```
**********************************************************
    2200 Thu Aug 22 18:59:20 2013
new/usr/src/pkg/manifests/driver-network-sfxge.mf
Merged sfxge driver
**********************************************************
```

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#

#
# Copyright (c) 2010, Oracle and/or its affiliates. All rights reserved.
# Copyright (c) 2012, Joyent, Inc. All rights reserved.
# Copyright 2012 Nexenta Systems, Inc. All rights reserved.
#

#
# The default for payload-bearing actions in this package is to appear in the
# global zone only.  See the include file for greater detail, as well as
# information about overriding the defaults.
#
<include global_zone_only_component>
set name=pkg.fmri value=pkg:/driver/network/sfxge@$(PKGVERS)
set name=pkg.description value="Solarflare 10GbE PCIE NIC Driver"
set name=pkg.summary value="Solarflare 10GbE PCIE NIC Driver"
set name=info.classification \
    value=org.opensolaris.category.2008:Drivers/Networking
set name=variant.arch value=$(ARCH)
dir path=kernel group=sys
dir path=kernel/drv group=sys
dir path=kernel/drv/$(ARCH64) group=sys
driver name=sfxge clone_perms="sfxge 0666 root sys" perms="* 0666 root sys" \
    alias=pciex1924,703 \
    alias=pciex1924,710 \
    alias=pciex1924,803 \
    alias=pciex1924,813
file path=kernel/drv/$(ARCH64)/sfxge group=sys
$(i386_ONLY)file path=kernel/drv/sfxge group=sys
file path=kernel/drv/sfxge.conf group=sys \
    original_name=SFCsfxge:kernel/drv/sfxge.conf preserve=renamenew
legacy pkg=SFCsfxge desc="Solarflare 10GbE PCIE NIC Driver" \
    name="Solarflare 10GbE PCIE NIC Driver"
license cr_Sun license=cr_Sun
license lic_CDDL license=lic_CDDL
#endif /* ! codereview */
```

```
**********************************************************
   44269 Thu Aug 22 18:59:20 2013
new/usr/src/uts/common/Makefile.files
Merged sfxge driver
**********************************************************
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#

#
# Copyright (c) 1991, 2010, Oracle and/or its affiliates. All rights reserved.
# Copyright (c) 2012 Nexenta Systems, Inc. All rights reserved.
# Copyright (c) 2012 by Delphix. All rights reserved.
# Copyright (c) 2013 by Saso Kiselkov. All rights reserved.
#

#
# This Makefile defines all file modules for the directory uts/common
# and its children. These are the source files which may be considered
# common to all SunOS systems.

i386_CORE_OBJS += \
                atomic.o        \
                avintr.o        \
                pic.o

sparc_CORE_OBJS +=

COMMON_CORE_OBJS +=             \
                beep.o          \
                bitset.o        \
                bp_map.o        \
                brand.o         \
                cpucaps.o       \
                cmt.o           \
                cmt_policy.o    \
                cpu.o           \
                cpu_event.o     \
                cpu_intr.o      \
                cpu_pm.o        \
                cpupart.o       \
                cap_util.o      \
                disp.o          \
                group.o         \
                kstat_fr.o      \
                iscsiboot_prop.o        \
                lgrp.o          \
                lgrp_topo.o     \
                mmapobj.o       \
```



```
                mutex.o         \
                page_lock.o     \
                page_retire.o   \
                panic.o         \
                param.o         \
                pg.o            \
                pghw.o          \
                putnext.o       \
                rctl_proc.o     \
                rwlock.o        \
                seg_kmem.o      \
                softint.o       \
                string.o        \
                strtol.o        \
                strtoul.o       \
                strtoll.o       \
                strtoull.o      \
                thread_intr.o   \
                vm_page.o       \
                vm_pagelist.o   \
                zlib_obj.o      \
                clock_tick.o

CORE_OBJS +=    $(COMMON_CORE_OBJS) $($(MACH)_CORE_OBJS)

ZLIB_OBJS =     zutil.o zmod.o zmod_subr.o \
                adler32.o crc32.o deflate.o inffast.o \
                inflate.o inftrees.o trees.o

GENUNIX_OBJS += \
                access.o        \
                acl.o           \
                acl_common.o    \
                adjtime.o       \
                alarm.o         \
                aio_subr.o      \
                auditsys.o      \
                audit_core.o    \
                audit_zone.o    \
                audit_memory.o  \
                autoconf.o      \
                avl.o           \
                bdev_dsort.o    \
                bio.o           \
                bitmap.o        \
                blabel.o        \
                brandsys.o      \
                bz2blocksort.o  \
                bz2compress.o   \
                bz2decompress.o \
                bz2randtable.o  \
                bz2bzlib.o      \
                bz2crctable.o   \
                bz2huffman.o    \
                callb.o         \
                callout.o       \
                chdir.o         \
                chmod.o         \
                chown.o         \
                cladm.o         \
                class.o         \
                clock.o         \
                clock_highres.o \
                clock_realtime.o\
                close.o         \
                compress.o      \
```

```
128                 condvar.o       \
129                 conf.o          \
130                 console.o       \
131                 contract.o      \
132                 copyops.o       \
133                 core.o          \
134                 corectl.o       \
135                 cred.o          \
136                 cs_stubs.o      \
137                 dacf.o          \
138                 dacf_clnt.o     \
139                 damap.o \
140                 cyclic.o        \
141                 ddi.o           \
142                 ddifm.o         \
143                 ddi_hp_impl.o   \
144                 ddi_hp_ndi.o    \
145                 ddi_intr.o      \
146                 ddi_intr_impl.o \
147                 ddi_intr_irm.o  \
148                 ddi_nodeid.o    \
149                 ddi_periodic.o  \
150                 devcfg.o        \
151                 devcache.o      \
152                 device.o        \
153                 devid.o         \
154                 devid_cache.o   \
155                 devid_scsi.o    \
156                 devid_smp.o     \
157                 devpolicy.o     \
158                 disp_lock.o     \
159                 dnlc.o          \
160                 driver.o        \
161                 dumpsubr.o      \
162                 driver_lyr.o    \
163                 dtrace_subr.o   \
164                 errorq.o        \
165                 etheraddr.o     \
166                 evchannels.o    \
167                 exacct.o        \
168                 exacct_core.o   \
169                 exec.o          \
170                 exit.o          \
171                 fbio.o          \
172                 fcntl.o         \
173                 fdbuffer.o      \
174                 fdsync.o        \
175                 fem.o           \
176                 ffs.o           \
177                 fio.o           \
178                 flock.o         \
179                 fm.o            \
180                 fork.o          \
181                 vpm.o           \
182                 fs_reparse.o    \
183                 fs_subr.o       \
184                 fsflush.o       \
185                 ftrace.o        \
186                 getcwd.o        \
187                 getdents.o      \
188                 getloadavg.o    \
189                 getpagesizes.o  \
190                 getpid.o        \
191                 gfs.o           \
192                 rusagesys.o     \
193                 gid.o           \
```



```
194                 groups.o        \
195                 grow.o          \
196                 hat_refmod.o    \
197                 id32.o          \
198                 id_space.o      \
199                 inet_ntop.o     \
200                 instance.o      \
201                 ioctl.o         \
202                 ip_cksum.o      \
203                 issetugid.o     \
204                 ippconf.o       \
205                 kcpc.o          \
206                 kdi.o           \
207                 kiconv.o        \
208                 klpd.o          \
209                 kmem.o          \
210                 ksyms_snapshot.o        \
211                 l_strplumb.o    \
212                 labelsys.o      \
213                 link.o          \
214                 list.o          \
215                 lockstat_subr.o \
216                 log_sysevent.o  \
217                 logsubr.o       \
218                 lookup.o        \
219                 lseek.o         \
220                 ltos.o          \
221                 lwp.o           \
222                 lwp_create.o    \
223                 lwp_info.o      \
224                 lwp_self.o      \
225                 lwp_sobj.o      \
226                 lwp_timer.o     \
227                 lwpsys.o        \
228                 main.o          \
229                 mmapobjsys.o    \
230                 memcntl.o       \
231                 memstr.o        \
232                 lgrpsys.o       \
233                 mkdir.o         \
234                 mknod.o         \
235                 mount.o         \
236                 move.o          \
237                 msacct.o        \
238                 multidata.o     \
239                 nbmlock.o       \
240                 ndifm.o         \
241                 nice.o          \
242                 netstack.o      \
243                 ntptime.o       \
244                 nvpair.o        \
245                 nvpair_alloc_system.o   \
246                 nvpair_alloc_fixed.o    \
247                 fnvpair.o       \
248                 octet.o         \
249                 open.o          \
250                 p_online.o      \
251                 pathconf.o      \
252                 pathname.o      \
253                 pause.o         \
254                 serializer.o    \
255                 pci_intr_lib.o  \
256                 pci_cap.o       \
257                 pcifm.o         \
258                 pgrp.o          \
259                 pgrpsys.o       \
```

```
260                 pid.o           \
261                 pkp_hash.o      \
262                 policy.o        \
263                 poll.o          \
264                 pool.o          \
265                 pool_pset.o     \
266                 port_subr.o     \
267                 ppriv.o         \
268                 printf.o        \
269                 priocntl.o      \
270                 priv.o          \
271                 priv_const.o    \
272                 proc.o          \
273                 procset.o       \
274                 processor_bind.o        \
275                 processor_info.o        \
276                 profil.o        \
277                 project.o       \
278                 qsort.o         \
279                 rctl.o          \
280                 rctlsys.o       \
281                 readlink.o      \
282                 refstr.o        \
283                 rename.o        \
284                 resolvepath.o   \
285                 retire_store.o  \
286                 process.o       \
287                 rlimit.o        \
288                 rmap.o          \
289                 rw.o            \
290                 rwstlock.o      \
291                 sad_conf.o      \
292                 sid.o           \
293                 sidsys.o        \
294                 sched.o         \
295                 schedctl.o      \
296                 sctp_crc32.o    \
297                 seg_dev.o       \
298                 seg_kp.o        \
299                 seg_kpm.o       \
300                 seg_map.o       \
301                 seg_vn.o        \
302                 seg_spt.o       \
303                 semaphore.o     \
304                 sendfile.o      \
305                 session.o       \
306                 share.o         \
307                 shuttle.o       \
308                 sig.o           \
309                 sigaction.o     \
310                 sigaltstack.o   \
311                 signotify.o     \
312                 sigpending.o    \
313                 sigprocmask.o   \
314                 sigqueue.o      \
315                 sigsendset.o    \
316                 sigsuspend.o    \
317                 sigtimedwait.o  \
318                 sleepq.o        \
319                 sock_conf.o     \
320                 space.o         \
321                 sscanf.o        \
322                 stat.o          \
323                 statfs.o        \
324                 statvfs.o       \
325                 stol.o          \
```



```
326                 str_conf.o      \
327                 strcalls.o      \
328                 stream.o        \
329                 streamio.o      \
330                 strext.o        \
331                 strsubr.o       \
332                 strsun.o        \
333                 subr.o          \
334                 sunddi.o        \
335                 sunmdi.o        \
336                 sunndi.o        \
337                 sunpci.o        \
338                 sunpm.o         \
339                 sundlpi.o       \
340                 suntpi.o        \
341                 swap_subr.o     \
342                 swap_vnops.o    \
343                 symlink.o       \
344                 sync.o          \
345                 sysclass.o      \
346                 sysconfig.o     \
347                 sysent.o        \
348                 sysfs.o         \
349                 systeminfo.o    \
350                 task.o          \
351                 taskq.o         \
352                 tasksys.o       \
353                 time.o          \
354                 timer.o         \
355                 times.o         \
356                 timers.o        \
357                 thread.o        \
358                 tlabel.o        \
359                 tnf_res.o       \
360                 turnstile.o     \
361                 tty_common.o    \
362                 u8_textprep.o   \
363                 uadmin.o        \
364                 uconv.o         \
365                 ucredsys.o      \
366                 uid.o           \
367                 umask.o         \
368                 umount.o        \
369                 uname.o         \
370                 unix_bb.o       \
371                 unlink.o        \
372                 urw.o           \
373                 utime.o         \
374                 utssys.o        \
375                 uucopy.o        \
376                 vfs.o           \
377                 vfs_conf.o      \
378                 vmem.o          \
379                 vm_anon.o       \
380                 vm_as.o         \
381                 vm_meter.o      \
382                 vm_pageout.o    \
383                 vm_pvn.o        \
384                 vm_rm.o         \
385                 vm_seg.o        \
386                 vm_subr.o       \
387                 vm_swap.o       \
388                 vm_usage.o      \
389                 vnode.o         \
390                 vuid_queue.o    \
391                 vuid_store.o    \
```

```
392                 waitq.o         \
393                 watchpoint.o    \
394                 yield.o         \
395                 scsi_confdata.o \
396                 xattr.o         \
397                 xattr_common.o  \
398                 xdr_mblk.o      \
399                 xdr_mem.o       \
400                 xdr.o           \
401                 xdr_array.o     \
402                 xdr_refer.o     \
403                 xhat.o          \
404                 zone.o

406 #
407 #       Stubs for the stand-alone linker/loader
408 #
409 sparc_GENSTUBS_OBJS =   \
410         kobj_stubs.o

412 i386_GENSTUBS_OBJS =

414 COMMON_GENSTUBS_OBJS =

416 GENSTUBS_OBJS += $(COMMON_GENSTUBS_OBJS) $($(MACH)_GENSTUBS_OBJS)

418 #
419 #       DTrace and DTrace Providers
420 #
421 DTRACE_OBJS += dtrace.o dtrace_isa.o dtrace_asm.o

423 SDT_OBJS += sdt_subr.o

425 PROFILE_OBJS += profile.o

427 SYSTRACE_OBJS += systrace.o

429 LOCKSTAT_OBJS += lockstat.o

431 FASTTRAP_OBJS += fasttrap.o fasttrap_isa.o

433 DCPC_OBJS += dcpc.o

435 #
436 #       Driver (pseudo-driver) Modules
437 #
438 IPP_OBJS +=     ippctl.o

440 AUDIO_OBJS += audio_client.o audio_ddi.o audio_engine.o \
441             audio_fltdata.o audio_format.o audio_ctrl.o \
442             audio_grc3.o audio_output.o audio_input.o \
443             audio_oss.o audio_sun.o

445 AUDIOEMU10K_OBJS += audioemu10k.o

447 AUDIOENS_OBJS += audioens.o

449 AUDIOVIA823X_OBJS += audiovia823x.o

451 AUDIOVIA97_OBJS += audiovia97.o

453 AUDIO1575_OBJS += audio1575.o

455 AUDIO810_OBJS += audio810.o

457 AUDIOCMI_OBJS += audiocmi.o
```



```
459 AUDIOCMIHD_OBJS += audiocmihd.o

461 AUDIOHD_OBJS += audiohd.o

463 AUDIOIXP_OBJS += audioixp.o

465 AUDIOLS_OBJS += audiols.o

467 AUDIOP16X_OBJS += audiop16x.o

469 AUDIOPCI_OBJS += audiopci.o

471 AUDIOSOLO_OBJS += audiosolo.o

473 AUDIOTS_OBJS += audiots.o

475 AC97_OBJS += ac97.o ac97_ad.o ac97_alc.o ac97_cmi.o

477 BLKDEV_OBJS += blkdev.o

479 CARDBUS_OBJS += cardbus.o cardbus_hp.o cardbus_cfg.o

481 CONSKBD_OBJS += conskbd.o

483 CONSMS_OBJS +=  consms.o

485 OLDPTY_OBJS +=  tty_ptyconf.o

487 PTC_OBJS +=     tty_pty.o

489 PTSL_OBJS +=    tty_pts.o

491 PTM_OBJS +=     ptm.o

493 MII_OBJS +=     mii.o mii_cicada.o mii_natsemi.o mii_intel.o mii_qualsemi.o \
494                 mii_marvell.o mii_realtek.o mii_other.o

496 PTS_OBJS +=     pts.o

498 PTY_OBJS +=     ptms_conf.o

500 SAD_OBJS +=     sad.o

502 MD4_OBJS +=     md4.o md4_mod.o

504 MD5_OBJS +=     md5.o md5_mod.o

506 SHA1_OBJS +=    sha1.o sha1_mod.o

508 SHA2_OBJS +=    sha2.o sha2_mod.o

510 IPGPC_OBJS +=   classifierddi.o classifier.o filters.o trie.o table.o \
511                 ba_table.o

513 DSCPMK_OBJS +=  dscpmk.o dscpmkddi.o

515 DLCOSMK_OBJS += dlcosmk.o dlcosmkddi.o

517 FLOWACCT_OBJS +=        flowacctddi.o flowacct.o

519 TOKENMT_OBJS += tokenmt.o tokenmtddi.o

521 TSWTCL_OBJS +=  tswtcl.o tswtclddi.o

523 ARP_OBJS +=     arpddi.o
```

```
525 ICMP_OBJS +=    icmpddi.o

527 ICMP6_OBJS +=   icmp6ddi.o

529 RTS_OBJS +=     rtsddi.o

531 IP_ICMP_OBJS =  icmp.o icmp_opt_data.o
532 IP_RTS_OBJS =   rts.o rts_opt_data.o
533 IP_TCP_OBJS =   tcp.o tcp_fusion.o tcp_opt_data.o tcp_sack.o tcp_stats.o \
534                 tcp_misc.o tcp_timers.o tcp_time_wait.o tcp_tpi.o tcp_output.o \
535                 tcp_input.o tcp_socket.o tcp_bind.o tcp_cluster.o tcp_tunables.o
536 IP_UDP_OBJS =   udp.o udp_opt_data.o udp_tunables.o udp_stats.o
537 IP_SCTP_OBJS =  sctp.o sctp_opt_data.o sctp_output.o \
538                 sctp_init.o sctp_input.o sctp_cookie.o \
539                 sctp_conn.o sctp_error.o sctp_snmp.o \
540                 sctp_tunables.o sctp_shutdown.o sctp_common.o \
541                 sctp_timer.o sctp_heartbeat.o sctp_hash.o \
542                 sctp_bind.o sctp_notify.o sctp_asconf.o \
543                 sctp_addr.o tn_ipopt.o tnet.o ip_netinfo.o \
544                 sctp_misc.o
545 IP_ILB_OBJS =   ilb.o ilb_nat.o ilb_conn.o ilb_alg_hash.o ilb_alg_rr.o

547 IP_OBJS +=      igmp.o ipmp.o ip.o ip6.o ip6_asp.o ip6_if.o ip6_ire.o \
548                 ip6_rts.o ip_if.o ip_ire.o ip_listutils.o ip_mroute.o \
549                 ip_multi.o ip2mac.o ip_ndp.o ip_rts.o ip_srcid.o \
550                 ipddi.o ipdrop.o mi.o nd.o tunables.o optcom.o snmpcom.o \
551                 ipsec_loader.o spd.o ipclassifier.o inet_common.o ip_squeue.o \
552                 squeue.o ip_sadb.o ip_ftable.o proto_set.o radix.o ip_dummy.o \
553                 ip_helper_stream.o ip_tunables.o \
554                 ip_output.o ip_input.o ip6_input.o ip6_output.o ip_arp.o \
555                 conn_opt.o ip_attr.o ip_dce.o \
556                 $(IP_ICMP_OBJS) \
557                 $(IP_RTS_OBJS) \
558                 $(IP_TCP_OBJS) \
559                 $(IP_UDP_OBJS) \
560                 $(IP_SCTP_OBJS) \
561                 $(IP_ILB_OBJS)

563 IP6_OBJS +=     ip6ddi.o

565 HOOK_OBJS +=    hook.o

567 NETI_OBJS +=    neti_impl.o neti_mod.o neti_stack.o

569 KEYSOCK_OBJS += keysockddi.o keysock.o keysock_opt_data.o

571 IPNET_OBJS +=   ipnet.o ipnet_bpf.o

573 SPDSOCK_OBJS += spdsockddi.o spdsock.o spdsock_opt_data.o

575 IPSECESP_OBJS += ipsecespddi.o ipsecesp.o

577 IPSECAH_OBJS += ipsecahddi.o ipsecah.o sadb.o

579 SPPP_OBJS +=    sppp.o sppp_dlpi.o sppp_mod.o s_common.o

581 SPPPTUN_OBJS += sppptun.o sppptun_mod.o

583 SPPPASYN_OBJS += spppasyn.o spppasyn_mod.o

585 SPPPCOMP_OBJS += spppcomp.o spppcomp_mod.o deflate.o bsd-comp.o vjcompress.o \
586                 zlib.o

588 TCP_OBJS +=     tcpddi.o
```



```
590 TCP6_OBJS +=    tcp6ddi.o

592 NCA_OBJS +=     ncaddi.o

594 SDP_SOCK_MOD_OBJS += sockmod_sdp.o socksdp.o socksdpsubr.o

596 SCTP_SOCK_MOD_OBJS += sockmod_sctp.o socksctp.o socksctpsubr.o

598 PFP_SOCK_MOD_OBJS += sockmod_pfp.o

600 RDS_SOCK_MOD_OBJS += sockmod_rds.o

602 RDS_OBJS +=     rdsddi.o rdssubr.o rds_opt.o rds_ioctl.o

604 RDSIB_OBJS +=   rdsib.o rdsib_ib.o rdsib_cm.o rdsib_ep.o rdsib_buf.o \
605                 rdsib_debug.o rdsib_sc.o

607 RDSV3_OBJS +=   af_rds.o rdsv3_ddi.o bind.o loop.o threads.o connection.o \
608                 transport.o cong.o sysctl.o message.o rds_recv.o send.o \
609                 stats.o info.o page.o rdma_transport.o ib_ring.o ib_rdma.o \
610                 ib_recv.o ib.o ib_send.o ib_sysctl.o ib_stats.o ib_cm.o \
611                 rdsv3_sc.o rdsv3_debug.o rdsv3_impl.o rdma.o rdsv3_af_thr.o

613 ISER_OBJS +=    iser.o iser_cm.o iser_cq.o iser_ib.o iser_idm.o \
614                 iser_resource.o iser_xfer.o

616 UDP_OBJS +=     udpddi.o

618 UDP6_OBJS +=    udp6ddi.o

620 SY_OBJS +=      gentty.o

622 TCO_OBJS +=     ticots.o

624 TCOO_OBJS +=    ticotsord.o

626 TCL_OBJS +=     ticlts.o

628 TL_OBJS +=      tl.o

630 DUMP_OBJS +=    dump.o

632 BPF_OBJS +=     bpf.o bpf_filter.o bpf_mod.o bpf_dlt.o bpf_mac.o

634 CLONE_OBJS +=   clone.o

636 CN_OBJS +=      cons.o

638 DLD_OBJS +=     dld_drv.o dld_proto.o dld_str.o dld_flow.o

640 DLS_OBJS +=     dls.o dls_link.o dls_mod.o dls_stat.o dls_mgmt.o

642 GLD_OBJS +=     gld.o gldutil.o

644 MAC_OBJS +=     mac.o  mac_bcast.o mac_client.o mac_datapath_setup.o mac_flow.o
645                 mac_hio.o mac_mod.o mac_ndd.o mac_provider.o mac_sched.o \
646                 mac_protect.o mac_soft_ring.o mac_stat.o mac_util.o

648 MAC_6TO4_OBJS +=        mac_6to4.o

650 MAC_ETHER_OBJS +=       mac_ether.o

652 MAC_IPV4_OBJS +=        mac_ipv4.o

654 MAC_IPV6_OBJS +=        mac_ipv6.o
```

```
656 MAC_WIFI_OBJS +=        mac_wifi.o

658 MAC_IB_OBJS +=          mac_ib.o

660 IPTUN_OBJS +=   iptun_dev.o iptun_ctl.o iptun.o

662 AGGR_OBJS +=    aggr_dev.o aggr_ctl.o aggr_grp.o aggr_port.o \
663                 aggr_send.o aggr_recv.o aggr_lacp.o

665 SOFTMAC_OBJS += softmac_main.o softmac_ctl.o softmac_capab.o \
666                 softmac_dev.o softmac_stat.o softmac_pkt.o softmac_fp.o

668 NET80211_OBJS += net80211.o net80211_proto.o net80211_input.o \
669                  net80211_output.o net80211_node.o net80211_crypto.o \
670                  net80211_crypto_none.o net80211_crypto_wep.o net80211_ioctl.o \
671                  net80211_crypto_tkip.o net80211_crypto_ccmp.o  \
672                  net80211_ht.o

674 VNIC_OBJS +=    vnic_ctl.o vnic_dev.o

676 SIMNET_OBJS +=  simnet.o

678 IB_OBJS +=      ibnex.o ibnex_ioctl.o ibnex_hca.o

680 IBCM_OBJS +=    ibcm_impl.o ibcm_sm.o ibcm_ti.o ibcm_utils.o ibcm_path.o \
681                 ibcm_arp.o ibcm_arp_link.o

683 IBDM_OBJS +=    ibdm.o

685 IBDMA_OBJS +=   ibdma.o

687 IBMF_OBJS +=    ibmf.o ibmf_impl.o ibmf_dr.o ibmf_wqe.o ibmf_ud_dest.o ibmf_mod.
688                 ibmf_send.o ibmf_recv.o ibmf_handlers.o ibmf_trans.o \
689                 ibmf_timers.o ibmf_msg.o ibmf_utils.o ibmf_rmpp.o \
690                 ibmf_saa.o ibmf_saa_impl.o ibmf_saa_utils.o ibmf_saa_events.o

692 IBTL_OBJS +=    ibtl_impl.o ibtl_util.o ibtl_mem.o ibtl_handlers.o ibtl_qp.o \
693                 ibtl_cq.o ibtl_wr.o ibtl_hca.o ibtl_chan.o ibtl_cm.o \
694                 ibtl_mcg.o ibtl_ibnex.o ibtl_srq.o ibtl_part.o

696 TAVOR_OBJS +=   tavor.o tavor_agents.o tavor_cfg.o tavor_ci.o tavor_cmd.o \
697                 tavor_cq.o tavor_event.o tavor_ioctl.o tavor_misc.o \
698                 tavor_mr.o tavor_qp.o tavor_qpmod.o tavor_rsrc.o \
699                 tavor_srq.o tavor_stats.o tavor_umap.o tavor_wr.o

701 HERMON_OBJS +=  hermon.o hermon_agents.o hermon_cfg.o hermon_ci.o hermon_cmd.o \
702                 hermon_cq.o hermon_event.o hermon_ioctl.o hermon_misc.o \
703                 hermon_mr.o hermon_qp.o hermon_qpmod.o hermon_rsrc.o \
704                 hermon_srq.o hermon_stats.o hermon_umap.o hermon_wr.o \
705                 hermon_fcoib.o hermon_fm.o

707 DAPLT_OBJS +=   daplt.o

709 SOL_OFS_OBJS += sol_cma.o sol_ib_cma.o sol_uobj.o \
710                 sol_ofs_debug_util.o sol_ofs_gen_util.o \
711                 sol_kverbs.o

713 SOL_UCMA_OBJS +=        sol_ucma.o

715 SOL_UVERBS_OBJS +=      sol_uverbs.o sol_uverbs_comp.o sol_uverbs_event.o \
716                         sol_uverbs_hca.o sol_uverbs_qp.o

718 SOL_UMAD_OBJS += sol_umad.o

720 KSTAT_OBJS +=   kstat.o
```



```
722 KSYMS_OBJS +=   ksyms.o

724 INSTANCE_OBJS += inst_sync.o

726 IWSCN_OBJS +=   iwscons.o

728 LOFI_OBJS +=    lofi.o LzmaDec.o

730 FSSNAP_OBJS +=  fssnap.o

732 FSSNAPIF_OBJS += fssnap_if.o

734 MM_OBJS +=      mem.o

736 PHYSMEM_OBJS += physmem.o

738 OPTIONS_OBJS += options.o

740 WINLOCK_OBJS += winlockio.o

742 PM_OBJS +=      pm.o
743 SRN_OBJS +=     srn.o

745 PSEUDO_OBJS +=  pseudonex.o

747 RAMDISK_OBJS += ramdisk.o

749 LLC1_OBJS += llc1.o

751 USBKBM_OBJS += usbkbm.o

753 USBWCM_OBJS += usbwcm.o

755 BOFI_OBJS += bofi.o

757 HID_OBJS += hid.o

759 HWA_RC_OBJS += hwarc.o

761 USBSKEL_OBJS += usbskel.o

763 USBVC_OBJS += usbvc.o usbvc_v4l2.o

765 HIDPARSER_OBJS += hidparser.o

767 USB_AC_OBJS += usb_ac.o

769 USB_AS_OBJS += usb_as.o

771 USB_AH_OBJS += usb_ah.o

773 USBMS_OBJS += usbms.o

775 USBPRN_OBJS += usbprn.o

777 UGEN_OBJS += ugen.o

779 USBSER_OBJS += usbser.o usbser_rseq.o

781 USBSACM_OBJS += usbsacm.o

783 USBSER_KEYSPAN_OBJS += usbser_keyspan.o keyspan_dsd.o keyspan_pipe.o

785 USBS49_FW_OBJS += keyspan_49fw.o

787 USBSPRL_OBJS += usbser_pl2303.o pl2303_dsd.o
```

```
789 WUSB_CA_OBJS += wusb_ca.o

791 USBFTDI_OBJS += usbser_uftdi.o uftdi_dsd.o

793 USBECM_OBJS += usbecm.o

795 WC_OBJS += wscons.o vcons.o

797 VCONS_CONF_OBJS += vcons_conf.o

799 SCSI_OBJS +=    scsi_capabilities.o scsi_confsubr.o scsi_control.o \
800                 scsi_data.o scsi_fm.o scsi_hba.o scsi_reset_notify.o \
801                 scsi_resource.o scsi_subr.o scsi_transport.o scsi_watch.o \
802                 smp_transport.o

804 SCSI_VHCI_OBJS +=               scsi_vhci.o mpapi_impl.o scsi_vhci_tpgs.o

806 SCSI_VHCI_F_SYM_OBJS +=         sym.o

808 SCSI_VHCI_F_TPGS_OBJS +=        tpgs.o

810 SCSI_VHCI_F_ASYM_SUN_OBJS +=    asym_sun.o

812 SCSI_VHCI_F_SYM_HDS_OBJS +=     sym_hds.o

814 SCSI_VHCI_F_TAPE_OBJS +=        tape.o

816 SCSI_VHCI_F_TPGS_TAPE_OBJS +=   tpgs_tape.o

818 SGEN_OBJS +=    sgen.o

820 SMP_OBJS +=     smp.o

822 SATA_OBJS +=    sata.o

824 USBA_OBJS +=    hcdi.o  usba.o  usbai.o hubdi.o parser.o genconsole.o \
825                 usbai_pipe_mgmt.o usbai_req.o usbai_util.o usbai_register.o \
826                 usba_devdb.o usba10_calls.o usba_ugen.o whcdi.o wa.o
827 USBA_WITHOUT_WUSB_OBJS +=       hcdi.o  usba.o  usbai.o hubdi.o parser.o gencons
828                 usbai_pipe_mgmt.o usbai_req.o usbai_util.o usbai_register.o \
829                 usba_devdb.o usba10_calls.o usba_ugen.o

831 USBA10_OBJS +=  usba10.o

833 RSM_OBJS +=     rsm.o   rsmka_pathmanager.o     rsmka_util.o

835 RSMOPS_OBJS +=  rsmops.o

837 S1394_OBJS +=   t1394.o t1394_errmsg.o s1394.o s1394_addr.o s1394_asynch.o \
838                 s1394_bus_reset.o s1394_cmp.o s1394_csr.o s1394_dev_disc.o \
839                 s1394_fa.o s1394_fcp.o \
840                 s1394_hotplug.o s1394_isoch.o s1394_misc.o h1394.o nx1394.o

842 HCI1394_OBJS += hci1394.o hci1394_async.o hci1394_attach.o hci1394_buf.o \
843                 hci1394_csr.o hci1394_detach.o hci1394_extern.o \
844                 hci1394_ioctl.o hci1394_isoch.o hci1394_isr.o \
845                 hci1394_ixl_comp.o hci1394_ixl_isr.o hci1394_ixl_misc.o \
846                 hci1394_ixl_update.o hci1394_misc.o hci1394_ohci.o \
847                 hci1394_q.o hci1394_s1394if.o hci1394_tlabel.o \
848                 hci1394_tlist.o hci1394_vendor.o

850 AV1394_OBJS +=  av1394.o av1394_as.o av1394_async.o av1394_cfgrom.o \
851                 av1394_cmp.o av1394_fcp.o av1394_isoch.o av1394_isoch_chan.o \
852                 av1394_isoch_recv.o av1394_isoch_xmit.o av1394_list.o \
853                 av1394_queue.o
```



```
855 DCAM1394_OBJS += dcam.o dcam_frame.o dcam_param.o dcam_reg.o \
856                 dcam_ring_buff.o

858 SCSA1394_OBJS += hba.o sbp2_driver.o sbp2_bus.o

860 SBP2_OBJS += cfgrom.o sbp2.o

862 PMODEM_OBJS += pmodem.o pmodem_cis.o cis.o cis_callout.o cis_handlers.o cis_para

864 DSW_OBJS +=     dsw.o dsw_dev.o ii_tree.o

866 NCALL_OBJS +=   ncall.o \
867                 ncall_stub.o

869 RDC_OBJS +=     rdc.o \
870                 rdc_dev.o \
871                 rdc_io.o \
872                 rdc_clnt.o \
873                 rdc_prot_xdr.o \
874                 rdc_svc.o \
875                 rdc_bitmap.o \
876                 rdc_health.o \
877                 rdc_subr.o \
878                 rdc_diskq.o

880 RDCSRV_OBJS +=  rdcsrv.o

882 RDCSTUB_OBJS += rdc_stub.o

884 SDBC_OBJS +=    sd_bcache.o \
885                 sd_bio.o \
886                 sd_conf.o \
887                 sd_ft.o \
888                 sd_hash.o \
889                 sd_io.o \
890                 sd_misc.o \
891                 sd_pcu.o \
892                 sd_tdaemon.o \
893                 sd_trace.o \
894                 sd_iob_impl0.o \
895                 sd_iob_impl1.o \
896                 sd_iob_impl2.o \
897                 sd_iob_impl3.o \
898                 sd_iob_impl4.o \
899                 sd_iob_impl5.o \
900                 sd_iob_impl6.o \
901                 sd_iob_impl7.o \
902                 safestore.o \
903                 safestore_ram.o

905 NSCTL_OBJS +=   nsctl.o \
906                 nsc_cache.o \
907                 nsc_disk.o \
908                 nsc_dev.o \
909                 nsc_freeze.o \
910                 nsc_gen.o \
911                 nsc_mem.o \
912                 nsc_ncallio.o \
913                 nsc_power.o \
914                 nsc_resv.o \
915                 nsc_rmspin.o \
916                 nsc_solaris.o \
917                 nsc_trap.o \
918                 nsc_list.o
919 UNISTAT_OBJS += spuni.o \
```

```
 920                 spcs_s_k.o

 922 NSKERN_OBJS +=  nsc_ddi.o \
 923                 nsc_proc.o \
 924                 nsc_raw.o \
 925                 nsc_thread.o \
 926                 nskernd.o

 928 SV_OBJS +=      sv.o

 930 PMCS_OBJS += pmcs_attach.o pmcs_ds.o pmcs_intr.o pmcs_nvram.o pmcs_sata.o \
 931                 pmcs_scsa.o pmcs_smhba.o pmcs_subr.o pmcs_fwlog.o

 933 PMCS8001FW_C_OBJS +=    pmcs_fw_hdr.o
 934 PMCS8001FW_OBJS +=              $(PMCS8001FW_C_OBJS) SPCBoot.o ila.o firmware.o

 936 #
 937 #       Build up defines and paths.

 939 ST_OBJS +=      st.o    st_conf.o

 941 EMLXS_OBJS +=   emlxs_clock.o emlxs_dfc.o emlxs_dhchap.o emlxs_diag.o \
 942                 emlxs_download.o emlxs_dump.o emlxs_els.o emlxs_event.o \
 943                 emlxs_fcf.o emlxs_fcp.o emlxs_fct.o emlxs_hba.o emlxs_ip.o \
 944                 emlxs_mbox.o emlxs_mem.o emlxs_msg.o emlxs_node.o \
 945                 emlxs_pkt.o emlxs_sli3.o emlxs_sli4.o emlxs_solaris.o \
 946                 emlxs_thread.o

 948 EMLXS_FW_OBJS +=        emlxs_fw.o

 950 OCE_OBJS +=     oce_buf.o oce_fm.o oce_gld.o oce_hw.o oce_intr.o oce_main.o \
 951                 oce_mbx.o oce_mq.o oce_queue.o oce_rx.o oce_stat.o oce_tx.o \
 952                 oce_utils.o

 954 FCT_OBJS += discovery.o fct.o

 956 QLT_OBJS += 2400.o 2500.o 8100.o qlt.o qlt_dma.o

 958 SRPT_OBJS += srpt_mod.o srpt_ch.o srpt_cm.o srpt_ioc.o srpt_stp.o

 960 FCOE_OBJS += fcoe.o fcoe_eth.o fcoe_fc.o

 962 FCOET_OBJS += fcoet.o fcoet_eth.o fcoet_fc.o

 964 FCOEI_OBJS += fcoei.o fcoei_eth.o fcoei_lv.o

 966 ISCSIT_SHARED_OBJS += \
 967                 iscsit_common.o

 969 ISCSIT_OBJS +=  $(ISCSIT_SHARED_OBJS) \
 970                 iscsit.o iscsit_tgt.o iscsit_sess.o iscsit_login.o \
 971                 iscsit_text.o iscsit_isns.o iscsit_radiusauth.o \
 972                 iscsit_radiuspacket.o iscsit_auth.o iscsit_authclient.o

 974 PPPT_OBJS +=    alua_ic_if.o pppt.o pppt_msg.o pppt_tgt.o

 976 STMF_OBJS += lun_map.o stmf.o

 978 STMF_SBD_OBJS += sbd.o sbd_scsi.o sbd_pgr.o sbd_zvol.o

 980 SYSMSG_OBJS +=  sysmsg.o

 982 SES_OBJS +=     ses.o ses_sen.o ses_safte.o ses_ses.o

 984 TNF_OBJS +=     tnf_buf.o       tnf_trace.o     tnf_writer.o    trace_init.o \
 985                 trace_funcs.o   tnf_probe.o     tnf.o
```



```
 987 LOGINDMUX_OBJS += logindmux.o

 989 DEVINFO_OBJS += devinfo.o

 991 DEVPOLL_OBJS += devpoll.o

 993 DEVPOOL_OBJS += devpool.o

 995 I8042_OBJS +=   i8042.o

 997 KB8042_OBJS +=  \
 998                 at_keyprocess.o \
 999                 kb8042.o        \
1000                 kb8042_keytables.o

1002 MOUSE8042_OBJS += mouse8042.o

1004 FDC_OBJS +=     fdc.o

1006 ASY_OBJS +=     asy.o

1008 ECPP_OBJS +=    ecpp.o

1010 VUIDM3P_OBJS += vuidmice.o vuidm3p.o

1012 VUIDM4P_OBJS += vuidmice.o vuidm4p.o

1014 VUIDM5P_OBJS += vuidmice.o vuidm5p.o

1016 VUIDPS2_OBJS += vuidmice.o vuidps2.o

1018 HPCSVC_OBJS += hpcsvc.o

1020 PCIE_MISC_OBJS += pcie.o pcie_fault.o pcie_hp.o pciehpc.o pcishpc.o pcie_pwr.o p

1022 PCIHPNEXUS_OBJS += pcihp.o

1024 OPENEEPR_OBJS += openprom.o

1026 RANDOM_OBJS += random.o

1028 PSHOT_OBJS += pshot.o

1030 GEN_DRV_OBJS += gen_drv.o

1032 TCLIENT_OBJS += tclient.o

1034 TPHCI_OBJS += tphci.o

1036 TVHCI_OBJS += tvhci.o

1038 EMUL64_OBJS += emul64.o emul64_bsd.o

1040 FCP_OBJS += fcp.o

1042 FCIP_OBJS += fcip.o

1044 FCSM_OBJS += fcsm.o

1046 FCTL_OBJS += fctl.o

1048 FP_OBJS += fp.o

1050 QLC_OBJS += ql_api.o ql_debug.o ql_hba_fru.o ql_init.o ql_iocb.o ql_ioctl.o \
1051         ql_isr.o ql_mbx.o ql_nx.o ql_xioctl.o ql_fw_table.o
```

```
QLC_FW_2200_OBJS += ql_fw_2200.o

QLC_FW_2300_OBJS += ql_fw_2300.o

QLC_FW_2400_OBJS += ql_fw_2400.o

QLC_FW_2500_OBJS += ql_fw_2500.o

QLC_FW_6322_OBJS += ql_fw_6322.o

QLC_FW_8100_OBJS += ql_fw_8100.o

QLGE_OBJS += qlge.o qlge_dbg.o qlge_flash.o qlge_fm.o qlge_gld.o qlge_mpi.o

ZCONS_OBJS += zcons.o

NV_SATA_OBJS += nv_sata.o

SI3124_OBJS += si3124.o

AHCI_OBJS += ahci.o

PCIIDE_OBJS += pci-ide.o

PCEPP_OBJS += pcepp.o

CPC_OBJS += cpc.o

CPUID_OBJS += cpuid_drv.o

SYSEVENT_OBJS += sysevent.o

BL_OBJS += bl.o

DRM_OBJS += drm_sunmod.o drm_kstat.o drm_agpsupport.o \
            drm_auth.o drm_bufs.o drm_context.o drm_dma.o \
            drm_drawable.o drm_drv.o drm_fops.o drm_ioctl.o drm_irq.o \
            drm_lock.o drm_memory.o drm_msg.o drm_pci.o drm_scatter.o \
            drm_cache.o drm_gem.o drm_mm.o ati_pcigart.o

FM_OBJS += devfm.o devfm_machdep.o

RTLS_OBJS +=    rtls.o

#
#                       exec modules
#
AOUTEXEC_OBJS +=aout.o

ELFEXEC_OBJS += elf.o elf_notes.o old_notes.o

INTPEXEC_OBJS +=intp.o

SHBINEXEC_OBJS +=shbin.o

JAVAEXEC_OBJS +=java.o

#
#                       file system modules
#
AUTOFS_OBJS +=  auto_vfsops.o auto_vnops.o auto_subr.o auto_xdr.o auto_sys.o

CACHEFS_OBJS += cachefs_cnode.o         cachefs_cod.o \
                cachefs_dir.o           cachefs_dlog.o  cachefs_filegrp.o \
                cachefs_fscache.o       cachefs_ioctl.o cachefs_log.o \
```



```
                cachefs_module.o \
                cachefs_noopc.o         cachefs_resource.o \
                cachefs_strict.o \
                cachefs_subr.o          cachefs_vfsops.o \
                cachefs_vnops.o

DCFS_OBJS +=    dc_vnops.o

DEVFS_OBJS +=   devfs_subr.o    devfs_vfsops.o  devfs_vnops.o

DEV_OBJS  +=    sdev_subr.o     sdev_vfsops.o   sdev_vnops.o    \
                sdev_ptsops.o   sdev_zvolops.o  sdev_comm.o     \
                sdev_profile.o  sdev_ncache.o   sdev_netops.o   \
                sdev_ipnetops.o \
                sdev_vtops.o

CTFS_OBJS +=    ctfs_all.o ctfs_cdir.o ctfs_ctl.o ctfs_event.o \
                ctfs_latest.o ctfs_root.o ctfs_sym.o ctfs_tdir.o ctfs_tmpl.o

OBJFS_OBJS +=   objfs_vfs.o     objfs_root.o    objfs_common.o \
                objfs_odir.o    objfs_data.o

FDFS_OBJS +=    fdops.o

FIFO_OBJS +=    fifosubr.o      fifovnops.o

PIPE_OBJS +=    pipe.o

HSFS_OBJS +=    hsfs_node.o     hsfs_subr.o     hsfs_vfsops.o   hsfs_vnops.o \
                hsfs_susp.o     hsfs_rrip.o     hsfs_susp_subr.o

LOFS_OBJS +=    lofs_subr.o     lofs_vfsops.o   lofs_vnops.o

NAMEFS_OBJS +=  namevfs.o       namevno.o

NFS_OBJS +=     nfs_client.o    nfs_common.o    nfs_dump.o \
                nfs_subr.o      nfs_vfsops.o    nfs_vnops.o \
                nfs_xdr.o       nfs_sys.o       nfs_strerror.o \
                nfs3_vfsops.o   nfs3_vnops.o    nfs3_xdr.o \
                nfs_acl_vnops.o nfs_acl_xdr.o   nfs4_vfsops.o \
                nfs4_vnops.o    nfs4_xdr.o      nfs4_idmap.o \
                nfs4_shadow.o   nfs4_subr.o \
                nfs4_attr.o     nfs4_rnode.o    nfs4_client.o \
                nfs4_acache.o   nfs4_common.o   nfs4_client_state.o \
                nfs4_callback.o nfs4_recovery.o nfs4_client_secinfo.o \
                nfs4_client_debug.o     nfs_stats.o \
                nfs4_acl.o      nfs4_stub_vnops.o       nfs_cmd.o

NFSSRV_OBJS +=  nfs_server.o    nfs_srv.o       nfs3_srv.o \
                nfs_acl_srv.o   nfs_auth.o      nfs_auth_xdr.o \
                nfs_export.o    nfs_log.o       nfs_log_xdr.o \
                nfs4_srv.o      nfs4_state.o    nfs4_srv_attr.o \
                nfs4_srv_ns.o   nfs4_db.o       nfs4_srv_deleg.o \
                nfs4_deleg_ops.o nfs4_srv_readdir.o nfs4_dispatch.o

SMBSRV_SHARED_OBJS += \
                smb_inet.o \
                smb_match.o \
                smb_msgbuf.o \
                smb_oem.o \
                smb_string.o \
                smb_utf8.o \
                smb_door_legacy.o \
                smb_xdr.o \
                smb_token.o \
                smb_token_xdr.o \
```

```
1184                 smb_sid.o \
1185                 smb_native.o \
1186                 smb_netbios_util.o

1188 SMBSRV_OBJS +=  $(SMBSRV_SHARED_OBJS)                  \
1189                 smb_acl.o                              \
1190                 smb_alloc.o                            \
1191                 smb_close.o                            \
1192                 smb_common_open.o                      \
1193                 smb_common_transact.o                  \
1194                 smb_create.o                           \
1195                 smb_delete.o                           \
1196                 smb_directory.o                        \
1197                 smb_dispatch.o                         \
1198                 smb_echo.o                             \
1199                 smb_fem.o                              \
1200                 smb_find.o                             \
1201                 smb_flush.o                            \
1202                 smb_fsinfo.o                           \
1203                 smb_fsops.o                            \
1204                 smb_init.o                             \
1205                 smb_kdoor.o                            \
1206                 smb_kshare.o                           \
1207                 smb_kutil.o                            \
1208                 smb_lock.o                             \
1209                 smb_lock_byte_range.o                  \
1210                 smb_locking_andx.o                     \
1211                 smb_logoff_andx.o                      \
1212                 smb_mangle_name.o                      \
1213                 smb_mbuf_marshaling.o                  \
1214                 smb_mbuf_util.o                        \
1215                 smb_negotiate.o                        \
1216                 smb_net.o                              \
1217                 smb_node.o                             \
1218                 smb_nt_cancel.o                        \
1219                 smb_nt_create_andx.o                   \
1220                 smb_nt_transact_create.o               \
1221                 smb_nt_transact_ioctl.o                \
1222                 smb_nt_transact_notify_change.o        \
1223                 smb_nt_transact_quota.o                \
1224                 smb_nt_transact_security.o             \
1225                 smb_odir.o                             \
1226                 smb_ofile.o                            \
1227                 smb_open_andx.o                        \
1228                 smb_opipe.o                            \
1229                 smb_oplock.o                           \
1230                 smb_pathname.o                         \
1231                 smb_print.o                            \
1232                 smb_process_exit.o                     \
1233                 smb_query_fileinfo.o                   \
1234                 smb_read.o                             \
1235                 smb_rename.o                           \
1236                 smb_sd.o                               \
1237                 smb_seek.o                             \
1238                 smb_server.o                           \
1239                 smb_session.o                          \
1240                 smb_session_setup_andx.o               \
1241                 smb_set_fileinfo.o                     \
1242                 smb_signing.o                          \
1243                 smb_tree.o                             \
1244                 smb_trans2_create_directory.o          \
1245                 smb_trans2_dfs.o                       \
1246                 smb_trans2_find.o                      \
1247                 smb_tree_connect.o                     \
1248                 smb_unlock_byte_range.o                \
1249                 smb_user.o                             \
```



```
1250                 smb_vfs.o                              \
1251                 smb_vops.o                             \
1252                 smb_vss.o                              \
1253                 smb_write.o                            \
1254                 smb_write_raw.o

1256 PCFS_OBJS +=    pc_alloc.o      pc_dir.o        pc_node.o       pc_subr.o \
1257                 pc_vfsops.o     pc_vnops.o

1259 PROC_OBJS +=    prcontrol.o     prioctl.o       prsubr.o        prusrio.o \
1260                 prvfsops.o      prvnops.o

1262 MNTFS_OBJS +=   mntvfsops.o     mntvnops.o

1264 SHAREFS_OBJS += sharetab.o      sharefs_vfsops.o        sharefs_vnops.o

1266 SPEC_OBJS +=    specsubr.o      specvfsops.o    specvnops.o

1268 SOCK_OBJS +=    socksubr.o      sockvfsops.o    sockparams.o    \
1269                 socksyscalls.o  socktpi.o       sockstr.o \
1270                 sockcommon_vnops.o      sockcommon_subr.o \
1271                 sockcommon_sops.o       sockcommon.o    \
1272                 sock_notsupp.o  socknotify.o \
1273                 nl7c.o          nl7curi.o       nl7chttp.o      nl7clogd.o \
1274                 nl7cnca.o       sodirect.o      sockfilter.o

1276 TMPFS_OBJS +=   tmp_dir.o       tmp_subr.o      tmp_tnode.o     tmp_vfsops.o \
1277                 tmp_vnops.o

1279 UDFS_OBJS +=    udf_alloc.o     udf_bmap.o      udf_dir.o       \
1280                 udf_inode.o     udf_subr.o      udf_vfsops.o    \
1281                 udf_vnops.o

1283 UFS_OBJS +=     ufs_alloc.o     ufs_bmap.o      ufs_dir.o       ufs_xattr.o \
1284                 ufs_inode.o     ufs_subr.o      ufs_tables.o    ufs_vfsops.o \
1285                 ufs_vnops.o     quota.o         quotacalls.o    quota_ufs.o \
1286                 ufs_filio.o     ufs_lockfs.o    ufs_thread.o    ufs_trans.o \
1287                 ufs_acl.o       ufs_panic.o     ufs_directio.o  ufs_log.o \
1288                 ufs_extvnops.o  ufs_snap.o      lufs.o          lufs_thread.o \
1289                 lufs_log.o      lufs_map.o      lufs_top.o      lufs_debug.o
1290 VSCAN_OBJS +=   vscan_drv.o     vscan_svc.o vscan_door.o

1292 NSMB_OBJS +=    smb_conn.o      smb_dev.o       smb_iod.o       smb_pass.o \
1293                 smb_rq.o        smb_sign.o      smb_smb.o       smb_subrs.o \
1294                 smb_time.o      smb_tran.o      smb_trantcp.o   smb_usr.o \
1295                 subr_mchain.o

1297 SMBFS_COMMON_OBJS += smbfs_ntacl.o
1298 SMBFS_OBJS +=    smbfs_vfsops.o  smbfs_vnops.o   smbfs_node.o    \
1299                 smbfs_acl.o     smbfs_client.o  smbfs_smb.o     \
1300                 smbfs_subr.o    smbfs_subr2.o   \
1301                 smbfs_rwlock.o  smbfs_xattr.o   \
1302                 $(SMBFS_COMMON_OBJS)


1305 #
1306 #                       LVM modules
1307 #
1308 MD_OBJS += md.o md_error.o md_ioctl.o md_mddb.o md_names.o \
1309         md_med.o md_rename.o md_subr.o

1311 MD_COMMON_OBJS = md_convert.o md_crc.o md_revchk.o

1313 MD_DERIVED_OBJS = metamed_xdr.o meta_basic_xdr.o

1315 SOFTPART_OBJS += sp.o sp_ioctl.o
```

```
1317 STRIPE_OBJS += stripe.o stripe_ioctl.o

1319 HOTSPARES_OBJS += hotspares.o

1321 RAID_OBJS += raid.o raid_ioctl.o raid_replay.o raid_resync.o raid_hotspare.o

1323 MIRROR_OBJS += mirror.o mirror_ioctl.o mirror_resync.o

1325 NOTIFY_OBJS += md_notify.o

1327 TRANS_OBJS += mdtrans.o trans_ioctl.o trans_log.o

1329 ZFS_COMMON_OBJS +=             \
1330         arc.o                  \
1331         bplist.o               \
1332         bpobj.o                \
1333         bptree.o               \
1334         dbuf.o                 \
1335         ddt.o                  \
1336         ddt_zap.o              \
1337         dmu.o                  \
1338         dmu_diff.o             \
1339         dmu_send.o             \
1340         dmu_object.o           \
1341         dmu_objset.o           \
1342         dmu_traverse.o         \
1343         dmu_tx.o               \
1344         dnode.o                \
1345         dnode_sync.o           \
1346         dsl_dir.o              \
1347         dsl_dataset.o          \
1348         dsl_deadlist.o         \
1349         dsl_destroy.o          \
1350         dsl_pool.o             \
1351         dsl_synctask.o         \
1352         dsl_userhold.o         \
1353         dmu_zfetch.o           \
1354         dsl_deleg.o            \
1355         dsl_prop.o             \
1356         dsl_scan.o             \
1357         zfeature.o             \
1358         gzip.o                 \
1359         lz4.o                  \
1360         lzjb.o                 \
1361         metaslab.o             \
1362         refcount.o             \
1363         rrwlock.o              \
1364         sa.o                   \
1365         sha256.o               \
1366         spa.o                  \
1367         spa_config.o           \
1368         spa_errlog.o           \
1369         spa_history.o          \
1370         spa_misc.o             \
1371         space_map.o            \
1372         txg.o                  \
1373         uberblock.o            \
1374         unique.o               \
1375         vdev.o                 \
1376         vdev_cache.o           \
1377         vdev_file.o            \
1378         vdev_label.o           \
1379         vdev_mirror.o          \
1380         vdev_missing.o         \
1381         vdev_queue.o           \
```



```
1382         vdev_raidz.o           \
1383         vdev_root.o            \
1384         zap.o                  \
1385         zap_leaf.o             \
1386         zap_micro.o            \
1387         zfs_byteswap.o         \
1388         zfs_debug.o            \
1389         zfs_fm.o               \
1390         zfs_fuid.o             \
1391         zfs_sa.o               \
1392         zfs_znode.o            \
1393         zil.o                  \
1394         zio.o                  \
1395         zio_checksum.o         \
1396         zio_compress.o         \
1397         zio_inject.o           \
1398         zle.o                  \
1399         zrlock.o

1401 ZFS_SHARED_OBJS +=             \
1402         zfeature_common.o      \
1403         zfs_comutil.o          \
1404         zfs_deleg.o            \
1405         zfs_fletcher.o         \
1406         zfs_namecheck.o        \
1407         zfs_prop.o             \
1408         zpool_prop.o           \
1409         zprop_common.o

1411 ZFS_OBJS +=                    \
1412         $(ZFS_COMMON_OBJS)     \
1413         $(ZFS_SHARED_OBJS)     \
1414         vdev_disk.o            \
1415         zfs_acl.o              \
1416         zfs_ctldir.o           \
1417         zfs_dir.o              \
1418         zfs_ioctl.o            \
1419         zfs_log.o              \
1420         zfs_onexit.o           \
1421         zfs_replay.o           \
1422         zfs_rlock.o            \
1423         zfs_vfsops.o           \
1424         zfs_vnops.o            \
1425         zvol.o

1427 ZUT_OBJS +=                    \
1428         zut.o

1430 #
1431 #                       streams modules
1432 #
1433 BUFMOD_OBJS     +=      bufmod.o

1435 CONNLD_OBJS +=  connld.o

1437 DEDUMP_OBJS +=  dedump.o

1439 DRCOMPAT_OBJS +=        drcompat.o

1441 LDLINUX_OBJS += ldlinux.o

1443 LDTERM_OBJS +=  ldterm.o uwidth.o

1445 PCKT_OBJS +=    pckt.o

1447 PFMOD_OBJS +=   pfmod.o
```

```
1449 PTEM_OBJS +=    ptem.o

1451 REDIRMOD_OBJS += strredirm.o

1453 TIMOD_OBJS +=   timod.o

1455 TIRDWR_OBJS +=  tirdwr.o

1457 TTCOMPAT_OBJS +=ttcompat.o

1459 LOG_OBJS +=     log.o

1461 PIPEMOD_OBJS += pipemod.o

1463 RPCMOD_OBJS +=  rpcmod.o        clnt_cots.o     clnt_clts.o \
1464                 clnt_gen.o      clnt_perr.o     mt_rpcinit.o    rpc_calmsg.o \
1465                 rpc_prot.o      rpc_sztypes.o   rpc_subr.o      rpcb_prot.o \
1466                 svc.o           svc_clts.o      svc_gen.o       svc_cots.o \
1467                 rpcsys.o        xdr_sizeof.o    clnt_rdma.o     svc_rdma.o \
1468                 xdr_rdma.o      rdma_subr.o     xdrrdma_sizeof.o

1470 TLIMOD_OBJS +=  tlimod.o        t_kalloc.o      t_kbind.o       t_kclose.o \
1471                 t_kconnect.o    t_kfree.o       t_kgtstate.o    t_kopen.o \
1472                 t_krcvudat.o    t_ksndudat.o    t_kspoll.o      t_kunbind.o \
1473                 t_kutil.o

1475 RLMOD_OBJS += rlmod.o

1477 TELMOD_OBJS += telmod.o

1479 CRYPTMOD_OBJS += cryptmod.o

1481 KB_OBJS +=      kbd.o           keytables.o

1483 #
1484 #                       ID mapping module
1485 #
1486 IDMAP_OBJS +=   idmap_mod.o     idmap_kapi.o    idmap_xdr.o     idmap_cache.o

1488 #
1489 #                       scheduling class modules
1490 #
1491 SDC_OBJS +=             sysdc.o

1493 RT_OBJS +=              rt.o
1494 RT_DPTBL_OBJS +=        rt_dptbl.o

1496 TS_OBJS +=              ts.o
1497 TS_DPTBL_OBJS +=        ts_dptbl.o

1499 IA_OBJS +=              ia.o

1501 FSS_OBJS +=             fss.o

1503 FX_OBJS +=              fx.o
1504 FX_DPTBL_OBJS +=        fx_dptbl.o

1506 #
1507 #                       Inter-Process Communication (IPC) modules
1508 #
1509 IPC_OBJS +=     ipc.o

1511 IPCMSG_OBJS +=  msg.o

1513 IPCSEM_OBJS +=  sem.o
```



```
1515 IPCSHM_OBJS +=  shm.o

1517 #
1518 #                       bignum module
1519 #
1520 COMMON_BIGNUM_OBJS += bignum_mod.o bignumimpl.o

1522 BIGNUM_OBJS += $(COMMON_BIGNUM_OBJS) $(BIGNUM_PSR_OBJS)

1524 #
1525 #                       kernel cryptographic framework
1526 #
1527 KCF_OBJS +=     kcf.o kcf_callprov.o kcf_cbufcall.o kcf_cipher.o kcf_crypto.o \
1528                 kcf_cryptoadm.o kcf_ctxops.o kcf_digest.o kcf_dual.o \
1529                 kcf_keys.o kcf_mac.o kcf_mech_tabs.o kcf_miscapi.o \
1530                 kcf_object.o kcf_policy.o kcf_prov_lib.o kcf_prov_tabs.o \
1531                 kcf_sched.o kcf_session.o kcf_sign.o kcf_spi.o kcf_verify.o \
1532                 kcf_random.o modes.o ecb.o cbc.o ctr.o ccm.o gcm.o \
1533                 fips_random.o

1535 CRYPTOADM_OBJS += cryptoadm.o

1537 CRYPTO_OBJS +=  crypto.o

1539 DPROV_OBJS +=   dprov.o

1541 DCA_OBJS +=     dca.o dca_3des.o dca_debug.o dca_dsa.o dca_kstat.o dca_rng.o \
1542                 dca_rsa.o

1544 AESPROV_OBJS += aes.o aes_impl.o aes_modes.o

1546 ARCFOURPROV_OBJS += arcfour.o arcfour_crypt.o

1548 BLOWFISHPROV_OBJS += blowfish.o blowfish_impl.o

1550 ECCPROV_OBJS += ecc.o ec.o ec2_163.o ec2_mont.o ecdecode.o ecl_mult.o \
1551                 ecp_384.o ecp_jac.o ec2_193.o ecl.o ecp_192.o ecp_521.o \
1552                 ecp_jm.o ec2_233.o ecl_curve.o ecp_224.o ecp_aff.o \
1553                 ecp_mont.o ec2_aff.o ec_naf.o ecl_gf.o ecp_256.o mp_gf2m.o \
1554                 mpi.o mplogic.o mpmontg.o mpprime.o oid.o \
1555                 secitem.o ec2_test.o ecp_test.o

1557 RSAPROV_OBJS += rsa.o rsa_impl.o pkcs1.o

1559 SWRANDPROV_OBJS += swrand.o

1561 #
1562 #                       kernel SSL
1563 #
1564 KSSL_OBJS +=    kssl.o kssl_ioctl.o

1566 KSSL_SOCKFIL_MOD_OBJS += ksslfilter.o ksslapi.o ksslrec.o

1568 #
1569 #                       misc. modules
1570 #

1572 C2AUDIT_OBJS += adr.o audit.o audit_event.o audit_io.o \
1573                 audit_path.o audit_start.o audit_syscalls.o audit_token.o \
1574                 audit_mem.o

1576 PCIC_OBJS +=    pcic.o

1578 RPCSEC_OBJS +=  secmod.o        sec_clnt.o      sec_svc.o       sec_gen.o \
1579                 auth_des.o      auth_kern.o     auth_none.o     auth_loopb.o\
```

```
1580                 authdesprt.o    authdesubr.o    authu_prot.o \
1581                 key_call.o      key_prot.o      svc_authu.o     svcauthdes.o

1583 RPCSEC_GSS_OBJS +=      rpcsec_gssmod.o rpcsec_gss.o rpcsec_gss_misc.o \
1584                 rpcsec_gss_utils.o svc_rpcsec_gss.o

1586 CONSCONFIG_OBJS += consconfig.o

1588 CONSCONFIG_DACF_OBJS  += consconfig_dacf.o consplat.o

1590 TEM_OBJS += tem.o tem_safe.o 6x10.o 7x14.o 12x22.o

1592 KBTRANS_OBJS +=                          \
1593                 kbtrans.o               \
1594                 kbtrans_keytables.o     \
1595                 kbtrans_polled.o        \
1596                 kbtrans_streams.o       \
1597                 usb_keytables.o

1599 KGSSD_OBJS +=   gssd_clnt_stubs.o gssd_handle.o gssd_prot.o \
1600                 gss_display_name.o gss_release_name.o gss_import_name.o \
1601                 gss_release_buffer.o gss_release_oid_set.o gen_oids.o gssdmod.o

1603 KGSSD_DERIVED_OBJS = gssd_xdr.o

1605 KGSS_DUMMY_OBJS += dmech.o

1607 KSOCKET_OBJS += ksocket.o ksocket_mod.o

1609 CRYPTO= cksumtypes.o decrypt.o encrypt.o encrypt_length.o etypes.o \
1610         nfold.o verify_checksum.o prng.o block_size.o make_checksum.o\
1611         checksum_length.o hmac.o default_state.o mandatory_sumtype.o

1613 # crypto/des
1614 CRYPTO_DES= f_cbc.o f_cksum.o f_parity.o weak_key.o d3_cbc.o ef_crypto.o

1616 CRYPTO_DK= checksum.o derive.o dk_decrypt.o dk_encrypt.o

1618 CRYPTO_ARCFOUR= k5_arcfour.o

1620 # crypto/enc_provider
1621 CRYPTO_ENC= des.o des3.o arcfour_provider.o aes_provider.o

1623 # crypto/hash_provider
1624 CRYPTO_HASH= hash_kef_generic.o hash_kmd5.o hash_crc32.o hash_ksha1.o

1626 # crypto/keyhash_provider
1627 CRYPTO_KEYHASH= descbc.o k5_kmd5des.o k_hmac_md5.o

1629 # crypto/crc32
1630 CRYPTO_CRC32= crc32.o

1632 # crypto/old
1633 CRYPTO_OLD= old_decrypt.o old_encrypt.o

1635 # crypto/raw
1636 CRYPTO_RAW= raw_decrypt.o raw_encrypt.o

1638 K5_KRB= kfree.o copy_key.o \
1639         parse.o init_ctx.o \
1640         ser_adata.o ser_addr.o \
1641         ser_auth.o ser_cksum.o \
1642         ser_key.o ser_princ.o \
1643         serialize.o unparse.o \
1644         ser_actx.o
```



```
1646 K5_OS=  timeofday.o toffset.o \
1647         init_os_ctx.o c_ustime.o

1649 SEAL=   seal.o unseal.o

1651 MECH=   delete_sec_context.o \
1652         import_sec_context.o \
1653         gssapi_krb5.o \
1654         k5seal.o k5unseal.o k5sealv3.o \
1655         ser_sctx.o \
1656         sign.o \
1657         util_crypt.o  \
1658         util_validate.o  util_ordering.o  \
1659         util_seqnum.o util_set.o util_seed.o \
1660         wrap_size_limit.o verify.o



1664 MECH_GEN= util_token.o


1667 KGSS_KRB5_OBJS += krb5mech.o \
1668         $(MECH) $(SEAL) $(MECH_GEN) \
1669         $(CRYPTO) $(CRYPTO_DES) $(CRYPTO_DK) $(CRYPTO_ARCFOUR) \
1670         $(CRYPTO_ENC) $(CRYPTO_HASH) \
1671         $(CRYPTO_KEYHASH) $(CRYPTO_CRC32) \
1672         $(CRYPTO_OLD) \
1673         $(CRYPTO_RAW) $(K5_KRB) $(K5_OS)

1675 DES_OBJS +=     des_crypt.o des_impl.o des_ks.o des_soft.o

1677 DLBOOT_OBJS +=  bootparam_xdr.o nfs_dlinet.o scan.o

1679 KRTLD_OBJS +=   kobj_bootflags.o getoptstr.o \
1680                 kobj.o kobj_kdi.o kobj_lm.o kobj_subr.o

1682 MOD_OBJS +=     modctl.o modsubr.o modsysfile.o modconf.o modhash.o

1684 STRPLUMB_OBJS += strplumb.o

1686 CPR_OBJS +=     cpr_driver.o cpr_dump.o \
1687                 cpr_main.o cpr_misc.o cpr_mod.o cpr_stat.o \
1688                 cpr_uthread.o

1690 PROF_OBJS +=    prf.o

1692 SE_OBJS += se_driver.o

1694 SYSACCT_OBJS += acct.o

1696 ACCTCTL_OBJS += acctctl.o

1698 EXACCTSYS_OBJS += exacctsys.o

1700 KAIO_OBJS += aio.o

1702 PCMCIA_OBJS += pcmcia.o cs.o cis.o cis_callout.o cis_handlers.o cis_params.o

1704 BUSRA_OBJS += busra.o

1706 PCS_OBJS += pcs.o

1708 PCAN_OBJS += pcan.o

1710 PCATA_OBJS += pcide.o pcdisk.o pclabel.o pcata.o
```

```
1712 PCSER_OBJS += pcser.o pcser_cis.o

1714 PCWL_OBJS += pcwl.o

1716 PSET_OBJS +=    pset.o

1718 OHCI_OBJS += ohci.o ohci_hub.o ohci_polled.o

1720 UHCI_OBJS += uhci.o uhciutil.o uhcitgt.o uhcihub.o uhcipolled.o

1722 EHCI_OBJS += ehci.o ehci_hub.o ehci_xfer.o ehci_intr.o ehci_util.o ehci_polled.o

1724 HUBD_OBJS += hubd.o

1726 USB_MID_OBJS += usb_mid.o

1728 USB_IA_OBJS += usb_ia.o

1730 UWBA_OBJS += uwba.o uwbai.o

1732 SCSA2USB_OBJS += scsa2usb.o usb_ms_bulkonly.o usb_ms_cbi.o

1734 HWAHC_OBJS += hwahc.o hwahc_util.o

1736 WUSB_DF_OBJS += wusb_df.o
1737 WUSB_FWMOD_OBJS += wusb_fwmod.o

1739 IPF_OBJS += ip_fil_solaris.o fil.o solaris.o ip_state.o ip_frag.o ip_nat.o \
1740             ip_proxy.o ip_auth.o ip_pool.o ip_htable.o ip_lookup.o \
1741             ip_log.o misc.o ip_compat.o ip_nat6.o drand48.o

1743 IBD_OBJS +=     ibd.o ibd_cm.o

1745 EIBNX_OBJS +=   enx_main.o enx_hdlrs.o enx_ibt.o enx_log.o enx_fip.o \
1746                 enx_misc.o enx_q.o enx_ctl.o

1748 EOIB_OBJS +=    eib_adm.o eib_chan.o eib_cmn.o eib_ctl.o eib_data.o \
1749                 eib_fip.o eib_ibt.o eib_log.o eib_mac.o eib_main.o \
1750                 eib_rsrc.o eib_svc.o eib_vnic.o

1752 DLPISTUB_OBJS += dlpistub.o

1754 SDP_OBJS +=     sdpddi.o

1756 TRILL_OBJS +=   trill.o

1758 CTF_OBJS += ctf_create.o ctf_decl.o ctf_error.o ctf_hash.o ctf_labels.o \
1759         ctf_lookup.o ctf_open.o ctf_types.o ctf_util.o ctf_subr.o ctf_mod.o

1761 SMBIOS_OBJS += smb_error.o smb_info.o smb_open.o smb_subr.o smb_dev.o

1763 RPCIB_OBJS += rpcib.o

1765 KMDB_OBJS += kdrv.o

1767 AFE_OBJS += afe.o

1769 BGE_OBJS += bge_main2.o bge_chip2.o bge_kstats.o bge_log.o bge_ndd.o \
1770                 bge_atomic.o bge_mii.o bge_send.o bge_recv2.o bge_mii_5906.o

1772 DMFE_OBJS += dmfe_log.o dmfe_main.o dmfe_mii.o

1774 EFE_OBJS += efe.o

1776 ELXL_OBJS += elxl.o
```



```
1778 HME_OBJS += hme.o

1780 IXGB_OBJS += ixgb.o ixgb_atomic.o ixgb_chip.o ixgb_gld.o ixgb_kstats.o \
1781                 ixgb_log.o ixgb_ndd.o ixgb_rx.o ixgb_tx.o ixgb_xmii.o

1783 NGE_OBJS += nge_main.o nge_atomic.o nge_chip.o nge_ndd.o nge_kstats.o \
1784                 nge_log.o nge_rx.o nge_tx.o nge_xmii.o

1786 PCN_OBJS += pcn.o

1788 RGE_OBJS += rge_main.o rge_chip.o rge_ndd.o rge_kstats.o rge_log.o rge_rxtx.o

1790 SFXGE_OBJS += efx_bootcfg.o efx_ev.o efx_intr.o efx_mac.o efx_mcdi.o efx_mon.o \
1791                   efx_nic.o efx_nvram.o efx_phy.o efx_port.o efx_rx.o efx_sram.o
1792                   efx_vpd.o efx_wol.o falcon_gmac.o falcon_i2c.o falcon_mac.o fa
1793                   falcon_nic.o falcon_nvram.o falcon_spi.o falcon_sram.o falcon_
1794                   falcon_xmac.o lm87.o max6647.o nullmon.o nullphy.o qt2022c2.o
1795                   sft9001.o sfx7101.o sfxge_bar.o sfxge_dma.o sfxge_err.o sfxge_
1796                   sfxge_gld_ndd.o sfxge_gld_v2.o sfxge_gld_v3.o sfxge_intr.o sfx
1797                   sfxge_mcdi.o sfxge_mon.o sfxge_nvram.o sfxge_pci.o sfxge_phy.o
1798                   sfxge_sram.o sfxge_tcp.o sfxge_tx.o sfxge_vpd.o sfxge.o siena_
1799                   siena_mon.o siena_nic.o siena_nvram.o siena_phy.o siena_sram.o
1800                   txc43128.o xphy.o

1802 #endif /* ! codereview */
1803 URTW_OBJS += urtw.o

1805 ARN_OBJS += arn_hw.o arn_eeprom.o arn_mac.o arn_calib.o arn_ani.o arn_phy.o arn_
1806                 arn_main.o arn_recv.o arn_xmit.o arn_rc.o

1808 ATH_OBJS += ath_aux.o ath_main.o ath_osdep.o ath_rate.o

1810 ATU_OBJS += atu.o

1812 IPW_OBJS += ipw2100_hw.o ipw2100.o

1814 IWI_OBJS += ipw2200_hw.o ipw2200.o

1816 IWH_OBJS += iwh.o

1818 IWK_OBJS += iwk2.o

1820 IWP_OBJS += iwp.o

1822 MWL_OBJS += mwl.o

1824 MWLFW_OBJS += mwlfw_mode.o

1826 WPI_OBJS += wpi.o

1828 RAL_OBJS += rt2560.o ral_rate.o

1830 RUM_OBJS += rum.o

1832 RWD_OBJS += rt2661.o

1834 RWN_OBJS += rt2860.o

1836 UATH_OBJS += uath.o

1838 UATHFW_OBJS += uathfw_mod.o

1840 URAL_OBJS += ural.o

1842 RTW_OBJS += rtw.o smc93cx6.o rtwphy.o rtwphyio.o
```

```
ZYD_OBJS += zyd.o zyd_usb.o zyd_hw.o zyd_fw.o

MXFE_OBJS += mxfe.o

MPTSAS_OBJS += mptsas.o mptsas_impl.o mptsas_init.o mptsas_raid.o mptsas_smhba.o

SFE_OBJS += sfe.o sfe_util.o

BFE_OBJS += bfe.o

BRIDGE_OBJS += bridge.o

IDM_SHARED_OBJS += base64.o

IDM_OBJS +=     $(IDM_SHARED_OBJS) \
                idm.o idm_impl.o idm_text.o idm_conn_sm.o idm_so.o

VR_OBJS += vr.o

ATGE_OBJS += atge_main.o atge_l1e.o atge_mii.o atge_l1.o atge_l1c.o

YGE_OBJS = yge.o

#
#       Build up defines and paths.
#
LINT_DEFS       += -Dunix

#
#       This duality can be removed when the native and target compilers
#       are the same (or at least recognize the same command line syntax!)
#       It is a bug in the current compilation system that the assember
#       can't process the -Y I, flag.
#
NATIVE_INC_PATH += $(INC_PATH) $(CCYFLAG)$(UTSBASE)/common
AS_INC_PATH     += $(INC_PATH) -I$(UTSBASE)/common
INCLUDE_PATH    += $(INC_PATH) $(CCYFLAG)$(UTSBASE)/common

PCIEB_OBJS += pcieb.o

#       Chelsio N110 10G NIC driver module
#
CH_OBJS = ch.o glue.o pe.o sge.o

CH_COM_OBJS =   ch_mac.o ch_subr.o cspi.o espi.o ixf1010.o mc3.o mc4.o mc5.o \
                mv88e1xxx.o mv88x201x.o my3126.o pm3393.o tp.o ulp.o \
                vsc7321.o vsc7326.o xpak.o

#
#       Chelsio Terminator 4 10G NIC nexus driver module
#
CXGBE_FW_OBJS  =        t4_fw.o t4_cfg.o
CXGBE_COM_OBJS =        t4_hw.o common.o
CXGBE_NEX_OBJS =        t4_nexus.o t4_sge.o t4_mac.o t4_ioctl.o shared.o \
                        t4_l2t.o adapter.o osdep.o

#
#       Chelsio Terminator 4 10G NIC driver module
#
CXGBE_OBJS =    cxgbe.o

#
#       PCI strings file
#
PCI_STRING_OBJS = pci_strings.o
```



```
NET_DACF_OBJS += net_dacf.o

#
#       Xframe 10G NIC driver module
#
XGE_OBJS = xge.o xgell.o

XGE_HAL_OBJS =  xgehal-channel.o xgehal-fifo.o xgehal-ring.o  xgehal-config.o \
                xgehal-driver.o  xgehal-mm.o xgehal-stats.o  xgehal-device.o \
                xge-queue.o  xgehal-mgmt.o xgehal-mgmtaux.o

#
#       e1000g module
#
E1000G_OBJS +=  e1000_80003es2lan.o e1000_82540.o e1000_82541.o e1000_82542.o \
                e1000_82543.o e1000_82571.o e1000_api.o e1000_ich8lan.o \
                e1000_mac.o e1000_manage.o e1000_nvm.o e1000_osdep.o \
                e1000_phy.o e1000g_debug.o e1000g_main.o e1000g_alloc.o \
                e1000g_tx.o e1000g_rx.o e1000g_stat.o

#
#       Intel 82575 1G NIC driver module
#
IGB_OBJS =      igb_82575.o igb_api.o igb_mac.o igb_manage.o \
                igb_nvm.o igb_osdep.o igb_phy.o igb_buf.o \
                igb_debug.o igb_gld.o igb_log.o igb_main.o \
                igb_rx.o igb_stat.o igb_tx.o

#
#       Intel Pro/100 NIC driver module
#
IPRB_OBJS =     iprb.o

#
#       Intel 10GbE PCIE NIC driver module
#
IXGBE_OBJS =    ixgbe_82598.o ixgbe_82599.o ixgbe_api.o         \
                ixgbe_common.o ixgbe_phy.o                      \
                ixgbe_buf.o ixgbe_debug.o ixgbe_gld.o           \
                ixgbe_log.o ixgbe_main.o                        \
                ixgbe_osdep.o ixgbe_rx.o ixgbe_stat.o           \
                ixgbe_tx.o  ixgbe_x540.o ixgbe_mbx.o

#
#       NIU 10G/1G driver module
#
NXGE_OBJS =     nxge_mac.o nxge_ipp.o nxge_rxdma.o              \
                nxge_txdma.o nxge_txc.o nxge_main.o             \
                nxge_hw.o nxge_fzc.o nxge_virtual.o             \
                nxge_send.o nxge_classify.o nxge_fflp.o         \
                nxge_fflp_hash.o nxge_ndd.o nxge_kstats.o       \
                nxge_zcp.o nxge_fm.o nxge_espc.o nxge_hv.o      \
                nxge_hio.o nxge_hio_guest.o nxge_intr.o

NXGE_NPI_OBJS = \
                npi.o npi_mac.o npi_ipp.o                       \
                npi_txdma.o npi_rxdma.o npi_txc.o               \
                npi_zcp.o npi_espc.o npi_fflp.o                 \
                npi_vir.o

NXGE_HCALL_OBJS =       \
                nxge_hcall.o

#
# Virtio modules
#
```

```
# Virtio core
VIRTIO_OBJS = virtio.o

# Virtio block driver
VIOBLK_OBJS = vioblk.o

#
#       kiconv modules
#
KICONV_EMEA_OBJS += kiconv_emea.o

KICONV_JA_OBJS += kiconv_ja.o

KICONV_KO_OBJS += kiconv_cck_common.o kiconv_ko.o

KICONV_SC_OBJS += kiconv_cck_common.o kiconv_sc.o

KICONV_TC_OBJS += kiconv_cck_common.o kiconv_tc.o

#
#       AAC module
#
AAC_OBJS = aac.o aac_ioctl.o

#
#       sdcard modules
#
SDA_OBJS =      sda_cmd.o sda_host.o sda_init.o sda_mem.o sda_mod.o sda_slot.o
SDHOST_OBJS =   sdhost.o

#
#       hxge 10G driver module
#
HXGE_OBJS =     hxge_main.o hxge_vmac.o hxge_send.o             \
                hxge_txdma.o hxge_rxdma.o hxge_virtual.o        \
                hxge_fm.o hxge_fzc.o hxge_hw.o hxge_kstats.o    \
                hxge_ndd.o hxge_pfc.o                           \
                hpi.o hpi_vmac.o hpi_rxdma.o hpi_txdma.o        \
                hpi_vir.o hpi_pfc.o

#
#       MEGARAID_SAS module
#
MEGA_SAS_OBJS = megaraid_sas.o

#
#       MR_SAS module
#
MR_SAS_OBJS = ld_pd_map.o mr_sas.o mr_sas_tbolt.o mr_sas_list.o

#
#       ISCSI_INITIATOR module
#
ISCSI_INITIATOR_OBJS =  chap.o iscsi_io.o iscsi_thread.o        \
                        iscsi_ioctl.o iscsid.o iscsi.o          \
                        iscsi_login.o isns_client.o iscsiAuthClient.o   \
                        iscsi_lun.o iscsiAuthClientGlue.o       \
                        iscsi_net.o nvfile.o iscsi_cmd.o        \
                        iscsi_queue.o persistent.o iscsi_conn.o \
                        iscsi_sess.o radius_auth.o iscsi_crc.o  \
                        iscsi_stats.o radius_packet.o iscsi_doorclt.o   \
                        iscsi_targetparam.o utils.o kifconf.o

#
#       ntxn 10Gb/1Gb NIC driver module
```



```
#
NTXN_OBJS =     unm_nic_init.o unm_gem.o unm_nic_hw.o unm_ndd.o \
                        unm_nic_main.o unm_nic_isr.o unm_nic_ctx.o niu.o

#
#       Myricom 10Gb NIC driver module
#
MYRI10GE_OBJS = myri10ge.o myri10ge_lro.o

#       nulldriver module
#
NULLDRIVER_OBJS =       nulldriver.o

TPM_OBJS =      tpm.o tpm_hcall.o
```

```
**********************************************************
   73708 Thu Aug 22 18:59:21 2013
new/usr/src/uts/common/Makefile.rules
Merged sfxge driver
**********************************************************
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#

#
# Copyright (c) 1991, 2010, Oracle and/or its affiliates. All rights reserved.
# Copyright (c) 2012 Nexenta Systems, Inc. All rights reserved.
#

#
# uts/common/Makefile.rules
#
#       This Makefile defines all the file build rules for the directory
# uts/common and its children. These are the source files which may
# be considered common to all SunOS systems.
#
#       The following two-level ordering must be maintained in this file.
#         Lines are sorted first in order of decreasing specificity based on
#         the first directory component.  That is, sun4u rules come before
#         sparc rules come before common rules.
#
#         Lines whose initial directory components are equal are sorted
#         alphabetically by the remaining components.

#
#       Section 1a: C objects build rules
#
$(OBJS_DIR)/%.o:                $(COMMONBASE)/crypto/aes/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(COMMONBASE)/crypto/arcfour/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(COMMONBASE)/crypto/blowfish/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(COMMONBASE)/crypto/ecc/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(COMMONBASE)/crypto/modes/%.c
```



```
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(COMMONBASE)/crypto/padding/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(COMMONBASE)/crypto/rng/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(COMMONBASE)/crypto/rsa/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(COMMONBASE)/bignum/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/common/bignum/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(COMMONBASE)/mpi/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(COMMONBASE)/acl/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(COMMONBASE)/avl/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(COMMONBASE)/ucode/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/common/brand/sn1/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/common/brand/solaris10/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/common/c2/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/common/conf/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/common/contract/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/common/cpr/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/common/ctf/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)
```

```
129 $(OBJS_DIR)/%.o:                $(COMMONBASE)/ctf/%.c
130         $(COMPILE.c) -o $@ $<
131         $(CTFCONVERT_O)

133 $(OBJS_DIR)/%.o:                $(COMMONBASE)/crypto/des/%.c
134         $(COMPILE.c) -o $@ $<
135         $(CTFCONVERT_O)

137 $(OBJS_DIR)/%.o:                $(COMMONBASE)/smbios/%.c
138         $(COMPILE.c) -o $@ $<
139         $(CTFCONVERT_O)

141 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/des/%.c
142         $(COMPILE.c) -o $@ $<
143         $(CTFCONVERT_O)

145 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/crypto/api/%.c
146         $(COMPILE.c) -o $@ $<
147         $(CTFCONVERT_O)

149 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/crypto/core/%.c
150         $(COMPILE.c) -o $@ $<
151         $(CTFCONVERT_O)

153 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/crypto/io/%.c
154         $(COMPILE.c) -o $@ $<
155         $(CTFCONVERT_O)

157 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/crypto/spi/%.c
158         $(COMPILE.c) -o $@ $<
159         $(CTFCONVERT_O)

161 $(OBJS_DIR)/%.o:                $(COMMONBASE)/pci/%.c
162         $(COMPILE.c) -o $@ $<
163         $(CTFCONVERT_O)

165 $(OBJS_DIR)/%.o:                $(COMMONBASE)/devid/%.c
166         $(COMPILE.c) -o $@ $<
167         $(CTFCONVERT_O)

169 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/disp/%.c
170         $(COMPILE.c) -o $@ $<
171         $(CTFCONVERT_O)

173 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/dtrace/%.c
174         $(COMPILE.c) -o $@ $<
175         $(CTFCONVERT_O)

177 $(OBJS_DIR)/%.o:                $(COMMONBASE)/exacct/%.c
178         $(COMPILE.c) -o $@ $<
179         $(CTFCONVERT_O)

181 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/exec/aout/%.c
182         $(COMPILE.c) -o $@ $<
183         $(CTFCONVERT_O)

185 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/exec/elf/%.c
186         $(COMPILE.c) -o $@ $<
187         $(CTFCONVERT_O)

189 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/exec/intp/%.c
190         $(COMPILE.c) -o $@ $<
191         $(CTFCONVERT_O)

193 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/exec/shbin/%.c
```



```
194         $(COMPILE.c) -o $@ $<
195         $(CTFCONVERT_O)

197 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/exec/java/%.c
198         $(COMPILE.c) -o $@ $<
199         $(CTFCONVERT_O)

201 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/fs/%.c
202         $(COMPILE.c) -o $@ $<
203         $(CTFCONVERT_O)

205 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/fs/autofs/%.c
206         $(COMPILE.c) -o $@ $<
207         $(CTFCONVERT_O)

209 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/fs/cachefs/%.c
210         $(COMPILE.c) -o $@ $<
211         $(CTFCONVERT_O)

213 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/fs/dcfs/%.c
214         $(COMPILE.c) -o $@ $<
215         $(CTFCONVERT_O)

217 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/fs/devfs/%.c
218         $(COMPILE.c) -o $@ $<
219         $(CTFCONVERT_O)

221 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/fs/ctfs/%.c
222         $(COMPILE.c) -o $@ $<
223         $(CTFCONVERT_O)

225 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/fs/doorfs/%.c
226         $(COMPILE.c) -o $@ $<
227         $(CTFCONVERT_O)

229 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/fs/dev/%.c
230         $(COMPILE.c) -o $@ $<
231         $(CTFCONVERT_O)

233 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/fs/fd/%.c
234         $(COMPILE.c) -o $@ $<
235         $(CTFCONVERT_O)

237 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/fs/fifofs/%.c
238         $(COMPILE.c) -o $@ $<
239         $(CTFCONVERT_O)

241 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/fs/hsfs/%.c
242         $(COMPILE.c) -o $@ $<
243         $(CTFCONVERT_O)

245 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/fs/lofs/%.c
246         $(COMPILE.c) -o $@ $<
247         $(CTFCONVERT_O)

249 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/fs/mntfs/%.c
250         $(COMPILE.c) -o $@ $<
251         $(CTFCONVERT_O)

253 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/fs/namefs/%.c
254         $(COMPILE.c) -o $@ $<
255         $(CTFCONVERT_O)

257 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/fs/nfs/%.c
258         $(COMPILE.c) -o $@ $<
259         $(CTFCONVERT_O)
```

```
261 $(OBJS_DIR)/%.o:                $(COMMONBASE)/smbsrv/%.c
262         $(COMPILE.c) -o $@ $<
263         $(CTFCONVERT_O)

265 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/fs/smbsrv/%.c
266         $(COMPILE.c) -o $@ $<
267         $(CTFCONVERT_O)

269 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/fs/objfs/%.c
270         $(COMPILE.c) -o $@ $<
271         $(CTFCONVERT_O)

273 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/fs/pcfs/%.c
274         $(COMPILE.c) -o $@ $<
275         $(CTFCONVERT_O)

277 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/fs/portfs/%.c
278         $(COMPILE.c) -o $@ $<
279         $(CTFCONVERT_O)

281 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/fs/proc/%.c
282         $(COMPILE.c) -o $@ $<
283         $(CTFCONVERT_O)

285 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/fs/sharefs/%.c
286         $(COMPILE.c) -o $@ $<
287         $(CTFCONVERT_O)

289 $(OBJS_DIR)/%.o:                $(COMMONBASE)/smbclnt/%.c
290         $(COMPILE.c) -o $@ $<
291         $(CTFCONVERT_O)

293 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/fs/smbclnt/netsmb/%.c
294         $(COMPILE.c) -o $@ $<
295         $(CTFCONVERT_O)

297 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/fs/smbclnt/smbfs/%.c
298         $(COMPILE.c) -o $@ $<
299         $(CTFCONVERT_O)

301 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/fs/sockfs/%.c
302         $(COMPILE.c) -o $@ $<
303         $(CTFCONVERT_O)

305 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/fs/specfs/%.c
306         $(COMPILE.c) -o $@ $<
307         $(CTFCONVERT_O)

309 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/fs/swapfs/%.c
310         $(COMPILE.c) -o $@ $<
311         $(CTFCONVERT_O)

313 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/fs/tmpfs/%.c
314         $(COMPILE.c) -o $@ $<
315         $(CTFCONVERT_O)

317 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/fs/udfs/%.c
318         $(COMPILE.c) -o $@ $<
319         $(CTFCONVERT_O)

321 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/fs/ufs/%.c
322         $(COMPILE.c) -o $@ $<
323         $(CTFCONVERT_O)

325 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/vscan/%.c
```



```
326         $(COMPILE.c) -o $@ $<
327         $(CTFCONVERT_O)

329 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/fs/zfs/%.c
330         $(COMPILE.c) -o $@ $<
331         $(CTFCONVERT_O)

333 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/fs/zut/%.c
334         $(COMPILE.c) -o $@ $<
335         $(CTFCONVERT_O)

337 $(OBJS_DIR)/%.o:                $(COMMONBASE)/xattr/%.c
338         $(COMPILE.c) -o $@ $<
339         $(CTFCONVERT_O)

341 $(OBJS_DIR)/%.o:                $(COMMONBASE)/zfs/%.c
342         $(COMPILE.c) -o $@ $<
343         $(CTFCONVERT_O)

345 $(OBJS_DIR)/%.o:        $(UTSBASE)/common/io/scsi/adapters/pmcs/%.c
346         $(COMPILE.c) -o $@ $<
347         $(CTFCONVERT_O)

349 $(OBJS_DIR)/%.o:        $(UTSBASE)/common/io/scsi/adapters/pmcs/%.bin
350         $(COMPILE.b) -o $@ $<
351         $(CTFCONVERT_O)

353 $(OBJS_DIR)/%.o:                $(COMMONBASE)/fsreparse/%.c
354         $(COMPILE.c) -o $@ $<
355         $(CTFCONVERT_O)

357 KMECHKRB5_BASE=$(UTSBASE)/common/gssapi/mechs/krb5

359 KGSSDFLAGS=-I $(UTSBASE)/common/gssapi/include

361 # Note, KRB5_DEFS can be assigned various preprocessor flags,
362 # typically -D defines on the make invocation.  The standard compiler
363 # flags will not be overwritten.
364 KGSSDFLAGS += $(KRB5_DEFS)

366 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/gssapi/%.c
367         $(COMPILE.c) $(KGSSDFLAGS) -o $@ $<
368         $(CTFCONVERT_O)

370 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/gssapi/mechs/dummy/%.c
371         $(COMPILE.c) $(KGSSDFLAGS) -o $@ $<
372         $(CTFCONVERT_O)

374 $(OBJS_DIR)/%.o:                $(KMECHKRB5_BASE)/%.c
375         $(COMPILE.c) $(KGSSDFLAGS) -o $@ $<
376         $(CTFCONVERT_O)

378 $(OBJS_DIR)/%.o:                $(KMECHKRB5_BASE)/crypto/%.c
379         $(COMPILE.c) $(KGSSDFLAGS) -o $@ $<
380         $(CTFCONVERT_O)

382 $(OBJS_DIR)/%.o:                $(KMECHKRB5_BASE)/crypto/des/%.c
383         $(COMPILE.c) $(KGSSDFLAGS) -o $@ $<
384         $(CTFCONVERT_O)

386 $(OBJS_DIR)/%.o:                $(KMECHKRB5_BASE)/crypto/arcfour/%.c
387         $(COMPILE.c) $(KGSSDFLAGS) -o $@ $<
388         $(CTFCONVERT_O)

390 $(OBJS_DIR)/%.o:                $(KMECHKRB5_BASE)/crypto/dk/%.c
391         $(COMPILE.c) $(KGSSDFLAGS) -o $@ $<
```

```
392         $(CTFCONVERT_O)

394 $(OBJS_DIR)/%.o:                $(KMECHKRB5_BASE)/crypto/enc_provider/%.c
395         $(COMPILE.c) $(KGSSDFLAGS) -o $@ $<
396         $(CTFCONVERT_O)

398 $(OBJS_DIR)/%.o:                $(KMECHKRB5_BASE)/crypto/hash_provider/%.c
399         $(COMPILE.c) $(KGSSDFLAGS) -o $@ $<
400         $(CTFCONVERT_O)

402 $(OBJS_DIR)/%.o:                $(KMECHKRB5_BASE)/crypto/keyhash_provider/%.c
403         $(COMPILE.c) $(KGSSDFLAGS) -o $@ $<
404         $(CTFCONVERT_O)

406 $(OBJS_DIR)/%.o:                $(KMECHKRB5_BASE)/crypto/raw/%.c
407         $(COMPILE.c) $(KGSSDFLAGS) -o $@ $<
408         $(CTFCONVERT_O)

410 $(OBJS_DIR)/%.o:                $(KMECHKRB5_BASE)/crypto/old/%.c
411         $(COMPILE.c) $(KGSSDFLAGS) -o $@ $<
412         $(CTFCONVERT_O)

414 $(OBJS_DIR)/%.o:                $(KMECHKRB5_BASE)/krb5/krb/%.c
415         $(COMPILE.c) $(KGSSDFLAGS) -o $@ $<
416         $(CTFCONVERT_O)

418 $(OBJS_DIR)/%.o:                $(KMECHKRB5_BASE)/krb5/os/%.c
419         $(COMPILE.c) $(KGSSDFLAGS) -o $@ $<
420         $(CTFCONVERT_O)

422 $(OBJS_DIR)/ser_sctx.o := CPPFLAGS += -DPROVIDE_KERNEL_IMPORT=1

424 $(OBJS_DIR)/%.o:                $(KMECHKRB5_BASE)/mech/%.c
425         $(COMPILE.c) $(KGSSDFLAGS) -o $@ $<
426         $(CTFCONVERT_O)

428 $(OBJS_DIR)/%.o:                $(KMECHKRB5_BASE)/profile/%.c
429         $(COMPILE.c) $(KGSSDFLAGS) -o $@ $<
430         $(CTFCONVERT_O)

432 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/avs/ncall/%.c
433         $(COMPILE.c) -o $@ $<
434         $(CTFCONVERT_O)

436 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/avs/ns/dsw/%.c
437         $(COMPILE.c) -o $@ $<
438         $(CTFCONVERT_O)

440 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/avs/ns/nsctl/%.c
441         $(COMPILE.c) -o $@ $<
442         $(CTFCONVERT_O)

444 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/avs/ns/rdc/%.c
445         $(COMPILE.c) -o $@ $<
446         $(CTFCONVERT_O)

448 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/avs/ns/sdbc/%.c
449         $(COMPILE.c) -o $@ $<
450         $(CTFCONVERT_O)

452 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/avs/ns/solaris/%.c
453         $(COMPILE.c) -o $@ $<
454         $(CTFCONVERT_O)

456 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/avs/ns/sv/%.c
457         $(COMPILE.c) -o $@ $<
```



```
458         $(CTFCONVERT_O)

460 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/avs/ns/unistat/%.c
461         $(COMPILE.c) -o $@ $<
462         $(CTFCONVERT_O)

464 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/idmap/%.c
465         $(COMPILE.c) -o $@ $<
466         $(CTFCONVERT_O)

468 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/inet/%.c
469         $(COMPILE.c) -o $@ $<
470         $(CTFCONVERT_O)

472 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/inet/arp/%.c
473         $(COMPILE.c) -o $@ $<
474         $(CTFCONVERT_O)

476 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/inet/ip/%.c
477         $(COMPILE.c) -o $@ $<
478         $(CTFCONVERT_O)

480 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/inet/ipnet/%.c
481         $(COMPILE.c) -o $@ $<
482         $(CTFCONVERT_O)

484 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/inet/iptun/%.c
485         $(COMPILE.c) -o $@ $<
486         $(CTFCONVERT_O)

488 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/inet/kssl/%.c
489         $(COMPILE.c) -o $@ $<
490         $(CTFCONVERT_O)

492 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/inet/sctp/%.c
493         $(COMPILE.c) -o $@ $<
494         $(CTFCONVERT_O)

496 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/inet/tcp/%.c
497         $(COMPILE.c) -o $@ $<
498         $(CTFCONVERT_O)

500 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/inet/ilb/%.c
501         $(COMPILE.c) -o $@ $<
502         $(CTFCONVERT_O)

504 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/inet/ipf/%.c
505         $(COMPILE.c) -o $@ $<
506         $(CTFCONVERT_O)

508 $(OBJS_DIR)/%.o:                $(COMMONBASE)/net/patricia/%.c
509         $(COMPILE.c) -o $@ $<
510         $(CTFCONVERT_O)

512 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/inet/udp/%.c
513         $(COMPILE.c) -o $@ $<
514         $(CTFCONVERT_O)

516 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/inet/nca/%.c
517         $(COMPILE.c) -o $@ $<
518         $(CTFCONVERT_O)

520 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/inet/sockmods/%.c
521         $(COMPILE.c) -o $@ $<
522         $(CTFCONVERT_O)
```

```
524 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/inet/dlpistub/%.c
525         $(COMPILE.c) -o $@ $<
526         $(CTFCONVERT_O)

528 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/%.c
529         $(COMPILE.c) -o $@ $<
530         $(CTFCONVERT_O)

532 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/1394/%.c
533         $(COMPILE.c) -o $@ $<
534         $(CTFCONVERT_O)

536 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/1394/adapters/%.c
537         $(COMPILE.c) -o $@ $<
538         $(CTFCONVERT_O)

540 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/1394/targets/av1394/%.c
541         $(COMPILE.c) -o $@ $<
542         $(CTFCONVERT_O)

544 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/1394/targets/dcam1394/%.c
545         $(COMPILE.c) -o $@ $<
546         $(CTFCONVERT_O)

548 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/1394/targets/scsa1394/%.c
549         $(COMPILE.c) -o $@ $<
550         $(CTFCONVERT_O)

552 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/sbp2/%.c
553         $(COMPILE.c) -o $@ $<
554         $(CTFCONVERT_O)

556 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/aac/%.c
557         $(COMPILE.c) -o $@ $<
558         $(CTFCONVERT_O)

560 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/afe/%.c
561         $(COMPILE.c) -o $@ $<
562         $(CTFCONVERT_O)

564 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/atge/%.c
565         $(COMPILE.c) -o $@ $<
566         $(CTFCONVERT_O)

568 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/arn/%.c
569         $(COMPILE.c) -o $@ $<
570         $(CTFCONVERT_O)

572 $(OBJS_DIR)/%.o:               $(UTSBASE)/common/io/ath/%.c
573         $(COMPILE.c) -o $@ $<
574         $(CTFCONVERT_O)

576 $(OBJS_DIR)/%.o:               $(UTSBASE)/common/io/atu/%.c
577         $(COMPILE.c) -o $@ $<
578         $(CTFCONVERT_O)

580 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/audio/impl/%.c
581         $(COMPILE.c) -o $@ $<
582         $(CTFCONVERT_O)

584 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/audio/ac97/%.c
585         $(COMPILE.c) -o $@ $<
586         $(CTFCONVERT_O)

588 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/audio/drv/audioens/%.c
589         $(COMPILE.c) -o $@ $<
```



```
590         $(CTFCONVERT_O)

592 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/audio/drv/audioemu10k/%.c
593         $(COMPILE.c) -o $@ $<
594         $(CTFCONVERT_O)

596 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/audio/drv/audio1575/%.c
597         $(COMPILE.c) -o $@ $<
598         $(CTFCONVERT_O)

600 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/audio/drv/audio810/%.c
601         $(COMPILE.c) -o $@ $<
602         $(CTFCONVERT_O)

604 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/audio/drv/audiocmi/%.c
605         $(COMPILE.c) -o $@ $<
606         $(CTFCONVERT_O)

608 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/audio/drv/audiocmihd/%.c
609         $(COMPILE.c) -o $@ $<
610         $(CTFCONVERT_O)

612 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/audio/drv/audiohd/%.c
613         $(COMPILE.c) -o $@ $<
614         $(CTFCONVERT_O)

616 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/audio/drv/audioixp/%.c
617         $(COMPILE.c) -o $@ $<
618         $(CTFCONVERT_O)

620 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/audio/drv/audiols/%.c
621         $(COMPILE.c) -o $@ $<
622         $(CTFCONVERT_O)

624 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/audio/drv/audiopci/%.c
625         $(COMPILE.c) -o $@ $<
626         $(CTFCONVERT_O)

628 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/audio/drv/audiop16x/%.c
629         $(COMPILE.c) -o $@ $<
630         $(CTFCONVERT_O)

632 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/audio/drv/audiosolo/%.c
633         $(COMPILE.c) -o $@ $<
634         $(CTFCONVERT_O)

636 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/audio/drv/audiots/%.c
637         $(COMPILE.c) -o $@ $<
638         $(CTFCONVERT_O)

640 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/audio/drv/audiovia823x/%.c
641         $(COMPILE.c) -o $@ $<
642         $(CTFCONVERT_O)

644 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/audio/drv/audiovia97/%.c
645         $(COMPILE.c) -o $@ $<
646         $(CTFCONVERT_O)

648 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/bfe/%.c
649         $(COMPILE.c) -o $@ $<
650         $(CTFCONVERT_O)

652 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/bge/%.c
653         $(COMPILE.c) -o $@ $<
654         $(CTFCONVERT_O)
```

```
656 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/blkdev/%.c
657         $(COMPILE.c) -o $@ $<
658         $(CTFCONVERT_O)

660 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/bpf/%.c
661         $(COMPILE.c) -o $@ $<
662         $(CTFCONVERT_O)

664 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/cardbus/%.c
665         $(COMPILE.c) -o $@ $<
666         $(CTFCONVERT_O)

668 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/comstar/stmf/%.c
669         $(COMPILE.c) -o $@ $<
670         $(CTFCONVERT_O)

672 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/comstar/port/fct/%.c
673         $(COMPILE.c) -o $@ $<
674         $(CTFCONVERT_O)

676 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/comstar/port/qlt/%.c
677         $(COMPILE.c) -o $@ $<
678         $(CTFCONVERT_O)

680 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/comstar/port/srpt/%.c
681         $(COMPILE.c) -o $@ $<
682         $(CTFCONVERT_O)

684 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/comstar/port/fcoet/%.c
685         $(COMPILE.c) -o $@ $<
686         $(CTFCONVERT_O)

688 $(OBJS_DIR)/%.o:                $(COMMONBASE)/iscsit/%.c
689         $(COMPILE.c) -o $@ $<
690         $(CTFCONVERT_O)

692 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/comstar/port/iscsit/%.c
693         $(COMPILE.c) -o $@ $<
694         $(CTFCONVERT_O)

696 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/comstar/port/pppt/%.c
697         $(COMPILE.c) -o $@ $<
698         $(CTFCONVERT_O)

700 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/comstar/lu/stmf_sbd/%.c
701         $(COMPILE.c) -o $@ $<
702         $(CTFCONVERT_O)

704 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/dld/%.c
705         $(COMPILE.c) -o $@ $<
706         $(CTFCONVERT_O)

708 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/dls/%.c
709         $(COMPILE.c) -o $@ $<
710         $(CTFCONVERT_O)

712 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/dmfe/%.c
713         $(COMPILE.c) -o $@ $<
714         $(CTFCONVERT_O)

716 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/drm/%.c
717         $(COMPILE.c) -o $@ $<
718         $(CTFCONVERT_O)

720 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/efe/%.c
721         $(COMPILE.c) -o $@ $<
```



```
722         $(CTFCONVERT_O)

724 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/elxl/%.c
725         $(COMPILE.c) -o $@ $<
726         $(CTFCONVERT_O)

728 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/fcoe/%.c
729         $(COMPILE.c) -o $@ $<
730         $(CTFCONVERT_O)

732 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/hme/%.c
733         $(COMPILE.c) -o $@ $<
734         $(CTFCONVERT_O)

736 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/pciex/%.c
737         $(COMPILE.c) -o $@ $<
738         $(CTFCONVERT_O)

740 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/hotplug/hpcsvc/%.c
741         $(COMPILE.c) -o $@ $<
742         $(CTFCONVERT_O)

744 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/pciex/hotplug/%.c
745         $(COMPILE.c) -o $@ $<
746         $(CTFCONVERT_O)

748 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/hotplug/pcihp/%.c
749         $(COMPILE.c) -o $@ $<
750         $(CTFCONVERT_O)

752 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/ib/clients/rds/%.c
753         $(COMPILE.c) -o $@ $<
754         $(CTFCONVERT_O)

756 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/ib/clients/rdsv3/%.c
757         $(COMPILE.c) -o $@ $<
758         $(CTFCONVERT_O)

760 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/ib/clients/iser/%.c
761         $(COMPILE.c) -o $@ $<
762         $(CTFCONVERT_O)

764 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/ib/clients/ibd/%.c
765         $(COMPILE.c) -o $@ $<
766         $(CTFCONVERT_O)

768 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/ib/clients/eoib/%.c
769         $(COMPILE.c) -o $@ $<
770         $(CTFCONVERT_O)

772 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/ib/clients/of/sol_ofs/%.c
773         $(COMPILE.c) -o $@ $<
774         $(CTFCONVERT_O)

776 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/ib/clients/of/sol_ucma/%.c
777         $(COMPILE.c) -o $@ $<
778         $(CTFCONVERT_O)

780 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/ib/clients/of/sol_umad/%.c
781         $(COMPILE.c) -o $@ $<
782         $(CTFCONVERT_O)

784 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/ib/clients/of/sol_uverbs/%.
785         $(COMPILE.c) -o $@ $<
786         $(CTFCONVERT_O)
```

```
788 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/ib/clients/sdp/%.c
789         $(COMPILE.c) -o $@ $<
790         $(CTFCONVERT_O)

792 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/ib/mgt/ibcm/%.c
793         $(COMPILE.c) -o $@ $<
794         $(CTFCONVERT_O)

796 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/ib/mgt/ibdm/%.c
797         $(COMPILE.c) -o $@ $<
798         $(CTFCONVERT_O)

800 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/ib/mgt/ibdma/%.c
801         $(COMPILE.c) -o $@ $<
802         $(CTFCONVERT_O)

804 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/ib/mgt/ibmf/%.c
805         $(COMPILE.c) -o $@ $<
806         $(CTFCONVERT_O)

808 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/ib/ibnex/%.c
809         $(COMPILE.c) -o $@ $<
810         $(CTFCONVERT_O)

812 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/ib/ibtl/%.c
813         $(COMPILE.c) -o $@ $<
814         $(CTFCONVERT_O)

816 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/ib/adapters/tavor/%.c
817         $(COMPILE.c) -o $@ $<
818         $(CTFCONVERT_O)

820 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/ib/adapters/hermon/%.c
821         $(COMPILE.c) -o $@ $<
822         $(CTFCONVERT_O)

824 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/ib/clients/daplt/%.c
825         $(COMPILE.c) -o $@ $<
826         $(CTFCONVERT_O)

828 $(OBJS_DIR)/%.o:                $(COMMONBASE)/iscsi/%.c
829         $(COMPILE.c) -o $@ $<
830         $(CTFCONVERT_O)

832 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/idm/%.c
833         $(COMPILE.c) -o $@ $<
834         $(CTFCONVERT_O)

836 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/ipw/%.c
837         $(COMPILE.c) -o $@ $<
838         $(CTFCONVERT_O)

840 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/iwh/%.c
841         $(COMPILE.c) -o $@ $<
842         $(CTFCONVERT_O)

844 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/iwi/%.c
845         $(COMPILE.c) -o $@ $<
846         $(CTFCONVERT_O)

848 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/iwk/%.c
849         $(COMPILE.c) -o $@ $<
850         $(CTFCONVERT_O)

852 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/iwp/%.c
853         $(COMPILE.c) -o $@ $<
```



```
854         $(CTFCONVERT_O)

856 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/kb8042/%.c
857         $(COMPILE.c) -o $@ $<
858         $(CTFCONVERT_O)

860 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/kbtrans/%.c
861         $(COMPILE.c) -o $@ $<
862         $(CTFCONVERT_O)

864 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/ksocket/%.c
865         $(COMPILE.c) -o $@ $<
866         $(CTFCONVERT_O)

868 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/aggr/%.c
869         $(COMPILE.c) -o $@ $<
870         $(CTFCONVERT_O)

872 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/lp/%.c
873         $(COMPILE.c) -o $@ $<
874         $(CTFCONVERT_O)

876 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/lvm/hotspares/%.c
877         $(COMPILE.c) -o $@ $<
878         $(CTFCONVERT_O)

880 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/lvm/md/%.c
881         $(COMPILE.c) -o $@ $<
882         $(CTFCONVERT_O)

884 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/lvm/mirror/%.c
885         $(COMPILE.c) -o $@ $<
886         $(CTFCONVERT_O)

888 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/lvm/notify/%.c
889         $(COMPILE.c) -o $@ $<
890         $(CTFCONVERT_O)

892 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/lvm/raid/%.c
893         $(COMPILE.c) -o $@ $<
894         $(CTFCONVERT_O)

896 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/lvm/softpart/%.c
897         $(COMPILE.c) -o $@ $<
898         $(CTFCONVERT_O)

900 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/lvm/stripe/%.c
901         $(COMPILE.c) -o $@ $<
902         $(CTFCONVERT_O)

904 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/lvm/trans/%.c
905         $(COMPILE.c) -o $@ $<
906         $(CTFCONVERT_O)

908 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/mac/%.c
909         $(COMPILE.c) -o $@ $<
910         $(CTFCONVERT_O)

912 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/mac/plugins/%.c
913         $(COMPILE.c) -o $@ $<
914         $(CTFCONVERT_O)

916 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/mega_sas/%.c
917         $(COMPILE.c) -o $@ $<
918         $(CTFCONVERT_O)
```

```
$(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/mii/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/mr_sas/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/scsi/adapters/mpt_sas/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/mxfe/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/mwl/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/mwl/mwl_fw/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/net80211/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/nge/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/nxge/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/nxge/npi/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/nxge/%.s
        $(COMPILE.s) -o $@ $<

$(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/pci-ide/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/pcmcia/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/pcan/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/pcn/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/pcwl/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/ppp/sppp/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)
```



```
$(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/ppp/spppasyn/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/ppp/sppptun/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/ral/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/rge/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/rtls/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/rsm/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/rtw/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/rum/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/rwd/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/rwn/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/sata/adapters/ahci/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/sata/adapters/nv_sata/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/sata/adapters/si3124/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/sata/impl/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/scsi/conf/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/scsi/impl/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/scsi/targets/%.c
```

```
1052         $(COMPILE.c) -o $@ $<
1053         $(CTFCONVERT_O)

1055 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/scsi/adapters/%.c
1056         $(COMPILE.c) -o $@ $<
1057         $(CTFCONVERT_O)

1059 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/scsi/adapters/blk2scsa/%.c
1060         $(COMPILE.c) -o $@ $<
1061         $(CTFCONVERT_O)

1063 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/scsi/adapters/scsi_vhci/%.c
1064         $(COMPILE.c) -o $@ $<
1065         $(CTFCONVERT_O)

1067 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/scsi/adapters/scsi_vhci/fop
1068         $(COMPILE.c) -o $@ $<
1069         $(CTFCONVERT_O)

1071 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/fibre-channel/ulp/%.c
1072         $(COMPILE.c) -o $@ $<
1073         $(CTFCONVERT_O)

1075 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/fibre-channel/impl/%.c
1076         $(COMPILE.c) -o $@ $<
1077         $(CTFCONVERT_O)

1079 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/fibre-channel/fca/qlc/%.c
1080         $(COMPILE.c) -o $@ $<
1081         $(CTFCONVERT_O)

1083 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/fibre-channel/fca/qlge/%.c
1084         $(COMPILE.c) -o $@ $<
1085         $(CTFCONVERT_O)

1087 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/fibre-channel/fca/emlxs/%.c
1088         $(COMPILE.c) -o $@ $<
1089         $(CTFCONVERT_O)

1091 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/fibre-channel/fca/oce/%.c
1092         $(COMPILE.c) -o $@ $<
1093         $(CTFCONVERT_O)

1095 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/fibre-channel/fca/fcoei/%.c
1096         $(COMPILE.c) -o $@ $<
1097         $(CTFCONVERT_O)

1099 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/sdcard/adapters/sdhost/%.c
1100         $(COMPILE.c) -o $@ $<
1101         $(CTFCONVERT_O)

1103 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/sdcard/impl/%.c
1104         $(COMPILE.c) -o $@ $<
1105         $(CTFCONVERT_O)

1107 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/sdcard/targets/sdcard/%.c
1108         $(COMPILE.c) -o $@ $<
1109         $(CTFCONVERT_O)

1111 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/sfe/%.c
1112         $(COMPILE.c) -o $@ $<
1113         $(CTFCONVERT_O)

1115 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/simnet/%.c
1116         $(COMPILE.c) -o $@ $<
1117         $(CTFCONVERT_O)
```



```
1119 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/softmac/%.c
1120         $(COMPILE.c) -o $@ $<
1121         $(CTFCONVERT_O)

1123 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/uath/%.c
1124         $(COMPILE.c) -o $@ $<
1125         $(CTFCONVERT_O)

1127 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/uath/uath_fw/%.c
1128         $(COMPILE.c) -o $@ $<
1129         $(CTFCONVERT_O)

1131 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/ural/%.c
1132         $(COMPILE.c) -o $@ $<
1133         $(CTFCONVERT_O)

1135 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/urtw/%.c
1136         $(COMPILE.c) -o $@ $<
1137         $(CTFCONVERT_O)

1139 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/usb/clients/audio/usb_ac/%.
1140         $(COMPILE.c) -o $@ $<
1141         $(CTFCONVERT_O)

1143 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/usb/clients/audio/usb_as/%.
1144         $(COMPILE.c) -o $@ $<
1145         $(CTFCONVERT_O)

1147 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/usb/clients/audio/usb_ah/%.
1148         $(COMPILE.c) -o $@ $<
1149         $(CTFCONVERT_O)

1151 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/usb/clients/usbskel/%.c
1152         $(COMPILE.c) -o $@ $<
1153         $(CTFCONVERT_O)

1155 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/usb/clients/video/usbvc/%.c
1156         $(COMPILE.c) -o $@ $<
1157         $(CTFCONVERT_O)

1159 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/usb/clients/hwarc/%.c
1160         $(COMPILE.c) -o $@ $<
1161         $(CTFCONVERT_O)

1163 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/usb/clients/hid/%.c
1164         $(COMPILE.c) -o $@ $<
1165         $(CTFCONVERT_O)

1167 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/usb/clients/hidparser/%.c
1168         $(COMPILE.c) -o $@ $<
1169         $(CTFCONVERT_O)

1171 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/usb/clients/printer/%.c
1172         $(COMPILE.c) -o $@ $<
1173         $(CTFCONVERT_O)

1175 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/usb/clients/usbkbm/%.c
1176         $(COMPILE.c) -o $@ $<
1177         $(CTFCONVERT_O)

1179 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/usb/clients/usbms/%.c
1180         $(COMPILE.c) -o $@ $<
1181         $(CTFCONVERT_O)

1183 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/usb/clients/usbinput/usbwcm
```

```
1184         $(COMPILE.c) -o $@ $<
1185         $(CTFCONVERT_O)

1187 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/usb/clients/ugen/%.c
1188         $(COMPILE.c) -o $@ $<
1189         $(CTFCONVERT_O)

1191 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/usb/clients/usbser/%.c
1192         $(COMPILE.c) -o $@ $<
1193         $(CTFCONVERT_O)

1195 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/usb/clients/usbser/usbsacm/
1196         $(COMPILE.c) -o $@ $<
1197         $(CTFCONVERT_O)

1199 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/usb/clients/usbser/usbftdi/
1200         $(COMPILE.c) -o $@ $<
1201         $(CTFCONVERT_O)

1203 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/usb/clients/usbser/usbser_k
1204         $(COMPILE.c) -o $@ $<
1205         $(CTFCONVERT_O)

1207 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/usb/clients/usbser/usbsprl/
1208         $(COMPILE.c) -o $@ $<
1209         $(CTFCONVERT_O)

1211 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/usb/clients/wusb_df/%.c
1212         $(COMPILE.c) -o $@ $<
1213         $(CTFCONVERT_O)

1215 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/usb/clients/hwa1480_fw/%.c
1216         $(COMPILE.c) -o $@ $<
1217         $(CTFCONVERT_O)

1219 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/usb/clients/wusb_ca/%.c
1220         $(COMPILE.c) -o $@ $<
1221         $(CTFCONVERT_O)

1223 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/usb/clients/usbecm/%.c
1224         $(COMPILE.c) -o $@ $<
1225         $(CTFCONVERT_O)

1227 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/usb/hcd/openhci/%.c
1228         $(COMPILE.c) -o $@ $<
1229         $(CTFCONVERT_O)

1231 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/usb/hcd/ehci/%.c
1232         $(COMPILE.c) -o $@ $<
1233         $(CTFCONVERT_O)

1235 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/usb/hcd/uhci/%.c
1236         $(COMPILE.c) -I../../common -o $@ $<
1237         $(CTFCONVERT_O)

1239 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/usb/hubd/%.c
1240         $(COMPILE.c) -o $@ $<
1241         $(CTFCONVERT_O)

1243 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/usb/scsa2usb/%.c
1244         $(COMPILE.c) -o $@ $<
1245         $(CTFCONVERT_O)

1247 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/usb/usb_mid/%.c
1248         $(COMPILE.c) -o $@ $<
1249         $(CTFCONVERT_O)
```



```
1251 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/usb/usb_ia/%.c
1252         $(COMPILE.c) -o $@ $<
1253         $(CTFCONVERT_O)

1255 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/usb/usba/%.c
1256         $(COMPILE.c) -o $@ $<
1257         $(CTFCONVERT_O)

1259 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/usb/usba10/%.c
1260         $(COMPILE.c) -o $@ $<
1261         $(CTFCONVERT_O)

1263 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/usb/hwa/hwahc/%.c
1264         $(COMPILE.c) -o $@ $<
1265         $(CTFCONVERT_O)

1267 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/uwb/uwba/%.c
1268         $(COMPILE.c) -o $@ $<
1269         $(CTFCONVERT_O)

1271 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/vuidmice/%.c
1272         $(COMPILE.c) -o $@ $<
1273         $(CTFCONVERT_O)

1275 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/vnic/%.c
1276         $(COMPILE.c) -o $@ $<
1277         $(CTFCONVERT_O)

1279 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/wpi/%.c
1280         $(COMPILE.c) -o $@ $<
1281         $(CTFCONVERT_O)

1283 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/zyd/%.c
1284         $(COMPILE.c) -o $@ $<
1285         $(CTFCONVERT_O)

1287 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/chxge/com/%.c
1288         $(COMPILE.c) -o $@ $<
1289         $(CTFCONVERT_O)

1291 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/chxge/%.c
1292         $(COMPILE.c) -o $@ $<
1293         $(CTFCONVERT_O)

1295 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/cxgbe/common/%.c
1296         $(COMPILE.c) -o $@ $<
1297         $(CTFCONVERT_O)

1299 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/cxgbe/shared/%.c
1300         $(COMPILE.c) -o $@ $<
1301         $(CTFCONVERT_O)

1303 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/cxgbe/firmware/%.c
1304         $(COMPILE.c) -o $@ $<
1305         $(CTFCONVERT_O)

1307 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/cxgbe/t4nex/%.c
1308         $(COMPILE.c) -o $@ $<
1309         $(CTFCONVERT_O)

1311 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/cxgbe/cxgbe/%.c
1312         $(COMPILE.c) -o $@ $<
1313         $(CTFCONVERT_O)

1315 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/ixgb/%.c
```

```
1316         $(COMPILE.c) -o $@ $<
1317         $(CTFCONVERT_O)

1319 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/xge/drv/%.c
1320         $(COMPILE.c) -o $@ $<
1321         $(CTFCONVERT_O)

1323 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/xge/hal/xgehal/%.c
1324         $(COMPILE.c) -o $@ $<
1325         $(CTFCONVERT_O)

1327 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/e1000g/%.c
1328         $(COMPILE.c) -o $@ $<
1329         $(CTFCONVERT_O)

1331 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/igb/%.c
1332         $(COMPILE.c) -o $@ $<
1333         $(CTFCONVERT_O)

1335 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/iprb/%.c
1336         $(COMPILE.c) -o $@ $<
1337         $(CTFCONVERT_O)

1339 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/ixgbe/%.c
1340         $(COMPILE.c) -o $@ $<
1341         $(CTFCONVERT_O)

1343 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/ntxn/%.c
1344         $(COMPILE.c) -o $@ $<
1345         $(CTFCONVERT_O)

1347 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/myri10ge/drv/%.c
1348         $(COMPILE.c) -o $@ $<
1349         $(CTFCONVERT_O)

1351 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/sfxge/%.c
1352         $(COMPILE.c) -o $@ $<
1353         $(CTFCONVERT_O)

1355 #endif /* ! codereview */
1356 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/ipp/%.c
1357         $(COMPILE.c) -o $@ $<
1358         $(CTFCONVERT_O)

1360 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/ipp/ipgpc/%.c
1361         $(COMPILE.c) -o $@ $<
1362         $(CTFCONVERT_O)

1364 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/ipp/dlcosmk/%.c
1365         $(COMPILE.c) -o $@ $<
1366         $(CTFCONVERT_O)

1368 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/ipp/flowacct/%.c
1369         $(COMPILE.c) -o $@ $<
1370         $(CTFCONVERT_O)

1372 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/ipp/dscpmk/%.c
1373         $(COMPILE.c) -o $@ $<
1374         $(CTFCONVERT_O)

1376 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/ipp/meters/%.c
1377         $(COMPILE.c) -o $@ $<
1378         $(CTFCONVERT_O)

1380 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/kiconv/kiconv_emea/%.c
1381         $(COMPILE.c) -o $@ $<
```



```
1382         $(CTFCONVERT_O)

1384 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/kiconv/kiconv_ja/%.c
1385         $(COMPILE.c) -o $@ $<
1386         $(CTFCONVERT_O)

1388 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/kiconv/kiconv_ko/%.c
1389         $(COMPILE.c) -o $@ $<
1390         $(CTFCONVERT_O)

1392 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/kiconv/kiconv_sc/%.c
1393         $(COMPILE.c) -o $@ $<
1394         $(CTFCONVERT_O)

1396 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/kiconv/kiconv_tc/%.c
1397         $(COMPILE.c) -o $@ $<
1398         $(CTFCONVERT_O)

1400 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/kmdb/%.c
1401         $(COMPILE.c) -o $@ $<
1402         $(CTFCONVERT_O)

1404 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/ktli/%.c
1405         $(COMPILE.c) -o $@ $<
1406         $(CTFCONVERT_O)

1408 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/scsi/adapters/iscsi/%.c
1409         $(COMPILE.c) -o $@ $<
1410         $(CTFCONVERT_O)

1412 $(OBJS_DIR)/%.o:                $(COMMONBASE)/iscsi/%.c
1413         $(COMPILE.c) -o $@ $<
1414         $(CTFCONVERT_O)

1416 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/inet/kifconf/%.c
1417         $(COMPILE.c) -o $@ $<
1418         $(CTFCONVERT_O)

1420 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/vr/%.c
1421         $(COMPILE.c) -o $@ $<
1422         $(CTFCONVERT_O)

1424 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/yge/%.c
1425         $(COMPILE.c) -o $@ $<
1426         $(CTFCONVERT_O)

1428 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/virtio/%.c
1429         $(COMPILE.c) -o $@ $<
1430         $(CTFCONVERT_O)

1432 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/vioblk/%.c
1433         $(COMPILE.c) -o $@ $<
1434         $(CTFCONVERT_O)

1436 #
1437 # krtld must refer to its own bzero/bcopy until the kernel is fully linked
1438 #
1439 $(OBJS_DIR)/bootrd.o            := CPPFLAGS += -DKOBJ_OVERRIDES
1440 $(OBJS_DIR)/doreloc.o           := CPPFLAGS += -DKOBJ_OVERRIDES
1441 $(OBJS_DIR)/kobj.o              := CPPFLAGS += -DKOBJ_OVERRIDES
1442 $(OBJS_DIR)/kobj_boot.o         := CPPFLAGS += -DKOBJ_OVERRIDES
1443 $(OBJS_DIR)/kobj_bootflags.o    := CPPFLAGS += -DKOBJ_OVERRIDES
1444 $(OBJS_DIR)/kobj_convrelstr.o   := CPPFLAGS += -DKOBJ_OVERRIDES
1445 $(OBJS_DIR)/kobj_isa.o          := CPPFLAGS += -DKOBJ_OVERRIDES
1446 $(OBJS_DIR)/kobj_kdi.o          := CPPFLAGS += -DKOBJ_OVERRIDES
1447 $(OBJS_DIR)/kobj_lm.o           := CPPFLAGS += -DKOBJ_OVERRIDES
```

```
$(OBJS_DIR)/kobj_reloc.o        := CPPFLAGS += -DKOBJ_OVERRIDES
$(OBJS_DIR)/kobj_stubs.o        := CPPFLAGS += -DKOBJ_OVERRIDES
$(OBJS_DIR)/kobj_subr.o         := CPPFLAGS += -DKOBJ_OVERRIDES

$(OBJS_DIR)/%.o:                $(UTSBASE)/common/krtld/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(COMMONBASE)/list/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(COMMONBASE)/lvm/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(COMMONBASE)/lzma/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(COMMONBASE)/crypto/md4/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(COMMONBASE)/crypto/md5/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(COMMONBASE)/net/dhcp/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(COMMONBASE)/nvpair/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/common/os/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/common/pcmcia/cis/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/common/pcmcia/cs/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/common/pcmcia/nexus/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/common/pcmcia/pcs/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/common/rpc/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/common/rpc/sec/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/common/rpc/sec_gss/%.c
        $(COMPILE.c) -o $@ $<
```



```
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(COMMONBASE)/crypto/sha1/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(COMMONBASE)/crypto/sha2/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/common/syscall/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/common/tnf/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(COMMONBASE)/tsol/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(COMMONBASE)/util/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(COMMONBASE)/unicode/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/common/vm/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/common/zmod/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/zlib_obj.o:         $(ZLIB_OBJS:%=$(OBJS_DIR)/%)
        $(LD) -r -Breduce -M$(UTSBASE)/common/zmod/mapfile -o $@ \
            $(ZLIB_OBJS:%=$(OBJS_DIR)/%)
        $(CTFMERGE) -t -f -L VERSION -o $@ $(ZLIB_OBJS:%=$(OBJS_DIR)/%)

$(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/hxge/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/tpm/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/tpm/%.s
        $(COMPILE.s) -o $@ $<

$(OBJS_DIR)/bz2%.o:             $(COMMONBASE)/bzip2/%.c
        $(COMPILE.c) -o $@ -I$(COMMONBASE)/bzip2 $<
        $(CTFCONVERT_O)

BZ2LINT = -erroff=%all -I$(UTSBASE)/common/bzip2

$(LINTS_DIR)/bz2%.ln:           $(COMMONBASE)/bzip2/%.c
        @($(LHEAD) $(LINT.c) -C $(LINTS_DIR)/‘basename $@ .ln‘ $(BZ2LINT) $< $(

#
# SVM
#
```

```
MD_XDR_CSRC     =       $(UTSBASE)/common/io/lvm/md
MD_XDR_XSRC     =       $(UTSBASE)/common/sys/lvm
RPCGENFLAGS     +=      -C -M -D_KERNEL -DSYSV

$(MD_XDR_CSRC)/meta_basic_xdr.c:        $(MD_XDR_XSRC)/meta_basic.x
        $(RPCGEN) $(RPCGENFLAGS) -c -i 100 $(MD_XDR_XSRC)/meta_basic.x | \
        nawk '{sub(/^#include "(\.\.\/\.\.\/)/,"#include \"\.\.\/\.\.\/\.\.\/\.\
        nawk '{sub(/meta_basic.h/, "md_basic.h"); print $$0}' >$@

$(MD_XDR_CSRC)/metamed_xdr.c:   $(MD_XDR_XSRC)/metamed.x
        $(RPCGEN) $(RPCGENFLAGS) -c -i 100 $(MD_XDR_XSRC)/metamed.x |  \
        nawk '{sub(/^#include "(\.\.\/\.\.\/)/,"#include \"\.\.\/\.\.\/\.\.\/\.\
        nawk '{sub(/metamed.h/, "mdmed.h"); print $$0}' >$@

#
#       Section 1b:     Lint 'objects'
#
$(LINTS_DIR)/%.ln:              $(COMMONBASE)/crypto/aes/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(COMMONBASE)/crypto/arcfour/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(COMMONBASE)/crypto/blowfish/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(COMMONBASE)/crypto/ecc/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(COMMONBASE)/crypto/modes/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(COMMONBASE)/crypto/padding/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(COMMONBASE)/crypto/rng/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(COMMONBASE)/crypto/rsa/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(COMMONBASE)/bignum/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/bignum/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(COMMONBASE)/mpi/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(COMMONBASE)/acl/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(COMMONBASE)/avl/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(COMMONBASE)/ucode/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/brand/sn1/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/brand/solaris10/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))
```



```
$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/c2/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/conf/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/contract/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/cpr/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/ctf/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(COMMONBASE)/ctf/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(COMMONBASE)/pci/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(COMMONBASE)/devid/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(COMMONBASE)/crypto/des/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(COMMONBASE)/smbios/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/avs/ncall/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/avs/ns/dsw/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/avs/ns/nsctl/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/avs/ns/rdc/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/avs/ns/sdbc/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/avs/ns/solaris/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/avs/ns/sv/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/avs/ns/unistat/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/des/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/crypto/api/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/crypto/core/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/crypto/io/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))
```

```
1712 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/crypto/spi/%.c
1713         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1715 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/disp/%.c
1716         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1718 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/dtrace/%.c
1719         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1721 $(LINTS_DIR)/%.ln:              $(COMMONBASE)/exacct/%.c
1722         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1724 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/exec/aout/%.c
1725         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1727 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/exec/elf/%.c
1728         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1730 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/exec/intp/%.c
1731         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1733 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/exec/shbin/%.c
1734         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1736 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/exec/java/%.c
1737         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1739 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/fs/%.c
1740         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1742 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/fs/autofs/%.c
1743         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1745 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/fs/cachefs/%.c
1746         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1748 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/fs/ctfs/%.c
1749         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1751 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/fs/doorfs/%.c
1752         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1754 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/fs/dcfs/%.c
1755         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1757 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/fs/devfs/%.c
1758         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1760 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/fs/dev/%.c
1761         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1763 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/fs/fd/%.c
1764         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1766 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/fs/fifofs/%.c
1767         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1769 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/fs/hsfs/%.c
1770         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1772 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/fs/lofs/%.c
1773         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1775 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/fs/mntfs/%.c
1776         @($(LHEAD) $(LINT.c) $< $(LTAIL))
```



```
1778 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/fs/namefs/%.c
1779         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1781 $(LINTS_DIR)/%.ln:              $(COMMONBASE)/smbsrv/%.c
1782         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1784 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/fs/smbsrv/%.c
1785         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1787 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/fs/nfs/%.c
1788         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1790 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/fs/objfs/%.c
1791         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1793 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/fs/pcfs/%.c
1794         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1796 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/fs/portfs/%.c
1797         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1799 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/fs/proc/%.c
1800         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1802 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/fs/sharefs/%.c
1803         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1805 $(LINTS_DIR)/%.ln:              $(COMMONBASE)/smbclnt/%.c
1806         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1808 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/fs/smbclnt/netsmb/%.c
1809         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1811 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/fs/smbclnt/smbfs/%.c
1812         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1814 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/fs/sockfs/%.c
1815         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1817 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/fs/specfs/%.c
1818         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1820 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/fs/swapfs/%.c
1821         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1823 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/fs/tmpfs/%.c
1824         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1826 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/fs/udfs/%.c
1827         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1829 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/fs/ufs/%.c
1830         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1832 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/fs/ufs_log/%.c
1833         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1835 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/vscan/%.c
1836         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1838 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/fs/zfs/%.c
1839         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1841 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/fs/zut/%.c
1842         @($(LHEAD) $(LINT.c) $< $(LTAIL))
```

```
$(LINTS_DIR)/%.ln:              $(COMMONBASE)/xattr/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(COMMONBASE)/zfs/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/gssapi/%.c
        @($(LHEAD) $(LINT.c) $(KGSSDFLAGS) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/gssapi/mechs/dummy/%.c
        @($(LHEAD) $(LINT.c) $(KGSSDFLAGS) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(KMECHKRB5_BASE)/%.c
        @($(LHEAD) $(LINT.c) $(KGSSDFLAGS) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(KMECHKRB5_BASE)/crypto/%.c
        @($(LHEAD) $(LINT.c) $(KGSSDFLAGS) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(KMECHKRB5_BASE)/crypto/des/%.c
        @($(LHEAD) $(LINT.c) $(KGSSDFLAGS) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(KMECHKRB5_BASE)/crypto/dk/%.c
        @($(LHEAD) $(LINT.c) $(KGSSDFLAGS) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(KMECHKRB5_BASE)/crypto/os/%.c
        @($(LHEAD) $(LINT.c) $(KGSSDFLAGS) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(KMECHKRB5_BASE)/crypto/arcfour/%.c
        @($(LHEAD) $(LINT.c) $(KGSSDFLAGS) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(KMECHKRB5_BASE)/crypto/enc_provider/%.c
        @($(LHEAD) $(LINT.c) $(KGSSDFLAGS) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(KMECHKRB5_BASE)/crypto/hash_provider/%.c
        @($(LHEAD) $(LINT.c) $(KGSSDFLAGS) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(KMECHKRB5_BASE)/crypto/keyhash_provider/%.c
        @($(LHEAD) $(LINT.c) $(KGSSDFLAGS) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(KMECHKRB5_BASE)/crypto/raw/%.c
        @($(LHEAD) $(LINT.c) $(KGSSDFLAGS) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(KMECHKRB5_BASE)/crypto/old/%.c
        @($(LHEAD) $(LINT.c) $(KGSSDFLAGS) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(KMECHKRB5_BASE)/krb5/krb/%.c
        @($(LHEAD) $(LINT.c) $(KGSSDFLAGS) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(KMECHKRB5_BASE)/krb5/os/%.c
        @($(LHEAD) $(LINT.c) $(KGSSDFLAGS) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(KMECHKRB5_BASE)/mech/%.c
        @($(LHEAD) $(LINT.c) $(KGSSDFLAGS) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/idmap/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/inet/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/inet/sockmods/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/inet/arp/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))
```



```
$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/inet/ip/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/inet/ipnet/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/inet/iptun/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/inet/ipf/%.c
        @($(LHEAD) $(LINT.c) $(IPFFLAGS) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/inet/kssl/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(COMMONBASE)/net/patricia/%.c
        @($(LHEAD) $(LINT.c) $(IPFFLAGS) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/inet/udp/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/inet/sctp/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/inet/tcp/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/inet/ilb/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/inet/nca/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/inet/dlpistub/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/1394/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/1394/adapters/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/1394/targets/av1394/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/1394/targets/dcam1394/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/1394/targets/scsa1394/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/sbp2/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/aac/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/afe/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/atge/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))
```

```
1976 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/arn/%.c
1977         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1979 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/ath/%.c
1980         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1982 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/atu/%.c
1983         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1985 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/audio/impl/%.c
1986         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1988 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/audio/ac97/%.c
1989         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1991 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/audio/drv/audio1575/%.c
1992         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1994 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/audio/drv/audio810/%.c
1995         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1997 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/audio/drv/audiocmi/%.c
1998         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2000 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/audio/drv/audiocmihd/%.c
2001         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2003 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/audio/drv/audioens/%.c
2004         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2006 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/audio/drv/audioemu10k/%.c
2007         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2009 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/audio/drv/audiohd/%.c
2010         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2012 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/audio/drv/audioixp/%.c
2013         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2015 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/audio/drv/audiols/%.c
2016         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2018 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/audio/drv/audiopci/%.c
2019         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2021 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/audio/drv/audiop16x/%.c
2022         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2024 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/audio/drv/audiosolo/%.c
2025         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2027 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/audio/drv/audiots/%.c
2028         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2030 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/audio/drv/audiovia823x/%.c
2031         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2033 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/audio/drv/audiovia97/%.c
2034         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2036 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/bfe/%.c
2037         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2039 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/bpf/%.c
2040         @($(LHEAD) $(LINT.c) $< $(LTAIL))
```



```
2042 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/bge/%.c
2043         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2045 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/blkdev/%.c
2046         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2048 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/cardbus/%.c
2049         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2051 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/comstar/lu/stmf_sbd/%.c
2052         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2054 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/comstar/port/fct/%.c
2055         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2057 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/comstar/port/qlt/%.c
2058         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2060 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/comstar/port/srpt/%.c
2061         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2063 $(LINTS_DIR)/%.ln:              $(COMMONBASE)/iscsit/%.c
2064         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2066 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/comstar/port/fcoet/%.c
2067         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2069 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/comstar/port/iscsit/%.c
2070         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2072 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/comstar/port/pppt/%.c
2073         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2075 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/comstar/stmf/%.c
2076         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2078 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/dld/%.c
2079         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2081 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/dls/%.c
2082         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2084 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/dmfe/%.c
2085         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2087 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/drm/%.c
2088         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2090 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/efe/%.c
2091         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2093 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/elxl/%.c
2094         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2096 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/fcoe/%.c
2097         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2099 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/hme/%.c
2100         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2102 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/pciex/%.c
2103         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2105 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/hotplug/hpcsvc/%.c
2106         @($(LHEAD) $(LINT.c) $< $(LTAIL))
```

```
2108 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/pciex/hotplug/%.c
2109         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2111 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/hotplug/pcihp/%.c
2112         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2114 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/ib/clients/rds/%.c
2115         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2117 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/ib/clients/rdsv3/%.c
2118         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2120 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/ib/clients/iser/%.c
2121         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2123 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/ib/clients/ibd/%.c
2124         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2126 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/ib/clients/eoib/%.c
2127         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2129 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/ib/clients/of/sol_ofs/%.c
2130         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2132 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/ib/clients/of/sol_ucma/%.c
2133         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2135 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/ib/clients/of/sol_umad/%.c
2136         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2138 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/ib/clients/of/sol_uverbs/%.
2139         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2141 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/ib/clients/sdp/%.c
2142         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2144 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/ib/mgt/ibcm/%.c
2145         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2147 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/ib/mgt/ibdm/%.c
2148         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2150 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/ib/mgt/ibdma/%.c
2151         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2153 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/ib/mgt/ibmf/%.c
2154         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2156 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/ib/ibnex/%.c
2157         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2159 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/ib/ibtl/%.c
2160         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2162 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/ib/adapters/tavor/%.c
2163         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2165 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/ib/adapters/hermon/%.c
2166         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2168 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/ib/clients/daplt/%.c
2169         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2171 $(LINTS_DIR)/%.ln:              $(COMMONBASE)/iscsi/%.c
2172         @($(LHEAD) $(LINT.c) $< $(LTAIL))
```



```
2174 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/idm/%.c
2175         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2177 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/ipw/%.c
2178         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2180 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/iwh/%.c
2181         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2183 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/iwi/%.c
2184         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2186 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/iwk/%.c
2187         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2189 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/iwp/%.c
2190         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2192 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/kb8042/%.c
2193         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2195 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/kbtrans/%.c
2196         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2198 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/ksocket/%.c
2199         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2201 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/aggr/%.c
2202         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2204 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/lp/%.c
2205         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2207 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/lvm/hotspares/%.c
2208         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2210 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/lvm/md/%.c
2211         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2213 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/lvm/mirror/%.c
2214         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2216 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/lvm/raid/%.c
2217         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2219 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/lvm/softpart/%.c
2220         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2222 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/lvm/stripe/%.c
2223         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2225 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/lvm/notify/%.c
2226         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2228 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/lvm/trans/%.c
2229         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2231 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/mac/%.c
2232         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2234 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/mac/plugins/%.c
2235         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2237 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/mega_sas/%.c
2238         @($(LHEAD) $(LINT.c) $< $(LTAIL))
```

```
2240 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/mii/%.c
2241         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2243 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/mr_sas/%.c
2244         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2246 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/scsi/adapters/mpt_sas/%.c
2247         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2249 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/mxfe/%.c
2250         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2252 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/mwl/%.c
2253         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2255 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/mwl/mwl_fw/%.c
2256         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2258 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/net80211/%.c
2259         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2261 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/nge/%.c
2262         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2264 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/nxge/%.c
2265         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2267 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/nxge/%.s
2268         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2270 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/nxge/npi/%.c
2271         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2273 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/pci-ide/%.c
2274         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2276 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/pcmcia/%.c
2277         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2279 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/pcan/%.c
2280         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2282 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/pcn/%.c
2283         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2285 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/pcwl/%.c
2286         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2288 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/ppp/sppp/%.c
2289         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2291 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/ppp/spppasyn/%.c
2292         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2294 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/ppp/spppptun/%.c
2295         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2297 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/ral/%.c
2298         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2300 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/rge/%.c
2301         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2303 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/rtls/%.c
2304         @($(LHEAD) $(LINT.c) $< $(LTAIL))
```



```
2306 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/rsm/%.c
2307         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2309 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/rtw/%.c
2310         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2312 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/rum/%.c
2313         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2315 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/rwd/%.c
2316         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2318 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/rwn/%.c
2319         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2321 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/sata/adapters/ahci/%.c
2322         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2324 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/sata/adapters/nv_sata/%.c
2325         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2327 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/sata/adapters/si3124/%.c
2328         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2330 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/sata/impl/%.c
2331         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2333 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/scsi/adapters/%.c
2334         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2336 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/scsi/adapters/blk2scsa/%.c
2337         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2339 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/scsi/adapters/pmcs/%.c
2340         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2342 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/scsi/adapters/scsi_vhci/%.c
2343         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2345 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/scsi/adapters/scsi_vhci/fop
2346         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2348 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/fibre-channel/ulp/%.c
2349         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2351 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/fibre-channel/impl/%.c
2352         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2354 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/fibre-channel/fca/qlc/%.c
2355         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2357 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/fibre-channel/fca/qlge/%.c
2358         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2360 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/fibre-channel/fca/emlxs/%.c
2361         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2363 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/fibre-channel/fca/oce/%.c
2364         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2366 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/fibre-channel/fca/fcoei/%.c
2367         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2369 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/scsi/conf/%.c
2370         @($(LHEAD) $(LINT.c) $< $(LTAIL))
```

```
2372 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/scsi/impl/%.c
2373         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2375 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/scsi/targets/%.c
2376         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2378 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/sdcard/adapters/sdhost/%.c
2379         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2381 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/sdcard/impl/%.c
2382         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2384 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/sdcard/targets/sdcard/%.c
2385         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2387 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/sfe/%.c
2388         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2390 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/simnet/%.c
2391         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2393 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/softmac/%.c
2394         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2396 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/uath/%.c
2397         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2399 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/uath/uath_fw/%.c
2400         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2402 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/ural/%.c
2403         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2405 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/urtw/%.c
2406         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2408 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/usb/clients/audio/usb_ac/%.
2409         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2411 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/usb/clients/audio/usb_as/%.
2412         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2414 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/usb/clients/audio/usb_ah/%.
2415         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2417 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/usb/clients/usbskel/%.c
2418         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2420 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/usb/clients/video/usbvc/%.c
2421         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2423 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/usb/clients/hwarc/%.c
2424         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2426 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/usb/clients/hid/%.c
2427         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2429 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/usb/clients/hidparser/%.c
2430         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2432 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/usb/clients/usbkbm/%.c
2433         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2435 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/usb/clients/usbms/%.c
2436         @($(LHEAD) $(LINT.c) $< $(LTAIL))
```



```
2438 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/usb/clients/usbinput/usbwcm
2439         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2441 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/usb/clients/ugen/%.c
2442         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2444 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/usb/clients/printer/%.c
2445         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2447 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/usb/clients/usbser/%.c
2448         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2450 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/usb/clients/usbser/usbsacm/
2451         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2453 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/usb/clients/usbser/usbftdi/
2454         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2456 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/usb/clients/usbser/usbser_k
2457         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2459 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/usb/clients/usbser/usbsprl/
2460         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2462 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/usb/clients/wusb_df/%.c
2463         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2465 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/usb/clients/hwa1480_fw/%.c
2466         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2468 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/usb/clients/wusb_ca/%.c
2469         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2471 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/usb/clients/usbecm/%.c
2472         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2474 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/usb/hcd/openhci/%.c
2475         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2477 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/usb/hcd/ehci/%.c
2478         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2480 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/usb/hcd/uhci/%.c
2481         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2483 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/usb/hubd/%.c
2484         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2486 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/usb/scsa2usb/%.c
2487         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2489 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/usb/usb_mid/%.c
2490         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2492 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/usb/usb_ia/%.c
2493         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2495 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/usb/usba/%.c
2496         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2498 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/usb/usba10/%.c
2499         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2501 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/uwb/uwba/%.c
2502         @($(LHEAD) $(LINT.c) $< $(LTAIL))
```

```
$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/usb/hwa/hwahc/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/vuidmice/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/vnic/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/wpi/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/zyd/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/chxge/com/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/chxge/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/cxgbe/common/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/cxgbe/shared/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/cxgbe/firmware/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/cxgbe/t4nex/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/cxgbe/cxgbe/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/ixgb/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/xge/drv/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/xge/hal/xgehal/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/e1000g/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/igb/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/iprb/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/ixgbe/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/ntxn/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/myri10ge/drv/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/sfxge/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))
```



```
#endif /* ! codereview */
$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/ipp/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/ipp/ipgpc/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/ipp/dlcosmk/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/ipp/flowacct/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/ipp/dscpmk/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/ipp/meters/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/kiconv/kiconv_emea/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/kiconv/kiconv_ja/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/kiconv/kiconv_ko/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/kiconv/kiconv_sc/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/kiconv/kiconv_tc/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/kmdb/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/krtld/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/ktli/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(COMMONBASE)/list/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(COMMONBASE)/lvm/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(COMMONBASE)/lzma/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(COMMONBASE)/crypto/md4/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(COMMONBASE)/crypto/md5/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(COMMONBASE)/net/dhcp/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(COMMONBASE)/nvpair/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/os/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))
```

```
$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/rpc/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/pcmcia/cs/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/pcmcia/cis/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/pcmcia/nexus/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/pcmcia/pcs/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/rpc/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/rpc/sec/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/rpc/sec_gss/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(COMMONBASE)/crypto/sha1/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(COMMONBASE)/crypto/sha2/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/syscall/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/tnf/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(COMMONBASE)/tsol/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(COMMONBASE)/util/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(COMMONBASE)/unicode/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/vm/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/scsi/adapters/iscsi/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(COMMONBASE)/iscsi/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/inet/kifconf/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/virtio/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/vioblk/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

ZMODLINTFLAGS = -erroff=E_CONSTANT_CONDITION
```



```
$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/zmod/%.c
        @($(LHEAD) $(LINT.c) $(ZMODLINTFLAGS) $< $(LTAIL))

$(LINTS_DIR)/zlib_obj.ln:       $(ZLIB_OBJS:%.o=$(LINTS_DIR)/%.ln) \
                                $(UTSBASE)/common/zmod/zlib_lint.c
        @($(LHEAD) $(LINT.c) -C $(LINTS_DIR)/zlib_obj \
            $(UTSBASE)/common/zmod/zlib_lint.c $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/hxge/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/tpm/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/tpm/%.s
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/vr/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/yge/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(COMMONBASE)/fsreparse/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))
```

****
580 Thu Aug 22 18:59:21 2013
new/usr/src/uts/common/io/sfxge/compat.h
Merged sfxge driver
****

```
#ifndef _SFXGE_COMPAT_H
#define _SFXGE_COMPAT_H

/*
 * Macro available in newer versions of /usr/include/sdt.h
 */
#include <sys/sdt.h>
#include <sys/inline.h>

#ifndef DTRACE_PROBE5
#define DTRACE_PROBE5(name, type1, arg1, type2, arg2, type3, arg3,      \
        type4, arg4, type5, arg5)                                       \
        {                                                               \
                extern void __dtrace_probe_##name(uintptr_t, uintptr_t, \
                    uintptr_t, uintptr_t, uintptr_t);                   \
                __dtrace_probe_##name((uintptr_t)(arg1),                \
                    (uintptr_t)(arg2), (uintptr_t)(arg3),               \
                    (uintptr_t)(arg4), (uintptr_t)(arg5));              \
        }
#endif  /* DTRACE_PROBE5 */

#endif
#endif /* ! codereview */
```

```
**********************************************************
   18729 Thu Aug 22 18:59:21 2013
new/usr/src/uts/common/io/sfxge/efsys.h
Merged sfxge driver
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2008-2013 Solarflare Communications Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#ifndef _SYS_EFSYS_H
#define _SYS_EFSYS_H

#ifdef  __cplusplus
extern "C" {
#endif

#include <sys/types.h>
#include <sys/sysmacros.h>
#include <sys/ddi.h>
#include <sys/sunddi.h>
#include <sys/cpuvar.h>
#include <sys/disp.h>
#include <sys/sdt.h>
#include <sys/kstat.h>
#include <sys/crc32.h>
#include <sys/note.h>
#include <sys/byteorder.h>

#define EFSYS_HAS_UINT64 1
#define EFSYS_USE_UINT64 0
#ifdef  _BIG_ENDIAN
#define EFSYS_IS_BIG_ENDIAN 1
#endif
#ifdef  _LITTLE_ENDIAN
#define EFSYS_IS_LITTLE_ENDIAN 1
#endif
#include "efx_types.h"

#ifdef _USE_GLD_V3_SOL10
#include "compat.h"
#endif

/* Modifiers used for DOS builds */
#define __cs
```



```
#define __far

/* Modifiers used for Windows builds */
#define __in
#define __in_opt
#define __in_ecount(_n)
#define __in_ecount_opt(_n)
#define __in_bcount(_n)
#define __in_bcount_opt(_n)

#define __out
#define __out_opt
#define __out_ecount(_n)
#define __out_ecount_opt(_n)
#define __out_bcount(_n)
#define __out_bcount_opt(_n)

#define __deref_out

#define __inout
#define __inout_opt
#define __inout_ecount(_n)
#define __inout_ecount_opt(_n)
#define __inout_bcount(_n)
#define __inout_bcount_opt(_n)
#define __inout_bcount_full_opt(_n)

#define __deref_out_bcount_opt(n)

#define __checkReturn

#define __drv_when(_p, _c)

/* Code inclusion options */


#define EFSYS_OPT_NAMES 1

#define EFSYS_OPT_FALCON 1
#define EFSYS_OPT_SIENA 1
#if DEBUG
#define EFSYS_OPT_CHECK_REG 1
#else
#define EFSYS_OPT_CHECK_REG 0
#endif

#define EFSYS_OPT_MCDI 1

#define EFSYS_OPT_MAC_FALCON_GMAC 1
#define EFSYS_OPT_MAC_FALCON_XMAC 1
#define EFSYS_OPT_MAC_STATS 1

#define EFSYS_OPT_LOOPBACK 1

#define EFSYS_OPT_MON_NULL 1
#define EFSYS_OPT_MON_LM87 1
#define EFSYS_OPT_MON_MAX6647 1
#define EFSYS_OPT_MON_SIENA 1
#define EFSYS_OPT_MON_STATS 1

#define EFSYS_OPT_PHY_NULL 1
#define EFSYS_OPT_PHY_QT2022C2 1
#define EFSYS_OPT_PHY_SFX7101 1
#define EFSYS_OPT_PHY_TXC43128 1
#define EFSYS_OPT_PHY_PM8358 1
#define EFSYS_OPT_PHY_SFT9001 1
```

```
#define EFSYS_OPT_PHY_QT2025C 1
#define EFSYS_OPT_PHY_STATS 1
#define EFSYS_OPT_PHY_PROPS 1
#define EFSYS_OPT_PHY_BIST 1
#define EFSYS_OPT_PHY_LED_CONTROL 1

#define EFSYS_OPT_VPD 1
#define EFSYS_OPT_NVRAM 1
#define EFSYS_OPT_NVRAM_FALCON_BOOTROM 1
#define EFSYS_OPT_NVRAM_SFT9001 0
#define EFSYS_OPT_NVRAM_SFX7101 0
#define EFSYS_OPT_BOOTCFG 1

#define EFSYS_OPT_PCIE_TUNE 1
#define EFSYS_OPT_DIAG 1
#define EFSYS_OPT_WOL 1
#define EFSYS_OPT_RX_SCALE 1
#define EFSYS_OPT_QSTATS 1

#define EFSYS_OPT_EV_PREFETCH 0

#define EFSYS_OPT_DECODE_INTR_FATAL 1

/* ID */

typedef struct __efsys_identifier_s     efsys_identifier_t;

/* DMA */

typedef uint64_t                efsys_dma_addr_t;

typedef struct efsys_mem_s {
        ddi_dma_handle_t        esm_dma_handle; /* DMA memory allocate/bind */
        ddi_acc_handle_t        esm_acc_handle; /* DMA memory read/write */
        caddr_t                 esm_base;
        efsys_dma_addr_t        esm_addr;
        size_t                  esm_size;
} efsys_mem_t;


#define EFSYS_MEM_ZERO(_esmp, _size)                                    \
        do {                                                            \
                (void) memset((_esmp)->esm_base, 0, (_size));           \
                                                                        \
        _NOTE(CONSTANTCONDITION)                                        \
        } while (B_FALSE)

#define EFSYS_MEM_READD(_esmp, _offset, _edp)                           \
        do {                                                            \
                uint32_t *addr;                                         \
                                                                        \
                _NOTE(CONSTANTCONDITION)                                \
                ASSERT(IS_P2ALIGNED(_offset, sizeof (efx_dword_t)));    \
                                                                        \
                addr = (void *)((_esmp)->esm_base + (_offset));         \
                                                                        \
                (_edp)->ed_u32[0] = ddi_get32((_esmp)->esm_acc_handle,  \
                    addr);                                              \
                                                                        \
                DTRACE_PROBE2(mem_readd, unsigned int, (_offset),       \
                    uint32_t, (_edp)->ed_u32[0]);                       \
                                                                        \
        _NOTE(CONSTANTCONDITION)                                        \
        } while (B_FALSE)

#define EFSYS_MEM_READQ(_esmp, _offset, _eqp)                           \
```



```
        do {                                                            \
                uint32_t *addr;                                         \
                                                                        \
                _NOTE(CONSTANTCONDITION)                                \
                ASSERT(IS_P2ALIGNED(_offset, sizeof (efx_qword_t)));    \
                                                                        \
                addr = (void *)((_esmp)->esm_base + (_offset));         \
                                                                        \
                (_eqp)->eq_u32[0] = ddi_get32((_esmp)->esm_acc_handle,  \
                    addr++);                                            \
                (_eqp)->eq_u32[1] = ddi_get32((_esmp)->esm_acc_handle,  \
                    addr);                                              \
                                                                        \
                DTRACE_PROBE3(mem_readq, unsigned int, (_offset),       \
                    uint32_t, (_eqp)->eq_u32[1],                        \
                    uint32_t, (_eqp)->eq_u32[0]);                       \
                                                                        \
        _NOTE(CONSTANTCONDITION)                                        \
        } while (B_FALSE)

#define EFSYS_MEM_READO(_esmp, _offset, _eop)                           \
        do {                                                            \
                uint32_t *addr;                                         \
                                                                        \
                _NOTE(CONSTANTCONDITION)                                \
                ASSERT(IS_P2ALIGNED(_offset, sizeof (efx_oword_t)));    \
                                                                        \
                addr = (void *)((_esmp)->esm_base + (_offset));         \
                                                                        \
                (_eop)->eo_u32[0] = ddi_get32((_esmp)->esm_acc_handle,  \
                    addr++);                                            \
                (_eop)->eo_u32[1] = ddi_get32((_esmp)->esm_acc_handle,  \
                    addr++);                                            \
                (_eop)->eo_u32[2] = ddi_get32((_esmp)->esm_acc_handle,  \
                    addr++);                                            \
                (_eop)->eo_u32[3] = ddi_get32((_esmp)->esm_acc_handle,  \
                    addr);                                              \
                                                                        \
                DTRACE_PROBE5(mem_reado, unsigned int, (_offset),       \
                    uint32_t, (_eop)->eo_u32[3],                        \
                    uint32_t, (_eop)->eo_u32[2],                        \
                    uint32_t, (_eop)->eo_u32[1],                        \
                    uint32_t, (_eop)->eo_u32[0]);                       \
                                                                        \
        _NOTE(CONSTANTCONDITION)                                        \
        } while (B_FALSE)

#define EFSYS_MEM_WRITED(_esmp, _offset, _edp)                          \
        do {                                                            \
                uint32_t *addr;                                         \
                                                                        \
                _NOTE(CONSTANTCONDITION)                                \
                ASSERT(IS_P2ALIGNED(_offset, sizeof (efx_dword_t)));    \
                                                                        \
                DTRACE_PROBE2(mem_writed, unsigned int, (_offset),      \
                    uint32_t, (_edp)->ed_u32[0]);                       \
                                                                        \
                addr = (void *)((_esmp)->esm_base + (_offset));         \
                                                                        \
                ddi_put32((_esmp)->esm_acc_handle, addr,                \
                    (_edp)->ed_u32[0]);                                 \
                                                                        \
        _NOTE(CONSTANTCONDITION)                                        \
        } while (B_FALSE)

#define EFSYS_MEM_WRITEQ(_esmp, _offset, _eqp)                          \
```

```
260         do {                                                        \
261                 uint32_t *addr;                                     \
262                                                                     \
263                 _NOTE(CONSTANTCONDITION)                            \
264                 ASSERT(IS_P2ALIGNED(_offset, sizeof (efx_qword_t)));    \
265                                                                     \
266                 DTRACE_PROBE3(mem_writeq, unsigned int, (_offset),      \
267                     uint32_t, (_eqp)->eq_u32[1],                    \
268                     uint32_t, (_eqp)->eq_u32[0]);                   \
269                                                                     \
270                 addr = (void *)((_esmp)->esm_base + (_offset));     \
271                                                                     \
272                 ddi_put32((_esmp)->esm_acc_handle, addr++,          \
273                     (_eqp)->eq_u32[0]);                             \
274                 ddi_put32((_esmp)->esm_acc_handle, addr,            \
275                     (_eqp)->eq_u32[1]);                             \
276                                                                     \
277         _NOTE(CONSTANTCONDITION)                                    \
278         } while (B_FALSE)

280 #define EFSYS_MEM_WRITEO(_esmp, _offset, _eop)                      \
281         do {                                                        \
282                 uint32_t *addr;                                     \
283                                                                     \
284                 _NOTE(CONSTANTCONDITION)                            \
285                 ASSERT(IS_P2ALIGNED(_offset, sizeof (efx_oword_t)));    \
286                                                                     \
287                 DTRACE_PROBE5(mem_writeo, unsigned int, (_offset),      \
288                     uint32_t, (_eop)->eo_u32[3],                    \
289                     uint32_t, (_eop)->eo_u32[2],                    \
290                     uint32_t, (_eop)->eo_u32[1],                    \
291                     uint32_t, (_eop)->eo_u32[0]);                   \
292                                                                     \
293                 addr = (void *)((_esmp)->esm_base + (_offset));     \
294                                                                     \
295                 ddi_put32((_esmp)->esm_acc_handle, addr++,          \
296                     (_eop)->eo_u32[0]);                             \
297                 ddi_put32((_esmp)->esm_acc_handle, addr++,          \
298                     (_eop)->eo_u32[1]);                             \
299                 ddi_put32((_esmp)->esm_acc_handle, addr++,          \
300                     (_eop)->eo_u32[2]);                             \
301                 ddi_put32((_esmp)->esm_acc_handle, addr,            \
302                     (_eop)->eo_u32[3]);                             \
303                                                                     \
304         _NOTE(CONSTANTCONDITION)                                    \
305         } while (B_FALSE)

307 #define EFSYS_MEM_ADDR(_esmp)                                       \
308         ((_esmp)->esm_addr)

310 /* BAR */

312 typedef struct efsys_bar_s {
313         kmutex_t                esb_lock;
314         ddi_acc_handle_t        esb_handle;
315         caddr_t                 esb_base;
316 } efsys_bar_t;

318 #define EFSYS_BAR_READD(_esbp, _offset, _edp, _lock)                \
319         do {                                                        \
320                 uint32_t *addr;                                     \
321                                                                     \
322                 _NOTE(CONSTANTCONDITION)                            \
323                 ASSERT(IS_P2ALIGNED(_offset, sizeof (efx_dword_t)));    \
324                                                                     \
325                 _NOTE(CONSTANTCONDITION)                            \
```



```
326                 if (_lock)                                          \
327                         mutex_enter(&((_esbp)->esb_lock));          \
328                                                                     \
329                 addr = (void *)((_esbp)->esb_base + (_offset));     \
330                                                                     \
331                 (_edp)->ed_u32[0] = ddi_get32((_esbp)->esb_handle,      \
332                     addr);                                          \
333                                                                     \
334                 DTRACE_PROBE2(bar_readd, unsigned int, (_offset),       \
335                     uint32_t, (_edp)->ed_u32[0]);                   \
336                                                                     \
337                 _NOTE(CONSTANTCONDITION)                            \
338                 if (_lock)                                          \
339                         mutex_exit(&((_esbp)->esb_lock));           \
340         _NOTE(CONSTANTCONDITION)                                    \
341         } while (B_FALSE)

343 #define EFSYS_BAR_READQ(_esbp, _offset, _eqp)                       \
344         do {                                                        \
345                 uint32_t *addr;                                     \
346                                                                     \
347                 _NOTE(CONSTANTCONDITION)                            \
348                 ASSERT(IS_P2ALIGNED(_offset, sizeof (efx_qword_t)));    \
349                                                                     \
350                 mutex_enter(&((_esbp)->esb_lock));                  \
351                                                                     \
352                 addr = (void *)((_esbp)->esb_base + (_offset));     \
353                                                                     \
354                 (_eqp)->eq_u32[0] = ddi_get32((_esbp)->esb_handle,      \
355                     addr++);                                        \
356                 (_eqp)->eq_u32[1] = ddi_get32((_esbp)->esb_handle,      \
357                     addr);                                          \
358                                                                     \
359                 DTRACE_PROBE3(bar_readq, unsigned int, (_offset),       \
360                     uint32_t, (_eqp)->eq_u32[1],                    \
361                     uint32_t, (_eqp)->eq_u32[0]);                   \
362                                                                     \
363                 mutex_exit(&((_esbp)->esb_lock));                   \
364         _NOTE(CONSTANTCONDITION)                                    \
365         } while (B_FALSE)

367 #define EFSYS_BAR_READO(_esbp, _offset, _eop, _lock)                \
368         do {                                                        \
369                 uint32_t *addr;                                     \
370                                                                     \
371                 _NOTE(CONSTANTCONDITION)                            \
372                 ASSERT(IS_P2ALIGNED(_offset, sizeof (efx_oword_t)));    \
373                                                                     \
374                 _NOTE(CONSTANTCONDITION)                            \
375                 if (_lock)                                          \
376                         mutex_enter(&((_esbp)->esb_lock));          \
377                                                                     \
378                 addr = (void *)((_esbp)->esb_base + (_offset));     \
379                                                                     \
380                 (_eop)->eo_u32[0] = ddi_get32((_esbp)->esb_handle,      \
381                     addr++);                                        \
382                 (_eop)->eo_u32[1] = ddi_get32((_esbp)->esb_handle,      \
383                     addr++);                                        \
384                 (_eop)->eo_u32[2] = ddi_get32((_esbp)->esb_handle,      \
385                     addr++);                                        \
386                 (_eop)->eo_u32[3] = ddi_get32((_esbp)->esb_handle,      \
387                     addr);                                          \
388                                                                     \
389                 DTRACE_PROBE5(bar_reado, unsigned int, (_offset),       \
390                     uint32_t, (_eop)->eo_u32[3],                    \
391                     uint32_t, (_eop)->eo_u32[2],                    \
```

```
392                     uint32_t, (_eop)->eo_u32[1],                    \
393                     uint32_t, (_eop)->eo_u32[0]);                   \
394                                                                     \
395                 _NOTE(CONSTANTCONDITION)                            \
396                 if (_lock)                                          \
397                         mutex_exit(&((_esbp)->esb_lock));           \
398         _NOTE(CONSTANTCONDITION)                                    \
399         } while (B_FALSE)

401 #define EFSYS_BAR_WRITED(_esbp, _offset, _edp, _lock)               \
402         do {                                                        \
403                 uint32_t *addr;                                     \
404                                                                     \
405                 _NOTE(CONSTANTCONDITION)                            \
406                 ASSERT(IS_P2ALIGNED(_offset, sizeof (efx_dword_t)));    \
407                                                                     \
408                 _NOTE(CONSTANTCONDITION)                            \
409                 if (_lock)                                          \
410                         mutex_enter(&((_esbp)->esb_lock));          \
411                                                                     \
412                 DTRACE_PROBE2(bar_writed, unsigned int, (_offset),      \
413                     uint32_t, (_edp)->ed_u32[0]);                   \
414                                                                     \
415                 addr = (void *)((_esbp)->esb_base + (_offset));     \
416                                                                     \
417                 ddi_put32((_esbp)->esb_handle, addr,                \
418                     (_edp)->ed_u32[0]);                             \
419                                                                     \
420                 _NOTE(CONSTANTCONDITION)                            \
421                 if (_lock)                                          \
422                         mutex_exit(&((_esbp)->esb_lock));           \
423         _NOTE(CONSTANTCONDITION)                                    \
424         } while (B_FALSE)

426 #define EFSYS_BAR_WRITEQ(_esbp, _offset, _eqp)                      \
427         do {                                                        \
428                 uint32_t *addr;                                     \
429                                                                     \
430                 _NOTE(CONSTANTCONDITION)                            \
431                 ASSERT(IS_P2ALIGNED(_offset, sizeof (efx_qword_t)));    \
432                                                                     \
433                 mutex_enter(&((_esbp)->esb_lock));                  \
434                                                                     \
435                 DTRACE_PROBE3(bar_writeq, unsigned int, (_offset),      \
436                     uint32_t, (_eqp)->eq_u32[1],                    \
437                     uint32_t, (_eqp)->eq_u32[0]);                   \
438                                                                     \
439                 addr = (void *)((_esbp)->esb_base + (_offset));     \
440                                                                     \
441                 ddi_put32((_esbp)->esb_handle, addr++,              \
442                     (_eqp)->eq_u32[0]);                             \
443                 ddi_put32((_esbp)->esb_handle, addr,                \
444                     (_eqp)->eq_u32[1]);                             \
445                                                                     \
446                 mutex_exit(&((_esbp)->esb_lock));                   \
447         _NOTE(CONSTANTCONDITION)                                    \
448         } while (B_FALSE)

450 #define EFSYS_BAR_WRITEO(_esbp, _offset, _eop, _lock)               \
451         do {                                                        \
452                 uint32_t *addr;                                     \
453                                                                     \
454                 _NOTE(CONSTANTCONDITION)                            \
455                 ASSERT(IS_P2ALIGNED(_offset, sizeof (efx_oword_t)));    \
456                                                                     \
457                 _NOTE(CONSTANTCONDITION)                            \
```



```
458                 if (_lock)                                          \
459                         mutex_enter(&((_esbp)->esb_lock));          \
460                                                                     \
461                 DTRACE_PROBE5(bar_writeo, unsigned int, (_offset),      \
462                     uint32_t, (_eop)->eo_u32[3],                    \
463                     uint32_t, (_eop)->eo_u32[2],                    \
464                     uint32_t, (_eop)->eo_u32[1],                    \
465                     uint32_t, (_eop)->eo_u32[0]);                   \
466                                                                     \
467                 addr = (void *)((_esbp)->esb_base + (_offset));     \
468                                                                     \
469                 ddi_put32((_esbp)->esb_handle, addr++,              \
470                     (_eop)->eo_u32[0]);                             \
471                 ddi_put32((_esbp)->esb_handle, addr++,              \
472                     (_eop)->eo_u32[1]);                             \
473                 ddi_put32((_esbp)->esb_handle, addr++,              \
474                     (_eop)->eo_u32[2]);                             \
475                 ddi_put32((_esbp)->esb_handle, addr,                \
476                     (_eop)->eo_u32[3]);                             \
477                                                                     \
478                 _NOTE(CONSTANTCONDITION)                            \
479                 if (_lock)                                          \
480                         mutex_exit(&((_esbp)->esb_lock));           \
481         _NOTE(CONSTANTCONDITION)                                    \
482         } while (B_FALSE)

484 /* SPIN */

486 #define EFSYS_SPIN(_us)                                             \
487         do {                                                        \
488                 drv_usecwait(_us);                                  \
489         _NOTE(CONSTANTCONDITION)                                    \
490         } while (B_FALSE)

492 #define EFSYS_SLEEP     EFSYS_SPIN

494 /* BARRIERS */

496 /* Strict ordering guaranteed by devacc.devacc_attr_dataorder */
497 #define EFSYS_MEM_READ_BARRIER()        membar_consumer()
498 /* TODO: Is ddi_put32() properly barriered? */
499 #define EFSYS_PIO_WRITE_BARRIER()

501 /* TIMESTAMP */

503 typedef clock_t efsys_timestamp_t;

505 #define EFSYS_TIMESTAMP(_usp)                                       \
506         do {                                                        \
507                 clock_t now;                                        \
508                                                                     \
509                 now = ddi_get_lbolt();                              \
510                 *(_usp) = drv_hztousec(now);                        \
511         _NOTE(CONSTANTCONDITION)                                    \
512         } while (B_FALSE)

514 /* KMEM */

516 #define EFSYS_KMEM_ALLOC(_esip, _size, _p)                          \
517         do {                                                        \
518                 (_esip) = (_esip);                                  \
519                 (_p) = kmem_zalloc((_size), KM_NOSLEEP);            \
520         _NOTE(CONSTANTCONDITION)                                    \
521         } while (B_FALSE)

523 #define EFSYS_KMEM_FREE(_esip, _size, _p)                           \
```

```
524         do {                                                        \
525                 (_esip) = (_esip);                                  \
526                 kmem_free((_p), (_size));                           \
527         _NOTE(CONSTANTCONDITION)                                    \
528         } while (B_FALSE)

530 /* LOCK */

532 typedef kmutex_t        efsys_lock_t;

534 #define EFSYS_LOCK_MAGIC        0x000010c4

536 #define EFSYS_LOCK(_lockp, _state)                                  \
537         do {                                                        \
538                 mutex_enter(_lockp);                                \
539                 (_state) = EFSYS_LOCK_MAGIC;                        \
540         _NOTE(CONSTANTCONDITION)                                    \
541         } while (B_FALSE)

543 #define EFSYS_UNLOCK(_lockp, _state)                                \
544         do {                                                        \
545                 if ((_state) != EFSYS_LOCK_MAGIC)                   \
546                         ASSERT(B_FALSE);                            \
547                 mutex_exit(_lockp);                                 \
548         _NOTE(CONSTANTCONDITION)                                    \
549         } while (B_FALSE)

551 /* PREEMPT */

553 #define EFSYS_PREEMPT_DISABLE(_state)                               \
554         do {                                                        \
555                 (_state) = ddi_enter_critical();                    \
556         _NOTE(CONSTANTCONDITION)                                    \
557         } while (B_FALSE)

559 #define EFSYS_PREEMPT_ENABLE(_state)                                \
560         do {                                                        \
561                 ddi_exit_critical(_state);                          \
562         _NOTE(CONSTANTCONDITION)                                    \
563         } while (B_FALSE)

565 /* STAT */

567 typedef kstat_named_t           efsys_stat_t;

569 #define EFSYS_STAT_INCR(_knp, _delta)                               \
570         do {                                                        \
571                 ((_knp)->value.ui64) += (_delta);                   \
572         _NOTE(CONSTANTCONDITION)                                    \
573         } while (B_FALSE)

575 #define EFSYS_STAT_DECR(_knp, _delta)                               \
576         do {                                                        \
577                 ((_knp)->value.ui64) -= (_delta);                   \
578         _NOTE(CONSTANTCONDITION)                                    \
579         } while (B_FALSE)

581 #define EFSYS_STAT_SET(_knp, _val)                                  \
582         do {                                                        \
583                 ((_knp)->value.ui64) = (_val);                      \
584         _NOTE(CONSTANTCONDITION)                                    \
585         } while (B_FALSE)

587 #define EFSYS_STAT_SET_QWORD(_knp, _valp)                           \
588         do {                                                        \
589                 ((_knp)->value.ui64) = LE_64((_valp)->eq_u64[0]);   \
```



```
590         _NOTE(CONSTANTCONDITION)                                    \
591         } while (B_FALSE)

593 #define EFSYS_STAT_SET_DWORD(_knp, _valp)                           \
594         do {                                                        \
595                 ((_knp)->value.ui64) = LE_32((_valp)->ed_u32[0]);   \
596         _NOTE(CONSTANTCONDITION)                                    \
597         } while (B_FALSE)

599 #define EFSYS_STAT_INCR_QWORD(_knp, _valp)                          \
600         do {                                                        \
601                 ((_knp)->value.ui64) += LE_64((_valp)->eq_u64[0]);  \
602         _NOTE(CONSTANTCONDITION)                                    \
603         } while (B_FALSE)

605 #define EFSYS_STAT_SUBR_QWORD(_knp, _valp)                          \
606         do {                                                        \
607                 ((_knp)->value.ui64) -= LE_64((_valp)->eq_u64[0]);  \
608         _NOTE(CONSTANTCONDITION)                                    \
609         } while (B_FALSE)

611 /* ERR */

613 extern void     sfxge_err(efsys_identifier_t *, unsigned int,
614                     uint32_t, uint32_t);

616 #if EFSYS_OPT_DECODE_INTR_FATAL
617 #define EFSYS_ERR(_esip, _code, _dword0, _dword1)                   \
618         do {                                                        \
619                 sfxge_err((_esip), (_code), (_dword0), (_dword1));  \
620         _NOTE(CONSTANTCONDITION)                                    \
621         } while (B_FALSE)
622 #endif

624 /* PROBE */

626 #define EFSYS_PROBE(_name)                                          \
627         DTRACE_PROBE(_name)

629 #define EFSYS_PROBE1(_name, _type1, _arg1)                          \
630         DTRACE_PROBE1(_name, _type1, _arg1)

632 #define EFSYS_PROBE2(_name, _type1, _arg1, _type2, _arg2)           \
633         DTRACE_PROBE2(_name, _type1, _arg1, _type2, _arg2)

635 #define EFSYS_PROBE3(_name, _type1, _arg1, _type2, _arg2,           \
636             _type3, _arg3)                                          \
637         DTRACE_PROBE3(_name, _type1, _arg1, _type2, _arg2,          \
638             _type3, _arg3)

640 #define EFSYS_PROBE4(_name, _type1, _arg1, _type2, _arg2,           \
641             _type3, _arg3, _type4, _arg4)                           \
642         DTRACE_PROBE4(_name, _type1, _arg1, _type2, _arg2,          \
643             _type3, _arg3, _type4, _arg4)

645 #define EFSYS_PROBE5(_name, _type1, _arg1, _type2, _arg2,           \
646             _type3, _arg3, _type4, _arg4, _type5, _arg5)            \
647         DTRACE_PROBE5(_name, _type1, _arg1, _type2, _arg2,          \
648             _type3, _arg3, _type4, _arg4, _type5, _arg5)

650 #ifdef DTRACE_PROBE6
651 #define EFSYS_PROBE6(_name, _type1, _arg1, _type2, _arg2,           \
652             _type3, _arg3, _type4, _arg4, _type5, _arg5,            \
653             _type6, _arg6)                                          \
654         DTRACE_PROBE6(_name, _type1, _arg1, _type2, _arg2,          \
655             _type3, _arg3, _type4, _arg4, _type5, _arg5,            \
```

```
656             _type6, _arg6)
657 #else
658 #define EFSYS_PROBE6(_name, _type1, _arg1, _type2, _arg2,               \
659             _type3, _arg3, _type4, _arg4, _type5, _arg5,                \
660             _type6, _arg6)                                              \
661         DTRACE_PROBE5(_name, _type1, _arg1, _type2, _arg2,              \
662             _type3, _arg3, _type4, _arg4, _type5, _arg5)
663 #endif

665 #ifdef DTRACE_PROBE7
666 #define EFSYS_PROBE7(_name, _type1, _arg1, _type2, _arg2,               \
667             _type3, _arg3, _type4, _arg4, _type5, _arg5,                \
668             _type6, _arg6, _type7, _arg7)                               \
669         DTRACE_PROBE7(_name, _type1, _arg1, _type2, _arg2,              \
670             _type3, _arg3, _type4, _arg4, _type5, _arg5,                \
671             _type6, _arg6, _type7, _arg7)
672 #else
673 #define EFSYS_PROBE7(_name, _type1, _arg1, _type2, _arg2,               \
674             _type3, _arg3, _type4, _arg4, _type5, _arg5,                \
675             _type6, _arg6, _type7, _arg7)                               \
676         DTRACE_PROBE5(_name, _type1, _arg1, _type2, _arg2,              \
677             _type3, _arg3, _type4, _arg4, _type5, _arg5)
678 #endif

680 /* ASSERT */

682 #define EFSYS_ASSERT(_exp)              ASSERT(_exp)
683 #define EFSYS_ASSERT3U(_x, _op, _y)     ASSERT3U(_x, _op, _y)
684 #define EFSYS_ASSERT3S(_x, _op, _y)     ASSERT3S(_x, _op, _y)
685 #define EFSYS_ASSERT3P(_x, _op, _y)     ASSERT3P(_x, _op, _y)

687 #ifdef  __cplusplus
688 }
689 #endif

691 #endif  /* _SYS_EFSYS_H */
692 #endif /* ! codereview */
```

```
**********************************************************
   41723 Thu Aug 22 18:59:21 2013
new/usr/src/uts/common/io/sfxge/efx.h
Merged sfxge driver
**********************************************************
```

```
/*-
 * Copyright 2006-2013 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#ifndef _SYS_EFX_H
#define _SYS_EFX_H

#include "efsys.h"

#ifdef  __cplusplus
extern "C" {
#endif

#define EFX_STATIC_ASSERT(_cond) ((void)sizeof(char[(_cond) ? 1 : -1]))

#define EFX_ARRAY_SIZE(_array) (sizeof(_array) / sizeof((_array)[0]))

#ifndef EFSYS_MEM_IS_NULL
#define EFSYS_MEM_IS_NULL(_esmp) ((_esmp)->esm_base == NULL)
#endif

typedef enum efx_family_e {
        EFX_FAMILY_INVALID,
        EFX_FAMILY_FALCON,
        EFX_FAMILY_SIENA,
        EFX_FAMILY_NTYPES
} efx_family_t;

extern  __checkReturn   int
efx_family(
        __in            uint16_t venid,
        __in            uint16_t devid,
        __out           efx_family_t *efp);

extern  __checkReturn   int
efx_infer_family(
        __in            efsys_bar_t *esbp,
        __out           efx_family_t *efp);

#define EFX_PCI_VENID_SFC               0x1924
```



```
#define EFX_PCI_DEVID_FALCON            0x0710
#define EFX_PCI_DEVID_BETHPAGE          0x0803
#define EFX_PCI_DEVID_SIENA             0x0813
#define EFX_PCI_DEVID_SIENA_F1_UNINIT   0x0810

#define EFX_MEM_BAR     2

/* Error codes */

enum {
        EFX_ERR_INVALID,
        EFX_ERR_SRAM_OOB,
        EFX_ERR_BUFID_DC_OOB,
        EFX_ERR_MEM_PERR,
        EFX_ERR_RBUF_OWN,
        EFX_ERR_TBUF_OWN,
        EFX_ERR_RDESQ_OWN,
        EFX_ERR_TDESQ_OWN,
        EFX_ERR_EVQ_OWN,
        EFX_ERR_EVFF_OFLO,
        EFX_ERR_ILL_ADDR,
        EFX_ERR_SRAM_PERR,
        EFX_ERR_NCODES
};

/* NIC */

typedef struct efx_nic_s        efx_nic_t;

extern  __checkReturn   int
efx_nic_create(
        __in            efx_family_t family,
        __in            efsys_identifier_t *esip,
        __in            efsys_bar_t *esbp,
        __in            efsys_lock_t *eslp,
        __deref_out     efx_nic_t **enpp);

extern  __checkReturn   int
efx_nic_probe(
        __in            efx_nic_t *enp);

#if EFSYS_OPT_PCIE_TUNE

extern  __checkReturn   int
efx_nic_pcie_tune(
        __in            efx_nic_t *enp,
        unsigned int    nlanes);

extern  __checkReturn   int
efx_nic_pcie_extended_sync(
        __in            efx_nic_t *enp);

#endif  /* EFSYS_OPT_PCIE_TUNE */

extern  __checkReturn   int
efx_nic_init(
        __in            efx_nic_t *enp);

extern  __checkReturn   int
efx_nic_reset(
        __in            efx_nic_t *enp);

#if EFSYS_OPT_DIAG

extern  __checkReturn   int
efx_nic_register_test(
```

```
128         __in            efx_nic_t *enp);

130 #endif  /* EFSYS_OPT_DIAG */

132 extern          void
133 efx_nic_fini(
134         __in            efx_nic_t *enp);

136 extern          void
137 efx_nic_unprobe(
138         __in            efx_nic_t *enp);

140 extern          void
141 efx_nic_destroy(
142         __in    efx_nic_t *enp);

144 #if EFSYS_OPT_MCDI

146 typedef struct efx_mcdi_req_s efx_mcdi_req_t;

148 typedef enum efx_mcdi_exception_e {
149         EFX_MCDI_EXCEPTION_MC_REBOOT,
150         EFX_MCDI_EXCEPTION_MC_BADASSERT,
151 } efx_mcdi_exception_t;

153 typedef struct efx_mcdi_transport_s {
154         void            *emt_context;
155         void            (*emt_execute)(void *, efx_mcdi_req_t *);
156         void            (*emt_ev_cpl)(void *);
157         void            (*emt_exception)(void *, efx_mcdi_exception_t);
158 } efx_mcdi_transport_t;

160 extern  __checkReturn   int
161 efx_mcdi_init(
162         __in            efx_nic_t *enp,
163         __in            const efx_mcdi_transport_t *mtp);

165 extern  __checkReturn   int
166 efx_mcdi_reboot(
167         __in            efx_nic_t *enp);

169 extern                  void
170 efx_mcdi_request_start(
171         __in            efx_nic_t *enp,
172         __in            efx_mcdi_req_t *emrp,
173         __in            boolean_t ev_cpl);

175 extern  __checkReturn   boolean_t
176 efx_mcdi_request_poll(
177         __in            efx_nic_t *enp);

179 extern  __checkReturn   boolean_t
180 efx_mcdi_request_abort(
181         __in            efx_nic_t *enp);

183 extern                  void
184 efx_mcdi_fini(
185         __in            efx_nic_t *enp);

187 #endif  /* EFSYS_OPT_MCDI */

189 /* INTR */

191 #define EFX_NINTR_FALCON 64
192 #define EFX_NINTR_SIENA 1024
```



```
194 typedef enum efx_intr_type_e {
195         EFX_INTR_INVALID = 0,
196         EFX_INTR_LINE,
197         EFX_INTR_MESSAGE,
198         EFX_INTR_NTYPES
199 } efx_intr_type_t;

201 #define EFX_INTR_SIZE   (sizeof (efx_oword_t))

203 extern  __checkReturn   int
204 efx_intr_init(
205         __in            efx_nic_t *enp,
206         __in            efx_intr_type_t type,
207         __in            efsys_mem_t *esmp);

209 extern                  void
210 efx_intr_enable(
211         __in            efx_nic_t *enp);

213 extern                  void
214 efx_intr_disable(
215         __in            efx_nic_t *enp);

217 extern                  void
218 efx_intr_disable_unlocked(
219         __in            efx_nic_t *enp);

221 #define EFX_INTR_NEVQS  32

223 extern __checkReturn    int
224 efx_intr_trigger(
225         __in            efx_nic_t *enp,
226         __in            unsigned int level);

228 extern                  void
229 efx_intr_status_line(
230         __in            efx_nic_t *enp,
231         __out           boolean_t *fatalp,
232         __out           uint32_t *maskp);

234 extern                  void
235 efx_intr_status_message(
236         __in            efx_nic_t *enp,
237         __in            unsigned int message,
238         __out           boolean_t *fatalp);

240 extern                  void
241 efx_intr_fatal(
242         __in            efx_nic_t *enp);

244 extern                  void
245 efx_intr_fini(
246         __in            efx_nic_t *enp);

248 /* MAC */

250 #if EFSYS_OPT_MAC_STATS

252 /* START MKCONFIG GENERATED EfxHeaderMacBlock bb8d39428b6fdcf5 */
253 typedef enum efx_mac_stat_e {
254         EFX_MAC_RX_OCTETS,
255         EFX_MAC_RX_PKTS,
256         EFX_MAC_RX_UNICST_PKTS,
257         EFX_MAC_RX_MULTICST_PKTS,
258         EFX_MAC_RX_BRDCST_PKTS,
259         EFX_MAC_RX_PAUSE_PKTS,
```

```
260         EFX_MAC_RX_LE_64_PKTS,
261         EFX_MAC_RX_65_TO_127_PKTS,
262         EFX_MAC_RX_128_TO_255_PKTS,
263         EFX_MAC_RX_256_TO_511_PKTS,
264         EFX_MAC_RX_512_TO_1023_PKTS,
265         EFX_MAC_RX_1024_TO_15XX_PKTS,
266         EFX_MAC_RX_GE_15XX_PKTS,
267         EFX_MAC_RX_ERRORS,
268         EFX_MAC_RX_FCS_ERRORS,
269         EFX_MAC_RX_DROP_EVENTS,
270         EFX_MAC_RX_FALSE_CARRIER_ERRORS,
271         EFX_MAC_RX_SYMBOL_ERRORS,
272         EFX_MAC_RX_ALIGN_ERRORS,
273         EFX_MAC_RX_INTERNAL_ERRORS,
274         EFX_MAC_RX_JABBER_PKTS,
275         EFX_MAC_RX_LANE0_CHAR_ERR,
276         EFX_MAC_RX_LANE1_CHAR_ERR,
277         EFX_MAC_RX_LANE2_CHAR_ERR,
278         EFX_MAC_RX_LANE3_CHAR_ERR,
279         EFX_MAC_RX_LANE0_DISP_ERR,
280         EFX_MAC_RX_LANE1_DISP_ERR,
281         EFX_MAC_RX_LANE2_DISP_ERR,
282         EFX_MAC_RX_LANE3_DISP_ERR,
283         EFX_MAC_RX_MATCH_FAULT,
284         EFX_MAC_RX_NODESC_DROP_CNT,
285         EFX_MAC_TX_OCTETS,
286         EFX_MAC_TX_PKTS,
287         EFX_MAC_TX_UNICST_PKTS,
288         EFX_MAC_TX_MULTICST_PKTS,
289         EFX_MAC_TX_BRDCST_PKTS,
290         EFX_MAC_TX_PAUSE_PKTS,
291         EFX_MAC_TX_LE_64_PKTS,
292         EFX_MAC_TX_65_TO_127_PKTS,
293         EFX_MAC_TX_128_TO_255_PKTS,
294         EFX_MAC_TX_256_TO_511_PKTS,
295         EFX_MAC_TX_512_TO_1023_PKTS,
296         EFX_MAC_TX_1024_TO_15XX_PKTS,
297         EFX_MAC_TX_GE_15XX_PKTS,
298         EFX_MAC_TX_ERRORS,
299         EFX_MAC_TX_SGL_COL_PKTS,
300         EFX_MAC_TX_MULT_COL_PKTS,
301         EFX_MAC_TX_EX_COL_PKTS,
302         EFX_MAC_TX_LATE_COL_PKTS,
303         EFX_MAC_TX_DEF_PKTS,
304         EFX_MAC_TX_EX_DEF_PKTS,
305         EFX_MAC_NSTATS
306 } efx_mac_stat_t;

308 /* END MKCONFIG GENERATED EfxHeaderMacBlock */

310 #endif  /* EFSYS_OPT_MAC_STATS */

312 typedef enum efx_link_mode_e {
313         EFX_LINK_UNKNOWN = 0,
314         EFX_LINK_DOWN,
315         EFX_LINK_10HDX,
316         EFX_LINK_10FDX,
317         EFX_LINK_100HDX,
318         EFX_LINK_100FDX,
319         EFX_LINK_1000HDX,
320         EFX_LINK_1000FDX,
321         EFX_LINK_10000FDX,
322         EFX_LINK_NMODES
323 } efx_link_mode_t;

325 #define EFX_MAC_SDU_MAX 9202
```



```
327 #define EFX_MAC_PDU(_sdu)                               \
328         P2ROUNDUP(((_sdu)                               \
329                   + /* EtherII */ 14                     \
330                   + /* VLAN */ 4                         \
331                   + /* CRC */ 4                          \
332                   + /* bug16011 */ 16),                  \
333                   (1 << 3))

335 #define EFX_MAC_PDU_MIN 60
336 #define EFX_MAC_PDU_MAX EFX_MAC_PDU(EFX_MAC_SDU_MAX)

338 extern  __checkReturn   int
339 efx_mac_pdu_set(
340         __in            efx_nic_t *enp,
341         __in            size_t pdu);

343 extern  __checkReturn   int
344 efx_mac_addr_set(
345         __in            efx_nic_t *enp,
346         __in            uint8_t *addr);

348 extern  __checkReturn   int
349 efx_mac_filter_set(
350         __in            efx_nic_t *enp,
351         __in            boolean_t unicst,
352         __in            boolean_t brdcst);

354 extern  __checkReturn   int
355 efx_mac_drain(
356         __in            efx_nic_t *enp,
357         __in            boolean_t enabled);

359 extern  __checkReturn   int
360 efx_mac_up(
361         __in            efx_nic_t *enp,
362         __out           boolean_t *mac_upp);

364 #define EFX_FCNTL_RESPOND       0x00000001
365 #define EFX_FCNTL_GENERATE      0x00000002

367 extern  __checkReturn   int
368 efx_mac_fcntl_set(
369         __in            efx_nic_t *enp,
370         __in            unsigned int fcntl,
371         __in            boolean_t autoneg);

373 extern                  void
374 efx_mac_fcntl_get(
375         __in            efx_nic_t *enp,
376         __out           unsigned int *fcntl_wantedp,
377         __out           unsigned int *fcntl_linkp);

379 #define EFX_MAC_HASH_BITS       (1 << 8)

381 extern  __checkReturn                   int
382 efx_mac_hash_set(
383         __in                            efx_nic_t *enp,
384         __in_ecount(EFX_MAC_HASH_BITS)  unsigned int const *bucket);

386 #if EFSYS_OPT_MAC_STATS

388 #if EFSYS_OPT_NAMES

390 extern  __checkReturn                   const char __cs *
391 efx_mac_stat_name(
```

```
392         __in                            efx_nic_t *enp,
393         __in                            unsigned int id);

395 #endif  /* EFSYS_OPT_NAMES */

397 #define EFX_MAC_STATS_SIZE 0x400

399 /*
400  * Upload mac statistics supported by the hardware into the given buffer.
401  *
402  * The reference buffer must be at least %EFX_MAC_STATS_SIZE bytes,
403  * and page aligned.
404  *
405  * The hardware will only DMA statistics that it understands (of course).
406  * Drivers should not make any assumptions about which statistics are
407  * supported, especially when the statistics are generated by firmware.
408  *
409  * Thus, drivers should zero this buffer before use, so that not-understood
410  * statistics read back as zero.
411  */
412 extern  __checkReturn                   int
413 efx_mac_stats_upload(
414         __in                            efx_nic_t *enp,
415         __in                            efsys_mem_t *esmp);

417 extern  __checkReturn                   int
418 efx_mac_stats_periodic(
419         __in                            efx_nic_t *enp,
420         __in                            efsys_mem_t *esmp,
421         __in                            uint16_t period_ms,
422         __in                            boolean_t events);

424 extern  __checkReturn                   int
425 efx_mac_stats_update(
426         __in                            efx_nic_t *enp,
427         __in                            efsys_mem_t *esmp,
428         __inout_ecount(EFX_MAC_NSTATS)  efsys_stat_t *stat,
429         __out_opt                       uint32_t *generationp);

431 #endif  /* EFSYS_OPT_MAC_STATS */

433 /* MON */

435 typedef enum efx_mon_type_e {
436         EFX_MON_INVALID = 0,
437         EFX_MON_NULL,
438         EFX_MON_LM87,
439         EFX_MON_MAX6647,
440         EFX_MON_SFC90X0,
441         EFX_MON_NTYPES
442 } efx_mon_type_t;

444 #if EFSYS_OPT_NAMES

446 extern          const char __cs *
447 efx_mon_name(
448         __in    efx_nic_t *enp);

450 #endif  /* EFSYS_OPT_NAMES */

452 extern  __checkReturn   int
453 efx_mon_init(
454         __in            efx_nic_t *enp);

456 #if EFSYS_OPT_MON_STATS
```



```
458 #define EFX_MON_STATS_SIZE 0x100

460 /* START MKCONFIG GENERATED MonitorHeaderStatsBlock 2a60d293f55d0285 */
461 typedef enum efx_mon_stat_e {
462         EFX_MON_STAT_2_5V,
463         EFX_MON_STAT_VCCP1,
464         EFX_MON_STAT_VCC,
465         EFX_MON_STAT_5V,
466         EFX_MON_STAT_12V,
467         EFX_MON_STAT_VCCP2,
468         EFX_MON_STAT_EXT_TEMP,
469         EFX_MON_STAT_INT_TEMP,
470         EFX_MON_STAT_AIN1,
471         EFX_MON_STAT_AIN2,
472         EFX_MON_STAT_INT_COOLING,
473         EFX_MON_STAT_EXT_COOLING,
474         EFX_MON_STAT_1V,
475         EFX_MON_STAT_1_2V,
476         EFX_MON_STAT_1_8V,
477         EFX_MON_STAT_3_3V,
478         EFX_MON_STAT_1_2VA,
479         EFX_MON_STAT_VREF,
480         EFX_MON_NSTATS
481 } efx_mon_stat_t;

483 /* END MKCONFIG GENERATED MonitorHeaderStatsBlock */

485 typedef enum efx_mon_stat_state_e {
486         EFX_MON_STAT_STATE_OK = 0,
487         EFX_MON_STAT_STATE_WARNING = 1,
488         EFX_MON_STAT_STATE_FATAL = 2,
489         EFX_MON_STAT_STATE_BROKEN = 3,
490 } efx_mon_stat_state_t;

492 typedef struct efx_mon_stat_value_t {
493         uint16_t        emsv_value;
494         uint16_t        emsv_state;
495 } efx_mon_stat_value_t;

497 #if EFSYS_OPT_NAMES

499 extern                                  const char __cs *
500 efx_mon_stat_name(
501         __in                            efx_nic_t *enp,
502         __in                            efx_mon_stat_t id);

504 #endif  /* EFSYS_OPT_NAMES */

506 extern  __checkReturn                   int
507 efx_mon_stats_update(
508         __in                            efx_nic_t *enp,
509         __in                            efsys_mem_t *esmp,
510         __out_ecount(EFX_MON_NSTATS)    efx_mon_stat_value_t *values);

512 #endif  /* EFSYS_OPT_MON_STATS */

514 extern          void
515 efx_mon_fini(
516         __in    efx_nic_t *enp);

518 /* PHY */

520 #define PMA_PMD_MMD     1
521 #define PCS_MMD         3
522 #define PHY_XS_MMD      4
523 #define DTE_XS_MMD      5
```

```
524 #define AN_MMD          7
525 #define CL22EXT_MMD     29

527 #define MAXMMD          ((1 << 5) - 1)

529 /* PHY types */
530 #define EFX_PHY_NULL            0x0
531 #define EFX_PHY_TXC43128        0x1
532 #define EFX_PHY_SFX7101         0x3
533 #define EFX_PHY_QT2022C2        0x4
534 #define EFX_PHY_SFT9001A        0x8
535 #define EFX_PHY_QT2025C         0x9
536 #define EFX_PHY_SFT9001B        0xa
537 #define EFX_PHY_QLX111V         0xc

539 extern  __checkReturn   int
540 efx_phy_verify(
541         __in            efx_nic_t *enp);

543 #if EFSYS_OPT_PHY_LED_CONTROL

545 typedef enum efx_phy_led_mode_e {
546         EFX_PHY_LED_DEFAULT = 0,
547         EFX_PHY_LED_OFF,
548         EFX_PHY_LED_ON,
549         EFX_PHY_LED_FLASH,
550         EFX_PHY_LED_NMODES
551 } efx_phy_led_mode_t;

553 extern  __checkReturn   int
554 efx_phy_led_set(
555         __in    efx_nic_t *enp,
556         __in    efx_phy_led_mode_t mode);

558 #endif  /* EFSYS_OPT_PHY_LED_CONTROL */

560 extern  __checkReturn   int
561 efx_port_init(
562         __in            efx_nic_t *enp);

564 #if EFSYS_OPT_LOOPBACK

566 typedef enum efx_loopback_type_e {
567         EFX_LOOPBACK_OFF = 0,
568         EFX_LOOPBACK_DATA = 1,
569         EFX_LOOPBACK_GMAC = 2,
570         EFX_LOOPBACK_XGMII = 3,
571         EFX_LOOPBACK_XGXS = 4,
572         EFX_LOOPBACK_XAUI = 5,
573         EFX_LOOPBACK_GMII = 6,
574         EFX_LOOPBACK_SGMII = 7,
575         EFX_LOOPBACK_XGBR = 8,
576         EFX_LOOPBACK_XFI = 9,
577         EFX_LOOPBACK_XAUI_FAR = 10,
578         EFX_LOOPBACK_GMII_FAR = 11,
579         EFX_LOOPBACK_SGMII_FAR = 12,
580         EFX_LOOPBACK_XFI_FAR = 13,
581         EFX_LOOPBACK_GPHY = 14,
582         EFX_LOOPBACK_PHY_XS = 15,
583         EFX_LOOPBACK_PCS = 16,
584         EFX_LOOPBACK_PMA_PMD = 17,
585         EFX_LOOPBACK_NTYPES
586 } efx_loopback_type_t;

588 #define EFX_LOOPBACK_MAC_MASK                   \
589         ((1 << EFX_LOOPBACK_DATA) |             \
```



```
590             (1 << EFX_LOOPBACK_GMAC) |          \
591             (1 << EFX_LOOPBACK_XGMII) |         \
592             (1 << EFX_LOOPBACK_XGXS) |          \
593             (1 << EFX_LOOPBACK_XAUI) |          \
594             (1 << EFX_LOOPBACK_GMII) |          \
595             (1 << EFX_LOOPBACK_SGMII) |         \
596             (1 << EFX_LOOPBACK_XGBR) |          \
597             (1 << EFX_LOOPBACK_XFI) |           \
598             (1 << EFX_LOOPBACK_XAUI_FAR) |      \
599             (1 << EFX_LOOPBACK_GMII_FAR) |      \
600             (1 << EFX_LOOPBACK_SGMII_FAR) |     \
601             (1 << EFX_LOOPBACK_XFI_FAR))

603 #define EFX_LOOPBACK_MASK                       \
604         ((1 << EFX_LOOPBACK_NTYPES) - 1)

606 extern  __checkReturn   int
607 efx_port_loopback_set(
608         __in    efx_nic_t *enp,
609         __in    efx_link_mode_t link_mode,
610         __in    efx_loopback_type_t type);

612 #if EFSYS_OPT_NAMES

614 extern  __checkReturn   const char __cs *
615 efx_loopback_type_name(
616         __in            efx_nic_t *enp,
617         __in            efx_loopback_type_t type);

619 #endif  /* EFSYS_OPT_NAMES */

621 #endif  /* EFSYS_OPT_LOOPBACK */

623 extern  __checkReturn   int
624 efx_port_poll(
625         __in            efx_nic_t *enp,
626         __out           efx_link_mode_t *link_modep);

628 extern          void
629 efx_port_fini(
630         __in    efx_nic_t *enp);

632 typedef enum efx_phy_cap_type_e {
633         EFX_PHY_CAP_INVALID = 0,
634         EFX_PHY_CAP_10HDX,
635         EFX_PHY_CAP_10FDX,
636         EFX_PHY_CAP_100HDX,
637         EFX_PHY_CAP_100FDX,
638         EFX_PHY_CAP_1000HDX,
639         EFX_PHY_CAP_1000FDX,
640         EFX_PHY_CAP_10000FDX,
641         EFX_PHY_CAP_PAUSE,
642         EFX_PHY_CAP_ASYM,
643         EFX_PHY_CAP_AN,
644         EFX_PHY_CAP_NTYPES
645 } efx_phy_cap_type_t;


648 #define EFX_PHY_CAP_CURRENT     0x00000000
649 #define EFX_PHY_CAP_DEFAULT     0x00000001
650 #define EFX_PHY_CAP_PERM        0x00000002

652 extern          void
653 efx_phy_adv_cap_get(
654         __in            efx_nic_t *enp,
655         __in            uint32_t flag,
```

```
656         __out           uint32_t *maskp);

658 extern  __checkReturn   int
659 efx_phy_adv_cap_set(
660         __in            efx_nic_t *enp,
661         __in            uint32_t mask);

663 extern                  void
664 efx_phy_lp_cap_get(
665         __in            efx_nic_t *enp,
666         __out           uint32_t *maskp);

668 extern  __checkReturn   int
669 efx_phy_oui_get(
670         __in            efx_nic_t *enp,
671         __out           uint32_t *ouip);

673 typedef enum efx_phy_media_type_e {
674         EFX_PHY_MEDIA_INVALID = 0,
675         EFX_PHY_MEDIA_XAUI,
676         EFX_PHY_MEDIA_CX4,
677         EFX_PHY_MEDIA_KX4,
678         EFX_PHY_MEDIA_XFP,
679         EFX_PHY_MEDIA_SFP_PLUS,
680         EFX_PHY_MEDIA_BASE_T,
681         EFX_PHY_MEDIA_NTYPES
682 } efx_phy_media_type_t;

684 /* Get the type of medium currently used.  If the board has ports for
685  * modules, a module is present, and we recognise the media type of
686  * the module, then this will be the media type of the module.
687  * Otherwise it will be the media type of the port.
688  */
689 extern                  void
690 efx_phy_media_type_get(
691         __in            efx_nic_t *enp,
692         __out           efx_phy_media_type_t *typep);

694 #if EFSYS_OPT_PHY_STATS

696 /* START MKCONFIG GENERATED PhyHeaderStatsBlock 30ed56ad501f8e36 */
697 typedef enum efx_phy_stat_e {
698         EFX_PHY_STAT_OUI,
699         EFX_PHY_STAT_PMA_PMD_LINK_UP,
700         EFX_PHY_STAT_PMA_PMD_RX_FAULT,
701         EFX_PHY_STAT_PMA_PMD_TX_FAULT,
702         EFX_PHY_STAT_PMA_PMD_REV_A,
703         EFX_PHY_STAT_PMA_PMD_REV_B,
704         EFX_PHY_STAT_PMA_PMD_REV_C,
705         EFX_PHY_STAT_PMA_PMD_REV_D,
706         EFX_PHY_STAT_PCS_LINK_UP,
707         EFX_PHY_STAT_PCS_RX_FAULT,
708         EFX_PHY_STAT_PCS_TX_FAULT,
709         EFX_PHY_STAT_PCS_BER,
710         EFX_PHY_STAT_PCS_BLOCK_ERRORS,
711         EFX_PHY_STAT_PHY_XS_LINK_UP,
712         EFX_PHY_STAT_PHY_XS_RX_FAULT,
713         EFX_PHY_STAT_PHY_XS_TX_FAULT,
714         EFX_PHY_STAT_PHY_XS_ALIGN,
715         EFX_PHY_STAT_PHY_XS_SYNC_A,
716         EFX_PHY_STAT_PHY_XS_SYNC_B,
717         EFX_PHY_STAT_PHY_XS_SYNC_C,
718         EFX_PHY_STAT_PHY_XS_SYNC_D,
719         EFX_PHY_STAT_AN_LINK_UP,
720         EFX_PHY_STAT_AN_MASTER,
721         EFX_PHY_STAT_AN_LOCAL_RX_OK,
```



```
722         EFX_PHY_STAT_AN_REMOTE_RX_OK,
723         EFX_PHY_STAT_CL22EXT_LINK_UP,
724         EFX_PHY_STAT_SNR_A,
725         EFX_PHY_STAT_SNR_B,
726         EFX_PHY_STAT_SNR_C,
727         EFX_PHY_STAT_SNR_D,
728         EFX_PHY_STAT_PMA_PMD_SIGNAL_A,
729         EFX_PHY_STAT_PMA_PMD_SIGNAL_B,
730         EFX_PHY_STAT_PMA_PMD_SIGNAL_C,
731         EFX_PHY_STAT_PMA_PMD_SIGNAL_D,
732         EFX_PHY_STAT_AN_COMPLETE,
733         EFX_PHY_STAT_PMA_PMD_REV_MAJOR,
734         EFX_PHY_STAT_PMA_PMD_REV_MINOR,
735         EFX_PHY_STAT_PMA_PMD_REV_MICRO,
736         EFX_PHY_STAT_PCS_FW_VERSION_0,
737         EFX_PHY_STAT_PCS_FW_VERSION_1,
738         EFX_PHY_STAT_PCS_FW_VERSION_2,
739         EFX_PHY_STAT_PCS_FW_VERSION_3,
740         EFX_PHY_STAT_PCS_FW_BUILD_YY,
741         EFX_PHY_STAT_PCS_FW_BUILD_MM,
742         EFX_PHY_STAT_PCS_FW_BUILD_DD,
743         EFX_PHY_STAT_PCS_OP_MODE,
744         EFX_PHY_NSTATS
745 } efx_phy_stat_t;

747 /* END MKCONFIG GENERATED PhyHeaderStatsBlock */

749 #if EFSYS_OPT_NAMES

751 extern                                  const char __cs *
752 efx_phy_stat_name(
753         __in                            efx_nic_t *enp,
754         __in                            efx_phy_stat_t stat);

756 #endif  /* EFSYS_OPT_NAMES */

758 #define EFX_PHY_STATS_SIZE 0x100

760 extern  __checkReturn                   int
761 efx_phy_stats_update(
762         __in                            efx_nic_t *enp,
763         __in                            efsys_mem_t *esmp,
764         __out_ecount(EFX_PHY_NSTATS)    uint32_t *stat);

766 #endif  /* EFSYS_OPT_PHY_STATS */

768 #if EFSYS_OPT_PHY_PROPS

770 #if EFSYS_OPT_NAMES

772 extern          const char __cs *
773 efx_phy_prop_name(
774         __in    efx_nic_t *enp,
775         __in    unsigned int id);

777 #endif  /* EFSYS_OPT_NAMES */

779 #define EFX_PHY_PROP_DEFAULT    0x00000001

781 extern  __checkReturn   int
782 efx_phy_prop_get(
783         __in            efx_nic_t *enp,
784         __in            unsigned int id,
785         __in            uint32_t flags,
786         __out           uint32_t *valp);
```

```
788 extern  __checkReturn   int
789 efx_phy_prop_set(
790         __in            efx_nic_t *enp,
791         __in            unsigned int id,
792         __in            uint32_t val);

794 #endif  /* EFSYS_OPT_PHY_PROPS */

796 #if EFSYS_OPT_PHY_BIST

798 typedef enum efx_phy_bist_type_e {
799         EFX_PHY_BIST_TYPE_UNKNOWN,
800         EFX_PHY_BIST_TYPE_NORMAL,
801         EFX_PHY_BIST_TYPE_CABLE_SHORT,
802         EFX_PHY_BIST_TYPE_CABLE_LONG,
803         EFX_PHY_BIST_TYPE_NTYPES,
804 } efx_phy_bist_type_t;

806 typedef enum efx_phy_bist_result_e {
807         EFX_PHY_BIST_RESULT_UNKNOWN,
808         EFX_PHY_BIST_RESULT_RUNNING,
809         EFX_PHY_BIST_RESULT_PASSED,
810         EFX_PHY_BIST_RESULT_FAILED,
811 } efx_phy_bist_result_t;

813 typedef enum efx_phy_cable_status_e {
814         EFX_PHY_CABLE_STATUS_OK,
815         EFX_PHY_CABLE_STATUS_INVALID,
816         EFX_PHY_CABLE_STATUS_OPEN,
817         EFX_PHY_CABLE_STATUS_INTRAPAIRSHORT,
818         EFX_PHY_CABLE_STATUS_INTERPAIRSHORT,
819         EFX_PHY_CABLE_STATUS_BUSY,
820 } efx_phy_cable_status_t;

822 typedef enum efx_phy_bist_value_e {
823         EFX_PHY_BIST_CABLE_LENGTH_A,
824         EFX_PHY_BIST_CABLE_LENGTH_B,
825         EFX_PHY_BIST_CABLE_LENGTH_C,
826         EFX_PHY_BIST_CABLE_LENGTH_D,
827         EFX_PHY_BIST_CABLE_STATUS_A,
828         EFX_PHY_BIST_CABLE_STATUS_B,
829         EFX_PHY_BIST_CABLE_STATUS_C,
830         EFX_PHY_BIST_CABLE_STATUS_D,
831         EFX_PHY_BIST_FAULT_CODE,
832         EFX_PHY_BIST_NVALUES,
833 } efx_phy_bist_value_t;

835 extern  __checkReturn           int
836 efx_phy_bist_start(
837         __in                    efx_nic_t *enp,
838         __in                    efx_phy_bist_type_t type);

840 extern  __checkReturn           int
841 efx_phy_bist_poll(
842         __in                    efx_nic_t *enp,
843         __in                    efx_phy_bist_type_t type,
844         __out                   efx_phy_bist_result_t *resultp,
845         __out_opt               uint32_t *value_maskp,
846         __out_ecount_opt(count) unsigned long *valuesp,
847         __in                    size_t count);

849 extern                          void
850 efx_phy_bist_stop(
851         __in                    efx_nic_t *enp,
852         __in                    efx_phy_bist_type_t type);
```



```
854 #endif  /* EFSYS_OPT_PHY_BIST */

856 #define EFX_FEATURE_IPV6                0x00000001
857 #define EFX_FEATURE_LFSR_HASH_INSERT    0x00000002
858 #define EFX_FEATURE_LINK_EVENTS         0x00000004
859 #define EFX_FEATURE_PERIODIC_MAC_STATS  0x00000008
860 #define EFX_FEATURE_WOL                 0x00000010
861 #define EFX_FEATURE_MCDI                0x00000020
862 #define EFX_FEATURE_LOOKAHEAD_SPLIT     0x00000040
863 #define EFX_FEATURE_MAC_HEADER_FILTERS  0x00000080
864 #define EFX_FEATURE_TURBO               0x00000100

866 typedef struct efx_nic_cfg_s {
867         uint32_t                enc_board_type;
868         uint32_t                enc_phy_type;
869 #if EFSYS_OPT_NAMES
870         char                    enc_phy_name[21];
871 #endif
872         char                    enc_phy_revision[21];
873         efx_mon_type_t          enc_mon_type;
874 #if EFSYS_OPT_MON_STATS
875         uint32_t                enc_mon_stat_mask;
876 #endif
877         unsigned int            enc_features;
878         uint8_t                 enc_mac_addr[6];
879         uint8_t                 enc_port;
880         uint32_t                enc_evq_limit;
881         uint32_t                enc_txq_limit;
882         uint32_t                enc_rxq_limit;
883         uint32_t                enc_buftbl_limit;
884         uint32_t                enc_evq_timer_quantum_ns;
885         uint32_t                enc_evq_timer_max_us;
886         uint32_t                enc_clk_mult;
887 #if EFSYS_OPT_LOOPBACK
888         uint32_t                enc_loopback_types[EFX_LINK_NMODES];
889 #endif  /* EFSYS_OPT_LOOPBACK */
890 #if EFSYS_OPT_PHY_FLAGS
891         uint32_t                enc_phy_flags_mask;
892 #endif  /* EFSYS_OPT_PHY_FLAGS */
893 #if EFSYS_OPT_PHY_LED_CONTROL
894         uint32_t                enc_led_mask;
895 #endif  /* EFSYS_OPT_PHY_LED_CONTROL */
896 #if EFSYS_OPT_PHY_STATS
897         uint64_t                enc_phy_stat_mask;
898 #endif  /* EFSYS_OPT_PHY_STATS */
899 #if EFSYS_OPT_PHY_PROPS
900         unsigned int            enc_phy_nprops;
901 #endif  /* EFSYS_OPT_PHY_PROPS */
902 #if EFSYS_OPT_SIENA
903         uint8_t                 enc_siena_channel;
904 #if EFSYS_OPT_PHY_STATS
905         uint32_t                enc_siena_phy_stat_mask;
906 #endif  /* EFSYS_OPT_PHY_STATS */
907 #if EFSYS_OPT_MON_STATS
908         uint32_t                enc_siena_mon_stat_mask;
909 #endif  /* EFSYS_OPT_MON_STATS */
910 #endif  /* EFSYS_OPT_SIENA */
911 #if EFSYS_OPT_PHY_BIST
912         uint32_t                enc_bist_mask;
913 #endif  /* EFSYS_OPT_PHY_BIST */
914 } efx_nic_cfg_t;

916 extern                  const efx_nic_cfg_t *
917 efx_nic_cfg_get(
918         __in            efx_nic_t *enp);
```

```
#if EFSYS_OPT_VPD

typedef enum efx_vpd_tag_e {
        EFX_VPD_ID = 0x02,
        EFX_VPD_END = 0x0f,
        EFX_VPD_RO = 0x10,
        EFX_VPD_RW = 0x11,
} efx_vpd_tag_t;

typedef uint16_t efx_vpd_keyword_t;

typedef struct efx_vpd_value_s {
        efx_vpd_tag_t           evv_tag;
        efx_vpd_keyword_t       evv_keyword;
        uint8_t                 evv_length;
        uint8_t                 evv_value[0x100];
} efx_vpd_value_t;


#define EFX_VPD_KEYWORD(x, y) ((x) | ((y) << 8))

extern  __checkReturn           int
efx_vpd_init(
        __in                    efx_nic_t *enp);

extern  __checkReturn           int
efx_vpd_size(
        __in                    efx_nic_t *enp,
        __out                   size_t *sizep);

extern  __checkReturn           int
efx_vpd_read(
        __in                    efx_nic_t *enp,
        __out_bcount(size)      caddr_t data,
        __in                    size_t size);

extern  __checkReturn           int
efx_vpd_verify(
        __in                    efx_nic_t *enp,
        __in_bcount(size)       caddr_t data,
        __in                    size_t size);

extern  __checkReturn           int
efx_vpd_reinit(
        __in                    efx_nic_t *enp,
        __in_bcount(size)       caddr_t data,
        __in                    size_t size);

extern  __checkReturn           int
efx_vpd_get(
        __in                    efx_nic_t *enp,
        __in_bcount(size)       caddr_t data,
        __in                    size_t size,
        __inout                 efx_vpd_value_t *evvp);

extern  __checkReturn           int
efx_vpd_set(
        __in                    efx_nic_t *enp,
        __inout_bcount(size)    caddr_t data,
        __in                    size_t size,
        __in                    efx_vpd_value_t *evvp);

extern  __checkReturn           int
efx_vpd_next(
        __in                    efx_nic_t *enp,
        __inout_bcount(size)    caddr_t data,
```



```
        __in                    size_t size,
        __out                   efx_vpd_value_t *evvp,
        __inout                 unsigned int *contp);

extern __checkReturn            int
efx_vpd_write(
        __in                    efx_nic_t *enp,
        __in_bcount(size)       caddr_t data,
        __in                    size_t size);

extern                          void
efx_vpd_fini(
        __in                    efx_nic_t *enp);

#endif  /* EFSYS_OPT_VPD */

/* NVRAM */

#if EFSYS_OPT_NVRAM

typedef enum efx_nvram_type_e {
        EFX_NVRAM_INVALID = 0,
        EFX_NVRAM_BOOTROM,
        EFX_NVRAM_BOOTROM_CFG,
        EFX_NVRAM_MC_FIRMWARE,
        EFX_NVRAM_MC_GOLDEN,
        EFX_NVRAM_PHY,
        EFX_NVRAM_NULLPHY,
        EFX_NVRAM_FPGA,
        EFX_NVRAM_NTYPES,
} efx_nvram_type_t;

extern  __checkReturn           int
efx_nvram_init(
        __in                    efx_nic_t *enp);

#if EFSYS_OPT_DIAG

extern  __checkReturn           int
efx_nvram_test(
        __in                    efx_nic_t *enp);

#endif  /* EFSYS_OPT_DIAG */

extern  __checkReturn           int
efx_nvram_size(
        __in                    efx_nic_t *enp,
        __in                    efx_nvram_type_t type,
        __out                   size_t *sizep);

extern  __checkReturn           int
efx_nvram_rw_start(
        __in                    efx_nic_t *enp,
        __in                    efx_nvram_type_t type,
        __out_opt               size_t *pref_chunkp);

extern                          void
efx_nvram_rw_finish(
        __in                    efx_nic_t *enp,
        __in                    efx_nvram_type_t type);

extern  __checkReturn           int
efx_nvram_get_version(
        __in                    efx_nic_t *enp,
        __in                    efx_nvram_type_t type,
        __out                   uint32_t *subtypep,
```

```
1052         __out_ecount(4)         uint16_t version[4]);

1054 extern  __checkReturn           int
1055 efx_nvram_read_chunk(
1056         __in                    efx_nic_t *enp,
1057         __in                    efx_nvram_type_t type,
1058         __in                    unsigned int offset,
1059         __out_bcount(size)      caddr_t data,
1060         __in                    size_t size);

1062 extern  __checkReturn           int
1063 efx_nvram_set_version(
1064         __in                    efx_nic_t *enp,
1065         __in                    efx_nvram_type_t type,
1066         __out                   uint16_t version[4]);

1068 extern   __checkReturn          int
1069 efx_nvram_erase(
1070         __in                    efx_nic_t *enp,
1071         __in                    efx_nvram_type_t type);

1073 extern  __checkReturn           int
1074 efx_nvram_write_chunk(
1075         __in                    efx_nic_t *enp,
1076         __in                    efx_nvram_type_t type,
1077         __in                    unsigned int offset,
1078         __in_bcount(size)       caddr_t data,
1079         __in                    size_t size);

1081 extern                          void
1082 efx_nvram_fini(
1083         __in                    efx_nic_t *enp);

1085 #endif  /* EFSYS_OPT_NVRAM */

1087 #if EFSYS_OPT_BOOTCFG

1089 extern                          int
1090 efx_bootcfg_read(
1091         __in                    efx_nic_t *enp,
1092         __out_bcount(size)      caddr_t data,
1093         __in                    size_t size);

1095 extern                          int
1096 efx_bootcfg_write(
1097         __in                    efx_nic_t *enp,
1098         __in_bcount(size)       caddr_t data,
1099         __in                    size_t size);

1101 #endif  /* EFSYS_OPT_BOOTCFG */

1103 #if EFSYS_OPT_WOL

1105 typedef enum efx_wol_type_e {
1106         EFX_WOL_TYPE_INVALID,
1107         EFX_WOL_TYPE_MAGIC,
1108         EFX_WOL_TYPE_BITMAP,
1109         EFX_WOL_TYPE_LINK,
1110         EFX_WOL_NTYPES,
1111 } efx_wol_type_t;

1113 typedef enum efx_lightsout_offload_type_e {
1114         EFX_LIGHTSOUT_OFFLOAD_TYPE_INVALID,
1115         EFX_LIGHTSOUT_OFFLOAD_TYPE_ARP,
1116         EFX_LIGHTSOUT_OFFLOAD_TYPE_NS,
1117 } efx_lightsout_offload_type_t;
```



```
1119 #define EFX_WOL_BITMAP_MASK_SIZE    (48)
1120 #define EFX_WOL_BITMAP_VALUE_SIZE   (128)

1122 typedef union efx_wol_param_u {
1123         struct {
1124                 uint8_t mac_addr[6];
1125         } ewp_magic;
1126         struct {
1127                 uint8_t mask[EFX_WOL_BITMAP_MASK_SIZE];   /* 1 bit per byte */
1128                 uint8_t value[EFX_WOL_BITMAP_VALUE_SIZE]; /* value to match */
1129                 uint8_t value_len;
1130         } ewp_bitmap;
1131 } efx_wol_param_t;

1133 typedef union efx_lightsout_offload_param_u {
1134         struct {
1135                 uint8_t mac_addr[6];
1136                 uint32_t ip;
1137         } elop_arp;
1138         struct {
1139                 uint8_t mac_addr[6];
1140                 uint32_t solicited_node[4];
1141                 uint32_t ip[4];
1142         } elop_ns;
1143 } efx_lightsout_offload_param_t;

1145 extern  __checkReturn   int
1146 efx_wol_init(
1147         __in            efx_nic_t *enp);

1149 extern  __checkReturn   int
1150 efx_wol_filter_clear(
1151         __in            efx_nic_t *enp);

1153 extern  __checkReturn   int
1154 efx_wol_filter_add(
1155         __in            efx_nic_t *enp,
1156         __in            efx_wol_type_t type,
1157         __in            efx_wol_param_t *paramp,
1158         __out           uint32_t *filter_idp);

1160 extern  __checkReturn   int
1161 efx_wol_filter_remove(
1162         __in            efx_nic_t *enp,
1163         __in            uint32_t filter_id);

1165 extern  __checkReturn   int
1166 efx_lightsout_offload_add(
1167         __in            efx_nic_t *enp,
1168         __in            efx_lightsout_offload_type_t type,
1169         __in            efx_lightsout_offload_param_t *paramp,
1170         __out           uint32_t *filter_idp);

1172 extern  __checkReturn   int
1173 efx_lightsout_offload_remove(
1174         __in            efx_nic_t *enp,
1175         __in            efx_lightsout_offload_type_t type,
1176         __in            uint32_t filter_id);

1178 extern                  void
1179 efx_wol_fini(
1180         __in            efx_nic_t *enp);

1182 #endif  /* EFSYS_OPT_WOL */
```

```
#if EFSYS_OPT_DIAG

typedef enum efx_pattern_type_t {
        EFX_PATTERN_BYTE_INCREMENT = 0,
        EFX_PATTERN_ALL_THE_SAME,
        EFX_PATTERN_BIT_ALTERNATE,
        EFX_PATTERN_BYTE_ALTERNATE,
        EFX_PATTERN_BYTE_CHANGING,
        EFX_PATTERN_BIT_SWEEP,
        EFX_PATTERN_NTYPES
} efx_pattern_type_t;

typedef                 void
(*efx_sram_pattern_fn_t)(
        __in            size_t row,
        __in            boolean_t negate,
        __out           efx_qword_t *eqp);

extern  __checkReturn   int
efx_sram_test(
        __in            efx_nic_t *enp,
        __in            efx_pattern_type_t type);

#endif  /* EFSYS_OPT_DIAG */

extern  __checkReturn   int
efx_sram_buf_tbl_set(
        __in            efx_nic_t *enp,
        __in            uint32_t id,
        __in            efsys_mem_t *esmp,
        __in            size_t n);

extern          void
efx_sram_buf_tbl_clear(
        __in    efx_nic_t *enp,
        __in    uint32_t id,
        __in    size_t n);

#define EFX_BUF_TBL_SIZE        0x20000

#define EFX_BUF_SIZE            4096

/* EV */

typedef struct efx_evq_s        efx_evq_t;

#if EFSYS_OPT_QSTATS

/* START MKCONFIG GENERATED EfxHeaderEventQueueBlock d5614a5d669c8ca3 */
typedef enum efx_ev_qstat_e {
        EV_ALL,
        EV_RX,
        EV_RX_OK,
        EV_RX_RECOVERY,
        EV_RX_FRM_TRUNC,
        EV_RX_TOBE_DISC,
        EV_RX_PAUSE_FRM_ERR,
        EV_RX_BUF_OWNER_ID_ERR,
        EV_RX_IPV4_HDR_CHKSUM_ERR,
        EV_RX_TCP_UDP_CHKSUM_ERR,
        EV_RX_ETH_CRC_ERR,
        EV_RX_IP_FRAG_ERR,
        EV_RX_MCAST_PKT,
        EV_RX_MCAST_HASH_MATCH,
        EV_RX_TCP_IPV4,
        EV_RX_TCP_IPV6,
```



```
        EV_RX_UDP_IPV4,
        EV_RX_UDP_IPV6,
        EV_RX_OTHER_IPV4,
        EV_RX_OTHER_IPV6,
        EV_RX_NON_IP,
        EV_RX_OVERRUN,
        EV_TX,
        EV_TX_WQ_FF_FULL,
        EV_TX_PKT_ERR,
        EV_TX_PKT_TOO_BIG,
        EV_TX_UNEXPECTED,
        EV_GLOBAL,
        EV_GLOBAL_PHY,
        EV_GLOBAL_MNT,
        EV_GLOBAL_RX_RECOVERY,
        EV_DRIVER,
        EV_DRIVER_SRM_UPD_DONE,
        EV_DRIVER_TX_DESCQ_FLS_DONE,
        EV_DRIVER_RX_DESCQ_FLS_DONE,
        EV_DRIVER_RX_DESCQ_FLS_FAILED,
        EV_DRIVER_RX_DSC_ERROR,
        EV_DRIVER_TX_DSC_ERROR,
        EV_DRV_GEN,
        EV_MCDI_RESPONSE,
        EV_NQSTATS
} efx_ev_qstat_t;

/* END MKCONFIG GENERATED EfxHeaderEventQueueBlock */

#endif  /* EFSYS_OPT_QSTATS */

extern  __checkReturn   int
efx_ev_init(
        __in            efx_nic_t *enp);

extern          void
efx_ev_fini(
        __in            efx_nic_t *enp);

#define EFX_MASK(_max, _min)    (-((_max) << 1) ^ -(_min))

#define EFX_EVQ_MAXNEVS         32768
#define EFX_EVQ_MINNEVS         512

#define EFX_EVQ_NEVS_MASK       EFX_MASK(EFX_EVQ_MAXNEVS, EFX_EVQ_MINNEVS)

#define EFX_EVQ_SIZE(_nevs)     ((_nevs) * sizeof (efx_qword_t))
#define EFX_EVQ_NBUFS(_nevs)    (EFX_EVQ_SIZE(_nevs) / EFX_BUF_SIZE)

extern  __checkReturn   int
efx_ev_qcreate(
        __in            efx_nic_t *enp,
        __in            unsigned int index,
        __in            efsys_mem_t *esmp,
        __in            size_t n,
        __in            uint32_t id,
        __deref_out     efx_evq_t **eepp);

extern          void
efx_ev_qpost(
        __in            efx_evq_t *eep,
        __in            uint16_t data);

typedef __checkReturn   boolean_t
(*efx_initialized_ev_t)(
        __in_opt        void *arg);
```

```
#define EFX_PKT_UNICAST         0x0004
#define EFX_PKT_START           0x0008

#define EFX_PKT_VLAN_TAGGED     0x0010
#define EFX_CKSUM_TCPUDP        0x0020
#define EFX_CKSUM_IPV4          0x0040
#define EFX_PKT_CONT            0x0080

#define EFX_CHECK_VLAN          0x0100
#define EFX_PKT_TCP             0x0200
#define EFX_PKT_UDP             0x0400
#define EFX_PKT_IPV4            0x0800

#define EFX_PKT_IPV6            0x1000
#define EFX_ADDR_MISMATCH       0x4000
#define EFX_DISCARD             0x8000

#define EFX_EV_RX_NLABELS       32
#define EFX_EV_TX_NLABELS       32

typedef __checkReturn   boolean_t
(*efx_rx_ev_t)(
        __in_opt        void *arg,
        __in            uint32_t label,
        __in            uint32_t id,
        __in            uint32_t size,
        __in            uint16_t flags);

typedef __checkReturn   boolean_t
(*efx_tx_ev_t)(
        __in_opt        void *arg,
        __in            uint32_t label,
        __in            uint32_t id);

#define EFX_EXCEPTION_RX_RECOVERY       0x00000001
#define EFX_EXCEPTION_RX_DSC_ERROR      0x00000002
#define EFX_EXCEPTION_TX_DSC_ERROR      0x00000003
#define EFX_EXCEPTION_UNKNOWN_SENSOREVT 0x00000004
#define EFX_EXCEPTION_FWALERT_SRAM      0x00000005
#define EFX_EXCEPTION_UNKNOWN_FWALERT   0x00000006

typedef __checkReturn   boolean_t
(*efx_exception_ev_t)(
        __in_opt        void *arg,
        __in            uint32_t label,
        __in            uint32_t data);

typedef __checkReturn   boolean_t
(*efx_rxq_flush_done_ev_t)(
        __in_opt        void *arg,
        __in            uint32_t label);

typedef __checkReturn   boolean_t
(*efx_rxq_flush_failed_ev_t)(
        __in_opt        void *arg,
        __in            uint32_t label);

typedef __checkReturn   boolean_t
(*efx_txq_flush_done_ev_t)(
        __in_opt        void *arg,
        __in            uint32_t label);

typedef __checkReturn   boolean_t
(*efx_software_ev_t)(
        __in_opt        void *arg,
```



```
        __in            uint16_t magic);

typedef __checkReturn   boolean_t
(*efx_sram_ev_t)(
        __in_opt        void *arg,
        __in            uint32_t code);

#define EFX_SRAM_CLEAR          0
#define EFX_SRAM_UPDATE         1
#define EFX_SRAM_ILLEGAL_CLEAR  2

typedef __checkReturn   boolean_t
(*efx_wake_up_ev_t)(
        __in_opt        void *arg,
        __in            uint32_t label);

typedef __checkReturn   boolean_t
(*efx_timer_ev_t)(
        __in_opt        void *arg,
        __in            uint32_t label);

typedef __checkReturn   boolean_t
(*efx_link_change_ev_t)(
        __in_opt        void *arg,
        __in            efx_link_mode_t link_mode);

#if EFSYS_OPT_MON_STATS

typedef __checkReturn   boolean_t
(*efx_monitor_ev_t)(
        __in_opt        void *arg,
        __in            efx_mon_stat_t id,
        __in            efx_mon_stat_value_t value);

#endif  /* EFSYS_OPT_MON_STATS */

#if EFSYS_OPT_MAC_STATS

typedef __checkReturn   boolean_t
(*efx_mac_stats_ev_t)(
        __in_opt        void *arg,
        __in            uint32_t generation
        );

#endif  /* EFSYS_OPT_MAC_STATS */

typedef struct efx_ev_callbacks_s {
        efx_initialized_ev_t            eec_initialized;
        efx_rx_ev_t                     eec_rx;
        efx_tx_ev_t                     eec_tx;
        efx_exception_ev_t              eec_exception;
        efx_rxq_flush_done_ev_t         eec_rxq_flush_done;
        efx_rxq_flush_failed_ev_t       eec_rxq_flush_failed;
        efx_txq_flush_done_ev_t         eec_txq_flush_done;
        efx_software_ev_t               eec_software;
        efx_sram_ev_t                   eec_sram;
        efx_wake_up_ev_t                eec_wake_up;
        efx_timer_ev_t                  eec_timer;
        efx_link_change_ev_t            eec_link_change;
#if EFSYS_OPT_MON_STATS
        efx_monitor_ev_t                eec_monitor;
#endif  /* EFSYS_OPT_MON_STATS */
#if EFSYS_OPT_MAC_STATS
        efx_mac_stats_ev_t              eec_mac_stats;
#endif  /* EFSYS_OPT_MON_STATS */
} efx_ev_callbacks_t;
```

```
1449 extern  __checkReturn   boolean_t
1450 efx_ev_qpending(
1451         __in            efx_evq_t *eep,
1452         __in            unsigned int count);

1454 #if EFSYS_OPT_EV_PREFETCH

1456 extern                  void
1457 efx_ev_qprefetch(
1458         __in            efx_evq_t *eep,
1459         __in            unsigned int count);

1461 #endif  /* EFSYS_OPT_EV_PREFETCH */

1463 extern                  void
1464 efx_ev_qpoll(
1465         __in            efx_evq_t *eep,
1466         __inout         unsigned int *countp,
1467         __in            const efx_ev_callbacks_t *eecp,
1468         __in_opt        void *arg);

1470 extern  __checkReturn   int
1471 efx_ev_qmoderate(
1472         __in            efx_evq_t *eep,
1473         __in            unsigned int us);

1475 extern  __checkReturn   int
1476 efx_ev_qprime(
1477         __in            efx_evq_t *eep,
1478         __in            unsigned int count);

1480 #if EFSYS_OPT_QSTATS

1482 #if EFSYS_OPT_NAMES

1484 extern          const char __cs *
1485 efx_ev_qstat_name(
1486         __in    efx_nic_t *enp,
1487         __in    unsigned int id);

1489 #endif  /* EFSYS_OPT_NAMES */

1491 extern                                  void
1492 efx_ev_qstats_update(
1493         __in                            efx_evq_t *eep,
1494         __inout_ecount(EV_NQSTATS)      efsys_stat_t *stat);

1496 #endif  /* EFSYS_OPT_QSTATS */

1498 extern          void
1499 efx_ev_qdestroy(
1500         __in    efx_evq_t *eep);

1502 /* RX */

1504 typedef struct efx_rxq_s        efx_rxq_t;

1506 extern  __checkReturn   int
1507 efx_rx_init(
1508         __in            efx_nic_t *enp);

1510 extern          void
1511 efx_rx_fini(
1512         __in            efx_nic_t *enp);
```



```
1514 #if EFSYS_OPT_RX_HDR_SPLIT
1515         __checkReturn   int
1516 efx_rx_hdr_split_enable(
1517         __in            efx_nic_t *enp,
1518         __in            unsigned int hdr_buf_size,
1519         __in            unsigned int pld_buf_size);

1521 #endif  /* EFSYS_OPT_RX_HDR_SPLIT */

1523 #if EFSYS_OPT_RX_SCATTER
1524         __checkReturn   int
1525 efx_rx_scatter_enable(
1526         __in            efx_nic_t *enp,
1527         __in            unsigned int buf_size);
1528 #endif  /* EFSYS_OPT_RX_SCATTER */

1530 #if EFSYS_OPT_RX_SCALE

1532 typedef enum efx_rx_hash_alg_e {
1533         EFX_RX_HASHALG_LFSR = 0,
1534         EFX_RX_HASHALG_TOEPLITZ
1535 } efx_rx_hash_alg_t;

1537 typedef enum efx_rx_hash_type_e {
1538         EFX_RX_HASH_IPV4 = 0,
1539         EFX_RX_HASH_TCPIPV4,
1540         EFX_RX_HASH_IPV6,
1541         EFX_RX_HASH_TCPIPV6,
1542 } efx_rx_hash_type_t;

1544 #define EFX_RSS_TBL_SIZE        128     /* Rows in RX indirection table */
1545 #define EFX_MAXRSS              64      /* RX indirection entry range */
1546 #define EFX_MAXRSS_LEGACY       16      /* See bug16611 and bug17213 */

1548 extern  __checkReturn   int
1549 efx_rx_scale_mode_set(
1550         __in    efx_nic_t *enp,
1551         __in    efx_rx_hash_alg_t alg,
1552         __in    efx_rx_hash_type_t type,
1553         __in    boolean_t insert);

1555 extern  __checkReturn   int
1556 efx_rx_scale_tbl_set(
1557         __in            efx_nic_t *enp,
1558         __in_ecount(n)  unsigned int *table,
1559         __in            size_t n);

1561 extern  __checkReturn   int
1562 efx_rx_scale_toeplitz_ipv4_key_set(
1563         __in            efx_nic_t *enp,
1564         __in_ecount(n)  uint8_t *key,
1565         __in            size_t n);

1567 extern  __checkReturn   int
1568 efx_rx_scale_toeplitz_ipv6_key_set(
1569         __in            efx_nic_t *enp,
1570         __in_ecount(n)  uint8_t *key,
1571         __in            size_t n);

1573 /*
1574  * The prefix is a byte array of one of the forms:
1575  *
1576  *  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
1577  * XX.XX.XX.XX.XX.XX.XX.XX.XX.XX.XX.XX.TT.TT.TT.TT
1578  * XX.XX.XX.XX.XX.XX.XX.XX.XX.XX.XX.XX.XX.XX.LL.LL
1579  *
```

```
 * where:
 *
 * TT.TT.TT.TT is a 32-bit Toeplitz hash
 * LL.LL is a 16-bit LFSR hash
 *
 * Hash values are in network (big-endian) byte order.
 */

#define EFX_RX_PREFIX_SIZE      16

#define EFX_RX_HASH_VALUE(_func, _buffer)                               \
        (((_func) == EFX_RX_HASHALG_LFSR) ?                             \
                ((uint16_t)((_buffer)[14] << 8) | (_buffer)[15])) :     \
                ((uint32_t)((_buffer)[12] << 24) |                      \
                    ((_buffer)[13] << 16) |                             \
                    ((_buffer)[14] << 8) |                              \
                    (_buffer)[15])))

#define EFX_RX_HASH_SIZE(_func)                                         \
        (((_func) == EFX_RX_HASHALG_LFSR) ?                             \
                sizeof (uint16_t) :                                     \
                sizeof (uint32_t))

#endif  /* EFSYS_OPT_RX_SCALE */

#define EFX_RXQ_MAXNDESCS               4096
#define EFX_RXQ_MINNDESCS               512

#define EFX_RXQ_NDESCS_MASK             EFX_MASK(EFX_RXQ_MAXNDESCS, EFX_RXQ_MINN

#define EFX_RXQ_SIZE(_ndescs)           ((_ndescs) * sizeof (efx_qword_t))
#define EFX_RXQ_NBUFS(_ndescs)          (EFX_RXQ_SIZE(_ndescs) / EFX_BUF_SIZE)
#define EFX_RXQ_LIMIT(_ndescs)          ((_ndescs) - 16)
#define EFX_RXQ_DC_NDESCS(_dcsize)      (8 << _dcsize)

typedef enum efx_rxq_type_e {
        EFX_RXQ_TYPE_DEFAULT,
        EFX_RXQ_TYPE_SPLIT_HEADER,
        EFX_RXQ_TYPE_SPLIT_PAYLOAD,
        EFX_RXQ_TYPE_SCATTER,
        EFX_RXQ_NTYPES
} efx_rxq_type_t;

extern  __checkReturn   int
efx_rx_qcreate(
        __in            efx_nic_t *enp,
        __in            unsigned int index,
        __in            unsigned int label,
        __in            efx_rxq_type_t type,
        __in            efsys_mem_t *esmp,
        __in            size_t n,
        __in            uint32_t id,
        __in            efx_evq_t *eep,
        __deref_out     efx_rxq_t **erpp);

typedef struct efx_buffer_s {
        efsys_dma_addr_t        eb_addr;
        size_t                  eb_size;
        boolean_t               eb_eop;
} efx_buffer_t;

extern                  void
efx_rx_qpost(
        __in            efx_rxq_t *erp,
        __in_ecount(n)  efsys_dma_addr_t *addrp,
        __in            size_t size,
```



```
        __in            unsigned int n,
        __in            unsigned int completed,
        __in            unsigned int added);

extern          void
efx_rx_qpush(
        __in    efx_rxq_t *erp,
        __in    unsigned int added);

extern          void
efx_rx_qflush(
        __in    efx_rxq_t *erp);

extern          void
efx_rx_qenable(
        __in    efx_rxq_t *erp);

extern          void
efx_rx_qdestroy(
        __in    efx_rxq_t *erp);

/* TX */

typedef struct efx_txq_s        efx_txq_t;

#if EFSYS_OPT_QSTATS

/* START MKCONFIG GENERATED EfxHeaderTransmitQueueBlock 536c5fa5014944bf */
typedef enum efx_tx_qstat_e {
        TX_POST,
        TX_UNALIGNED_SPLIT,
        TX_NQSTATS
} efx_tx_qstat_t;

/* END MKCONFIG GENERATED EfxHeaderTransmitQueueBlock */

#endif  /* EFSYS_OPT_QSTATS */

extern  __checkReturn   int
efx_tx_init(
        __in            efx_nic_t *enp);

extern          void
efx_tx_fini(
        __in    efx_nic_t *enp);

#define EFX_TXQ_MAXNDESCS               4096
#define EFX_TXQ_MINNDESCS               512

#define EFX_TXQ_NDESCS_MASK             EFX_MASK(EFX_TXQ_MAXNDESCS, EFX_TXQ_MINN

#define EFX_TXQ_SIZE(_ndescs)           ((_ndescs) * sizeof (efx_qword_t))
#define EFX_TXQ_NBUFS(_ndescs)          (EFX_TXQ_SIZE(_ndescs) / EFX_BUF_SIZE)
#define EFX_TXQ_LIMIT(_ndescs)          ((_ndescs) - 16)
#define EFX_TXQ_DC_NDESCS(_dcsize)      (8 << _dcsize)

extern  __checkReturn   int
efx_tx_qcreate(
        __in            efx_nic_t *enp,
        __in            unsigned int index,
        __in            unsigned int label,
        __in            efsys_mem_t *esmp,
        __in            size_t n,
        __in            uint32_t id,
        __in            uint16_t flags,
        __in            efx_evq_t *eep,
```

```
1712         __deref_out     efx_txq_t **etpp);

1714 extern  __checkReturn   int
1715 efx_tx_qpost(
1716         __in            efx_txq_t *etp,
1717         __in_ecount(n)  efx_buffer_t *eb,
1718         __in            unsigned int n,
1719         __in            unsigned int completed,
1720         __inout         unsigned int *addedp);

1722 extern  __checkReturn   int
1723 efx_tx_qpace(
1724         __in            efx_txq_t *etp,
1725         __in            unsigned int ns);

1727 extern          void
1728 efx_tx_qpush(
1729         __in    efx_txq_t *etp,
1730         __in    unsigned int added);

1732 extern          void
1733 efx_tx_qflush(
1734         __in    efx_txq_t *etp);

1736 extern          void
1737 efx_tx_qenable(
1738         __in    efx_txq_t *etp);

1740 #if EFSYS_OPT_QSTATS

1742 #if EFSYS_OPT_NAMES

1744 extern          const char __cs *
1745 efx_tx_qstat_name(
1746         __in    efx_nic_t *etp,
1747         __in    unsigned int id);

1749 #endif  /* EFSYS_OPT_NAMES */

1751 extern                                  void
1752 efx_tx_qstats_update(
1753         __in                            efx_txq_t *etp,
1754         __inout_ecount(TX_NQSTATS)      efsys_stat_t *stat);

1756 #endif  /* EFSYS_OPT_QSTATS */

1758 extern          void
1759 efx_tx_qdestroy(
1760         __in    efx_txq_t *etp);


1763 /* FILTER */

1765 #if EFSYS_OPT_FILTER

1767 typedef enum efx_filter_flag_e {
1768         EFX_FILTER_FLAG_RX_RSS = 0x01,          /* use RSS to spread across
1769                                                  * multiple queues */
1770         EFX_FILTER_FLAG_RX_SCATTER = 0x02,      /* enable RX scatter */
1771         EFX_FILTER_FLAG_RX_OVERRIDE_IP = 0x04,  /* MAC filter overrides
1772                                                  * any matching IP filter */
1773 } efx_filter_flag_t;

1775 typedef struct efx_filter_spec_s {
1776         uint8_t         efs_type;
1777         uint8_t         efs_flags;
```



```
1778         uint16_t        efs_dmaq_id;
1779         uint32_t        efs_dword[3];
1780 } efx_filter_spec_t;

1782 extern  __checkReturn   int
1783 efx_filter_init(
1784         __in            efx_nic_t *enp);

1786 extern                  void
1787 efx_filter_fini(
1788         __in            efx_nic_t *enp);

1790 extern  __checkReturn   int
1791 efx_rx_filter_insert(
1792         __in            efx_rxq_t *erp,
1793         __inout         efx_filter_spec_t *spec);

1795 extern  __checkReturn   int
1796 efx_rx_filter_remove(
1797         __in            efx_rxq_t *erp,
1798         __inout         efx_filter_spec_t *spec);

1800                         void
1801 efx_filter_restore(
1802         __in            efx_nic_t *enp);

1804 extern                  void
1805 efx_filter_spec_rx_ipv4_tcp_full(
1806         __inout         efx_filter_spec_t *spec,
1807         __in            unsigned int flags,
1808         __in            uint32_t src_ip,
1809         __in            uint16_t src_tcp,
1810         __in            uint32_t dest_ip,
1811         __in            uint16_t dest_tcp);

1813 extern                  void
1814 efx_filter_spec_rx_ipv4_tcp_wild(
1815         __inout         efx_filter_spec_t *spec,
1816         __in            unsigned int flags,
1817         __in            uint32_t dest_ip,
1818         __in            uint16_t dest_tcp);

1820 extern                  void
1821 efx_filter_spec_rx_ipv4_udp_full(
1822         __inout         efx_filter_spec_t *spec,
1823         __in            unsigned int flags,
1824         __in            uint32_t src_ip,
1825         __in            uint16_t src_udp,
1826         __in            uint32_t dest_ip,
1827         __in            uint16_t dest_udp);

1829 extern                  void
1830 efx_filter_spec_rx_ipv4_udp_wild(
1831         __inout         efx_filter_spec_t *spec,
1832         __in            unsigned int flags,
1833         __in            uint32_t dest_ip,
1834         __in            uint16_t dest_udp);

1836 extern                  void
1837 efx_filter_spec_rx_mac_full(
1838         __inout         efx_filter_spec_t *spec,
1839         __in            unsigned int flags,
1840         __in            uint16_t vlan_id,
1841         __in            uint8_t *dest_mac);

1843 extern                  void
```

```
1844 efx_filter_spec_rx_mac_wild(
1845         __inout         efx_filter_spec_t *spec,
1846         __in            unsigned int flags,
1847         __in            uint8_t *dest_mac);


1850 extern  __checkReturn   int
1851 efx_tx_filter_insert(
1852         __in            efx_txq_t *etp,
1853         __inout         efx_filter_spec_t *spec);

1855 extern  __checkReturn   int
1856 efx_tx_filter_remove(
1857         __in            efx_txq_t *etp,
1858         __inout         efx_filter_spec_t *spec);

1860 extern                  void
1861 efx_filter_spec_tx_ipv4_tcp_full(
1862         __inout         efx_filter_spec_t *spec,
1863         __in            uint32_t src_ip,
1864         __in            uint16_t src_tcp,
1865         __in            uint32_t dest_ip,
1866         __in            uint16_t dest_tcp);

1868 extern                  void
1869 efx_filter_spec_tx_ipv4_tcp_wild(
1870         __inout         efx_filter_spec_t *spec,
1871         __in            uint32_t src_ip,
1872         __in            uint16_t src_tcp);

1874 extern                  void
1875 efx_filter_spec_tx_ipv4_udp_full(
1876         __inout         efx_filter_spec_t *spec,
1877         __in            uint32_t src_ip,
1878         __in            uint16_t src_udp,
1879         __in            uint32_t dest_ip,
1880         __in            uint16_t dest_udp);

1882 extern                  void
1883 efx_filter_spec_tx_ipv4_udp_wild(
1884         __inout         efx_filter_spec_t *spec,
1885         __in            uint32_t src_ip,
1886         __in            uint16_t src_udp);

1888 extern                  void
1889 efx_filter_spec_tx_mac_full(
1890         __inout         efx_filter_spec_t *spec,
1891         __in            uint16_t vlan_id,
1892         __in            uint8_t *src_mac);

1894 extern                  void
1895 efx_filter_spec_tx_mac_wild(
1896         __inout         efx_filter_spec_t *spec,
1897         __in            uint8_t *src_mac);

1899 #endif  /* EFSYS_OPT_FILTER */


1902 #ifdef  __cplusplus
1903 }
1904 #endif

1906 #endif  /* _SYS_EFX_H */
1907 #endif /* ! codereview */
```

```
**********************************************************
    8263 Thu Aug 22 18:59:21 2013
new/usr/src/uts/common/io/sfxge/efx_bootcfg.c
Merged sfxge driver
**********************************************************
```

```
/*
 * Copyright 2009 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ‘‘AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#include "efsys.h"
#include "efx.h"
#include "efx_types.h"
#include "efx_impl.h"

#if EFSYS_OPT_BOOTCFG

/*
 * Maximum size of BOOTCFG block across all nics as understood by SFCgPXE.
 * A multiple of 0x100 so trailing 0xff characters don’t contrinbute to the
 * checksum.
 */
#define BOOTCFG_MAX_SIZE 0x1000

#define DHCP_END (uint8_t)0xff
#define DHCP_PAD (uint8_t)0

static  __checkReturn           uint8_t
efx_bootcfg_csum(
        __in                    efx_nic_t *enp,
        __in_bcount(size)       caddr_t data,
        __in                    size_t size)
{
        _NOTE(ARGUNUSED(enp))

        unsigned int pos;
        uint8_t checksum = 0;

        for (pos = 0; pos < size; pos++)
                checksum += data[pos];
        return (checksum);
}

static  __checkReturn           int
efx_bootcfg_verify(
        __in                    efx_nic_t *enp,
```



```
        __in_bcount(size)       caddr_t data,
        __in                    size_t size,
        __out                   size_t *usedp)
{
        size_t offset = 0;
        size_t used = 0;
        int rc;

        /* Start parsing tags immediatly after the checksum */
        for (offset = 1; offset < size; ) {
                uint8_t tag;
                uint8_t length;

                /* Consume tag */
                tag = data[offset];
                if (tag == DHCP_END) {
                        offset++;
                        used = offset;
                        break;
                }
                if (tag == DHCP_PAD) {
                        offset++;
                        continue;
                }

                /* Consume length */
                if (offset + 1 >= size) {
                        rc = ENOSPC;
                        goto fail1;
                }
                length = data[offset + 1];

                /* Consume *length */
                if (offset + 1 + length >= size) {
                        rc = ENOSPC;
                        goto fail2;
                }

                offset += 2 + length;
                used = offset;
        }

        /* Checksum the entire sector, including bytes after any DHCP_END */
        if (efx_bootcfg_csum(enp, data, size) != 0) {
                rc = EINVAL;
                goto fail3;
        }

        if (usedp != NULL)
                *usedp = used;

        return (0);

fail3:
        EFSYS_PROBE(fail3);
fail2:
        EFSYS_PROBE(fail2);
fail1:
        EFSYS_PROBE1(fail1, int, rc);

        return (rc);
}

                                int
efx_bootcfg_read(
        __in                    efx_nic_t *enp,
```

```
128         __out_bcount(size)      caddr_t data,
129         __in                    size_t size)
130 {
131         uint8_t *payload = NULL;
132         size_t used_bytes;
133         size_t sector_length;
134         int rc;

136         rc = efx_nvram_size(enp, EFX_NVRAM_BOOTROM_CFG, &sector_length);
137         if (rc != 0)
138                 goto fail1;

140         /*
141          * We need to read the entire BOOTCFG area to ensure we read all the
142          * tags, because legacy bootcfg sectors are not guaranteed to end with
143          * a DHCP_END character. If the user hasn't supplied a sufficiently
144          * large buffer then use our own buffer.
145          */
146         if (sector_length > BOOTCFG_MAX_SIZE)
147                 sector_length = BOOTCFG_MAX_SIZE;
148         if (sector_length > size) {
149                 EFSYS_KMEM_ALLOC(enp->en_esip, sector_length, payload);
150                 if (payload == NULL) {
151                         rc = ENOMEM;
152                         goto fail2;
153                 }
154         } else
155                 payload = (uint8_t *)data;

157         if ((rc = efx_nvram_rw_start(enp, EFX_NVRAM_BOOTROM_CFG, NULL)) != 0)
158                 goto fail3;

160         rc = efx_nvram_read_chunk(enp, EFX_NVRAM_BOOTROM_CFG, 0,
161                                   (caddr_t)payload, sector_length);

163         efx_nvram_rw_finish(enp, EFX_NVRAM_BOOTROM_CFG);

165         if (rc != 0)
166                 goto fail4;

168         /* Verify that the area is correctly formatted and checksummed */
169         rc = efx_bootcfg_verify(enp, (caddr_t)payload, sector_length,
170                                 &used_bytes);
171         if (rc != 0 || used_bytes == 0) {
172                 payload[0] = (uint8_t)~DHCP_END;
173                 payload[1] = DHCP_END;
174                 used_bytes = 2;
175         }

177         EFSYS_ASSERT(used_bytes >= 2);  /* checksum and DHCP_END */
178         EFSYS_ASSERT(used_bytes <= sector_length);

180         /*
181          * Legacy bootcfg sectors don't terminate with a DHCP_END character.
182          * Modify the returned payload so it does. BOOTCFG_MAX_SIZE is by
183          * definition large enough for any valid (per-port) bootcfg sector,
184          * so reinitialise the sector if there isn't room for the character.
185          */
186         if (payload[used_bytes - 1] != DHCP_END) {
187                 if (used_bytes + 1 > sector_length) {
188                         payload[0] = 0;
189                         used_bytes = 1;
190                 }

192                 payload[used_bytes] = DHCP_END;
193                 ++used_bytes;
```



```
194         }

196         /*
197          * Verify that the user supplied buffer is large enough for the
198          * entire used bootcfg area, then copy into the user supplied buffer.
199          */
200         if (used_bytes > size) {
201                 rc = ENOSPC;
202                 goto fail5;
203         }
204         if (sector_length > size) {
205                 memcpy(data, payload, used_bytes);
206                 EFSYS_KMEM_FREE(enp->en_esip, sector_length, payload);
207         }

209         /* Zero out the unused portion of the user buffer */
210         if (used_bytes < size)
211                 (void) memset(data + used_bytes, 0, size - used_bytes);

213         /*
214          * The checksum includes trailing data after any DHCP_END character,
215          * which we've just modified (by truncation or appending DHCP_END).
216          */
217         data[0] -= efx_bootcfg_csum(enp, data, size);

219         return (0);

221 fail5:
222         EFSYS_PROBE(fail5);
223 fail4:
224         EFSYS_PROBE(fail4);
225 fail3:
226         EFSYS_PROBE(fail3);

228         if (sector_length > size)
229                 EFSYS_KMEM_FREE(enp->en_esip, sector_length, payload);
230 fail2:
231         EFSYS_PROBE(fail2);
232 fail1:
233         EFSYS_PROBE1(fail1, int, rc);

235         return (rc);
236 }

238                                 int
239 efx_bootcfg_write(
240         __in                    efx_nic_t *enp,
241         __in_bcount(size)       caddr_t data,
242         __in                    size_t size)
243 {
244         uint8_t *chunk;
245         uint8_t checksum;
246         size_t sector_length;
247         size_t chunk_length;
248         size_t used_bytes;
249         size_t offset;
250         size_t remaining;
251         int rc;

253         rc = efx_nvram_size(enp, EFX_NVRAM_BOOTROM_CFG, &sector_length);
254         if (rc != 0)
255                 goto fail1;

257         if (sector_length > BOOTCFG_MAX_SIZE)
258                 sector_length = BOOTCFG_MAX_SIZE;
```

```
260         if ((rc = efx_bootcfg_verify(enp, data, size, &used_bytes)) != 0)
261                 goto fail2;

263         /* The caller *must* terminate their block with a DHCP_END character */
264         EFSYS_ASSERT(used_bytes >= 2);          /* checksum and DHCP_END */
265         if ((uint8_t)data[used_bytes - 1] != DHCP_END) {
266                 rc = ENOENT;
267                 goto fail3;
268         }

270         /* Check that the hardware has support for this much data */
271         if (used_bytes > MIN(sector_length, BOOTCFG_MAX_SIZE)) {
272                 rc = ENOSPC;
273                 goto fail4;
274         }

276         rc = efx_nvram_rw_start(enp, EFX_NVRAM_BOOTROM_CFG, &chunk_length);
277         if (rc != 0)
278                 goto fail5;

280         EFSYS_KMEM_ALLOC(enp->en_esip, chunk_length, chunk);
281         if (chunk == NULL) {
282                 rc = ENOMEM;
283                 goto fail6;
284         }

286         if ((rc = efx_nvram_erase(enp, EFX_NVRAM_BOOTROM_CFG)) != 0)
287                 goto fail7;

289         /*
290          * Write the entire sector_length bytes of data in chunks. Zero out
291          * all data following the DHCP_END, and adjust the checksum
292          */
293         checksum = efx_bootcfg_csum(enp, data, used_bytes);
294         for (offset = 0; offset < sector_length; offset += remaining) {
295                 remaining = MIN(chunk_length, sector_length - offset);

297                 /* Fill chunk */
298                 (void) memset(chunk, 0x0, chunk_length);
299                 if (offset < used_bytes)
300                         memcpy(chunk, data + offset,
301                             MIN(remaining, used_bytes - offset));

303                 /* Adjust checksum */
304                 if (offset == 0)
305                         chunk[0] -= checksum;

307                 if ((rc = efx_nvram_write_chunk(enp, EFX_NVRAM_BOOTROM_CFG,
308                             offset, (caddr_t)chunk, remaining)) != 0)
309                         goto fail8;
310         }

312         efx_nvram_rw_finish(enp, EFX_NVRAM_BOOTROM_CFG);

314         EFSYS_KMEM_FREE(enp->en_esip, chunk_length, chunk);

316         return (0);

318 fail8:
319         EFSYS_PROBE(fail8);
320 fail7:
321         EFSYS_PROBE(fail7);

323         EFSYS_KMEM_FREE(enp->en_esip, chunk_length, chunk);
324 fail6:
325         EFSYS_PROBE(fail6);
```



```
327         efx_nvram_rw_finish(enp, EFX_NVRAM_BOOTROM_CFG);
328 fail5:
329         EFSYS_PROBE(fail5);
330 fail4:
331         EFSYS_PROBE(fail4);
332 fail3:
333         EFSYS_PROBE(fail3);
334 fail2:
335         EFSYS_PROBE(fail2);
336 fail1:
337         EFSYS_PROBE1(fail1, int, rc);

339         return (rc);
340 }

342 #endif  /* EFSYS_OPT_BOOTCFG */
343 #endif /* ! codereview */
```

```
**********************************************************
   27740 Thu Aug 22 18:59:21 2013
new/usr/src/uts/common/io/sfxge/efx_ev.c
Merged sfxge driver
**********************************************************
```

```
/*
 * Copyright 2007-2013 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#include "efsys.h"
#include "efx.h"
#include "efx_types.h"
#include "efx_regs.h"
#include "efx_impl.h"

#if EFSYS_OPT_QSTATS
#define EFX_EV_QSTAT_INCR(_eep, _stat)                                  \
        do {                                                            \
                (_eep)->ee_stat[_stat]++;                               \
        _NOTE(CONSTANTCONDITION)                                        \
        } while (B_FALSE)
#else
#define EFX_EV_QSTAT_INCR(_eep, _stat)
#endif

        __checkReturn   int
efx_ev_init(
        __in            efx_nic_t *enp)
{
        efx_oword_t oword;
        int rc;

        EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
        EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_INTR);

        if (enp->en_mod_flags & EFX_MOD_EV) {
                rc = EINVAL;
                goto fail1;
        }

        EFSYS_ASSERT3U(enp->en_ev_qcount, ==, 0);

        /*
         * Program the event queue for receive and transmit queue
         * flush events.
```



```
         */
        EFX_BAR_READO(enp, FR_AZ_DP_CTRL_REG, &oword);
        EFX_SET_OWORD_FIELD(oword, FRF_AZ_FLS_EVQ_ID, 0);
        EFX_BAR_WRITEO(enp, FR_AZ_DP_CTRL_REG, &oword);

        enp->en_mod_flags |= EFX_MOD_EV;
        return (0);

fail1:
        EFSYS_PROBE1(fail1, int, rc);

        return (rc);
}

static  __checkReturn   boolean_t
efx_ev_rx_not_ok(
        __in            efx_evq_t *eep,
        __in            efx_qword_t *eqp,
        __in            uint32_t label,
        __in            uint32_t id,
        __inout         uint16_t *flagsp)
{
        boolean_t ignore = B_FALSE;

        if (EFX_QWORD_FIELD(*eqp, FSF_AZ_RX_EV_TOBE_DISC) != 0) {
                EFX_EV_QSTAT_INCR(eep, EV_RX_TOBE_DISC);
                EFSYS_PROBE(tobe_disc);
                /*
                 * Assume this is a unicast address mismatch, unless below
                 * we find either FSF_AZ_RX_EV_ETH_CRC_ERR or
                 * EV_RX_PAUSE_FRM_ERR is set.
                 */
                (*flagsp) |= EFX_ADDR_MISMATCH;
        }

        if (EFX_QWORD_FIELD(*eqp, FSF_AZ_RX_EV_FRM_TRUNC) != 0) {
                EFSYS_PROBE2(frm_trunc, uint32_t, label, uint32_t, id);
                EFX_EV_QSTAT_INCR(eep, EV_RX_FRM_TRUNC);
                (*flagsp) |= EFX_DISCARD;

#if (EFSYS_OPT_RX_HDR_SPLIT || EFSYS_OPT_RX_SCATTER)
                /*
                 * Lookout for payload queue ran dry errors and ignore them.
                 *
                 * Sadly for the header/data split cases, the descriptor
                 * pointer in this event refers to the header queue and
                 * therefore cannot be easily detected as duplicate.
                 * So we drop these and rely on the receive processing seeing
                 * a subsequent packet with FSF_AZ_RX_EV_SOP set to discard
                 * the partially received packet.
                 */
                if ((EFX_QWORD_FIELD(*eqp, FSF_AZ_RX_EV_SOP) == 0) &&
                    (EFX_QWORD_FIELD(*eqp, FSF_AZ_RX_EV_JUMBO_CONT) == 0) &&
                    (EFX_QWORD_FIELD(*eqp, FSF_AZ_RX_EV_BYTE_CNT) == 0))
                        ignore = B_TRUE;
#endif  /* EFSYS_OPT_RX_HDR_SPLIT || EFSYS_OPT_RX_SCATTER */
        }

        if (EFX_QWORD_FIELD(*eqp, FSF_AZ_RX_EV_ETH_CRC_ERR) != 0) {
                EFX_EV_QSTAT_INCR(eep, EV_RX_ETH_CRC_ERR);
                EFSYS_PROBE(crc_err);
                (*flagsp) &= ~EFX_ADDR_MISMATCH;
                (*flagsp) |= EFX_DISCARD;
        }

        if (EFX_QWORD_FIELD(*eqp, FSF_AZ_RX_EV_PAUSE_FRM_ERR) != 0) {
```

```
128                 EFX_EV_QSTAT_INCR(eep, EV_RX_PAUSE_FRM_ERR);
129                 EFSYS_PROBE(pause_frm_err);
130                 (*flagsp) &= ~EFX_ADDR_MISMATCH;
131                 (*flagsp) |= EFX_DISCARD;
132         }

134         if (EFX_QWORD_FIELD(*eqp, FSF_AZ_RX_EV_BUF_OWNER_ID_ERR) != 0) {
135                 EFX_EV_QSTAT_INCR(eep, EV_RX_BUF_OWNER_ID_ERR);
136                 EFSYS_PROBE(owner_id_err);
137                 (*flagsp) |= EFX_DISCARD;
138         }

140         if (EFX_QWORD_FIELD(*eqp, FSF_AZ_RX_EV_IP_HDR_CHKSUM_ERR) != 0) {
141                 EFX_EV_QSTAT_INCR(eep, EV_RX_IPV4_HDR_CHKSUM_ERR);
142                 EFSYS_PROBE(ipv4_err);
143                 (*flagsp) &= ~EFX_CKSUM_IPV4;
144         }

146         if (EFX_QWORD_FIELD(*eqp, FSF_AZ_RX_EV_TCP_UDP_CHKSUM_ERR) != 0) {
147                 EFX_EV_QSTAT_INCR(eep, EV_RX_TCP_UDP_CHKSUM_ERR);
148                 EFSYS_PROBE(udp_chk_err);
149                 (*flagsp) &= ~EFX_CKSUM_TCPUDP;
150         }

152         if (EFX_QWORD_FIELD(*eqp, FSF_AZ_RX_EV_IP_FRAG_ERR) != 0) {
153                 EFX_EV_QSTAT_INCR(eep, EV_RX_IP_FRAG_ERR);

155                 /*
156                  * If IP is fragmented FSF_AZ_RX_EV_IP_FRAG_ERR is set. This
157                  * causes FSF_AZ_RX_EV_PKT_OK to be clear. This is not an error
158                  * condition.
159                  */
160                 (*flagsp) &= ~(EFX_PKT_TCP | EFX_PKT_UDP | EFX_CKSUM_TCPUDP);
161         }

163         return (ignore);
164 }

166 static  __checkReturn   boolean_t
167 efx_ev_rx(
168         __in            efx_evq_t *eep,
169         __in            efx_qword_t *eqp,
170         __in            const efx_ev_callbacks_t *eecp,
171         __in_opt        void *arg)
172 {
173         efx_nic_t *enp = eep->ee_enp;
174         uint32_t id;
175         uint32_t size;
176         uint32_t label;
177         boolean_t ok;
178 #if (EFSYS_OPT_RX_HDR_SPLIT || EFSYS_OPT_RX_SCATTER)
179         boolean_t sop;
180         boolean_t jumbo_cont;
181 #endif  /* EFSYS_OPT_RX_HDR_SPLIT || EFSYS_OPT_RX_SCATTER */
182         uint32_t hdr_type;
183         boolean_t is_v6;
184         uint16_t flags;
185         boolean_t ignore;
186         boolean_t should_abort;

188         EFX_EV_QSTAT_INCR(eep, EV_RX);

190         /* Basic packet information */
191         id = EFX_QWORD_FIELD(*eqp, FSF_AZ_RX_EV_DESC_PTR);
192         size = EFX_QWORD_FIELD(*eqp, FSF_AZ_RX_EV_BYTE_CNT);
193         label = EFX_QWORD_FIELD(*eqp, FSF_AZ_RX_EV_Q_LABEL);
```



```
194         ok = (EFX_QWORD_FIELD(*eqp, FSF_AZ_RX_EV_PKT_OK) != 0);

196 #if (EFSYS_OPT_RX_HDR_SPLIT || EFSYS_OPT_RX_SCATTER)
197         sop = (EFX_QWORD_FIELD(*eqp, FSF_AZ_RX_EV_SOP) != 0);
198         jumbo_cont = (EFX_QWORD_FIELD(*eqp, FSF_AZ_RX_EV_JUMBO_CONT) != 0);
199 #endif  /* EFSYS_OPT_RX_HDR_SPLIT || EFSYS_OPT_RX_SCATTER */

201         hdr_type = EFX_QWORD_FIELD(*eqp, FSF_AZ_RX_EV_HDR_TYPE);

203         is_v6 = (enp->en_family != EFX_FAMILY_FALCON &&
204                  EFX_QWORD_FIELD(*eqp, FSF_CZ_RX_EV_IPV6_PKT) != 0);

206         /*
207          * If packet is marked as OK and packet type is TCP/IP or
208          * UDP/IP or other IP, then we can rely on the hardware checksums.
209          */
210         switch (hdr_type) {
211         case FSE_AZ_RX_EV_HDR_TYPE_IPV4V6_TCP:
212                 flags = EFX_PKT_TCP | EFX_CKSUM_TCPUDP;
213                 if (is_v6) {
214                         EFX_EV_QSTAT_INCR(eep, EV_RX_TCP_IPV6);
215                         flags |= EFX_PKT_IPV6;
216                 } else {
217                         EFX_EV_QSTAT_INCR(eep, EV_RX_TCP_IPV4);
218                         flags |= EFX_PKT_IPV4 | EFX_CKSUM_IPV4;
219                 }
220                 break;

222         case FSE_AZ_RX_EV_HDR_TYPE_IPV4V6_UDP:
223                 flags = EFX_PKT_UDP | EFX_CKSUM_TCPUDP;
224                 if (is_v6) {
225                         EFX_EV_QSTAT_INCR(eep, EV_RX_UDP_IPV6);
226                         flags |= EFX_PKT_IPV6;
227                 } else {
228                         EFX_EV_QSTAT_INCR(eep, EV_RX_UDP_IPV4);
229                         flags |= EFX_PKT_IPV4 | EFX_CKSUM_IPV4;
230                 }
231                 break;

233         case FSE_AZ_RX_EV_HDR_TYPE_IPV4V6_OTHER:
234                 if (is_v6) {
235                         EFX_EV_QSTAT_INCR(eep, EV_RX_OTHER_IPV6);
236                         flags = EFX_PKT_IPV6;
237                 } else {
238                         EFX_EV_QSTAT_INCR(eep, EV_RX_OTHER_IPV4);
239                         flags = EFX_PKT_IPV4 | EFX_CKSUM_IPV4;
240                 }
241                 break;

243         case FSE_AZ_RX_EV_HDR_TYPE_OTHER:
244                 EFX_EV_QSTAT_INCR(eep, EV_RX_NON_IP);
245                 flags = 0;
246                 break;

248         default:
249                 EFSYS_ASSERT(B_FALSE);
250                 flags = 0;
251                 break;
252         }

254 #if EFSYS_OPT_RX_SCATTER || EFSYS_OPT_RX_HDR_SPLIT
255         /* Report scatter and header/lookahead split buffer flags */
256         if (sop)
257                 flags |= EFX_PKT_START;
258         if (jumbo_cont)
259                 flags |= EFX_PKT_CONT;
```

```
260 #endif  /* EFSYS_OPT_RX_SCATTER || EFSYS_OPT_RX_HDR_SPLIT */

262         /* Detect errors included in the FSF_AZ_RX_EV_PKT_OK indication */
263         if (!ok) {
264                 ignore = efx_ev_rx_not_ok(eep, eqp, label, id, &flags);
265                 if (ignore) {
266                         EFSYS_PROBE4(rx_complete, uint32_t, label, uint32_t, id,
267                             uint32_t, size, uint16_t, flags);

269                         return (B_FALSE);
270                 }
271         }

273         /* If we're not discarding the packet then it is ok */
274         if (~flags & EFX_DISCARD)
275                 EFX_EV_QSTAT_INCR(eep, EV_RX_OK);

277         /* Detect multicast packets that didn't match the filter */
278         if (EFX_QWORD_FIELD(*eqp, FSF_AZ_RX_EV_MCAST_PKT) != 0) {
279                 EFX_EV_QSTAT_INCR(eep, EV_RX_MCAST_PKT);

281                 if (EFX_QWORD_FIELD(*eqp, FSF_AZ_RX_EV_MCAST_HASH_MATCH) != 0) {
282                         EFX_EV_QSTAT_INCR(eep, EV_RX_MCAST_HASH_MATCH);
283                 } else {
284                         EFSYS_PROBE(mcast_mismatch);
285                         flags |= EFX_ADDR_MISMATCH;
286                 }
287         } else {
288                 flags |= EFX_PKT_UNICAST;
289         }

291         /*
292          * The packet parser in Siena can abort parsing packets under
293          * certain error conditions, setting the PKT_NOT_PARSED bit
294          * (which clears PKT_OK). If this is set, then don't trust
295          * the PKT_TYPE field.
296          */
297         if (enp->en_family != EFX_FAMILY_FALCON && !ok) {
298                 uint32_t parse_err;

300                 parse_err = EFX_QWORD_FIELD(*eqp, FSF_CZ_RX_EV_PKT_NOT_PARSED);
301                 if (parse_err != 0)
302                         flags |= EFX_CHECK_VLAN;
303         }

305         if (~flags & EFX_CHECK_VLAN) {
306                 uint32_t pkt_type;

308                 pkt_type = EFX_QWORD_FIELD(*eqp, FSF_AZ_RX_EV_PKT_TYPE);
309                 if (pkt_type >= FSE_AZ_RX_EV_PKT_TYPE_VLAN)
310                         flags |= EFX_PKT_VLAN_TAGGED;
311         }

313         EFSYS_PROBE4(rx_complete, uint32_t, label, uint32_t, id,
314             uint32_t, size, uint16_t, flags);

316         EFSYS_ASSERT(eecp->eec_rx != NULL);
317         should_abort = eecp->eec_rx(arg, label, id, size, flags);

319         return (should_abort);
320 }

322 static  __checkReturn   boolean_t
323 efx_ev_tx(
324         __in            efx_evq_t *eep,
325         __in            efx_qword_t *eqp,
```



```
326         __in            const efx_ev_callbacks_t *eecp,
327         __in_opt        void *arg)
328 {
329         uint32_t id;
330         uint32_t label;
331         boolean_t should_abort;

333         EFX_EV_QSTAT_INCR(eep, EV_TX);

335         if (EFX_QWORD_FIELD(*eqp, FSF_AZ_TX_EV_COMP) != 0 &&
336             EFX_QWORD_FIELD(*eqp, FSF_AZ_TX_EV_PKT_ERR) == 0 &&
337             EFX_QWORD_FIELD(*eqp, FSF_AZ_TX_EV_PKT_TOO_BIG) == 0 &&
338             EFX_QWORD_FIELD(*eqp, FSF_AZ_TX_EV_WQ_FF_FULL) == 0) {

340                 id = EFX_QWORD_FIELD(*eqp, FSF_AZ_TX_EV_DESC_PTR);
341                 label = EFX_QWORD_FIELD(*eqp, FSF_AZ_TX_EV_Q_LABEL);

343                 EFSYS_PROBE2(tx_complete, uint32_t, label, uint32_t, id);

345                 EFSYS_ASSERT(eecp->eec_tx != NULL);
346                 should_abort = eecp->eec_tx(arg, label, id);

348                 return (should_abort);
349         }

351         if (EFX_QWORD_FIELD(*eqp, FSF_AZ_TX_EV_COMP) != 0)
352                 EFSYS_PROBE3(bad_event, unsigned int, eep->ee_index,
353                              uint32_t, EFX_QWORD_FIELD(*eqp, EFX_DWORD_1),
354                              uint32_t, EFX_QWORD_FIELD(*eqp, EFX_DWORD_0));

356         if (EFX_QWORD_FIELD(*eqp, FSF_AZ_TX_EV_PKT_ERR) != 0)
357                 EFX_EV_QSTAT_INCR(eep, EV_TX_PKT_ERR);

359         if (EFX_QWORD_FIELD(*eqp, FSF_AZ_TX_EV_PKT_TOO_BIG) != 0)
360                 EFX_EV_QSTAT_INCR(eep, EV_TX_PKT_TOO_BIG);

362         if (EFX_QWORD_FIELD(*eqp, FSF_AZ_TX_EV_WQ_FF_FULL) != 0)
363                 EFX_EV_QSTAT_INCR(eep, EV_TX_WQ_FF_FULL);

365         EFX_EV_QSTAT_INCR(eep, EV_TX_UNEXPECTED);
366         return (B_FALSE);
367 }

369 static  __checkReturn   boolean_t
370 efx_ev_global(
371         __in            efx_evq_t *eep,
372         __in            efx_qword_t *eqp,
373         __in            const efx_ev_callbacks_t *eecp,
374         __in_opt        void *arg)
375 {
376         efx_nic_t *enp = eep->ee_enp;
377         efx_port_t *epp = &(enp->en_port);
378         boolean_t should_abort;

380         EFX_EV_QSTAT_INCR(eep, EV_GLOBAL);
381         should_abort = B_FALSE;

383         /* Check for a link management event */
384         if (EFX_QWORD_FIELD(*eqp, FSF_BZ_GLB_EV_XG_MNT_INTR) != 0) {
385                 EFX_EV_QSTAT_INCR(eep, EV_GLOBAL_MNT);

387                 EFSYS_PROBE(xg_mgt);

389                 epp->ep_mac_poll_needed = B_TRUE;
390         }
```

```
392         return (should_abort);
393 }

395 static  __checkReturn   boolean_t
396 efx_ev_driver(
397         __in            efx_evq_t *eep,
398         __in            efx_qword_t *eqp,
399         __in            const efx_ev_callbacks_t *eecp,
400         __in_opt        void *arg)
401 {
402         boolean_t should_abort;

404         EFX_EV_QSTAT_INCR(eep, EV_DRIVER);
405         should_abort = B_FALSE;

407         switch (EFX_QWORD_FIELD(*eqp, FSF_AZ_DRIVER_EV_SUBCODE)) {
408         case FSE_AZ_TX_DESCQ_FLS_DONE_EV: {
409                 uint32_t label;

411                 EFX_EV_QSTAT_INCR(eep, EV_DRIVER_TX_DESCQ_FLS_DONE);

413                 label = EFX_QWORD_FIELD(*eqp, FSF_AZ_DRIVER_EV_SUBDATA);

415                 EFSYS_PROBE1(tx_descq_fls_done, uint32_t, label);

417                 EFSYS_ASSERT(eecp->eec_txq_flush_done != NULL);
418                 should_abort = eecp->eec_txq_flush_done(arg, label);

420                 break;
421         }
422         case FSE_AZ_RX_DESCQ_FLS_DONE_EV: {
423                 uint32_t label;
424                 uint32_t failed;

426                 label = EFX_QWORD_FIELD(*eqp, FSF_AZ_DRIVER_EV_RX_DESCQ_ID);
427                 failed = EFX_QWORD_FIELD(*eqp, FSF_AZ_DRIVER_EV_RX_FLUSH_FAIL);

429                 EFSYS_ASSERT(eecp->eec_rxq_flush_done != NULL);
430                 EFSYS_ASSERT(eecp->eec_rxq_flush_failed != NULL);

432                 if (failed) {
433                         EFX_EV_QSTAT_INCR(eep, EV_DRIVER_RX_DESCQ_FLS_FAILED);

435                         EFSYS_PROBE1(rx_descq_fls_failed, uint32_t, label);

437                         should_abort = eecp->eec_rxq_flush_failed(arg, label);
438                 } else {
439                         EFX_EV_QSTAT_INCR(eep, EV_DRIVER_RX_DESCQ_FLS_DONE);

441                         EFSYS_PROBE1(rx_descq_fls_done, uint32_t, label);

443                         should_abort = eecp->eec_rxq_flush_done(arg, label);
444                 }

446                 break;
447         }
448         case FSE_AZ_EVQ_INIT_DONE_EV:
449                 EFSYS_ASSERT(eecp->eec_initialized != NULL);
450                 should_abort = eecp->eec_initialized(arg);

452                 break;

454         case FSE_AZ_EVQ_NOT_EN_EV:
455                 EFSYS_PROBE(evq_not_en);
456                 break;
```



```
458         case FSE_AZ_SRM_UPD_DONE_EV: {
459                 uint32_t code;

461                 EFX_EV_QSTAT_INCR(eep, EV_DRIVER_SRM_UPD_DONE);

463                 code = EFX_QWORD_FIELD(*eqp, FSF_AZ_DRIVER_EV_SUBDATA);

465                 EFSYS_ASSERT(eecp->eec_sram != NULL);
466                 should_abort = eecp->eec_sram(arg, code);

468                 break;
469         }
470         case FSE_AZ_WAKE_UP_EV: {
471                 uint32_t id;

473                 id = EFX_QWORD_FIELD(*eqp, FSF_AZ_DRIVER_EV_SUBDATA);

475                 EFSYS_ASSERT(eecp->eec_wake_up != NULL);
476                 should_abort = eecp->eec_wake_up(arg, id);

478                 break;
479         }
480         case FSE_AZ_TX_PKT_NON_TCP_UDP:
481                 EFSYS_PROBE(tx_pkt_non_tcp_udp);
482                 break;

484         case FSE_AZ_TIMER_EV: {
485                 uint32_t id;

487                 id = EFX_QWORD_FIELD(*eqp, FSF_AZ_DRIVER_EV_SUBDATA);

489                 EFSYS_ASSERT(eecp->eec_timer != NULL);
490                 should_abort = eecp->eec_timer(arg, id);

492                 break;
493         }
494         case FSE_AZ_RX_DSC_ERROR_EV:
495                 EFX_EV_QSTAT_INCR(eep, EV_DRIVER_RX_DSC_ERROR);

497                 EFSYS_PROBE(rx_dsc_error);

499                 EFSYS_ASSERT(eecp->eec_exception != NULL);
500                 should_abort = eecp->eec_exception(arg,
501                         EFX_EXCEPTION_RX_DSC_ERROR, 0);

503                 break;

505         case FSE_AZ_TX_DSC_ERROR_EV:
506                 EFX_EV_QSTAT_INCR(eep, EV_DRIVER_TX_DSC_ERROR);

508                 EFSYS_PROBE(tx_dsc_error);

510                 EFSYS_ASSERT(eecp->eec_exception != NULL);
511                 should_abort = eecp->eec_exception(arg,
512                         EFX_EXCEPTION_TX_DSC_ERROR, 0);

514                 break;

516         default:
517                 break;
518         }

520         return (should_abort);
521 }

523 static  __checkReturn   boolean_t
```

```
524 efx_ev_drv_gen(
525         __in            efx_evq_t *eep,
526         __in            efx_qword_t *eqp,
527         __in            const efx_ev_callbacks_t *eecp,
528         __in_opt        void *arg)
529 {
530         uint32_t data;
531         boolean_t should_abort;

533         EFX_EV_QSTAT_INCR(eep, EV_DRV_GEN);

535         data = EFX_QWORD_FIELD(*eqp, FSF_AZ_EV_DATA_DW0);
536         if (data >= ((uint32_t)1 << 16)) {
537                 EFSYS_PROBE3(bad_event, unsigned int, eep->ee_index,
538                              uint32_t, EFX_QWORD_FIELD(*eqp, EFX_DWORD_1),
539                              uint32_t, EFX_QWORD_FIELD(*eqp, EFX_DWORD_0));
540                 return (B_TRUE);
541         }

543         EFSYS_ASSERT(eecp->eec_software != NULL);
544         should_abort = eecp->eec_software(arg, (uint16_t)data);

546         return (should_abort);
547 }

549 #if EFSYS_OPT_MCDI

551 static  __checkReturn   boolean_t
552 efx_ev_mcdi(
553         __in            efx_evq_t *eep,
554         __in            efx_qword_t *eqp,
555         __in            const efx_ev_callbacks_t *eecp,
556         __in_opt        void *arg)
557 {
558         efx_nic_t *enp = eep->ee_enp;
559         unsigned code;
560         boolean_t should_abort = B_FALSE;

562         EFSYS_ASSERT3U(enp->en_family, ==, EFX_FAMILY_SIENA);

564         if (enp->en_family != EFX_FAMILY_SIENA)
565                 goto out;

567         EFSYS_ASSERT(eecp->eec_link_change != NULL);
568         EFSYS_ASSERT(eecp->eec_exception != NULL);
569 #if EFSYS_OPT_MON_STATS
570         EFSYS_ASSERT(eecp->eec_monitor != NULL);
571 #endif

573         EFX_EV_QSTAT_INCR(eep, EV_MCDI_RESPONSE);

575         code = EFX_QWORD_FIELD(*eqp, MCDI_EVENT_CODE);
576         switch (code) {
577         case MCDI_EVENT_CODE_BADSSERT:
578                 efx_mcdi_ev_death(enp, EINTR);
579                 break;

581         case MCDI_EVENT_CODE_CMDDONE:
582                 efx_mcdi_ev_cpl(enp,
583                                 MCDI_EV_FIELD(eqp, CMDDONE_SEQ),
584                                 MCDI_EV_FIELD(eqp, CMDDONE_DATALEN),
585                                 MCDI_EV_FIELD(eqp, CMDDONE_ERRNO));
586                 break;

588         case MCDI_EVENT_CODE_LINKCHANGE: {
589                 efx_link_mode_t link_mode;
```



```
591                 siena_phy_link_ev(enp, eqp, &link_mode);
592                 should_abort = eecp->eec_link_change(arg, link_mode);
593                 break;
594         }
595         case MCDI_EVENT_CODE_SENSOREVT: {
596 #if EFSYS_OPT_MON_STATS
597                 efx_mon_stat_t id;
598                 efx_mon_stat_value_t value;
599                 int rc;

601                 if ((rc = siena_mon_ev(enp, eqp, &id, &value)) == 0)
602                         should_abort = eecp->eec_monitor(arg, id, value);
603                 else if (rc == ENOTSUP) {
604                         should_abort = eecp->eec_exception(arg,
605                                 EFX_EXCEPTION_UNKNOWN_SENSOREVT,
606                                 MCDI_EV_FIELD(eqp, DATA));
607                 } else
608                         EFSYS_ASSERT(rc == ENODEV);     /* Wrong port */
609 #else
610                 should_abort = B_FALSE;
611 #endif
612                 break;
613         }
614         case MCDI_EVENT_CODE_SCHEDERR:
615                 /* Informational only */
616                 break;

618         case MCDI_EVENT_CODE_REBOOT:
619                 efx_mcdi_ev_death(enp, EIO);
620                 break;

622         case MCDI_EVENT_CODE_MAC_STATS_DMA:
623 #if EFSYS_OPT_MAC_STATS
624                 if (eecp->eec_mac_stats != NULL) {
625                         eecp->eec_mac_stats(arg,
626                             MCDI_EV_FIELD(eqp, MAC_STATS_DMA_GENERATION));
627                 }
628 #endif
629                 break;

631         case MCDI_EVENT_CODE_FWALERT: {
632                 uint32_t reason = MCDI_EV_FIELD(eqp, FWALERT_REASON);

634                 if (reason == MCDI_EVENT_FWALERT_REASON_SRAM_ACCESS)
635                         should_abort = eecp->eec_exception(arg,
636                                 EFX_EXCEPTION_FWALERT_SRAM,
637                                 MCDI_EV_FIELD(eqp, FWALERT_DATA));
638                 else
639                         should_abort = eecp->eec_exception(arg,
640                                 EFX_EXCEPTION_UNKNOWN_FWALERT,
641                                 MCDI_EV_FIELD(eqp, DATA));
642                 break;
643         }

645         default:
646                 EFSYS_PROBE1(mc_pcol_error, int, code);
647                 break;
648         }

650 out:
651         return (should_abort);
652 }

654 #endif  /* EFSYS_OPT_MCDI */
```

```
656         __checkReturn   int
657 efx_ev_qprime(
658         __in            efx_evq_t *eep,
659         __in            unsigned int count)
660 {
661         efx_nic_t *enp = eep->ee_enp;
662         uint32_t rptr;
663         efx_dword_t dword;
664         int rc;

666         EFSYS_ASSERT3U(eep->ee_magic, ==, EFX_EVQ_MAGIC);

668         if (!(enp->en_mod_flags & EFX_MOD_INTR)) {
669                 rc = EINVAL;
670                 goto fail1;
671         }

673         rptr = count & eep->ee_mask;

675         EFX_POPULATE_DWORD_1(dword, FRF_AZ_EVQ_RPTR, rptr);

677         EFX_BAR_TBL_WRITED(enp, FR_AZ_EVQ_RPTR_REG, eep->ee_index,
678                            &dword, B_FALSE);

680         return (0);

682 fail1:
683         EFSYS_PROBE1(fail1, int, rc);

685         return (rc);
686 }

688         __checkReturn   boolean_t
689 efx_ev_qpending(
690         __in            efx_evq_t *eep,
691         __in            unsigned int count)
692 {
693         size_t offset;
694         efx_qword_t qword;

696         EFSYS_ASSERT3U(eep->ee_magic, ==, EFX_EVQ_MAGIC);

698         offset = (count & eep->ee_mask) * sizeof (efx_qword_t);
699         EFSYS_MEM_READQ(eep->ee_esmp, offset, &qword);

701         return (EFX_QWORD_FIELD(qword, EFX_DWORD_0) != 0xffffffff &&
702                 EFX_QWORD_FIELD(qword, EFX_DWORD_1) != 0xffffffff);
703 }

705 #if EFSYS_OPT_EV_PREFETCH

707                         void
708 efx_ev_qprefetch(
709         __in            efx_evq_t *eep,
710         __in            unsigned int count)
711 {
712         unsigned int offset;

714         EFSYS_ASSERT3U(eep->ee_magic, ==, EFX_EVQ_MAGIC);

716         offset = (count & eep->ee_mask) * sizeof (efx_qword_t);
717         EFSYS_MEM_PREFETCH(eep->ee_esmp, offset);
718 }

720 #endif  /* EFSYS_OPT_EV_PREFETCH */
```



```
722 #define EFX_EV_BATCH    8

724 #define EFX_EV_PRESENT(_qword)                                          \
725         (EFX_QWORD_FIELD((_qword), EFX_DWORD_0) != 0xffffffff &&        \
726         EFX_QWORD_FIELD((_qword), EFX_DWORD_1) != 0xffffffff)

728                         void
729 efx_ev_qpoll(
730         __in            efx_evq_t *eep,
731         __inout         unsigned int *countp,
732         __in            const efx_ev_callbacks_t *eecp,
733         __in_opt        void *arg)
734 {
735         efx_qword_t ev[EFX_EV_BATCH];
736         unsigned int batch;
737         unsigned int total;
738         unsigned int count;
739         unsigned int index;
740         size_t offset;

742         EFSYS_ASSERT3U(eep->ee_magic, ==, EFX_EVQ_MAGIC);
743         EFSYS_ASSERT(countp != NULL);
744         EFSYS_ASSERT(eecp != NULL);

746         count = *countp;
747         do {
748                 /* Read up until the end of the batch period */
749                 batch = EFX_EV_BATCH - (count & (EFX_EV_BATCH - 1));
750                 offset = (count & eep->ee_mask) * sizeof (efx_qword_t);
751                 for (total = 0; total < batch; ++total) {
752                         EFSYS_MEM_READQ(eep->ee_esmp, offset, &(ev[total]));

754                         if (!EFX_EV_PRESENT(ev[total]))
755                                 break;

757                         EFSYS_PROBE3(event, unsigned int, eep->ee_index,
758                             uint32_t, EFX_QWORD_FIELD(ev[total], EFX_DWORD_1),
759                             uint32_t, EFX_QWORD_FIELD(ev[total], EFX_DWORD_0));

761                         offset += sizeof (efx_qword_t);
762                 }

764 #if EFSYS_OPT_EV_PREFETCH && (EFSYS_OPT_EV_PREFETCH_PERIOD > 1)
765                 /*
766                  * Prefetch the next batch when we get within PREFETCH_PERIOD
767                  * of a completed batch. If the batch is smaller, then prefetch
768                  * immediately.
769                  */
770                 if (total == batch && total < EFSYS_OPT_EV_PREFETCH_PERIOD)
771                         EFSYS_MEM_PREFETCH(eep->ee_esmp, offset);
772 #endif  /* EFSYS_OPT_EV_PREFETCH */

774                 /* Process the batch of events */
775                 for (index = 0; index < total; ++index) {
776                         boolean_t should_abort;
777                         uint32_t code;
778                         efx_ev_handler_t handler;

780 #if EFSYS_OPT_EV_PREFETCH
781                         /* Prefetch if we've now reached the batch period */
782                         if (total == batch &&
783                             index + EFSYS_OPT_EV_PREFETCH_PERIOD == total) {
784                                 offset = (count + batch) & eep->ee_mask;
785                                 offset *= sizeof (efx_qword_t);

787                                 EFSYS_MEM_PREFETCH(eep->ee_esmp, offset);
```

```
788                         }
789 #endif  /* EFSYS_OPT_EV_PREFETCH */
790
791                         EFX_EV_QSTAT_INCR(eep, EV_ALL);
792
793                         code = EFX_QWORD_FIELD(ev[index], FSF_AZ_EV_CODE);
794                         handler = eep->ee_handler[code];
795                         EFSYS_ASSERT(handler != NULL);
796                         should_abort = handler(eep, &(ev[index]), eecp, arg);
797                         if (should_abort) {
798                                 /* Ignore subsequent events */
799                                 total = index + 1;
800                                 break;
801                         }
802                 }
803
804                 /*
805                  * Now that the hardware has most likely moved onto dma'ing
806                  * into the next cache line, clear the processed events. Take
807                  * care to only clear out events that we've processed
808                  */
809                 EFX_SET_QWORD(ev[0]);
810                 offset = (count & eep->ee_mask) * sizeof (efx_qword_t);
811                 for (index = 0; index < total; ++index) {
812                         EFSYS_MEM_WRITEQ(eep->ee_esmp, offset, &(ev[0]));
813                         offset += sizeof (efx_qword_t);
814                 }
815
816                 count += total;
817
818         } while (total == batch);
819
820         *countp = count;
821 }
822
823                 void
824 efx_ev_qpost(
825         __in    efx_evq_t *eep,
826         __in    uint16_t data)
827 {
828         efx_nic_t *enp = eep->ee_enp;
829         efx_qword_t ev;
830         efx_oword_t oword;
831
832         EFSYS_ASSERT3U(eep->ee_magic, ==, EFX_EVQ_MAGIC);
833
834         EFX_POPULATE_QWORD_2(ev, FSF_AZ_EV_CODE, FSE_AZ_EV_CODE_DRV_GEN_EV,
835             FSF_AZ_EV_DATA_DW0, (uint32_t)data);
836
837         EFX_POPULATE_OWORD_3(oword, FRF_AZ_DRV_EV_QID, eep->ee_index,
838             EFX_DWORD_0, EFX_QWORD_FIELD(ev, EFX_DWORD_0),
839             EFX_DWORD_1, EFX_QWORD_FIELD(ev, EFX_DWORD_1));
840
841         EFX_BAR_WRITEO(enp, FR_AZ_DRV_EV_REG, &oword);
842 }
843
844         __checkReturn   int
845 efx_ev_qmoderate(
846         __in            efx_evq_t *eep,
847         __in            unsigned int us)
848 {
849         efx_nic_t *enp = eep->ee_enp;
850         efx_nic_cfg_t *encp = &(enp->en_nic_cfg);
851         unsigned int locked;
852         efx_dword_t dword;
853         int rc;
```



```
854
855         EFSYS_ASSERT3U(eep->ee_magic, ==, EFX_EVQ_MAGIC);
856
857         if (us > encp->enc_evq_timer_max_us) {
858                 rc = EINVAL;
859                 goto fail1;
860         }
861
862         /* If the value is zero then disable the timer */
863         if (us == 0) {
864                 if (enp->en_family == EFX_FAMILY_FALCON)
865                         EFX_POPULATE_DWORD_2(dword,
866                             FRF_AB_TC_TIMER_MODE, FFE_AB_TIMER_MODE_DIS,
867                             FRF_AB_TC_TIMER_VAL, 0);
868                 else
869                         EFX_POPULATE_DWORD_2(dword,
870                             FRF_CZ_TC_TIMER_MODE, FFE_CZ_TIMER_MODE_DIS,
871                             FRF_CZ_TC_TIMER_VAL, 0);
872         } else {
873                 uint32_t timer_val;
874
875                 /* Calculate the timer value in quanta */
876                 timer_val = us * 1000 / encp->enc_evq_timer_quantum_ns;
877
878                 /* Moderation value is base 0 so we need to deduct 1 */
879                 if (timer_val > 0)
880                         timer_val--;
881
882                 if (enp->en_family == EFX_FAMILY_FALCON)
883                         EFX_POPULATE_DWORD_2(dword,
884                             FRF_AB_TC_TIMER_MODE, FFE_AB_TIMER_MODE_INT_HLDOFF,
885                             FRF_AB_TIMER_VAL, timer_val);
886                 else
887                         EFX_POPULATE_DWORD_2(dword,
888                             FRF_CZ_TC_TIMER_MODE, FFE_CZ_TIMER_MODE_INT_HLDOFF,
889                             FRF_CZ_TC_TIMER_VAL, timer_val);
890         }
891
892         locked = (eep->ee_index == 0) ? 1 : 0;
893
894         EFX_BAR_TBL_WRITED(enp, FR_BZ_TIMER_COMMAND_REGP0,
895             eep->ee_index, &dword, locked);
896
897         return (0);
898
899 fail1:
900         EFSYS_PROBE1(fail1, int, rc);
901
902         return (rc);
903 }
904
905         __checkReturn   int
906 efx_ev_qcreate(
907         __in            efx_nic_t *enp,
908         __in            unsigned int index,
909         __in            efsys_mem_t *esmp,
910         __in            size_t n,
911         __in            uint32_t id,
912         __deref_out     efx_evq_t **eepp)
913 {
914         efx_nic_cfg_t *encp = &(enp->en_nic_cfg);
915         uint32_t size;
916         efx_evq_t *eep;
917         efx_oword_t oword;
918         int rc;
```

```
920         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
921         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_EV);

923         EFSYS_ASSERT3U(enp->en_ev_qcount + 1, <, encp->enc_evq_limit);

925         if (!ISP2(n) || !(n & EFX_EVQ_NEVS_MASK)) {
926                 rc = EINVAL;
927                 goto fail1;
928         }
929         if (index >= encp->enc_evq_limit) {
930                 rc = EINVAL;
931                 goto fail2;
932         }
933 #if EFSYS_OPT_RX_SCALE
934         if (enp->en_intr.ei_type == EFX_INTR_LINE &&
935             index >= EFX_MAXRSS_LEGACY) {
936                 rc = EINVAL;
937                 goto fail3;
938         }
939 #endif
940         for (size = 0; (1 << size) <= (EFX_EVQ_MAXNEVS / EFX_EVQ_MINNEVS);
941             size++)
942                 if ((1 << size) == (int)(n / EFX_EVQ_MINNEVS))
943                         break;
944         if (id + (1 << size) >= encp->enc_buftbl_limit) {
945                 rc = EINVAL;
946                 goto fail4;
947         }

949         /* Allocate an EVQ object */
950         EFSYS_KMEM_ALLOC(enp->en_esip, sizeof (efx_evq_t), eep);
951         if (eep == NULL) {
952                 rc = ENOMEM;
953                 goto fail5;
954         }

956         eep->ee_magic = EFX_EVQ_MAGIC;
957         eep->ee_enp = enp;
958         eep->ee_index = index;
959         eep->ee_mask = n - 1;
960         eep->ee_esmp = esmp;

962         /* Set up the handler table */
963         eep->ee_handler[FSE_AZ_EV_CODE_RX_EV] = efx_ev_rx;
964         eep->ee_handler[FSE_AZ_EV_CODE_TX_EV] = efx_ev_tx;
965         eep->ee_handler[FSE_AZ_EV_CODE_DRIVER_EV] = efx_ev_driver;
966         eep->ee_handler[FSE_AZ_EV_CODE_GLOBAL_EV] = efx_ev_global;
967         eep->ee_handler[FSE_AZ_EV_CODE_DRV_GEN_EV] = efx_ev_drv_gen;
968 #if EFSYS_OPT_MCDI
969         eep->ee_handler[FSE_AZ_EV_CODE_MCDI_EVRESPONSE] = efx_ev_mcdi;
970 #endif  /* EFSYS_OPT_MCDI */

972         /* Set up the new event queue */
973         if (enp->en_family != EFX_FAMILY_FALCON) {
974                 EFX_POPULATE_OWORD_1(oword, FRF_CZ_TIMER_Q_EN, 1);
975                 EFX_BAR_TBL_WRITEO(enp, FR_AZ_TIMER_TBL, index, &oword);
976         }

978         EFX_POPULATE_OWORD_3(oword, FRF_AZ_EVQ_EN, 1, FRF_AZ_EVQ_SIZE, size,
979             FRF_AZ_EVQ_BUF_BASE_ID, id);

981         EFX_BAR_TBL_WRITEO(enp, FR_AZ_EVQ_PTR_TBL, index, &oword);

983         enp->en_ev_qcount++;
984         *eepp = eep;
985         return (0);
```



```
987 fail5:
988         EFSYS_PROBE(fail5);
989 fail4:
990         EFSYS_PROBE(fail4);
991 #if EFSYS_OPT_RX_SCALE
992 fail3:
993         EFSYS_PROBE(fail3);
994 #endif
995 fail2:
996         EFSYS_PROBE(fail2);
997 fail1:
998         EFSYS_PROBE1(fail1, int, rc);

1000        return (rc);
1001 }

1003 #if EFSYS_OPT_NAMES
1004 /* START MKCONFIG GENERATED EfxEventQueueStatNamesBlock 67e9bdcd920059bd */
1005 static const char       __cs * __cs __efx_ev_qstat_name[] = {
1006         "all",
1007         "rx",
1008         "rx_ok",
1009         "rx_recovery",
1010         "rx_frm_trunc",
1011         "rx_tobe_disc",
1012         "rx_pause_frm_err",
1013         "rx_buf_owner_id_err",
1014         "rx_ipv4_hdr_chksum_err",
1015         "rx_tcp_udp_chksum_err",
1016         "rx_eth_crc_err",
1017         "rx_ip_frag_err",
1018         "rx_mcast_pkt",
1019         "rx_mcast_hash_match",
1020         "rx_tcp_ipv4",
1021         "rx_tcp_ipv6",
1022         "rx_udp_ipv4",
1023         "rx_udp_ipv6",
1024         "rx_other_ipv4",
1025         "rx_other_ipv6",
1026         "rx_non_ip",
1027         "rx_overrun",
1028         "tx",
1029         "tx_wq_ff_full",
1030         "tx_pkt_err",
1031         "tx_pkt_too_big",
1032         "tx_unexpected",
1033         "global",
1034         "global_phy",
1035         "global_mnt",
1036         "global_rx_recovery",
1037         "driver",
1038         "driver_srm_upd_done",
1039         "driver_tx_descq_fls_done",
1040         "driver_rx_descq_fls_done",
1041         "driver_rx_descq_fls_failed",
1042         "driver_rx_dsc_error",
1043         "driver_tx_dsc_error",
1044         "drv_gen",
1045         "mcdi_response",
1046 };
1047 /* END MKCONFIG GENERATED EfxEventQueueStatNamesBlock */

1049                 const char __cs *
1050 efx_ev_qstat_name(
1051         __in    efx_nic_t *enp,
```

```
1052         __in    unsigned int id)
1053 {
1054         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
1055         EFSYS_ASSERT3U(id, <, EV_NQSTATS);

1057         return (__efx_ev_qstat_name[id]);
1058 }
1059 #endif  /* EFSYS_OPT_NAMES */

1061 #if EFSYS_OPT_QSTATS
1062                                         void
1063 efx_ev_qstats_update(
1064         __in                            efx_evq_t *eep,
1065         __inout_ecount(EV_NQSTATS)      efsys_stat_t *stat)
1066 {
1067         unsigned int id;

1069         EFSYS_ASSERT3U(eep->ee_magic, ==, EFX_EVQ_MAGIC);

1071         for (id = 0; id < EV_NQSTATS; id++) {
1072                 efsys_stat_t *essp = &stat[id];

1074                 EFSYS_STAT_INCR(essp, eep->ee_stat[id]);
1075                 eep->ee_stat[id] = 0;
1076         }
1077 }
1078 #endif  /* EFSYS_OPT_QSTATS */

1080                 void
1081 efx_ev_qdestroy(
1082         __in    efx_evq_t *eep)
1083 {
1084         efx_nic_t *enp = eep->ee_enp;
1085         efx_oword_t oword;

1087         EFSYS_ASSERT3U(eep->ee_magic, ==, EFX_EVQ_MAGIC);

1089         EFSYS_ASSERT(enp->en_ev_qcount != 0);
1090         --enp->en_ev_qcount;

1092         /* Purge event queue */
1093         EFX_ZERO_OWORD(oword);

1095         EFX_BAR_TBL_WRITEO(enp, FR_AZ_EVQ_PTR_TBL,
1096             eep->ee_index, &oword);

1098         if (enp->en_family != EFX_FAMILY_FALCON) {
1099                 EFX_ZERO_OWORD(oword);
1100                 EFX_BAR_TBL_WRITEO(enp, FR_AZ_TIMER_TBL,
1101                     eep->ee_index, &oword);
1102         }

1104         /* Free the EVQ object */
1105         EFSYS_KMEM_FREE(enp->en_esip, sizeof (efx_evq_t), eep);
1106 }

1108                 void
1109 efx_ev_fini(
1110         __in    efx_nic_t *enp)
1111 {
1112         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
1113         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_INTR);
1114         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_EV);
1115         EFSYS_ASSERT(!(enp->en_mod_flags & EFX_MOD_RX));
1116         EFSYS_ASSERT(!(enp->en_mod_flags & EFX_MOD_TX));
1117         EFSYS_ASSERT3U(enp->en_ev_qcount, ==, 0);
```



```
1119         enp->en_mod_flags &= ~EFX_MOD_EV;
1120 }
1121 #endif /* ! codereview */
```

```
**********************************************************
   21666 Thu Aug 22 18:59:21 2013
new/usr/src/uts/common/io/sfxge/efx_impl.h
Merged sfxge driver
**********************************************************
```

```
/*-
 * Copyright 2007-2013 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#ifndef _SYS_EFX_IMPL_H
#define _SYS_EFX_IMPL_H

#include "efsys.h"
#include "efx.h"
#include "efx_regs.h"

#if EFSYS_OPT_FALCON
#include "falcon_impl.h"
#endif  /* EFSYS_OPT_FALCON */

#if EFSYS_OPT_SIENA
#include "siena_impl.h"
#endif  /* EFSYS_OPT_SIENA */

#ifdef  __cplusplus
extern "C" {
#endif

#define EFX_MOD_MCDI    0x00000001
#define EFX_MOD_PROBE   0x00000002
#define EFX_MOD_NVRAM   0x00000004
#define EFX_MOD_VPD     0x00000008
#define EFX_MOD_NIC     0x00000010
#define EFX_MOD_INTR    0x00000020
#define EFX_MOD_EV      0x00000040
#define EFX_MOD_RX      0x00000080
#define EFX_MOD_TX      0x00000100
#define EFX_MOD_PORT    0x00000200
#define EFX_MOD_MON     0x00000400
#define EFX_MOD_WOL     0x00000800
#define EFX_MOD_FILTER  0x00001000

#define EFX_RESET_MAC   0x00000001
#define EFX_RESET_PHY   0x00000002
```



```
typedef enum efx_mac_type_e {
        EFX_MAC_INVALID = 0,
        EFX_MAC_FALCON_GMAC,
        EFX_MAC_FALCON_XMAC,
        EFX_MAC_SIENA,
        EFX_MAC_NTYPES
} efx_mac_type_t;

typedef struct efx_mac_ops_s {
        int             (*emo_reset)(efx_nic_t *); /* optional */
        int             (*emo_poll)(efx_nic_t *, efx_link_mode_t *);
        int             (*emo_up)(efx_nic_t *, boolean_t *);
        int             (*emo_reconfigure)(efx_nic_t *);
#if EFSYS_OPT_LOOPBACK
        int             (*emo_loopback_set)(efx_nic_t *, efx_link_mode_t,
                                            efx_loopback_type_t);
#endif  /* EFSYS_OPT_LOOPBACK */
#if EFSYS_OPT_MAC_STATS
        int             (*emo_stats_upload)(efx_nic_t *, efsys_mem_t *);
        int             (*emo_stats_periodic)(efx_nic_t *, efsys_mem_t *,
                                              uint16_t, boolean_t);
        int             (*emo_stats_update)(efx_nic_t *, efsys_mem_t *,
                                            efsys_stat_t *, uint32_t *);
#endif  /* EFSYS_OPT_MAC_STATS */
} efx_mac_ops_t;

typedef struct efx_phy_ops_s {
        int             (*epo_power)(efx_nic_t *, boolean_t); /* optional */
        int             (*epo_reset)(efx_nic_t *);
        int             (*epo_reconfigure)(efx_nic_t *);
        int             (*epo_verify)(efx_nic_t *);
        int             (*epo_uplink_check)(efx_nic_t *,
                                            boolean_t *); /* optional */
        int             (*epo_downlink_check)(efx_nic_t *, efx_link_mode_t *,
                                              unsigned int *, uint32_t *);
        int             (*epo_oui_get)(efx_nic_t *, uint32_t *);
#if EFSYS_OPT_PHY_STATS
        int             (*epo_stats_update)(efx_nic_t *, efsys_mem_t *,
                                            uint32_t *);
#endif  /* EFSYS_OPT_PHY_STATS */
#if EFSYS_OPT_PHY_PROPS
#if EFSYS_OPT_NAMES
        const char      __cs *(*epo_prop_name)(efx_nic_t *, unsigned int);
#endif  /* EFSYS_OPT_PHY_PROPS */
        int             (*epo_prop_get)(efx_nic_t *, unsigned int, uint32_t,
                                        uint32_t *);
        int             (*epo_prop_set)(efx_nic_t *, unsigned int, uint32_t);
#endif  /* EFSYS_OPT_PHY_PROPS */
#if EFSYS_OPT_PHY_BIST
        int             (*epo_bist_start)(efx_nic_t *, efx_phy_bist_type_t);
        int             (*epo_bist_poll)(efx_nic_t *, efx_phy_bist_type_t,
                                         efx_phy_bist_result_t *, uint32_t *,
                                         unsigned long *, size_t);
        void            (*epo_bist_stop)(efx_nic_t *, efx_phy_bist_type_t);
#endif  /* EFSYS_OPT_PHY_BIST */
} efx_phy_ops_t;

typedef struct efx_port_s {
        efx_mac_type_t          ep_mac_type;
        uint32_t                ep_phy_type;
        uint8_t                 ep_port;
        uint32_t                ep_mac_pdu;
        uint8_t                 ep_mac_addr[6];
        efx_link_mode_t         ep_link_mode;
        boolean_t               ep_unicst;
        boolean_t               ep_brdcst;
```

```
128         unsigned int            ep_fcntl;
129         boolean_t               ep_fcntl_autoneg;
130         efx_oword_t             ep_multicst_hash[2];
131 #if EFSYS_OPT_LOOPBACK
132         efx_loopback_type_t     ep_loopback_type;
133         efx_link_mode_t         ep_loopback_link_mode;
134 #endif  /* EFSYS_OPT_LOOPBACK */
135 #if EFSYS_OPT_PHY_FLAGS
136         uint32_t                ep_phy_flags;
137 #endif  /* EFSYS_OPT_PHY_FLAGS */
138 #if EFSYS_OPT_PHY_LED_CONTROL
139         efx_phy_led_mode_t      ep_phy_led_mode;
140 #endif  /* EFSYS_OPT_PHY_LED_CONTROL */
141         efx_phy_media_type_t    ep_fixed_port_type;
142         efx_phy_media_type_t    ep_module_type;
143         uint32_t                ep_adv_cap_mask;
144         uint32_t                ep_lp_cap_mask;
145         uint32_t                ep_default_adv_cap_mask;
146         uint32_t                ep_phy_cap_mask;
147 #if EFSYS_OPT_PHY_TXC43128 || EFSYS_OPT_PHY_QT2025C
148         union {
149                 struct {
150                         unsigned int    bug10934_count;
151                 } ep_txc43128;
152                 struct {
153                         unsigned int    bug17190_count;
154                 } ep_qt2025c;
155         };
156 #endif
157         boolean_t               ep_mac_poll_needed; /* falcon only */
158         boolean_t               ep_mac_up; /* falcon only */
159         uint32_t                ep_fwver; /* falcon only */
160         boolean_t               ep_mac_drain;
161         boolean_t               ep_mac_stats_pending;
162 #if EFSYS_OPT_PHY_BIST
163         efx_phy_bist_type_t     ep_current_bist;
164 #endif
165         efx_mac_ops_t           *ep_emop;
166         efx_phy_ops_t           *ep_epop;
167 } efx_port_t;

169 typedef struct efx_mon_ops_s {
170         int     (*emo_reset)(efx_nic_t *);
171         int     (*emo_reconfigure)(efx_nic_t *);
172 #if EFSYS_OPT_MON_STATS
173         int     (*emo_stats_update)(efx_nic_t *, efsys_mem_t *,
174                                     efx_mon_stat_value_t *);
175 #endif  /* EFSYS_OPT_MON_STATS */
176 } efx_mon_ops_t;

178 typedef struct efx_mon_s {
179         efx_mon_type_t  em_type;
180         efx_mon_ops_t   *em_emop;
181 } efx_mon_t;

183 typedef struct efx_intr_s {
184         efx_intr_type_t ei_type;
185         efsys_mem_t     *ei_esmp;
186         unsigned int    ei_level;
187 } efx_intr_t;

189 typedef struct efx_nic_ops_s {
190         int     (*eno_probe)(efx_nic_t *);
191         int     (*eno_reset)(efx_nic_t *);
192         int     (*eno_init)(efx_nic_t *);
193 #if EFSYS_OPT_DIAG
```



```
194         int     (*eno_sram_test)(efx_nic_t *, efx_sram_pattern_fn_t);
195         int     (*eno_register_test)(efx_nic_t *);
196 #endif  /* EFSYS_OPT_DIAG */
197         void    (*eno_fini)(efx_nic_t *);
198         void    (*eno_unprobe)(efx_nic_t *);
199 } efx_nic_ops_t;

201 #ifndef EFX_TXQ_LIMIT_TARGET
202 # define EFX_TXQ_LIMIT_TARGET 259
203 #endif
204 #ifndef EFX_RXQ_LIMIT_TARGET
205 # define EFX_RXQ_LIMIT_TARGET 768
206 #endif
207 #ifndef EFX_TXQ_DC_SIZE
208 #define EFX_TXQ_DC_SIZE 1 /* 16 descriptors */
209 #endif
210 #ifndef EFX_RXQ_DC_SIZE
211 #define EFX_RXQ_DC_SIZE 3 /* 64 descriptors */
212 #endif

214 #if EFSYS_OPT_FILTER

216 typedef enum efx_filter_type_e {
217         EFX_FILTER_RX_TCP_FULL, /* TCP/IPv4 4-tuple {dIP,dTCP,sIP,sTCP} */
218         EFX_FILTER_RX_TCP_WILD, /* TCP/IPv4 dest    {dIP,dTCP,  -,   -} */
219         EFX_FILTER_RX_UDP_FULL, /* UDP/IPv4 4-tuple {dIP,dUDP,sIP,sUDP} */
220         EFX_FILTER_RX_UDP_WILD, /* UDP/IPv4 dest    {dIP,dUDP,  -,   -} */

222 #if EFSYS_OPT_SIENA
223         EFX_FILTER_RX_MAC_FULL, /* Ethernet {dMAC,VLAN} */
224         EFX_FILTER_RX_MAC_WILD, /* Ethernet {dMAC,   -} */

226         EFX_FILTER_TX_TCP_FULL,         /* TCP/IPv4 {dIP,dTCP,sIP,sTCP} */
227         EFX_FILTER_TX_TCP_WILD,         /* TCP/IPv4 {  -,   -,sIP,sTCP} */
228         EFX_FILTER_TX_UDP_FULL,         /* UDP/IPv4 {dIP,dTCP,sIP,sTCP} */
229         EFX_FILTER_TX_UDP_WILD,         /* UDP/IPv4 source (host, port) */

231         EFX_FILTER_TX_MAC_FULL,         /* Ethernet source (MAC address, VLAN ID
232         EFX_FILTER_TX_MAC_WILD,         /* Ethernet source (MAC address) */
233 #endif /* EFSYS_OPT_SIENA */

235         EFX_FILTER_NTYPES
236 } efx_filter_type_t;

238 typedef enum efx_filter_tbl_id_e {
239         EFX_FILTER_TBL_RX_IP = 0,
240         EFX_FILTER_TBL_RX_MAC,
241         EFX_FILTER_TBL_TX_IP,
242         EFX_FILTER_TBL_TX_MAC,
243         EFX_FILTER_NTBLS
244 } efx_filter_tbl_id_t;

246 typedef struct efx_filter_tbl_s {
247         int                     eft_size;       /* number of entries */
248         int                     eft_used;       /* active count */
249         uint32_t                *eft_bitmap;    /* active bitmap */
250         efx_filter_spec_t       *eft_spec;      /* array of saved specs */
251 } efx_filter_tbl_t;

253 typedef struct efx_filter_s {
254         efx_filter_tbl_t        ef_tbl[EFX_FILTER_NTBLS];
255         unsigned int            ef_depth[EFX_FILTER_NTYPES];
256 } efx_filter_t;


259 extern  __checkReturn   int
```

```
260 efx_filter_insert_filter(
261         __in            efx_nic_t *enp,
262         __in            efx_filter_spec_t *spec,
263         __in            boolean_t replace);

265 extern  __checkReturn   int
266 efx_filter_remove_filter(
267         __in            efx_nic_t *enp,
268         __in            efx_filter_spec_t *spec);

270 extern                  void
271 efx_filter_remove_index(
272         __inout         efx_nic_t *enp,
273         __in            efx_filter_type_t type,
274         __in            int filter_idx);

276 extern                  void
277 efx_filter_redirect_index(
278         __inout         efx_nic_t *enp,
279         __in            efx_filter_type_t type,
280         __in            int filter_index,
281         __in            int rxq_index);

283 extern  __checkReturn   int
284 efx_filter_clear_tbl(
285         __in            efx_nic_t *enp,
286         __in            efx_filter_tbl_id_t tbl);

288 #endif  /* EFSYS_OPT_FILTER */

290 #if EFSYS_OPT_NVRAM
291 typedef struct efx_nvram_ops_s {
292 #if EFSYS_OPT_DIAG
293         int     (*envo_test)(efx_nic_t *);
294 #endif  /* EFSYS_OPT_DIAG */
295         int     (*envo_size)(efx_nic_t *, efx_nvram_type_t, size_t *);
296         int     (*envo_get_version)(efx_nic_t *, efx_nvram_type_t,
297                                     uint32_t *, uint16_t *);
298         int     (*envo_rw_start)(efx_nic_t *, efx_nvram_type_t, size_t *);
299         int     (*envo_read_chunk)(efx_nic_t *, efx_nvram_type_t,
300                                    unsigned int, caddr_t, size_t);
301         int     (*envo_erase)(efx_nic_t *, efx_nvram_type_t);
302         int     (*envo_write_chunk)(efx_nic_t *, efx_nvram_type_t,
303                                     unsigned int, caddr_t, size_t);
304         void    (*envo_rw_finish)(efx_nic_t *, efx_nvram_type_t);
305         int     (*envo_set_version)(efx_nic_t *, efx_nvram_type_t, uint16_t *);

307 } efx_nvram_ops_t;
308 #endif /* EFSYS_OPT_NVRAM */

310 #if EFSYS_OPT_VPD
311 typedef struct efx_vpd_ops_s {
312         int     (*evpdo_init)(efx_nic_t *);
313         int     (*evpdo_size)(efx_nic_t *, size_t *);
314         int     (*evpdo_read)(efx_nic_t *, caddr_t, size_t);
315         int     (*evpdo_verify)(efx_nic_t *, caddr_t, size_t);
316         int     (*evpdo_reinit)(efx_nic_t *, caddr_t, size_t);
317         int     (*evpdo_get)(efx_nic_t *, caddr_t, size_t, efx_vpd_value_t *);
318         int     (*evpdo_set)(efx_nic_t *, caddr_t, size_t, efx_vpd_value_t *);
319         int     (*evpdo_next)(efx_nic_t *, caddr_t, size_t, efx_vpd_value_t *,
320                               unsigned int *);
321         int     (*evpdo_write)(efx_nic_t *, caddr_t, size_t);
322         void    (*evpdo_fini)(efx_nic_t *);
323 } efx_vpd_ops_t;
324 #endif  /* EFSYS_OPT_VPD */
```



```
326 struct efx_nic_s {
327         uint32_t                enp_magic;
328         efx_family_t            enp_family;
329         uint32_t                enp_features;
330         efsys_identifier_t      *enp_esip;
331         efsys_lock_t            *enp_eslp;
332         efsys_bar_t             *enp_esbp;
333         unsigned int            enp_mod_flags;
334         unsigned int            enp_reset_flags;
335         efx_nic_cfg_t           enp_nic_cfg;
336         efx_port_t              enp_port;
337         efx_mon_t               enp_mon;
338         efx_intr_t              enp_intr;
339         uint32_t                enp_ev_qcount;
340         uint32_t                enp_rx_qcount;
341         uint32_t                enp_tx_qcount;
342         efx_nic_ops_t           *enp_enop;
343 #if EFSYS_OPT_FILTER
344         efx_filter_t            enp_filter;
345 #endif  /* EFSYS_OPT_FILTER */
346 #if EFSYS_OPT_NVRAM
347         efx_nvram_type_t        enp_nvram_locked;
348         efx_nvram_ops_t         *enp_envop;
349 #endif  /* EFSYS_OPT_NVRAM */
350 #if EFSYS_OPT_VPD
351         efx_vpd_ops_t           *enp_evpdop;
352 #endif  /* EFSYS_OPT_VPD */
353         union {
354 #if EFSYS_OPT_FALCON
355                 struct {
356                         falcon_spi_dev_t        enu_fsd[FALCON_SPI_NTYPES];
357                         falcon_i2c_t            enu_fip;
358                         boolean_t               enu_i2c_locked;
359 #if EFSYS_OPT_FALCON_NIC_CFG_OVERRIDE
360                         const uint8_t           *enu_forced_cfg;
361 #endif  /* EFSYS_OPT_FALCON_NIC_CFG_OVERRIDE */
362                         uint8_t                 enu_mon_devid;
363 #if EFSYS_OPT_PCIE_TUNE
364                         unsigned int            enu_nlanes;
365 #endif  /* EFSYS_OPT_PCIE_TUNE */
366                         uint16_t                enu_board_rev;
367                         boolean_t               enu_internal_sram;
368                         uint8_t                 enu_sram_num_bank;
369                         uint8_t                 enu_sram_bank_size;
370                 } falcon;
371 #endif  /* EFSYS_OPT_FALCON */
372 #if EFSYS_OPT_SIENA
373                 struct {
374 #if EFSYS_OPT_MCDI
375                         efx_mcdi_iface_t        enu_mip;
376 #endif  /* EFSYS_OPT_MCDI */
377 #if EFSYS_OPT_NVRAM || EFSYS_OPT_VPD
378                         unsigned int            enu_partn_mask;
379 #endif  /* EFSYS_OPT_NVRAM || EFSYS_OPT_VPD */
380 #if EFSYS_OPT_VPD
381                         caddr_t                 enu_svpd;
382                         size_t                  enu_svpd_length;
383 #endif  /* EFSYS_OPT_VPD */
384                 } siena;
385 #endif  /* EFSYS_OPT_SIENA */
386         } en_u;
387 };


390 #define EFX_NIC_MAGIC   0x02121996
```

```
392 typedef boolean_t (*efx_ev_handler_t)(efx_evq_t *, efx_qword_t *,
393     const efx_ev_callbacks_t *, void *);

395 struct efx_evq_s {
396         uint32_t                        ee_magic;
397         efx_nic_t                       *ee_enp;
398         unsigned int                    ee_index;
399         unsigned int                    ee_mask;
400         efsys_mem_t                     *ee_esmp;
401 #if EFSYS_OPT_QSTATS
402         uint32_t                        ee_stat[EV_NQSTATS];
403 #endif  /* EFSYS_OPT_QSTATS */
404         efx_ev_handler_t                ee_handler[1 << FSF_AZ_EV_CODE_WIDTH];
405 };

407 #define EFX_EVQ_MAGIC   0x08081997

409 #define EFX_EVQ_FALCON_TIMER_QUANTUM_NS 4968 /* 621 cycles */
410 #define EFX_EVQ_SIENA_TIMER_QUANTUM_NS  6144 /* 768 cycles */

412 struct efx_rxq_s {
413         uint32_t                        er_magic;
414         efx_nic_t                       *er_enp;
415         unsigned int                    er_index;
416         unsigned int                    er_mask;
417         efsys_mem_t                     *er_esmp;
418 };

420 #define EFX_RXQ_MAGIC   0x15022005

422 struct efx_txq_s {
423         uint32_t                        et_magic;
424         efx_nic_t                       *et_enp;
425         unsigned int                    et_index;
426         unsigned int                    et_mask;
427         efsys_mem_t                     *et_esmp;
428 #if EFSYS_OPT_QSTATS
429         uint32_t                        et_stat[TX_NQSTATS];
430 #endif  /* EFSYS_OPT_QSTATS */
431 };

433 #define EFX_TXQ_MAGIC   0x05092005

435 #define EFX_MAC_ADDR_COPY(_dst, _src)                                   \
436         do {                                                            \
437                 (_dst)[0] = (_src)[0];                                  \
438                 (_dst)[1] = (_src)[1];                                  \
439                 (_dst)[2] = (_src)[2];                                  \
440                 (_dst)[3] = (_src)[3];                                  \
441                 (_dst)[4] = (_src)[4];                                  \
442                 (_dst)[5] = (_src)[5];                                  \
443         _NOTE(CONSTANTCONDITION)                                        \
444         } while (B_FALSE)

446 #if EFSYS_OPT_CHECK_REG
447 #define EFX_CHECK_REG(_enp, _reg)                                       \
448         do {                                                            \
449                 const char __cs *name = #_reg;                          \
450                 char min = name[4];                                     \
451                 char max = name[5];                                     \
452                 char rev;                                               \
453                                                                         \
454                 switch ((_enp)->en_family) {                            \
455                 case EFX_FAMILY_FALCON:                                 \
456                         rev = 'B';                                      \
457                         break;                                          \
```



```
458                                                                         \
459                 case EFX_FAMILY_SIENA:                                  \
460                         rev = 'C';                                      \
461                         break;                                          \
462                                                                         \
463                 default:                                                \
464                         rev = '?';                                      \
465                         break;                                          \
466                 }                                                       \
467                                                                         \
468                 EFSYS_ASSERT3S(rev, >=, min);                           \
469                 EFSYS_ASSERT3S(rev, <=, max);                           \
470                                                                         \
471         _NOTE(CONSTANTCONDITION)                                        \
472         } while (B_FALSE)
473 #else
474 #define EFX_CHECK_REG(_enp, _reg) do {                                  \
475         _NOTE(CONSTANTCONDITION)                                        \
476         } while(B_FALSE)
477 #endif

479 #define EFX_BAR_READD(_enp, _reg, _edp, _lock)                          \
480         do {                                                            \
481                 EFX_CHECK_REG((_enp), (_reg));                          \
482                 EFSYS_BAR_READD((_enp)->en_esbp, _reg ## _OFST,         \
483                     (_edp), (_lock));                                   \
484                 EFSYS_PROBE3(efx_bar_readd, const char *, #_reg,        \
485                     uint32_t, _reg ## _OFST,                            \
486                     uint32_t, (_edp)->ed_u32[0]);                       \
487         _NOTE(CONSTANTCONDITION)                                        \
488         } while (B_FALSE)

490 #define EFX_BAR_WRITED(_enp, _reg, _edp, _lock)                         \
491         do {                                                            \
492                 EFX_CHECK_REG((_enp), (_reg));                          \
493                 EFSYS_PROBE3(efx_bar_writed, const char *, #_reg,       \
494                     uint32_t, _reg ## _OFST,                            \
495                     uint32_t, (_edp)->ed_u32[0]);                       \
496                 EFSYS_BAR_WRITED((_enp)->en_esbp, _reg ## _OFST,        \
497                     (_edp), (_lock));                                   \
498         _NOTE(CONSTANTCONDITION)                                        \
499         } while (B_FALSE)

501 #define EFX_BAR_READQ(_enp, _reg, _eqp)                                 \
502         do {                                                            \
503                 EFX_CHECK_REG((_enp), (_reg));                          \
504                 EFSYS_BAR_READQ((_enp)->en_esbp, _reg ## _OFST,         \
505                     (_eqp));                                            \
506                 EFSYS_PROBE4(efx_bar_readq, const char *, #_reg,        \
507                     uint32_t, _reg ## _OFST,                            \
508                     uint32_t, (_eqp)->eq_u32[1],                        \
509                     uint32_t, (_eqp)->eq_u32[0]);                       \
510         _NOTE(CONSTANTCONDITION)                                        \
511         } while (B_FALSE)

513 #define EFX_BAR_WRITEQ(_enp, _reg, _eqp)                                \
514         do {                                                            \
515                 EFX_CHECK_REG((_enp), (_reg));                          \
516                 EFSYS_PROBE4(efx_bar_writeq, const char *, #_reg,       \
517                     uint32_t, _reg ## _OFST,                            \
518                     uint32_t, (_eqp)->eq_u32[1],                        \
519                     uint32_t, (_eqp)->eq_u32[0]);                       \
520                 EFSYS_BAR_WRITEQ((_enp)->en_esbp, _reg ## _OFST,        \
521                     (_eqp));                                            \
522         _NOTE(CONSTANTCONDITION)                                        \
523         } while (B_FALSE)
```

```
#define EFX_BAR_READO(_enp, _reg, _eop)                                 \
        do {                                                            \
                EFX_CHECK_REG((_enp), (_reg));                          \
                EFSYS_BAR_READO((_enp)->en_esbp, _reg ## _OFST,         \
                    (_eop), B_TRUE);                                    \
                EFSYS_PROBE6(efx_bar_reado, const char *, #_reg,        \
                    uint32_t, _reg ## _OFST,                            \
                    uint32_t, (_eop)->eo_u32[3],                        \
                    uint32_t, (_eop)->eo_u32[2],                        \
                    uint32_t, (_eop)->eo_u32[1],                        \
                    uint32_t, (_eop)->eo_u32[0]);                       \
        _NOTE(CONSTANTCONDITION)                                        \
        } while (B_FALSE)

#define EFX_BAR_WRITEO(_enp, _reg, _eop)                                \
        do {                                                            \
                EFX_CHECK_REG((_enp), (_reg));                          \
                EFSYS_PROBE6(efx_bar_writeo, const char *, #_reg,       \
                    uint32_t, _reg ## _OFST,                            \
                    uint32_t, (_eop)->eo_u32[3],                        \
                    uint32_t, (_eop)->eo_u32[2],                        \
                    uint32_t, (_eop)->eo_u32[1],                        \
                    uint32_t, (_eop)->eo_u32[0]);                       \
                EFSYS_BAR_WRITEO((_enp)->en_esbp, _reg ## _OFST,        \
                    (_eop), B_TRUE);                                    \
        _NOTE(CONSTANTCONDITION)                                        \
        } while (B_FALSE)

#define EFX_BAR_TBL_READD(_enp, _reg, _index, _edp, _lock)              \
        do {                                                            \
                EFX_CHECK_REG((_enp), (_reg));                          \
                EFSYS_BAR_READD((_enp)->en_esbp,                        \
                    (_reg ## _OFST + ((_index) * _reg ## _STEP)),       \
                    (_edp), (_lock));                                   \
                EFSYS_PROBE4(efx_bar_tbl_readd, const char *, #_reg,    \
                    uint32_t, (_index),                                 \
                    uint32_t, _reg ## _OFST,                            \
                    uint32_t, (_edp)->ed_u32[0]);                       \
        _NOTE(CONSTANTCONDITION)                                        \
        } while (B_FALSE)

#define EFX_BAR_TBL_WRITED(_enp, _reg, _index, _edp, _lock)             \
        do {                                                            \
                EFX_CHECK_REG((_enp), (_reg));                          \
                EFSYS_PROBE4(efx_bar_tbl_writed, const char *, #_reg,   \
                    uint32_t, (_index),                                 \
                    uint32_t, _reg ## _OFST,                            \
                    uint32_t, (_edp)->ed_u32[0]);                       \
                EFSYS_BAR_WRITED((_enp)->en_esbp,                       \
                    (_reg ## _OFST + ((_index) * _reg ## _STEP)),       \
                    (_edp), (_lock));                                   \
        _NOTE(CONSTANTCONDITION)                                        \
        } while (B_FALSE)

#define EFX_BAR_TBL_WRITED3(_enp, _reg, _index, _edp, _lock)            \
        do {                                                            \
                EFX_CHECK_REG((_enp), (_reg));                          \
                EFSYS_PROBE4(efx_bar_tbl_writed, const char *, #_reg,   \
                    uint32_t, (_index),                                 \
                    uint32_t, _reg ## _OFST,                            \
                    uint32_t, (_edp)->ed_u32[0]);                       \
                EFSYS_BAR_WRITED((_enp)->en_esbp,                       \
                    (_reg ## _OFST +                                    \
                    (3 * sizeof (efx_dword_t)) +                        \
                    ((_index) * _reg ## _STEP)),                        \
```



```
                    (_edp), (_lock));                                   \
        _NOTE(CONSTANTCONDITION)                                        \
        } while (B_FALSE)

#define EFX_BAR_TBL_READQ(_enp, _reg, _index, _eqp)                     \
        do {                                                            \
                EFX_CHECK_REG((_enp), (_reg));                          \
                EFSYS_BAR_READQ((_enp)->en_esbp,                        \
                    (_reg ## _OFST + ((_index) * _reg ## _STEP)),       \
                    (_eqp));                                            \
                EFSYS_PROBE5(efx_bar_tbl_readq, const char *, #_reg,    \
                    uint32_t, (_index),                                 \
                    uint32_t, _reg ## _OFST,                            \
                    uint32_t, (_eqp)->eq_u32[1],                        \
                    uint32_t, (_eqp)->eq_u32[0]);                       \
        _NOTE(CONSTANTCONDITION)                                        \
        } while (B_FALSE)

#define EFX_BAR_TBL_WRITEQ(_enp, _reg, _index, _eqp)                    \
        do {                                                            \
                EFX_CHECK_REG((_enp), (_reg));                          \
                EFSYS_PROBE5(efx_bar_tbl_writeq, const char *, #_reg,   \
                    uint32_t, (_index),                                 \
                    uint32_t, _reg ## _OFST,                            \
                    uint32_t, (_eqp)->eq_u32[1],                        \
                    uint32_t, (_eqp)->eq_u32[0]);                       \
                EFSYS_BAR_WRITEQ((_enp)->en_esbp,                       \
                    (_reg ## _OFST + ((_index) * _reg ## _STEP)),       \
                    (_eqp));                                            \
        _NOTE(CONSTANTCONDITION)                                        \
        } while (B_FALSE)

#define EFX_BAR_TBL_READO(_enp, _reg, _index, _eop)                     \
        do {                                                            \
                EFX_CHECK_REG((_enp), (_reg));                          \
                EFSYS_BAR_READO((_enp)->en_esbp,                        \
                    (_reg ## _OFST + ((_index) * _reg ## _STEP)),       \
                    (_eop), B_TRUE);                                    \
                EFSYS_PROBE7(efx_bar_tbl_reado, const char *, #_reg,    \
                    uint32_t, (_index),                                 \
                    uint32_t, _reg ## _OFST,                            \
                    uint32_t, (_eop)->eo_u32[3],                        \
                    uint32_t, (_eop)->eo_u32[2],                        \
                    uint32_t, (_eop)->eo_u32[1],                        \
                    uint32_t, (_eop)->eo_u32[0]);                       \
        _NOTE(CONSTANTCONDITION)                                        \
        } while (B_FALSE)

#define EFX_BAR_TBL_WRITEO(_enp, _reg, _index, _eop)                    \
        do {                                                            \
                EFX_CHECK_REG((_enp), (_reg));                          \
                EFSYS_PROBE7(efx_bar_tbl_writeo, const char *, #_reg,   \
                    uint32_t, (_index),                                 \
                    uint32_t, _reg ## _OFST,                            \
                    uint32_t, (_eop)->eo_u32[3],                        \
                    uint32_t, (_eop)->eo_u32[2],                        \
                    uint32_t, (_eop)->eo_u32[1],                        \
                    uint32_t, (_eop)->eo_u32[0]);                       \
                EFSYS_BAR_WRITEO((_enp)->en_esbp,                       \
                    (_reg ## _OFST + ((_index) * _reg ## _STEP)),       \
                    (_eop), B_TRUE);                                    \
        _NOTE(CONSTANTCONDITION)                                        \
        } while (B_FALSE)

extern  __checkReturn   int
efx_mac_select(
```

```
656         __in            efx_nic_t *enp);

658 extern  __checkReturn   int
659 efx_phy_probe(
660         __in            efx_nic_t *enp);

662 extern                  void
663 efx_phy_unprobe(
664         __in            efx_nic_t *enp);

666 #if EFSYS_OPT_VPD

668 /* VPD utility functions */

670 extern  __checkReturn           int
671 efx_vpd_hunk_length(
672         __in_bcount(size)       caddr_t data,
673         __in                    size_t size,
674         __out                   size_t *lengthp);

676 extern  __checkReturn           int
677 efx_vpd_hunk_verify(
678         __in_bcount(size)       caddr_t data,
679         __in                    size_t size,
680         __out_opt               boolean_t *cksummedp);

682 extern  __checkReturn           int
683 efx_vpd_hunk_reinit(
684         __in                    caddr_t data,
685         __in                    size_t size,
686         __in                    boolean_t wantpid);

688 extern  __checkReturn           int
689 efx_vpd_hunk_get(
690         __in_bcount(size)       caddr_t data,
691         __in                    size_t size,
692         __in                    efx_vpd_tag_t tag,
693         __in                    efx_vpd_keyword_t keyword,
694         __out                   unsigned int *payloadp,
695         __out                   uint8_t *paylenp);

697 extern  __checkReturn                   int
698 efx_vpd_hunk_next(
699         __in_bcount(size)               caddr_t data,
700         __in                            size_t size,
701         __out                           efx_vpd_tag_t *tagp,
702         __out                           efx_vpd_keyword_t *keyword,
703         __out_bcount_opt(*paylenp)      unsigned int *payloadp,
704         __out_opt                       uint8_t *paylenp,
705         __inout                         unsigned int *contp);

707 extern  __checkReturn           int
708 efx_vpd_hunk_set(
709         __in_bcount(size)       caddr_t data,
710         __in                    size_t size,
711         __in                    efx_vpd_value_t *evvp);

713 #endif  /* EFSYS_OPT_VPD */

715 #if EFSYS_OPT_DIAG

717 extern  efx_sram_pattern_fn_t   __cs __efx_sram_pattern_fns[];

719 typedef struct efx_register_set_s {
720         unsigned int            address;
721         unsigned int            step;
```



```
722         unsigned int            rows;
723         efx_oword_t             mask;
724 } efx_register_set_t;

726 extern  __checkReturn   int
727 efx_nic_test_registers(
728         __in            efx_nic_t *enp,
729         __in            efx_register_set_t *rsp,
730         __in            size_t count);

732 extern  __checkReturn   int
733 efx_nic_test_tables(
734         __in            efx_nic_t *enp,
735         __in            efx_register_set_t *rsp,
736         __in            efx_pattern_type_t pattern,
737         __in            size_t count);

739 #endif  /* EFSYS_OPT_DIAG */

741 #ifdef  __cplusplus
742 }
743 #endif

745 #endif  /* _SYS_EFX_IMPL_H */
746 #endif /* ! codereview */
```

```
**********************************************************
    9648 Thu Aug 22 18:59:21 2013
new/usr/src/uts/common/io/sfxge/efx_intr.c
Merged sfxge driver
**********************************************************
```

```
/*
 * Copyright 2007-2013 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#include "efsys.h"
#include "efx.h"
#include "efx_types.h"
#include "efx_regs.h"
#include "efx_impl.h"

        __checkReturn   int
efx_intr_init(
        __in            efx_nic_t *enp,
        __in            efx_intr_type_t type,
        __in            efsys_mem_t *esmp)
{
        efx_intr_t *eip = &(enp->en_intr);
        efx_oword_t oword;
        int rc;

        EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
        EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_NIC);

        if (enp->en_mod_flags & EFX_MOD_INTR) {
                rc = EINVAL;
                goto fail1;
        }

        enp->en_mod_flags |= EFX_MOD_INTR;

        eip->ei_type = type;
        eip->ei_esmp = esmp;

        /*
         * bug17213 workaround.
         *
         * Under legacy interrupts, don't share a level between fatal
         * interrupts and event queue interrupts. Under MSI-X, they
         * must share, or we won't get an interrupt.
         */
```



```
        if (enp->en_family == EFX_FAMILY_SIENA &&
            eip->ei_type == EFX_INTR_LINE)
                eip->ei_level = 0x1f;
        else
                eip->ei_level = 0;

        /* Enable all the genuinely fatal interrupts */
        EFX_SET_OWORD(oword);
        EFX_SET_OWORD_FIELD(oword, FRF_AZ_ILL_ADR_INT_KER_EN, 0);
        EFX_SET_OWORD_FIELD(oword, FRF_AZ_RBUF_OWN_INT_KER_EN, 0);
        EFX_SET_OWORD_FIELD(oword, FRF_AZ_TBUF_OWN_INT_KER_EN, 0);
        if (enp->en_family >= EFX_FAMILY_SIENA)
                EFX_SET_OWORD_FIELD(oword, FRF_CZ_SRAM_PERR_INT_P_KER_EN, 0);
        EFX_BAR_WRITEO(enp, FR_AZ_FATAL_INTR_REG_KER, &oword);

        /* Set up the interrupt address register */
        EFX_POPULATE_OWORD_3(oword,
            FRF_AZ_NORM_INT_VEC_DIS_KER, (type == EFX_INTR_MESSAGE) ? 1 : 0,
            FRF_AZ_INT_ADR_KER_DW0, EFSYS_MEM_ADDR(esmp) & 0xffffffff,
            FRF_AZ_INT_ADR_KER_DW1, EFSYS_MEM_ADDR(esmp) >> 32);
        EFX_BAR_WRITEO(enp, FR_AZ_INT_ADR_REG_KER, &oword);

        return (0);

fail1:
        EFSYS_PROBE1(fail1, int, rc);

        return (rc);
}

                        void
efx_intr_enable(
        __in            efx_nic_t *enp)
{
        efx_intr_t *eip = &(enp->en_intr);
        efx_oword_t oword;

        EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
        EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_INTR);

        EFX_BAR_READO(enp, FR_AZ_INT_EN_REG_KER, &oword);

        EFX_SET_OWORD_FIELD(oword, FRF_AZ_KER_INT_LEVE_SEL, eip->ei_level);
        EFX_SET_OWORD_FIELD(oword, FRF_AZ_DRV_INT_EN_KER, 1);
        EFX_BAR_WRITEO(enp, FR_AZ_INT_EN_REG_KER, &oword);
}

                        void
efx_intr_disable(
        __in            efx_nic_t *enp)
{
        efx_oword_t oword;

        EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
        EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_INTR);

        EFX_BAR_READO(enp, FR_AZ_INT_EN_REG_KER, &oword);
        EFX_SET_OWORD_FIELD(oword, FRF_AZ_DRV_INT_EN_KER, 0);
        EFX_BAR_WRITEO(enp, FR_AZ_INT_EN_REG_KER, &oword);

        EFSYS_SPIN(10);
}

                        void
efx_intr_disable_unlocked(
        __in            efx_nic_t *enp)
```

```
128 {
129         efx_oword_t oword;

131         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
132         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_INTR);

134         EFSYS_BAR_READO(enp->en_esbp, FR_AZ_INT_EN_REG_KER_OFST,
135                         &oword, B_FALSE);
136         EFX_SET_OWORD_FIELD(oword, FRF_AZ_DRV_INT_EN_KER, 0);
137         EFSYS_BAR_WRITEO(enp->en_esbp, FR_AZ_INT_EN_REG_KER_OFST,
138             &oword, B_FALSE);
139 }

141         __checkReturn   int
142 efx_intr_trigger(
143         __in            efx_nic_t *enp,
144         __in            unsigned int level)
145 {
146         efx_intr_t *eip = &(enp->en_intr);
147         efx_oword_t oword;
148         unsigned int count;
149         uint32_t sel;
150         int rc;

152         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
153         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_INTR);

155         /* bug16757: No event queues can be initialized */
156         EFSYS_ASSERT(!(enp->en_mod_flags & EFX_MOD_EV));

158         switch (enp->en_family) {
159         case EFX_FAMILY_FALCON:
160                 if (level > EFX_NINTR_FALCON) {
161                         rc = EINVAL;
162                         goto fail1;
163                 }
164                 break;

166         case EFX_FAMILY_SIENA:
167                 if (level > EFX_NINTR_SIENA) {
168                         rc = EINVAL;
169                         goto fail1;
170                 }
171                 break;

173         default:
174                 EFSYS_ASSERT(B_FALSE);
175                 break;
176         }

178         if (level > EFX_MASK32(FRF_AZ_KER_INT_LEVE_SEL))
179                 return (ENOTSUP); /* avoid EFSYS_PROBE() */

181         sel = level;

183         /* Trigger a test interrupt */
184         EFX_BAR_READO(enp, FR_AZ_INT_EN_REG_KER, &oword);
185         EFX_SET_OWORD_FIELD(oword, FRF_AZ_KER_INT_LEVE_SEL, sel);
186         EFX_SET_OWORD_FIELD(oword, FRF_AZ_KER_INT_KER, 1);
187         EFX_BAR_WRITEO(enp, FR_AZ_INT_EN_REG_KER, &oword);

189         /*
190          * Wait up to 100ms for the interrupt to be raised before restoring
191          * KER_INT_LEVE_SEL. Ignore a failure to raise (the caller will
192          * observe this soon enough anyway), but always reset KER_INT_LEVE_SEL
193          */
```



```
194         count = 0;
195         do {
196                 EFSYS_SPIN(100);        /* 100us */

198                 EFX_BAR_READO(enp, FR_AZ_INT_EN_REG_KER, &oword);
199         } while (EFX_OWORD_FIELD(oword, FRF_AZ_KER_INT_KER) && ++count < 1000);

201         EFX_SET_OWORD_FIELD(oword, FRF_AZ_KER_INT_LEVE_SEL, eip->ei_level);
202         EFX_BAR_WRITEO(enp, FR_AZ_INT_EN_REG_KER, &oword);

204         return (0);

206 fail1:
207         EFSYS_PROBE1(fail1, int, rc);

209         return (rc);
210 }

212 static  __checkReturn   boolean_t
213 efx_intr_check_fatal(
214         __in            efx_nic_t *enp)
215 {
216         efx_intr_t *eip = &(enp->en_intr);
217         efsys_mem_t *esmp = eip->ei_esmp;
218         efx_oword_t oword;

220         /* Read the syndrome */
221         EFSYS_MEM_READO(esmp, 0, &oword);

223         if (EFX_OWORD_FIELD(oword, FSF_AZ_NET_IVEC_FATAL_INT) != 0) {
224                 EFSYS_PROBE(fatal);

226                 /* Clear the fatal interrupt condition */
227                 EFX_SET_OWORD_FIELD(oword, FSF_AZ_NET_IVEC_FATAL_INT, 0);
228                 EFSYS_MEM_WRITEO(esmp, 0, &oword);

230                 return (B_TRUE);
231         }

233         return (B_FALSE);
234 }

236                         void
237 efx_intr_status_line(
238         __in            efx_nic_t *enp,
239         __out           boolean_t *fatalp,
240         __out           uint32_t *qmaskp)
241 {
242         efx_intr_t *eip = &(enp->en_intr);
243         efx_dword_t dword;

245         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
246         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_INTR);

248         /*
249          * Read the queue mask and implicitly acknowledge the
250          * interrupt.
251          */
252         EFX_BAR_READD(enp, FR_BZ_INT_ISR0_REG, &dword, B_FALSE);
253         *qmaskp = EFX_DWORD_FIELD(dword, EFX_DWORD_0);

255         EFSYS_PROBE1(qmask, uint32_t, *qmaskp);

257         if (*qmaskp & (1U << eip->ei_level))
258                 *fatalp = efx_intr_check_fatal(enp);
259         else
```

```
260                 *fatalp = B_FALSE;
261 }
263                         void
264 efx_intr_status_message(
265         __in            efx_nic_t *enp,
266         __in            unsigned int message,
267         __out           boolean_t *fatalp)
268 {
269         efx_intr_t *eip = &(enp->en_intr);
271         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
272         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_INTR);
274         if (message == eip->ei_level)
275                 *fatalp = efx_intr_check_fatal(enp);
276         else
277                 *fatalp = B_FALSE;
278 }
280                 void
281 efx_intr_fatal(
282         __in    efx_nic_t *enp)
283 {
284 #if EFSYS_OPT_DECODE_INTR_FATAL
285         efx_oword_t fatal;
286         efx_oword_t mem_per;
288         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
289         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_INTR);
291         EFX_BAR_READO(enp, FR_AZ_FATAL_INTR_REG_KER, &fatal);
292         EFX_ZERO_OWORD(mem_per);
294         if (EFX_OWORD_FIELD(fatal, FRF_AZ_SRM_PERR_INT_KER) != 0 ||
295             EFX_OWORD_FIELD(fatal, FRF_AZ_MEM_PERR_INT_KER) != 0)
296                 EFX_BAR_READO(enp, FR_AZ_MEM_STAT_REG, &mem_per);
298         if (EFX_OWORD_FIELD(fatal, FRF_AZ_SRAM_OOB_INT_KER) != 0)
299                 EFSYS_ERR(enp->en_esip, EFX_ERR_SRAM_OOB, 0, 0);
301         if (EFX_OWORD_FIELD(fatal, FRF_AZ_BUFID_DC_OOB_INT_KER) != 0)
302                 EFSYS_ERR(enp->en_esip, EFX_ERR_BUFID_DC_OOB, 0, 0);
304         if (EFX_OWORD_FIELD(fatal, FRF_AZ_MEM_PERR_INT_KER) != 0)
305                 EFSYS_ERR(enp->en_esip, EFX_ERR_MEM_PERR,
306                     EFX_OWORD_FIELD(mem_per, EFX_DWORD_0),
307                     EFX_OWORD_FIELD(mem_per, EFX_DWORD_1));
309         if (EFX_OWORD_FIELD(fatal, FRF_AZ_RBUF_OWN_INT_KER) != 0)
310                 EFSYS_ERR(enp->en_esip, EFX_ERR_RBUF_OWN, 0, 0);
312         if (EFX_OWORD_FIELD(fatal, FRF_AZ_TBUF_OWN_INT_KER) != 0)
313                 EFSYS_ERR(enp->en_esip, EFX_ERR_TBUF_OWN, 0, 0);
315         if (EFX_OWORD_FIELD(fatal, FRF_AZ_RDESCQ_OWN_INT_KER) != 0)
316                 EFSYS_ERR(enp->en_esip, EFX_ERR_RDESQ_OWN, 0, 0);
318         if (EFX_OWORD_FIELD(fatal, FRF_AZ_TDESCQ_OWN_INT_KER) != 0)
319                 EFSYS_ERR(enp->en_esip, EFX_ERR_TDESQ_OWN, 0, 0);
321         if (EFX_OWORD_FIELD(fatal, FRF_AZ_EVQ_OWN_INT_KER) != 0)
322                 EFSYS_ERR(enp->en_esip, EFX_ERR_EVQ_OWN, 0, 0);
324         if (EFX_OWORD_FIELD(fatal, FRF_AZ_EVF_OFLO_INT_KER) != 0)
325                 EFSYS_ERR(enp->en_esip, EFX_ERR_EVFF_OFLO, 0, 0);
```



```
327         if (EFX_OWORD_FIELD(fatal, FRF_AZ_ILL_ADR_INT_KER) != 0)
328                 EFSYS_ERR(enp->en_esip, EFX_ERR_ILL_ADDR, 0, 0);
330         if (EFX_OWORD_FIELD(fatal, FRF_AZ_SRM_PERR_INT_KER) != 0)
331                 EFSYS_ERR(enp->en_esip, EFX_ERR_SRAM_PERR,
332                     EFX_OWORD_FIELD(mem_per, EFX_DWORD_0),
333                     EFX_OWORD_FIELD(mem_per, EFX_DWORD_1));
334 #else
335         EFSYS_ASSERT(0);
336 #endif
337 }
339                 void
340 efx_intr_fini(
341         __in    efx_nic_t *enp)
342 {
343         efx_oword_t oword;
345         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
346         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_NIC);
347         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_INTR);
349         /* Clear the interrupt address register */
350         EFX_ZERO_OWORD(oword);
351         EFX_BAR_WRITEO(enp, FR_AZ_INT_ADR_REG_KER, &oword);
353         enp->en_mod_flags &= ~EFX_MOD_INTR;
354 }
355 #endif /* ! codereview */
```

```
**********************************************************
   15640 Thu Aug 22 18:59:22 2013
new/usr/src/uts/common/io/sfxge/efx_mac.c
Merged sfxge driver
**********************************************************
```

```
/*
 * Copyright 2007-2013 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#include "efsys.h"
#include "efx.h"
#include "efx_types.h"
#include "efx_impl.h"

#if EFSYS_OPT_MAC_FALCON_GMAC
#include "falcon_gmac.h"
#endif

#if EFSYS_OPT_MAC_FALCON_XMAC
#include "falcon_xmac.h"
#endif

#if EFSYS_OPT_MAC_FALCON_GMAC
static efx_mac_ops_t    __cs __efx_falcon_gmac_ops = {
        falcon_gmac_reset,              /* emo_reset */
        falcon_mac_poll,                /* emo_poll */
        falcon_mac_up,                  /* emo_up */
        falcon_gmac_reconfigure,        /* emo_reconfigure */
#if EFSYS_OPT_LOOPBACK
        falcon_mac_loopback_set,        /* emo_loopback_set */
#endif  /* EFSYS_OPT_LOOPBACK */
#if EFSYS_OPT_MAC_STATS
        falcon_mac_stats_upload,        /* emo_stats_upload */
        NULL,                           /* emo_stats_periodic */
        falcon_gmac_stats_update        /* emo_stats_update */
#endif  /* EFSYS_OPT_MAC_STATS */
};
#endif  /* EFSYS_OPT_MAC_FALCON_GMAC */

#if EFSYS_OPT_MAC_FALCON_XMAC
static efx_mac_ops_t    __cs __efx_falcon_xmac_ops = {
        falcon_xmac_reset,              /* emo_reset */
        falcon_mac_poll,                /* emo_poll */
        falcon_mac_up,                  /* emo_up */
        falcon_xmac_reconfigure,        /* emo_reconfigure */
```



```
#if EFSYS_OPT_LOOPBACK
        falcon_mac_loopback_set,        /* emo_loopback_set */
#endif  /* EFSYS_OPT_LOOPBACK */
#if EFSYS_OPT_MAC_STATS
        falcon_mac_stats_upload,        /* emo_stats_upload */
        NULL,                           /* emo_stats_periodic */
        falcon_xmac_stats_update        /* emo_stats_update */
#endif  /* EFSYS_OPT_MAC_STATS */
};
#endif  /* EFSYS_OPT_MAC_FALCON_XMAC */

#if EFSYS_OPT_SIENA
static efx_mac_ops_t    __cs __efx_siena_mac_ops = {
        NULL,                           /* emo_reset */
        siena_mac_poll,                 /* emo_poll */
        siena_mac_up,                   /* emo_up */
        siena_mac_reconfigure,          /* emo_reconfigure */
#if EFSYS_OPT_LOOPBACK
        siena_mac_loopback_set,         /* emo_loopback_set */
#endif  /* EFSYS_OPT_LOOPBACK */
#if EFSYS_OPT_MAC_STATS
        siena_mac_stats_upload,         /* emo_stats_upload */
        siena_mac_stats_periodic,       /* emo_stats_periodic */
        siena_mac_stats_update          /* emo_stats_update */
#endif  /* EFSYS_OPT_MAC_STATS */
};
#endif  /* EFSYS_OPT_SIENA */

static efx_mac_ops_t    __cs * __cs __efx_mac_ops[] = {
        NULL,
#if EFSYS_OPT_MAC_FALCON_GMAC
        &__efx_falcon_gmac_ops,
#else
        NULL,
#endif  /* EFSYS_OPT_MAC_FALCON_GMAC */
#if EFSYS_OPT_MAC_FALCON_XMAC
        &__efx_falcon_xmac_ops,
#else
        NULL,
#endif  /* EFSYS_OPT_MAC_FALCON_XMAC */
#if EFSYS_OPT_SIENA
        &__efx_siena_mac_ops,
#else
        NULL,
#endif  /* EFSYS_OPT_SIENA */
};

        __checkReturn                   int
efx_mac_pdu_set(
        __in                            efx_nic_t *enp,
        __in                            size_t pdu)
{
        efx_port_t *epp = &(enp->en_port);
        efx_mac_ops_t *emop = epp->ep_emop;
        uint32_t old_pdu;
        int rc;

        EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
        EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_PORT);
        EFSYS_ASSERT(emop != NULL);

        if (pdu < EFX_MAC_PDU_MIN) {
                rc = EINVAL;
                goto fail1;
        }
```

```
128         if (pdu > EFX_MAC_PDU_MAX) {
129                 rc = EINVAL;
130                 goto fail2;
131         }

133         old_pdu = epp->ep_mac_pdu;
134         epp->ep_mac_pdu = (uint32_t)pdu;
135         if ((rc = emop->emo_reconfigure(enp)) != 0)
136                 goto fail3;

138         return (0);

140 fail3:
141         EFSYS_PROBE(fail3);

143         epp->ep_mac_pdu = old_pdu;

145 fail2:
146         EFSYS_PROBE(fail2);
147 fail1:
148         EFSYS_PROBE1(fail1, int, rc);

150         return (rc);
151 }

153         __checkReturn                   int
154 efx_mac_addr_set(
155         __in                            efx_nic_t *enp,
156         __in                            uint8_t *addr)
157 {
158         efx_port_t *epp = &(enp->en_port);
159         efx_mac_ops_t *emop = epp->ep_emop;
160         uint8_t old_addr[6];
161         uint32_t oui;
162         int rc;

164         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
165         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_PORT);

167         if (addr[0] & 0x01) {
168                 rc = EINVAL;
169                 goto fail1;
170         }

172         oui = addr[0] << 16 | addr[1] << 8 | addr[2];
173         if (oui == 0x000000) {
174                 rc = EINVAL;
175                 goto fail2;
176         }

178         EFX_MAC_ADDR_COPY(old_addr, epp->ep_mac_addr);
179         EFX_MAC_ADDR_COPY(epp->ep_mac_addr, addr);
180         if ((rc = emop->emo_reconfigure(enp)) != 0)
181                 goto fail3;

183         return (0);

185 fail3:
186         EFSYS_PROBE(fail3);

188         EFX_MAC_ADDR_COPY(epp->ep_mac_addr, old_addr);

190 fail2:
191         EFSYS_PROBE(fail2);
192 fail1:
193         EFSYS_PROBE1(fail1, int, rc);
```



```
195         return (rc);
196 }

198         __checkReturn                   int
199 efx_mac_filter_set(
200         __in                            efx_nic_t *enp,
201         __in                            boolean_t unicst,
202         __in                            boolean_t brdcst)
203 {
204         efx_port_t *epp = &(enp->en_port);
205         efx_mac_ops_t *emop = epp->ep_emop;
206         boolean_t old_unicst;
207         boolean_t old_brdcst;
208         int rc;

210         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
211         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_PORT);

213         old_unicst = unicst;
214         old_brdcst = brdcst;

216         epp->ep_unicst = unicst;
217         epp->ep_brdcst = brdcst;

219         if ((rc = emop->emo_reconfigure(enp)) != 0)
220                 goto fail1;

222         return (0);

224 fail1:
225         EFSYS_PROBE1(fail1, int, rc);

227         epp->ep_unicst = old_unicst;
228         epp->ep_brdcst = old_brdcst;

230         return (rc);
231 }

233         __checkReturn                   int
234 efx_mac_drain(
235         __in                            efx_nic_t *enp,
236         __in                            boolean_t enabled)
237 {
238         efx_port_t *epp = &(enp->en_port);
239         efx_mac_ops_t *emop = epp->ep_emop;
240         int rc;

242         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
243         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_PORT);
244         EFSYS_ASSERT(emop != NULL);

246         if (epp->ep_mac_drain == enabled)
247                 return (0);

249         epp->ep_mac_drain = enabled;

251         if (enabled && emop->emo_reset != NULL) {
252                 if ((rc = emop->emo_reset(enp)) != 0)
253                         goto fail1;

255                 EFSYS_ASSERT(enp->en_reset_flags & EFX_RESET_MAC);
256                 enp->en_reset_flags &= ~EFX_RESET_PHY;
257         }

259         if ((rc = emop->emo_reconfigure(enp)) != 0)
```

```
260                 goto fail2;

262         return (0);

264 fail2:
265         EFSYS_PROBE(fail2);
266 fail1:
267         EFSYS_PROBE1(fail1, int, rc);

269         return (rc);
270 }

272         __checkReturn   int
273 efx_mac_up(
274         __in            efx_nic_t *enp,
275         __out           boolean_t *mac_upp)
276 {
277         efx_port_t *epp = &(enp->en_port);
278         efx_mac_ops_t *emop = epp->ep_emop;
279         int rc;

281         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
282         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_PORT);

284         if ((rc = emop->emo_up(enp, mac_upp)) != 0)
285                 goto fail1;

287         return (0);

289 fail1:
290         EFSYS_PROBE1(fail1, int, rc);

292         return (rc);
293 }

295         __checkReturn                   int
296 efx_mac_fcntl_set(
297         __in                            efx_nic_t *enp,
298         __in                            unsigned int fcntl,
299         __in                            boolean_t autoneg)
300 {
301         efx_port_t *epp = &(enp->en_port);
302         efx_mac_ops_t *emop = epp->ep_emop;
303         efx_phy_ops_t *epop = epp->ep_epop;
304         unsigned int old_fcntl;
305         boolean_t old_autoneg;
306         unsigned int old_adv_cap;
307         int rc;

309         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
310         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_PORT);

312         if ((fcntl & ~(EFX_FCNTL_RESPOND | EFX_FCNTL_GENERATE)) != 0) {
313                 rc = EINVAL;
314                 goto fail1;
315         }

317         /*
318          * Ignore a request to set flow control autonegotiation
319          * if the PHY doesn't support it.
320          */
321         if (~epp->ep_phy_cap_mask & (1 << EFX_PHY_CAP_AN))
322                 autoneg = B_FALSE;

324         old_fcntl = epp->ep_fcntl;
325         old_autoneg = autoneg;
```



```
326         old_adv_cap = epp->ep_adv_cap_mask;

328         epp->ep_fcntl = fcntl;
329         epp->ep_fcntl_autoneg = autoneg;

331         /*
332          * If the PHY supports autonegotiation, then encode the flow control
333          * settings in the advertised capabilities, and restart AN. Otherwise,
334          * just push the new settings directly to the MAC.
335          */
336         if (epp->ep_phy_cap_mask & (1 << EFX_PHY_CAP_AN)) {
337                 if (fcntl & EFX_FCNTL_RESPOND)
338                         epp->ep_adv_cap_mask |=    (1 << EFX_PHY_CAP_PAUSE |
339                                                     1 << EFX_PHY_CAP_ASYM);
340                 else
341                         epp->ep_adv_cap_mask &=   ~(1 << EFX_PHY_CAP_PAUSE |
342                                                     1 << EFX_PHY_CAP_ASYM);

344                 if (fcntl & EFX_FCNTL_GENERATE)
345                         epp->ep_adv_cap_mask ^= (1 << EFX_PHY_CAP_ASYM);

347                 if ((rc = epop->epo_reconfigure(enp)) != 0)
348                         goto fail2;

350         } else {
351                 if ((rc = emop->emo_reconfigure(enp)) != 0)
352                         goto fail2;
353         }

355         return (0);

357 fail2:
358         EFSYS_PROBE(fail2);

360         epp->ep_fcntl = old_fcntl;
361         epp->ep_fcntl_autoneg = old_autoneg;
362         epp->ep_adv_cap_mask = old_adv_cap;

364 fail1:
365         EFSYS_PROBE1(fail1, int, rc);

367         return (rc);
368 }

370                         void
371 efx_mac_fcntl_get(
372         __in            efx_nic_t *enp,
373         __out           unsigned int *fcntl_wantedp,
374         __out           unsigned int *fcntl_linkp)
375 {
376         efx_port_t *epp = &(enp->en_port);
377         unsigned int wanted;

379         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
380         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_PORT);

382         /*
383          * If the PHY supports auto negotiation, then the requested flow
384          * control settings are encoded in the advertised capabilities.
385          */
386         if (epp->ep_phy_cap_mask & (1 << EFX_PHY_CAP_AN)) {
387                 wanted = 0;

389                 if (epp->ep_adv_cap_mask & (1 << EFX_PHY_CAP_PAUSE))
390                         wanted = EFX_FCNTL_RESPOND | EFX_FCNTL_GENERATE;
391                 if (epp->ep_adv_cap_mask & (1 << EFX_PHY_CAP_ASYM))
```

```
392                         wanted ^= EFX_FCNTL_GENERATE;
393         } else
394                 wanted = epp->ep_fcntl;

396         *fcntl_linkp = epp->ep_fcntl;
397         *fcntl_wantedp = wanted;
398 }

400         __checkReturn                   int
401 efx_mac_hash_set(
402         __in                            efx_nic_t *enp,
403         __in_ecount(EFX_MAC_HASH_BITS)  unsigned int const *bucket)
404 {
405         efx_port_t *epp = &(enp->en_port);
406         efx_mac_ops_t *emop = epp->ep_emop;
407         efx_oword_t old_hash[2];
408         unsigned int index;
409         int rc;

411         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
412         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_PORT);

414         memcpy(old_hash, epp->ep_multicst_hash, sizeof (old_hash));

416         /* Set the lower 128 bits of the hash */
417         EFX_ZERO_OWORD(epp->ep_multicst_hash[0]);
418         for (index = 0; index < 128; index++) {
419                 if (bucket[index] != 0)
420                         EFX_SET_OWORD_BIT(epp->ep_multicst_hash[0], index);
421         }

423         /* Set the upper 128 bits of the hash */
424         EFX_ZERO_OWORD(epp->ep_multicst_hash[1]);
425         for (index = 0; index < 128; index++) {
426                 if (bucket[index + 128] != 0)
427                         EFX_SET_OWORD_BIT(epp->ep_multicst_hash[1], index);
428         }

430         if ((rc = emop->emo_reconfigure(enp)) != 0)
431                 goto fail1;

433         return (0);

435 fail1:
436         EFSYS_PROBE1(fail1, int, rc);

438         memcpy(epp->ep_multicst_hash, old_hash, sizeof (old_hash));

440         return (rc);
441 }

443 #if EFSYS_OPT_MAC_STATS

445 #if EFSYS_OPT_NAMES

447 /* START MKCONFIG GENERATED EfxMacStatNamesBlock adf707adba80813e */
448 static const char       __cs * __cs __efx_mac_stat_name[] = {
449         "rx_octets",
450         "rx_pkts",
451         "rx_unicst_pkts",
452         "rx_multicst_pkts",
453         "rx_brdcst_pkts",
454         "rx_pause_pkts",
455         "rx_le_64_pkts",
456         "rx_65_to_127_pkts",
457         "rx_128_to_255_pkts",
```



```
458         "rx_256_to_511_pkts",
459         "rx_512_to_1023_pkts",
460         "rx_1024_to_15xx_pkts",
461         "rx_ge_15xx_pkts",
462         "rx_errors",
463         "rx_fcs_errors",
464         "rx_drop_events",
465         "rx_false_carrier_errors",
466         "rx_symbol_errors",
467         "rx_align_errors",
468         "rx_internal_errors",
469         "rx_jabber_pkts",
470         "rx_lane0_char_err",
471         "rx_lane1_char_err",
472         "rx_lane2_char_err",
473         "rx_lane3_char_err",
474         "rx_lane0_disp_err",
475         "rx_lane1_disp_err",
476         "rx_lane2_disp_err",
477         "rx_lane3_disp_err",
478         "rx_match_fault",
479         "rx_nodesc_drop_cnt",
480         "tx_octets",
481         "tx_pkts",
482         "tx_unicst_pkts",
483         "tx_multicst_pkts",
484         "tx_brdcst_pkts",
485         "tx_pause_pkts",
486         "tx_le_64_pkts",
487         "tx_65_to_127_pkts",
488         "tx_128_to_255_pkts",
489         "tx_256_to_511_pkts",
490         "tx_512_to_1023_pkts",
491         "tx_1024_to_15xx_pkts",
492         "tx_ge_15xx_pkts",
493         "tx_errors",
494         "tx_sgl_col_pkts",
495         "tx_mult_col_pkts",
496         "tx_ex_col_pkts",
497         "tx_late_col_pkts",
498         "tx_def_pkts",
499         "tx_ex_def_pkts",
500 };
501 /* END MKCONFIG GENERATED EfxMacStatNamesBlock */

503         __checkReturn                   const char __cs *
504 efx_mac_stat_name(
505         __in                            efx_nic_t *enp,
506         __in                            unsigned int id)
507 {
508         _NOTE(ARGUNUSED(enp))
509         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);

511         EFSYS_ASSERT3U(id, <, EFX_MAC_NSTATS);
512         return (__efx_mac_stat_name[id]);
513 }

515 #endif  /* EFSYS_OPT_STAT_NAME */

517         __checkReturn                   int
518 efx_mac_stats_upload(
519         __in                            efx_nic_t *enp,
520         __in                            efsys_mem_t *esmp)
521 {
522         efx_port_t *epp = &(enp->en_port);
523         efx_mac_ops_t *emop = epp->ep_emop;
```

```
524         int rc;

526         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
527         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_PORT);
528         EFSYS_ASSERT(emop != NULL);

530         /*
531          * Don't assert !ep_mac_stats_pending, because the client might
532          * have failed to finalise statistics when previously stopping
533          * the port.
534          */
535         if ((rc = emop->emo_stats_upload(enp, esmp)) != 0)
536                 goto fail1;

538         epp->ep_mac_stats_pending = B_TRUE;

540         return (0);

542 fail1:
543         EFSYS_PROBE1(fail1, int, rc);

545         return (rc);
546 }

548         __checkReturn                   int
549 efx_mac_stats_periodic(
550         __in                            efx_nic_t *enp,
551         __in                            efsys_mem_t *esmp,
552         __in                            uint16_t period_ms,
553         __in                            boolean_t events)
554 {
555         efx_port_t *epp = &(enp->en_port);
556         efx_mac_ops_t *emop = epp->ep_emop;
557         int rc;

559         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
560         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_PORT);

562         EFSYS_ASSERT(emop != NULL);

564         if (emop->emo_stats_periodic == NULL) {
565                 rc = EINVAL;
566                 goto fail1;
567         }

569         if ((rc = emop->emo_stats_periodic(enp, esmp, period_ms, events)) != 0)
570                 goto fail2;

572         return (0);

574 fail2:
575         EFSYS_PROBE(fail2);
576 fail1:
577         EFSYS_PROBE1(fail1, int, rc);

579         return (rc);
580 }


583         __checkReturn                   int
584 efx_mac_stats_update(
585         __in                            efx_nic_t *enp,
586         __in                            efsys_mem_t *esmp,
587         __inout_ecount(EFX_MAC_NSTATS)  efsys_stat_t *essp,
588         __in                            uint32_t *generationp)
589 {
```



```
590         efx_port_t *epp = &(enp->en_port);
591         efx_mac_ops_t *emop = epp->ep_emop;
592         int rc;

594         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
595         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_PORT);
596         EFSYS_ASSERT(emop != NULL);

598         rc = emop->emo_stats_update(enp, esmp, essp, generationp);
599         if (rc == 0)
600                 epp->ep_mac_stats_pending = B_FALSE;

602         return (rc);
603 }

605 #endif  /* EFSYS_OPT_MAC_STATS */

607         __checkReturn                   int
608 efx_mac_select(
609         __in                            efx_nic_t *enp)
610 {
611         efx_port_t *epp = &(enp->en_port);
612         efx_mac_type_t type = EFX_MAC_INVALID;
613         efx_mac_ops_t *emop;
614         int rc = EINVAL;

616 #if EFSYS_OPT_SIENA
617         if (enp->en_family == EFX_FAMILY_SIENA) {
618                 type = EFX_MAC_SIENA;
619                 goto chosen;
620         }
621 #endif

623 #if EFSYS_OPT_FALCON
624         switch (epp->ep_link_mode) {
625 #if EFSYS_OPT_MAC_FALCON_GMAC
626         case EFX_LINK_100HDX:
627         case EFX_LINK_100FDX:
628         case EFX_LINK_1000HDX:
629         case EFX_LINK_1000FDX:
630                 type = EFX_MAC_FALCON_GMAC;
631                 goto chosen;
632 #endif  /* EFSYS_OPT_FALCON_GMAC */

634 #if EFSYS_OPT_MAC_FALCON_XMAC
635         case EFX_LINK_10000FDX:
636                 type = EFX_MAC_FALCON_XMAC;
637                 goto chosen;
638 #endif  /* EFSYS_OPT_FALCON_XMAC */

640         default:
641 #if EFSYS_OPT_MAC_FALCON_GMAC && EFSYS_OPT_MAC_FALCON_XMAC
642                 /* Only initialise a MAC supported by the PHY */
643                 if (epp->ep_phy_cap_mask &
644                     ((1 << EFX_PHY_CAP_1000FDX) |
645                     (1 << EFX_PHY_CAP_1000HDX) |
646                     (1 << EFX_PHY_CAP_100FDX) |
647                     (1 << EFX_PHY_CAP_100HDX) |
648                     (1 << EFX_PHY_CAP_10FDX) |
649                     (1 << EFX_PHY_CAP_10FDX)))
650                         type = EFX_MAC_FALCON_GMAC;
651                 else
652                         type = EFX_MAC_FALCON_XMAC;
653 #elif EFSYS_OPT_MAC_FALCON_GMAC
654                 type = EFX_MAC_FALCON_GMAC;
655 #else
```

```
656                 type = EFX_MAC_FALCON_XMAC;
657 #endif
658                 goto chosen;
659         }
660 #endif  /* EFSYS_OPT_FALCON */

662 chosen:
663         EFSYS_ASSERT(type != EFX_MAC_INVALID);
664         EFSYS_ASSERT3U(type, <, EFX_MAC_NTYPES);
665         emop = epp->ep_emop = (efx_mac_ops_t *)__efx_mac_ops[type];
666         EFSYS_ASSERT(emop != NULL);

668         epp->ep_mac_type = type;

670         if (emop->emo_reset != NULL) {
671                 if ((rc = emop->emo_reset(enp)) != 0)
672                         goto fail1;

674                 EFSYS_ASSERT(enp->en_reset_flags & EFX_RESET_MAC);
675                 enp->en_reset_flags &= ~EFX_RESET_MAC;
676         }

678         return (0);

680 fail1:
681         EFSYS_PROBE1(fail1, int, rc);

683         return (rc);
684 }
685 #endif /* ! codereview */
```

```
**********************************************************
   19033 Thu Aug 22 18:59:22 2013
new/usr/src/uts/common/io/sfxge/efx_mcdi.c
Merged sfxge driver
**********************************************************
```

```
/*
 * Copyright 2008-2013 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#include "efsys.h"
#include "efx.h"
#include "efx_types.h"
#include "efx_regs.h"
#include "efx_regs_mcdi.h"
#include "efx_impl.h"

#if EFSYS_OPT_MCDI

/*
 * A reboot/assertion causes the MCDI status word to be set after the
 * command word is set or a REBOOT event is sent. If we notice a reboot
 * via these mechanisms then wait 10ms for the status word to be set.
 */
#define MCDI_STATUS_SLEEP_US    10000

                        void
efx_mcdi_request_start(
        __in            efx_nic_t *enp,
        __in            efx_mcdi_req_t *emrp,
        __in            boolean_t ev_cpl)
{
        efx_mcdi_iface_t *emip = &(enp->en_u.siena.enu_mip);
        efx_dword_t dword;
        unsigned int seq;
        unsigned int xflags;
        unsigned int pdur;
        unsigned int dbr;
        unsigned int pos;
        int state;

        EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
        EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_MCDI);
        EFSYS_ASSERT3U(enp->en_features, &, EFX_FEATURE_MCDI);

        switch (emip->emi_port) {
```



```
        case 1:
                pdur = MC_SMEM_P0_PDU_OFST >> 2;
                dbr = MC_SMEM_P0_DOORBELL_OFST >> 2;
                break;
        case 2:
                pdur = MC_SMEM_P1_PDU_OFST >> 2;
                dbr = MC_SMEM_P1_DOORBELL_OFST >> 2;
                break;
        default:
                EFSYS_ASSERT(0);
                pdur = dbr = 0;
        };

        /*
         * efx_mcdi_request_start() is naturally serialised against both
         * efx_mcdi_request_poll() and efx_mcdi_ev_cpl()/efx_mcdi_ev_death(),
         * by virtue of there only being one oustanding MCDI request.
         * Unfortunately, upper layers may also call efx_mcdi_request_abort()
         * at any time, to timeout a pending mcdi request, That request may
         * then subsequently complete, meaning efx_mcdi_ev_cpl() or
         * efx_mcdi_ev_death() may end up running in parallel with
         * efx_mcdi_request_start(). This race is handled by ensuring that
         * %emi_pending_req, %emi_ev_cpl and %emi_seq are protected by the
         * en_eslp lock.
         */
        EFSYS_LOCK(enp->en_eslp, state);
        EFSYS_ASSERT(emip->emi_pending_req == NULL);
        emip->emi_pending_req = emrp;
        emip->emi_ev_cpl = ev_cpl;
        emip->emi_poll_cnt = 0;
        seq = emip->emi_seq++ & 0xf;
        EFSYS_UNLOCK(enp->en_eslp, state);

        xflags = 0;
        if (ev_cpl)
                xflags |= MCDI_HEADER_XFLAGS_EVREQ;

        /* Construct the header in shared memory */
        EFX_POPULATE_DWORD_6(dword,
                             MCDI_HEADER_CODE, emrp->emr_cmd,
                             MCDI_HEADER_RESYNC, 1,
                             MCDI_HEADER_DATALEN, emrp->emr_in_length,
                             MCDI_HEADER_SEQ, seq,
                             MCDI_HEADER_RESPONSE, 0,
                             MCDI_HEADER_XFLAGS, xflags);
        EFX_BAR_TBL_WRITED(enp, FR_CZ_MC_TREG_SMEM, pdur, &dword, B_TRUE);

        for (pos = 0; pos < emrp->emr_in_length; pos += sizeof (efx_dword_t)) {
                memcpy(&dword, MCDI_IN(*emrp, efx_dword_t, pos),
                    MIN(sizeof (dword), emrp->emr_in_length - pos));
                EFX_BAR_TBL_WRITED(enp, FR_CZ_MC_TREG_SMEM,
                    pdur + 1 + (pos >> 2), &dword, B_FALSE);
        }

        /* Ring the doorbell */
        EFX_POPULATE_DWORD_1(dword, EFX_DWORD_0, 0xd004be11);
        EFX_BAR_TBL_WRITED(enp, FR_CZ_MC_TREG_SMEM, dbr, &dword, B_FALSE);
}

static                  void
efx_mcdi_request_copyout(
        __in            efx_nic_t *enp,
        __in            efx_mcdi_req_t *emrp)
{
        efx_mcdi_iface_t *emip = &(enp->en_u.siena.enu_mip);
        unsigned int pos;
```

```
128         unsigned int pdur;
129         efx_dword_t data;
130 
131         pdur = (emip->emi_port == 1)
132             ? MC_SMEM_P0_PDU_OFST >> 2
133             : MC_SMEM_P1_PDU_OFST >> 2;
134 
135         /* Copy payload out if caller supplied buffer */
136         if (emrp->emr_out_buf != NULL) {
137                 size_t bytes = MIN(emrp->emr_out_length_used,
138                                    emrp->emr_out_length);
139                 for (pos = 0; pos < bytes; pos += sizeof (efx_dword_t)) {
140                         EFX_BAR_TBL_READD(enp, FR_CZ_MC_TREG_SMEM,
141                             pdur + 1 + (pos >> 2), &data, B_FALSE);
142                         memcpy(MCDI_OUT(*emrp, efx_dword_t, pos), &data,
143                             MIN(sizeof (data), bytes - pos));
144                 }
145         }
146 }
147 
148 static                  int
149 efx_mcdi_request_errcode(
150         __in            unsigned int err)
151 {
152 
153         switch (err) {
154         case MC_CMD_ERR_ENOENT:
155                 return (ENOENT);
156         case MC_CMD_ERR_EINTR:
157                 return (EINTR);
158         case MC_CMD_ERR_EACCES:
159                 return (EACCES);
160         case MC_CMD_ERR_EBUSY:
161                 return (EBUSY);
162         case MC_CMD_ERR_EINVAL:
163                 return (EINVAL);
164         case MC_CMD_ERR_EDEADLK:
165                 return (EDEADLK);
166         case MC_CMD_ERR_ENOSYS:
167                 return (ENOTSUP);
168         case MC_CMD_ERR_ETIME:
169                 return (ETIMEDOUT);
170 #ifdef WITH_MCDI_V2
171         case MC_CMD_ERR_EAGAIN:
172                 return (EAGAIN);
173         case MC_CMD_ERR_ENOSPC:
174                 return (ENOSPC);
175 #endif
176         default:
177                 EFSYS_PROBE1(mc_pcol_error, int, err);
178                 return (EIO);
179         }
180 }
181 
182 static                  void
183 efx_mcdi_raise_exception(
184         __in            efx_nic_t *enp,
185         __in_opt        efx_mcdi_req_t *emrp,
186         __in            int rc)
187 {
188         efx_mcdi_iface_t *emip = &(enp->en_u.siena.enu_mip);
189         const efx_mcdi_transport_t *emtp = emip->emi_mtp;
190         efx_mcdi_exception_t exception;
191 
192         /* Reboot or Assertion failure only */
193         EFSYS_ASSERT(rc == EIO || rc == EINTR);
```



```
194 
195         /*
196          * If MC_CMD_REBOOT causes a reboot (dependent on parameters),
197          * then the EIO is not worthy of an exception.
198          */
199         if (emrp != NULL && emrp->emr_cmd == MC_CMD_REBOOT && rc == EIO)
200                 return;
201 
202         exception = (rc == EIO)
203                 ? EFX_MCDI_EXCEPTION_MC_REBOOT
204                 : EFX_MCDI_EXCEPTION_MC_BADASSERT;
205 
206         emtp->emt_exception(emtp->emt_context, exception);
207 }
208 
209 static                  int
210 efx_mcdi_poll_reboot(
211         __in            efx_nic_t *enp)
212 {
213 #if 1
214         /*
215          * XXX Bug 25922, bug 26099: This function is not being used
216          * properly.  Until its callers are fixed, it should always
217          * return 0.
218          */
219         _NOTE(ARGUNUSED(enp))
220         return (0);
221 #else
222         efx_mcdi_iface_t *emip = &(enp->en_u.siena.enu_mip);
223         unsigned int rebootr;
224         efx_dword_t dword;
225         uint32_t value;
226 
227         EFSYS_ASSERT(emip->emi_port == 1 || emip->emi_port == 2);
228         rebootr = ((emip->emi_port == 1)
229             ? MC_SMEM_P0_STATUS_OFST >> 2
230             : MC_SMEM_P1_STATUS_OFST >> 2);
231 
232         EFX_BAR_TBL_READD(enp, FR_CZ_MC_TREG_SMEM, rebootr, &dword, B_FALSE);
233         value = EFX_DWORD_FIELD(dword, EFX_DWORD_0);
234 
235         if (value == 0)
236                 return (0);
237 
238         EFX_ZERO_DWORD(dword);
239         EFX_BAR_TBL_WRITED(enp, FR_CZ_MC_TREG_SMEM, rebootr, &dword, B_FALSE);
240 
241         if (value == MC_STATUS_DWORD_ASSERT)
242                 return (EINTR);
243         else
244                 return (EIO);
245 #endif
246 }
247 
248         __checkReturn   boolean_t
249 efx_mcdi_request_poll(
250         __in            efx_nic_t *enp)
251 {
252         efx_mcdi_iface_t *emip = &(enp->en_u.siena.enu_mip);
253         efx_mcdi_req_t *emrp;
254         efx_dword_t dword;
255         unsigned int pdur;
256         unsigned int seq;
257         unsigned int length;
258         int state;
259         int rc;
```

```
261         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
262         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_MCDI);
263         EFSYS_ASSERT3U(enp->en_family, ==, EFX_FAMILY_SIENA);
264         EFSYS_ASSERT3U(enp->en_features, &, EFX_FEATURE_MCDI);

266         /* Serialise against post-watchdog efx_mcdi_ev* */
267         EFSYS_LOCK(enp->en_eslp, state);

269         EFSYS_ASSERT(emip->emi_pending_req != NULL);
270         EFSYS_ASSERT(!emip->emi_ev_cpl);
271         emrp = emip->emi_pending_req;

273         /* Check for reboot atomically w.r.t efx_mcdi_request_start */
274         if (emip->emi_poll_cnt++ == 0) {
275                 if ((rc = efx_mcdi_poll_reboot(enp)) != 0) {
276                         emip->emi_pending_req = NULL;
277                         EFSYS_UNLOCK(enp->en_eslp, state);

279                         goto fail1;
280                 }
281         }

283         EFSYS_ASSERT(emip->emi_port == 1 || emip->emi_port == 2);
284         pdur = (emip->emi_port == 1)
285             ? MC_SMEM_P0_PDU_OFST >> 2
286             : MC_SMEM_P1_PDU_OFST >> 2;

288         /* Read the command header */
289         EFX_BAR_TBL_READD(enp, FR_CZ_MC_TREG_SMEM, pdur, &dword, B_FALSE);
290         if (EFX_DWORD_FIELD(dword, MCDI_HEADER_RESPONSE) == 0) {
291                 EFSYS_UNLOCK(enp->en_eslp, state);
292                 return (B_FALSE);
293         }

295         /* Request complete */
296         emip->emi_pending_req = NULL;
297         seq = (emip->emi_seq - 1) & 0xf;

299         /* Check for synchronous reboot */
300         if (EFX_DWORD_FIELD(dword, MCDI_HEADER_ERROR) != 0 &&
301             EFX_DWORD_FIELD(dword, MCDI_HEADER_DATALEN) == 0) {
302                 /* Consume status word */
303                 EFSYS_SPIN(MCDI_STATUS_SLEEP_US);
304                 efx_mcdi_poll_reboot(enp);
305                 EFSYS_UNLOCK(enp->en_eslp, state);
306                 rc = EIO;
307                 goto fail2;
308         }

310         EFSYS_UNLOCK(enp->en_eslp, state);

312         /* Check that the returned data is consistent */
313         if (EFX_DWORD_FIELD(dword, MCDI_HEADER_CODE) != emrp->emr_cmd ||
314             EFX_DWORD_FIELD(dword, MCDI_HEADER_SEQ) != seq) {
315                 /* Response is for a different request */
316                 rc = EIO;
317                 goto fail3;
318         }

320         length = EFX_DWORD_FIELD(dword, MCDI_HEADER_DATALEN);
321         if (EFX_DWORD_FIELD(dword, MCDI_HEADER_ERROR)) {
322                 efx_dword_t errdword;
323                 int errcode;

325                 EFSYS_ASSERT3U(length, ==, 4);
```



```
326                 EFX_BAR_TBL_READD(enp, FR_CZ_MC_TREG_SMEM,
327                     pdur + 1 + (MC_CMD_ERR_CODE_OFST >> 2),
328                     &errdword, B_FALSE);
329                 errcode = EFX_DWORD_FIELD(errdword, EFX_DWORD_0);
330                 rc = efx_mcdi_request_errcode(errcode);
331                 EFSYS_PROBE2(mcdi_err, int, emrp->emr_cmd, int, errcode);
332                 goto fail4;

334         } else {
335                 emrp->emr_out_length_used = length;
336                 emrp->emr_rc = 0;
337                 efx_mcdi_request_copyout(enp, emrp);
338         }

340         goto out;

342 fail4:
343         EFSYS_PROBE(fail4);
344 fail3:
345         EFSYS_PROBE(fail3);
346 fail2:
347         EFSYS_PROBE(fail2);
348 fail1:
349         EFSYS_PROBE1(fail1, int, rc);

351         /* Fill out error state */
352         emrp->emr_rc = rc;
353         emrp->emr_out_length_used = 0;

355         /* Reboot/Assertion */
356         if (rc == EIO || rc == EINTR)
357                 efx_mcdi_raise_exception(enp, emrp, rc);

359 out:
360         return (B_TRUE);
361 }

363                         void
364 efx_mcdi_execute(
365         __in            efx_nic_t *enp,
366         __in            efx_mcdi_req_t *emrp)
367 {
368         efx_mcdi_iface_t *emip = &(enp->en_u.siena.enu_mip);
369         const efx_mcdi_transport_t *emtp = emip->emi_mtp;

371         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_MCDI);
372         EFSYS_ASSERT3U(enp->en_family, ==, EFX_FAMILY_SIENA);
373         EFSYS_ASSERT3U(enp->en_features, &, EFX_FEATURE_MCDI);

375         emtp->emt_execute(emtp->emt_context, emrp);
376 }

378                         void
379 efx_mcdi_ev_cpl(
380         __in            efx_nic_t *enp,
381         __in            unsigned int seq,
382         __in            unsigned int outlen,
383         __in            int errcode)
384 {
385         efx_mcdi_iface_t *emip = &(enp->en_u.siena.enu_mip);
386         const efx_mcdi_transport_t *emtp = emip->emi_mtp;
387         efx_mcdi_req_t *emrp;
388         int state;

390         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_MCDI);
391         EFSYS_ASSERT3U(enp->en_family, ==, EFX_FAMILY_SIENA);
```

```
392         EFSYS_ASSERT3U(enp->en_features, &, EFX_FEATURE_MCDI);

394         /*
395          * Serialise against efx_mcdi_request_poll()/efx_mcdi_request_start()
396          * when we're completing an aborted request.
397          */
398         EFSYS_LOCK(enp->en_eslp, state);
399         if (emip->emi_pending_req == NULL || !emip->emi_ev_cpl ||
400             (seq != ((emip->emi_seq - 1) & 0xf))) {
401                 EFSYS_ASSERT(emip->emi_aborted > 0);
402                 if (emip->emi_aborted > 0)
403                         --emip->emi_aborted;
404                 EFSYS_UNLOCK(enp->en_eslp, state);
405                 return;
406         }

408         emrp = emip->emi_pending_req;
409         emip->emi_pending_req = NULL;
410         EFSYS_UNLOCK(enp->en_eslp, state);

412         /*
413          * Fill out the remaining hdr fields, and copyout the payload
414          * if the user supplied an output buffer.
415          */
416         if (errcode != 0) {
417                 EFSYS_PROBE2(mcdi_err, int, emrp->emr_cmd,
418                     int, errcode);
419                 emrp->emr_out_length_used = 0;
420                 emrp->emr_rc = efx_mcdi_request_errcode(errcode);
421         } else {
422                 emrp->emr_out_length_used = outlen;
423                 emrp->emr_rc = 0;
424                 efx_mcdi_request_copyout(enp, emrp);
425         }

427         emtp->emt_ev_cpl(emtp->emt_context);
428 }

430                         void
431 efx_mcdi_ev_death(
432         __in            efx_nic_t *enp,
433         __in            int rc)
434 {
435         efx_mcdi_iface_t *emip = &(enp->en_u.siena.enu_mip);
436         const efx_mcdi_transport_t *emtp = emip->emi_mtp;
437         efx_mcdi_req_t *emrp = NULL;
438         boolean_t ev_cpl;
439         int state;

441         /*
442          * The MCDI request (if there is one) has been terminated, either
443          * by a BADASSERT or REBOOT event.
444          *
445          * If there is an oustanding event-completed MCDI operation, then we
446          * will never receive the completion event (because both MCDI
447          * completions and BADASSERT events are sent to the same evq). So
448          * complete this MCDI op.
449          *
450          * This function might run in parallel with efx_mcdi_request_poll()
451          * for poll completed mcdi requests, and also with
452          * efx_mcdi_request_start() for post-watchdog completions.
453          */
454         EFSYS_LOCK(enp->en_eslp, state);
455         emrp = emip->emi_pending_req;
456         ev_cpl = emip->emi_ev_cpl;
457         if (emrp != NULL && emip->emi_ev_cpl) {
```



```
458                 emip->emi_pending_req = NULL;

460                 emrp->emr_out_length_used = 0;
461                 emrp->emr_rc = rc;
462                 ++emip->emi_aborted;
463         }

465         /*
466          * Since we're running in parallel with a request, consume the
467          * status word before dropping the lock.
468          */
469         if (rc == EIO || rc == EINTR) {
470                 EFSYS_SPIN(MCDI_STATUS_SLEEP_US);
471                 (void) efx_mcdi_poll_reboot(enp);
472         }

474         EFSYS_UNLOCK(enp->en_eslp, state);

476         efx_mcdi_raise_exception(enp, emrp, rc);

478         if (emrp != NULL && ev_cpl)
479                 emtp->emt_ev_cpl(emtp->emt_context);
480 }

482         __checkReturn           int
483 efx_mcdi_version(
484         __in                    efx_nic_t *enp,
485         __out_ecount_opt(4)     uint16_t versionp[4],
486         __out_opt               uint32_t *buildp,
487         __out_opt               efx_mcdi_boot_t *statusp)
488 {
489         uint8_t outbuf[MAX(MC_CMD_GET_VERSION_OUT_LEN,
490                            MC_CMD_GET_BOOT_STATUS_OUT_LEN)];
491         efx_mcdi_req_t req;
492         efx_word_t *ver_words;
493         uint16_t version[4];
494         uint32_t build;
495         efx_mcdi_boot_t status;
496         int rc;

498         EFSYS_ASSERT3U(enp->en_family, ==, EFX_FAMILY_SIENA);
499         EFSYS_ASSERT3U(enp->en_features, &, EFX_FEATURE_MCDI);

501         EFX_STATIC_ASSERT(MC_CMD_GET_VERSION_IN_LEN == 0);
502         req.emr_cmd = MC_CMD_GET_VERSION;
503         req.emr_in_buf = NULL;
504         req.emr_in_length = 0;
505         req.emr_out_buf = outbuf;
506         req.emr_out_length = MC_CMD_GET_VERSION_OUT_LEN;

508         efx_mcdi_execute(enp, &req);

510         if (req.emr_rc != 0) {
511                 rc = req.emr_rc;
512                 goto fail1;
513         }

515         /* bootrom support */
516         if (req.emr_out_length_used == MC_CMD_GET_VERSION_V0_OUT_LEN) {
517                 version[0] = version[1] = version[2] = version[3] = 0;
518                 build = MCDI_OUT_DWORD(req, GET_VERSION_OUT_FIRMWARE);

520                 goto version;
521         }

523         if (req.emr_out_length_used < MC_CMD_GET_VERSION_OUT_LEN) {
```

```
524                 rc = EMSGSIZE;
525                 goto fail2;
526         }

528         ver_words = MCDI_OUT2(req, efx_word_t, GET_VERSION_OUT_VERSION);
529         version[0] = EFX_WORD_FIELD(ver_words[0], EFX_WORD_0);
530         version[1] = EFX_WORD_FIELD(ver_words[1], EFX_WORD_0);
531         version[2] = EFX_WORD_FIELD(ver_words[2], EFX_WORD_0);
532         version[3] = EFX_WORD_FIELD(ver_words[3], EFX_WORD_0);
533         build = MCDI_OUT_DWORD(req, GET_VERSION_OUT_FIRMWARE);

535 version:
536         /* The bootrom doesn't understand BOOT_STATUS */
537         if (build == MC_CMD_GET_VERSION_OUT_FIRMWARE_BOOTROM) {
538                 status = EFX_MCDI_BOOT_ROM;
539                 goto out;
540         }

542         req.emr_cmd = MC_CMD_GET_BOOT_STATUS;
543         EFX_STATIC_ASSERT(MC_CMD_GET_BOOT_STATUS_IN_LEN == 0);
544         req.emr_in_buf = NULL;
545         req.emr_in_length = 0;
546         req.emr_out_buf = outbuf;
547         req.emr_out_length = MC_CMD_GET_BOOT_STATUS_OUT_LEN;

549         efx_mcdi_execute(enp, &req);

551         if (req.emr_rc != 0) {
552                 rc = req.emr_rc;
553                 goto fail3;
554         }

556         if (req.emr_out_length_used < MC_CMD_GET_BOOT_STATUS_OUT_LEN) {
557                 rc = EMSGSIZE;
558                 goto fail4;
559         }

561         if (MCDI_OUT_DWORD_FIELD(req, GET_BOOT_STATUS_OUT_FLAGS,
562             GET_BOOT_STATUS_OUT_FLAGS_PRIMARY))
563                 status = EFX_MCDI_BOOT_PRIMARY;
564         else
565                 status = EFX_MCDI_BOOT_SECONDARY;

567 out:
568         if (versionp != NULL)
569                 memcpy(versionp, version, sizeof (version));
570         if (buildp != NULL)
571                 *buildp = build;
572         if (statusp != NULL)
573                 *statusp = status;

575         return (0);

577 fail4:
578         EFSYS_PROBE(fail4);
579 fail3:
580         EFSYS_PROBE(fail3);
581 fail2:
582         EFSYS_PROBE(fail2);
583 fail1:
584         EFSYS_PROBE1(fail1, int, rc);

586         return (rc);
587 }

589         __checkReturn   int
```



```
590 efx_mcdi_init(
591         __in            efx_nic_t *enp,
592         __in            const efx_mcdi_transport_t *mtp)
593 {
594         efx_mcdi_iface_t *emip = &(enp->en_u.siena.enu_mip);
595         efx_oword_t oword;
596         unsigned int portnum;
597         int rc;

599         EFSYS_ASSERT3U(enp->en_mod_flags, ==, 0);
600         enp->en_mod_flags |= EFX_MOD_MCDI;

602         if (enp->en_family == EFX_FAMILY_FALCON)
603                 return (0);

605         emip->emi_mtp = mtp;

607         /* Determine the port number to use for MCDI */
608         EFX_BAR_READO(enp, FR_AZ_CS_DEBUG_REG, &oword);
609         portnum = EFX_OWORD_FIELD(oword, FRF_CZ_CS_PORT_NUM);

611         if (portnum == 0) {
612                 /* Presumably booted from ROM; only MCDI port 1 will work */
613                 emip->emi_port = 1;
614         } else if (portnum <= 2) {
615                 emip->emi_port = portnum;
616         } else {
617                 rc = EINVAL;
618                 goto fail1;
619         }

621         /*
622          * Wipe the atomic reboot status so subsequent MCDI requests succeed.
623          * BOOT_STATUS is preserved so eno_nic_probe() can boot out of the
624          * assertion handler.
625          */
626         (void) efx_mcdi_poll_reboot(enp);

628         return (0);

630 fail1:
631         EFSYS_PROBE1(fail1, int, rc);

633         enp->en_mod_flags &= ~EFX_MOD_MCDI;

635         return (rc);
636 }


639         __checkReturn   int
640 efx_mcdi_reboot(
641         __in            efx_nic_t *enp)
642 {
643         uint8_t payload[MC_CMD_REBOOT_IN_LEN];
644         efx_mcdi_req_t req;
645         int rc;

647         /*
648          * We could require the caller to have caused en_mod_flags=0 to
649          * call this function. This doesn't help the other port though,
650          * who's about to get the MC ripped out from underneath them.
651          * Since they have to cope with the subsequent fallout of MCDI
652          * failures, we should as well.
653          */
654         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
```

```
656         req.emr_cmd = MC_CMD_REBOOT;
657         req.emr_in_buf = payload;
658         req.emr_in_length = MC_CMD_REBOOT_IN_LEN;
659         req.emr_out_buf = NULL;
660         req.emr_out_length = 0;

662         MCDI_IN_SET_DWORD(req, REBOOT_IN_FLAGS, 0);

664         efx_mcdi_execute(enp, &req);

666         /* Invert EIO */
667         if (req.emr_rc != EIO) {
668                 rc = EIO;
669                 goto fail1;
670         }

672         return (0);

674 fail1:
675         EFSYS_PROBE1(fail1, int, rc);

677         return (rc);
678 }

680         __checkReturn   boolean_t
681 efx_mcdi_request_abort(
682         __in            efx_nic_t *enp)
683 {
684         efx_mcdi_iface_t *emip = &(enp->en_u.siena.enu_mip);
685         efx_mcdi_req_t *emrp;
686         boolean_t aborted;
687         int state;

689         EFSYS_ASSERT3U(enp->en_family, ==, EFX_FAMILY_SIENA);
690         EFSYS_ASSERT3U(enp->en_features, &, EFX_FEATURE_MCDI);

692         /*
693          * efx_mcdi_ev_* may have already completed this event, and be
694          * spinning/blocked on the upper layer lock. So it *is* legitimate
695          * to for emi_pending_req to be NULL. If there is a pending event
696          * completed request, then provide a "credit" to allow
697          * efx_mcdi_ev_cpl() to accept a single spurious completion.
698          */
699         EFSYS_LOCK(enp->en_eslp, state);
700         emrp = emip->emi_pending_req;
701         aborted = (emrp != NULL);
702         if (aborted) {
703                 emip->emi_pending_req = NULL;

705                 /* Error the request */
706                 emrp->emr_out_length_used = 0;
707                 emrp->emr_rc = ETIMEDOUT;

709                 /* Provide a credit for seqno/emr_pending_req mismatches */
710                 if (emip->emi_ev_cpl)
711                         ++emip->emi_aborted;

713                 /*
714                  * The upper layer has called us, so we don't
715                  * need to complete the request.
716                  */
717         }
718         EFSYS_UNLOCK(enp->en_eslp, state);

720         return (aborted);
721 }
```



```
723                         void
724 efx_mcdi_fini(
725         __in            efx_nic_t *enp)
726 {
727         efx_mcdi_iface_t *emip = &(enp->en_u.siena.enu_mip);

729         EFSYS_ASSERT3U(enp->en_mod_flags, ==, EFX_MOD_MCDI);
730         enp->en_mod_flags &= ~EFX_MOD_MCDI;

732         if (~(enp->en_features) & EFX_FEATURE_MCDI)
733                 return;

735         emip->emi_mtp = NULL;
736         emip->emi_port = 0;
737         emip->emi_aborted = 0;
738 }

740 #endif  /* EFSYS_OPT_MCDI */
741 #endif /* ! codereview */
```

```
**********************************************************
    8216 Thu Aug 22 18:59:22 2013
new/usr/src/uts/common/io/sfxge/efx_mcdi.h
Merged sfxge driver
**********************************************************
```

```
/*-
 * Copyright 2009 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#ifndef _SYS_EFX_MCDI_H
#define _SYS_EFX_MCDI_H

#include "efx.h"
#include "efx_regs.h"
#include "efx_regs_mcdi.h"

#ifdef  __cplusplus
extern "C" {
#endif

/* Number of retries attempted for init code */
#define EFX_MCDI_REQ_RETRY_INIT 2

struct efx_mcdi_req_s {
        /* Inputs: Command #, input buffer and length */
        unsigned int    emr_cmd;
        uint8_t         *emr_in_buf;
        size_t          emr_in_length;
        /* Outputs: retcode, buffer, length, and length used*/
        int             emr_rc;
        uint8_t         *emr_out_buf;
        size_t          emr_out_length;
        size_t          emr_out_length_used;
};

typedef struct efx_mcdi_iface_s {
        const efx_mcdi_transport_t *emi_mtp;
        unsigned int            emi_port;
        unsigned int            emi_seq;
        efx_mcdi_req_t          *emi_pending_req;
        boolean_t               emi_ev_cpl;
        int                     emi_aborted;
        uint32_t                emi_poll_cnt;
} efx_mcdi_iface_t;
```



```
extern                  void
efx_mcdi_execute(
        __in            efx_nic_t *enp,
        __in            efx_mcdi_req_t *emrp);

extern                  void
efx_mcdi_ev_cpl(
        __in            efx_nic_t *enp,
        __in            unsigned int seq,
        __in            unsigned int outlen,
        __in            int errcode);

extern                  void
efx_mcdi_ev_death(
        __in            efx_nic_t *enp,
        __in            int rc);

typedef enum efx_mcdi_boot_e {
        EFX_MCDI_BOOT_PRIMARY,
        EFX_MCDI_BOOT_SECONDARY,
        EFX_MCDI_BOOT_ROM,
} efx_mcdi_boot_t;

extern  __checkReturn           int
efx_mcdi_version(
        __in                    efx_nic_t *enp,
        __out_ecount_opt(4)     uint16_t versionp[4],
        __out_opt               uint32_t *buildp,
        __out_opt               efx_mcdi_boot_t *statusp);

#define MCDI_IN(_emr, _type, _ofst)                                     \
        ((_type *)((_emr).emr_in_buf + (_ofst)))

#define MCDI_IN2(_emr, _type, _ofst)                                    \
        MCDI_IN(_emr, _type, MC_CMD_ ## _ofst ## _OFST)

#define MCDI_IN_SET_BYTE(_emr, _ofst, _value)                           \
        EFX_POPULATE_BYTE_1(*MCDI_IN2(_emr, efx_byte_t, _ofst),         \
                EFX_BYTE_0, _value)

#define MCDI_IN_SET_DWORD(_emr, _ofst, _value)                          \
        EFX_POPULATE_DWORD_1(*MCDI_IN2(_emr, efx_dword_t, _ofst),       \
                EFX_DWORD_0, _value)

#define MCDI_IN_POPULATE_DWORD_1(_emr, _ofst, _field1, _value1)         \
        EFX_POPULATE_DWORD_1(*MCDI_IN2(_emr, efx_dword_t, _ofst),       \
                MC_CMD_ ## _field1, _value1)

#define MCDI_IN_POPULATE_DWORD_2(_emr, _ofst, _field1, _value1,         \
                _field2, _value2)                                       \
        EFX_POPULATE_DWORD_2(*MCDI_IN2(_emr, efx_dword_t, _ofst),       \
                MC_CMD_ ## _field1, _value1,                            \
                MC_CMD_ ## _field2, _value2)

#define MCDI_IN_POPULATE_DWORD_3(_emr, _ofst, _field1, _value1,         \
                _field2, _value2, _field3, _value3)                     \
        EFX_POPULATE_DWORD_3(*MCDI_IN2(_emr, efx_dword_t, _ofst),       \
                MC_CMD_ ## _field1, _value1,                            \
                MC_CMD_ ## _field2, _value2,                            \
                MC_CMD_ ## _field3, _value3)

#define MCDI_IN_POPULATE_DWORD_4(_emr, _ofst, _field1, _value1,         \
                _field2, _value2, _field3, _value3, _field4, _value4)   \
        EFX_POPULATE_DWORD_4(*MCDI_IN2(_emr, efx_dword_t, _ofst),       \
                MC_CMD_ ## _field1, _value1,                            \
                MC_CMD_ ## _field2, _value2,                            \
```

```
128                 MC_CMD_ ## _field3, _value3,                            \
129                 MC_CMD_ ## _field4, _value4)
130
131 #define MCDI_IN_POPULATE_DWORD_5(_emr, _ofst, _field1, _value1,         \
132                 _field2, _value2, _field3, _value3, _field4, _value4,   \
133                 _field5, _value5)                                       \
134         EFX_POPULATE_DWORD_5(*MCDI_IN2(_emr, efx_dword_t, _ofst),       \
135                 MC_CMD_ ## _field1, _value1,                            \
136                 MC_CMD_ ## _field2, _value2,                            \
137                 MC_CMD_ ## _field3, _value3,                            \
138                 MC_CMD_ ## _field4, _value4,                            \
139                 MC_CMD_ ## _field5, _value5)
140
141 #define MCDI_IN_POPULATE_DWORD_6(_emr, _ofst, _field1, _value1,         \
142                 _field2, _value2, _field3, _value3, _field4, _value4,   \
143                 _field5, _value5, _field6, _value6)                     \
144         EFX_POPULATE_DWORD_6(*MCDI_IN2(_emr, efx_dword_t, _ofst),       \
145                 MC_CMD_ ## _field1, _value1,                            \
146                 MC_CMD_ ## _field2, _value2,                            \
147                 MC_CMD_ ## _field3, _value3,                            \
148                 MC_CMD_ ## _field4, _value4,                            \
149                 MC_CMD_ ## _field5, _value5,                            \
150                 MC_CMD_ ## _field6, _value6)
151
152 #define MCDI_IN_POPULATE_DWORD_7(_emr, _ofst, _field1, _value1,         \
153                 _field2, _value2, _field3, _value3, _field4, _value4,   \
154                 _field5, _value5, _field6, _value6, _field7, _value7)   \
155         EFX_POPULATE_DWORD_7(MCDI_IN2(_emr, efx_dword_t, _ofst),        \
156                 MC_CMD_ ## _field1, _value1,                            \
157                 MC_CMD_ ## _field2, _value2,                            \
158                 MC_CMD_ ## _field3, _value3,                            \
159                 MC_CMD_ ## _field4, _value4,                            \
160                 MC_CMD_ ## _field5, _value5,                            \
161                 MC_CMD_ ## _field6, _value6,                            \
162                 MC_CMD_ ## _field7, _value7)
163
164 #define MCDI_IN_POPULATE_DWORD_8(_emr, _ofst, _field1, _value1,         \
165                 _field2, _value2, _field3, _value3, _field4, _value4,   \
166                 _field5, _value5, _field6, _value6, _field7, _value7,   \
167                 _field8, _value8)                                       \
168         EFX_POPULATE_DWORD_8(*MCDI_IN2(_emr, efx_dword_t, _ofst),       \
169                 MC_CMD_ ## _field1, _value1,                            \
170                 MC_CMD_ ## _field2, _value2,                            \
171                 MC_CMD_ ## _field3, _value3,                            \
172                 MC_CMD_ ## _field4, _value4,                            \
173                 MC_CMD_ ## _field5, _value5,                            \
174                 MC_CMD_ ## _field6, _value6,                            \
175                 MC_CMD_ ## _field7, _value7,                            \
176                 MC_CMD_ ## _field8, _value8)
177
178 #define MCDI_IN_POPULATE_DWORD_9(_emr, _ofst, _field1, _value1,         \
179                 _field2, _value2, _field3, _value3, _field4, _value4,   \
180                 _field5, _value5, _field6, _value6, _field7, _value7,   \
181                 _field8, _value8, _field9, _value9)                     \
182         EFX_POPULATE_DWORD_9(*MCDI_IN2(_emr, efx_dword_t, _ofst),       \
183                 MC_CMD_ ## _field1, _value1,                            \
184                 MC_CMD_ ## _field2, _value2,                            \
185                 MC_CMD_ ## _field3, _value3,                            \
186                 MC_CMD_ ## _field4, _value4,                            \
187                 MC_CMD_ ## _field5, _value5,                            \
188                 MC_CMD_ ## _field6, _value6,                            \
189                 MC_CMD_ ## _field7, _value7,                            \
190                 MC_CMD_ ## _field8, _value8,                            \
191                 MC_CMD_ ## _field9, _value9)
192
193 #define MCDI_IN_POPULATE_DWORD_10(_emr, _ofst, _field1, _value1,        \
```



```
194                 _field2, _value2, _field3, _value3, _field4, _value4,   \
195                 _field5, _value5, _field6, _value6, _field7, _value7,   \
196                 _field8, _value8, _field9, _value9, _field10, _value10) \
197         EFX_POPULATE_DWORD_10(*MCDI_IN2(_emr, efx_dword_t, _ofst),      \
198                 MC_CMD_ ## _field1, _value1,                            \
199                 MC_CMD_ ## _field2, _value2,                            \
200                 MC_CMD_ ## _field3, _value3,                            \
201                 MC_CMD_ ## _field4, _value4,                            \
202                 MC_CMD_ ## _field5, _value5,                            \
203                 MC_CMD_ ## _field6, _value6,                            \
204                 MC_CMD_ ## _field7, _value7,                            \
205                 MC_CMD_ ## _field8, _value8,                            \
206                 MC_CMD_ ## _field9, _value9,                            \
207                 MC_CMD_ ## _field10, _value10)
208
209 #define MCDI_OUT(_emr, _type, _ofst)                                    \
210         ((_type *)((_emr).emr_out_buf + (_ofst)))
211
212 #define MCDI_OUT2(_emr, _type, _ofst)                                   \
213         MCDI_OUT(_emr, _type, MC_CMD_ ## _ofst ## _OFST)
214
215 #define MCDI_OUT_BYTE(_emr, _ofst)                                      \
216         EFX_BYTE_FIELD(*MCDI_OUT2(_emr, efx_byte_t, _ofst),             \
217                        EFX_BYTE_0)
218
219 #define MCDI_OUT_WORD(_emr, _ofst)                                      \
220         EFX_WORD_FIELD(*MCDI_OUT2(_emr, efx_word_t, _ofst),             \
221                        EFX_WORD_0)
222
223 #define MCDI_OUT_DWORD(_emr, _ofst)                                     \
224         EFX_DWORD_FIELD(*MCDI_OUT2(_emr, efx_dword_t, _ofst),           \
225                         EFX_DWORD_0)
226
227 #define MCDI_OUT_DWORD_FIELD(_emr, _ofst, _field)                       \
228         EFX_DWORD_FIELD(*MCDI_OUT2(_emr, efx_dword_t, _ofst),           \
229                         MC_CMD_ ## _field)
230
231 #define MCDI_EV_FIELD(_eqp, _field)                                     \
232         EFX_QWORD_FIELD(*_eqp, MCDI_EVENT_ ## _field)
233
234 #define MCDI_CMD_DWORD_FIELD(_edp, _field)                              \
235         EFX_DWORD_FIELD(*_edp, MC_CMD_ ## _field)
236
237 #ifdef  __cplusplus
238 }
239 #endif
240
241 #endif  /* _SYS_EFX_MCDI_H */
242 #endif /* ! codereview */
```

```
**********************************************************
    6142 Thu Aug 22 18:59:22 2013
new/usr/src/uts/common/io/sfxge/efx_mon.c
Merged sfxge driver
**********************************************************
```

```
/*
 * Copyright 2007-2013 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#include "efsys.h"
#include "efx.h"
#include "efx_types.h"
#include "efx_regs.h"
#include "efx_impl.h"

#if EFSYS_OPT_MON_NULL
#include "nullmon.h"
#endif

#if EFSYS_OPT_MON_LM87
#include "lm87.h"
#endif

#if EFSYS_OPT_MON_MAX6647
#include "max6647.h"
#endif

#if EFSYS_OPT_NAMES

static const char       __cs * __cs __efx_mon_name[] = {
        "",
        "nullmon",
        "lm87",
        "max6647",
        "sfx90x0"
};

                const char __cs *
efx_mon_name(
        __in    efx_nic_t *enp)
{
        efx_nic_cfg_t *encp = &(enp->en_nic_cfg);

        EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
```



```
        EFSYS_ASSERT(encp->enc_mon_type != EFX_MON_INVALID);
        EFSYS_ASSERT3U(encp->enc_mon_type, <, EFX_MON_NTYPES);
        return (__efx_mon_name[encp->enc_mon_type]);
}

#endif  /* EFSYS_OPT_NAMES */

#if EFSYS_OPT_MON_NULL
static efx_mon_ops_t    __cs __efx_mon_null_ops = {
        nullmon_reset,                  /* emo_reset */
        nullmon_reconfigure,            /* emo_reconfigure */
#if EFSYS_OPT_MON_STATS
        nullmon_stats_update            /* emo_stat_update */
#endif  /* EFSYS_OPT_MON_STATS */
};
#endif

#if EFSYS_OPT_MON_LM87
static efx_mon_ops_t    __cs __efx_mon_lm87_ops = {
        lm87_reset,                     /* emo_reset */
        lm87_reconfigure,               /* emo_reconfigure */
#if EFSYS_OPT_MON_STATS
        lm87_stats_update               /* emo_stat_update */
#endif  /* EFSYS_OPT_MON_STATS */
};
#endif

#if EFSYS_OPT_MON_MAX6647
static efx_mon_ops_t    __cs __efx_mon_max6647_ops = {
        max6647_reset,                  /* emo_reset */
        max6647_reconfigure,            /* emo_reconfigure */
#if EFSYS_OPT_MON_STATS
        max6647_stats_update            /* emo_stat_update */
#endif  /* EFSYS_OPT_MON_STATS */
};
#endif

#if EFSYS_OPT_MON_SIENA
static efx_mon_ops_t    __cs __efx_mon_siena_ops = {
        siena_mon_reset,                /* emo_reset */
        siena_mon_reconfigure,          /* emo_reconfigure */
#if EFSYS_OPT_MON_STATS
        siena_mon_stats_update          /* emo_stat_update */
#endif  /* EFSYS_OPT_MON_STATS */
};
#endif


static efx_mon_ops_t    __cs * __cs __efx_mon_ops[] = {
        NULL,
#if EFSYS_OPT_MON_NULL
        &__efx_mon_null_ops,
#else
        NULL,
#endif
#if EFSYS_OPT_MON_LM87
        &__efx_mon_lm87_ops,
#else
        NULL,
#endif
#if EFSYS_OPT_MON_MAX6647
        &__efx_mon_max6647_ops,
#else
        NULL,
#endif
#if EFSYS_OPT_MON_SIENA
```

```
128         &__efx_mon_siena_ops
129 #else
130         NULL
131 #endif
132 };

134         __checkReturn   int
135 efx_mon_init(
136         __in            efx_nic_t *enp)
137 {
138         efx_nic_cfg_t *encp = &(enp->en_nic_cfg);
139         efx_mon_t *emp = &(enp->en_mon);
140         efx_mon_ops_t *emop;
141         int rc;

143         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
144         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_PROBE);

146         if (enp->en_mod_flags & EFX_MOD_MON) {
147                 rc = EINVAL;
148                 goto fail1;
149         }

151         enp->en_mod_flags |= EFX_MOD_MON;

153         emp->em_type = encp->enc_mon_type;

155         EFSYS_ASSERT(encp->enc_mon_type != EFX_MON_INVALID);
156         EFSYS_ASSERT3U(emp->em_type, <, EFX_MON_NTYPES);
157         if ((emop = (efx_mon_ops_t *)__efx_mon_ops[emp->em_type]) == NULL) {
158                 rc = ENOTSUP;
159                 goto fail2;
160         }

162         if ((rc = emop->emo_reset(enp)) != 0)
163                 goto fail3;

165         if ((rc = emop->emo_reconfigure(enp)) != 0)
166                 goto fail4;

168         emp->em_emop = emop;
169         return (0);

171 fail4:
172         EFSYS_PROBE(fail5);

174         (void) emop->emo_reset(enp);

176 fail3:
177         EFSYS_PROBE(fail4);
178 fail2:
179         EFSYS_PROBE(fail3);

181         emp->em_type = EFX_MON_INVALID;

183         enp->en_mod_flags &= ~EFX_MOD_MON;

185 fail1:
186         EFSYS_PROBE1(fail1, int, rc);

188         return (rc);
189 }

191 #if EFSYS_OPT_MON_STATS

193 #if EFSYS_OPT_NAMES
```



```
195 /* START MKCONFIG GENERATED MonitorStatNamesBlock cc98f339ca74c6b8 */
196 static const char       __cs * __cs __mon_stat_name[] = {
197         "value_2_5v",
198         "value_vccp1",
199         "value_vcc",
200         "value_5v",
201         "value_12v",
202         "value_vccp2",
203         "value_ext_temp",
204         "value_int_temp",
205         "value_ain1",
206         "value_ain2",
207         "controller_cooling",
208         "ext_cooling",
209         "1v",
210         "1_2v",
211         "1_8v",
212         "3_3v",
213         "1_2va",
214         "vref",
215 };

217 /* END MKCONFIG GENERATED MonitorStatNamesBlock */

219 extern                                  const char __cs *
220 efx_mon_stat_name(
221         __in                            efx_nic_t *enp,
222         __in                            efx_mon_stat_t id)
223 {
224         _NOTE(ARGUNUSED(enp))
225         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);

227         EFSYS_ASSERT3U(id, <, EFX_MON_NSTATS);
228         return (__mon_stat_name[id]);
229 }

231 #endif  /* EFSYS_OPT_NAMES */

233         __checkReturn                   int
234 efx_mon_stats_update(
235         __in                            efx_nic_t *enp,
236         __in                            efsys_mem_t *esmp,
237         __out_ecount(EFX_MON_NSTATS)    efx_mon_stat_value_t *values)
238 {
239         efx_mon_t *emp = &(enp->en_mon);
240         efx_mon_ops_t *emop = emp->em_emop;

242         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
243         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_MON);

245         return (emop->emo_stats_update(enp, esmp, values));
246 }

248 #endif  /* EFSYS_OPT_MON_STATS */

250                 void
251 efx_mon_fini(
252         __in    efx_nic_t *enp)
253 {
254         efx_mon_t *emp = &(enp->en_mon);
255         efx_mon_ops_t *emop = emp->em_emop;
256         int rc;

258         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
259         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_PROBE);
```

```
260         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_MON);

262         emp->em_emop = NULL;

264         rc = emop->emo_reset(enp);
265         if (rc != 0)
266                 EFSYS_PROBE1(fail1, int, rc);

268         emp->em_type = EFX_MON_INVALID;

270         enp->en_mod_flags &= ~EFX_MOD_MON;
271 }
272 #endif /* ! codereview */
```

```
**********************************************************
   15352 Thu Aug 22 18:59:22 2013
new/usr/src/uts/common/io/sfxge/efx_nic.c
Merged sfxge driver
**********************************************************
```

```
/*
 * Copyright 2007-2013 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#include "efsys.h"
#include "efx.h"
#include "efx_types.h"
#include "efx_regs.h"
#include "efx_impl.h"

        __checkReturn   int
efx_family(
        __in            uint16_t venid,
        __in            uint16_t devid,
        __out           efx_family_t *efp)
{
#if EFSYS_OPT_FALCON
        if (venid == EFX_PCI_VENID_SFC && devid == EFX_PCI_DEVID_FALCON) {
                *efp = EFX_FAMILY_FALCON;
                return (0);
        }
#endif
#if EFSYS_OPT_SIENA
        if (venid == EFX_PCI_VENID_SFC && devid == EFX_PCI_DEVID_BETHPAGE) {
                *efp = EFX_FAMILY_SIENA;
                return (0);
        }
        if (venid == EFX_PCI_VENID_SFC && devid == EFX_PCI_DEVID_SIENA) {
                *efp = EFX_FAMILY_SIENA;
                return (0);
        }
        if (venid == EFX_PCI_VENID_SFC &&
            devid == EFX_PCI_DEVID_SIENA_F1_UNINIT) {
                *efp = EFX_FAMILY_SIENA;
                return (0);
        }
#endif
        return (ENOTSUP);
}
```



```
/*
 * To support clients which aren't provided with any PCI context infer
 * the hardware family by inspecting the hardware. Obviously the caller
 * must be damn sure they're really talking to a supported device.
 */
        __checkReturn   int
efx_infer_family(
        __in            efsys_bar_t *esbp,
        __out           efx_family_t *efp)
{
        efx_family_t family;
        efx_oword_t oword;
        unsigned int portnum;
        int rc;

        EFSYS_BAR_READO(esbp, FR_AZ_CS_DEBUG_REG_OFST, &oword, B_TRUE);
        portnum = EFX_OWORD_FIELD(oword, FRF_CZ_CS_PORT_NUM);
        switch (portnum) {
#if EFSYS_OPT_FALCON
        case 0:
                family = EFX_FAMILY_FALCON;
                break;
#endif
#if EFSYS_OPT_SIENA
        case 1:
        case 2:
                family = EFX_FAMILY_SIENA;
                break;
#endif
        default:
                rc = ENOTSUP;
                goto fail1;
        }

        if (efp != NULL)
                *efp = family;
        return (0);

fail1:
        EFSYS_PROBE1(fail1, int, rc);

        return (rc);
}

/*
 * The built-in default value device id for port 1 of Siena is 0x0810.
 * manftest needs to be able to cope with that.
 */

#define EFX_BIU_MAGIC0  0x01234567
#define EFX_BIU_MAGIC1  0xfedcba98

static  __checkReturn   int
efx_nic_biu_test(
        __in            efx_nic_t *enp)
{
        efx_oword_t oword;
        int rc;

        /*
         * Write magic values to scratch registers 0 and 1, then
         * verify that the values were written correctly.  Interleave
         * the accesses to ensure that the BIU is not just reading
         * back the cached value that was last written.
         */
        EFX_POPULATE_OWORD_1(oword, FRF_AZ_DRIVER_DW0, EFX_BIU_MAGIC0);
```

```
128         EFX_BAR_TBL_WRITEO(enp, FR_AZ_DRIVER_REG, 0, &oword);

130         EFX_POPULATE_OWORD_1(oword, FRF_AZ_DRIVER_DW0, EFX_BIU_MAGIC1);
131         EFX_BAR_TBL_WRITEO(enp, FR_AZ_DRIVER_REG, 1, &oword);

133         EFX_BAR_TBL_READO(enp, FR_AZ_DRIVER_REG, 0, &oword);
134         if (EFX_OWORD_FIELD(oword, FRF_AZ_DRIVER_DW0) != EFX_BIU_MAGIC0) {
135                 rc = EIO;
136                 goto fail1;
137         }

139         EFX_BAR_TBL_READO(enp, FR_AZ_DRIVER_REG, 1, &oword);
140         if (EFX_OWORD_FIELD(oword, FRF_AZ_DRIVER_DW0) != EFX_BIU_MAGIC1) {
141                 rc = EIO;
142                 goto fail2;
143         }

145         /*
146          * Perform the same test, with the values swapped.  This
147          * ensures that subsequent tests don't start with the correct
148          * values already written into the scratch registers.
149          */
150         EFX_POPULATE_OWORD_1(oword, FRF_AZ_DRIVER_DW0, EFX_BIU_MAGIC1);
151         EFX_BAR_TBL_WRITEO(enp, FR_AZ_DRIVER_REG, 0, &oword);

153         EFX_POPULATE_OWORD_1(oword, FRF_AZ_DRIVER_DW0, EFX_BIU_MAGIC0);
154         EFX_BAR_TBL_WRITEO(enp, FR_AZ_DRIVER_REG, 1, &oword);

156         EFX_BAR_TBL_READO(enp, FR_AZ_DRIVER_REG, 0, &oword);
157         if (EFX_OWORD_FIELD(oword, FRF_AZ_DRIVER_DW0) != EFX_BIU_MAGIC1) {
158                 rc = EIO;
159                 goto fail3;
160         }

162         EFX_BAR_TBL_READO(enp, FR_AZ_DRIVER_REG, 1, &oword);
163         if (EFX_OWORD_FIELD(oword, FRF_AZ_DRIVER_DW0) != EFX_BIU_MAGIC0) {
164                 rc = EIO;
165                 goto fail4;
166         }

168         return (0);

170 fail4:
171         EFSYS_PROBE(fail4);
172 fail3:
173         EFSYS_PROBE(fail3);
174 fail2:
175         EFSYS_PROBE(fail2);
176 fail1:
177         EFSYS_PROBE1(fail1, int, rc);

179         return (rc);
180 }

182 #if EFSYS_OPT_FALCON

184 static efx_nic_ops_t    __cs __efx_nic_falcon_ops = {
185         falcon_nic_probe,               /* eno_probe */
186         falcon_nic_reset,               /* eno_reset */
187         falcon_nic_init,                /* eno_init */
188 #if EFSYS_OPT_DIAG
189         falcon_sram_test,               /* eno_sram_test */
190         falcon_nic_register_test,       /* eno_register_test */
191 #endif  /* EFSYS_OPT_DIAG */
192         falcon_nic_fini,                /* eno_fini */
193         falcon_nic_unprobe,             /* eno_unprobe */
```



```
194 };

196 #endif  /* EFSYS_OPT_FALCON */

198 #if EFSYS_OPT_SIENA

200 static efx_nic_ops_t    __cs __efx_nic_siena_ops = {
201         siena_nic_probe,                /* eno_probe */
202         siena_nic_reset,                /* eno_reset */
203         siena_nic_init,                 /* eno_init */
204 #if EFSYS_OPT_DIAG
205         siena_sram_test,                /* eno_sram_test */
206         siena_nic_register_test,        /* eno_register_test */
207 #endif  /* EFSYS_OPT_DIAG */
208         siena_nic_fini,                 /* eno_fini */
209         siena_nic_unprobe,              /* eno_unprobe */
210 };

212 #endif  /* EFSYS_OPT_SIENA */

214         __checkReturn   int
215 efx_nic_create(
216         __in            efx_family_t family,
217         __in            efsys_identifier_t *esip,
218         __in            efsys_bar_t *esbp,
219         __in            efsys_lock_t *eslp,
220         __deref_out     efx_nic_t **enpp)
221 {
222         efx_nic_t *enp;
223         int rc;

225         EFSYS_ASSERT3U(family, >, EFX_FAMILY_INVALID);
226         EFSYS_ASSERT3U(family, <, EFX_FAMILY_NTYPES);

228         /* Allocate a NIC object */
229         EFSYS_KMEM_ALLOC(esip, sizeof (efx_nic_t), enp);

231         if (enp == NULL) {
232                 rc = ENOMEM;
233                 goto fail1;
234         }

236         enp->en_magic = EFX_NIC_MAGIC;

238         switch (family) {
239 #if EFSYS_OPT_FALCON
240         case EFX_FAMILY_FALCON:
241                 enp->en_enop = (efx_nic_ops_t *)&__efx_nic_falcon_ops;
242                 enp->en_features = 0;
243                 break;
244 #endif  /* EFSYS_OPT_FALCON */

246 #if EFSYS_OPT_SIENA
247         case EFX_FAMILY_SIENA:
248                 enp->en_enop = (efx_nic_ops_t *)&__efx_nic_siena_ops;
249                 enp->en_features =
250                     EFX_FEATURE_IPV6 |
251                     EFX_FEATURE_LFSR_HASH_INSERT |
252                     EFX_FEATURE_LINK_EVENTS |
253                     EFX_FEATURE_PERIODIC_MAC_STATS |
254                     EFX_FEATURE_WOL |
255                     EFX_FEATURE_MCDI |
256                     EFX_FEATURE_LOOKAHEAD_SPLIT |
257                     EFX_FEATURE_MAC_HEADER_FILTERS;
258                 break;
259 #endif  /* EFSYS_OPT_SIENA */
```

```
261         default:
262                 rc = ENOTSUP;
263                 goto fail2;
264         }

266         enp->en_family = family;
267         enp->en_esip = esip;
268         enp->en_esbp = esbp;
269         enp->en_eslp = eslp;

271         *enpp = enp;

273         return (0);

275 fail2:
276         EFSYS_PROBE(fail3);

278         enp->en_magic = 0;

280         /* Free the NIC object */
281         EFSYS_KMEM_FREE(esip, sizeof (efx_nic_t), enp);

283 fail1:
284         EFSYS_PROBE1(fail1, int, rc);

286         return (rc);
287 }

289         __checkReturn   int
290 efx_nic_probe(
291         __in            efx_nic_t *enp)
292 {
293         efx_nic_ops_t *enop;
294         efx_oword_t oword;
295         int rc;

297         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
298 #if EFSYS_OPT_MCDI
299         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_MCDI);
300 #endif  /* EFSYS_OPT_MCDI */
301         EFSYS_ASSERT(!(enp->en_mod_flags & EFX_MOD_PROBE));

303         /* Test BIU */
304         if ((rc = efx_nic_biu_test(enp)) != 0)
305                 goto fail1;

307         /* Clear the region register */
308         EFX_POPULATE_OWORD_4(oword,
309             FRF_AZ_ADR_REGION0, 0,
310             FRF_AZ_ADR_REGION1, (1 << 16),
311             FRF_AZ_ADR_REGION2, (2 << 16),
312             FRF_AZ_ADR_REGION3, (3 << 16));
313         EFX_BAR_WRITEO(enp, FR_AZ_ADR_REGION_REG, &oword);

315         enop = enp->en_enop;
316         if ((rc = enop->eno_probe(enp)) != 0)
317                 goto fail2;

319         if ((rc = efx_phy_probe(enp)) != 0)
320                 goto fail3;

322         enp->en_mod_flags |= EFX_MOD_PROBE;

324         return (0);
```



```
326 fail3:
327         EFSYS_PROBE(fail3);

329         enop->eno_unprobe(enp);

331 fail2:
332         EFSYS_PROBE(fail2);
333 fail1:
334         EFSYS_PROBE1(fail1, int, rc);

336         return (rc);
337 }

339 #if EFSYS_OPT_PCIE_TUNE

341         __checkReturn   int
342 efx_nic_pcie_tune(
343         __in            efx_nic_t *enp,
344         unsigned int    nlanes)
345 {
346         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
347         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_PROBE);
348         EFSYS_ASSERT(!(enp->en_mod_flags & EFX_MOD_NIC));

350 #if EFSYS_OPT_FALCON
351         if (enp->en_family == EFX_FAMILY_FALCON)
352                 return (falcon_nic_pcie_tune(enp, nlanes));
353 #endif
354         return (ENOTSUP);
355 }

357         __checkReturn   int
358 efx_nic_pcie_extended_sync(
359         __in            efx_nic_t *enp)
360 {
361         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
362         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_PROBE);
363         EFSYS_ASSERT(!(enp->en_mod_flags & EFX_MOD_NIC));

365 #if EFSYS_OPT_SIENA
366         if (enp->en_family == EFX_FAMILY_SIENA)
367                 return (siena_nic_pcie_extended_sync(enp));
368 #endif

370         return (ENOTSUP);
371 }

373 #endif  /* EFSYS_OPT_PCIE_TUNE */

375         __checkReturn   int
376 efx_nic_init(
377         __in            efx_nic_t *enp)
378 {
379         efx_nic_ops_t *enop = enp->en_enop;
380         int rc;

382         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
383         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_PROBE);

385         if (enp->en_mod_flags & EFX_MOD_NIC) {
386                 rc = EINVAL;
387                 goto fail1;
388         }

390         if ((rc = enop->eno_init(enp)) != 0)
391                 goto fail2;
```

```
393         enp->en_mod_flags |= EFX_MOD_NIC;

395         return (0);

397 fail2:
398         EFSYS_PROBE(fail2);
399 fail1:
400         EFSYS_PROBE1(fail1, int, rc);

402         return (rc);
403 }

405                         void
406 efx_nic_fini(
407         __in            efx_nic_t *enp)
408 {
409         efx_nic_ops_t *enop = enp->en_enop;

411         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
412         EFSYS_ASSERT(enp->en_mod_flags & EFX_MOD_PROBE);
413         EFSYS_ASSERT(enp->en_mod_flags & EFX_MOD_NIC);
414         EFSYS_ASSERT(!(enp->en_mod_flags & EFX_MOD_INTR));
415         EFSYS_ASSERT(!(enp->en_mod_flags & EFX_MOD_EV));
416         EFSYS_ASSERT(!(enp->en_mod_flags & EFX_MOD_RX));
417         EFSYS_ASSERT(!(enp->en_mod_flags & EFX_MOD_TX));

419         enop->eno_fini(enp);

421         enp->en_mod_flags &= ~EFX_MOD_NIC;
422 }

424                         void
425 efx_nic_unprobe(
426         __in            efx_nic_t *enp)
427 {
428         efx_nic_ops_t *enop = enp->en_enop;

430         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
431 #if EFSYS_OPT_MCDI
432         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_MCDI);
433 #endif  /* EFSYS_OPT_MCDI */
434         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_PROBE);
435         EFSYS_ASSERT(!(enp->en_mod_flags & EFX_MOD_NIC));
436         EFSYS_ASSERT(!(enp->en_mod_flags & EFX_MOD_INTR));
437         EFSYS_ASSERT(!(enp->en_mod_flags & EFX_MOD_EV));
438         EFSYS_ASSERT(!(enp->en_mod_flags & EFX_MOD_RX));
439         EFSYS_ASSERT(!(enp->en_mod_flags & EFX_MOD_TX));

441         efx_phy_unprobe(enp);

443         enop->eno_unprobe(enp);

445         enp->en_mod_flags &= ~EFX_MOD_PROBE;
446 }

448                         void
449 efx_nic_destroy(
450         __in    efx_nic_t *enp)
451 {
452         efsys_identifier_t *esip = enp->en_esip;

454         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
455         EFSYS_ASSERT3U(enp->en_mod_flags, ==, 0);

457         enp->en_family = 0;
```



```
458         enp->en_esip = NULL;
459         enp->en_esbp = NULL;
460         enp->en_eslp = NULL;

462         enp->en_enop = NULL;

464         enp->en_magic = 0;

466         /* Free the NIC object */
467         EFSYS_KMEM_FREE(esip, sizeof (efx_nic_t), enp);
468 }

470         __checkReturn   int
471 efx_nic_reset(
472         __in            efx_nic_t *enp)
473 {
474         efx_nic_ops_t *enop = enp->en_enop;
475         unsigned int mod_flags;
476         int rc;

478         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
479         EFSYS_ASSERT(enp->en_mod_flags & EFX_MOD_PROBE);
480         /*
481          * All modules except the MCDI, PROBE, NVRAM, VPD, MON (which we
482          * do not reset here) must have been shut down or never initialized.
483          *
484          * A rule of thumb here is: If the controller or MC reboots, is *any*
485          * state lost. If it's lost and needs reapplying, then the module
486          * *must* not be initialised during the reset.
487          */
488         mod_flags = enp->en_mod_flags;
489         mod_flags &= ~(EFX_MOD_MCDI | EFX_MOD_PROBE | EFX_MOD_NVRAM |
490                        EFX_MOD_VPD | EFX_MOD_MON);
491         EFSYS_ASSERT3U(mod_flags, ==, 0);
492         if (mod_flags != 0) {
493                 rc = EINVAL;
494                 goto fail1;
495         }

497         if ((rc = enop->eno_reset(enp)) != 0)
498                 goto fail2;

500         enp->en_reset_flags |= EFX_RESET_MAC;

502         return (0);

504 fail2:
505         EFSYS_PROBE(fail2);
506 fail1:
507         EFSYS_PROBE1(fail1, int, rc);

509         return (rc);
510 }

512                         const efx_nic_cfg_t *
513 efx_nic_cfg_get(
514         __in            efx_nic_t *enp)
515 {
516         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);

518         return (&(enp->en_nic_cfg));
519 }

521 #if EFSYS_OPT_DIAG

523         __checkReturn   int
```

```
524 efx_nic_register_test(
525         __in            efx_nic_t *enp)
526 {
527         efx_nic_ops_t *enop = enp->en_enop;
528         int rc;
529
530         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
531         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_PROBE);
532         EFSYS_ASSERT(!(enp->en_mod_flags & EFX_MOD_NIC));
533
534         if ((rc = enop->eno_register_test(enp)) != 0)
535                 goto fail1;
536
537         return (0);
538
539 fail1:
540         EFSYS_PROBE1(fail1, int, rc);
541
542         return (rc);
543 }
544
545         __checkReturn   int
546 efx_nic_test_registers(
547         __in            efx_nic_t *enp,
548         __in            efx_register_set_t *rsp,
549         __in            size_t count)
550 {
551         unsigned int bit;
552         efx_oword_t original;
553         efx_oword_t reg;
554         efx_oword_t buf;
555         int rc;
556
557         while (count > 0) {
558                 /* This function is only suitable for registers */
559                 EFSYS_ASSERT(rsp->rows == 1);
560
561                 /* bit sweep on and off */
562                 EFSYS_BAR_READO(enp->en_esbp, rsp->address, &original,
563                                 B_TRUE);
564                 for (bit = 0; bit < 128; bit++) {
565                         /* Is this bit in the mask? */
566                         if (~(rsp->mask.eo_u32[bit >> 5]) & (1 << bit))
567                                 continue;
568
569                         /* Test this bit can be set in isolation */
570                         reg = original;
571                         EFX_AND_OWORD(reg, rsp->mask);
572                         EFX_SET_OWORD_BIT(reg, bit);
573
574                         EFSYS_BAR_WRITEO(enp->en_esbp, rsp->address, &reg,
575                                          B_TRUE);
576                         EFSYS_BAR_READO(enp->en_esbp, rsp->address, &buf,
577                                         B_TRUE);
578
579                         EFX_AND_OWORD(buf, rsp->mask);
580                         if (memcmp(&reg, &buf, sizeof (reg))) {
581                                 rc = EIO;
582                                 goto fail1;
583                         }
584
585                         /* Test this bit can be cleared in isolation */
586                         EFX_OR_OWORD(reg, rsp->mask);
587                         EFX_CLEAR_OWORD_BIT(reg, bit);
588
589                         EFSYS_BAR_WRITEO(enp->en_esbp, rsp->address, &reg,
```



```
590                                          B_TRUE);
591                         EFSYS_BAR_READO(enp->en_esbp, rsp->address, &buf,
592                                         B_TRUE);
593
594                         EFX_AND_OWORD(buf, rsp->mask);
595                         if (memcmp(&reg, &buf, sizeof (reg))) {
596                                 rc = EIO;
597                                 goto fail2;
598                         }
599                 }
600
601                 /* Restore the old value */
602                 EFSYS_BAR_WRITEO(enp->en_esbp, rsp->address, &original,
603                                  B_TRUE);
604
605                 --count;
606                 ++rsp;
607         }
608
609         return (0);
610
611 fail2:
612         EFSYS_PROBE(fail2);
613 fail1:
614         EFSYS_PROBE1(fail1, int, rc);
615
616         /* Restore the old value */
617         EFSYS_BAR_WRITEO(enp->en_esbp, rsp->address, &original, B_TRUE);
618
619         return (rc);
620 }
621
622         __checkReturn   int
623 efx_nic_test_tables(
624         __in            efx_nic_t *enp,
625         __in            efx_register_set_t *rsp,
626         __in            efx_pattern_type_t pattern,
627         __in            size_t count)
628 {
629         efx_sram_pattern_fn_t func;
630         unsigned int index;
631         unsigned int address;
632         efx_oword_t reg;
633         efx_oword_t buf;
634         int rc;
635
636         EFSYS_ASSERT(pattern < EFX_PATTERN_NTYPES);
637         func = __efx_sram_pattern_fns[pattern];
638
639         while (count > 0) {
640                 /* Write */
641                 address = rsp->address;
642                 for (index = 0; index < rsp->rows; ++index) {
643                         func(2 * index + 0, B_FALSE, &reg.eo_qword[0]);
644                         func(2 * index + 1, B_FALSE, &reg.eo_qword[1]);
645                         EFX_AND_OWORD(reg, rsp->mask);
646                         EFSYS_BAR_WRITEO(enp->en_esbp, address, &reg, B_TRUE);
647
648                         address += rsp->step;
649                 }
650
651                 /* Read */
652                 address = rsp->address;
653                 for (index = 0; index < rsp->rows; ++index) {
654                         func(2 * index + 0, B_FALSE, &reg.eo_qword[0]);
655                         func(2 * index + 1, B_FALSE, &reg.eo_qword[1]);
```

```
656                         EFX_AND_OWORD(reg, rsp->mask);
657                         EFSYS_BAR_READO(enp->en_esbp, address, &buf, B_TRUE);
658                         if (memcmp(&reg, &buf, sizeof (reg))) {
659                                 rc = EIO;
660                                 goto fail1;
661                         }

663                         address += rsp->step;
664                 }

666                 ++rsp;
667                 --count;
668         }

670         return (0);

672 fail1:
673         EFSYS_PROBE1(fail1, int, rc);

675         return (rc);
676 }

678 #endif  /* EFSYS_OPT_DIAG */
679 #endif /* ! codereview */
```

****
8833 Thu Aug 22 18:59:22 2013
new/usr/src/uts/common/io/sfxge/efx_nvram.c
Merged sfxge driver
****

```
/*
 * Copyright 2009 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#include "efsys.h"
#include "efx.h"
#include "efx_types.h"
#include "efx_regs.h"
#include "efx_impl.h"

#if EFSYS_OPT_NVRAM

#if EFSYS_OPT_FALCON

static efx_nvram_ops_t  __cs    __efx_nvram_falcon_ops = {
#if EFSYS_OPT_DIAG
        falcon_nvram_test,              /* envo_test */
#endif  /* EFSYS_OPT_DIAG */
        falcon_nvram_size,              /* envo_size */
        falcon_nvram_get_version,       /* envo_get_version */
        falcon_nvram_rw_start,          /* envo_rw_start */
        falcon_nvram_read_chunk,        /* envo_read_chunk */
        falcon_nvram_erase,             /* envo_erase */
        falcon_nvram_write_chunk,       /* envo_write_chunk */
        falcon_nvram_rw_finish,         /* envo_rw_finish */
        falcon_nvram_set_version,       /* envo_set_version */
};

#endif  /* EFSYS_OPT_FALCON */

#if EFSYS_OPT_SIENA

static efx_nvram_ops_t  __cs    __efx_nvram_siena_ops = {
#if EFSYS_OPT_DIAG
        siena_nvram_test,               /* envo_test */
#endif  /* EFSYS_OPT_DIAG */
        siena_nvram_size,               /* envo_size */
        siena_nvram_get_version,        /* envo_get_version */
        siena_nvram_rw_start,           /* envo_rw_start */
        siena_nvram_read_chunk,         /* envo_read_chunk */
```



```
        siena_nvram_erase,              /* envo_erase */
        siena_nvram_write_chunk,        /* envo_write_chunk */
        siena_nvram_rw_finish,          /* envo_rw_finish */
        siena_nvram_set_version,        /* envo_set_version */
};

#endif  /* EFSYS_OPT_SIENA */

        __checkReturn   int
efx_nvram_init(
        __in            efx_nic_t *enp)
{
        efx_nvram_ops_t *envop;
        int rc;

        EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
        EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_PROBE);
        EFSYS_ASSERT(!(enp->en_mod_flags & EFX_MOD_NVRAM));

        switch (enp->en_family) {
#if EFSYS_OPT_FALCON
        case EFX_FAMILY_FALCON:
                envop = (efx_nvram_ops_t *)&__efx_nvram_falcon_ops;
                break;
#endif  /* EFSYS_OPT_FALCON */

#if EFSYS_OPT_SIENA
        case EFX_FAMILY_SIENA:
                envop = (efx_nvram_ops_t *)&__efx_nvram_siena_ops;
                break;
#endif  /* EFSYS_OPT_SIENA */

        default:
                EFSYS_ASSERT(0);
                rc = ENOTSUP;
                goto fail1;
        }

        enp->en_envop = envop;
        enp->en_mod_flags |= EFX_MOD_NVRAM;

        return (0);

fail1:
        EFSYS_PROBE1(fail1, int, rc);

        return (rc);
}

#if EFSYS_OPT_DIAG

        __checkReturn           int
efx_nvram_test(
        __in                    efx_nic_t *enp)
{
        efx_nvram_ops_t *envop = enp->en_envop;
        int rc;

        EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
        EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_NVRAM);

        if ((rc = envop->envo_test(enp)) != 0)
                goto fail1;

        return (0);
```

```
fail1:
        EFSYS_PROBE1(fail1, int, rc);

        return (rc);
}

#endif  /* EFSYS_OPT_DIAG */

        __checkReturn           int
efx_nvram_size(
        __in                    efx_nic_t *enp,
        __in                    efx_nvram_type_t type,
        __out                   size_t *sizep)
{
        efx_nvram_ops_t *envop = enp->en_envop;
        int rc;

        EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
        EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_NVRAM);

        EFSYS_ASSERT3U(type, <, EFX_NVRAM_NTYPES);

        if ((rc = envop->envo_size(enp, type, sizep)) != 0)
                goto fail1;

        return (0);

fail1:
        EFSYS_PROBE1(fail1, int, rc);

        return (rc);
}

        __checkReturn           int
efx_nvram_get_version(
        __in                    efx_nic_t *enp,
        __in                    efx_nvram_type_t type,
        __out                   uint32_t *subtypep,
        __out_ecount(4)         uint16_t version[4])
{
        efx_nvram_ops_t *envop = enp->en_envop;
        int rc;

        EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
        EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_PROBE);
        EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_NVRAM);

        EFSYS_ASSERT3U(type, <, EFX_NVRAM_NTYPES);

        if ((rc = envop->envo_get_version(enp, type, subtypep, version)) != 0)
                goto fail1;

        return (0);

fail1:
        EFSYS_PROBE1(fail1, int, rc);

        return (rc);
}

        __checkReturn           int
efx_nvram_rw_start(
        __in                    efx_nic_t *enp,
        __in                    efx_nvram_type_t type,
        __out_opt               size_t *chunk_sizep)
{
```



```
        efx_nvram_ops_t *envop = enp->en_envop;
        int rc;

        EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
        EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_NVRAM);

        EFSYS_ASSERT3U(type, <, EFX_NVRAM_NTYPES);
        EFSYS_ASSERT3U(type, !=, EFX_NVRAM_INVALID);

        EFSYS_ASSERT3U(enp->en_nvram_locked, ==, EFX_NVRAM_INVALID);

        if ((rc = envop->envo_rw_start(enp, type, chunk_sizep)) != 0)
                goto fail1;

        enp->en_nvram_locked = type;

        return (0);

fail1:
        EFSYS_PROBE1(fail1, int, rc);

        return (rc);
}

        __checkReturn           int
efx_nvram_read_chunk(
        __in                    efx_nic_t *enp,
        __in                    efx_nvram_type_t type,
        __in                    unsigned int offset,
        __out_bcount(size)      caddr_t data,
        __in                    size_t size)
{
        efx_nvram_ops_t *envop = enp->en_envop;
        int rc;

        EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
        EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_NVRAM);

        EFSYS_ASSERT3U(type, <, EFX_NVRAM_NTYPES);
        EFSYS_ASSERT3U(type, !=, EFX_NVRAM_INVALID);

        EFSYS_ASSERT3U(enp->en_nvram_locked, ==, type);

        if ((rc = envop->envo_read_chunk(enp, type, offset, data, size)) != 0)
                goto fail1;

        return (0);

fail1:
        EFSYS_PROBE1(fail1, int, rc);

        return (rc);
}

        __checkReturn           int
efx_nvram_erase(
        __in                    efx_nic_t *enp,
        __in                    efx_nvram_type_t type)
{
        efx_nvram_ops_t *envop = enp->en_envop;
        int rc;

        EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
        EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_NVRAM);

        EFSYS_ASSERT3U(type, <, EFX_NVRAM_NTYPES);
```

```
260         EFSYS_ASSERT3U(type, !=, EFX_NVRAM_INVALID);
262         EFSYS_ASSERT3U(enp->en_nvram_locked, ==, type);
264         if ((rc = envop->envo_erase(enp, type)) != 0)
265                 goto fail1;
267         return (0);
269 fail1:
270         EFSYS_PROBE1(fail1, int, rc);
272         return (rc);
273 }
275         __checkReturn           int
276 efx_nvram_write_chunk(
277         __in                    efx_nic_t *enp,
278         __in                    efx_nvram_type_t type,
279         __in                    unsigned int offset,
280         __in_bcount(size)       caddr_t data,
281         __in                    size_t size)
282 {
283         efx_nvram_ops_t *envop = enp->en_envop;
284         int rc;
286         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
287         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_NVRAM);
289         EFSYS_ASSERT3U(type, <, EFX_NVRAM_NTYPES);
290         EFSYS_ASSERT3U(type, !=, EFX_NVRAM_INVALID);
292         EFSYS_ASSERT3U(enp->en_nvram_locked, ==, type);
294         if ((rc = envop->envo_write_chunk(enp, type, offset, data, size)) != 0)
295                 goto fail1;
297         return (0);
299 fail1:
300         EFSYS_PROBE1(fail1, int, rc);
302         return (rc);
303 }
305                                 void
306 efx_nvram_rw_finish(
307         __in                    efx_nic_t *enp,
308         __in                    efx_nvram_type_t type)
309 {
310         efx_nvram_ops_t *envop = enp->en_envop;
312         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
313         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_NVRAM);
315         EFSYS_ASSERT3U(type, <, EFX_NVRAM_NTYPES);
316         EFSYS_ASSERT3U(type, !=, EFX_NVRAM_INVALID);
318         EFSYS_ASSERT3U(enp->en_nvram_locked, ==, type);
320         envop->envo_rw_finish(enp, type);
322         enp->en_nvram_locked = EFX_NVRAM_INVALID;
323 }
325         __checkReturn           int
```



```
326 efx_nvram_set_version(
327         __in                    efx_nic_t *enp,
328         __in                    efx_nvram_type_t type,
329         __out                   uint16_t version[4])
330 {
331         efx_nvram_ops_t *envop = enp->en_envop;
332         int rc;
334         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
335         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_PROBE);
336         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_NVRAM);
338         EFSYS_ASSERT3U(type, <, EFX_NVRAM_NTYPES);
340         /*
341          * The Siena implementation of envo_set_version() will attempt to
342          * acquire the NVRAM_UPDATE lock for the DYNAMIC_CONFIG sector.
343          * Therefore, you can't have already acquired the NVRAM_UPDATE lock.
344          */
345         EFSYS_ASSERT3U(enp->en_nvram_locked, ==, EFX_NVRAM_INVALID);
347         if ((rc = envop->envo_set_version(enp, type, version)) != 0)
348                 goto fail1;
350         return (0);
352 fail1:
353         EFSYS_PROBE1(fail1, int, rc);
355         return (rc);
356 }
358 void
359 efx_nvram_fini(
360         __in            efx_nic_t *enp)
361 {
362         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
363         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_PROBE);
364         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_NVRAM);
366         EFSYS_ASSERT3U(enp->en_nvram_locked, ==, EFX_NVRAM_INVALID);
368         enp->en_envop = NULL;
369         enp->en_mod_flags &= ~EFX_MOD_NVRAM;
370 }
372 #endif  /* EFSYS_OPT_NVRAM */
373 #endif /* ! codereview */
```

```
**********************************************************
   18370 Thu Aug 22 18:59:22 2013
new/usr/src/uts/common/io/sfxge/efx_phy.c
Merged sfxge driver
**********************************************************
```

```
/*
 * Copyright 2007-2013 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#include "efsys.h"
#include "efx.h"
#include "efx_types.h"
#include "efx_regs.h"
#include "efx_impl.h"
#if EFSYS_OPT_FALCON
#include "falcon_nvram.h"
#endif

#if EFSYS_OPT_MAC_FALCON_XMAC
#include "falcon_xmac.h"
#endif

#if EFSYS_OPT_MAC_FALCON_GMAC
#include "falcon_gmac.h"
#endif

#if EFSYS_OPT_PHY_NULL
#include "nullphy.h"
#endif

#if EFSYS_OPT_PHY_QT2022C2
#include "qt2022c2.h"
#endif

#if EFSYS_OPT_PHY_SFX7101
#include "sfx7101.h"
#endif

#if EFSYS_OPT_PHY_TXC43128
#include "txc43128.h"
#endif

#if EFSYS_OPT_PHY_SFT9001
#include "sft9001.h"
#endif
```



```
#if EFSYS_OPT_PHY_QT2025C
#include "qt2025c.h"
#endif

#if EFSYS_OPT_PHY_NULL
static efx_phy_ops_t    __cs __efx_phy_null_ops = {
        NULL,                           /* epo_power */
        nullphy_reset,                  /* epo_reset */
        nullphy_reconfigure,            /* epo_reconfigure */
        nullphy_verify,                 /* epo_verify */
        NULL,                           /* epo_uplink_check */
        nullphy_downlink_check,         /* epo_downlink_check */
        nullphy_oui_get,                /* epo_oui_get */
#if EFSYS_OPT_PHY_STATS
        nullphy_stats_update,           /* epo_stats_update */
#endif  /* EFSYS_OPT_PHY_STATS */
#if EFSYS_OPT_PHY_PROPS
#if EFSYS_OPT_NAMES
        nullphy_prop_name,              /* epo_prop_name */
#endif
        nullphy_prop_get,               /* epo_prop_get */
        nullphy_prop_set,               /* epo_prop_set */
#endif  /* EFSYS_OPT_PHY_PROPS */
#if EFSYS_OPT_PHY_BIST
        NULL,                           /* epo_bist_start */
        NULL,                           /* epo_bist_poll */
        NULL,                           /* epo_bist_stop */
#endif  /* EFSYS_OPT_PHY_BIST */
};
#endif  /* EFSYS_OPT_PHY_NULL */

#if EFSYS_OPT_PHY_QT2022C2
static efx_phy_ops_t    __cs __efx_phy_qt2022c2_ops = {
        NULL,                           /* epo_power */
        qt2022c2_reset,                 /* epo_reset */
        qt2022c2_reconfigure,           /* epo_reconfigure */
        qt2022c2_verify,                /* epo_verify */
        qt2022c2_uplink_check,          /* epo_uplink_check */
        qt2022c2_downlink_check,        /* epo_downlink_check */
        qt2022c2_oui_get,               /* epo_oui_get */
#if EFSYS_OPT_PHY_STATS
        qt2022c2_stats_update,          /* epo_stats_update */
#endif  /* EFSYS_OPT_PHY_STATS */
#if EFSYS_OPT_PHY_PROPS
#if EFSYS_OPT_NAMES
        qt2022c2_prop_name,             /* epo_prop_name */
#endif
        qt2022c2_prop_get,              /* epo_prop_get */
        qt2022c2_prop_set,              /* epo_prop_set */
#endif  /* EFSYS_OPT_PHY_PROPS */
#if EFSYS_OPT_PHY_BIST
        NULL,                           /* epo_bist_start */
        NULL,                           /* epo_bist_poll */
        NULL,                           /* epo_bist_stop */
#endif  /* EFSYS_OPT_PHY_BIST */
};
#endif  /* EFSYS_OPT_PHY_QT2022C2 */

#if EFSYS_OPT_PHY_SFX7101
static efx_phy_ops_t    __cs __efx_phy_sfx7101_ops = {
        sfx7101_power,                  /* epo_power */
        sfx7101_reset,                  /* epo_reset */
        sfx7101_reconfigure,            /* epo_reconfigure */
        sfx7101_verify,                 /* epo_verify */
        sfx7101_uplink_check,           /* epo_uplink_check */
```

```
128         sfx7101_downlink_check,         /* epo_downlink_check */
129         sfx7101_oui_get,                /* epo_oui_get */
130 #if EFSYS_OPT_PHY_STATS
131         sfx7101_stats_update,           /* epo_stats_update */
132 #endif  /* EFSYS_OPT_PHY_STATS */
133 #if EFSYS_OPT_PHY_PROPS
134 #if EFSYS_OPT_NAMES
135         sfx7101_prop_name,              /* epo_prop_name */
136 #endif
137         sfx7101_prop_get,               /* epo_prop_get */
138         sfx7101_prop_set,               /* epo_prop_set */
139 #endif  /* EFSYS_OPT_PHY_PROPS */
140 #if EFSYS_OPT_PHY_BIST
141         NULL,                           /* epo_bist_start */
142         NULL,                           /* epo_bist_poll */
143         NULL,                           /* epo_bist_stop */
144 #endif  /* EFSYS_OPT_PHY_BIST */
145 };
146 #endif  /* EFSYS_OPT_PHY_SFX7101 */

148 #if EFSYS_OPT_PHY_TXC43128
149 static efx_phy_ops_t    __cs __efx_phy_txc43128_ops = {
150         NULL,                           /* epo_power */
151         txc43128_reset,                 /* epo_reset */
152         txc43128_reconfigure,           /* epo_reconfigure */
153         txc43128_verify,                /* epo_verify */
154         txc43128_uplink_check,          /* epo_uplink_check */
155         txc43128_downlink_check,        /* epo_downlink_check */
156         txc43128_oui_get,               /* epo_oui_get */
157 #if EFSYS_OPT_PHY_STATS
158         txc43128_stats_update,          /* epo_stats_update */
159 #endif  /* EFSYS_OPT_PHY_STATS */
160 #if EFSYS_OPT_PHY_PROPS
161 #if EFSYS_OPT_NAMES
162         txc43128_prop_name,             /* epo_prop_name */
163 #endif
164         txc43128_prop_get,              /* epo_prop_get */
165         txc43128_prop_set,              /* epo_prop_set */
166 #endif  /* EFSYS_OPT_PHY_PROPS */
167 #if EFSYS_OPT_PHY_BIST
168         NULL,                           /* epo_bist_start */
169         NULL,                           /* epo_bist_poll */
170         NULL,                           /* epo_bist_stop */
171 #endif  /* EFSYS_OPT_PHY_BIST */
172 };
173 #endif  /* EFSYS_OPT_PHY_TXC43128 */

175 #if EFSYS_OPT_PHY_SFT9001
176 static efx_phy_ops_t    __cs __efx_phy_sft9001_ops = {
177         NULL,                           /* epo_power */
178         sft9001_reset,                  /* epo_reset */
179         sft9001_reconfigure,            /* epo_reconfigure */
180         sft9001_verify,                 /* epo_verify */
181         sft9001_uplink_check,           /* epo_uplink_check */
182         sft9001_downlink_check,         /* epo_downlink_check */
183         sft9001_oui_get,                /* epo_oui_get */
184 #if EFSYS_OPT_PHY_STATS
185         sft9001_stats_update,           /* epo_stats_update */
186 #endif  /* EFSYS_OPT_PHY_STATS */
187 #if EFSYS_OPT_PHY_PROPS
188 #if EFSYS_OPT_NAMES
189         sft9001_prop_name,              /* epo_prop_name */
190 #endif
191         sft9001_prop_get,               /* epo_prop_get */
192         sft9001_prop_set,               /* epo_prop_set */
193 #endif  /* EFSYS_OPT_PHY_PROPS */
```



```
194 #if EFSYS_OPT_PHY_BIST
195         sft9001_bist_start,             /* epo_bist_start */
196         sft9001_bist_poll,              /* epo_bist_poll */
197         sft9001_bist_stop,              /* epo_bist_stop */
198 #endif  /* EFSYS_OPT_PHY_BIST */
199 };
200 #endif  /* EFSYS_OPT_PHY_SFT9001 */

202 #if EFSYS_OPT_PHY_QT2025C
203 static efx_phy_ops_t    __cs __efx_phy_qt2025c_ops = {
204         NULL,                           /* epo_power */
205         qt2025c_reset,                  /* epo_reset */
206         qt2025c_reconfigure,            /* epo_reconfigure */
207         qt2025c_verify,                 /* epo_verify */
208         qt2025c_uplink_check,           /* epo_uplink_check */
209         qt2025c_downlink_check,         /* epo_downlink_check */
210         qt2025c_oui_get,                /* epo_oui_get */
211 #if EFSYS_OPT_PHY_STATS
212         qt2025c_stats_update,           /* epo_stats_update */
213 #endif  /* EFSYS_OPT_PHY_STATS */
214 #if EFSYS_OPT_PHY_PROPS
215 #if EFSYS_OPT_NAMES
216         qt2025c_prop_name,              /* epo_prop_name */
217 #endif
218         qt2025c_prop_get,               /* epo_prop_get */
219         qt2025c_prop_set,               /* epo_prop_set */
220 #endif  /* EFSYS_OPT_PHY_PROPS */
221 #if EFSYS_OPT_PHY_BIST
222         NULL,                           /* epo_bist_start */
223         NULL,                           /* epo_bist_poll */
224         NULL,                           /* epo_bist_stop */
225 #endif  /* EFSYS_OPT_PHY_BIST */
226 };
227 #endif  /* EFSYS_OPT_PHY_QT2025C */

229 #if EFSYS_OPT_SIENA
230 static efx_phy_ops_t    __cs __efx_phy_siena_ops = {
231         siena_phy_power,                /* epo_power */
232         NULL,                           /* epo_reset */
233         siena_phy_reconfigure,          /* epo_reconfigure */
234         siena_phy_verify,               /* epo_verify */
235         NULL,                           /* epo_uplink_check */
236         NULL,                           /* epo_downlink_check */
237         siena_phy_oui_get,              /* epo_oui_get */
238 #if EFSYS_OPT_PHY_STATS
239         siena_phy_stats_update,         /* epo_stats_update */
240 #endif  /* EFSYS_OPT_PHY_STATS */
241 #if EFSYS_OPT_PHY_PROPS
242 #if EFSYS_OPT_NAMES
243         siena_phy_prop_name,            /* epo_prop_name */
244 #endif
245         siena_phy_prop_get,             /* epo_prop_get */
246         siena_phy_prop_set,             /* epo_prop_set */
247 #endif  /* EFSYS_OPT_PHY_PROPS */
248 #if EFSYS_OPT_PHY_BIST
249         siena_phy_bist_start,           /* epo_bist_start */
250         siena_phy_bist_poll,            /* epo_bist_poll */
251         siena_phy_bist_stop,            /* epo_bist_stop */
252 #endif  /* EFSYS_OPT_PHY_BIST */
253 };
254 #endif  /* EFSYS_OPT_SIENA */

256         __checkReturn   int
257 efx_phy_probe(
258         __in            efx_nic_t *enp)
259 {
```

```
260         efx_port_t *epp = &(enp->en_port);
261         efx_nic_cfg_t *encp = &(enp->en_nic_cfg);
262         efx_phy_ops_t *epop;
263         int rc;

265         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);

267         epp->ep_port = encp->enc_port;
268         epp->ep_phy_type = encp->enc_phy_type;

270         /* Hook in operations structure */
271         switch (enp->en_family) {
272 #if EFSYS_OPT_FALCON
273         case EFX_FAMILY_FALCON:
274                 switch (epp->ep_phy_type) {
275 #if EFSYS_OPT_PHY_NULL
276                 case PHY_TYPE_NONE_DECODE:
277                         epop = (efx_phy_ops_t *)&__efx_phy_null_ops;
278                         break;
279 #endif
280 #if EFSYS_OPT_PHY_QT2022C2
281                 case PHY_TYPE_QT2022C2_DECODE:
282                         epop = (efx_phy_ops_t *)&__efx_phy_qt2022c2_ops;
283                         break;
284 #endif
285 #if EFSYS_OPT_PHY_SFX7101
286                 case PHY_TYPE_SFX7101_DECODE:
287                         epop = (efx_phy_ops_t *)&__efx_phy_sfx7101_ops;
288                         break;
289 #endif
290 #if EFSYS_OPT_PHY_TXC43128
291                 case PHY_TYPE_TXC43128_DECODE:
292                         epop = (efx_phy_ops_t *)&__efx_phy_txc43128_ops;
293                         break;
294 #endif
295 #if EFSYS_OPT_PHY_SFT9001
296                 case PHY_TYPE_SFT9001A_DECODE:
297                 case PHY_TYPE_SFT9001B_DECODE:
298                         epop = (efx_phy_ops_t *)&__efx_phy_sft9001_ops;
299                         break;
300 #endif
301 #if EFSYS_OPT_PHY_QT2025C
302                 case EFX_PHY_QT2025C:
303                         epop = (efx_phy_ops_t *)&__efx_phy_qt2025c_ops;
304                         break;
305 #endif
306                 default:
307                         rc = ENOTSUP;
308                         goto fail1;
309                 }
310                 break;
311 #endif  /* EFSYS_OPT_FALCON */
312 #if EFSYS_OPT_SIENA
313         case EFX_FAMILY_SIENA:
314                 epop = (efx_phy_ops_t *)&__efx_phy_siena_ops;
315                 break;
316 #endif  /* EFSYS_OPT_SIENA */
317         default:
318                 rc = ENOTSUP;
319                 goto fail1;
320         }

322         epp->ep_epop = epop;

324         return (0);
```



```
326 fail1:
327         EFSYS_PROBE1(fail1, int, rc);

329         epp->ep_port = 0;
330         epp->ep_phy_type = 0;

332         return (rc);
333 }

335         __checkReturn   int
336 efx_phy_verify(
337         __in            efx_nic_t *enp)
338 {
339         efx_port_t *epp = &(enp->en_port);
340         efx_phy_ops_t *epop = epp->ep_epop;

342         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
343         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_PORT);

345         return (epop->epo_verify(enp));
346 }

348 #if EFSYS_OPT_PHY_LED_CONTROL

350         __checkReturn   int
351 efx_phy_led_set(
352         __in            efx_nic_t *enp,
353         __in            efx_phy_led_mode_t mode)
354 {
355         efx_nic_cfg_t *encp = (&enp->en_nic_cfg);
356         efx_port_t *epp = &(enp->en_port);
357         efx_phy_ops_t *epop = epp->ep_epop;
358         uint32_t mask;
359         int rc;

361         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
362         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_PORT);

364         if (epp->ep_phy_led_mode == mode)
365                 goto done;

367         mask = (1 << EFX_PHY_LED_DEFAULT);
368         mask |= encp->enc_led_mask;

370         if (!((1 << mode) & mask)) {
371                 rc = ENOTSUP;
372                 goto fail1;
373         }

375         EFSYS_ASSERT3U(mode, <, EFX_PHY_LED_NMODES);
376         epp->ep_phy_led_mode = mode;

378         if ((rc = epop->epo_reconfigure(enp)) != 0)
379                 goto fail2;

381 done:
382         return (0);

384 fail2:
385         EFSYS_PROBE(fail2);
386 fail1:
387         EFSYS_PROBE1(fail1, int, rc);

389         return (rc);
390 }
391 #endif  /* EFSYS_OPT_PHY_LED_CONTROL */
```

```
393                         void
394 efx_phy_adv_cap_get(
395         __in            efx_nic_t *enp,
396         __in            uint32_t flag,
397         __out           uint32_t *maskp)
398 {
399         efx_port_t *epp = &(enp->en_port);

401         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
402         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_PROBE);

404         switch (flag) {
405         case EFX_PHY_CAP_CURRENT:
406                 *maskp = epp->ep_adv_cap_mask;
407                 break;
408         case EFX_PHY_CAP_DEFAULT:
409                 *maskp = epp->ep_default_adv_cap_mask;
410                 break;
411         case EFX_PHY_CAP_PERM:
412                 *maskp = epp->ep_phy_cap_mask;
413                 break;
414         default:
415                 EFSYS_ASSERT(B_FALSE);
416                 break;
417         }
418 }

420         __checkReturn   int
421 efx_phy_adv_cap_set(
422         __in            efx_nic_t *enp,
423         __in            uint32_t mask)
424 {
425         efx_port_t *epp = &(enp->en_port);
426         efx_phy_ops_t *epop = epp->ep_epop;
427         uint32_t old_mask;
428         int rc;

430         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
431         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_PORT);

433         if ((mask & ~epp->ep_phy_cap_mask) != 0) {
434                 rc = ENOTSUP;
435                 goto fail1;
436         }

438         if (epp->ep_adv_cap_mask == mask)
439                 goto done;

441         old_mask = epp->ep_adv_cap_mask;
442         epp->ep_adv_cap_mask = mask;

444         if ((rc = epop->epo_reconfigure(enp)) != 0)
445                 goto fail2;

447 done:
448         return (0);

450 fail2:
451         EFSYS_PROBE(fail2);

453         epp->ep_adv_cap_mask = old_mask;
454         /* Reconfigure for robustness */
455         if (epop->epo_reconfigure(enp) != 0) {
456                 /*
457                  * We may have an inconsistent view of our advertised speed
```



```
458                  * capabilities.
459                  */
460                 EFSYS_ASSERT(0);
461         }

463 fail1:
464         EFSYS_PROBE1(fail1, int, rc);

466         return (rc);
467 }

469         void
470 efx_phy_lp_cap_get(
471         __in            efx_nic_t *enp,
472         __out           uint32_t *maskp)
473 {
474         efx_port_t *epp = &(enp->en_port);

476         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
477         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_PORT);

479         *maskp = epp->ep_lp_cap_mask;
480 }

482         __checkReturn   int
483 efx_phy_oui_get(
484         __in            efx_nic_t *enp,
485         __out           uint32_t *ouip)
486 {
487         efx_port_t *epp = &(enp->en_port);
488         efx_phy_ops_t *epop = epp->ep_epop;

490         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
491         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_PORT);

493         return (epop->epo_oui_get(enp, ouip));
494 }

496                         void
497 efx_phy_media_type_get(
498         __in            efx_nic_t *enp,
499         __out           efx_phy_media_type_t *typep)
500 {
501         efx_port_t *epp = &(enp->en_port);

503         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
504         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_PORT);

506         if (epp->ep_module_type != EFX_PHY_MEDIA_INVALID)
507                 *typep = epp->ep_module_type;
508         else
509                 *typep = epp->ep_fixed_port_type;
510 }

512 #if EFSYS_OPT_PHY_STATS

514 #if EFSYS_OPT_NAMES

516 /* START MKCONFIG GENERATED PhyStatNamesBlock 271268f3da0e804f */
517 static const char       __cs * __cs __efx_phy_stat_name[] = {
518         "oui",
519         "pma_pmd_link_up",
520         "pma_pmd_rx_fault",
521         "pma_pmd_tx_fault",
522         "pma_pmd_rev_a",
523         "pma_pmd_rev_b",
```

```
524         "pma_pmd_rev_c",
525         "pma_pmd_rev_d",
526         "pcs_link_up",
527         "pcs_rx_fault",
528         "pcs_tx_fault",
529         "pcs_ber",
530         "pcs_block_errors",
531         "phy_xs_link_up",
532         "phy_xs_rx_fault",
533         "phy_xs_tx_fault",
534         "phy_xs_align",
535         "phy_xs_sync_a",
536         "phy_xs_sync_b",
537         "phy_xs_sync_c",
538         "phy_xs_sync_d",
539         "an_link_up",
540         "an_master",
541         "an_local_rx_ok",
542         "an_remote_rx_ok",
543         "cl22ext_link_up",
544         "snr_a",
545         "snr_b",
546         "snr_c",
547         "snr_d",
548         "pma_pmd_signal_a",
549         "pma_pmd_signal_b",
550         "pma_pmd_signal_c",
551         "pma_pmd_signal_d",
552         "an_complete",
553         "pma_pmd_rev_major",
554         "pma_pmd_rev_minor",
555         "pma_pmd_rev_micro",
556         "pcs_fw_version_0",
557         "pcs_fw_version_1",
558         "pcs_fw_version_2",
559         "pcs_fw_version_3",
560         "pcs_fw_build_yy",
561         "pcs_fw_build_mm",
562         "pcs_fw_build_dd",
563         "pcs_op_mode",
564 };

566 /* END MKCONFIG GENERATED PhyStatNamesBlock */

568                                     const char __cs *
569 efx_phy_stat_name(
570         __in                        efx_nic_t *enp,
571         __in                        efx_phy_stat_t type)
572 {
573         _NOTE(ARGUNUSED(enp))
574         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
575         EFSYS_ASSERT3U(type, <, EFX_PHY_NSTATS);

577         return (__efx_phy_stat_name[type]);
578 }

580 #endif  /* EFSYS_OPT_NAMES */

582         __checkReturn               int
583 efx_phy_stats_update(
584         __in                        efx_nic_t *enp,
585         __in                        efsys_mem_t *esmp,
586         __out_ecount(EFX_PHY_NSTATS)    uint32_t *stat)
587 {
588         efx_port_t *epp = &(enp->en_port);
589         efx_phy_ops_t *epop = epp->ep_epop;
```



```
591         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
592         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_PORT);

594         return (epop->epo_stats_update(enp, esmp, stat));
595 }

597 #endif  /* EFSYS_OPT_PHY_STATS */

599 #if EFSYS_OPT_PHY_PROPS

601 #if EFSYS_OPT_NAMES
602                     const char __cs *
603 efx_phy_prop_name(
604         __in    efx_nic_t *enp,
605         __in    unsigned int id)
606 {
607         efx_port_t *epp = &(enp->en_port);
608         efx_phy_ops_t *epop = epp->ep_epop;

610         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
611         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_PROBE);

613         return (epop->epo_prop_name(enp, id));
614 }
615 #endif  /* EFSYS_OPT_NAMES */

617         __checkReturn   int
618 efx_phy_prop_get(
619         __in            efx_nic_t *enp,
620         __in            unsigned int id,
621         __in            uint32_t flags,
622         __out           uint32_t *valp)
623 {
624         efx_port_t *epp = &(enp->en_port);
625         efx_phy_ops_t *epop = epp->ep_epop;

627         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
628         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_PORT);

630         return (epop->epo_prop_get(enp, id, flags, valp));
631 }

633         __checkReturn   int
634 efx_phy_prop_set(
635         __in            efx_nic_t *enp,
636         __in            unsigned int id,
637         __in            uint32_t val)
638 {
639         efx_port_t *epp = &(enp->en_port);
640         efx_phy_ops_t *epop = epp->ep_epop;

642         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
643         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_PORT);

645         return (epop->epo_prop_set(enp, id, val));
646 }
647 #endif  /* EFSYS_OPT_PHY_STATS */

649 #if EFSYS_OPT_PHY_BIST

651         __checkReturn           int
652 efx_phy_bist_start(
653         __in                    efx_nic_t *enp,
654         __in                    efx_phy_bist_type_t type)
655 {
```

```
656         efx_port_t *epp = &(enp->en_port);
657         efx_phy_ops_t *epop = epp->ep_epop;
658         int rc;

660         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
661         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_PORT);

663         EFSYS_ASSERT3U(type, !=, EFX_PHY_BIST_TYPE_UNKNOWN);
664         EFSYS_ASSERT3U(type, <, EFX_PHY_BIST_TYPE_NTYPES);
665         EFSYS_ASSERT3U(epp->ep_current_bist, ==, EFX_PHY_BIST_TYPE_UNKNOWN);

667         if (epop->epo_bist_start == NULL) {
668                 rc = ENOTSUP;
669                 goto fail1;
670         }

672         if ((rc = epop->epo_bist_start(enp, type)) != 0)
673                 goto fail2;

675         epp->ep_current_bist = type;

677         return (0);

679 fail2:
680         EFSYS_PROBE(fail2);
681 fail1:
682         EFSYS_PROBE1(fail1, int, rc);

684         return (rc);
685 }

687         __checkReturn           int
688 efx_phy_bist_poll(
689         __in                    efx_nic_t *enp,
690         __in                    efx_phy_bist_type_t type,
691         __out                   efx_phy_bist_result_t *resultp,
692         __out_opt               uint32_t *value_maskp,
693         __out_ecount_opt(count) unsigned long *valuesp,
694         __in                    size_t count)
695 {
696         efx_port_t *epp = &(enp->en_port);
697         efx_phy_ops_t *epop = epp->ep_epop;
698         int rc;

700         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
701         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_PORT);

703         EFSYS_ASSERT3U(type, !=, EFX_PHY_BIST_TYPE_UNKNOWN);
704         EFSYS_ASSERT3U(type, <, EFX_PHY_BIST_TYPE_NTYPES);
705         EFSYS_ASSERT3U(epp->ep_current_bist, ==, type);

707         EFSYS_ASSERT(epop->epo_bist_poll != NULL);
708         if (epop->epo_bist_poll == NULL) {
709                 rc = ENOTSUP;
710                 goto fail1;
711         }

713         if ((rc = epop->epo_bist_poll(enp, type, resultp, value_maskp,
714             valuesp, count)) != 0)
715                 goto fail2;

717         return (0);

719 fail2:
720         EFSYS_PROBE(fail2);
721 fail1:
```



```
722         EFSYS_PROBE1(fail1, int, rc);

724         return (rc);
725 }

727                         void
728 efx_phy_bist_stop(
729         __in            efx_nic_t *enp,
730         __in            efx_phy_bist_type_t type)
731 {
732         efx_port_t *epp = &(enp->en_port);
733         efx_phy_ops_t *epop = epp->ep_epop;

735         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
736         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_PORT);

738         EFSYS_ASSERT3U(type, !=, EFX_PHY_BIST_TYPE_UNKNOWN);
739         EFSYS_ASSERT3U(type, <, EFX_PHY_BIST_TYPE_NTYPES);
740         EFSYS_ASSERT3U(epp->ep_current_bist, ==, type);

742         EFSYS_ASSERT(epop->epo_bist_stop != NULL);

744         if (epop->epo_bist_stop != NULL)
745                 epop->epo_bist_stop(enp, type);

747         epp->ep_current_bist = EFX_PHY_BIST_TYPE_UNKNOWN;
748 }

750 #endif  /* EFSYS_OPT_PHY_BIST */
751                         void
752 efx_phy_unprobe(
753         __in    efx_nic_t *enp)
754 {
755         efx_port_t *epp = &(enp->en_port);

757         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);

759         epp->ep_epop = NULL;

761         epp->ep_adv_cap_mask = 0;

763         epp->ep_port = 0;
764         epp->ep_phy_type = 0;
765 }
766 #endif /* ! codereview */
```

```
**********************************************************
    5408 Thu Aug 22 18:59:22 2013
new/usr/src/uts/common/io/sfxge/efx_port.c
Merged sfxge driver
**********************************************************
```

```
/*
 * Copyright 2009 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#include "efsys.h"
#include "efx.h"
#include "efx_types.h"
#include "efx_impl.h"

        __checkReturn   int
efx_port_init(
        __in            efx_nic_t *enp)
{
        efx_port_t *epp = &(enp->en_port);
        efx_phy_ops_t *epop = epp->ep_epop;
        int rc;

        EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
        EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_PROBE);
        EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_NIC);

        if (enp->en_mod_flags & EFX_MOD_PORT) {
                rc = EINVAL;
                goto fail1;
        }

        enp->en_mod_flags |= EFX_MOD_PORT;

        epp->ep_mac_type = EFX_MAC_INVALID;
        epp->ep_link_mode = EFX_LINK_UNKNOWN;
        epp->ep_mac_poll_needed = B_TRUE;
        epp->ep_mac_drain = B_TRUE;

        /* Configure the MAC */
        if ((rc = efx_mac_select(enp)) != 0)
                goto fail1;

        epp->ep_emop->emo_reconfigure(enp);

        /*
```



```
         * Turn on the PHY if available, otherwise reset it, and
         * reconfigure it with the current configuration.
         */
        if (epop->epo_power != NULL) {
                if ((rc = epop->epo_power(enp, B_TRUE)) != 0)
                        goto fail2;
        } else {
                if ((rc = epop->epo_reset(enp)) != 0)
                        goto fail2;
        }

        EFSYS_ASSERT(enp->en_reset_flags & EFX_RESET_PHY);
        enp->en_reset_flags &= ~EFX_RESET_PHY;

        if ((rc = epop->epo_reconfigure(enp)) != 0)
                goto fail3;

        return (0);

fail3:
        EFSYS_PROBE(fail3);
fail2:
        EFSYS_PROBE(fail2);
fail1:
        EFSYS_PROBE1(fail1, int, rc);

        enp->en_mod_flags &= ~EFX_MOD_PORT;

        return (rc);
}

        __checkReturn   int
efx_port_poll(
        __in            efx_nic_t *enp,
        __out           efx_link_mode_t *link_modep)
{
        efx_port_t *epp = &(enp->en_port);
        efx_mac_ops_t *emop = epp->ep_emop;
        efx_link_mode_t ignore_link_mode;
        int rc;

        EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
        EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_PORT);

        EFSYS_ASSERT(emop != NULL);
        EFSYS_ASSERT(!epp->ep_mac_stats_pending);

        if (link_modep == NULL)
                link_modep = &ignore_link_mode;

        if ((rc = emop->emo_poll(enp, link_modep)) != 0)
                goto fail1;

        return (0);

fail1:
        EFSYS_PROBE1(fail1, int, rc);

        return (rc);
}

#if EFSYS_OPT_LOOPBACK

        __checkReturn   int
efx_port_loopback_set(
        __in            efx_nic_t *enp,
```

```
128         __in            efx_link_mode_t link_mode,
129         __in            efx_loopback_type_t loopback_type)
130 {
131         efx_port_t *epp = &(enp->en_port);
132         efx_nic_cfg_t *encp = &(enp->en_nic_cfg);
133         efx_mac_ops_t *emop = epp->ep_emop;
134         int rc;
135
136         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
137         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_PORT);
138         EFSYS_ASSERT(emop != NULL);
139
140         EFSYS_ASSERT(link_mode < EFX_LINK_NMODES);
141         if ((1 << loopback_type) & ~encp->enc_loopback_types[link_mode]) {
142                 rc = ENOTSUP;
143                 goto fail1;
144         }
145
146         if (epp->ep_loopback_type == loopback_type &&
147             epp->ep_loopback_link_mode == link_mode)
148                 return (0);
149
150         if ((rc = emop->emo_loopback_set(enp, link_mode, loopback_type)) != 0)
151                 goto fail2;
152
153         return (0);
154
155 fail2:
156         EFSYS_PROBE(fail2);
157 fail1:
158         EFSYS_PROBE1(fail1, int, rc);
159
160         return (rc);
161 }
162
163 #if EFSYS_OPT_NAMES
164
165 static const char       __cs * __cs __efx_loopback_type_name[] = {
166         "OFF",
167         "DATA",
168         "GMAC",
169         "XGMII",
170         "XGXS",
171         "XAUI",
172         "GMII",
173         "SGMII",
174         "XGBR",
175         "XFI",
176         "XAUI_FAR",
177         "GMII_FAR",
178         "SGMII_FAR",
179         "XFI_FAR",
180         "GPHY",
181         "PHY_XS",
182         "PCS",
183         "PMA_PMD",
184 };
185
186         __checkReturn   const char __cs *
187 efx_loopback_type_name(
188         __in            efx_nic_t *enp,
189         __in            efx_loopback_type_t type)
190 {
191         _NOTE(ARGUNUSED(enp))
192         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
193         EFSYS_ASSERT3U(type, <, EFX_LOOPBACK_NTYPES);
```



```
194
195         return (__efx_loopback_type_name[type]);
196 }
197
198 #endif  /* EFSYS_OPT_NAMES */
199
200 #endif  /* EFSYS_OPT_LOOPBACK */
201
202                         void
203 efx_port_fini(
204         __in            efx_nic_t *enp)
205 {
206         efx_port_t *epp = &(enp->en_port);
207         efx_phy_ops_t *epop = epp->ep_epop;
208
209         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
210         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_PROBE);
211         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_NIC);
212         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_PORT);
213
214         EFSYS_ASSERT(epp->ep_mac_drain);
215
216         epp->ep_emop = NULL;
217         epp->ep_mac_type = EFX_MAC_INVALID;
218         epp->ep_mac_drain = B_FALSE;
219         epp->ep_mac_poll_needed = B_FALSE;
220
221         /* Turn off the PHY */
222         if (epop->epo_power != NULL)
223                 (void) epop->epo_power(enp, B_FALSE);
224
225         enp->en_mod_flags &= ~EFX_MOD_PORT;
226 }
227 #endif /* ! codereview */
```

```
**********************************************************
  120994 Thu Aug 22 18:59:22 2013
new/usr/src/uts/common/io/sfxge/efx_regs.h
Merged sfxge driver
**********************************************************
```

```
/*-
 * Copyright 2007-2013 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#ifndef _SYS_EFX_REGS_H
#define _SYS_EFX_REGS_H


#ifdef  __cplusplus
extern "C" {
#endif


/**************************************************************************
 *
 * Falcon/Siena registers and descriptors
 *
 **************************************************************************
 */

/*
 * FR_AB_EE_VPD_CFG0_REG_SF(128bit):
 * SPI/VPD configuration register 0
 */
#define FR_AB_EE_VPD_CFG0_REG_SF_OFST 0x00000300
/* falcona0,falconb0=eeprom_flash */
/*
 * FR_AB_EE_VPD_CFG0_REG(128bit):
 * SPI/VPD configuration register 0
 */
#define FR_AB_EE_VPD_CFG0_REG_OFST 0x00000140
/* falcona0,falconb0=net_func_bar2,falcona0=char_func_bar0 */

#define FRF_AB_EE_SF_FASTRD_EN_LBN 127
#define FRF_AB_EE_SF_FASTRD_EN_WIDTH 1
#define FRF_AB_EE_SF_CLOCK_DIV_LBN 120
#define FRF_AB_EE_SF_CLOCK_DIV_WIDTH 7
#define FRF_AB_EE_VPD_WIP_POLL_LBN 119
#define FRF_AB_EE_VPD_WIP_POLL_WIDTH 1
#define FRF_AB_EE_EE_CLOCK_DIV_LBN 112
```



```
#define FRF_AB_EE_EE_CLOCK_DIV_WIDTH 7
#define FRF_AB_EE_EE_WR_TMR_VALUE_LBN 96
#define FRF_AB_EE_EE_WR_TMR_VALUE_WIDTH 16
#define FRF_AB_EE_VPDW_LENGTH_LBN 80
#define FRF_AB_EE_VPDW_LENGTH_WIDTH 15
#define FRF_AB_EE_VPDW_BASE_LBN 64
#define FRF_AB_EE_VPDW_BASE_WIDTH 15
#define FRF_AB_EE_VPD_WR_CMD_EN_LBN 56
#define FRF_AB_EE_VPD_WR_CMD_EN_WIDTH 8
#define FRF_AB_EE_VPD_BASE_LBN 32
#define FRF_AB_EE_VPD_BASE_WIDTH 24
#define FRF_AB_EE_VPD_LENGTH_LBN 16
#define FRF_AB_EE_VPD_LENGTH_WIDTH 15
#define FRF_AB_EE_VPD_AD_SIZE_LBN 8
#define FRF_AB_EE_VPD_AD_SIZE_WIDTH 5
#define FRF_AB_EE_VPD_ACCESS_ON_LBN 5
#define FRF_AB_EE_VPD_ACCESS_ON_WIDTH 1
#define FRF_AB_EE_VPD_ACCESS_BLOCK_LBN 4
#define FRF_AB_EE_VPD_ACCESS_BLOCK_WIDTH 1
#define FRF_AB_EE_VPD_DEV_SF_SEL_LBN 2
#define FRF_AB_EE_VPD_DEV_SF_SEL_WIDTH 1
#define FRF_AB_EE_VPD_EN_AD9_MODE_LBN 1
#define FRF_AB_EE_VPD_EN_AD9_MODE_WIDTH 1
#define FRF_AB_EE_VPD_EN_LBN 0
#define FRF_AB_EE_VPD_EN_WIDTH 1


/*
 * FR_AB_PCIE_SD_CTL0123_REG_SF(128bit):
 * PCIE SerDes control register 0 to 3
 */
#define FR_AB_PCIE_SD_CTL0123_REG_SF_OFST 0x00000320
/* falcona0,falconb0=eeprom_flash */
/*
 * FR_AB_PCIE_SD_CTL0123_REG(128bit):
 * PCIE SerDes control register 0 to 3
 */
#define FR_AB_PCIE_SD_CTL0123_REG_OFST 0x00000320
/* falcona0,falconb0=net_func_bar2,falcona0=char_func_bar0 */

#define FRF_AB_PCIE_TESTSIG_H_LBN 96
#define FRF_AB_PCIE_TESTSIG_H_WIDTH 19
#define FRF_AB_PCIE_TESTSIG_L_LBN 64
#define FRF_AB_PCIE_TESTSIG_L_WIDTH 19
#define FRF_AB_PCIE_OFFSET_LBN 56
#define FRF_AB_PCIE_OFFSET_WIDTH 8
#define FRF_AB_PCIE_OFFSETEN_H_LBN 55
#define FRF_AB_PCIE_OFFSETEN_H_WIDTH 1
#define FRF_AB_PCIE_OFFSETEN_L_LBN 54
#define FRF_AB_PCIE_OFFSETEN_L_WIDTH 1
#define FRF_AB_PCIE_HIVMODE_H_LBN 53
#define FRF_AB_PCIE_HIVMODE_H_WIDTH 1
#define FRF_AB_PCIE_HIVMODE_L_LBN 52
#define FRF_AB_PCIE_HIVMODE_L_WIDTH 1
#define FRF_AB_PCIE_PARRESET_H_LBN 51
#define FRF_AB_PCIE_PARRESET_H_WIDTH 1
#define FRF_AB_PCIE_PARRESET_L_LBN 50
#define FRF_AB_PCIE_PARRESET_L_WIDTH 1
#define FRF_AB_PCIE_LPBKWDRV_H_LBN 49
#define FRF_AB_PCIE_LPBKWDRV_H_WIDTH 1
#define FRF_AB_PCIE_LPBKWDRV_L_LBN 48
#define FRF_AB_PCIE_LPBKWDRV_L_WIDTH 1
#define FRF_AB_PCIE_LPBK_LBN 40
#define FRF_AB_PCIE_LPBK_WIDTH 8
#define FRF_AB_PCIE_PARLPBK_LBN 32
#define FRF_AB_PCIE_PARLPBK_WIDTH 8
```

```
#define FRF_AB_PCIE_RXTERMADJ_H_LBN 30
#define FRF_AB_PCIE_RXTERMADJ_H_WIDTH 2
#define FRF_AB_PCIE_RXTERMADJ_L_LBN 28
#define FRF_AB_PCIE_RXTERMADJ_L_WIDTH 2
#define FFE_AB_PCIE_RXTERMADJ_MIN15PCNT 3
#define FFE_AB_PCIE_RXTERMADJ_PL10PCNT 2
#define FFE_AB_PCIE_RXTERMADJ_MIN17PCNT 1
#define FFE_AB_PCIE_RXTERMADJ_NOMNL 0
#define FRF_AB_PCIE_TXTERMADJ_H_LBN 26
#define FRF_AB_PCIE_TXTERMADJ_H_WIDTH 2
#define FRF_AB_PCIE_TXTERMADJ_L_LBN 24
#define FRF_AB_PCIE_TXTERMADJ_L_WIDTH 2
#define FFE_AB_PCIE_TXTERMADJ_MIN15PCNT 3
#define FFE_AB_PCIE_TXTERMADJ_PL10PCNT 2
#define FFE_AB_PCIE_TXTERMADJ_MIN17PCNT 1
#define FFE_AB_PCIE_TXTERMADJ_NOMNL 0
#define FRF_AB_PCIE_RXEQCTL_H_LBN 18
#define FRF_AB_PCIE_RXEQCTL_H_WIDTH 2
#define FRF_AB_PCIE_RXEQCTL_L_LBN 16
#define FRF_AB_PCIE_RXEQCTL_L_WIDTH 2
#define FFE_AB_PCIE_RXEQCTL_OFF_ALT 3
#define FFE_AB_PCIE_RXEQCTL_OFF 2
#define FFE_AB_PCIE_RXEQCTL_MIN 1
#define FFE_AB_PCIE_RXEQCTL_MAX 0
#define FRF_AB_PCIE_HIDRV_LBN 8
#define FRF_AB_PCIE_HIDRV_WIDTH 8
#define FRF_AB_PCIE_LODRV_LBN 0
#define FRF_AB_PCIE_LODRV_WIDTH 8


/*
 * FR_AB_PCIE_SD_CTL45_REG_SF(128bit):
 * PCIE SerDes control register 4 and 5
 */
#define FR_AB_PCIE_SD_CTL45_REG_SF_OFST 0x00000330
/* falcona0,falconb0=eeprom_flash */
/*
 * FR_AB_PCIE_SD_CTL45_REG(128bit):
 * PCIE SerDes control register 4 and 5
 */
#define FR_AB_PCIE_SD_CTL45_REG_OFST 0x00000330
/* falcona0,falconb0=net_func_bar2,falcona0=char_func_bar0 */

#define FRF_AB_PCIE_DTX7_LBN 60
#define FRF_AB_PCIE_DTX7_WIDTH 4
#define FRF_AB_PCIE_DTX6_LBN 56
#define FRF_AB_PCIE_DTX6_WIDTH 4
#define FRF_AB_PCIE_DTX5_LBN 52
#define FRF_AB_PCIE_DTX5_WIDTH 4
#define FRF_AB_PCIE_DTX4_LBN 48
#define FRF_AB_PCIE_DTX4_WIDTH 4
#define FRF_AB_PCIE_DTX3_LBN 44
#define FRF_AB_PCIE_DTX3_WIDTH 4
#define FRF_AB_PCIE_DTX2_LBN 40
#define FRF_AB_PCIE_DTX2_WIDTH 4
#define FRF_AB_PCIE_DTX1_LBN 36
#define FRF_AB_PCIE_DTX1_WIDTH 4
#define FRF_AB_PCIE_DTX0_LBN 32
#define FRF_AB_PCIE_DTX0_WIDTH 4
#define FRF_AB_PCIE_DEQ7_LBN 28
#define FRF_AB_PCIE_DEQ7_WIDTH 4
#define FRF_AB_PCIE_DEQ6_LBN 24
#define FRF_AB_PCIE_DEQ6_WIDTH 4
#define FRF_AB_PCIE_DEQ5_LBN 20
#define FRF_AB_PCIE_DEQ5_WIDTH 4
#define FRF_AB_PCIE_DEQ4_LBN 16
```



```
#define FRF_AB_PCIE_DEQ4_WIDTH 4
#define FRF_AB_PCIE_DEQ3_LBN 12
#define FRF_AB_PCIE_DEQ3_WIDTH 4
#define FRF_AB_PCIE_DEQ2_LBN 8
#define FRF_AB_PCIE_DEQ2_WIDTH 4
#define FRF_AB_PCIE_DEQ1_LBN 4
#define FRF_AB_PCIE_DEQ1_WIDTH 4
#define FRF_AB_PCIE_DEQ0_LBN 0
#define FRF_AB_PCIE_DEQ0_WIDTH 4


/*
 * FR_AB_PCIE_PCS_CTL_STAT_REG_SF(128bit):
 * PCIE PCS control and status register
 */
#define FR_AB_PCIE_PCS_CTL_STAT_REG_SF_OFST 0x00000340
/* falcona0,falconb0=eeprom_flash */
/*
 * FR_AB_PCIE_PCS_CTL_STAT_REG(128bit):
 * PCIE PCS control and status register
 */
#define FR_AB_PCIE_PCS_CTL_STAT_REG_OFST 0x00000340
/* falcona0,falconb0=net_func_bar2,falcona0=char_func_bar0 */

#define FRF_AB_PCIE_PRBSERRCOUNT0_H_LBN 52
#define FRF_AB_PCIE_PRBSERRCOUNT0_H_WIDTH 4
#define FRF_AB_PCIE_PRBSERRCOUNT0_L_LBN 48
#define FRF_AB_PCIE_PRBSERRCOUNT0_L_WIDTH 4
#define FRF_AB_PCIE_PRBSERR_LBN 40
#define FRF_AB_PCIE_PRBSERR_WIDTH 8
#define FRF_AB_PCIE_PRBSERRH0_LBN 32
#define FRF_AB_PCIE_PRBSERRH0_WIDTH 8
#define FRF_AB_PCIE_FASTINIT_H_LBN 15
#define FRF_AB_PCIE_FASTINIT_H_WIDTH 1
#define FRF_AB_PCIE_FASTINIT_L_LBN 14
#define FRF_AB_PCIE_FASTINIT_L_WIDTH 1
#define FRF_AB_PCIE_CTCDISABLE_H_LBN 13
#define FRF_AB_PCIE_CTCDISABLE_H_WIDTH 1
#define FRF_AB_PCIE_CTCDISABLE_L_LBN 12
#define FRF_AB_PCIE_CTCDISABLE_L_WIDTH 1
#define FRF_AB_PCIE_PRBSSYNC_H_LBN 11
#define FRF_AB_PCIE_PRBSSYNC_H_WIDTH 1
#define FRF_AB_PCIE_PRBSSYNC_L_LBN 10
#define FRF_AB_PCIE_PRBSSYNC_L_WIDTH 1
#define FRF_AB_PCIE_PRBSERRACK_H_LBN 9
#define FRF_AB_PCIE_PRBSERRACK_H_WIDTH 1
#define FRF_AB_PCIE_PRBSERRACK_L_LBN 8
#define FRF_AB_PCIE_PRBSERRACK_L_WIDTH 1
#define FRF_AB_PCIE_PRBSSEL_LBN 0
#define FRF_AB_PCIE_PRBSSEL_WIDTH 8


/*
 * FR_AB_HW_INIT_REG_SF(128bit):
 * Hardware initialization register
 */
#define FR_AB_HW_INIT_REG_SF_OFST 0x00000350
/* falcona0,falconb0=eeprom_flash */
/*
 * FR_AZ_HW_INIT_REG(128bit):
 * Hardware initialization register
 */
#define FR_AZ_HW_INIT_REG_OFST 0x000000c0
/* falcona0,falconb0,sienaa0=net_func_bar2,falcona0=char_func_bar0 */

#define FRF_BB_BDMRD_CPLF_FULL_LBN 124
```

```
#define FRF_BB_BDMRD_CPLF_FULL_WIDTH 1
#define FRF_BB_PCIE_CPL_TIMEOUT_CTRL_LBN 121
#define FRF_BB_PCIE_CPL_TIMEOUT_CTRL_WIDTH 3
#define FRF_CZ_TX_MRG_TAGS_LBN 120
#define FRF_CZ_TX_MRG_TAGS_WIDTH 1
#define FRF_AZ_TRGT_MASK_ALL_LBN 100
#define FRF_AZ_TRGT_MASK_ALL_WIDTH 1
#define FRF_AZ_DOORBELL_DROP_LBN 92
#define FRF_AZ_DOORBELL_DROP_WIDTH 8
#define FRF_AB_TX_RREQ_MASK_EN_LBN 76
#define FRF_AB_TX_RREQ_MASK_EN_WIDTH 1
#define FRF_AB_PE_EIDLE_DIS_LBN 75
#define FRF_AB_PE_EIDLE_DIS_WIDTH 1
#define FRF_AZ_FC_BLOCKING_EN_LBN 45
#define FRF_AZ_FC_BLOCKING_EN_WIDTH 1
#define FRF_AZ_B2B_REQ_EN_LBN 44
#define FRF_AZ_B2B_REQ_EN_WIDTH 1
#define FRF_AZ_POST_WR_MASK_LBN 40
#define FRF_AZ_POST_WR_MASK_WIDTH 4
#define FRF_AZ_TLP_TC_LBN 34
#define FRF_AZ_TLP_TC_WIDTH 3
#define FRF_AZ_TLP_ATTR_LBN 32
#define FRF_AZ_TLP_ATTR_WIDTH 2
#define FRF_AB_INTB_VEC_LBN 24
#define FRF_AB_INTB_VEC_WIDTH 5
#define FRF_AB_INTA_VEC_LBN 16
#define FRF_AB_INTA_VEC_WIDTH 5
#define FRF_AZ_WD_TIMER_LBN 8
#define FRF_AZ_WD_TIMER_WIDTH 8
#define FRF_AZ_US_DISABLE_LBN 5
#define FRF_AZ_US_DISABLE_WIDTH 1
#define FRF_AZ_TLP_EP_LBN 4
#define FRF_AZ_TLP_EP_WIDTH 1
#define FRF_AZ_ATTR_SEL_LBN 3
#define FRF_AZ_ATTR_SEL_WIDTH 1
#define FRF_AZ_TD_SEL_LBN 1
#define FRF_AZ_TD_SEL_WIDTH 1
#define FRF_AZ_TLP_TD_LBN 0
#define FRF_AZ_TLP_TD_WIDTH 1


/*
 * FR_AB_NIC_STAT_REG_SF(128bit):
 * NIC status register
 */
#define FR_AB_NIC_STAT_REG_SF_OFST 0x00000360
/* falcona0,falconb0=eeprom_flash */
/*
 * FR_AB_NIC_STAT_REG(128bit):
 * NIC status register
 */
#define FR_AB_NIC_STAT_REG_OFST 0x00000200
/* falcona0,falconb0=net_func_bar2,falcona0=char_func_bar0 */

#define FRF_BB_AER_DIS_LBN 34
#define FRF_BB_AER_DIS_WIDTH 1
#define FRF_BB_EE_STRAP_EN_LBN 31
#define FRF_BB_EE_STRAP_EN_WIDTH 1
#define FRF_BB_EE_STRAP_LBN 24
#define FRF_BB_EE_STRAP_WIDTH 4
#define FRF_BB_REVISION_ID_LBN 17
#define FRF_BB_REVISION_ID_WIDTH 7
#define FRF_AB_ONCHIP_SRAM_LBN 16
#define FRF_AB_ONCHIP_SRAM_WIDTH 1
#define FRF_AB_SF_PRST_LBN 9
#define FRF_AB_SF_PRST_WIDTH 1
```



```
#define FRF_AB_EE_PRST_LBN 8
#define FRF_AB_EE_PRST_WIDTH 1
#define FRF_AB_ATE_MODE_LBN 3
#define FRF_AB_ATE_MODE_WIDTH 1
#define FRF_AB_STRAP_PINS_LBN 0
#define FRF_AB_STRAP_PINS_WIDTH 3


/*
 * FR_AB_GLB_CTL_REG_SF(128bit):
 * Global control register
 */
#define FR_AB_GLB_CTL_REG_SF_OFST 0x00000370
/* falcona0,falconb0=eeprom_flash */
/*
 * FR_AB_GLB_CTL_REG(128bit):
 * Global control register
 */
#define FR_AB_GLB_CTL_REG_OFST 0x00000220
/* falcona0,falconb0=net_func_bar2,falcona0=char_func_bar0 */

#define FRF_AB_EXT_PHY_RST_CTL_LBN 63
#define FRF_AB_EXT_PHY_RST_CTL_WIDTH 1
#define FRF_AB_XAUI_SD_RST_CTL_LBN 62
#define FRF_AB_XAUI_SD_RST_CTL_WIDTH 1
#define FRF_AB_PCIE_SD_RST_CTL_LBN 61
#define FRF_AB_PCIE_SD_RST_CTL_WIDTH 1
#define FRF_AA_PCIX_RST_CTL_LBN 60
#define FRF_AA_PCIX_RST_CTL_WIDTH 1
#define FRF_BB_BIU_RST_CTL_LBN 60
#define FRF_BB_BIU_RST_CTL_WIDTH 1
#define FRF_AB_PCIE_STKY_RST_CTL_LBN 59
#define FRF_AB_PCIE_STKY_RST_CTL_WIDTH 1
#define FRF_AB_PCIE_NSTKY_RST_CTL_LBN 58
#define FRF_AB_PCIE_NSTKY_RST_CTL_WIDTH 1
#define FRF_AB_PCIE_CORE_RST_CTL_LBN 57
#define FRF_AB_PCIE_CORE_RST_CTL_WIDTH 1
#define FRF_AB_XGRX_RST_CTL_LBN 56
#define FRF_AB_XGRX_RST_CTL_WIDTH 1
#define FRF_AB_XGTX_RST_CTL_LBN 55
#define FRF_AB_XGTX_RST_CTL_WIDTH 1
#define FRF_AB_EM_RST_CTL_LBN 54
#define FRF_AB_EM_RST_CTL_WIDTH 1
#define FRF_AB_EV_RST_CTL_LBN 53
#define FRF_AB_EV_RST_CTL_WIDTH 1
#define FRF_AB_SR_RST_CTL_LBN 52
#define FRF_AB_SR_RST_CTL_WIDTH 1
#define FRF_AB_RX_RST_CTL_LBN 51
#define FRF_AB_RX_RST_CTL_WIDTH 1
#define FRF_AB_TX_RST_CTL_LBN 50
#define FRF_AB_TX_RST_CTL_WIDTH 1
#define FRF_AB_EE_RST_CTL_LBN 49
#define FRF_AB_EE_RST_CTL_WIDTH 1
#define FRF_AB_CS_RST_CTL_LBN 48
#define FRF_AB_CS_RST_CTL_WIDTH 1
#define FRF_AB_HOT_RST_CTL_LBN 40
#define FRF_AB_HOT_RST_CTL_WIDTH 2
#define FRF_AB_RST_EXT_PHY_LBN 31
#define FRF_AB_RST_EXT_PHY_WIDTH 1
#define FRF_AB_RST_XAUI_SD_LBN 30
#define FRF_AB_RST_XAUI_SD_WIDTH 1
#define FRF_AB_RST_PCIE_SD_LBN 29
#define FRF_AB_RST_PCIE_SD_WIDTH 1
#define FRF_AA_RST_PCIX_LBN 28
#define FRF_AA_RST_PCIX_WIDTH 1
#define FRF_BB_RST_BIU_LBN 28
```

```
392 #define FRF_BB_RST_BIU_WIDTH 1
393 #define FRF_AB_RST_PCIE_STKY_LBN 27
394 #define FRF_AB_RST_PCIE_STKY_WIDTH 1
395 #define FRF_AB_RST_PCIE_NSTKY_LBN 26
396 #define FRF_AB_RST_PCIE_NSTKY_WIDTH 1
397 #define FRF_AB_RST_PCIE_CORE_LBN 25
398 #define FRF_AB_RST_PCIE_CORE_WIDTH 1
399 #define FRF_AB_RST_XGRX_LBN 24
400 #define FRF_AB_RST_XGRX_WIDTH 1
401 #define FRF_AB_RST_XGTX_LBN 23
402 #define FRF_AB_RST_XGTX_WIDTH 1
403 #define FRF_AB_RST_EM_LBN 22
404 #define FRF_AB_RST_EM_WIDTH 1
405 #define FRF_AB_RST_EV_LBN 21
406 #define FRF_AB_RST_EV_WIDTH 1
407 #define FRF_AB_RST_SR_LBN 20
408 #define FRF_AB_RST_SR_WIDTH 1
409 #define FRF_AB_RST_RX_LBN 19
410 #define FRF_AB_RST_RX_WIDTH 1
411 #define FRF_AB_RST_TX_LBN 18
412 #define FRF_AB_RST_TX_WIDTH 1
413 #define FRF_AB_RST_SF_LBN 17
414 #define FRF_AB_RST_SF_WIDTH 1
415 #define FRF_AB_RST_CS_LBN 16
416 #define FRF_AB_RST_CS_WIDTH 1
417 #define FRF_AB_INT_RST_DUR_LBN 4
418 #define FRF_AB_INT_RST_DUR_WIDTH 3
419 #define FRF_AB_EXT_PHY_RST_DUR_LBN 1
420 #define FRF_AB_EXT_PHY_RST_DUR_WIDTH 3
421 #define FFE_AB_EXT_PHY_RST_DUR_10240US 7
422 #define FFE_AB_EXT_PHY_RST_DUR_5120US 6
423 #define FFE_AB_EXT_PHY_RST_DUR_2560US 5
424 #define FFE_AB_EXT_PHY_RST_DUR_1280US 4
425 #define FFE_AB_EXT_PHY_RST_DUR_640US 3
426 #define FFE_AB_EXT_PHY_RST_DUR_320US 2
427 #define FFE_AB_EXT_PHY_RST_DUR_160US 1
428 #define FFE_AB_EXT_PHY_RST_DUR_80US 0
429 #define FRF_AB_SWRST_LBN 0
430 #define FRF_AB_SWRST_WIDTH 1


433 /*
434  * FR_AZ_IOM_IND_ADR_REG(32bit):
435  * IO-mapped indirect access address register
436  */
437 #define FR_AZ_IOM_IND_ADR_REG_OFST 0x00000000
438 /* falcona0,falconb0,sienaa0=net_func_bar0 */

440 #define FRF_AZ_IOM_AUTO_ADR_INC_EN_LBN 24
441 #define FRF_AZ_IOM_AUTO_ADR_INC_EN_WIDTH 1
442 #define FRF_AZ_IOM_IND_ADR_LBN 0
443 #define FRF_AZ_IOM_IND_ADR_WIDTH 24


446 /*
447  * FR_AZ_IOM_IND_DAT_REG(32bit):
448  * IO-mapped indirect access data register
449  */
450 #define FR_AZ_IOM_IND_DAT_REG_OFST 0x00000004
451 /* falcona0,falconb0,sienaa0=net_func_bar0 */

453 #define FRF_AZ_IOM_IND_DAT_LBN 0
454 #define FRF_AZ_IOM_IND_DAT_WIDTH 32


457 /*
```



```
458  * FR_AZ_ADR_REGION_REG(128bit):
459  * Address region register
460  */
461 #define FR_AZ_ADR_REGION_REG_OFST 0x00000000
462 /* falcona0,falconb0,sienaa0=net_func_bar2,falcona0=char_func_bar0 */

464 #define FRF_AZ_ADR_REGION3_LBN 96
465 #define FRF_AZ_ADR_REGION3_WIDTH 18
466 #define FRF_AZ_ADR_REGION2_LBN 64
467 #define FRF_AZ_ADR_REGION2_WIDTH 18
468 #define FRF_AZ_ADR_REGION1_LBN 32
469 #define FRF_AZ_ADR_REGION1_WIDTH 18
470 #define FRF_AZ_ADR_REGION0_LBN 0
471 #define FRF_AZ_ADR_REGION0_WIDTH 18


474 /*
475  * FR_AZ_INT_EN_REG_KER(128bit):
476  * Kernel driver Interrupt enable register
477  */
478 #define FR_AZ_INT_EN_REG_KER_OFST 0x00000010
479 /* falcona0,falconb0,sienaa0=net_func_bar2 */

481 #define FRF_AZ_KER_INT_LEVE_SEL_LBN 8
482 #define FRF_AZ_KER_INT_LEVE_SEL_WIDTH 6
483 #define FRF_AZ_KER_INT_CHAR_LBN 4
484 #define FRF_AZ_KER_INT_CHAR_WIDTH 1
485 #define FRF_AZ_KER_INT_KER_LBN 3
486 #define FRF_AZ_KER_INT_KER_WIDTH 1
487 #define FRF_AZ_DRV_INT_EN_KER_LBN 0
488 #define FRF_AZ_DRV_INT_EN_KER_WIDTH 1


491 /*
492  * FR_AZ_INT_EN_REG_CHAR(128bit):
493  * Char Driver interrupt enable register
494  */
495 #define FR_AZ_INT_EN_REG_CHAR_OFST 0x00000020
496 /* falconb0,sienaa0=net_func_bar2,falcona0=char_func_bar0 */

498 #define FRF_AZ_CHAR_INT_LEVE_SEL_LBN 8
499 #define FRF_AZ_CHAR_INT_LEVE_SEL_WIDTH 6
500 #define FRF_AZ_CHAR_INT_CHAR_LBN 4
501 #define FRF_AZ_CHAR_INT_CHAR_WIDTH 1
502 #define FRF_AZ_CHAR_INT_KER_LBN 3
503 #define FRF_AZ_CHAR_INT_KER_WIDTH 1
504 #define FRF_AZ_DRV_INT_EN_CHAR_LBN 0
505 #define FRF_AZ_DRV_INT_EN_CHAR_WIDTH 1


508 /*
509  * FR_AZ_INT_ADR_REG_KER(128bit):
510  * Interrupt host address for Kernel driver
511  */
512 #define FR_AZ_INT_ADR_REG_KER_OFST 0x00000030
513 /* falcona0,falconb0,sienaa0=net_func_bar2 */

515 #define FRF_AZ_NORM_INT_VEC_DIS_KER_LBN 64
516 #define FRF_AZ_NORM_INT_VEC_DIS_KER_WIDTH 1
517 #define FRF_AZ_INT_ADR_KER_LBN 0
518 #define FRF_AZ_INT_ADR_KER_WIDTH 64
519 #define FRF_AZ_INT_ADR_KER_DW0_LBN 0
520 #define FRF_AZ_INT_ADR_KER_DW0_WIDTH 32
521 #define FRF_AZ_INT_ADR_KER_DW1_LBN 32
522 #define FRF_AZ_INT_ADR_KER_DW1_WIDTH 32
```

```
525 /*
526  * FR_AZ_INT_ADR_REG_CHAR(128bit):
527  * Interrupt host address for Char driver
528  */
529 #define FR_AZ_INT_ADR_REG_CHAR_OFST 0x00000040
530 /* falconb0,sienaa0=net_func_bar2,falcona0=char_func_bar0 */

532 #define FRF_AZ_NORM_INT_VEC_DIS_CHAR_LBN 64
533 #define FRF_AZ_NORM_INT_VEC_DIS_CHAR_WIDTH 1
534 #define FRF_AZ_INT_ADR_CHAR_LBN 0
535 #define FRF_AZ_INT_ADR_CHAR_WIDTH 64
536 #define FRF_AZ_INT_ADR_CHAR_DW0_LBN 0
537 #define FRF_AZ_INT_ADR_CHAR_DW0_WIDTH 32
538 #define FRF_AZ_INT_ADR_CHAR_DW1_LBN 32
539 #define FRF_AZ_INT_ADR_CHAR_DW1_WIDTH 32


542 /*
543  * FR_AA_INT_ACK_KER(32bit):
544  * Kernel interrupt acknowledge register
545  */
546 #define FR_AA_INT_ACK_KER_OFST 0x00000050
547 /* falcona0=net_func_bar2 */

549 #define FRF_AA_INT_ACK_KER_FIELD_LBN 0
550 #define FRF_AA_INT_ACK_KER_FIELD_WIDTH 32


553 /*
554  * FR_BZ_INT_ISR0_REG(128bit):
555  * Function 0 Interrupt Acknowlege Status register
556  */
557 #define FR_BZ_INT_ISR0_REG_OFST 0x00000090
558 /* falconb0,sienaa0=net_func_bar2 */

560 #define FRF_BZ_INT_ISR_REG_LBN 0
561 #define FRF_BZ_INT_ISR_REG_WIDTH 64
562 #define FRF_BZ_INT_ISR_REG_DW0_LBN 0
563 #define FRF_BZ_INT_ISR_REG_DW0_WIDTH 32
564 #define FRF_BZ_INT_ISR_REG_DW1_LBN 32
565 #define FRF_BZ_INT_ISR_REG_DW1_WIDTH 32


568 /*
569  * FR_AB_EE_SPI_HCMD_REG(128bit):
570  * SPI host command register
571  */
572 #define FR_AB_EE_SPI_HCMD_REG_OFST 0x00000100
573 /* falcona0,falconb0=net_func_bar2,falcona0=char_func_bar0 */

575 #define FRF_AB_EE_SPI_HCMD_CMD_EN_LBN 31
576 #define FRF_AB_EE_SPI_HCMD_CMD_EN_WIDTH 1
577 #define FRF_AB_EE_WR_TIMER_ACTIVE_LBN 28
578 #define FRF_AB_EE_WR_TIMER_ACTIVE_WIDTH 1
579 #define FRF_AB_EE_SPI_HCMD_SF_SEL_LBN 24
580 #define FRF_AB_EE_SPI_HCMD_SF_SEL_WIDTH 1
581 #define FRF_AB_EE_SPI_HCMD_DABCNT_LBN 16
582 #define FRF_AB_EE_SPI_HCMD_DABCNT_WIDTH 5
583 #define FRF_AB_EE_SPI_HCMD_READ_LBN 15
584 #define FRF_AB_EE_SPI_HCMD_READ_WIDTH 1
585 #define FRF_AB_EE_SPI_HCMD_DUBCNT_LBN 12
586 #define FRF_AB_EE_SPI_HCMD_DUBCNT_WIDTH 2
587 #define FRF_AB_EE_SPI_HCMD_ADBCNT_LBN 8
588 #define FRF_AB_EE_SPI_HCMD_ADBCNT_WIDTH 2
589 #define FRF_AB_EE_SPI_HCMD_ENC_LBN 0
```



```
590 #define FRF_AB_EE_SPI_HCMD_ENC_WIDTH 8


593 /*
594  * FR_CZ_USR_EV_CFG(32bit):
595  * User Level Event Configuration register
596  */
597 #define FR_CZ_USR_EV_CFG_OFST 0x00000100
598 /* sienaa0=net_func_bar2 */

600 #define FRF_CZ_USREV_DIS_LBN 16
601 #define FRF_CZ_USREV_DIS_WIDTH 1
602 #define FRF_CZ_DFLT_EVQ_LBN 0
603 #define FRF_CZ_DFLT_EVQ_WIDTH 10


606 /*
607  * FR_AB_EE_SPI_HADR_REG(128bit):
608  * SPI host address register
609  */
610 #define FR_AB_EE_SPI_HADR_REG_OFST 0x00000110
611 /* falcona0,falconb0=net_func_bar2,falcona0=char_func_bar0 */

613 #define FRF_AB_EE_SPI_HADR_DUBYTE_LBN 24
614 #define FRF_AB_EE_SPI_HADR_DUBYTE_WIDTH 8
615 #define FRF_AB_EE_SPI_HADR_ADR_LBN 0
616 #define FRF_AB_EE_SPI_HADR_ADR_WIDTH 24


619 /*
620  * FR_AB_EE_SPI_HDATA_REG(128bit):
621  * SPI host data register
622  */
623 #define FR_AB_EE_SPI_HDATA_REG_OFST 0x00000120
624 /* falcona0,falconb0=net_func_bar2,falcona0=char_func_bar0 */

626 #define FRF_AB_EE_SPI_HDATA3_LBN 96
627 #define FRF_AB_EE_SPI_HDATA3_WIDTH 32
628 #define FRF_AB_EE_SPI_HDATA2_LBN 64
629 #define FRF_AB_EE_SPI_HDATA2_WIDTH 32
630 #define FRF_AB_EE_SPI_HDATA1_LBN 32
631 #define FRF_AB_EE_SPI_HDATA1_WIDTH 32
632 #define FRF_AB_EE_SPI_HDATA0_LBN 0
633 #define FRF_AB_EE_SPI_HDATA0_WIDTH 32


636 /*
637  * FR_AB_EE_BASE_PAGE_REG(128bit):
638  * Expansion ROM base mirror register
639  */
640 #define FR_AB_EE_BASE_PAGE_REG_OFST 0x00000130
641 /* falcona0,falconb0=net_func_bar2,falcona0=char_func_bar0 */

643 #define FRF_AB_EE_EXPROM_MASK_LBN 16
644 #define FRF_AB_EE_EXPROM_MASK_WIDTH 13
645 #define FRF_AB_EE_EXP_ROM_WINDOW_BASE_LBN 0
646 #define FRF_AB_EE_EXP_ROM_WINDOW_BASE_WIDTH 13


649 /*
650  * FR_AB_EE_VPD_SW_CNTL_REG(128bit):
651  * VPD access SW control register
652  */
653 #define FR_AB_EE_VPD_SW_CNTL_REG_OFST 0x00000150
654 /* falcona0,falconb0=net_func_bar2,falcona0=char_func_bar0 */
```

```
#define FRF_AB_EE_VPD_CYCLE_PENDING_LBN 31
#define FRF_AB_EE_VPD_CYCLE_PENDING_WIDTH 1
#define FRF_AB_EE_VPD_CYC_WRITE_LBN 28
#define FRF_AB_EE_VPD_CYC_WRITE_WIDTH 1
#define FRF_AB_EE_VPD_CYC_ADR_LBN 0
#define FRF_AB_EE_VPD_CYC_ADR_WIDTH 15


/*
 * FR_AB_EE_VPD_SW_DATA_REG(128bit):
 * VPD access SW data register
 */
#define FR_AB_EE_VPD_SW_DATA_REG_OFST 0x00000160
/* falcona0,falconb0=net_func_bar2,falcona0=char_func_bar0 */

#define FRF_AB_EE_VPD_CYC_DAT_LBN 0
#define FRF_AB_EE_VPD_CYC_DAT_WIDTH 32


/*
 * FR_BB_PCIE_CORE_INDIRECT_REG(64bit):
 * Indirect Access to PCIE Core registers
 */
#define FR_BB_PCIE_CORE_INDIRECT_REG_OFST 0x000001f0
/* falconb0=net_func_bar2 */

#define FRF_BB_PCIE_CORE_TARGET_DATA_LBN 32
#define FRF_BB_PCIE_CORE_TARGET_DATA_WIDTH 32
#define FRF_BB_PCIE_CORE_INDIRECT_ACCESS_DIR_LBN 15
#define FRF_BB_PCIE_CORE_INDIRECT_ACCESS_DIR_WIDTH 1
#define FRF_BB_PCIE_CORE_TARGET_REG_ADRS_LBN 0
#define FRF_BB_PCIE_CORE_TARGET_REG_ADRS_WIDTH 12


/*
 * FR_AB_GPIO_CTL_REG(128bit):
 * GPIO control register
 */
#define FR_AB_GPIO_CTL_REG_OFST 0x00000210
/* falcona0,falconb0=net_func_bar2,falcona0=char_func_bar0 */

#define FRF_AB_GPIO15_OEN_LBN 63
#define FRF_AB_GPIO15_OEN_WIDTH 1
#define FRF_AB_GPIO14_OEN_LBN 62
#define FRF_AB_GPIO14_OEN_WIDTH 1
#define FRF_AB_GPIO13_OEN_LBN 61
#define FRF_AB_GPIO13_OEN_WIDTH 1
#define FRF_AB_GPIO12_OEN_LBN 60
#define FRF_AB_GPIO12_OEN_WIDTH 1
#define FRF_AB_GPIO11_OEN_LBN 59
#define FRF_AB_GPIO11_OEN_WIDTH 1
#define FRF_AB_GPIO10_OEN_LBN 58
#define FRF_AB_GPIO10_OEN_WIDTH 1
#define FRF_AB_GPIO9_OEN_LBN 57
#define FRF_AB_GPIO9_OEN_WIDTH 1
#define FRF_AB_GPIO8_OEN_LBN 56
#define FRF_AB_GPIO8_OEN_WIDTH 1
#define FRF_AB_GPIO15_OUT_LBN 55
#define FRF_AB_GPIO15_OUT_WIDTH 1
#define FRF_AB_GPIO14_OUT_LBN 54
#define FRF_AB_GPIO14_OUT_WIDTH 1
#define FRF_AB_GPIO13_OUT_LBN 53
#define FRF_AB_GPIO13_OUT_WIDTH 1
#define FRF_AB_GPIO12_OUT_LBN 52
#define FRF_AB_GPIO12_OUT_WIDTH 1
#define FRF_AB_GPIO11_OUT_LBN 51
```



```
#define FRF_AB_GPIO11_OUT_WIDTH 1
#define FRF_AB_GPIO10_OUT_LBN 50
#define FRF_AB_GPIO10_OUT_WIDTH 1
#define FRF_AB_GPIO9_OUT_LBN 49
#define FRF_AB_GPIO9_OUT_WIDTH 1
#define FRF_AB_GPIO8_OUT_LBN 48
#define FRF_AB_GPIO8_OUT_WIDTH 1
#define FRF_AB_GPIO15_IN_LBN 47
#define FRF_AB_GPIO15_IN_WIDTH 1
#define FRF_AB_GPIO14_IN_LBN 46
#define FRF_AB_GPIO14_IN_WIDTH 1
#define FRF_AB_GPIO13_IN_LBN 45
#define FRF_AB_GPIO13_IN_WIDTH 1
#define FRF_AB_GPIO12_IN_LBN 44
#define FRF_AB_GPIO12_IN_WIDTH 1
#define FRF_AB_GPIO11_IN_LBN 43
#define FRF_AB_GPIO11_IN_WIDTH 1
#define FRF_AB_GPIO10_IN_LBN 42
#define FRF_AB_GPIO10_IN_WIDTH 1
#define FRF_AB_GPIO9_IN_LBN 41
#define FRF_AB_GPIO9_IN_WIDTH 1
#define FRF_AB_GPIO8_IN_LBN 40
#define FRF_AB_GPIO8_IN_WIDTH 1
#define FRF_AB_GPIO15_PWRUP_VALUE_LBN 39
#define FRF_AB_GPIO15_PWRUP_VALUE_WIDTH 1
#define FRF_AB_GPIO14_PWRUP_VALUE_LBN 38
#define FRF_AB_GPIO14_PWRUP_VALUE_WIDTH 1
#define FRF_AB_GPIO13_PWRUP_VALUE_LBN 37
#define FRF_AB_GPIO13_PWRUP_VALUE_WIDTH 1
#define FRF_AB_GPIO12_PWRUP_VALUE_LBN 36
#define FRF_AB_GPIO12_PWRUP_VALUE_WIDTH 1
#define FRF_AB_GPIO11_PWRUP_VALUE_LBN 35
#define FRF_AB_GPIO11_PWRUP_VALUE_WIDTH 1
#define FRF_AB_GPIO10_PWRUP_VALUE_LBN 34
#define FRF_AB_GPIO10_PWRUP_VALUE_WIDTH 1
#define FRF_AB_GPIO9_PWRUP_VALUE_LBN 33
#define FRF_AB_GPIO9_PWRUP_VALUE_WIDTH 1
#define FRF_AB_GPIO8_PWRUP_VALUE_LBN 32
#define FRF_AB_GPIO8_PWRUP_VALUE_WIDTH 1
#define FRF_BB_CLK156_OUT_EN_LBN 31
#define FRF_BB_CLK156_OUT_EN_WIDTH 1
#define FRF_BB_USE_NIC_CLK_LBN 30
#define FRF_BB_USE_NIC_CLK_WIDTH 1
#define FRF_AB_GPIO5_OEN_LBN 29
#define FRF_AB_GPIO5_OEN_WIDTH 1
#define FRF_AB_GPIO4_OEN_LBN 28
#define FRF_AB_GPIO4_OEN_WIDTH 1
#define FRF_AB_GPIO3_OEN_LBN 27
#define FRF_AB_GPIO3_OEN_WIDTH 1
#define FRF_AB_GPIO2_OEN_LBN 26
#define FRF_AB_GPIO2_OEN_WIDTH 1
#define FRF_AB_GPIO1_OEN_LBN 25
#define FRF_AB_GPIO1_OEN_WIDTH 1
#define FRF_AB_GPIO0_OEN_LBN 24
#define FRF_AB_GPIO0_OEN_WIDTH 1
#define FRF_AB_GPIO5_OUT_LBN 21
#define FRF_AB_GPIO5_OUT_WIDTH 1
#define FRF_AB_GPIO4_OUT_LBN 20
#define FRF_AB_GPIO4_OUT_WIDTH 1
#define FRF_AB_GPIO3_OUT_LBN 19
#define FRF_AB_GPIO3_OUT_WIDTH 1
#define FRF_AB_GPIO2_OUT_LBN 18
#define FRF_AB_GPIO2_OUT_WIDTH 1
#define FRF_AB_GPIO1_OUT_LBN 17
#define FRF_AB_GPIO1_OUT_WIDTH 1
#define FRF_AB_GPIO0_OUT_LBN 16
```

```
#define FRF_AB_GPIO0_OUT_WIDTH 1
#define FRF_AB_GPIO5_IN_LBN 13
#define FRF_AB_GPIO5_IN_WIDTH 1
#define FRF_AB_GPIO4_IN_LBN 12
#define FRF_AB_GPIO4_IN_WIDTH 1
#define FRF_AB_GPIO3_IN_LBN 11
#define FRF_AB_GPIO3_IN_WIDTH 1
#define FRF_AB_GPIO2_IN_LBN 10
#define FRF_AB_GPIO2_IN_WIDTH 1
#define FRF_AB_GPIO1_IN_LBN 9
#define FRF_AB_GPIO1_IN_WIDTH 1
#define FRF_AB_GPIO0_IN_LBN 8
#define FRF_AB_GPIO0_IN_WIDTH 1
#define FRF_AB_GPIO5_PWRUP_VALUE_LBN 5
#define FRF_AB_GPIO5_PWRUP_VALUE_WIDTH 1
#define FRF_AB_GPIO4_PWRUP_VALUE_LBN 4
#define FRF_AB_GPIO4_PWRUP_VALUE_WIDTH 1
#define FRF_AB_GPIO3_PWRUP_VALUE_LBN 3
#define FRF_AB_GPIO3_PWRUP_VALUE_WIDTH 1
#define FRF_AB_GPIO2_PWRUP_VALUE_LBN 2
#define FRF_AB_GPIO2_PWRUP_VALUE_WIDTH 1
#define FRF_AB_GPIO1_PWRUP_VALUE_LBN 1
#define FRF_AB_GPIO1_PWRUP_VALUE_WIDTH 1
#define FRF_AB_GPIO0_PWRUP_VALUE_LBN 0
#define FRF_AB_GPIO0_PWRUP_VALUE_WIDTH 1


/*
 * FR_AZ_FATAL_INTR_REG_KER(128bit):
 * Fatal interrupt register for Kernel
 */
#define FR_AZ_FATAL_INTR_REG_KER_OFST 0x00000230
/* falcona0,falconb0,sienaa0=net_func_bar2 */

#define FRF_CZ_SRAM_PERR_INT_P_KER_EN_LBN 44
#define FRF_CZ_SRAM_PERR_INT_P_KER_EN_WIDTH 1
#define FRF_AB_PCI_BUSERR_INT_KER_EN_LBN 43
#define FRF_AB_PCI_BUSERR_INT_KER_EN_WIDTH 1
#define FRF_CZ_MBU_PERR_INT_KER_EN_LBN 43
#define FRF_CZ_MBU_PERR_INT_KER_EN_WIDTH 1
#define FRF_AZ_SRAM_OOB_INT_KER_EN_LBN 42
#define FRF_AZ_SRAM_OOB_INT_KER_EN_WIDTH 1
#define FRF_AZ_BUFID_OOB_INT_KER_EN_LBN 41
#define FRF_AZ_BUFID_OOB_INT_KER_EN_WIDTH 1
#define FRF_AZ_MEM_PERR_INT_KER_EN_LBN 40
#define FRF_AZ_MEM_PERR_INT_KER_EN_WIDTH 1
#define FRF_AZ_RBUF_OWN_INT_KER_EN_LBN 39
#define FRF_AZ_RBUF_OWN_INT_KER_EN_WIDTH 1
#define FRF_AZ_TBUF_OWN_INT_KER_EN_LBN 38
#define FRF_AZ_TBUF_OWN_INT_KER_EN_WIDTH 1
#define FRF_AZ_RDESCQ_OWN_INT_KER_EN_LBN 37
#define FRF_AZ_RDESCQ_OWN_INT_KER_EN_WIDTH 1
#define FRF_AZ_TDESCQ_OWN_INT_KER_EN_LBN 36
#define FRF_AZ_TDESCQ_OWN_INT_KER_EN_WIDTH 1
#define FRF_AZ_EVQ_OWN_INT_KER_EN_LBN 35
#define FRF_AZ_EVQ_OWN_INT_KER_EN_WIDTH 1
#define FRF_AZ_EVF_OFLO_INT_KER_EN_LBN 34
#define FRF_AZ_EVF_OFLO_INT_KER_EN_WIDTH 1
#define FRF_AZ_ILL_ADR_INT_KER_EN_LBN 33
#define FRF_AZ_ILL_ADR_INT_KER_EN_WIDTH 1
#define FRF_AZ_SRM_PERR_INT_KER_EN_LBN 32
#define FRF_AZ_SRM_PERR_INT_KER_EN_WIDTH 1
#define FRF_CZ_SRAM_PERR_INT_P_KER_LBN 12
#define FRF_CZ_SRAM_PERR_INT_P_KER_WIDTH 1
#define FRF_AB_PCI_BUSERR_INT_KER_LBN 11
#define FRF_AB_PCI_BUSERR_INT_KER_WIDTH 1
```



```
#define FRF_CZ_MBU_PERR_INT_KER_LBN 11
#define FRF_CZ_MBU_PERR_INT_KER_WIDTH 1
#define FRF_AZ_SRAM_OOB_INT_KER_LBN 10
#define FRF_AZ_SRAM_OOB_INT_KER_WIDTH 1
#define FRF_AZ_BUFID_DC_OOB_INT_KER_LBN 9
#define FRF_AZ_BUFID_DC_OOB_INT_KER_WIDTH 1
#define FRF_AZ_MEM_PERR_INT_KER_LBN 8
#define FRF_AZ_MEM_PERR_INT_KER_WIDTH 1
#define FRF_AZ_RBUF_OWN_INT_KER_LBN 7
#define FRF_AZ_RBUF_OWN_INT_KER_WIDTH 1
#define FRF_AZ_TBUF_OWN_INT_KER_LBN 6
#define FRF_AZ_TBUF_OWN_INT_KER_WIDTH 1
#define FRF_AZ_RDESCQ_OWN_INT_KER_LBN 5
#define FRF_AZ_RDESCQ_OWN_INT_KER_WIDTH 1
#define FRF_AZ_TDESCQ_OWN_INT_KER_LBN 4
#define FRF_AZ_TDESCQ_OWN_INT_KER_WIDTH 1
#define FRF_AZ_EVQ_OWN_INT_KER_LBN 3
#define FRF_AZ_EVQ_OWN_INT_KER_WIDTH 1
#define FRF_AZ_EVF_OFLO_INT_KER_LBN 2
#define FRF_AZ_EVF_OFLO_INT_KER_WIDTH 1
#define FRF_AZ_ILL_ADR_INT_KER_LBN 1
#define FRF_AZ_ILL_ADR_INT_KER_WIDTH 1
#define FRF_AZ_SRM_PERR_INT_KER_LBN 0
#define FRF_AZ_SRM_PERR_INT_KER_WIDTH 1


/*
 * FR_AZ_FATAL_INTR_REG_CHAR(128bit):
 * Fatal interrupt register for Char
 */
#define FR_AZ_FATAL_INTR_REG_CHAR_OFST 0x00000240
/* falconb0,sienaa0=net_func_bar2,falcona0=char_func_bar0 */

#define FRF_CZ_SRAM_PERR_INT_P_CHAR_EN_LBN 44
#define FRF_CZ_SRAM_PERR_INT_P_CHAR_EN_WIDTH 1
#define FRF_AB_PCI_BUSERR_INT_CHAR_EN_LBN 43
#define FRF_AB_PCI_BUSERR_INT_CHAR_EN_WIDTH 1
#define FRF_CZ_MBU_PERR_INT_CHAR_EN_LBN 43
#define FRF_CZ_MBU_PERR_INT_CHAR_EN_WIDTH 1
#define FRF_AZ_SRAM_OOB_INT_CHAR_EN_LBN 42
#define FRF_AZ_SRAM_OOB_INT_CHAR_EN_WIDTH 1
#define FRF_AZ_BUFID_OOB_INT_CHAR_EN_LBN 41
#define FRF_AZ_BUFID_OOB_INT_CHAR_EN_WIDTH 1
#define FRF_AZ_MEM_PERR_INT_CHAR_EN_LBN 40
#define FRF_AZ_MEM_PERR_INT_CHAR_EN_WIDTH 1
#define FRF_AZ_RBUF_OWN_INT_CHAR_EN_LBN 39
#define FRF_AZ_RBUF_OWN_INT_CHAR_EN_WIDTH 1
#define FRF_AZ_TBUF_OWN_INT_CHAR_EN_LBN 38
#define FRF_AZ_TBUF_OWN_INT_CHAR_EN_WIDTH 1
#define FRF_AZ_RDESCQ_OWN_INT_CHAR_EN_LBN 37
#define FRF_AZ_RDESCQ_OWN_INT_CHAR_EN_WIDTH 1
#define FRF_AZ_TDESCQ_OWN_INT_CHAR_EN_LBN 36
#define FRF_AZ_TDESCQ_OWN_INT_CHAR_EN_WIDTH 1
#define FRF_AZ_EVQ_OWN_INT_CHAR_EN_LBN 35
#define FRF_AZ_EVQ_OWN_INT_CHAR_EN_WIDTH 1
#define FRF_AZ_EVF_OFLO_INT_CHAR_EN_LBN 34
#define FRF_AZ_EVF_OFLO_INT_CHAR_EN_WIDTH 1
#define FRF_AZ_ILL_ADR_INT_CHAR_EN_LBN 33
#define FRF_AZ_ILL_ADR_INT_CHAR_EN_WIDTH 1
#define FRF_AZ_SRM_PERR_INT_CHAR_EN_LBN 32
#define FRF_AZ_SRM_PERR_INT_CHAR_EN_WIDTH 1
#define FRF_CZ_SRAM_PERR_INT_P_CHAR_LBN 12
#define FRF_CZ_SRAM_PERR_INT_P_CHAR_WIDTH 1
#define FRF_AB_PCI_BUSERR_INT_CHAR_LBN 11
#define FRF_AB_PCI_BUSERR_INT_CHAR_WIDTH 1
#define FRF_CZ_MBU_PERR_INT_CHAR_LBN 11
```

```
#define FRF_CZ_MBU_PERR_INT_CHAR_WIDTH 1
#define FRF_AZ_SRAM_OOB_INT_CHAR_LBN 10
#define FRF_AZ_SRAM_OOB_INT_CHAR_WIDTH 1
#define FRF_AZ_BUFID_DC_OOB_INT_CHAR_LBN 9
#define FRF_AZ_BUFID_DC_OOB_INT_CHAR_WIDTH 1
#define FRF_AZ_MEM_PERR_INT_CHAR_LBN 8
#define FRF_AZ_MEM_PERR_INT_CHAR_WIDTH 1
#define FRF_AZ_RBUF_OWN_INT_CHAR_LBN 7
#define FRF_AZ_RBUF_OWN_INT_CHAR_WIDTH 1
#define FRF_AZ_TBUF_OWN_INT_CHAR_LBN 6
#define FRF_AZ_TBUF_OWN_INT_CHAR_WIDTH 1
#define FRF_AZ_RDESCQ_OWN_INT_CHAR_LBN 5
#define FRF_AZ_RDESCQ_OWN_INT_CHAR_WIDTH 1
#define FRF_AZ_TDESCQ_OWN_INT_CHAR_LBN 4
#define FRF_AZ_TDESCQ_OWN_INT_CHAR_WIDTH 1
#define FRF_AZ_EVQ_OWN_INT_CHAR_LBN 3
#define FRF_AZ_EVQ_OWN_INT_CHAR_WIDTH 1
#define FRF_AZ_EVF_OFLO_INT_CHAR_LBN 2
#define FRF_AZ_EVF_OFLO_INT_CHAR_WIDTH 1
#define FRF_AZ_ILL_ADR_INT_CHAR_LBN 1
#define FRF_AZ_ILL_ADR_INT_CHAR_WIDTH 1
#define FRF_AZ_SRM_PERR_INT_CHAR_LBN 0
#define FRF_AZ_SRM_PERR_INT_CHAR_WIDTH 1


/*
 * FR_AZ_DP_CTRL_REG(128bit):
 * Datapath control register
 */
#define FR_AZ_DP_CTRL_REG_OFST 0x00000250
/* falconb0,sienaa0=net_func_bar2,falcona0=char_func_bar0 */

#define FRF_AZ_FLS_EVQ_ID_LBN 0
#define FRF_AZ_FLS_EVQ_ID_WIDTH 12


/*
 * FR_AZ_MEM_STAT_REG(128bit):
 * Memory status register
 */
#define FR_AZ_MEM_STAT_REG_OFST 0x00000260
/* falcona0,falconb0,sienaa0=net_func_bar2,falcona0=char_func_bar0 */

#define FRF_AB_MEM_PERR_VEC_LBN 53
#define FRF_AB_MEM_PERR_VEC_WIDTH 40
#define FRF_AB_MEM_PERR_VEC_DW0_LBN 53
#define FRF_AB_MEM_PERR_VEC_DW0_WIDTH 32
#define FRF_AB_MEM_PERR_VEC_DW1_LBN 85
#define FRF_AB_MEM_PERR_VEC_DW1_WIDTH 6
#define FRF_AB_MBIST_CORR_LBN 38
#define FRF_AB_MBIST_CORR_WIDTH 15
#define FRF_AB_MBIST_ERR_LBN 0
#define FRF_AB_MBIST_ERR_WIDTH 40
#define FRF_AB_MBIST_ERR_DW0_LBN 0
#define FRF_AB_MBIST_ERR_DW0_WIDTH 32
#define FRF_AB_MBIST_ERR_DW1_LBN 32
#define FRF_AB_MBIST_ERR_DW1_WIDTH 6
#define FRF_CZ_MEM_PERR_VEC_LBN 0
#define FRF_CZ_MEM_PERR_VEC_WIDTH 35
#define FRF_CZ_MEM_PERR_VEC_DW0_LBN 0
#define FRF_CZ_MEM_PERR_VEC_DW0_WIDTH 32
#define FRF_CZ_MEM_PERR_VEC_DW1_LBN 32
#define FRF_CZ_MEM_PERR_VEC_DW1_WIDTH 3


/*
```



```
 * FR_PORT0_CS_DEBUG_REG(128bit):
 * Debug register
 */

#define FR_AZ_CS_DEBUG_REG_OFST 0x00000270
/* falcona0,falconb0,sienaa0=net_func_bar2,falcona0=char_func_bar0 */

#define FRF_AB_GLB_DEBUG2_SEL_LBN 50
#define FRF_AB_GLB_DEBUG2_SEL_WIDTH 3
#define FRF_AB_DEBUG_BLK_SEL2_LBN 47
#define FRF_AB_DEBUG_BLK_SEL2_WIDTH 3
#define FRF_AB_DEBUG_BLK_SEL1_LBN 44
#define FRF_AB_DEBUG_BLK_SEL1_WIDTH 3
#define FRF_AB_DEBUG_BLK_SEL0_LBN 41
#define FRF_AB_DEBUG_BLK_SEL0_WIDTH 3
#define FRF_CZ_CS_PORT_NUM_LBN 40
#define FRF_CZ_CS_PORT_NUM_WIDTH 2
#define FRF_AB_MISC_DEBUG_ADDR_LBN 36
#define FRF_AB_MISC_DEBUG_ADDR_WIDTH 5
#define FRF_CZ_CS_RESERVED_LBN 36
#define FRF_CZ_CS_RESERVED_WIDTH 4
#define FRF_AB_SERDES_DEBUG_ADDR_LBN 31
#define FRF_AB_SERDES_DEBUG_ADDR_WIDTH 5
#define FRF_CZ_CS_PORT_FPE_DW0_LBN 1
#define FRF_CZ_CS_PORT_FPE_DW0_WIDTH 32
#define FRF_CZ_CS_PORT_FPE_DW1_LBN 33
#define FRF_CZ_CS_PORT_FPE_DW1_WIDTH 3
#define FRF_CZ_CS_PORT_FPE_LBN 1
#define FRF_CZ_CS_PORT_FPE_WIDTH 35
#define FRF_AB_EM_DEBUG_ADDR_LBN 26
#define FRF_AB_EM_DEBUG_ADDR_WIDTH 5
#define FRF_AB_SR_DEBUG_ADDR_LBN 21
#define FRF_AB_SR_DEBUG_ADDR_WIDTH 5
#define FRF_AB_EV_DEBUG_ADDR_LBN 16
#define FRF_AB_EV_DEBUG_ADDR_WIDTH 5
#define FRF_AB_RX_DEBUG_ADDR_LBN 11
#define FRF_AB_RX_DEBUG_ADDR_WIDTH 5
#define FRF_AB_TX_DEBUG_ADDR_LBN 6
#define FRF_AB_TX_DEBUG_ADDR_WIDTH 5
#define FRF_AB_CS_BIU_DEBUG_ADDR_LBN 1
#define FRF_AB_CS_BIU_DEBUG_ADDR_WIDTH 5
#define FRF_AZ_CS_DEBUG_EN_LBN 0
#define FRF_AZ_CS_DEBUG_EN_WIDTH 1


/*
 * FR_AZ_DRIVER_REG(128bit):
 * Driver scratch register [0-7]
 */
#define FR_AZ_DRIVER_REG_OFST 0x00000280
/* falcona0,falconb0,sienaa0=net_func_bar2,falcona0=char_func_bar0 */
#define FR_AZ_DRIVER_REG_STEP 16
#define FR_AZ_DRIVER_REG_ROWS 8

#define FRF_AZ_DRIVER_DW0_LBN 0
#define FRF_AZ_DRIVER_DW0_WIDTH 32


/*
 * FR_AZ_ALTERA_BUILD_REG(128bit):
 * Altera build register
 */
#define FR_AZ_ALTERA_BUILD_REG_OFST 0x00000300
/* falcona0,falconb0,sienaa0=net_func_bar2,falcona0=char_func_bar0 */

#define FRF_AZ_ALTERA_BUILD_VER_LBN 0
```

```
#define FRF_AZ_ALTERA_BUILD_VER_WIDTH 32


/*
 * FR_AZ_CSR_SPARE_REG(128bit):
 * Spare register
 */
#define FR_AZ_CSR_SPARE_REG_OFST 0x00000310
/* falcona0,falconb0,sienaa0=net_func_bar2,falcona0=char_func_bar0 */

#define FRF_AZ_MEM_PERR_EN_TX_DATA_LBN 72
#define FRF_AZ_MEM_PERR_EN_TX_DATA_WIDTH 2
#define FRF_AZ_MEM_PERR_EN_LBN 64
#define FRF_AZ_MEM_PERR_EN_WIDTH 38
#define FRF_AZ_MEM_PERR_EN_DW0_LBN 64
#define FRF_AZ_MEM_PERR_EN_DW0_WIDTH 32
#define FRF_AZ_MEM_PERR_EN_DW1_LBN 96
#define FRF_AZ_MEM_PERR_EN_DW1_WIDTH 6
#define FRF_AZ_CSR_SPARE_BITS_LBN 0
#define FRF_AZ_CSR_SPARE_BITS_WIDTH 32


/*
 * FR_BZ_DEBUG_DATA_OUT_REG(128bit):
 * Live Debug and Debug 2 out ports
 */
#define FR_BZ_DEBUG_DATA_OUT_REG_OFST 0x00000350
/* falconb0,sienaa0=net_func_bar2 */

#define FRF_BZ_DEBUG2_PORT_LBN 25
#define FRF_BZ_DEBUG2_PORT_WIDTH 15
#define FRF_BZ_DEBUG1_PORT_LBN 0
#define FRF_BZ_DEBUG1_PORT_WIDTH 25


/*
 * FR_BZ_EVQ_RPTR_REGP0(32bit):
 * Event queue read pointer register
 */
#define FR_BZ_EVQ_RPTR_REGP0_OFST 0x00000400
/* falconb0,sienaa0=net_func_bar2 */
#define FR_BZ_EVQ_RPTR_REGP0_STEP 8192
#define FR_BZ_EVQ_RPTR_REGP0_ROWS 1024
/*
 * FR_AA_EVQ_RPTR_REG_KER(32bit):
 * Event queue read pointer register
 */
#define FR_AA_EVQ_RPTR_REG_KER_OFST 0x00011b00
/* falcona0=net_func_bar2 */
#define FR_AA_EVQ_RPTR_REG_KER_STEP 4
#define FR_AA_EVQ_RPTR_REG_KER_ROWS 4
/*
 * FR_AZ_EVQ_RPTR_REG(32bit):
 * Event queue read pointer register
 */
#define FR_AZ_EVQ_RPTR_REG_OFST 0x00fa0000
/* falconb0=net_func_bar2,sienaa0=net_func_bar2,falcona0=char_func_bar0 */
#define FR_AZ_EVQ_RPTR_REG_STEP 16
#define FR_AB_EVQ_RPTR_REG_ROWS 4096
#define FR_CZ_EVQ_RPTR_REG_ROWS 1024
/*
 * FR_BB_EVQ_RPTR_REGP123(32bit):
 * Event queue read pointer register
 */
#define FR_BB_EVQ_RPTR_REGP123_OFST 0x01000400
/* falconb0=net_func_bar2 */
```



```
#define FR_BB_EVQ_RPTR_REGP123_STEP 8192
#define FR_BB_EVQ_RPTR_REGP123_ROWS 3072

#define FRF_AZ_EVQ_RPTR_VLD_LBN 15
#define FRF_AZ_EVQ_RPTR_VLD_WIDTH 1
#define FRF_AZ_EVQ_RPTR_LBN 0
#define FRF_AZ_EVQ_RPTR_WIDTH 15


/*
 * FR_BZ_TIMER_COMMAND_REGP0(128bit):
 * Timer Command Registers
 */
#define FR_BZ_TIMER_COMMAND_REGP0_OFST 0x00000420
/* falconb0,sienaa0=net_func_bar2 */
#define FR_BZ_TIMER_COMMAND_REGP0_STEP 8192
#define FR_BZ_TIMER_COMMAND_REGP0_ROWS 1024
/*
 * FR_AA_TIMER_COMMAND_REG_KER(128bit):
 * Timer Command Registers
 */
#define FR_AA_TIMER_COMMAND_REG_KER_OFST 0x00000420
/* falcona0=net_func_bar2 */
#define FR_AA_TIMER_COMMAND_REG_KER_STEP 8192
#define FR_AA_TIMER_COMMAND_REG_KER_ROWS 4
/*
 * FR_AB_TIMER_COMMAND_REGP123(128bit):
 * Timer Command Registers
 */
#define FR_AB_TIMER_COMMAND_REGP123_OFST 0x01000420
/* falconb0=net_func_bar2,falcona0=char_func_bar0 */
#define FR_AB_TIMER_COMMAND_REGP123_STEP 8192
#define FR_AB_TIMER_COMMAND_REGP123_ROWS 3072
/*
 * FR_AA_TIMER_COMMAND_REGP0(128bit):
 * Timer Command Registers
 */
#define FR_AA_TIMER_COMMAND_REGP0_OFST 0x00008420
/* falcona0=char_func_bar0 */
#define FR_AA_TIMER_COMMAND_REGP0_STEP 8192
#define FR_AA_TIMER_COMMAND_REGP0_ROWS 1020

#define FRF_CZ_TC_TIMER_MODE_LBN 14
#define FRF_CZ_TC_TIMER_MODE_WIDTH 2
#define FRF_AB_TC_TIMER_MODE_LBN 12
#define FRF_AB_TC_TIMER_MODE_WIDTH 2
#define FRF_CZ_TC_TIMER_VAL_LBN 0
#define FRF_CZ_TC_TIMER_VAL_WIDTH 14
#define FRF_AB_TC_TIMER_VAL_LBN 0
#define FRF_AB_TC_TIMER_VAL_WIDTH 12


/*
 * FR_AZ_DRV_EV_REG(128bit):
 * Driver generated event register
 */
#define FR_AZ_DRV_EV_REG_OFST 0x00000440
/* falcona0,falconb0,sienaa0=net_func_bar2,falcona0=char_func_bar0 */

#define FRF_AZ_DRV_EV_QID_LBN 64
#define FRF_AZ_DRV_EV_QID_WIDTH 12
#define FRF_AZ_DRV_EV_DATA_LBN 0
#define FRF_AZ_DRV_EV_DATA_WIDTH 64
#define FRF_AZ_DRV_EV_DATA_DW0_LBN 0
#define FRF_AZ_DRV_EV_DATA_DW0_WIDTH 32
#define FRF_AZ_DRV_EV_DATA_DW1_LBN 32
```

```
#define FRF_AZ_DRV_EV_DATA_DW1_WIDTH 32


/*
 * FR_AZ_EVQ_CTL_REG(128bit):
 * Event queue control register
 */
#define FR_AZ_EVQ_CTL_REG_OFST 0x00000450
/* falcona0,falconb0,sienaa0=net_func_bar2,falcona0=char_func_bar0 */

#define FRF_CZ_RX_EVQ_WAKEUP_MASK_LBN 15
#define FRF_CZ_RX_EVQ_WAKEUP_MASK_WIDTH 10
#define FRF_BB_RX_EVQ_WAKEUP_MASK_LBN 15
#define FRF_BB_RX_EVQ_WAKEUP_MASK_WIDTH 6
#define FRF_AZ_EVQ_OWNERR_CTL_LBN 14
#define FRF_AZ_EVQ_OWNERR_CTL_WIDTH 1
#define FRF_AZ_EVQ_FIFO_AF_TH_LBN 7
#define FRF_AZ_EVQ_FIFO_AF_TH_WIDTH 7
#define FRF_AZ_EVQ_FIFO_NOTAF_TH_LBN 0
#define FRF_AZ_EVQ_FIFO_NOTAF_TH_WIDTH 7


/*
 * FR_AZ_EVQ_CNT1_REG(128bit):
 * Event counter 1 register
 */
#define FR_AZ_EVQ_CNT1_REG_OFST 0x00000460
/* falcona0,falconb0,sienaa0=net_func_bar2,falcona0=char_func_bar0 */

#define FRF_AZ_EVQ_CNT_PRE_FIFO_LBN 120
#define FRF_AZ_EVQ_CNT_PRE_FIFO_WIDTH 7
#define FRF_AZ_EVQ_CNT_TOBIU_LBN 100
#define FRF_AZ_EVQ_CNT_TOBIU_WIDTH 20
#define FRF_AZ_EVQ_TX_REQ_CNT_LBN 80
#define FRF_AZ_EVQ_TX_REQ_CNT_WIDTH 20
#define FRF_AZ_EVQ_RX_REQ_CNT_LBN 60
#define FRF_AZ_EVQ_RX_REQ_CNT_WIDTH 20
#define FRF_AZ_EVQ_EM_REQ_CNT_LBN 40
#define FRF_AZ_EVQ_EM_REQ_CNT_WIDTH 20
#define FRF_AZ_EVQ_CSR_REQ_CNT_LBN 20
#define FRF_AZ_EVQ_CSR_REQ_CNT_WIDTH 20
#define FRF_AZ_EVQ_ERR_REQ_CNT_LBN 0
#define FRF_AZ_EVQ_ERR_REQ_CNT_WIDTH 20


/*
 * FR_AZ_EVQ_CNT2_REG(128bit):
 * Event counter 2 register
 */
#define FR_AZ_EVQ_CNT2_REG_OFST 0x00000470
/* falcona0,falconb0,sienaa0=net_func_bar2,falcona0=char_func_bar0 */

#define FRF_AZ_EVQ_UPD_REQ_CNT_LBN 104
#define FRF_AZ_EVQ_UPD_REQ_CNT_WIDTH 20
#define FRF_AZ_EVQ_CLR_REQ_CNT_LBN 84
#define FRF_AZ_EVQ_CLR_REQ_CNT_WIDTH 20
#define FRF_AZ_EVQ_RDY_CNT_LBN 80
#define FRF_AZ_EVQ_RDY_CNT_WIDTH 4
#define FRF_AZ_EVQ_WU_REQ_CNT_LBN 60
#define FRF_AZ_EVQ_WU_REQ_CNT_WIDTH 20
#define FRF_AZ_EVQ_WET_REQ_CNT_LBN 40
#define FRF_AZ_EVQ_WET_REQ_CNT_WIDTH 20
#define FRF_AZ_EVQ_INIT_REQ_CNT_LBN 20
#define FRF_AZ_EVQ_INIT_REQ_CNT_WIDTH 20
#define FRF_AZ_EVQ_TM_REQ_CNT_LBN 0
#define FRF_AZ_EVQ_TM_REQ_CNT_WIDTH 20
```



```


/*
 * FR_CZ_USR_EV_REG(32bit):
 * Event mailbox register
 */
#define FR_CZ_USR_EV_REG_OFST 0x00000540
/* sienaa0=net_func_bar2 */
#define FR_CZ_USR_EV_REG_STEP 8192
#define FR_CZ_USR_EV_REG_ROWS 1024

#define FRF_CZ_USR_EV_DATA_LBN 0
#define FRF_CZ_USR_EV_DATA_WIDTH 32


/*
 * FR_AZ_BUF_TBL_CFG_REG(128bit):
 * Buffer table configuration register
 */
#define FR_AZ_BUF_TBL_CFG_REG_OFST 0x00000600
/* falcona0,falconb0,sienaa0=net_func_bar2,falcona0=char_func_bar0 */

#define FRF_AZ_BUF_TBL_MODE_LBN 3
#define FRF_AZ_BUF_TBL_MODE_WIDTH 1


/*
 * FR_AZ_SRM_RX_DC_CFG_REG(128bit):
 * SRAM receive descriptor cache configuration register
 */
#define FR_AZ_SRM_RX_DC_CFG_REG_OFST 0x00000610
/* falcona0,falconb0,sienaa0=net_func_bar2,falcona0=char_func_bar0 */

#define FRF_AZ_SRM_CLK_TMP_EN_LBN 21
#define FRF_AZ_SRM_CLK_TMP_EN_WIDTH 1
#define FRF_AZ_SRM_RX_DC_BASE_ADR_LBN 0
#define FRF_AZ_SRM_RX_DC_BASE_ADR_WIDTH 21


/*
 * FR_AZ_SRM_TX_DC_CFG_REG(128bit):
 * SRAM transmit descriptor cache configuration register
 */
#define FR_AZ_SRM_TX_DC_CFG_REG_OFST 0x00000620
/* falcona0,falconb0,sienaa0=net_func_bar2,falcona0=char_func_bar0 */

#define FRF_AZ_SRM_TX_DC_BASE_ADR_LBN 0
#define FRF_AZ_SRM_TX_DC_BASE_ADR_WIDTH 21


/*
 * FR_AZ_SRM_CFG_REG(128bit):
 * SRAM configuration register
 */
#define FR_AZ_SRM_CFG_REG_SF_OFST 0x00000380
/* falcona0,falconb0=eeprom_flash */
/*
 * FR_AZ_SRM_CFG_REG(128bit):
 * SRAM configuration register
 */
#define FR_AZ_SRM_CFG_REG_OFST 0x00000630
/* falcona0,falconb0,sienaa0=net_func_bar2,falcona0=char_func_bar0 */

#define FRF_AZ_SRM_OOB_ADR_INTEN_LBN 5
#define FRF_AZ_SRM_OOB_ADR_INTEN_WIDTH 1
#define FRF_AZ_SRM_OOB_BUF_INTEN_LBN 4
```

```
#define FRF_AZ_SRM_OOB_BUF_INTEN_WIDTH 1
#define FRF_AZ_SRM_INIT_EN_LBN 3
#define FRF_AZ_SRM_INIT_EN_WIDTH 1
#define FRF_AZ_SRM_NUM_BANK_LBN 2
#define FRF_AZ_SRM_NUM_BANK_WIDTH 1
#define FRF_AZ_SRM_BANK_SIZE_LBN 0
#define FRF_AZ_SRM_BANK_SIZE_WIDTH 2


/*
 * FR_AZ_BUF_TBL_UPD_REG(128bit):
 * Buffer table update register
 */
#define FR_AZ_BUF_TBL_UPD_REG_OFST 0x00000650
/* falcona0,falconb0,sienaa0=net_func_bar2,falcona0=char_func_bar0 */

#define FRF_AZ_BUF_UPD_CMD_LBN 63
#define FRF_AZ_BUF_UPD_CMD_WIDTH 1
#define FRF_AZ_BUF_CLR_CMD_LBN 62
#define FRF_AZ_BUF_CLR_CMD_WIDTH 1
#define FRF_AZ_BUF_CLR_END_ID_LBN 32
#define FRF_AZ_BUF_CLR_END_ID_WIDTH 20
#define FRF_AZ_BUF_CLR_START_ID_LBN 0
#define FRF_AZ_BUF_CLR_START_ID_WIDTH 20


/*
 * FR_AZ_SRM_UPD_EVQ_REG(128bit):
 * Buffer table update register
 */
#define FR_AZ_SRM_UPD_EVQ_REG_OFST 0x00000660
/* falcona0,falconb0,sienaa0=net_func_bar2,falcona0=char_func_bar0 */

#define FRF_AZ_SRM_UPD_EVQ_ID_LBN 0
#define FRF_AZ_SRM_UPD_EVQ_ID_WIDTH 12


/*
 * FR_AZ_SRAM_PARITY_REG(128bit):
 * SRAM parity register.
 */
#define FR_AZ_SRAM_PARITY_REG_OFST 0x00000670
/* falcona0,falconb0,sienaa0=net_func_bar2,falcona0=char_func_bar0 */

#define FRF_CZ_BYPASS_ECC_LBN 3
#define FRF_CZ_BYPASS_ECC_WIDTH 1
#define FRF_CZ_SEC_INT_LBN 2
#define FRF_CZ_SEC_INT_WIDTH 1
#define FRF_CZ_FORCE_SRAM_DOUBLE_ERR_LBN 1
#define FRF_CZ_FORCE_SRAM_DOUBLE_ERR_WIDTH 1
#define FRF_CZ_FORCE_SRAM_SINGLE_ERR_LBN 0
#define FRF_CZ_FORCE_SRAM_SINGLE_ERR_WIDTH 1
#define FRF_AB_FORCE_SRAM_PERR_LBN 0
#define FRF_AB_FORCE_SRAM_PERR_WIDTH 1


/*
 * FR_AZ_RX_CFG_REG(128bit):
 * Receive configuration register
 */
#define FR_AZ_RX_CFG_REG_OFST 0x00000800
/* falcona0,falconb0,sienaa0=net_func_bar2,falcona0=char_func_bar0 */

#define FRF_CZ_RX_HDR_SPLIT_EN_LBN 71
#define FRF_CZ_RX_HDR_SPLIT_EN_WIDTH 1
#define FRF_CZ_RX_HDR_SPLIT_PLD_BUF_SIZE_LBN 62
```



```
#define FRF_CZ_RX_HDR_SPLIT_PLD_BUF_SIZE_WIDTH 9
#define FRF_CZ_RX_HDR_SPLIT_HDR_BUF_SIZE_LBN 53
#define FRF_CZ_RX_HDR_SPLIT_HDR_BUF_SIZE_WIDTH 9
#define FRF_CZ_RX_PRE_RFF_IPG_LBN 49
#define FRF_CZ_RX_PRE_RFF_IPG_WIDTH 4
#define FRF_BZ_RX_TCP_SUP_LBN 48
#define FRF_BZ_RX_TCP_SUP_WIDTH 1
#define FRF_BZ_RX_INGR_EN_LBN 47
#define FRF_BZ_RX_INGR_EN_WIDTH 1
#define FRF_BZ_RX_IP_HASH_LBN 46
#define FRF_BZ_RX_IP_HASH_WIDTH 1
#define FRF_BZ_RX_HASH_ALG_LBN 45
#define FRF_BZ_RX_HASH_ALG_WIDTH 1
#define FRF_BZ_RX_HASH_INSRT_HDR_LBN 44
#define FRF_BZ_RX_HASH_INSRT_HDR_WIDTH 1
#define FRF_BZ_RX_DESC_PUSH_EN_LBN 43
#define FRF_BZ_RX_DESC_PUSH_EN_WIDTH 1
#define FRF_BZ_RX_RDW_PATCH_EN_LBN 42
#define FRF_BZ_RX_RDW_PATCH_EN_WIDTH 1
#define FRF_BB_RX_PCI_BURST_SIZE_LBN 39
#define FRF_BB_RX_PCI_BURST_SIZE_WIDTH 3
#define FRF_BZ_RX_OWNERR_CTL_LBN 38
#define FRF_BZ_RX_OWNERR_CTL_WIDTH 1
#define FRF_BZ_RX_XON_TX_TH_LBN 33
#define FRF_BZ_RX_XON_TX_TH_WIDTH 5
#define FRF_AA_RX_DESC_PUSH_EN_LBN 35
#define FRF_AA_RX_DESC_PUSH_EN_WIDTH 1
#define FRF_AA_RX_RDW_PATCH_EN_LBN 34
#define FRF_AA_RX_RDW_PATCH_EN_WIDTH 1
#define FRF_AA_RX_PCI_BURST_SIZE_LBN 31
#define FRF_AA_RX_PCI_BURST_SIZE_WIDTH 3
#define FRF_BZ_RX_XOFF_TX_TH_LBN 28
#define FRF_BZ_RX_XOFF_TX_TH_WIDTH 5
#define FRF_AA_RX_OWNERR_CTL_LBN 30
#define FRF_AA_RX_OWNERR_CTL_WIDTH 1
#define FRF_AA_RX_XON_TX_TH_LBN 25
#define FRF_AA_RX_XON_TX_TH_WIDTH 5
#define FRF_BZ_RX_USR_BUF_SIZE_LBN 19
#define FRF_BZ_RX_USR_BUF_SIZE_WIDTH 9
#define FRF_AA_RX_XOFF_TX_TH_LBN 20
#define FRF_AA_RX_XOFF_TX_TH_WIDTH 5
#define FRF_AA_RX_USR_BUF_SIZE_LBN 11
#define FRF_AA_RX_USR_BUF_SIZE_WIDTH 9
#define FRF_BZ_RX_XON_MAC_TH_LBN 10
#define FRF_BZ_RX_XON_MAC_TH_WIDTH 9
#define FRF_AA_RX_XON_MAC_TH_LBN 6
#define FRF_AA_RX_XON_MAC_TH_WIDTH 5
#define FRF_BZ_RX_XOFF_MAC_TH_LBN 1
#define FRF_BZ_RX_XOFF_MAC_TH_WIDTH 9
#define FRF_AA_RX_XOFF_MAC_TH_LBN 1
#define FRF_AA_RX_XOFF_MAC_TH_WIDTH 5
#define FRF_AZ_RX_XOFF_MAC_EN_LBN 0
#define FRF_AZ_RX_XOFF_MAC_EN_WIDTH 1


/*
 * FR_AZ_RX_FILTER_CTL_REG(128bit):
 * Receive filter control registers
 */
#define FR_AZ_RX_FILTER_CTL_REG_OFST 0x00000810
/* falconb0,sienaa0=net_func_bar2,falcona0=char_func_bar0 */

#define FRF_CZ_ETHERNET_WILDCARD_SEARCH_LIMIT_LBN 94
#define FRF_CZ_ETHERNET_WILDCARD_SEARCH_LIMIT_WIDTH 8
#define FRF_CZ_ETHERNET_FULL_SEARCH_LIMIT_LBN 86
#define FRF_CZ_ETHERNET_FULL_SEARCH_LIMIT_WIDTH 8
```

```
#define FRF_CZ_RX_FILTER_ALL_VLAN_ETHERTYPES_LBN 85
#define FRF_CZ_RX_FILTER_ALL_VLAN_ETHERTYPES_WIDTH 1
#define FRF_CZ_RX_VLAN_MATCH_ETHERTYPE_LBN 69
#define FRF_CZ_RX_VLAN_MATCH_ETHERTYPE_WIDTH 16
#define FRF_CZ_MULTICAST_NOMATCH_Q_ID_LBN 57
#define FRF_CZ_MULTICAST_NOMATCH_Q_ID_WIDTH 12
#define FRF_CZ_MULTICAST_NOMATCH_RSS_ENABLED_LBN 56
#define FRF_CZ_MULTICAST_NOMATCH_RSS_ENABLED_WIDTH 1
#define FRF_CZ_MULTICAST_NOMATCH_IP_OVERRIDE_LBN 55
#define FRF_CZ_MULTICAST_NOMATCH_IP_OVERRIDE_WIDTH 1
#define FRF_CZ_UNICAST_NOMATCH_Q_ID_LBN 43
#define FRF_CZ_UNICAST_NOMATCH_Q_ID_WIDTH 12
#define FRF_CZ_UNICAST_NOMATCH_RSS_ENABLED_LBN 42
#define FRF_CZ_UNICAST_NOMATCH_RSS_ENABLED_WIDTH 1
#define FRF_CZ_UNICAST_NOMATCH_IP_OVERRIDE_LBN 41
#define FRF_CZ_UNICAST_NOMATCH_IP_OVERRIDE_WIDTH 1
#define FRF_BZ_SCATTER_ENBL_NO_MATCH_Q_LBN 40
#define FRF_BZ_SCATTER_ENBL_NO_MATCH_Q_WIDTH 1
#define FRF_AZ_UDP_FULL_SRCH_LIMIT_LBN 32
#define FRF_AZ_UDP_FULL_SRCH_LIMIT_WIDTH 8
#define FRF_AZ_NUM_KER_LBN 24
#define FRF_AZ_NUM_KER_WIDTH 2
#define FRF_AZ_UDP_WILD_SRCH_LIMIT_LBN 16
#define FRF_AZ_UDP_WILD_SRCH_LIMIT_WIDTH 8
#define FRF_AZ_TCP_WILD_SRCH_LIMIT_LBN 8
#define FRF_AZ_TCP_WILD_SRCH_LIMIT_WIDTH 8
#define FRF_AZ_TCP_FULL_SRCH_LIMIT_LBN 0
#define FRF_AZ_TCP_FULL_SRCH_LIMIT_WIDTH 8


/*
 * FR_AZ_RX_FLUSH_DESCQ_REG(128bit):
 * Receive flush descriptor queue register
 */
#define FR_AZ_RX_FLUSH_DESCQ_REG_OFST 0x00000820
/* falcona0,falconb0,sienaa0=net_func_bar2,falcona0=char_func_bar0 */

#define FRF_AZ_RX_FLUSH_DESCQ_CMD_LBN 24
#define FRF_AZ_RX_FLUSH_DESCQ_CMD_WIDTH 1
#define FRF_AZ_RX_FLUSH_DESCQ_LBN 0
#define FRF_AZ_RX_FLUSH_DESCQ_WIDTH 12


/*
 * FR_BZ_RX_DESC_UPD_REGP0(128bit):
 * Receive descriptor update register.
 */
#define FR_BZ_RX_DESC_UPD_REGP0_OFST 0x00000830
/* falconb0,sienaa0=net_func_bar2 */
#define FR_BZ_RX_DESC_UPD_REGP0_STEP 8192
#define FR_BZ_RX_DESC_UPD_REGP0_ROWS 1024
/*
 * FR_AA_RX_DESC_UPD_REG_KER(128bit):
 * Receive descriptor update register.
 */
#define FR_AA_RX_DESC_UPD_REG_KER_OFST 0x00000830
/* falcona0=net_func_bar2 */
#define FR_AA_RX_DESC_UPD_REG_KER_STEP 8192
#define FR_AA_RX_DESC_UPD_REG_KER_ROWS 4
/*
 * FR_AB_RX_DESC_UPD_REGP123(128bit):
 * Receive descriptor update register.
 */
#define FR_AB_RX_DESC_UPD_REGP123_OFST 0x01000830
/* falconb0=net_func_bar2,falcona0=char_func_bar0 */
#define FR_AB_RX_DESC_UPD_REGP123_STEP 8192
```



```
#define FR_AB_RX_DESC_UPD_REGP123_ROWS 3072
/*
 * FR_AA_RX_DESC_UPD_REGP0(128bit):
 * Receive descriptor update register.
 */
#define FR_AA_RX_DESC_UPD_REGP0_OFST 0x00008830
/* falcona0=char_func_bar0 */
#define FR_AA_RX_DESC_UPD_REGP0_STEP 8192
#define FR_AA_RX_DESC_UPD_REGP0_ROWS 1020

#define FRF_AZ_RX_DESC_WPTR_LBN 96
#define FRF_AZ_RX_DESC_WPTR_WIDTH 12
#define FRF_AZ_RX_DESC_PUSH_CMD_LBN 95
#define FRF_AZ_RX_DESC_PUSH_CMD_WIDTH 1
#define FRF_AZ_RX_DESC_LBN 0
#define FRF_AZ_RX_DESC_WIDTH 64
#define FRF_AZ_RX_DESC_DW0_LBN 0
#define FRF_AZ_RX_DESC_DW0_WIDTH 32
#define FRF_AZ_RX_DESC_DW1_LBN 32
#define FRF_AZ_RX_DESC_DW1_WIDTH 32


/*
 * FR_AZ_RX_DC_CFG_REG(128bit):
 * Receive descriptor cache configuration register
 */
#define FR_AZ_RX_DC_CFG_REG_OFST 0x00000840
/* falcona0,falconb0,sienaa0=net_func_bar2,falcona0=char_func_bar0 */

#define FRF_AZ_RX_MAX_PF_LBN 2
#define FRF_AZ_RX_MAX_PF_WIDTH 2
#define FRF_AZ_RX_DC_SIZE_LBN 0
#define FRF_AZ_RX_DC_SIZE_WIDTH 2
#define FFE_AZ_RX_DC_SIZE_64 3
#define FFE_AZ_RX_DC_SIZE_32 2
#define FFE_AZ_RX_DC_SIZE_16 1
#define FFE_AZ_RX_DC_SIZE_8 0


/*
 * FR_AZ_RX_DC_PF_WM_REG(128bit):
 * Receive descriptor cache pre-fetch watermark register
 */
#define FR_AZ_RX_DC_PF_WM_REG_OFST 0x00000850
/* falcona0,falconb0,sienaa0=net_func_bar2,falcona0=char_func_bar0 */

#define FRF_AZ_RX_DC_PF_HWM_LBN 6
#define FRF_AZ_RX_DC_PF_HWM_WIDTH 6
#define FRF_AZ_RX_DC_PF_LWM_LBN 0
#define FRF_AZ_RX_DC_PF_LWM_WIDTH 6


/*
 * FR_BZ_RX_RSS_TKEY_REG(128bit):
 * RSS Toeplitz hash key
 */
#define FR_BZ_RX_RSS_TKEY_REG_OFST 0x00000860
/* falconb0,sienaa0=net_func_bar2 */

#define FRF_BZ_RX_RSS_TKEY_LBN 96
#define FRF_BZ_RX_RSS_TKEY_WIDTH 32
#define FRF_BZ_RX_RSS_TKEY_DW3_LBN 96
#define FRF_BZ_RX_RSS_TKEY_DW3_WIDTH 32
#define FRF_BZ_RX_RSS_TKEY_DW2_LBN 64
#define FRF_BZ_RX_RSS_TKEY_DW2_WIDTH 32
#define FRF_BZ_RX_RSS_TKEY_DW1_LBN 32
```

```
#define FRF_BZ_RX_RSS_TKEY_DW1_WIDTH 32
#define FRF_BZ_RX_RSS_TKEY_DW0_LBN 0
#define FRF_BZ_RX_RSS_TKEY_DW0_WIDTH 32


/*
 * FR_AZ_RX_NODESC_DROP_REG(128bit):
 * Receive dropped packet counter register
 */
#define FR_AZ_RX_NODESC_DROP_REG_OFST 0x00000880
/* falcona0,falconb0,sienaa0=net_func_bar2,falcona0=char_func_bar0 */

#define FRF_AZ_RX_NODESC_DROP_CNT_LBN 0
#define FRF_AZ_RX_NODESC_DROP_CNT_WIDTH 16


/*
 * FR_AZ_RX_SELF_RST_REG(128bit):
 * Receive self reset register
 */
#define FR_AZ_RX_SELF_RST_REG_OFST 0x00000890
/* falcona0,falconb0,sienaa0=net_func_bar2,falcona0=char_func_bar0 */

#define FRF_AZ_RX_ISCSI_DIS_LBN 17
#define FRF_AZ_RX_ISCSI_DIS_WIDTH 1
#define FRF_AB_RX_SW_RST_REG_LBN 16
#define FRF_AB_RX_SW_RST_REG_WIDTH 1
#define FRF_AB_RX_SELF_RST_EN_LBN 8
#define FRF_AB_RX_SELF_RST_EN_WIDTH 1
#define FRF_AZ_RX_MAX_PF_LAT_LBN 4
#define FRF_AZ_RX_MAX_PF_LAT_WIDTH 4
#define FRF_AZ_RX_MAX_LU_LAT_LBN 0
#define FRF_AZ_RX_MAX_LU_LAT_WIDTH 4


/*
 * FR_AZ_RX_DEBUG_REG(128bit):
 * undocumented register
 */
#define FR_AZ_RX_DEBUG_REG_OFST 0x000008a0
/* falcona0,falconb0,sienaa0=net_func_bar2,falcona0=char_func_bar0 */

#define FRF_AZ_RX_DEBUG_LBN 0
#define FRF_AZ_RX_DEBUG_WIDTH 64
#define FRF_AZ_RX_DEBUG_DW0_LBN 0
#define FRF_AZ_RX_DEBUG_DW0_WIDTH 32
#define FRF_AZ_RX_DEBUG_DW1_LBN 32
#define FRF_AZ_RX_DEBUG_DW1_WIDTH 32


/*
 * FR_AZ_RX_PUSH_DROP_REG(128bit):
 * Receive descriptor push dropped counter register
 */
#define FR_AZ_RX_PUSH_DROP_REG_OFST 0x000008b0
/* falcona0,falconb0,sienaa0=net_func_bar2,falcona0=char_func_bar0 */

#define FRF_AZ_RX_PUSH_DROP_CNT_LBN 0
#define FRF_AZ_RX_PUSH_DROP_CNT_WIDTH 32


/*
 * FR_CZ_RX_RSS_IPV6_REG1(128bit):
 * IPv6 RSS Toeplitz hash key low bytes
 */
#define FR_CZ_RX_RSS_IPV6_REG1_OFST 0x000008d0
```



```
/* sienaa0=net_func_bar2 */

#define FRF_CZ_RX_RSS_IPV6_TKEY_LO_LBN 0
#define FRF_CZ_RX_RSS_IPV6_TKEY_LO_WIDTH 128
#define FRF_CZ_RX_RSS_IPV6_TKEY_LO_DW0_LBN 0
#define FRF_CZ_RX_RSS_IPV6_TKEY_LO_DW0_WIDTH 32
#define FRF_CZ_RX_RSS_IPV6_TKEY_LO_DW1_LBN 32
#define FRF_CZ_RX_RSS_IPV6_TKEY_LO_DW1_WIDTH 32
#define FRF_CZ_RX_RSS_IPV6_TKEY_LO_DW2_LBN 64
#define FRF_CZ_RX_RSS_IPV6_TKEY_LO_DW2_WIDTH 32
#define FRF_CZ_RX_RSS_IPV6_TKEY_LO_DW3_LBN 96
#define FRF_CZ_RX_RSS_IPV6_TKEY_LO_DW3_WIDTH 32


/*
 * FR_CZ_RX_RSS_IPV6_REG2(128bit):
 * IPv6 RSS Toeplitz hash key middle bytes
 */
#define FR_CZ_RX_RSS_IPV6_REG2_OFST 0x000008e0
/* sienaa0=net_func_bar2 */

#define FRF_CZ_RX_RSS_IPV6_TKEY_MID_LBN 0
#define FRF_CZ_RX_RSS_IPV6_TKEY_MID_WIDTH 128
#define FRF_CZ_RX_RSS_IPV6_TKEY_MID_DW0_LBN 0
#define FRF_CZ_RX_RSS_IPV6_TKEY_MID_DW0_WIDTH 32
#define FRF_CZ_RX_RSS_IPV6_TKEY_MID_DW1_LBN 32
#define FRF_CZ_RX_RSS_IPV6_TKEY_MID_DW1_WIDTH 32
#define FRF_CZ_RX_RSS_IPV6_TKEY_MID_DW2_LBN 64
#define FRF_CZ_RX_RSS_IPV6_TKEY_MID_DW2_WIDTH 32
#define FRF_CZ_RX_RSS_IPV6_TKEY_MID_DW3_LBN 96
#define FRF_CZ_RX_RSS_IPV6_TKEY_MID_DW3_WIDTH 32


/*
 * FR_CZ_RX_RSS_IPV6_REG3(128bit):
 * IPv6 RSS Toeplitz hash key upper bytes and IPv6 RSS settings
 */
#define FR_CZ_RX_RSS_IPV6_REG3_OFST 0x000008f0
/* sienaa0=net_func_bar2 */

#define FRF_CZ_RX_RSS_IPV6_THASH_ENABLE_LBN 66
#define FRF_CZ_RX_RSS_IPV6_THASH_ENABLE_WIDTH 1
#define FRF_CZ_RX_RSS_IPV6_IP_THASH_ENABLE_LBN 65
#define FRF_CZ_RX_RSS_IPV6_IP_THASH_ENABLE_WIDTH 1
#define FRF_CZ_RX_RSS_IPV6_TCP_SUPPRESS_LBN 64
#define FRF_CZ_RX_RSS_IPV6_TCP_SUPPRESS_WIDTH 1
#define FRF_CZ_RX_RSS_IPV6_TKEY_HI_LBN 0
#define FRF_CZ_RX_RSS_IPV6_TKEY_HI_WIDTH 64
#define FRF_CZ_RX_RSS_IPV6_TKEY_HI_DW0_LBN 0
#define FRF_CZ_RX_RSS_IPV6_TKEY_HI_DW0_WIDTH 32
#define FRF_CZ_RX_RSS_IPV6_TKEY_HI_DW1_LBN 32
#define FRF_CZ_RX_RSS_IPV6_TKEY_HI_DW1_WIDTH 32


/*
 * FR_AZ_TX_FLUSH_DESCQ_REG(128bit):
 * Transmit flush descriptor queue register
 */
#define FR_AZ_TX_FLUSH_DESCQ_REG_OFST 0x00000a00
/* falcona0,falconb0,sienaa0=net_func_bar2,falcona0=char_func_bar0 */

#define FRF_AZ_TX_FLUSH_DESCQ_CMD_LBN 12
#define FRF_AZ_TX_FLUSH_DESCQ_CMD_WIDTH 1
#define FRF_AZ_TX_FLUSH_DESCQ_LBN 0
#define FRF_AZ_TX_FLUSH_DESCQ_WIDTH 12
```

```
1713 /*
1714  * FR_BZ_TX_DESC_UPD_REGP0(128bit):
1715  * Transmit descriptor update register.
1716  */
1717 #define FR_BZ_TX_DESC_UPD_REGP0_OFST 0x00000a10
1718 /* falconb0,sienaa0=net_func_bar2 */
1719 #define FR_BZ_TX_DESC_UPD_REGP0_STEP 8192
1720 #define FR_BZ_TX_DESC_UPD_REGP0_ROWS 1024
1721 /*
1722  * FR_AA_TX_DESC_UPD_REG_KER(128bit):
1723  * Transmit descriptor update register.
1724  */
1725 #define FR_AA_TX_DESC_UPD_REG_KER_OFST 0x00000a10
1726 /* falcona0=net_func_bar2 */
1727 #define FR_AA_TX_DESC_UPD_REG_KER_STEP 8192
1728 #define FR_AA_TX_DESC_UPD_REG_KER_ROWS 8
1729 /*
1730  * FR_AB_TX_DESC_UPD_REGP123(128bit):
1731  * Transmit descriptor update register.
1732  */
1733 #define FR_AB_TX_DESC_UPD_REGP123_OFST 0x01000a10
1734 /* falconb0=net_func_bar2,falcona0=char_func_bar0 */
1735 #define FR_AB_TX_DESC_UPD_REGP123_STEP 8192
1736 #define FR_AB_TX_DESC_UPD_REGP123_ROWS 3072
1737 /*
1738  * FR_AA_TX_DESC_UPD_REGP0(128bit):
1739  * Transmit descriptor update register.
1740  */
1741 #define FR_AA_TX_DESC_UPD_REGP0_OFST 0x00008a10
1742 /* falcona0=char_func_bar0 */
1743 #define FR_AA_TX_DESC_UPD_REGP0_STEP 8192
1744 #define FR_AA_TX_DESC_UPD_REGP0_ROWS 1020

1746 #define FRF_AZ_TX_DESC_WPTR_LBN 96
1747 #define FRF_AZ_TX_DESC_WPTR_WIDTH 12
1748 #define FRF_AZ_TX_DESC_PUSH_CMD_LBN 95
1749 #define FRF_AZ_TX_DESC_PUSH_CMD_WIDTH 1
1750 #define FRF_AZ_TX_DESC_LBN 0
1751 #define FRF_AZ_TX_DESC_WIDTH 95
1752 #define FRF_AZ_TX_DESC_DW0_LBN 0
1753 #define FRF_AZ_TX_DESC_DW0_WIDTH 32
1754 #define FRF_AZ_TX_DESC_DW1_LBN 32
1755 #define FRF_AZ_TX_DESC_DW1_WIDTH 32
1756 #define FRF_AZ_TX_DESC_DW2_LBN 64
1757 #define FRF_AZ_TX_DESC_DW2_WIDTH 31


1760 /*
1761  * FR_AZ_TX_DC_CFG_REG(128bit):
1762  * Transmit descriptor cache configuration register
1763  */
1764 #define FR_AZ_TX_DC_CFG_REG_OFST 0x00000a20
1765 /* falcona0,falconb0,sienaa0=net_func_bar2,falcona0=char_func_bar0 */

1767 #define FRF_AZ_TX_DC_SIZE_LBN 0
1768 #define FRF_AZ_TX_DC_SIZE_WIDTH 2
1769 #define FFE_AZ_TX_DC_SIZE_32 2
1770 #define FFE_AZ_TX_DC_SIZE_16 1
1771 #define FFE_AZ_TX_DC_SIZE_8 0


1774 /*
1775  * FR_AA_TX_CHKSM_CFG_REG(128bit):
1776  * Transmit checksum configuration register
1777  */
```



```
1778 #define FR_AA_TX_CHKSM_CFG_REG_OFST 0x00000a30
1779 /* falcona0=net_func_bar2,falcona0=char_func_bar0 */

1781 #define FRF_AA_TX_Q_CHKSM_DIS_96_127_LBN 96
1782 #define FRF_AA_TX_Q_CHKSM_DIS_96_127_WIDTH 32
1783 #define FRF_AA_TX_Q_CHKSM_DIS_64_95_LBN 64
1784 #define FRF_AA_TX_Q_CHKSM_DIS_64_95_WIDTH 32
1785 #define FRF_AA_TX_Q_CHKSM_DIS_32_63_LBN 32
1786 #define FRF_AA_TX_Q_CHKSM_DIS_32_63_WIDTH 32
1787 #define FRF_AA_TX_Q_CHKSM_DIS_0_31_LBN 0
1788 #define FRF_AA_TX_Q_CHKSM_DIS_0_31_WIDTH 32


1791 /*
1792  * FR_AZ_TX_CFG_REG(128bit):
1793  * Transmit configuration register
1794  */
1795 #define FR_AZ_TX_CFG_REG_OFST 0x00000a50
1796 /* falcona0,falconb0,sienaa0=net_func_bar2,falcona0=char_func_bar0 */

1798 #define FRF_CZ_TX_CONT_LOOKUP_THRESH_RANGE_LBN 114
1799 #define FRF_CZ_TX_CONT_LOOKUP_THRESH_RANGE_WIDTH 8
1800 #define FRF_CZ_TX_FILTER_TEST_MODE_BIT_LBN 113
1801 #define FRF_CZ_TX_FILTER_TEST_MODE_BIT_WIDTH 1
1802 #define FRF_CZ_TX_ETH_FILTER_WILD_SEARCH_RANGE_LBN 105
1803 #define FRF_CZ_TX_ETH_FILTER_WILD_SEARCH_RANGE_WIDTH 8
1804 #define FRF_CZ_TX_ETH_FILTER_FULL_SEARCH_RANGE_LBN 97
1805 #define FRF_CZ_TX_ETH_FILTER_FULL_SEARCH_RANGE_WIDTH 8
1806 #define FRF_CZ_TX_UDPIP_FILTER_WILD_SEARCH_RANGE_LBN 89
1807 #define FRF_CZ_TX_UDPIP_FILTER_WILD_SEARCH_RANGE_WIDTH 8
1808 #define FRF_CZ_TX_UDPIP_FILTER_FULL_SEARCH_RANGE_LBN 81
1809 #define FRF_CZ_TX_UDPIP_FILTER_FULL_SEARCH_RANGE_WIDTH 8
1810 #define FRF_CZ_TX_TCPIP_FILTER_WILD_SEARCH_RANGE_LBN 73
1811 #define FRF_CZ_TX_TCPIP_FILTER_WILD_SEARCH_RANGE_WIDTH 8
1812 #define FRF_CZ_TX_TCPIP_FILTER_FULL_SEARCH_RANGE_LBN 65
1813 #define FRF_CZ_TX_TCPIP_FILTER_FULL_SEARCH_RANGE_WIDTH 8
1814 #define FRF_CZ_TX_FILTER_ALL_VLAN_ETHERTYPES_BIT_LBN 64
1815 #define FRF_CZ_TX_FILTER_ALL_VLAN_ETHERTYPES_BIT_WIDTH 1
1816 #define FRF_CZ_TX_VLAN_MATCH_ETHERTYPE_RANGE_LBN 48
1817 #define FRF_CZ_TX_VLAN_MATCH_ETHERTYPE_RANGE_WIDTH 16
1818 #define FRF_CZ_TX_FILTER_EN_BIT_LBN 47
1819 #define FRF_CZ_TX_FILTER_EN_BIT_WIDTH 1
1820 #define FRF_AZ_TX_IP_ID_P0_OFS_LBN 16
1821 #define FRF_AZ_TX_IP_ID_P0_OFS_WIDTH 15
1822 #define FRF_AZ_TX_NO_EOP_DISC_EN_LBN 5
1823 #define FRF_AZ_TX_NO_EOP_DISC_EN_WIDTH 1
1824 #define FRF_AZ_TX_P1_PRI_EN_LBN 4
1825 #define FRF_AZ_TX_P1_PRI_EN_WIDTH 1
1826 #define FRF_AZ_TX_OWNERR_CTL_LBN 2
1827 #define FRF_AZ_TX_OWNERR_CTL_WIDTH 1
1828 #define FRF_AA_TX_NON_IP_DROP_DIS_LBN 1
1829 #define FRF_AA_TX_NON_IP_DROP_DIS_WIDTH 1
1830 #define FRF_AZ_TX_IP_ID_REP_EN_LBN 0
1831 #define FRF_AZ_TX_IP_ID_REP_EN_WIDTH 1


1834 /*
1835  * FR_AZ_TX_PUSH_DROP_REG(128bit):
1836  * Transmit push dropped register
1837  */
1838 #define FR_AZ_TX_PUSH_DROP_REG_OFST 0x00000a60
1839 /* falcona0,falconb0,sienaa0=net_func_bar2,falcona0=char_func_bar0 */

1841 #define FRF_AZ_TX_PUSH_DROP_CNT_LBN 0
1842 #define FRF_AZ_TX_PUSH_DROP_CNT_WIDTH 32
```

```
1845 /*
1846  * FR_AZ_TX_RESERVED_REG(128bit):
1847  * Transmit configuration register
1848  */
1849 #define FR_AZ_TX_RESERVED_REG_OFST 0x00000a80
1850 /* falcona0,falconb0,sienaa0=net_func_bar2,falcona0=char_func_bar0 */

1852 #define FRF_AZ_TX_EVT_CNT_LBN 121
1853 #define FRF_AZ_TX_EVT_CNT_WIDTH 7
1854 #define FRF_AZ_TX_PREF_AGE_CNT_LBN 119
1855 #define FRF_AZ_TX_PREF_AGE_CNT_WIDTH 2
1856 #define FRF_AZ_TX_RD_COMP_TMR_LBN 96
1857 #define FRF_AZ_TX_RD_COMP_TMR_WIDTH 23
1858 #define FRF_AZ_TX_PUSH_EN_LBN 89
1859 #define FRF_AZ_TX_PUSH_EN_WIDTH 1
1860 #define FRF_AZ_TX_PUSH_CHK_DIS_LBN 88
1861 #define FRF_AZ_TX_PUSH_CHK_DIS_WIDTH 1
1862 #define FRF_AZ_TX_D_FF_FULL_P0_LBN 85
1863 #define FRF_AZ_TX_D_FF_FULL_P0_WIDTH 1
1864 #define FRF_AZ_TX_DMAR_ST_P0_LBN 81
1865 #define FRF_AZ_TX_DMAR_ST_P0_WIDTH 1
1866 #define FRF_AZ_TX_DMAQ_ST_LBN 78
1867 #define FRF_AZ_TX_DMAQ_ST_WIDTH 1
1868 #define FRF_AZ_TX_RX_SPACER_LBN 64
1869 #define FRF_AZ_TX_RX_SPACER_WIDTH 8
1870 #define FRF_AZ_TX_DROP_ABORT_EN_LBN 60
1871 #define FRF_AZ_TX_DROP_ABORT_EN_WIDTH 1
1872 #define FRF_AZ_TX_SOFT_EVT_EN_LBN 59
1873 #define FRF_AZ_TX_SOFT_EVT_EN_WIDTH 1
1874 #define FRF_AZ_TX_PS_EVT_DIS_LBN 58
1875 #define FRF_AZ_TX_PS_EVT_DIS_WIDTH 1
1876 #define FRF_AZ_TX_RX_SPACER_EN_LBN 57
1877 #define FRF_AZ_TX_RX_SPACER_EN_WIDTH 1
1878 #define FRF_AZ_TX_XP_TIMER_LBN 52
1879 #define FRF_AZ_TX_XP_TIMER_WIDTH 5
1880 #define FRF_AZ_TX_PREF_SPACER_LBN 44
1881 #define FRF_AZ_TX_PREF_SPACER_WIDTH 8
1882 #define FRF_AZ_TX_PREF_WD_TMR_LBN 22
1883 #define FRF_AZ_TX_PREF_WD_TMR_WIDTH 22
1884 #define FRF_AZ_TX_ONLY1TAG_LBN 21
1885 #define FRF_AZ_TX_ONLY1TAG_WIDTH 1
1886 #define FRF_AZ_TX_PREF_THRESHOLD_LBN 19
1887 #define FRF_AZ_TX_PREF_THRESHOLD_WIDTH 2
1888 #define FRF_AZ_TX_ONE_PKT_PER_Q_LBN 18
1889 #define FRF_AZ_TX_ONE_PKT_PER_Q_WIDTH 1
1890 #define FRF_AZ_TX_DIS_NON_IP_EV_LBN 17
1891 #define FRF_AZ_TX_DIS_NON_IP_EV_WIDTH 1
1892 #define FRF_AA_TX_DMA_FF_THR_LBN 16
1893 #define FRF_AA_TX_DMA_FF_THR_WIDTH 1
1894 #define FRF_AZ_TX_DMA_SPACER_LBN 8
1895 #define FRF_AZ_TX_DMA_SPACER_WIDTH 8
1896 #define FRF_AA_TX_TCP_DIS_LBN 7
1897 #define FRF_AA_TX_TCP_DIS_WIDTH 1
1898 #define FRF_BZ_TX_FLUSH_MIN_LEN_EN_LBN 7
1899 #define FRF_BZ_TX_FLUSH_MIN_LEN_EN_WIDTH 1
1900 #define FRF_AA_TX_IP_DIS_LBN 6
1901 #define FRF_AA_TX_IP_DIS_WIDTH 1
1902 #define FRF_AZ_TX_MAX_CPL_LBN 2
1903 #define FRF_AZ_TX_MAX_CPL_WIDTH 2
1904 #define FFE_AZ_TX_MAX_CPL_16 3
1905 #define FFE_AZ_TX_MAX_CPL_8 2
1906 #define FFE_AZ_TX_MAX_CPL_4 1
1907 #define FFE_AZ_TX_MAX_CPL_NOLIMIT 0
1908 #define FRF_AZ_TX_MAX_PREF_LBN 0
1909 #define FRF_AZ_TX_MAX_PREF_WIDTH 2
```



```
1910 #define FFE_AZ_TX_MAX_PREF_32 3
1911 #define FFE_AZ_TX_MAX_PREF_16 2
1912 #define FFE_AZ_TX_MAX_PREF_8 1
1913 #define FFE_AZ_TX_MAX_PREF_OFF 0


1916 /*
1917  * FR_BZ_TX_PACE_REG(128bit):
1918  * Transmit pace control register
1919  */
1920 #define FR_BZ_TX_PACE_REG_OFST 0x00000a90
1921 /* falconb0,sienaa0=net_func_bar2 */
1922 /*
1923  * FR_AA_TX_PACE_REG(128bit):
1924  * Transmit pace control register
1925  */
1926 #define FR_AA_TX_PACE_REG_OFST 0x00f80000
1927 /* falcona0=char_func_bar0 */

1929 #define FRF_AZ_TX_PACE_SB_NOT_AF_LBN 19
1930 #define FRF_AZ_TX_PACE_SB_NOT_AF_WIDTH 10
1931 #define FRF_AZ_TX_PACE_SB_AF_LBN 9
1932 #define FRF_AZ_TX_PACE_SB_AF_WIDTH 10
1933 #define FRF_AZ_TX_PACE_FB_BASE_LBN 5
1934 #define FRF_AZ_TX_PACE_FB_BASE_WIDTH 4
1935 #define FRF_AZ_TX_PACE_BIN_TH_LBN 0
1936 #define FRF_AZ_TX_PACE_BIN_TH_WIDTH 5


1939 /*
1940  * FR_AZ_TX_PACE_DROP_QID_REG(128bit):
1941  * PACE Drop QID Counter
1942  */
1943 #define FR_AZ_TX_PACE_DROP_QID_REG_OFST 0x00000aa0
1944 /* falconb0,sienaa0=net_func_bar2,falcona0=char_func_bar0 */

1946 #define FRF_AZ_TX_PACE_QID_DRP_CNT_LBN 0
1947 #define FRF_AZ_TX_PACE_QID_DRP_CNT_WIDTH 16


1950 /*
1951  * FR_AB_TX_VLAN_REG(128bit):
1952  * Transmit VLAN tag register
1953  */
1954 #define FR_AB_TX_VLAN_REG_OFST 0x00000ae0
1955 /* falconb0=net_func_bar2,falcona0=char_func_bar0 */

1957 #define FRF_AB_TX_VLAN_EN_LBN 127
1958 #define FRF_AB_TX_VLAN_EN_WIDTH 1
1959 #define FRF_AB_TX_VLAN7_PORT1_EN_LBN 125
1960 #define FRF_AB_TX_VLAN7_PORT1_EN_WIDTH 1
1961 #define FRF_AB_TX_VLAN7_PORT0_EN_LBN 124
1962 #define FRF_AB_TX_VLAN7_PORT0_EN_WIDTH 1
1963 #define FRF_AB_TX_VLAN7_LBN 112
1964 #define FRF_AB_TX_VLAN7_WIDTH 12
1965 #define FRF_AB_TX_VLAN6_PORT1_EN_LBN 109
1966 #define FRF_AB_TX_VLAN6_PORT1_EN_WIDTH 1
1967 #define FRF_AB_TX_VLAN6_PORT0_EN_LBN 108
1968 #define FRF_AB_TX_VLAN6_PORT0_EN_WIDTH 1
1969 #define FRF_AB_TX_VLAN6_LBN 96
1970 #define FRF_AB_TX_VLAN6_WIDTH 12
1971 #define FRF_AB_TX_VLAN5_PORT1_EN_LBN 93
1972 #define FRF_AB_TX_VLAN5_PORT1_EN_WIDTH 1
1973 #define FRF_AB_TX_VLAN5_PORT0_EN_LBN 92
1974 #define FRF_AB_TX_VLAN5_PORT0_EN_WIDTH 1
1975 #define FRF_AB_TX_VLAN5_LBN 80
```

```
#define FRF_AB_TX_VLAN5_WIDTH 12
#define FRF_AB_TX_VLAN4_PORT1_EN_LBN 77
#define FRF_AB_TX_VLAN4_PORT1_EN_WIDTH 1
#define FRF_AB_TX_VLAN4_PORT0_EN_LBN 76
#define FRF_AB_TX_VLAN4_PORT0_EN_WIDTH 1
#define FRF_AB_TX_VLAN4_LBN 64
#define FRF_AB_TX_VLAN4_WIDTH 12
#define FRF_AB_TX_VLAN3_PORT1_EN_LBN 61
#define FRF_AB_TX_VLAN3_PORT1_EN_WIDTH 1
#define FRF_AB_TX_VLAN3_PORT0_EN_LBN 60
#define FRF_AB_TX_VLAN3_PORT0_EN_WIDTH 1
#define FRF_AB_TX_VLAN3_LBN 48
#define FRF_AB_TX_VLAN3_WIDTH 12
#define FRF_AB_TX_VLAN2_PORT1_EN_LBN 45
#define FRF_AB_TX_VLAN2_PORT1_EN_WIDTH 1
#define FRF_AB_TX_VLAN2_PORT0_EN_LBN 44
#define FRF_AB_TX_VLAN2_PORT0_EN_WIDTH 1
#define FRF_AB_TX_VLAN2_LBN 32
#define FRF_AB_TX_VLAN2_WIDTH 12
#define FRF_AB_TX_VLAN1_PORT1_EN_LBN 29
#define FRF_AB_TX_VLAN1_PORT1_EN_WIDTH 1
#define FRF_AB_TX_VLAN1_PORT0_EN_LBN 28
#define FRF_AB_TX_VLAN1_PORT0_EN_WIDTH 1
#define FRF_AB_TX_VLAN1_LBN 16
#define FRF_AB_TX_VLAN1_WIDTH 12
#define FRF_AB_TX_VLAN0_PORT1_EN_LBN 13
#define FRF_AB_TX_VLAN0_PORT1_EN_WIDTH 1
#define FRF_AB_TX_VLAN0_PORT0_EN_LBN 12
#define FRF_AB_TX_VLAN0_PORT0_EN_WIDTH 1
#define FRF_AB_TX_VLAN0_LBN 0
#define FRF_AB_TX_VLAN0_WIDTH 12


/*
 * FR_AZ_TX_IPFIL_PORTEN_REG(128bit):
 * Transmit filter control register
 */
#define FR_AZ_TX_IPFIL_PORTEN_REG_OFST 0x00000af0
/* falconb0,sienaa0=net_func_bar2,falcona0=char_func_bar0 */

#define FRF_AZ_TX_MADR0_FIL_EN_LBN 64
#define FRF_AZ_TX_MADR0_FIL_EN_WIDTH 1
#define FRF_AB_TX_IPFIL31_PORT_EN_LBN 62
#define FRF_AB_TX_IPFIL31_PORT_EN_WIDTH 1
#define FRF_AB_TX_IPFIL30_PORT_EN_LBN 60
#define FRF_AB_TX_IPFIL30_PORT_EN_WIDTH 1
#define FRF_AB_TX_IPFIL29_PORT_EN_LBN 58
#define FRF_AB_TX_IPFIL29_PORT_EN_WIDTH 1
#define FRF_AB_TX_IPFIL28_PORT_EN_LBN 56
#define FRF_AB_TX_IPFIL28_PORT_EN_WIDTH 1
#define FRF_AB_TX_IPFIL27_PORT_EN_LBN 54
#define FRF_AB_TX_IPFIL27_PORT_EN_WIDTH 1
#define FRF_AB_TX_IPFIL26_PORT_EN_LBN 52
#define FRF_AB_TX_IPFIL26_PORT_EN_WIDTH 1
#define FRF_AB_TX_IPFIL25_PORT_EN_LBN 50
#define FRF_AB_TX_IPFIL25_PORT_EN_WIDTH 1
#define FRF_AB_TX_IPFIL24_PORT_EN_LBN 48
#define FRF_AB_TX_IPFIL24_PORT_EN_WIDTH 1
#define FRF_AB_TX_IPFIL23_PORT_EN_LBN 46
#define FRF_AB_TX_IPFIL23_PORT_EN_WIDTH 1
#define FRF_AB_TX_IPFIL22_PORT_EN_LBN 44
#define FRF_AB_TX_IPFIL22_PORT_EN_WIDTH 1
#define FRF_AB_TX_IPFIL21_PORT_EN_LBN 42
#define FRF_AB_TX_IPFIL21_PORT_EN_WIDTH 1
#define FRF_AB_TX_IPFIL20_PORT_EN_LBN 40
#define FRF_AB_TX_IPFIL20_PORT_EN_WIDTH 1
```



```
#define FRF_AB_TX_IPFIL19_PORT_EN_LBN 38
#define FRF_AB_TX_IPFIL19_PORT_EN_WIDTH 1
#define FRF_AB_TX_IPFIL18_PORT_EN_LBN 36
#define FRF_AB_TX_IPFIL18_PORT_EN_WIDTH 1
#define FRF_AB_TX_IPFIL17_PORT_EN_LBN 34
#define FRF_AB_TX_IPFIL17_PORT_EN_WIDTH 1
#define FRF_AB_TX_IPFIL16_PORT_EN_LBN 32
#define FRF_AB_TX_IPFIL16_PORT_EN_WIDTH 1
#define FRF_AB_TX_IPFIL15_PORT_EN_LBN 30
#define FRF_AB_TX_IPFIL15_PORT_EN_WIDTH 1
#define FRF_AB_TX_IPFIL14_PORT_EN_LBN 28
#define FRF_AB_TX_IPFIL14_PORT_EN_WIDTH 1
#define FRF_AB_TX_IPFIL13_PORT_EN_LBN 26
#define FRF_AB_TX_IPFIL13_PORT_EN_WIDTH 1
#define FRF_AB_TX_IPFIL12_PORT_EN_LBN 24
#define FRF_AB_TX_IPFIL12_PORT_EN_WIDTH 1
#define FRF_AB_TX_IPFIL11_PORT_EN_LBN 22
#define FRF_AB_TX_IPFIL11_PORT_EN_WIDTH 1
#define FRF_AB_TX_IPFIL10_PORT_EN_LBN 20
#define FRF_AB_TX_IPFIL10_PORT_EN_WIDTH 1
#define FRF_AB_TX_IPFIL9_PORT_EN_LBN 18
#define FRF_AB_TX_IPFIL9_PORT_EN_WIDTH 1
#define FRF_AB_TX_IPFIL8_PORT_EN_LBN 16
#define FRF_AB_TX_IPFIL8_PORT_EN_WIDTH 1
#define FRF_AB_TX_IPFIL7_PORT_EN_LBN 14
#define FRF_AB_TX_IPFIL7_PORT_EN_WIDTH 1
#define FRF_AB_TX_IPFIL6_PORT_EN_LBN 12
#define FRF_AB_TX_IPFIL6_PORT_EN_WIDTH 1
#define FRF_AB_TX_IPFIL5_PORT_EN_LBN 10
#define FRF_AB_TX_IPFIL5_PORT_EN_WIDTH 1
#define FRF_AB_TX_IPFIL4_PORT_EN_LBN 8
#define FRF_AB_TX_IPFIL4_PORT_EN_WIDTH 1
#define FRF_AB_TX_IPFIL3_PORT_EN_LBN 6
#define FRF_AB_TX_IPFIL3_PORT_EN_WIDTH 1
#define FRF_AB_TX_IPFIL2_PORT_EN_LBN 4
#define FRF_AB_TX_IPFIL2_PORT_EN_WIDTH 1
#define FRF_AB_TX_IPFIL1_PORT_EN_LBN 2
#define FRF_AB_TX_IPFIL1_PORT_EN_WIDTH 1
#define FRF_AB_TX_IPFIL0_PORT_EN_LBN 0
#define FRF_AB_TX_IPFIL0_PORT_EN_WIDTH 1


/*
 * FR_AB_TX_IPFIL_TBL(128bit):
 * Transmit IP source address filter table
 */
#define FR_AB_TX_IPFIL_TBL_OFST 0x00000b00
/* falconb0=net_func_bar2,falcona0=char_func_bar0 */
#define FR_AB_TX_IPFIL_TBL_STEP 16
#define FR_AB_TX_IPFIL_TBL_ROWS 16

#define FRF_AB_TX_IPFIL_MASK_1_LBN 96
#define FRF_AB_TX_IPFIL_MASK_1_WIDTH 32
#define FRF_AB_TX_IP_SRC_ADR_1_LBN 64
#define FRF_AB_TX_IP_SRC_ADR_1_WIDTH 32
#define FRF_AB_TX_IPFIL_MASK_0_LBN 32
#define FRF_AB_TX_IPFIL_MASK_0_WIDTH 32
#define FRF_AB_TX_IP_SRC_ADR_0_LBN 0
#define FRF_AB_TX_IP_SRC_ADR_0_WIDTH 32


/*
 * FR_AB_MD_TXD_REG(128bit):
 * PHY management transmit data register
 */
#define FR_AB_MD_TXD_REG_OFST 0x00000c00
```

```
2108 /* falcona0,falconb0=net_func_bar2,falcona0=char_func_bar0 */

2110 #define FRF_AB_MD_TXD_LBN 0
2111 #define FRF_AB_MD_TXD_WIDTH 16


2114 /*
2115  * FR_AB_MD_RXD_REG(128bit):
2116  * PHY management receive data register
2117  */
2118 #define FR_AB_MD_RXD_REG_OFST 0x00000c10
2119 /* falcona0,falconb0=net_func_bar2,falcona0=char_func_bar0 */

2121 #define FRF_AB_MD_RXD_LBN 0
2122 #define FRF_AB_MD_RXD_WIDTH 16


2125 /*
2126  * FR_AB_MD_CS_REG(128bit):
2127  * PHY management configuration & status register
2128  */
2129 #define FR_AB_MD_CS_REG_OFST 0x00000c20
2130 /* falcona0,falconb0=net_func_bar2,falcona0=char_func_bar0 */

2132 #define FRF_AB_MD_RD_EN_LBN 15
2133 #define FRF_AB_MD_RD_EN_WIDTH 1
2134 #define FRF_AB_MD_WR_EN_LBN 14
2135 #define FRF_AB_MD_WR_EN_WIDTH 1
2136 #define FRF_AB_MD_ADDR_CMD_LBN 13
2137 #define FRF_AB_MD_ADDR_CMD_WIDTH 1
2138 #define FRF_AB_MD_PT_LBN 7
2139 #define FRF_AB_MD_PT_WIDTH 3
2140 #define FRF_AB_MD_PL_LBN 6
2141 #define FRF_AB_MD_PL_WIDTH 1
2142 #define FRF_AB_MD_INT_CLR_LBN 5
2143 #define FRF_AB_MD_INT_CLR_WIDTH 1
2144 #define FRF_AB_MD_GC_LBN 4
2145 #define FRF_AB_MD_GC_WIDTH 1
2146 #define FRF_AB_MD_PRSP_LBN 3
2147 #define FRF_AB_MD_PRSP_WIDTH 1
2148 #define FRF_AB_MD_RIC_LBN 2
2149 #define FRF_AB_MD_RIC_WIDTH 1
2150 #define FRF_AB_MD_RDC_LBN 1
2151 #define FRF_AB_MD_RDC_WIDTH 1
2152 #define FRF_AB_MD_WRC_LBN 0
2153 #define FRF_AB_MD_WRC_WIDTH 1


2156 /*
2157  * FR_AB_MD_PHY_ADR_REG(128bit):
2158  * PHY management PHY address register
2159  */
2160 #define FR_AB_MD_PHY_ADR_REG_OFST 0x00000c30
2161 /* falcona0,falconb0=net_func_bar2,falcona0=char_func_bar0 */

2163 #define FRF_AB_MD_PHY_ADR_LBN 0
2164 #define FRF_AB_MD_PHY_ADR_WIDTH 16


2167 /*
2168  * FR_AB_MD_ID_REG(128bit):
2169  * PHY management ID register
2170  */
2171 #define FR_AB_MD_ID_REG_OFST 0x00000c40
2172 /* falcona0,falconb0=net_func_bar2,falcona0=char_func_bar0 */
```



```
2174 #define FRF_AB_MD_PRT_ADR_LBN 11
2175 #define FRF_AB_MD_PRT_ADR_WIDTH 5
2176 #define FRF_AB_MD_DEV_ADR_LBN 6
2177 #define FRF_AB_MD_DEV_ADR_WIDTH 5


2180 /*
2181  * FR_AB_MD_STAT_REG(128bit):
2182  * PHY management status & mask register
2183  */
2184 #define FR_AB_MD_STAT_REG_OFST 0x00000c50
2185 /* falcona0,falconb0=net_func_bar2,falcona0=char_func_bar0 */

2187 #define FRF_AB_MD_PINT_LBN 4
2188 #define FRF_AB_MD_PINT_WIDTH 1
2189 #define FRF_AB_MD_DONE_LBN 3
2190 #define FRF_AB_MD_DONE_WIDTH 1
2191 #define FRF_AB_MD_BSERR_LBN 2
2192 #define FRF_AB_MD_BSERR_WIDTH 1
2193 #define FRF_AB_MD_LNFL_LBN 1
2194 #define FRF_AB_MD_LNFL_WIDTH 1
2195 #define FRF_AB_MD_BSY_LBN 0
2196 #define FRF_AB_MD_BSY_WIDTH 1


2199 /*
2200  * FR_AB_MAC_STAT_DMA_REG(128bit):
2201  * Port MAC statistical counter DMA register
2202  */
2203 #define FR_AB_MAC_STAT_DMA_REG_OFST 0x00000c60
2204 /* falcona0,falconb0=net_func_bar2,falcona0=char_func_bar0 */

2206 #define FRF_AB_MAC_STAT_DMA_CMD_LBN 48
2207 #define FRF_AB_MAC_STAT_DMA_CMD_WIDTH 1
2208 #define FRF_AB_MAC_STAT_DMA_ADR_LBN 0
2209 #define FRF_AB_MAC_STAT_DMA_ADR_WIDTH 48
2210 #define FRF_AB_MAC_STAT_DMA_ADR_DW0_LBN 0
2211 #define FRF_AB_MAC_STAT_DMA_ADR_DW0_WIDTH 32
2212 #define FRF_AB_MAC_STAT_DMA_ADR_DW1_LBN 32
2213 #define FRF_AB_MAC_STAT_DMA_ADR_DW1_WIDTH 16


2216 /*
2217  * FR_AB_MAC_CTRL_REG(128bit):
2218  * Port MAC control register
2219  */
2220 #define FR_AB_MAC_CTRL_REG_OFST 0x00000c80
2221 /* falcona0,falconb0=net_func_bar2,falcona0=char_func_bar0 */

2223 #define FRF_AB_MAC_XOFF_VAL_LBN 16
2224 #define FRF_AB_MAC_XOFF_VAL_WIDTH 16
2225 #define FRF_BB_TXFIFO_DRAIN_EN_LBN 7
2226 #define FRF_BB_TXFIFO_DRAIN_EN_WIDTH 1
2227 #define FRF_AB_MAC_XG_DISTXCRC_LBN 5
2228 #define FRF_AB_MAC_XG_DISTXCRC_WIDTH 1
2229 #define FRF_AB_MAC_BCAD_ACPT_LBN 4
2230 #define FRF_AB_MAC_BCAD_ACPT_WIDTH 1
2231 #define FRF_AB_MAC_UC_PROM_LBN 3
2232 #define FRF_AB_MAC_UC_PROM_WIDTH 1
2233 #define FRF_AB_MAC_LINK_STATUS_LBN 2
2234 #define FRF_AB_MAC_LINK_STATUS_WIDTH 1
2235 #define FRF_AB_MAC_SPEED_LBN 0
2236 #define FRF_AB_MAC_SPEED_WIDTH 2
2237 #define FRF_AB_MAC_SPEED_10M 0
2238 #define FRF_AB_MAC_SPEED_100M 1
2239 #define FRF_AB_MAC_SPEED_1G 2
```

```
#define FRF_AB_MAC_SPEED_10G 3

/*
 * FR_BB_GEN_MODE_REG(128bit):
 * General Purpose mode register (external interrupt mask)
 */
#define FR_BB_GEN_MODE_REG_OFST 0x00000c90
/* falconb0=net_func_bar2 */

#define FRF_BB_XFP_PHY_INT_POL_SEL_LBN 3
#define FRF_BB_XFP_PHY_INT_POL_SEL_WIDTH 1
#define FRF_BB_XG_PHY_INT_POL_SEL_LBN 2
#define FRF_BB_XG_PHY_INT_POL_SEL_WIDTH 1
#define FRF_BB_XFP_PHY_INT_MASK_LBN 1
#define FRF_BB_XFP_PHY_INT_MASK_WIDTH 1
#define FRF_BB_XG_PHY_INT_MASK_LBN 0
#define FRF_BB_XG_PHY_INT_MASK_WIDTH 1


/*
 * FR_AB_MAC_MC_HASH_REG0(128bit):
 * Multicast address hash table
 */
#define FR_AB_MAC_MC_HASH0_REG_OFST 0x00000ca0
/* falcona0,falconb0=net_func_bar2,falcona0=char_func_bar0 */

#define FRF_AB_MAC_MCAST_HASH0_LBN 0
#define FRF_AB_MAC_MCAST_HASH0_WIDTH 128
#define FRF_AB_MAC_MCAST_HASH0_DW0_LBN 0
#define FRF_AB_MAC_MCAST_HASH0_DW0_WIDTH 32
#define FRF_AB_MAC_MCAST_HASH0_DW1_LBN 32
#define FRF_AB_MAC_MCAST_HASH0_DW1_WIDTH 32
#define FRF_AB_MAC_MCAST_HASH0_DW2_LBN 64
#define FRF_AB_MAC_MCAST_HASH0_DW2_WIDTH 32
#define FRF_AB_MAC_MCAST_HASH0_DW3_LBN 96
#define FRF_AB_MAC_MCAST_HASH0_DW3_WIDTH 32


/*
 * FR_AB_MAC_MC_HASH_REG1(128bit):
 * Multicast address hash table
 */
#define FR_AB_MAC_MC_HASH1_REG_OFST 0x00000cb0
/* falcona0,falconb0=net_func_bar2,falcona0=char_func_bar0 */

#define FRF_AB_MAC_MCAST_HASH1_LBN 0
#define FRF_AB_MAC_MCAST_HASH1_WIDTH 128
#define FRF_AB_MAC_MCAST_HASH1_DW0_LBN 0
#define FRF_AB_MAC_MCAST_HASH1_DW0_WIDTH 32
#define FRF_AB_MAC_MCAST_HASH1_DW1_LBN 32
#define FRF_AB_MAC_MCAST_HASH1_DW1_WIDTH 32
#define FRF_AB_MAC_MCAST_HASH1_DW2_LBN 64
#define FRF_AB_MAC_MCAST_HASH1_DW2_WIDTH 32
#define FRF_AB_MAC_MCAST_HASH1_DW3_LBN 96
#define FRF_AB_MAC_MCAST_HASH1_DW3_WIDTH 32


/*
 * FR_AB_GM_CFG1_REG(32bit):
 * GMAC configuration register 1
 */
#define FR_AB_GM_CFG1_REG_OFST 0x00000e00
/* falcona0,falconb0=net_func_bar2,falcona0=char_func_bar0 */

#define FRF_AB_GM_SW_RST_LBN 31
#define FRF_AB_GM_SW_RST_WIDTH 1
```



```
#define FRF_AB_GM_SIM_RST_LBN 30
#define FRF_AB_GM_SIM_RST_WIDTH 1
#define FRF_AB_GM_RST_RX_MAC_CTL_LBN 19
#define FRF_AB_GM_RST_RX_MAC_CTL_WIDTH 1
#define FRF_AB_GM_RST_TX_MAC_CTL_LBN 18
#define FRF_AB_GM_RST_TX_MAC_CTL_WIDTH 1
#define FRF_AB_GM_RST_RX_FUNC_LBN 17
#define FRF_AB_GM_RST_RX_FUNC_WIDTH 1
#define FRF_AB_GM_RST_TX_FUNC_LBN 16
#define FRF_AB_GM_RST_TX_FUNC_WIDTH 1
#define FRF_AB_GM_LOOP_LBN 8
#define FRF_AB_GM_LOOP_WIDTH 1
#define FRF_AB_GM_RX_FC_EN_LBN 5
#define FRF_AB_GM_RX_FC_EN_WIDTH 1
#define FRF_AB_GM_TX_FC_EN_LBN 4
#define FRF_AB_GM_TX_FC_EN_WIDTH 1
#define FRF_AB_GM_SYNC_RXEN_LBN 3
#define FRF_AB_GM_SYNC_RXEN_WIDTH 1
#define FRF_AB_GM_RX_EN_LBN 2
#define FRF_AB_GM_RX_EN_WIDTH 1
#define FRF_AB_GM_SYNC_TXEN_LBN 1
#define FRF_AB_GM_SYNC_TXEN_WIDTH 1
#define FRF_AB_GM_TX_EN_LBN 0
#define FRF_AB_GM_TX_EN_WIDTH 1


/*
 * FR_AB_GM_CFG2_REG(32bit):
 * GMAC configuration register 2
 */
#define FR_AB_GM_CFG2_REG_OFST 0x00000e10
/* falcona0,falconb0=net_func_bar2,falcona0=char_func_bar0 */

#define FRF_AB_GM_PAMBL_LEN_LBN 12
#define FRF_AB_GM_PAMBL_LEN_WIDTH 4
#define FRF_AB_GM_IF_MODE_LBN 8
#define FRF_AB_GM_IF_MODE_WIDTH 2
#define FRF_AB_GM_IF_MODE_BYTE_MODE 2
#define FRF_AB_GM_IF_MODE_NIBBLE_MODE 1
#define FRF_AB_GM_HUGE_FRM_EN_LBN 5
#define FRF_AB_GM_HUGE_FRM_EN_WIDTH 1
#define FRF_AB_GM_LEN_CHK_LBN 4
#define FRF_AB_GM_LEN_CHK_WIDTH 1
#define FRF_AB_GM_PAD_CRC_EN_LBN 2
#define FRF_AB_GM_PAD_CRC_EN_WIDTH 1
#define FRF_AB_GM_CRC_EN_LBN 1
#define FRF_AB_GM_CRC_EN_WIDTH 1
#define FRF_AB_GM_FD_LBN 0
#define FRF_AB_GM_FD_WIDTH 1


/*
 * FR_AB_GM_IPG_REG(32bit):
 * GMAC IPG register
 */
#define FR_AB_GM_IPG_REG_OFST 0x00000e20
/* falcona0,falconb0=net_func_bar2,falcona0=char_func_bar0 */

#define FRF_AB_GM_NONB2B_IPG1_LBN 24
#define FRF_AB_GM_NONB2B_IPG1_WIDTH 7
#define FRF_AB_GM_NONB2B_IPG2_LBN 16
#define FRF_AB_GM_NONB2B_IPG2_WIDTH 7
#define FRF_AB_GM_MIN_IPG_ENF_LBN 8
#define FRF_AB_GM_MIN_IPG_ENF_WIDTH 8
#define FRF_AB_GM_B2B_IPG_LBN 0
#define FRF_AB_GM_B2B_IPG_WIDTH 7
```

```
2374 /*
2375  * FR_AB_GM_HD_REG(32bit):
2376  * GMAC half duplex register
2377  */
2378 #define FR_AB_GM_HD_REG_OFST 0x00000e30
2379 /* falcona0,falconb0=net_func_bar2,falcona0=char_func_bar0 */

2381 #define FRF_AB_GM_ALT_BOFF_VAL_LBN 20
2382 #define FRF_AB_GM_ALT_BOFF_VAL_WIDTH 4
2383 #define FRF_AB_GM_ALT_BOFF_EN_LBN 19
2384 #define FRF_AB_GM_ALT_BOFF_EN_WIDTH 1
2385 #define FRF_AB_GM_BP_NO_BOFF_LBN 18
2386 #define FRF_AB_GM_BP_NO_BOFF_WIDTH 1
2387 #define FRF_AB_GM_DIS_BOFF_LBN 17
2388 #define FRF_AB_GM_DIS_BOFF_WIDTH 1
2389 #define FRF_AB_GM_EXDEF_TX_EN_LBN 16
2390 #define FRF_AB_GM_EXDEF_TX_EN_WIDTH 1
2391 #define FRF_AB_GM_RTRY_LIMIT_LBN 12
2392 #define FRF_AB_GM_RTRY_LIMIT_WIDTH 4
2393 #define FRF_AB_GM_COL_WIN_LBN 0
2394 #define FRF_AB_GM_COL_WIN_WIDTH 10


2397 /*
2398  * FR_AB_GM_MAX_FLEN_REG(32bit):
2399  * GMAC maximum frame length register
2400  */
2401 #define FR_AB_GM_MAX_FLEN_REG_OFST 0x00000e40
2402 /* falcona0,falconb0=net_func_bar2,falcona0=char_func_bar0 */

2404 #define FRF_AB_GM_MAX_FLEN_LBN 0
2405 #define FRF_AB_GM_MAX_FLEN_WIDTH 16


2408 /*
2409  * FR_AB_GM_TEST_REG(32bit):
2410  * GMAC test register
2411  */
2412 #define FR_AB_GM_TEST_REG_OFST 0x00000e70
2413 /* falcona0,falconb0=net_func_bar2,falcona0=char_func_bar0 */

2415 #define FRF_AB_GM_MAX_BOFF_LBN 3
2416 #define FRF_AB_GM_MAX_BOFF_WIDTH 1
2417 #define FRF_AB_GM_REG_TX_FLOW_EN_LBN 2
2418 #define FRF_AB_GM_REG_TX_FLOW_EN_WIDTH 1
2419 #define FRF_AB_GM_TEST_PAUSE_LBN 1
2420 #define FRF_AB_GM_TEST_PAUSE_WIDTH 1
2421 #define FRF_AB_GM_SHORT_SLOT_LBN 0
2422 #define FRF_AB_GM_SHORT_SLOT_WIDTH 1


2425 /*
2426  * FR_AB_GM_ADR1_REG(32bit):
2427  * GMAC station address register 1
2428  */
2429 #define FR_AB_GM_ADR1_REG_OFST 0x00000f00
2430 /* falcona0,falconb0=net_func_bar2,falcona0=char_func_bar0 */

2432 #define FRF_AB_GM_ADR_B0_LBN 24
2433 #define FRF_AB_GM_ADR_B0_WIDTH 8
2434 #define FRF_AB_GM_ADR_B1_LBN 16
2435 #define FRF_AB_GM_ADR_B1_WIDTH 8
2436 #define FRF_AB_GM_ADR_B2_LBN 8
2437 #define FRF_AB_GM_ADR_B2_WIDTH 8
```



```
2438 #define FRF_AB_GM_ADR_B3_LBN 0
2439 #define FRF_AB_GM_ADR_B3_WIDTH 8


2442 /*
2443  * FR_AB_GM_ADR2_REG(32bit):
2444  * GMAC station address register 2
2445  */
2446 #define FR_AB_GM_ADR2_REG_OFST 0x00000f10
2447 /* falcona0,falconb0=net_func_bar2,falcona0=char_func_bar0 */

2449 #define FRF_AB_GM_ADR_B4_LBN 24
2450 #define FRF_AB_GM_ADR_B4_WIDTH 8
2451 #define FRF_AB_GM_ADR_B5_LBN 16
2452 #define FRF_AB_GM_ADR_B5_WIDTH 8


2455 /*
2456  * FR_AB_GMF_CFG0_REG(32bit):
2457  * GMAC FIFO configuration register 0
2458  */
2459 #define FR_AB_GMF_CFG0_REG_OFST 0x00000f20
2460 /* falcona0,falconb0=net_func_bar2,falcona0=char_func_bar0 */

2462 #define FRF_AB_GMF_FTFENRPLY_LBN 20
2463 #define FRF_AB_GMF_FTFENRPLY_WIDTH 1
2464 #define FRF_AB_GMF_STFENRPLY_LBN 19
2465 #define FRF_AB_GMF_STFENRPLY_WIDTH 1
2466 #define FRF_AB_GMF_FRFENRPLY_LBN 18
2467 #define FRF_AB_GMF_FRFENRPLY_WIDTH 1
2468 #define FRF_AB_GMF_SRFENRPLY_LBN 17
2469 #define FRF_AB_GMF_SRFENRPLY_WIDTH 1
2470 #define FRF_AB_GMF_WTMENRPLY_LBN 16
2471 #define FRF_AB_GMF_WTMENRPLY_WIDTH 1
2472 #define FRF_AB_GMF_FTFENREQ_LBN 12
2473 #define FRF_AB_GMF_FTFENREQ_WIDTH 1
2474 #define FRF_AB_GMF_STFENREQ_LBN 11
2475 #define FRF_AB_GMF_STFENREQ_WIDTH 1
2476 #define FRF_AB_GMF_FRFENREQ_LBN 10
2477 #define FRF_AB_GMF_FRFENREQ_WIDTH 1
2478 #define FRF_AB_GMF_SRFENREQ_LBN 9
2479 #define FRF_AB_GMF_SRFENREQ_WIDTH 1
2480 #define FRF_AB_GMF_WTMENREQ_LBN 8
2481 #define FRF_AB_GMF_WTMENREQ_WIDTH 1
2482 #define FRF_AB_GMF_HSTRSTFT_LBN 4
2483 #define FRF_AB_GMF_HSTRSTFT_WIDTH 1
2484 #define FRF_AB_GMF_HSTRSTST_LBN 3
2485 #define FRF_AB_GMF_HSTRSTST_WIDTH 1
2486 #define FRF_AB_GMF_HSTRSTFR_LBN 2
2487 #define FRF_AB_GMF_HSTRSTFR_WIDTH 1
2488 #define FRF_AB_GMF_HSTRSTSR_LBN 1
2489 #define FRF_AB_GMF_HSTRSTSR_WIDTH 1
2490 #define FRF_AB_GMF_HSTRSTWT_LBN 0
2491 #define FRF_AB_GMF_HSTRSTWT_WIDTH 1


2494 /*
2495  * FR_AB_GMF_CFG1_REG(32bit):
2496  * GMAC FIFO configuration register 1
2497  */
2498 #define FR_AB_GMF_CFG1_REG_OFST 0x00000f30
2499 /* falcona0,falconb0=net_func_bar2,falcona0=char_func_bar0 */

2501 #define FRF_AB_GMF_CFGFRTH_LBN 16
2502 #define FRF_AB_GMF_CFGFRTH_WIDTH 5
2503 #define FRF_AB_GMF_CFGXOFFRTX_LBN 0
```

```
#define FRF_AB_GMF_CFGXOFFRTX_WIDTH 16


/*
 * FR_AB_GMF_CFG2_REG(32bit):
 * GMAC FIFO configuration register 2
 */
#define FR_AB_GMF_CFG2_REG_OFST 0x00000f40
/* falcona0,falconb0=net_func_bar2,falcona0=char_func_bar0 */

#define FRF_AB_GMF_CFGHWM_LBN 16
#define FRF_AB_GMF_CFGHWM_WIDTH 6
#define FRF_AB_GMF_CFGLWM_LBN 0
#define FRF_AB_GMF_CFGLWM_WIDTH 6


/*
 * FR_AB_GMF_CFG3_REG(32bit):
 * GMAC FIFO configuration register 3
 */
#define FR_AB_GMF_CFG3_REG_OFST 0x00000f50
/* falcona0,falconb0=net_func_bar2,falcona0=char_func_bar0 */

#define FRF_AB_GMF_CFGHWMFT_LBN 16
#define FRF_AB_GMF_CFGHWMFT_WIDTH 6
#define FRF_AB_GMF_CFGFTTH_LBN 0
#define FRF_AB_GMF_CFGFTTH_WIDTH 6


/*
 * FR_AB_GMF_CFG4_REG(32bit):
 * GMAC FIFO configuration register 4
 */
#define FR_AB_GMF_CFG4_REG_OFST 0x00000f60
/* falcona0,falconb0=net_func_bar2,falcona0=char_func_bar0 */

#define FRF_AB_GMF_HSTFLTRFRM_LBN 0
#define FRF_AB_GMF_HSTFLTRFRM_WIDTH 18


/*
 * FR_AB_GMF_CFG5_REG(32bit):
 * GMAC FIFO configuration register 5
 */
#define FR_AB_GMF_CFG5_REG_OFST 0x00000f70
/* falcona0,falconb0=net_func_bar2,falcona0=char_func_bar0 */

#define FRF_AB_GMF_CFGHDPLX_LBN 22
#define FRF_AB_GMF_CFGHDPLX_WIDTH 1
#define FRF_AB_GMF_SRFULL_LBN 21
#define FRF_AB_GMF_SRFULL_WIDTH 1
#define FRF_AB_GMF_HSTSRFULLCLR_LBN 20
#define FRF_AB_GMF_HSTSRFULLCLR_WIDTH 1
#define FRF_AB_GMF_CFGBYTMODE_LBN 19
#define FRF_AB_GMF_CFGBYTMODE_WIDTH 1
#define FRF_AB_GMF_HSTDRPLT64_LBN 18
#define FRF_AB_GMF_HSTDRPLT64_WIDTH 1
#define FRF_AB_GMF_HSTFLTRFRMDC_LBN 0
#define FRF_AB_GMF_HSTFLTRFRMDC_WIDTH 18


/*
 * FR_BB_TX_SRC_MAC_TBL(128bit):
 * Transmit IP source address filter table
 */
#define FR_BB_TX_SRC_MAC_TBL_OFST 0x00001000
```



```
/* falconb0=net_func_bar2 */
#define FR_BB_TX_SRC_MAC_TBL_STEP 16
#define FR_BB_TX_SRC_MAC_TBL_ROWS 16

#define FRF_BB_TX_SRC_MAC_ADR_1_LBN 64
#define FRF_BB_TX_SRC_MAC_ADR_1_WIDTH 48
#define FRF_BB_TX_SRC_MAC_ADR_1_DW0_LBN 64
#define FRF_BB_TX_SRC_MAC_ADR_1_DW0_WIDTH 32
#define FRF_BB_TX_SRC_MAC_ADR_1_DW1_LBN 96
#define FRF_BB_TX_SRC_MAC_ADR_1_DW1_WIDTH 16
#define FRF_BB_TX_SRC_MAC_ADR_0_LBN 0
#define FRF_BB_TX_SRC_MAC_ADR_0_WIDTH 48
#define FRF_BB_TX_SRC_MAC_ADR_0_DW0_LBN 0
#define FRF_BB_TX_SRC_MAC_ADR_0_DW0_WIDTH 32
#define FRF_BB_TX_SRC_MAC_ADR_0_DW1_LBN 32
#define FRF_BB_TX_SRC_MAC_ADR_0_DW1_WIDTH 16


/*
 * FR_BB_TX_SRC_MAC_CTL_REG(128bit):
 * Transmit MAC source address filter control
 */
#define FR_BB_TX_SRC_MAC_CTL_REG_OFST 0x00001100
/* falconb0=net_func_bar2 */

#define FRF_BB_TX_SRC_DROP_CTR_LBN 16
#define FRF_BB_TX_SRC_DROP_CTR_WIDTH 16
#define FRF_BB_TX_SRC_FLTR_EN_LBN 15
#define FRF_BB_TX_SRC_FLTR_EN_WIDTH 1
#define FRF_BB_TX_DROP_CTR_CLR_LBN 12
#define FRF_BB_TX_DROP_CTR_CLR_WIDTH 1
#define FRF_BB_TX_MAC_QID_SEL_LBN 0
#define FRF_BB_TX_MAC_QID_SEL_WIDTH 3


/*
 * FR_AB_XM_ADR_LO_REG(128bit):
 * XGMAC address register low
 */
#define FR_AB_XM_ADR_LO_REG_OFST 0x00001200
/* falcona0,falconb0=net_func_bar2,falcona0=char_func_bar0 */

#define FRF_AB_XM_ADR_LO_LBN 0
#define FRF_AB_XM_ADR_LO_WIDTH 32


/*
 * FR_AB_XM_ADR_HI_REG(128bit):
 * XGMAC address register high
 */
#define FR_AB_XM_ADR_HI_REG_OFST 0x00001210
/* falcona0,falconb0=net_func_bar2,falcona0=char_func_bar0 */

#define FRF_AB_XM_ADR_HI_LBN 0
#define FRF_AB_XM_ADR_HI_WIDTH 16


/*
 * FR_AB_XM_GLB_CFG_REG(128bit):
 * XGMAC global configuration
 */
#define FR_AB_XM_GLB_CFG_REG_OFST 0x00001220
/* falcona0,falconb0=net_func_bar2,falcona0=char_func_bar0 */

#define FRF_AB_XM_RMTFLT_GEN_LBN 17
#define FRF_AB_XM_RMTFLT_GEN_WIDTH 1
```

```
2636 #define FRF_AB_XM_DEBUG_MODE_LBN 16
2637 #define FRF_AB_XM_DEBUG_MODE_WIDTH 1
2638 #define FRF_AB_XM_RX_STAT_EN_LBN 11
2639 #define FRF_AB_XM_RX_STAT_EN_WIDTH 1
2640 #define FRF_AB_XM_TX_STAT_EN_LBN 10
2641 #define FRF_AB_XM_TX_STAT_EN_WIDTH 1
2642 #define FRF_AB_XM_RX_JUMBO_MODE_LBN 6
2643 #define FRF_AB_XM_RX_JUMBO_MODE_WIDTH 1
2644 #define FRF_AB_XM_WAN_MODE_LBN 5
2645 #define FRF_AB_XM_WAN_MODE_WIDTH 1
2646 #define FRF_AB_XM_INTCLR_MODE_LBN 3
2647 #define FRF_AB_XM_INTCLR_MODE_WIDTH 1
2648 #define FRF_AB_XM_CORE_RST_LBN 0
2649 #define FRF_AB_XM_CORE_RST_WIDTH 1


2652 /*
2653  * FR_AB_XM_TX_CFG_REG(128bit):
2654  * XGMAC transmit configuration
2655  */
2656 #define FR_AB_XM_TX_CFG_REG_OFST 0x00001230
2657 /* falcona0,falconb0=net_func_bar2,falcona0=char_func_bar0 */

2659 #define FRF_AB_XM_TX_PROG_LBN 24
2660 #define FRF_AB_XM_TX_PROG_WIDTH 1
2661 #define FRF_AB_XM_IPG_LBN 16
2662 #define FRF_AB_XM_IPG_WIDTH 4
2663 #define FRF_AB_XM_FCNTL_LBN 10
2664 #define FRF_AB_XM_FCNTL_WIDTH 1
2665 #define FRF_AB_XM_TXCRC_LBN 8
2666 #define FRF_AB_XM_TXCRC_WIDTH 1
2667 #define FRF_AB_XM_EDRC_LBN 6
2668 #define FRF_AB_XM_EDRC_WIDTH 1
2669 #define FRF_AB_XM_AUTO_PAD_LBN 5
2670 #define FRF_AB_XM_AUTO_PAD_WIDTH 1
2671 #define FRF_AB_XM_TX_PRMBL_LBN 2
2672 #define FRF_AB_XM_TX_PRMBL_WIDTH 1
2673 #define FRF_AB_XM_TXEN_LBN 1
2674 #define FRF_AB_XM_TXEN_WIDTH 1
2675 #define FRF_AB_XM_TX_RST_LBN 0
2676 #define FRF_AB_XM_TX_RST_WIDTH 1


2679 /*
2680  * FR_AB_XM_RX_CFG_REG(128bit):
2681  * XGMAC receive configuration
2682  */
2683 #define FR_AB_XM_RX_CFG_REG_OFST 0x00001240
2684 /* falcona0,falconb0=net_func_bar2,falcona0=char_func_bar0 */

2686 #define FRF_AB_XM_PASS_LENERR_LBN 26
2687 #define FRF_AB_XM_PASS_LENERR_WIDTH 1
2688 #define FRF_AB_XM_PASS_CRC_ERR_LBN 25
2689 #define FRF_AB_XM_PASS_CRC_ERR_WIDTH 1
2690 #define FRF_AB_XM_PASS_PRMBLE_ERR_LBN 24
2691 #define FRF_AB_XM_PASS_PRMBLE_ERR_WIDTH 1
2692 #define FRF_AB_XM_REJ_BCAST_LBN 20
2693 #define FRF_AB_XM_REJ_BCAST_WIDTH 1
2694 #define FRF_AB_XM_ACPT_ALL_MCAST_LBN 11
2695 #define FRF_AB_XM_ACPT_ALL_MCAST_WIDTH 1
2696 #define FRF_AB_XM_ACPT_ALL_UCAST_LBN 9
2697 #define FRF_AB_XM_ACPT_ALL_UCAST_WIDTH 1
2698 #define FRF_AB_XM_AUTO_DEPAD_LBN 8
2699 #define FRF_AB_XM_AUTO_DEPAD_WIDTH 1
2700 #define FRF_AB_XM_RXCRC_LBN 3
2701 #define FRF_AB_XM_RXCRC_WIDTH 1
```



```
2702 #define FRF_AB_XM_RX_PRMBL_LBN 2
2703 #define FRF_AB_XM_RX_PRMBL_WIDTH 1
2704 #define FRF_AB_XM_RXEN_LBN 1
2705 #define FRF_AB_XM_RXEN_WIDTH 1
2706 #define FRF_AB_XM_RX_RST_LBN 0
2707 #define FRF_AB_XM_RX_RST_WIDTH 1


2710 /*
2711  * FR_AB_XM_MGT_INT_MASK(128bit):
2712  * documentation to be written for sum_XM_MGT_INT_MASK
2713  */
2714 #define FR_AB_XM_MGT_INT_MASK_OFST 0x00001250
2715 /* falcona0,falconb0=net_func_bar2,falcona0=char_func_bar0 */

2717 #define FRF_AB_XM_MSK_STA_INTR_LBN 16
2718 #define FRF_AB_XM_MSK_STA_INTR_WIDTH 1
2719 #define FRF_AB_XM_MSK_STAT_CNTR_HF_LBN 9
2720 #define FRF_AB_XM_MSK_STAT_CNTR_HF_WIDTH 1
2721 #define FRF_AB_XM_MSK_STAT_CNTR_OF_LBN 8
2722 #define FRF_AB_XM_MSK_STAT_CNTR_OF_WIDTH 1
2723 #define FRF_AB_XM_MSK_PRMBLE_ERR_LBN 2
2724 #define FRF_AB_XM_MSK_PRMBLE_ERR_WIDTH 1
2725 #define FRF_AB_XM_MSK_RMTFLT_LBN 1
2726 #define FRF_AB_XM_MSK_RMTFLT_WIDTH 1
2727 #define FRF_AB_XM_MSK_LCLFLT_LBN 0
2728 #define FRF_AB_XM_MSK_LCLFLT_WIDTH 1


2731 /*
2732  * FR_AB_XM_FC_REG(128bit):
2733  * XGMAC flow control register
2734  */
2735 #define FR_AB_XM_FC_REG_OFST 0x00001270
2736 /* falcona0,falconb0=net_func_bar2,falcona0=char_func_bar0 */

2738 #define FRF_AB_XM_PAUSE_TIME_LBN 16
2739 #define FRF_AB_XM_PAUSE_TIME_WIDTH 16
2740 #define FRF_AB_XM_RX_MAC_STAT_LBN 11
2741 #define FRF_AB_XM_RX_MAC_STAT_WIDTH 1
2742 #define FRF_AB_XM_TX_MAC_STAT_LBN 10
2743 #define FRF_AB_XM_TX_MAC_STAT_WIDTH 1
2744 #define FRF_AB_XM_MCNTL_PASS_LBN 8
2745 #define FRF_AB_XM_MCNTL_PASS_WIDTH 2
2746 #define FRF_AB_XM_REJ_CNTL_UCAST_LBN 6
2747 #define FRF_AB_XM_REJ_CNTL_UCAST_WIDTH 1
2748 #define FRF_AB_XM_REJ_CNTL_MCAST_LBN 5
2749 #define FRF_AB_XM_REJ_CNTL_MCAST_WIDTH 1
2750 #define FRF_AB_XM_ZPAUSE_LBN 2
2751 #define FRF_AB_XM_ZPAUSE_WIDTH 1
2752 #define FRF_AB_XM_XMIT_PAUSE_LBN 1
2753 #define FRF_AB_XM_XMIT_PAUSE_WIDTH 1
2754 #define FRF_AB_XM_DIS_FCNTL_LBN 0
2755 #define FRF_AB_XM_DIS_FCNTL_WIDTH 1


2758 /*
2759  * FR_AB_XM_PAUSE_TIME_REG(128bit):
2760  * XGMAC pause time register
2761  */
2762 #define FR_AB_XM_PAUSE_TIME_REG_OFST 0x00001290
2763 /* falcona0,falconb0=net_func_bar2,falcona0=char_func_bar0 */

2765 #define FRF_AB_XM_TX_PAUSE_CNT_LBN 16
2766 #define FRF_AB_XM_TX_PAUSE_CNT_WIDTH 16
2767 #define FRF_AB_XM_RX_PAUSE_CNT_LBN 0
```

```
#define FRF_AB_XM_RX_PAUSE_CNT_WIDTH 16


/*
 * FR_AB_XM_TX_PARAM_REG(128bit):
 * XGMAC transmit parameter register
 */
#define FR_AB_XM_TX_PARAM_REG_OFST 0x000012d0
/* falcona0,falconb0=net_func_bar2,falcona0=char_func_bar0 */

#define FRF_AB_XM_TX_JUMBO_MODE_LBN 31
#define FRF_AB_XM_TX_JUMBO_MODE_WIDTH 1
#define FRF_AB_XM_MAX_TX_FRM_SIZE_HI_LBN 19
#define FRF_AB_XM_MAX_TX_FRM_SIZE_HI_WIDTH 11
#define FRF_AB_XM_MAX_TX_FRM_SIZE_LO_LBN 16
#define FRF_AB_XM_MAX_TX_FRM_SIZE_LO_WIDTH 3
#define FRF_AB_XM_PAD_CHAR_LBN 0
#define FRF_AB_XM_PAD_CHAR_WIDTH 8


/*
 * FR_AB_XM_RX_PARAM_REG(128bit):
 * XGMAC receive parameter register
 */
#define FR_AB_XM_RX_PARAM_REG_OFST 0x000012e0
/* falcona0,falconb0=net_func_bar2,falcona0=char_func_bar0 */

#define FRF_AB_XM_MAX_RX_FRM_SIZE_HI_LBN 3
#define FRF_AB_XM_MAX_RX_FRM_SIZE_HI_WIDTH 11
#define FRF_AB_XM_MAX_RX_FRM_SIZE_LO_LBN 0
#define FRF_AB_XM_MAX_RX_FRM_SIZE_LO_WIDTH 3


/*
 * FR_AB_XM_MGT_INT_MSK_REG(128bit):
 * XGMAC management interrupt mask register
 */
#define FR_AB_XM_MGT_INT_REG_OFST 0x000012f0
/* falcona0,falconb0=net_func_bar2,falcona0=char_func_bar0 */

#define FRF_AB_XM_STAT_CNTR_OF_LBN 9
#define FRF_AB_XM_STAT_CNTR_OF_WIDTH 1
#define FRF_AB_XM_STAT_CNTR_HF_LBN 8
#define FRF_AB_XM_STAT_CNTR_HF_WIDTH 1
#define FRF_AB_XM_PRMBLE_ERR_LBN 2
#define FRF_AB_XM_PRMBLE_ERR_WIDTH 1
#define FRF_AB_XM_RMTFLT_LBN 1
#define FRF_AB_XM_RMTFLT_WIDTH 1
#define FRF_AB_XM_LCLFLT_LBN 0
#define FRF_AB_XM_LCLFLT_WIDTH 1


/*
 * FR_AB_XX_PWR_RST_REG(128bit):
 * XGXS/XAUI powerdown/reset register
 */
#define FR_AB_XX_PWR_RST_REG_OFST 0x00001300
/* falcona0,falconb0=net_func_bar2,falcona0=char_func_bar0 */

#define FRF_AB_XX_PWRDND_SIG_LBN 31
#define FRF_AB_XX_PWRDND_SIG_WIDTH 1
#define FRF_AB_XX_PWRDNC_SIG_LBN 30
#define FRF_AB_XX_PWRDNC_SIG_WIDTH 1
#define FRF_AB_XX_PWRDNB_SIG_LBN 29
#define FRF_AB_XX_PWRDNB_SIG_WIDTH 1
#define FRF_AB_XX_PWRDNA_SIG_LBN 28
```



```
#define FRF_AB_XX_PWRDNA_SIG_WIDTH 1
#define FRF_AB_XX_SIM_MODE_LBN 27
#define FRF_AB_XX_SIM_MODE_WIDTH 1
#define FRF_AB_XX_RSTPLLCD_SIG_LBN 25
#define FRF_AB_XX_RSTPLLCD_SIG_WIDTH 1
#define FRF_AB_XX_RSTPLLAB_SIG_LBN 24
#define FRF_AB_XX_RSTPLLAB_SIG_WIDTH 1
#define FRF_AB_XX_RESETD_SIG_LBN 23
#define FRF_AB_XX_RESETD_SIG_WIDTH 1
#define FRF_AB_XX_RESETC_SIG_LBN 22
#define FRF_AB_XX_RESETC_SIG_WIDTH 1
#define FRF_AB_XX_RESETB_SIG_LBN 21
#define FRF_AB_XX_RESETB_SIG_WIDTH 1
#define FRF_AB_XX_RESETA_SIG_LBN 20
#define FRF_AB_XX_RESETA_SIG_WIDTH 1
#define FRF_AB_XX_RSTXGXSRX_SIG_LBN 18
#define FRF_AB_XX_RSTXGXSRX_SIG_WIDTH 1
#define FRF_AB_XX_RSTXGXSTX_SIG_LBN 17
#define FRF_AB_XX_RSTXGXSTX_SIG_WIDTH 1
#define FRF_AB_XX_SD_RST_ACT_LBN 16
#define FRF_AB_XX_SD_RST_ACT_WIDTH 1
#define FRF_AB_XX_PWRDND_EN_LBN 15
#define FRF_AB_XX_PWRDND_EN_WIDTH 1
#define FRF_AB_XX_PWRDNC_EN_LBN 14
#define FRF_AB_XX_PWRDNC_EN_WIDTH 1
#define FRF_AB_XX_PWRDNB_EN_LBN 13
#define FRF_AB_XX_PWRDNB_EN_WIDTH 1
#define FRF_AB_XX_PWRDNA_EN_LBN 12
#define FRF_AB_XX_PWRDNA_EN_WIDTH 1
#define FRF_AB_XX_RSTPLLCD_EN_LBN 9
#define FRF_AB_XX_RSTPLLCD_EN_WIDTH 1
#define FRF_AB_XX_RSTPLLAB_EN_LBN 8
#define FRF_AB_XX_RSTPLLAB_EN_WIDTH 1
#define FRF_AB_XX_RESETD_EN_LBN 7
#define FRF_AB_XX_RESETD_EN_WIDTH 1
#define FRF_AB_XX_RESETC_EN_LBN 6
#define FRF_AB_XX_RESETC_EN_WIDTH 1
#define FRF_AB_XX_RESETB_EN_LBN 5
#define FRF_AB_XX_RESETB_EN_WIDTH 1
#define FRF_AB_XX_RESETA_EN_LBN 4
#define FRF_AB_XX_RESETA_EN_WIDTH 1
#define FRF_AB_XX_RSTXGXSRX_EN_LBN 2
#define FRF_AB_XX_RSTXGXSRX_EN_WIDTH 1
#define FRF_AB_XX_RSTXGXSTX_EN_LBN 1
#define FRF_AB_XX_RSTXGXSTX_EN_WIDTH 1
#define FRF_AB_XX_RST_XX_EN_LBN 0
#define FRF_AB_XX_RST_XX_EN_WIDTH 1


/*
 * FR_AB_XX_SD_CTL_REG(128bit):
 * XGXS/XAUI powerdown/reset control register
 */
#define FR_AB_XX_SD_CTL_REG_OFST 0x00001310
/* falcona0,falconb0=net_func_bar2,falcona0=char_func_bar0 */

#define FRF_AB_XX_TERMADJ1_LBN 17
#define FRF_AB_XX_TERMADJ1_WIDTH 1
#define FRF_AB_XX_TERMADJ0_LBN 16
#define FRF_AB_XX_TERMADJ0_WIDTH 1
#define FRF_AB_XX_HIDRVD_LBN 15
#define FRF_AB_XX_HIDRVD_WIDTH 1
#define FRF_AB_XX_LODRVD_LBN 14
#define FRF_AB_XX_LODRVD_WIDTH 1
#define FRF_AB_XX_HIDRVC_LBN 13
#define FRF_AB_XX_HIDRVC_WIDTH 1
```

```
#define FRF_AB_XX_LODRVC_LBN 12
#define FRF_AB_XX_LODRVC_WIDTH 1
#define FRF_AB_XX_HIDRVB_LBN 11
#define FRF_AB_XX_HIDRVB_WIDTH 1
#define FRF_AB_XX_LODRVB_LBN 10
#define FRF_AB_XX_LODRVB_WIDTH 1
#define FRF_AB_XX_HIDRVA_LBN 9
#define FRF_AB_XX_HIDRVA_WIDTH 1
#define FRF_AB_XX_LODRVA_LBN 8
#define FRF_AB_XX_LODRVA_WIDTH 1
#define FRF_AB_XX_LPBKD_LBN 3
#define FRF_AB_XX_LPBKD_WIDTH 1
#define FRF_AB_XX_LPBKC_LBN 2
#define FRF_AB_XX_LPBKC_WIDTH 1
#define FRF_AB_XX_LPBKB_LBN 1
#define FRF_AB_XX_LPBKB_WIDTH 1
#define FRF_AB_XX_LPBKA_LBN 0
#define FRF_AB_XX_LPBKA_WIDTH 1


/*
 * FR_AB_XX_TXDRV_CTL_REG(128bit):
 * XAUI SerDes transmit drive control register
 */
#define FR_AB_XX_TXDRV_CTL_REG_OFST 0x00001320
/* falcona0,falconb0=net_func_bar2,falcona0=char_func_bar0 */

#define FRF_AB_XX_DEQD_LBN 28
#define FRF_AB_XX_DEQD_WIDTH 4
#define FRF_AB_XX_DEQC_LBN 24
#define FRF_AB_XX_DEQC_WIDTH 4
#define FRF_AB_XX_DEQB_LBN 20
#define FRF_AB_XX_DEQB_WIDTH 4
#define FRF_AB_XX_DEQA_LBN 16
#define FRF_AB_XX_DEQA_WIDTH 4
#define FRF_AB_XX_DTXD_LBN 12
#define FRF_AB_XX_DTXD_WIDTH 4
#define FRF_AB_XX_DTXC_LBN 8
#define FRF_AB_XX_DTXC_WIDTH 4
#define FRF_AB_XX_DTXB_LBN 4
#define FRF_AB_XX_DTXB_WIDTH 4
#define FRF_AB_XX_DTXA_LBN 0
#define FRF_AB_XX_DTXA_WIDTH 4


/*
 * FR_AB_XX_PRBS_CTL_REG(128bit):
 * documentation to be written for sum_XX_PRBS_CTL_REG
 */
#define FR_AB_XX_PRBS_CTL_REG_OFST 0x00001330
/* falcona0,falconb0=net_func_bar2,falcona0=char_func_bar0 */

#define FRF_AB_XX_CH3_RX_PRBS_SEL_LBN 30
#define FRF_AB_XX_CH3_RX_PRBS_SEL_WIDTH 2
#define FRF_AB_XX_CH3_RX_PRBS_INV_LBN 29
#define FRF_AB_XX_CH3_RX_PRBS_INV_WIDTH 1
#define FRF_AB_XX_CH3_RX_PRBS_CHKEN_LBN 28
#define FRF_AB_XX_CH3_RX_PRBS_CHKEN_WIDTH 1
#define FRF_AB_XX_CH2_RX_PRBS_SEL_LBN 26
#define FRF_AB_XX_CH2_RX_PRBS_SEL_WIDTH 2
#define FRF_AB_XX_CH2_RX_PRBS_INV_LBN 25
#define FRF_AB_XX_CH2_RX_PRBS_INV_WIDTH 1
#define FRF_AB_XX_CH2_RX_PRBS_CHKEN_LBN 24
#define FRF_AB_XX_CH2_RX_PRBS_CHKEN_WIDTH 1
#define FRF_AB_XX_CH1_RX_PRBS_SEL_LBN 22
#define FRF_AB_XX_CH1_RX_PRBS_SEL_WIDTH 2
```



```
#define FRF_AB_XX_CH1_RX_PRBS_INV_LBN 21
#define FRF_AB_XX_CH1_RX_PRBS_INV_WIDTH 1
#define FRF_AB_XX_CH1_RX_PRBS_CHKEN_LBN 20
#define FRF_AB_XX_CH1_RX_PRBS_CHKEN_WIDTH 1
#define FRF_AB_XX_CH0_RX_PRBS_SEL_LBN 18
#define FRF_AB_XX_CH0_RX_PRBS_SEL_WIDTH 2
#define FRF_AB_XX_CH0_RX_PRBS_INV_LBN 17
#define FRF_AB_XX_CH0_RX_PRBS_INV_WIDTH 1
#define FRF_AB_XX_CH0_RX_PRBS_CHKEN_LBN 16
#define FRF_AB_XX_CH0_RX_PRBS_CHKEN_WIDTH 1
#define FRF_AB_XX_CH3_TX_PRBS_SEL_LBN 14
#define FRF_AB_XX_CH3_TX_PRBS_SEL_WIDTH 2
#define FRF_AB_XX_CH3_TX_PRBS_INV_LBN 13
#define FRF_AB_XX_CH3_TX_PRBS_INV_WIDTH 1
#define FRF_AB_XX_CH3_TX_PRBS_CHKEN_LBN 12
#define FRF_AB_XX_CH3_TX_PRBS_CHKEN_WIDTH 1
#define FRF_AB_XX_CH2_TX_PRBS_SEL_LBN 10
#define FRF_AB_XX_CH2_TX_PRBS_SEL_WIDTH 2
#define FRF_AB_XX_CH2_TX_PRBS_INV_LBN 9
#define FRF_AB_XX_CH2_TX_PRBS_INV_WIDTH 1
#define FRF_AB_XX_CH2_TX_PRBS_CHKEN_LBN 8
#define FRF_AB_XX_CH2_TX_PRBS_CHKEN_WIDTH 1
#define FRF_AB_XX_CH1_TX_PRBS_SEL_LBN 6
#define FRF_AB_XX_CH1_TX_PRBS_SEL_WIDTH 2
#define FRF_AB_XX_CH1_TX_PRBS_INV_LBN 5
#define FRF_AB_XX_CH1_TX_PRBS_INV_WIDTH 1
#define FRF_AB_XX_CH1_TX_PRBS_CHKEN_LBN 4
#define FRF_AB_XX_CH1_TX_PRBS_CHKEN_WIDTH 1
#define FRF_AB_XX_CH0_TX_PRBS_SEL_LBN 2
#define FRF_AB_XX_CH0_TX_PRBS_SEL_WIDTH 2
#define FRF_AB_XX_CH0_TX_PRBS_INV_LBN 1
#define FRF_AB_XX_CH0_TX_PRBS_INV_WIDTH 1
#define FRF_AB_XX_CH0_TX_PRBS_CHKEN_LBN 0
#define FRF_AB_XX_CH0_TX_PRBS_CHKEN_WIDTH 1


/*
 * FR_AB_XX_PRBS_CHK_REG(128bit):
 * documentation to be written for sum_XX_PRBS_CHK_REG
 */
#define FR_AB_XX_PRBS_CHK_REG_OFST 0x00001340
/* falcona0,falconb0=net_func_bar2,falcona0=char_func_bar0 */

#define FRF_AB_XX_REV_LB_EN_LBN 16
#define FRF_AB_XX_REV_LB_EN_WIDTH 1
#define FRF_AB_XX_CH3_DEG_DET_LBN 15
#define FRF_AB_XX_CH3_DEG_DET_WIDTH 1
#define FRF_AB_XX_CH3_LFSR_LOCK_IND_LBN 14
#define FRF_AB_XX_CH3_LFSR_LOCK_IND_WIDTH 1
#define FRF_AB_XX_CH3_PRBS_FRUN_LBN 13
#define FRF_AB_XX_CH3_PRBS_FRUN_WIDTH 1
#define FRF_AB_XX_CH3_ERR_CHK_LBN 12
#define FRF_AB_XX_CH3_ERR_CHK_WIDTH 1
#define FRF_AB_XX_CH2_DEG_DET_LBN 11
#define FRF_AB_XX_CH2_DEG_DET_WIDTH 1
#define FRF_AB_XX_CH2_LFSR_LOCK_IND_LBN 10
#define FRF_AB_XX_CH2_LFSR_LOCK_IND_WIDTH 1
#define FRF_AB_XX_CH2_PRBS_FRUN_LBN 9
#define FRF_AB_XX_CH2_PRBS_FRUN_WIDTH 1
#define FRF_AB_XX_CH2_ERR_CHK_LBN 8
#define FRF_AB_XX_CH2_ERR_CHK_WIDTH 1
#define FRF_AB_XX_CH1_DEG_DET_LBN 7
#define FRF_AB_XX_CH1_DEG_DET_WIDTH 1
#define FRF_AB_XX_CH1_LFSR_LOCK_IND_LBN 6
#define FRF_AB_XX_CH1_LFSR_LOCK_IND_WIDTH 1
#define FRF_AB_XX_CH1_PRBS_FRUN_LBN 5
```

```
#define FRF_AB_XX_CH1_PRBS_FRUN_WIDTH 1
#define FRF_AB_XX_CH1_ERR_CHK_LBN 4
#define FRF_AB_XX_CH1_ERR_CHK_WIDTH 1
#define FRF_AB_XX_CH0_DEG_DET_LBN 3
#define FRF_AB_XX_CH0_DEG_DET_WIDTH 1
#define FRF_AB_XX_CH0_LFSR_LOCK_IND_LBN 2
#define FRF_AB_XX_CH0_LFSR_LOCK_IND_WIDTH 1
#define FRF_AB_XX_CH0_PRBS_FRUN_LBN 1
#define FRF_AB_XX_CH0_PRBS_FRUN_WIDTH 1
#define FRF_AB_XX_CH0_ERR_CHK_LBN 0
#define FRF_AB_XX_CH0_ERR_CHK_WIDTH 1


/*
 * FR_AB_XX_PRBS_ERR_REG(128bit):
 * documentation to be written for sum_XX_PRBS_ERR_REG
 */
#define FR_AB_XX_PRBS_ERR_REG_OFST 0x00001350
/* falcona0,falconb0=net_func_bar2,falcona0=char_func_bar0 */

#define FRF_AB_XX_CH3_PRBS_ERR_CNT_LBN 24
#define FRF_AB_XX_CH3_PRBS_ERR_CNT_WIDTH 8
#define FRF_AB_XX_CH2_PRBS_ERR_CNT_LBN 16
#define FRF_AB_XX_CH2_PRBS_ERR_CNT_WIDTH 8
#define FRF_AB_XX_CH1_PRBS_ERR_CNT_LBN 8
#define FRF_AB_XX_CH1_PRBS_ERR_CNT_WIDTH 8
#define FRF_AB_XX_CH0_PRBS_ERR_CNT_LBN 0
#define FRF_AB_XX_CH0_PRBS_ERR_CNT_WIDTH 8


/*
 * FR_AB_XX_CORE_STAT_REG(128bit):
 * XAUI XGXS core status register
 */
#define FR_AB_XX_CORE_STAT_REG_OFST 0x00001360
/* falcona0,falconb0=net_func_bar2,falcona0=char_func_bar0 */

#define FRF_AB_XX_FORCE_SIG3_LBN 31
#define FRF_AB_XX_FORCE_SIG3_WIDTH 1
#define FRF_AB_XX_FORCE_SIG3_VAL_LBN 30
#define FRF_AB_XX_FORCE_SIG3_VAL_WIDTH 1
#define FRF_AB_XX_FORCE_SIG2_LBN 29
#define FRF_AB_XX_FORCE_SIG2_WIDTH 1
#define FRF_AB_XX_FORCE_SIG2_VAL_LBN 28
#define FRF_AB_XX_FORCE_SIG2_VAL_WIDTH 1
#define FRF_AB_XX_FORCE_SIG1_LBN 27
#define FRF_AB_XX_FORCE_SIG1_WIDTH 1
#define FRF_AB_XX_FORCE_SIG1_VAL_LBN 26
#define FRF_AB_XX_FORCE_SIG1_VAL_WIDTH 1
#define FRF_AB_XX_FORCE_SIG0_LBN 25
#define FRF_AB_XX_FORCE_SIG0_WIDTH 1
#define FRF_AB_XX_FORCE_SIG0_VAL_LBN 24
#define FRF_AB_XX_FORCE_SIG0_VAL_WIDTH 1
#define FRF_AB_XX_XGXS_LB_EN_LBN 23
#define FRF_AB_XX_XGXS_LB_EN_WIDTH 1
#define FRF_AB_XX_XGMII_LB_EN_LBN 22
#define FRF_AB_XX_XGMII_LB_EN_WIDTH 1
#define FRF_AB_XX_MATCH_FAULT_LBN 21
#define FRF_AB_XX_MATCH_FAULT_WIDTH 1
#define FRF_AB_XX_ALIGN_DONE_LBN 20
#define FRF_AB_XX_ALIGN_DONE_WIDTH 1
#define FRF_AB_XX_SYNC_STAT3_LBN 19
#define FRF_AB_XX_SYNC_STAT3_WIDTH 1
#define FRF_AB_XX_SYNC_STAT2_LBN 18
#define FRF_AB_XX_SYNC_STAT2_WIDTH 1
#define FRF_AB_XX_SYNC_STAT1_LBN 17
```



```
#define FRF_AB_XX_SYNC_STAT1_WIDTH 1
#define FRF_AB_XX_SYNC_STAT0_LBN 16
#define FRF_AB_XX_SYNC_STAT0_WIDTH 1
#define FRF_AB_XX_COMMA_DET_CH3_LBN 15
#define FRF_AB_XX_COMMA_DET_CH3_WIDTH 1
#define FRF_AB_XX_COMMA_DET_CH2_LBN 14
#define FRF_AB_XX_COMMA_DET_CH2_WIDTH 1
#define FRF_AB_XX_COMMA_DET_CH1_LBN 13
#define FRF_AB_XX_COMMA_DET_CH1_WIDTH 1
#define FRF_AB_XX_COMMA_DET_CH0_LBN 12
#define FRF_AB_XX_COMMA_DET_CH0_WIDTH 1
#define FRF_AB_XX_CGRP_ALIGN_CH3_LBN 11
#define FRF_AB_XX_CGRP_ALIGN_CH3_WIDTH 1
#define FRF_AB_XX_CGRP_ALIGN_CH2_LBN 10
#define FRF_AB_XX_CGRP_ALIGN_CH2_WIDTH 1
#define FRF_AB_XX_CGRP_ALIGN_CH1_LBN 9
#define FRF_AB_XX_CGRP_ALIGN_CH1_WIDTH 1
#define FRF_AB_XX_CGRP_ALIGN_CH0_LBN 8
#define FRF_AB_XX_CGRP_ALIGN_CH0_WIDTH 1
#define FRF_AB_XX_CHAR_ERR_CH3_LBN 7
#define FRF_AB_XX_CHAR_ERR_CH3_WIDTH 1
#define FRF_AB_XX_CHAR_ERR_CH2_LBN 6
#define FRF_AB_XX_CHAR_ERR_CH2_WIDTH 1
#define FRF_AB_XX_CHAR_ERR_CH1_LBN 5
#define FRF_AB_XX_CHAR_ERR_CH1_WIDTH 1
#define FRF_AB_XX_CHAR_ERR_CH0_LBN 4
#define FRF_AB_XX_CHAR_ERR_CH0_WIDTH 1
#define FRF_AB_XX_DISPERR_CH3_LBN 3
#define FRF_AB_XX_DISPERR_CH3_WIDTH 1
#define FRF_AB_XX_DISPERR_CH2_LBN 2
#define FRF_AB_XX_DISPERR_CH2_WIDTH 1
#define FRF_AB_XX_DISPERR_CH1_LBN 1
#define FRF_AB_XX_DISPERR_CH1_WIDTH 1
#define FRF_AB_XX_DISPERR_CH0_LBN 0
#define FRF_AB_XX_DISPERR_CH0_WIDTH 1


/*
 * FR_AA_RX_DESC_PTR_TBL_KER(128bit):
 * Receive descriptor pointer table
 */
#define FR_AA_RX_DESC_PTR_TBL_KER_OFST 0x00011800
/* falcona0=net_func_bar2 */
#define FR_AA_RX_DESC_PTR_TBL_KER_STEP 16
#define FR_AA_RX_DESC_PTR_TBL_KER_ROWS 4
/*
 * FR_AZ_RX_DESC_PTR_TBL(128bit):
 * Receive descriptor pointer table
 */
#define FR_AZ_RX_DESC_PTR_TBL_OFST 0x00f40000
/* sienaa0=net_func_bar2,falconb0=net_func_bar2,falcona0=char_func_bar0 */
#define FR_AZ_RX_DESC_PTR_TBL_STEP 16
#define FR_CZ_RX_DESC_PTR_TBL_ROWS 1024
#define FR_AB_RX_DESC_PTR_TBL_ROWS 4096

#define FRF_CZ_RX_HDR_SPLIT_LBN 90
#define FRF_CZ_RX_HDR_SPLIT_WIDTH 1
#define FRF_AZ_RX_RESET_LBN 89
#define FRF_AZ_RX_RESET_WIDTH 1
#define FRF_AZ_RX_ISCSI_DDIG_EN_LBN 88
#define FRF_AZ_RX_ISCSI_DDIG_EN_WIDTH 1
#define FRF_AZ_RX_ISCSI_HDIG_EN_LBN 87
#define FRF_AZ_RX_ISCSI_HDIG_EN_WIDTH 1
#define FRF_AZ_RX_DESC_PREF_ACT_LBN 86
#define FRF_AZ_RX_DESC_PREF_ACT_WIDTH 1
#define FRF_AZ_RX_DC_HW_RPTR_LBN 80
```

```
#define FRF_AZ_RX_DC_HW_RPTR_WIDTH 6
#define FRF_AZ_RX_DESCQ_HW_RPTR_LBN 68
#define FRF_AZ_RX_DESCQ_HW_RPTR_WIDTH 12
#define FRF_AZ_RX_DESCQ_SW_WPTR_LBN 56
#define FRF_AZ_RX_DESCQ_SW_WPTR_WIDTH 12
#define FRF_AZ_RX_DESCQ_BUF_BASE_ID_LBN 36
#define FRF_AZ_RX_DESCQ_BUF_BASE_ID_WIDTH 20
#define FRF_AZ_RX_DESCQ_EVQ_ID_LBN 24
#define FRF_AZ_RX_DESCQ_EVQ_ID_WIDTH 12
#define FRF_AZ_RX_DESCQ_OWNER_ID_LBN 10
#define FRF_AZ_RX_DESCQ_OWNER_ID_WIDTH 14
#define FRF_AZ_RX_DESCQ_LABEL_LBN 5
#define FRF_AZ_RX_DESCQ_LABEL_WIDTH 5
#define FRF_AZ_RX_DESCQ_SIZE_LBN 3
#define FRF_AZ_RX_DESCQ_SIZE_WIDTH 2
#define FFE_AZ_RX_DESCQ_SIZE_4K 3
#define FFE_AZ_RX_DESCQ_SIZE_2K 2
#define FFE_AZ_RX_DESCQ_SIZE_1K 1
#define FFE_AZ_RX_DESCQ_SIZE_512 0
#define FRF_AZ_RX_DESCQ_TYPE_LBN 2
#define FRF_AZ_RX_DESCQ_TYPE_WIDTH 1
#define FRF_AZ_RX_DESCQ_JUMBO_LBN 1
#define FRF_AZ_RX_DESCQ_JUMBO_WIDTH 1
#define FRF_AZ_RX_DESCQ_EN_LBN 0
#define FRF_AZ_RX_DESCQ_EN_WIDTH 1


/*
 * FR_AA_TX_DESC_PTR_TBL_KER(128bit):
 * Transmit descriptor pointer
 */
#define FR_AA_TX_DESC_PTR_TBL_KER_OFST 0x00011900
/* falcona0=net_func_bar2 */
#define FR_AA_TX_DESC_PTR_TBL_KER_STEP 16
#define FR_AA_TX_DESC_PTR_TBL_KER_ROWS 8
/*
 * FR_AZ_TX_DESC_PTR_TBL(128bit):
 * Transmit descriptor pointer
 */
#define FR_AZ_TX_DESC_PTR_TBL_OFST 0x00f50000
/* falconb0=net_func_bar2,sienaa0=net_func_bar2,falcona0=char_func_bar0 */
#define FR_AZ_TX_DESC_PTR_TBL_STEP 16
#define FR_AB_TX_DESC_PTR_TBL_ROWS 4096
#define FR_CZ_TX_DESC_PTR_TBL_ROWS 1024

#define FRF_CZ_TX_DPT_Q_MASK_WIDTH_LBN 94
#define FRF_CZ_TX_DPT_Q_MASK_WIDTH_WIDTH 2
#define FRF_CZ_TX_DPT_ETH_FILT_EN_LBN 93
#define FRF_CZ_TX_DPT_ETH_FILT_EN_WIDTH 1
#define FRF_CZ_TX_DPT_IP_FILT_EN_LBN 92
#define FRF_CZ_TX_DPT_IP_FILT_EN_WIDTH 1
#define FRF_BZ_TX_NON_IP_DROP_DIS_LBN 91
#define FRF_BZ_TX_NON_IP_DROP_DIS_WIDTH 1
#define FRF_BZ_TX_IP_CHKSM_DIS_LBN 90
#define FRF_BZ_TX_IP_CHKSM_DIS_WIDTH 1
#define FRF_BZ_TX_TCP_CHKSM_DIS_LBN 89
#define FRF_BZ_TX_TCP_CHKSM_DIS_WIDTH 1
#define FRF_AZ_TX_DESCQ_EN_LBN 88
#define FRF_AZ_TX_DESCQ_EN_WIDTH 1
#define FRF_AZ_TX_ISCSI_DDIG_EN_LBN 87
#define FRF_AZ_TX_ISCSI_DDIG_EN_WIDTH 1
#define FRF_AZ_TX_ISCSI_HDIG_EN_LBN 86
#define FRF_AZ_TX_ISCSI_HDIG_EN_WIDTH 1
#define FRF_AZ_TX_DC_HW_RPTR_LBN 80
#define FRF_AZ_TX_DC_HW_RPTR_WIDTH 6
#define FRF_AZ_TX_DESCQ_HW_RPTR_LBN 68
```



```
#define FRF_AZ_TX_DESCQ_HW_RPTR_WIDTH 12
#define FRF_AZ_TX_DESCQ_SW_WPTR_LBN 56
#define FRF_AZ_TX_DESCQ_SW_WPTR_WIDTH 12
#define FRF_AZ_TX_DESCQ_BUF_BASE_ID_LBN 36
#define FRF_AZ_TX_DESCQ_BUF_BASE_ID_WIDTH 20
#define FRF_AZ_TX_DESCQ_EVQ_ID_LBN 24
#define FRF_AZ_TX_DESCQ_EVQ_ID_WIDTH 12
#define FRF_AZ_TX_DESCQ_OWNER_ID_LBN 10
#define FRF_AZ_TX_DESCQ_OWNER_ID_WIDTH 14
#define FRF_AZ_TX_DESCQ_LABEL_LBN 5
#define FRF_AZ_TX_DESCQ_LABEL_WIDTH 5
#define FRF_AZ_TX_DESCQ_SIZE_LBN 3
#define FRF_AZ_TX_DESCQ_SIZE_WIDTH 2
#define FFE_AZ_TX_DESCQ_SIZE_4K 3
#define FFE_AZ_TX_DESCQ_SIZE_2K 2
#define FFE_AZ_TX_DESCQ_SIZE_1K 1
#define FFE_AZ_TX_DESCQ_SIZE_512 0
#define FRF_AZ_TX_DESCQ_TYPE_LBN 1
#define FRF_AZ_TX_DESCQ_TYPE_WIDTH 2
#define FRF_AZ_TX_DESCQ_FLUSH_LBN 0
#define FRF_AZ_TX_DESCQ_FLUSH_WIDTH 1


/*
 * FR_AA_EVQ_PTR_TBL_KER(128bit):
 * Event queue pointer table
 */
#define FR_AA_EVQ_PTR_TBL_KER_OFST 0x00011a00
/* falcona0=net_func_bar2 */
#define FR_AA_EVQ_PTR_TBL_KER_STEP 16
#define FR_AA_EVQ_PTR_TBL_KER_ROWS 4
/*
 * FR_AZ_EVQ_PTR_TBL(128bit):
 * Event queue pointer table
 */
#define FR_AZ_EVQ_PTR_TBL_OFST 0x00f60000
/* sienaa0=net_func_bar2,falconb0=net_func_bar2,falcona0=char_func_bar0 */
#define FR_AZ_EVQ_PTR_TBL_STEP 16
#define FR_CZ_EVQ_PTR_TBL_ROWS 1024
#define FR_AB_EVQ_PTR_TBL_ROWS 4096

#define FRF_BZ_EVQ_RPTR_IGN_LBN 40
#define FRF_BZ_EVQ_RPTR_IGN_WIDTH 1
#define FRF_AZ_EVQ_WKUP_OR_INT_EN_LBN 39
#define FRF_AZ_EVQ_WKUP_OR_INT_EN_WIDTH 1
#define FRF_AZ_EVQ_NXT_WPTR_LBN 24
#define FRF_AZ_EVQ_NXT_WPTR_WIDTH 15
#define FRF_AZ_EVQ_EN_LBN 23
#define FRF_AZ_EVQ_EN_WIDTH 1
#define FRF_AZ_EVQ_SIZE_LBN 20
#define FRF_AZ_EVQ_SIZE_WIDTH 3
#define FFE_AZ_EVQ_SIZE_32K 6
#define FFE_AZ_EVQ_SIZE_16K 5
#define FFE_AZ_EVQ_SIZE_8K 4
#define FFE_AZ_EVQ_SIZE_4K 3
#define FFE_AZ_EVQ_SIZE_2K 2
#define FFE_AZ_EVQ_SIZE_1K 1
#define FFE_AZ_EVQ_SIZE_512 0
#define FRF_AZ_EVQ_BUF_BASE_ID_LBN 0
#define FRF_AZ_EVQ_BUF_BASE_ID_WIDTH 20


/*
 * FR_AA_BUF_HALF_TBL_KER(64bit):
 * Buffer table in half buffer table mode direct access by driver
 */
```

```
#define FR_AA_BUF_HALF_TBL_KER_OFST 0x00018000
/* falcona0=net_func_bar2 */
#define FR_AA_BUF_HALF_TBL_KER_STEP 8
#define FR_AA_BUF_HALF_TBL_KER_ROWS 4096
/*
 * FR_AZ_BUF_HALF_TBL(64bit):
 * Buffer table in half buffer table mode direct access by driver
 */
#define FR_AZ_BUF_HALF_TBL_OFST 0x00800000
/* sienaa0=net_func_bar2,falconb0=net_func_bar2,falcona0=char_func_bar0 */
#define FR_AZ_BUF_HALF_TBL_STEP 8
#define FR_CZ_BUF_HALF_TBL_ROWS 147456
#define FR_AB_BUF_HALF_TBL_ROWS 524288

#define FRF_AZ_BUF_ADR_HBUF_ODD_LBN 44
#define FRF_AZ_BUF_ADR_HBUF_ODD_WIDTH 20
#define FRF_AZ_BUF_OWNER_ID_HBUF_ODD_LBN 32
#define FRF_AZ_BUF_OWNER_ID_HBUF_ODD_WIDTH 12
#define FRF_AZ_BUF_ADR_HBUF_EVEN_LBN 12
#define FRF_AZ_BUF_ADR_HBUF_EVEN_WIDTH 20
#define FRF_AZ_BUF_OWNER_ID_HBUF_EVEN_LBN 0
#define FRF_AZ_BUF_OWNER_ID_HBUF_EVEN_WIDTH 12


/*
 * FR_AA_BUF_FULL_TBL_KER(64bit):
 * Buffer table in full buffer table mode direct access by driver
 */
#define FR_AA_BUF_FULL_TBL_KER_OFST 0x00018000
/* falcona0=net_func_bar2 */
#define FR_AA_BUF_FULL_TBL_KER_STEP 8
#define FR_AA_BUF_FULL_TBL_KER_ROWS 4096
/*
 * FR_AZ_BUF_FULL_TBL(64bit):
 * Buffer table in full buffer table mode direct access by driver
 */
#define FR_AZ_BUF_FULL_TBL_OFST 0x00800000
/* sienaa0=net_func_bar2,falconb0=net_func_bar2,falcona0=char_func_bar0 */
#define FR_AZ_BUF_FULL_TBL_STEP 8

#define FR_CZ_BUF_FULL_TBL_ROWS 147456
#define FR_AB_BUF_FULL_TBL_ROWS 917504

#define FRF_AZ_BUF_FULL_UNUSED_LBN 51
#define FRF_AZ_BUF_FULL_UNUSED_WIDTH 13
#define FRF_AZ_IP_DAT_BUF_SIZE_LBN 50
#define FRF_AZ_IP_DAT_BUF_SIZE_WIDTH 1
#define FRF_AZ_BUF_ADR_REGION_LBN 48
#define FRF_AZ_BUF_ADR_REGION_WIDTH 2
#define FFE_AZ_BUF_ADR_REGN3 3
#define FFE_AZ_BUF_ADR_REGN2 2
#define FFE_AZ_BUF_ADR_REGN1 1
#define FFE_AZ_BUF_ADR_REGN0 0
#define FRF_AZ_BUF_ADR_FBUF_LBN 14
#define FRF_AZ_BUF_ADR_FBUF_WIDTH 34
#define FRF_AZ_BUF_ADR_FBUF_DW0_LBN 14
#define FRF_AZ_BUF_ADR_FBUF_DW0_WIDTH 32
#define FRF_AZ_BUF_ADR_FBUF_DW1_LBN 46
#define FRF_AZ_BUF_ADR_FBUF_DW1_WIDTH 2
#define FRF_AZ_BUF_OWNER_ID_FBUF_LBN 0
#define FRF_AZ_BUF_OWNER_ID_FBUF_WIDTH 14


/*
 * FR_AZ_RX_FILTER_TBL0(128bit):
 * TCP/IPv4 Receive filter table
```



```
 */
#define FR_AZ_RX_FILTER_TBL0_OFST 0x00f00000
/* falconb0,sienaa0=net_func_bar2,falcona0=char_func_bar0 */
#define FR_AZ_RX_FILTER_TBL0_STEP 32
#define FR_AZ_RX_FILTER_TBL0_ROWS 8192
/*
 * FR_AB_RX_FILTER_TBL1(128bit):
 * TCP/IPv4 Receive filter table
 */
#define FR_AB_RX_FILTER_TBL1_OFST 0x00f00010
/* falconb0=net_func_bar2,falcona0=char_func_bar0 */
#define FR_AB_RX_FILTER_TBL1_STEP 32
#define FR_AB_RX_FILTER_TBL1_ROWS 8192

#define FRF_BZ_RSS_EN_LBN 110
#define FRF_BZ_RSS_EN_WIDTH 1
#define FRF_BZ_SCATTER_EN_LBN 109
#define FRF_BZ_SCATTER_EN_WIDTH 1
#define FRF_AZ_TCP_UDP_LBN 108
#define FRF_AZ_TCP_UDP_WIDTH 1
#define FRF_AZ_RXQ_ID_LBN 96
#define FRF_AZ_RXQ_ID_WIDTH 12
#define FRF_AZ_DEST_IP_LBN 64
#define FRF_AZ_DEST_IP_WIDTH 32
#define FRF_AZ_DEST_PORT_TCP_LBN 48
#define FRF_AZ_DEST_PORT_TCP_WIDTH 16
#define FRF_AZ_SRC_IP_LBN 16
#define FRF_AZ_SRC_IP_WIDTH 32
#define FRF_AZ_SRC_TCP_DEST_UDP_LBN 0
#define FRF_AZ_SRC_TCP_DEST_UDP_WIDTH 16


/*
 * FR_CZ_RX_MAC_FILTER_TBL0(128bit):
 * Receive Ethernet filter table
 */
#define FR_CZ_RX_MAC_FILTER_TBL0_OFST 0x00f00010
/* sienaa0=net_func_bar2 */
#define FR_CZ_RX_MAC_FILTER_TBL0_STEP 32
#define FR_CZ_RX_MAC_FILTER_TBL0_ROWS 512

#define FRF_CZ_RMFT_RSS_EN_LBN 75
#define FRF_CZ_RMFT_RSS_EN_WIDTH 1
#define FRF_CZ_RMFT_SCATTER_EN_LBN 74
#define FRF_CZ_RMFT_SCATTER_EN_WIDTH 1
#define FRF_CZ_RMFT_IP_OVERRIDE_LBN 73
#define FRF_CZ_RMFT_IP_OVERRIDE_WIDTH 1
#define FRF_CZ_RMFT_RXQ_ID_LBN 61
#define FRF_CZ_RMFT_RXQ_ID_WIDTH 12
#define FRF_CZ_RMFT_WILDCARD_MATCH_LBN 60
#define FRF_CZ_RMFT_WILDCARD_MATCH_WIDTH 1
#define FRF_CZ_RMFT_DEST_MAC_LBN 12
#define FRF_CZ_RMFT_DEST_MAC_WIDTH 48
#define FRF_CZ_RMFT_DEST_MAC_DW0_LBN 12
#define FRF_CZ_RMFT_DEST_MAC_DW0_WIDTH 32
#define FRF_CZ_RMFT_DEST_MAC_DW1_LBN 44
#define FRF_CZ_RMFT_DEST_MAC_DW1_WIDTH 16
#define FRF_CZ_RMFT_VLAN_ID_LBN 0
#define FRF_CZ_RMFT_VLAN_ID_WIDTH 12


/*
 * FR_AZ_TIMER_TBL(128bit):
 * Timer table
 */
#define FR_AZ_TIMER_TBL_OFST 0x00f70000
```

```
/* sienaa0=net_func_bar2,falconb0=net_func_bar2,falcona0=char_func_bar0 */
#define FR_AZ_TIMER_TBL_STEP 16
#define FR_CZ_TIMER_TBL_ROWS 1024
#define FR_AB_TIMER_TBL_ROWS 4096

#define FRF_CZ_TIMER_Q_EN_LBN 33
#define FRF_CZ_TIMER_Q_EN_WIDTH 1
#define FRF_CZ_INT_ARMD_LBN 32
#define FRF_CZ_INT_ARMD_WIDTH 1
#define FRF_CZ_INT_PEND_LBN 31
#define FRF_CZ_INT_PEND_WIDTH 1
#define FRF_CZ_HOST_NOTIFY_MODE_LBN 30
#define FRF_CZ_HOST_NOTIFY_MODE_WIDTH 1
#define FRF_CZ_RELOAD_TIMER_VAL_LBN 16
#define FRF_CZ_RELOAD_TIMER_VAL_WIDTH 14
#define FRF_CZ_TIMER_MODE_LBN 14
#define FRF_CZ_TIMER_MODE_WIDTH 2
#define FFE_CZ_TIMER_MODE_INT_HLDOFF 3
#define FFE_CZ_TIMER_MODE_TRIG_START 2
#define FFE_CZ_TIMER_MODE_IMMED_START 1
#define FFE_CZ_TIMER_MODE_DIS 0
#define FRF_AB_TIMER_MODE_LBN 12
#define FRF_AB_TIMER_MODE_WIDTH 2
#define FFE_AB_TIMER_MODE_INT_HLDOFF 2
#define FFE_AB_TIMER_MODE_TRIG_START 2
#define FFE_AB_TIMER_MODE_IMMED_START 1
#define FFE_AB_TIMER_MODE_DIS 0
#define FRF_CZ_TIMER_VAL_LBN 0
#define FRF_CZ_TIMER_VAL_WIDTH 14
#define FRF_AB_TIMER_VAL_LBN 0
#define FRF_AB_TIMER_VAL_WIDTH 12


/*
 * FR_BZ_TX_PACE_TBL(128bit):
 * Transmit pacing table
 */
#define FR_BZ_TX_PACE_TBL_OFST 0x00f80000
/* sienaa0=net_func_bar2,falconb0=net_func_bar2 */
#define FR_AZ_TX_PACE_TBL_STEP 16
#define FR_CZ_TX_PACE_TBL_ROWS 1024
#define FR_BB_TX_PACE_TBL_ROWS 4096
/*
 * FR_AA_TX_PACE_TBL(128bit):
 * Transmit pacing table
 */
#define FR_AA_TX_PACE_TBL_OFST 0x00f80040
/* falcona0=char_func_bar0 */
/* FR_AZ_TX_PACE_TBL_STEP 16 */
#define FR_AA_TX_PACE_TBL_ROWS 4092

#define FRF_AZ_TX_PACE_LBN 0
#define FRF_AZ_TX_PACE_WIDTH 5


/*
 * FR_BZ_RX_INDIRECTION_TBL(7bit):
 * RX Indirection Table
 */
#define FR_BZ_RX_INDIRECTION_TBL_OFST 0x00fb0000
/* falconb0,sienaa0=net_func_bar2 */
#define FR_BZ_RX_INDIRECTION_TBL_STEP 16
#define FR_BZ_RX_INDIRECTION_TBL_ROWS 128

#define FRF_BZ_IT_QUEUE_LBN 0
#define FRF_BZ_IT_QUEUE_WIDTH 6
```



```


/*
 * FR_CZ_TX_FILTER_TBL0(128bit):
 * TCP/IPv4 Transmit filter table
 */
#define FR_CZ_TX_FILTER_TBL0_OFST 0x00fc0000
/* sienaa0=net_func_bar2 */
#define FR_CZ_TX_FILTER_TBL0_STEP 16
#define FR_CZ_TX_FILTER_TBL0_ROWS 8192

#define FRF_CZ_TIFT_TCP_UDP_LBN 108
#define FRF_CZ_TIFT_TCP_UDP_WIDTH 1
#define FRF_CZ_TIFT_TXQ_ID_LBN 96
#define FRF_CZ_TIFT_TXQ_ID_WIDTH 12
#define FRF_CZ_TIFT_DEST_IP_LBN 64
#define FRF_CZ_TIFT_DEST_IP_WIDTH 32
#define FRF_CZ_TIFT_DEST_PORT_TCP_LBN 48
#define FRF_CZ_TIFT_DEST_PORT_TCP_WIDTH 16
#define FRF_CZ_TIFT_SRC_IP_LBN 16
#define FRF_CZ_TIFT_SRC_IP_WIDTH 32
#define FRF_CZ_TIFT_SRC_TCP_DEST_UDP_LBN 0
#define FRF_CZ_TIFT_SRC_TCP_DEST_UDP_WIDTH 16


/*
 * FR_CZ_TX_MAC_FILTER_TBL0(128bit):
 * Transmit Ethernet filter table
 */
#define FR_CZ_TX_MAC_FILTER_TBL0_OFST 0x00fe0000
/* sienaa0=net_func_bar2 */
#define FR_CZ_TX_MAC_FILTER_TBL0_STEP 16
#define FR_CZ_TX_MAC_FILTER_TBL0_ROWS 512

#define FRF_CZ_TMFT_TXQ_ID_LBN 61
#define FRF_CZ_TMFT_TXQ_ID_WIDTH 12
#define FRF_CZ_TMFT_WILDCARD_MATCH_LBN 60
#define FRF_CZ_TMFT_WILDCARD_MATCH_WIDTH 1
#define FRF_CZ_TMFT_SRC_MAC_LBN 12
#define FRF_CZ_TMFT_SRC_MAC_WIDTH 48
#define FRF_CZ_TMFT_SRC_MAC_DW0_LBN 12
#define FRF_CZ_TMFT_SRC_MAC_DW0_WIDTH 32
#define FRF_CZ_TMFT_SRC_MAC_DW1_LBN 44
#define FRF_CZ_TMFT_SRC_MAC_DW1_WIDTH 16
#define FRF_CZ_TMFT_VLAN_ID_LBN 0
#define FRF_CZ_TMFT_VLAN_ID_WIDTH 12


/*
 * FR_CZ_MC_TREG_SMEM(32bit):
 * MC Shared Memory
 */
#define FR_CZ_MC_TREG_SMEM_OFST 0x00ff0000
/* sienaa0=net_func_bar2 */
#define FR_CZ_MC_TREG_SMEM_STEP 4
#define FR_CZ_MC_TREG_SMEM_ROWS 512

#define FRF_CZ_MC_TREG_SMEM_ROW_LBN 0
#define FRF_CZ_MC_TREG_SMEM_ROW_WIDTH 32


/*
 * FR_BB_MSIX_VECTOR_TABLE(128bit):
 * MSIX Vector Table
 */
#define FR_BB_MSIX_VECTOR_TABLE_OFST 0x00ff0000
```

```
/* falconb0=net_func_bar2 */
#define FR_BZ_MSIX_VECTOR_TABLE_STEP 16
#define FR_BB_MSIX_VECTOR_TABLE_ROWS 64
/*
 * FR_CZ_MSIX_VECTOR_TABLE(128bit):
 * MSIX Vector Table
 */
#define FR_CZ_MSIX_VECTOR_TABLE_OFST 0x00000000
/* sienaa0=pci_f0_bar4 */
/* FR_BZ_MSIX_VECTOR_TABLE_STEP 16 */
#define FR_CZ_MSIX_VECTOR_TABLE_ROWS 1024

#define FRF_BZ_MSIX_VECTOR_RESERVED_LBN 97
#define FRF_BZ_MSIX_VECTOR_RESERVED_WIDTH 31
#define FRF_BZ_MSIX_VECTOR_MASK_LBN 96
#define FRF_BZ_MSIX_VECTOR_MASK_WIDTH 1
#define FRF_BZ_MSIX_MESSAGE_DATA_LBN 64
#define FRF_BZ_MSIX_MESSAGE_DATA_WIDTH 32
#define FRF_BZ_MSIX_MESSAGE_ADDRESS_HI_LBN 32
#define FRF_BZ_MSIX_MESSAGE_ADDRESS_HI_WIDTH 32
#define FRF_BZ_MSIX_MESSAGE_ADDRESS_LO_LBN 0
#define FRF_BZ_MSIX_MESSAGE_ADDRESS_LO_WIDTH 32


/*
 * FR_BB_MSIX_PBA_TABLE(32bit):
 * MSIX Pending Bit Array
 */
#define FR_BB_MSIX_PBA_TABLE_OFST 0x00ff2000
/* falconb0=net_func_bar2 */
#define FR_BZ_MSIX_PBA_TABLE_STEP 4
#define FR_BB_MSIX_PBA_TABLE_ROWS 2
/*
 * FR_CZ_MSIX_PBA_TABLE(32bit):
 * MSIX Pending Bit Array
 */
#define FR_CZ_MSIX_PBA_TABLE_OFST 0x00008000
/* sienaa0=pci_f0_bar4 */
/* FR_BZ_MSIX_PBA_TABLE_STEP 4 */
#define FR_CZ_MSIX_PBA_TABLE_ROWS 32

#define FRF_BZ_MSIX_PBA_PEND_DWORD_LBN 0
#define FRF_BZ_MSIX_PBA_PEND_DWORD_WIDTH 32


/*
 * FR_AZ_SRM_DBG_REG(64bit):
 * SRAM debug access
 */
#define FR_AZ_SRM_DBG_REG_OFST 0x03000000
/* sienaa0=net_func_bar2,falconb0=net_func_bar2,falcona0=char_func_bar0 */
#define FR_AZ_SRM_DBG_REG_STEP 8

#define FR_CZ_SRM_DBG_REG_ROWS 262144
#define FR_AB_SRM_DBG_REG_ROWS 2097152

#define FRF_AZ_SRM_DBG_LBN 0
#define FRF_AZ_SRM_DBG_WIDTH 64
#define FRF_AZ_SRM_DBG_DW0_LBN 0
#define FRF_AZ_SRM_DBG_DW0_WIDTH 32
#define FRF_AZ_SRM_DBG_DW1_LBN 32
#define FRF_AZ_SRM_DBG_DW1_WIDTH 32


/*
 * FR_AA_INT_ACK_CHAR(32bit):
```



```
 * CHAR interrupt acknowledge register
 */
#define FR_AA_INT_ACK_CHAR_OFST 0x00000060
/* falcona0=char_func_bar0 */

#define FRF_AA_INT_ACK_CHAR_FIELD_LBN 0
#define FRF_AA_INT_ACK_CHAR_FIELD_WIDTH 32


/* FS_DRIVER_EV */
#define FSF_AZ_DRIVER_EV_SUBCODE_LBN 56
#define FSF_AZ_DRIVER_EV_SUBCODE_WIDTH 4
#define FSE_AZ_TX_DSC_ERROR_EV 15
#define FSE_AZ_RX_DSC_ERROR_EV 14
#define FSE_AZ_RX_RECOVER_EV 11
#define FSE_AZ_TIMER_EV 10
#define FSE_AZ_TX_PKT_NON_TCP_UDP 9
#define FSE_AZ_WAKE_UP_EV 6
#define FSE_AZ_SRM_UPD_DONE_EV 5
#define FSE_AZ_EVQ_NOT_EN_EV 3
#define FSE_AZ_EVQ_INIT_DONE_EV 2
#define FSE_AZ_RX_DESCQ_FLS_DONE_EV 1
#define FSE_AZ_TX_DESCQ_FLS_DONE_EV 0
#define FSF_AZ_DRIVER_EV_SUBDATA_LBN 0
#define FSF_AZ_DRIVER_EV_SUBDATA_WIDTH 14


/* FS_EVENT_ENTRY */
#define FSF_AZ_EV_CODE_LBN 60
#define FSF_AZ_EV_CODE_WIDTH 4
#define FSE_AZ_EV_CODE_USER_EV 8
#define FSE_AZ_EV_CODE_DRV_GEN_EV 7
#define FSE_AZ_EV_CODE_GLOBAL_EV 6
#define FSE_AZ_EV_CODE_DRIVER_EV 5
#define FSE_AZ_EV_CODE_TX_EV 2
#define FSE_AZ_EV_CODE_RX_EV 0
#define FSF_AZ_EV_DATA_LBN 0
#define FSF_AZ_EV_DATA_WIDTH 60
#define FSF_AZ_EV_DATA_DW0_LBN 0
#define FSF_AZ_EV_DATA_DW0_WIDTH 32
#define FSF_AZ_EV_DATA_DW1_LBN 32
#define FSF_AZ_EV_DATA_DW1_WIDTH 28


/* FS_GLOBAL_EV */
#define FSF_AA_GLB_EV_RX_RECOVERY_LBN 12
#define FSF_AA_GLB_EV_RX_RECOVERY_WIDTH 1
#define FSF_BZ_GLB_EV_XG_MNT_INTR_LBN 11
#define FSF_BZ_GLB_EV_XG_MNT_INTR_WIDTH 1
#define FSF_AZ_GLB_EV_XFP_PHY0_INTR_LBN 10
#define FSF_AZ_GLB_EV_XFP_PHY0_INTR_WIDTH 1
#define FSF_AZ_GLB_EV_XG_PHY0_INTR_LBN 9
#define FSF_AZ_GLB_EV_XG_PHY0_INTR_WIDTH 1
#define FSF_AZ_GLB_EV_G_PHY0_INTR_LBN 7
#define FSF_AZ_GLB_EV_G_PHY0_INTR_WIDTH 1


/* FS_RX_EV */
#define FSF_CZ_RX_EV_PKT_NOT_PARSED_LBN 58
#define FSF_CZ_RX_EV_PKT_NOT_PARSED_WIDTH 1
#define FSF_CZ_RX_EV_IPV6_PKT_LBN 57
#define FSF_CZ_RX_EV_IPV6_PKT_WIDTH 1
#define FSF_AZ_RX_EV_PKT_OK_LBN 56
#define FSF_AZ_RX_EV_PKT_OK_WIDTH 1
#define FSF_AZ_RX_EV_PAUSE_FRM_ERR_LBN 55
#define FSF_AZ_RX_EV_PAUSE_FRM_ERR_WIDTH 1
```

```
#define FSF_AZ_RX_EV_BUF_OWNER_ID_ERR_LBN 54
#define FSF_AZ_RX_EV_BUF_OWNER_ID_ERR_WIDTH 1
#define FSF_AZ_RX_EV_IP_FRAG_ERR_LBN 53
#define FSF_AZ_RX_EV_IP_FRAG_ERR_WIDTH 1
#define FSF_AZ_RX_EV_IP_HDR_CHKSUM_ERR_LBN 52
#define FSF_AZ_RX_EV_IP_HDR_CHKSUM_ERR_WIDTH 1
#define FSF_AZ_RX_EV_TCP_UDP_CHKSUM_ERR_LBN 51
#define FSF_AZ_RX_EV_TCP_UDP_CHKSUM_ERR_WIDTH 1
#define FSF_AZ_RX_EV_ETH_CRC_ERR_LBN 50
#define FSF_AZ_RX_EV_ETH_CRC_ERR_WIDTH 1
#define FSF_AZ_RX_EV_FRM_TRUNC_LBN 49
#define FSF_AZ_RX_EV_FRM_TRUNC_WIDTH 1
#define FSF_AZ_RX_EV_TOBE_DISC_LBN 47
#define FSF_AZ_RX_EV_TOBE_DISC_WIDTH 1
#define FSF_AZ_RX_EV_PKT_TYPE_LBN 44
#define FSF_AZ_RX_EV_PKT_TYPE_WIDTH 3
#define FSE_AZ_RX_EV_PKT_TYPE_VLAN_JUMBO 5
#define FSE_AZ_RX_EV_PKT_TYPE_VLAN_LLC 4
#define FSE_AZ_RX_EV_PKT_TYPE_VLAN 3
#define FSE_AZ_RX_EV_PKT_TYPE_JUMBO 2
#define FSE_AZ_RX_EV_PKT_TYPE_LLC 1
#define FSE_AZ_RX_EV_PKT_TYPE_ETH 0
#define FSF_AZ_RX_EV_HDR_TYPE_LBN 42
#define FSF_AZ_RX_EV_HDR_TYPE_WIDTH 2
#define FSE_AZ_RX_EV_HDR_TYPE_OTHER 3
#define FSE_AZ_RX_EV_HDR_TYPE_IPV4_OTHER 2
#define FSE_AZ_RX_EV_HDR_TYPE_IPV4V6_OTHER 2
#define FSE_AZ_RX_EV_HDR_TYPE_IPV4_UDP 1
#define FSE_AZ_RX_EV_HDR_TYPE_IPV4V6_UDP 1
#define FSE_AZ_RX_EV_HDR_TYPE_IPV4_TCP 0
#define FSE_AZ_RX_EV_HDR_TYPE_IPV4V6_TCP 0
#define FSF_AZ_RX_EV_DESC_Q_EMPTY_LBN 41
#define FSF_AZ_RX_EV_DESC_Q_EMPTY_WIDTH 1
#define FSF_AZ_RX_EV_MCAST_HASH_MATCH_LBN 40
#define FSF_AZ_RX_EV_MCAST_HASH_MATCH_WIDTH 1
#define FSF_AZ_RX_EV_MCAST_PKT_LBN 39
#define FSF_AZ_RX_EV_MCAST_PKT_WIDTH 1
#define FSF_AA_RX_EV_RECOVERY_FLAG_LBN 37
#define FSF_AA_RX_EV_RECOVERY_FLAG_WIDTH 1
#define FSF_AZ_RX_EV_Q_LABEL_LBN 32
#define FSF_AZ_RX_EV_Q_LABEL_WIDTH 5
#define FSF_AZ_RX_EV_JUMBO_CONT_LBN 31
#define FSF_AZ_RX_EV_JUMBO_CONT_WIDTH 1
#define FSF_AZ_RX_EV_PORT_LBN 30
#define FSF_AZ_RX_EV_PORT_WIDTH 1
#define FSF_AZ_RX_EV_BYTE_CNT_LBN 16
#define FSF_AZ_RX_EV_BYTE_CNT_WIDTH 14
#define FSF_AZ_RX_EV_SOP_LBN 15
#define FSF_AZ_RX_EV_SOP_WIDTH 1
#define FSF_AZ_RX_EV_ISCSI_PKT_OK_LBN 14
#define FSF_AZ_RX_EV_ISCSI_PKT_OK_WIDTH 1
#define FSF_AZ_RX_EV_ISCSI_DDIG_ERR_LBN 13
#define FSF_AZ_RX_EV_ISCSI_DDIG_ERR_WIDTH 1
#define FSF_AZ_RX_EV_ISCSI_HDIG_ERR_LBN 12
#define FSF_AZ_RX_EV_ISCSI_HDIG_ERR_WIDTH 1
#define FSF_AZ_RX_EV_DESC_PTR_LBN 0
#define FSF_AZ_RX_EV_DESC_PTR_WIDTH 12


/* FS_RX_KER_DESC */
#define FSF_AZ_RX_KER_BUF_SIZE_LBN 48
#define FSF_AZ_RX_KER_BUF_SIZE_WIDTH 14
#define FSF_AZ_RX_KER_BUF_REGION_LBN 46
#define FSF_AZ_RX_KER_BUF_REGION_WIDTH 2
#define FSF_AZ_RX_KER_BUF_ADDR_LBN 0
#define FSF_AZ_RX_KER_BUF_ADDR_WIDTH 46
```



```
#define FSF_AZ_RX_KER_BUF_ADDR_DW0_LBN 0
#define FSF_AZ_RX_KER_BUF_ADDR_DW0_WIDTH 32
#define FSF_AZ_RX_KER_BUF_ADDR_DW1_LBN 32
#define FSF_AZ_RX_KER_BUF_ADDR_DW1_WIDTH 14


/* FS_RX_USER_DESC */
#define FSF_AZ_RX_USER_2BYTE_OFFSET_LBN 20
#define FSF_AZ_RX_USER_2BYTE_OFFSET_WIDTH 12
#define FSF_AZ_RX_USER_BUF_ID_LBN 0
#define FSF_AZ_RX_USER_BUF_ID_WIDTH 20


/* FS_TX_EV */
#define FSF_AZ_TX_EV_PKT_ERR_LBN 38
#define FSF_AZ_TX_EV_PKT_ERR_WIDTH 1
#define FSF_AZ_TX_EV_PKT_TOO_BIG_LBN 37
#define FSF_AZ_TX_EV_PKT_TOO_BIG_WIDTH 1
#define FSF_AZ_TX_EV_Q_LABEL_LBN 32
#define FSF_AZ_TX_EV_Q_LABEL_WIDTH 5
#define FSF_AZ_TX_EV_PORT_LBN 16
#define FSF_AZ_TX_EV_PORT_WIDTH 1
#define FSF_AZ_TX_EV_WQ_FF_FULL_LBN 15
#define FSF_AZ_TX_EV_WQ_FF_FULL_WIDTH 1
#define FSF_AZ_TX_EV_BUF_OWNER_ID_ERR_LBN 14
#define FSF_AZ_TX_EV_BUF_OWNER_ID_ERR_WIDTH 1
#define FSF_AZ_TX_EV_COMP_LBN 12
#define FSF_AZ_TX_EV_COMP_WIDTH 1
#define FSF_AZ_TX_EV_DESC_PTR_LBN 0
#define FSF_AZ_TX_EV_DESC_PTR_WIDTH 12


/* FS_TX_KER_DESC */
#define FSF_AZ_TX_KER_CONT_LBN 62
#define FSF_AZ_TX_KER_CONT_WIDTH 1
#define FSF_AZ_TX_KER_BYTE_COUNT_LBN 48
#define FSF_AZ_TX_KER_BYTE_COUNT_WIDTH 14
#define FSF_AZ_TX_KER_BUF_REGION_LBN 46
#define FSF_AZ_TX_KER_BUF_REGION_WIDTH 2
#define FSF_AZ_TX_KER_BUF_ADDR_LBN 0
#define FSF_AZ_TX_KER_BUF_ADDR_WIDTH 46
#define FSF_AZ_TX_KER_BUF_ADDR_DW0_LBN 0
#define FSF_AZ_TX_KER_BUF_ADDR_DW0_WIDTH 32
#define FSF_AZ_TX_KER_BUF_ADDR_DW1_LBN 32
#define FSF_AZ_TX_KER_BUF_ADDR_DW1_WIDTH 14


/* FS_TX_USER_DESC */
#define FSF_AZ_TX_USER_SW_EV_EN_LBN 48
#define FSF_AZ_TX_USER_SW_EV_EN_WIDTH 1
#define FSF_AZ_TX_USER_CONT_LBN 46
#define FSF_AZ_TX_USER_CONT_WIDTH 1
#define FSF_AZ_TX_USER_BYTE_CNT_LBN 33
#define FSF_AZ_TX_USER_BYTE_CNT_WIDTH 13
#define FSF_AZ_TX_USER_BUF_ID_LBN 13
#define FSF_AZ_TX_USER_BUF_ID_WIDTH 20
#define FSF_AZ_TX_USER_BYTE_OFS_LBN 0
#define FSF_AZ_TX_USER_BYTE_OFS_WIDTH 13


/* FS_USER_EV */
#define FSF_CZ_USER_QID_LBN 32
#define FSF_CZ_USER_QID_WIDTH 10
#define FSF_CZ_USER_EV_REG_VALUE_LBN 0
#define FSF_CZ_USER_EV_REG_VALUE_WIDTH 32
```

```
3825 /* FS_NET_IVEC */
3826 #define FSF_AZ_NET_IVEC_FATAL_INT_LBN 64
3827 #define FSF_AZ_NET_IVEC_FATAL_INT_WIDTH 1
3828 #define FSF_AZ_NET_IVEC_INT_Q_LBN 40
3829 #define FSF_AZ_NET_IVEC_INT_Q_WIDTH 4
3830 #define FSF_AZ_NET_IVEC_INT_FLAG_LBN 32
3831 #define FSF_AZ_NET_IVEC_INT_FLAG_WIDTH 1
3832 #define FSF_AZ_NET_IVEC_EVQ_FIFO_HF_LBN 1
3833 #define FSF_AZ_NET_IVEC_EVQ_FIFO_HF_WIDTH 1
3834 #define FSF_AZ_NET_IVEC_EVQ_FIFO_AF_LBN 0
3835 #define FSF_AZ_NET_IVEC_EVQ_FIFO_AF_WIDTH 1


3838 /* DRIVER_EV */
3839 /* Sub-fields of an RX flush completion event */
3840 #define FSF_AZ_DRIVER_EV_RX_FLUSH_FAIL_LBN 12
3841 #define FSF_AZ_DRIVER_EV_RX_FLUSH_FAIL_WIDTH 1
3842 #define FSF_AZ_DRIVER_EV_RX_DESCQ_ID_LBN 0
3843 #define FSF_AZ_DRIVER_EV_RX_DESCQ_ID_WIDTH 12



3847 /*************************************************************************
3848  *
3849  * Falcon non-volatile configuration
3850  *
3851  *************************************************************************
3852  */


3855 #define FR_AZ_TX_PACE_TBL_OFST FR_BZ_TX_PACE_TBL_OFST


3858 #ifdef  __cplusplus
3859 }
3860 #endif




3865 #endif /* _SYS_EFX_REGS_H */
3866 #endif /* ! codereview */
```

```
**********************************************************
  107677 Thu Aug 22 18:59:22 2013
new/usr/src/uts/common/io/sfxge/efx_regs_mcdi.h
Merged sfxge driver
**********************************************************
```

```
/*-
 * Copyright 2008-2013 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

/*! \cidoxg_firmware_mc_cmd */

#ifndef _SIENA_MC_DRIVER_PCOL_H
#define _SIENA_MC_DRIVER_PCOL_H


/* Values to be written into FMCR_CZ_RESET_STATE_REG to control boot. */
/* Power-on reset state */
#define MC_FW_STATE_POR (1)
/* If this is set in MC_RESET_STATE_REG then it should be
 * possible to jump into IMEM without loading code from flash. */
#define MC_FW_WARM_BOOT_OK (2)
/* The MC main image has started to boot. */
#define MC_FW_STATE_BOOTING (4)
/* The Scheduler has started. */
#define MC_FW_STATE_SCHED (8)

/* Siena MC shared memmory offsets */
/* The 'doorbell' addresses are hard-wired to alert the MC when written */
#define MC_SMEM_P0_DOORBELL_OFST        0x000
#define MC_SMEM_P1_DOORBELL_OFST        0x004
/* The rest of these are firmware-defined */
#define MC_SMEM_P0_PDU_OFST             0x008
#define MC_SMEM_P1_PDU_OFST             0x108
#define MC_SMEM_PDU_LEN                 0x100
#define MC_SMEM_P0_PTP_TIME_OFST        0x7f0
#define MC_SMEM_P0_STATUS_OFST          0x7f8
#define MC_SMEM_P1_STATUS_OFST          0x7fc

/* Values to be written to the per-port status dword in shared
 * memory on reboot and assert */
#define MC_STATUS_DWORD_REBOOT (0xb007b007)
#define MC_STATUS_DWORD_ASSERT (0xdeaddead)

/* The current version of the MCDI protocol.
 *
```



```
 * Note that the ROM burnt into the card only talks V0, so at the very
 * least every driver must support version 0 and MCDI_PCOL_VERSION
 */
#ifdef WITH_MCDI_V2
#define MCDI_PCOL_VERSION 2
#else
#define MCDI_PCOL_VERSION 1
#endif

/* Unused commands: 0x23, 0x27, 0x30, 0x31 */

/**
 * MCDI version 1
 *
 * Each MCDI request starts with an MCDI_HEADER, which is a 32byte
 * structure, filled in by the client.
 *
 *       0       7  8     16    20     22  23  24    31
 *      | CODE | R | LEN | SEQ | Rsvd | E | R | XFLAGS |
 *               |                      |   |
 *               |                      |   \--- Response
 *               |                      \------- Error
 *               \------------------------------ Resync (always set)
 *
 * The client writes it's request into MC shared memory, and rings the
 * doorbell. Each request is completed by either by the MC writting
 * back into shared memory, or by writting out an event.
 *
 * All MCDI commands support completion by shared memory response. Each
 * request may also contain additional data (accounted for by HEADER.LEN),
 * and some response's may also contain additional data (again, accounted
 * for by HEADER.LEN).
 *
 * Some MCDI commands support completion by event, in which any associated
 * response data is included in the event.
 *
 * The protocol requires one response to be delivered for every request, a
 * request should not be sent unless the response for the previous request
 * has been received (either by polling shared memory, or by receiving
 * an event).
 */

/** Request/Response structure */
#define MCDI_HEADER_OFST 0
#define MCDI_HEADER_CODE_LBN 0
#define MCDI_HEADER_CODE_WIDTH 7
#define MCDI_HEADER_RESYNC_LBN 7
#define MCDI_HEADER_RESYNC_WIDTH 1
#define MCDI_HEADER_DATALEN_LBN 8
#define MCDI_HEADER_DATALEN_WIDTH 8
#define MCDI_HEADER_SEQ_LBN 16
#define MCDI_HEADER_RSVD_LBN 20
#define MCDI_HEADER_RSVD_WIDTH 2
#define MCDI_HEADER_SEQ_WIDTH 4
#define MCDI_HEADER_ERROR_LBN 22
#define MCDI_HEADER_ERROR_WIDTH 1
#define MCDI_HEADER_RESPONSE_LBN 23
#define MCDI_HEADER_RESPONSE_WIDTH 1
#define MCDI_HEADER_XFLAGS_LBN 24
#define MCDI_HEADER_XFLAGS_WIDTH 8
/* Request response using event */
#define MCDI_HEADER_XFLAGS_EVREQ 0x01

/* Maximum number of payload bytes */
#ifdef WITH_MCDI_V2
#define MCDI_CTL_SDU_LEN_MAX 0x400
```

```
#else
#define MCDI_CTL_SDU_LEN_MAX 0xfc
#endif

/* The MC can generate events for two reasons:
 *   - To complete a shared memory request if XFLAGS_EVREQ was set
 *   - As a notification (link state, i2c event), controlled
 *     via MC_CMD_LOG_CTRL
 *
 * Both events share a common structure:
 *
 *  0      32     33      36    44     52     60
 * | Data | Cont | Level | Src | Code | Rsvd |
 *           |
 *           \ There is another event pending in this notification
 *
 * If Code==CMDDONE, then the fields are further interpreted as:
 *
 *   - LEVEL==INFO    Command succeeded
 *   - LEVEL==ERR     Command failed
 *
 *    0     8         16      24     32
 *   | Seq | Datalen | Errno | Rsvd |
 *
 *   These fields are taken directly out of the standard MCDI header, i.e.,
 *   LEVEL==ERR, Datalen == 0 => Reboot
 *
 * Events can be squirted out of the UART (using LOG_CTRL) without a
 * MCDI header.  An event can be distinguished from a MCDI response by
 * examining the first byte which is 0xc0.  This corresponds to the
 * non-existent MCDI command MC_CMD_DEBUG_LOG.
 *
 *      0         7        8
 *     | command | Resync |     = 0xc0
 *
 * Since the event is written in big-endian byte order, this works
 * providing bits 56-63 of the event are 0xc0.
 *
 *      56     60  63
 *     | Rsvd | Code |    = 0xc0
 *
 * Which means for convenience the event code is 0xc for all MC
 * generated events.
 */
#define FSE_AZ_EV_CODE_MCDI_EVRESPONSE 0xc


/* Non-existent command target */
#define MC_CMD_ERR_ENOENT 2
/* assert() has killed the MC */
#define MC_CMD_ERR_EINTR 4
/* Caller does not hold required locks */
#define MC_CMD_ERR_EACCES 13
/* Resource is currently unavailable (e.g. lock contention) */
#define MC_CMD_ERR_EBUSY 16
/* Invalid argument to target */
#define MC_CMD_ERR_EINVAL 22
/* Non-recursive resource is already acquired */
#define MC_CMD_ERR_EDEADLK 35
/* Operation not implemented */
#define MC_CMD_ERR_ENOSYS 38
/* Operation timed out */
#define MC_CMD_ERR_ETIME 62

#define MC_CMD_ERR_CODE_OFST 0
```



```
/* We define 8 "escape" commands to allow
   for command number space extension */

#define MC_CMD_CMD_SPACE_ESCAPE_0             0x78
#define MC_CMD_CMD_SPACE_ESCAPE_1             0x79
#define MC_CMD_CMD_SPACE_ESCAPE_2             0x7A
#define MC_CMD_CMD_SPACE_ESCAPE_3             0x7B
#define MC_CMD_CMD_SPACE_ESCAPE_4             0x7C
#define MC_CMD_CMD_SPACE_ESCAPE_5             0x7D
#define MC_CMD_CMD_SPACE_ESCAPE_6             0x7E
#define MC_CMD_CMD_SPACE_ESCAPE_7             0x7F

/* Vectors in the boot ROM */
/* Point to the copycode entry point. */
#define MC_BOOTROM_COPYCODE_VEC (0x7f4)
/* Points to the recovery mode entry point. */
#define MC_BOOTROM_NOFLASH_VEC (0x7f8)

/* The command set exported by the boot ROM (MCDI v0) */
#define MC_CMD_GET_VERSION_V0_SUPPORTED_FUNCS {         \
        (1 << MC_CMD_READ32)    |                       \
        (1 << MC_CMD_WRITE32)   |                       \
        (1 << MC_CMD_COPYCODE)  |                       \
        (1 << MC_CMD_GET_VERSION),                      \
        0, 0, 0 }

#define MC_CMD_SENSOR_INFO_OUT_OFFSET_OFST(_x) \
        (MC_CMD_SENSOR_ENTRY_OFST + (_x))

#define MC_CMD_DBI_WRITE_IN_ADDRESS_OFST(n) (  \
        (MC_CMD_DBI_WRITE_IN_DBIWROP_OFST+     \
         MC_CMD_DBIWROP_TYPEDEF_ADDRESS_OFST)+ \
         ((n)*MC_CMD_DBIWROP_TYPEDEF_LEN))

#define MC_CMD_DBI_WRITE_IN_BYTE_MASK_OFST(n) (  \
        (MC_CMD_DBI_WRITE_IN_DBIWROP_OFST+       \
         MC_CMD_DBIWROP_TYPEDEF_BYTE_MASK_OFST)+ \
         ((n)*MC_CMD_DBIWROP_TYPEDEF_LEN))

#define MC_CMD_DBI_WRITE_IN_VALUE_OFST(n) (  \
        (MC_CMD_DBI_WRITE_IN_DBIWROP_OFST+   \
         MC_CMD_DBIWROP_TYPEDEF_VALUE_OFST)+ \
         ((n)*MC_CMD_DBIWROP_TYPEDEF_LEN))


#ifdef WITH_MCDI_V2

/* Version 2 adds an optional argument to error returns: the errno value
 * may be followed by the (0-based) number of the first argument that
 * could not be processed.
 */
#define MC_CMD_ERR_ARG_OFST 4

/* Try again */
#define MC_CMD_ERR_EAGAIN 11
/* No space */
#define MC_CMD_ERR_ENOSPC 28

#endif

/* MCDI_EVENT structuredef */
#define MCDI_EVENT_LEN 8
#define MCDI_EVENT_CONT_LBN 32
#define MCDI_EVENT_CONT_WIDTH 1
#define MCDI_EVENT_LEVEL_LBN 33
#define MCDI_EVENT_LEVEL_WIDTH 3
```

```
#define MCDI_EVENT_LEVEL_INFO  0x0 /* enum */
#define MCDI_EVENT_LEVEL_WARN 0x1 /* enum */
#define MCDI_EVENT_LEVEL_ERR 0x2 /* enum */
#define MCDI_EVENT_LEVEL_FATAL 0x3 /* enum */
#define MCDI_EVENT_DATA_OFST 0
#define MCDI_EVENT_CMDDONE_SEQ_LBN 0
#define MCDI_EVENT_CMDDONE_SEQ_WIDTH 8
#define MCDI_EVENT_CMDDONE_DATALEN_LBN 8
#define MCDI_EVENT_CMDDONE_DATALEN_WIDTH 8
#define MCDI_EVENT_CMDDONE_ERRNO_LBN 16
#define MCDI_EVENT_CMDDONE_ERRNO_WIDTH 8
#define MCDI_EVENT_LINKCHANGE_LP_CAP_LBN 0
#define MCDI_EVENT_LINKCHANGE_LP_CAP_WIDTH 16
#define MCDI_EVENT_LINKCHANGE_SPEED_LBN 16
#define MCDI_EVENT_LINKCHANGE_SPEED_WIDTH 4
#define MCDI_EVENT_LINKCHANGE_SPEED_100M  0x1 /* enum */
#define MCDI_EVENT_LINKCHANGE_SPEED_1G  0x2 /* enum */
#define MCDI_EVENT_LINKCHANGE_SPEED_10G  0x3 /* enum */
#define MCDI_EVENT_LINKCHANGE_FCNTL_LBN 20
#define MCDI_EVENT_LINKCHANGE_FCNTL_WIDTH 4
#define MCDI_EVENT_LINKCHANGE_LINK_FLAGS_LBN 24
#define MCDI_EVENT_LINKCHANGE_LINK_FLAGS_WIDTH 8
#define MCDI_EVENT_SENSOREVT_MONITOR_LBN 0
#define MCDI_EVENT_SENSOREVT_MONITOR_WIDTH 8
#define MCDI_EVENT_SENSOREVT_STATE_LBN 8
#define MCDI_EVENT_SENSOREVT_STATE_WIDTH 8
#define MCDI_EVENT_SENSOREVT_VALUE_LBN 16
#define MCDI_EVENT_SENSOREVT_VALUE_WIDTH 16
#define MCDI_EVENT_FWALERT_DATA_LBN 8
#define MCDI_EVENT_FWALERT_DATA_WIDTH 24
#define MCDI_EVENT_FWALERT_REASON_LBN 0
#define MCDI_EVENT_FWALERT_REASON_WIDTH 8
#define MCDI_EVENT_FWALERT_REASON_SRAM_ACCESS 0x1 /* enum */
#define MCDI_EVENT_FLR_VF_LBN 0
#define MCDI_EVENT_FLR_VF_WIDTH 8
#define MCDI_EVENT_TX_ERR_TXQ_LBN 0
#define MCDI_EVENT_TX_ERR_TXQ_WIDTH 12
#define MCDI_EVENT_TX_ERR_TYPE_LBN 12
#define MCDI_EVENT_TX_ERR_TYPE_WIDTH 4
#define MCDI_EVENT_TX_ERR_DL_FAIL 0x1 /* enum */
#define MCDI_EVENT_TX_ERR_NO_EOP 0x2 /* enum */
#define MCDI_EVENT_TX_ERR_2BIG 0x3 /* enum */
#define MCDI_EVENT_TX_ERR_INFO_LBN 16
#define MCDI_EVENT_TX_ERR_INFO_WIDTH 16
#define MCDI_EVENT_TX_FLUSH_TXQ_LBN 0
#define MCDI_EVENT_TX_FLUSH_TXQ_WIDTH 12
#define MCDI_EVENT_PTP_ERR_TYPE_LBN 0
#define MCDI_EVENT_PTP_ERR_TYPE_WIDTH 8
#define MCDI_EVENT_PTP_ERR_PLL_LOST 0x1 /* enum */
#define MCDI_EVENT_PTP_ERR_FILTER 0x2 /* enum */
#define MCDI_EVENT_PTP_ERR_FIFO 0x3 /* enum */
#define MCDI_EVENT_PTP_ERR_QUEUE 0x4 /* enum */
#define MCDI_EVENT_DATA_LBN 0
#define MCDI_EVENT_DATA_WIDTH 32
#define MCDI_EVENT_SRC_LBN 36
#define MCDI_EVENT_SRC_WIDTH 8
#define MCDI_EVENT_EV_CODE_LBN 60
#define MCDI_EVENT_EV_CODE_WIDTH 4
#define MCDI_EVENT_CODE_LBN 44
#define MCDI_EVENT_CODE_WIDTH 8
#define MCDI_EVENT_CODE_BADSSERT 0x1 /* enum */
#define MCDI_EVENT_CODE_PMNOTICE 0x2 /* enum */
#define MCDI_EVENT_CODE_CMDDONE 0x3 /* enum */
#define MCDI_EVENT_CODE_LINKCHANGE 0x4 /* enum */
#define MCDI_EVENT_CODE_SENSOREVT 0x5 /* enum */
#define MCDI_EVENT_CODE_SCHEDERR 0x6 /* enum */
```



```
#define MCDI_EVENT_CODE_REBOOT 0x7 /* enum */
#define MCDI_EVENT_CODE_MAC_STATS_DMA 0x8 /* enum */
#define MCDI_EVENT_CODE_FWALERT 0x9 /* enum */
#define MCDI_EVENT_CODE_FLR 0xa /* enum */
#define MCDI_EVENT_CODE_TX_ERR 0xb /* enum */
#define MCDI_EVENT_CODE_TX_FLUSH  0xc /* enum */
#define MCDI_EVENT_CODE_PTP_RX  0xd /* enum */
#define MCDI_EVENT_CODE_PTP_FAULT  0xe /* enum */
#define MCDI_EVENT_CODE_VCAL_FAIL  0x13 /* enum */
#define MCDI_EVENT_CMDDONE_DATA_OFST 0
#define MCDI_EVENT_CMDDONE_DATA_LBN 0
#define MCDI_EVENT_CMDDONE_DATA_WIDTH 32
#define MCDI_EVENT_LINKCHANGE_DATA_OFST 0
#define MCDI_EVENT_LINKCHANGE_DATA_LBN 0
#define MCDI_EVENT_LINKCHANGE_DATA_WIDTH 32
#define MCDI_EVENT_SENSOREVT_DATA_OFST 0
#define MCDI_EVENT_SENSOREVT_DATA_LBN 0
#define MCDI_EVENT_SENSOREVT_DATA_WIDTH 32
#define MCDI_EVENT_MAC_STATS_DMA_GENERATION_OFST 0
#define MCDI_EVENT_MAC_STATS_DMA_GENERATION_LBN 0
#define MCDI_EVENT_MAC_STATS_DMA_GENERATION_WIDTH 32
#define MCDI_EVENT_TX_ERR_DATA_OFST 0
#define MCDI_EVENT_TX_ERR_DATA_LBN 0
#define MCDI_EVENT_TX_ERR_DATA_WIDTH 32
#define MCDI_EVENT_PTP_SECONDS_OFST 0
#define MCDI_EVENT_PTP_SECONDS_LBN 0
#define MCDI_EVENT_PTP_SECONDS_WIDTH 32
#define MCDI_EVENT_PTP_NANOSECONDS_OFST 0
#define MCDI_EVENT_PTP_NANOSECONDS_LBN 0
#define MCDI_EVENT_PTP_NANOSECONDS_WIDTH 32
#define MCDI_EVENT_PTP_UUID_OFST 0
#define MCDI_EVENT_PTP_UUID_LBN 0
#define MCDI_EVENT_PTP_UUID_WIDTH 32


/***********************************/
/* MC_CMD_READ32
 * Read multiple 32byte words from MC memory.
 */
#define MC_CMD_READ32 0x1

/* MC_CMD_READ32_IN msgrequest */
#define MC_CMD_READ32_IN_LEN 8
#define MC_CMD_READ32_IN_ADDR_OFST 0
#define MC_CMD_READ32_IN_NUMWORDS_OFST 4

/* MC_CMD_READ32_OUT msgresponse */
#define MC_CMD_READ32_OUT_LENMIN 4
#define MC_CMD_READ32_OUT_LENMAX 252
#define MC_CMD_READ32_OUT_LEN(num) (0+4*(num))
#define MC_CMD_READ32_OUT_BUFFER_OFST 0
#define MC_CMD_READ32_OUT_BUFFER_LEN 4
#define MC_CMD_READ32_OUT_BUFFER_MINNUM 1
#define MC_CMD_READ32_OUT_BUFFER_MAXNUM 63


/***********************************/
/* MC_CMD_WRITE32
 * Write multiple 32byte words to MC memory.
 */
#define MC_CMD_WRITE32 0x2

/* MC_CMD_WRITE32_IN msgrequest */
#define MC_CMD_WRITE32_IN_LENMIN 8
#define MC_CMD_WRITE32_IN_LENMAX 252
#define MC_CMD_WRITE32_IN_LEN(num) (4+4*(num))
```

```
#define MC_CMD_WRITE32_IN_ADDR_OFST 0
#define MC_CMD_WRITE32_IN_BUFFER_OFST 4
#define MC_CMD_WRITE32_IN_BUFFER_LEN 4
#define MC_CMD_WRITE32_IN_BUFFER_MINNUM 1
#define MC_CMD_WRITE32_IN_BUFFER_MAXNUM 62

/* MC_CMD_WRITE32_OUT msgresponse */
#define MC_CMD_WRITE32_OUT_LEN 0


/***********************************/
/* MC_CMD_COPYCODE
 * Copy MC code between two locations and jump.
 */
#define MC_CMD_COPYCODE 0x3

/* MC_CMD_COPYCODE_IN msgrequest */
#define MC_CMD_COPYCODE_IN_LEN 16
#define MC_CMD_COPYCODE_IN_SRC_ADDR_OFST 0
#define MC_CMD_COPYCODE_IN_DEST_ADDR_OFST 4
#define MC_CMD_COPYCODE_IN_NUMWORDS_OFST 8
#define MC_CMD_COPYCODE_IN_JUMP_OFST 12
#define MC_CMD_COPYCODE_JUMP_NONE 0x1 /* enum */

/* MC_CMD_COPYCODE_OUT msgresponse */
#define MC_CMD_COPYCODE_OUT_LEN 0


/***********************************/
/* MC_CMD_SET_FUNC
 */
#define MC_CMD_SET_FUNC  0x4

/* MC_CMD_SET_FUNC_IN msgrequest */
#define MC_CMD_SET_FUNC_IN_LEN 4
#define MC_CMD_SET_FUNC_IN_FUNC_OFST 0

/* MC_CMD_SET_FUNC_OUT msgresponse */
#define MC_CMD_SET_FUNC_OUT_LEN 0


/***********************************/
/* MC_CMD_GET_BOOT_STATUS
 */
#define MC_CMD_GET_BOOT_STATUS 0x5

/* MC_CMD_GET_BOOT_STATUS_IN msgrequest */
#define MC_CMD_GET_BOOT_STATUS_IN_LEN 0

/* MC_CMD_GET_BOOT_STATUS_OUT msgresponse */
#define MC_CMD_GET_BOOT_STATUS_OUT_LEN 8
#define MC_CMD_GET_BOOT_STATUS_OUT_BOOT_OFFSET_OFST 0
#define MC_CMD_GET_BOOT_STATUS_OUT_FLAGS_OFST 4
#define MC_CMD_GET_BOOT_STATUS_OUT_FLAGS_WATCHDOG_LBN 0
#define MC_CMD_GET_BOOT_STATUS_OUT_FLAGS_WATCHDOG_WIDTH 1
#define MC_CMD_GET_BOOT_STATUS_OUT_FLAGS_PRIMARY_LBN 1
#define MC_CMD_GET_BOOT_STATUS_OUT_FLAGS_PRIMARY_WIDTH 1
#define MC_CMD_GET_BOOT_STATUS_OUT_FLAGS_BACKUP_LBN 2
#define MC_CMD_GET_BOOT_STATUS_OUT_FLAGS_BACKUP_WIDTH 1


/***********************************/
/* MC_CMD_GET_ASSERTS
 * Get and clear any assertion status.
 */
#define MC_CMD_GET_ASSERTS  0x6
```



```
/* MC_CMD_GET_ASSERTS_IN msgrequest */
#define MC_CMD_GET_ASSERTS_IN_LEN 4
#define MC_CMD_GET_ASSERTS_IN_CLEAR_OFST 0

/* MC_CMD_GET_ASSERTS_OUT msgresponse */
#define MC_CMD_GET_ASSERTS_OUT_LEN 140
#define MC_CMD_GET_ASSERTS_OUT_GLOBAL_FLAGS_OFST 0
#define MC_CMD_GET_ASSERTS_FLAGS_NO_FAILS 0x1 /* enum */
#define MC_CMD_GET_ASSERTS_FLAGS_SYS_FAIL 0x2 /* enum */
#define MC_CMD_GET_ASSERTS_FLAGS_THR_FAIL 0x3 /* enum */
#define MC_CMD_GET_ASSERTS_FLAGS_WDOG_FIRED 0x4 /* enum */
#define MC_CMD_GET_ASSERTS_OUT_SAVED_PC_OFFS_OFST 4
#define MC_CMD_GET_ASSERTS_OUT_GP_REGS_OFFS_OFST 8
#define MC_CMD_GET_ASSERTS_OUT_GP_REGS_OFFS_LEN 4
#define MC_CMD_GET_ASSERTS_OUT_GP_REGS_OFFS_NUM 31
#define MC_CMD_GET_ASSERTS_OUT_THREAD_OFFS_OFST 132
#define MC_CMD_GET_ASSERTS_OUT_RESERVED_OFST 136


/***********************************/
/* MC_CMD_LOG_CTRL
 * Configure the output stream for various events and messages.
 */
#define MC_CMD_LOG_CTRL  0x7

/* MC_CMD_LOG_CTRL_IN msgrequest */
#define MC_CMD_LOG_CTRL_IN_LEN 8
#define MC_CMD_LOG_CTRL_IN_LOG_DEST_OFST 0
#define MC_CMD_LOG_CTRL_IN_LOG_DEST_UART 0x1 /* enum */
#define MC_CMD_LOG_CTRL_IN_LOG_DEST_EVQ 0x2 /* enum */
#define MC_CMD_LOG_CTRL_IN_LOG_DEST_EVQ_OFST 4

/* MC_CMD_LOG_CTRL_OUT msgresponse */
#define MC_CMD_LOG_CTRL_OUT_LEN 0


/***********************************/
/* MC_CMD_GET_VERSION
 * Get version information about the MC firmware.
 */
#define MC_CMD_GET_VERSION  0x8

/* MC_CMD_GET_VERSION_IN msgrequest */
#define MC_CMD_GET_VERSION_IN_LEN 0

/* MC_CMD_GET_VERSION_V0_OUT msgresponse */
#define MC_CMD_GET_VERSION_V0_OUT_LEN 4
#define MC_CMD_GET_VERSION_OUT_FIRMWARE_OFST 0
#define MC_CMD_GET_VERSION_OUT_FIRMWARE_ANY 0xffffffff /* enum */
#define MC_CMD_GET_VERSION_OUT_FIRMWARE_BOOTROM 0xb0070000 /* enum */

/* MC_CMD_GET_VERSION_OUT msgresponse */
#define MC_CMD_GET_VERSION_OUT_LEN 32
/*            MC_CMD_GET_VERSION_OUT_FIRMWARE_OFST 0 */
/*            Enum values, see field(s): */
/*               MC_CMD_GET_VERSION_V0_OUT/MC_CMD_GET_VERSION_OUT_FIRMWARE */
#define MC_CMD_GET_VERSION_OUT_PCOL_OFST 4
#define MC_CMD_GET_VERSION_OUT_SUPPORTED_FUNCS_OFST 8
#define MC_CMD_GET_VERSION_OUT_SUPPORTED_FUNCS_LEN 16
#define MC_CMD_GET_VERSION_OUT_VERSION_OFST 24
#define MC_CMD_GET_VERSION_OUT_VERSION_LEN 8
#define MC_CMD_GET_VERSION_OUT_VERSION_LO_OFST 24
#define MC_CMD_GET_VERSION_OUT_VERSION_HI_OFST 28
```

```
524 /***********************************/
525 /* MC_CMD_GET_FPGAREG
526  * Read multiple bytes from PTP FPGA.
527  */
528 #define MC_CMD_GET_FPGAREG  0x9

530 /* MC_CMD_GET_FPGAREG_IN msgrequest */
531 #define MC_CMD_GET_FPGAREG_IN_LEN 8
532 #define MC_CMD_GET_FPGAREG_IN_ADDR_OFST 0
533 #define MC_CMD_GET_FPGAREG_IN_NUMBYTES_OFST 4

535 /* MC_CMD_GET_FPGAREG_OUT msgresponse */
536 #define MC_CMD_GET_FPGAREG_OUT_LENMIN 1
537 #define MC_CMD_GET_FPGAREG_OUT_LENMAX 252
538 #define MC_CMD_GET_FPGAREG_OUT_LEN(num) (0+1*(num))
539 #define MC_CMD_GET_FPGAREG_OUT_BUFFER_OFST 0
540 #define MC_CMD_GET_FPGAREG_OUT_BUFFER_LEN 1
541 #define MC_CMD_GET_FPGAREG_OUT_BUFFER_MINNUM 1
542 #define MC_CMD_GET_FPGAREG_OUT_BUFFER_MAXNUM 252


545 /***********************************/
546 /* MC_CMD_PUT_FPGAREG
547  * Write multiple bytes to PTP FPGA.
548  */
549 #define MC_CMD_PUT_FPGAREG  0xa

551 /* MC_CMD_PUT_FPGAREG_IN msgrequest */
552 #define MC_CMD_PUT_FPGAREG_IN_LENMIN 5
553 #define MC_CMD_PUT_FPGAREG_IN_LENMAX 252
554 #define MC_CMD_PUT_FPGAREG_IN_LEN(num) (4+1*(num))
555 #define MC_CMD_PUT_FPGAREG_IN_ADDR_OFST 0
556 #define MC_CMD_PUT_FPGAREG_IN_BUFFER_OFST 4
557 #define MC_CMD_PUT_FPGAREG_IN_BUFFER_LEN 1
558 #define MC_CMD_PUT_FPGAREG_IN_BUFFER_MINNUM 1
559 #define MC_CMD_PUT_FPGAREG_IN_BUFFER_MAXNUM 248

561 /* MC_CMD_PUT_FPGAREG_OUT msgresponse */
562 #define MC_CMD_PUT_FPGAREG_OUT_LEN 0


565 /***********************************/
566 /* MC_CMD_PTP
567  * Perform PTP operation
568  */
569 #define MC_CMD_PTP  0xb

571 /* MC_CMD_PTP_IN msgrequest */
572 #define MC_CMD_PTP_IN_LEN 1
573 #define MC_CMD_PTP_IN_OP_OFST 0
574 #define MC_CMD_PTP_IN_OP_LEN 1
575 #define MC_CMD_PTP_OP_ENABLE 0x1 /* enum */
576 #define MC_CMD_PTP_OP_DISABLE 0x2 /* enum */
577 #define MC_CMD_PTP_OP_TRANSMIT 0x3 /* enum */
578 #define MC_CMD_PTP_OP_READ_NIC_TIME 0x4 /* enum */
579 #define MC_CMD_PTP_OP_STATUS 0x5 /* enum */
580 #define MC_CMD_PTP_OP_ADJUST 0x6 /* enum */
581 #define MC_CMD_PTP_OP_SYNCHRONIZE 0x7 /* enum */
582 #define MC_CMD_PTP_OP_MANFTEST_BASIC 0x8 /* enum */
583 #define MC_CMD_PTP_OP_MANFTEST_PACKET 0x9 /* enum */
584 #define MC_CMD_PTP_OP_RESET_STATS 0xa /* enum */
585 #define MC_CMD_PTP_OP_DEBUG 0xb /* enum */
586 #define MC_CMD_PTP_OP_MAX 0xc /* enum */

588 /* MC_CMD_PTP_IN_ENABLE msgrequest */
589 #define MC_CMD_PTP_IN_ENABLE_LEN 16
```



```
590 #define MC_CMD_PTP_IN_CMD_OFST 0
591 #define MC_CMD_PTP_IN_PERIPH_ID_OFST 4
592 #define MC_CMD_PTP_IN_ENABLE_QUEUE_OFST 8
593 #define MC_CMD_PTP_IN_ENABLE_MODE_OFST 12
594 #define MC_CMD_PTP_MODE_V1 0x0 /* enum */
595 #define MC_CMD_PTP_MODE_V1_VLAN 0x1 /* enum */
596 #define MC_CMD_PTP_MODE_V2 0x2 /* enum */
597 #define MC_CMD_PTP_MODE_V2_VLAN 0x3 /* enum */

599 /* MC_CMD_PTP_IN_DISABLE msgrequest */
600 #define MC_CMD_PTP_IN_DISABLE_LEN 8
601 /*            MC_CMD_PTP_IN_CMD_OFST 0 */
602 /*            MC_CMD_PTP_IN_PERIPH_ID_OFST 4 */

604 /* MC_CMD_PTP_IN_TRANSMIT msgrequest */
605 #define MC_CMD_PTP_IN_TRANSMIT_LENMIN 13
606 #define MC_CMD_PTP_IN_TRANSMIT_LENMAX 252
607 #define MC_CMD_PTP_IN_TRANSMIT_LEN(num) (12+1*(num))
608 /*            MC_CMD_PTP_IN_CMD_OFST 0 */
609 /*            MC_CMD_PTP_IN_PERIPH_ID_OFST 4 */
610 #define MC_CMD_PTP_IN_TRANSMIT_LENGTH_OFST 8
611 #define MC_CMD_PTP_IN_TRANSMIT_PACKET_OFST 12
612 #define MC_CMD_PTP_IN_TRANSMIT_PACKET_LEN 1
613 #define MC_CMD_PTP_IN_TRANSMIT_PACKET_MINNUM 1
614 #define MC_CMD_PTP_IN_TRANSMIT_PACKET_MAXNUM 240

616 /* MC_CMD_PTP_IN_READ_NIC_TIME msgrequest */
617 #define MC_CMD_PTP_IN_READ_NIC_TIME_LEN 8
618 /*            MC_CMD_PTP_IN_CMD_OFST 0 */
619 /*            MC_CMD_PTP_IN_PERIPH_ID_OFST 4 */

621 /* MC_CMD_PTP_IN_STATUS msgrequest */
622 #define MC_CMD_PTP_IN_STATUS_LEN 8
623 /*            MC_CMD_PTP_IN_CMD_OFST 0 */
624 /*            MC_CMD_PTP_IN_PERIPH_ID_OFST 4 */

626 /* MC_CMD_PTP_IN_ADJUST msgrequest */
627 #define MC_CMD_PTP_IN_ADJUST_LEN 24
628 /*            MC_CMD_PTP_IN_CMD_OFST 0 */
629 /*            MC_CMD_PTP_IN_PERIPH_ID_OFST 4 */
630 #define MC_CMD_PTP_IN_ADJUST_FREQ_OFST 8
631 #define MC_CMD_PTP_IN_ADJUST_FREQ_LEN 8
632 #define MC_CMD_PTP_IN_ADJUST_FREQ_LO_OFST 8
633 #define MC_CMD_PTP_IN_ADJUST_FREQ_HI_OFST 12
634 #define MC_CMD_PTP_IN_ADJUST_BITS 0x28 /* enum */
635 #define MC_CMD_PTP_IN_ADJUST_SECONDS_OFST 16
636 #define MC_CMD_PTP_IN_ADJUST_NANOSECONDS_OFST 20

638 /* MC_CMD_PTP_IN_SYNCHRONIZE msgrequest */
639 #define MC_CMD_PTP_IN_SYNCHRONIZE_LEN 20
640 /*            MC_CMD_PTP_IN_CMD_OFST 0 */
641 /*            MC_CMD_PTP_IN_PERIPH_ID_OFST 4 */
642 #define MC_CMD_PTP_IN_SYNCHRONIZE_NUMTIMESETS_OFST 8
643 #define MC_CMD_PTP_IN_SYNCHRONIZE_START_ADDR_OFST 12
644 #define MC_CMD_PTP_IN_SYNCHRONIZE_START_ADDR_LEN 8
645 #define MC_CMD_PTP_IN_SYNCHRONIZE_START_ADDR_LO_OFST 12
646 #define MC_CMD_PTP_IN_SYNCHRONIZE_START_ADDR_HI_OFST 16

648 /* MC_CMD_PTP_IN_MANFTEST_BASIC msgrequest */
649 #define MC_CMD_PTP_IN_MANFTEST_BASIC_LEN 8
650 /*            MC_CMD_PTP_IN_CMD_OFST 0 */
651 /*            MC_CMD_PTP_IN_PERIPH_ID_OFST 4 */

653 /* MC_CMD_PTP_IN_MANFTEST_PACKET msgrequest */
654 #define MC_CMD_PTP_IN_MANFTEST_PACKET_LEN 12
655 /*            MC_CMD_PTP_IN_CMD_OFST 0 */
```

```
656 /*            MC_CMD_PTP_IN_PERIPH_ID_OFST 4 */
657 #define MC_CMD_PTP_IN_MANFTEST_PACKET_TEST_ENABLE_OFST 8

659 /* MC_CMD_PTP_IN_RESET_STATS msgrequest */
660 #define MC_CMD_PTP_IN_RESET_STATS_LEN 8
661 /*            MC_CMD_PTP_IN_CMD_OFST 0 */
662 /*            MC_CMD_PTP_IN_PERIPH_ID_OFST 4 */

664 /* MC_CMD_PTP_IN_DEBUG msgrequest */
665 #define MC_CMD_PTP_IN_DEBUG_LEN 12
666 /*            MC_CMD_PTP_IN_CMD_OFST 0 */
667 /*            MC_CMD_PTP_IN_PERIPH_ID_OFST 4 */
668 #define MC_CMD_PTP_IN_DEBUG_DEBUG_PARAM_OFST 8

670 /* MC_CMD_PTP_OUT msgresponse */
671 #define MC_CMD_PTP_OUT_LEN 0

673 /* MC_CMD_PTP_OUT_TRANSMIT msgresponse */
674 #define MC_CMD_PTP_OUT_TRANSMIT_LEN 8
675 #define MC_CMD_PTP_OUT_TRANSMIT_SECONDS_OFST 0
676 #define MC_CMD_PTP_OUT_TRANSMIT_NANOSECONDS_OFST 4

678 /* MC_CMD_PTP_OUT_READ_NIC_TIME msgresponse */
679 #define MC_CMD_PTP_OUT_READ_NIC_TIME_LEN 8
680 #define MC_CMD_PTP_OUT_READ_NIC_TIME_SECONDS_OFST 0
681 #define MC_CMD_PTP_OUT_READ_NIC_TIME_NANOSECONDS_OFST 4

683 /* MC_CMD_PTP_OUT_STATUS msgresponse */
684 #define MC_CMD_PTP_OUT_STATUS_LEN 64
685 #define MC_CMD_PTP_OUT_STATUS_CLOCK_FREQ_OFST 0
686 #define MC_CMD_PTP_OUT_STATUS_STATS_TX_OFST 4
687 #define MC_CMD_PTP_OUT_STATUS_STATS_RX_OFST 8
688 #define MC_CMD_PTP_OUT_STATUS_STATS_TS_OFST 12
689 #define MC_CMD_PTP_OUT_STATUS_STATS_FM_OFST 16
690 #define MC_CMD_PTP_OUT_STATUS_STATS_NFM_OFST 20
691 #define MC_CMD_PTP_OUT_STATUS_STATS_PPS_OFLOW_OFST 24
692 #define MC_CMD_PTP_OUT_STATUS_STATS_PPS_BAD_OFST 28
693 #define MC_CMD_PTP_OUT_STATUS_STATS_PPS_PER_MIN_OFST 32
694 #define MC_CMD_PTP_OUT_STATUS_STATS_PPS_PER_MAX_OFST 36
695 #define MC_CMD_PTP_OUT_STATUS_STATS_PPS_PER_LAST_OFST 40
696 #define MC_CMD_PTP_OUT_STATUS_STATS_PPS_PER_MEAN_OFST 44
697 #define MC_CMD_PTP_OUT_STATUS_STATS_PPS_OFF_MIN_OFST 48
698 #define MC_CMD_PTP_OUT_STATUS_STATS_PPS_OFF_MAX_OFST 52
699 #define MC_CMD_PTP_OUT_STATUS_STATS_PPS_OFF_LAST_OFST 56
700 #define MC_CMD_PTP_OUT_STATUS_STATS_PPS_OFF_MEAN_OFST 60

702 /* MC_CMD_PTP_OUT_SYNCHRONIZE msgresponse */
703 #define MC_CMD_PTP_OUT_SYNCHRONIZE_LENMIN 20
704 #define MC_CMD_PTP_OUT_SYNCHRONIZE_LENMAX 240
705 #define MC_CMD_PTP_OUT_SYNCHRONIZE_LEN(num) (0+20*(num))
706 #define MC_CMD_PTP_OUT_SYNCHRONIZE_TIMESET_OFST 0
707 #define MC_CMD_PTP_OUT_SYNCHRONIZE_TIMESET_LEN 20
708 #define MC_CMD_PTP_OUT_SYNCHRONIZE_TIMESET_MINNUM 1
709 #define MC_CMD_PTP_OUT_SYNCHRONIZE_TIMESET_MAXNUM 12
710 #define MC_CMD_PTP_OUT_SYNCHRONIZE_HOSTSTART_OFST 0
711 #define MC_CMD_PTP_OUT_SYNCHRONIZE_SECONDS_OFST 4
712 #define MC_CMD_PTP_OUT_SYNCHRONIZE_NANOSECONDS_OFST 8
713 #define MC_CMD_PTP_OUT_SYNCHRONIZE_HOSTEND_OFST 12
714 #define MC_CMD_PTP_OUT_SYNCHRONIZE_WAITNS_OFST 16

716 /* MC_CMD_PTP_OUT_MANFTEST_BASIC msgresponse */
717 #define MC_CMD_PTP_OUT_MANFTEST_BASIC_LEN 8
718 #define MC_CMD_PTP_OUT_MANFTEST_BASIC_TEST_RESULT_OFST 0
719 #define MC_CMD_PTP_MANF_SUCCESS 0x0 /* enum */
720 #define MC_CMD_PTP_MANF_FPGA_LOAD 0x1 /* enum */
721 #define MC_CMD_PTP_MANF_FPGA_VERSION 0x2 /* enum */
```



```
722 #define MC_CMD_PTP_MANF_FPGA_REGISTERS 0x3 /* enum */
723 #define MC_CMD_PTP_MANF_OSCILLATOR 0x4 /* enum */
724 #define MC_CMD_PTP_MANF_TIMESTAMPS 0x5 /* enum */
725 #define MC_CMD_PTP_MANF_PACKET_COUNT 0x6 /* enum */
726 #define MC_CMD_PTP_MANF_FILTER_COUNT 0x7 /* enum */
727 #define MC_CMD_PTP_MANF_PACKET_ENOUGH 0x8 /* enum */
728 #define MC_CMD_PTP_MANF_GPIO_TRIGGER 0x9 /* enum */
729 #define MC_CMD_PTP_OUT_MANFTEST_BASIC_TEST_EXTOSC_OFST 4

731 /* MC_CMD_PTP_OUT_MANFTEST_PACKET msgresponse */
732 #define MC_CMD_PTP_OUT_MANFTEST_PACKET_LEN 12
733 #define MC_CMD_PTP_OUT_MANFTEST_PACKET_TEST_RESULT_OFST 0
734 #define MC_CMD_PTP_OUT_MANFTEST_PACKET_TEST_FPGACOUNT_OFST 4
735 #define MC_CMD_PTP_OUT_MANFTEST_PACKET_TEST_FILTERCOUNT_OFST 8


738 /***********************************/
739 /* MC_CMD_CSR_READ32
740  * Read 32bit words from the indirect memory map.
741  */
742 #define MC_CMD_CSR_READ32  0xc

744 /* MC_CMD_CSR_READ32_IN msgrequest */
745 #define MC_CMD_CSR_READ32_IN_LEN 12
746 #define MC_CMD_CSR_READ32_IN_ADDR_OFST 0
747 #define MC_CMD_CSR_READ32_IN_STEP_OFST 4
748 #define MC_CMD_CSR_READ32_IN_NUMWORDS_OFST 8

750 /* MC_CMD_CSR_READ32_OUT msgresponse */
751 #define MC_CMD_CSR_READ32_OUT_LENMIN 4
752 #define MC_CMD_CSR_READ32_OUT_LENMAX 252
753 #define MC_CMD_CSR_READ32_OUT_LEN(num) (0+4*(num))
754 #define MC_CMD_CSR_READ32_OUT_BUFFER_OFST 0
755 #define MC_CMD_CSR_READ32_OUT_BUFFER_LEN 4
756 #define MC_CMD_CSR_READ32_OUT_BUFFER_MINNUM 1
757 #define MC_CMD_CSR_READ32_OUT_BUFFER_MAXNUM 63


760 /***********************************/
761 /* MC_CMD_CSR_WRITE32
762  * Write 32bit dwords to the indirect memory map.
763  */
764 #define MC_CMD_CSR_WRITE32  0xd

766 /* MC_CMD_CSR_WRITE32_IN msgrequest */
767 #define MC_CMD_CSR_WRITE32_IN_LENMIN 12
768 #define MC_CMD_CSR_WRITE32_IN_LENMAX 252
769 #define MC_CMD_CSR_WRITE32_IN_LEN(num) (8+4*(num))
770 #define MC_CMD_CSR_WRITE32_IN_ADDR_OFST 0
771 #define MC_CMD_CSR_WRITE32_IN_STEP_OFST 4
772 #define MC_CMD_CSR_WRITE32_IN_BUFFER_OFST 8
773 #define MC_CMD_CSR_WRITE32_IN_BUFFER_LEN 4
774 #define MC_CMD_CSR_WRITE32_IN_BUFFER_MINNUM 1
775 #define MC_CMD_CSR_WRITE32_IN_BUFFER_MAXNUM 61

777 /* MC_CMD_CSR_WRITE32_OUT msgresponse */
778 #define MC_CMD_CSR_WRITE32_OUT_LEN 4
779 #define MC_CMD_CSR_WRITE32_OUT_STATUS_OFST 0


782 /***********************************/
783 /* MC_CMD_STACKINFO
784  * Get stack information.
785  */
786 #define MC_CMD_STACKINFO  0xf
```

```
788 /* MC_CMD_STACKINFO_IN msgrequest */
789 #define MC_CMD_STACKINFO_IN_LEN 0

791 /* MC_CMD_STACKINFO_OUT msgresponse */
792 #define MC_CMD_STACKINFO_OUT_LENMIN 12
793 #define MC_CMD_STACKINFO_OUT_LENMAX 252
794 #define MC_CMD_STACKINFO_OUT_LEN(num) (0+12*(num))
795 #define MC_CMD_STACKINFO_OUT_THREAD_INFO_OFST 0
796 #define MC_CMD_STACKINFO_OUT_THREAD_INFO_LEN 12
797 #define MC_CMD_STACKINFO_OUT_THREAD_INFO_MINNUM 1
798 #define MC_CMD_STACKINFO_OUT_THREAD_INFO_MAXNUM 21


801 /***********************************/
802 /* MC_CMD_MDIO_READ
803  * MDIO register read.
804  */
805 #define MC_CMD_MDIO_READ  0x10

807 /* MC_CMD_MDIO_READ_IN msgrequest */
808 #define MC_CMD_MDIO_READ_IN_LEN 16
809 #define MC_CMD_MDIO_READ_IN_BUS_OFST 0
810 #define MC_CMD_MDIO_BUS_INTERNAL 0x0 /* enum */
811 #define MC_CMD_MDIO_BUS_EXTERNAL 0x1 /* enum */
812 #define MC_CMD_MDIO_READ_IN_PRTAD_OFST 4
813 #define MC_CMD_MDIO_READ_IN_DEVAD_OFST 8
814 #define MC_CMD_MDIO_CLAUSE22 0x20 /* enum */
815 #define MC_CMD_MDIO_READ_IN_ADDR_OFST 12

817 /* MC_CMD_MDIO_READ_OUT msgresponse */
818 #define MC_CMD_MDIO_READ_OUT_LEN 8
819 #define MC_CMD_MDIO_READ_OUT_VALUE_OFST 0
820 #define MC_CMD_MDIO_READ_OUT_STATUS_OFST 4
821 #define MC_CMD_MDIO_STATUS_GOOD 0x8 /* enum */


824 /***********************************/
825 /* MC_CMD_MDIO_WRITE
826  * MDIO register write.
827  */
828 #define MC_CMD_MDIO_WRITE  0x11

830 /* MC_CMD_MDIO_WRITE_IN msgrequest */
831 #define MC_CMD_MDIO_WRITE_IN_LEN 20
832 #define MC_CMD_MDIO_WRITE_IN_BUS_OFST 0
833 /*               MC_CMD_MDIO_BUS_INTERNAL 0x0 */
834 /*               MC_CMD_MDIO_BUS_EXTERNAL 0x1 */
835 #define MC_CMD_MDIO_WRITE_IN_PRTAD_OFST 4
836 #define MC_CMD_MDIO_WRITE_IN_DEVAD_OFST 8
837 /*               MC_CMD_MDIO_CLAUSE22 0x20 */
838 #define MC_CMD_MDIO_WRITE_IN_ADDR_OFST 12
839 #define MC_CMD_MDIO_WRITE_IN_VALUE_OFST 16

841 /* MC_CMD_MDIO_WRITE_OUT msgresponse */
842 #define MC_CMD_MDIO_WRITE_OUT_LEN 4
843 #define MC_CMD_MDIO_WRITE_OUT_STATUS_OFST 0
844 /*               MC_CMD_MDIO_STATUS_GOOD 0x8 */


847 /***********************************/
848 /* MC_CMD_DBI_WRITE
849  * Write DBI register(s).
850  */
851 #define MC_CMD_DBI_WRITE  0x12

853 /* MC_CMD_DBI_WRITE_IN msgrequest */
```



```
854 #define MC_CMD_DBI_WRITE_IN_LENMIN 12
855 #define MC_CMD_DBI_WRITE_IN_LENMAX 252
856 #define MC_CMD_DBI_WRITE_IN_LEN(num) (0+12*(num))
857 #define MC_CMD_DBI_WRITE_IN_DBIWROP_OFST 0
858 #define MC_CMD_DBI_WRITE_IN_DBIWROP_LEN 12
859 #define MC_CMD_DBI_WRITE_IN_DBIWROP_MINNUM 1
860 #define MC_CMD_DBI_WRITE_IN_DBIWROP_MAXNUM 21

862 /* MC_CMD_DBI_WRITE_OUT msgresponse */
863 #define MC_CMD_DBI_WRITE_OUT_LEN 0

865 /* MC_CMD_DBIWROP_TYPEDEF structuredef */
866 #define MC_CMD_DBIWROP_TYPEDEF_LEN 12
867 #define MC_CMD_DBIWROP_TYPEDEF_ADDRESS_OFST 0
868 #define MC_CMD_DBIWROP_TYPEDEF_ADDRESS_LBN 0
869 #define MC_CMD_DBIWROP_TYPEDEF_ADDRESS_WIDTH 32
870 #define MC_CMD_DBIWROP_TYPEDEF_BYTE_MASK_OFST 4
871 #define MC_CMD_DBIWROP_TYPEDEF_BYTE_MASK_LBN 32
872 #define MC_CMD_DBIWROP_TYPEDEF_BYTE_MASK_WIDTH 32
873 #define MC_CMD_DBIWROP_TYPEDEF_VALUE_OFST 8
874 #define MC_CMD_DBIWROP_TYPEDEF_VALUE_LBN 64
875 #define MC_CMD_DBIWROP_TYPEDEF_VALUE_WIDTH 32


878 /***********************************/
879 /* MC_CMD_PORT_READ32
880  * Read a 32-bit register from the indirect port register map.
881  */
882 #define MC_CMD_PORT_READ32  0x14

884 /* MC_CMD_PORT_READ32_IN msgrequest */
885 #define MC_CMD_PORT_READ32_IN_LEN 4
886 #define MC_CMD_PORT_READ32_IN_ADDR_OFST 0

888 /* MC_CMD_PORT_READ32_OUT msgresponse */
889 #define MC_CMD_PORT_READ32_OUT_LEN 8
890 #define MC_CMD_PORT_READ32_OUT_VALUE_OFST 0
891 #define MC_CMD_PORT_READ32_OUT_STATUS_OFST 4


894 /***********************************/
895 /* MC_CMD_PORT_WRITE32
896  * Write a 32-bit register to the indirect port register map.
897  */
898 #define MC_CMD_PORT_WRITE32  0x15

900 /* MC_CMD_PORT_WRITE32_IN msgrequest */
901 #define MC_CMD_PORT_WRITE32_IN_LEN 8
902 #define MC_CMD_PORT_WRITE32_IN_ADDR_OFST 0
903 #define MC_CMD_PORT_WRITE32_IN_VALUE_OFST 4

905 /* MC_CMD_PORT_WRITE32_OUT msgresponse */
906 #define MC_CMD_PORT_WRITE32_OUT_LEN 4
907 #define MC_CMD_PORT_WRITE32_OUT_STATUS_OFST 0


910 /***********************************/
911 /* MC_CMD_PORT_READ128
912  * Read a 128-bit register from the indirect port register map.
913  */
914 #define MC_CMD_PORT_READ128  0x16

916 /* MC_CMD_PORT_READ128_IN msgrequest */
917 #define MC_CMD_PORT_READ128_IN_LEN 4
918 #define MC_CMD_PORT_READ128_IN_ADDR_OFST 0
```

```
 920 /* MC_CMD_PORT_READ128_OUT msgresponse */
 921 #define MC_CMD_PORT_READ128_OUT_LEN 20
 922 #define MC_CMD_PORT_READ128_OUT_VALUE_OFST 0
 923 #define MC_CMD_PORT_READ128_OUT_VALUE_LEN 16
 924 #define MC_CMD_PORT_READ128_OUT_STATUS_OFST 16


 927 /***********************************/
 928 /* MC_CMD_PORT_WRITE128
 929  * Write a 128-bit register to the indirect port register map.
 930  */
 931 #define MC_CMD_PORT_WRITE128  0x17

 933 /* MC_CMD_PORT_WRITE128_IN msgrequest */
 934 #define MC_CMD_PORT_WRITE128_IN_LEN 20
 935 #define MC_CMD_PORT_WRITE128_IN_ADDR_OFST 0
 936 #define MC_CMD_PORT_WRITE128_IN_VALUE_OFST 4
 937 #define MC_CMD_PORT_WRITE128_IN_VALUE_LEN 16

 939 /* MC_CMD_PORT_WRITE128_OUT msgresponse */
 940 #define MC_CMD_PORT_WRITE128_OUT_LEN 4
 941 #define MC_CMD_PORT_WRITE128_OUT_STATUS_OFST 0


 944 /***********************************/
 945 /* MC_CMD_GET_BOARD_CFG
 946  * Returns the MC firmware configuration structure.
 947  */
 948 #define MC_CMD_GET_BOARD_CFG  0x18

 950 /* MC_CMD_GET_BOARD_CFG_IN msgrequest */
 951 #define MC_CMD_GET_BOARD_CFG_IN_LEN 0

 953 /* MC_CMD_GET_BOARD_CFG_OUT msgresponse */
 954 #define MC_CMD_GET_BOARD_CFG_OUT_LENMIN 96
 955 #define MC_CMD_GET_BOARD_CFG_OUT_LENMAX 136
 956 #define MC_CMD_GET_BOARD_CFG_OUT_LEN(num) (72+2*(num))
 957 #define MC_CMD_GET_BOARD_CFG_OUT_BOARD_TYPE_OFST 0
 958 #define MC_CMD_GET_BOARD_CFG_OUT_BOARD_NAME_OFST 4
 959 #define MC_CMD_GET_BOARD_CFG_OUT_BOARD_NAME_LEN 32
 960 #define MC_CMD_GET_BOARD_CFG_OUT_CAPABILITIES_PORT0_OFST 36
 961 #define MC_CMD_CAPABILITIES_SMALL_BUF_TBL_LBN 0x0 /* enum */
 962 #define MC_CMD_CAPABILITIES_SMALL_BUF_TBL_WIDTH 0x1 /* enum */
 963 #define MC_CMD_CAPABILITIES_TURBO_LBN 0x1 /* enum */
 964 #define MC_CMD_CAPABILITIES_TURBO_WIDTH 0x1 /* enum */
 965 #define MC_CMD_CAPABILITIES_TURBO_ACTIVE_LBN 0x2 /* enum */
 966 #define MC_CMD_CAPABILITIES_TURBO_ACTIVE_WIDTH 0x1 /* enum */
 967 #define MC_CMD_CAPABILITIES_PTP_LBN 0x3 /* enum */
 968 #define MC_CMD_CAPABILITIES_PTP_WIDTH 0x1 /* enum */
 969 #define MC_CMD_GET_BOARD_CFG_OUT_CAPABILITIES_PORT1_OFST 40
 970 /*            Enum values, see field(s): */
 971 /*               CAPABILITIES_PORT0 */
 972 #define MC_CMD_GET_BOARD_CFG_OUT_MAC_ADDR_BASE_PORT0_OFST 44
 973 #define MC_CMD_GET_BOARD_CFG_OUT_MAC_ADDR_BASE_PORT0_LEN 6
 974 #define MC_CMD_GET_BOARD_CFG_OUT_MAC_ADDR_BASE_PORT1_OFST 50
 975 #define MC_CMD_GET_BOARD_CFG_OUT_MAC_ADDR_BASE_PORT1_LEN 6
 976 #define MC_CMD_GET_BOARD_CFG_OUT_MAC_COUNT_PORT0_OFST 56
 977 #define MC_CMD_GET_BOARD_CFG_OUT_MAC_COUNT_PORT1_OFST 60
 978 #define MC_CMD_GET_BOARD_CFG_OUT_MAC_STRIDE_PORT0_OFST 64
 979 #define MC_CMD_GET_BOARD_CFG_OUT_MAC_STRIDE_PORT1_OFST 68
 980 #define MC_CMD_GET_BOARD_CFG_OUT_FW_SUBTYPE_LIST_OFST 72
 981 #define MC_CMD_GET_BOARD_CFG_OUT_FW_SUBTYPE_LIST_LEN 2
 982 #define MC_CMD_GET_BOARD_CFG_OUT_FW_SUBTYPE_LIST_MINNUM 12
 983 #define MC_CMD_GET_BOARD_CFG_OUT_FW_SUBTYPE_LIST_MAXNUM 32
```



```
 986 /***********************************/
 987 /* MC_CMD_DBI_READX
 988  * Read DBI register(s).
 989  */
 990 #define MC_CMD_DBI_READX  0x19

 992 /* MC_CMD_DBI_READX_IN msgrequest */
 993 #define MC_CMD_DBI_READX_IN_LENMIN 8
 994 #define MC_CMD_DBI_READX_IN_LENMAX 248
 995 #define MC_CMD_DBI_READX_IN_LEN(num) (0+8*(num))
 996 #define MC_CMD_DBI_READX_IN_DBIRDOP_OFST 0
 997 #define MC_CMD_DBI_READX_IN_DBIRDOP_LEN 8
 998 #define MC_CMD_DBI_READX_IN_DBIRDOP_LO_OFST 0
 999 #define MC_CMD_DBI_READX_IN_DBIRDOP_HI_OFST 4
1000 #define MC_CMD_DBI_READX_IN_DBIRDOP_MINNUM 1
1001 #define MC_CMD_DBI_READX_IN_DBIRDOP_MAXNUM 31

1003 /* MC_CMD_DBI_READX_OUT msgresponse */
1004 #define MC_CMD_DBI_READX_OUT_LENMIN 4
1005 #define MC_CMD_DBI_READX_OUT_LENMAX 252
1006 #define MC_CMD_DBI_READX_OUT_LEN(num) (0+4*(num))
1007 #define MC_CMD_DBI_READX_OUT_VALUE_OFST 0
1008 #define MC_CMD_DBI_READX_OUT_VALUE_LEN 4
1009 #define MC_CMD_DBI_READX_OUT_VALUE_MINNUM 1
1010 #define MC_CMD_DBI_READX_OUT_VALUE_MAXNUM 63


1013 /***********************************/
1014 /* MC_CMD_SET_RAND_SEED
1015  * Set the 16byte seed for the MC pseudo-random generator.
1016  */
1017 #define MC_CMD_SET_RAND_SEED  0x1a

1019 /* MC_CMD_SET_RAND_SEED_IN msgrequest */
1020 #define MC_CMD_SET_RAND_SEED_IN_LEN 16
1021 #define MC_CMD_SET_RAND_SEED_IN_SEED_OFST 0
1022 #define MC_CMD_SET_RAND_SEED_IN_SEED_LEN 16

1024 /* MC_CMD_SET_RAND_SEED_OUT msgresponse */
1025 #define MC_CMD_SET_RAND_SEED_OUT_LEN 0


1028 /***********************************/
1029 /* MC_CMD_LTSSM_HIST
1030  * Retrieve the history of the PCIE LTSSM.
1031  */
1032 #define MC_CMD_LTSSM_HIST  0x1b

1034 /* MC_CMD_LTSSM_HIST_IN msgrequest */
1035 #define MC_CMD_LTSSM_HIST_IN_LEN 0

1037 /* MC_CMD_LTSSM_HIST_OUT msgresponse */
1038 #define MC_CMD_LTSSM_HIST_OUT_LENMIN 0
1039 #define MC_CMD_LTSSM_HIST_OUT_LENMAX 252
1040 #define MC_CMD_LTSSM_HIST_OUT_LEN(num) (0+4*(num))
1041 #define MC_CMD_LTSSM_HIST_OUT_DATA_OFST 0
1042 #define MC_CMD_LTSSM_HIST_OUT_DATA_LEN 4
1043 #define MC_CMD_LTSSM_HIST_OUT_DATA_MINNUM 0
1044 #define MC_CMD_LTSSM_HIST_OUT_DATA_MAXNUM 63


1047 /***********************************/
1048 /* MC_CMD_DRV_ATTACH
1049  * Inform MCPU that this port is managed on the host.
1050  */
1051 #define MC_CMD_DRV_ATTACH  0x1c
```

```
/* MC_CMD_DRV_ATTACH_IN msgrequest */
#define MC_CMD_DRV_ATTACH_IN_LEN 8
#define MC_CMD_DRV_ATTACH_IN_NEW_STATE_OFST 0
#define MC_CMD_DRV_ATTACH_IN_UPDATE_OFST 4

/* MC_CMD_DRV_ATTACH_OUT msgresponse */
#define MC_CMD_DRV_ATTACH_OUT_LEN 4
#define MC_CMD_DRV_ATTACH_OUT_OLD_STATE_OFST 0


/***********************************/
/* MC_CMD_NCSI_PROD
 * Trigger an NC-SI event.
 */
#define MC_CMD_NCSI_PROD  0x1d

/* MC_CMD_NCSI_PROD_IN msgrequest */
#define MC_CMD_NCSI_PROD_IN_LEN 4
#define MC_CMD_NCSI_PROD_IN_EVENTS_OFST 0
#define MC_CMD_NCSI_PROD_LINKCHANGE 0x0 /* enum */
#define MC_CMD_NCSI_PROD_RESET 0x1 /* enum */
#define MC_CMD_NCSI_PROD_DRVATTACH 0x2 /* enum */
#define MC_CMD_NCSI_PROD_IN_LINKCHANGE_LBN 0
#define MC_CMD_NCSI_PROD_IN_LINKCHANGE_WIDTH 1
#define MC_CMD_NCSI_PROD_IN_RESET_LBN 1
#define MC_CMD_NCSI_PROD_IN_RESET_WIDTH 1
#define MC_CMD_NCSI_PROD_IN_DRVATTACH_LBN 2
#define MC_CMD_NCSI_PROD_IN_DRVATTACH_WIDTH 1

/* MC_CMD_NCSI_PROD_OUT msgresponse */
#define MC_CMD_NCSI_PROD_OUT_LEN 0


/***********************************/
/* MC_CMD_SHMUART
 * Route UART output to circular buffer in shared memory instead.
 */
#define MC_CMD_SHMUART  0x1f

/* MC_CMD_SHMUART_IN msgrequest */
#define MC_CMD_SHMUART_IN_LEN 4
#define MC_CMD_SHMUART_IN_FLAG_OFST 0

/* MC_CMD_SHMUART_OUT msgresponse */
#define MC_CMD_SHMUART_OUT_LEN 0


/***********************************/
/* MC_CMD_PORT_RESET
 * Generic per-port reset.
 */
#define MC_CMD_PORT_RESET  0x20

/* MC_CMD_PORT_RESET_IN msgrequest */
#define MC_CMD_PORT_RESET_IN_LEN 0

/* MC_CMD_PORT_RESET_OUT msgresponse */
#define MC_CMD_PORT_RESET_OUT_LEN 0


/***********************************/
/* MC_CMD_PCIE_CREDITS
 * Read instantaneous and minimum flow control thresholds.
 */
#define MC_CMD_PCIE_CREDITS  0x21
```



```
/* MC_CMD_PCIE_CREDITS_IN msgrequest */
#define MC_CMD_PCIE_CREDITS_IN_LEN 8
#define MC_CMD_PCIE_CREDITS_IN_POLL_PERIOD_OFST 0
#define MC_CMD_PCIE_CREDITS_IN_WIPE_OFST 4

/* MC_CMD_PCIE_CREDITS_OUT msgresponse */
#define MC_CMD_PCIE_CREDITS_OUT_LEN 16
#define MC_CMD_PCIE_CREDITS_OUT_CURRENT_P_HDR_OFST 0
#define MC_CMD_PCIE_CREDITS_OUT_CURRENT_P_HDR_LEN 2
#define MC_CMD_PCIE_CREDITS_OUT_CURRENT_P_DATA_OFST 2
#define MC_CMD_PCIE_CREDITS_OUT_CURRENT_P_DATA_LEN 2
#define MC_CMD_PCIE_CREDITS_OUT_CURRENT_NP_HDR_OFST 4
#define MC_CMD_PCIE_CREDITS_OUT_CURRENT_NP_HDR_LEN 2
#define MC_CMD_PCIE_CREDITS_OUT_CURRENT_NP_DATA_OFST 6
#define MC_CMD_PCIE_CREDITS_OUT_CURRENT_NP_DATA_LEN 2
#define MC_CMD_PCIE_CREDITS_OUT_MINIMUM_P_HDR_OFST 8
#define MC_CMD_PCIE_CREDITS_OUT_MINIMUM_P_HDR_LEN 2
#define MC_CMD_PCIE_CREDITS_OUT_MINIMUM_P_DATA_OFST 10
#define MC_CMD_PCIE_CREDITS_OUT_MINIMUM_P_DATA_LEN 2
#define MC_CMD_PCIE_CREDITS_OUT_MINIMUM_NP_HDR_OFST 12
#define MC_CMD_PCIE_CREDITS_OUT_MINIMUM_NP_HDR_LEN 2
#define MC_CMD_PCIE_CREDITS_OUT_MINIMUM_NP_DATA_OFST 14
#define MC_CMD_PCIE_CREDITS_OUT_MINIMUM_NP_DATA_LEN 2


/***********************************/
/* MC_CMD_RXD_MONITOR
 * Get histogram of RX queue fill level.
 */
#define MC_CMD_RXD_MONITOR  0x22

/* MC_CMD_RXD_MONITOR_IN msgrequest */
#define MC_CMD_RXD_MONITOR_IN_LEN 12
#define MC_CMD_RXD_MONITOR_IN_QID_OFST 0
#define MC_CMD_RXD_MONITOR_IN_POLL_PERIOD_OFST 4
#define MC_CMD_RXD_MONITOR_IN_WIPE_OFST 8

/* MC_CMD_RXD_MONITOR_OUT msgresponse */
#define MC_CMD_RXD_MONITOR_OUT_LEN 80
#define MC_CMD_RXD_MONITOR_OUT_QID_OFST 0
#define MC_CMD_RXD_MONITOR_OUT_RING_FILL_OFST 4
#define MC_CMD_RXD_MONITOR_OUT_CACHE_FILL_OFST 8
#define MC_CMD_RXD_MONITOR_OUT_RING_LT_1_OFST 12
#define MC_CMD_RXD_MONITOR_OUT_RING_LT_2_OFST 16
#define MC_CMD_RXD_MONITOR_OUT_RING_LT_4_OFST 20
#define MC_CMD_RXD_MONITOR_OUT_RING_LT_8_OFST 24
#define MC_CMD_RXD_MONITOR_OUT_RING_LT_16_OFST 28
#define MC_CMD_RXD_MONITOR_OUT_RING_LT_32_OFST 32
#define MC_CMD_RXD_MONITOR_OUT_RING_LT_64_OFST 36
#define MC_CMD_RXD_MONITOR_OUT_RING_LT_128_OFST 40
#define MC_CMD_RXD_MONITOR_OUT_RING_LT_256_OFST 44
#define MC_CMD_RXD_MONITOR_OUT_RING_GE_256_OFST 48
#define MC_CMD_RXD_MONITOR_OUT_CACHE_LT_1_OFST 52
#define MC_CMD_RXD_MONITOR_OUT_CACHE_LT_2_OFST 56
#define MC_CMD_RXD_MONITOR_OUT_CACHE_LT_4_OFST 60
#define MC_CMD_RXD_MONITOR_OUT_CACHE_LT_8_OFST 64
#define MC_CMD_RXD_MONITOR_OUT_CACHE_LT_16_OFST 68
#define MC_CMD_RXD_MONITOR_OUT_CACHE_LT_32_OFST 72
#define MC_CMD_RXD_MONITOR_OUT_CACHE_GE_32_OFST 76


/***********************************/
/* MC_CMD_PUTS
 * puts(3) implementation over MCDI
 */
```

```
#define MC_CMD_PUTS  0x23

/* MC_CMD_PUTS_IN msgrequest */
#define MC_CMD_PUTS_IN_LENMIN 13
#define MC_CMD_PUTS_IN_LENMAX 252
#define MC_CMD_PUTS_IN_LEN(num) (12+1*(num))
#define MC_CMD_PUTS_IN_DEST_OFST 0
#define MC_CMD_PUTS_IN_UART_LBN 0
#define MC_CMD_PUTS_IN_UART_WIDTH 1
#define MC_CMD_PUTS_IN_PORT_LBN 1
#define MC_CMD_PUTS_IN_PORT_WIDTH 1
#define MC_CMD_PUTS_IN_DHOST_OFST 4
#define MC_CMD_PUTS_IN_DHOST_LEN 6
#define MC_CMD_PUTS_IN_STRING_OFST 12
#define MC_CMD_PUTS_IN_STRING_LEN 1
#define MC_CMD_PUTS_IN_STRING_MINNUM 1
#define MC_CMD_PUTS_IN_STRING_MAXNUM 240

/* MC_CMD_PUTS_OUT msgresponse */
#define MC_CMD_PUTS_OUT_LEN 0


/***********************************/
/* MC_CMD_GET_PHY_CFG
 * Report PHY configuration.
 */
#define MC_CMD_GET_PHY_CFG  0x24

/* MC_CMD_GET_PHY_CFG_IN msgrequest */
#define MC_CMD_GET_PHY_CFG_IN_LEN 0

/* MC_CMD_GET_PHY_CFG_OUT msgresponse */
#define MC_CMD_GET_PHY_CFG_OUT_LEN 72
#define MC_CMD_GET_PHY_CFG_OUT_FLAGS_OFST 0
#define MC_CMD_GET_PHY_CFG_OUT_PRESENT_LBN 0
#define MC_CMD_GET_PHY_CFG_OUT_PRESENT_WIDTH 1
#define MC_CMD_GET_PHY_CFG_OUT_BIST_CABLE_SHORT_LBN 1
#define MC_CMD_GET_PHY_CFG_OUT_BIST_CABLE_SHORT_WIDTH 1
#define MC_CMD_GET_PHY_CFG_OUT_BIST_CABLE_LONG_LBN 2
#define MC_CMD_GET_PHY_CFG_OUT_BIST_CABLE_LONG_WIDTH 1
#define MC_CMD_GET_PHY_CFG_OUT_LOWPOWER_LBN 3
#define MC_CMD_GET_PHY_CFG_OUT_LOWPOWER_WIDTH 1
#define MC_CMD_GET_PHY_CFG_OUT_POWEROFF_LBN 4
#define MC_CMD_GET_PHY_CFG_OUT_POWEROFF_WIDTH 1
#define MC_CMD_GET_PHY_CFG_OUT_TXDIS_LBN 5
#define MC_CMD_GET_PHY_CFG_OUT_TXDIS_WIDTH 1
#define MC_CMD_GET_PHY_CFG_OUT_BIST_LBN 6
#define MC_CMD_GET_PHY_CFG_OUT_BIST_WIDTH 1
#define MC_CMD_GET_PHY_CFG_OUT_TYPE_OFST 4
#define MC_CMD_GET_PHY_CFG_OUT_SUPPORTED_CAP_OFST 8
#define MC_CMD_PHY_CAP_10HDX_LBN 1
#define MC_CMD_PHY_CAP_10HDX_WIDTH 1
#define MC_CMD_PHY_CAP_10FDX_LBN 2
#define MC_CMD_PHY_CAP_10FDX_WIDTH 1
#define MC_CMD_PHY_CAP_100HDX_LBN 3
#define MC_CMD_PHY_CAP_100HDX_WIDTH 1
#define MC_CMD_PHY_CAP_100FDX_LBN 4
#define MC_CMD_PHY_CAP_100FDX_WIDTH 1
#define MC_CMD_PHY_CAP_1000HDX_LBN 5
#define MC_CMD_PHY_CAP_1000HDX_WIDTH 1
#define MC_CMD_PHY_CAP_1000FDX_LBN 6
#define MC_CMD_PHY_CAP_1000FDX_WIDTH 1
#define MC_CMD_PHY_CAP_10000FDX_LBN 7
#define MC_CMD_PHY_CAP_10000FDX_WIDTH 1
#define MC_CMD_PHY_CAP_PAUSE_LBN 8
#define MC_CMD_PHY_CAP_PAUSE_WIDTH 1
```



```
#define MC_CMD_PHY_CAP_ASYM_LBN 9
#define MC_CMD_PHY_CAP_ASYM_WIDTH 1
#define MC_CMD_PHY_CAP_AN_LBN 10
#define MC_CMD_PHY_CAP_AN_WIDTH 1
#define MC_CMD_GET_PHY_CFG_OUT_CHANNEL_OFST 12
#define MC_CMD_GET_PHY_CFG_OUT_PRT_OFST 16
#define MC_CMD_GET_PHY_CFG_OUT_STATS_MASK_OFST 20
#define MC_CMD_GET_PHY_CFG_OUT_NAME_OFST 24
#define MC_CMD_GET_PHY_CFG_OUT_NAME_LEN 20
#define MC_CMD_GET_PHY_CFG_OUT_MEDIA_TYPE_OFST 44
#define MC_CMD_MEDIA_XAUI 0x1 /* enum */
#define MC_CMD_MEDIA_CX4 0x2 /* enum */
#define MC_CMD_MEDIA_KX4 0x3 /* enum */
#define MC_CMD_MEDIA_XFP 0x4 /* enum */
#define MC_CMD_MEDIA_SFP_PLUS 0x5 /* enum */
#define MC_CMD_MEDIA_BASE_T 0x6 /* enum */
#define MC_CMD_GET_PHY_CFG_OUT_MMD_MASK_OFST 48
#define MC_CMD_MMD_CLAUSE22 0x0 /* enum */
#define MC_CMD_MMD_CLAUSE45_PMAPMD 0x1 /* enum */
#define MC_CMD_MMD_CLAUSE45_WIS 0x2 /* enum */
#define MC_CMD_MMD_CLAUSE45_PCS 0x3 /* enum */
#define MC_CMD_MMD_CLAUSE45_PHYXS 0x4 /* enum */
#define MC_CMD_MMD_CLAUSE45_DTEXS 0x5 /* enum */
#define MC_CMD_MMD_CLAUSE45_TC 0x6 /* enum */
#define MC_CMD_MMD_CLAUSE45_AN 0x7 /* enum */
#define MC_CMD_MMD_CLAUSE45_C22EXT 0x1d /* enum */
#define MC_CMD_MMD_CLAUSE45_VEND1 0x1e /* enum */
#define MC_CMD_MMD_CLAUSE45_VEND2 0x1f /* enum */
#define MC_CMD_GET_PHY_CFG_OUT_REVISION_OFST 52
#define MC_CMD_GET_PHY_CFG_OUT_REVISION_LEN 20


/***********************************/
/* MC_CMD_START_BIST
 * Start a BIST test on the PHY.
 */
#define MC_CMD_START_BIST  0x25

/* MC_CMD_START_BIST_IN msgrequest */
#define MC_CMD_START_BIST_IN_LEN 4
#define MC_CMD_START_BIST_IN_TYPE_OFST 0
#define MC_CMD_PHY_BIST_CABLE_SHORT 0x1 /* enum */
#define MC_CMD_PHY_BIST_CABLE_LONG 0x2 /* enum */
#define MC_CMD_BPX_SERDES_BIST 0x3 /* enum */
#define MC_CMD_MC_LOOPBACK_BIST 0x4 /* enum */
#define MC_CMD_PHY_BIST 0x5 /* enum */

/* MC_CMD_START_BIST_OUT msgresponse */
#define MC_CMD_START_BIST_OUT_LEN 0


/***********************************/
/* MC_CMD_POLL_BIST
 * Poll for BIST completion.
 */
#define MC_CMD_POLL_BIST  0x26

/* MC_CMD_POLL_BIST_IN msgrequest */
#define MC_CMD_POLL_BIST_IN_LEN 0

/* MC_CMD_POLL_BIST_OUT msgresponse */
#define MC_CMD_POLL_BIST_OUT_LEN 8
#define MC_CMD_POLL_BIST_OUT_RESULT_OFST 0
#define MC_CMD_POLL_BIST_RUNNING 0x1 /* enum */
#define MC_CMD_POLL_BIST_PASSED 0x2 /* enum */
#define MC_CMD_POLL_BIST_FAILED 0x3 /* enum */
```

```
#define MC_CMD_POLL_BIST_TIMEOUT 0x4 /* enum */
#define MC_CMD_POLL_BIST_OUT_PRIVATE_OFST 4

/* MC_CMD_POLL_BIST_OUT_SFT9001 msgresponse */
#define MC_CMD_POLL_BIST_OUT_SFT9001_LEN 36
/*            MC_CMD_POLL_BIST_OUT_RESULT_OFST 0 */
/*            Enum values, see field(s): */
/*               MC_CMD_POLL_BIST_OUT/MC_CMD_POLL_BIST_OUT_RESULT */
#define MC_CMD_POLL_BIST_OUT_SFT9001_CABLE_LENGTH_A_OFST 4
#define MC_CMD_POLL_BIST_OUT_SFT9001_CABLE_LENGTH_B_OFST 8
#define MC_CMD_POLL_BIST_OUT_SFT9001_CABLE_LENGTH_C_OFST 12
#define MC_CMD_POLL_BIST_OUT_SFT9001_CABLE_LENGTH_D_OFST 16
#define MC_CMD_POLL_BIST_OUT_SFT9001_CABLE_STATUS_A_OFST 20
#define MC_CMD_POLL_BIST_SFT9001_PAIR_OK 0x1 /* enum */
#define MC_CMD_POLL_BIST_SFT9001_PAIR_OPEN 0x2 /* enum */
#define MC_CMD_POLL_BIST_SFT9001_INTRA_PAIR_SHORT 0x3 /* enum */
#define MC_CMD_POLL_BIST_SFT9001_INTER_PAIR_SHORT 0x4 /* enum */
#define MC_CMD_POLL_BIST_SFT9001_PAIR_BUSY 0x9 /* enum */
#define MC_CMD_POLL_BIST_OUT_SFT9001_CABLE_STATUS_B_OFST 24
/*            Enum values, see field(s): */
/*               CABLE_STATUS_A */
#define MC_CMD_POLL_BIST_OUT_SFT9001_CABLE_STATUS_C_OFST 28
/*            Enum values, see field(s): */
/*               CABLE_STATUS_A */
#define MC_CMD_POLL_BIST_OUT_SFT9001_CABLE_STATUS_D_OFST 32
/*            Enum values, see field(s): */
/*               CABLE_STATUS_A */

/* MC_CMD_POLL_BIST_OUT_MRSFP msgresponse */
#define MC_CMD_POLL_BIST_OUT_MRSFP_LEN 8
/*            MC_CMD_POLL_BIST_OUT_RESULT_OFST 0 */
/*            Enum values, see field(s): */
/*               MC_CMD_POLL_BIST_OUT/MC_CMD_POLL_BIST_OUT_RESULT */
#define MC_CMD_POLL_BIST_OUT_MRSFP_TEST_OFST 4
#define MC_CMD_POLL_BIST_MRSFP_TEST_COMPLETE 0x0 /* enum */
#define MC_CMD_POLL_BIST_MRSFP_TEST_BUS_SWITCH_OFF_I2C_WRITE 0x1 /* enum */
#define MC_CMD_POLL_BIST_MRSFP_TEST_BUS_SWITCH_OFF_I2C_NO_ACCESS_IO_EXP 0x2 /* e
#define MC_CMD_POLL_BIST_MRSFP_TEST_BUS_SWITCH_OFF_I2C_NO_ACCESS_MODULE 0x3 /* e
#define MC_CMD_POLL_BIST_MRSFP_TEST_IO_EXP_I2C_CONFIGURE 0x4 /* enum */
#define MC_CMD_POLL_BIST_MRSFP_TEST_BUS_SWITCH_I2C_NO_CROSSTALK 0x5 /* enum */
#define MC_CMD_POLL_BIST_MRSFP_TEST_MODULE_PRESENCE 0x6 /* enum */
#define MC_CMD_POLL_BIST_MRSFP_TEST_MODULE_ID_I2C_ACCESS 0x7 /* enum */
#define MC_CMD_POLL_BIST_MRSFP_TEST_MODULE_ID_SANE_VALUE 0x8 /* enum */


/***********************************/
/* MC_CMD_FLUSH_RX_QUEUES
 * Flush receive queue(s).
 */
#define MC_CMD_FLUSH_RX_QUEUES  0x27

/* MC_CMD_FLUSH_RX_QUEUES_IN msgrequest */
#define MC_CMD_FLUSH_RX_QUEUES_IN_LENMIN 4
#define MC_CMD_FLUSH_RX_QUEUES_IN_LENMAX 252
#define MC_CMD_FLUSH_RX_QUEUES_IN_LEN(num) (0+4*(num))
#define MC_CMD_FLUSH_RX_QUEUES_IN_QID_OFST_OFST 0
#define MC_CMD_FLUSH_RX_QUEUES_IN_QID_OFST_LEN 4
#define MC_CMD_FLUSH_RX_QUEUES_IN_QID_OFST_MINNUM 1
#define MC_CMD_FLUSH_RX_QUEUES_IN_QID_OFST_MAXNUM 63

/* MC_CMD_FLUSH_RX_QUEUES_OUT msgresponse */
#define MC_CMD_FLUSH_RX_QUEUES_OUT_LEN 0


/***********************************/
/* MC_CMD_GET_LOOPBACK_MODES
```



```
 * Get port's loopback modes.
 */
#define MC_CMD_GET_LOOPBACK_MODES  0x28

/* MC_CMD_GET_LOOPBACK_MODES_IN msgrequest */
#define MC_CMD_GET_LOOPBACK_MODES_IN_LEN 0

/* MC_CMD_GET_LOOPBACK_MODES_OUT msgresponse */
#define MC_CMD_GET_LOOPBACK_MODES_OUT_LEN 32
#define MC_CMD_GET_LOOPBACK_MODES_OUT_100M_OFST 0
#define MC_CMD_GET_LOOPBACK_MODES_OUT_100M_LEN 8
#define MC_CMD_GET_LOOPBACK_MODES_OUT_100M_LO_OFST 0
#define MC_CMD_GET_LOOPBACK_MODES_OUT_100M_HI_OFST 4
#define MC_CMD_LOOPBACK_NONE  0x0 /* enum */
#define MC_CMD_LOOPBACK_DATA  0x1 /* enum */
#define MC_CMD_LOOPBACK_GMAC  0x2 /* enum */
#define MC_CMD_LOOPBACK_XGMII 0x3 /* enum */
#define MC_CMD_LOOPBACK_XGXS  0x4 /* enum */
#define MC_CMD_LOOPBACK_XAUI  0x5 /* enum */
#define MC_CMD_LOOPBACK_GMII  0x6 /* enum */
#define MC_CMD_LOOPBACK_SGMII  0x7 /* enum */
#define MC_CMD_LOOPBACK_XGBR  0x8 /* enum */
#define MC_CMD_LOOPBACK_XFI  0x9 /* enum */
#define MC_CMD_LOOPBACK_XAUI_FAR  0xa /* enum */
#define MC_CMD_LOOPBACK_GMII_FAR  0xb /* enum */
#define MC_CMD_LOOPBACK_SGMII_FAR  0xc /* enum */
#define MC_CMD_LOOPBACK_XFI_FAR  0xd /* enum */
#define MC_CMD_LOOPBACK_GPHY  0xe /* enum */
#define MC_CMD_LOOPBACK_PHYXS  0xf /* enum */
#define MC_CMD_LOOPBACK_PCS  0x10 /* enum */
#define MC_CMD_LOOPBACK_PMAPMD  0x11 /* enum */
#define MC_CMD_LOOPBACK_XPORT  0x12 /* enum */
#define MC_CMD_LOOPBACK_XGMII_WS  0x13 /* enum */
#define MC_CMD_LOOPBACK_XAUI_WS  0x14 /* enum */
#define MC_CMD_LOOPBACK_XAUI_WS_FAR  0x15 /* enum */
#define MC_CMD_LOOPBACK_XAUI_WS_NEAR  0x16 /* enum */
#define MC_CMD_LOOPBACK_GMII_WS  0x17 /* enum */
#define MC_CMD_LOOPBACK_XFI_WS  0x18 /* enum */
#define MC_CMD_LOOPBACK_XFI_WS_FAR  0x19 /* enum */
#define MC_CMD_LOOPBACK_PHYXS_WS  0x1a /* enum */
#define MC_CMD_GET_LOOPBACK_MODES_OUT_1G_OFST 8
#define MC_CMD_GET_LOOPBACK_MODES_OUT_1G_LEN 8
#define MC_CMD_GET_LOOPBACK_MODES_OUT_1G_LO_OFST 8
#define MC_CMD_GET_LOOPBACK_MODES_OUT_1G_HI_OFST 12
/*            Enum values, see field(s): */
/*               100M */
#define MC_CMD_GET_LOOPBACK_MODES_OUT_10G_OFST 16
#define MC_CMD_GET_LOOPBACK_MODES_OUT_10G_LEN 8
#define MC_CMD_GET_LOOPBACK_MODES_OUT_10G_LO_OFST 16
#define MC_CMD_GET_LOOPBACK_MODES_OUT_10G_HI_OFST 20
/*            Enum values, see field(s): */
/*               100M */
#define MC_CMD_GET_LOOPBACK_MODES_OUT_SUGGESTED_OFST 24
#define MC_CMD_GET_LOOPBACK_MODES_OUT_SUGGESTED_LEN 8
#define MC_CMD_GET_LOOPBACK_MODES_OUT_SUGGESTED_LO_OFST 24
#define MC_CMD_GET_LOOPBACK_MODES_OUT_SUGGESTED_HI_OFST 28
/*            Enum values, see field(s): */
/*               100M */


/***********************************/
/* MC_CMD_GET_LINK
 * Read the unified MAC/PHY link state.
 */
#define MC_CMD_GET_LINK  0x29
```

```
1448 /* MC_CMD_GET_LINK_IN msgrequest */
1449 #define MC_CMD_GET_LINK_IN_LEN 0

1451 /* MC_CMD_GET_LINK_OUT msgresponse */
1452 #define MC_CMD_GET_LINK_OUT_LEN 28
1453 #define MC_CMD_GET_LINK_OUT_CAP_OFST 0
1454 #define MC_CMD_GET_LINK_OUT_LP_CAP_OFST 4
1455 #define MC_CMD_GET_LINK_OUT_LINK_SPEED_OFST 8
1456 #define MC_CMD_GET_LINK_OUT_LOOPBACK_MODE_OFST 12
1457 /*            Enum values, see field(s): */
1458 /*               MC_CMD_GET_LOOPBACK_MODES/MC_CMD_GET_LOOPBACK_MODES_OUT/100M */
1459 #define MC_CMD_GET_LINK_OUT_FLAGS_OFST 16
1460 #define MC_CMD_GET_LINK_OUT_LINK_UP_LBN 0
1461 #define MC_CMD_GET_LINK_OUT_LINK_UP_WIDTH 1
1462 #define MC_CMD_GET_LINK_OUT_FULL_DUPLEX_LBN 1
1463 #define MC_CMD_GET_LINK_OUT_FULL_DUPLEX_WIDTH 1
1464 #define MC_CMD_GET_LINK_OUT_BPX_LINK_LBN 2
1465 #define MC_CMD_GET_LINK_OUT_BPX_LINK_WIDTH 1
1466 #define MC_CMD_GET_LINK_OUT_PHY_LINK_LBN 3
1467 #define MC_CMD_GET_LINK_OUT_PHY_LINK_WIDTH 1
1468 #define MC_CMD_GET_LINK_OUT_FCNTL_OFST 20
1469 #define MC_CMD_FCNTL_OFF 0x0 /* enum */
1470 #define MC_CMD_FCNTL_RESPOND 0x1 /* enum */
1471 #define MC_CMD_FCNTL_BIDIR 0x2 /* enum */
1472 #define MC_CMD_GET_LINK_OUT_MAC_FAULT_OFST 24
1473 #define MC_CMD_MAC_FAULT_XGMII_LOCAL_LBN 0
1474 #define MC_CMD_MAC_FAULT_XGMII_LOCAL_WIDTH 1
1475 #define MC_CMD_MAC_FAULT_XGMII_REMOTE_LBN 1
1476 #define MC_CMD_MAC_FAULT_XGMII_REMOTE_WIDTH 1
1477 #define MC_CMD_MAC_FAULT_SGMII_REMOTE_LBN 2
1478 #define MC_CMD_MAC_FAULT_SGMII_REMOTE_WIDTH 1
1479 #define MC_CMD_MAC_FAULT_PENDING_RECONFIG_LBN 3
1480 #define MC_CMD_MAC_FAULT_PENDING_RECONFIG_WIDTH 1


1483 /***********************************/
1484 /* MC_CMD_SET_LINK
1485  * Write the unified MAC/PHY link configuration.
1486  */
1487 #define MC_CMD_SET_LINK  0x2a

1489 /* MC_CMD_SET_LINK_IN msgrequest */
1490 #define MC_CMD_SET_LINK_IN_LEN 16
1491 #define MC_CMD_SET_LINK_IN_CAP_OFST 0
1492 #define MC_CMD_SET_LINK_IN_FLAGS_OFST 4
1493 #define MC_CMD_SET_LINK_IN_LOWPOWER_LBN 0
1494 #define MC_CMD_SET_LINK_IN_LOWPOWER_WIDTH 1
1495 #define MC_CMD_SET_LINK_IN_POWEROFF_LBN 1
1496 #define MC_CMD_SET_LINK_IN_POWEROFF_WIDTH 1
1497 #define MC_CMD_SET_LINK_IN_TXDIS_LBN 2
1498 #define MC_CMD_SET_LINK_IN_TXDIS_WIDTH 1
1499 #define MC_CMD_SET_LINK_IN_LOOPBACK_MODE_OFST 8
1500 /*            Enum values, see field(s): */
1501 /*               MC_CMD_GET_LOOPBACK_MODES/MC_CMD_GET_LOOPBACK_MODES_OUT/100M */
1502 #define MC_CMD_SET_LINK_IN_LOOPBACK_SPEED_OFST 12

1504 /* MC_CMD_SET_LINK_OUT msgresponse */
1505 #define MC_CMD_SET_LINK_OUT_LEN 0


1508 /***********************************/
1509 /* MC_CMD_SET_ID_LED
1510  * Set indentification LED state.
1511  */
1512 #define MC_CMD_SET_ID_LED  0x2b
```



```
1514 /* MC_CMD_SET_ID_LED_IN msgrequest */
1515 #define MC_CMD_SET_ID_LED_IN_LEN 4
1516 #define MC_CMD_SET_ID_LED_IN_STATE_OFST 0
1517 #define MC_CMD_LED_OFF  0x0 /* enum */
1518 #define MC_CMD_LED_ON  0x1 /* enum */
1519 #define MC_CMD_LED_DEFAULT  0x2 /* enum */

1521 /* MC_CMD_SET_ID_LED_OUT msgresponse */
1522 #define MC_CMD_SET_ID_LED_OUT_LEN 0


1525 /***********************************/
1526 /* MC_CMD_SET_MAC
1527  * Set MAC configuration.
1528  */
1529 #define MC_CMD_SET_MAC  0x2c

1531 /* MC_CMD_SET_MAC_IN msgrequest */
1532 #define MC_CMD_SET_MAC_IN_LEN 24
1533 #define MC_CMD_SET_MAC_IN_MTU_OFST 0
1534 #define MC_CMD_SET_MAC_IN_DRAIN_OFST 4
1535 #define MC_CMD_SET_MAC_IN_ADDR_OFST 8
1536 #define MC_CMD_SET_MAC_IN_ADDR_LEN 8
1537 #define MC_CMD_SET_MAC_IN_ADDR_LO_OFST 8
1538 #define MC_CMD_SET_MAC_IN_ADDR_HI_OFST 12
1539 #define MC_CMD_SET_MAC_IN_REJECT_OFST 16
1540 #define MC_CMD_SET_MAC_IN_REJECT_UNCST_LBN 0
1541 #define MC_CMD_SET_MAC_IN_REJECT_UNCST_WIDTH 1
1542 #define MC_CMD_SET_MAC_IN_REJECT_BRDCST_LBN 1
1543 #define MC_CMD_SET_MAC_IN_REJECT_BRDCST_WIDTH 1
1544 #define MC_CMD_SET_MAC_IN_FCNTL_OFST 20
1545 /*               MC_CMD_FCNTL_OFF 0x0 */
1546 /*               MC_CMD_FCNTL_RESPOND 0x1 */
1547 /*               MC_CMD_FCNTL_BIDIR 0x2 */
1548 #define MC_CMD_FCNTL_AUTO 0x3 /* enum */

1550 /* MC_CMD_SET_MAC_OUT msgresponse */
1551 #define MC_CMD_SET_MAC_OUT_LEN 0


1554 /***********************************/
1555 /* MC_CMD_PHY_STATS
1556  * Get generic PHY statistics.
1557  */
1558 #define MC_CMD_PHY_STATS  0x2d

1560 /* MC_CMD_PHY_STATS_IN msgrequest */
1561 #define MC_CMD_PHY_STATS_IN_LEN 8
1562 #define MC_CMD_PHY_STATS_IN_DMA_ADDR_OFST 0
1563 #define MC_CMD_PHY_STATS_IN_DMA_ADDR_LEN 8
1564 #define MC_CMD_PHY_STATS_IN_DMA_ADDR_LO_OFST 0
1565 #define MC_CMD_PHY_STATS_IN_DMA_ADDR_HI_OFST 4

1567 /* MC_CMD_PHY_STATS_OUT_DMA msgresponse */
1568 #define MC_CMD_PHY_STATS_OUT_DMA_LEN 0

1570 /* MC_CMD_PHY_STATS_OUT_NO_DMA msgresponse */
1571 #define MC_CMD_PHY_STATS_OUT_NO_DMA_LEN (((MC_CMD_PHY_NSTATS*32))>>3)
1572 #define MC_CMD_PHY_STATS_OUT_NO_DMA_STATISTICS_OFST 0
1573 #define MC_CMD_PHY_STATS_OUT_NO_DMA_STATISTICS_LEN 4
1574 #define MC_CMD_PHY_STATS_OUT_NO_DMA_STATISTICS_NUM MC_CMD_PHY_NSTATS
1575 #define MC_CMD_OUI  0x0 /* enum */
1576 #define MC_CMD_PMA_PMD_LINK_UP  0x1 /* enum */
1577 #define MC_CMD_PMA_PMD_RX_FAULT  0x2 /* enum */
1578 #define MC_CMD_PMA_PMD_TX_FAULT  0x3 /* enum */
1579 #define MC_CMD_PMA_PMD_SIGNAL  0x4 /* enum */
```

```
#define MC_CMD_PMA_PMD_SNR_A  0x5 /* enum */
#define MC_CMD_PMA_PMD_SNR_B  0x6 /* enum */
#define MC_CMD_PMA_PMD_SNR_C  0x7 /* enum */
#define MC_CMD_PMA_PMD_SNR_D  0x8 /* enum */
#define MC_CMD_PCS_LINK_UP  0x9 /* enum */
#define MC_CMD_PCS_RX_FAULT  0xa /* enum */
#define MC_CMD_PCS_TX_FAULT  0xb /* enum */
#define MC_CMD_PCS_BER  0xc /* enum */
#define MC_CMD_PCS_BLOCK_ERRORS  0xd /* enum */
#define MC_CMD_PHYXS_LINK_UP  0xe /* enum */
#define MC_CMD_PHYXS_RX_FAULT  0xf /* enum */
#define MC_CMD_PHYXS_TX_FAULT  0x10 /* enum */
#define MC_CMD_PHYXS_ALIGN  0x11 /* enum */
#define MC_CMD_PHYXS_SYNC  0x12 /* enum */
#define MC_CMD_AN_LINK_UP  0x13 /* enum */
#define MC_CMD_AN_COMPLETE  0x14 /* enum */
#define MC_CMD_AN_10GBT_STATUS  0x15 /* enum */
#define MC_CMD_CL22_LINK_UP  0x16 /* enum */
#define MC_CMD_PHY_NSTATS  0x17 /* enum */


/***********************************/
/* MC_CMD_MAC_STATS
 * Get generic MAC statistics.
 */
#define MC_CMD_MAC_STATS  0x2e

/* MC_CMD_MAC_STATS_IN msgrequest */
#define MC_CMD_MAC_STATS_IN_LEN 16
#define MC_CMD_MAC_STATS_IN_DMA_ADDR_OFST 0
#define MC_CMD_MAC_STATS_IN_DMA_ADDR_LEN 8
#define MC_CMD_MAC_STATS_IN_DMA_ADDR_LO_OFST 0
#define MC_CMD_MAC_STATS_IN_DMA_ADDR_HI_OFST 4
#define MC_CMD_MAC_STATS_IN_CMD_OFST 8
#define MC_CMD_MAC_STATS_IN_DMA_LBN 0
#define MC_CMD_MAC_STATS_IN_DMA_WIDTH 1
#define MC_CMD_MAC_STATS_IN_CLEAR_LBN 1
#define MC_CMD_MAC_STATS_IN_CLEAR_WIDTH 1
#define MC_CMD_MAC_STATS_IN_PERIODIC_CHANGE_LBN 2
#define MC_CMD_MAC_STATS_IN_PERIODIC_CHANGE_WIDTH 1
#define MC_CMD_MAC_STATS_IN_PERIODIC_ENABLE_LBN 3
#define MC_CMD_MAC_STATS_IN_PERIODIC_ENABLE_WIDTH 1
#define MC_CMD_MAC_STATS_IN_PERIODIC_CLEAR_LBN 4
#define MC_CMD_MAC_STATS_IN_PERIODIC_CLEAR_WIDTH 1
#define MC_CMD_MAC_STATS_IN_PERIODIC_NOEVENT_LBN 5
#define MC_CMD_MAC_STATS_IN_PERIODIC_NOEVENT_WIDTH 1
#define MC_CMD_MAC_STATS_IN_PERIOD_MS_LBN 16
#define MC_CMD_MAC_STATS_IN_PERIOD_MS_WIDTH 16
#define MC_CMD_MAC_STATS_IN_DMA_LEN_OFST 12

/* MC_CMD_MAC_STATS_OUT_DMA msgresponse */
#define MC_CMD_MAC_STATS_OUT_DMA_LEN 0

/* MC_CMD_MAC_STATS_OUT_NO_DMA msgresponse */
#define MC_CMD_MAC_STATS_OUT_NO_DMA_LEN (((MC_CMD_MAC_NSTATS*64))>>3)
#define MC_CMD_MAC_STATS_OUT_NO_DMA_STATISTICS_OFST 0
#define MC_CMD_MAC_STATS_OUT_NO_DMA_STATISTICS_LEN 8
#define MC_CMD_MAC_STATS_OUT_NO_DMA_STATISTICS_LO_OFST 0
#define MC_CMD_MAC_STATS_OUT_NO_DMA_STATISTICS_HI_OFST 4
#define MC_CMD_MAC_STATS_OUT_NO_DMA_STATISTICS_NUM MC_CMD_MAC_NSTATS
#define MC_CMD_MAC_GENERATION_START  0x0 /* enum */
#define MC_CMD_MAC_TX_PKTS  0x1 /* enum */
#define MC_CMD_MAC_TX_PAUSE_PKTS  0x2 /* enum */
#define MC_CMD_MAC_TX_CONTROL_PKTS  0x3 /* enum */
#define MC_CMD_MAC_TX_UNICAST_PKTS  0x4 /* enum */
#define MC_CMD_MAC_TX_MULTICAST_PKTS  0x5 /* enum */
```



```
#define MC_CMD_MAC_TX_BROADCAST_PKTS  0x6 /* enum */
#define MC_CMD_MAC_TX_BYTES  0x7 /* enum */
#define MC_CMD_MAC_TX_BAD_BYTES  0x8 /* enum */
#define MC_CMD_MAC_TX_LT64_PKTS  0x9 /* enum */
#define MC_CMD_MAC_TX_64_PKTS  0xa /* enum */
#define MC_CMD_MAC_TX_65_TO_127_PKTS  0xb /* enum */
#define MC_CMD_MAC_TX_128_TO_255_PKTS  0xc /* enum */
#define MC_CMD_MAC_TX_256_TO_511_PKTS  0xd /* enum */
#define MC_CMD_MAC_TX_512_TO_1023_PKTS  0xe /* enum */
#define MC_CMD_MAC_TX_1024_TO_15XX_PKTS  0xf /* enum */
#define MC_CMD_MAC_TX_15XX_TO_JUMBO_PKTS  0x10 /* enum */
#define MC_CMD_MAC_TX_GTJUMBO_PKTS  0x11 /* enum */
#define MC_CMD_MAC_TX_BAD_FCS_PKTS  0x12 /* enum */
#define MC_CMD_MAC_TX_SINGLE_COLLISION_PKTS  0x13 /* enum */
#define MC_CMD_MAC_TX_MULTIPLE_COLLISION_PKTS  0x14 /* enum */
#define MC_CMD_MAC_TX_EXCESSIVE_COLLISION_PKTS  0x15 /* enum */
#define MC_CMD_MAC_TX_LATE_COLLISION_PKTS  0x16 /* enum */
#define MC_CMD_MAC_TX_DEFERRED_PKTS  0x17 /* enum */
#define MC_CMD_MAC_TX_EXCESSIVE_DEFERRED_PKTS  0x18 /* enum */
#define MC_CMD_MAC_TX_NON_TCPUDP_PKTS  0x19 /* enum */
#define MC_CMD_MAC_TX_MAC_SRC_ERR_PKTS  0x1a /* enum */
#define MC_CMD_MAC_TX_IP_SRC_ERR_PKTS  0x1b /* enum */
#define MC_CMD_MAC_RX_PKTS  0x1c /* enum */
#define MC_CMD_MAC_RX_PAUSE_PKTS  0x1d /* enum */
#define MC_CMD_MAC_RX_GOOD_PKTS  0x1e /* enum */
#define MC_CMD_MAC_RX_CONTROL_PKTS  0x1f /* enum */
#define MC_CMD_MAC_RX_UNICAST_PKTS  0x20 /* enum */
#define MC_CMD_MAC_RX_MULTICAST_PKTS  0x21 /* enum */
#define MC_CMD_MAC_RX_BROADCAST_PKTS  0x22 /* enum */
#define MC_CMD_MAC_RX_BYTES  0x23 /* enum */
#define MC_CMD_MAC_RX_BAD_BYTES  0x24 /* enum */
#define MC_CMD_MAC_RX_64_PKTS  0x25 /* enum */
#define MC_CMD_MAC_RX_65_TO_127_PKTS  0x26 /* enum */
#define MC_CMD_MAC_RX_128_TO_255_PKTS  0x27 /* enum */
#define MC_CMD_MAC_RX_256_TO_511_PKTS  0x28 /* enum */
#define MC_CMD_MAC_RX_512_TO_1023_PKTS  0x29 /* enum */
#define MC_CMD_MAC_RX_1024_TO_15XX_PKTS  0x2a /* enum */
#define MC_CMD_MAC_RX_15XX_TO_JUMBO_PKTS  0x2b /* enum */
#define MC_CMD_MAC_RX_GTJUMBO_PKTS  0x2c /* enum */
#define MC_CMD_MAC_RX_UNDERSIZE_PKTS  0x2d /* enum */
#define MC_CMD_MAC_RX_BAD_FCS_PKTS  0x2e /* enum */
#define MC_CMD_MAC_RX_OVERFLOW_PKTS  0x2f /* enum */
#define MC_CMD_MAC_RX_FALSE_CARRIER_PKTS  0x30 /* enum */
#define MC_CMD_MAC_RX_SYMBOL_ERROR_PKTS  0x31 /* enum */
#define MC_CMD_MAC_RX_ALIGN_ERROR_PKTS  0x32 /* enum */
#define MC_CMD_MAC_RX_LENGTH_ERROR_PKTS  0x33 /* enum */
#define MC_CMD_MAC_RX_INTERNAL_ERROR_PKTS  0x34 /* enum */
#define MC_CMD_MAC_RX_JABBER_PKTS  0x35 /* enum */
#define MC_CMD_MAC_RX_NODESC_DROPS  0x36 /* enum */
#define MC_CMD_MAC_RX_LANES01_CHAR_ERR  0x37 /* enum */
#define MC_CMD_MAC_RX_LANES23_CHAR_ERR  0x38 /* enum */
#define MC_CMD_MAC_RX_LANES01_DISP_ERR  0x39 /* enum */
#define MC_CMD_MAC_RX_LANES23_DISP_ERR  0x3a /* enum */
#define MC_CMD_MAC_RX_MATCH_FAULT  0x3b /* enum */
#define MC_CMD_GMAC_DMABUF_START  0x40 /* enum */
#define MC_CMD_GMAC_DMABUF_END    0x5f /* enum */
#define MC_CMD_MAC_GENERATION_END 0x60 /* enum */
#define MC_CMD_MAC_NSTATS  0x61 /* enum */


/***********************************/
/* MC_CMD_SRIOV
 * to be documented
 */
#define MC_CMD_SRIOV  0x30
```

```
/* MC_CMD_SRIOV_IN msgrequest */
#define MC_CMD_SRIOV_IN_LEN 12
#define MC_CMD_SRIOV_IN_ENABLE_OFST 0
#define MC_CMD_SRIOV_IN_VI_BASE_OFST 4
#define MC_CMD_SRIOV_IN_VF_COUNT_OFST 8

/* MC_CMD_SRIOV_OUT msgresponse */
#define MC_CMD_SRIOV_OUT_LEN 8
#define MC_CMD_SRIOV_OUT_VI_SCALE_OFST 0
#define MC_CMD_SRIOV_OUT_VF_TOTAL_OFST 4

/* MC_CMD_MEMCPY_RECORD_TYPEDEF structuredef */
#define MC_CMD_MEMCPY_RECORD_TYPEDEF_LEN 32
#define MC_CMD_MEMCPY_RECORD_TYPEDEF_NUM_RECORDS_OFST 0
#define MC_CMD_MEMCPY_RECORD_TYPEDEF_NUM_RECORDS_LBN 0
#define MC_CMD_MEMCPY_RECORD_TYPEDEF_NUM_RECORDS_WIDTH 32
#define MC_CMD_MEMCPY_RECORD_TYPEDEF_TO_RID_OFST 4
#define MC_CMD_MEMCPY_RECORD_TYPEDEF_TO_RID_LBN 32
#define MC_CMD_MEMCPY_RECORD_TYPEDEF_TO_RID_WIDTH 32
#define MC_CMD_MEMCPY_RECORD_TYPEDEF_TO_ADDR_OFST 8
#define MC_CMD_MEMCPY_RECORD_TYPEDEF_TO_ADDR_LEN 8
#define MC_CMD_MEMCPY_RECORD_TYPEDEF_TO_ADDR_LO_OFST 8
#define MC_CMD_MEMCPY_RECORD_TYPEDEF_TO_ADDR_HI_OFST 12
#define MC_CMD_MEMCPY_RECORD_TYPEDEF_TO_ADDR_LBN 64
#define MC_CMD_MEMCPY_RECORD_TYPEDEF_TO_ADDR_WIDTH 64
#define MC_CMD_MEMCPY_RECORD_TYPEDEF_FROM_RID_OFST 16
#define MC_CMD_MEMCPY_RECORD_TYPEDEF_RID_INLINE 0x100 /* enum */
#define MC_CMD_MEMCPY_RECORD_TYPEDEF_FROM_RID_LBN 128
#define MC_CMD_MEMCPY_RECORD_TYPEDEF_FROM_RID_WIDTH 32
#define MC_CMD_MEMCPY_RECORD_TYPEDEF_FROM_ADDR_OFST 20
#define MC_CMD_MEMCPY_RECORD_TYPEDEF_FROM_ADDR_LEN 8
#define MC_CMD_MEMCPY_RECORD_TYPEDEF_FROM_ADDR_LO_OFST 20
#define MC_CMD_MEMCPY_RECORD_TYPEDEF_FROM_ADDR_HI_OFST 24
#define MC_CMD_MEMCPY_RECORD_TYPEDEF_FROM_ADDR_LBN 160
#define MC_CMD_MEMCPY_RECORD_TYPEDEF_FROM_ADDR_WIDTH 64
#define MC_CMD_MEMCPY_RECORD_TYPEDEF_LENGTH_OFST 28
#define MC_CMD_MEMCPY_RECORD_TYPEDEF_LENGTH_LBN 224
#define MC_CMD_MEMCPY_RECORD_TYPEDEF_LENGTH_WIDTH 32


/***********************************/
/* MC_CMD_MEMCPY
 * Perform memory copy operation.
 */
#define MC_CMD_MEMCPY  0x31

/* MC_CMD_MEMCPY_IN msgrequest */
#define MC_CMD_MEMCPY_IN_LENMIN 32
#define MC_CMD_MEMCPY_IN_LENMAX 224
#define MC_CMD_MEMCPY_IN_LEN(num) (0+32*(num))
#define MC_CMD_MEMCPY_IN_RECORD_OFST 0
#define MC_CMD_MEMCPY_IN_RECORD_LEN 32
#define MC_CMD_MEMCPY_IN_RECORD_MINNUM 1
#define MC_CMD_MEMCPY_IN_RECORD_MAXNUM 7

/* MC_CMD_MEMCPY_OUT msgresponse */
#define MC_CMD_MEMCPY_OUT_LEN 0


/***********************************/
/* MC_CMD_WOL_FILTER_SET
 * Set a WoL filter.
 */
#define MC_CMD_WOL_FILTER_SET  0x32

/* MC_CMD_WOL_FILTER_SET_IN msgrequest */
```



```
#define MC_CMD_WOL_FILTER_SET_IN_LEN 192
#define MC_CMD_WOL_FILTER_SET_IN_FILTER_MODE_OFST 0
#define MC_CMD_FILTER_MODE_SIMPLE    0x0 /* enum */
#define MC_CMD_FILTER_MODE_STRUCTURED 0xffffffff /* enum */
#define MC_CMD_WOL_FILTER_SET_IN_WOL_TYPE_OFST 4
#define MC_CMD_WOL_TYPE_MAGIC      0x0 /* enum */
#define MC_CMD_WOL_TYPE_WIN_MAGIC 0x2 /* enum */
#define MC_CMD_WOL_TYPE_IPV4_SYN   0x3 /* enum */
#define MC_CMD_WOL_TYPE_IPV6_SYN   0x4 /* enum */
#define MC_CMD_WOL_TYPE_BITMAP     0x5 /* enum */
#define MC_CMD_WOL_TYPE_LINK       0x6 /* enum */
#define MC_CMD_WOL_TYPE_MAX        0x7 /* enum */
#define MC_CMD_WOL_FILTER_SET_IN_DATA_OFST 8
#define MC_CMD_WOL_FILTER_SET_IN_DATA_LEN 4
#define MC_CMD_WOL_FILTER_SET_IN_DATA_NUM 46

/* MC_CMD_WOL_FILTER_SET_IN_MAGIC msgrequest */
#define MC_CMD_WOL_FILTER_SET_IN_MAGIC_LEN 16
/*            MC_CMD_WOL_FILTER_SET_IN_FILTER_MODE_OFST 0 */
/*            MC_CMD_WOL_FILTER_SET_IN_WOL_TYPE_OFST 4 */
#define MC_CMD_WOL_FILTER_SET_IN_MAGIC_MAC_OFST 8
#define MC_CMD_WOL_FILTER_SET_IN_MAGIC_MAC_LEN 8
#define MC_CMD_WOL_FILTER_SET_IN_MAGIC_MAC_LO_OFST 8
#define MC_CMD_WOL_FILTER_SET_IN_MAGIC_MAC_HI_OFST 12

/* MC_CMD_WOL_FILTER_SET_IN_IPV4_SYN msgrequest */
#define MC_CMD_WOL_FILTER_SET_IN_IPV4_SYN_LEN 20
/*            MC_CMD_WOL_FILTER_SET_IN_FILTER_MODE_OFST 0 */
/*            MC_CMD_WOL_FILTER_SET_IN_WOL_TYPE_OFST 4 */
#define MC_CMD_WOL_FILTER_SET_IN_IPV4_SYN_SRC_IP_OFST 8
#define MC_CMD_WOL_FILTER_SET_IN_IPV4_SYN_DST_IP_OFST 12
#define MC_CMD_WOL_FILTER_SET_IN_IPV4_SYN_SRC_PORT_OFST 16
#define MC_CMD_WOL_FILTER_SET_IN_IPV4_SYN_SRC_PORT_LEN 2
#define MC_CMD_WOL_FILTER_SET_IN_IPV4_SYN_DST_PORT_OFST 18
#define MC_CMD_WOL_FILTER_SET_IN_IPV4_SYN_DST_PORT_LEN 2

/* MC_CMD_WOL_FILTER_SET_IN_IPV6_SYN msgrequest */
#define MC_CMD_WOL_FILTER_SET_IN_IPV6_SYN_LEN 44
/*            MC_CMD_WOL_FILTER_SET_IN_FILTER_MODE_OFST 0 */
/*            MC_CMD_WOL_FILTER_SET_IN_WOL_TYPE_OFST 4 */
#define MC_CMD_WOL_FILTER_SET_IN_IPV6_SYN_SRC_IP_OFST 8
#define MC_CMD_WOL_FILTER_SET_IN_IPV6_SYN_SRC_IP_LEN 16
#define MC_CMD_WOL_FILTER_SET_IN_IPV6_SYN_DST_IP_OFST 24
#define MC_CMD_WOL_FILTER_SET_IN_IPV6_SYN_DST_IP_LEN 16
#define MC_CMD_WOL_FILTER_SET_IN_IPV6_SYN_SRC_PORT_OFST 40
#define MC_CMD_WOL_FILTER_SET_IN_IPV6_SYN_SRC_PORT_LEN 2
#define MC_CMD_WOL_FILTER_SET_IN_IPV6_SYN_DST_PORT_OFST 42
#define MC_CMD_WOL_FILTER_SET_IN_IPV6_SYN_DST_PORT_LEN 2

/* MC_CMD_WOL_FILTER_SET_IN_BITMAP msgrequest */
#define MC_CMD_WOL_FILTER_SET_IN_BITMAP_LEN 187
/*            MC_CMD_WOL_FILTER_SET_IN_FILTER_MODE_OFST 0 */
/*            MC_CMD_WOL_FILTER_SET_IN_WOL_TYPE_OFST 4 */
#define MC_CMD_WOL_FILTER_SET_IN_BITMAP_MASK_OFST 8
#define MC_CMD_WOL_FILTER_SET_IN_BITMAP_MASK_LEN 48
#define MC_CMD_WOL_FILTER_SET_IN_BITMAP_BITMAP_OFST 56
#define MC_CMD_WOL_FILTER_SET_IN_BITMAP_BITMAP_LEN 128
#define MC_CMD_WOL_FILTER_SET_IN_BITMAP_LEN_OFST 184
#define MC_CMD_WOL_FILTER_SET_IN_BITMAP_LEN_LEN 1
#define MC_CMD_WOL_FILTER_SET_IN_BITMAP_LAYER3_OFST 185
#define MC_CMD_WOL_FILTER_SET_IN_BITMAP_LAYER3_LEN 1
#define MC_CMD_WOL_FILTER_SET_IN_BITMAP_LAYER4_OFST 186
#define MC_CMD_WOL_FILTER_SET_IN_BITMAP_LAYER4_LEN 1

/* MC_CMD_WOL_FILTER_SET_IN_LINK msgrequest */
#define MC_CMD_WOL_FILTER_SET_IN_LINK_LEN 12
```

```
1844 /*            MC_CMD_WOL_FILTER_SET_IN_FILTER_MODE_OFST 0 */
1845 /*            MC_CMD_WOL_FILTER_SET_IN_WOL_TYPE_OFST 4 */
1846 #define MC_CMD_WOL_FILTER_SET_IN_LINK_MASK_OFST 8
1847 #define MC_CMD_WOL_FILTER_SET_IN_LINK_UP_LBN 0
1848 #define MC_CMD_WOL_FILTER_SET_IN_LINK_UP_WIDTH 1
1849 #define MC_CMD_WOL_FILTER_SET_IN_LINK_DOWN_LBN 1
1850 #define MC_CMD_WOL_FILTER_SET_IN_LINK_DOWN_WIDTH 1

1852 /* MC_CMD_WOL_FILTER_SET_OUT msgresponse */
1853 #define MC_CMD_WOL_FILTER_SET_OUT_LEN 4
1854 #define MC_CMD_WOL_FILTER_SET_OUT_FILTER_ID_OFST 0


1857 /***********************************/
1858 /* MC_CMD_WOL_FILTER_REMOVE
1859  * Remove a WoL filter.
1860  */
1861 #define MC_CMD_WOL_FILTER_REMOVE  0x33

1863 /* MC_CMD_WOL_FILTER_REMOVE_IN msgrequest */
1864 #define MC_CMD_WOL_FILTER_REMOVE_IN_LEN 4
1865 #define MC_CMD_WOL_FILTER_REMOVE_IN_FILTER_ID_OFST 0

1867 /* MC_CMD_WOL_FILTER_REMOVE_OUT msgresponse */
1868 #define MC_CMD_WOL_FILTER_REMOVE_OUT_LEN 0


1871 /***********************************/
1872 /* MC_CMD_WOL_FILTER_RESET
1873  * Reset (i.e. remove all) WoL filters.
1874  */
1875 #define MC_CMD_WOL_FILTER_RESET  0x34

1877 /* MC_CMD_WOL_FILTER_RESET_IN msgrequest */
1878 #define MC_CMD_WOL_FILTER_RESET_IN_LEN 4
1879 #define MC_CMD_WOL_FILTER_RESET_IN_MASK_OFST 0
1880 #define MC_CMD_WOL_FILTER_RESET_IN_WAKE_FILTERS 0x1 /* enum */
1881 #define MC_CMD_WOL_FILTER_RESET_IN_LIGHTSOUT_OFFLOADS 0x2 /* enum */

1883 /* MC_CMD_WOL_FILTER_RESET_OUT msgresponse */
1884 #define MC_CMD_WOL_FILTER_RESET_OUT_LEN 0


1887 /***********************************/
1888 /* MC_CMD_SET_MCAST_HASH
1889  * Set the MCASH hash value.
1890  */
1891 #define MC_CMD_SET_MCAST_HASH  0x35

1893 /* MC_CMD_SET_MCAST_HASH_IN msgrequest */
1894 #define MC_CMD_SET_MCAST_HASH_IN_LEN 32
1895 #define MC_CMD_SET_MCAST_HASH_IN_HASH0_OFST 0
1896 #define MC_CMD_SET_MCAST_HASH_IN_HASH0_LEN 16
1897 #define MC_CMD_SET_MCAST_HASH_IN_HASH1_OFST 16
1898 #define MC_CMD_SET_MCAST_HASH_IN_HASH1_LEN 16

1900 /* MC_CMD_SET_MCAST_HASH_OUT msgresponse */
1901 #define MC_CMD_SET_MCAST_HASH_OUT_LEN 0


1904 /***********************************/
1905 /* MC_CMD_NVRAM_TYPES
1906  * Get virtual NVRAM partitions information.
1907  */
1908 #define MC_CMD_NVRAM_TYPES  0x36
```



```
1910 /* MC_CMD_NVRAM_TYPES_IN msgrequest */
1911 #define MC_CMD_NVRAM_TYPES_IN_LEN 0

1913 /* MC_CMD_NVRAM_TYPES_OUT msgresponse */
1914 #define MC_CMD_NVRAM_TYPES_OUT_LEN 4
1915 #define MC_CMD_NVRAM_TYPES_OUT_TYPES_OFST 0
1916 #define MC_CMD_NVRAM_TYPE_DISABLED_CALLISTO 0x0 /* enum */
1917 #define MC_CMD_NVRAM_TYPE_MC_FW 0x1 /* enum */
1918 #define MC_CMD_NVRAM_TYPE_MC_FW_BACKUP 0x2 /* enum */
1919 #define MC_CMD_NVRAM_TYPE_STATIC_CFG_PORT0 0x3 /* enum */
1920 #define MC_CMD_NVRAM_TYPE_STATIC_CFG_PORT1 0x4 /* enum */
1921 #define MC_CMD_NVRAM_TYPE_DYNAMIC_CFG_PORT0 0x5 /* enum */
1922 #define MC_CMD_NVRAM_TYPE_DYNAMIC_CFG_PORT1 0x6 /* enum */
1923 #define MC_CMD_NVRAM_TYPE_EXP_ROM 0x7 /* enum */
1924 #define MC_CMD_NVRAM_TYPE_EXP_ROM_CFG_PORT0 0x8 /* enum */
1925 #define MC_CMD_NVRAM_TYPE_EXP_ROM_CFG_PORT1 0x9 /* enum */
1926 #define MC_CMD_NVRAM_TYPE_PHY_PORT0 0xa /* enum */
1927 #define MC_CMD_NVRAM_TYPE_PHY_PORT1 0xb /* enum */
1928 #define MC_CMD_NVRAM_TYPE_LOG 0xc /* enum */
1929 #define MC_CMD_NVRAM_TYPE_FPGA 0xd /* enum */


1932 /***********************************/
1933 /* MC_CMD_NVRAM_INFO
1934  * Read info about a virtual NVRAM partition.
1935  */
1936 #define MC_CMD_NVRAM_INFO  0x37

1938 /* MC_CMD_NVRAM_INFO_IN msgrequest */
1939 #define MC_CMD_NVRAM_INFO_IN_LEN 4
1940 #define MC_CMD_NVRAM_INFO_IN_TYPE_OFST 0
1941 /*            Enum values, see field(s): */
1942 /*               MC_CMD_NVRAM_TYPES/MC_CMD_NVRAM_TYPES_OUT/TYPES */

1944 /* MC_CMD_NVRAM_INFO_OUT msgresponse */
1945 #define MC_CMD_NVRAM_INFO_OUT_LEN 24
1946 #define MC_CMD_NVRAM_INFO_OUT_TYPE_OFST 0
1947 /*            Enum values, see field(s): */
1948 /*               MC_CMD_NVRAM_TYPES/MC_CMD_NVRAM_TYPES_OUT/TYPES */
1949 #define MC_CMD_NVRAM_INFO_OUT_SIZE_OFST 4
1950 #define MC_CMD_NVRAM_INFO_OUT_ERASESIZE_OFST 8
1951 #define MC_CMD_NVRAM_INFO_OUT_FLAGS_OFST 12
1952 #define MC_CMD_NVRAM_INFO_OUT_PROTECTED_LBN 0
1953 #define MC_CMD_NVRAM_INFO_OUT_PROTECTED_WIDTH 1
1954 #define MC_CMD_NVRAM_INFO_OUT_PHYSDEV_OFST 16
1955 #define MC_CMD_NVRAM_INFO_OUT_PHYSADDR_OFST 20


1958 /***********************************/
1959 /* MC_CMD_NVRAM_UPDATE_START
1960  * Start a group of update operations on a virtual NVRAM partition.
1961  */
1962 #define MC_CMD_NVRAM_UPDATE_START  0x38

1964 /* MC_CMD_NVRAM_UPDATE_START_IN msgrequest */
1965 #define MC_CMD_NVRAM_UPDATE_START_IN_LEN 4
1966 #define MC_CMD_NVRAM_UPDATE_START_IN_TYPE_OFST 0
1967 /*            Enum values, see field(s): */
1968 /*               MC_CMD_NVRAM_TYPES/MC_CMD_NVRAM_TYPES_OUT/TYPES */

1970 /* MC_CMD_NVRAM_UPDATE_START_OUT msgresponse */
1971 #define MC_CMD_NVRAM_UPDATE_START_OUT_LEN 0


1974 /***********************************/
1975 /* MC_CMD_NVRAM_READ
```

```
 * Read data from a virtual NVRAM partition.
 */
#define MC_CMD_NVRAM_READ  0x39

/* MC_CMD_NVRAM_READ_IN msgrequest */
#define MC_CMD_NVRAM_READ_IN_LEN 12
#define MC_CMD_NVRAM_READ_IN_TYPE_OFST 0
/*            Enum values, see field(s): */
/*               MC_CMD_NVRAM_TYPES/MC_CMD_NVRAM_TYPES_OUT/TYPES */
#define MC_CMD_NVRAM_READ_IN_OFFSET_OFST 4
#define MC_CMD_NVRAM_READ_IN_LENGTH_OFST 8

/* MC_CMD_NVRAM_READ_OUT msgresponse */
#define MC_CMD_NVRAM_READ_OUT_LENMIN 1
#define MC_CMD_NVRAM_READ_OUT_LENMAX 252
#define MC_CMD_NVRAM_READ_OUT_LEN(num) (0+1*(num))
#define MC_CMD_NVRAM_READ_OUT_READ_BUFFER_OFST 0
#define MC_CMD_NVRAM_READ_OUT_READ_BUFFER_LEN 1
#define MC_CMD_NVRAM_READ_OUT_READ_BUFFER_MINNUM 1
#define MC_CMD_NVRAM_READ_OUT_READ_BUFFER_MAXNUM 252


/***********************************/
/* MC_CMD_NVRAM_WRITE
 * Write data to a virtual NVRAM partition.
 */
#define MC_CMD_NVRAM_WRITE  0x3a

/* MC_CMD_NVRAM_WRITE_IN msgrequest */
#define MC_CMD_NVRAM_WRITE_IN_LENMIN 13
#define MC_CMD_NVRAM_WRITE_IN_LENMAX 252
#define MC_CMD_NVRAM_WRITE_IN_LEN(num) (12+1*(num))
#define MC_CMD_NVRAM_WRITE_IN_TYPE_OFST 0
/*            Enum values, see field(s): */
/*               MC_CMD_NVRAM_TYPES/MC_CMD_NVRAM_TYPES_OUT/TYPES */
#define MC_CMD_NVRAM_WRITE_IN_OFFSET_OFST 4
#define MC_CMD_NVRAM_WRITE_IN_LENGTH_OFST 8
#define MC_CMD_NVRAM_WRITE_IN_WRITE_BUFFER_OFST 12
#define MC_CMD_NVRAM_WRITE_IN_WRITE_BUFFER_LEN 1
#define MC_CMD_NVRAM_WRITE_IN_WRITE_BUFFER_MINNUM 1
#define MC_CMD_NVRAM_WRITE_IN_WRITE_BUFFER_MAXNUM 240

/* MC_CMD_NVRAM_WRITE_OUT msgresponse */
#define MC_CMD_NVRAM_WRITE_OUT_LEN 0


/***********************************/
/* MC_CMD_NVRAM_ERASE
 * Erase sector(s) from a virtual NVRAM partition.
 */
#define MC_CMD_NVRAM_ERASE  0x3b

/* MC_CMD_NVRAM_ERASE_IN msgrequest */
#define MC_CMD_NVRAM_ERASE_IN_LEN 12
#define MC_CMD_NVRAM_ERASE_IN_TYPE_OFST 0
/*            Enum values, see field(s): */
/*               MC_CMD_NVRAM_TYPES/MC_CMD_NVRAM_TYPES_OUT/TYPES */
#define MC_CMD_NVRAM_ERASE_IN_OFFSET_OFST 4
#define MC_CMD_NVRAM_ERASE_IN_LENGTH_OFST 8

/* MC_CMD_NVRAM_ERASE_OUT msgresponse */
#define MC_CMD_NVRAM_ERASE_OUT_LEN 0


/***********************************/
/* MC_CMD_NVRAM_UPDATE_FINISH
```



```
 * Finish a group of update operations on a virtual NVRAM partition.
 */
#define MC_CMD_NVRAM_UPDATE_FINISH  0x3c

/* MC_CMD_NVRAM_UPDATE_FINISH_IN msgrequest */
#define MC_CMD_NVRAM_UPDATE_FINISH_IN_LEN 8
#define MC_CMD_NVRAM_UPDATE_FINISH_IN_TYPE_OFST 0
/*            Enum values, see field(s): */
/*               MC_CMD_NVRAM_TYPES/MC_CMD_NVRAM_TYPES_OUT/TYPES */
#define MC_CMD_NVRAM_UPDATE_FINISH_IN_REBOOT_OFST 4

/* MC_CMD_NVRAM_UPDATE_FINISH_OUT msgresponse */
#define MC_CMD_NVRAM_UPDATE_FINISH_OUT_LEN 0


/***********************************/
/* MC_CMD_REBOOT
 * Reboot the MC.
 */
#define MC_CMD_REBOOT  0x3d

/* MC_CMD_REBOOT_IN msgrequest */
#define MC_CMD_REBOOT_IN_LEN 4
#define MC_CMD_REBOOT_IN_FLAGS_OFST 0
#define MC_CMD_REBOOT_FLAGS_AFTER_ASSERTION 0x1 /* enum */

/* MC_CMD_REBOOT_OUT msgresponse */
#define MC_CMD_REBOOT_OUT_LEN 0


/***********************************/
/* MC_CMD_SCHEDINFO
 * Request scheduler info.
 */
#define MC_CMD_SCHEDINFO  0x3e

/* MC_CMD_SCHEDINFO_IN msgrequest */
#define MC_CMD_SCHEDINFO_IN_LEN 0

/* MC_CMD_SCHEDINFO_OUT msgresponse */
#define MC_CMD_SCHEDINFO_OUT_LENMIN 4
#define MC_CMD_SCHEDINFO_OUT_LENMAX 252
#define MC_CMD_SCHEDINFO_OUT_LEN(num) (0+4*(num))
#define MC_CMD_SCHEDINFO_OUT_DATA_OFST 0
#define MC_CMD_SCHEDINFO_OUT_DATA_LEN 4
#define MC_CMD_SCHEDINFO_OUT_DATA_MINNUM 1
#define MC_CMD_SCHEDINFO_OUT_DATA_MAXNUM 63


/***********************************/
/* MC_CMD_REBOOT_MODE
 */
#define MC_CMD_REBOOT_MODE  0x3f

/* MC_CMD_REBOOT_MODE_IN msgrequest */
#define MC_CMD_REBOOT_MODE_IN_LEN 4
#define MC_CMD_REBOOT_MODE_IN_VALUE_OFST 0
#define MC_CMD_REBOOT_MODE_NORMAL 0x0 /* enum */
#define MC_CMD_REBOOT_MODE_SNAPPER 0x3 /* enum */

/* MC_CMD_REBOOT_MODE_OUT msgresponse */
#define MC_CMD_REBOOT_MODE_OUT_LEN 4
#define MC_CMD_REBOOT_MODE_OUT_VALUE_OFST 0


/***********************************/
```

```
2108 /* MC_CMD_SENSOR_INFO
2109  * Returns information about every available sensor.
2110  */
2111 #define MC_CMD_SENSOR_INFO  0x41

2113 /* MC_CMD_SENSOR_INFO_IN msgrequest */
2114 #define MC_CMD_SENSOR_INFO_IN_LEN 0

2116 /* MC_CMD_SENSOR_INFO_OUT msgresponse */
2117 #define MC_CMD_SENSOR_INFO_OUT_LENMIN 12
2118 #define MC_CMD_SENSOR_INFO_OUT_LENMAX 252
2119 #define MC_CMD_SENSOR_INFO_OUT_LEN(num) (4+8*(num))
2120 #define MC_CMD_SENSOR_INFO_OUT_MASK_OFST 0
2121 #define MC_CMD_SENSOR_CONTROLLER_TEMP  0x0 /* enum */
2122 #define MC_CMD_SENSOR_PHY_COMMON_TEMP  0x1 /* enum */
2123 #define MC_CMD_SENSOR_CONTROLLER_COOLING  0x2 /* enum */
2124 #define MC_CMD_SENSOR_PHY0_TEMP  0x3 /* enum */
2125 #define MC_CMD_SENSOR_PHY0_COOLING  0x4 /* enum */
2126 #define MC_CMD_SENSOR_PHY1_TEMP  0x5 /* enum */
2127 #define MC_CMD_SENSOR_PHY1_COOLING  0x6 /* enum */
2128 #define MC_CMD_SENSOR_IN_1V0  0x7 /* enum */
2129 #define MC_CMD_SENSOR_IN_1V2  0x8 /* enum */
2130 #define MC_CMD_SENSOR_IN_1V8  0x9 /* enum */
2131 #define MC_CMD_SENSOR_IN_2V5  0xa /* enum */
2132 #define MC_CMD_SENSOR_IN_3V3  0xb /* enum */
2133 #define MC_CMD_SENSOR_IN_12V0  0xc /* enum */
2134 #define MC_CMD_SENSOR_IN_1V2A  0xd /* enum */
2135 #define MC_CMD_SENSOR_IN_VREF  0xe /* enum */
2136 #define MC_CMD_SENSOR_ENTRY_OFST 4
2137 #define MC_CMD_SENSOR_ENTRY_LEN 8
2138 #define MC_CMD_SENSOR_ENTRY_LO_OFST 4
2139 #define MC_CMD_SENSOR_ENTRY_HI_OFST 8
2140 #define MC_CMD_SENSOR_ENTRY_MINNUM 1
2141 #define MC_CMD_SENSOR_ENTRY_MAXNUM 31

2143 /* MC_CMD_SENSOR_INFO_ENTRY_TYPEDEF structuredef */
2144 #define MC_CMD_SENSOR_INFO_ENTRY_TYPEDEF_LEN 8
2145 #define MC_CMD_SENSOR_INFO_ENTRY_TYPEDEF_MIN1_OFST 0
2146 #define MC_CMD_SENSOR_INFO_ENTRY_TYPEDEF_MIN1_LEN 2
2147 #define MC_CMD_SENSOR_INFO_ENTRY_TYPEDEF_MIN1_LBN 0
2148 #define MC_CMD_SENSOR_INFO_ENTRY_TYPEDEF_MIN1_WIDTH 16
2149 #define MC_CMD_SENSOR_INFO_ENTRY_TYPEDEF_MAX1_OFST 2
2150 #define MC_CMD_SENSOR_INFO_ENTRY_TYPEDEF_MAX1_LEN 2
2151 #define MC_CMD_SENSOR_INFO_ENTRY_TYPEDEF_MAX1_LBN 16
2152 #define MC_CMD_SENSOR_INFO_ENTRY_TYPEDEF_MAX1_WIDTH 16
2153 #define MC_CMD_SENSOR_INFO_ENTRY_TYPEDEF_MIN2_OFST 4
2154 #define MC_CMD_SENSOR_INFO_ENTRY_TYPEDEF_MIN2_LEN 2
2155 #define MC_CMD_SENSOR_INFO_ENTRY_TYPEDEF_MIN2_LBN 32
2156 #define MC_CMD_SENSOR_INFO_ENTRY_TYPEDEF_MIN2_WIDTH 16
2157 #define MC_CMD_SENSOR_INFO_ENTRY_TYPEDEF_MAX2_OFST 6
2158 #define MC_CMD_SENSOR_INFO_ENTRY_TYPEDEF_MAX2_LEN 2
2159 #define MC_CMD_SENSOR_INFO_ENTRY_TYPEDEF_MAX2_LBN 48
2160 #define MC_CMD_SENSOR_INFO_ENTRY_TYPEDEF_MAX2_WIDTH 16


2163 /***********************************/
2164 /* MC_CMD_READ_SENSORS
2165  * Returns the current reading from each sensor.
2166  */
2167 #define MC_CMD_READ_SENSORS  0x42

2169 /* MC_CMD_READ_SENSORS_IN msgrequest */
2170 #define MC_CMD_READ_SENSORS_IN_LEN 8
2171 #define MC_CMD_READ_SENSORS_IN_DMA_ADDR_OFST 0
2172 #define MC_CMD_READ_SENSORS_IN_DMA_ADDR_LEN 8
2173 #define MC_CMD_READ_SENSORS_IN_DMA_ADDR_LO_OFST 0
```



```
2174 #define MC_CMD_READ_SENSORS_IN_DMA_ADDR_HI_OFST 4

2176 /* MC_CMD_READ_SENSORS_OUT msgresponse */
2177 #define MC_CMD_READ_SENSORS_OUT_LEN 0

2179 /* MC_CMD_SENSOR_VALUE_ENTRY_TYPEDEF structuredef */
2180 #define MC_CMD_SENSOR_VALUE_ENTRY_TYPEDEF_LEN 3
2181 #define MC_CMD_SENSOR_VALUE_ENTRY_TYPEDEF_VALUE_OFST 0
2182 #define MC_CMD_SENSOR_VALUE_ENTRY_TYPEDEF_VALUE_LEN 2
2183 #define MC_CMD_SENSOR_VALUE_ENTRY_TYPEDEF_VALUE_LBN 0
2184 #define MC_CMD_SENSOR_VALUE_ENTRY_TYPEDEF_VALUE_WIDTH 16
2185 #define MC_CMD_SENSOR_VALUE_ENTRY_TYPEDEF_STATE_OFST 2
2186 #define MC_CMD_SENSOR_VALUE_ENTRY_TYPEDEF_STATE_LEN 1
2187 #define MC_CMD_SENSOR_STATE_OK  0x0 /* enum */
2188 #define MC_CMD_SENSOR_STATE_WARNING  0x1 /* enum */
2189 #define MC_CMD_SENSOR_STATE_FATAL  0x2 /* enum */
2190 #define MC_CMD_SENSOR_STATE_BROKEN  0x3 /* enum */
2191 #define MC_CMD_SENSOR_VALUE_ENTRY_TYPEDEF_STATE_LBN 16
2192 #define MC_CMD_SENSOR_VALUE_ENTRY_TYPEDEF_STATE_WIDTH 8


2195 /***********************************/
2196 /* MC_CMD_GET_PHY_STATE
2197  * Report current state of PHY.
2198  */
2199 #define MC_CMD_GET_PHY_STATE  0x43

2201 /* MC_CMD_GET_PHY_STATE_IN msgrequest */
2202 #define MC_CMD_GET_PHY_STATE_IN_LEN 0

2204 /* MC_CMD_GET_PHY_STATE_OUT msgresponse */
2205 #define MC_CMD_GET_PHY_STATE_OUT_LEN 4
2206 #define MC_CMD_GET_PHY_STATE_OUT_STATE_OFST 0
2207 #define MC_CMD_PHY_STATE_OK 0x1 /* enum */
2208 #define MC_CMD_PHY_STATE_ZOMBIE 0x2 /* enum */


2211 /***********************************/
2212 /* MC_CMD_SETUP_8021QBB
2213  * 802.1Qbb control.
2214  */
2215 #define MC_CMD_SETUP_8021QBB  0x44

2217 /* MC_CMD_SETUP_8021QBB_IN msgrequest */
2218 #define MC_CMD_SETUP_8021QBB_IN_LEN 32
2219 #define MC_CMD_SETUP_8021QBB_IN_TXQS_OFST 0
2220 #define MC_CMD_SETUP_8021QBB_IN_TXQS_LEN 32

2222 /* MC_CMD_SETUP_8021QBB_OUT msgresponse */
2223 #define MC_CMD_SETUP_8021QBB_OUT_LEN 0


2226 /***********************************/
2227 /* MC_CMD_WOL_FILTER_GET
2228  * Retrieve ID of any WoL filters.
2229  */
2230 #define MC_CMD_WOL_FILTER_GET  0x45

2232 /* MC_CMD_WOL_FILTER_GET_IN msgrequest */
2233 #define MC_CMD_WOL_FILTER_GET_IN_LEN 0

2235 /* MC_CMD_WOL_FILTER_GET_OUT msgresponse */
2236 #define MC_CMD_WOL_FILTER_GET_OUT_LEN 4
2237 #define MC_CMD_WOL_FILTER_GET_OUT_FILTER_ID_OFST 0
```

```
2240 /***********************************/
2241 /* MC_CMD_ADD_LIGHTSOUT_OFFLOAD
2242  * Add a protocol offload to NIC for lights-out state.
2243  */
2244 #define MC_CMD_ADD_LIGHTSOUT_OFFLOAD  0x46

2246 /* MC_CMD_ADD_LIGHTSOUT_OFFLOAD_IN msgrequest */
2247 #define MC_CMD_ADD_LIGHTSOUT_OFFLOAD_IN_LENMIN 8
2248 #define MC_CMD_ADD_LIGHTSOUT_OFFLOAD_IN_LENMAX 252
2249 #define MC_CMD_ADD_LIGHTSOUT_OFFLOAD_IN_LEN(num) (4+4*(num))
2250 #define MC_CMD_ADD_LIGHTSOUT_OFFLOAD_IN_PROTOCOL_OFST 0
2251 #define MC_CMD_LIGHTSOUT_OFFLOAD_PROTOCOL_ARP 0x1 /* enum */
2252 #define MC_CMD_LIGHTSOUT_OFFLOAD_PROTOCOL_NS  0x2 /* enum */
2253 #define MC_CMD_ADD_LIGHTSOUT_OFFLOAD_IN_DATA_OFST 4
2254 #define MC_CMD_ADD_LIGHTSOUT_OFFLOAD_IN_DATA_LEN 4
2255 #define MC_CMD_ADD_LIGHTSOUT_OFFLOAD_IN_DATA_MINNUM 1
2256 #define MC_CMD_ADD_LIGHTSOUT_OFFLOAD_IN_DATA_MAXNUM 62

2258 /* MC_CMD_ADD_LIGHTSOUT_OFFLOAD_IN_ARP msgrequest */
2259 #define MC_CMD_ADD_LIGHTSOUT_OFFLOAD_IN_ARP_LEN 14
2260 /*            MC_CMD_ADD_LIGHTSOUT_OFFLOAD_IN_PROTOCOL_OFST 0 */
2261 #define MC_CMD_ADD_LIGHTSOUT_OFFLOAD_IN_ARP_MAC_OFST 4
2262 #define MC_CMD_ADD_LIGHTSOUT_OFFLOAD_IN_ARP_MAC_LEN 6
2263 #define MC_CMD_ADD_LIGHTSOUT_OFFLOAD_IN_ARP_IP_OFST 10

2265 /* MC_CMD_ADD_LIGHTSOUT_OFFLOAD_IN_NS msgrequest */
2266 #define MC_CMD_ADD_LIGHTSOUT_OFFLOAD_IN_NS_LEN 42
2267 /*            MC_CMD_ADD_LIGHTSOUT_OFFLOAD_IN_PROTOCOL_OFST 0 */
2268 #define MC_CMD_ADD_LIGHTSOUT_OFFLOAD_IN_NS_MAC_OFST 4
2269 #define MC_CMD_ADD_LIGHTSOUT_OFFLOAD_IN_NS_MAC_LEN 6
2270 #define MC_CMD_ADD_LIGHTSOUT_OFFLOAD_IN_NS_SNIPV6_OFST 10
2271 #define MC_CMD_ADD_LIGHTSOUT_OFFLOAD_IN_NS_SNIPV6_LEN 16
2272 #define MC_CMD_ADD_LIGHTSOUT_OFFLOAD_IN_NS_IPV6_OFST 26
2273 #define MC_CMD_ADD_LIGHTSOUT_OFFLOAD_IN_NS_IPV6_LEN 16

2275 /* MC_CMD_ADD_LIGHTSOUT_OFFLOAD_OUT msgresponse */
2276 #define MC_CMD_ADD_LIGHTSOUT_OFFLOAD_OUT_LEN 4
2277 #define MC_CMD_ADD_LIGHTSOUT_OFFLOAD_OUT_FILTER_ID_OFST 0


2280 /***********************************/
2281 /* MC_CMD_REMOVE_LIGHTSOUT_OFFLOAD
2282  * Remove a protocol offload from NIC for lights-out state.
2283  */
2284 #define MC_CMD_REMOVE_LIGHTSOUT_OFFLOAD  0x47

2286 /* MC_CMD_REMOVE_LIGHTSOUT_OFFLOAD_IN msgrequest */
2287 #define MC_CMD_REMOVE_LIGHTSOUT_OFFLOAD_IN_LEN 8
2288 #define MC_CMD_REMOVE_LIGHTSOUT_OFFLOAD_IN_PROTOCOL_OFST 0
2289 #define MC_CMD_REMOVE_LIGHTSOUT_OFFLOAD_IN_FILTER_ID_OFST 4

2291 /* MC_CMD_REMOVE_LIGHTSOUT_OFFLOAD_OUT msgresponse */
2292 #define MC_CMD_REMOVE_LIGHTSOUT_OFFLOAD_OUT_LEN 0


2295 /***********************************/
2296 /* MC_CMD_MAC_RESET_RESTORE
2297  * Restore MAC after block reset.
2298  */
2299 #define MC_CMD_MAC_RESET_RESTORE  0x48

2301 /* MC_CMD_MAC_RESET_RESTORE_IN msgrequest */
2302 #define MC_CMD_MAC_RESET_RESTORE_IN_LEN 0

2304 /* MC_CMD_MAC_RESET_RESTORE_OUT msgresponse */
2305 #define MC_CMD_MAC_RESET_RESTORE_OUT_LEN 0
```



```
2308 /***********************************/
2309 /* MC_CMD_TESTASSERT
2310  */
2311 #define MC_CMD_TESTASSERT   0x49

2313 /* MC_CMD_TESTASSERT_IN msgrequest */
2314 #define MC_CMD_TESTASSERT_IN_LEN 0

2316 /* MC_CMD_TESTASSERT_OUT msgresponse */
2317 #define MC_CMD_TESTASSERT_OUT_LEN 0


2320 /***********************************/
2321 /* MC_CMD_WORKAROUND
2322  * Enable/Disable a given workaround.
2323  */
2324 #define MC_CMD_WORKAROUND  0x4a

2326 /* MC_CMD_WORKAROUND_IN msgrequest */
2327 #define MC_CMD_WORKAROUND_IN_LEN 8
2328 #define MC_CMD_WORKAROUND_IN_TYPE_OFST 0
2329 #define MC_CMD_WORKAROUND_BUG17230 0x1 /* enum */
2330 #define MC_CMD_WORKAROUND_IN_ENABLED_OFST 4

2332 /* MC_CMD_WORKAROUND_OUT msgresponse */
2333 #define MC_CMD_WORKAROUND_OUT_LEN 0


2336 /***********************************/
2337 /* MC_CMD_GET_PHY_MEDIA_INFO
2338  * Read media-specific data from PHY.
2339  */
2340 #define MC_CMD_GET_PHY_MEDIA_INFO  0x4b

2342 /* MC_CMD_GET_PHY_MEDIA_INFO_IN msgrequest */
2343 #define MC_CMD_GET_PHY_MEDIA_INFO_IN_LEN 4
2344 #define MC_CMD_GET_PHY_MEDIA_INFO_IN_PAGE_OFST 0

2346 /* MC_CMD_GET_PHY_MEDIA_INFO_OUT msgresponse */
2347 #define MC_CMD_GET_PHY_MEDIA_INFO_OUT_LENMIN 5
2348 #define MC_CMD_GET_PHY_MEDIA_INFO_OUT_LENMAX 252
2349 #define MC_CMD_GET_PHY_MEDIA_INFO_OUT_LEN(num) (4+1*(num))
2350 #define MC_CMD_GET_PHY_MEDIA_INFO_OUT_DATALEN_OFST 0
2351 #define MC_CMD_GET_PHY_MEDIA_INFO_OUT_DATA_OFST 4
2352 #define MC_CMD_GET_PHY_MEDIA_INFO_OUT_DATA_LEN 1
2353 #define MC_CMD_GET_PHY_MEDIA_INFO_OUT_DATA_MINNUM 1
2354 #define MC_CMD_GET_PHY_MEDIA_INFO_OUT_DATA_MAXNUM 248


2357 /***********************************/
2358 /* MC_CMD_NVRAM_TEST
2359  * Test a particular NVRAM partition.
2360  */
2361 #define MC_CMD_NVRAM_TEST  0x4c

2363 /* MC_CMD_NVRAM_TEST_IN msgrequest */
2364 #define MC_CMD_NVRAM_TEST_IN_LEN 4
2365 #define MC_CMD_NVRAM_TEST_IN_TYPE_OFST 0
2366 /*            Enum values, see field(s): */
2367 /*               MC_CMD_NVRAM_TYPES/MC_CMD_NVRAM_TYPES_OUT/TYPES */

2369 /* MC_CMD_NVRAM_TEST_OUT msgresponse */
2370 #define MC_CMD_NVRAM_TEST_OUT_LEN 4
2371 #define MC_CMD_NVRAM_TEST_OUT_RESULT_OFST 0
```

```
#define MC_CMD_NVRAM_TEST_PASS 0x0 /* enum */
#define MC_CMD_NVRAM_TEST_FAIL 0x1 /* enum */
#define MC_CMD_NVRAM_TEST_NOTSUPP 0x2 /* enum */


/***********************************/
/* MC_CMD_MRSFP_TWEAK
 * Read status and/or set parameters for the 'mrsfp' driver.
 */
#define MC_CMD_MRSFP_TWEAK  0x4d

/* MC_CMD_MRSFP_TWEAK_IN_EQ_CONFIG msgrequest */
#define MC_CMD_MRSFP_TWEAK_IN_EQ_CONFIG_LEN 16
#define MC_CMD_MRSFP_TWEAK_IN_EQ_CONFIG_TXEQ_LEVEL_OFST 0
#define MC_CMD_MRSFP_TWEAK_IN_EQ_CONFIG_TXEQ_DT_CFG_OFST 4
#define MC_CMD_MRSFP_TWEAK_IN_EQ_CONFIG_RXEQ_BOOST_OFST 8
#define MC_CMD_MRSFP_TWEAK_IN_EQ_CONFIG_RXEQ_DT_CFG_OFST 12

/* MC_CMD_MRSFP_TWEAK_IN_READ_ONLY msgrequest */
#define MC_CMD_MRSFP_TWEAK_IN_READ_ONLY_LEN 0

/* MC_CMD_MRSFP_TWEAK_OUT msgresponse */
#define MC_CMD_MRSFP_TWEAK_OUT_LEN 12
#define MC_CMD_MRSFP_TWEAK_OUT_IOEXP_INPUTS_OFST 0
#define MC_CMD_MRSFP_TWEAK_OUT_IOEXP_OUTPUTS_OFST 4
#define MC_CMD_MRSFP_TWEAK_OUT_IOEXP_DIRECTION_OFST 8
#define MC_CMD_MRSFP_TWEAK_OUT_IOEXP_DIRECTION_OUT 0x0 /* enum */
#define MC_CMD_MRSFP_TWEAK_OUT_IOEXP_DIRECTION_IN 0x1 /* enum */


/***********************************/
/* MC_CMD_SENSOR_SET_LIMS
 * Adjusts the sensor limits.
 */
#define MC_CMD_SENSOR_SET_LIMS  0x4e

/* MC_CMD_SENSOR_SET_LIMS_IN msgrequest */
#define MC_CMD_SENSOR_SET_LIMS_IN_LEN 20
#define MC_CMD_SENSOR_SET_LIMS_IN_SENSOR_OFST 0
/*            Enum values, see field(s): */
/*               MC_CMD_SENSOR_INFO/MC_CMD_SENSOR_INFO_OUT/MASK */
#define MC_CMD_SENSOR_SET_LIMS_IN_LOW0_OFST 4
#define MC_CMD_SENSOR_SET_LIMS_IN_HI0_OFST 8
#define MC_CMD_SENSOR_SET_LIMS_IN_LOW1_OFST 12
#define MC_CMD_SENSOR_SET_LIMS_IN_HI1_OFST 16

/* MC_CMD_SENSOR_SET_LIMS_OUT msgresponse */
#define MC_CMD_SENSOR_SET_LIMS_OUT_LEN 0


/***********************************/
/* MC_CMD_GET_RESOURCE_LIMITS
 */
#define MC_CMD_GET_RESOURCE_LIMITS  0x4f

/* MC_CMD_GET_RESOURCE_LIMITS_IN msgrequest */
#define MC_CMD_GET_RESOURCE_LIMITS_IN_LEN 0

/* MC_CMD_GET_RESOURCE_LIMITS_OUT msgresponse */
#define MC_CMD_GET_RESOURCE_LIMITS_OUT_LEN 16
#define MC_CMD_GET_RESOURCE_LIMITS_OUT_BUFTBL_OFST 0
#define MC_CMD_GET_RESOURCE_LIMITS_OUT_EVQ_OFST 4
#define MC_CMD_GET_RESOURCE_LIMITS_OUT_RXQ_OFST 8
#define MC_CMD_GET_RESOURCE_LIMITS_OUT_TXQ_OFST 12

/* MC_CMD_RESOURCE_SPECIFIER enum */
```



```
#define MC_CMD_RESOURCE_INSTANCE_ANY 0xffffffff /* enum */
#define MC_CMD_RESOURCE_INSTANCE_NONE 0xfffffffe /* enum */


/***********************************/
/* MC_CMD_INIT_EVQ
 */
#define MC_CMD_INIT_EVQ  0x50

/* MC_CMD_INIT_EVQ_IN msgrequest */
#define MC_CMD_INIT_EVQ_IN_LENMIN 36
#define MC_CMD_INIT_EVQ_IN_LENMAX 540
#define MC_CMD_INIT_EVQ_IN_LEN(num) (28+8*(num))
#define MC_CMD_INIT_EVQ_IN_SIZE_OFST 0
#define MC_CMD_INIT_EVQ_IN_INSTANCE_OFST 4
#define MC_CMD_INIT_EVQ_IN_TMR_LOAD_OFST 8
#define MC_CMD_INIT_EVQ_IN_TMR_RELOAD_OFST 12
#define MC_CMD_INIT_EVQ_IN_FLAGS_OFST 16
#define MC_CMD_INIT_EVQ_IN_FLAG_INTERRUPTING_LBN 0
#define MC_CMD_INIT_EVQ_IN_FLAG_INTERRUPTING_WIDTH 1
#define MC_CMD_INIT_EVQ_IN_FLAG_RPTR_DOS_LBN 1
#define MC_CMD_INIT_EVQ_IN_FLAG_RPTR_DOS_WIDTH 1
#define MC_CMD_INIT_EVQ_IN_TMR_MODE_OFST 20
#define MC_CMD_INIT_EVQ_IN_TMR_MODE_DIS 0x0 /* enum */
#define MC_CMD_INIT_EVQ_IN_TMR_IMMED_START 0x1 /* enum */
#define MC_CMD_INIT_EVQ_IN_TMR_TRIG_START 0x2 /* enum */
#define MC_CMD_INIT_EVQ_IN_TMR_INT_HLDOFF 0x3 /* enum */
#define MC_CMD_INIT_EVQ_IN_TARGET_EVQ_OFST 24
#define MC_CMD_INIT_EVQ_IN_IRQ_NUM_OFST 24
#define MC_CMD_INIT_EVQ_IN_DMA_ADDR_OFST 28
#define MC_CMD_INIT_EVQ_IN_DMA_ADDR_LEN 8
#define MC_CMD_INIT_EVQ_IN_DMA_ADDR_LO_OFST 28
#define MC_CMD_INIT_EVQ_IN_DMA_ADDR_HI_OFST 32
#define MC_CMD_INIT_EVQ_IN_DMA_ADDR_MINNUM 1
#define MC_CMD_INIT_EVQ_IN_DMA_ADDR_MAXNUM 64

/* MC_CMD_INIT_EVQ_OUT msgresponse */
#define MC_CMD_INIT_EVQ_OUT_LEN 4
#define MC_CMD_INIT_EVQ_OUT_IRQ_OFST 0

/* QUEUE_CRC_MODE structuredef */
#define QUEUE_CRC_MODE_LEN 1
#define QUEUE_CRC_MODE_MODE_LBN 0
#define QUEUE_CRC_MODE_MODE_WIDTH 4
#define QUEUE_CRC_MODE_NONE  0x0 /* enum */
#define QUEUE_CRC_MODE_FCOE  0x1 /* enum */
#define QUEUE_CRC_MODE_ISCSI_HDR  0x2 /* enum */
#define QUEUE_CRC_MODE_ISCSI  0x3 /* enum */
#define QUEUE_CRC_MODE_FCOIPOE  0x4 /* enum */
#define QUEUE_CRC_MODE_MPA  0x5 /* enum */
#define QUEUE_CRC_MODE_SPARE_LBN 4
#define QUEUE_CRC_MODE_SPARE_WIDTH 4


/***********************************/
/* MC_CMD_INIT_RXQ
 */
#define MC_CMD_INIT_RXQ  0x51

/* MC_CMD_INIT_RXQ_IN msgrequest */
#define MC_CMD_INIT_RXQ_IN_LENMIN 32
#define MC_CMD_INIT_RXQ_IN_LENMAX 248
#define MC_CMD_INIT_RXQ_IN_LEN(num) (24+8*(num))
#define MC_CMD_INIT_RXQ_IN_SIZE_OFST 0
#define MC_CMD_INIT_RXQ_IN_TARGET_EVQ_OFST 4
#define MC_CMD_INIT_RXQ_IN_LABEL_OFST 8
```

```
#define MC_CMD_INIT_RXQ_IN_INSTANCE_OFST 12
#define MC_CMD_INIT_RXQ_IN_FLAGS_OFST 16
#define MC_CMD_INIT_RXQ_IN_FLAG_BUFF_MODE_LBN 0
#define MC_CMD_INIT_RXQ_IN_FLAG_BUFF_MODE_WIDTH 1
#define MC_CMD_INIT_RXQ_IN_FLAG_HDR_SPLIT_LBN 1
#define MC_CMD_INIT_RXQ_IN_FLAG_HDR_SPLIT_WIDTH 1
#define MC_CMD_INIT_RXQ_IN_FLAG_PKT_EDIT_LBN 2
#define MC_CMD_INIT_RXQ_IN_FLAG_PKT_EDIT_WIDTH 1
#define MC_CMD_INIT_RXQ_IN_CRC_MODE_LBN 3
#define MC_CMD_INIT_RXQ_IN_CRC_MODE_WIDTH 4
#define MC_CMD_INIT_RXQ_IN_OWNER_ID_OFST 20
#define MC_CMD_INIT_RXQ_IN_DMA_ADDR_OFST 24
#define MC_CMD_INIT_RXQ_IN_DMA_ADDR_LEN 8
#define MC_CMD_INIT_RXQ_IN_DMA_ADDR_LO_OFST 24
#define MC_CMD_INIT_RXQ_IN_DMA_ADDR_HI_OFST 28
#define MC_CMD_INIT_RXQ_IN_DMA_ADDR_MINNUM 1
#define MC_CMD_INIT_RXQ_IN_DMA_ADDR_MAXNUM 28

/* MC_CMD_INIT_RXQ_OUT msgresponse */
#define MC_CMD_INIT_RXQ_OUT_LEN 0


/***********************************/
/* MC_CMD_INIT_TXQ
 */
#define MC_CMD_INIT_TXQ  0x52

/* MC_CMD_INIT_TXQ_IN msgrequest */
#define MC_CMD_INIT_TXQ_IN_LENMIN 32
#define MC_CMD_INIT_TXQ_IN_LENMAX 248
#define MC_CMD_INIT_TXQ_IN_LEN(num) (24+8*(num))
#define MC_CMD_INIT_TXQ_IN_SIZE_OFST 0
#define MC_CMD_INIT_TXQ_IN_TARGET_EVQ_OFST 4
#define MC_CMD_INIT_TXQ_IN_LABEL_OFST 8
#define MC_CMD_INIT_TXQ_IN_INSTANCE_OFST 12
#define MC_CMD_INIT_TXQ_IN_FLAGS_OFST 16
#define MC_CMD_INIT_TXQ_IN_FLAG_BUFF_MODE_LBN 0
#define MC_CMD_INIT_TXQ_IN_FLAG_BUFF_MODE_WIDTH 1
#define MC_CMD_INIT_TXQ_IN_FLAG_IP_CSUM_DIS_LBN 1
#define MC_CMD_INIT_TXQ_IN_FLAG_IP_CSUM_DIS_WIDTH 1
#define MC_CMD_INIT_TXQ_IN_FLAG_TCP_CSUM_DIS_LBN 2
#define MC_CMD_INIT_TXQ_IN_FLAG_TCP_CSUM_DIS_WIDTH 1
#define MC_CMD_INIT_TXQ_IN_CRC_MODE_LBN 4
#define MC_CMD_INIT_TXQ_IN_CRC_MODE_WIDTH 4
#define MC_CMD_INIT_TXQ_IN_OWNER_ID_OFST 20
#define MC_CMD_INIT_TXQ_IN_DMA_ADDR_OFST 24
#define MC_CMD_INIT_TXQ_IN_DMA_ADDR_LEN 8
#define MC_CMD_INIT_TXQ_IN_DMA_ADDR_LO_OFST 24
#define MC_CMD_INIT_TXQ_IN_DMA_ADDR_HI_OFST 28
#define MC_CMD_INIT_TXQ_IN_DMA_ADDR_MINNUM 1
#define MC_CMD_INIT_TXQ_IN_DMA_ADDR_MAXNUM 28

/* MC_CMD_INIT_TXQ_OUT msgresponse */
#define MC_CMD_INIT_TXQ_OUT_LEN 0


/***********************************/
/* MC_CMD_FINI_EVQ
 */
#define MC_CMD_FINI_EVQ  0x55

/* MC_CMD_FINI_EVQ_IN msgrequest */
#define MC_CMD_FINI_EVQ_IN_LEN 4
#define MC_CMD_FINI_EVQ_IN_INSTANCE_OFST 0

/* MC_CMD_FINI_EVQ_OUT msgresponse */
```



```
#define MC_CMD_FINI_EVQ_OUT_LEN 0


/***********************************/
/* MC_CMD_FINI_RXQ
 */
#define MC_CMD_FINI_RXQ  0x56

/* MC_CMD_FINI_RXQ_IN msgrequest */
#define MC_CMD_FINI_RXQ_IN_LEN 4
#define MC_CMD_FINI_RXQ_IN_INSTANCE_OFST 0

/* MC_CMD_FINI_RXQ_OUT msgresponse */
#define MC_CMD_FINI_RXQ_OUT_LEN 0


/***********************************/
/* MC_CMD_FINI_TXQ
 */
#define MC_CMD_FINI_TXQ  0x57

/* MC_CMD_FINI_TXQ_IN msgrequest */
#define MC_CMD_FINI_TXQ_IN_LEN 4
#define MC_CMD_FINI_TXQ_IN_INSTANCE_OFST 0

/* MC_CMD_FINI_TXQ_OUT msgresponse */
#define MC_CMD_FINI_TXQ_OUT_LEN 0


/***********************************/
/* MC_CMD_DRIVER_EVENT
 */
#define MC_CMD_DRIVER_EVENT  0x5a

/* MC_CMD_DRIVER_EVENT_IN msgrequest */
#define MC_CMD_DRIVER_EVENT_IN_LEN 12
#define MC_CMD_DRIVER_EVENT_IN_EVQ_OFST 0
#define MC_CMD_DRIVER_EVENT_IN_DATA_OFST 4
#define MC_CMD_DRIVER_EVENT_IN_DATA_LEN 8
#define MC_CMD_DRIVER_EVENT_IN_DATA_LO_OFST 4
#define MC_CMD_DRIVER_EVENT_IN_DATA_HI_OFST 8


/***********************************/
/* MC_CMD_PROXY_CMD
 */
#define MC_CMD_PROXY_CMD  0x5b

/* MC_CMD_PROXY_CMD_IN msgrequest */
#define MC_CMD_PROXY_CMD_IN_LEN 4
#define MC_CMD_PROXY_CMD_IN_TARGET_OFST 0


/***********************************/
/* MC_CMD_ALLOC_OWNER_IDS
 */
#define MC_CMD_ALLOC_OWNER_IDS  0x54

/* MC_CMD_ALLOC_OWNER_IDS_IN msgrequest */
#define MC_CMD_ALLOC_OWNER_IDS_IN_LEN 4
#define MC_CMD_ALLOC_OWNER_IDS_IN_NIDS_OFST 0

/* MC_CMD_ALLOC_OWNER_IDS_OUT msgresponse */
#define MC_CMD_ALLOC_OWNER_IDS_OUT_LEN 12
#define MC_CMD_ALLOC_OWNER_IDS_OUT_HANDLE_OFST 0
#define MC_CMD_ALLOC_OWNER_IDS_OUT_NIDS_OFST 4
```

```
2636 #define MC_CMD_ALLOC_OWNER_IDS_OUT_BASE_OFST 8


2639 /***********************************/
2640 /* MC_CMD_FREE_OWNER_IDS
2641  */
2642 #define MC_CMD_FREE_OWNER_IDS  0x59

2644 /* MC_CMD_FREE_OWNER_IDS_IN msgrequest */
2645 #define MC_CMD_FREE_OWNER_IDS_IN_LEN 4
2646 #define MC_CMD_FREE_OWNER_IDS_IN_HANDLE_OFST 0

2648 /* MC_CMD_FREE_OWNER_IDS_OUT msgresponse */
2649 #define MC_CMD_FREE_OWNER_IDS_OUT_LEN 0


2652 /***********************************/
2653 /* MC_CMD_ALLOC_BUFTBL_CHUNK
2654  */
2655 #define MC_CMD_ALLOC_BUFTBL_CHUNK  0x5c

2657 /* MC_CMD_ALLOC_BUFTBL_CHUNK_IN msgrequest */
2658 #define MC_CMD_ALLOC_BUFTBL_CHUNK_IN_LEN 8
2659 #define MC_CMD_ALLOC_BUFTBL_CHUNK_IN_OWNER_OFST 0
2660 #define MC_CMD_ALLOC_BUFTBL_CHUNK_IN_PAGE_SIZE_OFST 4

2662 /* MC_CMD_ALLOC_BUFTBL_CHUNK_OUT msgresponse */
2663 #define MC_CMD_ALLOC_BUFTBL_CHUNK_OUT_LEN 12
2664 #define MC_CMD_ALLOC_BUFTBL_CHUNK_OUT_HANDLE_OFST 0
2665 #define MC_CMD_ALLOC_BUFTBL_CHUNK_OUT_NUMENTRIES_OFST 4
2666 #define MC_CMD_ALLOC_BUFTBL_CHUNK_OUT_ID_OFST 8


2669 /***********************************/
2670 /* MC_CMD_PROGRAM_BUFTBL_ENTRIES
2671  */
2672 #define MC_CMD_PROGRAM_BUFTBL_ENTRIES  0x5d

2674 /* MC_CMD_PROGRAM_BUFTBL_ENTRIES_IN msgrequest */
2675 #define MC_CMD_PROGRAM_BUFTBL_ENTRIES_IN_LENMIN 20
2676 #define MC_CMD_PROGRAM_BUFTBL_ENTRIES_IN_LENMAX 252
2677 #define MC_CMD_PROGRAM_BUFTBL_ENTRIES_IN_LEN(num) (12+8*(num))
2678 #define MC_CMD_PROGRAM_BUFTBL_ENTRIES_IN_HANDLE_OFST 0
2679 #define MC_CMD_PROGRAM_BUFTBL_ENTRIES_IN_FIRSTID_OFST 4
2680 #define MC_CMD_PROGRAM_BUFTBL_ENTRIES_IN_NUMENTRIES_OFST 8
2681 #define MC_CMD_PROGRAM_BUFTBL_ENTRIES_IN_ENTRY_OFST 12
2682 #define MC_CMD_PROGRAM_BUFTBL_ENTRIES_IN_ENTRY_LEN 8
2683 #define MC_CMD_PROGRAM_BUFTBL_ENTRIES_IN_ENTRY_LO_OFST 12
2684 #define MC_CMD_PROGRAM_BUFTBL_ENTRIES_IN_ENTRY_HI_OFST 16
2685 #define MC_CMD_PROGRAM_BUFTBL_ENTRIES_IN_ENTRY_MINNUM 1
2686 #define MC_CMD_PROGRAM_BUFTBL_ENTRIES_IN_ENTRY_MAXNUM 30

2688 /* MC_CMD_PROGRAM_BUFTBL_ENTRIES_OUT msgresponse */
2689 #define MC_CMD_PROGRAM_BUFTBL_ENTRIES_OUT_LEN 0


2692 /***********************************/
2693 /* MC_CMD_FREE_BUFTBL_CHUNK
2694  */
2695 #define MC_CMD_FREE_BUFTBL_CHUNK  0x5e

2697 /* MC_CMD_FREE_BUFTBL_CHUNK_IN msgrequest */
2698 #define MC_CMD_FREE_BUFTBL_CHUNK_IN_LEN 4
2699 #define MC_CMD_FREE_BUFTBL_CHUNK_IN_HANDLE_OFST 0

2701 /* MC_CMD_FREE_BUFTBL_CHUNK_OUT msgresponse */
```



```
2702 #define MC_CMD_FREE_BUFTBL_CHUNK_OUT_LEN 0


2705 /***********************************/
2706 /* MC_CMD_GET_PF_COUNT
2707  */
2708 #define MC_CMD_GET_PF_COUNT  0x60

2710 /* MC_CMD_GET_PF_COUNT_IN msgrequest */
2711 #define MC_CMD_GET_PF_COUNT_IN_LEN 0

2713 /* MC_CMD_GET_PF_COUNT_OUT msgresponse */
2714 #define MC_CMD_GET_PF_COUNT_OUT_LEN 1
2715 #define MC_CMD_GET_PF_COUNT_OUT_PF_COUNT_OFST 0
2716 #define MC_CMD_GET_PF_COUNT_OUT_PF_COUNT_LEN 1


2719 /***********************************/
2720 /* MC_CMD_FILTER_OP
2721  */
2722 #define MC_CMD_FILTER_OP  0x61

2724 /* MC_CMD_FILTER_OP_IN msgrequest */
2725 #define MC_CMD_FILTER_OP_IN_LEN 100
2726 #define MC_CMD_FILTER_OP_IN_OP_OFST 0
2727 #define MC_CMD_FILTER_OP_IN_OP_INSERT  0x0 /* enum */
2728 #define MC_CMD_FILTER_OP_IN_OP_REMOVE  0x1 /* enum */
2729 #define MC_CMD_FILTER_OP_IN_OP_SUBSCRIBE  0x2 /* enum */
2730 #define MC_CMD_FILTER_OP_IN_OP_UNSUBSCRIBE  0x3 /* enum */
2731 #define MC_CMD_FILTER_OP_IN_HANDLE_OFST 4
2732 #define MC_CMD_FILTER_OP_IN_MATCH_FIELDS_OFST 8
2733 #define MC_CMD_FILTER_OP_IN_MATCH_SRC_IP_LBN 0
2734 #define MC_CMD_FILTER_OP_IN_MATCH_SRC_IP_WIDTH 1
2735 #define MC_CMD_FILTER_OP_IN_MATCH_DST_IP_LBN 1
2736 #define MC_CMD_FILTER_OP_IN_MATCH_DST_IP_WIDTH 1
2737 #define MC_CMD_FILTER_OP_IN_MATCH_SRC_MAC_LBN 2
2738 #define MC_CMD_FILTER_OP_IN_MATCH_SRC_MAC_WIDTH 1
2739 #define MC_CMD_FILTER_OP_IN_MATCH_SRC_PORT_LBN 3
2740 #define MC_CMD_FILTER_OP_IN_MATCH_SRC_PORT_WIDTH 1
2741 #define MC_CMD_FILTER_OP_IN_MATCH_DST_MAC_LBN 4
2742 #define MC_CMD_FILTER_OP_IN_MATCH_DST_MAC_WIDTH 1
2743 #define MC_CMD_FILTER_OP_IN_MATCH_DST_PORT_LBN 5
2744 #define MC_CMD_FILTER_OP_IN_MATCH_DST_PORT_WIDTH 1
2745 #define MC_CMD_FILTER_OP_IN_MATCH_ETHER_TYPE_LBN 6
2746 #define MC_CMD_FILTER_OP_IN_MATCH_ETHER_TYPE_WIDTH 1
2747 #define MC_CMD_FILTER_OP_IN_MATCH_INNER_VLAN_LBN 7
2748 #define MC_CMD_FILTER_OP_IN_MATCH_INNER_VLAN_WIDTH 1
2749 #define MC_CMD_FILTER_OP_IN_MATCH_OUTER_VLAN_LBN 8
2750 #define MC_CMD_FILTER_OP_IN_MATCH_OUTER_VLAN_WIDTH 1
2751 #define MC_CMD_FILTER_OP_IN_MATCH_IP_PROTO_LBN 9
2752 #define MC_CMD_FILTER_OP_IN_MATCH_IP_PROTO_WIDTH 1
2753 #define MC_CMD_FILTER_OP_IN_MATCH_FWDEF0_LBN 10
2754 #define MC_CMD_FILTER_OP_IN_MATCH_FWDEF0_WIDTH 1
2755 #define MC_CMD_FILTER_OP_IN_MATCH_FWDEF1_LBN 11
2756 #define MC_CMD_FILTER_OP_IN_MATCH_FWDEF1_WIDTH 1
2757 #define MC_CMD_FILTER_OP_IN_RX_DEST_OFST 12
2758 #define MC_CMD_FILTER_OP_IN_RX_DEST_DROP  0x0 /* enum */
2759 #define MC_CMD_FILTER_OP_IN_RX_DEST_HOST  0x1 /* enum */
2760 #define MC_CMD_FILTER_OP_IN_RX_DEST_MC  0x2 /* enum */
2761 #define MC_CMD_FILTER_OP_IN_RX_DEST_TX0  0x3 /* enum */
2762 #define MC_CMD_FILTER_OP_IN_RX_DEST_TX1  0x4 /* enum */
2763 #define MC_CMD_FILTER_OP_IN_RX_QUEUE_OFST 16
2764 #define MC_CMD_FILTER_OP_IN_RX_FLAGS_OFST 20
2765 #define MC_CMD_FILTER_OP_IN_RX_FLAG_RSS_LBN 0
2766 #define MC_CMD_FILTER_OP_IN_RX_FLAG_RSS_WIDTH 1
2767 #define MC_CMD_FILTER_OP_IN_RSS_CONTEXT_OFST 24
```

```
#define MC_CMD_FILTER_OP_IN_TX_DOMAIN_OFST 28
#define MC_CMD_FILTER_OP_IN_TX_DEST_OFST 32
#define MC_CMD_FILTER_OP_IN_TX_DEST_MAC_LBN 0
#define MC_CMD_FILTER_OP_IN_TX_DEST_MAC_WIDTH 1
#define MC_CMD_FILTER_OP_IN_TX_DEST_PM_LBN 1
#define MC_CMD_FILTER_OP_IN_TX_DEST_PM_WIDTH 1
#define MC_CMD_FILTER_OP_IN_SRC_MAC_OFST 36
#define MC_CMD_FILTER_OP_IN_SRC_MAC_LEN 6
#define MC_CMD_FILTER_OP_IN_SRC_PORT_OFST 42
#define MC_CMD_FILTER_OP_IN_SRC_PORT_LEN 2
#define MC_CMD_FILTER_OP_IN_DST_MAC_OFST 44
#define MC_CMD_FILTER_OP_IN_DST_MAC_LEN 6
#define MC_CMD_FILTER_OP_IN_DST_PORT_OFST 50
#define MC_CMD_FILTER_OP_IN_DST_PORT_LEN 2
#define MC_CMD_FILTER_OP_IN_ETHER_TYPE_OFST 52
#define MC_CMD_FILTER_OP_IN_ETHER_TYPE_LEN 2
#define MC_CMD_FILTER_OP_IN_INNER_VLAN_OFST 54
#define MC_CMD_FILTER_OP_IN_INNER_VLAN_LEN 2
#define MC_CMD_FILTER_OP_IN_OUTER_VLAN_OFST 56
#define MC_CMD_FILTER_OP_IN_OUTER_VLAN_LEN 2
#define MC_CMD_FILTER_OP_IN_IP_PROTO_OFST 58
#define MC_CMD_FILTER_OP_IN_IP_PROTO_LEN 2
#define MC_CMD_FILTER_OP_IN_FWDEF0_OFST 60
#define MC_CMD_FILTER_OP_IN_FWDEF1_OFST 64
#define MC_CMD_FILTER_OP_IN_SRC_IP_OFST 68
#define MC_CMD_FILTER_OP_IN_SRC_IP_LEN 16
#define MC_CMD_FILTER_OP_IN_DST_IP_OFST 84
#define MC_CMD_FILTER_OP_IN_DST_IP_LEN 16

/* MC_CMD_FILTER_OP_OUT msgresponse */
#define MC_CMD_FILTER_OP_OUT_LEN 8
#define MC_CMD_FILTER_OP_OUT_OP_OFST 0
#define MC_CMD_FILTER_OP_OUT_OP_INSERT  0x0 /* enum */
#define MC_CMD_FILTER_OP_OUT_OP_REMOVE  0x1 /* enum */
#define MC_CMD_FILTER_OP_OUT_OP_SUBSCRIBE  0x2 /* enum */
#define MC_CMD_FILTER_OP_OUT_OP_UNSUBSCRIBE  0x3 /* enum */
#define MC_CMD_FILTER_OP_OUT_HANDLE_OFST 4


/***********************************/
/* MC_CMD_SET_PF_COUNT
 */
#define MC_CMD_SET_PF_COUNT  0x62

/* MC_CMD_SET_PF_COUNT_IN msgrequest */
#define MC_CMD_SET_PF_COUNT_IN_LEN 4
#define MC_CMD_SET_PF_COUNT_IN_PF_COUNT_OFST 0

/* MC_CMD_SET_PF_COUNT_OUT msgresponse */
#define MC_CMD_SET_PF_COUNT_OUT_LEN 0


/***********************************/
/* MC_CMD_GET_PORT_ASSIGNMENT
 */
#define MC_CMD_GET_PORT_ASSIGNMENT  0x63

/* MC_CMD_GET_PORT_ASSIGNMENT_IN msgrequest */
#define MC_CMD_GET_PORT_ASSIGNMENT_IN_LEN 0

/* MC_CMD_GET_PORT_ASSIGNMENT_OUT msgresponse */
#define MC_CMD_GET_PORT_ASSIGNMENT_OUT_LEN 4
#define MC_CMD_GET_PORT_ASSIGNMENT_OUT_PORT_OFST 0


/***********************************/
```



```
/* MC_CMD_SET_PORT_ASSIGNMENT
 */
#define MC_CMD_SET_PORT_ASSIGNMENT  0x64

/* MC_CMD_SET_PORT_ASSIGNMENT_IN msgrequest */
#define MC_CMD_SET_PORT_ASSIGNMENT_IN_LEN 4
#define MC_CMD_SET_PORT_ASSIGNMENT_IN_PORT_OFST 0

/* MC_CMD_SET_PORT_ASSIGNMENT_OUT msgresponse */
#define MC_CMD_SET_PORT_ASSIGNMENT_OUT_LEN 0


/***********************************/
/* MC_CMD_ALLOC_VIS
 */
#define MC_CMD_ALLOC_VIS  0x65

/* MC_CMD_ALLOC_VIS_IN msgrequest */
#define MC_CMD_ALLOC_VIS_IN_LEN 4
#define MC_CMD_ALLOC_VIS_IN_VI_COUNT_OFST 0

/* MC_CMD_ALLOC_VIS_OUT msgresponse */
#define MC_CMD_ALLOC_VIS_OUT_LEN 0


/***********************************/
/* MC_CMD_FREE_VIS
 */
#define MC_CMD_FREE_VIS  0x66

/* MC_CMD_FREE_VIS_IN msgrequest */
#define MC_CMD_FREE_VIS_IN_LEN 0

/* MC_CMD_FREE_VIS_OUT msgresponse */
#define MC_CMD_FREE_VIS_OUT_LEN 0


/***********************************/
/* MC_CMD_GET_SRIOV_CFG
 */
#define MC_CMD_GET_SRIOV_CFG  0x67

/* MC_CMD_GET_SRIOV_CFG_IN msgrequest */
#define MC_CMD_GET_SRIOV_CFG_IN_LEN 0

/* MC_CMD_GET_SRIOV_CFG_OUT msgresponse */
#define MC_CMD_GET_SRIOV_CFG_OUT_LEN 20
#define MC_CMD_GET_SRIOV_CFG_OUT_VF_CURRENT_OFST 0
#define MC_CMD_GET_SRIOV_CFG_OUT_VF_MAX_OFST 4
#define MC_CMD_GET_SRIOV_CFG_OUT_FLAGS_OFST 8
#define MC_CMD_GET_SRIOV_CFG_OUT_VF_ENABLED_LBN 0
#define MC_CMD_GET_SRIOV_CFG_OUT_VF_ENABLED_WIDTH 1
#define MC_CMD_GET_SRIOV_CFG_OUT_VF_OFFSET_OFST 12
#define MC_CMD_GET_SRIOV_CFG_OUT_VF_STRIDE_OFST 16


/***********************************/
/* MC_CMD_SET_SRIOV_CFG
 */
#define MC_CMD_SET_SRIOV_CFG  0x68

/* MC_CMD_SET_SRIOV_CFG_IN msgrequest */
#define MC_CMD_SET_SRIOV_CFG_IN_LEN 20
#define MC_CMD_SET_SRIOV_CFG_IN_VF_CURRENT_OFST 0
#define MC_CMD_SET_SRIOV_CFG_IN_VF_MAX_OFST 4
#define MC_CMD_SET_SRIOV_CFG_IN_FLAGS_OFST 8
```

```
#define MC_CMD_SET_SRIOV_CFG_IN_VF_ENABLED_LBN 0
#define MC_CMD_SET_SRIOV_CFG_IN_VF_ENABLED_WIDTH 1
#define MC_CMD_SET_SRIOV_CFG_IN_VF_OFFSET_OFST 12
#define MC_CMD_SET_SRIOV_CFG_IN_VF_STRIDE_OFST 16

/* MC_CMD_SET_SRIOV_CFG_OUT msgresponse */
#define MC_CMD_SET_SRIOV_CFG_OUT_LEN 0


/***********************************/
/* MC_CMD_GET_VI_COUNT
 */
#define MC_CMD_GET_VI_COUNT  0x69

/* MC_CMD_GET_VI_COUNT_IN msgrequest */
#define MC_CMD_GET_VI_COUNT_IN_LEN 0

/* MC_CMD_GET_VI_COUNT_OUT msgresponse */
#define MC_CMD_GET_VI_COUNT_OUT_LEN 4
#define MC_CMD_GET_VI_COUNT_OUT_VI_COUNT_OFST 0


/***********************************/
/* MC_CMD_GET_VECTOR_CFG
 */
#define MC_CMD_GET_VECTOR_CFG  0x70

/* MC_CMD_GET_VECTOR_CFG_IN msgrequest */
#define MC_CMD_GET_VECTOR_CFG_IN_LEN 0

/* MC_CMD_GET_VECTOR_CFG_OUT msgresponse */
#define MC_CMD_GET_VECTOR_CFG_OUT_LEN 12
#define MC_CMD_GET_VECTOR_CFG_OUT_VEC_BASE_OFST 0
#define MC_CMD_GET_VECTOR_CFG_OUT_VECS_PER_PF_OFST 4
#define MC_CMD_GET_VECTOR_CFG_OUT_VECS_PER_VF_OFST 8


/***********************************/
/* MC_CMD_SET_VECTOR_CFG
 */
#define MC_CMD_SET_VECTOR_CFG  0x71

/* MC_CMD_SET_VECTOR_CFG_IN msgrequest */
#define MC_CMD_SET_VECTOR_CFG_IN_LEN 12
#define MC_CMD_SET_VECTOR_CFG_IN_VEC_BASE_OFST 0
#define MC_CMD_SET_VECTOR_CFG_IN_VECS_PER_PF_OFST 4
#define MC_CMD_SET_VECTOR_CFG_IN_VECS_PER_VF_OFST 8

/* MC_CMD_SET_VECTOR_CFG_OUT msgresponse */
#define MC_CMD_SET_VECTOR_CFG_OUT_LEN 0


/***********************************/
/* MC_CMD_ALLOC_PIOBUF
 */
#define MC_CMD_ALLOC_PIOBUF  0x72

/* MC_CMD_ALLOC_PIOBUF_IN msgrequest */
#define MC_CMD_ALLOC_PIOBUF_IN_LEN 0

/* MC_CMD_ALLOC_PIOBUF_OUT msgresponse */
#define MC_CMD_ALLOC_PIOBUF_OUT_LEN 4
#define MC_CMD_ALLOC_PIOBUF_OUT_PIOBUF_HANDLE_OFST 0


/***********************************/
```



```
/* MC_CMD_FREE_PIOBUF
 */
#define MC_CMD_FREE_PIOBUF  0x73

/* MC_CMD_FREE_PIOBUF_IN msgrequest */
#define MC_CMD_FREE_PIOBUF_IN_LEN 4
#define MC_CMD_FREE_PIOBUF_IN_PIOBUF_HANDLE_OFST 0

/* MC_CMD_FREE_PIOBUF_OUT msgresponse */
#define MC_CMD_FREE_PIOBUF_OUT_LEN 0


/***********************************/
/* MC_CMD_V2_EXTN
 */
#define MC_CMD_V2_EXTN  0x7f

/* MC_CMD_V2_EXTN_IN msgrequest */
#define MC_CMD_V2_EXTN_IN_LEN 4
#define MC_CMD_V2_EXTN_IN_EXTENDED_CMD_LBN 0
#define MC_CMD_V2_EXTN_IN_EXTENDED_CMD_WIDTH 15
#define MC_CMD_V2_EXTN_IN_UNUSED_LBN 15
#define MC_CMD_V2_EXTN_IN_UNUSED_WIDTH 1
#define MC_CMD_V2_EXTN_IN_ACTUAL_LEN_LBN 16
#define MC_CMD_V2_EXTN_IN_ACTUAL_LEN_WIDTH 10
#define MC_CMD_V2_EXTN_IN_UNUSED2_LBN 26
#define MC_CMD_V2_EXTN_IN_UNUSED2_WIDTH 6


/***********************************/
/* MC_CMD_TCM_BUCKET_ALLOC
 */
#define MC_CMD_TCM_BUCKET_ALLOC  0x80

/* MC_CMD_TCM_BUCKET_ALLOC_IN msgrequest */
#define MC_CMD_TCM_BUCKET_ALLOC_IN_LEN 0

/* MC_CMD_TCM_BUCKET_ALLOC_OUT msgresponse */
#define MC_CMD_TCM_BUCKET_ALLOC_OUT_LEN 4
#define MC_CMD_TCM_BUCKET_ALLOC_OUT_BUCKET_OFST 0


/***********************************/
/* MC_CMD_TCM_BUCKET_FREE
 */
#define MC_CMD_TCM_BUCKET_FREE  0x81

/* MC_CMD_TCM_BUCKET_FREE_IN msgrequest */
#define MC_CMD_TCM_BUCKET_FREE_IN_LEN 4
#define MC_CMD_TCM_BUCKET_FREE_IN_BUCKET_OFST 0

/* MC_CMD_TCM_BUCKET_FREE_OUT msgresponse */
#define MC_CMD_TCM_BUCKET_FREE_OUT_LEN 0


/***********************************/
/* MC_CMD_TCM_BUCKET_INIT
 */
#define MC_CMD_TCM_BUCKET_INIT  0x82

/* MC_CMD_TCM_BUCKET_INIT_IN msgrequest */
#define MC_CMD_TCM_BUCKET_INIT_IN_LEN 8
#define MC_CMD_TCM_BUCKET_INIT_IN_BUCKET_OFST 0
#define MC_CMD_TCM_BUCKET_INIT_IN_RATE_OFST 4

/* MC_CMD_TCM_BUCKET_INIT_OUT msgresponse */
```

```
3032 #define MC_CMD_TCM_BUCKET_INIT_OUT_LEN 0


3035 /***********************************/
3036 /* MC_CMD_TCM_TXQ_INIT
3037  */
3038 #define MC_CMD_TCM_TXQ_INIT  0x83

3040 /* MC_CMD_TCM_TXQ_INIT_IN msgrequest */
3041 #define MC_CMD_TCM_TXQ_INIT_IN_LEN 28
3042 #define MC_CMD_TCM_TXQ_INIT_IN_QID_OFST 0
3043 #define MC_CMD_TCM_TXQ_INIT_IN_LABEL_OFST 4
3044 #define MC_CMD_TCM_TXQ_INIT_IN_PQ_FLAGS_OFST 8
3045 #define MC_CMD_TCM_TXQ_INIT_IN_RP_BKT_OFST 12
3046 #define MC_CMD_TCM_TXQ_INIT_IN_MAX_BKT1_OFST 16
3047 #define MC_CMD_TCM_TXQ_INIT_IN_MAX_BKT2_OFST 20
3048 #define MC_CMD_TCM_TXQ_INIT_IN_MIN_BKT_OFST 24

3050 /* MC_CMD_TCM_TXQ_INIT_OUT msgresponse */
3051 #define MC_CMD_TCM_TXQ_INIT_OUT_LEN 0

3053 #endif /* _SIENA_MC_DRIVER_PCOL_H */
3054 /*! \cidoxg_end */
3055 #endif /* ! codereview */
```

****
56501 Thu Aug 22 18:59:23 2013
new/usr/src/uts/common/io/sfxge/efx_regs_pci.h
Merged sfxge driver
****

```
/*-
 * Copyright 2007-2013 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#ifndef _SYS_EFX_REGS_PCI_H
#define _SYS_EFX_REGS_PCI_H

#ifdef  __cplusplus
extern "C" {
#endif

/*
 * PC_VEND_ID_REG(16bit):
 * Vendor ID register
 */

#define PCR_AZ_VEND_ID_REG 0x00000000
/* falcona0,falconb0,sienaa0,hunta0=pci_f0_config */

#define PCRF_AZ_VEND_ID_LBN 0
#define PCRF_AZ_VEND_ID_WIDTH 16


/*
 * PC_DEV_ID_REG(16bit):
 * Device ID register
 */

#define PCR_AZ_DEV_ID_REG 0x00000002
/* falcona0,falconb0,sienaa0,hunta0=pci_f0_config */

#define PCRF_AZ_DEV_ID_LBN 0
#define PCRF_AZ_DEV_ID_WIDTH 16


/*
 * PC_CMD_REG(16bit):
 * Command register
 */
```



```
#define PCR_AZ_CMD_REG 0x00000004
/* falcona0,falconb0,sienaa0,hunta0=pci_f0_config */

#define PCRF_AZ_INTX_DIS_LBN 10
#define PCRF_AZ_INTX_DIS_WIDTH 1
#define PCRF_AZ_FB2B_EN_LBN 9
#define PCRF_AZ_FB2B_EN_WIDTH 1
#define PCRF_AZ_SERR_EN_LBN 8
#define PCRF_AZ_SERR_EN_WIDTH 1
#define PCRF_AZ_IDSEL_CTL_LBN 7
#define PCRF_AZ_IDSEL_CTL_WIDTH 1
#define PCRF_AZ_PERR_EN_LBN 6
#define PCRF_AZ_PERR_EN_WIDTH 1
#define PCRF_AZ_VGA_PAL_SNP_LBN 5
#define PCRF_AZ_VGA_PAL_SNP_WIDTH 1
#define PCRF_AZ_MWI_EN_LBN 4
#define PCRF_AZ_MWI_EN_WIDTH 1
#define PCRF_AZ_SPEC_CYC_LBN 3
#define PCRF_AZ_SPEC_CYC_WIDTH 1
#define PCRF_AZ_MST_EN_LBN 2
#define PCRF_AZ_MST_EN_WIDTH 1
#define PCRF_AZ_MEM_EN_LBN 1
#define PCRF_AZ_MEM_EN_WIDTH 1
#define PCRF_AZ_IO_EN_LBN 0
#define PCRF_AZ_IO_EN_WIDTH 1


/*
 * PC_STAT_REG(16bit):
 * Status register
 */

#define PCR_AZ_STAT_REG 0x00000006
/* falcona0,falconb0,sienaa0,hunta0=pci_f0_config */

#define PCRF_AZ_DET_PERR_LBN 15
#define PCRF_AZ_DET_PERR_WIDTH 1
#define PCRF_AZ_SIG_SERR_LBN 14
#define PCRF_AZ_SIG_SERR_WIDTH 1
#define PCRF_AZ_GOT_MABRT_LBN 13
#define PCRF_AZ_GOT_MABRT_WIDTH 1
#define PCRF_AZ_GOT_TABRT_LBN 12
#define PCRF_AZ_GOT_TABRT_WIDTH 1
#define PCRF_AZ_SIG_TABRT_LBN 11
#define PCRF_AZ_SIG_TABRT_WIDTH 1
#define PCRF_AZ_DEVSEL_TIM_LBN 9
#define PCRF_AZ_DEVSEL_TIM_WIDTH 2
#define PCRF_AZ_MDAT_PERR_LBN 8
#define PCRF_AZ_MDAT_PERR_WIDTH 1
#define PCRF_AZ_FB2B_CAP_LBN 7
#define PCRF_AZ_FB2B_CAP_WIDTH 1
#define PCRF_AZ_66MHZ_CAP_LBN 5
#define PCRF_AZ_66MHZ_CAP_WIDTH 1
#define PCRF_AZ_CAP_LIST_LBN 4
#define PCRF_AZ_CAP_LIST_WIDTH 1
#define PCRF_AZ_INTX_STAT_LBN 3
#define PCRF_AZ_INTX_STAT_WIDTH 1


/*
 * PC_REV_ID_REG(8bit):
 * Class code & revision ID register
 */

#define PCR_AZ_REV_ID_REG 0x00000008
/* falcona0,falconb0,sienaa0,hunta0=pci_f0_config */
```

```
129 #define PCRF_AZ_REV_ID_LBN 0
130 #define PCRF_AZ_REV_ID_WIDTH 8


133 /*
134  * PC_CC_REG(24bit):
135  * Class code register
136  */

138 #define PCR_AZ_CC_REG 0x00000009
139 /* falcona0,falconb0,sienaa0,hunta0=pci_f0_config */

141 #define PCRF_AZ_BASE_CC_LBN 16
142 #define PCRF_AZ_BASE_CC_WIDTH 8
143 #define PCRF_AZ_SUB_CC_LBN 8
144 #define PCRF_AZ_SUB_CC_WIDTH 8
145 #define PCRF_AZ_PROG_IF_LBN 0
146 #define PCRF_AZ_PROG_IF_WIDTH 8


149 /*
150  * PC_CACHE_LSIZE_REG(8bit):
151  * Cache line size
152  */

154 #define PCR_AZ_CACHE_LSIZE_REG 0x0000000c
155 /* falcona0,falconb0,sienaa0,hunta0=pci_f0_config */

157 #define PCRF_AZ_CACHE_LSIZE_LBN 0
158 #define PCRF_AZ_CACHE_LSIZE_WIDTH 8


161 /*
162  * PC_MST_LAT_REG(8bit):
163  * Master latency timer register
164  */

166 #define PCR_AZ_MST_LAT_REG 0x0000000d
167 /* falcona0,falconb0,sienaa0,hunta0=pci_f0_config */

169 #define PCRF_AZ_MST_LAT_LBN 0
170 #define PCRF_AZ_MST_LAT_WIDTH 8


173 /*
174  * PC_HDR_TYPE_REG(8bit):
175  * Header type register
176  */

178 #define PCR_AZ_HDR_TYPE_REG 0x0000000e
179 /* falcona0,falconb0,sienaa0,hunta0=pci_f0_config */

181 #define PCRF_AZ_MULT_FUNC_LBN 7
182 #define PCRF_AZ_MULT_FUNC_WIDTH 1
183 #define PCRF_AZ_TYPE_LBN 0
184 #define PCRF_AZ_TYPE_WIDTH 7


187 /*
188  * PC_BIST_REG(8bit):
189  * BIST register
190  */

192 #define PCR_AZ_BIST_REG 0x0000000f
193 /* falcona0,falconb0,sienaa0,hunta0=pci_f0_config */
```



```
195 #define PCRF_AZ_BIST_LBN 0
196 #define PCRF_AZ_BIST_WIDTH 8


199 /*
200  * PC_BAR0_REG(32bit):
201  * Primary function base address register 0
202  */

204 #define PCR_AZ_BAR0_REG 0x00000010
205 /* falcona0,falconb0,sienaa0,hunta0=pci_f0_config */

207 #define PCRF_AZ_BAR0_LBN 4
208 #define PCRF_AZ_BAR0_WIDTH 28
209 #define PCRF_AZ_BAR0_PREF_LBN 3
210 #define PCRF_AZ_BAR0_PREF_WIDTH 1
211 #define PCRF_AZ_BAR0_TYPE_LBN 1
212 #define PCRF_AZ_BAR0_TYPE_WIDTH 2
213 #define PCRF_AZ_BAR0_IOM_LBN 0
214 #define PCRF_AZ_BAR0_IOM_WIDTH 1


217 /*
218  * PC_BAR1_REG(32bit):
219  * Primary function base address register 1, BAR1 is not implemented so read onl
220  */

222 #define PCR_DZ_BAR1_REG 0x00000014
223 /* hunta0=pci_f0_config */

225 #define PCRF_DZ_BAR1_LBN 0
226 #define PCRF_DZ_BAR1_WIDTH 32


229 /*
230  * PC_BAR2_LO_REG(32bit):
231  * Primary function base address register 2 low bits
232  */

234 #define PCR_AZ_BAR2_LO_REG 0x00000018
235 /* falcona0,falconb0,sienaa0,hunta0=pci_f0_config */

237 #define PCRF_AZ_BAR2_LO_LBN 4
238 #define PCRF_AZ_BAR2_LO_WIDTH 28
239 #define PCRF_AZ_BAR2_PREF_LBN 3
240 #define PCRF_AZ_BAR2_PREF_WIDTH 1
241 #define PCRF_AZ_BAR2_TYPE_LBN 1
242 #define PCRF_AZ_BAR2_TYPE_WIDTH 2
243 #define PCRF_AZ_BAR2_IOM_LBN 0
244 #define PCRF_AZ_BAR2_IOM_WIDTH 1


247 /*
248  * PC_BAR2_HI_REG(32bit):
249  * Primary function base address register 2 high bits
250  */

252 #define PCR_AZ_BAR2_HI_REG 0x0000001c
253 /* falcona0,falconb0,sienaa0,hunta0=pci_f0_config */

255 #define PCRF_AZ_BAR2_HI_LBN 0
256 #define PCRF_AZ_BAR2_HI_WIDTH 32


259 /*
```

```
 * PC_BAR4_LO_REG(32bit):
 * Primary function base address register 2 low bits
 */

#define PCR_CZ_BAR4_LO_REG 0x00000020
/* sienaa0,hunta0=pci_f0_config */

#define PCRF_CZ_BAR4_LO_LBN 4
#define PCRF_CZ_BAR4_LO_WIDTH 28
#define PCRF_CZ_BAR4_PREF_LBN 3
#define PCRF_CZ_BAR4_PREF_WIDTH 1
#define PCRF_CZ_BAR4_TYPE_LBN 1
#define PCRF_CZ_BAR4_TYPE_WIDTH 2
#define PCRF_CZ_BAR4_IOM_LBN 0
#define PCRF_CZ_BAR4_IOM_WIDTH 1


/*
 * PC_BAR4_HI_REG(32bit):
 * Primary function base address register 2 high bits
 */

#define PCR_CZ_BAR4_HI_REG 0x00000024
/* sienaa0,hunta0=pci_f0_config */

#define PCRF_CZ_BAR4_HI_LBN 0
#define PCRF_CZ_BAR4_HI_WIDTH 32


/*
 * PC_SS_VEND_ID_REG(16bit):
 * Sub-system vendor ID register
 */

#define PCR_AZ_SS_VEND_ID_REG 0x0000002c
/* falcona0,falconb0,sienaa0,hunta0=pci_f0_config */

#define PCRF_AZ_SS_VEND_ID_LBN 0
#define PCRF_AZ_SS_VEND_ID_WIDTH 16


/*
 * PC_SS_ID_REG(16bit):
 * Sub-system ID register
 */

#define PCR_AZ_SS_ID_REG 0x0000002e
/* falcona0,falconb0,sienaa0,hunta0=pci_f0_config */

#define PCRF_AZ_SS_ID_LBN 0
#define PCRF_AZ_SS_ID_WIDTH 16


/*
 * PC_EXPROM_BAR_REG(32bit):
 * Expansion ROM base address register
 */

#define PCR_AZ_EXPROM_BAR_REG 0x00000030
/* falcona0,falconb0,sienaa0,hunta0=pci_f0_config */

#define PCRF_AZ_EXPROM_BAR_LBN 11
#define PCRF_AZ_EXPROM_BAR_WIDTH 21
#define PCRF_AB_EXPROM_MIN_SIZE_LBN 2
#define PCRF_AB_EXPROM_MIN_SIZE_WIDTH 9
#define PCRF_CZ_EXPROM_MIN_SIZE_LBN 1
```



```
#define PCRF_CZ_EXPROM_MIN_SIZE_WIDTH 10
#define PCRF_AB_EXPROM_FEATURE_ENABLE_LBN 1
#define PCRF_AB_EXPROM_FEATURE_ENABLE_WIDTH 1
#define PCRF_AZ_EXPROM_EN_LBN 0
#define PCRF_AZ_EXPROM_EN_WIDTH 1


/*
 * PC_CAP_PTR_REG(8bit):
 * Capability pointer register
 */

#define PCR_AZ_CAP_PTR_REG 0x00000034
/* falcona0,falconb0,sienaa0,hunta0=pci_f0_config */

#define PCRF_AZ_CAP_PTR_LBN 0
#define PCRF_AZ_CAP_PTR_WIDTH 8


/*
 * PC_INT_LINE_REG(8bit):
 * Interrupt line register
 */

#define PCR_AZ_INT_LINE_REG 0x0000003c
/* falcona0,falconb0,sienaa0,hunta0=pci_f0_config */

#define PCRF_AZ_INT_LINE_LBN 0
#define PCRF_AZ_INT_LINE_WIDTH 8


/*
 * PC_INT_PIN_REG(8bit):
 * Interrupt pin register
 */

#define PCR_AZ_INT_PIN_REG 0x0000003d
/* falcona0,falconb0,sienaa0,hunta0=pci_f0_config */

#define PCRF_AZ_INT_PIN_LBN 0
#define PCRF_AZ_INT_PIN_WIDTH 8


/*
 * PC_PM_CAP_ID_REG(8bit):
 * Power management capability ID
 */

#define PCR_AC_PM_CAP_ID_REG 0x00000040
/* falcona0,falconb0,sienaa0=pci_f0_config */

#define PCR_DZ_PM_CAP_ID_REG 0x00000080
/* hunta0=pci_f0_config */

#define PCRF_AZ_PM_CAP_ID_LBN 0
#define PCRF_AZ_PM_CAP_ID_WIDTH 8


/*
 * PC_PM_NXT_PTR_REG(8bit):
 * Power management next item pointer
 */

#define PCR_AC_PM_NXT_PTR_REG 0x00000041
/* falcona0,falconb0,sienaa0=pci_f0_config */
```

```
392 #define PCR_DZ_PM_NXT_PTR_REG 0x00000081
393 /* hunta0=pci_f0_config */

395 #define PCRF_AZ_PM_NXT_PTR_LBN 0
396 #define PCRF_AZ_PM_NXT_PTR_WIDTH 8


399 /*
400  * PC_PM_CAP_REG(16bit):
401  * Power management capabilities register
402  */

404 #define PCR_AC_PM_CAP_REG 0x00000042
405 /* falcona0,falconb0,sienaa0=pci_f0_config */

407 #define PCR_DZ_PM_CAP_REG 0x00000082
408 /* hunta0=pci_f0_config */

410 #define PCRF_AZ_PM_PME_SUPT_LBN 11
411 #define PCRF_AZ_PM_PME_SUPT_WIDTH 5
412 #define PCRF_AZ_PM_D2_SUPT_LBN 10
413 #define PCRF_AZ_PM_D2_SUPT_WIDTH 1
414 #define PCRF_AZ_PM_D1_SUPT_LBN 9
415 #define PCRF_AZ_PM_D1_SUPT_WIDTH 1
416 #define PCRF_AZ_PM_AUX_CURR_LBN 6
417 #define PCRF_AZ_PM_AUX_CURR_WIDTH 3
418 #define PCRF_AZ_PM_DSI_LBN 5
419 #define PCRF_AZ_PM_DSI_WIDTH 1
420 #define PCRF_AZ_PM_PME_CLK_LBN 3
421 #define PCRF_AZ_PM_PME_CLK_WIDTH 1
422 #define PCRF_AZ_PM_PME_VER_LBN 0
423 #define PCRF_AZ_PM_PME_VER_WIDTH 3


426 /*
427  * PC_PM_CS_REG(16bit):
428  * Power management control & status register
429  */

431 #define PCR_AC_PM_CS_REG 0x00000044
432 /* falcona0,falconb0,sienaa0=pci_f0_config */

434 #define PCR_DZ_PM_CS_REG 0x00000084
435 /* hunta0=pci_f0_config */

437 #define PCRF_AZ_PM_PME_STAT_LBN 15
438 #define PCRF_AZ_PM_PME_STAT_WIDTH 1
439 #define PCRF_AZ_PM_DAT_SCALE_LBN 13
440 #define PCRF_AZ_PM_DAT_SCALE_WIDTH 2
441 #define PCRF_AZ_PM_DAT_SEL_LBN 9
442 #define PCRF_AZ_PM_DAT_SEL_WIDTH 4
443 #define PCRF_AZ_PM_PME_EN_LBN 8
444 #define PCRF_AZ_PM_PME_EN_WIDTH 1
445 #define PCRF_CZ_NO_SOFT_RESET_LBN 3
446 #define PCRF_CZ_NO_SOFT_RESET_WIDTH 1
447 #define PCRF_AZ_PM_PWR_ST_LBN 0
448 #define PCRF_AZ_PM_PWR_ST_WIDTH 2


451 /*
452  * PC_MSI_CAP_ID_REG(8bit):
453  * MSI capability ID
454  */

456 #define PCR_AC_MSI_CAP_ID_REG 0x00000050
457 /* falcona0,falconb0,sienaa0=pci_f0_config */
```



```
459 #define PCR_DZ_MSI_CAP_ID_REG 0x00000090
460 /* hunta0=pci_f0_config */

462 #define PCRF_AZ_MSI_CAP_ID_LBN 0
463 #define PCRF_AZ_MSI_CAP_ID_WIDTH 8


466 /*
467  * PC_MSI_NXT_PTR_REG(8bit):
468  * MSI next item pointer
469  */

471 #define PCR_AC_MSI_NXT_PTR_REG 0x00000051
472 /* falcona0,falconb0,sienaa0=pci_f0_config */

474 #define PCR_DZ_MSI_NXT_PTR_REG 0x00000091
475 /* hunta0=pci_f0_config */

477 #define PCRF_AZ_MSI_NXT_PTR_LBN 0
478 #define PCRF_AZ_MSI_NXT_PTR_WIDTH 8


481 /*
482  * PC_MSI_CTL_REG(16bit):
483  * MSI control register
484  */

486 #define PCR_AC_MSI_CTL_REG 0x00000052
487 /* falcona0,falconb0,sienaa0=pci_f0_config */

489 #define PCR_DZ_MSI_CTL_REG 0x00000092
490 /* hunta0=pci_f0_config */

492 #define PCRF_AZ_MSI_64_EN_LBN 7
493 #define PCRF_AZ_MSI_64_EN_WIDTH 1
494 #define PCRF_AZ_MSI_MULT_MSG_EN_LBN 4
495 #define PCRF_AZ_MSI_MULT_MSG_EN_WIDTH 3
496 #define PCRF_AZ_MSI_MULT_MSG_CAP_LBN 1
497 #define PCRF_AZ_MSI_MULT_MSG_CAP_WIDTH 3
498 #define PCRF_AZ_MSI_EN_LBN 0
499 #define PCRF_AZ_MSI_EN_WIDTH 1


502 /*
503  * PC_MSI_ADR_LO_REG(32bit):
504  * MSI low 32 bits address register
505  */

507 #define PCR_AC_MSI_ADR_LO_REG 0x00000054
508 /* falcona0,falconb0,sienaa0=pci_f0_config */

510 #define PCR_DZ_MSI_ADR_LO_REG 0x00000094
511 /* hunta0=pci_f0_config */

513 #define PCRF_AZ_MSI_ADR_LO_LBN 2
514 #define PCRF_AZ_MSI_ADR_LO_WIDTH 30


517 /*
518  * PC_VPD_CAP_CTL_REG(8bit):
519  * VPD control and capabilities register
520  */

522 #define PCR_DZ_VPD_CAP_CTL_REG 0x00000054
523 /* hunta0=pci_f0_config */
```

```
525 #define PCR_CC_VPD_CAP_CTL_REG 0x000000d0
526 /* sienaa0=pci_f0_config */

528 #define PCRF_CZ_VPD_FLAG_LBN 31
529 #define PCRF_CZ_VPD_FLAG_WIDTH 1
530 #define PCRF_CZ_VPD_ADDR_LBN 16
531 #define PCRF_CZ_VPD_ADDR_WIDTH 15
532 #define PCRF_CZ_VPD_NXT_PTR_LBN 8
533 #define PCRF_CZ_VPD_NXT_PTR_WIDTH 8
534 #define PCRF_CZ_VPD_CAP_ID_LBN 0
535 #define PCRF_CZ_VPD_CAP_ID_WIDTH 8


538 /*
539  * PC_VPD_CAP_DATA_REG(32bit):
540  * documentation to be written for sum_PC_VPD_CAP_DATA_REG
541  */

543 #define PCR_DZ_VPD_CAP_DATA_REG 0x00000058
544 /* hunta0=pci_f0_config */

546 #define PCR_AB_VPD_CAP_DATA_REG 0x000000b4
547 /* falcona0,falconb0=pci_f0_config */

549 #define PCR_CC_VPD_CAP_DATA_REG 0x000000d4
550 /* sienaa0=pci_f0_config */

552 #define PCRF_AZ_VPD_DATA_LBN 0
553 #define PCRF_AZ_VPD_DATA_WIDTH 32


556 /*
557  * PC_MSI_ADR_HI_REG(32bit):
558  * MSI high 32 bits address register
559  */

561 #define PCR_AC_MSI_ADR_HI_REG 0x00000058
562 /* falcona0,falconb0,sienaa0=pci_f0_config */

564 #define PCR_DZ_MSI_ADR_HI_REG 0x00000098
565 /* hunta0=pci_f0_config */

567 #define PCRF_AZ_MSI_ADR_HI_LBN 0
568 #define PCRF_AZ_MSI_ADR_HI_WIDTH 32


571 /*
572  * PC_MSI_DAT_REG(16bit):
573  * MSI data register
574  */

576 #define PCR_AC_MSI_DAT_REG 0x0000005c
577 /* falcona0,falconb0,sienaa0=pci_f0_config */

579 #define PCR_DZ_MSI_DAT_REG 0x0000009c
580 /* hunta0=pci_f0_config */

582 #define PCRF_AZ_MSI_DAT_LBN 0
583 #define PCRF_AZ_MSI_DAT_WIDTH 16


586 /*
587  * PC_PCIE_CAP_LIST_REG(16bit):
588  * PCIe capability list register
589  */
```



```
591 #define PCR_AB_PCIE_CAP_LIST_REG 0x00000060
592 /* falcona0,falconb0=pci_f0_config */

594 #define PCR_CC_PCIE_CAP_LIST_REG 0x00000070
595 /* sienaa0=pci_f0_config */

597 #define PCR_DZ_PCIE_CAP_LIST_REG 0x000000c0
598 /* hunta0=pci_f0_config */

600 #define PCRF_AZ_PCIE_NXT_PTR_LBN 8
601 #define PCRF_AZ_PCIE_NXT_PTR_WIDTH 8
602 #define PCRF_AZ_PCIE_CAP_ID_LBN 0
603 #define PCRF_AZ_PCIE_CAP_ID_WIDTH 8


606 /*
607  * PC_PCIE_CAP_REG(16bit):
608  * PCIe capability register
609  */

611 #define PCR_AB_PCIE_CAP_REG 0x00000062
612 /* falcona0,falconb0=pci_f0_config */

614 #define PCR_CC_PCIE_CAP_REG 0x00000072
615 /* sienaa0=pci_f0_config */

617 #define PCR_DZ_PCIE_CAP_REG 0x000000c2
618 /* hunta0=pci_f0_config */

620 #define PCRF_AZ_PCIE_INT_MSG_NUM_LBN 9
621 #define PCRF_AZ_PCIE_INT_MSG_NUM_WIDTH 5
622 #define PCRF_AZ_PCIE_SLOT_IMP_LBN 8
623 #define PCRF_AZ_PCIE_SLOT_IMP_WIDTH 1
624 #define PCRF_AZ_PCIE_DEV_PORT_TYPE_LBN 4
625 #define PCRF_AZ_PCIE_DEV_PORT_TYPE_WIDTH 4
626 #define PCRF_AZ_PCIE_CAP_VER_LBN 0
627 #define PCRF_AZ_PCIE_CAP_VER_WIDTH 4


630 /*
631  * PC_DEV_CAP_REG(32bit):
632  * PCIe device capabilities register
633  */

635 #define PCR_AB_DEV_CAP_REG 0x00000064
636 /* falcona0,falconb0=pci_f0_config */

638 #define PCR_CC_DEV_CAP_REG 0x00000074
639 /* sienaa0=pci_f0_config */

641 #define PCR_DZ_DEV_CAP_REG 0x000000c4
642 /* hunta0=pci_f0_config */

644 #define PCRF_CZ_CAP_FN_LEVEL_RESET_LBN 28
645 #define PCRF_CZ_CAP_FN_LEVEL_RESET_WIDTH 1
646 #define PCRF_AZ_CAP_SLOT_PWR_SCL_LBN 26
647 #define PCRF_AZ_CAP_SLOT_PWR_SCL_WIDTH 2
648 #define PCRF_AZ_CAP_SLOT_PWR_VAL_LBN 18
649 #define PCRF_AZ_CAP_SLOT_PWR_VAL_WIDTH 8
650 #define PCRF_CZ_ROLE_BASE_ERR_REPORTING_LBN 15
651 #define PCRF_CZ_ROLE_BASE_ERR_REPORTING_WIDTH 1
652 #define PCRF_AB_PWR_IND_LBN 14
653 #define PCRF_AB_PWR_IND_WIDTH 1
654 #define PCRF_AB_ATTN_IND_LBN 13
655 #define PCRF_AB_ATTN_IND_WIDTH 1
```

```
#define PCRF_AB_ATTN_BUTTON_LBN 12
#define PCRF_AB_ATTN_BUTTON_WIDTH 1
#define PCRF_AZ_ENDPT_L1_LAT_LBN 9
#define PCRF_AZ_ENDPT_L1_LAT_WIDTH 3
#define PCRF_AZ_ENDPT_L0_LAT_LBN 6
#define PCRF_AZ_ENDPT_L0_LAT_WIDTH 3
#define PCRF_AZ_TAG_FIELD_LBN 5
#define PCRF_AZ_TAG_FIELD_WIDTH 1
#define PCRF_AZ_PHAN_FUNC_LBN 3
#define PCRF_AZ_PHAN_FUNC_WIDTH 2
#define PCRF_AZ_MAX_PAYL_SIZE_SUPT_LBN 0
#define PCRF_AZ_MAX_PAYL_SIZE_SUPT_WIDTH 3


/*
 * PC_DEV_CTL_REG(16bit):
 * PCIe device control register
 */

#define PCR_AB_DEV_CTL_REG 0x00000068
/* falcona0,falconb0=pci_f0_config */

#define PCR_CC_DEV_CTL_REG 0x00000078
/* sienaa0=pci_f0_config */

#define PCR_DZ_DEV_CTL_REG 0x000000c8
/* hunta0=pci_f0_config */

#define PCRF_CZ_FN_LEVEL_RESET_LBN 15
#define PCRF_CZ_FN_LEVEL_RESET_WIDTH 1
#define PCRF_AZ_MAX_RD_REQ_SIZE_LBN 12
#define PCRF_AZ_MAX_RD_REQ_SIZE_WIDTH 3
#define PCFE_AZ_MAX_RD_REQ_SIZE_4096 5
#define PCFE_AZ_MAX_RD_REQ_SIZE_2048 4
#define PCFE_AZ_MAX_RD_REQ_SIZE_1024 3
#define PCFE_AZ_MAX_RD_REQ_SIZE_512 2
#define PCFE_AZ_MAX_RD_REQ_SIZE_256 1
#define PCFE_AZ_MAX_RD_REQ_SIZE_128 0
#define PCFE_DZ_OTHER other
#define PCRF_AZ_EN_NO_SNOOP_LBN 11
#define PCRF_AZ_EN_NO_SNOOP_WIDTH 1
#define PCRF_AZ_AUX_PWR_PM_EN_LBN 10
#define PCRF_AZ_AUX_PWR_PM_EN_WIDTH 1
#define PCRF_AZ_PHAN_FUNC_EN_LBN 9
#define PCRF_AZ_PHAN_FUNC_EN_WIDTH 1
#define PCRF_AB_DEV_CAP_REG_RSVD0_LBN 8
#define PCRF_AB_DEV_CAP_REG_RSVD0_WIDTH 1
#define PCRF_CZ_EXTENDED_TAG_EN_LBN 8
#define PCRF_CZ_EXTENDED_TAG_EN_WIDTH 1
#define PCRF_AZ_MAX_PAYL_SIZE_LBN 5
#define PCRF_AZ_MAX_PAYL_SIZE_WIDTH 3
#define PCFE_AZ_MAX_PAYL_SIZE_4096 5
#define PCFE_AZ_MAX_PAYL_SIZE_2048 4
#define PCFE_AZ_MAX_PAYL_SIZE_1024 3
#define PCFE_AZ_MAX_PAYL_SIZE_512 2
#define PCFE_AZ_MAX_PAYL_SIZE_256 1
#define PCFE_AZ_MAX_PAYL_SIZE_128 0
#define PCFE_DZ_OTHER other
#define PCRF_AZ_EN_RELAX_ORDER_LBN 4
#define PCRF_AZ_EN_RELAX_ORDER_WIDTH 1
#define PCRF_AZ_UNSUP_REQ_RPT_EN_LBN 3
#define PCRF_AZ_UNSUP_REQ_RPT_EN_WIDTH 1
#define PCRF_AZ_FATAL_ERR_RPT_EN_LBN 2
#define PCRF_AZ_FATAL_ERR_RPT_EN_WIDTH 1
#define PCRF_AZ_NONFATAL_ERR_RPT_EN_LBN 1
#define PCRF_AZ_NONFATAL_ERR_RPT_EN_WIDTH 1
```



```
#define PCRF_AZ_CORR_ERR_RPT_EN_LBN 0
#define PCRF_AZ_CORR_ERR_RPT_EN_WIDTH 1


/*
 * PC_DEV_STAT_REG(16bit):
 * PCIe device status register
 */

#define PCR_AB_DEV_STAT_REG 0x0000006a
/* falcona0,falconb0=pci_f0_config */

#define PCR_CC_DEV_STAT_REG 0x0000007a
/* sienaa0=pci_f0_config */

#define PCR_DZ_DEV_STAT_REG 0x000000ca
/* hunta0=pci_f0_config */

#define PCRF_AZ_TRNS_PEND_LBN 5
#define PCRF_AZ_TRNS_PEND_WIDTH 1
#define PCRF_AZ_AUX_PWR_DET_LBN 4
#define PCRF_AZ_AUX_PWR_DET_WIDTH 1
#define PCRF_AZ_UNSUP_REQ_DET_LBN 3
#define PCRF_AZ_UNSUP_REQ_DET_WIDTH 1
#define PCRF_AZ_FATAL_ERR_DET_LBN 2
#define PCRF_AZ_FATAL_ERR_DET_WIDTH 1
#define PCRF_AZ_NONFATAL_ERR_DET_LBN 1
#define PCRF_AZ_NONFATAL_ERR_DET_WIDTH 1
#define PCRF_AZ_CORR_ERR_DET_LBN 0
#define PCRF_AZ_CORR_ERR_DET_WIDTH 1


/*
 * PC_LNK_CAP_REG(32bit):
 * PCIe link capabilities register
 */

#define PCR_AB_LNK_CAP_REG 0x0000006c
/* falcona0,falconb0=pci_f0_config */

#define PCR_CC_LNK_CAP_REG 0x0000007c
/* sienaa0=pci_f0_config */

#define PCR_DZ_LNK_CAP_REG 0x000000cc
/* hunta0=pci_f0_config */

#define PCRF_AZ_PORT_NUM_LBN 24
#define PCRF_AZ_PORT_NUM_WIDTH 8
#define PCRF_CZ_LINK_BWDITH_NOTIF_CAP_LBN 21
#define PCRF_CZ_LINK_BWDITH_NOTIF_CAP_WIDTH 1
#define PCRF_CZ_DATA_LINK_ACTIVE_RPT_CAP_LBN 20
#define PCRF_CZ_DATA_LINK_ACTIVE_RPT_CAP_WIDTH 1
#define PCRF_CZ_SURPISE_DOWN_RPT_CAP_LBN 19
#define PCRF_CZ_SURPISE_DOWN_RPT_CAP_WIDTH 1
#define PCRF_CZ_CLOCK_PWR_MNGMNT_CAP_LBN 18
#define PCRF_CZ_CLOCK_PWR_MNGMNT_CAP_WIDTH 1
#define PCRF_AZ_DEF_L1_EXIT_LAT_LBN 15
#define PCRF_AZ_DEF_L1_EXIT_LAT_WIDTH 3
#define PCRF_AZ_DEF_L0_EXIT_LATPORT_NUM_LBN 12
#define PCRF_AZ_DEF_L0_EXIT_LATPORT_NUM_WIDTH 3
#define PCRF_AZ_AS_LNK_PM_SUPT_LBN 10
#define PCRF_AZ_AS_LNK_PM_SUPT_WIDTH 2
#define PCRF_AZ_MAX_LNK_WIDTH_LBN 4
#define PCRF_AZ_MAX_LNK_WIDTH_WIDTH 6
#define PCRF_AZ_MAX_LNK_SP_LBN 0
#define PCRF_AZ_MAX_LNK_SP_WIDTH 4
```

```
790 /*
791  * PC_LNK_CTL_REG(16bit):
792  * PCIe link control register
793  */

795 #define PCR_AB_LNK_CTL_REG 0x00000070
796 /* falcona0,falconb0=pci_f0_config */

798 #define PCR_CC_LNK_CTL_REG 0x00000080
799 /* sienaa0=pci_f0_config */

801 #define PCR_DZ_LNK_CTL_REG 0x000000d0
802 /* hunta0=pci_f0_config */

804 #define PCRF_AZ_EXT_SYNC_LBN 7
805 #define PCRF_AZ_EXT_SYNC_WIDTH 1
806 #define PCRF_AZ_COMM_CLK_CFG_LBN 6
807 #define PCRF_AZ_COMM_CLK_CFG_WIDTH 1
808 #define PCRF_AB_LNK_CTL_REG_RSVD0_LBN 5
809 #define PCRF_AB_LNK_CTL_REG_RSVD0_WIDTH 1
810 #define PCRF_CZ_LNK_RETRAIN_LBN 5
811 #define PCRF_CZ_LNK_RETRAIN_WIDTH 1
812 #define PCRF_AZ_LNK_DIS_LBN 4
813 #define PCRF_AZ_LNK_DIS_WIDTH 1
814 #define PCRF_AZ_RD_COM_BDRY_LBN 3
815 #define PCRF_AZ_RD_COM_BDRY_WIDTH 1
816 #define PCRF_AZ_ACT_ST_LNK_PM_CTL_LBN 0
817 #define PCRF_AZ_ACT_ST_LNK_PM_CTL_WIDTH 2


820 /*
821  * PC_LNK_STAT_REG(16bit):
822  * PCIe link status register
823  */

825 #define PCR_AB_LNK_STAT_REG 0x00000072
826 /* falcona0,falconb0=pci_f0_config */

828 #define PCR_CC_LNK_STAT_REG 0x00000082
829 /* sienaa0=pci_f0_config */

831 #define PCR_DZ_LNK_STAT_REG 0x000000d2
832 /* hunta0=pci_f0_config */

834 #define PCRF_AZ_SLOT_CLK_CFG_LBN 12
835 #define PCRF_AZ_SLOT_CLK_CFG_WIDTH 1
836 #define PCRF_AZ_LNK_TRAIN_LBN 11
837 #define PCRF_AZ_LNK_TRAIN_WIDTH 1
838 #define PCRF_AB_TRAIN_ERR_LBN 10
839 #define PCRF_AB_TRAIN_ERR_WIDTH 1
840 #define PCRF_AZ_LNK_WIDTH_LBN 4
841 #define PCRF_AZ_LNK_WIDTH_WIDTH 6
842 #define PCRF_AZ_LNK_SP_LBN 0
843 #define PCRF_AZ_LNK_SP_WIDTH 4


846 /*
847  * PC_SLOT_CAP_REG(32bit):
848  * PCIe slot capabilities register
849  */

851 #define PCR_AB_SLOT_CAP_REG 0x00000074
852 /* falcona0,falconb0=pci_f0_config */
```



```
854 #define PCRF_AB_SLOT_NUM_LBN 19
855 #define PCRF_AB_SLOT_NUM_WIDTH 13
856 #define PCRF_AB_SLOT_PWR_LIM_SCL_LBN 15
857 #define PCRF_AB_SLOT_PWR_LIM_SCL_WIDTH 2
858 #define PCRF_AB_SLOT_PWR_LIM_VAL_LBN 7
859 #define PCRF_AB_SLOT_PWR_LIM_VAL_WIDTH 8
860 #define PCRF_AB_SLOT_HP_CAP_LBN 6
861 #define PCRF_AB_SLOT_HP_CAP_WIDTH 1
862 #define PCRF_AB_SLOT_HP_SURP_LBN 5
863 #define PCRF_AB_SLOT_HP_SURP_WIDTH 1
864 #define PCRF_AB_SLOT_PWR_IND_PRST_LBN 4
865 #define PCRF_AB_SLOT_PWR_IND_PRST_WIDTH 1
866 #define PCRF_AB_SLOT_ATTN_IND_PRST_LBN 3
867 #define PCRF_AB_SLOT_ATTN_IND_PRST_WIDTH 1
868 #define PCRF_AB_SLOT_MRL_SENS_PRST_LBN 2
869 #define PCRF_AB_SLOT_MRL_SENS_PRST_WIDTH 1
870 #define PCRF_AB_SLOT_PWR_CTL_PRST_LBN 1
871 #define PCRF_AB_SLOT_PWR_CTL_PRST_WIDTH 1
872 #define PCRF_AB_SLOT_ATTN_BUT_PRST_LBN 0
873 #define PCRF_AB_SLOT_ATTN_BUT_PRST_WIDTH 1


876 /*
877  * PC_SLOT_CTL_REG(16bit):
878  * PCIe slot control register
879  */

881 #define PCR_AB_SLOT_CTL_REG 0x00000078
882 /* falcona0,falconb0=pci_f0_config */

884 #define PCRF_AB_SLOT_PWR_CTLR_CTL_LBN 10
885 #define PCRF_AB_SLOT_PWR_CTLR_CTL_WIDTH 1
886 #define PCRF_AB_SLOT_PWR_IND_CTL_LBN 8
887 #define PCRF_AB_SLOT_PWR_IND_CTL_WIDTH 2
888 #define PCRF_AB_SLOT_ATT_IND_CTL_LBN 6
889 #define PCRF_AB_SLOT_ATT_IND_CTL_WIDTH 2
890 #define PCRF_AB_SLOT_HP_INT_EN_LBN 5
891 #define PCRF_AB_SLOT_HP_INT_EN_WIDTH 1
892 #define PCRF_AB_SLOT_CMD_COMP_INT_EN_LBN 4
893 #define PCRF_AB_SLOT_CMD_COMP_INT_EN_WIDTH 1
894 #define PCRF_AB_SLOT_PRES_DET_CHG_EN_LBN 3
895 #define PCRF_AB_SLOT_PRES_DET_CHG_EN_WIDTH 1
896 #define PCRF_AB_SLOT_MRL_SENS_CHG_EN_LBN 2
897 #define PCRF_AB_SLOT_MRL_SENS_CHG_EN_WIDTH 1
898 #define PCRF_AB_SLOT_PWR_FLTDET_EN_LBN 1
899 #define PCRF_AB_SLOT_PWR_FLTDET_EN_WIDTH 1
900 #define PCRF_AB_SLOT_ATTN_BUT_EN_LBN 0
901 #define PCRF_AB_SLOT_ATTN_BUT_EN_WIDTH 1


904 /*
905  * PC_SLOT_STAT_REG(16bit):
906  * PCIe slot status register
907  */

909 #define PCR_AB_SLOT_STAT_REG 0x0000007a
910 /* falcona0,falconb0=pci_f0_config */

912 #define PCRF_AB_PRES_DET_ST_LBN 6
913 #define PCRF_AB_PRES_DET_ST_WIDTH 1
914 #define PCRF_AB_MRL_SENS_ST_LBN 5
915 #define PCRF_AB_MRL_SENS_ST_WIDTH 1
916 #define PCRF_AB_SLOT_PWR_IND_LBN 4
917 #define PCRF_AB_SLOT_PWR_IND_WIDTH 1
918 #define PCRF_AB_SLOT_ATTN_IND_LBN 3
919 #define PCRF_AB_SLOT_ATTN_IND_WIDTH 1
```

```
#define PCRF_AB_SLOT_MRL_SENS_LBN 2
#define PCRF_AB_SLOT_MRL_SENS_WIDTH 1
#define PCRF_AB_PWR_FLTDET_LBN 1
#define PCRF_AB_PWR_FLTDET_WIDTH 1
#define PCRF_AB_ATTN_BUTDET_LBN 0
#define PCRF_AB_ATTN_BUTDET_WIDTH 1


/*
 * PC_MSIX_CAP_ID_REG(8bit):
 * MSIX Capability ID
 */

#define PCR_BB_MSIX_CAP_ID_REG 0x00000090
/* falconb0=pci_f0_config */

#define PCR_CZ_MSIX_CAP_ID_REG 0x000000b0
/* sienaa0,hunta0=pci_f0_config */

#define PCRF_BZ_MSIX_CAP_ID_LBN 0
#define PCRF_BZ_MSIX_CAP_ID_WIDTH 8


/*
 * PC_MSIX_NXT_PTR_REG(8bit):
 * MSIX Capability Next Capability Ptr
 */

#define PCR_BB_MSIX_NXT_PTR_REG 0x00000091
/* falconb0=pci_f0_config */

#define PCR_CZ_MSIX_NXT_PTR_REG 0x000000b1
/* sienaa0,hunta0=pci_f0_config */

#define PCRF_BZ_MSIX_NXT_PTR_LBN 0
#define PCRF_BZ_MSIX_NXT_PTR_WIDTH 8


/*
 * PC_MSIX_CTL_REG(16bit):
 * MSIX control register
 */

#define PCR_BB_MSIX_CTL_REG 0x00000092
/* falconb0=pci_f0_config */

#define PCR_CZ_MSIX_CTL_REG 0x000000b2
/* sienaa0,hunta0=pci_f0_config */

#define PCRF_BZ_MSIX_EN_LBN 15
#define PCRF_BZ_MSIX_EN_WIDTH 1
#define PCRF_BZ_MSIX_FUNC_MASK_LBN 14
#define PCRF_BZ_MSIX_FUNC_MASK_WIDTH 1
#define PCRF_BZ_MSIX_TBL_SIZE_LBN 0
#define PCRF_BZ_MSIX_TBL_SIZE_WIDTH 11


/*
 * PC_DEV_CAP2_REG(16bit):
 * PCIe Device Capabilities 2
 */

#define PCR_CC_DEV_CAP2_REG 0x00000094
/* sienaa0=pci_f0_config */

#define PCR_DZ_DEV_CAP2_REG 0x000000e4
```



```
/* hunta0=pci_f0_config */

#define PCRF_CZ_CMPL_TIMEOUT_DIS_LBN 4
#define PCRF_CZ_CMPL_TIMEOUT_DIS_WIDTH 1
#define PCRF_CZ_CMPL_TIMEOUT_LBN 0
#define PCRF_CZ_CMPL_TIMEOUT_WIDTH 4
#define PCFE_CZ_CMPL_TIMEOUT_17000_TO_6400MS 14
#define PCFE_CZ_CMPL_TIMEOUT_4000_TO_1300MS 13
#define PCFE_CZ_CMPL_TIMEOUT_1000_TO_3500MS 10
#define PCFE_CZ_CMPL_TIMEOUT_260_TO_900MS 9
#define PCFE_CZ_CMPL_TIMEOUT_65_TO_210MS 6
#define PCFE_CZ_CMPL_TIMEOUT_16_TO_55MS 5
#define PCFE_CZ_CMPL_TIMEOUT_1_TO_10MS 2
#define PCFE_CZ_CMPL_TIMEOUT_50_TO_100US 1
#define PCFE_CZ_CMPL_TIMEOUT_DEFAULT 0


/*
 * PC_MSIX_TBL_BASE_REG(32bit):
 * MSIX Capability Vector Table Base
 */

#define PCR_BB_MSIX_TBL_BASE_REG 0x00000094
/* falconb0=pci_f0_config */

#define PCR_CZ_MSIX_TBL_BASE_REG 0x000000b4
/* sienaa0,hunta0=pci_f0_config */

#define PCRF_BZ_MSIX_TBL_OFF_LBN 3
#define PCRF_BZ_MSIX_TBL_OFF_WIDTH 29
#define PCRF_BZ_MSIX_TBL_BIR_LBN 0
#define PCRF_BZ_MSIX_TBL_BIR_WIDTH 3


/*
 * PC_DEV_CTL2_REG(16bit):
 * PCIe Device Control 2
 */

#define PCR_CC_DEV_CTL2_REG 0x00000098
/* sienaa0=pci_f0_config */

#define PCR_DZ_DEV_CTL2_REG 0x000000e8
/* hunta0=pci_f0_config */

#define PCRF_CZ_CMPL_TIMEOUT_DIS_CTL_LBN 4
#define PCRF_CZ_CMPL_TIMEOUT_DIS_CTL_WIDTH 1
#define PCRF_CZ_CMPL_TIMEOUT_CTL_LBN 0
#define PCRF_CZ_CMPL_TIMEOUT_CTL_WIDTH 4


/*
 * PC_MSIX_PBA_BASE_REG(32bit):
 * MSIX Capability PBA Base
 */

#define PCR_BB_MSIX_PBA_BASE_REG 0x00000098
/* falconb0=pci_f0_config */

#define PCR_CZ_MSIX_PBA_BASE_REG 0x000000b8
/* sienaa0,hunta0=pci_f0_config */

#define PCRF_BZ_MSIX_PBA_OFF_LBN 3
#define PCRF_BZ_MSIX_PBA_OFF_WIDTH 29
#define PCRF_BZ_MSIX_PBA_BIR_LBN 0
#define PCRF_BZ_MSIX_PBA_BIR_WIDTH 3
```

```
1054 /*
1055  * PC_LNK_CTL2_REG(16bit):
1056  * PCIe Link Control 2
1057  */

1059 #define PCR_CC_LNK_CTL2_REG 0x000000a0
1060 /* sienaa0=pci_f0_config */

1062 #define PCR_DZ_LNK_CTL2_REG 0x000000f0
1063 /* hunta0=pci_f0_config */

1065 #define PCRF_CZ_POLLING_DEEMPH_LVL_LBN 12
1066 #define PCRF_CZ_POLLING_DEEMPH_LVL_WIDTH 1
1067 #define PCRF_CZ_COMPLIANCE_SOS_CTL_LBN 11
1068 #define PCRF_CZ_COMPLIANCE_SOS_CTL_WIDTH 1
1069 #define PCRF_CZ_ENTER_MODIFIED_COMPLIANCE_CTL_LBN 10
1070 #define PCRF_CZ_ENTER_MODIFIED_COMPLIANCE_CTL_WIDTH 1
1071 #define PCRF_CZ_TRANSMIT_MARGIN_LBN 7
1072 #define PCRF_CZ_TRANSMIT_MARGIN_WIDTH 3
1073 #define PCRF_CZ_SELECT_DEEMPH_LBN 6
1074 #define PCRF_CZ_SELECT_DEEMPH_WIDTH 1
1075 #define PCRF_CZ_HW_AUTONOMOUS_SPEED_DIS_LBN 5
1076 #define PCRF_CZ_HW_AUTONOMOUS_SPEED_DIS_WIDTH 1
1077 #define PCRF_CZ_ENTER_COMPLIANCE_CTL_LBN 4
1078 #define PCRF_CZ_ENTER_COMPLIANCE_CTL_WIDTH 1
1079 #define PCRF_CZ_TGT_LNK_SPEED_CTL_LBN 0
1080 #define PCRF_CZ_TGT_LNK_SPEED_CTL_WIDTH 4


1083 /*
1084  * PC_LNK_STAT2_REG(16bit):
1085  * PCIe Link Status 2
1086  */

1088 #define PCR_CC_LNK_STAT2_REG 0x000000a2
1089 /* sienaa0=pci_f0_config */

1091 #define PCR_DZ_LNK_STAT2_REG 0x000000f2
1092 /* hunta0=pci_f0_config */

1094 #define PCRF_CZ_CURRENT_DEEMPH_LBN 0
1095 #define PCRF_CZ_CURRENT_DEEMPH_WIDTH 1


1098 /*
1099  * PC_VPD_CAP_ID_REG(8bit):
1100  * VPD data register
1101  */

1103 #define PCR_AB_VPD_CAP_ID_REG 0x000000b0
1104 /* falcona0,falconb0=pci_f0_config */

1106 #define PCRF_AB_VPD_CAP_ID_LBN 0
1107 #define PCRF_AB_VPD_CAP_ID_WIDTH 8


1110 /*
1111  * PC_VPD_NXT_PTR_REG(8bit):
1112  * VPD next item pointer
1113  */

1115 #define PCR_AB_VPD_NXT_PTR_REG 0x000000b1
1116 /* falcona0,falconb0=pci_f0_config */
```



```
1118 #define PCRF_AB_VPD_NXT_PTR_LBN 0
1119 #define PCRF_AB_VPD_NXT_PTR_WIDTH 8


1122 /*
1123  * PC_VPD_ADDR_REG(16bit):
1124  * VPD address register
1125  */

1127 #define PCR_AB_VPD_ADDR_REG 0x000000b2
1128 /* falcona0,falconb0=pci_f0_config */

1130 #define PCRF_AB_VPD_FLAG_LBN 15
1131 #define PCRF_AB_VPD_FLAG_WIDTH 1
1132 #define PCRF_AB_VPD_ADDR_LBN 0
1133 #define PCRF_AB_VPD_ADDR_WIDTH 15


1136 /*
1137  * PC_AER_CAP_HDR_REG(32bit):
1138  * AER capability header register
1139  */

1141 #define PCR_AZ_AER_CAP_HDR_REG 0x00000100
1142 /* falcona0,falconb0,sienaa0,hunta0=pci_f0_config */

1144 #define PCRF_AZ_AERCAPHDR_NXT_PTR_LBN 20
1145 #define PCRF_AZ_AERCAPHDR_NXT_PTR_WIDTH 12
1146 #define PCRF_AZ_AERCAPHDR_VER_LBN 16
1147 #define PCRF_AZ_AERCAPHDR_VER_WIDTH 4
1148 #define PCRF_AZ_AERCAPHDR_ID_LBN 0
1149 #define PCRF_AZ_AERCAPHDR_ID_WIDTH 16


1152 /*
1153  * PC_AER_UNCORR_ERR_STAT_REG(32bit):
1154  * AER Uncorrectable error status register
1155  */

1157 #define PCR_AZ_AER_UNCORR_ERR_STAT_REG 0x00000104
1158 /* falcona0,falconb0,sienaa0,hunta0=pci_f0_config */

1160 #define PCRF_AZ_UNSUPT_REQ_ERR_STAT_LBN 20
1161 #define PCRF_AZ_UNSUPT_REQ_ERR_STAT_WIDTH 1
1162 #define PCRF_AZ_ECRC_ERR_STAT_LBN 19
1163 #define PCRF_AZ_ECRC_ERR_STAT_WIDTH 1
1164 #define PCRF_AZ_MALF_TLP_STAT_LBN 18
1165 #define PCRF_AZ_MALF_TLP_STAT_WIDTH 1
1166 #define PCRF_AZ_RX_OVF_STAT_LBN 17
1167 #define PCRF_AZ_RX_OVF_STAT_WIDTH 1
1168 #define PCRF_AZ_UNEXP_COMP_STAT_LBN 16
1169 #define PCRF_AZ_UNEXP_COMP_STAT_WIDTH 1
1170 #define PCRF_AZ_COMP_ABRT_STAT_LBN 15
1171 #define PCRF_AZ_COMP_ABRT_STAT_WIDTH 1
1172 #define PCRF_AZ_COMP_TIMEOUT_STAT_LBN 14
1173 #define PCRF_AZ_COMP_TIMEOUT_STAT_WIDTH 1
1174 #define PCRF_AZ_FC_PROTO_ERR_STAT_LBN 13
1175 #define PCRF_AZ_FC_PROTO_ERR_STAT_WIDTH 1
1176 #define PCRF_AZ_PSON_TLP_STAT_LBN 12
1177 #define PCRF_AZ_PSON_TLP_STAT_WIDTH 1
1178 #define PCRF_AZ_DL_PROTO_ERR_STAT_LBN 4
1179 #define PCRF_AZ_DL_PROTO_ERR_STAT_WIDTH 1
1180 #define PCRF_AB_TRAIN_ERR_STAT_LBN 0
1181 #define PCRF_AB_TRAIN_ERR_STAT_WIDTH 1
```

```
1184 /*
1185  * PC_AER_UNCORR_ERR_MASK_REG(32bit):
1186  * AER Uncorrectable error mask register
1187  */

1189 #define PCR_AZ_AER_UNCORR_ERR_MASK_REG 0x00000108
1190 /* falcona0,falconb0,sienaa0,hunta0=pci_f0_config */

1192 #define PCRF_AZ_UNSUPT_REQ_ERR_MASK_LBN 20
1193 #define PCRF_AZ_UNSUPT_REQ_ERR_MASK_WIDTH 1
1194 #define PCRF_AZ_ECRC_ERR_MASK_LBN 19
1195 #define PCRF_AZ_ECRC_ERR_MASK_WIDTH 1
1196 #define PCRF_AZ_MALF_TLP_MASK_LBN 18
1197 #define PCRF_AZ_MALF_TLP_MASK_WIDTH 1
1198 #define PCRF_AZ_RX_OVF_MASK_LBN 17
1199 #define PCRF_AZ_RX_OVF_MASK_WIDTH 1
1200 #define PCRF_AZ_UNEXP_COMP_MASK_LBN 16
1201 #define PCRF_AZ_UNEXP_COMP_MASK_WIDTH 1
1202 #define PCRF_AZ_COMP_ABRT_MASK_LBN 15
1203 #define PCRF_AZ_COMP_ABRT_MASK_WIDTH 1
1204 #define PCRF_AZ_COMP_TIMEOUT_MASK_LBN 14
1205 #define PCRF_AZ_COMP_TIMEOUT_MASK_WIDTH 1
1206 #define PCRF_AZ_FC_PROTO_ERR_MASK_LBN 13
1207 #define PCRF_AZ_FC_PROTO_ERR_MASK_WIDTH 1
1208 #define PCRF_AZ_PSON_TLP_MASK_LBN 12
1209 #define PCRF_AZ_PSON_TLP_MASK_WIDTH 1
1210 #define PCRF_AZ_DL_PROTO_ERR_MASK_LBN 4
1211 #define PCRF_AZ_DL_PROTO_ERR_MASK_WIDTH 1
1212 #define PCRF_AB_TRAIN_ERR_MASK_LBN 0
1213 #define PCRF_AB_TRAIN_ERR_MASK_WIDTH 1


1216 /*
1217  * PC_AER_UNCORR_ERR_SEV_REG(32bit):
1218  * AER Uncorrectable error severity register
1219  */

1221 #define PCR_AZ_AER_UNCORR_ERR_SEV_REG 0x0000010c
1222 /* falcona0,falconb0,sienaa0,hunta0=pci_f0_config */

1224 #define PCRF_AZ_UNSUPT_REQ_ERR_SEV_LBN 20
1225 #define PCRF_AZ_UNSUPT_REQ_ERR_SEV_WIDTH 1
1226 #define PCRF_AZ_ECRC_ERR_SEV_LBN 19
1227 #define PCRF_AZ_ECRC_ERR_SEV_WIDTH 1
1228 #define PCRF_AZ_MALF_TLP_SEV_LBN 18
1229 #define PCRF_AZ_MALF_TLP_SEV_WIDTH 1
1230 #define PCRF_AZ_RX_OVF_SEV_LBN 17
1231 #define PCRF_AZ_RX_OVF_SEV_WIDTH 1
1232 #define PCRF_AZ_UNEXP_COMP_SEV_LBN 16
1233 #define PCRF_AZ_UNEXP_COMP_SEV_WIDTH 1
1234 #define PCRF_AZ_COMP_ABRT_SEV_LBN 15
1235 #define PCRF_AZ_COMP_ABRT_SEV_WIDTH 1
1236 #define PCRF_AZ_COMP_TIMEOUT_SEV_LBN 14
1237 #define PCRF_AZ_COMP_TIMEOUT_SEV_WIDTH 1
1238 #define PCRF_AZ_FC_PROTO_ERR_SEV_LBN 13
1239 #define PCRF_AZ_FC_PROTO_ERR_SEV_WIDTH 1
1240 #define PCRF_AZ_PSON_TLP_SEV_LBN 12
1241 #define PCRF_AZ_PSON_TLP_SEV_WIDTH 1
1242 #define PCRF_AZ_DL_PROTO_ERR_SEV_LBN 4
1243 #define PCRF_AZ_DL_PROTO_ERR_SEV_WIDTH 1
1244 #define PCRF_AB_TRAIN_ERR_SEV_LBN 0
1245 #define PCRF_AB_TRAIN_ERR_SEV_WIDTH 1


1248 /*
1249  * PC_AER_CORR_ERR_STAT_REG(32bit):
```



```
1250  * AER Correctable error status register
1251  */

1253 #define PCR_AZ_AER_CORR_ERR_STAT_REG 0x00000110
1254 /* falcona0,falconb0,sienaa0,hunta0=pci_f0_config */

1256 #define PCRF_CZ_ADVSY_NON_FATAL_STAT_LBN 13
1257 #define PCRF_CZ_ADVSY_NON_FATAL_STAT_WIDTH 1
1258 #define PCRF_AZ_RPLY_TMR_TOUT_STAT_LBN 12
1259 #define PCRF_AZ_RPLY_TMR_TOUT_STAT_WIDTH 1
1260 #define PCRF_AZ_RPLAY_NUM_RO_STAT_LBN 8
1261 #define PCRF_AZ_RPLAY_NUM_RO_STAT_WIDTH 1
1262 #define PCRF_AZ_BAD_DLLP_STAT_LBN 7
1263 #define PCRF_AZ_BAD_DLLP_STAT_WIDTH 1
1264 #define PCRF_AZ_BAD_TLP_STAT_LBN 6
1265 #define PCRF_AZ_BAD_TLP_STAT_WIDTH 1
1266 #define PCRF_AZ_RX_ERR_STAT_LBN 0
1267 #define PCRF_AZ_RX_ERR_STAT_WIDTH 1


1270 /*
1271  * PC_AER_CORR_ERR_MASK_REG(32bit):
1272  * AER Correctable error status register
1273  */

1275 #define PCR_AZ_AER_CORR_ERR_MASK_REG 0x00000114
1276 /* falcona0,falconb0,sienaa0,hunta0=pci_f0_config */

1278 #define PCRF_CZ_ADVSY_NON_FATAL_MASK_LBN 13
1279 #define PCRF_CZ_ADVSY_NON_FATAL_MASK_WIDTH 1
1280 #define PCRF_AZ_RPLY_TMR_TOUT_MASK_LBN 12
1281 #define PCRF_AZ_RPLY_TMR_TOUT_MASK_WIDTH 1
1282 #define PCRF_AZ_RPLAY_NUM_RO_MASK_LBN 8
1283 #define PCRF_AZ_RPLAY_NUM_RO_MASK_WIDTH 1
1284 #define PCRF_AZ_BAD_DLLP_MASK_LBN 7
1285 #define PCRF_AZ_BAD_DLLP_MASK_WIDTH 1
1286 #define PCRF_AZ_BAD_TLP_MASK_LBN 6
1287 #define PCRF_AZ_BAD_TLP_MASK_WIDTH 1
1288 #define PCRF_AZ_RX_ERR_MASK_LBN 0
1289 #define PCRF_AZ_RX_ERR_MASK_WIDTH 1


1292 /*
1293  * PC_AER_CAP_CTL_REG(32bit):
1294  * AER capability and control register
1295  */

1297 #define PCR_AZ_AER_CAP_CTL_REG 0x00000118
1298 /* falcona0,falconb0,sienaa0,hunta0=pci_f0_config */

1300 #define PCRF_AZ_ECRC_CHK_EN_LBN 8
1301 #define PCRF_AZ_ECRC_CHK_EN_WIDTH 1
1302 #define PCRF_AZ_ECRC_CHK_CAP_LBN 7
1303 #define PCRF_AZ_ECRC_CHK_CAP_WIDTH 1
1304 #define PCRF_AZ_ECRC_GEN_EN_LBN 6
1305 #define PCRF_AZ_ECRC_GEN_EN_WIDTH 1
1306 #define PCRF_AZ_ECRC_GEN_CAP_LBN 5
1307 #define PCRF_AZ_ECRC_GEN_CAP_WIDTH 1
1308 #define PCRF_AZ_1ST_ERR_PTR_LBN 0
1309 #define PCRF_AZ_1ST_ERR_PTR_WIDTH 5


1312 /*
1313  * PC_AER_HDR_LOG_REG(128bit):
1314  * AER Header log register
1315  */
```

```
#define PCR_AZ_AER_HDR_LOG_REG 0x0000011c
/* falcona0,falconb0,sienaa0,hunta0=pci_f0_config */

#define PCRF_AZ_HDR_LOG_LBN 0
#define PCRF_AZ_HDR_LOG_WIDTH 128


/*
 * PC_DEVSN_CAP_HDR_REG(32bit):
 * Device serial number capability header register
 */

#define PCR_DZ_DEVSN_CAP_HDR_REG 0x00000130
/* hunta0=pci_f0_config */

#define PCR_CC_DEVSN_CAP_HDR_REG 0x00000140
/* sienaa0=pci_f0_config */

#define PCRF_CZ_DEVSNCAPHDR_NXT_PTR_LBN 20
#define PCRF_CZ_DEVSNCAPHDR_NXT_PTR_WIDTH 12
#define PCRF_CZ_DEVSNCAPHDR_VER_LBN 16
#define PCRF_CZ_DEVSNCAPHDR_VER_WIDTH 4
#define PCRF_CZ_DEVSNCAPHDR_ID_LBN 0
#define PCRF_CZ_DEVSNCAPHDR_ID_WIDTH 16


/*
 * PC_DEVSN_DWORD0_REG(32bit):
 * Device serial number DWORD0
 */

#define PCR_DZ_DEVSN_DWORD0_REG 0x00000134
/* hunta0=pci_f0_config */

#define PCR_CC_DEVSN_DWORD0_REG 0x00000144
/* sienaa0=pci_f0_config */

#define PCRF_CZ_DEVSN_DWORD0_LBN 0
#define PCRF_CZ_DEVSN_DWORD0_WIDTH 32


/*
 * PC_DEVSN_DWORD1_REG(32bit):
 * Device serial number DWORD0
 */

#define PCR_DZ_DEVSN_DWORD1_REG 0x00000138
/* hunta0=pci_f0_config */

#define PCR_CC_DEVSN_DWORD1_REG 0x00000148
/* sienaa0=pci_f0_config */

#define PCRF_CZ_DEVSN_DWORD1_LBN 0
#define PCRF_CZ_DEVSN_DWORD1_WIDTH 32


/*
 * PC_ARI_CAP_HDR_REG(32bit):
 * ARI capability header register
 */

#define PCR_DZ_ARI_CAP_HDR_REG 0x00000140
/* hunta0=pci_f0_config */

#define PCR_CC_ARI_CAP_HDR_REG 0x00000150
```



```
/* sienaa0=pci_f0_config */

#define PCRF_CZ_ARICAPHDR_NXT_PTR_LBN 20
#define PCRF_CZ_ARICAPHDR_NXT_PTR_WIDTH 12
#define PCRF_CZ_ARICAPHDR_VER_LBN 16
#define PCRF_CZ_ARICAPHDR_VER_WIDTH 4
#define PCRF_CZ_ARICAPHDR_ID_LBN 0
#define PCRF_CZ_ARICAPHDR_ID_WIDTH 16


/*
 * PC_ARI_CAP_REG(16bit):
 * ARI Capabilities
 */

#define PCR_DZ_ARI_CAP_REG 0x00000144
/* hunta0=pci_f0_config */

#define PCR_CC_ARI_CAP_REG 0x00000154
/* sienaa0=pci_f0_config */

#define PCRF_CZ_ARI_NXT_FN_NUM_LBN 8
#define PCRF_CZ_ARI_NXT_FN_NUM_WIDTH 8
#define PCRF_CZ_ARI_ACS_FNGRP_CAP_LBN 1
#define PCRF_CZ_ARI_ACS_FNGRP_CAP_WIDTH 1
#define PCRF_CZ_ARI_MFVC_FNGRP_CAP_LBN 0
#define PCRF_CZ_ARI_MFVC_FNGRP_CAP_WIDTH 1


/*
 * PC_ARI_CTL_REG(16bit):
 * ARI Control
 */

#define PCR_DZ_ARI_CTL_REG 0x00000146
/* hunta0=pci_f0_config */

#define PCR_CC_ARI_CTL_REG 0x00000156
/* sienaa0=pci_f0_config */

#define PCRF_CZ_ARI_FN_GRP_LBN 4
#define PCRF_CZ_ARI_FN_GRP_WIDTH 3
#define PCRF_CZ_ARI_ACS_FNGRP_EN_LBN 1
#define PCRF_CZ_ARI_ACS_FNGRP_EN_WIDTH 1
#define PCRF_CZ_ARI_MFVC_FNGRP_EN_LBN 0
#define PCRF_CZ_ARI_MFVC_FNGRP_EN_WIDTH 1


/*
 * PC_SRIOV_CAP_HDR_REG(32bit):
 * SRIOV capability header register
 */

#define PCR_CC_SRIOV_CAP_HDR_REG 0x00000160
/* sienaa0=pci_f0_config */

#define PCR_DZ_SRIOV_CAP_HDR_REG 0x00000200
/* hunta0=pci_f0_config */

#define PCRF_CZ_SRIOVCAPHDR_NXT_PTR_LBN 20
#define PCRF_CZ_SRIOVCAPHDR_NXT_PTR_WIDTH 12
#define PCRF_CZ_SRIOVCAPHDR_VER_LBN 16
#define PCRF_CZ_SRIOVCAPHDR_VER_WIDTH 4
#define PCRF_CZ_SRIOVCAPHDR_ID_LBN 0
#define PCRF_CZ_SRIOVCAPHDR_ID_WIDTH 16
```

```
1449 /*
1450  * PC_SRIOV_CAP_REG(32bit):
1451  * SRIOV Capabilities
1452  */

1454 #define PCR_CC_SRIOV_CAP_REG 0x00000164
1455 /* sienaa0=pci_f0_config */

1457 #define PCR_DZ_SRIOV_CAP_REG 0x00000204
1458 /* hunta0=pci_f0_config */

1460 #define PCRF_CZ_VF_MIGR_INT_MSG_NUM_LBN 21
1461 #define PCRF_CZ_VF_MIGR_INT_MSG_NUM_WIDTH 11
1462 #define PCRF_CZ_VF_MIGR_CAP_LBN 0
1463 #define PCRF_CZ_VF_MIGR_CAP_WIDTH 1


1466 /*
1467  * PC_SRIOV_CTL_REG(16bit):
1468  * SRIOV Control
1469  */

1471 #define PCR_CC_SRIOV_CTL_REG 0x00000168
1472 /* sienaa0=pci_f0_config */

1474 #define PCR_DZ_SRIOV_CTL_REG 0x00000208
1475 /* hunta0=pci_f0_config */

1477 #define PCRF_CZ_VF_ARI_CAP_HRCHY_LBN 4
1478 #define PCRF_CZ_VF_ARI_CAP_HRCHY_WIDTH 1
1479 #define PCRF_CZ_VF_MSE_LBN 3
1480 #define PCRF_CZ_VF_MSE_WIDTH 1
1481 #define PCRF_CZ_VF_MIGR_INT_EN_LBN 2
1482 #define PCRF_CZ_VF_MIGR_INT_EN_WIDTH 1
1483 #define PCRF_CZ_VF_MIGR_EN_LBN 1
1484 #define PCRF_CZ_VF_MIGR_EN_WIDTH 1
1485 #define PCRF_CZ_VF_EN_LBN 0
1486 #define PCRF_CZ_VF_EN_WIDTH 1


1489 /*
1490  * PC_SRIOV_STAT_REG(16bit):
1491  * SRIOV Status
1492  */

1494 #define PCR_CC_SRIOV_STAT_REG 0x0000016a
1495 /* sienaa0=pci_f0_config */

1497 #define PCR_DZ_SRIOV_STAT_REG 0x0000020a
1498 /* hunta0=pci_f0_config */

1500 #define PCRF_CZ_VF_MIGR_STAT_LBN 0
1501 #define PCRF_CZ_VF_MIGR_STAT_WIDTH 1


1504 /*
1505  * PC_SRIOV_INITIALVFS_REG(16bit):
1506  * SRIOV Initial VFs
1507  */

1509 #define PCR_CC_SRIOV_INITIALVFS_REG 0x0000016c
1510 /* sienaa0=pci_f0_config */

1512 #define PCR_DZ_SRIOV_INITIALVFS_REG 0x0000020c
1513 /* hunta0=pci_f0_config */
```



```
1515 #define PCRF_CZ_VF_INITIALVFS_LBN 0
1516 #define PCRF_CZ_VF_INITIALVFS_WIDTH 16


1519 /*
1520  * PC_SRIOV_TOTALVFS_REG(10bit):
1521  * SRIOV Total VFs
1522  */

1524 #define PCR_CC_SRIOV_TOTALVFS_REG 0x0000016e
1525 /* sienaa0=pci_f0_config */

1527 #define PCR_DZ_SRIOV_TOTALVFS_REG 0x0000020e
1528 /* hunta0=pci_f0_config */

1530 #define PCRF_CZ_VF_TOTALVFS_LBN 0
1531 #define PCRF_CZ_VF_TOTALVFS_WIDTH 16


1534 /*
1535  * PC_SRIOV_NUMVFS_REG(16bit):
1536  * SRIOV Number of VFs
1537  */

1539 #define PCR_CC_SRIOV_NUMVFS_REG 0x00000170
1540 /* sienaa0=pci_f0_config */

1542 #define PCR_DZ_SRIOV_NUMVFS_REG 0x00000210
1543 /* hunta0=pci_f0_config */

1545 #define PCRF_CZ_VF_NUMVFS_LBN 0
1546 #define PCRF_CZ_VF_NUMVFS_WIDTH 16


1549 /*
1550  * PC_SRIOV_FN_DPND_LNK_REG(16bit):
1551  * SRIOV Function dependency link
1552  */

1554 #define PCR_CC_SRIOV_FN_DPND_LNK_REG 0x00000172
1555 /* sienaa0=pci_f0_config */

1557 #define PCR_DZ_SRIOV_FN_DPND_LNK_REG 0x00000212
1558 /* hunta0=pci_f0_config */

1560 #define PCRF_CZ_SRIOV_FN_DPND_LNK_LBN 0
1561 #define PCRF_CZ_SRIOV_FN_DPND_LNK_WIDTH 8


1564 /*
1565  * PC_SRIOV_1STVF_OFFSET_REG(16bit):
1566  * SRIOV First VF Offset
1567  */

1569 #define PCR_CC_SRIOV_1STVF_OFFSET_REG 0x00000174
1570 /* sienaa0=pci_f0_config */

1572 #define PCR_DZ_SRIOV_1STVF_OFFSET_REG 0x00000214
1573 /* hunta0=pci_f0_config */

1575 #define PCRF_CZ_VF_1STVF_OFFSET_LBN 0
1576 #define PCRF_CZ_VF_1STVF_OFFSET_WIDTH 16


1579 /*
```

```
1580  * PC_SRIOV_VFSTRIDE_REG(16bit):
1581  * SRIOV VF Stride
1582  */

1584 #define PCR_CC_SRIOV_VFSTRIDE_REG 0x00000176
1585 /* sienaa0=pci_f0_config */

1587 #define PCR_DZ_SRIOV_VFSTRIDE_REG 0x00000216
1588 /* hunta0=pci_f0_config */

1590 #define PCRF_CZ_VF_VFSTRIDE_LBN 0
1591 #define PCRF_CZ_VF_VFSTRIDE_WIDTH 16


1594 /*
1595  * PC_SRIOV_DEVID_REG(16bit):
1596  * SRIOV VF Device ID
1597  */

1599 #define PCR_CC_SRIOV_DEVID_REG 0x0000017a
1600 /* sienaa0=pci_f0_config */

1602 #define PCR_DZ_SRIOV_DEVID_REG 0x0000021a
1603 /* hunta0=pci_f0_config */

1605 #define PCRF_CZ_VF_DEVID_LBN 0
1606 #define PCRF_CZ_VF_DEVID_WIDTH 16


1609 /*
1610  * PC_SRIOV_SUP_PAGESZ_REG(16bit):
1611  * SRIOV Supported Page Sizes
1612  */

1614 #define PCR_CC_SRIOV_SUP_PAGESZ_REG 0x0000017c
1615 /* sienaa0=pci_f0_config */

1617 #define PCR_DZ_SRIOV_SUP_PAGESZ_REG 0x0000021c
1618 /* hunta0=pci_f0_config */

1620 #define PCRF_CZ_VF_SUP_PAGESZ_LBN 0
1621 #define PCRF_CZ_VF_SUP_PAGESZ_WIDTH 16


1624 /*
1625  * PC_SRIOV_SYS_PAGESZ_REG(32bit):
1626  * SRIOV System Page Size
1627  */

1629 #define PCR_CC_SRIOV_SYS_PAGESZ_REG 0x00000180
1630 /* sienaa0=pci_f0_config */

1632 #define PCR_DZ_SRIOV_SYS_PAGESZ_REG 0x00000220
1633 /* hunta0=pci_f0_config */

1635 #define PCRF_CZ_VF_SYS_PAGESZ_LBN 0
1636 #define PCRF_CZ_VF_SYS_PAGESZ_WIDTH 16


1639 /*
1640  * PC_SRIOV_BAR0_REG(32bit):
1641  * SRIOV VF Bar0
1642  */

1644 #define PCR_CC_SRIOV_BAR0_REG 0x00000184
1645 /* sienaa0=pci_f0_config */
```



```
1647 #define PCR_DZ_SRIOV_BAR0_REG 0x00000224
1648 /* hunta0=pci_f0_config */

1650 #define PCRF_CC_VF_BAR_ADDRESS_LBN 0
1651 #define PCRF_CC_VF_BAR_ADDRESS_WIDTH 32
1652 #define PCRF_DZ_VF_BAR0_ADDRESS_LBN 0
1653 #define PCRF_DZ_VF_BAR0_ADDRESS_WIDTH 32


1656 /*
1657  * PC_SRIOV_BAR1_REG(32bit):
1658  * SRIOV Bar1
1659  */

1661 #define PCR_CC_SRIOV_BAR1_REG 0x00000188
1662 /* sienaa0=pci_f0_config */

1664 #define PCR_DZ_SRIOV_BAR1_REG 0x00000228
1665 /* hunta0=pci_f0_config */

1667 /* defined as PCRF_CC_VF_BAR_ADDRESS_LBN 0; */
1668 /* defined as PCRF_CC_VF_BAR_ADDRESS_WIDTH 32 */
1669 #define PCRF_DZ_VF_BAR1_ADDRESS_LBN 0
1670 #define PCRF_DZ_VF_BAR1_ADDRESS_WIDTH 32


1673 /*
1674  * PC_SRIOV_BAR2_REG(32bit):
1675  * SRIOV Bar2
1676  */

1678 #define PCR_CC_SRIOV_BAR2_REG 0x0000018c
1679 /* sienaa0=pci_f0_config */

1681 #define PCR_DZ_SRIOV_BAR2_REG 0x0000022c
1682 /* hunta0=pci_f0_config */

1684 /* defined as PCRF_CC_VF_BAR_ADDRESS_LBN 0; */
1685 /* defined as PCRF_CC_VF_BAR_ADDRESS_WIDTH 32 */
1686 #define PCRF_DZ_VF_BAR2_ADDRESS_LBN 0
1687 #define PCRF_DZ_VF_BAR2_ADDRESS_WIDTH 32


1690 /*
1691  * PC_SRIOV_BAR3_REG(32bit):
1692  * SRIOV Bar3
1693  */

1695 #define PCR_CC_SRIOV_BAR3_REG 0x00000190
1696 /* sienaa0=pci_f0_config */

1698 #define PCR_DZ_SRIOV_BAR3_REG 0x00000230
1699 /* hunta0=pci_f0_config */

1701 /* defined as PCRF_CC_VF_BAR_ADDRESS_LBN 0; */
1702 /* defined as PCRF_CC_VF_BAR_ADDRESS_WIDTH 32 */
1703 #define PCRF_DZ_VF_BAR3_ADDRESS_LBN 0
1704 #define PCRF_DZ_VF_BAR3_ADDRESS_WIDTH 32


1707 /*
1708  * PC_SRIOV_BAR4_REG(32bit):
1709  * SRIOV Bar4
1710  */
```

```
#define PCR_CC_SRIOV_BAR4_REG 0x00000194
/* sienaa0=pci_f0_config */

#define PCR_DZ_SRIOV_BAR4_REG 0x00000234
/* hunta0=pci_f0_config */

/* defined as PCRF_CC_VF_BAR_ADDRESS_LBN 0; */
/* defined as PCRF_CC_VF_BAR_ADDRESS_WIDTH 32 */
#define PCRF_DZ_VF_BAR4_ADDRESS_LBN 0
#define PCRF_DZ_VF_BAR4_ADDRESS_WIDTH 32


/*
 * PC_SRIOV_BAR5_REG(32bit):
 * SRIOV Bar5
 */

#define PCR_CC_SRIOV_BAR5_REG 0x00000198
/* sienaa0=pci_f0_config */

#define PCR_DZ_SRIOV_BAR5_REG 0x00000238
/* hunta0=pci_f0_config */

/* defined as PCRF_CC_VF_BAR_ADDRESS_LBN 0; */
/* defined as PCRF_CC_VF_BAR_ADDRESS_WIDTH 32 */
#define PCRF_DZ_VF_BAR5_ADDRESS_LBN 0
#define PCRF_DZ_VF_BAR5_ADDRESS_WIDTH 32


/*
 * PC_SRIOV_MIBR_SARRAY_OFFSET_REG(32bit):
 * SRIOV VF Migration State Array Offset
 */

#define PCR_CC_SRIOV_MIBR_SARRAY_OFFSET_REG 0x0000019c
/* sienaa0=pci_f0_config */

#define PCR_DZ_SRIOV_MIBR_SARRAY_OFFSET_REG 0x0000023c
/* hunta0=pci_f0_config */

#define PCRF_CZ_VF_MIGR_OFFSET_LBN 3
#define PCRF_CZ_VF_MIGR_OFFSET_WIDTH 29
#define PCRF_CZ_VF_MIGR_BIR_LBN 0
#define PCRF_CZ_VF_MIGR_BIR_WIDTH 3


/*
 * PC_LTR_CAP_HDR_REG(32bit):
 * Latency Tolerance Reporting Cap Header Reg
 */

#define PCR_DZ_LTR_CAP_HDR_REG 0x00000240
/* hunta0=pci_f0_config */

#define PCRF_DZ_LTR_NXT_PTR_LBN 20
#define PCRF_DZ_LTR_NXT_PTR_WIDTH 12
#define PCRF_DZ_LTR_VERSION_LBN 16
#define PCRF_DZ_LTR_VERSION_WIDTH 4
#define PCRF_DZ_LTR_EXT_CAP_ID_LBN 0
#define PCRF_DZ_LTR_EXT_CAP_ID_WIDTH 16


/*
 * PC_LTR_MAX_SNOOP_REG(32bit):
 * LTR Maximum Snoop/No Snoop Register
 */
```



```
#define PCR_DZ_LTR_MAX_SNOOP_REG 0x00000244
/* hunta0=pci_f0_config */

#define PCRF_DZ_LTR_MAX_NOSNOOP_SCALE_LBN 26
#define PCRF_DZ_LTR_MAX_NOSNOOP_SCALE_WIDTH 3
#define PCRF_DZ_LTR_MAX_NOSNOOP_LAT_LBN 16
#define PCRF_DZ_LTR_MAX_NOSNOOP_LAT_WIDTH 10
#define PCRF_DZ_LTR_MAX_SNOOP_SCALE_LBN 10
#define PCRF_DZ_LTR_MAX_SNOOP_SCALE_WIDTH 3
#define PCRF_DZ_LTR_MAX_SNOOP_LAT_LBN 0
#define PCRF_DZ_LTR_MAX_SNOOP_LAT_WIDTH 10


/*
 * PC_TPH_CAP_HDR_REG(32bit):
 * TPH Capability Header Register
 */

#define PCR_DZ_TPH_CAP_HDR_REG 0x00000274
/* hunta0=pci_f0_config */

#define PCRF_DZ_TPH_NXT_PTR_LBN 20
#define PCRF_DZ_TPH_NXT_PTR_WIDTH 12
#define PCRF_DZ_TPH_VERSION_LBN 16
#define PCRF_DZ_TPH_VERSION_WIDTH 4
#define PCRF_DZ_TPH_EXT_CAP_ID_LBN 0
#define PCRF_DZ_TPH_EXT_CAP_ID_WIDTH 16


/*
 * PC_TPH_REQ_CAP_REG(32bit):
 * TPH Requester Capability Register
 */

#define PCR_DZ_TPH_REQ_CAP_REG 0x00000278
/* hunta0=pci_f0_config */

#define PCRF_DZ_ST_TBLE_SIZE_LBN 16
#define PCRF_DZ_ST_TBLE_SIZE_WIDTH 11
#define PCRF_DZ_ST_TBLE_LOC_LBN 9
#define PCRF_DZ_ST_TBLE_LOC_WIDTH 2
#define PCRF_DZ_EXT_TPH_MODE_SUP_LBN 8
#define PCRF_DZ_EXT_TPH_MODE_SUP_WIDTH 1
#define PCRF_DZ_TPH_DEV_MODE_SUP_LBN 2
#define PCRF_DZ_TPH_DEV_MODE_SUP_WIDTH 1
#define PCRF_DZ_TPH_INT_MODE_SUP_LBN 1
#define PCRF_DZ_TPH_INT_MODE_SUP_WIDTH 1
#define PCRF_DZ_TPH_NOST_MODE_SUP_LBN 0
#define PCRF_DZ_TPH_NOST_MODE_SUP_WIDTH 1


/*
 * PC_TPH_REQ_CTL_REG(32bit):
 * TPH Requester Control Register
 */

#define PCR_DZ_TPH_REQ_CTL_REG 0x0000027c
/* hunta0=pci_f0_config */

#define PCRF_DZ_TPH_REQ_ENABLE_LBN 8
#define PCRF_DZ_TPH_REQ_ENABLE_WIDTH 2
#define PCRF_DZ_TPH_ST_MODE_LBN 0
#define PCRF_DZ_TPH_ST_MODE_WIDTH 3
```

```
1844 /*
1845  * PC_SEC_PCIE_CAP_REG(32bit):
1846  * Secondary PCIE Capability Register
1847  */

1849 #define PCR_DZ_SEC_PCIE_CAP_REG 0x00000300
1850 /* hunta0=pci_f0_config */

1852 #define PCRF_DZ_SEC_NXT_PTR_LBN 20
1853 #define PCRF_DZ_SEC_NXT_PTR_WIDTH 12
1854 #define PCRF_DZ_SEC_VERSION_LBN 16
1855 #define PCRF_DZ_SEC_VERSION_WIDTH 4
1856 #define PCRF_DZ_SEC_EXT_CAP_ID_LBN 0
1857 #define PCRF_DZ_SEC_EXT_CAP_ID_WIDTH 16


1860 /*
1861  * PC_LINK_CONTROL3_REG(32bit):
1862  * Link Control 3.
1863  */

1865 #define PCR_DZ_LINK_CONTROL3_REG 0x00000304
1866 /* hunta0=pci_f0_config */

1868 #define PCRF_DZ_LINK_EQ_INT_EN_LBN 1
1869 #define PCRF_DZ_LINK_EQ_INT_EN_WIDTH 1
1870 #define PCRF_DZ_PERFORM_EQL_LBN 0
1871 #define PCRF_DZ_PERFORM_EQL_WIDTH 1


1874 /*
1875  * PC_LANE_ERROR_STAT_REG(32bit):
1876  * Lane Error Status Register.
1877  */

1879 #define PCR_DZ_LANE_ERROR_STAT_REG 0x00000308
1880 /* hunta0=pci_f0_config */

1882 #define PCRF_DZ_LANE_STATUS_LBN 0
1883 #define PCRF_DZ_LANE_STATUS_WIDTH 8


1886 /*
1887  * PC_LANE01_EQU_CONTROL_REG(32bit):
1888  * Lanes 0,1 Equalization Control Register.
1889  */

1891 #define PCR_DZ_LANE01_EQU_CONTROL_REG 0x0000030c
1892 /* hunta0=pci_f0_config */

1894 #define PCRF_DZ_LANE1_EQ_CTRL_LBN 16
1895 #define PCRF_DZ_LANE1_EQ_CTRL_WIDTH 16
1896 #define PCRF_DZ_LANE0_EQ_CTRL_LBN 0
1897 #define PCRF_DZ_LANE0_EQ_CTRL_WIDTH 16


1900 /*
1901  * PC_LANE23_EQU_CONTROL_REG(32bit):
1902  * Lanes 2,3 Equalization Control Register.
1903  */

1905 #define PCR_DZ_LANE23_EQU_CONTROL_REG 0x00000310
1906 /* hunta0=pci_f0_config */

1908 #define PCRF_DZ_LANE3_EQ_CTRL_LBN 16
1909 #define PCRF_DZ_LANE3_EQ_CTRL_WIDTH 16
```



```
1910 #define PCRF_DZ_LANE2_EQ_CTRL_LBN 0
1911 #define PCRF_DZ_LANE2_EQ_CTRL_WIDTH 16


1914 /*
1915  * PC_LANE45_EQU_CONTROL_REG(32bit):
1916  * Lanes 4,5 Equalization Control Register.
1917  */

1919 #define PCR_DZ_LANE45_EQU_CONTROL_REG 0x00000314
1920 /* hunta0=pci_f0_config */

1922 #define PCRF_DZ_LANE5_EQ_CTRL_LBN 16
1923 #define PCRF_DZ_LANE5_EQ_CTRL_WIDTH 16
1924 #define PCRF_DZ_LANE4_EQ_CTRL_LBN 0
1925 #define PCRF_DZ_LANE4_EQ_CTRL_WIDTH 16


1928 /*
1929  * PC_LANE67_EQU_CONTROL_REG(32bit):
1930  * Lanes 6,7 Equalization Control Register.
1931  */

1933 #define PCR_DZ_LANE67_EQU_CONTROL_REG 0x00000318
1934 /* hunta0=pci_f0_config */

1936 #define PCRF_DZ_LANE7_EQ_CTRL_LBN 16
1937 #define PCRF_DZ_LANE7_EQ_CTRL_WIDTH 16
1938 #define PCRF_DZ_LANE6_EQ_CTRL_LBN 0
1939 #define PCRF_DZ_LANE6_EQ_CTRL_WIDTH 16


1942 /*
1943  * PC_ACK_LAT_TMR_REG(32bit):
1944  * ACK latency timer & replay timer register
1945  */

1947 #define PCR_AC_ACK_LAT_TMR_REG 0x00000700
1948 /* falcona0,falconb0,sienaa0=pci_f0_config */

1950 #define PCRF_AC_RT_LBN 16
1951 #define PCRF_AC_RT_WIDTH 16
1952 #define PCRF_AC_ALT_LBN 0
1953 #define PCRF_AC_ALT_WIDTH 16


1956 /*
1957  * PC_OTHER_MSG_REG(32bit):
1958  * Other message register
1959  */

1961 #define PCR_AC_OTHER_MSG_REG 0x00000704
1962 /* falcona0,falconb0,sienaa0=pci_f0_config */

1964 #define PCRF_AC_OM_CRPT3_LBN 24
1965 #define PCRF_AC_OM_CRPT3_WIDTH 8
1966 #define PCRF_AC_OM_CRPT2_LBN 16
1967 #define PCRF_AC_OM_CRPT2_WIDTH 8
1968 #define PCRF_AC_OM_CRPT1_LBN 8
1969 #define PCRF_AC_OM_CRPT1_WIDTH 8
1970 #define PCRF_AC_OM_CRPT0_LBN 0
1971 #define PCRF_AC_OM_CRPT0_WIDTH 8


1974 /*
1975  * PC_FORCE_LNK_REG(24bit):
```

```
1976  * Port force link register
1977  */

1979 #define PCR_AC_FORCE_LNK_REG 0x00000708
1980 /* falcona0,falconb0,sienaa0=pci_f0_config */

1982 #define PCRF_AC_LFS_LBN 16
1983 #define PCRF_AC_LFS_WIDTH 6
1984 #define PCRF_AC_FL_LBN 15
1985 #define PCRF_AC_FL_WIDTH 1
1986 #define PCRF_AC_LN_LBN 0
1987 #define PCRF_AC_LN_WIDTH 8


1990 /*
1991  * PC_ACK_FREQ_REG(32bit):
1992  * ACK frequency register
1993  */

1995 #define PCR_AC_ACK_FREQ_REG 0x0000070c
1996 /* falcona0,falconb0,sienaa0=pci_f0_config */

1998 #define PCRF_CC_ALLOW_L1_WITHOUT_L0S_LBN 30
1999 #define PCRF_CC_ALLOW_L1_WITHOUT_L0S_WIDTH 1
2000 #define PCRF_AC_L1_ENTR_LAT_LBN 27
2001 #define PCRF_AC_L1_ENTR_LAT_WIDTH 3
2002 #define PCRF_AC_L0_ENTR_LAT_LBN 24
2003 #define PCRF_AC_L0_ENTR_LAT_WIDTH 3
2004 #define PCRF_CC_COMM_NFTS_LBN 16
2005 #define PCRF_CC_COMM_NFTS_WIDTH 8
2006 #define PCRF_AB_ACK_FREQ_REG_RSVD0_LBN 16
2007 #define PCRF_AB_ACK_FREQ_REG_RSVD0_WIDTH 3
2008 #define PCRF_AC_MAX_FTS_LBN 8
2009 #define PCRF_AC_MAX_FTS_WIDTH 8
2010 #define PCRF_AC_ACK_FREQ_LBN 0
2011 #define PCRF_AC_ACK_FREQ_WIDTH 8


2014 /*
2015  * PC_PORT_LNK_CTL_REG(32bit):
2016  * Port link control register
2017  */

2019 #define PCR_AC_PORT_LNK_CTL_REG 0x00000710
2020 /* falcona0,falconb0,sienaa0=pci_f0_config */

2022 #define PCRF_AB_LRE_LBN 27
2023 #define PCRF_AB_LRE_WIDTH 1
2024 #define PCRF_AB_ESYNC_LBN 26
2025 #define PCRF_AB_ESYNC_WIDTH 1
2026 #define PCRF_AB_CRPT_LBN 25
2027 #define PCRF_AB_CRPT_WIDTH 1
2028 #define PCRF_AB_XB_LBN 24
2029 #define PCRF_AB_XB_WIDTH 1
2030 #define PCRF_AC_LC_LBN 16
2031 #define PCRF_AC_LC_WIDTH 6
2032 #define PCRF_AC_LDR_LBN 8
2033 #define PCRF_AC_LDR_WIDTH 4
2034 #define PCRF_AC_FLM_LBN 7
2035 #define PCRF_AC_FLM_WIDTH 1
2036 #define PCRF_AC_LKD_LBN 6
2037 #define PCRF_AC_LKD_WIDTH 1
2038 #define PCRF_AC_DLE_LBN 5
2039 #define PCRF_AC_DLE_WIDTH 1
2040 #define PCRF_AB_PORT_LNK_CTL_REG_RSVD0_LBN 4
2041 #define PCRF_AB_PORT_LNK_CTL_REG_RSVD0_WIDTH 1
```



```
2042 #define PCRF_AC_RA_LBN 3
2043 #define PCRF_AC_RA_WIDTH 1
2044 #define PCRF_AC_LE_LBN 2
2045 #define PCRF_AC_LE_WIDTH 1
2046 #define PCRF_AC_SD_LBN 1
2047 #define PCRF_AC_SD_WIDTH 1
2048 #define PCRF_AC_OMR_LBN 0
2049 #define PCRF_AC_OMR_WIDTH 1


2052 /*
2053  * PC_LN_SKEW_REG(32bit):
2054  * Lane skew register
2055  */

2057 #define PCR_AC_LN_SKEW_REG 0x00000714
2058 /* falcona0,falconb0,sienaa0=pci_f0_config */

2060 #define PCRF_AC_DIS_LBN 31
2061 #define PCRF_AC_DIS_WIDTH 1
2062 #define PCRF_AB_RST_LBN 30
2063 #define PCRF_AB_RST_WIDTH 1
2064 #define PCRF_AC_AD_LBN 25
2065 #define PCRF_AC_AD_WIDTH 1
2066 #define PCRF_AC_FCD_LBN 24
2067 #define PCRF_AC_FCD_WIDTH 1
2068 #define PCRF_AC_LS2_LBN 16
2069 #define PCRF_AC_LS2_WIDTH 8
2070 #define PCRF_AC_LS1_LBN 8
2071 #define PCRF_AC_LS1_WIDTH 8
2072 #define PCRF_AC_LS0_LBN 0
2073 #define PCRF_AC_LS0_WIDTH 8


2076 /*
2077  * PC_SYM_NUM_REG(16bit):
2078  * Symbol number register
2079  */

2081 #define PCR_AC_SYM_NUM_REG 0x00000718
2082 /* falcona0,falconb0,sienaa0=pci_f0_config */

2084 #define PCRF_CC_MAX_FUNCTIONS_LBN 29
2085 #define PCRF_CC_MAX_FUNCTIONS_WIDTH 3
2086 #define PCRF_CC_FC_WATCHDOG_TMR_LBN 24
2087 #define PCRF_CC_FC_WATCHDOG_TMR_WIDTH 5
2088 #define PCRF_CC_ACK_NAK_TMR_MOD_LBN 19
2089 #define PCRF_CC_ACK_NAK_TMR_MOD_WIDTH 5
2090 #define PCRF_CC_REPLAY_TMR_MOD_LBN 14
2091 #define PCRF_CC_REPLAY_TMR_MOD_WIDTH 5
2092 #define PCRF_AB_ES_LBN 12
2093 #define PCRF_AB_ES_WIDTH 3
2094 #define PCRF_AB_SYM_NUM_REG_RSVD0_LBN 11
2095 #define PCRF_AB_SYM_NUM_REG_RSVD0_WIDTH 1
2096 #define PCRF_CC_NUM_SKP_SYMS_LBN 8
2097 #define PCRF_CC_NUM_SKP_SYMS_WIDTH 3
2098 #define PCRF_AB_TS2_LBN 4
2099 #define PCRF_AB_TS2_WIDTH 4
2100 #define PCRF_AC_TS1_LBN 0
2101 #define PCRF_AC_TS1_WIDTH 4


2104 /*
2105  * PC_SYM_TMR_FLT_MSK_REG(16bit):
2106  * Symbol timer and Filter Mask Register
2107  */
```

```
#define PCR_CC_SYM_TMR_FLT_MSK_REG 0x0000071c
/* sienaa0=pci_f0_config */

#define PCRF_CC_DEFAULT_FLT_MSK1_LBN 16
#define PCRF_CC_DEFAULT_FLT_MSK1_WIDTH 16
#define PCRF_CC_FC_WDOG_TMR_DIS_LBN 15
#define PCRF_CC_FC_WDOG_TMR_DIS_WIDTH 1
#define PCRF_CC_SI1_LBN 8
#define PCRF_CC_SI1_WIDTH 3
#define PCRF_CC_SKIP_INT_VAL_LBN 0
#define PCRF_CC_SKIP_INT_VAL_WIDTH 11
#define PCRF_CC_SI0_LBN 0
#define PCRF_CC_SI0_WIDTH 8


/*
 * PC_SYM_TMR_REG(16bit):
 * Symbol timer register
 */

#define PCR_AB_SYM_TMR_REG 0x0000071c
/* falcona0,falconb0=pci_f0_config */

#define PCRF_AB_ET_LBN 11
#define PCRF_AB_ET_WIDTH 4
#define PCRF_AB_SI1_LBN 8
#define PCRF_AB_SI1_WIDTH 3
#define PCRF_AB_SI0_LBN 0
#define PCRF_AB_SI0_WIDTH 8


/*
 * PC_PHY_STAT_REG(32bit):
 * PHY status register
 */

#define PCR_AB_PHY_STAT_REG 0x00000720
/* falcona0,falconb0=pci_f0_config */

#define PCR_CC_PHY_STAT_REG 0x00000810
/* sienaa0=pci_f0_config */

#define PCRF_AC_SSL_LBN 3
#define PCRF_AC_SSL_WIDTH 1
#define PCRF_AC_SSR_LBN 2
#define PCRF_AC_SSR_WIDTH 1
#define PCRF_AC_SSCL_LBN 1
#define PCRF_AC_SSCL_WIDTH 1
#define PCRF_AC_SSCD_LBN 0
#define PCRF_AC_SSCD_WIDTH 1


/*
 * PC_FLT_MSK_REG(32bit):
 * Filter Mask Register 2
 */

#define PCR_CC_FLT_MSK_REG 0x00000720
/* sienaa0=pci_f0_config */

#define PCRF_CC_DEFAULT_FLT_MSK2_LBN 0
#define PCRF_CC_DEFAULT_FLT_MSK2_WIDTH 32


/*
```



```
 * PC_PHY_CTL_REG(32bit):
 * PHY control register
 */

#define PCR_AB_PHY_CTL_REG 0x00000724
/* falcona0,falconb0=pci_f0_config */

#define PCR_CC_PHY_CTL_REG 0x00000814
/* sienaa0=pci_f0_config */

#define PCRF_AC_BD_LBN 31
#define PCRF_AC_BD_WIDTH 1
#define PCRF_AC_CDS_LBN 30
#define PCRF_AC_CDS_WIDTH 1
#define PCRF_AC_DWRAP_LB_LBN 29
#define PCRF_AC_DWRAP_LB_WIDTH 1
#define PCRF_AC_EBD_LBN 28
#define PCRF_AC_EBD_WIDTH 1
#define PCRF_AC_SNR_LBN 27
#define PCRF_AC_SNR_WIDTH 1
#define PCRF_AC_RX_NOT_DET_LBN 2
#define PCRF_AC_RX_NOT_DET_WIDTH 1
#define PCRF_AC_FORCE_LOS_VAL_LBN 1
#define PCRF_AC_FORCE_LOS_VAL_WIDTH 1
#define PCRF_AC_FORCE_LOS_EN_LBN 0
#define PCRF_AC_FORCE_LOS_EN_WIDTH 1


/*
 * PC_DEBUG0_REG(32bit):
 * Debug register 0
 */

#define PCR_AC_DEBUG0_REG 0x00000728
/* falcona0,falconb0,sienaa0=pci_f0_config */

#define PCRF_AC_CDI03_LBN 24
#define PCRF_AC_CDI03_WIDTH 8
#define PCRF_AC_CDI0_LBN 0
#define PCRF_AC_CDI0_WIDTH 32
#define PCRF_AC_CDI02_LBN 16
#define PCRF_AC_CDI02_WIDTH 8
#define PCRF_AC_CDI01_LBN 8
#define PCRF_AC_CDI01_WIDTH 8
#define PCRF_AC_CDI00_LBN 0
#define PCRF_AC_CDI00_WIDTH 8


/*
 * PC_DEBUG1_REG(32bit):
 * Debug register 1
 */

#define PCR_AC_DEBUG1_REG 0x0000072c
/* falcona0,falconb0,sienaa0=pci_f0_config */

#define PCRF_AC_CDI13_LBN 24
#define PCRF_AC_CDI13_WIDTH 8
#define PCRF_AC_CDI1_LBN 0
#define PCRF_AC_CDI1_WIDTH 32
#define PCRF_AC_CDI12_LBN 16
#define PCRF_AC_CDI12_WIDTH 8
#define PCRF_AC_CDI11_LBN 8
#define PCRF_AC_CDI11_WIDTH 8
#define PCRF_AC_CDI10_LBN 0
#define PCRF_AC_CDI10_WIDTH 8
```

```
2242 /*
2243  * PC_XPFCC_STAT_REG(24bit):
2244  * documentation to be written for sum_PC_XPFCC_STAT_REG
2245  */

2247 #define PCR_AC_XPFCC_STAT_REG 0x00000730
2248 /* falcona0,falconb0,sienaa0=pci_f0_config */

2250 #define PCRF_AC_XPDC_LBN 12
2251 #define PCRF_AC_XPDC_WIDTH 8
2252 #define PCRF_AC_XPHC_LBN 0
2253 #define PCRF_AC_XPHC_WIDTH 12


2256 /*
2257  * PC_XNPFCC_STAT_REG(24bit):
2258  * documentation to be written for sum_PC_XNPFCC_STAT_REG
2259  */

2261 #define PCR_AC_XNPFCC_STAT_REG 0x00000734
2262 /* falcona0,falconb0,sienaa0=pci_f0_config */

2264 #define PCRF_AC_XNPDC_LBN 12
2265 #define PCRF_AC_XNPDC_WIDTH 8
2266 #define PCRF_AC_XNPHC_LBN 0
2267 #define PCRF_AC_XNPHC_WIDTH 12


2270 /*
2271  * PC_XCFCC_STAT_REG(24bit):
2272  * documentation to be written for sum_PC_XCFCC_STAT_REG
2273  */

2275 #define PCR_AC_XCFCC_STAT_REG 0x00000738
2276 /* falcona0,falconb0,sienaa0=pci_f0_config */

2278 #define PCRF_AC_XCDC_LBN 12
2279 #define PCRF_AC_XCDC_WIDTH 8
2280 #define PCRF_AC_XCHC_LBN 0
2281 #define PCRF_AC_XCHC_WIDTH 12


2284 /*
2285  * PC_Q_STAT_REG(8bit):
2286  * documentation to be written for sum_PC_Q_STAT_REG
2287  */

2289 #define PCR_AC_Q_STAT_REG 0x0000073c
2290 /* falcona0,falconb0,sienaa0=pci_f0_config */

2292 #define PCRF_AC_RQNE_LBN 2
2293 #define PCRF_AC_RQNE_WIDTH 1
2294 #define PCRF_AC_XRNE_LBN 1
2295 #define PCRF_AC_XRNE_WIDTH 1
2296 #define PCRF_AC_RCNR_LBN 0
2297 #define PCRF_AC_RCNR_WIDTH 1


2300 /*
2301  * PC_VC_XMIT_ARB1_REG(32bit):
2302  * VC Transmit Arbitration Register 1
2303  */

2305 #define PCR_CC_VC_XMIT_ARB1_REG 0x00000740
```



```
2306 /* sienaa0=pci_f0_config */



2310 /*
2311  * PC_VC_XMIT_ARB2_REG(32bit):
2312  * VC Transmit Arbitration Register 2
2313  */

2315 #define PCR_CC_VC_XMIT_ARB2_REG 0x00000744
2316 /* sienaa0=pci_f0_config */



2320 /*
2321  * PC_VC0_P_RQ_CTL_REG(32bit):
2322  * VC0 Posted Receive Queue Control
2323  */

2325 #define PCR_CC_VC0_P_RQ_CTL_REG 0x00000748
2326 /* sienaa0=pci_f0_config */



2330 /*
2331  * PC_VC0_NP_RQ_CTL_REG(32bit):
2332  * VC0 Non-Posted Receive Queue Control
2333  */

2335 #define PCR_CC_VC0_NP_RQ_CTL_REG 0x0000074c
2336 /* sienaa0=pci_f0_config */



2340 /*
2341  * PC_VC0_C_RQ_CTL_REG(32bit):
2342  * VC0 Completion Receive Queue Control
2343  */

2345 #define PCR_CC_VC0_C_RQ_CTL_REG 0x00000750
2346 /* sienaa0=pci_f0_config */



2350 /*
2351  * PC_GEN2_REG(32bit):
2352  * Gen2 Register
2353  */

2355 #define PCR_CC_GEN2_REG 0x0000080c
2356 /* sienaa0=pci_f0_config */

2358 #define PCRF_CC_SET_DE_EMPHASIS_LBN 20
2359 #define PCRF_CC_SET_DE_EMPHASIS_WIDTH 1
2360 #define PCRF_CC_CFG_TX_COMPLIANCE_LBN 19
2361 #define PCRF_CC_CFG_TX_COMPLIANCE_WIDTH 1
2362 #define PCRF_CC_CFG_TX_SWING_LBN 18
2363 #define PCRF_CC_CFG_TX_SWING_WIDTH 1
2364 #define PCRF_CC_DIR_SPEED_CHANGE_LBN 17
2365 #define PCRF_CC_DIR_SPEED_CHANGE_WIDTH 1
2366 #define PCRF_CC_LANE_ENABLE_LBN 8
2367 #define PCRF_CC_LANE_ENABLE_WIDTH 9
2368 #define PCRF_CC_NUM_FTS_LBN 0
2369 #define PCRF_CC_NUM_FTS_WIDTH 8
```

```
2372 #ifdef  __cplusplus
2373 }
2374 #endif

2376 #endif /* _SYS_EFX_REGS_PCI_H */
2377 #endif /* ! codereview */
```

```
**********************************************************
   19544 Thu Aug 22 18:59:23 2013
new/usr/src/uts/common/io/sfxge/efx_rx.c
Merged sfxge driver
**********************************************************
```

```
/*
 * Copyright 2007-2013 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#include "efsys.h"
#include "efx.h"
#include "efx_types.h"
#include "efx_regs.h"
#include "efx_impl.h"

        __checkReturn   int
efx_rx_init(
        __in            efx_nic_t *enp)
{
        efx_oword_t oword;
        unsigned int index;
        int rc;

        EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
        EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_NIC);

        if (!(enp->en_mod_flags & EFX_MOD_EV)) {
                rc = EINVAL;
                goto fail1;
        }

        if (enp->en_mod_flags & EFX_MOD_RX) {
                rc = EINVAL;
                goto fail2;
        }

        EFX_BAR_READO(enp, FR_AZ_RX_CFG_REG, &oword);

        EFX_SET_OWORD_FIELD(oword, FRF_BZ_RX_DESC_PUSH_EN, 0);
        EFX_SET_OWORD_FIELD(oword, FRF_BZ_RX_HASH_ALG, 0);
        EFX_SET_OWORD_FIELD(oword, FRF_BZ_RX_IP_HASH, 0);
        EFX_SET_OWORD_FIELD(oword, FRF_BZ_RX_TCP_SUP, 0);
        EFX_SET_OWORD_FIELD(oword, FRF_BZ_RX_HASH_INSRT_HDR, 0);
        EFX_SET_OWORD_FIELD(oword, FRF_BZ_RX_USR_BUF_SIZE, 0x3000 / 32);
        EFX_BAR_WRITEO(enp, FR_AZ_RX_CFG_REG, &oword);
```



```
        /* Zero the RSS table */
        for (index = 0; index < FR_BZ_RX_INDIRECTION_TBL_ROWS;
            index++) {
                EFX_ZERO_OWORD(oword);
                EFX_BAR_TBL_WRITEO(enp, FR_BZ_RX_INDIRECTION_TBL,
                                   index, &oword);
        }

        enp->en_mod_flags |= EFX_MOD_RX;
        return (0);

fail2:
        EFSYS_PROBE(fail2);
fail1:
        EFSYS_PROBE1(fail1, int, rc);

        return (rc);
}

#if EFSYS_OPT_RX_HDR_SPLIT
        __checkReturn   int
efx_rx_hdr_split_enable(
        __in            efx_nic_t *enp,
        __in            unsigned int hdr_buf_size,
        __in            unsigned int pld_buf_size)
{
        unsigned int nhdr32;
        unsigned int npld32;
        efx_oword_t oword;
        int rc;

        EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
        EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_RX);
        EFSYS_ASSERT3U(enp->en_family, >=, EFX_FAMILY_SIENA);

        nhdr32 = hdr_buf_size / 32;
        if ((nhdr32 == 0) ||
            (nhdr32 >= (1 << FRF_CZ_RX_HDR_SPLIT_HDR_BUF_SIZE_WIDTH)) ||
            ((hdr_buf_size % 32) != 0)) {
                rc = EINVAL;
                goto fail1;
        }

        npld32 = pld_buf_size / 32;
        if ((npld32 == 0) ||
            (npld32 >= (1 << FRF_CZ_RX_HDR_SPLIT_PLD_BUF_SIZE_WIDTH)) ||
            ((pld_buf_size % 32) != 0)) {
                rc = EINVAL;
                goto fail2;
        }

        if (enp->en_rx_qcount > 0) {
                rc = EBUSY;
                goto fail3;
        }

        EFX_BAR_READO(enp, FR_AZ_RX_CFG_REG, &oword);

        EFX_SET_OWORD_FIELD(oword, FRF_CZ_RX_HDR_SPLIT_EN, 1);
        EFX_SET_OWORD_FIELD(oword, FRF_CZ_RX_HDR_SPLIT_HDR_BUF_SIZE, nhdr32);
        EFX_SET_OWORD_FIELD(oword, FRF_CZ_RX_HDR_SPLIT_PLD_BUF_SIZE, npld32);

        EFX_BAR_WRITEO(enp, FR_AZ_RX_CFG_REG, &oword);

        return (0);
```

```
fail3:
        EFSYS_PROBE(fail3);
fail2:
        EFSYS_PROBE(fail2);
fail1:
        EFSYS_PROBE1(fail1, int, rc);

        return (rc);
}
#endif  /* EFSYS_OPT_RX_HDR_SPLIT */


#if EFSYS_OPT_RX_SCATTER
        __checkReturn   int
efx_rx_scatter_enable(
        __in            efx_nic_t *enp,
        __in            unsigned int buf_size)
{
        unsigned int nbuf32;
        efx_oword_t oword;
        int rc;

        EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
        EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_RX);
        EFSYS_ASSERT3U(enp->en_family, >=, EFX_FAMILY_FALCON);

        nbuf32 = buf_size / 32;
        if ((nbuf32 == 0) ||
            (nbuf32 >= (1 << FRF_BZ_RX_USR_BUF_SIZE_WIDTH)) ||
            ((buf_size % 32) != 0)) {
                rc = EINVAL;
                goto fail1;
        }

        if (enp->en_rx_qcount > 0) {
                rc = EBUSY;
                goto fail2;
        }

        /* Set scatter buffer size */
        EFX_BAR_READO(enp, FR_AZ_RX_CFG_REG, &oword);
        EFX_SET_OWORD_FIELD(oword, FRF_BZ_RX_USR_BUF_SIZE, nbuf32);
        EFX_BAR_WRITEO(enp, FR_AZ_RX_CFG_REG, &oword);

        /* Enable scatter for packets not matching a filter */
        EFX_BAR_READO(enp, FR_AZ_RX_FILTER_CTL_REG, &oword);
        EFX_SET_OWORD_FIELD(oword, FRF_BZ_SCATTER_ENBL_NO_MATCH_Q, 1);
        EFX_BAR_WRITEO(enp, FR_AZ_RX_FILTER_CTL_REG, &oword);

        return (0);

fail2:
        EFSYS_PROBE(fail2);
fail1:
        EFSYS_PROBE1(fail1, int, rc);

        return (rc);
}
#endif  /* EFSYS_OPT_RX_SCATTER */


#define EFX_RX_LFSR_HASH(_enp, _insert)                                 \
        do {                                                            \
                efx_oword_t oword;                                      \
                                                                        \
```



```
                EFX_BAR_READO((_enp), FR_AZ_RX_CFG_REG, &oword);        \
                EFX_SET_OWORD_FIELD(oword, FRF_BZ_RX_HASH_ALG, 0);      \
                EFX_SET_OWORD_FIELD(oword, FRF_BZ_RX_IP_HASH, 0);       \
                EFX_SET_OWORD_FIELD(oword, FRF_BZ_RX_TCP_SUP, 0);       \
                EFX_SET_OWORD_FIELD(oword, FRF_BZ_RX_HASH_INSRT_HDR,    \
                    (_insert) ? 1 : 0);                                 \
                EFX_BAR_WRITEO((_enp), FR_AZ_RX_CFG_REG, &oword);       \
                                                                        \
                if ((_enp)->en_family == EFX_FAMILY_SIENA) {            \
                        EFX_BAR_READO((_enp), FR_CZ_RX_RSS_IPV6_REG3,   \
                            &oword);                                    \
                        EFX_SET_OWORD_FIELD(oword,                      \
                            FRF_CZ_RX_RSS_IPV6_THASH_ENABLE, 0);        \
                        EFX_BAR_WRITEO((_enp), FR_CZ_RX_RSS_IPV6_REG3,  \
                            &oword);                                    \
                }                                                       \
                                                                        \
                _NOTE(CONSTANTCONDITION)                                \
        } while (B_FALSE)

#define EFX_RX_TOEPLITZ_IPV4_HASH(_enp, _insert, _ip, _tcp)             \
        do {                                                            \
                efx_oword_t oword;                                      \
                                                                        \
                EFX_BAR_READO((_enp), FR_AZ_RX_CFG_REG, &oword);        \
                EFX_SET_OWORD_FIELD(oword, FRF_BZ_RX_HASH_ALG, 1);      \
                EFX_SET_OWORD_FIELD(oword, FRF_BZ_RX_IP_HASH,           \
                    (_ip) ? 1 : 0);                                     \
                EFX_SET_OWORD_FIELD(oword, FRF_BZ_RX_TCP_SUP,           \
                    (_tcp) ? 0 : 1);                                    \
                EFX_SET_OWORD_FIELD(oword, FRF_BZ_RX_HASH_INSRT_HDR,    \
                    (_insert) ? 1 : 0);                                 \
                EFX_BAR_WRITEO((_enp), FR_AZ_RX_CFG_REG, &oword);       \
                                                                        \
                _NOTE(CONSTANTCONDITION)                                \
        } while (B_FALSE)

#define EFX_RX_TOEPLITZ_IPV6_HASH(_enp, _ip, _tcp, _rc)                 \
        do {                                                            \
                efx_oword_t oword;                                      \
                                                                        \
                if ((_enp)->en_family == EFX_FAMILY_FALCON) {           \
                        (_rc) = ((_ip) || (_tcp)) ? ENOTSUP : 0;        \
                        break;                                          \
                }                                                       \
                                                                        \
                EFX_BAR_READO((_enp), FR_CZ_RX_RSS_IPV6_REG3, &oword);  \
                EFX_SET_OWORD_FIELD(oword,                              \
                    FRF_CZ_RX_RSS_IPV6_THASH_ENABLE, 1);                \
                EFX_SET_OWORD_FIELD(oword,                              \
                    FRF_CZ_RX_RSS_IPV6_IP_THASH_ENABLE, (_ip) ? 1 : 0); \
                EFX_SET_OWORD_FIELD(oword,                              \
                    FRF_CZ_RX_RSS_IPV6_TCP_SUPPRESS, (_tcp) ? 0 : 1);   \
                EFX_BAR_WRITEO((_enp), FR_CZ_RX_RSS_IPV6_REG3, &oword); \
                                                                        \
                (_rc) = 0;                                              \
                                                                        \
                _NOTE(CONSTANTCONDITION)                                \
        } while (B_FALSE)


#if EFSYS_OPT_RX_SCALE
        __checkReturn   int
efx_rx_scale_mode_set(
        __in            efx_nic_t *enp,
        __in            efx_rx_hash_alg_t alg,
```

```
260         __in            efx_rx_hash_type_t type,
261         __in            boolean_t insert)
262 {
263         int rc;

265         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
266         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_RX);
267         EFSYS_ASSERT3U(enp->en_family, >=, EFX_FAMILY_FALCON);

269         switch (alg) {
270         case EFX_RX_HASHALG_LFSR:
271                 EFX_RX_LFSR_HASH(enp, insert);
272                 break;

274         case EFX_RX_HASHALG_TOEPLITZ:
275                 EFX_RX_TOEPLITZ_IPV4_HASH(enp, insert,
276                     type & (1 << EFX_RX_HASH_IPV4),
277                     type & (1 << EFX_RX_HASH_TCPIPV4));

279                 EFX_RX_TOEPLITZ_IPV6_HASH(enp,
280                     type & (1 << EFX_RX_HASH_IPV6),
281                     type & (1 << EFX_RX_HASH_TCPIPV6),
282                     rc);
283                 if (rc != 0)
284                         goto fail1;

286                 break;

288         default:
289                 rc = EINVAL;
290                 goto fail2;
291         }

293         return (0);

295 fail2:
296         EFSYS_PROBE(fail2);
297 fail1:
298         EFSYS_PROBE1(fail1, int, rc);

300         EFX_RX_LFSR_HASH(enp, B_FALSE);

302         return (rc);
303 }
304 #endif

306 #if EFSYS_OPT_RX_SCALE
307         __checkReturn   int
308 efx_rx_scale_toeplitz_ipv4_key_set(
309         __in            efx_nic_t *enp,
310         __in_ecount(n)  uint8_t *key,
311         __in            size_t n)
312 {
313         efx_oword_t oword;
314         unsigned int byte;
315         unsigned int offset;
316         int rc;

318         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
319         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_RX);

321         byte = 0;

323         /* Write toeplitz hash key */
324         EFX_ZERO_OWORD(oword);
325         for (offset = (FRF_BZ_RX_RSS_TKEY_LBN + FRF_BZ_RX_RSS_TKEY_WIDTH) / 8;
```



```
326             offset > 0 && byte < n;
327             --offset)
328                 oword.eo_u8[offset - 1] = key[byte++];

330         EFX_BAR_WRITEO(enp, FR_BZ_RX_RSS_TKEY_REG, &oword);

332         byte = 0;

334         /* Verify toeplitz hash key */
335         EFX_BAR_READO(enp, FR_BZ_RX_RSS_TKEY_REG, &oword);
336         for (offset = (FRF_BZ_RX_RSS_TKEY_LBN + FRF_BZ_RX_RSS_TKEY_WIDTH) / 8;
337             offset > 0 && byte < n;
338             --offset) {
339                 if (oword.eo_u8[offset - 1] != key[byte++]) {
340                         rc = EFAULT;
341                         goto fail1;
342                 }
343         }

345         return (0);

347 fail1:
348         EFSYS_PROBE1(fail1, int, rc);

350         return (rc);
351 }
352 #endif

354 #if EFSYS_OPT_RX_SCALE
355         __checkReturn   int
356 efx_rx_scale_toeplitz_ipv6_key_set(
357         __in            efx_nic_t *enp,
358         __in_ecount(n)  uint8_t *key,
359         __in            size_t n)
360 {
361         efx_oword_t oword;
362         unsigned int byte;
363         int offset;
364         int rc;

366         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
367         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_RX);

369         byte = 0;

371         /* Write toeplitz hash key 3 */
372         EFX_BAR_READO(enp, FR_CZ_RX_RSS_IPV6_REG3, &oword);
373         for (offset = (FRF_CZ_RX_RSS_IPV6_TKEY_HI_LBN +
374             FRF_CZ_RX_RSS_IPV6_TKEY_HI_WIDTH) / 8;
375             offset > 0 && byte < n;
376             --offset)
377                 oword.eo_u8[offset - 1] = key[byte++];

379         EFX_BAR_WRITEO(enp, FR_CZ_RX_RSS_IPV6_REG3, &oword);

381         /* Write toeplitz hash key 2 */
382         EFX_ZERO_OWORD(oword);
383         for (offset = (FRF_CZ_RX_RSS_IPV6_TKEY_MID_LBN +
384             FRF_CZ_RX_RSS_IPV6_TKEY_MID_WIDTH) / 8;
385             offset > 0 && byte < n;
386             --offset)
387                 oword.eo_u8[offset - 1] = key[byte++];

389         EFX_BAR_WRITEO(enp, FR_CZ_RX_RSS_IPV6_REG2, &oword);

391         /* Write toeplitz hash key 1 */
```

```
392         EFX_ZERO_OWORD(oword);
393         for (offset = (FRF_CZ_RX_RSS_IPV6_TKEY_LO_LBN +
394             FRF_CZ_RX_RSS_IPV6_TKEY_LO_WIDTH) / 8;
395             offset > 0 && byte < n;
396             --offset)
397                 oword.eo_u8[offset - 1] = key[byte++];

399         EFX_BAR_WRITEO(enp, FR_CZ_RX_RSS_IPV6_REG1, &oword);

401         byte = 0;

403         /* Verify toeplitz hash key 3 */
404         EFX_BAR_READO(enp, FR_CZ_RX_RSS_IPV6_REG3, &oword);
405         for (offset = (FRF_CZ_RX_RSS_IPV6_TKEY_HI_LBN +
406             FRF_CZ_RX_RSS_IPV6_TKEY_HI_WIDTH) / 8;
407             offset > 0 && byte < n;
408             --offset) {
409                 if (oword.eo_u8[offset - 1] != key[byte++]) {
410                         rc = EFAULT;
411                         goto fail1;
412                 }
413         }

415         /* Verify toeplitz hash key 2 */
416         EFX_BAR_READO(enp, FR_CZ_RX_RSS_IPV6_REG2, &oword);
417         for (offset = (FRF_CZ_RX_RSS_IPV6_TKEY_MID_LBN +
418             FRF_CZ_RX_RSS_IPV6_TKEY_MID_WIDTH) / 8;
419             offset > 0 && byte < n;
420             --offset) {
421                 if (oword.eo_u8[offset - 1] != key[byte++]) {
422                         rc = EFAULT;
423                         goto fail2;
424                 }
425         }

427         /* Verify toeplitz hash key 1 */
428         EFX_BAR_READO(enp, FR_CZ_RX_RSS_IPV6_REG1, &oword);
429         for (offset = (FRF_CZ_RX_RSS_IPV6_TKEY_LO_LBN +
430             FRF_CZ_RX_RSS_IPV6_TKEY_LO_WIDTH) / 8;
431             offset > 0 && byte < n;
432             --offset) {
433                 if (oword.eo_u8[offset - 1] != key[byte++]) {
434                         rc = EFAULT;
435                         goto fail3;
436                 }
437         }

439         return (0);

441 fail3:
442         EFSYS_PROBE(fail3);
443 fail2:
444         EFSYS_PROBE(fail2);
445 fail1:
446         EFSYS_PROBE1(fail1, int, rc);

448         return (rc);
449 }
450 #endif

452 #if EFSYS_OPT_RX_SCALE
453         __checkReturn   int
454 efx_rx_scale_tbl_set(
455         __in            efx_nic_t *enp,
456         __in_ecount(n)  unsigned int *table,
457         __in            size_t n)
```



```
458 {
459         efx_oword_t oword;
460         int index;
461         int rc;

463         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
464         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_RX);

466         EFX_STATIC_ASSERT(EFX_RSS_TBL_SIZE == FR_BZ_RX_INDIRECTION_TBL_ROWS);
467         EFX_STATIC_ASSERT(EFX_MAXRSS == (1 << FRF_BZ_IT_QUEUE_WIDTH));

469         if (n > FR_BZ_RX_INDIRECTION_TBL_ROWS) {
470                 rc = EINVAL;
471                 goto fail1;
472         }

474         for (index = 0; index < FR_BZ_RX_INDIRECTION_TBL_ROWS; index++) {
475                 uint32_t byte;

477                 /* Calculate the entry to place in the table */
478                 byte = (uint32_t)table[index % n];

480                 EFSYS_PROBE2(table, int, index, uint32_t, byte);

482                 EFX_POPULATE_OWORD_1(oword, FRF_BZ_IT_QUEUE, byte);

484                 /* Write the table */
485                 EFX_BAR_TBL_WRITEO(enp, FR_BZ_RX_INDIRECTION_TBL,
486                                    index, &oword);
487         }

489         for (index = FR_BZ_RX_INDIRECTION_TBL_ROWS - 1; index >= 0; --index) {
490                 uint32_t byte;

492                 /* Determine if we're starting a new batch */
493                 byte = (uint32_t)table[index % n];

495                 /* Read the table */
496                 EFX_BAR_TBL_READO(enp, FR_BZ_RX_INDIRECTION_TBL,
497                                   index, &oword);

499                 /* Verify the entry */
500                 if (EFX_OWORD_FIELD(oword, FRF_BZ_IT_QUEUE) != byte) {
501                         rc = EFAULT;
502                         goto fail2;
503                 }
504         }

506         return (0);

508 fail2:
509         EFSYS_PROBE(fail2);
510 fail1:
511         EFSYS_PROBE1(fail1, int, rc);

513         return (rc);
514 }
515 #endif

517 #if EFSYS_OPT_FILTER
518 extern  __checkReturn   int
519 efx_rx_filter_insert(
520         __in            efx_rxq_t *erp,
521         __inout         efx_filter_spec_t *spec)
522 {
523         EFSYS_ASSERT3U(erp->er_magic, ==, EFX_RXQ_MAGIC);
```

```
524         EFSYS_ASSERT3P(spec, !=, NULL);

526         spec->efs_dmaq_id = (uint16_t)erp->er_index;
527         return (efx_filter_insert_filter(erp->er_enp, spec, B_FALSE));
528 }
529 #endif

531 #if EFSYS_OPT_FILTER
532 extern  __checkReturn   int
533 efx_rx_filter_remove(
534         __in            efx_rxq_t *erp,
535         __inout         efx_filter_spec_t *spec)
536 {
537         EFSYS_ASSERT3U(erp->er_magic, ==, EFX_RXQ_MAGIC);
538         EFSYS_ASSERT3P(spec, !=, NULL);

540         spec->efs_dmaq_id = (uint16_t)erp->er_index;
541         return (efx_filter_remove_filter(erp->er_enp, spec));
542 }
543 #endif

545 extern                  void
546 efx_rx_qpost(
547         __in            efx_rxq_t *erp,
548         __in_ecount(n)  efsys_dma_addr_t *addrp,
549         __in            size_t size,
550         __in            unsigned int n,
551         __in            unsigned int completed,
552         __in            unsigned int added)
553 {
554         efx_qword_t qword;
555         unsigned int i;
556         unsigned int offset;
557         unsigned int id;

559         EFSYS_ASSERT3U(erp->er_magic, ==, EFX_RXQ_MAGIC);

561         /* The client driver must not overfill the queue */
562         EFSYS_ASSERT3U(added - completed + n, <=,
563             EFX_RXQ_LIMIT(erp->er_mask + 1));

565         id = added & (erp->er_mask);
566         for (i = 0; i < n; i++) {
567                 EFSYS_PROBE4(rx_post, unsigned int, erp->er_index,
568                     unsigned int, id, efsys_dma_addr_t, addrp[i],
569                     size_t, size);

571                 EFX_POPULATE_QWORD_3(qword,
572                     FSF_AZ_RX_KER_BUF_SIZE, (uint32_t)(size),
573                     FSF_AZ_RX_KER_BUF_ADDR_DW0,
574                     (uint32_t)(addrp[i] & 0xffffffff),
575                     FSF_AZ_RX_KER_BUF_ADDR_DW1,
576                     (uint32_t)(addrp[i] >> 32));

578                 offset = id * sizeof (efx_qword_t);
579                 EFSYS_MEM_WRITEQ(erp->er_esmp, offset, &qword);

581                 id = (id + 1) & (erp->er_mask);
582         }
583 }

585                 void
586 efx_rx_qpush(
587         __in    efx_rxq_t *erp,
588         __in    unsigned int added)
589 {
```



```
590         efx_nic_t *enp = erp->er_enp;
591         uint32_t wptr;
592         efx_oword_t oword;
593         efx_dword_t dword;

595         EFSYS_ASSERT3U(erp->er_magic, ==, EFX_RXQ_MAGIC);

597         /* Guarantee ordering of memory (descriptors) and PIO (doorbell) */
598         EFSYS_PIO_WRITE_BARRIER();

600         /* Push the populated descriptors out */
601         wptr = added & erp->er_mask;

603         EFX_POPULATE_OWORD_1(oword, FRF_AZ_RX_DESC_WPTR, wptr);

605         /* Only write the third DWORD */
606         EFX_POPULATE_DWORD_1(dword,
607             EFX_DWORD_0, EFX_OWORD_FIELD(oword, EFX_DWORD_3));
608         EFX_BAR_TBL_WRITED3(enp, FR_BZ_RX_DESC_UPD_REGP0,
609                             erp->er_index, &dword, B_FALSE);
610 }

612                 void
613 efx_rx_qflush(
614         __in    efx_rxq_t *erp)
615 {
616         efx_nic_t *enp = erp->er_enp;
617         efx_oword_t oword;
618         uint32_t label;

620         EFSYS_ASSERT3U(erp->er_magic, ==, EFX_RXQ_MAGIC);

622         label = erp->er_index;

624         /* Flush the queue */
625         EFX_POPULATE_OWORD_2(oword, FRF_AZ_RX_FLUSH_DESCQ_CMD, 1,
626             FRF_AZ_RX_FLUSH_DESCQ, label);
627         EFX_BAR_WRITEO(enp, FR_AZ_RX_FLUSH_DESCQ_REG, &oword);
628 }

630                 void
631 efx_rx_qenable(
632         __in    efx_rxq_t *erp)
633 {
634         efx_nic_t *enp = erp->er_enp;
635         efx_oword_t oword;

637         EFSYS_ASSERT3U(erp->er_magic, ==, EFX_RXQ_MAGIC);

639         EFX_BAR_TBL_READO(enp, FR_AZ_RX_DESC_PTR_TBL,
640                           erp->er_index, &oword);

642         EFX_SET_OWORD_FIELD(oword, FRF_AZ_RX_DC_HW_RPTR, 0);
643         EFX_SET_OWORD_FIELD(oword, FRF_AZ_RX_DESCQ_HW_RPTR, 0);
644         EFX_SET_OWORD_FIELD(oword, FRF_AZ_RX_DESCQ_EN, 1);

646         EFX_BAR_TBL_WRITEO(enp, FR_AZ_RX_DESC_PTR_TBL,
647                           erp->er_index, &oword);
648 }

650         __checkReturn   int
651 efx_rx_qcreate(
652         __in            efx_nic_t *enp,
653         __in            unsigned int index,
654         __in            unsigned int label,
655         __in            efx_rxq_type_t type,
```

```
656         __in            efsys_mem_t *esmp,
657         __in            size_t n,
658         __in            uint32_t id,
659         __in            efx_evq_t *eep,
660         __deref_out     efx_rxq_t **erpp)
661 {
662         efx_nic_cfg_t *encp = &(enp->en_nic_cfg);
663         efx_rxq_t *erp;
664         efx_oword_t oword;
665         uint32_t size;
666         boolean_t split;
667         boolean_t jumbo;
668         int rc;

670         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
671         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_RX);

673         EFX_STATIC_ASSERT(EFX_EV_RX_NLABELS ==
674             (1 << FRF_AZ_RX_DESCQ_LABEL_WIDTH));
675         EFSYS_ASSERT3U(label, <, EFX_EV_RX_NLABELS);
676         EFSYS_ASSERT3U(enp->en_rx_qcount + 1, <, encp->enc_rxq_limit);

678         if (!ISP2(n) || !(n & EFX_RXQ_NDESCS_MASK)) {
679                 rc = EINVAL;
680                 goto fail1;
681         }
682         if (index >= encp->enc_rxq_limit) {
683                 rc = EINVAL;
684                 goto fail2;
685         }
686         for (size = 0; (1 << size) <= (EFX_RXQ_MAXNDESCS / EFX_RXQ_MINNDESCS);
687             size++)
688                 if ((1 << size) == (int)(n / EFX_RXQ_MINNDESCS))
689                         break;
690         if (id + (1 << size) >= encp->enc_buftbl_limit) {
691                 rc = EINVAL;
692                 goto fail3;
693         }

695         switch (type) {
696         case EFX_RXQ_TYPE_DEFAULT:
697                 split = B_FALSE;
698                 jumbo = B_FALSE;
699                 break;

701 #if EFSYS_OPT_RX_HDR_SPLIT
702         case EFX_RXQ_TYPE_SPLIT_HEADER:
703                 if ((enp->en_family < EFX_FAMILY_SIENA) || ((index & 1) != 0)) {
704                         rc = EINVAL;
705                         goto fail4;
706                 }
707                 split = B_TRUE;
708                 jumbo = B_TRUE;
709                 break;

711         case EFX_RXQ_TYPE_SPLIT_PAYLOAD:
712                 if ((enp->en_family < EFX_FAMILY_SIENA) || ((index & 1) == 0)) {
713                         rc = EINVAL;
714                         goto fail4;
715                 }
716                 split = B_FALSE;
717                 jumbo = B_TRUE;
718                 break;
719 #endif  /* EFSYS_OPT_RX_HDR_SPLIT */

721 #if EFSYS_OPT_RX_SCATTER
```



```
722         case EFX_RXQ_TYPE_SCATTER:
723                 if (enp->en_family < EFX_FAMILY_SIENA) {
724                         rc = EINVAL;
725                         goto fail4;
726                 }
727                 split = B_FALSE;
728                 jumbo = B_TRUE;
729                 break;
730 #endif  /* EFSYS_OPT_RX_SCATTER */

732         default:
733                 rc = EINVAL;
734                 goto fail4;
735         }

737         /* Allocate an RXQ object */
738         EFSYS_KMEM_ALLOC(enp->en_esip, sizeof (efx_rxq_t), erp);

740         if (erp == NULL) {
741                 rc = ENOMEM;
742                 goto fail5;
743         }

745         erp->er_magic = EFX_RXQ_MAGIC;
746         erp->er_enp = enp;
747         erp->er_index = index;
748         erp->er_mask = n - 1;
749         erp->er_esmp = esmp;

751         /* Set up the new descriptor queue */
752         EFX_POPULATE_OWORD_10(oword,
753             FRF_CZ_RX_HDR_SPLIT, split,
754             FRF_AZ_RX_ISCSI_DDIG_EN, 0,
755             FRF_AZ_RX_ISCSI_HDIG_EN, 0,
756             FRF_AZ_RX_DESCQ_BUF_BASE_ID, id,
757             FRF_AZ_RX_DESCQ_EVQ_ID, eep->ee_index,
758             FRF_AZ_RX_DESCQ_OWNER_ID, 0,
759             FRF_AZ_RX_DESCQ_LABEL, label,
760             FRF_AZ_RX_DESCQ_SIZE, size,
761             FRF_AZ_RX_DESCQ_TYPE, 0,
762             FRF_AZ_RX_DESCQ_JUMBO, jumbo);

764         EFX_BAR_TBL_WRITEO(enp, FR_AZ_RX_DESC_PTR_TBL,
765                            erp->er_index, &oword);

767         enp->en_rx_qcount++;
768         *erpp = erp;
769         return (0);

771 fail5:
772         EFSYS_PROBE(fail5);
773 fail4:
774         EFSYS_PROBE(fail4);
775 fail3:
776         EFSYS_PROBE(fail3);
777 fail2:
778         EFSYS_PROBE(fail2);
779 fail1:
780         EFSYS_PROBE1(fail1, int, rc);

782         return (rc);
783 }

785                 void
786 efx_rx_qdestroy(
787         __in    efx_rxq_t *erp)
```

```
788 {
789         efx_nic_t *enp = erp->er_enp;
790         efx_oword_t oword;

792         EFSYS_ASSERT3U(erp->er_magic, ==, EFX_RXQ_MAGIC);

794         EFSYS_ASSERT(enp->en_rx_qcount != 0);
795         --enp->en_rx_qcount;

797         /* Purge descriptor queue */
798         EFX_ZERO_OWORD(oword);

800         EFX_BAR_TBL_WRITEO(enp, FR_AZ_RX_DESC_PTR_TBL,
801                            erp->er_index, &oword);

803         /* Free the RXQ object */
804         EFSYS_KMEM_FREE(enp->en_esip, sizeof (efx_rxq_t), erp);
805 }

807                 void
808 efx_rx_fini(
809         __in    efx_nic_t *enp)
810 {
811         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
812         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_NIC);
813         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_RX);
814         EFSYS_ASSERT3U(enp->en_rx_qcount, ==, 0);

816         enp->en_mod_flags &= ~EFX_MOD_RX;
817 }
818 #endif /* ! codereview */
```

```
**********************************************************
    7302 Thu Aug 22 18:59:23 2013
new/usr/src/uts/common/io/sfxge/efx_sram.c
Merged sfxge driver
**********************************************************
```

```
/*
 * Copyright 2007-2013 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#include "efsys.h"
#include "efx.h"
#include "efx_types.h"
#include "efx_regs.h"
#include "efx_impl.h"

        __checkReturn   int
efx_sram_buf_tbl_set(
        __in            efx_nic_t *enp,
        __in            uint32_t id,
        __in            efsys_mem_t *esmp,
        __in            size_t n)
{
        efx_qword_t qword;
        uint32_t start = id;
        uint32_t stop = start + n;
        efsys_dma_addr_t addr;
        efx_oword_t oword;
        unsigned int count;
        int rc;

        EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
        EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_NIC);

        if (stop >= EFX_BUF_TBL_SIZE) {
                rc = EFBIG;
                goto fail1;
        }

        /* Add the entries into the buffer table */
        addr = EFSYS_MEM_ADDR(esmp);
        for (id = start; id != stop; id++) {
                EFX_POPULATE_QWORD_5(qword,
                    FRF_AZ_IP_DAT_BUF_SIZE, 0, FRF_AZ_BUF_ADR_REGION, 0,
                    FRF_AZ_BUF_ADR_FBUF_DW0,
                    (uint32_t)((addr >> 12) & 0xffffffff),
```



```
                    FRF_AZ_BUF_ADR_FBUF_DW1,
                    (uint32_t)((addr >> 12) >> 32),
                    FRF_AZ_BUF_OWNER_ID_FBUF, 0);

                EFX_BAR_TBL_WRITEQ(enp, FR_AZ_BUF_FULL_TBL,
                                   id, &qword);

                addr += EFX_BUF_SIZE;
        }

        EFSYS_PROBE2(buf, uint32_t, start, uint32_t, stop - 1);

        /* Flush the write buffer */
        EFX_POPULATE_OWORD_2(oword, FRF_AZ_BUF_UPD_CMD, 1,
            FRF_AZ_BUF_CLR_CMD, 0);
        EFX_BAR_WRITEO(enp, FR_AZ_BUF_TBL_UPD_REG, &oword);

        /* Poll for the last entry being written to the buffer table */
        EFSYS_ASSERT3U(id, ==, stop);
        addr -= EFX_BUF_SIZE;

        count = 0;
        do {
                EFSYS_PROBE1(wait, unsigned int, count);

                /* Spin for 1 ms */
                EFSYS_SPIN(1000);

                EFX_BAR_TBL_READQ(enp, FR_AZ_BUF_FULL_TBL,
                                  id - 1, &qword);

                if (EFX_QWORD_FIELD(qword, FRF_AZ_BUF_ADR_FBUF_DW0) ==
                    (uint32_t)((addr >> 12) & 0xffffffff) &&
                    EFX_QWORD_FIELD(qword, FRF_AZ_BUF_ADR_FBUF_DW1) ==
                    (uint32_t)((addr >> 12) >> 32))
                        goto verify;

        } while (++count < 100);

        rc = ETIMEDOUT;
        goto fail2;

verify:
        /* Verify the rest of the entries in the buffer table */
        while (--id != start) {
                addr -= EFX_BUF_SIZE;

                /* Read the buffer table entry */
                EFX_BAR_TBL_READQ(enp, FR_AZ_BUF_FULL_TBL,
                                  id - 1, &qword);

                if (EFX_QWORD_FIELD(qword, FRF_AZ_BUF_ADR_FBUF_DW0) !=
                    (uint32_t)((addr >> 12) & 0xffffffff) ||
                    EFX_QWORD_FIELD(qword, FRF_AZ_BUF_ADR_FBUF_DW1) !=
                    (uint32_t)((addr >> 12) >> 32)) {
                        rc = EFAULT;
                        goto fail3;
                }
        }

        return (0);

fail3:
        EFSYS_PROBE(fail3);

        id = stop;
```

```
129 fail2:
130         EFSYS_PROBE(fail2);

132         EFX_POPULATE_OWORD_4(oword, FRF_AZ_BUF_UPD_CMD, 0,
133             FRF_AZ_BUF_CLR_CMD, 1, FRF_AZ_BUF_CLR_END_ID, id - 1,
134             FRF_AZ_BUF_CLR_START_ID, start);
135         EFX_BAR_WRITEO(enp, FR_AZ_BUF_TBL_UPD_REG, &oword);

137 fail1:
138         EFSYS_PROBE1(fail1, int, rc);

140         return (rc);
141 }

143                 void
144 efx_sram_buf_tbl_clear(
145         __in    efx_nic_t *enp,
146         __in    uint32_t id,
147         __in    size_t n)
148 {
149         efx_oword_t oword;
150         uint32_t start = id;
151         uint32_t stop = start + n;

153         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
154         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_NIC);

156         EFSYS_ASSERT3U(stop, <, EFX_BUF_TBL_SIZE);

158         EFSYS_PROBE2(buf, uint32_t, start, uint32_t, stop - 1);

160         EFX_POPULATE_OWORD_4(oword, FRF_AZ_BUF_UPD_CMD, 0,
161             FRF_AZ_BUF_CLR_CMD, 1, FRF_AZ_BUF_CLR_END_ID, stop - 1,
162             FRF_AZ_BUF_CLR_START_ID, start);
163         EFX_BAR_WRITEO(enp, FR_AZ_BUF_TBL_UPD_REG, &oword);
164 }


167 #if EFSYS_OPT_DIAG

169 static                  void
170 efx_sram_byte_increment_set(
171         __in            size_t row,
172         __in            boolean_t negate,
173         __out           efx_qword_t *eqp)
174 {
175         size_t offset = row * FR_AZ_SRM_DBG_REG_STEP;
176         unsigned int index;

178         _NOTE(ARGUNUSED(negate))

180         for (index = 0; index < sizeof (efx_qword_t); index++)
181                 eqp->eq_u8[index] = offset + index;
182 }

184 static                  void
185 efx_sram_all_the_same_set(
186         __in            size_t row,
187         __in            boolean_t negate,
188         __out           efx_qword_t *eqp)
189 {
190         _NOTE(ARGUNUSED(row))

192         if (negate)
193                 EFX_SET_QWORD(*eqp);
```



```
194         else
195                 EFX_ZERO_QWORD(*eqp);
196 }

198 static                  void
199 efx_sram_bit_alternate_set(
200         __in            size_t row,
201         __in            boolean_t negate,
202         __out           efx_qword_t *eqp)
203 {
204         _NOTE(ARGUNUSED(row))

206         EFX_POPULATE_QWORD_2(*eqp,
207             EFX_DWORD_0, (negate) ? 0x55555555 : 0xaaaaaaaa,
208             EFX_DWORD_1, (negate) ? 0x55555555 : 0xaaaaaaaa);
209 }

211 static                  void
212 efx_sram_byte_alternate_set(
213         __in            size_t row,
214         __in            boolean_t negate,
215         __out           efx_qword_t *eqp)
216 {
217         _NOTE(ARGUNUSED(row))

219         EFX_POPULATE_QWORD_2(*eqp,
220             EFX_DWORD_0, (negate) ? 0x00ff00ff : 0xff00ff00,
221             EFX_DWORD_1, (negate) ? 0x00ff00ff : 0xff00ff00);
222 }

224 static                  void
225 efx_sram_byte_changing_set(
226         __in            size_t row,
227         __in            boolean_t negate,
228         __out           efx_qword_t *eqp)
229 {
230         size_t offset = row * FR_AZ_SRM_DBG_REG_STEP;
231         unsigned int index;

233         for (index = 0; index < sizeof (efx_qword_t); index++) {
234                 uint8_t byte;

236                 if (offset / 256 == 0)
237                         byte = (uint8_t)((offset % 257) % 256);
238                 else
239                         byte = (uint8_t)(~((offset - 8) % 257) % 256);

241                 eqp->eq_u8[index] = (negate) ? ~byte : byte;
242         }
243 }

245 static                  void
246 efx_sram_bit_sweep_set(
247         __in            size_t row,
248         __in            boolean_t negate,
249         __out           efx_qword_t *eqp)
250 {
251         size_t offset = row * FR_AZ_SRM_DBG_REG_STEP;

253         if (negate) {
254                 EFX_SET_QWORD(*eqp);
255                 EFX_CLEAR_QWORD_BIT(*eqp, (offset / sizeof (efx_qword_t)) % 64);
256         } else {
257                 EFX_ZERO_QWORD(*eqp);
258                 EFX_SET_QWORD_BIT(*eqp, (offset / sizeof (efx_qword_t)) % 64);
259         }
```

```
260 }

262 efx_sram_pattern_fn_t   __cs __efx_sram_pattern_fns[] = {
263         efx_sram_byte_increment_set,
264         efx_sram_all_the_same_set,
265         efx_sram_bit_alternate_set,
266         efx_sram_byte_alternate_set,
267         efx_sram_byte_changing_set,
268         efx_sram_bit_sweep_set
269 };

271         __checkReturn   int
272 efx_sram_test(
273         __in            efx_nic_t *enp,
274         __in            efx_pattern_type_t type)
275 {
276         efx_nic_ops_t *enop = enp->en_enop;
277         efx_sram_pattern_fn_t func;

279         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);

281         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_NIC);

283         EFSYS_ASSERT(!(enp->en_mod_flags & EFX_MOD_RX));
284         EFSYS_ASSERT(!(enp->en_mod_flags & EFX_MOD_TX));
285         EFSYS_ASSERT(!(enp->en_mod_flags & EFX_MOD_EV));

287         /* Select pattern generator */
288         EFSYS_ASSERT3U(type, <, EFX_PATTERN_NTYPES);
289         func = __efx_sram_pattern_fns[type];

291         return (enop->eno_sram_test(enp, func));
292 }

294 #endif  /* EFSYS_OPT_DIAG */
295 #endif /* ! codereview */
```

```
**********************************************************
   12006 Thu Aug 22 18:59:23 2013
new/usr/src/uts/common/io/sfxge/efx_tx.c
Merged sfxge driver
**********************************************************
```

```
/*
 * Copyright 2007-2013 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#include "efsys.h"
#include "efx.h"
#include "efx_types.h"
#include "efx_regs.h"
#include "efx_impl.h"

#if EFSYS_OPT_QSTATS
#define EFX_TX_QSTAT_INCR(_etp, _stat)                                  \
        do {                                                            \
                (_etp)->et_stat[_stat]++;                               \
        _NOTE(CONSTANTCONDITION)                                        \
        } while (B_FALSE)
#else
#define EFX_TX_QSTAT_INCR(_etp, _stat)
#endif



        __checkReturn   int
efx_tx_init(
        __in            efx_nic_t *enp)
{
        efx_oword_t oword;
        int rc;

        EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
        EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_NIC);

        if (!(enp->en_mod_flags & EFX_MOD_EV)) {
                rc = EINVAL;
                goto fail1;
        }

        if (enp->en_mod_flags & EFX_MOD_TX) {
                rc = EINVAL;
                goto fail2;
```



```
        }

        EFSYS_ASSERT3U(enp->en_tx_qcount, ==, 0);

        /*
         * Disable the timer-based TX DMA backoff and allow TX DMA to be
         * controlled by the RX FIFO fill level (although always allow a
         * minimal trickle).
         */
        EFX_BAR_READO(enp, FR_AZ_TX_RESERVED_REG, &oword);
        EFX_SET_OWORD_FIELD(oword, FRF_AZ_TX_RX_SPACER, 0xfe);
        EFX_SET_OWORD_FIELD(oword, FRF_AZ_TX_RX_SPACER_EN, 1);
        EFX_SET_OWORD_FIELD(oword, FRF_AZ_TX_ONE_PKT_PER_Q, 1);
        EFX_SET_OWORD_FIELD(oword, FRF_AZ_TX_PUSH_EN, 0);
        EFX_SET_OWORD_FIELD(oword, FRF_AZ_TX_DIS_NON_IP_EV, 1);
        EFX_SET_OWORD_FIELD(oword, FRF_AZ_TX_PREF_THRESHOLD, 2);
        EFX_SET_OWORD_FIELD(oword, FRF_AZ_TX_PREF_WD_TMR, 0x3fffff);

        /*
         * Filter all packets less than 14 bytes to avoid parsing
         * errors.
         */
        EFX_SET_OWORD_FIELD(oword, FRF_BZ_TX_FLUSH_MIN_LEN_EN, 1);
        EFX_BAR_WRITEO(enp, FR_AZ_TX_RESERVED_REG, &oword);

        /*
         * Do not set TX_NO_EOP_DISC_EN, since it limits packets to 16
         * descriptors (which is bad).
         */
        EFX_BAR_READO(enp, FR_AZ_TX_CFG_REG, &oword);
        EFX_SET_OWORD_FIELD(oword, FRF_AZ_TX_NO_EOP_DISC_EN, 0);
        EFX_BAR_WRITEO(enp, FR_AZ_TX_CFG_REG, &oword);

        enp->en_mod_flags |= EFX_MOD_TX;
        return (0);

fail2:
        EFSYS_PROBE(fail2);
fail1:
        EFSYS_PROBE1(fail1, int, rc);

        return (rc);
}

#if EFSYS_OPT_FILTER
extern  __checkReturn   int
efx_tx_filter_insert(
        __in            efx_txq_t *etp,
        __inout         efx_filter_spec_t *spec)
{
        EFSYS_ASSERT3U(etp->et_magic, ==, EFX_TXQ_MAGIC);
        EFSYS_ASSERT3P(spec, !=, NULL);

        spec->efs_dmaq_id = (uint16_t)etp->et_index;
        return (efx_filter_insert_filter(etp->et_enp, spec, B_FALSE));
}
#endif

#if EFSYS_OPT_FILTER
extern  __checkReturn   int
efx_tx_filter_remove(
        __in            efx_txq_t *etp,
        __inout         efx_filter_spec_t *spec)
{
        EFSYS_ASSERT3U(etp->et_magic, ==, EFX_TXQ_MAGIC);
        EFSYS_ASSERT3P(spec, !=, NULL);
```

```
129         spec->efs_dmaq_id = (uint16_t)etp->et_index;
130         return (efx_filter_remove_filter(etp->et_enp, spec));
131 }
132 #endif

134 #define EFX_TX_DESC(_etp, _addr, _size, _eop, _added)                   \
135         do {                                                            \
136                 unsigned int id;                                        \
137                 size_t offset;                                          \
138                 efx_qword_t qword;                                      \
139                                                                         \
140                 id = (_added)++ & (_etp)->et_mask;                      \
141                 offset = id * sizeof (efx_qword_t);                     \
142                                                                         \
143                 EFSYS_PROBE5(tx_post, unsigned int, (_etp)->et_index,   \
144                     unsigned int, id, efsys_dma_addr_t, (_addr),        \
145                     size_t, (_size), boolean_t, (_eop));                \
146                                                                         \
147                 EFX_POPULATE_QWORD_4(qword,                             \
148                     FSF_AZ_TX_KER_CONT, (_eop) ? 0 : 1,                 \
149                     FSF_AZ_TX_KER_BYTE_COUNT, (uint32_t)(_size),        \
150                     FSF_AZ_TX_KER_BUF_ADDR_DW0,                         \
151                     (uint32_t)((_addr) & 0xffffffff),                   \
152                     FSF_AZ_TX_KER_BUF_ADDR_DW1,                         \
153                     (uint32_t)((_addr) >> 32));                         \
154                 EFSYS_MEM_WRITEQ((_etp)->et_esmp, offset, &qword);      \
155                                                                         \
156                 _NOTE(CONSTANTCONDITION)                                \
157         } while (B_FALSE)

159         __checkReturn   int
160 efx_tx_qpost(
161         __in            efx_txq_t *etp,
162         __in_ecount(n)  efx_buffer_t *eb,
163         __in            unsigned int n,
164         __in            unsigned int completed,
165         __inout         unsigned int *addedp)
166 {
167         unsigned int added = *addedp;
168         unsigned int i;
169         int rc = ENOSPC;

171         EFSYS_ASSERT3U(etp->et_magic, ==, EFX_TXQ_MAGIC);

173         if (added - completed + n > EFX_TXQ_LIMIT(etp->et_mask + 1))
174                 goto fail1;

176         for (i = 0; i < n; i++) {
177                 efx_buffer_t *ebp = &eb[i];
178                 efsys_dma_addr_t start = ebp->eb_addr;
179                 size_t size = ebp->eb_size;
180                 efsys_dma_addr_t end = start + size;

182                 /* Fragments must not span 4k boundaries. */
183                 EFSYS_ASSERT(P2ROUNDUP(start + 1, 4096) >= end);

185                 EFX_TX_DESC(etp, start, size, ebp->eb_eop, added);
186         }

188         EFX_TX_QSTAT_INCR(etp, TX_POST);

190         *addedp = added;
191         return (0);

193 fail1:
```



```
194         EFSYS_PROBE1(fail1, int, rc);

196         return (rc);
197 }

199                 void
200 efx_tx_qpush(
201         __in    efx_txq_t *etp,
202         __in    unsigned int added)
203 {
204         efx_nic_t *enp = etp->et_enp;
205         uint32_t wptr;
206         efx_dword_t dword;
207         efx_oword_t oword;

209         EFSYS_ASSERT3U(etp->et_magic, ==, EFX_TXQ_MAGIC);

211         /* Guarantee ordering of memory (descriptors) and PIO (doorbell) */
212         EFSYS_PIO_WRITE_BARRIER();

214         /* Push the populated descriptors out */
215         wptr = added & etp->et_mask;

217         EFX_POPULATE_OWORD_1(oword, FRF_AZ_TX_DESC_WPTR, wptr);

219         /* Only write the third DWORD */
220         EFX_POPULATE_DWORD_1(dword,
221             EFX_DWORD_0, EFX_OWORD_FIELD(oword, EFX_DWORD_3));
222         EFX_BAR_TBL_WRITED3(enp, FR_BZ_TX_DESC_UPD_REGP0,
223                             etp->et_index, &dword, B_FALSE);
224 }

226 #define EFX_MAX_PACE_VALUE 20

228         __checkReturn   int
229 efx_tx_qpace(
230         __in            efx_txq_t *etp,
231         __in            unsigned int ns)
232 {
233         efx_nic_t *enp = etp->et_enp;
234         efx_nic_cfg_t *encp = &(enp->en_nic_cfg);
235         efx_oword_t oword;
236         unsigned int pace_val;
237         unsigned int timer_period;
238         int rc;

240         EFSYS_ASSERT3U(etp->et_magic, ==, EFX_TXQ_MAGIC);

242         if (ns == 0) {
243                 pace_val = 0;
244         } else {
245                 /*
246                  * The pace_val to write into the table is s.t
247                  * ns <= timer_period * (2 ^ pace_val)
248                  */
249                 timer_period = 104 / encp->enc_clk_mult;
250                 for (pace_val = 1; pace_val <= EFX_MAX_PACE_VALUE; pace_val++) {
251                         if ((timer_period << pace_val) >= ns)
252                                 break;
253                 }
254         }
255         if (pace_val > EFX_MAX_PACE_VALUE) {
256                 rc = EINVAL;
257                 goto fail1;
258         }
```

```
260         /* Update the pacing table */
261         EFX_POPULATE_OWORD_1(oword, FRF_AZ_TX_PACE, pace_val);
262         EFX_BAR_TBL_WRITEO(enp, FR_AZ_TX_PACE_TBL, etp->et_index, &oword);

264         return (0);

266 fail1:
267         EFSYS_PROBE1(fail1, int, rc);

269         return (rc);
270 }

272                 void
273 efx_tx_qflush(
274         __in    efx_txq_t *etp)
275 {
276         efx_nic_t *enp = etp->et_enp;
277         efx_oword_t oword;
278         uint32_t label;

280         EFSYS_ASSERT3U(etp->et_magic, ==, EFX_TXQ_MAGIC);

282         efx_tx_qpace(etp, 0);

284         label = etp->et_index;

286         /* Flush the queue */
287         EFX_POPULATE_OWORD_2(oword, FRF_AZ_TX_FLUSH_DESCQ_CMD, 1,
288             FRF_AZ_TX_FLUSH_DESCQ, label);
289         EFX_BAR_WRITEO(enp, FR_AZ_TX_FLUSH_DESCQ_REG, &oword);
290 }

292                 void
293 efx_tx_qenable(
294         __in    efx_txq_t *etp)
295 {
296         efx_nic_t *enp = etp->et_enp;
297         efx_oword_t oword;

299         EFSYS_ASSERT3U(etp->et_magic, ==, EFX_TXQ_MAGIC);

301         EFX_BAR_TBL_READO(enp, FR_AZ_TX_DESC_PTR_TBL,
302                           etp->et_index, &oword);

304         EFSYS_PROBE5(tx_descq_ptr, unsigned int, etp->et_index,
305             uint32_t, EFX_OWORD_FIELD(oword, EFX_DWORD_3),
306             uint32_t, EFX_OWORD_FIELD(oword, EFX_DWORD_2),
307             uint32_t, EFX_OWORD_FIELD(oword, EFX_DWORD_1),
308             uint32_t, EFX_OWORD_FIELD(oword, EFX_DWORD_0));

310         EFX_SET_OWORD_FIELD(oword, FRF_AZ_TX_DC_HW_RPTR, 0);
311         EFX_SET_OWORD_FIELD(oword, FRF_AZ_TX_DESCQ_HW_RPTR, 0);
312         EFX_SET_OWORD_FIELD(oword, FRF_AZ_TX_DESCQ_EN, 1);

314         EFX_BAR_TBL_WRITEO(enp, FR_AZ_TX_DESC_PTR_TBL,
315                           etp->et_index, &oword);
316 }

318         __checkReturn   int
319 efx_tx_qcreate(
320         __in            efx_nic_t *enp,
321         __in            unsigned int index,
322         __in            unsigned int label,
323         __in            efsys_mem_t *esmp,
324         __in            size_t n,
325         __in            uint32_t id,
```



```
326         __in            uint16_t flags,
327         __in            efx_evq_t *eep,
328         __deref_out     efx_txq_t **etpp)
329 {
330         efx_nic_cfg_t *encp = &(enp->en_nic_cfg);
331         efx_txq_t *etp;
332         efx_oword_t oword;
333         uint32_t size;
334         int rc;

336         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
337         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_TX);

339         EFX_STATIC_ASSERT(EFX_EV_TX_NLABELS ==
340             (1 << FRF_AZ_TX_DESCQ_LABEL_WIDTH));
341         EFSYS_ASSERT3U(label, <, EFX_EV_TX_NLABELS);
342         EFSYS_ASSERT3U(enp->en_tx_qcount + 1, <, encp->enc_txq_limit);

344         if (!ISP2(n) || !(n & EFX_TXQ_NDESCS_MASK)) {
345                 rc = EINVAL;
346                 goto fail1;
347         }
348         if (index >= encp->enc_txq_limit) {
349                 rc = EINVAL;
350                 goto fail2;
351         }
352         for (size = 0; (1 << size) <= (EFX_TXQ_MAXNDESCS / EFX_TXQ_MINNDESCS);
353             size++)
354                 if ((1 << size) == (int)(n / EFX_TXQ_MINNDESCS))
355                         break;
356         if (id + (1 << size) >= encp->enc_buftbl_limit) {
357                 rc = EINVAL;
358                 goto fail3;
359         }

361         /* Allocate an TXQ object */
362         EFSYS_KMEM_ALLOC(enp->en_esip, sizeof (efx_txq_t), etp);

364         if (etp == NULL) {
365                 rc = ENOMEM;
366                 goto fail4;
367         }

369         etp->et_magic = EFX_TXQ_MAGIC;
370         etp->et_enp = enp;
371         etp->et_index = index;
372         etp->et_mask = n - 1;
373         etp->et_esmp = esmp;

375         /* Set up the new descriptor queue */
376         EFX_POPULATE_OWORD_6(oword,
377             FRF_AZ_TX_DESCQ_BUF_BASE_ID, id,
378             FRF_AZ_TX_DESCQ_EVQ_ID, eep->ee_index,
379             FRF_AZ_TX_DESCQ_OWNER_ID, 0,
380             FRF_AZ_TX_DESCQ_LABEL, label,
381             FRF_AZ_TX_DESCQ_SIZE, size,
382             FRF_AZ_TX_DESCQ_TYPE, 0);

384         EFX_SET_OWORD_FIELD(oword, FRF_BZ_TX_NON_IP_DROP_DIS, 1);
385         EFX_SET_OWORD_FIELD(oword, FRF_BZ_TX_IP_CHKSM_DIS,
386             (flags & EFX_CKSUM_IPV4) ? 0 : 1);
387         EFX_SET_OWORD_FIELD(oword, FRF_BZ_TX_TCP_CHKSM_DIS,
388             (flags & EFX_CKSUM_TCPUDP) ? 0 : 1);

390         EFX_BAR_TBL_WRITEO(enp, FR_AZ_TX_DESC_PTR_TBL,
391             etp->et_index, &oword);
```

```
393         enp->en_tx_qcount++;
394         *etpp = etp;
395         return (0);

397 fail4:
398         EFSYS_PROBE(fail4);
399 fail3:
400         EFSYS_PROBE(fail3);
401 fail2:
402         EFSYS_PROBE(fail2);
403 fail1:
404         EFSYS_PROBE1(fail1, int, rc);

406         return (rc);
407 }

409 #if EFSYS_OPT_NAMES
410 /* START MKCONFIG GENERATED EfxTransmitQueueStatNamesBlock 78ca9ab00287fffb */
411 static const char       __cs * __cs __efx_tx_qstat_name[] = {
412         "post",
413         "unaligned_split",
414 };
415 /* END MKCONFIG GENERATED EfxTransmitQueueStatNamesBlock */

417                 const char __cs *
418 efx_tx_qstat_name(
419         __in    efx_nic_t *enp,
420         __in    unsigned int id)
421 {
422         _NOTE(ARGUNUSED(enp))
423         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
424         EFSYS_ASSERT3U(id, <, TX_NQSTATS);

426         return (__efx_tx_qstat_name[id]);
427 }
428 #endif  /* EFSYS_OPT_NAMES */

430 #if EFSYS_OPT_QSTATS
431                                         void
432 efx_tx_qstats_update(
433         __in                            efx_txq_t *etp,
434         __inout_ecount(TX_NQSTATS)      efsys_stat_t *stat)
435 {
436         unsigned int id;

438         EFSYS_ASSERT3U(etp->et_magic, ==, EFX_TXQ_MAGIC);

440         for (id = 0; id < TX_NQSTATS; id++) {
441                 efsys_stat_t *essp = &stat[id];

443                 EFSYS_STAT_INCR(essp, etp->et_stat[id]);
444                 etp->et_stat[id] = 0;
445         }
446 }
447 #endif  /* EFSYS_OPT_QSTATS */

449                 void
450 efx_tx_qdestroy(
451         __in    efx_txq_t *etp)
452 {
453         efx_nic_t *enp = etp->et_enp;
454         efx_oword_t oword;

456         EFSYS_ASSERT3U(etp->et_magic, ==, EFX_TXQ_MAGIC);
```



```
458         EFSYS_ASSERT(enp->en_tx_qcount != 0);
459         --enp->en_tx_qcount;

461         /* Purge descriptor queue */
462         EFX_ZERO_OWORD(oword);

464         EFX_BAR_TBL_WRITEO(enp, FR_AZ_TX_DESC_PTR_TBL,
465                            etp->et_index, &oword);

467         /* Free the TXQ object */
468         EFSYS_KMEM_FREE(enp->en_esip, sizeof (efx_txq_t), etp);
469 }

471                 void
472 efx_tx_fini(
473         __in    efx_nic_t *enp)
474 {
475         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
476         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_NIC);
477         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_TX);
478         EFSYS_ASSERT3U(enp->en_tx_qcount, ==, 0);

480         enp->en_mod_flags &= ~EFX_MOD_TX;
481 }
482 #endif /* ! codereview */
```

```
**********************************************************
   56990 Thu Aug 22 18:59:23 2013
new/usr/src/uts/common/io/sfxge/efx_types.h
Merged sfxge driver
**********************************************************
```

```
/*-
 * Copyright 2007-2013 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 *
 * Ackowledgement to Fen Systems Ltd.
 */

#ifndef _SYS_EFX_TYPES_H
#define _SYS_EFX_TYPES_H

#include "efsys.h"

#ifdef  __cplusplus
extern "C" {
#endif

/*
 * Bitfield access
 *
 * Solarflare NICs make extensive use of bitfields up to 128 bits
 * wide.  Since there is no native 128-bit datatype on most systems,
 * and since 64-bit datatypes are inefficient on 32-bit systems and
 * vice versa, we wrap accesses in a way that uses the most efficient
 * datatype.
 *
 * The NICs are PCI devices and therefore little-endian.  Since most
 * of the quantities that we deal with are DMAed to/from host memory,
 * we define    our datatypes (efx_oword_t, efx_qword_t and efx_dword_t)
 * to be little-endian.
 *
 * In the less common case of using PIO for individual register
 * writes, we construct the little-endian datatype in host memory and
 * then use non-swapping register access primitives, rather than
 * constructing a native-endian datatype and relying on implicit
 * byte-swapping.  (We use a similar strategy for register reads.)
 */

/*
 * NOTE: Field definitions here and elsewhere are done in terms of a lowest
 *       bit number (LBN) and a width.
 */
```



```
#define EFX_DUMMY_FIELD_LBN 0
#define EFX_DUMMY_FIELD_WIDTH 0

#define EFX_BYTE_0_LBN 0
#define EFX_BYTE_0_WIDTH 8

#define EFX_BYTE_1_LBN 8
#define EFX_BYTE_1_WIDTH 8

#define EFX_BYTE_2_LBN 16
#define EFX_BYTE_2_WIDTH 8

#define EFX_BYTE_3_LBN 24
#define EFX_BYTE_3_WIDTH 8

#define EFX_BYTE_4_LBN 32
#define EFX_BYTE_4_WIDTH 8

#define EFX_BYTE_5_LBN 40
#define EFX_BYTE_5_WIDTH 8

#define EFX_BYTE_6_LBN 48
#define EFX_BYTE_6_WIDTH 8

#define EFX_BYTE_7_LBN 56
#define EFX_BYTE_7_WIDTH 8

#define EFX_WORD_0_LBN 0
#define EFX_WORD_0_WIDTH 16

#define EFX_WORD_1_LBN 16
#define EFX_WORD_1_WIDTH 16

#define EFX_WORD_2_LBN 32
#define EFX_WORD_2_WIDTH 16

#define EFX_WORD_3_LBN 48
#define EFX_WORD_3_WIDTH 16

#define EFX_DWORD_0_LBN 0
#define EFX_DWORD_0_WIDTH 32

#define EFX_DWORD_1_LBN 32
#define EFX_DWORD_1_WIDTH 32

#define EFX_DWORD_2_LBN 64
#define EFX_DWORD_2_WIDTH 32

#define EFX_DWORD_3_LBN 96
#define EFX_DWORD_3_WIDTH 32

#define EFX_QWORD_0_LBN 0
#define EFX_QWORD_0_WIDTH 64

#define EFX_QWORD_1_LBN 64
#define EFX_QWORD_1_WIDTH 64

/* Specified attribute (i.e. LBN ow WIDTH) of the specified field */
#define EFX_VAL(_field, _attribute)                                     \
        _field ## _ ## _attribute

/* Lowest bit number of the specified field */
#define EFX_LOW_BIT(_field)                                             \
        EFX_VAL(_field, LBN)
```

```
/* Width of the specified field */
#define EFX_WIDTH(_field)                                               \
        EFX_VAL(_field, WIDTH)

/* Highest bit number of the specified field */
#define EFX_HIGH_BIT(_field)                                            \
        (EFX_LOW_BIT(_field) + EFX_WIDTH(_field) - 1)

/*
 * 64-bit mask equal in width to the specified field.
 *
 * For example, a field with width 5 would have a mask of 0x000000000000001f.
 */
#define EFX_MASK64(_field)                                              \
        ((EFX_WIDTH(_field) == 64) ? ~((uint64_t)0) :                   \
            (((((uint64_t)1) << EFX_WIDTH(_field))) - 1))
/*
 * 32-bit mask equal in width to the specified field.
 *
 * For example, a field with width 5 would have a mask of 0x0000001f.
 */
#define EFX_MASK32(_field)                                              \
        ((EFX_WIDTH(_field) == 32) ? ~((uint32_t)0) :                   \
            (((((uint32_t)1) << EFX_WIDTH(_field))) - 1))

/*
 * 16-bit mask equal in width to the specified field.
 *
 * For example, a field with width 5 would have a mask of 0x001f.
 */
#define EFX_MASK16(_field)                                              \
        ((EFX_WIDTH(_field) == 16) ? 0xffffu :                          \
            (uint16_t)((1 << EFX_WIDTH(_field)) - 1))

/*
 * 8-bit mask equal in width to the specified field.
 *
 * For example, a field with width 5 would have a mask of 0x1f.
 */
#define EFX_MASK8(_field)                                               \
        ((uint8_t)((1 << EFX_WIDTH(_field)) - 1))

#pragma pack(1)

/*
 * A byte (i.e. 8-bit) datatype
 */
typedef union efx_byte_u {
        uint8_t eb_u8[1];
} efx_byte_t;

/*
 * A word (i.e. 16-bit) datatype
 *
 * This datatype is defined to be little-endian.
 */
typedef union efx_word_u {
        efx_byte_t ew_byte[2];
        uint16_t ew_u16[1];
        uint8_t ew_u8[2];
} efx_word_t;

/*
 * A doubleword (i.e. 32-bit) datatype
 *
 * This datatype is defined to be little-endian.
```



```
 */
typedef union efx_dword_u {
        efx_byte_t ed_byte[4];
        efx_word_t ed_word[2];
        uint32_t ed_u32[1];
        uint16_t ed_u16[2];
        uint8_t ed_u8[4];
} efx_dword_t;

/*
 * A quadword (i.e. 64-bit) datatype
 *
 * This datatype is defined to be little-endian.
 */
typedef union efx_qword_u {
        efx_byte_t eq_byte[8];
        efx_word_t eq_word[4];
        efx_dword_t eq_dword[2];
#if EFSYS_HAS_UINT64
        uint64_t eq_u64[1];
#endif
        uint32_t eq_u32[2];
        uint16_t eq_u16[4];
        uint8_t eq_u8[8];
} efx_qword_t;

/*
 * An octword (i.e. 128-bit) datatype
 *
 * This datatype is defined to be little-endian.
 */
typedef union efx_oword_u {
        efx_byte_t eo_byte[16];
        efx_word_t eo_word[8];
        efx_dword_t eo_dword[4];
        efx_qword_t eo_qword[2];
#if EFSYS_HAS_UINT64
        uint64_t eo_u64[2];
#endif
        uint32_t eo_u32[4];
        uint16_t eo_u16[8];
        uint8_t eo_u8[16];
} efx_oword_t;

#pragma pack()

#define __SWAP16(_x)                            \
        ((((_x) & 0xff) << 8) |                 \
        (((_x) >> 8) & 0xff))

#define __SWAP32(_x)                            \
        ((__SWAP16((_x) & 0xffff) << 16) |      \
        __SWAP16(((_x) >> 16) & 0xffff))

#define __SWAP64(_x)                            \
        ((__SWAP32((_x) & 0xffffffff) << 32) |  \
        __SWAP32(((_x) >> 32) & 0xffffffff))

#define __NOSWAP16(_x)          (_x)
#define __NOSWAP32(_x)          (_x)
#define __NOSWAP64(_x)          (_x)

#if EFSYS_IS_BIG_ENDIAN

#define __CPU_TO_LE_16(_x)      (uint16_t)__SWAP16(_x)
#define __LE_TO_CPU_16(_x)      (uint16_t)__SWAP16(_x)
```

```
260 #define __CPU_TO_BE_16(_x)      (uint16_t)__NOSWAP16(_x)
261 #define __BE_TO_CPU_16(_x)      (uint16_t)__NOSWAP16(_x)

263 #define __CPU_TO_LE_32(_x)      (uint32_t)__SWAP32(_x)
264 #define __LE_TO_CPU_32(_x)      (uint32_t)__SWAP32(_x)
265 #define __CPU_TO_BE_32(_x)      (uint32_t)__NOSWAP32(_x)
266 #define __BE_TO_CPU_32(_x)      (uint32_t)__NOSWAP32(_x)

268 #define __CPU_TO_LE_64(_x)      (uint64_t)__SWAP64(_x)
269 #define __LE_TO_CPU_64(_x)      (uint64_t)__SWAP64(_x)
270 #define __CPU_TO_BE_64(_x)      (uint64_t)__NOSWAP64(_x)
271 #define __BE_TO_CPU_64(_x)      (uint64_t)__NOSWAP64(_x)

273 #elif EFSYS_IS_LITTLE_ENDIAN

275 #define __CPU_TO_LE_16(_x)      (uint16_t)__NOSWAP16(_x)
276 #define __LE_TO_CPU_16(_x)      (uint16_t)__NOSWAP16(_x)
277 #define __CPU_TO_BE_16(_x)      (uint16_t)__SWAP16(_x)
278 #define __BE_TO_CPU_16(_x)      (uint16_t)__SWAP16(_x)

280 #define __CPU_TO_LE_32(_x)      (uint32_t)__NOSWAP32(_x)
281 #define __LE_TO_CPU_32(_x)      (uint32_t)__NOSWAP32(_x)
282 #define __CPU_TO_BE_32(_x)      (uint32_t)__SWAP32(_x)
283 #define __BE_TO_CPU_32(_x)      (uint32_t)__SWAP32(_x)

285 #define __CPU_TO_LE_64(_x)      (uint64_t)__NOSWAP64(_x)
286 #define __LE_TO_CPU_64(_x)      (uint64_t)__NOSWAP64(_x)
287 #define __CPU_TO_BE_64(_x)      (uint64_t)__SWAP64(_x)
288 #define __BE_TO_CPU_64(_x)      (uint64_t)__SWAP64(_x)

290 #else

292 #error "Neither of EFSYS_IS_{BIG,LITTLE}_ENDIAN is set"

294 #endif

296 #define __NATIVE_8(_x)  (uint8_t)(_x)

298 /* Format string for printing an efx_byte_t */
299 #define EFX_BYTE_FMT "0x%02x"

301 /* Format string for printing an efx_word_t */
302 #define EFX_WORD_FMT "0x%04x"

304 /* Format string for printing an efx_dword_t */
305 #define EFX_DWORD_FMT "0x%08x"

307 /* Format string for printing an efx_qword_t */
308 #define EFX_QWORD_FMT "0x%08x:%08x"

310 /* Format string for printing an efx_oword_t */
311 #define EFX_OWORD_FMT "0x%08x:%08x:%08x:%08x"

313 /* Parameters for printing an efx_byte_t */
314 #define EFX_BYTE_VAL(_byte)                                     \
315         ((unsigned int)__NATIVE_8((_byte).eb_u8[0]))

317 /* Parameters for printing an efx_word_t */
318 #define EFX_WORD_VAL(_word)                                     \
319         ((unsigned int)__LE_TO_CPU_16((_word).ew_u16[0]))

321 /* Parameters for printing an efx_dword_t */
322 #define EFX_DWORD_VAL(_dword)                                   \
323         ((unsigned int)__LE_TO_CPU_32((_dword).ed_u32[0]))

325 /* Parameters for printing an efx_qword_t */
```



```
326 #define EFX_QWORD_VAL(_qword)                                   \
327         ((unsigned int)__LE_TO_CPU_32((_qword).eq_u32[1])),     \
328         ((unsigned int)__LE_TO_CPU_32((_qword).eq_u32[0]))

330 /* Parameters for printing an efx_oword_t */
331 #define EFX_OWORD_VAL(_oword)                                   \
332         ((unsigned int)__LE_TO_CPU_32((_oword).eo_u32[3])),     \
333         ((unsigned int)__LE_TO_CPU_32((_oword).eo_u32[2])),     \
334         ((unsigned int)__LE_TO_CPU_32((_oword).eo_u32[1])),     \
335         ((unsigned int)__LE_TO_CPU_32((_oword).eo_u32[0]))

337 /*
338  * Stop lint complaining about some shifts.
339  */
340 #ifdef  __lint
341 extern int fix_lint;
342 #define FIX_LINT(_x)    (_x + fix_lint)
343 #else
344 #define FIX_LINT(_x)    (_x)
345 #endif

347 /*
348  * Extract bit field portion [low,high) from the native-endian element
349  * which contains bits [min,max).
350  *
351  * For example, suppose "element" represents the high 32 bits of a
352  * 64-bit value, and we wish to extract the bits belonging to the bit
353  * field occupying bits 28-45 of this 64-bit value.
354  *
355  * Then EFX_EXTRACT(_element, 32, 63, 28, 45) would give
356  *
357  *   (_element) << 4
358  *
359  * The result will contain the relevant bits filled in in the range
360  * [0,high-low), with garbage in bits [high-low+1,...).
361  */
362 #define EFX_EXTRACT_NATIVE(_element, _min, _max, _low, _high)           \
363         ((FIX_LINT(_low > _max) || FIX_LINT(_high < _min)) ?            \
364                 0U :                                                    \
365                 ((_low > _min) ?                                        \
366                         ((_element) >> (_low - _min)) :                 \
367                         ((_element) << (_min - _low))))

369 /*
370  * Extract bit field portion [low,high) from the 64-bit little-endian
371  * element which contains bits [min,max)
372  */
373 #define EFX_EXTRACT64(_element, _min, _max, _low, _high)                \
374         EFX_EXTRACT_NATIVE(__LE_TO_CPU_64(_element), _min, _max, _low, _high)

376 /*
377  * Extract bit field portion [low,high) from the 32-bit little-endian
378  * element which contains bits [min,max)
379  */
380 #define EFX_EXTRACT32(_element, _min, _max, _low, _high)                \
381         EFX_EXTRACT_NATIVE(__LE_TO_CPU_32(_element), _min, _max, _low, _high)

383 /*
384  * Extract bit field portion [low,high) from the 16-bit little-endian
385  * element which contains bits [min,max)
386  */
387 #define EFX_EXTRACT16(_element, _min, _max, _low, _high)                \
388         EFX_EXTRACT_NATIVE(__LE_TO_CPU_16(_element), _min, _max, _low, _high)

390 /*
391  * Extract bit field portion [low,high) from the 8-bit
```

```
392  * element which contains bits [min,max)
393  */
394 #define EFX_EXTRACT8(_element, _min, _max, _low, _high)                 \
395         EFX_EXTRACT_NATIVE(__NATIVE_8(_element), _min, _max, _low, _high)
396
397 #define EFX_EXTRACT_OWORD64(_oword, _low, _high)                        \
398         (EFX_EXTRACT64((_oword).eo_u64[0], FIX_LINT(0), FIX_LINT(63),   \
399             _low, _high) |                                              \
400         EFX_EXTRACT64((_oword).eo_u64[1], FIX_LINT(64), FIX_LINT(127),  \
401             _low, _high))
402
403 #define EFX_EXTRACT_OWORD32(_oword, _low, _high)                        \
404         (EFX_EXTRACT32((_oword).eo_u32[0], FIX_LINT(0), FIX_LINT(31),   \
405             _low, _high) |                                              \
406         EFX_EXTRACT32((_oword).eo_u32[1], FIX_LINT(32), FIX_LINT(63),   \
407             _low, _high) |                                              \
408         EFX_EXTRACT32((_oword).eo_u32[2], FIX_LINT(64), FIX_LINT(95),   \
409             _low, _high) |                                              \
410         EFX_EXTRACT32((_oword).eo_u32[3], FIX_LINT(96), FIX_LINT(127),  \
411             _low, _high))
412
413 #define EFX_EXTRACT_QWORD64(_qword, _low, _high)                        \
414         (EFX_EXTRACT64((_qword).eq_u64[0], FIX_LINT(0), FIX_LINT(63),   \
415             _low, _high))
416
417 #define EFX_EXTRACT_QWORD32(_qword, _low, _high)                        \
418         (EFX_EXTRACT32((_qword).eq_u32[0], FIX_LINT(0), FIX_LINT(31),   \
419             _low, _high) |                                              \
420         EFX_EXTRACT32((_qword).eq_u32[1], FIX_LINT(32), FIX_LINT(63),   \
421             _low, _high))
422
423 #define EFX_EXTRACT_DWORD(_dword, _low, _high)                          \
424         (EFX_EXTRACT32((_dword).ed_u32[0], FIX_LINT(0), FIX_LINT(31),   \
425             _low, _high))
426
427 #define EFX_EXTRACT_WORD(_word, _low, _high)                            \
428         (EFX_EXTRACT16((_word).ew_u16[0], FIX_LINT(0), FIX_LINT(15),    \
429             _low, _high))
430
431 #define EFX_EXTRACT_BYTE(_byte, _low, _high)                            \
432         (EFX_EXTRACT8((_byte).eb_u8[0], FIX_LINT(0), FIX_LINT(7),       \
433             _low, _high))
434
435
436 #define EFX_OWORD_FIELD64(_oword, _field)                               \
437         ((uint32_t)EFX_EXTRACT_OWORD64(_oword, EFX_LOW_BIT(_field),     \
438             EFX_HIGH_BIT(_field)) & EFX_MASK32(_field))
439
440 #define EFX_OWORD_FIELD32(_oword, _field)                               \
441         (EFX_EXTRACT_OWORD32(_oword, EFX_LOW_BIT(_field),               \
442             EFX_HIGH_BIT(_field)) & EFX_MASK32(_field))
443
444 #define EFX_QWORD_FIELD64(_qword, _field)                               \
445         ((uint32_t)EFX_EXTRACT_QWORD64(_qword, EFX_LOW_BIT(_field),     \
446             EFX_HIGH_BIT(_field)) & EFX_MASK32(_field))
447
448 #define EFX_QWORD_FIELD32(_qword, _field)                               \
449         (EFX_EXTRACT_QWORD32(_qword, EFX_LOW_BIT(_field),               \
450             EFX_HIGH_BIT(_field)) & EFX_MASK32(_field))
451
452 #define EFX_DWORD_FIELD(_dword, _field)                                 \
453         (EFX_EXTRACT_DWORD(_dword, EFX_LOW_BIT(_field),                 \
454             EFX_HIGH_BIT(_field)) & EFX_MASK32(_field))
455
456 #define EFX_WORD_FIELD(_word, _field)                                   \
457         (EFX_EXTRACT_WORD(_word, EFX_LOW_BIT(_field),                   \
```



```
458             EFX_HIGH_BIT(_field)) & EFX_MASK16(_field))
459
460 #define EFX_BYTE_FIELD(_byte, _field)                                   \
461         (EFX_EXTRACT_BYTE(_byte, EFX_LOW_BIT(_field),                   \
462             EFX_HIGH_BIT(_field)) & EFX_MASK8(_field))
463
464
465 #define EFX_OWORD_IS_EQUAL64(_oword_a, _oword_b)                        \
466         ((_oword_a).eo_u64[0] == (_oword_b).eo_u64[0] &&                \
467             (_oword_a).eo_u64[1] == (_oword_b).eo_u64[1])
468
469 #define EFX_OWORD_IS_EQUAL32(_oword_a, _oword_b)                        \
470         ((_oword_a).eo_u32[0] == (_oword_b).eo_u32[0] &&                \
471             (_oword_a).eo_u32[1] == (_oword_b).eo_u32[1] &&             \
472             (_oword_a).eo_u32[2] == (_oword_b).eo_u32[2] &&             \
473             (_oword_a).eo_u32[3] == (_oword_b).eo_u32[3])
474
475 #define EFX_QWORD_IS_EQUAL64(_qword_a, _qword_b)                        \
476         ((_qword_a).eq_u64[0] == (_qword_b).eq_u64[0])
477
478 #define EFX_QWORD_IS_EQUAL32(_qword_a, _qword_b)                        \
479         ((_qword_a).eq_u32[0] == (_qword_b).eq_u32[0] &&                \
480             (_qword_a).eq_u32[1] == (_qword_b).eq_u32[1])
481
482 #define EFX_DWORD_IS_EQUAL(_dword_a, _dword_b)                          \
483         ((_dword_a).ed_u32[0] == (_dword_b).ed_u32[0])
484
485 #define EFX_WORD_IS_EQUAL(_word_a, _word_b)                             \
486         ((_word_a).ew_u16[0] == (_word_b).ew_u16[0])
487
488 #define EFX_BYTE_IS_EQUAL(_byte_a, _byte_b)                             \
489         ((_byte_a).eb_u8[0] == (_byte_b).eb_u8[0])
490
491
492 #define EFX_OWORD_IS_ZERO64(_oword)                                     \
493         (((_oword).eo_u64[0] |                                          \
494             (_oword).eo_u64[1]) == 0)
495
496 #define EFX_OWORD_IS_ZERO32(_oword)                                     \
497         (((_oword).eo_u32[0] |                                          \
498             (_oword).eo_u32[1] |                                        \
499             (_oword).eo_u32[2] |                                        \
500             (_oword).eo_u32[3]) == 0)
501
502 #define EFX_QWORD_IS_ZERO64(_qword)                                     \
503         (((_qword).eq_u64[0]) == 0)
504
505 #define EFX_QWORD_IS_ZERO32(_qword)                                     \
506         (((_qword).eq_u32[0] |                                          \
507             (_qword).eq_u32[1]) == 0)
508
509 #define EFX_DWORD_IS_ZERO(_dword)                                       \
510         (((_dword).ed_u32[0]) == 0)
511
512 #define EFX_WORD_IS_ZERO(_word)                                         \
513         (((_word).ew_u16[0]) == 0)
514
515 #define EFX_BYTE_IS_ZERO(_byte)                                         \
516         (((_byte).eb_u8[0]) == 0)
517
518
519 #define EFX_OWORD_IS_SET64(_oword)                                      \
520         (((_oword).eo_u64[0] &                                          \
521             (_oword).eo_u64[1]) == ~((uint64_t)0))
522
523 #define EFX_OWORD_IS_SET32(_oword)                                      \
```

```
524         (((_oword).eo_u32[0] &                                          \
525             (_oword).eo_u32[1] &                                        \
526             (_oword).eo_u32[2] &                                        \
527             (_oword).eo_u32[3]) == ~((uint32_t)0))
528
529 #define EFX_QWORD_IS_SET64(_qword)                                      \
530         (((_qword).eq_u64[0]) == ~((uint32_t)0))
531
532 #define EFX_QWORD_IS_SET32(_qword)                                      \
533         (((_qword).eq_u32[0] &                                          \
534             (_qword).eq_u32[1]) == ~((uint32_t)0))
535
536 #define EFX_DWORD_IS_SET(_dword)                                        \
537         ((_dword).ed_u32[0] == ~((uint32_t)0))
538
539 #define EFX_WORD_IS_SET(_word)                                          \
540         ((_word).ew_u16[0] == ~((uint16_t)0))
541
542 #define EFX_BYTE_IS_SET(_byte)                                          \
543         ((_byte).eb_u8[0] == ~((uint8_t)0))
544
545 /*
546  * Construct bit field portion
547  *
548  * Creates the portion of the bit field [low,high) that lies within
549  * the range [min,max).
550  */
551
552 #define EFX_INSERT_NATIVE64(_min, _max, _low, _high, _value)            \
553         (((_low > _max) || (_high < _min)) ?                            \
554                 0U :                                                    \
555                 ((_low > _min) ?                                        \
556                         (((uint64_t)(_value)) << (_low - _min)) :       \
557                         (((uint64_t)(_value)) >> (_min - _low))))
558
559 #define EFX_INSERT_NATIVE32(_min, _max, _low, _high, _value)            \
560         (((_low > _max) || (_high < _min)) ?                            \
561                 0U :                                                    \
562                 ((_low > _min) ?                                        \
563                         (((uint32_t)(_value)) << (_low - _min)) :       \
564                         (((uint32_t)(_value)) >> (_min - _low))))
565
566 #define EFX_INSERT_NATIVE16(_min, _max, _low, _high, _value)            \
567         (((_low > _max) || (_high < _min)) ?                            \
568                 0U :                                                    \
569                 (uint16_t)((_low > _min) ?                              \
570                                 ((_value) << (_low - _min)) :           \
571                                 ((_value) >> (_min - _low))))
572
573 #define EFX_INSERT_NATIVE8(_min, _max, _low, _high, _value)             \
574         (((_low > _max) || (_high < _min)) ?                            \
575                 0U :                                                    \
576                 (uint8_t)((_low > _min) ?                               \
577                                 ((_value) << (_low - _min)) :   \
578                                 ((_value) >> (_min - _low))))
579
580 /*
581  * Construct bit field portion
582  *
583  * Creates the portion of the named bit field that lies within the
584  * range [min,max).
585  */
586 #define EFX_INSERT_FIELD_NATIVE64(_min, _max, _field, _value)           \
587         EFX_INSERT_NATIVE64(_min, _max, EFX_LOW_BIT(_field),            \
588             EFX_HIGH_BIT(_field), _value)
```



```
589
590 #define EFX_INSERT_FIELD_NATIVE32(_min, _max, _field, _value)           \
591         EFX_INSERT_NATIVE32(_min, _max, EFX_LOW_BIT(_field),            \
592             EFX_HIGH_BIT(_field), _value)
593
594 #define EFX_INSERT_FIELD_NATIVE16(_min, _max, _field, _value)           \
595         EFX_INSERT_NATIVE16(_min, _max, EFX_LOW_BIT(_field),            \
596             EFX_HIGH_BIT(_field), _value)
597
598 #define EFX_INSERT_FIELD_NATIVE8(_min, _max, _field, _value)            \
599         EFX_INSERT_NATIVE8(_min, _max, EFX_LOW_BIT(_field),             \
600             EFX_HIGH_BIT(_field), _value)
601
602 /*
603  * Construct bit field
604  *
605  * Creates the portion of the named bit fields that lie within the
606  * range [min,max).
607  */
608 #define EFX_INSERT_FIELDS64(_min, _max,                                 \
609             _field1, _value1, _field2, _value2, _field3, _value3,       \
610             _field4, _value4, _field5, _value5, _field6, _value6,       \
611             _field7, _value7, _field8, _value8, _field9, _value9,       \
612             _field10, _value10)                                         \
613         __CPU_TO_LE_64(                                                 \
614             EFX_INSERT_FIELD_NATIVE64(_min, _max, _field1, _value1) |   \
615             EFX_INSERT_FIELD_NATIVE64(_min, _max, _field2, _value2) |   \
616             EFX_INSERT_FIELD_NATIVE64(_min, _max, _field3, _value3) |   \
617             EFX_INSERT_FIELD_NATIVE64(_min, _max, _field4, _value4) |   \
618             EFX_INSERT_FIELD_NATIVE64(_min, _max, _field5, _value5) |   \
619             EFX_INSERT_FIELD_NATIVE64(_min, _max, _field6, _value6) |   \
620             EFX_INSERT_FIELD_NATIVE64(_min, _max, _field7, _value7) |   \
621             EFX_INSERT_FIELD_NATIVE64(_min, _max, _field8, _value8) |   \
622             EFX_INSERT_FIELD_NATIVE64(_min, _max, _field9, _value9) |   \
623             EFX_INSERT_FIELD_NATIVE64(_min, _max, _field10, _value10))
624
625 #define EFX_INSERT_FIELDS32(_min, _max,                                 \
626             _field1, _value1, _field2, _value2, _field3, _value3,       \
627             _field4, _value4, _field5, _value5, _field6, _value6,       \
628             _field7, _value7, _field8, _value8, _field9, _value9,       \
629             _field10, _value10)                                         \
630         __CPU_TO_LE_32(                                                 \
631             EFX_INSERT_FIELD_NATIVE32(_min, _max, _field1, _value1) |   \
632             EFX_INSERT_FIELD_NATIVE32(_min, _max, _field2, _value2) |   \
633             EFX_INSERT_FIELD_NATIVE32(_min, _max, _field3, _value3) |   \
634             EFX_INSERT_FIELD_NATIVE32(_min, _max, _field4, _value4) |   \
635             EFX_INSERT_FIELD_NATIVE32(_min, _max, _field5, _value5) |   \
636             EFX_INSERT_FIELD_NATIVE32(_min, _max, _field6, _value6) |   \
637             EFX_INSERT_FIELD_NATIVE32(_min, _max, _field7, _value7) |   \
638             EFX_INSERT_FIELD_NATIVE32(_min, _max, _field8, _value8) |   \
639             EFX_INSERT_FIELD_NATIVE32(_min, _max, _field9, _value9) |   \
640             EFX_INSERT_FIELD_NATIVE32(_min, _max, _field10, _value10))
641
642 #define EFX_INSERT_FIELDS16(_min, _max,                                 \
643             _field1, _value1, _field2, _value2, _field3, _value3,       \
644             _field4, _value4, _field5, _value5, _field6, _value6,       \
645             _field7, _value7, _field8, _value8, _field9, _value9,       \
646             _field10, _value10)                                         \
647         __CPU_TO_LE_16(                                                 \
648             EFX_INSERT_FIELD_NATIVE16(_min, _max, _field1, _value1) |   \
649             EFX_INSERT_FIELD_NATIVE16(_min, _max, _field2, _value2) |   \
650             EFX_INSERT_FIELD_NATIVE16(_min, _max, _field3, _value3) |   \
651             EFX_INSERT_FIELD_NATIVE16(_min, _max, _field4, _value4) |   \
652             EFX_INSERT_FIELD_NATIVE16(_min, _max, _field5, _value5) |   \
653             EFX_INSERT_FIELD_NATIVE16(_min, _max, _field6, _value6) |   \
654             EFX_INSERT_FIELD_NATIVE16(_min, _max, _field7, _value7) |   \
655             EFX_INSERT_FIELD_NATIVE16(_min, _max, _field8, _value8) |   \
```

```
656             EFX_INSERT_FIELD_NATIVE16(_min, _max, _field9, _value9) |   \
657             EFX_INSERT_FIELD_NATIVE16(_min, _max, _field10, _value10))
658
659 #define EFX_INSERT_FIELDS8(_min, _max,                                  \
660             _field1, _value1, _field2, _value2, _field3, _value3,       \
661             _field4, _value4, _field5, _value5, _field6, _value6,       \
662             _field7, _value7, _field8, _value8, _field9, _value9,       \
663             _field10, _value10)                                         \
664         __NATIVE_8(                                                     \
665             EFX_INSERT_FIELD_NATIVE8(_min, _max, _field1, _value1) |    \
666             EFX_INSERT_FIELD_NATIVE8(_min, _max, _field2, _value2) |    \
667             EFX_INSERT_FIELD_NATIVE8(_min, _max, _field3, _value3) |    \
668             EFX_INSERT_FIELD_NATIVE8(_min, _max, _field4, _value4) |    \
669             EFX_INSERT_FIELD_NATIVE8(_min, _max, _field5, _value5) |    \
670             EFX_INSERT_FIELD_NATIVE8(_min, _max, _field6, _value6) |    \
671             EFX_INSERT_FIELD_NATIVE8(_min, _max, _field7, _value7) |    \
672             EFX_INSERT_FIELD_NATIVE8(_min, _max, _field8, _value8) |    \
673             EFX_INSERT_FIELD_NATIVE8(_min, _max, _field9, _value9) |    \
674             EFX_INSERT_FIELD_NATIVE8(_min, _max, _field10, _value10))
675
676 #define EFX_POPULATE_OWORD64(_oword,                                    \
677             _field1, _value1, _field2, _value2, _field3, _value3,       \
678             _field4, _value4, _field5, _value5, _field6, _value6,       \
679             _field7, _value7, _field8, _value8, _field9, _value9,       \
680             _field10, _value10)                                         \
681         do {                                                            \
682                 _NOTE(CONSTANTCONDITION)                                \
683                 (_oword).eo_u64[0] = EFX_INSERT_FIELDS64(0, 63,         \
684                     _field1, _value1, _field2, _value2,                 \
685                     _field3, _value3, _field4, _value4,                 \
686                     _field5, _value5, _field6, _value6,                 \
687                     _field7, _value7, _field8, _value8,                 \
688                     _field9, _value9, _field10, _value10);              \
689                 _NOTE(CONSTANTCONDITION)                                \
690                 (_oword).eo_u64[1] = EFX_INSERT_FIELDS64(64, 127,       \
691                     _field1, _value1, _field2, _value2,                 \
692                     _field3, _value3, _field4, _value4,                 \
693                     _field5, _value5, _field6, _value6,                 \
694                     _field7, _value7, _field8, _value8,                 \
695                     _field9, _value9, _field10, _value10);              \
696         _NOTE(CONSTANTCONDITION)                                        \
697         } while (B_FALSE)
698
699 #define EFX_POPULATE_OWORD32(_oword,                                    \
700             _field1, _value1, _field2, _value2, _field3, _value3,       \
701             _field4, _value4, _field5, _value5, _field6, _value6,       \
702             _field7, _value7, _field8, _value8, _field9, _value9,       \
703             _field10, _value10)                                         \
704         do {                                                            \
705                 _NOTE(CONSTANTCONDITION)                                \
706                 (_oword).eo_u32[0] = EFX_INSERT_FIELDS32(0, 31,         \
707                     _field1, _value1, _field2, _value2,                 \
708                     _field3, _value3, _field4, _value4,                 \
709                     _field5, _value5, _field6, _value6,                 \
710                     _field7, _value7, _field8, _value8,                 \
711                     _field9, _value9, _field10, _value10);              \
712                 _NOTE(CONSTANTCONDITION)                                \
713                 (_oword).eo_u32[1] = EFX_INSERT_FIELDS32(32, 63,        \
714                     _field1, _value1, _field2, _value2,                 \
715                     _field3, _value3, _field4, _value4,                 \
716                     _field5, _value5, _field6, _value6,                 \
717                     _field7, _value7, _field8, _value8,                 \
718                     _field9, _value9, _field10, _value10);              \
719                 _NOTE(CONSTANTCONDITION)                                \
720                 (_oword).eo_u32[2] = EFX_INSERT_FIELDS32(64, 95,        \
721                     _field1, _value1, _field2, _value2,                 \
```



```
722                     _field3, _value3, _field4, _value4,                 \
723                     _field5, _value5, _field6, _value6,                 \
724                     _field7, _value7, _field8, _value8,                 \
725                     _field9, _value9, _field10, _value10);              \
726                 _NOTE(CONSTANTCONDITION)                                \
727                 (_oword).eo_u32[3] = EFX_INSERT_FIELDS32(96, 127,       \
728                     _field1, _value1, _field2, _value2,                 \
729                     _field3, _value3, _field4, _value4,                 \
730                     _field5, _value5, _field6, _value6,                 \
731                     _field7, _value7, _field8, _value8,                 \
732                     _field9, _value9, _field10, _value10);              \
733         _NOTE(CONSTANTCONDITION)                                        \
734         } while (B_FALSE)
735
736 #define EFX_POPULATE_QWORD64(_qword,                                    \
737             _field1, _value1, _field2, _value2, _field3, _value3,       \
738             _field4, _value4, _field5, _value5, _field6, _value6,       \
739             _field7, _value7, _field8, _value8, _field9, _value9,       \
740             _field10, _value10)                                         \
741         do {                                                            \
742                 _NOTE(CONSTANTCONDITION)                                \
743                 (_qword).eq_u64[0] = EFX_INSERT_FIELDS64(0, 63,         \
744                     _field1, _value1, _field2, _value2,                 \
745                     _field3, _value3, _field4, _value4,                 \
746                     _field5, _value5, _field6, _value6,                 \
747                     _field7, _value7, _field8, _value8,                 \
748                     _field9, _value9, _field10, _value10);              \
749         _NOTE(CONSTANTCONDITION)                                        \
750         } while (B_FALSE)
751
752 #define EFX_POPULATE_QWORD32(_qword,                                    \
753             _field1, _value1, _field2, _value2, _field3, _value3,       \
754             _field4, _value4, _field5, _value5, _field6, _value6,       \
755             _field7, _value7, _field8, _value8, _field9, _value9,       \
756             _field10, _value10)                                         \
757         do {                                                            \
758                 _NOTE(CONSTANTCONDITION)                                \
759                 (_qword).eq_u32[0] = EFX_INSERT_FIELDS32(0, 31,         \
760                     _field1, _value1, _field2, _value2,                 \
761                     _field3, _value3, _field4, _value4,                 \
762                     _field5, _value5, _field6, _value6,                 \
763                     _field7, _value7, _field8, _value8,                 \
764                     _field9, _value9, _field10, _value10);              \
765                 _NOTE(CONSTANTCONDITION)                                \
766                 (_qword).eq_u32[1] = EFX_INSERT_FIELDS32(32, 63,        \
767                     _field1, _value1, _field2, _value2,                 \
768                     _field3, _value3, _field4, _value4,                 \
769                     _field5, _value5, _field6, _value6,                 \
770                     _field7, _value7, _field8, _value8,                 \
771                     _field9, _value9, _field10, _value10);              \
772         _NOTE(CONSTANTCONDITION)                                        \
773         } while (B_FALSE)
774
775 #define EFX_POPULATE_DWORD(_dword,                                      \
776             _field1, _value1, _field2, _value2, _field3, _value3,       \
777             _field4, _value4, _field5, _value5, _field6, _value6,       \
778             _field7, _value7, _field8, _value8, _field9, _value9,       \
779             _field10, _value10)                                         \
780         do {                                                            \
781                 _NOTE(CONSTANTCONDITION)                                \
782                 (_dword).ed_u32[0] = EFX_INSERT_FIELDS32(0, 31,         \
783                     _field1, _value1, _field2, _value2,                 \
784                     _field3, _value3, _field4, _value4,                 \
785                     _field5, _value5, _field6, _value6,                 \
786                     _field7, _value7, _field8, _value8,                 \
787                     _field9, _value9, _field10, _value10);              \
```

```
788         _NOTE(CONSTANTCONDITION)                                        \
789         } while (B_FALSE)

791 #define EFX_POPULATE_WORD(_word,                                        \
792             _field1, _value1, _field2, _value2, _field3, _value3,       \
793             _field4, _value4, _field5, _value5, _field6, _value6,       \
794             _field7, _value7, _field8, _value8, _field9, _value9,       \
795             _field10, _value10)                                         \
796         do {                                                            \
797                 _NOTE(CONSTANTCONDITION)                                \
798                 (_word).ew_u16[0] = EFX_INSERT_FIELDS16(0, 15,          \
799                     _field1, _value1, _field2, _value2,                 \
800                     _field3, _value3, _field4, _value4,                 \
801                     _field5, _value5, _field6, _value6,                 \
802                     _field7, _value7, _field8, _value8,                 \
803                     _field9, _value9, _field10, _value10);              \
804         _NOTE(CONSTANTCONDITION)                                        \
805         } while (B_FALSE)

807 #define EFX_POPULATE_BYTE(_byte,                                        \
808             _field1, _value1, _field2, _value2, _field3, _value3,       \
809             _field4, _value4, _field5, _value5, _field6, _value6,       \
810             _field7, _value7, _field8, _value8, _field9, _value9,       \
811             _field10, _value10)                                         \
812         do {                                                            \
813                 _NOTE(CONSTANTCONDITION)                                \
814                 (_byte).eb_u8[0] = EFX_INSERT_FIELDS8(0, 7,             \
815                     _field1, _value1, _field2, _value2,                 \
816                     _field3, _value3, _field4, _value4,                 \
817                     _field5, _value5, _field6, _value6,                 \
818                     _field7, _value7, _field8, _value8,                 \
819                     _field9, _value9, _field10, _value10);              \
820         _NOTE(CONSTANTCONDITION)                                        \
821         } while (B_FALSE)

823 /* Populate an octword field with various numbers of arguments */
824 #define EFX_POPULATE_OWORD_10 EFX_POPULATE_OWORD

826 #define EFX_POPULATE_OWORD_9(_oword,                                    \
827             _field1, _value1, _field2, _value2, _field3, _value3,       \
828             _field4, _value4, _field5, _value5, _field6, _value6,       \
829             _field7, _value7, _field8, _value8, _field9, _value9)       \
830         EFX_POPULATE_OWORD_10(_oword, EFX_DUMMY_FIELD, 0,               \
831             _field1, _value1, _field2, _value2, _field3, _value3,       \
832             _field4, _value4, _field5, _value5, _field6, _value6,       \
833             _field7, _value7, _field8, _value8, _field9, _value9)

835 #define EFX_POPULATE_OWORD_8(_oword,                                    \
836             _field1, _value1, _field2, _value2, _field3, _value3,       \
837             _field4, _value4, _field5, _value5, _field6, _value6,       \
838             _field7, _value7, _field8, _value8)                         \
839         EFX_POPULATE_OWORD_9(_oword, EFX_DUMMY_FIELD, 0,                \
840             _field1, _value1, _field2, _value2, _field3, _value3,       \
841             _field4, _value4, _field5, _value5, _field6, _value6,       \
842             _field7, _value7, _field8, _value8)

844 #define EFX_POPULATE_OWORD_7(_oword,                                    \
845             _field1, _value1, _field2, _value2, _field3, _value3,       \
846             _field4, _value4, _field5, _value5, _field6, _value6,       \
847             _field7, _value7)                                           \
848         EFX_POPULATE_OWORD_8(_oword, EFX_DUMMY_FIELD, 0,                \
849             _field1, _value1, _field2, _value2, _field3, _value3,       \
850             _field4, _value4, _field5, _value5, _field6, _value6,       \
851             _field7, _value7)

853 #define EFX_POPULATE_OWORD_6(_oword,                                    \
```



```
854             _field1, _value1, _field2, _value2, _field3, _value3,       \
855             _field4, _value4, _field5, _value5, _field6, _value6)       \
856         EFX_POPULATE_OWORD_7(_oword, EFX_DUMMY_FIELD, 0,                \
857             _field1, _value1, _field2, _value2, _field3, _value3,       \
858             _field4, _value4, _field5, _value5, _field6, _value6)

860 #define EFX_POPULATE_OWORD_5(_oword,                                    \
861             _field1, _value1, _field2, _value2, _field3, _value3,       \
862             _field4, _value4, _field5, _value5)                         \
863         EFX_POPULATE_OWORD_6(_oword, EFX_DUMMY_FIELD, 0,                \
864             _field1, _value1, _field2, _value2, _field3, _value3,       \
865             _field4, _value4, _field5, _value5)

867 #define EFX_POPULATE_OWORD_4(_oword,                                    \
868             _field1, _value1, _field2, _value2, _field3, _value3,       \
869             _field4, _value4)                                           \
870         EFX_POPULATE_OWORD_5(_oword, EFX_DUMMY_FIELD, 0,                \
871             _field1, _value1, _field2, _value2, _field3, _value3,       \
872             _field4, _value4)

874 #define EFX_POPULATE_OWORD_3(_oword,                                    \
875             _field1, _value1, _field2, _value2, _field3, _value3)       \
876         EFX_POPULATE_OWORD_4(_oword, EFX_DUMMY_FIELD, 0,                \
877             _field1, _value1, _field2, _value2, _field3, _value3)

879 #define EFX_POPULATE_OWORD_2(_oword,                                    \
880             _field1, _value1, _field2, _value2)                         \
881         EFX_POPULATE_OWORD_3(_oword, EFX_DUMMY_FIELD, 0,                \
882             _field1, _value1, _field2, _value2)

884 #define EFX_POPULATE_OWORD_1(_oword,                                    \
885             _field1, _value1)                                           \
886         EFX_POPULATE_OWORD_2(_oword, EFX_DUMMY_FIELD, 0,                \
887             _field1, _value1)

889 #define EFX_ZERO_OWORD(_oword)                                          \
890         EFX_POPULATE_OWORD_1(_oword, EFX_DUMMY_FIELD, 0)

892 #define EFX_SET_OWORD64(_oword)                                         \
893         EFX_POPULATE_OWORD_2(_oword,                                    \
894             EFX_QWORD_0, (uint64_t)-1, EFX_QWORD_1, (uint64_t)-1)

896 #define EFX_SET_OWORD32(_oword)                                         \
897         EFX_POPULATE_OWORD_4(_oword,                                    \
898             EFX_DWORD_0, 0xffffffff, EFX_DWORD_1, 0xffffffff,           \
899             EFX_DWORD_2, 0xffffffff, EFX_DWORD_3, 0xffffffff)

901 /* Populate a quadword field with various numbers of arguments */
902 #define EFX_POPULATE_QWORD_10 EFX_POPULATE_QWORD

904 #define EFX_POPULATE_QWORD_9(_qword,                                    \
905             _field1, _value1, _field2, _value2, _field3, _value3,       \
906             _field4, _value4, _field5, _value5, _field6, _value6,       \
907             _field7, _value7, _field8, _value8, _field9, _value9)       \
908         EFX_POPULATE_QWORD_10(_qword, EFX_DUMMY_FIELD, 0,               \
909             _field1, _value1, _field2, _value2, _field3, _value3,       \
910             _field4, _value4, _field5, _value5, _field6, _value6,       \
911             _field7, _value7, _field8, _value8, _field9, _value9)

913 #define EFX_POPULATE_QWORD_8(_qword,                                    \
914             _field1, _value1, _field2, _value2, _field3, _value3,       \
915             _field4, _value4, _field5, _value5, _field6, _value6,       \
916             _field7, _value7, _field8, _value8)                         \
917         EFX_POPULATE_QWORD_9(_qword, EFX_DUMMY_FIELD, 0,                \
918             _field1, _value1, _field2, _value2, _field3, _value3,       \
919             _field4, _value4, _field5, _value5, _field6, _value6,       \
```

```
920             _field7, _value7, _field8, _value8)

922 #define EFX_POPULATE_QWORD_7(_qword,                                    \
923             _field1, _value1, _field2, _value2, _field3, _value3,       \
924             _field4, _value4, _field5, _value5, _field6, _value6,       \
925             _field7, _value7)                                           \
926         EFX_POPULATE_QWORD_8(_qword, EFX_DUMMY_FIELD, 0,                \
927             _field1, _value1, _field2, _value2, _field3, _value3,       \
928             _field4, _value4, _field5, _value5, _field6, _value6,       \
929             _field7, _value7)

931 #define EFX_POPULATE_QWORD_6(_qword,                                    \
932             _field1, _value1, _field2, _value2, _field3, _value3,       \
933             _field4, _value4, _field5, _value5, _field6, _value6)       \
934         EFX_POPULATE_QWORD_7(_qword, EFX_DUMMY_FIELD, 0,                \
935             _field1, _value1, _field2, _value2, _field3, _value3,       \
936             _field4, _value4, _field5, _value5, _field6, _value6)

938 #define EFX_POPULATE_QWORD_5(_qword,                                    \
939             _field1, _value1, _field2, _value2, _field3, _value3,       \
940             _field4, _value4, _field5, _value5)                         \
941         EFX_POPULATE_QWORD_6(_qword, EFX_DUMMY_FIELD, 0,                \
942             _field1, _value1, _field2, _value2, _field3, _value3,       \
943             _field4, _value4, _field5, _value5)

945 #define EFX_POPULATE_QWORD_4(_qword,                                    \
946             _field1, _value1, _field2, _value2, _field3, _value3,       \
947             _field4, _value4)                                           \
948         EFX_POPULATE_QWORD_5(_qword, EFX_DUMMY_FIELD, 0,                \
949             _field1, _value1, _field2, _value2, _field3, _value3,       \
950             _field4, _value4)

952 #define EFX_POPULATE_QWORD_3(_qword,                                    \
953             _field1, _value1, _field2, _value2, _field3, _value3)       \
954         EFX_POPULATE_QWORD_4(_qword, EFX_DUMMY_FIELD, 0,                \
955             _field1, _value1, _field2, _value2, _field3, _value3)

957 #define EFX_POPULATE_QWORD_2(_qword,                                    \
958             _field1, _value1, _field2, _value2)                         \
959         EFX_POPULATE_QWORD_3(_qword, EFX_DUMMY_FIELD, 0,                \
960             _field1, _value1, _field2, _value2)

962 #define EFX_POPULATE_QWORD_1(_qword,                                    \
963             _field1, _value1)                                           \
964         EFX_POPULATE_QWORD_2(_qword, EFX_DUMMY_FIELD, 0,                \
965             _field1, _value1)

967 #define EFX_ZERO_QWORD(_qword)                                          \
968         EFX_POPULATE_QWORD_1(_qword, EFX_DUMMY_FIELD, 0)

970 #define EFX_SET_QWORD64(_qword)                                         \
971         EFX_POPULATE_QWORD_1(_qword,                                    \
972             EFX_QWORD_0, (uint64_t)-1)

974 #define EFX_SET_QWORD32(_qword)                                         \
975         EFX_POPULATE_QWORD_2(_qword,                                    \
976             EFX_DWORD_0, 0xffffffff, EFX_DWORD_1, 0xffffffff)

978 /* Populate a dword field with various numbers of arguments */
979 #define EFX_POPULATE_DWORD_10 EFX_POPULATE_DWORD

981 #define EFX_POPULATE_DWORD_9(_dword,                                    \
982             _field1, _value1, _field2, _value2, _field3, _value3,       \
983             _field4, _value4, _field5, _value5, _field6, _value6,       \
984             _field7, _value7, _field8, _value8, _field9, _value9)       \
985         EFX_POPULATE_DWORD_10(_dword, EFX_DUMMY_FIELD, 0,               \
```



```
986             _field1, _value1, _field2, _value2, _field3, _value3,       \
987             _field4, _value4, _field5, _value5, _field6, _value6,       \
988             _field7, _value7, _field8, _value8, _field9, _value9)

990 #define EFX_POPULATE_DWORD_8(_dword,                                    \
991             _field1, _value1, _field2, _value2, _field3, _value3,       \
992             _field4, _value4, _field5, _value5, _field6, _value6,       \
993             _field7, _value7, _field8, _value8)                         \
994         EFX_POPULATE_DWORD_9(_dword, EFX_DUMMY_FIELD, 0,                \
995             _field1, _value1, _field2, _value2, _field3, _value3,       \
996             _field4, _value4, _field5, _value5, _field6, _value6,       \
997             _field7, _value7, _field8, _value8)

999 #define EFX_POPULATE_DWORD_7(_dword,                                    \
1000             _field1, _value1, _field2, _value2, _field3, _value3,      \
1001             _field4, _value4, _field5, _value5, _field6, _value6,      \
1002             _field7, _value7)                                          \
1003         EFX_POPULATE_DWORD_8(_dword, EFX_DUMMY_FIELD, 0,               \
1004             _field1, _value1, _field2, _value2, _field3, _value3,      \
1005             _field4, _value4, _field5, _value5, _field6, _value6,      \
1006             _field7, _value7)

1008 #define EFX_POPULATE_DWORD_6(_dword,                                   \
1009             _field1, _value1, _field2, _value2, _field3, _value3,      \
1010             _field4, _value4, _field5, _value5, _field6, _value6)      \
1011         EFX_POPULATE_DWORD_7(_dword, EFX_DUMMY_FIELD, 0,               \
1012             _field1, _value1, _field2, _value2, _field3, _value3,      \
1013             _field4, _value4, _field5, _value5, _field6, _value6)

1015 #define EFX_POPULATE_DWORD_5(_dword,                                   \
1016             _field1, _value1, _field2, _value2, _field3, _value3,      \
1017             _field4, _value4, _field5, _value5)                        \
1018         EFX_POPULATE_DWORD_6(_dword, EFX_DUMMY_FIELD, 0,               \
1019             _field1, _value1, _field2, _value2, _field3, _value3,      \
1020             _field4, _value4, _field5, _value5)

1022 #define EFX_POPULATE_DWORD_4(_dword,                                   \
1023             _field1, _value1, _field2, _value2, _field3, _value3,      \
1024             _field4, _value4)                                          \
1025         EFX_POPULATE_DWORD_5(_dword, EFX_DUMMY_FIELD, 0,               \
1026             _field1, _value1, _field2, _value2, _field3, _value3,      \
1027             _field4, _value4)

1029 #define EFX_POPULATE_DWORD_3(_dword,                                   \
1030             _field1, _value1, _field2, _value2, _field3, _value3)      \
1031         EFX_POPULATE_DWORD_4(_dword, EFX_DUMMY_FIELD, 0,               \
1032             _field1, _value1, _field2, _value2, _field3, _value3)

1034 #define EFX_POPULATE_DWORD_2(_dword,                                   \
1035             _field1, _value1, _field2, _value2)                        \
1036         EFX_POPULATE_DWORD_3(_dword, EFX_DUMMY_FIELD, 0,               \
1037             _field1, _value1, _field2, _value2)

1039 #define EFX_POPULATE_DWORD_1(_dword,                                   \
1040             _field1, _value1)                                          \
1041         EFX_POPULATE_DWORD_2(_dword, EFX_DUMMY_FIELD, 0,               \
1042             _field1, _value1)

1044 #define EFX_ZERO_DWORD(_dword)                                         \
1045         EFX_POPULATE_DWORD_1(_dword, EFX_DUMMY_FIELD, 0)

1047 #define EFX_SET_DWORD(_dword)                                          \
1048         EFX_POPULATE_DWORD_1(_dword,                                   \
1049             EFX_DWORD_0, 0xffffffff)

1051 /* Populate a word field with various numbers of arguments */
```

```
#define EFX_POPULATE_WORD_10 EFX_POPULATE_WORD

#define EFX_POPULATE_WORD_9(_word,                                      \
            _field1, _value1, _field2, _value2, _field3, _value3,       \
            _field4, _value4, _field5, _value5, _field6, _value6,       \
            _field7, _value7, _field8, _value8, _field9, _value9)       \
        EFX_POPULATE_WORD_10(_word, EFX_DUMMY_FIELD, 0,                 \
            _field1, _value1, _field2, _value2, _field3, _value3,       \
            _field4, _value4, _field5, _value5, _field6, _value6,       \
            _field7, _value7, _field8, _value8, _field9, _value9)

#define EFX_POPULATE_WORD_8(_word,                                      \
            _field1, _value1, _field2, _value2, _field3, _value3,       \
            _field4, _value4, _field5, _value5, _field6, _value6,       \
            _field7, _value7, _field8, _value8)                         \
        EFX_POPULATE_WORD_9(_word, EFX_DUMMY_FIELD, 0,                  \
            _field1, _value1, _field2, _value2, _field3, _value3,       \
            _field4, _value4, _field5, _value5, _field6, _value6,       \
            _field7, _value7, _field8, _value8)

#define EFX_POPULATE_WORD_7(_word,                                      \
            _field1, _value1, _field2, _value2, _field3, _value3,       \
            _field4, _value4, _field5, _value5, _field6, _value6,       \
            _field7, _value7)                                           \
        EFX_POPULATE_WORD_8(_word, EFX_DUMMY_FIELD, 0,                  \
            _field1, _value1, _field2, _value2, _field3, _value3,       \
            _field4, _value4, _field5, _value5, _field6, _value6,       \
            _field7, _value7)

#define EFX_POPULATE_WORD_6(_word,                                      \
            _field1, _value1, _field2, _value2, _field3, _value3,       \
            _field4, _value4, _field5, _value5, _field6, _value6)       \
        EFX_POPULATE_WORD_7(_word, EFX_DUMMY_FIELD, 0,                  \
            _field1, _value1, _field2, _value2, _field3, _value3,       \
            _field4, _value4, _field5, _value5, _field6, _value6)

#define EFX_POPULATE_WORD_5(_word,                                      \
            _field1, _value1, _field2, _value2, _field3, _value3,       \
            _field4, _value4, _field5, _value5)                         \
        EFX_POPULATE_WORD_6(_word, EFX_DUMMY_FIELD, 0,                  \
            _field1, _value1, _field2, _value2, _field3, _value3,       \
            _field4, _value4, _field5, _value5)

#define EFX_POPULATE_WORD_4(_word,                                      \
            _field1, _value1, _field2, _value2, _field3, _value3,       \
            _field4, _value4)                                           \
        EFX_POPULATE_WORD_5(_word, EFX_DUMMY_FIELD, 0,                  \
            _field1, _value1, _field2, _value2, _field3, _value3,       \
            _field4, _value4)

#define EFX_POPULATE_WORD_3(_word,                                      \
            _field1, _value1, _field2, _value2, _field3, _value3)       \
        EFX_POPULATE_WORD_4(_word, EFX_DUMMY_FIELD, 0,                  \
            _field1, _value1, _field2, _value2, _field3, _value3)

#define EFX_POPULATE_WORD_2(_word,                                      \
            _field1, _value1, _field2, _value2)                         \
        EFX_POPULATE_WORD_3(_word, EFX_DUMMY_FIELD, 0,                  \
            _field1, _value1, _field2, _value2)

#define EFX_POPULATE_WORD_1(_word,                                      \
            _field1, _value1)                                           \
        EFX_POPULATE_WORD_2(_word, EFX_DUMMY_FIELD, 0,                  \
            _field1, _value1)

#define EFX_ZERO_WORD(_word)                                            \
```



```
        EFX_POPULATE_WORD_1(_word, EFX_DUMMY_FIELD, 0)

#define EFX_SET_WORD(_word)                                             \
        EFX_POPULATE_WORD_1(_word,                                      \
            EFX_WORD_0, 0xffff)

/* Populate a byte field with various numbers of arguments */
#define EFX_POPULATE_BYTE_10 EFX_POPULATE_BYTE

#define EFX_POPULATE_BYTE_9(_byte,                                      \
            _field1, _value1, _field2, _value2, _field3, _value3,       \
            _field4, _value4, _field5, _value5, _field6, _value6,       \
            _field7, _value7, _field8, _value8, _field9, _value9)       \
        EFX_POPULATE_BYTE_10(_byte, EFX_DUMMY_FIELD, 0,                 \
            _field1, _value1, _field2, _value2, _field3, _value3,       \
            _field4, _value4, _field5, _value5, _field6, _value6,       \
            _field7, _value7, _field8, _value8, _field9, _value9)

#define EFX_POPULATE_BYTE_8(_byte,                                      \
            _field1, _value1, _field2, _value2, _field3, _value3,       \
            _field4, _value4, _field5, _value5, _field6, _value6,       \
            _field7, _value7, _field8, _value8)                         \
        EFX_POPULATE_BYTE_9(_byte, EFX_DUMMY_FIELD, 0,                  \
            _field1, _value1, _field2, _value2, _field3, _value3,       \
            _field4, _value4, _field5, _value5, _field6, _value6,       \
            _field7, _value7, _field8, _value8)

#define EFX_POPULATE_BYTE_7(_byte,                                      \
            _field1, _value1, _field2, _value2, _field3, _value3,       \
            _field4, _value4, _field5, _value5, _field6, _value6,       \
            _field7, _value7)                                           \
        EFX_POPULATE_BYTE_8(_byte, EFX_DUMMY_FIELD, 0,                  \
            _field1, _value1, _field2, _value2, _field3, _value3,       \
            _field4, _value4, _field5, _value5, _field6, _value6,       \
            _field7, _value7)

#define EFX_POPULATE_BYTE_6(_byte,                                      \
            _field1, _value1, _field2, _value2, _field3, _value3,       \
            _field4, _value4, _field5, _value5, _field6, _value6)       \
        EFX_POPULATE_BYTE_7(_byte, EFX_DUMMY_FIELD, 0,                  \
            _field1, _value1, _field2, _value2, _field3, _value3,       \
            _field4, _value4, _field5, _value5, _field6, _value6)

#define EFX_POPULATE_BYTE_5(_byte,                                      \
            _field1, _value1, _field2, _value2, _field3, _value3,       \
            _field4, _value4, _field5, _value5)                         \
        EFX_POPULATE_BYTE_6(_byte, EFX_DUMMY_FIELD, 0,                  \
            _field1, _value1, _field2, _value2, _field3, _value3,       \
            _field4, _value4, _field5, _value5)

#define EFX_POPULATE_BYTE_4(_byte,                                      \
            _field1, _value1, _field2, _value2, _field3, _value3,       \
            _field4, _value4)                                           \
        EFX_POPULATE_BYTE_5(_byte, EFX_DUMMY_FIELD, 0,                  \
            _field1, _value1, _field2, _value2, _field3, _value3,       \
            _field4, _value4)

#define EFX_POPULATE_BYTE_3(_byte,                                      \
            _field1, _value1, _field2, _value2, _field3, _value3)       \
        EFX_POPULATE_BYTE_4(_byte, EFX_DUMMY_FIELD, 0,                  \
            _field1, _value1, _field2, _value2, _field3, _value3)

#define EFX_POPULATE_BYTE_2(_byte,                                      \
            _field1, _value1, _field2, _value2)                         \
        EFX_POPULATE_BYTE_3(_byte, EFX_DUMMY_FIELD, 0,                  \
            _field1, _value1, _field2, _value2)
```

```
#define EFX_POPULATE_BYTE_1(_byte,                                      \
            _field1, _value1)                                           \
        EFX_POPULATE_BYTE_2(_byte, EFX_DUMMY_FIELD, 0,                  \
            _field1, _value1)

#define EFX_ZERO_BYTE(_byte)                                            \
        EFX_POPULATE_BYTE_1(_byte, EFX_DUMMY_FIELD, 0)

#define EFX_SET_BYTE(_byte)                                             \
        EFX_POPULATE_BYTE_1(_byte,                                      \
            EFX_BYTE_0, 0xff)

/*
 * Modify a named field within an already-populated structure.  Used
 * for read-modify-write operations.
 */

#define EFX_INSERT_FIELD64(_min, _max, _field, _value)                  \
        __CPU_TO_LE_64(EFX_INSERT_FIELD_NATIVE64(_min, _max, _field, _value))

#define EFX_INSERT_FIELD32(_min, _max, _field, _value)                  \
        __CPU_TO_LE_32(EFX_INSERT_FIELD_NATIVE32(_min, _max, _field, _value))

#define EFX_INSERT_FIELD16(_min, _max, _field, _value)                  \
        __CPU_TO_LE_16(EFX_INSERT_FIELD_NATIVE16(_min, _max, _field, _value))

#define EFX_INSERT_FIELD8(_min, _max, _field, _value)                   \
        __NATIVE_8(EFX_INSERT_FIELD_NATIVE8(_min, _max, _field, _value))

#define EFX_INPLACE_MASK64(_min, _max, _field)                          \
        EFX_INSERT_FIELD64(_min, _max, _field, EFX_MASK64(_field))

#define EFX_INPLACE_MASK32(_min, _max, _field)                          \
        EFX_INSERT_FIELD32(_min, _max, _field, EFX_MASK32(_field))

#define EFX_INPLACE_MASK16(_min, _max, _field)                          \
        EFX_INSERT_FIELD16(_min, _max, _field, EFX_MASK16(_field))

#define EFX_INPLACE_MASK8(_min, _max, _field)                           \
        EFX_INSERT_FIELD8(_min, _max, _field, EFX_MASK8(_field))

#define EFX_SET_OWORD_FIELD64(_oword, _field, _value)                   \
        do {                                                            \
                _NOTE(CONSTANTCONDITION)                                \
                (_oword).eo_u64[0] = (((_oword).eo_u64[0] &             \
                    ~EFX_INPLACE_MASK64(0, 63, _field)) |               \
                    EFX_INSERT_FIELD64(0, 63, _field, _value));         \
                _NOTE(CONSTANTCONDITION)                                \
                (_oword).eo_u64[1] = (((_oword).eo_u64[1] &             \
                    ~EFX_INPLACE_MASK64(64, 127, _field)) |             \
                    EFX_INSERT_FIELD64(64, 127, _field, _value));       \
        _NOTE(CONSTANTCONDITION)                                        \
        } while (B_FALSE)

#define EFX_SET_OWORD_FIELD32(_oword, _field, _value)                   \
        do {                                                            \
                _NOTE(CONSTANTCONDITION)                                \
                (_oword).eo_u32[0] = (((_oword).eo_u32[0] &             \
                    ~EFX_INPLACE_MASK32(0, 31, _field)) |               \
                    EFX_INSERT_FIELD32(0, 31, _field, _value));         \
                _NOTE(CONSTANTCONDITION)                                \
                (_oword).eo_u32[1] = (((_oword).eo_u32[1] &             \
                    ~EFX_INPLACE_MASK32(32, 63, _field)) |              \
                    EFX_INSERT_FIELD32(32, 63, _field, _value));        \
                _NOTE(CONSTANTCONDITION)                                \
```



```
                (_oword).eo_u32[2] = (((_oword).eo_u32[2] &             \
                    ~EFX_INPLACE_MASK32(64, 95, _field)) |              \
                    EFX_INSERT_FIELD32(64, 95, _field, _value));        \
                _NOTE(CONSTANTCONDITION)                                \
                (_oword).eo_u32[3] = (((_oword).eo_u32[3] &             \
                    ~EFX_INPLACE_MASK32(96, 127, _field)) |             \
                    EFX_INSERT_FIELD32(96, 127, _field, _value));       \
        _NOTE(CONSTANTCONDITION)                                        \
        } while (B_FALSE)

#define EFX_SET_QWORD_FIELD64(_qword, _field, _value)                   \
        do {                                                            \
                _NOTE(CONSTANTCONDITION)                                \
                (_qword).eq_u64[0] = (((_qword).eq_u64[0] &             \
                    ~EFX_INPLACE_MASK64(0, 63, _field)) |               \
                    EFX_INSERT_FIELD64(0, 63, _field, _value));         \
        _NOTE(CONSTANTCONDITION)                                        \
        } while (B_FALSE)

#define EFX_SET_QWORD_FIELD32(_qword, _field, _value)                   \
        do {                                                            \
                _NOTE(CONSTANTCONDITION)                                \
                (_qword).eq_u32[0] = (((_qword).eq_u32[0] &             \
                    ~EFX_INPLACE_MASK32(0, 31, _field)) |               \
                    EFX_INSERT_FIELD32(0, 31, _field, _value));         \
                _NOTE(CONSTANTCONDITION)                                \
                (_qword).eq_u32[1] = (((_qword).eq_u32[1] &             \
                    ~EFX_INPLACE_MASK32(32, 63, _field)) |              \
                    EFX_INSERT_FIELD32(32, 63, _field, _value));        \
        _NOTE(CONSTANTCONDITION)                                        \
        } while (B_FALSE)

#define EFX_SET_DWORD_FIELD(_dword, _field, _value)                     \
        do {                                                            \
                _NOTE(CONSTANTCONDITION)                                \
                (_dword).ed_u32[0] = (((_dword).ed_u32[0] &             \
                    ~EFX_INPLACE_MASK32(0, 31, _field)) |               \
                    EFX_INSERT_FIELD32(0, 31, _field, _value));         \
        _NOTE(CONSTANTCONDITION)                                        \
        } while (B_FALSE)

#define EFX_SET_WORD_FIELD(_word, _field, _value)                       \
        do {                                                            \
                _NOTE(CONSTANTCONDITION)                                \
                (_word).ew_u16[0] = (((_word).ew_u16[0] &               \
                    ~EFX_INPLACE_MASK16(0, 15, _field)) |               \
                    EFX_INSERT_FIELD16(0, 15, _field, _value));         \
        _NOTE(CONSTANTCONDITION)                                        \
        } while (B_FALSE)

#define EFX_SET_BYTE_FIELD(_byte, _field, _value)                       \
        do {                                                            \
                _NOTE(CONSTANTCONDITION)                                \
                (_byte).eb_u8[0] = (((_byte).eb_u8[0] &                 \
                    ~EFX_INPLACE_MASK8(0, 7, _field)) |                 \
                    EFX_INSERT_FIELD8(0, 7, _field, _value));           \
        _NOTE(CONSTANTCONDITION)                                        \
        } while (B_FALSE)

/*
 * Set or clear a numbered bit within an octword.
 */

#define EFX_SHIFT64(_bit, _base)                                        \
        (((_bit) >= (_base) && (_bit) < (_base) + 64) ?                 \
                ((uint64_t)1 << ((_bit) - (_base))) :                   \
```

```
1316                0U)
1317
1318 #define EFX_SHIFT32(_bit, _base)                                        \
1319         (((_bit) >= (_base) && (_bit) < (_base) + 32) ?                 \
1320                 ((uint32_t)1 << ((_bit) - (_base))) :                   \
1321                 0U)
1322
1323 #define EFX_SHIFT16(_bit, _base)                                        \
1324         (((_bit) >= (_base) && (_bit) < (_base) + 16) ?                 \
1325                 (uint16_t)(1 << ((_bit) - (_base))) :                   \
1326                 0U)
1327
1328 #define EFX_SHIFT8(_bit, _base)                                         \
1329         (((_bit) >= (_base) && (_bit) < (_base) + 8) ?                  \
1330                 (uint8_t)(1 << ((_bit) - (_base))) :                    \
1331                 0U)
1332
1333 #define EFX_SET_OWORD_BIT64(_oword, _bit)                               \
1334         do {                                                            \
1335                 _NOTE(CONSTANTCONDITION)                                \
1336                 (_oword).eo_u64[0] |=                                   \
1337                     __CPU_TO_LE_64(EFX_SHIFT64(_bit, FIX_LINT(0)));     \
1338                 (_oword).eo_u64[1] |=                                   \
1339                     __CPU_TO_LE_64(EFX_SHIFT64(_bit, FIX_LINT(64)));    \
1340         _NOTE(CONSTANTCONDITION)                                        \
1341         } while (B_FALSE)
1342
1343 #define EFX_SET_OWORD_BIT32(_oword, _bit)                               \
1344         do {                                                            \
1345                 _NOTE(CONSTANTCONDITION)                                \
1346                 (_oword).eo_u32[0] |=                                   \
1347                     __CPU_TO_LE_32(EFX_SHIFT32(_bit, FIX_LINT(0)));     \
1348                 (_oword).eo_u32[1] |=                                   \
1349                     __CPU_TO_LE_32(EFX_SHIFT32(_bit, FIX_LINT(32)));    \
1350                 (_oword).eo_u32[2] |=                                   \
1351                     __CPU_TO_LE_32(EFX_SHIFT32(_bit, FIX_LINT(64)));    \
1352                 (_oword).eo_u32[3] |=                                   \
1353                     __CPU_TO_LE_32(EFX_SHIFT32(_bit, FIX_LINT(96)));    \
1354         _NOTE(CONSTANTCONDITION)                                        \
1355         } while (B_FALSE)
1356
1357 #define EFX_CLEAR_OWORD_BIT64(_oword, _bit)                             \
1358         do {                                                            \
1359                 _NOTE(CONSTANTCONDITION)                                \
1360                 (_oword).eo_u64[0] &=                                   \
1361                     __CPU_TO_LE_64(~EFX_SHIFT64(_bit, FIX_LINT(0)));    \
1362                 (_oword).eo_u64[1] &=                                   \
1363                     __CPU_TO_LE_64(~EFX_SHIFT64(_bit, FIX_LINT(64)));   \
1364         _NOTE(CONSTANTCONDITION)                                        \
1365         } while (B_FALSE)
1366
1367 #define EFX_CLEAR_OWORD_BIT32(_oword, _bit)                             \
1368         do {                                                            \
1369                 _NOTE(CONSTANTCONDITION)                                \
1370                 (_oword).eo_u32[0] &=                                   \
1371                     __CPU_TO_LE_32(~EFX_SHIFT32(_bit, FIX_LINT(0)));    \
1372                 (_oword).eo_u32[1] &=                                   \
1373                     __CPU_TO_LE_32(~EFX_SHIFT32(_bit, FIX_LINT(32)));   \
1374                 (_oword).eo_u32[2] &=                                   \
1375                     __CPU_TO_LE_32(~EFX_SHIFT32(_bit, FIX_LINT(64)));   \
1376                 (_oword).eo_u32[3] &=                                   \
1377                     __CPU_TO_LE_32(~EFX_SHIFT32(_bit, FIX_LINT(96)));   \
1378         _NOTE(CONSTANTCONDITION)                                        \
1379         } while (B_FALSE)
1380
1381 #define EFX_SET_QWORD_BIT64(_qword, _bit)                               \
```



```
1382         do {                                                           \
1383                 _NOTE(CONSTANTCONDITION)                               \
1384                 (_qword).eq_u64[0] |=                                  \
1385                     __CPU_TO_LE_64(EFX_SHIFT64(_bit, FIX_LINT(0)));    \
1386         _NOTE(CONSTANTCONDITION)                                       \
1387         } while (B_FALSE)
1388
1389 #define EFX_SET_QWORD_BIT32(_qword, _bit)                              \
1390         do {                                                           \
1391                 _NOTE(CONSTANTCONDITION)                               \
1392                 (_qword).eq_u32[0] |=                                  \
1393                     __CPU_TO_LE_32(EFX_SHIFT32(_bit, FIX_LINT(0)));    \
1394                 (_qword).eq_u32[1] |=                                  \
1395                     __CPU_TO_LE_32(EFX_SHIFT32(_bit, FIX_LINT(32)));   \
1396         _NOTE(CONSTANTCONDITION)                                       \
1397         } while (B_FALSE)
1398
1399 #define EFX_CLEAR_QWORD_BIT64(_qword, _bit)                            \
1400         do {                                                           \
1401                 _NOTE(CONSTANTCONDITION)                               \
1402                 (_qword).eq_u64[0] &=                                  \
1403                     __CPU_TO_LE_64(~EFX_SHIFT64(_bit, FIX_LINT(0)));   \
1404         _NOTE(CONSTANTCONDITION)                                       \
1405         } while (B_FALSE)
1406
1407 #define EFX_CLEAR_QWORD_BIT32(_qword, _bit)                            \
1408         do {                                                           \
1409                 _NOTE(CONSTANTCONDITION)                               \
1410                 (_qword).eq_u32[0] &=                                  \
1411                     __CPU_TO_LE_32(~EFX_SHIFT32(_bit, FIX_LINT(0)));   \
1412                 (_qword).eq_u32[1] &=                                  \
1413                     __CPU_TO_LE_32(~EFX_SHIFT32(_bit, FIX_LINT(32)));  \
1414         _NOTE(CONSTANTCONDITION)                                       \
1415         } while (B_FALSE)
1416
1417 #define EFX_SET_DWORD_BIT(_dword, _bit)                                \
1418         do {                                                           \
1419                 (_dword).ed_u32[0] |=                                  \
1420                     __CPU_TO_LE_32(EFX_SHIFT32(_bit, FIX_LINT(0)));    \
1421         _NOTE(CONSTANTCONDITION)                                       \
1422         } while (B_FALSE)
1423
1424 #define EFX_CLEAR_DWORD_BIT(_dword, _bit)                              \
1425         do {                                                           \
1426                 (_dword).ed_u32[0] &=                                  \
1427                     __CPU_TO_LE_32(~EFX_SHIFT32(_bit, FIX_LINT(0)));   \
1428         _NOTE(CONSTANTCONDITION)                                       \
1429         } while (B_FALSE)
1430
1431 #define EFX_SET_WORD_BIT(_word, _bit)                                  \
1432         do {                                                           \
1433                 (_word).ew_u16[0] |=                                   \
1434                     __CPU_TO_LE_16(EFX_SHIFT16(_bit, FIX_LINT(0)));    \
1435         _NOTE(CONSTANTCONDITION)                                       \
1436         } while (B_FALSE)
1437
1438 #define EFX_CLEAR_WORD_BIT(_word, _bit)                                \
1439         do {                                                           \
1440                 (_word).ew_u32[0] &=                                   \
1441                     __CPU_TO_LE_16(~EFX_SHIFT16(_bit, FIX_LINT(0)));   \
1442         _NOTE(CONSTANTCONDITION)                                       \
1443         } while (B_FALSE)
1444
1445 #define EFX_SET_BYTE_BIT(_byte, _bit)                                  \
1446         do {                                                           \
1447                 (_byte).eb_u8[0] |=                                    \
```

```
1448                    __NATIVE_8(EFX_SHIFT8(_bit, FIX_LINT(0)));          \
1449            _NOTE(CONSTANTCONDITION)                                    \
1450            } while (B_FALSE)

1452 #define EFX_CLEAR_BYTE_BIT(_byte, _bit)                                 \
1453            do {                                                        \
1454                    (_byte).eb_u8[0] &=                                 \
1455                            __NATIVE_8(~EFX_SHIFT8(_bit, FIX_LINT(0)));         \
1456            _NOTE(CONSTANTCONDITION)                                    \
1457            } while (B_FALSE)

1459 #define EFX_OR_OWORD64(_oword1, _oword2)                                \
1460            do {                                                        \
1461                    (_oword1).eo_u64[0] |= (_oword2).eo_u64[0];         \
1462                    (_oword1).eo_u64[1] |= (_oword2).eo_u64[1];         \
1463            _NOTE(CONSTANTCONDITION)                                    \
1464            } while (B_FALSE)

1466 #define EFX_OR_OWORD32(_oword1, _oword2)                                \
1467            do {                                                        \
1468                    (_oword1).eo_u32[0] |= (_oword2).eo_u32[0];         \
1469                    (_oword1).eo_u32[1] |= (_oword2).eo_u32[1];         \
1470                    (_oword1).eo_u32[2] |= (_oword2).eo_u32[2];         \
1471                    (_oword1).eo_u32[3] |= (_oword2).eo_u32[3];         \
1472            _NOTE(CONSTANTCONDITION)                                    \
1473            } while (B_FALSE)

1475 #define EFX_AND_OWORD64(_oword1, _oword2)                               \
1476            do {                                                        \
1477                    (_oword1).eo_u64[0] &= (_oword2).eo_u64[0];         \
1478                    (_oword1).eo_u64[1] &= (_oword2).eo_u64[1];         \
1479            _NOTE(CONSTANTCONDITION)                                    \
1480            } while (B_FALSE)

1482 #define EFX_AND_OWORD32(_oword1, _oword2)                               \
1483            do {                                                        \
1484                    (_oword1).eo_u32[0] &= (_oword2).eo_u32[0];         \
1485                    (_oword1).eo_u32[1] &= (_oword2).eo_u32[1];         \
1486                    (_oword1).eo_u32[2] &= (_oword2).eo_u32[2];         \
1487                    (_oword1).eo_u32[3] &= (_oword2).eo_u32[3];         \
1488            _NOTE(CONSTANTCONDITION)                                    \
1489            } while (B_FALSE)

1491 #define EFX_OR_QWORD64(_qword1, _qword2)                                \
1492            do {                                                        \
1493                    (_qword1).eq_u64[0] |= (_qword2).eq_u64[0];         \
1494            _NOTE(CONSTANTCONDITION)                                    \
1495            } while (B_FALSE)

1497 #define EFX_OR_QWORD32(_qword1, _qword2)                                \
1498            do {                                                        \
1499                    (_qword1).eq_u32[0] |= (_qword2).eq_u32[0];         \
1500                    (_qword1).eq_u32[1] |= (_qword2).eq_u32[1];         \
1501            _NOTE(CONSTANTCONDITION)                                    \
1502            } while (B_FALSE)

1504 #define EFX_AND_QWORD64(_qword1, _qword2)                               \
1505            do {                                                        \
1506                    (_qword1).eq_u64[0] &= (_qword2).eq_u64[0];         \
1507            _NOTE(CONSTANTCONDITION)                                    \
1508            } while (B_FALSE)

1510 #define EFX_AND_QWORD32(_qword1, _qword2)                               \
1511            do {                                                        \
1512                    (_qword1).eq_u32[0] &= (_qword2).eq_u32[0];         \
1513                    (_qword1).eq_u32[1] &= (_qword2).eq_u32[1];         \
```



```
1514            _NOTE(CONSTANTCONDITION)                                    \
1515            } while (B_FALSE)

1517 #define EFX_OR_DWORD(_dword1, _dword2)                                  \
1518            do {                                                        \
1519                    (_dword1).ed_u32[0] |= (_dword2).ed_u32[0];         \
1520            _NOTE(CONSTANTCONDITION)                                    \
1521            } while (B_FALSE)

1523 #define EFX_AND_DWORD(_dword1, _dword2)                                 \
1524            do {                                                        \
1525                    (_dword1).ed_u32[0] &= (_dword2).ed_u32[0];         \
1526            _NOTE(CONSTANTCONDITION)                                    \
1527            } while (B_FALSE)

1529 #define EFX_OR_WORD(_word1, _word2)                                     \
1530            do {                                                        \
1531                    (_word1).ew_u16[0] |= (_word2).ew_u16[0];           \
1532            _NOTE(CONSTANTCONDITION)                                    \
1533            } while (B_FALSE)

1535 #define EFX_AND_WORD(_word1, _word2)                                    \
1536            do {                                                        \
1537                    (_word1).ew_u16[0] &= (_word2).ew_u16[0];           \
1538            _NOTE(CONSTANTCONDITION)                                    \
1539            } while (B_FALSE)

1541 #define EFX_OR_BYTE(_byte1, _byte2)                                     \
1542            do {                                                        \
1543                    (_byte1).eb_u8[0] &= (_byte2).eb_u8[0];             \
1544            _NOTE(CONSTANTCONDITION)                                    \
1545            } while (B_FALSE)

1547 #define EFX_AND_BYTE(_byte1, _byte2)                                    \
1548            do {                                                        \
1549                    (_byte1).eb_u8[0] &= (_byte2).eb_u8[0];             \
1550            _NOTE(CONSTANTCONDITION)                                    \
1551            } while (B_FALSE)

1553 #if EFSYS_USE_UINT64
1554 #define EFX_OWORD_FIELD         EFX_OWORD_FIELD64
1555 #define EFX_QWORD_FIELD         EFX_QWORD_FIELD64
1556 #define EFX_OWORD_IS_EQUAL      EFX_OWORD_IS_EQUAL64
1557 #define EFX_QWORD_IS_EQUAL      EFX_QWORD_IS_EQUAL64
1558 #define EFX_OWORD_IS_ZERO       EFX_OWORD_IS_ZERO64
1559 #define EFX_QWORD_IS_ZERO       EFX_QWORD_IS_ZERO64
1560 #define EFX_OWORD_IS_SET        EFX_OWORD_IS_SET64
1561 #define EFX_QWORD_IS_SET        EFX_QWORD_IS_SET64
1562 #define EFX_POPULATE_OWORD      EFX_POPULATE_OWORD64
1563 #define EFX_POPULATE_QWORD      EFX_POPULATE_QWORD64
1564 #define EFX_SET_OWORD           EFX_SET_OWORD64
1565 #define EFX_SET_QWORD           EFX_SET_QWORD64
1566 #define EFX_SET_OWORD_FIELD     EFX_SET_OWORD_FIELD64
1567 #define EFX_SET_QWORD_FIELD     EFX_SET_QWORD_FIELD64
1568 #define EFX_SET_OWORD_BIT       EFX_SET_OWORD_BIT64
1569 #define EFX_CLEAR_OWORD_BIT     EFX_CLEAR_OWORD_BIT64
1570 #define EFX_SET_QWORD_BIT       EFX_SET_QWORD_BIT64
1571 #define EFX_CLEAR_QWORD_BIT     EFX_CLEAR_QWORD_BIT64
1572 #define EFX_OR_OWORD            EFX_OR_OWORD64
1573 #define EFX_AND_OWORD           EFX_AND_OWORD64
1574 #define EFX_OR_QWORD            EFX_OR_QWORD64
1575 #define EFX_AND_QWORD           EFX_OR_QWORD64
1576 #else
1577 #define EFX_OWORD_FIELD         EFX_OWORD_FIELD32
1578 #define EFX_QWORD_FIELD         EFX_QWORD_FIELD32
1579 #define EFX_OWORD_IS_EQUAL      EFX_OWORD_IS_EQUAL32
```

```
#define EFX_QWORD_IS_EQUAL      EFX_QWORD_IS_EQUAL32
#define EFX_OWORD_IS_ZERO       EFX_OWORD_IS_ZERO32
#define EFX_QWORD_IS_ZERO       EFX_QWORD_IS_ZERO32
#define EFX_OWORD_IS_SET        EFX_OWORD_IS_SET32
#define EFX_QWORD_IS_SET        EFX_QWORD_IS_SET32
#define EFX_POPULATE_OWORD      EFX_POPULATE_OWORD32
#define EFX_POPULATE_QWORD      EFX_POPULATE_QWORD32
#define EFX_SET_OWORD           EFX_SET_OWORD32
#define EFX_SET_QWORD           EFX_SET_QWORD32
#define EFX_SET_OWORD_FIELD     EFX_SET_OWORD_FIELD32
#define EFX_SET_QWORD_FIELD     EFX_SET_QWORD_FIELD32
#define EFX_SET_OWORD_BIT       EFX_SET_OWORD_BIT32
#define EFX_CLEAR_OWORD_BIT     EFX_CLEAR_OWORD_BIT32
#define EFX_SET_QWORD_BIT       EFX_SET_QWORD_BIT32
#define EFX_CLEAR_QWORD_BIT     EFX_CLEAR_QWORD_BIT32
#define EFX_OR_OWORD            EFX_OR_OWORD32
#define EFX_AND_OWORD           EFX_AND_OWORD32
#define EFX_OR_QWORD            EFX_OR_QWORD32
#define EFX_AND_QWORD           EFX_OR_QWORD32
#endif

#ifdef  __cplusplus
}
#endif

#endif  /* _SYS_EFX_TYPES_H */
#endif /* ! codereview */
```

```
**********************************************************
   20253 Thu Aug 22 18:59:23 2013
new/usr/src/uts/common/io/sfxge/efx_vpd.c
Merged sfxge driver
**********************************************************
```

```
/*
 * Copyright 2009 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#include "efsys.h"
#include "efx.h"
#include "efx_types.h"
#include "efx_regs.h"
#include "efx_impl.h"

#if EFSYS_OPT_VPD

#define TAG_TYPE_LBN 7
#define TAG_TYPE_WIDTH 1
#define TAG_TYPE_LARGE_ITEM_DECODE 1
#define TAG_TYPE_SMALL_ITEM_DECODE 0

#define TAG_SMALL_ITEM_NAME_LBN 3
#define TAG_SMALL_ITEM_NAME_WIDTH 4
#define TAG_SMALL_ITEM_SIZE_LBN 0
#define TAG_SMALL_ITEM_SIZE_WIDTH 3

#define TAG_LARGE_ITEM_NAME_LBN 0
#define TAG_LARGE_ITEM_NAME_WIDTH 7

#define TAG_NAME_END_DECODE 0x0f
#define TAG_NAME_ID_STRING_DECODE 0x02
#define TAG_NAME_VPD_R_DECODE 0x10
#define TAG_NAME_VPD_W_DECODE 0x11

#if EFSYS_OPT_FALCON

static efx_vpd_ops_t    __cs    __efx_vpd_falcon_ops = {
        NULL,                   /* evpdo_init */
        falcon_vpd_size,        /* evpdo_size */
        falcon_vpd_read,        /* evpdo_read */
        falcon_vpd_verify,      /* evpdo_verify */
        NULL,                   /* evpdo_reinit */
        falcon_vpd_get,         /* evpdo_get */
        falcon_vpd_set,         /* evpdo_set */
```



```
        falcon_vpd_next,        /* evpdo_next */
        falcon_vpd_write,       /* evpdo_write */
        NULL,                   /* evpdo_fini */
};

#endif  /* EFSYS_OPT_FALCON */

#if EFSYS_OPT_SIENA

static efx_vpd_ops_t    __cs    __efx_vpd_siena_ops = {
        siena_vpd_init,         /* evpdo_init */
        siena_vpd_size,         /* evpdo_size */
        siena_vpd_read,         /* evpdo_read */
        siena_vpd_verify,       /* evpdo_verify */
        siena_vpd_reinit,       /* evpdo_reinit */
        siena_vpd_get,          /* evpdo_get */
        siena_vpd_set,          /* evpdo_set */
        siena_vpd_next,         /* evpdo_next */
        siena_vpd_write,        /* evpdo_write */
        siena_vpd_fini,         /* evpdo_fini */
};

#endif  /* EFSYS_OPT_SIENA */

        __checkReturn           int
efx_vpd_init(
        __in                    efx_nic_t *enp)
{
        efx_vpd_ops_t *evpdop;
        int rc;

        EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
        EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_PROBE);
        EFSYS_ASSERT(!(enp->en_mod_flags & EFX_MOD_VPD));

        switch (enp->en_family) {
#if EFSYS_OPT_FALCON
        case EFX_FAMILY_FALCON:
                evpdop = (efx_vpd_ops_t *)&__efx_vpd_falcon_ops;
                break;
#endif  /* EFSYS_OPT_FALCON */

#if EFSYS_OPT_SIENA
        case EFX_FAMILY_SIENA:
                evpdop = (efx_vpd_ops_t *)&__efx_vpd_siena_ops;
                break;
#endif  /* EFSYS_OPT_SIENA */

        default:
                EFSYS_ASSERT(0);
                rc = ENOTSUP;
                goto fail1;
        }

        if (evpdop->evpdo_init != NULL) {
                if ((rc = evpdop->evpdo_init(enp)) != 0)
                        goto fail2;
        }

        enp->en_evpdop = evpdop;
        enp->en_mod_flags |= EFX_MOD_VPD;

        return (0);

fail2:
        EFSYS_PROBE(fail2);
```

```
128 fail1:
129         EFSYS_PROBE1(fail1, int, rc);
130
131         return (rc);
132 }
133
134         __checkReturn           int
135 efx_vpd_size(
136         __in                    efx_nic_t *enp,
137         __out                   size_t *sizep)
138 {
139         efx_vpd_ops_t *evpdop = enp->en_evpdop;
140         int rc;
141
142         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
143         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_VPD);
144
145         if ((rc = evpdop->evpdo_size(enp, sizep)) != 0)
146                 goto fail1;
147
148         return (0);
149
150 fail1:
151         EFSYS_PROBE1(fail1, int, rc);
152
153         return (rc);
154 }
155
156         __checkReturn           int
157 efx_vpd_read(
158         __in                    efx_nic_t *enp,
159         __out_bcount(size)      caddr_t data,
160         __in                    size_t size)
161 {
162         efx_vpd_ops_t *evpdop = enp->en_evpdop;
163         int rc;
164
165         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
166         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_VPD);
167
168         if ((rc = evpdop->evpdo_read(enp, data, size)) != 0)
169                 goto fail1;
170
171         return (0);
172
173 fail1:
174         EFSYS_PROBE1(fail1, int, rc);
175
176         return (rc);
177 }
178
179         __checkReturn           int
180 efx_vpd_verify(
181         __in                    efx_nic_t *enp,
182         __in_bcount(size)       caddr_t data,
183         __in                    size_t size)
184 {
185         efx_vpd_ops_t *evpdop = enp->en_evpdop;
186         int rc;
187
188         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
189         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_VPD);
190
191         if ((rc = evpdop->evpdo_verify(enp, data, size)) != 0)
192                 goto fail1;
```



```
194         return (0);
195
196 fail1:
197         EFSYS_PROBE1(fail1, int, rc);
198
199         return (rc);
200 }
201
202         __checkReturn           int
203 efx_vpd_reinit(
204         __in                    efx_nic_t *enp,
205         __in_bcount(size)       caddr_t data,
206         __in                    size_t size)
207 {
208         efx_vpd_ops_t *evpdop = enp->en_evpdop;
209         int rc;
210
211         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
212         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_VPD);
213
214         if (evpdop->evpdo_reinit == NULL) {
215                 rc = ENOTSUP;
216                 goto fail1;
217         }
218
219         if ((rc = evpdop->evpdo_reinit(enp, data, size)) != 0)
220                 goto fail2;
221
222         return (0);
223
224 fail2:
225         EFSYS_PROBE(fail2);
226 fail1:
227         EFSYS_PROBE1(fail1, int, rc);
228
229         return (rc);
230 }
231
232         __checkReturn           int
233 efx_vpd_get(
234         __in                    efx_nic_t *enp,
235         __in_bcount(size)       caddr_t data,
236         __in                    size_t size,
237         __inout                 efx_vpd_value_t *evvp)
238 {
239         efx_vpd_ops_t *evpdop = enp->en_evpdop;
240         int rc;
241
242         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
243         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_VPD);
244
245         if ((rc = evpdop->evpdo_get(enp, data, size, evvp)) != 0)
246                 goto fail1;
247
248         return (0);
249
250 fail1:
251         EFSYS_PROBE1(fail1, int, rc);
252
253         return (rc);
254 }
255
256         __checkReturn           int
257 efx_vpd_set(
258         __in                    efx_nic_t *enp,
259         __inout_bcount(size)    caddr_t data,
```

```
260         __in                    size_t size,
261         __in                    efx_vpd_value_t *evvp)
262 {
263         efx_vpd_ops_t *evpdop = enp->en_evpdop;
264         int rc;

266         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
267         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_VPD);

269         if ((rc = evpdop->evpdo_set(enp, data, size, evvp)) != 0)
270                 goto fail1;

272         return (0);

274 fail1:
275         EFSYS_PROBE1(fail1, int, rc);

277         return (rc);
278 }

280         __checkReturn           int
281 efx_vpd_next(
282         __in                    efx_nic_t *enp,
283         __inout_bcount(size)    caddr_t data,
284         __in                    size_t size,
285         __out                   efx_vpd_value_t *evvp,
286         __inout                 unsigned int *contp)
287 {
288         efx_vpd_ops_t *evpdop = enp->en_evpdop;
289         int rc;

291         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
292         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_VPD);

294         if ((rc = evpdop->evpdo_next(enp, data, size, evvp, contp)) != 0)
295                 goto fail1;

297         return (0);

299 fail1:
300         EFSYS_PROBE1(fail1, int, rc);

302         return (rc);
303 }

305         __checkReturn           int
306 efx_vpd_write(
307         __in                    efx_nic_t *enp,
308         __in_bcount(size)       caddr_t data,
309         __in                    size_t size)
310 {
311         efx_vpd_ops_t *evpdop = enp->en_evpdop;
312         int rc;

314         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
315         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_VPD);

317         if ((rc = evpdop->evpdo_write(enp, data, size)) != 0)
318                 goto fail1;

320         return (0);

322 fail1:
323         EFSYS_PROBE1(fail1, int, rc);

325         return (rc);
```



```
326 }

328 static  __checkReturn           int
329 efx_vpd_next_tag(
330         __in                    caddr_t data,
331         __in                    size_t size,
332         __inout                 unsigned int *offsetp,
333         __out                   efx_vpd_tag_t *tagp,
334         __out                   uint16_t *lengthp)
335 {
336         efx_byte_t byte;
337         efx_word_t word;
338         uint8_t name;
339         uint16_t length;
340         size_t headlen;
341         int rc;

343         if (*offsetp >= size) {
344                 rc = EFAULT;
345                 goto fail1;
346         }

348         EFX_POPULATE_BYTE_1(byte, EFX_BYTE_0, data[*offsetp]);

350         switch (EFX_BYTE_FIELD(byte, TAG_TYPE)) {
351         case TAG_TYPE_SMALL_ITEM_DECODE:
352                 headlen = 1;

354                 name = EFX_BYTE_FIELD(byte, TAG_SMALL_ITEM_NAME);
355                 length = (uint16_t)EFX_BYTE_FIELD(byte, TAG_SMALL_ITEM_SIZE);

357                 break;

359         case TAG_TYPE_LARGE_ITEM_DECODE:
360                 headlen = 3;

362                 if (*offsetp + headlen > size) {
363                         rc = EFAULT;
364                         goto fail2;
365                 }

367                 name = EFX_BYTE_FIELD(byte, TAG_LARGE_ITEM_NAME);
368                 EFX_POPULATE_WORD_2(word,
369                                     EFX_BYTE_0, data[*offsetp + 1],
370                                     EFX_BYTE_1, data[*offsetp + 2]);
371                 length = EFX_WORD_FIELD(word, EFX_WORD_0);

373                 break;

375         default:
376                 rc = EFAULT;
377                 goto fail2;
378         }

380         if (*offsetp + headlen + length > size) {
381                 rc = EFAULT;
382                 goto fail3;
383         }

385         EFX_STATIC_ASSERT(TAG_NAME_END_DECODE == EFX_VPD_END);
386         EFX_STATIC_ASSERT(TAG_NAME_ID_STRING_DECODE == EFX_VPD_ID);
387         EFX_STATIC_ASSERT(TAG_NAME_VPD_R_DECODE == EFX_VPD_RO);
388         EFX_STATIC_ASSERT(TAG_NAME_VPD_W_DECODE == EFX_VPD_RW);
389         if (name != EFX_VPD_END && name != EFX_VPD_ID &&
390             name != EFX_VPD_RO) {
391                 rc = EFAULT;
```

```
392                 goto fail4;
393         }

395         *tagp = name;
396         *lengthp = length;
397         *offsetp += headlen;

399         return (0);

401 fail4:
402         EFSYS_PROBE(fail4);
403 fail3:
404         EFSYS_PROBE(fail3);
405 fail2:
406         EFSYS_PROBE(fail2);
407 fail1:
408         EFSYS_PROBE1(fail1, int, rc);

410         return (rc);
411 }

413 static  __checkReturn           int
414 efx_vpd_next_keyword(
415         __in_bcount(size)       caddr_t tag,
416         __in                    size_t size,
417         __in                    unsigned int pos,
418         __out                   efx_vpd_keyword_t *keywordp,
419         __out                   uint8_t *lengthp)
420 {
421         efx_vpd_keyword_t keyword;
422         uint8_t length;
423         int rc;

425         if (pos + 3U > size) {
426                 rc = EFAULT;
427                 goto fail1;
428         }

430         keyword = EFX_VPD_KEYWORD(tag[pos], tag[pos + 1]);
431         length = tag[pos + 2];

433         if (length == 0 || pos + 3U + length > size) {
434                 rc = EFAULT;
435                 goto fail2;
436         }

438         *keywordp = keyword;
439         *lengthp = length;

441         return (0);

443 fail2:
444         EFSYS_PROBE(fail2);
445 fail1:
446         EFSYS_PROBE1(fail1, int, rc);

448         return (rc);
449 }

451         __checkReturn           int
452 efx_vpd_hunk_length(
453         __in_bcount(size)       caddr_t data,
454         __in                    size_t size,
455         __out                   size_t *lengthp)
456 {
457         efx_vpd_tag_t tag;
```



```
458         unsigned int offset;
459         uint16_t taglen;
460         int rc;

462         offset = 0;
463         _NOTE(CONSTANTCONDITION)
464         while (1) {
465                 if ((rc = efx_vpd_next_tag(data, size, &offset,
466                     &tag, &taglen)) != 0)
467                         goto fail1;
468                 offset += taglen;
469                 if (tag == EFX_VPD_END)
470                         break;
471         }

473         *lengthp = offset;

475         return (0);

477 fail1:
478         EFSYS_PROBE1(fail1, int, rc);

480         return (rc);
481 }

483         __checkReturn           int
484 efx_vpd_hunk_verify(
485         __in_bcount(size)       caddr_t data,
486         __in                    size_t size,
487         __out_opt               boolean_t *cksummedp)
488 {
489         efx_vpd_tag_t tag;
490         efx_vpd_keyword_t keyword;
491         unsigned int offset;
492         unsigned int pos;
493         unsigned int i;
494         uint16_t taglen;
495         uint8_t keylen;
496         uint8_t cksum;
497         boolean_t cksummed = B_FALSE;
498         int rc;

500         /*
501          * Parse every tag,keyword in the existing VPD. If the csum is present,
502          * the assert it is correct, and is the final keyword in the RO block.
503          */
504         offset = 0;
505         _NOTE(CONSTANTCONDITION)
506         while (1) {
507                 if ((rc = efx_vpd_next_tag(data, size, &offset,
508                     &tag, &taglen)) != 0)
509                         goto fail1;
510                 if (tag == EFX_VPD_END)
511                         break;
512                 else if (tag == EFX_VPD_ID)
513                         goto done;

515                 for (pos = 0; pos != taglen; pos += 3 + keylen) {
516                         /* RV keyword must be the last in the block */
517                         if (cksummed)
518                                 goto fail2;

520                         if ((rc = efx_vpd_next_keyword(data + offset,
521                             taglen, pos, &keyword, &keylen)) != 0)
522                                 goto fail3;
```

```
524                         if (keyword == EFX_VPD_KEYWORD('R', 'V')) {
525                                 cksum = 0;
526                                 for (i = 0; i < offset + pos + 4; i++)
527                                         cksum += data[i];
528
529                                 if (cksum != 0) {
530                                         rc = EFAULT;
531                                         goto fail4;
532                                 }
533
534                                 cksummed = B_TRUE;
535                         }
536                 }
537
538         done:
539                 offset += taglen;
540         }
541
542         if (!cksummed) {
543                 rc = EFAULT;
544                 goto fail5;
545         }
546
547         if (cksummedp != NULL)
548                 *cksummedp = cksummed;
549
550         return (0);
551
552 fail5:
553         EFSYS_PROBE(fail5);
554 fail4:
555         EFSYS_PROBE(fail4);
556 fail3:
557         EFSYS_PROBE(fail3);
558 fail2:
559         EFSYS_PROBE(fail2);
560 fail1:
561         EFSYS_PROBE1(fail1, int, rc);
562
563         return (rc);
564 }
565
566 static  uint8_t __cs    __efx_vpd_blank_pid[] = {
567         /* Large resource type ID length 1 */
568         0x82, 0x01, 0x00,
569         /* Product name ' ' */
570         0x32,
571 };
572
573 static uint8_t __cs     __efx_vpd_blank_r[] = {
574         /* Large resource type VPD-R length 4 */
575         0x90, 0x04, 0x00,
576         /* RV keyword length 1 */
577         'R', 'V', 0x01,
578         /* RV payload checksum */
579         0x00,
580 };
581
582         __checkReturn           int
583 efx_vpd_hunk_reinit(
584         __in                    caddr_t data,
585         __in                    size_t size,
586         __in                    boolean_t wantpid)
587 {
588         unsigned int offset = 0;
589         unsigned int pos;
```



```
590         efx_byte_t byte;
591         uint8_t cksum;
592         int rc;
593
594         if (size < 0x100) {
595                 rc = ENOSPC;
596                 goto fail1;
597         }
598
599         if (wantpid) {
600                 memcpy(data + offset, __efx_vpd_blank_pid,
601                     sizeof (__efx_vpd_blank_pid));
602                 offset += sizeof (__efx_vpd_blank_pid);
603         }
604
605         memcpy(data + offset, __efx_vpd_blank_r, sizeof (__efx_vpd_blank_r));
606         offset += sizeof (__efx_vpd_blank_r);
607
608         /* Update checksum */
609         cksum = 0;
610         for (pos = 0; pos < offset; pos++)
611                 cksum += data[pos];
612         data[offset - 1] -= cksum;
613
614         /* Append trailing tag */
615         EFX_POPULATE_BYTE_3(byte,
616                             TAG_TYPE, TAG_TYPE_SMALL_ITEM_DECODE,
617                             TAG_SMALL_ITEM_NAME, TAG_NAME_END_DECODE,
618                             TAG_SMALL_ITEM_SIZE, 0);
619         data[offset] = EFX_BYTE_FIELD(byte, EFX_BYTE_0);
620         offset++;
621
622         return (0);
623
624 fail1:
625         EFSYS_PROBE1(fail1, int, rc);
626
627         return (rc);
628 }
629
630         __checkReturn                   int
631 efx_vpd_hunk_next(
632         __in_bcount(size)               caddr_t data,
633         __in                            size_t size,
634         __out                           efx_vpd_tag_t *tagp,
635         __out                           efx_vpd_keyword_t *keywordp,
636         __out_bcount_opt(*paylenp)      unsigned int *payloadp,
637         __out_opt                       uint8_t *paylenp,
638         __inout                         unsigned int *contp)
639 {
640         efx_vpd_tag_t tag;
641         efx_vpd_keyword_t keyword = 0;
642         unsigned int offset;
643         unsigned int pos;
644         unsigned int index;
645         uint16_t taglen;
646         uint8_t keylen;
647         uint8_t paylen;
648         int rc;
649
650         offset = index = 0;
651         _NOTE(CONSTANTCONDITION)
652         while (1) {
653                 if ((rc = efx_vpd_next_tag(data, size, &offset,
654                     &tag, &taglen)) != 0)
655                         goto fail1;
```

```
656                 if (tag == EFX_VPD_END)
657                         break;
659                 if (tag == EFX_VPD_ID) {
660                         if (index == *contp) {
661                                 EFSYS_ASSERT3U(taglen, <, 0x100);
662                                 paylen = (uint8_t)MIN(taglen, 0xff);
664                                 goto done;
665                         }
666                 } else {
667                         for (pos = 0; pos != taglen; pos += 3 + keylen) {
668                                 if ((rc = efx_vpd_next_keyword(data + offset,
669                                     taglen, pos, &keyword, &keylen)) != 0)
670                                         goto fail2;
672                                 if (index == *contp) {
673                                         offset += pos + 3;
674                                         paylen = keylen;
676                                         goto done;
677                                 }
678                         }
679                 }
681                 offset += taglen;
682         }
684         *contp = 0;
685         return (0);
687 done:
688         *tagp = tag;
689         *keywordp = keyword;
690         if (payloadp != NULL)
691                 *payloadp = offset;
692         if (paylenp != NULL)
693                 *paylenp = paylen;
695         ++(*contp);
696         return (0);
698 fail2:
699         EFSYS_PROBE(fail2);
700 fail1:
701         EFSYS_PROBE1(fail1, int, rc);
703         return (rc);
704 }
706         __checkReturn           int
707 efx_vpd_hunk_get(
708         __in_bcount(size)       caddr_t data,
709         __in                    size_t size,
710         __in                    efx_vpd_tag_t tag,
711         __in                    efx_vpd_keyword_t keyword,
712         __out                   unsigned int *payloadp,
713         __out                   uint8_t *paylenp)
714 {
715         efx_vpd_tag_t itag;
716         efx_vpd_keyword_t ikeyword;
717         unsigned int offset;
718         unsigned int pos;
719         uint16_t taglen;
720         uint8_t keylen;
721         int rc;
```



```
723         offset = 0;
724         _NOTE(CONSTANTCONDITION)
725         while (1) {
726                 if ((rc = efx_vpd_next_tag(data, size, &offset,
727                     &itag, &taglen)) != 0)
728                         goto fail1;
729                 if (itag == EFX_VPD_END)
730                         break;
732                 if (itag == tag) {
733                         if (itag == EFX_VPD_ID) {
734                                 EFSYS_ASSERT3U(taglen, <, 0x100);
736                                 *paylenp = (uint8_t)MIN(taglen, 0xff);
737                                 *payloadp = offset;
738                                 return (0);
739                         }
741                         for (pos = 0; pos != taglen; pos += 3 + keylen) {
742                                 if ((rc = efx_vpd_next_keyword(data + offset,
743                                     taglen, pos, &ikeyword, &keylen)) != 0)
744                                         goto fail2;
746                                 if (ikeyword == keyword) {
747                                         *paylenp = keylen;
748                                         *payloadp = offset + pos + 3;
749                                         return (0);
750                                 }
751                         }
752                 }
754                 offset += taglen;
755         }
757         /* Not an error */
758         return (ENOENT);
760 fail2:
761         EFSYS_PROBE(fail2);
762 fail1:
763         EFSYS_PROBE1(fail1, int, rc);
765         return (rc);
766 }
768         __checkReturn           int
769 efx_vpd_hunk_set(
770         __in_bcount(size)       caddr_t data,
771         __in                    size_t size,
772         __in                    efx_vpd_value_t *evvp)
773 {
774         efx_word_t word;
775         efx_vpd_tag_t tag;
776         efx_vpd_keyword_t keyword;
777         unsigned int offset;
778         unsigned int pos;
779         unsigned int taghead;
780         unsigned int source;
781         unsigned int dest;
782         unsigned int i;
783         uint16_t taglen;
784         uint8_t keylen;
785         uint8_t cksum;
786         size_t used;
787         int rc;
```

```
789         switch (evvp->evv_tag) {
790         case EFX_VPD_ID:
791                 if (evvp->evv_keyword != 0) {
792                         rc = EINVAL;
793                         goto fail1;
794                 }
796                 /* Can't delete the ID keyword */
797                 if (evvp->evv_length == 0) {
798                         rc = EINVAL;
799                         goto fail1;
800                 }
801                 break;
803         case EFX_VPD_RO:
804                 if (evvp->evv_keyword == EFX_VPD_KEYWORD('R', 'V')) {
805                         rc = EINVAL;
806                         goto fail1;
807                 }
808                 break;
810         default:
811                 rc = EINVAL;
812                 goto fail1;
813         }
815         /* Determine total size of all current tags */
816         if ((rc = efx_vpd_hunk_length(data, size, &used)) != 0)
817                 goto fail2;
819         offset = 0;
820         _NOTE(CONSTANTCONDITION)
821         while (1) {
822                 taghead = offset;
823                 if ((rc = efx_vpd_next_tag(data, size, &offset,
824                     &tag, &taglen)) != 0)
825                         goto fail3;
826                 if (tag == EFX_VPD_END)
827                         break;
828                 else if (tag != evvp->evv_tag) {
829                         offset += taglen;
830                         continue;
831                 }
833                 /* We only support modifying large resource tags */
834                 if (offset - taghead != 3) {
835                         rc = EINVAL;
836                         goto fail4;
837                 }
839                 /*
840                  * Work out the offset of the byte immediately after the
841                  * old (=source) and new (=dest) new keyword/tag
842                  */
843                 pos = 0;
844                 if (tag == EFX_VPD_ID) {
845                         source = offset + taglen;
846                         dest = offset + evvp->evv_length;
847                         goto check_space;
848                 }
850                 EFSYS_ASSERT3U(tag, ==, EFX_VPD_RO);
851                 source = dest = 0;
852                 for (pos = 0; pos != taglen; pos += 3 + keylen) {
853                         if ((rc = efx_vpd_next_keyword(data + offset,
```



```
854                             taglen, pos, &keyword, &keylen)) != 0)
855                                 goto fail5;
857                         if (keyword == evvp->evv_keyword &&
858                             evvp->evv_length == 0) {
859                                 /* Deleting this keyword */
860                                 source = offset + pos + 3 + keylen;
861                                 dest = offset + pos;
862                                 break;
864                         } else if (keyword == evvp->evv_keyword) {
865                                 /* Adjusting this keyword */
866                                 source = offset + pos + 3 + keylen;
867                                 dest = offset + pos + 3 + evvp->evv_length;
868                                 break;
870                         } else if (keyword == EFX_VPD_KEYWORD('R', 'V')) {
871                                 /* The RV keyword must be at the end */
872                                 EFSYS_ASSERT3U(pos + 3 + keylen, ==, taglen);
874                                 /*
875                                  * The keyword doesn't already exist. If the
876                                  * user deleting a non-existant keyword then
877                                  * this is a no-op.
878                                  */
879                                 if (evvp->evv_length == 0)
880                                         return (0);
882                                 /* Insert this keyword before the RV keyword */
883                                 source = offset + pos;
884                                 dest = offset + pos + 3 + evvp->evv_length;
885                                 break;
886                         }
887                 }
889         check_space:
890                 if (used + dest > size + source) {
891                         rc = ENOSPC;
892                         goto fail6;
893                 }
895                 /* Move trailing data */
896                 (void) memmove(data + dest, data + source, used - source);
898                 /* Copy contents */
899                 memcpy(data + dest - evvp->evv_length, evvp->evv_value,
900                     evvp->evv_length);
902                 /* Insert new keyword header if required */
903                 if (tag != EFX_VPD_ID && evvp->evv_length > 0) {
904                         EFX_POPULATE_WORD_1(word, EFX_WORD_0,
905                                             evvp->evv_keyword);
906                         data[offset + pos + 0] =
907                             EFX_WORD_FIELD(word, EFX_BYTE_0);
908                         data[offset + pos + 1] =
909                             EFX_WORD_FIELD(word, EFX_BYTE_1);
910                         data[offset + pos + 2] = evvp->evv_length;
911                 }
913                 /* Modify tag length (large resource type) */
914                 taglen += (dest - source);
915                 EFX_POPULATE_WORD_1(word, EFX_WORD_0, taglen);
916                 data[offset - 2] = EFX_WORD_FIELD(word, EFX_BYTE_0);
917                 data[offset - 1] = EFX_WORD_FIELD(word, EFX_BYTE_1);
919                 goto checksum;
```

```
920         }

922         /* Unable to find the matching tag */
923         rc = ENOENT;
924         goto fail7;

926 checksum:
927         /* Find the RV tag, and update the checksum */
928         offset = 0;
929         _NOTE(CONSTANTCONDITION)
930         while (1) {
931                 if ((rc = efx_vpd_next_tag(data, size, &offset,
932                     &tag, &taglen)) != 0)
933                         goto fail8;
934                 if (tag == EFX_VPD_END)
935                         break;
936                 if (tag == EFX_VPD_RO) {
937                         for (pos = 0; pos != taglen; pos += 3 + keylen) {
938                                 if ((rc = efx_vpd_next_keyword(data + offset,
939                                     taglen, pos, &keyword, &keylen)) != 0)
940                                         goto fail9;

942                                 if (keyword == EFX_VPD_KEYWORD('R', 'V')) {
943                                         cksum = 0;
944                                         for (i = 0; i < offset + pos + 3; i++)
945                                                 cksum += data[i];
946                                         data[i] = -cksum;
947                                         break;
948                                 }
949                         }
950                 }

952                 offset += taglen;
953         }

955         /* Zero out the unused portion */
956         (void) memset(data + offset + taglen, 0xff, size - offset - taglen);

958         return (0);

960 fail9:
961         EFSYS_PROBE(fail9);
962 fail8:
963         EFSYS_PROBE(fail8);
964 fail7:
965         EFSYS_PROBE(fail7);
966 fail6:
967         EFSYS_PROBE(fail6);
968 fail5:
969         EFSYS_PROBE(fail5);
970 fail4:
971         EFSYS_PROBE(fail4);
972 fail3:
973         EFSYS_PROBE(fail3);
974 fail2:
975         EFSYS_PROBE(fail2);
976 fail1:
977         EFSYS_PROBE1(fail1, int, rc);

979         return (rc);
980 }

982                                 void
983 efx_vpd_fini(
984         __in                    efx_nic_t *enp)
985 {
```



```
986         efx_vpd_ops_t *evpdop = enp->en_evpdop;

988         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
989         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_PROBE);
990         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_VPD);

992         if (evpdop->evpdo_fini != NULL)
993                 evpdop->evpdo_fini(enp);

995         enp->en_evpdop = NULL;
996         enp->en_mod_flags &= ~EFX_MOD_VPD;
997 }

999 #endif  /* EFSYS_OPT_VPD */
1000 #endif /* ! codereview */
```

```
**********************************************************
   10114 Thu Aug 22 18:59:23 2013
new/usr/src/uts/common/io/sfxge/efx_wol.c
Merged sfxge driver
**********************************************************
```

```
/*
 * Copyright 2009 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#include "efsys.h"
#include "efx.h"
#include "efx_types.h"
#include "efx_impl.h"

#if EFSYS_OPT_WOL

        __checkReturn   int
efx_wol_init(
        __in            efx_nic_t *enp)
{
        efx_nic_cfg_t *encp = &(enp->en_nic_cfg);
        int rc;

        EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
        EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_PROBE);
        EFSYS_ASSERT(!(enp->en_mod_flags & EFX_MOD_WOL));

        if (~(encp->enc_features) & EFX_FEATURE_WOL) {
                rc = ENOTSUP;
                goto fail1;
        }

        /* Current implementation is Siena specific */
        EFSYS_ASSERT3U(enp->en_family, ==, EFX_FAMILY_SIENA);

        enp->en_mod_flags |= EFX_MOD_WOL;

        return (0);

fail1:
        EFSYS_PROBE1(fail1, int, rc);

        return (rc);
}
```



```
        __checkReturn   int
efx_wol_filter_clear(
        __in            efx_nic_t *enp)
{
        efx_mcdi_req_t req;
        uint8_t payload[MC_CMD_WOL_FILTER_RESET_IN_LEN];
        int rc;

        EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
        EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_WOL);

        req.emr_cmd = MC_CMD_WOL_FILTER_RESET;
        req.emr_in_buf = payload;
        req.emr_in_length = MC_CMD_WOL_FILTER_RESET_IN_LEN;
        req.emr_out_buf = NULL;
        req.emr_out_length = 0;

        MCDI_IN_SET_DWORD(req, WOL_FILTER_RESET_IN_MASK,
                          MC_CMD_WOL_FILTER_RESET_IN_WAKE_FILTERS |
                          MC_CMD_WOL_FILTER_RESET_IN_LIGHTSOUT_OFFLOADS);

        efx_mcdi_execute(enp, &req);

        if (req.emr_rc != 0) {
                rc = req.emr_rc;
                goto fail1;
        }

        return (0);

fail1:
        EFSYS_PROBE1(fail1, int, rc);

        return (rc);
}

        __checkReturn   int
efx_wol_filter_add(
        __in            efx_nic_t *enp,
        __in            efx_wol_type_t type,
        __in            efx_wol_param_t *paramp,
        __out           uint32_t *filter_idp)
{
        efx_mcdi_req_t req;
        uint8_t payload[MAX(MC_CMD_WOL_FILTER_SET_IN_LEN,
                            MC_CMD_WOL_FILTER_SET_OUT_LEN)];
        efx_byte_t link_mask;
        int rc;

        EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
        EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_WOL);

        req.emr_cmd = MC_CMD_WOL_FILTER_SET;
        (void) memset(payload, '\0', sizeof (payload));
        req.emr_in_buf = payload;
        req.emr_in_length = MC_CMD_WOL_FILTER_SET_IN_LEN;
        req.emr_out_buf = payload;
        req.emr_out_length = MC_CMD_WOL_FILTER_SET_OUT_LEN;

        switch (type) {
        case EFX_WOL_TYPE_MAGIC:
                MCDI_IN_SET_DWORD(req, WOL_FILTER_SET_IN_FILTER_MODE,
                                  MC_CMD_FILTER_MODE_SIMPLE);
                MCDI_IN_SET_DWORD(req, WOL_FILTER_SET_IN_WOL_TYPE,
                                  MC_CMD_WOL_TYPE_MAGIC);
                EFX_MAC_ADDR_COPY(
```

```
128                     MCDI_IN2(req, uint8_t, WOL_FILTER_SET_IN_MAGIC_MAC),
129                     paramp->ewp_magic.mac_addr);
130             break;

132         case EFX_WOL_TYPE_BITMAP: {
133             uint32_t swapped = 0;
134             efx_dword_t *dwordp;
135             unsigned int pos, bit;

137             MCDI_IN_SET_DWORD(req, WOL_FILTER_SET_IN_FILTER_MODE,
138                               MC_CMD_FILTER_MODE_SIMPLE);
139             MCDI_IN_SET_DWORD(req, WOL_FILTER_SET_IN_WOL_TYPE,
140                               MC_CMD_WOL_TYPE_BITMAP);

142             /*
143              * MC bitmask is supposed to be bit swapped
144              * amongst 32 bit words(!)
145              */

147             dwordp = MCDI_IN2(req, efx_dword_t,
148                               WOL_FILTER_SET_IN_BITMAP_MASK);

150             EFSYS_ASSERT3U(EFX_WOL_BITMAP_MASK_SIZE % 4, ==, 0);

152             for (pos = 0; pos < EFX_WOL_BITMAP_MASK_SIZE; ++pos) {
153                 uint8_t native = paramp->ewp_bitmap.mask[pos];

155                 for (bit = 0; bit < 8; ++bit) {
156                     swapped <<= 1;
157                     swapped |= (native & 0x1);
158                     native >>= 1;
159                 }

161                 if ((pos & 3) == 3) {
162                     EFX_POPULATE_DWORD_1(dwordp[pos >> 2],
163                         EFX_DWORD_0, swapped);
164                     swapped = 0;
165                 }
166             }

168             memcpy(MCDI_IN2(req, uint8_t, WOL_FILTER_SET_IN_BITMAP_BITMAP),
169                 paramp->ewp_bitmap.value,
170                 sizeof (paramp->ewp_bitmap.value));

172             EFSYS_ASSERT3U(paramp->ewp_bitmap.value_len, <=,
173                            sizeof (paramp->ewp_bitmap.value));
174             MCDI_IN_SET_DWORD(req, WOL_FILTER_SET_IN_BITMAP_LEN,
175                               paramp->ewp_bitmap.value_len);
176             }
177             break;

179         case EFX_WOL_TYPE_LINK:
180             MCDI_IN_SET_DWORD(req, WOL_FILTER_SET_IN_FILTER_MODE,
181                               MC_CMD_FILTER_MODE_SIMPLE);
182             MCDI_IN_SET_DWORD(req, WOL_FILTER_SET_IN_WOL_TYPE,
183                               MC_CMD_WOL_TYPE_LINK);

185             EFX_ZERO_BYTE(link_mask);
186             EFX_SET_BYTE_FIELD(link_mask, MC_CMD_WOL_FILTER_SET_IN_LINK_UP,
187                 1);
188             MCDI_IN_SET_BYTE(req, WOL_FILTER_SET_IN_LINK_MASK,
189                 link_mask.eb_u8[0]);
190             break;

192         default:
193             EFSYS_ASSERT3U(type, !=, type);
```



```
194         }

196         efx_mcdi_execute(enp, &req);

198         if (req.emr_rc != 0) {
199                 rc = req.emr_rc;
200                 goto fail1;
201         }

203         if (req.emr_out_length_used < MC_CMD_WOL_FILTER_SET_OUT_LEN) {
204                 rc = EMSGSIZE;
205                 goto fail2;
206         }

208         *filter_idp = MCDI_OUT_DWORD(req, WOL_FILTER_SET_OUT_FILTER_ID);

210         return (0);

212 fail2:
213         EFSYS_PROBE(fail2);
214 fail1:
215         EFSYS_PROBE1(fail1, int, rc);

217         return (rc);
218 }

220         __checkReturn   int
221 efx_wol_filter_remove(
222         __in            efx_nic_t *enp,
223         __in            uint32_t filter_id)
224 {
225         efx_mcdi_req_t req;
226         uint8_t payload[MC_CMD_WOL_FILTER_REMOVE_IN_LEN];
227         int rc;

229         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
230         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_WOL);

232         req.emr_cmd = MC_CMD_WOL_FILTER_REMOVE;
233         req.emr_in_buf = payload;
234         req.emr_in_length = MC_CMD_WOL_FILTER_REMOVE_IN_LEN;
235         EFX_STATIC_ASSERT(MC_CMD_WOL_FILTER_REMOVE_OUT_LEN == 0);
236         req.emr_out_buf = NULL;
237         req.emr_out_length = 0;

239         MCDI_IN_SET_DWORD(req, WOL_FILTER_REMOVE_IN_FILTER_ID, filter_id);

241         efx_mcdi_execute(enp, &req);

243         if (req.emr_rc != 0) {
244                 rc = req.emr_rc;
245                 goto fail1;
246         }

248         return (0);

250 fail1:
251         EFSYS_PROBE1(fail1, int, rc);

253         return (rc);
254 }


257         __checkReturn   int
258 efx_lightsout_offload_add(
259         __in            efx_nic_t *enp,
```

```
260         __in            efx_lightsout_offload_type_t type,
261         __in            efx_lightsout_offload_param_t *paramp,
262         __out           uint32_t *filter_idp)
263 {
264         efx_mcdi_req_t req;
265         uint8_t payload[MAX(MAX(MC_CMD_ADD_LIGHTSOUT_OFFLOAD_IN_ARP_LEN,
266                                 MC_CMD_ADD_LIGHTSOUT_OFFLOAD_IN_NS_LEN),
267                             MC_CMD_ADD_LIGHTSOUT_OFFLOAD_OUT_LEN)];
268         int rc;

270         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
271         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_WOL);

273         req.emr_cmd = MC_CMD_ADD_LIGHTSOUT_OFFLOAD;
274         req.emr_in_buf = payload;
275         req.emr_in_length = sizeof (type);
276         req.emr_out_buf = payload;
277         req.emr_out_length = MC_CMD_ADD_LIGHTSOUT_OFFLOAD_OUT_LEN;

279         switch (type) {
280         case EFX_LIGHTSOUT_OFFLOAD_TYPE_ARP:
281                 req.emr_in_length = MC_CMD_ADD_LIGHTSOUT_OFFLOAD_IN_ARP_LEN;
282                 MCDI_IN_SET_DWORD(req, ADD_LIGHTSOUT_OFFLOAD_IN_PROTOCOL,
283                                   MC_CMD_LIGHTSOUT_OFFLOAD_PROTOCOL_ARP);
284                 EFX_MAC_ADDR_COPY(MCDI_IN2(req, uint8_t,
285                                            ADD_LIGHTSOUT_OFFLOAD_IN_ARP_MAC),
286                                   paramp->elop_arp.mac_addr);
287                 MCDI_IN_SET_DWORD(req, ADD_LIGHTSOUT_OFFLOAD_IN_ARP_IP,
288                                   paramp->elop_arp.ip);
289                 break;
290         case EFX_LIGHTSOUT_OFFLOAD_TYPE_NS:
291                 req.emr_in_length = MC_CMD_ADD_LIGHTSOUT_OFFLOAD_IN_NS_LEN;
292                 MCDI_IN_SET_DWORD(req, ADD_LIGHTSOUT_OFFLOAD_IN_PROTOCOL,
293                                   MC_CMD_LIGHTSOUT_OFFLOAD_PROTOCOL_NS);
294                 EFX_MAC_ADDR_COPY(MCDI_IN2(req, uint8_t,
295                                            ADD_LIGHTSOUT_OFFLOAD_IN_NS_MAC),
296                                   paramp->elop_ns.mac_addr);
297                 memcpy(MCDI_IN2(req, uint8_t,
298                     ADD_LIGHTSOUT_OFFLOAD_IN_NS_SNIPV6),
299                     paramp->elop_ns.solicited_node,
300                     sizeof (paramp->elop_ns.solicited_node));
301                 memcpy(MCDI_IN2(req, uint8_t, ADD_LIGHTSOUT_OFFLOAD_IN_NS_IPV6),
302                     paramp->elop_ns.ip, sizeof (paramp->elop_ns.ip));
303                 break;
304         default:
305                 EFSYS_ASSERT3U(type, !=, type);
306         }

308         efx_mcdi_execute(enp, &req);

310         if (req.emr_rc != 0) {
311                 rc = req.emr_rc;
312                 goto fail1;
313         }

315         if (req.emr_out_length_used < MC_CMD_ADD_LIGHTSOUT_OFFLOAD_OUT_LEN) {
316                 rc = EMSGSIZE;
317                 goto fail2;
318         }

320         *filter_idp = MCDI_OUT_DWORD(req, ADD_LIGHTSOUT_OFFLOAD_OUT_FILTER_ID);

322         return (0);

324 fail2:
325         EFSYS_PROBE(fail2);
```



```
326 fail1:
327         EFSYS_PROBE1(fail1, int, rc);

329         return (rc);
330 }


333         __checkReturn   int
334 efx_lightsout_offload_remove(
335         __in            efx_nic_t *enp,
336         __in            efx_lightsout_offload_type_t type,
337         __in            uint32_t filter_id)
338 {
339         efx_mcdi_req_t req;
340         uint8_t payload[MC_CMD_REMOVE_LIGHTSOUT_OFFLOAD_IN_LEN];
341         int rc;

343         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
344         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_WOL);

346         req.emr_cmd = MC_CMD_REMOVE_LIGHTSOUT_OFFLOAD;
347         req.emr_in_buf = payload;
348         req.emr_in_length = sizeof (payload);
349         EFX_STATIC_ASSERT(MC_CMD_REMOVE_LIGHTSOUT_OFFLOAD_OUT_LEN == 0);
350         req.emr_out_buf = NULL;
351         req.emr_out_length = 0;

353         switch (type) {
354         case EFX_LIGHTSOUT_OFFLOAD_TYPE_ARP:
355                 MCDI_IN_SET_DWORD(req, REMOVE_LIGHTSOUT_OFFLOAD_IN_PROTOCOL,
356                                   MC_CMD_LIGHTSOUT_OFFLOAD_PROTOCOL_ARP);
357                 break;
358         case EFX_LIGHTSOUT_OFFLOAD_TYPE_NS:
359                 MCDI_IN_SET_DWORD(req, REMOVE_LIGHTSOUT_OFFLOAD_IN_PROTOCOL,
360                                   MC_CMD_LIGHTSOUT_OFFLOAD_PROTOCOL_NS);
361                 break;
362         default:
363                 EFSYS_ASSERT3U(type, !=, type);
364         }

366         MCDI_IN_SET_DWORD(req, REMOVE_LIGHTSOUT_OFFLOAD_IN_FILTER_ID,
367                           filter_id);

369         efx_mcdi_execute(enp, &req);

371         if (req.emr_rc != 0) {
372                 rc = req.emr_rc;
373                 goto fail1;
374         }

376         return (0);

378 fail1:
379         EFSYS_PROBE1(fail1, int, rc);

381         return (rc);
382 }


385                         void
386 efx_wol_fini(
387         __in            efx_nic_t *enp)
388 {
389         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
390         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_PROBE);
391         EFSYS_ASSERT3U(enp->en_mod_flags, &, EFX_MOD_WOL);
```

```
393         enp->en_mod_flags &= ~EFX_MOD_WOL;
394 }

396 #endif  /* EFSYS_OPT_WOL */
397 #endif /* ! codereview */
```

```
**********************************************************
   12512 Thu Aug 22 18:59:23 2013
new/usr/src/uts/common/io/sfxge/falcon_gmac.c
Merged sfxge driver
**********************************************************
```

```
/*
 * Copyright 2008-2013 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#include "efsys.h"
#include "efx.h"
#include "efx_types.h"
#include "efx_regs.h"
#include "efx_impl.h"
#include "falcon_gmac.h"
#include "falcon_stats.h"

#if EFSYS_OPT_MAC_FALCON_GMAC

        __checkReturn   int
falcon_gmac_reset(
        __in            efx_nic_t *enp)
{
        efx_port_t *epp = &(enp->en_port);
        efx_oword_t oword;
        int rc;

        EFSYS_ASSERT3U(epp->ep_mac_type, ==, EFX_MAC_FALCON_GMAC);

        EFSYS_PROBE(reset);

        /* Ensure that the GMAC registers are accessible */
        EFX_BAR_READO(enp, FR_AB_NIC_STAT_REG, &oword);
        EFX_SET_OWORD_FIELD(oword, FRF_BB_EE_STRAP_EN, 1);
        EFX_SET_OWORD_FIELD(oword, FRF_BB_EE_STRAP, 0x3);
        EFX_BAR_WRITEO(enp, FR_AB_NIC_STAT_REG, &oword);

        if ((rc = falcon_mac_wrapper_disable(enp)) != 0)
            goto fail1;

        enp->en_reset_flags |= EFX_RESET_MAC;

        return (0);

fail1:
```



```
        EFSYS_PROBE1(fail1, int, rc);

        return (rc);
}

static                  void
falcon_gmac_core_reconfigure(
        __in    efx_nic_t *enp)
{
        efx_port_t *epp = &(enp->en_port);
        efx_oword_t oword;
        boolean_t full_duplex;
        boolean_t byte_mode;

        full_duplex = (epp->ep_link_mode == EFX_LINK_100FDX ||
            epp->ep_link_mode == EFX_LINK_1000FDX);
        byte_mode = (epp->ep_link_mode == EFX_LINK_1000HDX ||
            epp->ep_link_mode == EFX_LINK_1000FDX);
#if EFSYS_OPT_LOOPBACK
        byte_mode |= (epp->ep_loopback_type == EFX_LOOPBACK_GMAC);

        EFX_POPULATE_OWORD_5(oword,
            FRF_AB_GM_LOOP,
            (epp->ep_loopback_type == EFX_LOOPBACK_GMAC) ? 1 : 0,
            FRF_AB_GM_TX_EN, 1,
            FRF_AB_GM_TX_FC_EN, (epp->ep_fcntl & EFX_FCNTL_GENERATE) ? 1 : 0,
            FRF_AB_GM_RX_EN, 1,
            FRF_AB_GM_RX_FC_EN, (epp->ep_fcntl & EFX_FCNTL_RESPOND) ? 1 : 0);

#else   /* EFSYS_OPT_LOOPBACK */

        EFX_POPULATE_OWORD_4(oword,
            FRF_AB_GM_TX_EN, 1,
            FRF_AB_GM_TX_FC_EN, (epp->ep_fcntl & EFX_FCNTL_GENERATE) ? 1 : 0,
            FRF_AB_GM_RX_EN, 1,
            FRF_AB_GM_RX_FC_EN, (epp->ep_fcntl & EFX_FCNTL_RESPOND) ? 1 : 0);

#endif  /* EFSYS_OPT_LOOPBACK */

        EFX_BAR_WRITEO(enp, FR_AB_GM_CFG1_REG, &oword);
        EFSYS_SPIN(10);

        EFX_POPULATE_OWORD_4(oword,
            FRF_AB_GM_IF_MODE, (byte_mode) ?
            FRF_AB_GM_IF_MODE_BYTE_MODE : FRF_AB_GM_IF_MODE_NIBBLE_MODE,
            FRF_AB_GM_PAD_CRC_EN, 1,
            FRF_AB_GM_FD, full_duplex ? 1 : 0,
            FRF_AB_GM_PAMBL_LEN, 7);

        EFX_BAR_WRITEO(enp, FR_AB_GM_CFG2_REG, &oword);
        EFSYS_SPIN(10);

        EFX_POPULATE_OWORD_1(oword, FRF_AB_GM_MAX_FLEN, epp->ep_mac_pdu);

        EFX_BAR_WRITEO(enp, FR_AB_GM_MAX_FLEN_REG, &oword);
        EFSYS_SPIN(10);

        EFX_POPULATE_OWORD_5(oword,
            FRF_AB_GMF_FTFENREQ, 1,
            FRF_AB_GMF_STFENREQ, 1,
            FRF_AB_GMF_FRFENREQ, 1,
            FRF_AB_GMF_SRFENREQ, 1,
            FRF_AB_GMF_WTMENREQ, 1);

        EFX_BAR_WRITEO(enp, FR_AB_GMF_CFG0_REG, &oword);
        EFSYS_SPIN(10);
```

```
129         EFX_POPULATE_OWORD_2(oword,
130             FRF_AB_GMF_CFGFRTH, 0x12,
131             FRF_AB_GMF_CFGXOFFRTX, 0xffff);
132
133         EFX_BAR_WRITEO(enp, FR_AB_GMF_CFG1_REG, &oword);
134         EFSYS_SPIN(10);
135
136         EFX_POPULATE_OWORD_2(oword,
137             FRF_AB_GMF_CFGHWM, 0x3f,
138             FRF_AB_GMF_CFGLWM, 0x0a);
139
140         EFX_BAR_WRITEO(enp, FR_AB_GMF_CFG2_REG, &oword);
141         EFSYS_SPIN(10);
142
143         EFX_POPULATE_OWORD_2(oword,
144             FRF_AB_GMF_CFGHWMFT, 0x1c,
145             FRF_AB_GMF_CFGFTTH, 0x08);
146
147         EFX_BAR_WRITEO(enp, FR_AB_GMF_CFG3_REG, &oword);
148         EFSYS_SPIN(10);
149
150         EFX_POPULATE_OWORD_1(oword, FRF_AB_GMF_HSTFLTRFRM, 0x1000);
151
152         EFX_BAR_WRITEO(enp, FR_AB_GMF_CFG4_REG, &oword);
153         EFSYS_SPIN(10);
154
155         EFX_BAR_READO(enp, FR_AB_GMF_CFG5_REG, &oword);
156         EFX_SET_OWORD_FIELD(oword, FRF_AB_GMF_CFGBYTMODE, byte_mode ? 1 : 0);
157         EFX_SET_OWORD_FIELD(oword, FRF_AB_GMF_CFGHDPLX, full_duplex ? 0 : 1);
158         EFX_SET_OWORD_FIELD(oword,
159             FRF_AB_GMF_HSTDRPLT64, full_duplex ? 0 : 1);
160         EFX_SET_OWORD_FIELD(oword, FRF_AB_GMF_HSTFLTRFRMDC, 0x3efff);
161         EFX_BAR_WRITEO(enp, FR_AB_GMF_CFG5_REG, &oword);
162         EFSYS_SPIN(10);
163
164         EFX_POPULATE_OWORD_4(oword,
165             FRF_AB_GM_ADR_B0, epp->ep_mac_addr[5],
166             FRF_AB_GM_ADR_B1, epp->ep_mac_addr[4],
167             FRF_AB_GM_ADR_B2, epp->ep_mac_addr[3],
168             FRF_AB_GM_ADR_B3, epp->ep_mac_addr[2]);
169
170         EFX_BAR_WRITEO(enp, FR_AB_GM_ADR1_REG, &oword);
171         EFSYS_SPIN(10);
172
173         EFX_POPULATE_OWORD_2(oword,
174             FRF_AB_GM_ADR_B4, epp->ep_mac_addr[1],
175             FRF_AB_GM_ADR_B5, epp->ep_mac_addr[0]);
176
177         EFX_BAR_WRITEO(enp, FR_AB_GM_ADR2_REG, &oword);
178         EFSYS_SPIN(10);
179 }
180
181         __checkReturn   int
182 falcon_gmac_downlink_check(
183         __in            efx_nic_t *enp,
184         __out           boolean_t *upp)
185 {
186         efx_port_t *epp = &(enp->en_port);
187
188         _NOTE(ARGUNUSED(upp))
189
190         EFSYS_ASSERT3U(epp->ep_mac_type, ==, EFX_MAC_FALCON_GMAC);
191
192         return (ENOTSUP);
193 }
```



```
195         __checkReturn   int
196 falcon_gmac_reconfigure(
197         __in            efx_nic_t *enp)
198 {
199         efx_port_t *epp = &(enp->en_port);
200
201         EFSYS_ASSERT3U(epp->ep_mac_type, ==, EFX_MAC_FALCON_GMAC);
202         EFSYS_ASSERT3U(epp->ep_link_mode, !=, EFX_LINK_10000FDX);
203
204         EFSYS_PROBE(reconfigure);
205
206         falcon_gmac_core_reconfigure(enp);
207         falcon_mac_wrapper_enable(enp);
208
209         return (0);
210 }
211
212                 void
213 falcon_gmac_downlink_reset(
214         __in    efx_nic_t *enp,
215         __in    boolean_t hold)
216 {
217         _NOTE(ARGUNUSED(enp, hold))
218 }
219
220 #if EFSYS_OPT_MAC_STATS
221 static          uint32_t
222 falcon_gmac_stat_read(
223         __in    efsys_mem_t *esmp,
224         __in    unsigned int offset,
225         __in    unsigned int width)
226 {
227         uint32_t val;
228
229         switch (width) {
230         case 2: {
231                 efx_dword_t dword;
232
233                 EFSYS_MEM_READD(esmp, offset, &dword);
234
235                 val = (uint16_t)EFX_DWORD_FIELD(dword, EFX_WORD_0);
236                 break;
237         }
238         case 4: {
239                 efx_dword_t dword;
240
241                 EFSYS_MEM_READD(esmp, offset, &dword);
242
243                 val = EFX_DWORD_FIELD(dword, EFX_DWORD_0);
244                 break;
245         }
246         case 6: {
247                 efx_qword_t qword;
248
249                 EFSYS_MEM_READQ(esmp, offset, &qword);
250
251                 val = EFX_QWORD_FIELD(qword, EFX_DWORD_0);
252                 break;
253         }
254         default:
255                 EFSYS_ASSERT(B_FALSE);
256
257                 val = 0;
258                 break;
259         }
```

```
261         return (val);
262 }
263
264 #define GMAC_STAT_READ(_esmp, _id)                                      \
265         falcon_gmac_stat_read((_esmp), G_STAT_OFFSET(_id),              \
266             G_STAT_WIDTH(_id))
267
268 #define GMAC_STAT_INCR(_esmp, _stat, _id)                               \
269         do {                                                            \
270                 delta = GMAC_STAT_READ(_esmp, _id);                     \
271                 EFSYS_STAT_INCR(&(_stat), delta);                       \
272         _NOTE(CONSTANTCONDITION)                                        \
273         } while (0)
274
275 #define GMAC_STAT_DECR(_esmp, _stat, _id)                               \
276         do {                                                            \
277                 delta = GMAC_STAT_READ(_esmp, _id);                     \
278                 EFSYS_STAT_DECR(&(_stat), delta);                       \
279         _NOTE(CONSTANTCONDITION)                                        \
280         } while (0)
281
282         __checkReturn                   int
283 falcon_gmac_stats_update(
284         __in                            efx_nic_t *enp,
285         __in                            efsys_mem_t *esmp,
286         __inout_ecount(EFX_MAC_NSTATS)  efsys_stat_t *stat,
287         __out_opt                       uint32_t *generationp)
288 {
289         efx_port_t *epp = &(enp->en_port);
290         efx_oword_t oword;
291         uint32_t delta;
292
293         _NOTE(ARGUNUSED(generationp));
294         EFSYS_ASSERT3U(epp->ep_mac_type, ==, EFX_MAC_FALCON_GMAC);
295
296         /* Check the DMA flag */
297         if (GMAC_STAT_READ(esmp, DMA_DONE) != DMA_IS_DONE)
298                 return (EAGAIN);
299         EFSYS_MEM_READ_BARRIER();
300
301         /* RX */
302         GMAC_STAT_INCR(esmp, stat[EFX_MAC_RX_OCTETS], RX_GOOD_OCT);
303         GMAC_STAT_INCR(esmp, stat[EFX_MAC_RX_OCTETS], RX_BAD_OCT);
304
305         GMAC_STAT_INCR(esmp, stat[EFX_MAC_RX_PKTS], RX_GOOD_LT_64_PKT);
306         GMAC_STAT_INCR(esmp, stat[EFX_MAC_RX_PKTS], RX_64_PKT);
307         GMAC_STAT_INCR(esmp, stat[EFX_MAC_RX_PKTS], RX_65_TO_127_PKT);
308         GMAC_STAT_INCR(esmp, stat[EFX_MAC_RX_PKTS], RX_128_TO_255_PKT);
309         GMAC_STAT_INCR(esmp, stat[EFX_MAC_RX_PKTS], RX_256_TO_511_PKT);
310         GMAC_STAT_INCR(esmp, stat[EFX_MAC_RX_PKTS], RX_512_TO_1023_PKT);
311         GMAC_STAT_INCR(esmp, stat[EFX_MAC_RX_PKTS], RX_1024_TO_15XX_PKT);
312         GMAC_STAT_INCR(esmp, stat[EFX_MAC_RX_PKTS], RX_15XX_TO_JUMBO_PKT);
313         GMAC_STAT_INCR(esmp, stat[EFX_MAC_RX_PKTS], RX_GT_JUMBO_PKT);
314
315         GMAC_STAT_INCR(esmp, stat[EFX_MAC_RX_UNICST_PKTS], RX_UCAST_PKT);
316         GMAC_STAT_INCR(esmp, stat[EFX_MAC_RX_MULTICST_PKTS], RX_MCAST_PKT);
317         GMAC_STAT_INCR(esmp, stat[EFX_MAC_RX_BRDCST_PKTS], RX_BCAST_PKT);
318         GMAC_STAT_INCR(esmp, stat[EFX_MAC_RX_PAUSE_PKTS], RX_PAUSE_PKT);
319
320         GMAC_STAT_INCR(esmp, stat[EFX_MAC_RX_LE_64_PKTS], RX_GOOD_LT_64_PKT);
321         GMAC_STAT_INCR(esmp, stat[EFX_MAC_RX_LE_64_PKTS], RX_BAD_LT_64_PKT);
322         GMAC_STAT_INCR(esmp, stat[EFX_MAC_RX_LE_64_PKTS], RX_64_PKT);
323         GMAC_STAT_INCR(esmp, stat[EFX_MAC_RX_65_TO_127_PKTS],
324             RX_65_TO_127_PKT);
325         GMAC_STAT_INCR(esmp, stat[EFX_MAC_RX_128_TO_255_PKTS],
```



```
326             RX_128_TO_255_PKT);
327         GMAC_STAT_INCR(esmp, stat[EFX_MAC_RX_256_TO_511_PKTS],
328             RX_256_TO_511_PKT);
329         GMAC_STAT_INCR(esmp, stat[EFX_MAC_RX_512_TO_1023_PKTS],
330             RX_512_TO_1023_PKT);
331         GMAC_STAT_INCR(esmp, stat[EFX_MAC_RX_1024_TO_15XX_PKTS],
332             RX_1024_TO_15XX_PKT);
333         GMAC_STAT_INCR(esmp, stat[EFX_MAC_RX_GE_15XX_PKTS],
334             RX_15XX_TO_JUMBO_PKT);
335         GMAC_STAT_INCR(esmp, stat[EFX_MAC_RX_GE_15XX_PKTS], RX_GT_JUMBO_PKT);
336
337         GMAC_STAT_INCR(esmp, stat[EFX_MAC_RX_ERRORS], RX_BAD_PKT);
338         GMAC_STAT_DECR(esmp, stat[EFX_MAC_RX_ERRORS], RX_PAUSE_PKT);
339         GMAC_STAT_INCR(esmp, stat[EFX_MAC_RX_FCS_ERRORS],
340             RX_FCS_ERR_64_TO_15XX_PKT);
341         GMAC_STAT_INCR(esmp, stat[EFX_MAC_RX_FCS_ERRORS],
342             RX_FCS_ERR_15XX_TO_JUMBO_PKT);
343         GMAC_STAT_INCR(esmp, stat[EFX_MAC_RX_FCS_ERRORS],
344             RX_FCS_ERR_GT_JUMBO_PKT);
345         GMAC_STAT_INCR(esmp, stat[EFX_MAC_RX_DROP_EVENTS], RX_MISS_PKT);
346         GMAC_STAT_INCR(esmp, stat[EFX_MAC_RX_FALSE_CARRIER_ERRORS],
347             RX_FALSE_CRS);
348
349         EFX_BAR_READO(enp, FR_AZ_RX_NODESC_DROP_REG, &oword);
350         delta = (uint32_t)EFX_OWORD_FIELD(oword, FRF_AZ_RX_NODESC_DROP_CNT);
351         EFSYS_STAT_INCR(&(stat[EFX_MAC_RX_NODESC_DROP_CNT]), delta);
352
353         /* TX */
354         GMAC_STAT_INCR(esmp, stat[EFX_MAC_TX_OCTETS], TX_GOOD_BAD_OCT);
355
356         GMAC_STAT_INCR(esmp, stat[EFX_MAC_TX_PKTS], TX_LT_64_PKT);
357         GMAC_STAT_INCR(esmp, stat[EFX_MAC_TX_PKTS], TX_64_PKT);
358         GMAC_STAT_INCR(esmp, stat[EFX_MAC_TX_PKTS], TX_65_TO_127_PKT);
359         GMAC_STAT_INCR(esmp, stat[EFX_MAC_TX_PKTS], TX_128_TO_255_PKT);
360         GMAC_STAT_INCR(esmp, stat[EFX_MAC_TX_PKTS], TX_256_TO_511_PKT);
361         GMAC_STAT_INCR(esmp, stat[EFX_MAC_TX_PKTS], TX_512_TO_1023_PKT);
362         GMAC_STAT_INCR(esmp, stat[EFX_MAC_TX_PKTS], TX_1024_TO_15XX_PKT);
363         GMAC_STAT_INCR(esmp, stat[EFX_MAC_TX_PKTS], TX_15XX_TO_JUMBO_PKT);
364         GMAC_STAT_INCR(esmp, stat[EFX_MAC_TX_PKTS], TX_GT_JUMBO_PKT);
365
366         GMAC_STAT_INCR(esmp, stat[EFX_MAC_TX_UNICST_PKTS], TX_UCAST_PKT);
367         GMAC_STAT_INCR(esmp, stat[EFX_MAC_TX_MULTICST_PKTS], TX_MCAST_PKT);
368         GMAC_STAT_INCR(esmp, stat[EFX_MAC_TX_BRDCST_PKTS], TX_BCAST_PKT);
369         GMAC_STAT_INCR(esmp, stat[EFX_MAC_TX_PAUSE_PKTS], TX_PAUSE_PKT);
370         GMAC_STAT_INCR(esmp, stat[EFX_MAC_TX_LE_64_PKTS], TX_LT_64_PKT);
371         GMAC_STAT_INCR(esmp, stat[EFX_MAC_TX_LE_64_PKTS], TX_64_PKT);
372
373         GMAC_STAT_INCR(esmp, stat[EFX_MAC_TX_65_TO_127_PKTS],
374             TX_65_TO_127_PKT);
375         GMAC_STAT_INCR(esmp, stat[EFX_MAC_TX_128_TO_255_PKTS],
376             TX_128_TO_255_PKT);
377         GMAC_STAT_INCR(esmp, stat[EFX_MAC_TX_256_TO_511_PKTS],
378             TX_256_TO_511_PKT);
379         GMAC_STAT_INCR(esmp, stat[EFX_MAC_TX_512_TO_1023_PKTS],
380             TX_512_TO_1023_PKT);
381         GMAC_STAT_INCR(esmp, stat[EFX_MAC_TX_1024_TO_15XX_PKTS],
382             TX_1024_TO_15XX_PKT);
383         GMAC_STAT_INCR(esmp, stat[EFX_MAC_TX_GE_15XX_PKTS],
384             TX_15XX_TO_JUMBO_PKT);
385         GMAC_STAT_INCR(esmp, stat[EFX_MAC_TX_GE_15XX_PKTS],
386             TX_GT_JUMBO_PKT);
387
388         GMAC_STAT_INCR(esmp, stat[EFX_MAC_TX_ERRORS], TX_BAD_PKT);
389         GMAC_STAT_DECR(esmp, stat[EFX_MAC_TX_ERRORS], TX_PAUSE_PKT);
390
391         GMAC_STAT_INCR(esmp, stat[EFX_MAC_TX_SGL_COL_PKTS], TX_SGL_COL_PKT);
```

```
392         GMAC_STAT_INCR(esmp, stat[EFX_MAC_TX_MULT_COL_PKTS], TX_MULT_COL_PKT);
393         GMAC_STAT_INCR(esmp, stat[EFX_MAC_TX_EX_COL_PKTS], TX_EX_COL_PKT);
394         GMAC_STAT_INCR(esmp, stat[EFX_MAC_TX_LATE_COL_PKTS], TX_LATE_COL);
395         GMAC_STAT_INCR(esmp, stat[EFX_MAC_TX_DEF_PKTS], TX_DEF_PKT);
396         GMAC_STAT_INCR(esmp, stat[EFX_MAC_TX_EX_DEF_PKTS], TX_EX_DEF_PKT);

398         return (0);
399 }

401 #endif  /* EFSYS_OPT_MAC_STATS */

403 #endif  /* EFSYS_OPT_MAC_FALCON_GMAC */
404 #endif /* ! codereview */
```

```
**********************************************************
    2143 Thu Aug 22 18:59:23 2013
new/usr/src/uts/common/io/sfxge/falcon_gmac.h
Merged sfxge driver
**********************************************************
```

```
/*-
 * Copyright 2008-2013 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#ifndef _SYS_FALCON_GMAC_H
#define _SYS_FALCON_GMAC_H

#include "efx.h"

#ifdef  __cplusplus
extern "C" {
#endif

#if EFSYS_OPT_MAC_FALCON_GMAC

#if EFSYS_OPT_LOOPBACK

#define FALCON_GMAC_LOOPBACK_MASK
        (1 << EFX_LOOPBACK_GMAC)

#endif  /* EFSYS_OPT_LOOPBACK */

#define GMAC_INTR_SUPPORTED     B_FALSE

extern  __checkReturn   int
falcon_gmac_reset(
        __in            efx_nic_t *enp);

extern  __checkReturn   int
falcon_gmac_reconfigure(
        __in            efx_nic_t *enp);

#if EFSYS_OPT_MAC_STATS

extern  __checkReturn                   int
falcon_gmac_stats_update(
        __in                            efx_nic_t *enp,
        __in                            efsys_mem_t *esmp,
        __inout_ecount(EFX_MAC_NSTATS)  efsys_stat_t *essp,
        __out_opt                       uint32_t *generationp);
```



```
#endif

#endif  /* EFSYS_OPT_MAC_GMAC */

#ifdef  __cplusplus
}
#endif

#endif  /* _SYS_FALCON_GMAC_H */
#endif /* ! codereview */
```

```
**********************************************************
   16852 Thu Aug 22 18:59:23 2013
new/usr/src/uts/common/io/sfxge/falcon_i2c.c
Merged sfxge driver
**********************************************************
```

```
/*
 * Copyright 2007-2013 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#include "efsys.h"
#include "efx.h"
#include "efx_types.h"
#include "efx_regs.h"
#include "efx_impl.h"

#if EFSYS_OPT_FALCON

#define I2C_READ_CMD(_devid)    (((_devid) << 1) | 1)
#define I2C_WRITE_CMD(_devid)   (((_devid) << 1) | 0)

#define I2C_PERIOD      100

#define I2C_RETRY_LIMIT 10

static                  void
falcon_i2c_set_sda_scl(
        __in            efx_nic_t *enp,
        __in            falcon_i2c_t *fip,
        __inout_opt     efsys_timestamp_t *timep)
{
        efx_oword_t oword;
        efsys_timestamp_t delta = 0;

        EFSYS_SPIN(I2C_PERIOD);
        delta += I2C_PERIOD;

        /* Set the pin state */
        EFX_BAR_READO(enp, FR_AB_GPIO_CTL_REG, &oword);

        EFX_SET_OWORD_FIELD(oword, FRF_AB_GPIO3_OUT, 0);
        EFX_SET_OWORD_FIELD(oword, FRF_AB_GPIO3_OEN,
                            fip->fi_sda ? 0 : 1);

        EFX_SET_OWORD_FIELD(oword, FRF_AB_GPIO0_OEN, 1);
        EFX_SET_OWORD_FIELD(oword, FRF_AB_GPIO0_OUT,
```



```
                            fip->fi_scl ? 1 : 0);

        EFX_BAR_WRITEO(enp, FR_AB_GPIO_CTL_REG, &oword);

        EFSYS_SPIN(I2C_PERIOD);
        delta += I2C_PERIOD;

        if (timep != NULL)
                *timep += delta;
}

static  __checkReturn   boolean_t
falcon_i2c_get_sda(
        __in            efx_nic_t *enp,
        __inout_opt     efsys_timestamp_t *timep)
{
        efx_oword_t oword;
        efsys_timestamp_t delta = 0;
        boolean_t state;

        EFSYS_SPIN(I2C_PERIOD);
        delta += I2C_PERIOD;

        /* Get the pin state */
        EFX_BAR_READO(enp, FR_AB_GPIO_CTL_REG, &oword);
        state = (EFX_OWORD_FIELD(oword, FRF_AB_GPIO3_IN) != 0);

        EFSYS_SPIN(I2C_PERIOD);
        delta += I2C_PERIOD;

        if (timep != NULL)
                *timep += delta;

        return (state);
}

static  __checkReturn   boolean_t
falcon_i2c_get_scl(
        __in            efx_nic_t *enp,
        __inout_opt     efsys_timestamp_t *timep)
{
        efx_oword_t oword;
        efsys_timestamp_t delta = 0;
        boolean_t state;

        EFSYS_SPIN(I2C_PERIOD);
        delta += I2C_PERIOD;

        /* Get the pin state */
        EFX_BAR_READO(enp, FR_AB_GPIO_CTL_REG, &oword);
        state = (EFX_OWORD_FIELD(oword, FRF_AB_GPIO0_IN) != 0);

        EFSYS_SPIN(I2C_PERIOD);
        delta += I2C_PERIOD;

        if (timep != NULL)
                *timep += delta;

        return (state);
}

static                  void
falcon_i2c_sda_write(
        __in            efx_nic_t *enp,
        __in            boolean_t on,
        __inout_opt     efsys_timestamp_t *timep)
```

```
128 {
129         falcon_i2c_t *fip = &(enp->en_u.falcon.enu_fip);
130
131         fip->fi_sda = on;
132         falcon_i2c_set_sda_scl(enp, fip, timep);
133 }
134
135 static                  void
136 falcon_i2c_scl_write(
137         __in            efx_nic_t *enp,
138         __in            boolean_t on,
139         __inout_opt     efsys_timestamp_t *timep)
140 {
141         falcon_i2c_t *fip = &(enp->en_u.falcon.enu_fip);
142
143         fip->fi_scl = on;
144         falcon_i2c_set_sda_scl(enp, fip, timep);
145 }
146
147 static                  void
148 falcon_i2c_start(
149         __in            efx_nic_t *enp,
150         __inout_opt     efsys_timestamp_t *timep)
151 {
152         falcon_i2c_t *fip = &(enp->en_u.falcon.enu_fip);
153
154         EFSYS_ASSERT(timep != NULL);
155
156         EFSYS_ASSERT(fip->fi_sda);
157
158         falcon_i2c_scl_write(enp, B_TRUE, timep);
159         falcon_i2c_sda_write(enp, B_FALSE, timep);
160         falcon_i2c_scl_write(enp, B_FALSE, timep);
161         falcon_i2c_sda_write(enp, B_TRUE, timep);
162 }
163
164 static                  void
165 falcon_i2c_bit_out(
166         __in            efx_nic_t *enp,
167         __in            boolean_t bit,
168         __inout_opt     efsys_timestamp_t *timep)
169 {
170         falcon_i2c_t *fip = &(enp->en_u.falcon.enu_fip);
171
172         EFSYS_ASSERT(timep != NULL);
173
174         EFSYS_ASSERT(!(fip->fi_scl));
175
176         falcon_i2c_sda_write(enp, bit, timep);
177         falcon_i2c_scl_write(enp, B_TRUE, timep);
178         falcon_i2c_scl_write(enp, B_FALSE, timep);
179         falcon_i2c_sda_write(enp, B_TRUE, timep);
180 }
181
182 static                  boolean_t
183 falcon_i2c_bit_in(
184         __in            efx_nic_t *enp,
185         __inout_opt     efsys_timestamp_t *timep)
186 {
187         falcon_i2c_t *fip = &(enp->en_u.falcon.enu_fip);
188         boolean_t bit;
189
190         EFSYS_ASSERT(timep != NULL);
191
192         EFSYS_ASSERT(!(fip->fi_scl));
193         EFSYS_ASSERT(fip->fi_sda);
```



```
194
195         falcon_i2c_scl_write(enp, B_TRUE, timep);
196         bit = falcon_i2c_get_sda(enp, timep);
197         falcon_i2c_scl_write(enp, B_FALSE, timep);
198
199         return (bit);
200 }
201
202 static                  void
203 falcon_i2c_stop(
204         __in            efx_nic_t *enp,
205         __inout_opt     efsys_timestamp_t *timep)
206 {
207         falcon_i2c_t *fip = &(enp->en_u.falcon.enu_fip);
208
209         EFSYS_ASSERT(timep != NULL);
210
211         EFSYS_ASSERT(!(fip->fi_scl));
212
213         falcon_i2c_sda_write(enp, B_FALSE, timep);
214         falcon_i2c_scl_write(enp, B_TRUE, timep);
215         falcon_i2c_sda_write(enp, B_TRUE, timep);
216 }
217
218 static                  void
219 falcon_i2c_release(
220         __in            efx_nic_t *enp,
221         __inout_opt     efsys_timestamp_t *timep)
222 {
223         falcon_i2c_t *fip = &(enp->en_u.falcon.enu_fip);
224
225         EFSYS_ASSERT(timep != NULL);
226
227         falcon_i2c_scl_write(enp, B_TRUE, timep);
228         falcon_i2c_sda_write(enp, B_TRUE, timep);
229
230         EFSYS_ASSERT(falcon_i2c_get_sda(enp, timep));
231         EFSYS_ASSERT(falcon_i2c_get_scl(enp, timep));
232
233         EFSYS_ASSERT(fip->fi_scl);
234         EFSYS_ASSERT(fip->fi_sda);
235
236         EFSYS_SPIN(10000);
237         *timep += 10000;
238 }
239
240 static  __checkReturn   int
241 falcon_i2c_wait(
242         __in            efx_nic_t *enp)
243 {
244         falcon_i2c_t *fip = &(enp->en_u.falcon.enu_fip);
245         int count;
246         int rc;
247
248         if (enp->en_u.falcon.enu_i2c_locked) {
249                 rc = EBUSY;
250                 goto fail1;
251         }
252
253         EFSYS_ASSERT(fip->fi_scl);
254         EFSYS_ASSERT(fip->fi_sda);
255
256         count = 0;
257         do {
258                 EFSYS_PROBE1(wait, unsigned int, count);
```

```
260                 if (falcon_i2c_get_scl(enp, NULL) &&
261                     falcon_i2c_get_sda(enp, NULL))
262                         goto done;
263
264                 /* Spin for 1 ms */
265                 EFSYS_SPIN(1000);
266
267         } while (++count < 20);
268
269         rc = ETIMEDOUT;
270         goto fail2;
271
272 done:
273         return (0);
274
275 fail2:
276         EFSYS_PROBE(fail2);
277 fail1:
278         EFSYS_PROBE1(fail1, int, rc);
279
280         return (rc);
281 }
282
283 static  __checkReturn   int
284 falcon_i2c_byte_out(
285         __in            efx_nic_t *enp,
286         __in            uint8_t byte,
287         __inout_opt     efsys_timestamp_t *timep)
288 {
289         unsigned int bit;
290
291         for (bit = 0; bit < 8; bit++) {
292                 falcon_i2c_bit_out(enp, ((byte & 0x80) != 0), timep);
293                 byte <<= 1;
294         }
295
296         /* Check for acknowledgement */
297         return ((falcon_i2c_bit_in(enp, timep)) ? EIO : 0);
298 }
299
300 static                  uint8_t
301 falcon_i2c_byte_in(
302         __in            efx_nic_t *enp,
303         __in            boolean_t ack,
304         __inout_opt     efsys_timestamp_t *timep)
305 {
306         uint8_t byte;
307         unsigned int bit;
308
309         byte = 0;
310         for (bit = 0; bit < 8; bit++) {
311                 byte <<= 1;
312                 byte |= falcon_i2c_bit_in(enp, timep);
313         }
314
315         /* Send acknowledgement */
316         falcon_i2c_bit_out(enp, !ack, timep);
317
318         return (byte);
319 }
320
321         __checkReturn   int
322 falcon_i2c_check(
323         __in            efx_nic_t *enp,
324         __in            uint8_t devid)
325 {
```



```
326         int pstate;
327         int lstate;
328         efsys_timestamp_t start;
329         efsys_timestamp_t end;
330         efsys_timestamp_t expected;
331         unsigned int attempt;
332         int rc;
333
334         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
335         EFSYS_ASSERT3U(enp->en_family, ==, EFX_FAMILY_FALCON);
336
337         EFSYS_LOCK(enp->en_eslp, lstate);
338         EFSYS_PREEMPT_DISABLE(pstate);
339
340         /* Check the state of the I2C bus */
341         if ((rc = falcon_i2c_wait(enp)) != 0)
342                 goto fail1;
343
344         attempt = 0;
345
346 again:
347         EFSYS_TIMESTAMP(&start);
348         expected = 0;
349
350         /* Select device and write cycle */
351         falcon_i2c_start(enp, &expected);
352
353         if ((rc = falcon_i2c_byte_out(enp, I2C_WRITE_CMD(devid),
354             &expected)) != 0)
355                 goto fail2;
356
357         /* Abort the cycle since we're only testing the target is there */
358         falcon_i2c_stop(enp, &expected);
359         falcon_i2c_release(enp, &expected);
360
361         EFSYS_TIMESTAMP(&end);
362
363         /* The transaction may have timed out */
364         if (end - start > expected * 2) {
365                 if (++attempt < I2C_RETRY_LIMIT) {
366                         EFSYS_PROBE1(retry, unsigned int, attempt);
367
368                         goto again;
369                 }
370
371                 rc = ETIMEDOUT;
372                 goto fail3;
373         }
374
375         EFSYS_PREEMPT_ENABLE(pstate);
376         EFSYS_UNLOCK(enp->en_eslp, lstate);
377
378         EFSYS_PROBE(success);
379         return (0);
380
381 fail3:
382         EFSYS_PROBE(fail3);
383
384         goto fail1;
385
386 fail2:
387         falcon_i2c_stop(enp, &expected);
388         falcon_i2c_release(enp, &expected);
389
390         EFSYS_TIMESTAMP(&end);
```

```
392         /* The transaction may have timed out */
393         if (end - start > expected * 2) {
394                 if (++attempt < I2C_RETRY_LIMIT) {
395                         EFSYS_PROBE1(retry, unsigned int, attempt);
396
397                         goto again;
398                 }
399
400                 rc = ETIMEDOUT;
401         }
402
403         EFSYS_PROBE(fail2);
404
405 fail1:
406         EFSYS_PROBE1(fail1, int, rc);
407
408         EFSYS_PREEMPT_ENABLE(pstate);
409         EFSYS_UNLOCK(enp->en_eslp, lstate);
410         return (rc);
411 }
412
413         __checkReturn           int
414 falcon_i2c_read(
415         __in                    efx_nic_t *enp,
416         __in                    uint8_t devid,
417         __in                    uint8_t addr,
418         __out_bcount(size)      caddr_t base,
419         __in                    size_t size)
420 {
421         int pstate;
422         int lstate;
423         efsys_timestamp_t start;
424         efsys_timestamp_t end;
425         efsys_timestamp_t expected;
426         unsigned int attempt;
427         unsigned int i;
428         int rc;
429
430         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
431         EFSYS_ASSERT3U(enp->en_family, ==, EFX_FAMILY_FALCON);
432
433         EFSYS_ASSERT(size != 0);
434
435         EFSYS_LOCK(enp->en_eslp, lstate);
436         EFSYS_PREEMPT_DISABLE(pstate);
437
438         /* Check the state of the I2C bus */
439         if ((rc = falcon_i2c_wait(enp)) != 0)
440                 goto fail1;
441
442         attempt = 0;
443
444 again:
445         EFSYS_TIMESTAMP(&start);
446         expected = 0;
447
448         /* Select device and write cycle */
449         falcon_i2c_start(enp, &expected);
450
451         if ((rc = falcon_i2c_byte_out(enp, I2C_WRITE_CMD(devid),
452             &expected)) != 0)
453                 goto fail2;
454
455         /* Write address */
456         if ((rc = falcon_i2c_byte_out(enp, addr, &expected)) != 0)
457                 goto fail3;
```



```
459         /* Select device and read cycle */
460         falcon_i2c_start(enp, &expected);
461
462         if ((rc = falcon_i2c_byte_out(enp, I2C_READ_CMD(devid),
463             &expected)) != 0)
464                 goto fail4;
465
466         /* Read bytes */
467         for (i = 0; i < size; i++)
468                 *(uint8_t *)(base + i) =
469                         falcon_i2c_byte_in(enp, (i < size - 1), &expected);
470
471         falcon_i2c_stop(enp, &expected);
472         falcon_i2c_release(enp, &expected);
473
474         EFSYS_TIMESTAMP(&end);
475
476         /* The transaction may have timed out */
477         if (end - start > expected * 2) {
478                 if (++attempt < I2C_RETRY_LIMIT) {
479                         EFSYS_PROBE1(retry, unsigned int, attempt);
480
481                         goto again;
482                 }
483
484                 rc = ETIMEDOUT;
485                 goto fail5;
486         }
487
488         EFSYS_PREEMPT_ENABLE(pstate);
489         EFSYS_UNLOCK(enp->en_eslp, lstate);
490         return (0);
491
492 fail5:
493         EFSYS_PROBE(fail5);
494
495         goto fail1;
496
497 fail4:
498         EFSYS_PROBE(fail4);
499 fail3:
500         EFSYS_PROBE(fail3);
501 fail2:
502         falcon_i2c_stop(enp, &expected);
503         falcon_i2c_release(enp, &expected);
504
505         EFSYS_TIMESTAMP(&end);
506
507         /* The transaction may have timed out */
508         if (end - start > expected * 2) {
509                 if (++attempt < I2C_RETRY_LIMIT) {
510                         EFSYS_PROBE1(retry, unsigned int, attempt);
511
512                         goto again;
513                 }
514
515                 rc = ETIMEDOUT;
516         }
517
518         EFSYS_PROBE(fail2);
519
520 fail1:
521         EFSYS_PROBE1(fail1, int, rc);
522
523         EFSYS_PREEMPT_ENABLE(pstate);
```

```
524         EFSYS_UNLOCK(enp->en_eslp, lstate);
525         return (rc);
526 }

528         __checkReturn           int
529 falcon_i2c_write(
530         __in                    efx_nic_t *enp,
531         __in                    uint8_t devid,
532         __in                    uint8_t addr,
533         __in_bcount(size)       caddr_t base,
534         __in                    size_t size)
535 {
536         int pstate;
537         int lstate;
538         efsys_timestamp_t start;
539         efsys_timestamp_t end;
540         efsys_timestamp_t expected;
541         unsigned int attempt;
542         unsigned int i;
543         int rc;

545         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
546         EFSYS_ASSERT3U(enp->en_family, ==, EFX_FAMILY_FALCON);

548         EFSYS_ASSERT(size != 0);

550         EFSYS_LOCK(enp->en_eslp, lstate);
551         EFSYS_PREEMPT_DISABLE(pstate);

553         /* Check the state of the I2C bus */
554         if ((rc = falcon_i2c_wait(enp)) != 0)
555                 goto fail1;

557         attempt = 0;

559 again:
560         EFSYS_TIMESTAMP(&start);
561         expected = 0;

563         /* Select device and write cycle */
564         falcon_i2c_start(enp, &expected);

566         if ((rc = falcon_i2c_byte_out(enp, I2C_WRITE_CMD(devid),
567             &expected)) != 0)
568                 goto fail2;

570         /* Write address */
571         if ((rc = falcon_i2c_byte_out(enp, addr, &expected)) != 0)
572                 goto fail3;

574         /* Write bytes */
575         for (i = 0; i < size; i++) {
576                 if ((rc = falcon_i2c_byte_out(enp, *(uint8_t *)(base + i),
577                     &expected)) != 0)
578                         goto fail4;
579         }

581         falcon_i2c_stop(enp, &expected);
582         falcon_i2c_release(enp, &expected);

584         EFSYS_TIMESTAMP(&end);

586         /* The transaction may have timed out */
587         if (end - start > expected * 2) {
588                 if (++attempt < I2C_RETRY_LIMIT) {
589                         EFSYS_PROBE1(retry, unsigned int, attempt);
```



```
591                         goto again;
592                 }

594                 rc = ETIMEDOUT;
595                 goto fail5;
596         }

598         EFSYS_PREEMPT_ENABLE(pstate);
599         EFSYS_UNLOCK(enp->en_eslp, lstate);
600         return (0);

602 fail5:
603         EFSYS_PROBE(fail5);

605         goto fail1;

607 fail4:
608         EFSYS_PROBE(fail4);
609 fail3:
610         EFSYS_PROBE(fail3);
611 fail2:
612         falcon_i2c_stop(enp, &expected);
613         falcon_i2c_release(enp, &expected);

615         EFSYS_TIMESTAMP(&end);

617         /* The transaction may have timed out */
618         if (end - start > expected * 2) {
619                 if (++attempt < I2C_RETRY_LIMIT) {
620                         EFSYS_PROBE1(retry, unsigned int, attempt);

622                         goto again;
623                 }

625                 rc = ETIMEDOUT;
626         }

628         EFSYS_PROBE(fail2);

630 fail1:
631         EFSYS_PROBE1(fail1, int, rc);

633         EFSYS_PREEMPT_ENABLE(pstate);
634         EFSYS_UNLOCK(enp->en_eslp, lstate);
635         return (rc);
636 }

638 #if EFSYS_OPT_PHY_NULL

640         __checkReturn           int
641 falcon_i2c_recv(
642         __in                    efx_nic_t *enp,
643         __in                    uint8_t devid,
644         __out_bcount(size)      caddr_t base,
645         __in                    size_t size)
646 {
647         int pstate;
648         int lstate;
649         efsys_timestamp_t start;
650         efsys_timestamp_t end;
651         efsys_timestamp_t expected;
652         unsigned int attempt;
653         unsigned int i;
654         int rc;
```

```
656         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
657         EFSYS_ASSERT3U(enp->en_family, ==, EFX_FAMILY_FALCON);

659         EFSYS_ASSERT(size != 0);

661         EFSYS_LOCK(enp->en_eslp, lstate);
662         EFSYS_PREEMPT_DISABLE(pstate);

664         /* Check the state of the I2C bus */
665         if ((rc = falcon_i2c_wait(enp)) != 0)
666                 goto fail1;

668         attempt = 0;

670 again:
671         EFSYS_TIMESTAMP(&start);
672         expected = 0;

674         /* Select device and read cycle */
675         falcon_i2c_start(enp, &expected);

677         if ((rc = falcon_i2c_byte_out(enp, I2C_READ_CMD(devid), &expected))
678             != 0)
679                 goto fail2;

681         /* Read bytes */
682         for (i = 0; i < size; i++)
683                 *(uint8_t *)(base + i) = falcon_i2c_byte_in(enp, (i < size - 1),
684                     &expected);

686         falcon_i2c_stop(enp, &expected);
687         falcon_i2c_release(enp, &expected);

689         EFSYS_TIMESTAMP(&end);

691         /* The transaction may have timed out */
692         if (end - start > expected * 2) {
693                 if (++attempt < I2C_RETRY_LIMIT) {
694                         EFSYS_PROBE1(retry, unsigned int, attempt);

696                         goto again;
697                 }

699                 rc = ETIMEDOUT;
700                 goto fail3;
701         }

703         EFSYS_PREEMPT_ENABLE(pstate);
704         EFSYS_UNLOCK(enp->en_eslp, lstate);
705         return (0);

707 fail3:
708         EFSYS_PROBE(fail5);

710         goto fail1;

712 fail2:
713         falcon_i2c_stop(enp, &expected);
714         falcon_i2c_release(enp, &expected);

716         EFSYS_TIMESTAMP(&end);

718         /* The transaction may have timed out */
719         if (end - start > expected * 2) {
720                 if (++attempt < I2C_RETRY_LIMIT) {
721                         EFSYS_PROBE1(retry, unsigned int, attempt);
```



```
723                         goto again;
724                 }

726                 rc = ETIMEDOUT;
727         }

729         EFSYS_PROBE(fail2);

731 fail1:
732         EFSYS_PROBE1(fail1, int, rc);

734         EFSYS_PREEMPT_ENABLE(pstate);
735         EFSYS_UNLOCK(enp->en_eslp, lstate);
736         return (rc);
737 }

739         __checkReturn           int
740 falcon_i2c_send(
741         __in                    efx_nic_t *enp,
742         __in                    uint8_t devid,
743         __in_bcount(size)       caddr_t base,
744         __in                    size_t size)
745 {
746         int pstate;
747         int lstate;
748         efsys_timestamp_t start;
749         efsys_timestamp_t end;
750         efsys_timestamp_t expected;
751         unsigned int attempt;
752         unsigned int i;
753         int rc;

755         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
756         EFSYS_ASSERT3U(enp->en_family, ==, EFX_FAMILY_FALCON);

758         EFSYS_ASSERT(size != 0);

760         EFSYS_LOCK(enp->en_eslp, lstate);
761         EFSYS_PREEMPT_DISABLE(pstate);

763         /* Check the state of the I2C bus */
764         if ((rc = falcon_i2c_wait(enp)) != 0)
765                 goto fail1;

767         attempt = 0;

769 again:
770         EFSYS_TIMESTAMP(&start);
771         expected = 0;

773         /* Select device and write cycle */
774         falcon_i2c_start(enp, &expected);

776         if ((rc = falcon_i2c_byte_out(enp, I2C_WRITE_CMD(devid), &expected))
777             != 0)
778                 goto fail2;

780         /* Write bytes */
781         for (i = 0; i < size; i++)
782                 if ((rc = falcon_i2c_byte_out(enp,      *(uint8_t *)(base + i),
783                     &expected)) != 0)
784                         goto fail3;

786         falcon_i2c_stop(enp, &expected);
787         falcon_i2c_release(enp, &expected);
```

```
789         EFSYS_TIMESTAMP(&end);

791         /* The transaction may have timed out */
792         if (end - start > expected * 2) {
793                 if (++attempt < I2C_RETRY_LIMIT) {
794                         EFSYS_PROBE1(retry, unsigned int, attempt);

796                         goto again;
797                 }

799                 rc = ETIMEDOUT;
800                 goto fail4;
801         }

803         EFSYS_PREEMPT_ENABLE(pstate);
804         EFSYS_UNLOCK(enp->en_eslp, lstate);
805         return (0);

807 fail4:
808         EFSYS_PROBE(fail5);

810         goto fail1;

812 fail3:
813         EFSYS_PROBE(fail3);
814 fail2:
815         falcon_i2c_stop(enp, &expected);
816         falcon_i2c_release(enp, &expected);

818         EFSYS_TIMESTAMP(&end);

820         /* The transaction may have timed out */
821         if (end - start > expected * 2) {
822                 if (++attempt < I2C_RETRY_LIMIT) {
823                         EFSYS_PROBE1(retry, unsigned int, attempt);

825                         goto again;
826                 }

828                 rc = ETIMEDOUT;
829         }

831         EFSYS_PROBE(fail2);

833 fail1:
834         EFSYS_PROBE1(fail1, int, rc);

836         EFSYS_PREEMPT_ENABLE(pstate);
837         EFSYS_UNLOCK(enp->en_eslp, lstate);
838         return (rc);
839 }
840 #endif  /* EFSYS_OPT_PHY_NULL */

842 #endif  /* EFSYS_OPT_FALCON */
843 #endif /* ! codereview */
```

```
**********************************************************
    8258 Thu Aug 22 18:59:23 2013
new/usr/src/uts/common/io/sfxge/falcon_impl.h
Merged sfxge driver
**********************************************************
```

```
/*-
 * Copyright 2007-2013 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#ifndef _SYS_FALCON_IMPL_H
#define _SYS_FALCON_IMPL_H

#include "efx.h"
#include "efx_regs.h"

#ifdef  __cplusplus
extern "C" {
#endif

typedef struct falcon_i2c_s {
        boolean_t       fi_sda;
        boolean_t       fi_scl;
} falcon_i2c_t;

typedef struct falcon_spi_dev_s {
        uint32_t        fsd_sf_sel;
        size_t          fsd_size;
        uint32_t        fsd_adbcnt;
        boolean_t       fsd_munge;
        uint32_t        fsd_erase_cmd;
        size_t          fsd_erase_size;
        size_t          fsd_write_size;
} falcon_spi_dev_t;

extern  __checkReturn   int
falcon_nic_probe(
        __in            efx_nic_t *enp);

#if EFSYS_OPT_PCIE_TUNE

extern                  int
falcon_nic_pcie_tune(
        __in            efx_nic_t *enp,
        __in            unsigned int nlanes);
```



```
#endif

#define FALCON_NIC_CFG_RAW_SZ 0x400

extern  __checkReturn   int
falcon_nic_cfg_raw_read_verify(
        __in            efx_nic_t *enp,
        __in            uint32_t offset,
        __in            uint32_t size,
        __out           uint8_t *cfg);

extern  __checkReturn   int
falcon_nic_cfg_build(
        __in            efx_nic_t *enp,
        __out           efx_nic_cfg_t *encp);

extern  __checkReturn   int
falcon_nic_reset(
        __in            efx_nic_t *enp);

extern  __checkReturn   int
falcon_nic_init(
        __in            efx_nic_t *enp);

#if EFSYS_OPT_DIAG

extern  __checkReturn   int
falcon_nic_register_test(
        __in            efx_nic_t *enp);

#endif  /* EFSYS_OPT_DIAG */

extern                  void
falcon_nic_fini(
        __in            efx_nic_t *enp);

extern                  void
falcon_nic_unprobe(
        __in            efx_nic_t *enp);

extern  __checkReturn   int
falcon_nic_mac_reset(
        __in            efx_nic_t *enp);

extern                  void
falcon_mac_wrapper_enable(
        __in            efx_nic_t *enp);

extern  __checkReturn   int
falcon_mac_wrapper_disable(
        __in            efx_nic_t *enp);

#if EFSYS_OPT_LOOPBACK

extern  __checkReturn   int
falcon_mac_loopback_set(
        __in            efx_nic_t *enp,
        __in            efx_link_mode_t link_mode,
        __in            efx_loopback_type_t loopback_type);

#endif  /* EFSYS_OPT_LOOPBACK */

extern                  void
falcon_nic_phy_reset(
        __in            efx_nic_t *enp);
```

```
128 extern  __checkReturn           int
129 falcon_nvram_init(
130         __in                    efx_nic_t *enp);

132 #if EFSYS_OPT_NVRAM

134 #if EFSYS_OPT_DIAG

136 extern  __checkReturn           int
137 falcon_nvram_test(
138         __in                    efx_nic_t *enp);

140 #endif  /* EFSYS_OPT_DIAG */

142 extern  __checkReturn           int
143 falcon_nvram_size(
144         __in                    efx_nic_t *enp,
145         __in                    efx_nvram_type_t type,
146         __out                   size_t *sizep);

148 extern  __checkReturn           int
149 falcon_nvram_get_version(
150         __in                    efx_nic_t *enp,
151         __in                    efx_nvram_type_t type,
152         __out                   uint32_t *subtypep,
153         __out_ecount(4)         uint16_t version[4]);

155 extern  __checkReturn           int
156 falcon_nvram_rw_start(
157         __in                    efx_nic_t *enp,
158         __in                    efx_nvram_type_t type,
159         __out                   size_t *pref_chunkp);

161 extern  __checkReturn           int
162 falcon_nvram_read_chunk(
163         __in                    efx_nic_t *enp,
164         __in                    efx_nvram_type_t type,
165         __in                    unsigned int offset,
166         __out_bcount(size)      caddr_t data,
167         __in                    size_t size);

169 extern   __checkReturn          int
170 falcon_nvram_erase(
171         __in                    efx_nic_t *enp,
172         __in                    efx_nvram_type_t type);

174 extern  __checkReturn           int
175 falcon_nvram_write_chunk(
176         __in                    efx_nic_t *enp,
177         __in                    efx_nvram_type_t type,
178         __in                    unsigned int offset,
179         __in_bcount(size)       caddr_t data,
180         __in                    size_t size);

182 extern                          void
183 falcon_nvram_rw_finish(
184         __in                    efx_nic_t *enp,
185         __in                    efx_nvram_type_t type);

187 extern  __checkReturn           int
188 falcon_nvram_set_version(
189         __in                    efx_nic_t *enp,
190         __in                    efx_nvram_type_t type,
191         __out                   uint16_t version[4]);

193 #endif  /* EFSYS_OPT_NVRAM */
```



```
195 extern                          void
196 falcon_nvram_fini(
197         __in                    efx_nic_t *enp);

199 #if EFSYS_OPT_VPD

201 extern  __checkReturn           int
202 falcon_vpd_size(
203         __in                    efx_nic_t *enp,
204         __out                   size_t *sizep);

206 extern  __checkReturn           int
207 falcon_vpd_read(
208         __in                    efx_nic_t *enp,
209         __out_bcount(size)      caddr_t data,
210         __in                    size_t size);

212 extern  __checkReturn           int
213 falcon_vpd_verify(
214         __in                    efx_nic_t *enp,
215         __in_bcount(size)       caddr_t data,
216         __in                    size_t size);

218 extern  __checkReturn           int
219 falcon_vpd_get(
220         __in                    efx_nic_t *enp,
221         __in_bcount(size)       caddr_t data,
222         __in                    size_t size,
223         __inout                 efx_vpd_value_t *evvp);

225 extern  __checkReturn           int
226 falcon_vpd_set(
227         __in                    efx_nic_t *enp,
228         __in_bcount(size)       caddr_t data,
229         __in                    size_t size,
230         __in                    efx_vpd_value_t *evvp);

232 extern  __checkReturn           int
233 falcon_vpd_next(
234         __in                    efx_nic_t *enp,
235         __in_bcount(size)       caddr_t data,
236         __in                    size_t size,
237         __out                   efx_vpd_value_t *evvp,
238         __inout                 unsigned int *contp);

240 extern __checkReturn            int
241 falcon_vpd_write(
242         __in                    efx_nic_t *enp,
243         __in_bcount(size)       caddr_t data,
244         __in                    size_t size);

246 #endif  /* EFSYS_OPT_VPD */

248 extern  __checkReturn   int
249 falcon_sram_init(
250         __in            efx_nic_t *enp);

252 #if EFSYS_OPT_DIAG

254 extern  __checkReturn   int
255 falcon_sram_test(
256         __in            efx_nic_t *enp,
257         __in            efx_sram_pattern_fn_t func);

259 #endif  /* EFSYS_OPT_DIAG */
```

```
261 extern          void
262 falcon_sram_fini(
263         __in    efx_nic_t *enp);

265 extern  __checkReturn   int
266 falcon_i2c_check(
267         __in            efx_nic_t *enp,
268         __in            uint8_t devid);

270 extern  __checkReturn           int
271 falcon_i2c_read(
272         __in                    efx_nic_t *enp,
273         __in                    uint8_t devid,
274         __in                    uint8_t addr,
275         __out_bcount(size)      caddr_t base,
276         __in                    size_t size);

278 extern  __checkReturn           int
279 falcon_i2c_write(
280         __in                    efx_nic_t *enp,
281         __in                    uint8_t devid,
282         __in                    uint8_t addr,
283         __in_bcount(size)       caddr_t base,
284         __in                    size_t size);

286 #if EFSYS_OPT_PHY_NULL

288 extern  __checkReturn           int
289 falcon_i2c_recv(
290         __in                    efx_nic_t *enp,
291         __in                    uint8_t devid,
292         __out_bcount(size)      caddr_t base,
293         __in                    size_t size);

295 extern  __checkReturn           int
296 falcon_i2c_send(
297         __in                    efx_nic_t *enp,
298         __in                    uint8_t devid,
299         __in_bcount(size)       caddr_t base,
300         __in                    size_t size);

302 #endif  /* EFSYS_OPT_PHY_NULL */

304 extern  __checkReturn   int
305 falcon_mdio_write(
306         __in            efx_nic_t *enp,
307         __in            uint8_t port,
308         __in            uint8_t mmd,
309         __in            uint16_t reg,
310         __in            efx_word_t *ewp);

312 extern  __checkReturn   int
313 falcon_mdio_read(
314         __in            efx_nic_t *enp,
315         __in            uint8_t port,
316         __in            uint8_t mmd,
317         __in            uint16_t reg,
318         __out           efx_word_t *ewp);

320 #if EFSYS_OPT_MAC_STATS

322 extern  __checkReturn                   int
323 falcon_mac_stats_upload(
324         __in                            efx_nic_t *enp,
325         __in                            efsys_mem_t *esmp);
```



```
327 #endif  /* EFSYS_OPT_MAC_STATS */

329 extern  __checkReturn   int
330 falcon_mac_poll(
331         __in            efx_nic_t *enp,
332         __out           efx_link_mode_t *link_modep);

334 extern  __checkReturn   int
335 falcon_mac_up(
336         __in            efx_nic_t *enp,
337         __out           boolean_t *mac_upp);

339 #if EFSYS_OPT_LOOPBACK

341 extern  __checkReturn   int
342 falcon_mac_loopback_set(
343         __in            efx_nic_t *enp,
344         __in            efx_link_mode_t link_mode,
345         __in            efx_loopback_type_t loopback_type);

347 #endif  /* EFSYS_OPT_LOOPBACK */

349 typedef enum falcon_spi_type_e {
350         FALCON_SPI_FLASH = 0,
351         FALCON_SPI_EEPROM,
352         FALCON_SPI_NTYPES
353 } falcon_spi_type_t;

355 extern  __checkReturn           int
356 falcon_spi_dev_read(
357         __in                    efx_nic_t *enp,
358         __in                    falcon_spi_type_t type,
359         __in                    uint32_t addr,
360         __out_bcount(size)      caddr_t base,
361         __in                    size_t size);

363 extern  __checkReturn           int
364 falcon_spi_dev_write(
365         __in                    efx_nic_t *enp,
366         __in                    falcon_spi_type_t type,
367         __in                    uint32_t addr,
368         __in_bcount(size)       caddr_t base,
369         __in                    size_t size);

371 extern  __checkReturn           int
372 falcon_spi_dev_erase(
373         __in                    efx_nic_t *enp,
374         __in                    falcon_spi_type_t type,
375         __in                    uint32_t addr,
376         __in                    size_t size);

378 #ifdef  __cplusplus
379 }
380 #endif

382 #endif  /* _SYS_FALCON_IMPL_H */
383 #endif /* ! codereview */
```

```
**********************************************************
   10032 Thu Aug 22 18:59:24 2013
new/usr/src/uts/common/io/sfxge/falcon_mac.c
Merged sfxge driver
**********************************************************
```

```
/*
 * Copyright 2007-2013 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#include "efsys.h"
#include "efx.h"
#include "efx_types.h"
#include "efx_regs.h"
#include "efx_impl.h"

#if EFSYS_OPT_FALCON

#if EFSYS_OPT_MAC_FALCON_GMAC
#include "falcon_gmac.h"
#endif

#if EFSYS_OPT_MAC_FALCON_XMAC
#include "falcon_xmac.h"
#endif

        __checkReturn   int
falcon_mac_poll(
        __in            efx_nic_t *enp,
        __out           efx_link_mode_t *link_modep)
{
        efx_port_t *epp = &(enp->en_port);
        efx_phy_ops_t *epop = epp->ep_epop;
        efx_link_mode_t link_mode;
        unsigned int fcntl;
        uint32_t lp_cap_mask;
        boolean_t mac_up = B_TRUE;
        boolean_t reconfigure_mac = B_FALSE;
        int rc;

        /* Poll PHY for link state changes */
        rc = epop->epo_downlink_check(enp, &link_mode, &fcntl, &lp_cap_mask);
        if (rc != 0) {
                fcntl = epp->ep_fcntl;
                lp_cap_mask = epp->ep_lp_cap_mask;
                link_mode = EFX_LINK_UNKNOWN;
```



```
        }

#if EFSYS_OPT_LOOPBACK
        /* Ignore the phy link state in MAC level loopback */
        switch (epp->ep_loopback_type) {
        case EFX_LOOPBACK_GMAC:
                link_mode = EFX_LINK_1000FDX;
                break;

        case EFX_LOOPBACK_XGMII:
        case EFX_LOOPBACK_XGXS:
        case EFX_LOOPBACK_XAUI:
                link_mode = EFX_LINK_10000FDX;
                break;

        default:
                break;
        }

        if (epp->ep_loopback_type != EFX_LOOPBACK_OFF) {
                /*
                 * We've already configured the correct MAC for the correct
                 * speed, so all we need to do is wait for the phy loopback
                 * to come up. Keep ep_link_mode et al. at the values forced
                 * by falcon_mac_loopback_set(), but return the current value
                 * of link up vs link down.
                 */
                goto poll_mac;
        }
#endif

        /* Hook in the correct MAC and reconfigure for the new link speed. */
        epp->ep_lp_cap_mask = lp_cap_mask;

        if (link_mode != epp->ep_link_mode) {
                epp->ep_link_mode = link_mode;

                if ((rc = efx_mac_select(enp)) != 0)
                        goto fail1;

                reconfigure_mac = B_TRUE;
        }

        if (fcntl != epp->ep_fcntl) {
                epp->ep_fcntl = fcntl;
                reconfigure_mac = B_TRUE;
        }

        if (reconfigure_mac)
                epp->ep_emop->emo_reconfigure(enp);

#if EFSYS_OPT_LOOPBACK
poll_mac:
#endif
        /* The XMAC requires additional polling */
        if (epp->ep_mac_type == EFX_MAC_FALCON_XMAC)
                falcon_xmac_poll(enp, &mac_up);
        else
                mac_up = B_TRUE;

        epp->ep_mac_up = mac_up;
        *link_modep = link_mode;

        return (0);

fail1:
```

```
128         EFSYS_PROBE1(fail1, int, rc);

130         return (rc);
131 }

133         __checkReturn   int
134 falcon_mac_up(
135         __in            efx_nic_t *enp,
136         __out           boolean_t *mac_upp)
137 {
138         efx_port_t *epp = &(enp->en_port);

140         /* falcon_mac_poll() *must* be run on Falcon */
141         *mac_upp = epp->ep_mac_up;

143         return (0);
144 }

146                         void
147 falcon_mac_wrapper_enable(
148         __in            efx_nic_t *enp)
149 {
150         efx_port_t *epp = &(enp->en_port);
151         efx_oword_t oword;
152         uint32_t speed;
153         boolean_t drain = epp->ep_mac_drain;

155         switch (epp->ep_link_mode) {
156         case EFX_LINK_100FDX:
157         case EFX_LINK_100HDX:
158                 speed = FRF_AB_MAC_SPEED_100M;
159                 break;

161         case EFX_LINK_1000FDX:
162         case EFX_LINK_1000HDX:
163                 speed = FRF_AB_MAC_SPEED_1G;
164                 break;

166         case EFX_LINK_10000FDX:
167                 speed = FRF_AB_MAC_SPEED_10G;
168                 break;

170         default:
171 #if EFSYS_OPT_LOOPBACK
172                 EFSYS_ASSERT3U(epp->ep_loopback_type, ==, EFX_LOOPBACK_OFF);
173 #endif  /* EFSYS_OPT_LOOPBACK */
174                 drain = B_TRUE;
175                 speed = FRF_AB_MAC_SPEED_10M;
176         };

178         /* Bring the mac wrapper out of reset and configure it */
179         switch (epp->ep_mac_type) {
180         case EFX_MAC_FALCON_GMAC:
181                 EFX_POPULATE_OWORD_6(oword,
182                     FRF_AB_MAC_XOFF_VAL, 0xffff,
183                     FRF_AB_MAC_XG_DISTXCRC, 0,
184                     FRF_AB_MAC_BCAD_ACPT, (epp->ep_brdcst) ? 1 : 0,
185                     FRF_AB_MAC_UC_PROM, (epp->ep_unicst) ? 1 : 0,
186                     FRF_AB_MAC_LINK_STATUS, 1,
187                     FRF_AB_MAC_SPEED, speed);
188                 break;

190         case EFX_MAC_FALCON_XMAC:
191                 EFX_POPULATE_OWORD_6(oword,
192                     FRF_AB_MAC_XOFF_VAL, 0xffff,
193                     FRF_AB_MAC_XG_DISTXCRC, 0,
```



```
194                     FRF_AB_MAC_BCAD_ACPT, 1,
195                     FRF_AB_MAC_UC_PROM, 0,
196                     FRF_AB_MAC_LINK_STATUS, 1,
197                     FRF_AB_MAC_SPEED, speed);
198                 break;

200         default:
201                 EFSYS_ASSERT(B_FALSE);
202                 break;
203         }

205         /* Open TX_DRAIN if the link is up */
206         if (enp->en_family == EFX_FAMILY_FALCON)
207                 EFX_SET_OWORD_FIELD(oword, FRF_BB_TXFIFO_DRAIN_EN,
208                                     drain ? 1 : 0);
209         EFX_BAR_WRITEO(enp, FR_AB_MAC_CTRL_REG, &oword);

211         /* Push multicast hash. Set the broadcast bit (0xff) appropriately */
212         EFX_BAR_WRITEO(enp, FR_AB_MAC_MC_HASH0_REG,
213             &(epp->ep_multicst_hash[0]));
214         memcpy(&oword, &(epp->ep_multicst_hash[1]), sizeof (oword));
215         if (epp->ep_brdcst)
216                 EFX_SET_OWORD_BIT(oword, 0x7f);
217         EFX_BAR_WRITEO(enp, FR_AB_MAC_MC_HASH1_REG, &oword);

219         /* Configure RX <-> mac_wrapper link */
220         EFX_BAR_READO(enp, FR_AZ_RX_CFG_REG, &oword);
221         EFX_SET_OWORD_FIELD(oword, FRF_BZ_RX_XON_TX_TH, 25);
222         EFX_SET_OWORD_FIELD(oword, FRF_BZ_RX_XOFF_TX_TH, 20);
223         /* Send XON and XOFF at ~3 * max MTU away from empty/full */
224         EFX_SET_OWORD_FIELD(oword, FRF_BZ_RX_XON_MAC_TH, 27648 >> 8);
225         EFX_SET_OWORD_FIELD(oword, FRF_BZ_RX_XOFF_MAC_TH, 54272 >> 8);
226         EFX_SET_OWORD_FIELD(oword, FRF_AZ_RX_XOFF_MAC_EN,
227             (epp->ep_fcntl & EFX_FCNTL_GENERATE) ? 1 : 0);

229         if (enp->en_family == EFX_FAMILY_FALCON)
230                 EFX_SET_OWORD_FIELD(oword, FRF_BZ_RX_INGR_EN, drain ? 0 : 1);

232         EFX_BAR_WRITEO(enp, FR_AZ_RX_CFG_REG, &oword);
233 }

235         __checkReturn   int
236 falcon_mac_wrapper_disable(
237         __in            efx_nic_t *enp)
238 {
239         efx_oword_t oword;
240         int rc;

242         EFX_BAR_READO(enp, FR_AZ_RX_CFG_REG, &oword);
243         EFX_SET_OWORD_FIELD(oword, FRF_BZ_RX_INGR_EN, 0);
244         EFX_BAR_WRITEO(enp, FR_AZ_RX_CFG_REG, &oword);

246         /* There is no point in draining more than once */
247         EFX_BAR_READO(enp, FR_AB_MAC_CTRL_REG, &oword);
248         if (EFX_OWORD_FIELD(oword, FRF_BB_TXFIFO_DRAIN_EN))
249                 return (0);

251         if ((rc = falcon_nic_mac_reset(enp)) != 0)
252                 goto fail1;

254         return (0);

256 fail1:
257         EFSYS_PROBE1(fail1, int, rc);

259         return (rc);
```

```
}

#if EFSYS_OPT_LOOPBACK

        __checkReturn   int
falcon_mac_loopback_set(
        __in            efx_nic_t *enp,
        __in            efx_link_mode_t link_mode,
        __in            efx_loopback_type_t loopback_type)
{
        efx_port_t *epp = &(enp->en_port);
        efx_phy_ops_t *epop = epp->ep_epop;
        const efx_nic_cfg_t *encp = &(enp->en_nic_cfg);
        efx_loopback_type_t old_loopback_type;
        efx_link_mode_t old_loopback_link_mode;
        uint32_t phy_loopback_mask;
        boolean_t phy_loopback_changed;
        int rc;

        EFSYS_ASSERT3U(enp->en_family, ==, EFX_FAMILY_FALCON);

        old_loopback_type = epp->ep_loopback_type;
        old_loopback_link_mode = epp->ep_loopback_link_mode;

        phy_loopback_mask = EFX_LOOPBACK_MASK &
                ~(EFX_LOOPBACK_MAC_MASK | (1 << EFX_LOOPBACK_OFF));
        phy_loopback_changed = (phy_loopback_mask &
                ((1 << loopback_type) ^ (1 << epp->ep_loopback_type))) != 0;
        epp->ep_loopback_type = loopback_type;
        epp->ep_loopback_link_mode = link_mode;

        if (loopback_type == EFX_LOOPBACK_OFF)
                link_mode = EFX_LINK_DOWN;
        else if (link_mode == EFX_LINK_UNKNOWN) {
                for (link_mode = EFX_LINK_NMODES - 1;
                    link_mode > EFX_LINK_UNKNOWN; --link_mode) {
                if ((1 << loopback_type) &
                    encp->enc_loopback_types[link_mode])
                        break;
                }
        }

        EFSYS_ASSERT(((1 << loopback_type) & ~FALCON_GMAC_LOOPBACK_MASK) ||
                        link_mode == EFX_LINK_1000FDX);
        EFSYS_ASSERT(((1 << loopback_type) & ~FALCON_XMAC_LOOPBACK_MASK) ||
                        link_mode == EFX_LINK_10000FDX);

        /*
         * In loopback, a PHY typically requires the correct MAC and link
         * to be initialized before they will report link up. Determine
         * the expected link speed and select the correct MAC. Ensure that
         * ep_link_mode != LINK_DOWN in loopback so that TXDRAIN isn't enabled,
         * because we'll never again run efx_mac_select() to subsequently
         * reset the EM block.
         */
        epp->ep_link_mode = link_mode;
        if ((rc = efx_mac_select(enp)) != 0)
                goto fail1;

        /*
         * Don't reset and reconfigure the PHY unless it's
         * configuration has actually changed
         */
        if (phy_loopback_changed) {
                if ((rc = epop->epo_reset(enp)) != 0)
                        goto fail2;
```



```
                EFSYS_ASSERT(enp->en_reset_flags & EFX_RESET_PHY);
                enp->en_reset_flags &= ~EFX_RESET_PHY;

                if ((rc = epop->epo_reconfigure(enp)) != 0)
                        goto fail3;
        }

        epp->ep_emop->emo_reconfigure(enp);

        /* Ensure that the MAC is subsequently polled */
        epp->ep_mac_poll_needed = B_TRUE;
        epp->ep_mac_up = B_FALSE;

        return (0);

fail3:
        EFSYS_PROBE(fail3);
fail2:
        EFSYS_PROBE(fail2);
fail1:
        EFSYS_PROBE1(fail1, int, rc);

        epp->ep_loopback_type = old_loopback_type;
        epp->ep_loopback_link_mode = old_loopback_link_mode;
        epp->ep_link_mode = EFX_LINK_DOWN;

        return (rc);
}

#endif  /* EFSYS_OPT_LOOPBACK */

#if EFSYS_OPT_MAC_STATS

        __checkReturn                   int
falcon_mac_stats_upload(
        __in                            efx_nic_t *enp,
        __in                            efsys_mem_t *esmp)
{
        efx_oword_t oword;

        EFSYS_ASSERT3U(enp->en_family, ==, EFX_FAMILY_FALCON);

        EFX_POPULATE_OWORD_3(oword,
            FRF_AB_MAC_STAT_DMA_ADR_DW0, EFSYS_MEM_ADDR(esmp) & 0xffffffff,
            FRF_AB_MAC_STAT_DMA_ADR_DW1, EFSYS_MEM_ADDR(esmp) >> 32,
            FRF_AB_MAC_STAT_DMA_CMD, 1);

        EFX_BAR_WRITEO(enp, FR_AB_MAC_STAT_DMA_REG, &oword);

        return (0);
}

#endif  /* EFSYS_OPT_MAC_STATS */

#endif  /* EFSYS_OPT_FALCON */
#endif /* ! codereview */
```

```
**********************************************************
    5073 Thu Aug 22 18:59:24 2013
new/usr/src/uts/common/io/sfxge/falcon_mdio.c
Merged sfxge driver
**********************************************************
```

```
/*
 * Copyright 2007-2013 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#include "efsys.h"
#include "efx.h"
#include "efx_types.h"
#include "efx_regs.h"
#include "efx_impl.h"

#if EFSYS_OPT_FALCON

static  __checkReturn   int
falcon_mdio_wait(
        __in            efx_nic_t *enp)
{
        efx_oword_t oword;
        unsigned int count;
        int rc;

        count = 0;
        do {
                EFSYS_PROBE1(wait, unsigned int, count);

                EFX_BAR_READO(enp, FR_AB_MD_STAT_REG, &oword);

                if (EFX_OWORD_FIELD(oword, FRF_AB_MD_BSY) == 0)
                        goto done;

                /* Spin for 100 us */
                EFSYS_SPIN(100);

        } while (++count < 100);

        rc = ETIMEDOUT;
        goto fail1;

done:
        if (EFX_OWORD_FIELD(oword, FRF_AB_MD_LNFL) != 0) {
                rc = EIO;
```



```
                goto fail2;
        }

        if (EFX_OWORD_FIELD(oword, FRF_AB_MD_BSERR) != 0) {
                rc = EIO;
                goto fail3;
        }

        return (0);

fail3:
        EFSYS_PROBE(fail3);
fail2:
        EFSYS_PROBE(fail2);
fail1:
        EFSYS_PROBE1(fail1, int, rc);

        return (rc);
}

        __checkReturn   int
falcon_mdio_write(
        __in            efx_nic_t *enp,
        __in            uint8_t port,
        __in            uint8_t mmd,
        __in            uint16_t reg,
        __in            efx_word_t *ewp)
{
        int state;
        efx_oword_t oword;
        uint32_t prtad;
        uint32_t devad;
        uint32_t addr;
        uint16_t val;
        int rc;

        EFSYS_LOCK(enp->en_eslp, state);

        /* Check MDIO is not currently being accessed */
        if ((rc = falcon_mdio_wait(enp)) != 0)
                goto fail1;

        prtad = port;
        devad = mmd;

        /* Write address */
        EFX_POPULATE_OWORD_2(oword, FRF_AB_MD_PRT_ADR, prtad,
            FRF_AB_MD_DEV_ADR, devad);
        EFX_BAR_WRITEO(enp, FR_AB_MD_ID_REG, &oword);

        addr = reg;

        EFX_POPULATE_OWORD_1(oword, FRF_AB_MD_PHY_ADR, addr);
        EFX_BAR_WRITEO(enp, FR_AB_MD_PHY_ADR_REG, &oword);

        /* Write data */
        val = EFX_WORD_FIELD(*ewp, EFX_WORD_0);

        EFSYS_PROBE4(mdio_write, uint8_t, port, uint8_t, mmd,
            uint16_t, reg, uint16_t, val);

        EFX_POPULATE_OWORD_1(oword, FRF_AB_MD_TXD, (uint32_t)val);
        EFX_BAR_WRITEO(enp, FR_AB_MD_TXD_REG, &oword);

        /* Select clause 45 write cycle */
        EFX_POPULATE_OWORD_2(oword, FRF_AB_MD_WRC, 1, FRF_AB_MD_GC, 0);
```

```
128         EFX_BAR_WRITEO(enp, FR_AB_MD_CS_REG, &oword);

130         /* Wait for the cycle to complete */
131         if ((rc = falcon_mdio_wait(enp)) != 0)
132                 goto fail2;

134         EFSYS_UNLOCK(enp->en_eslp, state);
135         return (0);

137 fail2:
138         EFSYS_PROBE(fail2);
139 fail1:
140         EFSYS_PROBE1(fail1, int, rc);

142         EFSYS_UNLOCK(enp->en_eslp, state);
143         return (rc);
144 }

146         __checkReturn   int
147 falcon_mdio_read(
148         __in            efx_nic_t *enp,
149         __in            uint8_t port,
150         __in            uint8_t mmd,
151         __in            uint16_t reg,
152         __out           efx_word_t *ewp)
153 {
154         int state;
155         efx_oword_t oword;
156         uint32_t prtad;
157         uint32_t devad;
158         uint32_t addr;
159         uint16_t val;
160         int rc;

162         EFSYS_LOCK(enp->en_eslp, state);

164         /* Check MDIO is not currently being accessed */
165         if ((rc = falcon_mdio_wait(enp)) != 0)
166                 goto fail1;

168         prtad = port;
169         devad = mmd;

171         /* Write address */
172         EFX_POPULATE_OWORD_2(oword, FRF_AB_MD_PRT_ADR, prtad,
173             FRF_AB_MD_DEV_ADR, devad);
174         EFX_BAR_WRITEO(enp, FR_AB_MD_ID_REG, &oword);

176         addr = reg;

178         EFX_POPULATE_OWORD_1(oword, FRF_AB_MD_PHY_ADR, addr);
179         EFX_BAR_WRITEO(enp, FR_AB_MD_PHY_ADR_REG, &oword);

181         /* Clear the data register */
182         EFX_SET_OWORD(oword);
183         EFX_BAR_WRITEO(enp, FR_AB_MD_RXD_REG, &oword);

185         /* Select clause 45 read cycle */
186         EFX_POPULATE_OWORD_2(oword, FRF_AB_MD_RDC, 1, FRF_AB_MD_GC, 0);
187         EFX_BAR_WRITEO(enp, FR_AB_MD_CS_REG, &oword);

189         /* Wait for the cycle to complete */
190         if ((rc = falcon_mdio_wait(enp)) != 0)
191                 goto fail2;

193         /* Read data */
```



```
194         EFX_BAR_READO(enp, FR_AB_MD_RXD_REG, &oword);
195         val = (uint16_t)EFX_OWORD_FIELD(oword, FRF_AB_MD_RXD);

197         EFSYS_PROBE4(mdio_read, uint8_t, port, uint8_t, mmd,
198             uint16_t, reg, uint16_t, val);

200         EFX_POPULATE_WORD_1(*ewp, EFX_WORD_0, val);

202         EFSYS_UNLOCK(enp->en_eslp, state);
203         return (0);

205 fail2:
206         EFSYS_PROBE(fail2);
207 fail1:
208         EFSYS_PROBE1(fail1, int, rc);

210         EFSYS_UNLOCK(enp->en_eslp, state);
211         return (rc);
212 }

214 #endif  /* EFSYS_OPT_FALCON */
215 #endif /* ! codereview */
```

```
**********************************************************
   30467 Thu Aug 22 18:59:24 2013
new/usr/src/uts/common/io/sfxge/falcon_nic.c
Merged sfxge driver
**********************************************************
```

```
/*
 * Copyright 2007-2013 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#include "efsys.h"
#include "efx.h"
#include "falcon_nvram.h"
#include "efx_types.h"
#include "efx_regs.h"
#include "efx_regs_pci.h"
#include "efx_impl.h"

#if EFSYS_OPT_FALCON

#if EFSYS_OPT_MAC_FALCON_XMAC
#include "falcon_xmac.h"
#endif

#if EFSYS_OPT_MAC_FALCON_GMAC
#include "falcon_gmac.h"
#endif

#if EFSYS_OPT_MON_LM87
#include "lm87.h"
#endif

#if EFSYS_OPT_MON_MAX6647
#include "max6647.h"
#endif

#if EFSYS_OPT_PHY_NULL
#include "nullphy.h"
#endif

#if EFSYS_OPT_PHY_QT2022C2
#include "qt2022c2.h"
#endif

#if EFSYS_OPT_PHY_SFX7101
#include "sfx7101.h"
```



```
#endif

#if EFSYS_OPT_PHY_TXC43128
#include "txc43128.h"
#endif

#if EFSYS_OPT_PHY_SFT9001
#include "sft9001.h"
#endif

#if EFSYS_OPT_PHY_QT2025C
#include "qt2025c.h"
#endif

static  __checkReturn           int
falcon_nic_cfg_raw_read(
        __in                    efx_nic_t *enp,
        __in                    uint32_t offset,
        __in                    uint32_t size,
        __out                   void *cfg)
{
        EFSYS_ASSERT3U(offset + size, <=, FALCON_NIC_CFG_RAW_SZ);

#if EFSYS_OPT_FALCON_NIC_CFG_OVERRIDE
        if (enp->en_u.falcon.enu_forced_cfg != NULL) {
                memcpy(cfg, enp->en_u.falcon.enu_forced_cfg + offset, size);
                return (0);
        }
#endif  /* EFSYS_OPT_FALCON_NIC_CFG_OVERRIDE */

        return falcon_spi_dev_read(enp, FALCON_SPI_FLASH, offset,
                    (caddr_t)cfg, size);
}

        __checkReturn           int
falcon_nic_cfg_raw_read_verify(
        __in                    efx_nic_t *enp,
        __in                    uint32_t offset,
        __in                    uint32_t size,
        __out                   uint8_t *cfg)
{
        uint32_t csum_offset;
        uint16_t magic, version, cksum;
        int rc;
        efx_word_t word;

        if ((rc = falcon_nic_cfg_raw_read(enp, CFG_MAGIC_REG_SF_OFST,
                    sizeof (word), &word)) != 0)
                goto fail1;

        magic = EFX_WORD_FIELD(word, MAGIC);

        if ((rc = falcon_nic_cfg_raw_read(enp, CFG_VERSION_REG_SF_OFST,
                    sizeof (word), &word)) != 0)
                goto fail2;

        version = EFX_WORD_FIELD(word, VERSION);

        cksum = 0;
        for (csum_offset = (version < 4) ? CFG_MAGIC_REG_SF_OFST : 0;
            csum_offset < FALCON_NIC_CFG_RAW_SZ;
            csum_offset += sizeof (efx_word_t)) {

                if ((rc = falcon_nic_cfg_raw_read(enp, csum_offset,
                            sizeof (word), &word)) != 0)
                        goto fail3;
```

```
129                 cksum += EFX_WORD_FIELD(word, EFX_WORD_0);
130         }
131         if (magic != MAGIC_DECODE || version < 2 || cksum != 0xffff) {
132                 rc = EINVAL;
133                 goto fail4;
134         }

136         if ((rc = falcon_nic_cfg_raw_read(enp, offset, size, cfg)) != 0)
137                 goto fail5;

139         return (0);

141 fail5:
142         EFSYS_PROBE(fail5);
143 fail4:
144         EFSYS_PROBE(fail4);
145 fail3:
146         EFSYS_PROBE(fail3);
147 fail2:
148         EFSYS_PROBE(fail2);
149 fail1:
150         EFSYS_PROBE1(fail1, int, rc);

152         return (rc);
153 }

155         __checkReturn   int
156 falcon_nic_probe(
157         __in            efx_nic_t *enp)
158 {
159         efx_port_t *epp = &(enp->en_port);
160         efx_nic_cfg_t *encp = &(enp->en_nic_cfg);
161         int rc;

163         EFSYS_ASSERT3U(enp->en_family, ==, EFX_FAMILY_FALCON);

165         /* Initialise the nvram */
166         if ((rc = falcon_nvram_init(enp)) != 0)
167                 goto fail1;

169         /* Probe the board configuration */
170         if ((rc = falcon_nic_cfg_build(enp, encp)) != 0)
171                 goto fail2;
172         epp->ep_adv_cap_mask = epp->ep_default_adv_cap_mask;

174         return (0);

176 fail2:
177         EFSYS_PROBE(fail2);
178 fail1:
179         EFSYS_PROBE1(fail1, int, rc);

181         return (rc);
182 }

184 #define FALCON_NIC_CFG_BUILD_LOWEST_REG                         \
185         MIN(CFG_BOARD_REV_REG_SF_OFST,                          \
186             MIN(CFG_BOARD_TYPE_REG_SF_OFST,                     \
187                 MIN(CFG_PHY_PORT_REG_SF_OFST,                   \
188                     MIN(CFG_PHY_TYPE_REG_SF_OFST,               \
189                         MIN(MAC_ADDRESS_SF_OFST,                \
190                             MIN(NIC_STAT_SF_OFST,               \
191                                 SRAM_CFG_SF_OFST))))))
```



```
194 #define FALCON_NIC_CFG_BUILD_HIGHEST_REG                        \
195         MAX(CFG_BOARD_REV_REG_SF_OFST + 1,                      \
196             MAX(CFG_BOARD_TYPE_REG_SF_OFST + 1,                 \
197                 MAX(CFG_PHY_PORT_REG_SF_OFST + 1,               \
198                     MAX(CFG_PHY_TYPE_REG_SF_OFST + 1,           \
199                         MAX(MAC_ADDRESS_SF_OFST + 6,            \
200                             MAX(NIC_STAT_SF_OFST + 16,          \
201                                 SRAM_CFG_SF_OFST + 16))))))

203 #define FALCON_NIC_CFG_BUILD_NEEDED_CFG_SIZE                    \
204         (FALCON_NIC_CFG_BUILD_HIGHEST_REG -                     \
205             FALCON_NIC_CFG_BUILD_LOWEST_REG)

207         __checkReturn   int
208 falcon_nic_cfg_build(
209         __in            efx_nic_t *enp,
210         __out           efx_nic_cfg_t *encp)
211 {
212         efx_port_t *epp = &(enp->en_port);
213         uint8_t cfg[FALCON_NIC_CFG_BUILD_NEEDED_CFG_SIZE];
214         uint8_t *origin = cfg - FALCON_NIC_CFG_BUILD_LOWEST_REG;
215         efx_oword_t *owordp;
216         uint8_t major;
217         uint8_t minor;
218         int rc;

220         EFSYS_ASSERT3U(enp->en_family, ==, EFX_FAMILY_FALCON);

222         (void) memset(encp, 0, sizeof (efx_nic_cfg_t));

224         if ((rc = falcon_nic_cfg_raw_read_verify(enp,
225             FALCON_NIC_CFG_BUILD_LOWEST_REG,
226             FALCON_NIC_CFG_BUILD_NEEDED_CFG_SIZE, cfg)) != 0)
227                 goto fail1;

229         encp->enc_board_type = EFX_BYTE_FIELD(
230                 ((efx_byte_t *)origin)[CFG_BOARD_TYPE_REG_SF_OFST],
231                 EFX_BYTE_0);

233         /* Read board revision */
234         major = EFX_BYTE_FIELD(
235             ((efx_byte_t *)origin)[CFG_BOARD_REV_REG_SF_OFST],
236             BOARD_REV_MAJOR);
237         minor = EFX_BYTE_FIELD(
238             ((efx_byte_t *)origin)[CFG_BOARD_REV_REG_SF_OFST],
239             BOARD_REV_MINOR);
240         enp->en_u.falcon.enu_board_rev = (major << 4) | minor;

242         /* Sram mode */
243         owordp = (efx_oword_t *)(origin + NIC_STAT_SF_OFST);
244         enp->en_u.falcon.enu_internal_sram =
245             EFX_OWORD_FIELD(*owordp, FRF_AB_ONCHIP_SRAM) != 0;
246         if (enp->en_u.falcon.enu_internal_sram) {
247                 enp->en_u.falcon.enu_sram_num_bank = 0;
248                 enp->en_u.falcon.enu_sram_bank_size = 0;

250                 /* Resource limits */
251                 encp->enc_evq_limit = 64;       /* Interrupt-capable */
252                 encp->enc_txq_limit = 512;
253                 encp->enc_rxq_limit = 384;
254                 encp->enc_buftbl_limit = 4096;
255         } else {
256                 uint32_t sram_rows;

258                 owordp = (efx_oword_t *)(origin + SRAM_CFG_SF_OFST);
259                 enp->en_u.falcon.enu_sram_num_bank =
```

```
260                     (uint8_t)EFX_OWORD_FIELD(*owordp, FRF_AZ_SRM_NUM_BANK);
261                 enp->en_u.falcon.enu_sram_bank_size =
262                     (uint8_t)EFX_OWORD_FIELD(*owordp, FRF_AZ_SRM_BANK_SIZE);
263                 sram_rows = (enp->en_u.falcon.enu_sram_num_bank + 1)
264                     << (18 + enp->en_u.falcon.enu_sram_bank_size);

266                 /* Resource limits */
267                 encp->enc_evq_limit = 64;       /* Interrupt-capable */
268                 encp->enc_txq_limit = EFX_TXQ_LIMIT_TARGET;
269                 encp->enc_rxq_limit = EFX_RXQ_LIMIT_TARGET;
270                 encp->enc_buftbl_limit = sram_rows -
271                     (encp->enc_txq_limit * EFX_TXQ_DC_NDESCS(EFX_TXQ_DC_SIZE) +
272                     encp->enc_rxq_limit * EFX_RXQ_DC_NDESCS(EFX_RXQ_DC_SIZE));
273         }

275         encp->enc_clk_mult = 1;
276         encp->enc_evq_timer_quantum_ns =
277                 EFX_EVQ_FALCON_TIMER_QUANTUM_NS / encp->enc_clk_mult;
278         encp->enc_evq_timer_max_us = (encp->enc_evq_timer_quantum_ns <<
279                 FRF_AB_TIMER_VAL_WIDTH) / 1000;

281         /* Determine system monitor configuration */
282         switch (encp->enc_board_type) {
283 #if EFSYS_OPT_MON_LM87
284         case BOARD_TYPE_SFE4002_DECODE:
285         case BOARD_TYPE_SFE4003_DECODE:
286         case BOARD_TYPE_SFE4005_DECODE:
287         case BOARD_TYPE_SFN4112F_DECODE:
288                 encp->enc_mon_type = EFX_MON_LM87;
289                 enp->en_u.falcon.enu_mon_devid = LM87_DEVID;
290 #if EFSYS_OPT_MON_STATS
291                 encp->enc_mon_stat_mask = LM87_STAT_MASK;
292 #endif
293                 break;
294 #endif  /* EFSYS_OPT_MON_LM87 */

296 #if EFSYS_OPT_MON_MAX6647
297         case BOARD_TYPE_SFE4001_DECODE:
298                 encp->enc_mon_type = EFX_MON_MAX6647;
299                 enp->en_u.falcon.enu_mon_devid = MAX6647_DEVID;
300 #if EFSYS_OPT_MON_STATS
301                 encp->enc_mon_stat_mask = MAX6647_STAT_MASK;
302 #endif
303                 break;
304 #endif  /* EFSYS_OPT_MON_MAX6647 */

306 #if EFSYS_OPT_MON_MAX6647
307         case BOARD_TYPE_SFN4111T_DECODE:
308                 encp->enc_mon_type = EFX_MON_MAX6647;
309 #if EFSYS_OPT_MON_STATS
310                 encp->enc_mon_stat_mask = MAX6647_STAT_MASK;
311 #endif
312                 /*
313                  * MAX6646 chips are identical to MAX6647 chips in every way
314                  * that matters to the driver so we pretend that the chip
315                  * is always a MAX6647, but adjust the identifier accordingly
316                  */
317                 enp->en_u.falcon.enu_mon_devid = (major == 0 && minor < 5) ?
318                     MAX6647_DEVID : MAX6646_DEVID;
319                 break;
320 #endif  /* EFSYS_OPT_MON_MAX6647 */

322         default:
323                 encp->enc_mon_type = EFX_MON_NULL;
324                 enp->en_u.falcon.enu_mon_devid = 0;
325 #if EFSYS_OPT_MON_STATS
```



```
326                 encp->enc_mon_stat_mask = 0;
327 #endif
328                 break;
329         }

331         /* Copy the feature flags */
332         encp->enc_features = enp->en_features;

334         /* Read PHY MII prt and type, and mac address */
335         encp->enc_port = EFX_BYTE_FIELD(
336                 ((efx_byte_t *)origin)[CFG_PHY_PORT_REG_SF_OFST], EFX_BYTE_0);
337         encp->enc_phy_type = EFX_BYTE_FIELD(
338                 ((efx_byte_t *)origin)[CFG_PHY_TYPE_REG_SF_OFST], EFX_BYTE_0);

340         EFX_MAC_ADDR_COPY(encp->enc_mac_addr, origin + MAC_ADDRESS_SF_OFST);

342         /* Populate phy capabalities */
343         EFX_STATIC_ASSERT(EFX_PHY_NULL == PHY_TYPE_NONE_DECODE);
344         EFX_STATIC_ASSERT(EFX_PHY_TXC43128 == PHY_TYPE_TXC43128_DECODE);
345         EFX_STATIC_ASSERT(EFX_PHY_SFX7101 == PHY_TYPE_SFX7101_DECODE);
346         EFX_STATIC_ASSERT(EFX_PHY_QT2022C2 == PHY_TYPE_QT2022C2_DECODE);
347         EFX_STATIC_ASSERT(EFX_PHY_SFT9001A == PHY_TYPE_SFT9001A_DECODE);
348         EFX_STATIC_ASSERT(EFX_PHY_QT2025C == PHY_TYPE_QT2025C_DECODE);
349         EFX_STATIC_ASSERT(EFX_PHY_SFT9001B == PHY_TYPE_SFT9001B_DECODE);

351         switch (encp->enc_phy_type) {
352 #if EFSYS_OPT_PHY_NULL
353         case PHY_TYPE_NONE_DECODE:
354                 epp->ep_fixed_port_type = EFX_PHY_MEDIA_XAUI;
355                 epp->ep_default_adv_cap_mask = NULLPHY_ADV_CAP_MASK;
356                 epp->ep_phy_cap_mask =
357                         NULLPHY_ADV_CAP_MASK | NULLPHY_ADV_CAP_PERM;
358 #if EFSYS_OPT_NAMES
359                 (void) strncpy(encp->enc_phy_name, "nullphy",
360                     sizeof (encp->enc_phy_name));
361 #endif  /* EFSYS_OPT_NAMES */
362 #if EFSYS_OPT_PHY_LED_CONTROL
363                 encp->enc_led_mask = NULLPHY_LED_MASK;
364 #endif  /* EFSYS_OPT_PHY_LED_CONTROL */
365 #if EFSYS_OPT_LOOPBACK
366                 encp->enc_loopback_types[EFX_LINK_10000FDX] =
367                     NULLPHY_LOOPBACK_MASK;
368 #endif  /* EFSYS_OPT_LOOPBACK */
369 #if EFSYS_OPT_PHY_STATS
370                 encp->enc_phy_stat_mask = NULLPHY_STAT_MASK;
371 #endif  /* EFSYS_OPT_PHY_STATS */
372 #if EFSYS_OPT_PHY_PROPS
373                 encp->enc_phy_nprops = NULLPHY_NPROPS;
374 #endif  /* EFSYS_OPT_PHY_PROPS */
375 #if EFSYS_OPT_PHY_BIST
376                 encp->enc_bist_mask = NULLPHY_BIST_MASK;
377 #endif  /* EFSYS_OPT_PHY_BIST */
378                 break;
379 #endif  /* EFSYS_OPT_PHY_NULL */

381 #if EFSYS_OPT_PHY_QT2022C2
382         case PHY_TYPE_QT2022C2_DECODE:
383                 epp->ep_fixed_port_type = EFX_PHY_MEDIA_XFP;
384                 epp->ep_default_adv_cap_mask = QT2022C2_ADV_CAP_MASK;
385                 epp->ep_phy_cap_mask =
386                         QT2022C2_ADV_CAP_MASK | QT2022C2_ADV_CAP_PERM;
387 #if EFSYS_OPT_NAMES
388                 (void) strncpy(encp->enc_phy_name, "qt2022c2",
389                     sizeof (encp->enc_phy_name));
390 #endif  /* EFSYS_OPT_NAMES */
391 #if EFSYS_OPT_PHY_LED_CONTROL
```

```
392                 encp->enc_led_mask = QT2022C2_LED_MASK;
393 #endif  /* EFSYS_OPT_PHY_LED_CONTROL */
394 #if EFSYS_OPT_LOOPBACK
395                 encp->enc_loopback_types[EFX_LINK_10000FDX] =
396                     QT2022C2_LOOPBACK_MASK;
397 #endif  /* EFSYS_OPT_LOOPBACK */
398 #if EFSYS_OPT_PHY_STATS
399                 encp->enc_phy_stat_mask = QT2022C2_STAT_MASK;
400 #endif  /* EFSYS_OPT_PHY_STATS */
401 #if EFSYS_OPT_PHY_PROPS
402                 encp->enc_phy_nprops = QT2022C2_NPROPS;
403 #endif  /* EFSYS_OPT_PHY_PROPS */
404 #if EFSYS_OPT_PHY_BIST
405                 encp->enc_bist_mask = QT2022C2_BIST_MASK;
406 #endif  /* EFSYS_OPT_PHY_BIST */
407                 break;
408 #endif  /* EFSYS_OPT_PHY_QT2022C2 */

410 #if EFSYS_OPT_PHY_SFX7101
411         case PHY_TYPE_SFX7101_DECODE:
412                 epp->ep_fixed_port_type = EFX_PHY_MEDIA_BASE_T;
413                 epp->ep_default_adv_cap_mask = SFX7101_ADV_CAP_MASK;
414                 epp->ep_phy_cap_mask =
415                         SFX7101_ADV_CAP_MASK | SFX7101_ADV_CAP_PERM;
416 #if EFSYS_OPT_NAMES
417                 (void) strncpy(encp->enc_phy_name, "sfx7101",
418                     sizeof (encp->enc_phy_name));
419 #endif  /* EFSYS_OPT_NAMES */
420 #if EFSYS_OPT_PHY_LED_CONTROL
421                 encp->enc_led_mask = SFX7101_LED_MASK;
422 #endif  /* EFSYS_OPT_PHY_LED_CONTROL */
423 #if EFSYS_OPT_LOOPBACK
424                 encp->enc_loopback_types[EFX_LINK_10000FDX] =
425                     SFX7101_LOOPBACK_MASK;
426 #endif  /* EFSYS_OPT_LOOPBACK */
427 #if EFSYS_OPT_PHY_STATS
428                 encp->enc_phy_stat_mask = SFX7101_STAT_MASK;
429 #endif  /* EFSYS_OPT_PHY_STATS */
430 #if EFSYS_OPT_PHY_PROPS
431                 encp->enc_phy_nprops = SFX7101_NPROPS;
432 #endif  /* EFSYS_OPT_PHY_PROPS */
433 #if EFSYS_OPT_PHY_BIST
434                 encp->enc_bist_mask = SFX7101_BIST_MASK;
435 #endif  /* EFSYS_OPT_PHY_BIST */
436                 break;
437 #endif  /* EFSYS_OPT_PHY_SFX7101 */

439 #if EFSYS_OPT_PHY_TXC43128
440         case PHY_TYPE_TXC43128_DECODE:
441                 epp->ep_fixed_port_type = EFX_PHY_MEDIA_CX4;
442                 epp->ep_default_adv_cap_mask = TXC43128_ADV_CAP_MASK;
443                 epp->ep_phy_cap_mask =
444                         TXC43128_ADV_CAP_MASK | TXC43128_ADV_CAP_PERM;
445 #if EFSYS_OPT_NAMES
446                 (void) strncpy(encp->enc_phy_name, "txc43128",
447                     sizeof (encp->enc_phy_name));
448 #endif  /* EFSYS_OPT_NAMES */
449 #if EFSYS_OPT_PHY_LED_CONTROL
450                 encp->enc_led_mask = TXC43128_LED_MASK;
451 #endif  /* EFSYS_OPT_PHY_LED_CONTROL */
452 #if EFSYS_OPT_LOOPBACK
453                 encp->enc_loopback_types[EFX_LINK_10000FDX] =
454                     TXC43128_LOOPBACK_MASK;
455 #endif  /* EFSYS_OPT_LOOPBACK */
456 #if EFSYS_OPT_PHY_STATS
457                 encp->enc_phy_stat_mask = TXC43128_STAT_MASK;
```



```
458 #endif  /* EFSYS_OPT_PHY_STATS */
459 #if EFSYS_OPT_PHY_PROPS
460                 encp->enc_phy_nprops = TXC43128_NPROPS;
461 #endif  /* EFSYS_OPT_PHY_PROPS */
462 #if EFSYS_OPT_PHY_BIST
463                 encp->enc_bist_mask = TXC43128_BIST_MASK;
464 #endif  /* EFSYS_OPT_PHY_BIST */
465                 break;
466 #endif  /* EFSYS_OPT_PHY_TXC43128 */

468 #if EFSYS_OPT_PHY_SFT9001
469         case PHY_TYPE_SFT9001A_DECODE:
470         case PHY_TYPE_SFT9001B_DECODE:
471                 epp->ep_fixed_port_type = EFX_PHY_MEDIA_BASE_T;
472                 epp->ep_default_adv_cap_mask = SFT9001_ADV_CAP_MASK;
473                 epp->ep_phy_cap_mask =
474                         SFT9001_ADV_CAP_MASK | SFT9001_ADV_CAP_PERM;
475 #if EFSYS_OPT_NAMES
476                 (void) strncpy(encp->enc_phy_name, "sft9001",
477                     sizeof (encp->enc_phy_name));
478 #endif  /* EFSYS_OPT_NAMES */
479 #if EFSYS_OPT_PHY_LED_CONTROL
480                 encp->enc_led_mask = SFT9001_LED_MASK;
481 #endif  /* EFSYS_OPT_PHY_LED_CONTROL */
482 #if EFSYS_OPT_LOOPBACK
483                 encp->enc_loopback_types[EFX_LINK_10000FDX] =
484                     SFT9001_10G_LOOPBACK_MASK;
485                 encp->enc_loopback_types[EFX_LINK_1000FDX] =
486                     SFT9001_1G_LOOPBACK_MASK;
487 #endif  /* EFSYS_OPT_LOOPBACK */
488 #if EFSYS_OPT_PHY_STATS
489                 encp->enc_phy_stat_mask = SFT9001_STAT_MASK;
490 #endif  /* EFSYS_OPT_PHY_STATS */
491 #if EFSYS_OPT_PHY_PROPS
492                 encp->enc_phy_nprops = SFT9001_NPROPS;
493 #endif  /* EFSYS_OPT_PHY_PROPS */
494 #if EFSYS_OPT_PHY_BIST
495                 encp->enc_bist_mask = SFT9001_BIST_MASK;
496 #endif  /* EFSYS_OPT_PHY_BIST */
497                 break;
498 #endif  /* EFSYS_OPT_PHY_SFT9001 */

500 #if EFSYS_OPT_PHY_QT2025C
501         case EFX_PHY_QT2025C:
502                 epp->ep_fixed_port_type = EFX_PHY_MEDIA_SFP_PLUS;
503                 epp->ep_default_adv_cap_mask = QT2025C_ADV_CAP_MASK;
504                 epp->ep_phy_cap_mask =
505                         QT2025C_ADV_CAP_MASK | QT2025C_ADV_CAP_PERM;
506 #if EFSYS_OPT_NAMES
507                 (void) strncpy(encp->enc_phy_name, "qt2025c",
508                     sizeof (encp->enc_phy_name));
509 #endif  /* EFSYS_OPT_NAMES */
510 #if EFSYS_OPT_PHY_LED_CONTROL
511                 encp->enc_led_mask = QT2025C_LED_MASK;
512 #endif  /* EFSYS_OPT_PHY_LED_CONTROL */
513 #if EFSYS_OPT_LOOPBACK
514                 encp->enc_loopback_types[EFX_LINK_10000FDX] =
515                     QT2025C_LOOPBACK_MASK;
516 #endif  /* EFSYS_OPT_LOOPBACK */
517 #if EFSYS_OPT_PHY_STATS
518                 encp->enc_phy_stat_mask = QT2025C_STAT_MASK;
519 #endif  /* EFSYS_OPT_PHY_STATS */
520 #if EFSYS_OPT_PHY_PROPS
521                 encp->enc_phy_nprops = QT2025C_NPROPS;
522 #endif  /* EFSYS_OPT_PHY_PROPS */
523 #if EFSYS_OPT_PHY_BIST
```

```
524                 encp->enc_bist_mask = QT2025C_BIST_MASK;
525 #endif  /* EFSYS_OPT_PHY_BIST */
526                 break;
527 #endif  /* EFSYS_OPT_PHY_QT2025C */

529         default:
530                 rc = ENOTSUP;
531                 goto fail2;
532         }

534 #if EFSYS_OPT_LOOPBACK
535         encp->enc_loopback_types[EFX_LINK_UNKNOWN] =
536             (1 << EFX_LOOPBACK_OFF) |
537             encp->enc_loopback_types[EFX_LINK_1000FDX] |
538             encp->enc_loopback_types[EFX_LINK_10000FDX];
539 #endif  /* EFSYS_OPT_LOOPBACK */

541         return (0);

543 fail2:
544         EFSYS_PROBE(fail2);
545 fail1:
546         EFSYS_PROBE1(fail1, int, rc);

548         return (rc);
549 }

551 #if EFSYS_OPT_PCIE_TUNE

553 static                  void
554 falcon_nic_pcie_core_read(
555         __in            efx_nic_t *enp,
556         __in            uint32_t addr,
557         __out           efx_dword_t *edp)
558 {
559         int lstate;
560         efx_oword_t oword;
561         uint32_t val;

563         EFSYS_ASSERT3U(enp->en_family, ==, EFX_FAMILY_FALCON);

565         EFSYS_LOCK(enp->en_eslp, lstate);

567         EFX_POPULATE_OWORD_2(oword, FRF_BB_PCIE_CORE_TARGET_REG_ADRS, addr,
568             FRF_BB_PCIE_CORE_INDIRECT_ACCESS_DIR, 0);
569         EFX_BAR_WRITEO(enp, FR_BB_PCIE_CORE_INDIRECT_REG, &oword);

571         EFSYS_SPIN(10);

573         EFX_BAR_READO(enp, FR_BB_PCIE_CORE_INDIRECT_REG, &oword);
574         val = EFX_OWORD_FIELD(oword, FRF_BB_PCIE_CORE_TARGET_DATA);

576         EFX_POPULATE_DWORD_1(*edp, EFX_DWORD_0, val);

578         EFSYS_UNLOCK(enp->en_eslp, lstate);
579 }

581 static                  void
582 falcon_nic_pcie_core_write(
583         __in            efx_nic_t *enp,
584         __in            uint32_t addr,
585         __out           efx_dword_t *edp)
586 {
587         int lstate;
588         efx_oword_t oword;
589         uint32_t val;
```



```
591         EFSYS_ASSERT3U(enp->en_family, ==, EFX_FAMILY_FALCON);

593         EFSYS_LOCK(enp->en_eslp, lstate);

595         val = EFX_DWORD_FIELD(*edp, EFX_DWORD_0);

597         EFX_POPULATE_OWORD_3(oword, FRF_BB_PCIE_CORE_TARGET_REG_ADRS, addr,
598             FRF_BB_PCIE_CORE_TARGET_DATA, val,
599             FRF_BB_PCIE_CORE_INDIRECT_ACCESS_DIR, 0);
600         EFX_BAR_WRITEO(enp, FR_BB_PCIE_CORE_INDIRECT_REG, &oword);

602         EFSYS_UNLOCK(enp->en_eslp, lstate);
603 }
604 #endif  /* EFSYS_OPT_PCIE_TUNE || EFSYS_OPT_DIAG */

606 #if EFSYS_OPT_PCIE_TUNE

608 typedef struct falcon_pcie_rpl_s {
609         size_t          fpr_tlp_size;
610         uint32_t        fpr_value[4];
611 } falcon_pcie_rpl_t;

613 /*  TLP         1x      2x      4x      8x */
614 static falcon_pcie_rpl_t        __cs falcon_nic_pcie_rpl[] = {
615         { 128,  { 421,  257,    174,    166 } },
616         { 256,  { 698,  391,    241,    225 } },
617         { 512,  { 903,  498,    295,    193 } },
618         { 1024, { 1670, 881,    487,    290 } }
619 };

621 static                  void
622 falcon_nic_pcie_rpl_tl_set(
623         __in            efx_nic_t *enp,
624         __in            unsigned int nlanes)
625 {
626         uint32_t index;
627         uint32_t current;
628         falcon_pcie_rpl_t *fprp;
629         uint32_t expected;
630         efx_dword_t dword;

632         EFSYS_ASSERT3U(nlanes, >, 0);
633         EFSYS_ASSERT3U(nlanes, <=, 8);
634         EFSYS_ASSERT(ISP2(nlanes));

636         /* Get the appropriate set of replay timer values */
637         falcon_nic_pcie_core_read(enp, PCR_AB_DEV_CTL_REG, &dword);

639         index = EFX_DWORD_FIELD(dword, PCRF_AZ_MAX_PAYL_SIZE);
640         if (index >= 4) {
641                 EFSYS_PROBE1(fail1, int, EIO);
642                 return;
643         }

645         fprp = (falcon_pcie_rpl_t *)&(falcon_nic_pcie_rpl[index]);

647         EFSYS_PROBE1(pcie_tlp_size, size_t, fprp->fpr_tlp_size);

649         for (index = 0; index < 4; index++)
650                 if ((1 << index) == nlanes)
651                         break;

653         /* Get the current replay timer value */
654         falcon_nic_pcie_core_read(enp, PCR_AC_ACK_LAT_TMR_REG, &dword);
655         current = EFX_DWORD_FIELD(dword, PCRF_AC_RT);
```

```
657         /* Get the appropriate replay timer value from the set */
658         expected = fprp->fpr_value[index];
659 
660         EFSYS_PROBE2(pcie_rpl_tl, uint32_t, current, uint32_t, expected);
661 
662         EFSYS_PROBE1(pcie_ack_tl,
663             uint32_t, EFX_DWORD_FIELD(dword, PCRF_AC_ALT));
664 
665         if (expected > current) {
666                 EFX_SET_DWORD_FIELD(dword, PCRF_AC_RT, expected);
667                 falcon_nic_pcie_core_write(enp, PCR_AC_ACK_LAT_TMR_REG,
668                     &dword);
669         }
670 }
671 
672 static                  void
673 falcon_nic_pcie_ack_freq_set(
674         __in            efx_nic_t *enp,
675         __in            uint32_t freq)
676 {
677         efx_dword_t dword;
678 
679         falcon_nic_pcie_core_read(enp, PCR_AC_ACK_FREQ_REG, &dword);
680 
681         EFSYS_PROBE2(pcie_ack_freq,
682             uint32_t, EFX_DWORD_FIELD(dword, PCRF_AC_ACK_FREQ),
683             uint32_t, freq);
684 
685         /* Set to zero to ensure that we always ACK after timeout */
686         EFX_SET_DWORD_FIELD(dword, PCRF_AC_ACK_FREQ, freq);
687         falcon_nic_pcie_core_write(enp, PCR_AC_ACK_FREQ_REG, &dword);
688 }
689 
690                         int
691 falcon_nic_pcie_tune(
692         __in            efx_nic_t *enp,
693         __in            unsigned int nlanes)
694 {
695         EFSYS_ASSERT3U(enp->en_family, ==, EFX_FAMILY_FALCON);
696 
697         enp->en_u.falcon.enu_nlanes = nlanes;
698         return (0);
699 }
700 
701 #endif  /* EFSYS_OPT_PCIE_TUNE */
702 
703         __checkReturn   int
704 falcon_nic_reset(
705         __in            efx_nic_t *enp)
706 {
707         efx_nic_cfg_t *encp = &(enp->en_nic_cfg);
708         falcon_i2c_t *fip = &(enp->en_u.falcon.enu_fip);
709         efx_oword_t oword;
710         unsigned int count;
711         int rc;
712 
713         EFSYS_ASSERT3U(enp->en_family, ==, EFX_FAMILY_FALCON);
714 
715         /* Check the reset register */
716         EFX_BAR_READO(enp, FR_AB_GLB_CTL_REG, &oword);
717 
718         /* Select units for reset */
719         EFX_SET_OWORD_FIELD(oword, FRF_AB_RST_CS, 1);
720         EFX_SET_OWORD_FIELD(oword, FRF_AB_RST_TX, 1);
721         EFX_SET_OWORD_FIELD(oword, FRF_AB_RST_XGTX, 1);
```



```
722         EFX_SET_OWORD_FIELD(oword, FRF_AB_RST_RX, 1);
723         EFX_SET_OWORD_FIELD(oword, FRF_AB_RST_XGRX, 1);
724         EFX_SET_OWORD_FIELD(oword, FRF_AB_RST_SR, 1);
725         EFX_SET_OWORD_FIELD(oword, FRF_AB_RST_EV, 1);
726         EFX_SET_OWORD_FIELD(oword, FRF_AB_RST_EM, 1);
727         EFX_SET_OWORD_FIELD(oword, FRF_AB_RST_PCIE_STKY, 1);
728         EFX_SET_OWORD_FIELD(oword, FRF_BB_RST_BIU, 1);
729         EFX_SET_OWORD_FIELD(oword, FRF_AB_RST_XAUI_SD, 1);
730 
731         /* Initiate reset */
732         EFX_BAR_WRITEO(enp, FR_AB_GLB_CTL_REG, &oword);
733 
734         /* Wait for the reset to complete */
735         count = 0;
736         do {
737                 EFSYS_PROBE1(wait, unsigned int, count);
738 
739                 /* Spin for 10 us */
740                 EFSYS_SPIN(10);
741 
742                 /* Test for reset complete */
743                 EFX_BAR_READO(enp, FR_AB_GLB_CTL_REG, &oword);
744                 if (EFX_OWORD_FIELD(oword, FRF_AB_RST_CS) == 0 &&
745                     EFX_OWORD_FIELD(oword, FRF_AB_RST_TX) == 0 &&
746                     EFX_OWORD_FIELD(oword, FRF_AB_RST_XGTX) == 0 &&
747                     EFX_OWORD_FIELD(oword, FRF_AB_RST_RX) == 0 &&
748                     EFX_OWORD_FIELD(oword, FRF_AB_RST_XGRX) == 0 &&
749                     EFX_OWORD_FIELD(oword, FRF_AB_RST_SR) == 0 &&
750                     EFX_OWORD_FIELD(oword, FRF_AB_RST_EV) == 0 &&
751                     EFX_OWORD_FIELD(oword, FRF_AB_RST_EM) == 0 &&
752                     EFX_OWORD_FIELD(oword, FRF_AB_RST_PCIE_STKY) == 0 &&
753                     EFX_OWORD_FIELD(oword, FRF_BB_RST_BIU) == 0 &&
754                     EFX_OWORD_FIELD(oword, FRF_AB_RST_XAUI_SD) == 0)
755                         goto done;
756         } while (++count < 1000);
757 
758         rc = ETIMEDOUT;
759         goto fail1;
760 
761 done:
762         /* GPIO initialization */
763         EFX_BAR_READO(enp, FR_AB_GPIO_CTL_REG, &oword);
764 
765         /* Set I2C SCL to 1 */
766         EFX_SET_OWORD_FIELD(oword, FRF_AB_GPIO0_OUT, 1);
767         EFX_SET_OWORD_FIELD(oword, FRF_AB_GPIO0_OEN, 1);
768 
769         /* Select external 1G MAC clock if it is available */
770         EFX_SET_OWORD_FIELD(oword, FRF_BB_USE_NIC_CLK,
771             (encp->enc_board_type == BOARD_TYPE_SFN4111T_DECODE));
772 
773         EFX_BAR_WRITEO(enp, FR_AB_GPIO_CTL_REG, &oword);
774 
775         fip->fi_sda = B_TRUE;
776         fip->fi_scl = B_TRUE;
777 
778         return (0);
779 
780 fail1:
781         EFSYS_PROBE1(fail1, int, rc);
782 
783         return (rc);
784 }
785 
786 static                  void
787 falcon_nic_timer_tbl_watchdog(
```

```
788         __in            efx_nic_t *enp)
789 {
790         efx_oword_t oword;

792         if (enp->en_family != EFX_FAMILY_FALCON)
793                 return;

795         /*
796          * Ensure that PCI writes to the event queue read pointers are spaced
797          * out far enough to avoid write loss.
798          */
799         EFX_BAR_READO(enp, FR_AZ_HW_INIT_REG, &oword);
800         EFX_SET_OWORD_FIELD(oword, FRF_AZ_POST_WR_MASK, 0xf);
801         EFX_SET_OWORD_FIELD(oword, FRF_AZ_WD_TIMER, 0x10);
802         EFX_BAR_WRITEO(enp, FR_AZ_HW_INIT_REG, &oword);
803 }

805 static                  void
806 falcon_rx_reset_recovery_enable(
807         __in            efx_nic_t *enp)
808 {
809         efx_oword_t oword;

811         /*
812          * Set number of channels for receive path and also set filter table
813          * search limits to 8, to reduce the frequency of RX_RECOVERY events
814          */
815         EFX_POPULATE_OWORD_4(oword,
816                              FRF_AZ_UDP_FULL_SRCH_LIMIT, 8,
817                              FRF_AZ_UDP_WILD_SRCH_LIMIT, 8,
818                              FRF_AZ_TCP_FULL_SRCH_LIMIT, 8,
819                              FRF_AZ_TCP_WILD_SRCH_LIMIT, 8);
820         EFX_BAR_WRITEO(enp, FR_AZ_RX_FILTER_CTL_REG, &oword);

822         /*
823          * Enable RX Self-Reset functionality.
824          * Disable ISCSI digest to reduce RX_RECOVERY frequency
825          */
826         EFX_BAR_READO(enp, FR_AZ_RX_SELF_RST_REG, &oword);
827         EFX_SET_OWORD_FIELD(oword, FRF_AZ_RX_ISCSI_DIS, 1);
828         EFX_SET_OWORD_FIELD(oword, FRF_AB_RX_SW_RST_REG, 1);
829         EFX_BAR_WRITEO(enp, FR_AZ_RX_SELF_RST_REG, &oword);
830 }

832         __checkReturn   int
833 falcon_nic_init(
834         __in            efx_nic_t *enp)
835 {
836         int rc;

838         if ((rc = falcon_sram_init(enp)) != 0)
839                 goto fail1;

841 #if EFSYS_OPT_PCIE_TUNE
842         /* Tune up the PCIe core */
843         if (enp->en_u.falcon.enu_nlanes > 0) {
844                 falcon_nic_pcie_rpl_tl_set(enp, enp->en_u.falcon.enu_nlanes);
845                 falcon_nic_pcie_ack_freq_set(enp, 0);
846         }
847 #endif
848         /* Fix NMI's due to premature timer_tbl biu watchdog */
849         falcon_nic_timer_tbl_watchdog(enp);

851         /* Enable RX_RECOVERY feature */
852         falcon_rx_reset_recovery_enable(enp);
```



```
854         return (0);

856 fail1:
857         EFSYS_PROBE1(fail1, int, rc);

859         return (rc);
860 }

862         __checkReturn   int
863 falcon_nic_mac_reset(
864         __in            efx_nic_t *enp)
865 {
866         efx_oword_t oword;
867         unsigned int count;
868         int rc;

870         EFSYS_ASSERT3U(enp->en_family, ==, EFX_FAMILY_FALCON);

872         EFX_BAR_READO(enp, FR_AB_MAC_CTRL_REG, &oword);
873         EFX_SET_OWORD_FIELD(oword, FRF_BB_TXFIFO_DRAIN_EN, 1);
874         EFX_BAR_WRITEO(enp, FR_AB_MAC_CTRL_REG, &oword);

876         /* Reset the MAC and EM units */
877         EFX_BAR_READO(enp, FR_AB_GLB_CTL_REG, &oword);

879         EFX_SET_OWORD_FIELD(oword, FRF_AB_RST_EM, 1);
880         EFX_SET_OWORD_FIELD(oword, FRF_AB_RST_XGRX, 1);
881         EFX_SET_OWORD_FIELD(oword, FRF_AB_RST_XGTX, 1);

883         /* Initiate reset */
884         EFX_BAR_WRITEO(enp, FR_AB_GLB_CTL_REG, &oword);

886         /* Wait for the reset to complete */
887         count = 0;
888         do {
889                 EFSYS_PROBE1(wait, unsigned int, count);

891                 /* Spin for 10 us */
892                 EFSYS_SPIN(10);

894                 /* Test for reset complete */
895                 EFX_BAR_READO(enp, FR_AB_GLB_CTL_REG, &oword);
896                 if (EFX_OWORD_FIELD(oword, FRF_AB_RST_EM) == 0 &&
897                     EFX_OWORD_FIELD(oword, FRF_AB_RST_XGRX) == 0 &&
898                     EFX_OWORD_FIELD(oword, FRF_AB_RST_XGTX) == 0)
899                         goto done;
900         } while (++count < 1000);

902         rc = ETIMEDOUT;
903         goto fail1;

905 done:
906         enp->en_reset_flags |= EFX_RESET_MAC;

908         return (0);

910 fail1:
911         EFSYS_PROBE1(fail1, int, rc);
912         return (rc);
913 }

915                         void
916 falcon_nic_phy_reset(
917         __in            efx_nic_t *enp)
918 {
919         efx_oword_t oword;
```

```
920         int state;

922         EFSYS_ASSERT3U(enp->en_family, ==, EFX_FAMILY_FALCON);

924         /* Set PHY_RSTn to 0 */
925         EFSYS_LOCK(enp->en_eslp, state);
926         EFX_BAR_READO(enp, FR_AB_GPIO_CTL_REG, &oword);
927         EFX_SET_OWORD_FIELD(oword, FRF_AB_GPIO2_OUT, 0);
928         EFX_SET_OWORD_FIELD(oword, FRF_AB_GPIO2_OEN, 1);
929         EFX_BAR_WRITEO(enp, FR_AB_GPIO_CTL_REG, &oword);
930         EFSYS_UNLOCK(enp->en_eslp, state);

932         EFSYS_SLEEP(500000);

934         EFSYS_LOCK(enp->en_eslp, state);
935         EFX_BAR_READO(enp, FR_AB_GPIO_CTL_REG, &oword);
936         EFX_SET_OWORD_FIELD(oword, FRF_AB_GPIO2_OEN, 0);
937         EFX_BAR_WRITEO(enp, FR_AB_GPIO_CTL_REG, &oword);
938         EFSYS_UNLOCK(enp->en_eslp, state);

940         EFSYS_SLEEP(100000);

942         enp->en_reset_flags |= EFX_RESET_PHY;
943 }

945                         void
946 falcon_nic_fini(
947         __in            efx_nic_t *enp)
948 {
949         EFSYS_ASSERT3U(enp->en_family, ==, EFX_FAMILY_FALCON);

951 #if EFSYS_OPT_PCIE_TUNE
952         falcon_nic_pcie_ack_freq_set(enp, 1);
953 #endif  /* EFSYS_OPT_PCIE_TUNE */

955         falcon_sram_fini(enp);
956 }

958                         void
959 falcon_nic_unprobe(
960         __in            efx_nic_t *enp)
961 {
962         EFSYS_ASSERT3U(enp->en_family, ==, EFX_FAMILY_FALCON);

964         falcon_nvram_fini(enp);
965 }

967 #if EFSYS_OPT_DIAG

969 static efx_register_set_t __cs  __falcon_b0_registers[] = {
970         { FR_AZ_ADR_REGION_REG_OFST, 0, 1 },
971         { FR_AZ_RX_CFG_REG_OFST, 0, 1 },
972         { FR_AZ_TX_CFG_REG_OFST, 0, 1 },
973         { FR_AZ_TX_RESERVED_REG_OFST, 0, 1 },
974         { FR_AB_MAC_CTRL_REG_OFST, 0, 1 },
975         { FR_AZ_SRM_TX_DC_CFG_REG_OFST, 0, 1 },
976         { FR_AZ_RX_DC_CFG_REG_OFST, 0, 1 },
977         { FR_AZ_RX_DC_PF_WM_REG_OFST, 0, 1 },
978         { FR_AZ_DP_CTRL_REG_OFST, 0, 1 },
979         { FR_AB_GM_CFG2_REG_OFST, 0, 1 },
980         { FR_AB_GMF_CFG0_REG_OFST, 0, 1 },
981         { FR_AB_XM_GLB_CFG_REG_OFST, 0, 1 },
982         { FR_AB_XM_TX_CFG_REG_OFST, 0, 1 },
983         { FR_AB_XM_RX_CFG_REG_OFST, 0, 1 },
984         { FR_AB_XM_RX_PARAM_REG_OFST, 0, 1 },
985         { FR_AB_XM_FC_REG_OFST, 0, 1 },
```



```
986         { FR_AB_XM_ADR_LO_REG_OFST, 0, 1 },
987         { FR_AB_XX_SD_CTL_REG_OFST, 0, 1 },
988 };

990 static const uint32_t __cs      __falcon_b0_register_masks[] = {
991         0x0003FFFF, 0x0003FFFF, 0x0003FFFF, 0x0003FFFF,
992         0xFFFFFFFE, 0x00017FFF, 0x00000000, 0x00000000,
993         0x7FFF0037, 0x00000000, 0x00000000, 0x00000000,
994         0xFFFEFE80, 0x1FFFFFFF, 0x020000FE, 0x007FFFFF,
995         0xFFFF0000, 0x00000000, 0x00000000, 0x00000000,
996         0x001FFFFF, 0x00000000, 0x00000000, 0x00000000,
997         0x0000000F, 0x00000000, 0x00000000, 0x00000000,
998         0x000003FF, 0x00000000, 0x00000000, 0x00000000,
999         0x00000FFF, 0x00000000, 0x00000000, 0x00000000,
1000        0x00007337, 0x00000000, 0x00000000, 0x00000000,
1001        0x00001F1F, 0x00000000, 0x00000000, 0x00000000,
1002        0x00000C68, 0x00000000, 0x00000000, 0x00000000,
1003        0x00080164, 0x00000000, 0x00000000, 0x00000000,
1004        0x07100A0C, 0x00000000, 0x00000000, 0x00000000,
1005        0x00001FF8, 0x00000000, 0x00000000, 0x00000000,
1006        0xFFFF0001, 0x00000000, 0x00000000, 0x00000000,
1007        0xFFFFFFFF, 0x00000000, 0x00000000, 0x00000000,
1008        0x0003FF0F, 0x00000000, 0x00000000, 0x00000000,
1009 };

1011 static efx_register_set_t __cs  __falcon_b0_tables[] = {
1012         { FR_AZ_RX_FILTER_TBL0_OFST, FR_AZ_RX_FILTER_TBL0_STEP,
1013             FR_AZ_RX_FILTER_TBL0_ROWS },
1014         { FR_AB_RX_FILTER_TBL1_OFST, FR_AB_RX_FILTER_TBL1_STEP,
1015             FR_AB_RX_FILTER_TBL1_ROWS },
1016         { FR_AZ_RX_DESC_PTR_TBL_OFST,
1017             FR_AZ_RX_DESC_PTR_TBL_STEP, FR_AB_RX_DESC_PTR_TBL_ROWS },
1018         { FR_AZ_TX_DESC_PTR_TBL_OFST,
1019             FR_AZ_TX_DESC_PTR_TBL_STEP, FR_AB_TX_DESC_PTR_TBL_ROWS },
1020 };

1022 static const uint32_t __cs      __falcon_b0_table_masks[] = {
1023         0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0x000003FF,
1024         0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0x000003FF,
1025         0xFFFFFFFE, 0x0FFFFFFF, 0x01800000, 0x00000000,
1026         0x3FFFFFFE, 0x0FFFFFFF, 0x0C000000, 0x00000000,
1027 };

1029         __checkReturn   int
1030 falcon_nic_register_test(
1031         __in            efx_nic_t *enp)
1032 {
1033         efx_register_set_t *rsp;
1034         const uint32_t *dwordp;
1035         unsigned int nitems;
1036         unsigned int count;
1037         int rc;

1039         /* Fill out the register mask entries */
1040         EFX_STATIC_ASSERT(EFX_ARRAY_SIZE(__falcon_b0_register_masks)
1041                           == EFX_ARRAY_SIZE(__falcon_b0_registers) * 4);

1043         nitems = EFX_ARRAY_SIZE(__falcon_b0_registers);
1044         dwordp = __falcon_b0_register_masks;
1045         for (count = 0; count < nitems; ++count) {
1046                 rsp = __falcon_b0_registers + count;
1047                 rsp->mask.eo_u32[0] = *dwordp++;
1048                 rsp->mask.eo_u32[1] = *dwordp++;
1049                 rsp->mask.eo_u32[2] = *dwordp++;
1050                 rsp->mask.eo_u32[3] = *dwordp++;
1051         }
```

```
1053         /* Fill out the register table entries */
1054         EFX_STATIC_ASSERT(EFX_ARRAY_SIZE(__falcon_b0_table_masks)
1055                           == EFX_ARRAY_SIZE(__falcon_b0_tables) * 4);

1057         nitems = EFX_ARRAY_SIZE(__falcon_b0_tables);
1058         dwordp = __falcon_b0_table_masks;
1059         for (count = 0; count < nitems; ++count) {
1060                 rsp = __falcon_b0_tables + count;
1061                 rsp->mask.eo_u32[0] = *dwordp++;
1062                 rsp->mask.eo_u32[1] = *dwordp++;
1063                 rsp->mask.eo_u32[2] = *dwordp++;
1064                 rsp->mask.eo_u32[3] = *dwordp++;
1065         }

1067         if ((rc = efx_nic_test_registers(enp, __falcon_b0_registers,
1068             EFX_ARRAY_SIZE(__falcon_b0_registers))) != 0)
1069                 goto fail1;

1071         if ((rc = efx_nic_test_tables(enp, __falcon_b0_tables,
1072             EFX_PATTERN_BYTE_ALTERNATE,
1073             EFX_ARRAY_SIZE(__falcon_b0_tables))) != 0)
1074                 goto fail2;

1076         if ((rc = efx_nic_test_tables(enp, __falcon_b0_tables,
1077             EFX_PATTERN_BYTE_CHANGING,
1078             EFX_ARRAY_SIZE(__falcon_b0_tables))) != 0)
1079                 goto fail3;

1081         if ((rc = efx_nic_test_tables(enp, __falcon_b0_tables,
1082             EFX_PATTERN_BIT_SWEEP, EFX_ARRAY_SIZE(__falcon_b0_tables))) != 0)
1083                 goto fail4;

1085         return (0);

1087 fail4:
1088         EFSYS_PROBE(fail4);
1089 fail3:
1090         EFSYS_PROBE(fail3);
1091 fail2:
1092         EFSYS_PROBE(fail2);
1093 fail1:
1094         EFSYS_PROBE1(fail1, int, rc);

1096         return (rc);
1097 }

1099 #endif  /* EFSYS_OPT_DIAG */

1101 #endif  /* EFSYS_OPT_FALCON */
1102 #endif /* ! codereview */
```

```
**********************************************************
   19002 Thu Aug 22 18:59:24 2013
new/usr/src/uts/common/io/sfxge/falcon_nvram.c
Merged sfxge driver
**********************************************************
```

```
/*
 * Copyright 2007-2013 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#include "efsys.h"
#include "efx.h"
#include "falcon_nvram.h"
#include "efx_types.h"
#include "efx_impl.h"

#if EFSYS_OPT_FALCON

#if EFSYS_OPT_MON_LM87
#include "lm87.h"
#endif

#if EFSYS_OPT_MON_MAX6647
#include "max6647.h"
#endif

#if EFSYS_OPT_NVRAM_SFX7101
#include "sfx7101.h"
#endif

#if EFSYS_OPT_NVRAM_SFT9001
#include "sft9001.h"
#endif

#define FALCON_NVRAM_INIT_LOWEST_REG                            \
        MIN(CFG_VERSION_REG_SF_OFST,                            \
            MIN(CFG_FLASH_DEV_REG_SF_OFST,                      \
                CFG_EEPROM_DEV_REG_SF_OFST))

#define FALCON_NVRAM_INIT_HIGHEST_REG                           \
        MAX(CFG_VERSION_REG_SF_OFST,                            \
            MAX(CFG_FLASH_DEV_REG_SF_OFST,                      \
                CFG_EEPROM_DEV_REG_SF_OFST))

#define FALCON_NVRAM_INIT_NEEDED_CFG_SIZE                       \
        (FALCON_NVRAM_INIT_HIGHEST_REG + sizeof (efx_oword_t)   \
            - FALCON_NVRAM_INIT_LOWEST_REG)

        __checkReturn   int
falcon_nvram_init(
        __in            efx_nic_t *enp)
{
        falcon_spi_dev_t *fsdp;
        efx_oword_t oword;
        uint16_t version;
        int rc;
        uint8_t cfg[FALCON_NVRAM_INIT_NEEDED_CFG_SIZE];
        uint8_t *origin = cfg - FALCON_NVRAM_INIT_LOWEST_REG;

        /* All boards have flash and EEPROM */
        EFX_BAR_READO(enp, FR_AB_NIC_STAT_REG, &oword);
        EFX_SET_OWORD_FIELD(oword, FRF_AB_SF_PRST, 1);
        EFX_SET_OWORD_FIELD(oword, FRF_AB_EE_PRST, 1);
        EFX_BAR_WRITEO(enp, FR_AB_NIC_STAT_REG, &oword);

        /* Set up partial flash parameters */
        fsdp = &(enp->en_u.falcon.enu_fsd[FALCON_SPI_FLASH]);
        fsdp->fsd_sf_sel = 1;
        fsdp->fsd_adbcnt = 3;
        fsdp->fsd_munge = B_FALSE;

        if ((rc = falcon_nic_cfg_raw_read_verify(enp,
            FALCON_NVRAM_INIT_LOWEST_REG,
            FALCON_NVRAM_INIT_NEEDED_CFG_SIZE, cfg)) != 0)
                goto fail1;

        version = EFX_WORD_FIELD(
                *(efx_word_t *)(origin + CFG_VERSION_REG_SF_OFST), VERSION);

        if (version < 3) {
                fsdp->fsd_size = (size_t)1 << 17;
                fsdp->fsd_erase_cmd = 0x52;
                fsdp->fsd_erase_size = (size_t)1 << 15;
                fsdp->fsd_write_size = (size_t)1 << 8;
        } else {
                efx_dword_t *dwordp =
                        (efx_dword_t *)(origin + CFG_FLASH_DEV_REG_SF_OFST);

                if (EFX_DWORD_FIELD(*dwordp, SPI_DEV_ADBCNT) != 3 ||
                    EFX_DWORD_FIELD(*dwordp, SPI_DEV_SIZE) >= 24) {
                        rc = EINVAL;
                        goto fail2;
                }

                fsdp->fsd_size = (size_t)1 << EFX_DWORD_FIELD(*dwordp,
                    SPI_DEV_SIZE);
                fsdp->fsd_erase_cmd = EFX_DWORD_FIELD(*dwordp,
                                                      SPI_DEV_ERASE_CMD);
                fsdp->fsd_erase_size = (size_t)1 << EFX_DWORD_FIELD(*dwordp,
                    SPI_DEV_ERASE_SIZE);
                fsdp->fsd_write_size = (size_t)1 << EFX_DWORD_FIELD(*dwordp,
                    SPI_DEV_WRITE_SIZE);
        }

        /* Configure the EEPROM */
        fsdp = &(enp->en_u.falcon.enu_fsd[FALCON_SPI_EEPROM]);

        fsdp->fsd_sf_sel = 0;
        if (version < 3) {
                fsdp->fsd_adbcnt = 1;
                fsdp->fsd_size = (size_t)1 << 9;
                fsdp->fsd_munge = B_TRUE;
```

```
128                 fsdp->fsd_erase_cmd = 0;
129                 fsdp->fsd_erase_size = 1;
130                 fsdp->fsd_write_size = (size_t)1 << 3;
131         } else {
132                 efx_dword_t *dwordp =
133                         (efx_dword_t *)(origin + CFG_EEPROM_DEV_REG_SF_OFST);

135                 fsdp->fsd_adbcnt = EFX_DWORD_FIELD(*dwordp, SPI_DEV_ADBCNT);
136                 fsdp->fsd_size = (size_t)1 << EFX_DWORD_FIELD(*dwordp,
137                     SPI_DEV_SIZE);
138                 fsdp->fsd_munge = (EFX_DWORD_FIELD(*dwordp, SPI_DEV_SIZE) >
139                     fsdp->fsd_adbcnt * 8);
140                 fsdp->fsd_erase_cmd = EFX_DWORD_FIELD(*dwordp,
141                     SPI_DEV_ERASE_CMD);
142                 fsdp->fsd_erase_size = (size_t)1 << EFX_DWORD_FIELD(*dwordp,
143                     SPI_DEV_ERASE_SIZE);
144                 fsdp->fsd_write_size = (size_t)1 << EFX_DWORD_FIELD(*dwordp,
145                     SPI_DEV_WRITE_SIZE);
146         }

148         return (0);

150 fail2:
151         EFSYS_PROBE(fail2);
152 fail1:
153         EFSYS_PROBE1(fail1, int, rc);

155         (void) memset(enp->en_u.falcon.enu_fsd, 0,
156             sizeof (enp->en_u.falcon.enu_fsd));

158         return (rc);
159 }

161 #if EFSYS_OPT_NVRAM

163 typedef struct falcon_nvram_ops_s {
164         int     (*fnvo_size)(efx_nic_t *, size_t *);
165         int     (*fnvo_get_version)(efx_nic_t *, uint32_t *, uint16_t *);
166         int     (*fnvo_rw_start)(efx_nic_t *, size_t *);
167         int     (*fnvo_read_chunk)(efx_nic_t *, unsigned int,
168                     caddr_t, size_t);
169         int     (*fnvo_erase)(efx_nic_t *);
170         int     (*fnvo_write_chunk)(efx_nic_t *, unsigned int,
171                     caddr_t, size_t);
172         void    (*fnvo_rw_finish)(efx_nic_t *);
173 } falcon_nvram_ops_t;

175 #if EFSYS_OPT_NVRAM_FALCON_BOOTROM

177 #define FALCON_GPXE_IMAGE_OFFSET        0x8000
178 #define FALCON_GPXE_IMAGE_SIZE          0x18000

180 static  __checkReturn           int
181 falcon_nvram_bootrom_size(
182         __in                    efx_nic_t *enp,
183         __out                   size_t *sizep)
184 {
185         _NOTE(ARGUNUSED(enp))
186         EFSYS_ASSERT(sizep != NULL);
187         *sizep = FALCON_GPXE_IMAGE_SIZE;

189         return (0);
190 }

192 static  __checkReturn           int
193 falcon_nvram_bootrom_get_version(
```



```
194         __in                    efx_nic_t *enp,
195         __out                   uint32_t *subtypep,
196         __out_ecount(4)         uint16_t version[4])
197 {
198         const char prefix[] = "Solarstorm Boot Manager (v";
199         char buf[16], p;
200         size_t current, needle;
201         uint16_t *versionp;
202         int rc;

204         version[0] = version[1] = version[2] = version[3] = 0;
205         versionp = NULL;
206         needle = 0;

208         /*
209          * Search from [current, end) for prefix, and return the
210          * trailing four decimal number.
211          */
212         for (current = 0; current < 0x600; current++) {
213                 if (current % sizeof (buf) == 0) {
214                         if ((rc = falcon_spi_dev_read(enp, FALCON_SPI_FLASH,
215                             FALCON_GPXE_IMAGE_OFFSET + current, buf,
216                             sizeof (buf))) != 0)
217                                 break;
218                 }

220                 p = buf[current % sizeof (buf)];
221                 if (versionp == NULL) {
222                         if (prefix[needle] == p) {
223                                 ++needle;
224                                 if (needle == sizeof (prefix) - 1)
225                                         versionp = version;
226                         } else
227                                 needle = 0;
228                 } else {
229                         if (p == ')' && versionp == version + 3)
230                                 goto done;
231                         else if (p >= '0' && p <= '9')
232                                 *versionp = (*versionp * 10) + (p - '0');
233                         else if (p == '.' && versionp != version + 3)
234                                 ++versionp;
235                         else
236                                 /* Invalid format */
237                                 break;
238                 }
239         }

241         version[0] = version[1] = version[2] = version[3] = 0;

243 done:
244         *subtypep = 0;  /* Falcon bootrom is type 0 */

246         return (0);
247 }

249 static  __checkReturn           int
250 falcon_nvram_bootrom_rw_start(
251         __in                    efx_nic_t *enp,
252         __out                   size_t *chunk_sizep)
253 {
254         _NOTE(ARGUNUSED(enp))
255         if (chunk_sizep != NULL)
256                 *chunk_sizep = sizeof (efx_oword_t);

258         return (0);
259 }
```

```
261 static  __checkReturn          int
262 falcon_nvram_bootrom_read_chunk(
263         __in                    efx_nic_t *enp,
264         __in                    unsigned int offset,
265         __out_bcount(size)      caddr_t data,
266         __in                    size_t size)
267 {
268         int rc;

270         EFSYS_ASSERT3U(size + offset, <=, FALCON_GPXE_IMAGE_SIZE);

272         if ((rc = falcon_spi_dev_read(enp, FALCON_SPI_FLASH,
273             FALCON_GPXE_IMAGE_OFFSET + offset, data, size)) != 0)
274                 goto fail1;

276         return (0);

278 fail1:
279         EFSYS_PROBE1(fail1, int, rc);

281         return (rc);
282 }

284 static   __checkReturn          int
285 falcon_nvram_bootrom_erase(
286         __in                    efx_nic_t *enp)
287 {
288         int rc;

290         if ((rc = falcon_spi_dev_erase(enp, FALCON_SPI_FLASH,
291             FALCON_GPXE_IMAGE_OFFSET, FALCON_GPXE_IMAGE_SIZE)) != 0)
292                 goto fail1;

294         return (0);

296 fail1:
297         EFSYS_PROBE1(fail1, int, rc);

299         return (rc);
300 }

302 static  __checkReturn           int
303 falcon_nvram_bootrom_write_chunk(
304         __in                    efx_nic_t *enp,
305         __in                    unsigned int offset,
306         __out_bcount(size)      caddr_t base,
307         __in                    size_t size)
308 {
309         int rc;

311         if ((rc = falcon_spi_dev_write(enp, FALCON_SPI_FLASH,
312             FALCON_GPXE_IMAGE_OFFSET + offset, base, size)) != 0)
313                 goto fail1;

315         return (0);

317 fail1:
318         EFSYS_PROBE1(fail1, int, rc);

320         return (rc);
321 }

323 static falcon_nvram_ops_t       __cs    __falcon_nvram_bootrom_ops = {
324         falcon_nvram_bootrom_size,              /* fnvo_size */
325         falcon_nvram_bootrom_get_version,       /* fnvo_get_version */
```



```
326         falcon_nvram_bootrom_rw_start,          /* fnvo_rw_start */
327         falcon_nvram_bootrom_read_chunk,        /* fnvo_read_chunk */
328         falcon_nvram_bootrom_erase,             /* fnvo_erase */
329         falcon_nvram_bootrom_write_chunk,       /* fnvo_write_chunk */
330         NULL,                                   /* fnvo_rw_finish */
331 };

333 #define FALCON_GPXE_CFG_OFFSET          0x800

335 static  __checkReturn           int
336 falcon_nvram_bootrom_cfg_size(
337         __in                    efx_nic_t *enp,
338         __out                   size_t *sizep)
339 {
340         falcon_spi_dev_t *fsdp =
341                 &(enp->en_u.falcon.enu_fsd[FALCON_SPI_EEPROM]);
342         int rc;

344         EFSYS_ASSERT(sizep != NULL);
345         EFSYS_ASSERT(fsdp != NULL);

347         if (fsdp->fsd_size < FALCON_GPXE_CFG_OFFSET) {
348                 *sizep = 0;
349                 rc = ENOTSUP;
350                 goto fail1;
351         }

353         *sizep = fsdp->fsd_size - FALCON_GPXE_CFG_OFFSET;

355         return (0);

357 fail1:
358         EFSYS_PROBE1(fail1, int, rc);

360         return (rc);
361 }

363 static  __checkReturn           int
364 falcon_nvram_bootrom_cfg_get_version(
365         __in                    efx_nic_t *enp,
366         __out                   uint32_t *subtypep,
367         __out_ecount(4)         uint16_t version[4])
368 {
369         falcon_spi_dev_t *fsdp =
370                 &(enp->en_u.falcon.enu_fsd[FALCON_SPI_EEPROM]);
371         int rc;

373         EFSYS_ASSERT(fsdp != NULL);
374         if (fsdp->fsd_size < FALCON_GPXE_CFG_OFFSET) {
375                 rc = ENOTSUP;
376                 goto fail1;
377         }

379         /* gpxecfg is not versioned */
380         *subtypep = 0;
381         version[0] = version[1] = version[2] = version[3];

383         return (0);

385 fail1:
386         EFSYS_PROBE1(fail1, int, rc);

388         return (rc);
389 }

391 static  __checkReturn           int
```

```
392 falcon_nvram_bootrom_cfg_rw_start(
393         __in                    efx_nic_t *enp,
394         __out                   size_t *chunk_sizep)
395 {
396         falcon_spi_dev_t *fsdp =
397                 &(enp->en_u.falcon.enu_fsd[FALCON_SPI_EEPROM]);
398         int rc;
399
400         EFSYS_ASSERT(fsdp != NULL);
401         if (fsdp->fsd_size < FALCON_GPXE_CFG_OFFSET) {
402                 rc = ENOTSUP;
403                 goto fail1;
404         }
405
406         if (chunk_sizep != NULL)
407                 *chunk_sizep = sizeof (efx_oword_t);
408
409         return (0);
410
411 fail1:
412         EFSYS_PROBE1(fail1, int, rc);
413
414         return (rc);
415 }
416
417
418 static  __checkReturn           int
419 falcon_nvram_bootrom_cfg_read_chunk(
420         __in                    efx_nic_t *enp,
421         __in                    unsigned int offset,
422         __out_bcount(size)      caddr_t data,
423         __in                    size_t size)
424 {
425         falcon_spi_dev_t *fsdp =
426                 &(enp->en_u.falcon.enu_fsd[FALCON_SPI_EEPROM]);
427         int rc;
428
429         EFSYS_ASSERT(fsdp != NULL);
430         EFSYS_ASSERT3U(fsdp->fsd_size, >=, FALCON_GPXE_CFG_OFFSET);
431         EFSYS_ASSERT3U(offset + size, <=,
432             fsdp->fsd_size - FALCON_GPXE_CFG_OFFSET);
433
434         if ((rc = falcon_spi_dev_read(enp, FALCON_SPI_EEPROM,
435             FALCON_GPXE_CFG_OFFSET + offset, data, size)) != 0)
436                 goto fail1;
437
438         return (0);
439
440 fail1:
441         EFSYS_PROBE1(fail1, int, rc);
442
443         return (rc);
444 }
445
446 static  __checkReturn           int
447 falcon_nvram_bootrom_cfg_write_chunk(
448         __in                    efx_nic_t *enp,
449         __in                    unsigned int offset,
450         __in_bcount(size)       caddr_t base,
451         __in                    size_t size)
452 {
453         falcon_spi_dev_t *fsdp =
454                 &(enp->en_u.falcon.enu_fsd[FALCON_SPI_EEPROM]);
455         int rc;
456
457         EFSYS_ASSERT(fsdp != NULL);
```



```
458         EFSYS_ASSERT3U(fsdp->fsd_size, >=, FALCON_GPXE_CFG_OFFSET);
459         EFSYS_ASSERT3U(offset + size, <=,
460             fsdp->fsd_size - FALCON_GPXE_CFG_OFFSET);
461
462         if ((rc = falcon_spi_dev_write(enp, FALCON_SPI_EEPROM,
463             FALCON_GPXE_CFG_OFFSET + offset, base, size)) != 0)
464                 goto fail1;
465
466         return (0);
467
468 fail1:
469         EFSYS_PROBE1(fail1, int, rc);
470
471         return (rc);
472 }
473
474 static falcon_nvram_ops_t       __cs    __falcon_nvram_bootrom_cfg_ops = {
475         falcon_nvram_bootrom_cfg_size,          /* fnvo_size */
476         falcon_nvram_bootrom_cfg_get_version,   /* fnvo_get_version */
477         falcon_nvram_bootrom_cfg_rw_start,      /* fnvo_rw_start */
478         falcon_nvram_bootrom_cfg_read_chunk,    /* fnvo_read_chunk */
479         NULL,                                   /* fnvo_erase */
480         falcon_nvram_bootrom_cfg_write_chunk,   /* fnvo_write_chunk */
481         NULL,                                   /* fnvo_rw_finish */
482 };
483
484 #endif  /* EFSYS_OPT_NVRAM_FALCON_BOOTROM */
485
486 #if EFSYS_OPT_NVRAM_SFX7101
487
488 static falcon_nvram_ops_t       __cs    __falcon_sfx7101_ops = {
489         sfx7101_nvram_size,             /* fnvo_size */
490         sfx7101_nvram_get_version,      /* fnvo_get_version */
491         sfx7101_nvram_rw_start,         /* fnvo_rw_start */
492         sfx7101_nvram_read_chunk,       /* fnvo_read_chunk */
493         sfx7101_nvram_erase,            /* fnvo_erase */
494         sfx7101_nvram_write_chunk,      /* fnvo_write_chunk */
495         sfx7101_nvram_rw_finish,        /* fnvo_rw_finish */
496 };
497
498 #endif  /* EFSYS_OPT_NVRAM_SFX7101 */
499
500 #if EFSYS_OPT_NVRAM_SFT9001
501
502 static falcon_nvram_ops_t       __cs    __falcon_sft9001_ops = {
503         sft9001_nvram_size,             /* fnvo_size */
504         sft9001_nvram_get_version,      /* fnvo_get_version */
505         sft9001_nvram_rw_start,         /* fnvo_rw_start */
506         sft9001_nvram_read_chunk,       /* fnvo_read_chunk */
507         sft9001_nvram_erase,            /* fnvo_erase */
508         sft9001_nvram_write_chunk,      /* fnvo_write_chunk */
509         sft9001_nvram_rw_finish,        /* fnvo_rw_finish */
510 };
511
512 #endif  /* EFSYS_OPT_NVRAM_SFT9001 */
513
514 static  __checkReturn           int
515 falcon_nvram_get_ops(
516         __in                    efx_nic_t *enp,
517         __in                    efx_nvram_type_t type,
518         __out                   falcon_nvram_ops_t **fnvopp)
519 {
520         efx_nic_cfg_t *encp = &(enp->en_nic_cfg);
521         falcon_nvram_ops_t *fnvop;
522         int rc;
```

```
524         EFSYS_ASSERT3U(type, <, EFX_NVRAM_NTYPES);

526         switch (type) {
527 #if EFSYS_OPT_NVRAM_FALCON_BOOTROM
528         case EFX_NVRAM_BOOTROM_CFG:
529                 fnvop = (falcon_nvram_ops_t *)&__falcon_nvram_bootrom_cfg_ops;
530                 goto done;

532         case EFX_NVRAM_BOOTROM:
533                 fnvop = (falcon_nvram_ops_t *)&__falcon_nvram_bootrom_ops;
534                 goto done;
535 #endif
536         case EFX_NVRAM_PHY:
537                 switch (encp->enc_phy_type) {
538 #if EFSYS_OPT_NVRAM_SFX7101
539                 case EFX_PHY_SFX7101:
540                         fnvop = (falcon_nvram_ops_t *)&__falcon_sfx7101_ops;
541                         goto done;
542 #endif  /* EFSYS_OPT_NVRAM_SFX7101 */

544 #if EFSYS_OPT_NVRAM_SFT9001
545                 case EFX_PHY_SFT9001B:
546                         fnvop = (falcon_nvram_ops_t *)&__falcon_sft9001_ops;
547                         goto done;
548 #endif  /* EFSYS_OPT_NVRAM_SFT9001 */

550                 default:
551                         break;
552                 }

554                 break;

556         default:
557                 break;
558         }

560         rc = ENOTSUP;
561         goto fail1;

563 done:
564         *fnvopp = fnvop;

566         return (0);

568 fail1:
569         EFSYS_PROBE1(fail1, int, rc);

571         return (rc);
572 }

574 #if EFSYS_OPT_DIAG

576         __checkReturn           int
577 falcon_nvram_test(
578         __in                    efx_nic_t *enp)
579 {
580         efx_nic_cfg_t enc;
581         int rc;

583         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
584         EFSYS_ASSERT3U(enp->en_family, ==, EFX_FAMILY_FALCON);

586         if ((rc = falcon_nic_cfg_build(enp, &enc)) != 0)
587                 goto fail1;

589         return (0);
```



```
591 fail1:
592         EFSYS_PROBE1(fail1, int, rc);

594         return (rc);
595 }

597 #endif  /* EFSYS_OPT_DIAG */

599         __checkReturn           int
600 falcon_nvram_size(
601         __in                    efx_nic_t *enp,
602         __in                    efx_nvram_type_t type,
603         __out                   size_t *sizep)
604 {
605         falcon_nvram_ops_t *fnvop;
606         int rc;

608         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
609         EFSYS_ASSERT3U(enp->en_family, ==, EFX_FAMILY_FALCON);

611         if ((rc = falcon_nvram_get_ops(enp, type, &fnvop)) != 0)
612                 goto fail1;

614         if ((rc = fnvop->fnvo_size(enp, sizep)) != 0)
615                 goto fail2;

617         return (0);

619 fail2:
620         EFSYS_PROBE(fail2);
621 fail1:
622         EFSYS_PROBE1(fail1, int, rc);

624         return (rc);
625 }

627         __checkReturn           int
628 falcon_nvram_get_version(
629         __in                    efx_nic_t *enp,
630         __in                    efx_nvram_type_t type,
631         __out                   uint32_t *subtypep,
632         __out_ecount(4)         uint16_t version[4])
633 {
634         falcon_nvram_ops_t *fnvop;
635         int rc;

637         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
638         EFSYS_ASSERT3U(enp->en_family, ==, EFX_FAMILY_FALCON);

640         if ((rc = falcon_nvram_get_ops(enp, type, &fnvop)) != 0)
641                 goto fail1;

643         if ((rc = fnvop->fnvo_get_version(enp, subtypep, version)) != 0)
644                 goto fail2;

646         return (0);

648 fail2:
649         EFSYS_PROBE(fail2);
650 fail1:
651         EFSYS_PROBE1(fail1, int, rc);

653         return (rc);
654 }
```

```
656         __checkReturn           int
657 falcon_nvram_rw_start(
658         __in                    efx_nic_t *enp,
659         __in                    efx_nvram_type_t type,
660         __out                   size_t *chunk_sizep)
661 {
662         falcon_nvram_ops_t *fnvop;
663         int rc;

665         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
666         EFSYS_ASSERT3U(enp->en_family, ==, EFX_FAMILY_FALCON);

668         if ((rc = falcon_nvram_get_ops(enp, type, &fnvop)) != 0)
669                 goto fail1;

671         if ((rc = fnvop->fnvo_rw_start(enp, chunk_sizep)) != 0)
672                 goto fail2;

674         return (0);

676 fail2:
677         EFSYS_PROBE(fail2);
678 fail1:
679         EFSYS_PROBE1(fail1, int, rc);

681         return (rc);
682 }

684         __checkReturn           int
685 falcon_nvram_read_chunk(
686         __in                    efx_nic_t *enp,
687         __in                    efx_nvram_type_t type,
688         __in                    unsigned int offset,
689         __out_bcount(size)      caddr_t data,
690         __in                    size_t size)
691 {
692         falcon_nvram_ops_t *fnvop;
693         int rc;

695         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
696         EFSYS_ASSERT3U(enp->en_family, ==, EFX_FAMILY_FALCON);

698         if ((rc = falcon_nvram_get_ops(enp, type, &fnvop)) != 0)
699                 goto fail1;

701         if ((rc = fnvop->fnvo_read_chunk(enp, offset, data, size)) != 0)
702                 goto fail2;

704         return (0);

706 fail2:
707         EFSYS_PROBE(fail2);
708 fail1:
709         EFSYS_PROBE1(fail1, int, rc);

711         return (rc);
712 }

714         __checkReturn           int
715 falcon_nvram_erase(
716         __in                    efx_nic_t *enp,
717         __in                    efx_nvram_type_t type)
718 {
719         falcon_nvram_ops_t *fnvop;
720         int rc;
```



```
722         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
723         EFSYS_ASSERT3U(enp->en_family, ==, EFX_FAMILY_FALCON);

725         if ((rc = falcon_nvram_get_ops(enp, type, &fnvop)) != 0)
726                 goto fail1;

728         if (fnvop->fnvo_erase != NULL) {
729                 if ((rc = fnvop->fnvo_erase(enp)) != 0)
730                         goto fail2;
731         }

733         return (0);

735 fail2:
736         EFSYS_PROBE(fail2);
737 fail1:
738         EFSYS_PROBE1(fail1, int, rc);

740         return (rc);
741 }

743         __checkReturn           int
744 falcon_nvram_write_chunk(
745         __in                    efx_nic_t *enp,
746         __in                    efx_nvram_type_t type,
747         __in                    unsigned int offset,
748         __in_bcount(size)       caddr_t data,
749         __in                    size_t size)
750 {
751         falcon_nvram_ops_t *fnvop;
752         int rc;

754         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
755         EFSYS_ASSERT3U(enp->en_family, ==, EFX_FAMILY_FALCON);

757         if ((rc = falcon_nvram_get_ops(enp, type, &fnvop)) != 0)
758                 goto fail1;

760         if ((rc = fnvop->fnvo_write_chunk(enp, offset, data, size)) != 0)
761                 goto fail2;

763         return (0);

765 fail2:
766         EFSYS_PROBE(fail2);
767 fail1:
768         EFSYS_PROBE1(fail1, int, rc);

770         return (rc);
771 }

773                                 void
774 falcon_nvram_rw_finish(
775         __in                    efx_nic_t *enp,
776         __in                    efx_nvram_type_t type)
777 {
778         falcon_nvram_ops_t *fnvop;
779         int rc;

781         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
782         EFSYS_ASSERT3U(enp->en_family, ==, EFX_FAMILY_FALCON);

784         rc = falcon_nvram_get_ops(enp, type, &fnvop);
785         EFSYS_ASSERT(rc == 0);
786         if (rc == 0) {
787                 if (fnvop->fnvo_rw_finish != NULL)
```

```
788                         fnvop->fnvo_rw_finish(enp);
789         }
790 }

792         __checkReturn           int
793 falcon_nvram_set_version(
794         __in                    efx_nic_t *enp,
795         __in                    efx_nvram_type_t type,
796         __out                   uint16_t version[4])
797 {
798         falcon_nvram_ops_t *fnvop;
799         uint32_t subtype;
800         uint16_t old_version[4];
801         int rc;

803         _NOTE(ARGUNUSED(enp))
804         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
805         EFSYS_ASSERT3U(enp->en_family, ==, EFX_FAMILY_FALCON);

807         /*
808          * There is no room on Falcon to version anything, so it's
809          * inferred where possible from the underlying entity.
810          */
811         if ((rc = falcon_nvram_get_ops(enp, type, &fnvop)) != 0)
812                 goto fail1;

814         /*
815          * The user *really should be setting the version correctly
816          */
817         if ((rc = fnvop->fnvo_get_version(enp, &subtype, old_version)) != 0)
818                 goto fail2;
819         EFSYS_ASSERT(memcmp(old_version, version, sizeof (old_version)) == 0);

821         return (0);

823 fail2:
824         EFSYS_PROBE(fail2);
825 fail1:
826         EFSYS_PROBE1(fail1, int, rc);

828         return (rc);
829 }

831 #endif  /* EFSYS_OPT_NVRAM */

833                 void
834 falcon_nvram_fini(
835         __in    efx_nic_t *enp)
836 {
837         (void) memset(&enp->en_u.falcon.enu_fsd, 0,
838             sizeof (enp->en_u.falcon.enu_fsd));
839 }

841 #endif  /* EFSYS_OPT_FALCON */
842 #endif /* ! codereview */
```

```
**********************************************************
    3984 Thu Aug 22 18:59:24 2013
new/usr/src/uts/common/io/sfxge/falcon_nvram.h
Merged sfxge driver
**********************************************************
```

```
/*-
 * Copyright 2007-2013 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#ifndef _SYS_FALCON_NVRAM_H
#define _SYS_FALCON_NVRAM_H

#ifdef  __cplusplus
extern "C" {
#endif

/* PCI subsystem vendor ID */
#define PC_SS_VEND_ID_REG_SF_OFST 0x12c

/* PCI subsystem device ID */
#define PC_SS_ID_REG_SF_OFST 0x12e

/* PCIe serial number */
#define PCI_SN_SF_OFST 0x1c4

/* NVRAM and VPD configuration */
#define EE_VPD_CFG0_REG_SF_OFST 0x300

/* MAC address */
#define MAC_ADDRESS_SF_OFST 0x310

/* NIC stat */
#define NIC_STAT_SF_OFST 0x360

/* Sram config */
#define SRAM_CFG_SF_OFST 0x380

/* Magic number */
#define CFG_MAGIC_REG_SF_OFST 0x3a0

#define MAGIC_LBN 0
#define MAGIC_WIDTH 16
#define MAGIC_DECODE 0xfa1c

/* Version */
```



```
#define CFG_VERSION_REG_SF_OFST 0x3a2

#define VERSION_LBN 0
#define VERSION_WIDTH 16

/* Checksum */
#define CFG_CKSUM_REG_SF_OFST 0x3a4

#define CKSUM_LBN 0
#define CKSUM_WIDTH 16

/* PHY address */
#define CFG_PHY_PORT_REG_SF_OFST 0x3a8

#define PHY_PORT_LBN 0
#define PHY_PORT_WIDTH 8
#define PHY_PORT_INVALID_DECODE 0xff

/* PHY type */
#define CFG_PHY_TYPE_REG_SF_OFST 0x3a9

#define PHY_TYPE_LBN 0
#define PHY_TYPE_WIDTH 8
#define PHY_TYPE_NONE_DECODE 0x00
#define PHY_TYPE_TXC43128_DECODE 0x01
#define PHY_TYPE_88E1111_DECODE 0x02
#define PHY_TYPE_SFX7101_DECODE 0x03
#define PHY_TYPE_QT2022C2_DECODE 0x04
#define PHY_TYPE_SFT9001A_DECODE 0x08
#define PHY_TYPE_QT2025C_DECODE 0x09
#define PHY_TYPE_SFT9001B_DECODE 0x0a

/* ASIC revision */
#define CFG_ASIC_REV_REG_SF_OFST 0x3ac

#define ASIC_REV_MINOR_LBN 0
#define ASIC_REV_MINOR_WIDTH 8
#define ASIC_REV_MAJOR_LBN 8
#define ASIC_REV_MAJOR_WIDTH 16

/* Board revision */
#define CFG_BOARD_REV_REG_SF_OFST 0x3ae

#define BOARD_REV_MINOR_LBN 0
#define BOARD_REV_MINOR_WIDTH 4
#define BOARD_REV_MAJOR_LBN 4
#define BOARD_REV_MAJOR_WIDTH 4

/* Board type */
#define CFG_BOARD_TYPE_REG_SF_OFST 0x3af

#define BOARD_TYPE_LBN 0
#define BOARD_TYPE_WIDTH 8
#define BOARD_TYPE_SFE4001_DECODE 0x01
#define BOARD_TYPE_SFE4002_DECODE 0x02
#define BOARD_TYPE_SFE4003_DECODE 0x03
#define BOARD_TYPE_SFE4005_DECODE 0x04
#define BOARD_TYPE_SFN4111T_DECODE 0x51
#define BOARD_TYPE_SFN4112F_DECODE 0x52

/* EEPROM information */
#define CFG_EEPROM_DEV_REG_SF_OFST 0x3c0

/* FLASH information */
#define CFG_FLASH_DEV_REG_SF_OFST 0x3c4
```

```
128 #define SPI_DEV_SIZE_LBN 0
129 #define SPI_DEV_SIZE_WIDTH 5
130 #define SPI_DEV_ADBCNT_LBN 6
131 #define SPI_DEV_ADBCNT_WIDTH 2
132 #define SPI_DEV_ERASE_CMD_LBN 8
133 #define SPI_DEV_ERASE_CMD_WIDTH 8
134 #define SPI_DEV_ERASE_SIZE_LBN 16
135 #define SPI_DEV_ERASE_SIZE_WIDTH 5
136 #define SPI_DEV_WRITE_SIZE_LBN 24
137 #define SPI_DEV_WRITE_SIZE_WIDTH 5

139 #ifdef  __cplusplus
140 }
141 #endif

143 #endif  /* _SYS_FALCON_NVRAM_H */
144 #endif /* ! codereview */
```

```
**********************************************************
    9968 Thu Aug 22 18:59:24 2013
new/usr/src/uts/common/io/sfxge/falcon_spi.c
Merged sfxge driver
**********************************************************
```

```
/*
 * Copyright 2007-2013 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#include "efsys.h"
#include "efx.h"
#include "efx_types.h"
#include "efx_regs.h"
#include "efx_impl.h"

#if EFSYS_OPT_FALCON

/* Commands common to all known devices */

#define SPI_CMD_WRSR 0x01       /* write status register */
#define SPI_CMD_WRITE 0x02      /* Write data to memory array */
#define SPI_CMD_READ 0x03       /* Read data from memory array */
#define SPI_CMD_WRDI 0x04       /* reset write enable latch */
#define SPI_CMD_RDSR 0x05       /* read status register */
#define SPI_CMD_WREN 0x06       /* set write enable latch */

#define SPI_CMD_SECE 0x52       /* erase one sector */

#define SPI_STATUS_WPEN_LBN 7
#define SPI_STATUS_WPEN_WIDTH 1
#define SPI_STATUS_BP2_LBN 4
#define SPI_STATUS_BP2_WIDTH 1
#define SPI_STATUS_BP1_LBN 3
#define SPI_STATUS_BP1_WIDTH 1
#define SPI_STATUS_BP0_LBN 2
#define SPI_STATUS_BP0_WIDTH 1
#define SPI_STATUS_WEN_LBN 1
#define SPI_STATUS_WEN_WIDTH 1
#define SPI_STATUS_NRDY_LBN 0
#define SPI_STATUS_NRDY_WIDTH 1

static  __checkReturn   int
falcon_spi_wait(
        __in            efx_nic_t *enp)
{
```



```
        unsigned int count;
        int rc;

        count = 0;
        do {
                efx_oword_t oword;

                EFSYS_PROBE1(wait, unsigned int, count);

                EFX_BAR_READO(enp, FR_AB_EE_SPI_HCMD_REG, &oword);

                if (EFX_OWORD_FIELD(oword, FRF_AB_EE_SPI_HCMD_CMD_EN) == 0)
                        goto done;

                /* Spin for 10 us */
                EFSYS_SPIN(10);
        } while (++count < 10000);

        rc = ETIMEDOUT;
        goto fail1;

done:
        return (0);

fail1:
        EFSYS_PROBE1(fail1, int, rc);

        return (rc);
}

#define FALCON_SPI_COPY(_dst, _src, _size)                              \
        do {                                                            \
                unsigned int index;                                     \
                                                                        \
                for (index = 0; index < (unsigned int)(_size); index++) \
                        *((uint8_t *)(_dst) + index) =                  \
                            *((uint8_t *)(_src) + index);               \
                                                                        \
        _NOTE(CONSTANTCONDITION)                                        \
        } while (B_FALSE)

static  __checkReturn                   int
falcon_spi_cmd(
        __in                            efx_nic_t *enp,
        __in                            uint32_t sf_sel,
        __in                            uint32_t adbcnt,
        __in                            uint32_t cmd,
        __in                            uint32_t addr,
        __in                            boolean_t munge,
        __drv_when(cmd == SPI_CMD_WRSR || cmd == SPI_CMD_WRITE,
            __drv_in(__byte_readableTo(dabcnt)))
        __drv_when(cmd == SPI_CMD_RDSR || cmd == SPI_CMD_READ,
            __drv_out(__byte_readableTo(dabcnt)))
                                        caddr_t base,
        __in                            uint32_t dabcnt)
{
        uint32_t enc;
        efx_oword_t oword;
        int rc;

        EFSYS_ASSERT3U(dabcnt, <=,  sizeof (efx_oword_t));

        /* Wait for the SPI to become available */
        if ((rc = falcon_spi_wait(enp)) != 0)
                goto fail1;
```

```
128         /* Program the data register */
129         if (cmd == SPI_CMD_WRSR || cmd == SPI_CMD_WRITE) {
130                 EFSYS_ASSERT(base != NULL);
131                 EFSYS_ASSERT(dabcnt != 0);
132                 EFX_ZERO_OWORD(oword);
133                 FALCON_SPI_COPY(&oword, base, dabcnt);
134                 EFX_BAR_WRITEO(enp, FR_AB_EE_SPI_HDATA_REG, &oword);
135         }

137         /* Program the address register */
138         if (cmd == SPI_CMD_READ || cmd == SPI_CMD_WRITE ||
139             cmd == SPI_CMD_SECE) {
140                 EFX_POPULATE_OWORD_1(oword, FRF_AB_EE_SPI_HADR_ADR, addr);
141                 EFX_BAR_WRITEO(enp, FR_AB_EE_SPI_HADR_REG, &oword);

143                 enc = (munge) ? (cmd | ((addr >> 8) << 3)) : cmd;
144         } else {
145                 enc = cmd;
146         }

148         /* Issue command */
149         EFX_POPULATE_OWORD_6(oword, FRF_AB_EE_SPI_HCMD_CMD_EN, 1,
150             FRF_AB_EE_SPI_HCMD_SF_SEL, sf_sel,
151             FRF_AB_EE_SPI_HCMD_DABCNT, dabcnt,
152             FRF_AB_EE_SPI_HCMD_DUBCNT, 0,
153             FRF_AB_EE_SPI_HCMD_ADBCNT, adbcnt,
154             FRF_AB_EE_SPI_HCMD_ENC, enc);

156         EFX_SET_OWORD_FIELD(oword,
157             FRF_AB_EE_SPI_HCMD_READ,
158             (cmd == SPI_CMD_RDSR || cmd == SPI_CMD_READ) ? 1 : 0);

160         EFX_BAR_WRITEO(enp, FR_AB_EE_SPI_HCMD_REG, &oword);

162         /* Wait for read to complete */
163         if ((rc = falcon_spi_wait(enp)) != 0)
164                 goto fail2;

166         /* Read the data register */
167         if (cmd == SPI_CMD_RDSR || cmd == SPI_CMD_READ) {
168                 EFX_BAR_READO(enp, FR_AB_EE_SPI_HDATA_REG, &oword);
169                 EFSYS_ASSERT(base != NULL);
170                 FALCON_SPI_COPY(base, &oword, dabcnt);
171         }

173         return (0);

175 fail2:
176         EFSYS_PROBE(fail2);
177 fail1:
178         EFSYS_PROBE1(fail1, int, rc);

180         return (rc);
181 }

183 static  __checkReturn   int
184 falcon_spi_dev_wait(
185         __in            efx_nic_t *enp,
186         __in            falcon_spi_dev_t *fsdp,
187         __in            unsigned int us,
188         __in            unsigned int n)
189 {
190         unsigned int count;
191         int rc;

193         count = 0;
```



```
194         do {
195                 efx_byte_t byte;

197                 EFSYS_PROBE1(wait, unsigned int, count);

199                 if ((rc = falcon_spi_cmd(enp, fsdp->fsd_sf_sel,
200                     fsdp->fsd_adbcnt, SPI_CMD_RDSR, 0, fsdp->fsd_munge,
201                     (caddr_t)&byte, sizeof (efx_byte_t))) != 0)
202                         goto fail1;

204                 if (EFX_BYTE_FIELD(byte, SPI_STATUS_NRDY) == 0)
205                         goto done;

207                 EFSYS_SPIN(us);
208         } while (++count < n);

210         rc = ETIMEDOUT;
211         goto fail2;

213 done:
214         return (0);

216 fail2:
217         EFSYS_PROBE(fail2);
218 fail1:
219         EFSYS_PROBE1(fail1, int, rc);

221         return (rc);
222 }

224         __checkReturn           int
225 falcon_spi_dev_read(
226         __in                    efx_nic_t *enp,
227         __in                    falcon_spi_type_t type,
228         __in                    uint32_t addr,
229         __out_bcount(size)      caddr_t base,
230         __in                    size_t size)
231 {
232         falcon_spi_dev_t *fsdp = &(enp->en_u.falcon.enu_fsd[type]);
233         uint32_t end = addr + size;
234         int state;
235         int rc;

237         EFSYS_ASSERT3U(type, <, FALCON_SPI_NTYPES);

239         EFSYS_ASSERT3U(enp->en_family, ==, EFX_FAMILY_FALCON);

241         if (fsdp == NULL) {
242                 rc = ENODEV;
243                 goto fail1;
244         }

246         EFSYS_LOCK(enp->en_eslp, state);

248         while (addr != end) {
249                 uint32_t dabcnt = MIN(end - addr, sizeof (efx_oword_t));

251                 if ((rc = falcon_spi_cmd(enp, fsdp->fsd_sf_sel,
252                     fsdp->fsd_adbcnt, SPI_CMD_READ, addr, fsdp->fsd_munge,
253                     base, dabcnt)) != 0)
254                         goto fail2;

256                 EFSYS_ASSERT3U(addr, <, end);
257                 addr += dabcnt;
258                 base += dabcnt;
259         }
```

```
261         EFSYS_UNLOCK(enp->en_eslp, state);
262         return (0);

264 fail2:
265         EFSYS_PROBE(fail2);

267         EFSYS_UNLOCK(enp->en_eslp, state);

269 fail1:
270         EFSYS_PROBE1(fail1, int, rc);

272         return (rc);
273 }

275         __checkReturn           int
276 falcon_spi_dev_write(
277         __in                    efx_nic_t *enp,
278         __in                    falcon_spi_type_t type,
279         __in                    uint32_t addr,
280         __in_bcount(size)       caddr_t base,
281         __in                    size_t size)
282 {
283         falcon_spi_dev_t *fsdp = &(enp->en_u.falcon.enu_fsd[type]);
284         uint32_t end = addr + size;
285         int state;
286         int rc;

288         EFSYS_ASSERT3U(type, <, FALCON_SPI_NTYPES);

290         EFSYS_ASSERT3U(enp->en_family, ==, EFX_FAMILY_FALCON);

292         if (fsdp == NULL) {
293                 rc = ENODEV;
294                 goto fail1;
295         }

297         EFSYS_LOCK(enp->en_eslp, state);

299         while (addr != end) {
300                 efx_oword_t oword;
301                 uint32_t dabcnt;
302                 unsigned int index;

304                 if ((rc = falcon_spi_cmd(enp, fsdp->fsd_sf_sel,
305                             fsdp->fsd_adbcnt, SPI_CMD_WREN, 0, fsdp->fsd_munge,
306                             NULL, 0)) != 0)
307                         goto fail2;

309                 dabcnt = MIN(end - addr, fsdp->fsd_write_size -
310                     (addr & (fsdp->fsd_write_size - 1)));
311                 dabcnt = MIN(dabcnt, sizeof (efx_oword_t));

313                 if ((rc = falcon_spi_cmd(enp, fsdp->fsd_sf_sel,
314                             fsdp->fsd_adbcnt, SPI_CMD_WRITE, addr,
315                             fsdp->fsd_munge, base, dabcnt)) != 0)
316                         goto fail3;

318                 if ((rc = falcon_spi_dev_wait(enp, fsdp, 1000, 20)) != 0)
319                         goto fail4;

321                 if ((rc = falcon_spi_cmd(enp, fsdp->fsd_sf_sel,
322                             fsdp->fsd_adbcnt, SPI_CMD_READ, addr,
323                             fsdp->fsd_munge, (caddr_t)&oword, dabcnt)) != 0)
324                         goto fail5;
```



```
326                 for (index = 0; index < dabcnt; index++) {
327                         if (oword.eo_u8[index] != *(uint8_t *)(base + index)) {
328                                 rc = EIO;
329                                 goto fail6;
330                         }
331                 }

333                 EFSYS_ASSERT3U(addr, <, end);
334                 addr += dabcnt;
335                 base += dabcnt;
336         }

338         EFSYS_UNLOCK(enp->en_eslp, state);
339         return (0);

341 fail6:
342         EFSYS_PROBE(fail6);
343 fail5:
344         EFSYS_PROBE(fail5);
345 fail4:
346         EFSYS_PROBE(fail4);
347 fail3:
348         EFSYS_PROBE(fail3);
349 fail2:
350         EFSYS_PROBE(fail2);

352         EFSYS_UNLOCK(enp->en_eslp, state);
353 fail1:
354         EFSYS_PROBE1(fail1, int, rc);

356         return (rc);
357 }

359         __checkReturn   int
360 falcon_spi_dev_erase(
361         __in            efx_nic_t *enp,
362         __in            falcon_spi_type_t type,
363         __in            uint32_t addr,
364         __in            size_t size)
365 {
366         falcon_spi_dev_t *fsdp = &(enp->en_u.falcon.enu_fsd[type]);
367         uint32_t end = addr + size;
368         int state;
369         int rc;

371         EFSYS_ASSERT3U(type, <, FALCON_SPI_NTYPES);

373         EFSYS_ASSERT3U(enp->en_family, ==, EFX_FAMILY_FALCON);

375         if (fsdp == NULL) {
376                 rc = ENODEV;
377                 goto fail1;
378         }

380         if (fsdp->fsd_erase_cmd == 0) {
381                 rc = ENOTSUP;
382                 goto fail2;
383         }

385         if (!IS_P2ALIGNED(addr, fsdp->fsd_erase_size) ||
386             !IS_P2ALIGNED(size, fsdp->fsd_erase_size)) {
387                 rc = EINVAL;
388                 goto fail3;
389         }

391         EFSYS_LOCK(enp->en_eslp, state);
```

```
393         while (addr != end) {
394                 if ((rc = falcon_spi_cmd(enp, fsdp->fsd_sf_sel,
395                             fsdp->fsd_adbcnt, SPI_CMD_WREN, 0, fsdp->fsd_munge,
396                             NULL, 0)) != 0)
397                         goto fail4;

399                 if ((rc = falcon_spi_cmd(enp, fsdp->fsd_sf_sel,
400                             fsdp->fsd_adbcnt, fsdp->fsd_erase_cmd, addr,
401                             fsdp->fsd_munge, NULL, 0)) != 0)
402                         goto fail5;

404                 if ((rc = falcon_spi_dev_wait(enp, fsdp, 40000, 100)) != 0)
405                         goto fail6;

407                 EFSYS_ASSERT3U(addr, <, end);
408                 addr += fsdp->fsd_erase_size;
409         }

411         EFSYS_UNLOCK(enp->en_eslp, state);
412         return (0);

414 fail6:
415         EFSYS_PROBE(fail6);
416 fail5:
417         EFSYS_PROBE(fail5);
418 fail4:
419         EFSYS_PROBE(fail4);

421         EFSYS_UNLOCK(enp->en_eslp, state);

423 fail3:
424         EFSYS_PROBE(fail3);
425 fail2:
426         EFSYS_PROBE(fail2);
427 fail1:
428         EFSYS_PROBE1(fail1, int, rc);

430         return (rc);
431 }

433 #endif  /* EFSYS_OPT_FALCON */
434 #endif /* ! codereview */
```

```
**********************************************************
    6602 Thu Aug 22 18:59:24 2013
new/usr/src/uts/common/io/sfxge/falcon_sram.c
Merged sfxge driver
**********************************************************
/*
 * Copyright 2007-2013 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#include "efsys.h"
#include "efx.h"
#include "efx_types.h"
#include "efx_regs.h"
#include "efx_impl.h"

#if EFSYS_OPT_FALCON

static  __checkReturn   int
falcon_sram_reset(
        __in            efx_nic_t *enp)
{
        efx_oword_t oword;
        unsigned int count;
        int rc;

        /* Initiate SRAM reset */
        EFX_POPULATE_OWORD_3(oword,
                             FRF_AZ_SRM_NUM_BANK,
                             enp->en_u.falcon.enu_sram_num_bank,
                             FRF_AZ_SRM_BANK_SIZE,
                             enp->en_u.falcon.enu_sram_bank_size,
                             FRF_AZ_SRM_INIT_EN, 1);
        EFX_BAR_WRITEO(enp, FR_AZ_SRM_CFG_REG, &oword);

        /* Wait for SRAM reset to complete */
        count = 0;
        do {
                EFSYS_PROBE1(wait, unsigned int, count);

                /* Spin for 1 ms */
                EFSYS_SPIN(1000);

                /* Check for reset complete */
                EFX_BAR_READO(enp, FR_AZ_SRM_CFG_REG, &oword);
                if (EFX_OWORD_FIELD(oword, FRF_AZ_SRM_INIT_EN) == 0)
```



```
                        goto done;
        } while (++count < 100);

        rc = ETIMEDOUT;
        goto fail1;

done:
        return (0);

fail1:
        EFSYS_PROBE1(fail1, int, rc);

        return (rc);
}

        __checkReturn   int
falcon_sram_init(
        __in            efx_nic_t *enp)
{
        efx_nic_cfg_t *encp = &(enp->en_nic_cfg);
        efx_oword_t oword;
        uint32_t tx_base, rx_base;
        int rc;

        EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
        EFSYS_ASSERT3U(enp->en_family, ==, EFX_FAMILY_FALCON);

        /* Position the descriptor caches */
        if (enp->en_u.falcon.enu_internal_sram) {
                tx_base = 0x130000;
                rx_base = 0x100000;
        } else {
                rx_base = encp->enc_buftbl_limit * 8;
                tx_base = rx_base + (encp->enc_rxq_limit * 64 * 8);
        }

        /* Select internal SRAM */
        EFX_BAR_READO(enp, FR_AB_NIC_STAT_REG, &oword);
        EFX_SET_OWORD_FIELD(oword, FRF_AB_ONCHIP_SRAM,
            enp->en_u.falcon.enu_internal_sram ? 1 : 0);
        EFX_BAR_WRITEO(enp, FR_AB_NIC_STAT_REG, &oword);

        /* Sleep/Wake any external SRAM */
        EFX_BAR_READO(enp, FR_AB_GPIO_CTL_REG, &oword);
        EFX_SET_OWORD_FIELD(oword, FRF_AB_GPIO1_OEN, 1);
        EFX_SET_OWORD_FIELD(oword, FRF_AB_GPIO1_OUT,
            enp->en_u.falcon.enu_internal_sram ? 1 : 0);
        EFX_BAR_WRITEO(enp, FR_AB_GPIO_CTL_REG, &oword);

        /* Clear the SRAM contents */
        if ((rc = falcon_sram_reset(enp)) != 0)
                goto fail1;

        /* Initialize the buffer table */
        EFX_POPULATE_OWORD_1(oword, FRF_AZ_BUF_TBL_MODE, 1);
        EFX_BAR_WRITEO(enp, FR_AZ_BUF_TBL_CFG_REG, &oword);

        /* Initialize the transmit descriptor cache */
        EFX_POPULATE_OWORD_1(oword, FRF_AZ_SRM_TX_DC_BASE_ADR, tx_base);
        EFX_BAR_WRITEO(enp, FR_AZ_SRM_TX_DC_CFG_REG, &oword);

        EFX_POPULATE_OWORD_1(oword, FRF_AZ_TX_DC_SIZE, 1); /* 16 descriptors */
        EFX_BAR_WRITEO(enp, FR_AZ_TX_DC_CFG_REG, &oword);

        /* Initialize the receive descriptor cache */
        EFX_POPULATE_OWORD_1(oword, FRF_AZ_SRM_RX_DC_BASE_ADR, rx_base);
```

```
128         EFX_BAR_WRITEO(enp, FR_AZ_SRM_RX_DC_CFG_REG, &oword);

130         EFX_POPULATE_OWORD_1(oword, FRF_AZ_RX_DC_SIZE, 3); /* 64 descriptors */
131         EFX_BAR_WRITEO(enp, FR_AZ_RX_DC_CFG_REG, &oword);

133         /* Set receive descriptor pre-fetch low water mark */
134         EFX_POPULATE_OWORD_1(oword, FRF_AZ_RX_DC_PF_LWM, 56);
135         EFX_BAR_WRITEO(enp, FR_AZ_RX_DC_PF_WM_REG, &oword);

137         /* Set the event queue to use for SRAM updates */
138         EFX_POPULATE_OWORD_1(oword, FRF_AZ_SRM_UPD_EVQ_ID, 0);
139         EFX_BAR_WRITEO(enp, FR_AZ_SRM_UPD_EVQ_REG, &oword);

141         return (0);

143 fail1:
144         EFSYS_PROBE1(fail1, int, rc);

146         return (rc);
147 }

149 #if EFSYS_OPT_DIAG

151         __checkReturn   int
152 falcon_sram_test(
153         __in            efx_nic_t *enp,
154         __in            efx_sram_pattern_fn_t func)
155 {
156         efx_qword_t qword;
157         efx_qword_t verify;
158         size_t rows = 0x1000;
159         unsigned int rptr;
160         unsigned int wptr;
161         int rc;

163         EFSYS_ASSERT3U(enp->en_family, ==, EFX_FAMILY_FALCON);

165         /*
166          * Write the pattern through the SRAM debug aperture. Write
167          * in 64 entry batches, waiting 1us in between each batch
168          * to guarantee not to overflow the SRAM fifo
169          */
170         for (wptr = 0, rptr = 0; wptr < rows; ++wptr) {
171                 func(wptr, B_FALSE, &qword);
172                 EFX_BAR_TBL_WRITEQ(enp, FR_AZ_SRM_DBG_REG, wptr, &qword);

174                 if ((wptr - rptr) < 64 && wptr < rows - 1)
175                         continue;

177                 EFSYS_SPIN(1);

179                 for (; rptr <= wptr; ++rptr) {
180                         func(rptr, B_FALSE, &qword);
181                         EFX_BAR_TBL_READQ(enp, FR_AZ_SRM_DBG_REG, rptr,
182                             &verify);

184                         if (!EFX_QWORD_IS_EQUAL(verify, qword)) {
185                                 rc = EFAULT;
186                                 goto fail1;
187                         }
188                 }
189         }

191         /* And do the same negated */
192         for (wptr = 0, rptr = 0; wptr < rows; ++wptr) {
193                 func(wptr, B_TRUE, &qword);
```



```
194                 EFX_BAR_TBL_WRITEQ(enp, FR_AZ_SRM_DBG_REG, wptr, &qword);

196                 if ((wptr - rptr) < 64 && wptr < rows - 1)
197                         continue;

199                 EFSYS_SPIN(1);

201                 for (; rptr <= wptr; ++rptr) {
202                         func(rptr, B_TRUE, &qword);
203                         EFX_BAR_TBL_READQ(enp, FR_AZ_SRM_DBG_REG, rptr,
204                             &verify);

206                         if (!EFX_QWORD_IS_EQUAL(verify, qword)) {
207                                 rc = EFAULT;
208                                 goto fail2;
209                         }
210                 }
211         }

213         return (0);

215 fail2:
216         EFSYS_PROBE(fail2);
217 fail1:
218         EFSYS_PROBE1(fail1, int, rc);

220         return (rc);
221 }

223 #endif  /* EFSYS_OPT_DIAG */

225                 void
226 falcon_sram_fini(
227         __in    efx_nic_t *enp)
228 {
229         efx_oword_t oword;

231         EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
232         EFSYS_ASSERT3U(enp->en_family, ==, EFX_FAMILY_FALCON);

234         EFSYS_ASSERT(!(enp->en_mod_flags & EFX_MOD_EV));
235         EFSYS_ASSERT(!(enp->en_mod_flags & EFX_MOD_RX));
236         EFSYS_ASSERT(!(enp->en_mod_flags & EFX_MOD_TX));

238         /* Allow the GPIO1 pin to float */
239         EFX_BAR_READO(enp, FR_AB_GPIO_CTL_REG, &oword);
240         EFX_SET_OWORD_FIELD(oword, FRF_AB_GPIO1_OEN, 0);
241         EFX_BAR_WRITEO(enp, FR_AB_GPIO_CTL_REG, &oword);
242 }

244 #endif  /* EFS_OPT_FALCON */
245 #endif /* ! codereview */
```

```
**********************************************************
    8895 Thu Aug 22 18:59:24 2013
new/usr/src/uts/common/io/sfxge/falcon_stats.h
Merged sfxge driver
**********************************************************
```

```
/*-
 * Copyright 2008-2013 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#ifndef _SYS_FALCON_STATS_H
#define _SYS_FALCON_STATS_H

#ifdef  __cplusplus
extern "C" {
#endif

/*
 * Falcon GMAC stats
 */
#define G_RX_GOOD_OCT_OFFSET 0x0
#define G_RX_GOOD_OCT_WIDTH 6
#define G_RX_BAD_OCT_OFFSET 0x8
#define G_RX_BAD_OCT_WIDTH 6
#define G_RX_MISS_PKT_OFFSET 0x10
#define G_RX_MISS_PKT_WIDTH 4
#define G_RX_FALSE_CRS_OFFSET 0x14
#define G_RX_FALSE_CRS_WIDTH 4
#define G_RX_PAUSE_PKT_OFFSET 0x18
#define G_RX_PAUSE_PKT_WIDTH 4
#define G_RX_BAD_PKT_OFFSET 0x1C
#define G_RX_BAD_PKT_WIDTH 4
#define G_RX_UCAST_PKT_OFFSET 0x20
#define G_RX_UCAST_PKT_WIDTH 4
#define G_RX_MCAST_PKT_OFFSET 0x24
#define G_RX_MCAST_PKT_WIDTH 4
#define G_RX_BCAST_PKT_OFFSET 0x28
#define G_RX_BCAST_PKT_WIDTH 4
#define G_RX_GOOD_LT_64_PKT_OFFSET 0x2C
#define G_RX_GOOD_LT_64_PKT_WIDTH 4
#define G_RX_BAD_LT_64_PKT_OFFSET 0x30
#define G_RX_BAD_LT_64_PKT_WIDTH 4
#define G_RX_64_PKT_OFFSET 0x34
#define G_RX_64_PKT_WIDTH 4
#define G_RX_65_TO_127_PKT_OFFSET 0x38
#define G_RX_65_TO_127_PKT_WIDTH 4
```



```
#define G_RX_128_TO_255_PKT_OFFSET 0x3C
#define G_RX_128_TO_255_PKT_WIDTH 4
#define G_RX_256_TO_511_PKT_OFFSET 0x40
#define G_RX_256_TO_511_PKT_WIDTH 4
#define G_RX_512_TO_1023_PKT_OFFSET 0x44
#define G_RX_512_TO_1023_PKT_WIDTH 4
#define G_RX_1024_TO_15XX_PKT_OFFSET 0x48
#define G_RX_1024_TO_15XX_PKT_WIDTH 4
#define G_RX_15XX_TO_JUMBO_PKT_OFFSET 0x4C
#define G_RX_15XX_TO_JUMBO_PKT_WIDTH 4
#define G_RX_GT_JUMBO_PKT_OFFSET 0x50
#define G_RX_GT_JUMBO_PKT_WIDTH 4
#define G_RX_FCS_ERR_64_TO_15XX_PKT_OFFSET 0x54
#define G_RX_FCS_ERR_64_TO_15XX_PKT_WIDTH 4
#define G_RX_FCS_ERR_15XX_TO_JUMBO_PKT_OFFSET 0x58
#define G_RX_FCS_ERR_15XX_TO_JUMBO_PKT_WIDTH 4
#define G_RX_FCS_ERR_GT_JUMBO_PKT_OFFSET 0x5C
#define G_RX_FCS_ERR_GT_JUMBO_PKT_WIDTH 4

#define G_TX_GOOD_BAD_OCT_OFFSET 0x80
#define G_TX_GOOD_BAD_OCT_WIDTH 6
#define G_TX_GOOD_OCT_OFFSET 0x88
#define G_TX_GOOD_OCT_WIDTH 6
#define G_TX_SGL_COL_PKT_OFFSET 0x90
#define G_TX_SGL_COL_PKT_WIDTH 4
#define G_TX_MULT_COL_PKT_OFFSET 0x94
#define G_TX_MULT_COL_PKT_WIDTH 4
#define G_TX_EX_COL_PKT_OFFSET 0x98
#define G_TX_EX_COL_PKT_WIDTH 4
#define G_TX_DEF_PKT_OFFSET 0x9C
#define G_TX_DEF_PKT_WIDTH 4
#define G_TX_LATE_COL_OFFSET 0xA0
#define G_TX_LATE_COL_WIDTH 4
#define G_TX_EX_DEF_PKT_OFFSET 0xA4
#define G_TX_EX_DEF_PKT_WIDTH 4
#define G_TX_PAUSE_PKT_OFFSET 0xA8
#define G_TX_PAUSE_PKT_WIDTH 4
#define G_TX_BAD_PKT_OFFSET 0xAC
#define G_TX_BAD_PKT_WIDTH 4
#define G_TX_UCAST_PKT_OFFSET 0xB0
#define G_TX_UCAST_PKT_WIDTH 4
#define G_TX_MCAST_PKT_OFFSET 0xB4
#define G_TX_MCAST_PKT_WIDTH 4
#define G_TX_BCAST_PKT_OFFSET 0xB8
#define G_TX_BCAST_PKT_WIDTH 4
#define G_TX_LT_64_PKT_OFFSET 0xBC
#define G_TX_LT_64_PKT_WIDTH 4
#define G_TX_64_PKT_OFFSET 0xC0
#define G_TX_64_PKT_WIDTH 4
#define G_TX_65_TO_127_PKT_OFFSET 0xC4
#define G_TX_65_TO_127_PKT_WIDTH 4
#define G_TX_128_TO_255_PKT_OFFSET 0xC8
#define G_TX_128_TO_255_PKT_WIDTH 4
#define G_TX_256_TO_511_PKT_OFFSET 0xCC
#define G_TX_256_TO_511_PKT_WIDTH 4
#define G_TX_512_TO_1023_PKT_OFFSET 0xD0
#define G_TX_512_TO_1023_PKT_WIDTH 4
#define G_TX_1024_TO_15XX_PKT_OFFSET 0xD4
#define G_TX_1024_TO_15XX_PKT_WIDTH 4
#define G_TX_15XX_TO_JUMBO_PKT_OFFSET 0xD8
#define G_TX_15XX_TO_JUMBO_PKT_WIDTH 4
#define G_TX_GT_JUMBO_PKT_OFFSET 0xDC
#define G_TX_GT_JUMBO_PKT_WIDTH 4
#define G_TX_NON_TCP_UDP_PKT_OFFSET 0xE0
#define G_TX_NON_TCP_UDP_PKT_WIDTH 2
#define G_TX_MAC_SRC_ERR_PKT_OFFSET 0xE4
```

```
#define G_TX_MAC_SRC_ERR_PKT_WIDTH 2
#define G_TX_IP_SRC_ERR_PKT_OFFSET 0xE8
#define G_TX_IP_SRC_ERR_PKT_WIDTH 2

#define G_DMA_DONE_OFFSET 0xEC
#define G_DMA_DONE_WIDTH 4

#define G_STAT_OFFSET(_id) G_ ## _id ## _OFFSET
#define G_STAT_WIDTH(_id) G_ ## _id ## _WIDTH

/*
 * Falcon XMAC stats
 */
#define XG_RX_OCTETS_OFFSET 0x0
#define XG_RX_OCTETS_WIDTH 6
#define XG_RX_OCTETS_OK_OFFSET 0x8
#define XG_RX_OCTETS_OK_WIDTH 6
#define XG_RX_PKTS_OFFSET 0x10
#define XG_RX_PKTS_WIDTH 4
#define XG_RX_PKTS_OK_OFFSET 0x14
#define XG_RX_PKTS_OK_WIDTH 4
#define XG_RX_BROADCAST_PKTS_OFFSET 0x18
#define XG_RX_BROADCAST_PKTS_WIDTH 4
#define XG_RX_MULTICAST_PKTS_OFFSET 0x1C
#define XG_RX_MULTICAST_PKTS_WIDTH 4
#define XG_RX_UNICAST_PKTS_OFFSET 0x20
#define XG_RX_UNICAST_PKTS_WIDTH 4
#define XG_RX_UNDERSIZE_PKTS_OFFSET 0x24
#define XG_RX_UNDERSIZE_PKTS_WIDTH 4
#define XG_RX_OVERSIZE_PKTS_OFFSET 0x28
#define XG_RX_OVERSIZE_PKTS_WIDTH 4
#define XG_RX_JABBER_PKTS_OFFSET 0x2C
#define XG_RX_JABBER_PKTS_WIDTH 4
#define XG_RX_UNDERSIZE_FCS_ERROR_PKTS_OFFSET 0x30
#define XG_RX_UNDERSIZE_FCS_ERROR_PKTS_WIDTH 4
#define XG_RX_DROP_EVENTS_OFFSET 0x34
#define XG_RX_DROP_EVENTS_WIDTH 4
#define XG_RX_FCS_ERROR_PKTS_OFFSET 0x38
#define XG_RX_FCS_ERROR_PKTS_WIDTH 4
#define XG_RX_ALIGN_ERROR_OFFSET 0x3C
#define XG_RX_ALIGN_ERROR_WIDTH 4
#define XG_RX_SYMBOL_ERROR_OFFSET 0x40
#define XG_RX_SYMBOL_ERROR_WIDTH 4
#define XG_RX_INTERNAL_MAC_ERROR_OFFSET 0x44
#define XG_RX_INTERNAL_MAC_ERROR_WIDTH 4
#define XG_RX_CONTROL_PKTS_OFFSET 0x48
#define XG_RX_CONTROL_PKTS_WIDTH 4
#define XG_RX_PAUSE_PKTS_OFFSET 0x4C
#define XG_RX_PAUSE_PKTS_WIDTH 4
#define XG_RX_PKTS_64_OCTETS_OFFSET 0x50
#define XG_RX_PKTS_64_OCTETS_WIDTH 4
#define XG_RX_PKTS_65_TO_127_OCTETS_OFFSET 0x54
#define XG_RX_PKTS_65_TO_127_OCTETS_WIDTH 4
#define XG_RX_PKTS_128_TO_255_OCTETS_OFFSET 0x58
#define XG_RX_PKTS_128_TO_255_OCTETS_WIDTH 4
#define XG_RX_PKTS_256_TO_511_OCTETS_OFFSET 0x5C
#define XG_RX_PKTS_256_TO_511_OCTETS_WIDTH 4
#define XG_RX_PKTS_512_TO_1023_OCTETS_OFFSET 0x60
#define XG_RX_PKTS_512_TO_1023_OCTETS_WIDTH 4
#define XG_RX_PKTS_1024_TO_15XX_OCTETS_OFFSET 0x64
#define XG_RX_PKTS_1024_TO_15XX_OCTETS_WIDTH 4
#define XG_RX_PKTS_15XX_TO_MAX_OCTETS_OFFSET 0x68
#define XG_RX_PKTS_15XX_TO_MAX_OCTETS_WIDTH 4
#define XG_RX_LENGTH_ERROR_OFFSET 0x6C
#define XG_RX_LENGTH_ERROR_WIDTH 4
```



```
#define XG_TX_PKTS_OFFSET 0x80
#define XG_TX_PKTS_WIDTH 4
#define XG_TX_OCTETS_OFFSET 0x88
#define XG_TX_OCTETS_WIDTH 6
#define XG_TX_MULTICAST_PKTS_OFFSET 0x90
#define XG_TX_MULTICAST_PKTS_WIDTH 4
#define XG_TX_BROADCAST_PKTS_OFFSET 0x94
#define XG_TX_BROADCAST_PKTS_WIDTH 4
#define XG_TX_UNICAST_PKTS_OFFSET 0x98
#define XG_TX_UNICAST_PKTS_WIDTH 4
#define XG_TX_CONTROL_PKTS_OFFSET 0x9C
#define XG_TX_CONTROL_PKTS_WIDTH 4
#define XG_TX_PAUSE_PKTS_OFFSET 0xA0
#define XG_TX_PAUSE_PKTS_WIDTH 4
#define XG_TX_PKTS_64_OCTETS_OFFSET 0xA4
#define XG_TX_PKTS_64_OCTETS_WIDTH 4
#define XG_TX_PKTS_65_TO_127_OCTETS_OFFSET 0xA8
#define XG_TX_PKTS_65_TO_127_OCTETS_WIDTH 4
#define XG_TX_PKTS_128_TO_255_OCTETS_OFFSET 0xAC
#define XG_TX_PKTS_128_TO_255_OCTETS_WIDTH 4
#define XG_TX_PKTS_256_TO_511_OCTETS_OFFSET 0xB0
#define XG_TX_PKTS_256_TO_511_OCTETS_WIDTH 4
#define XG_TX_PKTS_512_TO_1023_OCTETS_OFFSET 0xB4
#define XG_TX_PKTS_512_TO_1023_OCTETS_WIDTH 4
#define XG_TX_PKTS_1024_TO_15XX_OCTETS_OFFSET 0xB8
#define XG_TX_PKTS_1024_TO_15XX_OCTETS_WIDTH 4
#define XG_TX_PKTS_15XX_TO_MAX_OCTETS_OFFSET 0xBC
#define XG_TX_PKTS_15XX_TO_MAX_OCTETS_WIDTH 4
#define XG_TX_UNDERSIZE_PKTS_OFFSET 0xC0
#define XG_TX_UNDERSIZE_PKTS_WIDTH 4
#define XG_TX_OVERSIZE_PKTS_OFFSET 0xC4
#define XG_TX_OVERSIZE_PKTS_WIDTH 4
#define XG_TX_NON_TCP_UDP_PKTS_OFFSET 0xC8
#define XG_TX_NON_TCP_UDP_PKTS_WIDTH 2
#define XG_TX_MAC_SRC_ERR_PKTS_OFFSET 0xCC
#define XG_TX_MAC_SRC_ERR_PKTS_WIDTH 2
#define XG_TX_IP_SRC_ERR_PKTS_OFFSET 0xD0
#define XG_TX_IP_SRC_ERR_PKTS_WIDTH 2

#define XG_DMA_DONE_OFFSET 0xD4
#define XG_DMA_DONE_WIDTH 4

#define DMA_IS_DONE 0xffffffff

#define XG_STAT_OFFSET(_id) XG_ ## _id ## _OFFSET
#define XG_STAT_WIDTH(_id) XG_ ## _id ## _WIDTH

#ifdef  __cplusplus
}
#endif

#endif  /* _SYS_FALCON_STATS_H */
#endif /* ! codereview */
```

```
**********************************************************
    6725 Thu Aug 22 18:59:24 2013
new/usr/src/uts/common/io/sfxge/falcon_vpd.c
Merged sfxge driver
**********************************************************
```

```
/*
 * Copyright 2009 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

/*
 * Falcon conviniently uses an EEPROM to store it's VPD configuration,
 * and it stores the VPD contents in native VPD format. This code does
 * not cope with the presence of an RW block in VPD at all.
 */

#include "efsys.h"
#include "efx.h"
#include "falcon_nvram.h"
#include "efx_types.h"
#include "efx_impl.h"

#if EFSYS_OPT_FALCON

#if EFSYS_OPT_VPD

typedef struct {
        size_t          fvpdd_base;
        size_t          fvpdd_size;
        boolean_t       fvpdd_writable;
} falcon_vpd_dimension_t;

static                          void
falcon_vpd_dimension(
        __in                    efx_nic_t *enp,
        __out                   falcon_vpd_dimension_t *dimp)
{
        efx_oword_t oword;

#if EFSYS_OPT_FALCON_NIC_CFG_OVERRIDE
        if (enp->en_u.falcon.enu_forced_cfg != NULL) {
                memcpy(&oword, (enp->en_u.falcon.enu_forced_cfg
                                + EE_VPD_CFG0_REG_SF_OFST), sizeof (oword));
        }
        else
#endif  /* EFSYS_OPT_FALCON_NIC_CFG_OVERRIDE */
        {
                EFX_BAR_READO(enp, FR_AB_EE_VPD_CFG0_REG, &oword);
        }

        dimp->fvpdd_base = EFX_OWORD_FIELD(oword, FRF_AB_EE_VPD_BASE);
        dimp->fvpdd_size = EFX_OWORD_FIELD(oword, FRF_AB_EE_VPD_LENGTH);
        if (dimp->fvpdd_size != 0)
                /* Non-zero "lengths" are actually maximum dword offsets */
                dimp->fvpdd_size += 4;
        dimp->fvpdd_writable =
                EFX_OWORD_FIELD(oword, FRF_AB_EE_VPDW_LENGTH) != 0;
}

        __checkReturn           int
falcon_vpd_size(
        __in                    efx_nic_t *enp,
        __out                   size_t *sizep)
{
        falcon_vpd_dimension_t dim;
        int rc;

        EFSYS_ASSERT3U(enp->en_family, ==, EFX_FAMILY_FALCON);

        falcon_vpd_dimension(enp, &dim);
        if (dim.fvpdd_size == 0 || dim.fvpdd_writable) {
                rc = ENOTSUP;
                goto fail1;
        }

        *sizep = dim.fvpdd_size;

        return (0);

fail1:
        EFSYS_PROBE1(fail1, int, rc);

        *sizep = 0;

        return (rc);
}

        __checkReturn           int
falcon_vpd_read(
        __in                    efx_nic_t *enp,
        __out_bcount(size)      caddr_t data,
        __in                    size_t size)
{
        falcon_vpd_dimension_t dim;
        int rc;

        EFSYS_ASSERT3U(enp->en_family, ==, EFX_FAMILY_FALCON);

        falcon_vpd_dimension(enp, &dim);
        if (size < dim.fvpdd_size || size == 0) {
                rc = ENOTSUP;
                goto fail1;
        }

        if ((rc = falcon_spi_dev_read(enp, FALCON_SPI_EEPROM,
                    dim.fvpdd_base, data, size)) != 0)
                goto fail2;

        return (0);

fail2:
        EFSYS_PROBE(fail2);
```

```
fail1:
        EFSYS_PROBE1(fail1, int, rc);

        return (rc);
}

        __checkReturn           int
falcon_vpd_verify(
        __in                    efx_nic_t *enp,
        __in_bcount(size)       caddr_t data,
        __in                    size_t size)
{
        falcon_vpd_dimension_t dim;
        boolean_t cksummed;
        int rc;

        EFSYS_ASSERT3U(enp->en_family, ==, EFX_FAMILY_FALCON);

        falcon_vpd_dimension(enp, &dim);
        EFSYS_ASSERT3U(dim.fvpdd_size, <=, size);
        EFSYS_ASSERT(!dim.fvpdd_writable);

        if ((rc = efx_vpd_hunk_verify(data, size, &cksummed)) != 0)
                goto fail1;

        if (!cksummed) {
                rc = EFAULT;
                goto fail2;
        }

        return (0);

fail2:
        EFSYS_PROBE(fail2);
fail1:
        EFSYS_PROBE1(fail1, int, rc);

        return (rc);
}

        __checkReturn           int
falcon_vpd_get(
        __in                    efx_nic_t *enp,
        __in_bcount(size)       caddr_t data,
        __in                    size_t size,
        __inout                 efx_vpd_value_t *evvp)
{
        falcon_vpd_dimension_t dim;
        unsigned int offset;
        uint8_t length;
        int rc;

        EFSYS_ASSERT3U(enp->en_family, ==, EFX_FAMILY_FALCON);

        if (evvp->evv_tag != EFX_VPD_ID && evvp->evv_tag != EFX_VPD_RO) {
                rc = EINVAL;
                goto fail1;
        }

        falcon_vpd_dimension(enp, &dim);
        EFSYS_ASSERT3U(dim.fvpdd_size, <=, size);
        EFSYS_ASSERT(!dim.fvpdd_writable);

        if ((rc = efx_vpd_hunk_get(data, size, evvp->evv_tag,
            evvp->evv_keyword, &offset, &length)) != 0)
                goto fail2;
```



```
        /* Copy out */
        evvp->evv_length = length;
        memcpy(evvp->evv_value, data + offset, length);

        return (0);

fail2:
        EFSYS_PROBE(fail2);
fail1:
        EFSYS_PROBE1(fail1, int, rc);

        return (rc);
}

        __checkReturn           int
falcon_vpd_set(
        __in                    efx_nic_t *enp,
        __in_bcount(size)       caddr_t data,
        __in                    size_t size,
        __in                    efx_vpd_value_t *evvp)
{
        falcon_vpd_dimension_t dim;
        int rc;

        EFSYS_ASSERT3U(enp->en_family, ==, EFX_FAMILY_FALCON);

        falcon_vpd_dimension(enp, &dim);
        EFSYS_ASSERT3U(dim.fvpdd_size, <=, size);

        if ((rc = efx_vpd_hunk_set(data, size, evvp)) != 0)
                goto fail1;

fail1:
        EFSYS_PROBE1(fail1, int, rc);

        return (rc);
}

        __checkReturn           int
falcon_vpd_next(
        __in                    efx_nic_t *enp,
        __in_bcount(size)       caddr_t data,
        __in                    size_t size,
        __out                   efx_vpd_value_t *evvp,
        __inout                 unsigned int *contp)
{
        falcon_vpd_dimension_t dim;
        unsigned int offset;
        int rc;

        EFSYS_ASSERT3U(enp->en_family, ==, EFX_FAMILY_FALCON);

        falcon_vpd_dimension(enp, &dim);
        EFSYS_ASSERT3U(dim.fvpdd_size, <=, size);
        EFSYS_ASSERT(!dim.fvpdd_writable);

        /* Find the (tag, keyword) */
        if ((rc = efx_vpd_hunk_next(data, size, &evvp->evv_tag,
            &evvp->evv_keyword, &offset, &evvp->evv_length, contp)) != 0)
                goto fail1;

        /* Copyout */
        memcpy(evvp->evv_value, data + offset, evvp->evv_length);

        return (0);
```

```
fail1:
        EFSYS_PROBE1(fail1, int, rc);

        return (rc);
}

        __checkReturn           int
falcon_vpd_write(
        __in                    efx_nic_t *enp,
        __in_bcount(size)       caddr_t data,
        __in                    size_t size)
{
        falcon_vpd_dimension_t dim;
        int rc;

        EFSYS_ASSERT3U(enp->en_family, ==, EFX_FAMILY_FALCON);

        falcon_vpd_dimension(enp, &dim);
        if (dim.fvpdd_size != size) {
                /* User hasn't provided sufficient data */
                rc = EINVAL;
                goto fail1;
        }

        if ((rc = falcon_spi_dev_write(enp, FALCON_SPI_EEPROM,
                    dim.fvpdd_base, data, dim.fvpdd_size)) != 0)
                goto fail2;

        return (0);

fail2:
        EFSYS_PROBE(fail2);
fail1:
        EFSYS_PROBE1(fail1, int, rc);

        return (rc);
}

#endif  /* EFSYS_OPT_FALCON */

#endif  /* EFSYS_OPT_VPD */
#endif /* ! codereview */
```

```
**********************************************************
   16289 Thu Aug 22 18:59:24 2013
new/usr/src/uts/common/io/sfxge/falcon_xmac.c
Merged sfxge driver
**********************************************************
```

```
/*
 * Copyright 2008-2013 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#include "efsys.h"
#include "efx.h"
#include "efx_types.h"
#include "efx_regs.h"
#include "efx_impl.h"
#include "falcon_stats.h"
#include "falcon_xmac.h"

#if EFSYS_OPT_MAC_FALCON_XMAC

static  __checkReturn   int
falcon_xmac_xgxs_reset(
        __in            efx_nic_t *enp)
{
        efx_oword_t oword;
        unsigned int count;
        int rc;

        /* Reset the XGMAC via the Vendor Register */
        EFX_POPULATE_OWORD_1(oword, FRF_AB_XX_RST_XX_EN, 1);
        EFX_BAR_WRITEO(enp, FR_AB_XX_PWR_RST_REG, &oword);

        count = 0;
        do {
                EFSYS_PROBE1(wait, unsigned int, count);

                /* Spin for 10us */
                EFSYS_SPIN(10);

                /* Test for reset complete */
                EFX_BAR_READO(enp, FR_AB_XX_PWR_RST_REG, &oword);
                if (EFX_OWORD_FIELD(oword, FRF_AB_XX_RST_XX_EN) == 0 &&
                    EFX_OWORD_FIELD(oword, FRF_AB_XX_SD_RST_ACT) == 0)
                        goto done;
        } while (++count < 1000);
```



```
        rc = ETIMEDOUT;
        goto fail1;

done:
        return (0);

fail1:
        EFSYS_PROBE1(fail1, int, rc);

        return (rc);
}

static                  void
falcon_xmac_xgxs_reconfigure(
        __in            efx_nic_t *enp)
{
#if EFSYS_OPT_LOOPBACK
        efx_port_t *epp = &(enp->en_port);
#endif  /* EFSYS_OPT_LOOPBACK */
        efx_oword_t oword;
        uint32_t force_sig;
        uint32_t force_sig_val;
        uint32_t xgxs_lb_en;
        uint32_t xgmii_lb_en;
        uint32_t lpbk;
        boolean_t need_reset;

#if EFSYS_OPT_LOOPBACK
        force_sig =
            (epp->ep_loopback_type == EFX_LOOPBACK_XGXS ||
            epp->ep_loopback_type == EFX_LOOPBACK_XAUI) ?
            1 : 0;
        force_sig_val =
            (epp->ep_loopback_type == EFX_LOOPBACK_XGXS ||
            epp->ep_loopback_type == EFX_LOOPBACK_XAUI) ?
            1 : 0;
        xgxs_lb_en = (epp->ep_loopback_type == EFX_LOOPBACK_XGXS) ?
            1 : 0;
        xgmii_lb_en = (epp->ep_loopback_type == EFX_LOOPBACK_XGMII) ?
            1 : 0;
        lpbk = (epp->ep_loopback_type == EFX_LOOPBACK_XAUI) ?
            1 : 0;
#else   /* EFSYS_OPT_LOOPBACK */
        force_sig = 0;
        force_sig_val = 0;
        xgxs_lb_en = 0;
        xgmii_lb_en = 0;
        lpbk = 0;
#endif  /* EFSYS_OPT_LOOPBACK */

        EFX_BAR_READO(enp, FR_AB_XX_CORE_STAT_REG, &oword);
        need_reset =
            (EFX_OWORD_FIELD(oword, FRF_AB_XX_FORCE_SIG0) != force_sig ||
            EFX_OWORD_FIELD(oword, FRF_AB_XX_FORCE_SIG0_VAL) != force_sig_val ||
            EFX_OWORD_FIELD(oword, FRF_AB_XX_FORCE_SIG1) != force_sig ||
            EFX_OWORD_FIELD(oword, FRF_AB_XX_FORCE_SIG1_VAL) != force_sig_val ||
            EFX_OWORD_FIELD(oword, FRF_AB_XX_FORCE_SIG2) != force_sig ||
            EFX_OWORD_FIELD(oword, FRF_AB_XX_FORCE_SIG2_VAL) != force_sig_val ||
            EFX_OWORD_FIELD(oword, FRF_AB_XX_FORCE_SIG3) != force_sig ||
            EFX_OWORD_FIELD(oword, FRF_AB_XX_FORCE_SIG3_VAL) != force_sig_val ||
            EFX_OWORD_FIELD(oword, FRF_AB_XX_XGXS_LB_EN) != xgxs_lb_en ||
            EFX_OWORD_FIELD(oword, FRF_AB_XX_XGMII_LB_EN) != xgmii_lb_en);

        EFX_BAR_READO(enp, FR_AB_XX_SD_CTL_REG, &oword);
        need_reset |= (EFX_OWORD_FIELD(oword, FRF_AB_XX_LPBKA) != lpbk ||
            EFX_OWORD_FIELD(oword, FRF_AB_XX_LPBKB) != lpbk ||
```

```
128             EFX_OWORD_FIELD(oword, FRF_AB_XX_LPBKC) != lpbk ||
129             EFX_OWORD_FIELD(oword, FRF_AB_XX_LPBKD) != lpbk);

131         if (need_reset)
132                 (void) falcon_xmac_xgxs_reset(enp);

134         EFX_BAR_READO(enp, FR_AB_XX_CORE_STAT_REG, &oword);
135         EFX_SET_OWORD_FIELD(oword, FRF_AB_XX_FORCE_SIG0, force_sig);
136         EFX_SET_OWORD_FIELD(oword, FRF_AB_XX_FORCE_SIG0_VAL, force_sig_val);
137         EFX_SET_OWORD_FIELD(oword, FRF_AB_XX_FORCE_SIG1, force_sig);
138         EFX_SET_OWORD_FIELD(oword, FRF_AB_XX_FORCE_SIG1_VAL, force_sig_val);
139         EFX_SET_OWORD_FIELD(oword, FRF_AB_XX_FORCE_SIG2, force_sig);
140         EFX_SET_OWORD_FIELD(oword, FRF_AB_XX_FORCE_SIG2_VAL, force_sig_val);
141         EFX_SET_OWORD_FIELD(oword, FRF_AB_XX_FORCE_SIG3, force_sig);
142         EFX_SET_OWORD_FIELD(oword, FRF_AB_XX_FORCE_SIG3_VAL, force_sig_val);
143         EFX_SET_OWORD_FIELD(oword, FRF_AB_XX_XGXS_LB_EN, xgxs_lb_en);
144         EFX_SET_OWORD_FIELD(oword, FRF_AB_XX_XGMII_LB_EN, xgmii_lb_en);
145         EFX_BAR_WRITEO(enp, FR_AB_XX_CORE_STAT_REG, &oword);

147         EFX_BAR_READO(enp, FR_AB_XX_SD_CTL_REG, &oword);
148         EFX_SET_OWORD_FIELD(oword, FRF_AB_XX_LPBKD, lpbk);
149         EFX_SET_OWORD_FIELD(oword, FRF_AB_XX_LPBKB, lpbk);
150         EFX_SET_OWORD_FIELD(oword, FRF_AB_XX_LPBKC, lpbk);
151         EFX_SET_OWORD_FIELD(oword, FRF_AB_XX_LPBKA, lpbk);
152         EFX_BAR_WRITEO(enp, FR_AB_XX_SD_CTL_REG, &oword);
153 }

155 static                  void
156 falcon_xmac_core_reconfigure(
157         __in            efx_nic_t *enp)
158 {
159         efx_port_t *epp = &(enp->en_port);
160         efx_oword_t oword;

162         EFX_POPULATE_OWORD_3(oword,
163             FRF_AB_XM_RX_JUMBO_MODE, 1,
164             FRF_AB_XM_TX_STAT_EN, 1,
165             FRF_AB_XM_RX_STAT_EN, 1);

167         EFX_BAR_WRITEO(enp, FR_AB_XM_GLB_CFG_REG, &oword);

169         EFX_POPULATE_OWORD_6(oword,
170             FRF_AB_XM_TXEN, 1,
171             FRF_AB_XM_TX_PRMBL, 1,
172             FRF_AB_XM_AUTO_PAD, 1,
173             FRF_AB_XM_TXCRC, 1,
174             FRF_AB_XM_FCNTL, 1,
175             FRF_AB_XM_IPG, 3);

177         EFX_BAR_WRITEO(enp, FR_AB_XM_TX_CFG_REG, &oword);

179         EFX_POPULATE_OWORD_6(oword,
180             FRF_AB_XM_RXEN, 1,
181             FRF_AB_XM_AUTO_DEPAD, 0,
182             FRF_AB_XM_REJ_BCAST, (epp->ep_brdcst) ? 0 : 1,
183             FRF_AB_XM_ACPT_ALL_MCAST, 1,
184             FRF_AB_XM_ACPT_ALL_UCAST, (epp->ep_unicst) ? 1 : 0,
185             FRF_AB_XM_PASS_CRC_ERR, 1);

187         EFX_BAR_WRITEO(enp, FR_AB_XM_RX_CFG_REG, &oword);

189         EFX_POPULATE_OWORD_2(oword,
190             FRF_AB_XM_PAUSE_TIME, 0xfffe,
191             FRF_AB_XM_DIS_FCNTL, (epp->ep_fcntl != 0) ? 0 : 1);

193         EFX_BAR_WRITEO(enp, FR_AB_XM_FC_REG, &oword);
```



```
195         EFX_POPULATE_OWORD_1(oword, FRF_AB_XM_ADR_LO,
196             ((uint32_t)epp->ep_mac_addr[0] << 0) |
197             ((uint32_t)epp->ep_mac_addr[1] << 8) |
198             ((uint32_t)epp->ep_mac_addr[2] << 16) |
199             ((uint32_t)epp->ep_mac_addr[3] << 24));

201         EFX_BAR_WRITEO(enp, FR_AB_XM_ADR_LO_REG, &oword);

203         EFX_POPULATE_OWORD_1(oword, FRF_AB_XM_ADR_HI,
204             ((uint32_t)epp->ep_mac_addr[4] << 0) |
205             ((uint32_t)epp->ep_mac_addr[5] << 8));

207         EFX_BAR_WRITEO(enp, FR_AB_XM_ADR_HI_REG, &oword);

209         EFX_POPULATE_OWORD_1(oword,
210             FRF_AB_XM_MAX_RX_FRM_SIZE_HI, epp->ep_mac_pdu >> 3);

212         EFX_BAR_WRITEO(enp, FR_AB_XM_RX_PARAM_REG, &oword);

214         EFX_POPULATE_OWORD_2(oword,
215             FRF_AB_XM_MAX_TX_FRM_SIZE_HI, epp->ep_mac_pdu >> 3,
216             FRF_AB_XM_TX_JUMBO_MODE, 1);

218         EFX_BAR_WRITEO(enp, FR_AB_XM_TX_PARAM_REG, &oword);
219 }

221         __checkReturn   int
222 falcon_xmac_reset(
223         __in            efx_nic_t *enp)
224 {
225         efx_port_t *epp = &(enp->en_port);
226         efx_oword_t oword;
227         int rc;

229         EFSYS_ASSERT3U(epp->ep_mac_type, ==, EFX_MAC_FALCON_XMAC);

231         EFSYS_PROBE(reset);

233         /* Ensure that the XMAC registers are accessible */
234         EFX_BAR_READO(enp, FR_AB_NIC_STAT_REG, &oword);
235         EFX_SET_OWORD_FIELD(oword, FRF_BB_EE_STRAP_EN, 1);
236         EFX_SET_OWORD_FIELD(oword, FRF_BB_EE_STRAP, 0x5);
237         EFX_BAR_WRITEO(enp, FR_AB_NIC_STAT_REG, &oword);

239         if ((rc = falcon_mac_wrapper_disable(enp)) != 0)
240             goto fail1;

242         /*
243          * Force the PHY XAUI state machine to restart after the EM block
244          * reset. Don't do this if we're now in a MAC level loopback.
245          */
246 #if EFSYS_OPT_LOOPBACK
247         if ((1 << epp->ep_loopback_type) & ~EFX_LOOPBACK_MAC_MASK)
248 #endif
249                 if ((rc = falcon_xmac_xgxs_reset(enp)) != 0)
250                         goto fail2;

252         epp->ep_mac_poll_needed = B_TRUE;

254         enp->en_reset_flags |= EFX_RESET_MAC;

256         return (0);

258 fail2:
259         EFSYS_PROBE(fail2);
```

```
260 fail1:
261         EFSYS_PROBE1(fail1, int, rc);

263         return (rc);
264 }

266         __checkReturn   int
267 falcon_xmac_reconfigure(
268         __in            efx_nic_t *enp)
269 {
270         efx_port_t *epp = &(enp->en_port);

272         EFSYS_PROBE(reconfigure);

274         EFSYS_ASSERT3U(epp->ep_mac_type, ==, EFX_MAC_FALCON_XMAC);
275         EFSYS_ASSERT(epp->ep_link_mode == EFX_LINK_UNKNOWN ||
276                      epp->ep_link_mode == EFX_LINK_DOWN ||
277                      epp->ep_link_mode == EFX_LINK_10000FDX);

279         falcon_xmac_xgxs_reconfigure(enp);
280         falcon_xmac_core_reconfigure(enp);

282         falcon_mac_wrapper_enable(enp);

284         return (0);
285 }

287 #if EFSYS_OPT_MAC_STATS
288 static          uint32_t
289 falcon_xmac_stat_read(
290         __in    efsys_mem_t *esmp,
291         __in    unsigned int offset,
292         __in    unsigned int width)
293 {
294         uint32_t val;

296         switch (width) {
297         case 2: {
298                 efx_dword_t dword;

300                 EFSYS_MEM_READD(esmp, offset, &dword);

302                 val = (uint16_t)EFX_DWORD_FIELD(dword, EFX_WORD_0);
303                 break;
304         }
305         case 4: {
306                 efx_dword_t dword;

308                 EFSYS_MEM_READD(esmp, offset, &dword);

310                 val = EFX_DWORD_FIELD(dword, EFX_DWORD_0);
311                 break;
312         }
313         case 6: {
314                 efx_qword_t qword;

316                 EFSYS_MEM_READQ(esmp, offset, &qword);

318                 val = EFX_QWORD_FIELD(qword, EFX_DWORD_0);
319                 break;
320         }
321         default:
322                 EFSYS_ASSERT(B_FALSE);

324                 val = 0;
325                 break;
```



```
326         }

328         return (val);
329 }

331 #define XMAC_STAT_READ(_esmp, _id)                                      \
332         falcon_xmac_stat_read((_esmp), XG_STAT_OFFSET(_id),             \
333             XG_STAT_WIDTH(_id))

335 #define XMAC_STAT_INCR(_esmp, _stat, _id)                               \
336         do {                                                            \
337                 delta = XMAC_STAT_READ(_esmp, _id);                     \
338                 EFSYS_STAT_INCR(&(_stat), delta);                       \
339         _NOTE(CONSTANTCONDITION)                                        \
340         } while (0)

342 #define XMAC_STAT_DECR(_esmp, _stat, _id)                               \
343         do {                                                            \
344                 delta = XMAC_STAT_READ(_esmp, _id);                     \
345                 EFSYS_STAT_DECR(&(_stat), delta);                       \
346         _NOTE(CONSTANTCONDITION)                                        \
347         } while (0)

349         __checkReturn                   int
350 falcon_xmac_stats_update(
351         __in                            efx_nic_t *enp,
352         __in                            efsys_mem_t *esmp,
353         __in_ecount(EFX_MAC_NSTATS)     efsys_stat_t *stat,
354         __out_opt                       uint32_t *generationp)
355 {
356         efx_port_t *epp = &(enp->en_port);
357         efx_oword_t oword;
358         uint32_t delta;

360         _NOTE(ARGUNUSED(generationp));
361         EFSYS_ASSERT3U(epp->ep_mac_type, ==, EFX_MAC_FALCON_XMAC);

363         /* Check the DMA flag */
364         if (XMAC_STAT_READ(esmp, DMA_DONE) != DMA_IS_DONE)
365                 return (EAGAIN);
366         EFSYS_MEM_READ_BARRIER();

368         /* RX */
369         XMAC_STAT_INCR(esmp, stat[EFX_MAC_RX_OCTETS], RX_OCTETS);
370         XMAC_STAT_INCR(esmp, stat[EFX_MAC_RX_PKTS], RX_PKTS);
371         XMAC_STAT_INCR(esmp, stat[EFX_MAC_RX_UNICST_PKTS], RX_UNICAST_PKTS);
372         XMAC_STAT_INCR(esmp, stat[EFX_MAC_RX_MULTICST_PKTS],
373             RX_MULTICAST_PKTS);
374         XMAC_STAT_INCR(esmp, stat[EFX_MAC_RX_BRDCST_PKTS], RX_BROADCAST_PKTS);
375         XMAC_STAT_INCR(esmp, stat[EFX_MAC_RX_PAUSE_PKTS], RX_PAUSE_PKTS);
376         XMAC_STAT_INCR(esmp, stat[EFX_MAC_RX_LE_64_PKTS], RX_UNDERSIZE_PKTS);

378         XMAC_STAT_INCR(esmp, stat[EFX_MAC_RX_LE_64_PKTS], RX_PKTS_64_OCTETS);
379         XMAC_STAT_INCR(esmp, stat[EFX_MAC_RX_65_TO_127_PKTS],
380             RX_PKTS_65_TO_127_OCTETS);
381         XMAC_STAT_INCR(esmp, stat[EFX_MAC_RX_128_TO_255_PKTS],
382             RX_PKTS_128_TO_255_OCTETS);
383         XMAC_STAT_INCR(esmp, stat[EFX_MAC_RX_256_TO_511_PKTS],
384             RX_PKTS_256_TO_511_OCTETS);
385         XMAC_STAT_INCR(esmp, stat[EFX_MAC_RX_512_TO_1023_PKTS],
386             RX_PKTS_512_TO_1023_OCTETS);
387         XMAC_STAT_INCR(esmp, stat[EFX_MAC_RX_1024_TO_15XX_PKTS],
388             RX_PKTS_1024_TO_15XX_OCTETS);

390         XMAC_STAT_INCR(esmp, stat[EFX_MAC_RX_GE_15XX_PKTS],
391             RX_PKTS_15XX_TO_MAX_OCTETS);
```

```
392         XMAC_STAT_DECR(esmp, stat[EFX_MAC_RX_GE_15XX_PKTS],
393             RX_OVERSIZE_PKTS);
394
395         XMAC_STAT_INCR(esmp, stat[EFX_MAC_RX_ERRORS], RX_PKTS);
396         XMAC_STAT_DECR(esmp, stat[EFX_MAC_RX_ERRORS], RX_PKTS_OK);
397
398         XMAC_STAT_INCR(esmp, stat[EFX_MAC_RX_FCS_ERRORS],
399             RX_UNDERSIZE_FCS_ERROR_PKTS);
400         XMAC_STAT_INCR(esmp, stat[EFX_MAC_RX_FCS_ERRORS], RX_FCS_ERROR_PKTS);
401         XMAC_STAT_INCR(esmp, stat[EFX_MAC_RX_DROP_EVENTS], RX_DROP_EVENTS);
402         XMAC_STAT_INCR(esmp, stat[EFX_MAC_RX_SYMBOL_ERRORS], RX_SYMBOL_ERROR);
403         XMAC_STAT_INCR(esmp, stat[EFX_MAC_RX_ALIGN_ERRORS], RX_ALIGN_ERROR);
404         XMAC_STAT_INCR(esmp, stat[EFX_MAC_RX_INTERNAL_ERRORS],
405             RX_INTERNAL_MAC_ERROR);
406
407         EFX_BAR_READO(enp, FR_AZ_RX_NODESC_DROP_REG, &oword);
408         delta = (uint32_t)EFX_OWORD_FIELD(oword, FRF_AZ_RX_NODESC_DROP_CNT);
409         EFSYS_STAT_INCR(&(stat[EFX_MAC_RX_NODESC_DROP_CNT]), delta);
410
411         /* TX */
412         XMAC_STAT_INCR(esmp, stat[EFX_MAC_TX_OCTETS], TX_OCTETS);
413         XMAC_STAT_INCR(esmp, stat[EFX_MAC_TX_PKTS], TX_PKTS);
414         XMAC_STAT_INCR(esmp, stat[EFX_MAC_TX_UNICST_PKTS], TX_UNICAST_PKTS);
415         XMAC_STAT_INCR(esmp, stat[EFX_MAC_TX_MULTICST_PKTS],
416             TX_MULTICAST_PKTS);
417         XMAC_STAT_INCR(esmp, stat[EFX_MAC_TX_BRDCST_PKTS], TX_BROADCAST_PKTS);
418         XMAC_STAT_INCR(esmp, stat[EFX_MAC_TX_PAUSE_PKTS], TX_PAUSE_PKTS);
419
420         XMAC_STAT_INCR(esmp, stat[EFX_MAC_TX_LE_64_PKTS], TX_UNDERSIZE_PKTS);
421         XMAC_STAT_INCR(esmp, stat[EFX_MAC_TX_LE_64_PKTS], TX_PKTS_64_OCTETS);
422         XMAC_STAT_INCR(esmp, stat[EFX_MAC_TX_65_TO_127_PKTS],
423             TX_PKTS_65_TO_127_OCTETS);
424         XMAC_STAT_INCR(esmp, stat[EFX_MAC_TX_128_TO_255_PKTS],
425             TX_PKTS_128_TO_255_OCTETS);
426         XMAC_STAT_INCR(esmp, stat[EFX_MAC_TX_256_TO_511_PKTS],
427             TX_PKTS_256_TO_511_OCTETS);
428         XMAC_STAT_INCR(esmp, stat[EFX_MAC_TX_512_TO_1023_PKTS],
429             TX_PKTS_512_TO_1023_OCTETS);
430         XMAC_STAT_INCR(esmp, stat[EFX_MAC_TX_1024_TO_15XX_PKTS],
431             TX_PKTS_1024_TO_15XX_OCTETS);
432         XMAC_STAT_INCR(esmp, stat[EFX_MAC_TX_GE_15XX_PKTS],
433             TX_PKTS_15XX_TO_MAX_OCTETS);
434         XMAC_STAT_INCR(esmp, stat[EFX_MAC_TX_GE_15XX_PKTS],
435             TX_OVERSIZE_PKTS);
436
437         XMAC_STAT_INCR(esmp, stat[EFX_MAC_TX_ERRORS], TX_MAC_SRC_ERR_PKTS);
438         XMAC_STAT_INCR(esmp, stat[EFX_MAC_TX_ERRORS], TX_IP_SRC_ERR_PKTS);
439
440         return (0);
441 }
442 #endif  /* EFSYS_OPT_MAC_STATS */
443
444 static                  void
445 falcon_xmac_downlink_check(
446         __in            efx_nic_t *enp,
447         __out           boolean_t *upp)
448 {
449         efx_port_t *epp = &(enp->en_port);
450         efx_oword_t oword;
451         boolean_t align_done;
452         boolean_t sync_stat;
453
454         EFSYS_ASSERT3U(epp->ep_mac_type, ==, EFX_MAC_FALCON_XMAC);
455
456         /* XGXS state irrelevent in XGMII loopback */
457 #if EFSYS_OPT_LOOPBACK
```



```
458         if (epp->ep_loopback_type == EFX_LOOPBACK_XGMII) {
459                 *upp = B_TRUE;
460                 return;
461         }
462 #endif  /* EFSYS_OPT_LOOPBACK */
463
464         EFX_BAR_READO(enp, FR_AB_XX_CORE_STAT_REG, &oword);
465
466         align_done = (EFX_OWORD_FIELD(oword, FRF_AB_XX_ALIGN_DONE) != 0);
467         sync_stat = (EFX_OWORD_FIELD(oword, FRF_AB_XX_SYNC_STAT0) != 0 &&
468             EFX_OWORD_FIELD(oword, FRF_AB_XX_SYNC_STAT1) != 0 &&
469             EFX_OWORD_FIELD(oword, FRF_AB_XX_SYNC_STAT2) != 0 &&
470             EFX_OWORD_FIELD(oword, FRF_AB_XX_SYNC_STAT3) != 0);
471
472         *upp = (align_done && sync_stat);
473
474         EFX_SET_OWORD_FIELD(oword, FRF_AB_XX_COMMA_DET_CH0, 1);
475         EFX_SET_OWORD_FIELD(oword, FRF_AB_XX_COMMA_DET_CH1, 1);
476         EFX_SET_OWORD_FIELD(oword, FRF_AB_XX_COMMA_DET_CH2, 1);
477         EFX_SET_OWORD_FIELD(oword, FRF_AB_XX_COMMA_DET_CH3, 1);
478
479         EFX_SET_OWORD_FIELD(oword, FRF_AB_XX_CHAR_ERR_CH0, 1);
480         EFX_SET_OWORD_FIELD(oword, FRF_AB_XX_CHAR_ERR_CH1, 1);
481         EFX_SET_OWORD_FIELD(oword, FRF_AB_XX_CHAR_ERR_CH2, 1);
482         EFX_SET_OWORD_FIELD(oword, FRF_AB_XX_CHAR_ERR_CH3, 1);
483
484         EFX_SET_OWORD_FIELD(oword, FRF_AB_XX_DISPERR_CH0, 1);
485         EFX_SET_OWORD_FIELD(oword, FRF_AB_XX_DISPERR_CH1, 1);
486         EFX_SET_OWORD_FIELD(oword, FRF_AB_XX_DISPERR_CH2, 1);
487         EFX_SET_OWORD_FIELD(oword, FRF_AB_XX_DISPERR_CH3, 1);
488
489         EFX_BAR_WRITEO(enp, FR_AB_XX_CORE_STAT_REG, &oword);
490 }
491
492                         void
493 falcon_xmac_poll(
494         __in            efx_nic_t *enp,
495         __out           boolean_t *mac_upp)
496 {
497         efx_port_t *epp = &(enp->en_port);
498         efx_phy_ops_t *epop = epp->ep_epop;
499         boolean_t mac_up = B_TRUE;
500         boolean_t ok;
501
502         /* Poll the mac link state if required */
503         if (epp->ep_mac_poll_needed) {
504                 falcon_xmac_downlink_check(enp, &mac_up);
505                 if (!mac_up)
506                         goto done;
507 #if EFSYS_OPT_LOOPBACK
508                 /* PHYXS link state irrelevent in MAC loopback */
509                 if ((1 << epp->ep_loopback_type) & EFX_LOOPBACK_MAC_MASK)
510                         goto done;
511 #endif
512                 if (epop->epo_uplink_check != NULL) {
513                         if (epop->epo_uplink_check(enp, &ok) == 0)
514                                 mac_up = ok;
515                 }
516         }
517
518 done:
519         *mac_upp = mac_up;
520
521         /*
522          * If the XAUI link (and therefore wireside link) are UP, then we
523          * can use XGMT interrupts rather than polling the link state to spot
```

```
524          * downwards transitions. To spot upwards transitions, we must poll
525          */
526         epp->ep_mac_poll_needed = !mac_up;
527         if (mac_up) {
528                 efx_oword_t oword;

530                 /* ACK the interrupt */
531                 EFX_BAR_READO(enp, FR_AB_XM_MGT_INT_REG, &oword);
532         } else {
533                 (void) falcon_xmac_xgxs_reset(enp);
534         }
535 }

537 #endif  /* EFSYS_OPT_MAC_FALCON_XMAC */
538 #endif /* ! codereview */
```

```
**********************************************************
    2294 Thu Aug 22 18:59:24 2013
new/usr/src/uts/common/io/sfxge/falcon_xmac.h
Merged sfxge driver
**********************************************************
```

```
/*-
 * Copyright 2008-2013 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#ifndef _SYS_FALCON_XMAC_H
#define _SYS_FALCON_XMAC_H

#include "efx.h"

#ifdef  __cplusplus
extern "C" {
#endif

#if EFSYS_OPT_MAC_FALCON_XMAC

#if EFSYS_OPT_LOOPBACK

#define FALCON_XMAC_LOOPBACK_MASK                                       \
        ((1 << EFX_LOOPBACK_XGMII) |                                    \
        (1 << EFX_LOOPBACK_XGXS) |                                      \
        (1 << EFX_LOOPBACK_XAUI))

#endif  /* EFSYS_OPT_LOOPBACK */

#define XMAC_INTR_SUPPORTED     B_TRUE

extern  __checkReturn   int
falcon_xmac_reset(
        __in            efx_nic_t *enp);

extern  __checkReturn   int
falcon_xmac_reconfigure(
        __in    efx_nic_t *enp);

#if EFSYS_OPT_MAC_STATS

extern  __checkReturn                   int
falcon_xmac_stats_update(
        __in                            efx_nic_t *enp,
        __in                            efsys_mem_t *esmp,
```



```
        __in_ecount(EFX_MAC_NSTATS)     efsys_stat_t *essp,
        __out_opt                       uint32_t *generationp);

#endif

#endif  /* EFSYS_OPT_MAC_XMAC */

extern                  void
falcon_xmac_poll(
        __in            efx_nic_t *enp,
        __out           boolean_t *mac_upp);

#ifdef  __cplusplus
}
#endif

#endif  /* _SYS_FALCON_XMAC_H */
#endif /* ! codereview */
```

```
**********************************************************
    5505 Thu Aug 22 18:59:25 2013
new/usr/src/uts/common/io/sfxge/lm87.c
Merged sfxge driver
**********************************************************
```

```
/*
 * Copyright 2007-2013 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#include "efsys.h"
#include "efx.h"
#include "efx_types.h"
#include "efx_regs.h"
#include "efx_impl.h"
#include "lm87.h"
#include "lm87_impl.h"

#if EFSYS_OPT_MON_LM87

        __checkReturn   int
lm87_reset(
        __in            efx_nic_t *enp)
{
        uint8_t devid = enp->en_u.falcon.enu_mon_devid;
        efx_byte_t byte;
        int rc;

        /* Check we can communicate with the chip */
        if ((rc = falcon_i2c_check(enp, devid)) != 0)
                goto fail1;

        /* Reset the chip */
        EFX_POPULATE_BYTE_1(byte, INIT, 1);
        if ((rc = falcon_i2c_write(enp, devid, CONFIG1_REG, (caddr_t)&byte,
            1)) != 0)
                goto fail2;

        /* Check the company identifier and revision */
        if ((rc = falcon_i2c_read(enp, devid, ID_REG, (caddr_t)&byte, 1)) != 0)
                goto fail3;

        if (EFX_BYTE_FIELD(byte, EFX_BYTE_0) != ID_DECODE) {
                rc = ENODEV;
                goto fail4;
        }
```



```
        if ((rc = falcon_i2c_read(enp, devid, REV_REG, (caddr_t)&byte, 1)) != 0)
                goto fail5;

        if (EFX_BYTE_FIELD(byte, EFX_BYTE_0) != REV_DECODE) {
                rc = ENODEV;
                goto fail6;
        }

        return (0);

fail6:
        EFSYS_PROBE(fail6);
fail5:
        EFSYS_PROBE(fail5);
fail4:
        EFSYS_PROBE(fail4);
fail3:
        EFSYS_PROBE(fail3);
fail2:
        EFSYS_PROBE(fail2);
fail1:
        EFSYS_PROBE1(fail1, int, rc);

        return (rc);
}

        __checkReturn   int
lm87_reconfigure(
        __in            efx_nic_t *enp)
{
        uint8_t devid = enp->en_u.falcon.enu_mon_devid;
        efx_byte_t byte;
        int rc;

        /* Configure the channel mode to select AIN1/2 rather than FAN1/2 */
        EFX_POPULATE_BYTE_2(byte, FAN1_AIN1, 1, FAN2_AIN2, 1);
        if ((rc = falcon_i2c_write(enp, devid, CHANNEL_MODE_REG, (caddr_t)&byte,
            1)) != 0)
                goto fail1;

        /* Mask out all interrupts */
        EFX_SET_BYTE(byte);
        if ((rc = falcon_i2c_write(enp, devid, INTERRUPT_MASK1_REG,
            (caddr_t)&byte, 1)) != 0)
                goto fail2;
        if ((rc = falcon_i2c_write(enp, devid, INTERRUPT_MASK2_REG,
            (caddr_t)&byte, 1)) != 0)
                goto fail3;

        /* Start monitoring */
        EFX_POPULATE_BYTE_1(byte, START, 1);
        if ((rc = falcon_i2c_write(enp, devid, CONFIG1_REG, (caddr_t)&byte,
            1)) != 0)
                goto fail4;

        return (0);

fail4:
        EFSYS_PROBE(fail4);
fail3:
        EFSYS_PROBE(fail3);
fail2:
        EFSYS_PROBE(fail2);
fail1:
        EFSYS_PROBE1(fail1, int, rc);
```

```
129         return (rc);
130 }

132 #if EFSYS_OPT_MON_STATS

134 #define LM87_STAT_SET(_enp, _mask, _value, _id, _rc)                    \
135         do {                                                            \
136                 uint8_t devid = enp->en_u.falcon.enu_mon_devid;         \
137                 efx_mon_stat_value_t *value =                           \
138                         (_value) + EFX_MON_STAT_ ## _id;                \
139                 efx_byte_t byte;                                        \
140                                                                         \
141                 if (((_rc) = falcon_i2c_read((_enp), devid,             \
142                     VALUE_ ## _id ## _REG, (caddr_t)&byte, 1)) == 0) {  \
143                         uint8_t val = EFX_BYTE_FIELD(byte, EFX_BYTE_0); \
144                         value->emsv_value = (uint16_t)val;              \
145                         value->emsv_state = 0;                          \
146                 }                                                       \
147                                                                         \
148                 (_mask) |= (1 << (EFX_MON_STAT_ ## _id));               \
149                 _NOTE(CONSTANTCONDITION)                                \
150         } while (B_FALSE)

152         __checkReturn                   int
153 lm87_stats_update(
154         __in                            efx_nic_t *enp,
155         __in                            efsys_mem_t *esmp,
156         __out_ecount(EFX_MON_NSTATS)    efx_mon_stat_value_t *values)
157 {
158         uint32_t mask = 0;
159         int rc;

161         _NOTE(ARGUNUSED(esmp))

163         LM87_STAT_SET(enp, mask, values, 2_5V, rc);
164         if (rc != 0)
165                 goto fail1;

167         LM87_STAT_SET(enp, mask, values, VCCP1, rc);
168         if (rc != 0)
169                 goto fail2;

171         LM87_STAT_SET(enp, mask, values, VCC, rc);
172         if (rc != 0)
173                 goto fail3;

175         LM87_STAT_SET(enp, mask, values, 5V, rc);
176         if (rc != 0)
177                 goto fail4;

179         LM87_STAT_SET(enp, mask, values, 12V, rc);
180         if (rc != 0)
181                 goto fail5;

183         LM87_STAT_SET(enp, mask, values, VCCP2, rc);
184         if (rc != 0)
185                 goto fail6;

187         LM87_STAT_SET(enp, mask, values, EXT_TEMP, rc);
188         if (rc != 0)
189                 goto fail7;

191         LM87_STAT_SET(enp, mask, values, INT_TEMP, rc);
192         if (rc != 0)
193                 goto fail8;
```



```
195         LM87_STAT_SET(enp, mask, values, AIN1, rc);
196         if (rc != 0)
197                 goto fail9;

199         LM87_STAT_SET(enp, mask, values, AIN2, rc);
200         if (rc != 0)
201                 goto fail10;

203         EFSYS_ASSERT3U(mask, ==, LM87_STAT_MASK);

205         return (0);

207 fail10:
208         EFSYS_PROBE(fail10);
209 fail9:
210         EFSYS_PROBE(fail9);
211 fail8:
212         EFSYS_PROBE(fail8);
213 fail7:
214         EFSYS_PROBE(fail7);
215 fail6:
216         EFSYS_PROBE(fail6);
217 fail5:
218         EFSYS_PROBE(fail5);
219 fail4:
220         EFSYS_PROBE(fail4);
221 fail3:
222         EFSYS_PROBE(fail3);
223 fail2:
224         EFSYS_PROBE(fail2);
225 fail1:
226         EFSYS_PROBE1(fail1, int, rc);

228         return (rc);
229 }
230 #endif  /* EFSYS_OPT_MON_STATS */

232 #endif  /* EFSYS_OPT_MON_LM87 */
233 #endif /* ! codereview */
```

```
**********************************************************
    2445 Thu Aug 22 18:59:25 2013
new/usr/src/uts/common/io/sfxge/lm87.h
Merged sfxge driver
**********************************************************
```

```
/*-
 * Copyright 2007-2013 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#ifndef _SYS_LM87_H
#define _SYS_LM87_H

#include "efx.h"

#ifdef  __cplusplus
extern "C" {
#endif

#if EFSYS_OPT_MON_LM87

#define LM87_DEVID 0x2e

extern  __checkReturn   int
lm87_reset(
        __in            efx_nic_t *enp);

extern  __checkReturn   int
lm87_reconfigure(
        __in            efx_nic_t *enp);

#if EFSYS_OPT_MON_STATS

/* START MKCONFIG GENERATED Lm87MonitorHeaderStatsMask ebc33ca917ba24d7 */
#define LM87_STAT_MASK \
        (1ULL << EFX_MON_STAT_2_5V) | \
        (1ULL << EFX_MON_STAT_VCCP1) | \
        (1ULL << EFX_MON_STAT_VCC) | \
        (1ULL << EFX_MON_STAT_5V) | \
        (1ULL << EFX_MON_STAT_12V) | \
        (1ULL << EFX_MON_STAT_VCCP2) | \
        (1ULL << EFX_MON_STAT_EXT_TEMP) | \
        (1ULL << EFX_MON_STAT_INT_TEMP) | \
        (1ULL << EFX_MON_STAT_AIN1) | \
        (1ULL << EFX_MON_STAT_AIN2)
```



```
/* END MKCONFIG GENERATED Lm87MonitorHeaderStatsMask */

extern  __checkReturn                   int
lm87_stats_update(
        __in                            efx_nic_t *enp,
        __in                            efsys_mem_t *esmp,
        __out_ecount(EFX_MON_NSTATS)    efx_mon_stat_value_t *values);

#endif  /* EFSYS_OPT_MON_STATS */

#endif  /* EFSYS_OPT_MON_LM87 */

#ifdef  __cplusplus
}
#endif

#endif  /* _SYS_LM87_H */
#endif /* ! codereview */
```

```
**********************************************************
    2375 Thu Aug 22 18:59:25 2013
new/usr/src/uts/common/io/sfxge/lm87_impl.h
Merged sfxge driver
**********************************************************
```

```
/*-
 * Copyright 2007-2013 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ``AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#ifndef _SYS_LM87_IMPL_H
#define _SYS_LM87_IMPL_H

#ifdef  __cplusplus
extern "C" {
#endif

#if EFSYS_OPT_MON_LM87

#define TEST_REG 0x15
#define SHUTDOWN_LBN 0
#define SHUTDOWN_WIDTH 1

#define CHANNEL_MODE_REG 0x16
#define FAN1_AIN1_LBN 0
#define FAN1_AIN1_WIDTH 1
#define FAN2_AIN2_LBN 1
#define FAN2_AIN2_WIDTH 1

#define CONFIG1_REG 0x40
#define START_LBN 0
#define START_WIDTH 1
#define INT_EN_LBN 1
#define INT_EN_WIDTH 1
#define INIT_LBN 7
#define INIT_WIDTH 1

#define INTERRUPT_MASK1_REG 0x43
#define INTERRUPT_MASK2_REG 0x44

#define VALUE_2_5V_REG 0x20
#define VALUE_VCCP1_REG 0x21
#define VALUE_VCC_REG 0x22
#define VALUE_5V_REG 0x23
#define VALUE_12V_REG 0x24
#define VALUE_VCCP2_REG 0x25
```



```
#define VALUE_EXT_TEMP_REG 0x26
#define VALUE_INT_TEMP_REG 0x27
#define VALUE_AIN1_REG 0x28
#define VALUE_AIN2_REG 0x29

#define ID_REG 0x3e
#define ID_DECODE 0x02

#define REV_REG 0x3f
#define REV_DECODE 0x06

#endif  /* EFSYS_OPT_MON_LM87 */

#ifdef  __cplusplus
}
#endif

#endif  /* _SYS_LM87_IMPL_H */
#endif /* ! codereview */
```

```
**********************************************************
    4695 Thu Aug 22 18:59:25 2013
new/usr/src/uts/common/io/sfxge/max6647.c
Merged sfxge driver
**********************************************************
```

```
/*
 * Copyright 2007-2013 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#include "efsys.h"
#include "efx.h"
#include "efx_types.h"
#include "efx_regs.h"
#include "efx_impl.h"
#include "max6647.h"
#include "max6647_impl.h"

#if EFSYS_OPT_MON_MAX6647

        __checkReturn   int
max6647_reset(
        __in            efx_nic_t *enp)
{
        uint8_t devid = enp->en_u.falcon.enu_mon_devid;
        efx_byte_t byte;
        int rc;

        if ((rc = falcon_i2c_check(enp, devid)) != 0)
                goto fail1;

        /* Reset the chip */
        EFX_POPULATE_BYTE_2(byte, MASK, 1, NRUN, 1);
        if ((rc = falcon_i2c_write(enp, devid, WCA_REG, (caddr_t)&byte,
            1)) != 0)
                goto fail2;

        EFSYS_SPIN(10);

        EFX_POPULATE_BYTE_2(byte, MASK, 1, NRUN, 0);
        if ((rc = falcon_i2c_write(enp, devid, WCA_REG, (caddr_t)&byte,
            1)) != 0)
                goto fail3;

        /* Check the company identifier and revision */
        if ((rc = falcon_i2c_read(enp, devid, MFID_REG, (caddr_t)&byte,
```



```
            1)) != 0)
                goto fail2;

        if (EFX_BYTE_FIELD(byte, EFX_BYTE_0) != MFID_DECODE) {
                rc = ENODEV;
                goto fail3;
        }

        if ((rc = falcon_i2c_read(enp, devid, REVID_REG, (caddr_t)&byte,
            1)) != 0)
                goto fail4;

        if (EFX_BYTE_FIELD(byte, EFX_BYTE_0) != REVID_DECODE) {
                rc = ENODEV;
                goto fail5;
        }

        return (0);

fail5:
        EFSYS_PROBE(fail5);
fail4:
        EFSYS_PROBE(fail4);
fail3:
        EFSYS_PROBE(fail3);
fail2:
        EFSYS_PROBE(fail2);
fail1:
        EFSYS_PROBE1(fail1, int, rc);

        return (rc);
}

        __checkReturn   int
max6647_reconfigure(
        __in            efx_nic_t *enp)
{
        uint8_t devid = enp->en_u.falcon.enu_mon_devid;
        efx_byte_t byte;
        int rc;

        /* Clear any latched status */
        if ((rc = falcon_i2c_read(enp, devid, RSL_REG, (caddr_t)&byte,
            1)) != 0)
                goto fail1;

        /* Override the default alert limit */
        EFX_POPULATE_BYTE_1(byte, EFX_BYTE_0, 90);
        if ((rc = falcon_i2c_write(enp, devid, WLHO_REG, (caddr_t)&byte,
            1)) != 0)
                goto fail2;

        /* Read it back and verify */
        if ((rc = falcon_i2c_read(enp, devid, RLHN_REG, (caddr_t)&byte,
            1)) != 0)
                goto fail3;

        if (EFX_BYTE_FIELD(byte, EFX_BYTE_0) != 90) {
                rc = EFAULT;
                goto fail4;
        }

        /* Enable the alert signal */
        EFX_POPULATE_BYTE_2(byte, MASK, 0, NRUN, 0);
        if ((rc = falcon_i2c_write(enp, devid, WCA_REG, (caddr_t)&byte,
            1)) != 0)
```

```
128                 goto fail5;

130         return (0);

132 fail5:
133         EFSYS_PROBE(fail5);
134 fail4:
135         EFSYS_PROBE(fail4);
136 fail3:
137         EFSYS_PROBE(fail3);
138 fail2:
139         EFSYS_PROBE(fail2);
140 fail1:
141         EFSYS_PROBE1(fail1, int, rc);

143         return (rc);
144 }

146 #if EFSYS_OPT_MON_STATS

148 #define MAX6647_STAT_SET(_enp, _value, _id, _rc)                        \
149         do {                                                            \
150                 uint8_t devid = enp->en_u.falcon.enu_mon_devid;         \
151                 efx_byte_t byte;                                        \
152                                                                         \
153                 if (((_rc) = falcon_i2c_read((_enp), devid,             \
154                     _id ## _REG, (caddr_t)&byte, 1)) == 0) {            \
155                         (_value)->emsv_value =                          \
156                                 (uint16_t)EFX_BYTE_FIELD(byte,          \
157                                                          EFX_BYTE_0);   \
158                         (_value)->emsv_state = 0;                       \
159                 }                                                       \
160                 _NOTE(CONSTANTCONDITION)                                \
161         } while (B_FALSE)

163         __checkReturn                   int
164 max6647_stats_update(
165         __in                            efx_nic_t *enp,
166         __in                            efsys_mem_t *esmp,
167         __out_ecount(EFX_MON_NSTATS)    efx_mon_stat_value_t *values)
168 {
169         int rc;

171         _NOTE(ARGUNUSED(esmp))

173         MAX6647_STAT_SET(enp, values + EFX_MON_STAT_INT_TEMP, RLTS, rc);
174         if (rc != 0)
175                 goto fail1;

177         MAX6647_STAT_SET(enp, values + EFX_MON_STAT_EXT_TEMP, RRTE, rc);
178         if (rc != 0)
179                 goto fail2;

181         return (0);

183 fail2:
184         EFSYS_PROBE(fail2);
185 fail1:
186         EFSYS_PROBE1(fail1, int, rc);

188         return (rc);
189 }
190 #endif  /* EFSYS_OPT_MON_STATS */

192 #endif  /* EFSYS_OPT_MON_MAX6647 */
193 #endif /* ! codereview */
```

```
**********************************************************
    2246 Thu Aug 22 18:59:25 2013
new/usr/src/uts/common/io/sfxge/max6647.h
Merged sfxge driver
**********************************************************
```

```
/*-
 * Copyright 2007-2013 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#ifndef _SYS_MAX6647_H
#define _SYS_MAX6647_H

#include "efx.h"

#ifdef  __cplusplus
extern "C" {
#endif

#if EFSYS_OPT_MON_MAX6647

#define MAX6646_DEVID 0x4d
#define MAX6647_DEVID 0x4e

extern  __checkReturn   int
max6647_reset(
        __in            efx_nic_t *enp);

extern  __checkReturn   int
max6647_reconfigure(
        __in            efx_nic_t *enp);

#if EFSYS_OPT_MON_STATS

/* START MKCONFIG GENERATED Max6647MonitorHeaderStatsMask b4d91f25c4d293e1 */
#define MAX6647_STAT_MASK \
        (1ULL << EFX_MON_STAT_INT_TEMP) | \
        (1ULL << EFX_MON_STAT_EXT_TEMP)

/* END MKCONFIG GENERATED Max6647MonitorHeaderStatsMask */

extern  __checkReturn                   int
max6647_stats_update(
        __in                            efx_nic_t *enp,
        __in                            efsys_mem_t *esmp,
        __out_ecount(EFX_MON_NSTATS)    efx_mon_stat_value_t *values);
```



```
#endif  /* EFSYS_OPT_MON_STATS */

#endif  /* EFSYS_OPT_MON_MAX6647 */

#ifdef  __cplusplus
}
#endif

#endif  /* _SYS_MAX6647_H */
#endif /* ! codereview */
```

```
**********************************************************
    2622 Thu Aug 22 18:59:25 2013
new/usr/src/uts/common/io/sfxge/max6647_impl.h
Merged sfxge driver
**********************************************************
```

```
/*-
 * Copyright 2007-2013 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#ifndef _SYS_MAX6647_IMPL_H
#define _SYS_MAX6647_IMPL_H

#ifdef  __cplusplus
extern "C" {
#endif

#if EFSYS_OPT_MON_MAX6647 || EFSYS_OPT_PHY_SFX7101

#define RLTS_REG 0x00

#define RRTE_REG 0x01

#define RSL_REG 0x02
#define BUSY_LBN 7
#define BUSY_WIDTH 1
#define LHIGH_LBN 6
#define LHIGH_WIDTH 1
#define LLOW_LBN 5
#define LLOW_WIDTH 1
#define RHIGH_LBN 4
#define RHIGH_WIDTH 1
#define RLOW_LBN 3
#define RLOW_WIDTH 1
#define FAULT_LBN 2
#define FAULT_WIDTH 1
#define EOT_LBN 1
#define EOT_WIDTH 1
#define IOT_LBN 0
#define IOT_WIDTH 1

#define RCL_REG 0x03
#define MASK_LBN 7
#define MASK_WIDTH 1
#define NRUN_LBN 6
#define NRUN_WIDTH 1
```



```
#define RCRA_REG 0x04

#define RLHN_REG 0x05

#define RLLI_REG 0x06

#define RRHI_REG 0x07

#define RRLS_REG 0x08

#define WCA_REG 0x09

#define WCRW_REG 0x0a

#define WLHO_REG 0x0b

#define WRHA_REG 0x0c

#define WRLN_REG 0x0e

#define OSHT_REG 0x0f

#define REET_REG 0x10

#define RIET_REG 0x11

#define RWOE_REG 0x19

#define RWOI_REG 0x20

#define HYS_REG 0x21

#define QUEUE_REG 0x22

#define MFID_REG 0xfe
#define MFID_DECODE 0x4d

#define REVID_REG 0xff
#define REVID_DECODE 0x59

#endif  /* EFSYS_OPT_MON_MAX6647 || EFSYS_OPT_PHY_SFX7101 */

#ifdef  __cplusplus
}
#endif

#endif  /* _SYS_MAX6647_IMPL_H */
#endif /* ! codereview */
```

```
**********************************************************
    2031 Thu Aug 22 18:59:25 2013
new/usr/src/uts/common/io/sfxge/nullmon.c
Merged sfxge driver
**********************************************************
```

```
/*
 * Copyright 2007-2013 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#include "efsys.h"
#include "efx.h"
#include "efx_types.h"
#include "efx_regs.h"
#include "efx_impl.h"
#include "nullmon.h"

#if EFSYS_OPT_MON_NULL

        __checkReturn   int
nullmon_reset(
        __in            efx_nic_t *enp)
{
        _NOTE(ARGUNUSED(enp))

        return (0);
}

        __checkReturn   int
nullmon_reconfigure(
        __in            efx_nic_t *enp)
{
        _NOTE(ARGUNUSED(enp))

        return (0);
}

#if EFSYS_OPT_MON_STATS

        __checkReturn                   int
nullmon_stats_update(
        __in                            efx_nic_t *enp,
        __in                            efsys_mem_t *esmp,
        __out_ecount(EFX_MON_NSTATS)    efx_mon_stat_value_t *values)
{
        _NOTE(ARGUNUSED(enp, esmp, values))
```



```
        return (ENOTSUP);
}

#endif  /* EFSYS_OPT_MON_STATS */

#endif  /* EFSYS_OPT_MON_NULL */
#endif /* ! codereview */
```

```
**********************************************************
    2005 Thu Aug 22 18:59:25 2013
new/usr/src/uts/common/io/sfxge/nullmon.h
Merged sfxge driver
**********************************************************
```

```
/*-
 * Copyright 2007-2013 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ‘‘AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#ifndef _SYS_NULLMON_H
#define _SYS_NULLMON_H

#include "efx.h"

#ifdef  __cplusplus
extern "C" {
#endif

#if EFSYS_OPT_MON_NULL

#if EFSYS_OPT_MON_STATS
#define NULLMON_NSTATS  0
#endif

extern  __checkReturn   int
nullmon_reset(
        __in            efx_nic_t *enp);

extern  __checkReturn   int
nullmon_reconfigure(
        __in            efx_nic_t *enp);

#if EFSYS_OPT_MON_STATS

extern  __checkReturn                   int
nullmon_stats_update(
        __in                            efx_nic_t *enp,
        __in                            efsys_mem_t *esmp,
        __out_ecount(EFX_MON_NSTATS)    efx_mon_stat_value_t *values);

#endif  /* EFSYS_OPT_MON_STATS */

#endif  /* EFSYS_OPT_MON_NULL */

#ifdef  __cplusplus
```



```
}
#endif

#endif  /* _SYS_NULLMON_H */
#endif /* ! codereview */
```

```
**********************************************************
    4178 Thu Aug 22 18:59:25 2013
new/usr/src/uts/common/io/sfxge/nullphy.c
Merged sfxge driver
**********************************************************
```

```
/*
 * Copyright 2007-2013 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#include "efsys.h"
#include "efx.h"
#include "efx_types.h"
#include "efx_regs.h"
#include "efx_impl.h"
#include "nullphy.h"
#include "nullphy_impl.h"

#if EFSYS_OPT_PHY_NULL

        __checkReturn   int
nullphy_reset(
        __in            efx_nic_t *enp)
{
        _NOTE(ARGUNUSED(enp))

        enp->en_reset_flags |= EFX_RESET_PHY;

        return (0);
}

        __checkReturn   int
nullphy_reconfigure(
        __in            efx_nic_t *enp)
{
        efx_port_t *epp = &(enp->en_port);
        efx_word_t word;
        efx_word_t check;
        int rc;

        _NOTE(ARGUNUSED(enp))

        EFX_POPULATE_WORD_3(word, HOSTPORT_EQ, 1, PORTSEL, 0, CX4uC_RESET, 1);

        if ((rc = falcon_i2c_send(enp, PCF8575, (caddr_t)&word.ew_byte[0],
            sizeof (word.ew_byte) / sizeof (efx_byte_t))) != 0)
```



```
                goto fail1;

        if ((rc = falcon_i2c_recv(enp, PCF8575, (caddr_t)&check.ew_byte[0],
            sizeof (check.ew_byte) / sizeof (efx_byte_t))) != 0)
                goto fail2;

        if (EFX_WORD_FIELD(check, EFX_WORD_0) !=
            EFX_WORD_FIELD(word, EFX_WORD_0)) {
                rc = EFAULT;
                goto fail3;
        }

        EFSYS_ASSERT3U(epp->ep_adv_cap_mask, ==, NULLPHY_ADV_CAP_MASK);

        return (0);

fail3:
        EFSYS_PROBE(fail3);
fail2:
        EFSYS_PROBE(fail2);
fail1:
        EFSYS_PROBE1(fail1, int, rc);

        return (rc);
}

        __checkReturn   int
nullphy_verify(
        __in            efx_nic_t *enp)
{
        _NOTE(ARGUNUSED(enp))

        return (ENOTSUP);
}

        __checkReturn   int
nullphy_downlink_check(
        __in            efx_nic_t *enp,
        __out           efx_link_mode_t *modep,
        __out           unsigned int *fcntlp,
        __out           uint32_t *lp_cap_maskp)
{
        efx_port_t *epp = &(enp->en_port);

        *modep = EFX_LINK_10000FDX;
        *fcntlp = epp->ep_fcntl;
        *lp_cap_maskp = NULLPHY_ADV_CAP_MASK;

        return (0);
}

        __checkReturn   int
nullphy_lp_cap_get(
        __in            efx_nic_t *enp,
        __out           uint32_t *maskp)
{
        _NOTE(ARGUNUSED(enp, maskp))

        return (ENOTSUP);
}

        __checkReturn   int
nullphy_oui_get(
        __in            efx_nic_t *enp,
        __out           uint32_t *ouip)
{
```

```
128         _NOTE(ARGUNUSED(enp, ouip))

130         return (ENOTSUP);
131 }

133 #if EFSYS_OPT_PHY_STATS

135         __checkReturn                   int
136 nullphy_stats_update(
137         __in                            efx_nic_t *enp,
138         __in                            efsys_mem_t *esmp,
139         __out_ecount(EFX_PHY_NSTATS)    uint32_t *stat)
140 {
141         _NOTE(ARGUNUSED(enp, esmp, stat))

143         return (ENOTSUP);
144 }
145 #endif  /* EFSYS_OPT_PHY_STATS */

147 #if EFSYS_OPT_PHY_PROPS

149 #if EFSYS_OPT_NAMES
150                 const char __cs *
151 nullphy_prop_name(
152         __in    efx_nic_t *enp,
153         __in    unsigned int id)
154 {
155         _NOTE(ARGUNUSED(enp, id))

157         EFSYS_ASSERT(B_FALSE);

159         return (NULL);
160 }
161 #endif  /* EFSYS_OPT_NAMES */

163         __checkReturn   int
164 nullphy_prop_get(
165         __in            efx_nic_t *enp,
166         __in            unsigned int id,
167         __in            uint32_t flags,
168         __out           uint32_t *valp)
169 {
170         _NOTE(ARGUNUSED(enp, id, flags, valp))

172         EFSYS_ASSERT(B_FALSE);

174         return (ENOTSUP);
175 }

177         __checkReturn   int
178 nullphy_prop_set(
179         __in            efx_nic_t *enp,
180         __in            unsigned int id,
181         __in            uint32_t val)
182 {
183         _NOTE(ARGUNUSED(enp, id, val))

185         EFSYS_ASSERT(B_FALSE);

187         return (ENOTSUP);
188 }
189 #endif  /* EFSYS_OPT_PHY_PROPS */

191 #endif  /* EFSYS_OPT_PHY_NULL */
192 #endif /* ! codereview */
```

```
**********************************************************
    3043 Thu Aug 22 18:59:25 2013
new/usr/src/uts/common/io/sfxge/nullphy.h
Merged sfxge driver
**********************************************************
```

```
/*-
 * Copyright 2007-2013 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#ifndef _SYS_NULLPHY_H
#define _SYS_NULLPHY_H

#include "efx.h"

#ifdef  __cplusplus
extern "C" {
#endif

#if EFSYS_OPT_PHY_NULL

#define NULLPHY_LOOPBACK_MASK                                           \
            FALCON_XMAC_LOOPBACK_MASK

#define NULLPHY_LED_MASK        0

#define NULLPHY_NSTATS          0

#define NULLPHY_NPROPS          0

#define NULLPHY_ADV_CAP_MASK                                            \
            ((1 << EFX_PHY_CAP_10000FDX) |                              \
            (1 << EFX_PHY_CAP_PAUSE))

#define NULLPHY_ADV_CAP_PERM    0

#define NULLPHY_BIST_MASK       0

extern  __checkReturn   int
nullphy_reset(
        __in            efx_nic_t *enp);

extern  __checkReturn   int
nullphy_reconfigure(
        __in            efx_nic_t *enp);
```



```
extern  __checkReturn   int
nullphy_verify(
        __in            efx_nic_t *enp);

extern  __checkReturn   int
nullphy_downlink_check(
        __in            efx_nic_t *enp,
        __out           efx_link_mode_t *modep,
        __out           unsigned int *fcntlp,
        __out           uint32_t *lp_cap_maskp);

extern  __checkReturn   int
nullphy_oui_get(
        __in            efx_nic_t *enp,
        __out           uint32_t *ouip);

#if EFSYS_OPT_PHY_STATS

#define NULLPHY_STAT_MASK       0

extern  __checkReturn                   int
nullphy_stats_update(
        __in                            efx_nic_t *enp,
        __in                            efsys_mem_t *esmp,
        __out_ecount(EFX_PHY_NSTATS)    uint32_t *stat);

#endif  /* EFSYS_OPT_PHY_STATS */

#if EFSYS_OPT_PHY_PROPS

#if EFSYS_OPT_NAMES

extern          const char __cs *
nullphy_prop_name(
        __in            efx_nic_t *enp,
        __in            unsigned int id);

#endif

extern  __checkReturn   int
nullphy_prop_get(
        __in            efx_nic_t *enp,
        __in            unsigned int id,
        __in            uint32_t flags,
        __out           uint32_t *valp);

extern  __checkReturn   int
nullphy_prop_set(
        __in            efx_nic_t *enp,
        __in            unsigned int id,
        __in            uint32_t val);

#endif  /* EFSYS_OPT_PHY_PROPS */

#endif  /* EFSYS_OPT_PHY_NULL */

#ifdef  __cplusplus
}
#endif

#endif  /* _SYS_NULLPHY_H */
#endif /* ! codereview */
```

```
**********************************************************
    2252 Thu Aug 22 18:59:25 2013
new/usr/src/uts/common/io/sfxge/nullphy_impl.h
Merged sfxge driver
**********************************************************
```

```
/*-
 * Copyright 2008-2013 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#ifndef _SYS_NULLPHY_IMPL_H
#define _SYS_NULLPHY_IMPL_H

#ifdef  __cplusplus
extern "C" {
#endif

#if EFSYS_OPT_PHY_NULL

/* IO expender */
#define PCF8575 0x20

#define PORT0_EXTLOOP_LBN 0
#define PORT0_EXTLOOP_WIDTH 1
#define PORT1_EXTLOOP_LBN 1
#define PORT1_EXTLOOP_WIDTH 1
#define HOSTPORT_LOOP_LBN 2
#define HOSTPORT_LOOP_WIDTH 1
#define BCAST_LBN 3
#define BCAST_WIDTH 1
#define PORT0_EQ_LBN 4
#define PORT0_EQ_WIDTH 1
#define PORT1_EQ_LBN 5
#define PORT1_EQ_WIDTH 1
#define HOSTPORT_EQ_LBN 6
#define HOSTPORT_EQ_WIDTH 1
#define PORTSEL_LBN 7
#define PORTSEL_WIDTH 1
#define PORT0_PRE_LBN 8
#define PORT0_PRE_WIDTH 2
#define PORT1_PRE_LBN 10
#define PORT1_PRE_WIDTH 2
#define HOSTPORT_PRE_LBN 12
#define HOSTPORT_PRE_WIDTH 2
#define CX4uC_RESET_LBN 15
#define CX4uC_RESET_WIDTH 1
```



```
#endif  /* EFSYS_OPT_PHY_NULL */

#ifdef  __cplusplus
}
#endif

#endif  /* _SYS_NULLPHY_IMPL_H */
#endif /* ! codereview */
```

****
13167 Thu Aug 22 18:59:26 2013
new/usr/src/uts/common/io/sfxge/qt2022c2.c
Merged sfxge driver
****

```
/*
 * Copyright 2007-2013 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#include "efsys.h"
#include "efx.h"
#include "efx_types.h"
#include "efx_regs.h"
#include "efx_impl.h"
#include "xphy.h"
#include "qt2022c2.h"
#include "qt2022c2_impl.h"

#if EFSYS_OPT_PHY_QT2022C2

static  __checkReturn   int
qt2022c2_led_cfg(
        __in            efx_nic_t *enp)
{
        efx_port_t *epp = &(enp->en_port);
        efx_word_t led1;
        efx_word_t led2;
        efx_word_t led3;
        int rc;

#if EFSYS_OPT_PHY_LED_CONTROL

        switch (epp->ep_phy_led_mode) {
        case EFX_PHY_LED_DEFAULT:
                EFX_POPULATE_WORD_2(led1, PMA_PMD_LED_CFG, LED_CFG_LSA_DECODE,
                    PMA_PMD_LED_PATH, LED_PATH_RX_DECODE);
                EFX_POPULATE_WORD_2(led2, PMA_PMD_LED_CFG, LED_CFG_LSA_DECODE,
                    PMA_PMD_LED_PATH, LED_PATH_TX_DECODE);
                EFX_POPULATE_WORD_1(led3, PMA_PMD_LED_CFG, LED_CFG_OFF_DECODE);
                break;

        case EFX_PHY_LED_OFF:
                EFX_POPULATE_WORD_1(led1, PMA_PMD_LED_CFG, LED_CFG_OFF_DECODE);
                EFX_POPULATE_WORD_1(led2, PMA_PMD_LED_CFG, LED_CFG_OFF_DECODE);
                EFX_POPULATE_WORD_1(led3, PMA_PMD_LED_CFG, LED_CFG_OFF_DECODE);
```



```
                break;

        case EFX_PHY_LED_ON:
                EFX_POPULATE_WORD_1(led1, PMA_PMD_LED_CFG, LED_CFG_ON_DECODE);
                EFX_POPULATE_WORD_1(led2, PMA_PMD_LED_CFG, LED_CFG_ON_DECODE);
                EFX_POPULATE_WORD_1(led3, PMA_PMD_LED_CFG, LED_CFG_ON_DECODE);
                break;

        default:
                EFSYS_ASSERT(B_FALSE);
                break;
        }

#else   /* EFSYS_OPT_PHY_LED_CONTROL */

        EFX_POPULATE_WORD_2(led1, PMA_PMD_LED_CFG, LED_CFG_LSA_DECODE,
            PMA_PMD_LED_PATH, LED_PATH_RX_DECODE);
        EFX_POPULATE_WORD_2(led2, PMA_PMD_LED_CFG, LED_CFG_LSA_DECODE,
            PMA_PMD_LED_PATH, LED_PATH_TX_DECODE);
        EFX_POPULATE_WORD_1(led3, PMA_PMD_LED_CFG, LED_CFG_OFF_DECODE);

#endif  /* EFSYS_OPT_PHY_LED_CONTROL */

        if ((rc = falcon_mdio_write(enp, epp->ep_port, PMA_PMD_MMD,
            PMA_PMD_LED1_REG, &led1)) != 0)
                goto fail1;

        if ((rc = falcon_mdio_write(enp, epp->ep_port, PMA_PMD_MMD,
            PMA_PMD_LED2_REG, &led2)) != 0)
                goto fail2;

        if ((rc = falcon_mdio_write(enp, epp->ep_port, PMA_PMD_MMD,
            PMA_PMD_LED3_REG, &led3)) != 0)
                goto fail3;

        return (0);

fail3:
        EFSYS_PROBE(fail2);
fail2:
        EFSYS_PROBE(fail2);
fail1:
        EFSYS_PROBE1(fail1, int, rc);

        return (rc);
}

#if EFSYS_OPT_LOOPBACK
static  __checkReturn   int
qt2022c2_loopback_cfg(
        __in            efx_nic_t *enp)
{
        efx_port_t *epp = &(enp->en_port);
        int rc;

        switch (epp->ep_loopback_type) {
        case EFX_LOOPBACK_PHY_XS: {
                efx_word_t word;

                if ((rc = falcon_mdio_read(enp, epp->ep_port, PHY_XS_MMD,
                    PHY_XS_VENDOR0_REG, &word)) != 0)
                        goto fail1;

                EFX_SET_WORD_FIELD(word, XAUI_SYSTEM_LOOPBACK, 1);

                if ((rc = falcon_mdio_write(enp, epp->ep_port, PHY_XS_MMD,
```

```
128                     PHY_XS_VENDOR0_REG, &word)) != 0)
129                         goto fail2;

131                 break;
132         }
133         case EFX_LOOPBACK_PCS:
134                 if ((rc = xphy_mmd_loopback_set(enp, epp->ep_port, PCS_MMD,
135                     B_TRUE)) != 0)
136                         goto fail1;

138                 break;

140         case EFX_LOOPBACK_PMA_PMD:
141                 if ((rc = xphy_mmd_loopback_set(enp, epp->ep_port, PMA_PMD_MMD,
142                     B_TRUE)) != 0)
143                         goto fail1;

145                 break;

147         default:
148                 break;
149         }

151         return (0);

153 fail2:
154         EFSYS_PROBE(fail2);
155 fail1:
156         EFSYS_PROBE1(fail1, int, rc);

158         return (rc);
159 }
160 #endif  /* EFSYS_OPT_LOOPBACK */

162         __checkReturn   int
163 qt2022c2_reset(
164         __in            efx_nic_t *enp)
165 {
166         /* Pull the external reset line */
167         falcon_nic_phy_reset(enp);

169         return (0);
170 }

172         __checkReturn   int
173 qt2022c2_reconfigure(
174         __in            efx_nic_t *enp)
175 {
176         efx_port_t *epp = &(enp->en_port);
177         int rc;

179         if ((rc = xphy_pkg_wait(enp, epp->ep_port, QT2022C2_MMD_MASK)) != 0)
180                 goto fail1;

182         if ((rc = qt2022c2_led_cfg(enp)) != 0)
183                 goto fail2;

185         EFSYS_ASSERT3U(epp->ep_adv_cap_mask, ==, QT2022C2_ADV_CAP_MASK);

187 #if EFSYS_OPT_LOOPBACK
188         if ((rc = qt2022c2_loopback_cfg(enp)) != 0)
189                 goto fail3;
190 #endif  /* EFSYS_OPT_LOOPBACK */

192         return (0);
```



```
194 #if EFSYS_OPT_LOOPBACK
195 fail3:
196         EFSYS_PROBE(fail3);
197 #endif  /* EFSYS_OPT_LOOPBACK */

199 fail2:
200         EFSYS_PROBE(fail2);
201 fail1:
202         EFSYS_PROBE1(fail1, int, rc);

204         return (rc);
205 }

207         __checkReturn   int
208 qt2022c2_verify(
209         __in            efx_nic_t *enp)
210 {
211         efx_port_t *epp = &(enp->en_port);
212         int rc;

214         if ((rc = xphy_pkg_verify(enp, epp->ep_port, QT2022C2_MMD_MASK)) != 0)
215                 goto fail1;

217         return (0);

219 fail1:
220         EFSYS_PROBE1(fail1, int, rc);

222         return (rc);
223 }

225         __checkReturn   int
226 qt2022c2_uplink_check(
227         __in            efx_nic_t *enp,
228         __out           boolean_t *upp)
229 {
230         efx_port_t *epp = &(enp->en_port);
231         efx_word_t word;
232         int rc;

234         if (epp->ep_mac_type != EFX_MAC_FALCON_XMAC) {
235                 rc = ENOTSUP;
236                 goto fail1;
237         }

239         if ((rc = falcon_mdio_read(enp, epp->ep_port, PHY_XS_MMD,
240             PHY_XS_LANE_STATUS_REG, &word)) != 0)
241                 goto fail2;

243         *upp = ((EFX_WORD_FIELD(word, PHY_XS_ALIGNED) != 0) &&
244             (EFX_WORD_FIELD(word, PHY_XS_LANE0_SYNC) != 0) &&
245             (EFX_WORD_FIELD(word, PHY_XS_LANE1_SYNC) != 0) &&
246             (EFX_WORD_FIELD(word, PHY_XS_LANE2_SYNC) != 0) &&
247             (EFX_WORD_FIELD(word, PHY_XS_LANE3_SYNC) != 0));

249         return (0);

251 fail2:
252         EFSYS_PROBE(fail2);
253 fail1:
254         EFSYS_PROBE1(fail1, int, rc);

256         return (rc);
257 }

259         __checkReturn   int
```

```
qt2022c2_downlink_check(
        __in            efx_nic_t *enp,
        __out           efx_link_mode_t *modep,
        __out           unsigned int *fcntlp,
        __out           uint32_t *lp_cap_maskp)
{
        efx_port_t *epp = &(enp->en_port);
        boolean_t up;
        int rc;

#if EFSYS_OPT_LOOPBACK
        switch (epp->ep_loopback_type) {
        case EFX_LOOPBACK_PHY_XS:
                rc = xphy_mmd_fault(enp, epp->ep_port, &up);
                if (rc != 0)
                        goto fail1;

                *modep = (up) ? EFX_LINK_10000FDX : EFX_LINK_DOWN;
                goto done;

        case EFX_LOOPBACK_PCS:
                rc = xphy_mmd_check(enp, epp->ep_port, PHY_XS_MMD, &up);
                if (rc != 0)
                        goto fail1;

                *modep = (up) ? EFX_LINK_10000FDX : EFX_LINK_DOWN;
                goto done;

        default:
                break;
        }
#endif /* EFSYS_OPT_LOOPBACK */

        if ((rc = xphy_mmd_check(enp, epp->ep_port, PCS_MMD, &up)) != 0)
                goto fail1;

        *modep = (up) ? EFX_LINK_10000FDX : EFX_LINK_DOWN;

#if EFSYS_OPT_LOOPBACK
done:
#endif
        *fcntlp = epp->ep_fcntl;
        *lp_cap_maskp = epp->ep_lp_cap_mask;

        return (0);

fail1:
        EFSYS_PROBE1(fail1, int, rc);

        return (rc);
}

        __checkReturn   int
qt2022c2_oui_get(
        __in            efx_nic_t *enp,
        __out           uint32_t *ouip)
{
        efx_port_t *epp = &(enp->en_port);
        int rc;

        if ((rc = xphy_mmd_oui_get(enp, epp->ep_port, PMA_PMD_MMD, ouip)) != 0)
                goto fail1;

        return (0);

fail1:
```



```
        EFSYS_PROBE1(fail1, int, rc);

        return (rc);
}

#if EFSYS_OPT_PHY_STATS

#define QT2022C2_STAT_SET(_stat, _mask, _id, _val)                      \
        do {                                                            \
                (_mask) |= (1 << (_id));                                \
                (_stat)[_id] = (uint32_t)(_val);                        \
        _NOTE(CONSTANTCONDITION)                                        \
        } while (B_FALSE)

static  __checkReturn                   int
qt2022c2_pma_pmd_stats_update(
        __in                            efx_nic_t *enp,
        __inout                         uint64_t *maskp,
        __inout_ecount(EFX_PHY_NSTATS)  uint32_t *stat)
{
        efx_port_t *epp = &(enp->en_port);
        efx_word_t word;
        int rc;

        if ((rc = falcon_mdio_read(enp, epp->ep_port, PMA_PMD_MMD,
            PMA_PMD_STATUS1_REG, &word)) != 0)
                goto fail1;

        QT2022C2_STAT_SET(stat, *maskp, EFX_PHY_STAT_PMA_PMD_LINK_UP,
            (EFX_WORD_FIELD(word, PMA_PMD_LINK_UP) != 0) ? 1 : 0);

        if ((rc = falcon_mdio_read(enp, epp->ep_port, PMA_PMD_MMD,
            PMA_PMD_STATUS2_REG, &word)) != 0)
                goto fail2;

        QT2022C2_STAT_SET(stat, *maskp, EFX_PHY_STAT_PMA_PMD_RX_FAULT,
            (EFX_WORD_FIELD(word, PMA_PMD_RX_FAULT) != 0) ? 1 : 0);
        QT2022C2_STAT_SET(stat, *maskp, EFX_PHY_STAT_PMA_PMD_TX_FAULT,
            (EFX_WORD_FIELD(word, PMA_PMD_TX_FAULT) != 0) ? 1 : 0);

        return (0);

fail2:
        EFSYS_PROBE(fail2);
fail1:
        EFSYS_PROBE1(fail1, int, rc);

        return (rc);
}

static  __checkReturn                   int
qt2022c2_pcs_stats_update(
        __in                            efx_nic_t *enp,
        __inout                         uint64_t *maskp,
        __inout_ecount(EFX_PHY_NSTATS)  uint32_t *stat)
{
        efx_port_t *epp = &(enp->en_port);
        efx_word_t word;
        int rc;

        if ((rc = falcon_mdio_read(enp, epp->ep_port, PCS_MMD,
            PCS_STATUS1_REG, &word)) != 0)
                goto fail1;

        QT2022C2_STAT_SET(stat, *maskp, EFX_PHY_STAT_PCS_LINK_UP,
            (EFX_WORD_FIELD(word, PCS_LINK_UP) != 0) ? 1 : 0);
```

```
393         if ((rc = falcon_mdio_read(enp, epp->ep_port, PCS_MMD,
394             PCS_STATUS2_REG, &word)) != 0)
395                 goto fail2;
396
397         QT2022C2_STAT_SET(stat, *maskp, EFX_PHY_STAT_PCS_RX_FAULT,
398             (EFX_WORD_FIELD(word, PCS_RX_FAULT) != 0) ? 1 : 0);
399         QT2022C2_STAT_SET(stat, *maskp, EFX_PHY_STAT_PCS_TX_FAULT,
400             (EFX_WORD_FIELD(word, PCS_TX_FAULT) != 0) ? 1 : 0);
401
402         if ((rc = falcon_mdio_read(enp, epp->ep_port, PCS_MMD,
403             PCS_10GBASE_R_STATUS2_REG, &word)) != 0)
404                 goto fail3;
405
406         QT2022C2_STAT_SET(stat, *maskp, EFX_PHY_STAT_PCS_BER,
407             EFX_WORD_FIELD(word, PCS_BER));
408         QT2022C2_STAT_SET(stat, *maskp, EFX_PHY_STAT_PCS_BLOCK_ERRORS,
409             EFX_WORD_FIELD(word, PCS_ERR));
410
411         return (0);
412
413 fail3:
414         EFSYS_PROBE(fail3);
415 fail2:
416         EFSYS_PROBE(fail2);
417 fail1:
418         EFSYS_PROBE1(fail1, int, rc);
419
420         return (rc);
421 }
422
423 static  __checkReturn                   int
424 qt2022c2_phy_xs_stats_update(
425         __in                            efx_nic_t *enp,
426         __inout                         uint64_t *maskp,
427         __inout_ecount(EFX_PHY_NSTATS)  uint32_t *stat)
428 {
429         efx_port_t *epp = &(enp->en_port);
430         efx_word_t word;
431         int rc;
432
433         if ((rc = falcon_mdio_read(enp, epp->ep_port, PHY_XS_MMD,
434             PHY_XS_STATUS1_REG, &word)) != 0)
435                 goto fail1;
436
437         QT2022C2_STAT_SET(stat, *maskp, EFX_PHY_STAT_PHY_XS_LINK_UP,
438             (EFX_WORD_FIELD(word, PHY_XS_LINK_UP) != 0) ? 1 : 0);
439
440         if ((rc = falcon_mdio_read(enp, epp->ep_port, PHY_XS_MMD,
441             PHY_XS_STATUS2_REG, &word)) != 0)
442                 goto fail2;
443
444         QT2022C2_STAT_SET(stat, *maskp, EFX_PHY_STAT_PHY_XS_RX_FAULT,
445             (EFX_WORD_FIELD(word, PHY_XS_RX_FAULT) != 0) ? 1 : 0);
446         QT2022C2_STAT_SET(stat, *maskp, EFX_PHY_STAT_PHY_XS_TX_FAULT,
447             (EFX_WORD_FIELD(word, PHY_XS_TX_FAULT) != 0) ? 1 : 0);
448
449         if ((rc = falcon_mdio_read(enp, epp->ep_port, PHY_XS_MMD,
450             PHY_XS_LANE_STATUS_REG, &word)) != 0)
451                 goto fail3;
452
453         QT2022C2_STAT_SET(stat, *maskp, EFX_PHY_STAT_PHY_XS_ALIGN,
454             (EFX_WORD_FIELD(word, PHY_XS_ALIGNED) != 0) ? 1 : 0);
455         QT2022C2_STAT_SET(stat, *maskp, EFX_PHY_STAT_PHY_XS_SYNC_A,
456             (EFX_WORD_FIELD(word, PHY_XS_LANE0_SYNC) != 0) ? 1 : 0);
457         QT2022C2_STAT_SET(stat, *maskp, EFX_PHY_STAT_PHY_XS_SYNC_B,
```



```
458             (EFX_WORD_FIELD(word, PHY_XS_LANE1_SYNC) != 0) ? 1 : 0);
459         QT2022C2_STAT_SET(stat, *maskp, EFX_PHY_STAT_PHY_XS_SYNC_C,
460             (EFX_WORD_FIELD(word, PHY_XS_LANE2_SYNC) != 0) ? 1 : 0);
461         QT2022C2_STAT_SET(stat, *maskp, EFX_PHY_STAT_PHY_XS_SYNC_D,
462             (EFX_WORD_FIELD(word, PHY_XS_LANE3_SYNC) != 0) ? 1 : 0);
463
464         return (0);
465
466 fail3:
467         EFSYS_PROBE(fail3);
468 fail2:
469         EFSYS_PROBE(fail2);
470 fail1:
471         EFSYS_PROBE1(fail1, int, rc);
472
473         return (rc);
474 }
475
476         __checkReturn                   int
477 qt2022c2_stats_update(
478         __in                            efx_nic_t *enp,
479         __in                            efsys_mem_t *esmp,
480         __out_ecount(EFX_PHY_NSTATS)    uint32_t *stat)
481 {
482         efx_port_t *epp = &(enp->en_port);
483         efx_nic_cfg_t *encp = &(enp->en_nic_cfg);
484         uint32_t oui;
485         uint64_t mask = 0;
486         int rc;
487
488         _NOTE(ARGUNUSED(esmp))
489
490         if ((rc = xphy_mmd_oui_get(enp, epp->ep_port, PMA_PMD_MMD, &oui)) != 0)
491                 goto fail1;
492
493         QT2022C2_STAT_SET(stat, mask, EFX_PHY_STAT_OUI, oui);
494
495         if ((rc = qt2022c2_pma_pmd_stats_update(enp, &mask, stat)) != 0)
496                 goto fail2;
497
498         if ((rc = qt2022c2_pcs_stats_update(enp, &mask, stat)) != 0)
499                 goto fail3;
500
501         if ((rc = qt2022c2_phy_xs_stats_update(enp, &mask, stat)) != 0)
502                 goto fail4;
503
504         /* Ensure all the supported statistics are up to date */
505         EFSYS_ASSERT(mask == encp->enc_phy_stat_mask);
506
507         return (0);
508
509 fail4:
510         EFSYS_PROBE(fail4);
511 fail3:
512         EFSYS_PROBE(fail3);
513 fail2:
514         EFSYS_PROBE(fail2);
515 fail1:
516         EFSYS_PROBE1(fail1, int, rc);
517
518         return (rc);
519 }
520 #endif  /* EFSYS_OPT_PHY_STATS */
521
522 #if EFSYS_OPT_PHY_PROPS
```

```
524 #if EFSYS_OPT_NAMES
525                 const char __cs *
526 qt2022c2_prop_name(
527         __in    efx_nic_t *enp,
528         __in    unsigned int id)
529 {
530         _NOTE(ARGUNUSED(enp, id))

532         EFSYS_ASSERT(B_FALSE);

534         return (NULL);
535 }
536 #endif  /* EFSYS_OPT_NAMES */

538         __checkReturn   int
539 qt2022c2_prop_get(
540         __in            efx_nic_t *enp,
541         __in            unsigned int id,
542         __in            uint32_t flags,
543         __out           uint32_t *valp)
544 {
545         _NOTE(ARGUNUSED(enp, id, flags, valp))

547         EFSYS_ASSERT(B_FALSE);

549         return (ENOTSUP);
550 }

552         __checkReturn   int
553 qt2022c2_prop_set(
554         __in            efx_nic_t *enp,
555         __in            unsigned int id,
556         __in            uint32_t val)
557 {
558         _NOTE(ARGUNUSED(enp, id, val))

560         EFSYS_ASSERT(B_FALSE);

562         return (ENOTSUP);
563 }
564 #endif  /* EFSYS_OPT_PHY_PROPS */

566 #endif  /* EFSYS_OPT_PHY_QT2022C2 */
567 #endif /* ! codereview */
```

```
**********************************************************
    4163 Thu Aug 22 18:59:26 2013
new/usr/src/uts/common/io/sfxge/qt2022c2.h
Merged sfxge driver
**********************************************************
```

```
/*-
 * Copyright 2007-2013 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#ifndef _SYS_QT2022C2_H
#define _SYS_QT2022C2_H

#include "efx.h"

#ifdef  __cplusplus
extern "C" {
#endif

#if EFSYS_OPT_PHY_QT2022C2

#define QT2022C2_LOOPBACK_MASK                                          \
            ((1 << EFX_LOOPBACK_PHY_XS) |                               \
            (1 << EFX_LOOPBACK_PCS) |                                   \
            (1 << EFX_LOOPBACK_PMA_PMD) |                               \
            FALCON_XMAC_LOOPBACK_MASK)

#define QT2022C2_LED_MASK                                               \
            ((1 << EFX_PHY_LED_OFF) |                                   \
            (1 << EFX_PHY_LED_ON))

#define QT2022C2_NPROPS         0

#define QT2022C2_ADV_CAP_MASK                                           \
            ((1 << EFX_PHY_CAP_10000FDX) |                              \
            (1 << EFX_PHY_CAP_PAUSE))

#define QT2022C2_ADV_CAP_PERM   0

#define QT2022C2_BIST_MASK      0

extern  __checkReturn   int
qt2022c2_reset(
        __in            efx_nic_t *enp);

extern  __checkReturn   int
```



```
qt2022c2_reconfigure(
        __in            efx_nic_t *enp);

extern  __checkReturn   int
qt2022c2_verify(
        __in            efx_nic_t *enp);

extern  __checkReturn   int
qt2022c2_uplink_check(
        __in            efx_nic_t *enp,
        __out           boolean_t *upp);

extern  __checkReturn   int
qt2022c2_downlink_check(
        __in            efx_nic_t *enp,
        __out           efx_link_mode_t *modep,
        __out           unsigned int *fcntlp,
        __out           uint32_t *lp_cap_maskp);

extern  __checkReturn   int
qt2022c2_oui_get(
        __in            efx_nic_t *enp,
        __out           uint32_t *ouip);

#if EFSYS_OPT_PHY_STATS

/* START MKCONFIG GENERATED Qt2022c2PhyHeaderStatsMask 5655dc14f9b46071 */
#define QT2022C2_STAT_MASK \
        (1ULL << EFX_PHY_STAT_OUI) | \
        (1ULL << EFX_PHY_STAT_PMA_PMD_LINK_UP) | \
        (1ULL << EFX_PHY_STAT_PMA_PMD_RX_FAULT) | \
        (1ULL << EFX_PHY_STAT_PMA_PMD_TX_FAULT) | \
        (1ULL << EFX_PHY_STAT_PCS_LINK_UP) | \
        (1ULL << EFX_PHY_STAT_PCS_RX_FAULT) | \
        (1ULL << EFX_PHY_STAT_PCS_TX_FAULT) | \
        (1ULL << EFX_PHY_STAT_PCS_BER) | \
        (1ULL << EFX_PHY_STAT_PCS_BLOCK_ERRORS) | \
        (1ULL << EFX_PHY_STAT_PHY_XS_LINK_UP) | \
        (1ULL << EFX_PHY_STAT_PHY_XS_RX_FAULT) | \
        (1ULL << EFX_PHY_STAT_PHY_XS_TX_FAULT) | \
        (1ULL << EFX_PHY_STAT_PHY_XS_ALIGN) | \
        (1ULL << EFX_PHY_STAT_PHY_XS_SYNC_A) | \
        (1ULL << EFX_PHY_STAT_PHY_XS_SYNC_B) | \
        (1ULL << EFX_PHY_STAT_PHY_XS_SYNC_C) | \
        (1ULL << EFX_PHY_STAT_PHY_XS_SYNC_D)

/* END MKCONFIG GENERATED Qt2022c2PhyHeaderStatsMask */

extern  __checkReturn                   int
qt2022c2_stats_update(
        __in                            efx_nic_t *enp,
        __in                            efsys_mem_t *esmp,
        __out_ecount(EFX_PHY_NSTATS)    uint32_t *stat);

#endif  /* EFSYS_OPT_PHY_STATS */

#if EFSYS_OPT_PHY_PROPS

#if EFSYS_OPT_NAMES

extern          const char __cs *
qt2022c2_prop_name(
        __in    efx_nic_t *enp,
        __in    unsigned int id);

#endif
```

```
129 extern  __checkReturn   int
130 qt2022c2_prop_get(
131         __in            efx_nic_t *enp,
132         __in            unsigned int id,
133         __in            uint32_t flags,
134         __out           uint32_t *valp);

136 extern  __checkReturn   int
137 qt2022c2_prop_set(
138         __in            efx_nic_t *enp,
139         __in            unsigned int id,
140         __in            uint32_t val);

142 #endif  /* EFSYS_OPT_PHY_PROPS */

144 #endif  /* EFSYS_OPT_PHY_QT2022C2 */

146 #ifdef  __cplusplus
147 }
148 #endif

150 #endif  /* _SYS_QT2022C2_H */
151 #endif /* ! codereview */
```

```
**********************************************************
    2297 Thu Aug 22 18:59:26 2013
new/usr/src/uts/common/io/sfxge/qt2022c2_impl.h
Merged sfxge driver
**********************************************************
```

```
/*-
 * Copyright 2007-2013 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#ifndef _SYS_QT2022C2_IMPL_H
#define _SYS_QT2022C2_IMPL_H

#ifdef  __cplusplus
extern "C" {
#endif

#if EFSYS_OPT_PHY_QT2022C2

#define QT2022C2_MMD_MASK                                               \
            ((1 << PMA_PMD_MMD) |                                       \
            (1 << PCS_MMD) |                                            \
            (1 << PHY_XS_MMD))

#define PMA_PMD_LED1_REG 0xd006 /* Green */
#define PMA_PMD_LED2_REG 0xd007 /* Amber */
#define PMA_PMD_LED3_REG 0xd008 /* Red */

#define PMA_PMD_LED_CFG_LBN 0
#define PMA_PMD_LED_CFG_WIDTH 3
#define LED_CFG_LS_DECODE 0x1
#define LED_CFG_LA_DECODE 0x2
#define LED_CFG_LSA_DECODE 0x3
#define LED_CFG_OFF_DECODE 0x4
#define LED_CFG_ON_DECODE 0x5
#define PMA_PMD_LED_PATH_LBN 3
#define PMA_PMD_LED_PATH_WIDTH 1
#define LED_PATH_TX_DECODE 0x0
#define LED_PATH_RX_DECODE 0x1

#define PHY_XS_VENDOR0_REG 0xc000
#define XAUI_SYSTEM_LOOPBACK_LBN 14
#define XAUI_SYSTEM_LOOPBACK_WIDTH 1

#endif  /* EFSYS_OPT_PHY_QT2022C2 */
```



```
#ifdef  __cplusplus
}
#endif

#endif  /* _SYS_QT2022C2_IMPL_H */
#endif /* ! codereview */
```

```
**********************************************************
   26081 Thu Aug 22 18:59:26 2013
new/usr/src/uts/common/io/sfxge/qt2025c.c
Merged sfxge driver
**********************************************************
```

```
/*
 * Copyright 2008-2013 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#include "efsys.h"
#include "efx.h"
#include "efx_types.h"
#include "efx_regs.h"
#include "efx_impl.h"
#include "xphy.h"
#include "qt2025c.h"
#include "qt2025c_impl.h"
#include "falcon_impl.h"

#if EFSYS_OPT_PHY_QT2025C

static  __checkReturn   int
qt2025c_led_cfg(
        __in            efx_nic_t *enp)
{
        efx_port_t *epp = &(enp->en_port);
        efx_word_t led1;
        efx_word_t led2;
        int rc;

#if EFSYS_OPT_PHY_LED_CONTROL

        switch (epp->ep_phy_led_mode) {
        case EFX_PHY_LED_DEFAULT:
                EFX_POPULATE_WORD_2(led1, PMA_PMD_LED_CFG, LED_CFG_LA_DECODE,
                    PMA_PMD_LED_PATH, LED_PATH_RX_DECODE);
                EFX_POPULATE_WORD_2(led2, PMA_PMD_LED_CFG, LED_CFG_LS_DECODE,
                    PMA_PMD_LED_PATH, LED_PATH_RX_DECODE);
                break;

        case EFX_PHY_LED_OFF:
                EFX_POPULATE_WORD_1(led1, PMA_PMD_LED_CFG, LED_CFG_OFF_DECODE);
                EFX_POPULATE_WORD_1(led2, PMA_PMD_LED_CFG, LED_CFG_OFF_DECODE);
                break;
```



```
        case EFX_PHY_LED_ON:
                EFX_POPULATE_WORD_1(led1, PMA_PMD_LED_CFG, LED_CFG_ON_DECODE);
                EFX_POPULATE_WORD_1(led2, PMA_PMD_LED_CFG, LED_CFG_ON_DECODE);
                break;

        default:
                EFSYS_ASSERT(B_FALSE);
                break;
        }

#else   /* EFSYS_OPT_PHY_LED_CONTROL */

        EFX_POPULATE_WORD_2(led1, PMA_PMD_LED_CFG, LED_CFG_LA_DECODE,
            PMA_PMD_LED_PATH, LED_PATH_RX_DECODE);
        EFX_POPULATE_WORD_2(led2, PMA_PMD_LED_CFG, LED_CFG_LS_DECODE,
            PMA_PMD_LED_PATH, LED_PATH_RX_DECODE);

#endif  /* EFSYS_OPT_PHY_LED_CONTROL */

        if ((rc = falcon_mdio_write(enp, epp->ep_port, PMA_PMD_MMD,
            PMA_PMD_LED1_REG, &led1)) != 0)
                goto fail1;

        if ((rc = falcon_mdio_write(enp, epp->ep_port, PMA_PMD_MMD,
            PMA_PMD_LED2_REG, &led2)) != 0)
                goto fail2;

        return (0);

fail2:
        EFSYS_PROBE(fail2);
fail1:
        EFSYS_PROBE1(fail1, int, rc);

        return (rc);
}

#if EFSYS_OPT_LOOPBACK
static  __checkReturn   int
qt2025c_loopback_cfg(
        __in            efx_nic_t *enp)
{
        efx_port_t *epp = &(enp->en_port);
        efx_word_t word;
        int rc;

        /* Set static mode appropriately */
        if ((rc = falcon_mdio_read(enp, epp->ep_port, PMA_PMD_MMD,
            PMA_FTX_CTRL2_REG, &word)) != 0)
                goto fail1;

        EFX_SET_WORD_FIELD(word, FTX_STATIC,
            (epp->ep_loopback_type == EFX_LOOPBACK_PCS ||
            epp->ep_loopback_type == EFX_LOOPBACK_PMA_PMD) ? 1 : 0);

        /* Set static mode appropriately */
        if ((rc = falcon_mdio_write(enp, epp->ep_port, PMA_PMD_MMD,
            PMA_FTX_CTRL2_REG, &word)) != 0)
                goto fail2;

        switch (epp->ep_loopback_type) {
        case EFX_LOOPBACK_PHY_XS: {
                efx_word_t xsword;

                if ((rc = falcon_mdio_read(enp, epp->ep_port, PHY_XS_MMD,
                    XGXS_VENDOR_SPECIFIC_1_REG, &xsword)) != 0)
```

```
128                         goto fail3;

130                 EFX_SET_WORD_FIELD(xsword, XGXS_SYSLPBK, 1);

132                 if ((rc = falcon_mdio_write(enp, epp->ep_port, PHY_XS_MMD,
133                     XGXS_VENDOR_SPECIFIC_1_REG, &xsword)) != 0)
134                         goto fail4;

136                 break;
137         }
138         case EFX_LOOPBACK_PCS:
139                 if ((rc = xphy_mmd_loopback_set(enp, epp->ep_port, PCS_MMD,
140                     B_TRUE)) != 0)
141                         goto fail3;

143                 break;

145         case EFX_LOOPBACK_PMA_PMD:
146                 if ((rc = xphy_mmd_loopback_set(enp, epp->ep_port, PMA_PMD_MMD,
147                     B_TRUE)) != 0)
148                         goto fail3;

150                 break;

152         default:
153                 break;
154         }

156         return (0);

158 fail4:
159         EFSYS_PROBE(fail4);
160 fail3:
161         EFSYS_PROBE(fail3);
162 fail2:
163         EFSYS_PROBE(fail2);
164 fail1:
165         EFSYS_PROBE1(fail1, int, rc);

167         return (rc);
168 }
169 #endif  /* EFSYS_OPT_LOOPBACK */

171 static  __checkReturn   int
172 qt2025c_version_get(
173         __in            efx_nic_t *enp,
174         __out_ecount(4) uint8_t version[])
175 {
176         efx_port_t *epp = &(enp->en_port);
177         efx_word_t word;
178         int rc;

180         if ((rc = falcon_mdio_read(enp, epp->ep_port, PCS_MMD,
181             FW_VERSION1_REG, &word)) != 0)
182                 goto fail1;

184         version[0] = (uint8_t)EFX_WORD_FIELD(word, EFX_BYTE_0) >> 4;
185         version[1] = (uint8_t)EFX_WORD_FIELD(word, EFX_BYTE_0) & 0x0f;

187         if ((rc = falcon_mdio_read(enp, epp->ep_port, PCS_MMD,
188             FW_VERSION2_REG, &word)) != 0)
189                 goto fail2;

191         version[2] = (uint8_t)EFX_WORD_FIELD(word, EFX_BYTE_0);

193         if ((rc = falcon_mdio_read(enp, epp->ep_port, PCS_MMD,
```



```
194             FW_VERSION3_REG, &word)) != 0)
195                 goto fail3;

197         version[3] = (uint8_t)EFX_WORD_FIELD(word, EFX_BYTE_0);

199         return (0);

201 fail3:
202         EFSYS_PROBE(fail3);
203 fail2:
204         EFSYS_PROBE(fail2);
205 fail1:
206         EFSYS_PROBE1(fail1, int, rc);

208         return (rc);
209 }

211 /*
212  * Convinience function so that the large number of MDIO writes
213  * in qt2025c_select_phy_mode are condensed, and easier to compare
214  * directly with the data sheet.
215  */
216 static  __checkReturn   int
217 qt2025c_mdio_write(
218         __in            efx_nic_t *enp,
219         __in            uint8_t mmd,
220         __in            uint16_t addr,
221         __in            uint16_t value)
222 {
223         efx_port_t *epp = &(enp->en_port);
224         efx_word_t word;
225         int rc;

227         EFX_POPULATE_WORD_1(word, EFX_WORD_0, value);
228         rc = falcon_mdio_write(enp, epp->ep_port, mmd, addr, &word);

230         return (rc);
231 }

233 static  __checkReturn   int
234 qt2025c_restart_firmware(
235         __in            efx_nic_t *enp)
236 {
237         int rc;

239         /* Restart microcontroller execution of firmware from RAM */
240         if ((rc = qt2025c_mdio_write(enp, 3, 0xe854, 0x00c0)) != 0)
241                 goto fail1;

243         if ((rc = qt2025c_mdio_write(enp, 3, 0xe854, 0x0040)) != 0)
244                 goto fail2;

246         EFSYS_SLEEP(50000);     /* 50ms */

248         return (0);
249 fail2:
250         EFSYS_PROBE(fail2);
251 fail1:
252         EFSYS_PROBE1(fail1, int, rc);

254         return (rc);
255 }

257 static  __checkReturn   int
258 qt2025c_wait_heartbeat(
259         __in            efx_nic_t *enp)
```

```
{
        efx_port_t *epp = &(enp->en_port);
        efx_word_t word;
        unsigned int count;
        uint8_t heartb;
        int rc;

        /* Read the initial heartbeat */
        if ((rc = falcon_mdio_read(enp, epp->ep_port, PCS_MMD,
            FW_HEARTBEAT_REG, &word)) != 0)
                goto fail1;

        heartb = EFX_WORD_FIELD(word, FW_HEARTB);

        /* Wait for the heartbeat to start updating */
        count = 0;
        do {
                EFSYS_PROBE1(wait, unsigned int, count);

                EFSYS_SLEEP(100000);    /* 100ms */

                if ((rc = falcon_mdio_read(enp, epp->ep_port, PCS_MMD,
                    FW_HEARTBEAT_REG, &word)) != 0)
                        goto fail2;

                if (EFX_WORD_FIELD(word, FW_HEARTB) != heartb)
                        return (0);

        } while (++count < 50);         /* For up to 5s */

        rc = ENOTACTIVE;
        EFSYS_PROBE(fail3);
fail2:
        EFSYS_PROBE(fail2);
fail1:
        EFSYS_PROBE1(fail1, int, rc);

        return (rc);
}

static  __checkReturn   int
qt2025c_wait_firmware(
        __in            efx_nic_t *enp)
{
        efx_port_t *epp = &(enp->en_port);
        efx_word_t word;
        unsigned int count;
        uint16_t status;
        int rc;

        count = 0;
        do {
                EFSYS_PROBE1(wait, unsigned int, count);

                EFSYS_SLEEP(100000);    /* 100ms */

                if ((rc = falcon_mdio_read(enp, epp->ep_port, PCS_MMD,
                    UC8051_STATUS_REG, &word)) != 0)
                        goto fail1;

                status = EFX_WORD_FIELD(word, UC_STATUS);
                if (status >= UC_STATUS_FW_SAVE_DECODE)
                        return (0);

        } while (++count < 25);         /* For up to 2.5s */
```



```
        rc = ENOTACTIVE;
        EFSYS_PROBE(fail2);
fail1:
        EFSYS_PROBE1(fail1, int, rc);

        return (rc);
}

static  __checkReturn   int
qt2025c_wait_reset(
        __in            efx_nic_t *enp)
{
        boolean_t retry = B_TRUE;
        int rc;

        /*
         * Bug17689: occasionally heartbeat starts but firmware status
         * code never progresses beyond 0x00.  Try again, once, after
         * restarting execution of the firmware image.
         */
again:
        if ((rc = qt2025c_wait_heartbeat(enp)) != 0)
                goto fail1;

        rc = qt2025c_wait_firmware(enp);
        if (rc == ENOTACTIVE && retry) {
                if ((rc = qt2025c_restart_firmware(enp)) != 0)
                        goto fail2;

                retry = B_FALSE;
                goto again;
        }
        if (rc != 0)
                goto fail3;

        return (0);

fail3:
        EFSYS_PROBE(fail3);
fail2:
        EFSYS_PROBE(fail2);
fail1:
        EFSYS_PROBE1(fail1, int, rc);

        return (rc);
}

        __checkReturn   int
qt2025c_reset(
        __in            efx_nic_t *enp)
{
        efx_port_t *epp = &(enp->en_port);
        uint8_t version[4];
        int rc;

        /* Pull the external reset line */
        falcon_nic_phy_reset(enp);

        /* Wait for the reset to complete */
        if ((rc = qt2025c_wait_reset(enp)) != 0)
                goto fail1;

        /* Read the firmware version again post-reset */
        if ((rc = qt2025c_version_get(enp, version)) != 0)
                goto fail2;
        epp->ep_fwver = \
```

```
392             ((uint32_t)version[0] << 24 | (uint32_t)version[1] << 16 | \
393             (uint32_t)version[2] << 8 | (uint32_t)version[3]);

395         return (0);

397 fail2:
398         EFSYS_PROBE(fail2);
399 fail1:
400         EFSYS_PROBE1(fail1, int, rc);

402         return (rc);
403 }

405 static  __checkReturn   int
406 qt2025c_select_phy_mode(
407         __in            efx_nic_t *enp)
408 {
409         efx_port_t *epp = &(enp->en_port);
410         efx_word_t word;
411         uint16_t phy_op_mode;
412         int i;
413         int rc;

415         /*
416          * Only 2.0.1.0+ PHY firmware supports the more optimal SFP+
417          * Self-Configure mode.  Don't attempt any switching if we encounter
418          * older firmware.
419          *
420          * NOTE: This code uses explicit integers for MMD's and register
421          * addresses so that it is trivially comparable to the firmware
422          * release notes.
423          */
424         if (epp->ep_fwver < 0x02000100)
425                 return (0);

427         /*
428          * In general we will get optimal behaviour in "SFP+ Self-Configure"
429          * mode; however, that powers down most of the PHY when no module is
430          * present, so we must use a different mode (any fixed mode will do)
431          * to be sure that loopbacks will work.
432          */
433         phy_op_mode = 0x0038;
434 #if EFSYS_OPT_LOOPBACK
435         if (epp->ep_loopback_type != EFX_LOOPBACK_OFF)
436                 phy_op_mode = 0x0020;
437 #endif  /* EFSYS_OPT_LOOPBACK */

439         /* Only change mode if really necessary */
440         if ((rc = falcon_mdio_read(enp, epp->ep_port, 1, 0xc319, &word)) != 0)
441                 goto fail1;
442         if ((EFX_WORD_FIELD(word, EFX_WORD_0) & 0x0038) == phy_op_mode)
443                 return (0);

445         /* Do a full reset and wait for it to complete */
446         if ((rc = qt2025c_mdio_write(enp, 1, 0x0000, 0x8000)) != 0)
447                 goto fail2;
448         EFSYS_SLEEP(50000);             /* 50ms */

450         if ((rc = qt2025c_wait_reset(enp)) != 0)
451                 goto fail3;

453         /*
454          * This sequence replicates the register writes configured in the boot
455          * EEPROM (including the differences between board revisions), except
456          * that the operating mode is changed, and the PHY is prevented from
457          * unnecessarily reloading the main firmware image again.
```



```
458          */
459         if ((rc = qt2025c_mdio_write(enp, 1, 0xc300, 0)) != 0)
460                 goto fail4;

462         /*
463          * (Note: this portion of the boot EEPROM sequence, which bit-bashes 9
464          * STOPs onto the firmware/module I2C bus to reset it, varies across
465          * board revisions, as the bus is connected to different GPIO/LED
466          * outputs on the PHY.)
467          */
468         if (enp->en_u.falcon.enu_board_rev < 2) {
469                 if ((rc = qt2025c_mdio_write(enp, 1, 0xc303, 0x4498)) != 0)
470                         goto fail5;
471                 for (i = 0; i < 9; i++) {
472                         rc = qt2025c_mdio_write(enp, 1, 0xc303, 0x4488);
473                         if (rc != 0)
474                                 goto fail5;
475                         rc = qt2025c_mdio_write(enp, 1, 0xc303, 0x4480);
476                         if (rc != 0)
477                                 goto fail5;
478                         rc = qt2025c_mdio_write(enp, 1, 0xc303, 0x4490);
479                         if (rc != 0)
480                                 goto fail5;
481                         rc = qt2025c_mdio_write(enp, 1, 0xc303, 0x4498);
482                         if (rc != 0)
483                                 goto fail5;
484                 }
485         } else {
486                 if ((rc = qt2025c_mdio_write(enp, 1, 0xc303, 0x0920)) != 0)
487                         goto fail5;
488                 if ((rc = qt2025c_mdio_write(enp, 1, 0xd008, 0x0004)) != 0)
489                         goto fail5;
490                 for (i = 0; i < 9; i++) {
491                         rc = qt2025c_mdio_write(enp, 1, 0xc303, 0x0900);
492                         if (rc != 0)
493                                 goto fail5;
494                         rc = qt2025c_mdio_write(enp, 1, 0xd008, 0x0005);
495                         if (rc != 0)
496                                 goto fail5;
497                         rc = qt2025c_mdio_write(enp, 1, 0xc303, 0x0920);
498                         if (rc != 0)
499                                 goto fail5;
500                         rc = qt2025c_mdio_write(enp, 1, 0xd008, 0x0004);
501                         if (rc != 0)
502                                 goto fail5;
503                 }
504                 if ((rc = qt2025c_mdio_write(enp, 1, 0xc303, 0x4900)) != 0)
505                         goto fail5;
506         }

508         if ((rc = qt2025c_mdio_write(enp, 1, 0xc303, 0x4900)) != 0)
509                 goto fail6;
510         if ((rc = qt2025c_mdio_write(enp, 1, 0xc302, 0x0004)) != 0)
511                 goto fail6;
512         if ((rc = qt2025c_mdio_write(enp, 1, 0xc316, 0x0013)) != 0)
513                 goto fail6;
514         if ((rc = qt2025c_mdio_write(enp, 1, 0xc318, 0x0054)) != 0)
515                 goto fail6;
516         if ((rc = qt2025c_mdio_write(enp, 1, 0xc319, phy_op_mode)) != 0)
517                 goto fail6;
518         if ((rc = qt2025c_mdio_write(enp, 1, 0xc31a, 0x0098)) != 0)
519                 goto fail6;
520         if ((rc = qt2025c_mdio_write(enp, 3, 0x0026, 0x0e00)) != 0)
521                 goto fail6;
522         if ((rc = qt2025c_mdio_write(enp, 3, 0x0027, 0x0013)) != 0)
523                 goto fail6;
```

```
524         if ((rc = qt2025c_mdio_write(enp, 3, 0x0028, 0xa528)) != 0)
525                 goto fail6;
526         if ((rc = qt2025c_mdio_write(enp, 1, 0xd006, 0x000a)) != 0)
527                 goto fail6;
528         if ((rc = qt2025c_mdio_write(enp, 1, 0xd007, 0x0009)) != 0)
529                 goto fail6;
530         if ((rc = qt2025c_mdio_write(enp, 1, 0xd008, 0x0004)) != 0)
531                 goto fail6;
532         /*
533          * This additional write prevents the PHY boot ROM doing another
534          * pointless reload of the firmware image (the microcontroller's
535          * code memory is not affected by the microcontroller reset).
536          */
537         if ((rc = qt2025c_mdio_write(enp, 1, 0xc317, 0x00ff)) != 0)
538                 goto fail7;
539         if ((rc = qt2025c_mdio_write(enp, 1, 0xc300, 0x0002)) != 0)
540                 goto fail7;
541         EFSYS_SLEEP(20000);             /* 20ms */

543         /* Restart microcontroller execution from RAM */
544         if ((rc = qt2025c_restart_firmware(enp)) != 0)
545                 goto fail8;

547         /* Wait for the microcontroller to be ready again */
548         if ((rc = qt2025c_wait_reset(enp)) != 0)
549                 goto fail9;

551         return (0);

553 fail9:
554         EFSYS_PROBE(fail9);
555 fail8:
556         EFSYS_PROBE(fail8);
557 fail7:
558         EFSYS_PROBE(fail7);
559 fail6:
560         EFSYS_PROBE(fail6);
561 fail5:
562         EFSYS_PROBE(fail5);
563 fail4:
564         EFSYS_PROBE(fail4);
565 fail3:
566         EFSYS_PROBE(fail3);
567 fail2:
568         EFSYS_PROBE(fail2);
569 fail1:
570         EFSYS_PROBE1(fail1, int, rc);

572         return (rc);
573 }

575         __checkReturn   int
576 qt2025c_reconfigure(
577         __in            efx_nic_t *enp)
578 {
579         efx_port_t *epp = &(enp->en_port);
580         int rc;

582         if ((rc = qt2025c_select_phy_mode(enp)) != 0)
583                 goto fail1;

585         if ((rc = xphy_pkg_wait(enp, epp->ep_port, QT2025C_MMD_MASK)) != 0)
586                 goto fail2;

588         if ((rc = qt2025c_led_cfg(enp)) != 0)
589                 goto fail3;
```



```
591         EFSYS_ASSERT3U(epp->ep_adv_cap_mask, ==, QT2025C_ADV_CAP_MASK);

593 #if EFSYS_OPT_LOOPBACK
594         if ((rc = qt2025c_loopback_cfg(enp)) != 0)
595                 goto fail4;
596 #endif  /* EFSYS_OPT_LOOPBACK */

598         return (0);

600 #if EFSYS_OPT_LOOPBACK
601 fail4:
602         EFSYS_PROBE(fail4);
603 #endif  /* EFSYS_OPT_LOOPBACK */
604 fail3:
605         EFSYS_PROBE(fail3);
606 fail2:
607         EFSYS_PROBE(fail2);
608 fail1:
609         EFSYS_PROBE1(fail1, int, rc);

611         return (rc);
612 }

614         __checkReturn   int
615 qt2025c_verify(
616         __in            efx_nic_t *enp)
617 {
618         efx_port_t *epp = &(enp->en_port);
619         int rc;

621         if ((rc = xphy_pkg_verify(enp, epp->ep_port, QT2025C_MMD_MASK)) != 0)
622                 goto fail1;

624         return (0);

626 fail1:
627         EFSYS_PROBE1(fail1, int, rc);

629         return (rc);
630 }

632         __checkReturn   int
633 qt2025c_uplink_check(
634         __in            efx_nic_t *enp,
635         __out           boolean_t *upp)
636 {
637         efx_port_t *epp = &(enp->en_port);
638         efx_word_t word;
639         int rc;

641         if (epp->ep_mac_type != EFX_MAC_FALCON_XMAC) {
642                 rc = ENOTSUP;
643                 goto fail1;
644         }

646         if ((rc = falcon_mdio_read(enp, epp->ep_port, PHY_XS_MMD,
647             PHY_XS_LANE_STATUS_REG, &word)) != 0)
648                 goto fail2;

650         *upp = ((EFX_WORD_FIELD(word, PHY_XS_ALIGNED) != 0) &&
651             (EFX_WORD_FIELD(word, PHY_XS_LANE0_SYNC) != 0) &&
652             (EFX_WORD_FIELD(word, PHY_XS_LANE1_SYNC) != 0) &&
653             (EFX_WORD_FIELD(word, PHY_XS_LANE2_SYNC) != 0) &&
654             (EFX_WORD_FIELD(word, PHY_XS_LANE3_SYNC) != 0));
```

```
656         return (0);

658 fail2:
659         EFSYS_PROBE(fail2);
660 fail1:
661         EFSYS_PROBE1(fail1, int, rc);

663         return (rc);
664 }

666 /* Recover from bug17190 by moving into and out of PMA loopback */
667 static  __checkReturn   int
668 qt2025c_bug17190_workaround(
669         __in            efx_nic_t *enp)
670 {
671         efx_port_t *epp = &(enp->en_port);
672         efx_word_t word;
673         int rc;

675         if ((rc = falcon_mdio_read(enp, epp->ep_port,
676             PMA_PMD_MMD, MMD_CONTROL1_REG, &word)) != 0)
677                 goto fail1;
678         EFX_SET_WORD_FIELD(word, MMD_PMA_LOOPBACK, 1);
679         if ((rc = falcon_mdio_write(enp, epp->ep_port,
680             PMA_PMD_MMD, MMD_CONTROL1_REG, &word)) != 0)
681                 goto fail2;

683         EFSYS_SLEEP(100000);    /* 100ms */

685         EFX_SET_WORD_FIELD(word, MMD_PMA_LOOPBACK, 0);
686         if ((rc = falcon_mdio_write(enp, epp->ep_port,
687             PMA_PMD_MMD, MMD_CONTROL1_REG, &word)) != 0)
688                 goto fail3;

690         return (0);

692 fail3:
693         EFSYS_PROBE(fail3);
694 fail2:
695         EFSYS_PROBE(fail2);
696 fail1:
697         EFSYS_PROBE1(fail1, int, rc);

699         return (rc);
700 }

702         __checkReturn   int
703 qt2025c_downlink_check(
704         __in            efx_nic_t *enp,
705         __out           efx_link_mode_t *modep,
706         __out           unsigned int *fcntlp,
707         __out           uint32_t *lp_cap_maskp)
708 {
709         efx_port_t *epp = &(enp->en_port);
710         uint8_t mmd;
711         uint8_t mmds;
712         boolean_t broken;
713         boolean_t ok;
714         boolean_t up;
715         int rc;

717         mmds = (1 << PMA_PMD_MMD) | (1 << PCS_MMD) | (1 << PHY_XS_MMD);
718 #if EFSYS_OPT_LOOPBACK
719         switch (epp->ep_loopback_type) {
720         case EFX_LOOPBACK_PHY_XS:
721                 mmds = 0;
```



```
722                 break;
723         case EFX_LOOPBACK_PCS:
724                 mmds = (1 << PHY_XS_MMD);
725                 break;
726         case EFX_LOOPBACK_PMA_PMD:
727                 mmds = (1 << PHY_XS_MMD) | (1 << PCS_MMD);
728                 break;
729         default:
730                 break;
731         }
732 #endif /* EFSYS_OPT_LOOPBACK */

734         /*
735          * bug17190 is a persistent failure of PCS block lock when PMA
736          * and PHYXS are up. Only check for this when not in phy loopback.
737          */
738         broken = !!(mmds & (1 << PMA_PMD_MMD));

740         if (!mmds) {
741                 /* Use the fault state instead */
742                 if ((rc = xphy_mmd_fault(enp, epp->ep_port, &up)) != 0)
743                         goto fail1;

745         } else {
746                 up = B_TRUE;
747                 for (mmd = 0; mmd < MAXMMD; mmd++) {
748                         if (~mmds & (1 << mmd))
749                                 continue;

751                         rc = xphy_mmd_check(enp, epp->ep_port, mmd, &ok);
752                         if (rc != 0)
753                                 goto fail2;
754                         up &= ok;
755                         broken &= (ok == (mmd != PCS_MMD));
756                 }
757         }

759         /* Recover from bug17190 if we've been stuck there for 3 polls. */
760         if (!broken)
761                 epp->ep_qt2025c.bug17190_count = 0;
762         else if (++(epp->ep_qt2025c.bug17190_count) >= 3) {
763                 epp->ep_qt2025c.bug17190_count = 0;
764                 EFSYS_PROBE(bug17190_recovery);

766                 if ((rc = qt2025c_bug17190_workaround(enp)) != 0)
767                         goto fail3;
768         }

770         *modep = (up) ? EFX_LINK_10000FDX : EFX_LINK_DOWN;
771         *fcntlp = epp->ep_fcntl;
772         *lp_cap_maskp = epp->ep_lp_cap_mask;

774         return (0);

776 fail3:
777         EFSYS_PROBE(fail3);
778 fail2:
779         EFSYS_PROBE(fail2);
780 fail1:
781         EFSYS_PROBE1(fail1, int, rc);

783         return (rc);
784 }

786         __checkReturn   int
787 qt2025c_oui_get(
```

```
788         __in            efx_nic_t *enp,
789         __out           uint32_t *ouip)
790 {
791         efx_port_t *epp = &(enp->en_port);
792         int rc;

794         if ((rc = xphy_mmd_oui_get(enp, epp->ep_port, PMA_PMD_MMD, ouip)) != 0)
795                 goto fail1;

797         return (0);

799 fail1:
800         EFSYS_PROBE1(fail1, int, rc);

802         return (rc);
803 }

805 #if EFSYS_OPT_PHY_STATS

807 #define QT2025C_STAT_SET(_stat, _mask, _id, _val)                       \
808         do {                                                            \
809                 (_mask) |= (1ULL << (_id));                             \
810                 (_stat)[_id] = (uint32_t)(_val);                        \
811         _NOTE(CONSTANTCONDITION)                                        \
812         } while (B_FALSE)

814 static  __checkReturn   int
815 qt2025c_build_get(
816         __in            efx_nic_t *enp,
817         __out           uint8_t *yp,
818         __out           uint8_t *mp,
819         __out           uint8_t *dp)
820 {
821         efx_port_t *epp = &(enp->en_port);
822         efx_word_t word;
823         int rc;

825         if ((rc = falcon_mdio_read(enp, epp->ep_port, PCS_MMD,
826             FW_BUILD1_REG, &word)) != 0)
827                 goto fail1;

829         *yp = (uint8_t)EFX_WORD_FIELD(word, EFX_BYTE_0);

831         if ((rc = falcon_mdio_read(enp, epp->ep_port, PCS_MMD,
832             FW_BUILD2_REG, &word)) != 0)
833                 goto fail2;

835         *mp = (uint8_t)EFX_WORD_FIELD(word, EFX_BYTE_0);

837         if ((rc = falcon_mdio_read(enp, epp->ep_port, PCS_MMD,
838             FW_BUILD3_REG, &word)) != 0)
839                 goto fail3;

841         *dp = (uint8_t)EFX_WORD_FIELD(word, EFX_BYTE_0);

843         return (0);

845 fail3:
846         EFSYS_PROBE(fail3);
847 fail2:
848         EFSYS_PROBE(fail2);
849 fail1:
850         EFSYS_PROBE1(fail1, int, rc);

852         return (rc);
853 }
```



```
855 static  __checkReturn                   int
856 qt2025c_pma_pmd_stats_update(
857         __in                            efx_nic_t *enp,
858         __inout                         uint64_t *maskp,
859         __inout_ecount(EFX_PHY_NSTATS)  uint32_t *stat)
860 {
861         efx_port_t *epp = &(enp->en_port);
862         efx_word_t word;
863         int rc;

865         if ((rc = falcon_mdio_read(enp, epp->ep_port, PMA_PMD_MMD,
866             PMA_PMD_STATUS1_REG, &word)) != 0)
867                 goto fail1;

869         QT2025C_STAT_SET(stat, *maskp, EFX_PHY_STAT_PMA_PMD_LINK_UP,
870             (EFX_WORD_FIELD(word, PMA_PMD_LINK_UP) != 0) ? 1 : 0);

872         if ((rc = falcon_mdio_read(enp, epp->ep_port, PMA_PMD_MMD,
873             PMA_PMD_STATUS2_REG, &word)) != 0)
874                 goto fail2;

876         QT2025C_STAT_SET(stat, *maskp, EFX_PHY_STAT_PMA_PMD_RX_FAULT,
877             (EFX_WORD_FIELD(word, PMA_PMD_RX_FAULT) != 0) ? 1 : 0);
878         QT2025C_STAT_SET(stat, *maskp, EFX_PHY_STAT_PMA_PMD_TX_FAULT,
879             (EFX_WORD_FIELD(word, PMA_PMD_TX_FAULT) != 0) ? 1 : 0);

881         return (0);

883 fail2:
884         EFSYS_PROBE(fail2);
885 fail1:
886         EFSYS_PROBE1(fail1, int, rc);

888         return (rc);
889 }

891 static  __checkReturn                   int
892 qt2025c_pcs_stats_update(
893         __in                            efx_nic_t *enp,
894         __inout                         uint64_t *maskp,
895         __inout_ecount(EFX_PHY_NSTATS)  uint32_t *stat)
896 {
897         efx_port_t *epp = &(enp->en_port);
898         efx_word_t word;
899         uint8_t version[4];
900         uint8_t yy;
901         uint8_t mm;
902         uint8_t dd;
903         int rc;

905         if ((rc = falcon_mdio_read(enp, epp->ep_port, PCS_MMD,
906             PCS_STATUS1_REG, &word)) != 0)
907                 goto fail1;

909         QT2025C_STAT_SET(stat, *maskp, EFX_PHY_STAT_PCS_LINK_UP,
910             (EFX_WORD_FIELD(word, PCS_LINK_UP) != 0) ? 1 : 0);

912         if ((rc = falcon_mdio_read(enp, epp->ep_port, PCS_MMD,
913             PCS_STATUS2_REG, &word)) != 0)
914                 goto fail2;

916         QT2025C_STAT_SET(stat, *maskp, EFX_PHY_STAT_PCS_RX_FAULT,
917             (EFX_WORD_FIELD(word, PCS_RX_FAULT) != 0) ? 1 : 0);
918         QT2025C_STAT_SET(stat, *maskp, EFX_PHY_STAT_PCS_TX_FAULT,
919             (EFX_WORD_FIELD(word, PCS_TX_FAULT) != 0) ? 1 : 0);
```

```
921         if ((rc = falcon_mdio_read(enp, epp->ep_port, PCS_MMD,
922             PCS_10GBASE_R_STATUS2_REG, &word)) != 0)
923                 goto fail3;
924
925         QT2025C_STAT_SET(stat, *maskp, EFX_PHY_STAT_PCS_BER,
926             EFX_WORD_FIELD(word, PCS_BER));
927         QT2025C_STAT_SET(stat, *maskp, EFX_PHY_STAT_PCS_BLOCK_ERRORS,
928             EFX_WORD_FIELD(word, PCS_ERR));
929
930         if ((rc = falcon_mdio_read(enp, epp->ep_port, PCS_MMD,
931             OP_MODE_REG, &word)) != 0)
932                 goto fail4;
933
934         QT2025C_STAT_SET(stat, *maskp, EFX_PHY_STAT_PCS_OP_MODE,
935             EFX_WORD_FIELD(word, OP_MODE_CURRENT));
936
937         if ((rc = qt2025c_version_get(enp, version)) != 0)
938                 goto fail5;
939
940         QT2025C_STAT_SET(stat, *maskp, EFX_PHY_STAT_PCS_FW_VERSION_0,
941             version[0]);
942         QT2025C_STAT_SET(stat, *maskp, EFX_PHY_STAT_PCS_FW_VERSION_1,
943             version[1]);
944         QT2025C_STAT_SET(stat, *maskp, EFX_PHY_STAT_PCS_FW_VERSION_2,
945             version[2]);
946         QT2025C_STAT_SET(stat, *maskp, EFX_PHY_STAT_PCS_FW_VERSION_3,
947             version[3]);
948
949         if ((rc = qt2025c_build_get(enp, &yy, &mm, &dd)) != 0)
950                 goto fail6;
951
952         QT2025C_STAT_SET(stat, *maskp, EFX_PHY_STAT_PCS_FW_BUILD_YY, yy);
953         QT2025C_STAT_SET(stat, *maskp, EFX_PHY_STAT_PCS_FW_BUILD_MM, mm);
954         QT2025C_STAT_SET(stat, *maskp, EFX_PHY_STAT_PCS_FW_BUILD_DD, dd);
955
956         return (0);
957
958 fail6:
959         EFSYS_PROBE(fail6);
960 fail5:
961         EFSYS_PROBE(fail5);
962 fail4:
963         EFSYS_PROBE(fail4);
964 fail3:
965         EFSYS_PROBE(fail3);
966 fail2:
967         EFSYS_PROBE(fail2);
968 fail1:
969         EFSYS_PROBE1(fail1, int, rc);
970
971         return (rc);
972 }
973
974 static  __checkReturn                   int
975 qt2025c_phy_xs_stats_update(
976         __in                            efx_nic_t *enp,
977         __inout                         uint64_t *maskp,
978         __inout_ecount(EFX_PHY_NSTATS)  uint32_t *stat)
979 {
980         efx_port_t *epp = &(enp->en_port);
981         efx_word_t word;
982         int rc;
983
984         if ((rc = falcon_mdio_read(enp, epp->ep_port, PHY_XS_MMD,
985             PHY_XS_STATUS1_REG, &word)) != 0)
```



```
986                 goto fail1;
987
988         QT2025C_STAT_SET(stat, *maskp, EFX_PHY_STAT_PHY_XS_LINK_UP,
989             (EFX_WORD_FIELD(word, PHY_XS_LINK_UP) != 0) ? 1 : 0);
990
991         if ((rc = falcon_mdio_read(enp, epp->ep_port, PHY_XS_MMD,
992             PHY_XS_STATUS2_REG, &word)) != 0)
993                 goto fail2;
994
995         QT2025C_STAT_SET(stat, *maskp, EFX_PHY_STAT_PHY_XS_RX_FAULT,
996             (EFX_WORD_FIELD(word, PHY_XS_RX_FAULT) != 0) ? 1 : 0);
997         QT2025C_STAT_SET(stat, *maskp, EFX_PHY_STAT_PHY_XS_TX_FAULT,
998             (EFX_WORD_FIELD(word, PHY_XS_TX_FAULT) != 0) ? 1 : 0);
999
1000        if ((rc = falcon_mdio_read(enp, epp->ep_port, PHY_XS_MMD,
1001            PHY_XS_LANE_STATUS_REG, &word)) != 0)
1002                goto fail3;
1003
1004        QT2025C_STAT_SET(stat, *maskp, EFX_PHY_STAT_PHY_XS_ALIGN,
1005            (EFX_WORD_FIELD(word, PHY_XS_ALIGNED) != 0) ? 1 : 0);
1006        QT2025C_STAT_SET(stat, *maskp, EFX_PHY_STAT_PHY_XS_SYNC_A,
1007            (EFX_WORD_FIELD(word, PHY_XS_LANE0_SYNC) != 0) ? 1 : 0);
1008        QT2025C_STAT_SET(stat, *maskp, EFX_PHY_STAT_PHY_XS_SYNC_B,
1009            (EFX_WORD_FIELD(word, PHY_XS_LANE1_SYNC) != 0) ? 1 : 0);
1010        QT2025C_STAT_SET(stat, *maskp, EFX_PHY_STAT_PHY_XS_SYNC_C,
1011            (EFX_WORD_FIELD(word, PHY_XS_LANE2_SYNC) != 0) ? 1 : 0);
1012        QT2025C_STAT_SET(stat, *maskp, EFX_PHY_STAT_PHY_XS_SYNC_D,
1013            (EFX_WORD_FIELD(word, PHY_XS_LANE3_SYNC) != 0) ? 1 : 0);
1014
1015        return (0);
1016
1017 fail3:
1018        EFSYS_PROBE(fail3);
1019 fail2:
1020        EFSYS_PROBE(fail2);
1021 fail1:
1022        EFSYS_PROBE1(fail1, int, rc);
1023
1024        return (rc);
1025 }
1026
1027        __checkReturn                   int
1028 qt2025c_stats_update(
1029        __in                            efx_nic_t *enp,
1030        __in                            efsys_mem_t *esmp,
1031        __out_ecount(EFX_PHY_NSTATS)    uint32_t *stat)
1032 {
1033        efx_port_t *epp = &(enp->en_port);
1034        efx_nic_cfg_t *encp = &(enp->en_nic_cfg);
1035        uint64_t mask = 0;
1036        uint32_t oui;
1037        int rc;
1038
1039        _NOTE(ARGUNUSED(esmp))
1040
1041        if ((rc = xphy_mmd_oui_get(enp, epp->ep_port, PMA_PMD_MMD, &oui)) != 0)
1042                goto fail1;
1043
1044        QT2025C_STAT_SET(stat, mask, EFX_PHY_STAT_OUI, oui);
1045
1046        if ((rc = qt2025c_pma_pmd_stats_update(enp, &mask, stat)) != 0)
1047                goto fail2;
1048
1049        if ((rc = qt2025c_pcs_stats_update(enp, &mask, stat)) != 0)
1050                goto fail3;
```

```
1052         if ((rc = qt2025c_phy_xs_stats_update(enp, &mask, stat)) != 0)
1053                 goto fail4;

1055         /* Ensure all the supported statistics are up to date */
1056         EFSYS_ASSERT(mask == encp->enc_phy_stat_mask);

1058         return (0);

1060 fail4:
1061         EFSYS_PROBE(fail4);
1062 fail3:
1063         EFSYS_PROBE(fail3);
1064 fail2:
1065         EFSYS_PROBE(fail2);
1066 fail1:
1067         EFSYS_PROBE1(fail1, int, rc);

1069         return (rc);
1070 }
1071 #endif  /* EFSYS_OPT_PHY_STATS */

1073 #if EFSYS_OPT_PHY_PROPS

1075 #if EFSYS_OPT_NAMES
1076                 const char __cs *
1077 qt2025c_prop_name(
1078         __in    efx_nic_t *enp,
1079         __in    unsigned int id)
1080 {
1081         _NOTE(ARGUNUSED(enp, id))

1083         EFSYS_ASSERT(B_FALSE);

1085         return (NULL);
1086 }
1087 #endif  /* EFSYS_OPT_NAMES */

1089         __checkReturn   int
1090 qt2025c_prop_get(
1091         __in            efx_nic_t *enp,
1092         __in            unsigned int id,
1093         __in            uint32_t flags,
1094         __out           uint32_t *valp)
1095 {
1096         _NOTE(ARGUNUSED(enp, id, flags, valp))

1098         EFSYS_ASSERT(B_FALSE);

1100         return (ENOTSUP);
1101 }

1103         __checkReturn   int
1104 qt2025c_prop_set(
1105         __in            efx_nic_t *enp,
1106         __in            unsigned int id,
1107         __in            uint32_t val)
1108 {
1109         _NOTE(ARGUNUSED(enp, id, val))

1111         EFSYS_ASSERT(B_FALSE);

1113         return (ENOTSUP);
1114 }
1115 #endif  /* EFSYS_OPT_PHY_PROPS */

1117 #endif  /* EFSYS_OPT_PHY_QT2025C */
```



```
1118 #endif /* ! codereview */
```

```
**********************************************************
    4554 Thu Aug 22 18:59:26 2013
new/usr/src/uts/common/io/sfxge/qt2025c.h
Merged sfxge driver
**********************************************************
```

```
/*-
 * Copyright 2008-2013 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#ifndef _SYS_QT2025C_H
#define _SYS_QT2025C_H

#include "efx.h"

#ifdef  __cplusplus
extern "C" {
#endif

#if EFSYS_OPT_PHY_QT2025C

#define QT2025C_LOOPBACK_MASK                                           \
            ((1 << EFX_LOOPBACK_PHY_XS) |                               \
            (1 << EFX_LOOPBACK_PCS) |                                   \
            (1 << EFX_LOOPBACK_PMA_PMD) |                               \
            FALCON_XMAC_LOOPBACK_MASK)

#define QT2025C_LED_MASK                                                \
            ((1 << EFX_PHY_LED_OFF) |                                   \
            (1 << EFX_PHY_LED_ON))

#define QT2025C_NPROPS          0

#define QT2025C_ADV_CAP_MASK                                            \
            ((1 << EFX_PHY_CAP_10000FDX) |                              \
            (1 << EFX_PHY_CAP_PAUSE))

#define QT2025C_ADV_CAP_PERM    0

#define QT2025C_BIST_MASK       0

extern  __checkReturn   int
qt2025c_reset(
        __in            efx_nic_t *enp);

extern  __checkReturn   int
```



```
qt2025c_reconfigure(
        __in            efx_nic_t *enp);

extern  __checkReturn   int
qt2025c_verify(
        __in            efx_nic_t *enp);

extern  __checkReturn   int
qt2025c_uplink_check(
        __in            efx_nic_t *enp,
        __out           boolean_t *upp);

extern  __checkReturn   int
qt2025c_downlink_check(
        __in            efx_nic_t *enp,
        __out           efx_link_mode_t *modep,
        __out           unsigned int *fcntlp,
        __out           uint32_t *lp_cap_maskp);

extern                  void
qt2025c_downlink_reset(
        __in            efx_nic_t *enp);

extern  __checkReturn   int
qt2025c_oui_get(
        __in            efx_nic_t *enp,
        __out           uint32_t *ouip);

#if EFSYS_OPT_PHY_STATS

/* START MKCONFIG GENERATED Qt2025cPhyHeaderStatsMask 2eb4a092d85bd5ac */
#define QT2025C_STAT_MASK \
        (1ULL << EFX_PHY_STAT_OUI) | \
        (1ULL << EFX_PHY_STAT_PMA_PMD_LINK_UP) | \
        (1ULL << EFX_PHY_STAT_PMA_PMD_RX_FAULT) | \
        (1ULL << EFX_PHY_STAT_PMA_PMD_TX_FAULT) | \
        (1ULL << EFX_PHY_STAT_PCS_LINK_UP) | \
        (1ULL << EFX_PHY_STAT_PCS_RX_FAULT) | \
        (1ULL << EFX_PHY_STAT_PCS_TX_FAULT) | \
        (1ULL << EFX_PHY_STAT_PCS_BER) | \
        (1ULL << EFX_PHY_STAT_PCS_BLOCK_ERRORS) | \
        (1ULL << EFX_PHY_STAT_PCS_FW_VERSION_0) | \
        (1ULL << EFX_PHY_STAT_PCS_FW_VERSION_1) | \
        (1ULL << EFX_PHY_STAT_PCS_FW_VERSION_2) | \
        (1ULL << EFX_PHY_STAT_PCS_FW_VERSION_3) | \
        (1ULL << EFX_PHY_STAT_PCS_FW_BUILD_YY) | \
        (1ULL << EFX_PHY_STAT_PCS_FW_BUILD_MM) | \
        (1ULL << EFX_PHY_STAT_PCS_FW_BUILD_DD) | \
        (1ULL << EFX_PHY_STAT_PCS_OP_MODE) | \
        (1ULL << EFX_PHY_STAT_PHY_XS_LINK_UP) | \
        (1ULL << EFX_PHY_STAT_PHY_XS_RX_FAULT) | \
        (1ULL << EFX_PHY_STAT_PHY_XS_TX_FAULT) | \
        (1ULL << EFX_PHY_STAT_PHY_XS_ALIGN) | \
        (1ULL << EFX_PHY_STAT_PHY_XS_SYNC_A) | \
        (1ULL << EFX_PHY_STAT_PHY_XS_SYNC_B) | \
        (1ULL << EFX_PHY_STAT_PHY_XS_SYNC_C) | \
        (1ULL << EFX_PHY_STAT_PHY_XS_SYNC_D)
/* END MKCONFIG GENERATED Qt2025cPhyHeaderStatsMask */

extern  __checkReturn                   int
qt2025c_stats_update(
        __in                            efx_nic_t *enp,
        __in                            efsys_mem_t *esmp,
        __out_ecount(EFX_PHY_NSTATS)    uint32_t *stat);
```

```
128 #endif  /* EFSYS_OPT_PHY_STATS */

130 #if EFSYS_OPT_PHY_PROPS

132 #if EFSYS_OPT_NAMES

134 extern          const char __cs *
135 qt2025c_prop_name(
136         __in    efx_nic_t *enp,
137         __in    unsigned int id);

139 #endif

141 extern  __checkReturn   int
142 qt2025c_prop_get(
143         __in            efx_nic_t *enp,
144         __in            unsigned int id,
145         __in            uint32_t flags,
146         __out           uint32_t *valp);

148 extern  __checkReturn   int
149 qt2025c_prop_set(
150         __in            efx_nic_t *enp,
151         __in            unsigned int id,
152         __in            uint32_t val);

154 #endif  /* EFSYS_OPT_PHY_PROPS */

156 #endif  /* EFSYS_OPT_PHY_QT2025C */

158 #ifdef  __cplusplus
159 }
160 #endif

162 #endif  /* _SYS_QT2025C_H */
163 #endif /* ! codereview */
```

```
**********************************************************
    6905 Thu Aug 22 18:59:26 2013
new/usr/src/uts/common/io/sfxge/qt2025c_impl.h
Merged sfxge driver
**********************************************************
```

```
/*-
 * Copyright 2008-2013 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#ifndef _SYS_QT2025C_IMPL_H
#define _SYS_QT2025C_IMPL_H

#ifdef  __cplusplus
extern "C" {
#endif

#if EFSYS_OPT_PHY_QT2025C

#define QT2025C_MMD_MASK                                        \
            ((1 << PMA_PMD_MMD) |                               \
            (1 << PCS_MMD) |                                    \
            (1 << PHY_XS_MMD))

/* PMA/PMD */

#define MDIO_RESETS_REG 0xc300
#define KRDLL_RESETN_LBN 0
#define KRDLL_RESETN_WIDTH 1
#define MICRO_RESETN_LBN 1
#define MICRO_RESETN_WIDTH 1
#define EDC_RESETN_LBN 2
#define EDC_RESETN_WIDTH 1
#define PSDLL_RESETN_LBN 3
#define PSDLL_RESETN_WIDTH 1
#define PRDLL_RESETN_LBN 4
#define PRDLL_RESETN_WIDTH 1
#define RESLRN_RESETN_LBN 5
#define RESLRN_RESETN_WIDTH 1
#define IB_TX_RESETN_LBN 6
#define IB_TX_RESETN_WIDTH 1
#define IB_RX_RESETN_LBN 7
#define IB_RX_RESETN_WIDTH 1
#define SINUS_RESETN_LBN 8
#define SINUS_RESETN_WIDTH 1
```



```
#define PMA_UC8051_CFG_REG 0xc302
#define EREFCLK_FREQ_LBN 0
#define EREFCLK_FREQ_WIDTH 1
#define EREFCLK_FREQ_156_25_DECODE 0
#define EREFCLK_FREQ_52_08_DECODE 1
#define SREFCLK_FREQ_LBN 1
#define SREFCLK_FREQ_WIDTH 1
#define SREFCLK_FREQ_155_52_DECODE 0
#define SREFCLK_FREQ_51_84_DECODE 1
#define UC_CLK_SEL_LBN 2
#define UC_CLK_SEL_WIDTH 3
#define UC_CLK_156_25_DECODE 1
#define EEPROM_BOOT_MODE_LBN 5
#define EEPROM_BOOT_MODE_WIDTH 1
#define EEPROM_BOOT_0_DECODE 0
#define EEPROM_BOOT_1K_DECODE 1
#define UC_JTAG_CTRL_LBN 6
#define UC_JTAG_CTRL_WIDTH 1

#define PMA_FTX_CTRL2_REG 0xc309
#define FTX_STATIC_LBN 13
#define FTX_STATIC_WIDTH 1

#define PMA_UC8051_I2C_FREQ_REG 0xc316
#define UC_I2C_DIV_LBN 0
#define UC_I2C_DIV_WIDTH 8

#define PMA_UC8051_I2C_SLV_ADDR_REG 0xc318
#define UC_I2C_SLV_ADDR_LBN 0
#define UC_I2C_SLV_ADDR_WIDTH 7

#define PMA_UC8051_SPARE2_REG 0xc319
#define DISABLE_PORT_LBN 1
#define DISABLE_PORT_WIDTH 1
#define OP_MODE_CONFIG_LBN 3
#define OP_MODE_CONFIG_WIDTH 3
#define OP_MODE_LINEAR_DECODE 0
#define OP_MODE_LIMITING_DECODE 1
#define OP_MODE_KR_DECODE 2
#define OP_MODE_DIRECT_DECODE 4
#define OP_MODE_LOW_POWER_DECODE 5
#define OP_MODE_AUTO_DECODE 7
#define DATA_RATE_LBN 7
#define DATA_RATE_WIDTH 1
#define DATA_RATE_10G_DECODE 0
#define DATA_RATE_1_25G_DECODE 1

#define PMA_UC8051_SPARE3_REG 0xc31a
#define XDRV_OVRD_LBN 1
#define XDRV_OVRD_WIDTH 1
#define FTX_OVRD_LBN 2
#define FTX_OVRD_WIDTH 1
#define MODULE_TYPE_LBN 3
#define MODULE_TYPE_WIDTH 3
#define MODULE_TYPE_X2_DECODE 1
#define MODULE_TYPE_XFP_DECODE 2
#define MODULE_TYPE_SFP_DECODE 3

#define PRD_CODE1_REG 0xd000
#define PRODUCT_CODE_LBN 0
#define PRODUCT_CODE_WIDTH 16

#define PMA_PMD_LED1_REG 0xd006 /* Green */
#define PMA_PMD_LED2_REG 0xd007 /* Amber */
#define PMA_PMD_LED3_REG 0xd008 /* Red */
```

```
#define PMA_PMD_LED_CFG_LBN 0
#define PMA_PMD_LED_CFG_WIDTH 3
#define LED_CFG_LS_DECODE 0x1
#define LED_CFG_LA_DECODE 0x2
#define LED_CFG_LSA_DECODE 0x3
#define LED_CFG_OFF_DECODE 0x4
#define LED_CFG_ON_DECODE 0x5
#define PMA_PMD_LED_PATH_LBN 3
#define PMA_PMD_LED_PATH_WIDTH 1
#define LED_PATH_TX_DECODE 0x0
#define LED_PATH_RX_DECODE 0x1

/* PCS */

#define KR_BYPASS_CTRL_REG 0x0026
#define FREEZE_DIS_LBN 8
#define FREEZE_DIS_WIDTH 8
#define FREEZE_DIS_DECODE 0x0e

#define BOOT_CTRL_REG 0x0027
#define LED1_CTRL_LBN 0
#define LED1_CTRL_WIDTH 1
#define LED1_CTRL_FW_DECODE 0
#define LED1_CTRL_HW_DECODE 1
#define FTX_FR4_LOSS_LBN 1
#define FTX_FR4_LOSS_WIDTH 3
#define FRX_FR4_LOSS_LBN 4
#define FRX_FR4_LOSS_WIDTH 3
#define GE_PRBS_MODE_LBN 15
#define GE_PRBS_MODE_WIDTH 1

#define FW_CFG_REG 0x0028
#define FW_CFG_KEY_LBN 0
#define FW_CFG_KEY_WIDTH 16
#define FW_CFG_KEY_DECODE 0xa528

#define MICRO_RAM1_BASE 0x8000
#define MICRO_RAM1_SIZE 0x4000

#define OP_MODE_REG 0xd70c
#define OP_MODE_CURRENT_LBN 0
#define OP_MODE_CURRENT_WIDTH 4

#define LED_CFG_REG 0xd70d
#define LED3_CFG_LBN 0
#define LED3_CFG_WIDTH 2
#define LED2_CFG_LBN 2
#define LED2_CFG_WIDTH 2
#define LED1_CFG_LBN 4
#define LED1_CFG_WIDTH 2
#define LED_RXTX_DECODE 3
#define LED_RX_DECODE 2
#define LED_TX_DECODE 1
#define LED_ON_DECODE 0

#define CKSUM_STATUS1_REG 0xd716
#define CKSUM_STATUS1_BYTE_LBN 0
#define CKSUM_STATUS1_BYTE_WIDTH 1
#define CKSUM_STATUS1_BYTE_BAD_DECODE 0xde

#define CKSUM_STATUS2_REG 0xd717
#define CKSUM_STATUS2_BYTE_LBN 0
#define CKSUM_STATUS2_BYTE_WIDTH 1
#define CKSUM_STATUS2_BYTE_BAD_DECODE 0xad

#define FW_HEARTBEAT_REG 0xd7ee
```



```
#define FW_HEARTB_LBN 0
#define FW_HEARTB_WIDTH 8

#define FW_VERSION1_REG 0xd7f3
#define FW_VERSION2_REG 0xd7f4
#define FW_VERSION3_REG 0xd7f5

#define FW_BUILD1_REG 0xd7f6
#define FW_BUILD2_REG 0xd7f7
#define FW_BUILD3_REG 0xd7f8

#define UC8051_STATUS_REG 0xd7fd
#define UC_STATUS_LBN 0
#define UC_STATUS_WIDTH 8
#define UC_STATUS_INVALID_DECODE 0x00
#define UC_STATUS_FW_START_DECODE 0x10
#define UC_STATUS_FW_SAVE_DECODE 0x20
#define UC_STATUS_INIT_DECODE 0x40
#define UC_STATUS_AQ_IN_PROG_DECODE 0x50
#define UC_STATUS_AQ_COMPLETE_DECODE 0x60
#define UC_STATUS_TRACK_IN_PROG_DECODE 0x70
#define UC_STATUS_IN_AN_DECODE 0xa0

#define MICRO_GEN_CTL_REG 0xe854
#define UC_INT0_CNT_LBN 0
#define UC_INT0_CNT_WIDTH 1
#define UC_INT1_CNT_LBN 1
#define UC_INT1_CNT_WIDTH 1
#define UC_MDIO_SW_LBN 3
#define UC_MDIO_SW_WIDTH 1
#define UC_DIS_ROM_LBN 4
#define UC_DIS_ROM_WIDTH 1
#define UC_UPDATE_LBN 5
#define UC_UPDATE_WIDTH 1
#define UC_RUN_RAM_LBN 6
#define UC_RUN_RAM_WIDTH 1
#define UC_RST_LBN 7
#define UC_RST_WIDTH 1

/* PHY_XS */

#define XGXS_VENDOR_SPECIFIC_1_REG 0xc000
#define XGXS_SYSLPBK_LBN 14
#define XGXS_SYSLPBK_WIDTH 1

#define MICRO_RAM2_BASE 0x8000
#define MICRO_RAM2_SIZE 0x2000

#endif  /* EFSYS_OPT_PHY_QT2025C */

#ifdef  __cplusplus
}
#endif

#endif  /* _SYS_QT2025C_IMPL_H */
#endif /* ! codereview */
```

```
**********************************************************
   47639 Thu Aug 22 18:59:26 2013
new/usr/src/uts/common/io/sfxge/sft9001.c
Merged sfxge driver
**********************************************************
```

```
/*
 * Copyright 2008-2013 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#include "efsys.h"
#include "efx.h"
#include "efx_types.h"
#include "efx_regs.h"
#include "efx_impl.h"
#include "falcon_nvram.h"
#include "sft9001.h"
#include "sft9001_impl.h"
#include "xphy.h"
#include "falcon_impl.h"

#if EFSYS_OPT_PHY_SFT9001

static  __checkReturn   int
sft9001_short_reach_set(
        __in            efx_nic_t *enp,
        __in            boolean_t on)
{
        efx_port_t *epp = &(enp->en_port);
        efx_word_t word;
        int rc;

        if ((rc = falcon_mdio_read(enp, epp->ep_port,
            PMA_PMD_MMD, PMA_PMD_PWR_BACKOFF_REG, &word)) != 0)
                goto fail1;

        EFX_SET_WORD_FIELD(word, SHORT_REACH, (on) ? 1 : 0);

        if ((rc = falcon_mdio_write(enp, epp->ep_port,
            PMA_PMD_MMD, PMA_PMD_PWR_BACKOFF_REG, &word)) != 0)
                goto fail2;

        return (0);

fail2:
        EFSYS_PROBE(fail2);
```



```
fail1:
        EFSYS_PROBE1(fail1, int, rc);

        return (rc);
}

static  __checkReturn   int
sft9001_short_reach_get(
        __in            efx_nic_t *enp,
        __out           boolean_t *onp)
{
        efx_port_t *epp = &(enp->en_port);
        efx_word_t word;
        int rc;

        if ((rc = falcon_mdio_read(enp, epp->ep_port,
            PMA_PMD_MMD, PMA_PMD_PWR_BACKOFF_REG, &word)) != 0)
                goto fail1;

        *onp = (EFX_WORD_FIELD(word, SHORT_REACH) != 0);

        return (0);

fail1:
        EFSYS_PROBE1(fail1, int, rc);

        return (rc);
}

static  __checkReturn   int
sft9001_robust_set(
        __in            efx_nic_t *enp,
        __in            boolean_t on)
{
        efx_port_t *epp = &(enp->en_port);
        efx_word_t word;
        int rc;

        if ((rc = falcon_mdio_read(enp, epp->ep_port, PMA_PMD_MMD,
            PMA_PMD_XCONTROL_REG, &word)) != 0)
                goto fail1;

        EFX_SET_WORD_FIELD(word, ROBUST, (on) ? 1 : 0);

        if ((rc = falcon_mdio_write(enp, epp->ep_port, PMA_PMD_MMD,
            PMA_PMD_XCONTROL_REG, &word)) != 0)
                goto fail2;

        return (0);

fail2:
        EFSYS_PROBE(fail2);
fail1:
        EFSYS_PROBE1(fail1, int, rc);

        return (rc);
}

static  __checkReturn   int
sft9001_robust_get(
        __in            efx_nic_t *enp,
        __out           boolean_t *onp)
{
        efx_port_t *epp = &(enp->en_port);
        efx_word_t word;
        int rc;
```

```
129         if ((rc = falcon_mdio_read(enp, epp->ep_port, PMA_PMD_MMD,
130             PMA_PMD_XCONTROL_REG, &word)) != 0)
131                 goto fail1;
132
133         *onp = (EFX_WORD_FIELD(word, ROBUST) != 0);
134
135         return (0);
136
137 fail1:
138         EFSYS_PROBE1(fail1, int, rc);
139
140         return (rc);
141 }
142
143 static  __checkReturn   int
144 sft9001_an_set(
145         __in            efx_nic_t *enp,
146         __in            boolean_t on)
147 {
148         efx_port_t *epp = &(enp->en_port);
149         efx_word_t word;
150         int rc;
151
152         if ((rc = falcon_mdio_read(enp, epp->ep_port, AN_MMD,
153             AN_CONTROL1_REG, &word)) != 0)
154                 goto fail1;
155
156         if (on) {
157                 EFX_SET_WORD_FIELD(word, AN_ENABLE, 1);
158                 EFX_SET_WORD_FIELD(word, AN_RESTART, 1);
159         } else {
160                 EFX_SET_WORD_FIELD(word, AN_ENABLE, 0);
161         }
162
163         if ((rc = falcon_mdio_write(enp, epp->ep_port, AN_MMD,
164             AN_CONTROL1_REG, &word)) != 0)
165                 goto fail2;
166
167         return (0);
168
169 fail2:
170         EFSYS_PROBE(fail2);
171 fail1:
172         EFSYS_PROBE1(fail1, int, rc);
173
174         return (rc);
175 }
176
177 static  __checkReturn   int
178 sft9001_gmii_cfg(
179         __in            efx_nic_t *enp)
180 {
181         efx_port_t *epp = &(enp->en_port);
182         efx_word_t word;
183         int rc;
184
185         if ((rc = falcon_mdio_read(enp, epp->ep_port, PMA_PMD_MMD,
186             PMA_PMD_XCONTROL_REG, &word)) != 0)
187                 goto fail1;
188
189         EFX_SET_WORD_FIELD(word, GMII_EN, 1);
190
191         if ((rc = falcon_mdio_write(enp, epp->ep_port, PMA_PMD_MMD,
192             PMA_PMD_XCONTROL_REG, &word)) != 0)
193                 goto fail2;
```



```
194
195         return (0);
196
197 fail2:
198         EFSYS_PROBE(fail2);
199 fail1:
200         EFSYS_PROBE1(fail1, int, rc);
201
202         return (rc);
203 }
204
205 static  __checkReturn   int
206 sft9001_clock_cfg(
207         __in            efx_nic_t *enp)
208 {
209         efx_port_t *epp = &(enp->en_port);
210         efx_word_t word;
211         int rc;
212
213         if ((rc = falcon_mdio_read(enp, epp->ep_port, PMA_PMD_MMD,
214             PMA_PMD_XCONTROL_REG, &word)) != 0)
215                 goto fail1;
216
217         /* Select 312MHz clock on CLK312_OUT_{P,N} */
218         EFX_SET_WORD_FIELD(word, CLK312_OUT_SEL, SEL_312MHZ_DECODE);
219         EFX_SET_WORD_FIELD(word, CLK312_OUT_EN, 1);
220
221         /* Select 125MHz clock on TEST_CLKOUT_{P,N} */
222         EFX_SET_WORD_FIELD(word, TEST_CLKOUT_SEL, SEL_125MHZ_DECODE);
223         EFX_SET_WORD_FIELD(word, TEST_CLKOUT_EN, 1);
224
225         if ((rc = falcon_mdio_write(enp, epp->ep_port, PMA_PMD_MMD,
226             PMA_PMD_XCONTROL_REG, &word)) != 0)
227                 goto fail2;
228
229         return (0);
230
231 fail2:
232         EFSYS_PROBE(fail2);
233 fail1:
234         EFSYS_PROBE1(fail1, int, rc);
235
236         return (rc);
237 }
238
239 static  __checkReturn   int
240 sft9001_adv_cap_cfg(
241         __in            efx_nic_t *enp)
242 {
243         efx_port_t *epp = &(enp->en_port);
244         efx_word_t word;
245         int rc;
246
247         /* Check base page */
248         if ((rc = falcon_mdio_read(enp, epp->ep_port, AN_MMD,
249             AN_ADV_BP_CAP_REG, &word)) != 0)
250                 goto fail1;
251
252         EFSYS_ASSERT(!(epp->ep_adv_cap_mask & (1 << EFX_PHY_CAP_10HDX)));
253         if (EFX_WORD_FIELD(word, AN_ADV_TA_10BASE_T) != 0)
254                 EFX_SET_WORD_FIELD(word, AN_ADV_TA_10BASE_T, 0);
255
256         EFSYS_ASSERT(!(epp->ep_adv_cap_mask & (1 << EFX_PHY_CAP_10FDX)));
257         if (EFX_WORD_FIELD(word, AN_ADV_TA_10BASE_T_FDX) != 0)
258                 EFX_SET_WORD_FIELD(word, AN_ADV_TA_10BASE_T_FDX, 0);
```

```
260         if (EFX_WORD_FIELD(word, AN_ADV_TA_100BASE_TX) == 0 &&
261             epp->ep_adv_cap_mask & (1 << EFX_PHY_CAP_100HDX))
262                 EFX_SET_WORD_FIELD(word, AN_ADV_TA_100BASE_TX, 1);
263         else if (EFX_WORD_FIELD(word, AN_ADV_TA_100BASE_TX) != 0 &&
264             !(epp->ep_adv_cap_mask & (1 << EFX_PHY_CAP_100HDX)))
265                 EFX_SET_WORD_FIELD(word, AN_ADV_TA_100BASE_TX, 0);

267         if (EFX_WORD_FIELD(word, AN_ADV_TA_100BASE_TX_FDX) == 0 &&
268             epp->ep_adv_cap_mask & (1 << EFX_PHY_CAP_100FDX))
269                 EFX_SET_WORD_FIELD(word, AN_ADV_TA_100BASE_TX_FDX, 1);
270         else if (EFX_WORD_FIELD(word, AN_ADV_TA_100BASE_TX_FDX) != 0 &&
271             !(epp->ep_adv_cap_mask & (1 << EFX_PHY_CAP_100FDX)))
272                 EFX_SET_WORD_FIELD(word, AN_ADV_TA_100BASE_TX_FDX, 0);

274         if (EFX_WORD_FIELD(word, AN_ADV_TA_PAUSE) == 0 &&
275             epp->ep_adv_cap_mask & (1 << EFX_PHY_CAP_PAUSE))
276                 EFX_SET_WORD_FIELD(word, AN_ADV_TA_PAUSE, 1);
277         else if (EFX_WORD_FIELD(word, AN_ADV_TA_PAUSE) != 0 &&
278             !(epp->ep_adv_cap_mask & (1 << EFX_PHY_CAP_PAUSE)))
279                 EFX_SET_WORD_FIELD(word, AN_ADV_TA_PAUSE, 0);

281         if (EFX_WORD_FIELD(word, AN_ADV_TA_ASM_DIR) == 0 &&
282             epp->ep_adv_cap_mask & (1 << EFX_PHY_CAP_ASYM))
283                 EFX_SET_WORD_FIELD(word, AN_ADV_TA_ASM_DIR, 1);
284         else if (EFX_WORD_FIELD(word, AN_ADV_TA_ASM_DIR) != 0 &&
285             !(epp->ep_adv_cap_mask & (1 << EFX_PHY_CAP_ASYM)))
286                 EFX_SET_WORD_FIELD(word, AN_ADV_TA_ASM_DIR, 0);

288         if ((rc = falcon_mdio_write(enp, epp->ep_port, AN_MMD,
289             AN_ADV_BP_CAP_REG, &word)) != 0)
290                 goto fail2;

292         /* Check 1G operation */
293         if ((rc = falcon_mdio_read(enp, epp->ep_port, CL22EXT_MMD,
294             CL22EXT_MS_CONTROL_REG, &word)) != 0)
295                 goto fail3;

297         EFSYS_ASSERT(!(epp->ep_adv_cap_mask & (1 << EFX_PHY_CAP_1000HDX)));
298         if (EFX_WORD_FIELD(word, CL22EXT_1000BASE_T_ADV) != 0)
299                 EFX_SET_WORD_FIELD(word, CL22EXT_1000BASE_T_ADV, 0);

301         if (EFX_WORD_FIELD(word, CL22EXT_1000BASE_T_FDX_ADV) == 0 &&
302             epp->ep_adv_cap_mask & (1 << EFX_PHY_CAP_1000FDX))
303                 EFX_SET_WORD_FIELD(word, CL22EXT_1000BASE_T_FDX_ADV, 1);
304         else if (EFX_WORD_FIELD(word, CL22EXT_1000BASE_T_FDX_ADV) != 0 &&
305             !(epp->ep_adv_cap_mask & (1 << EFX_PHY_CAP_1000FDX)))
306                 EFX_SET_WORD_FIELD(word, CL22EXT_1000BASE_T_FDX_ADV, 0);

308         if ((rc = falcon_mdio_write(enp, epp->ep_port, CL22EXT_MMD,
309             CL22EXT_MS_CONTROL_REG, &word)) != 0)
310                 goto fail4;

312         /* Check 10G operation */
313         if ((rc = falcon_mdio_read(enp, epp->ep_port, AN_MMD,
314             AN_10G_BASE_T_CONTROL_REG, &word)) != 0)
315                 goto fail5;

317         if (EFX_WORD_FIELD(word, AN_10G_BASE_T_ADV) == 0 &&
318             epp->ep_adv_cap_mask & (1 << EFX_PHY_CAP_10000FDX))
319                 EFX_SET_WORD_FIELD(word, AN_10G_BASE_T_ADV, 1);
320         else if (EFX_WORD_FIELD(word, AN_10G_BASE_T_ADV) != 0 &&
321             !(epp->ep_adv_cap_mask & (1 << EFX_PHY_CAP_10000FDX)))
322                 EFX_SET_WORD_FIELD(word, AN_10G_BASE_T_ADV, 0);

324         if ((rc = falcon_mdio_write(enp, epp->ep_port, AN_MMD,
325             AN_10G_BASE_T_CONTROL_REG, &word)) != 0)
```



```
326                 goto fail6;

328         return (0);

330 fail6:
331         EFSYS_PROBE(fail6);
332 fail5:
333         EFSYS_PROBE(fail5);
334 fail4:
335         EFSYS_PROBE(fail4);
336 fail3:
337         EFSYS_PROBE(fail3);
338 fail2:
339         EFSYS_PROBE(fail2);
340 fail1:
341         EFSYS_PROBE1(fail1, int, rc);

343         return (rc);
344 }

346 #if EFSYS_OPT_LOOPBACK
347 static  __checkReturn   int
348 sft9001_loopback_cfg(
349         __in            efx_nic_t *enp)
350 {
351         efx_port_t *epp = &(enp->en_port);
352         efx_word_t word;
353         int rc;

355         switch (epp->ep_loopback_type) {
356         case EFX_LOOPBACK_PHY_XS:
357                 if ((rc = falcon_mdio_read(enp, epp->ep_port,
358                     PHY_XS_MMD, PHY_XS_TEST1_REG, &word)) != 0)
359                         goto fail1;

361                 EFX_SET_WORD_FIELD(word, PHY_XS_NE_LOOPBACK, 1);

363                 if ((rc = falcon_mdio_write(enp, epp->ep_port,
364                     PHY_XS_MMD, PHY_XS_TEST1_REG, &word)) != 0)
365                         goto fail2;

367                 break;

369         case EFX_LOOPBACK_PCS:
370                 if ((rc = xphy_mmd_loopback_set(enp, epp->ep_port, PCS_MMD,
371                     B_TRUE)) != 0)
372                         goto fail1;

374                 break;

376         case EFX_LOOPBACK_PMA_PMD:
377                 if ((rc = xphy_mmd_loopback_set(enp, epp->ep_port, PMA_PMD_MMD,
378                     B_TRUE)) != 0)
379                         goto fail1;

381                 break;

383         case EFX_LOOPBACK_GPHY:
384                 if ((rc = falcon_mdio_read(enp, epp->ep_port,
385                     CL22EXT_MMD, CL22EXT_CONTROL_REG, &word)) != 0)
386                         goto fail1;

388                 EFX_SET_WORD_FIELD(word, CL22EXT_NE_LOOPBACK, 1);

390                 if ((rc = falcon_mdio_write(enp, epp->ep_port,
391                     CL22EXT_MMD, CL22EXT_CONTROL_REG, &word)) != 0)
```

```
392                         goto fail2;

394                 break;

396         default:
397                 break;
398         }

400         return (0);

402 fail2:
403         EFSYS_PROBE(fail2);
404 fail1:
405         EFSYS_PROBE1(fail1, int, rc);

407         return (rc);
408 }
409 #endif  /* EFSYS_OPT_LOOPBACK */

411 static  __checkReturn   int
412 sft9001_led_cfg(
413         __in            efx_nic_t *enp)
414 {
415         efx_port_t *epp = &(enp->en_port);
416         efx_word_t word;
417         int rc;

419 #if EFSYS_OPT_PHY_LED_CONTROL

421         switch (epp->ep_phy_led_mode) {
422         case EFX_PHY_LED_DEFAULT:
423                 EFX_POPULATE_WORD_8(word,
424                     LED_TMODE, LED_NORMAL_DECODE,
425                     LED_SPARE, LED_NORMAL_DECODE,
426                     LED_MS, LED_NORMAL_DECODE,
427                     LED_RX, LED_NORMAL_DECODE,
428                     LED_TX, LED_NORMAL_DECODE,
429                     LED_SPEED0, LED_NORMAL_DECODE,
430                     LED_SPEED1, LED_NORMAL_DECODE,
431                     LED_LINK, LED_NORMAL_DECODE);
432                 break;

434         case EFX_PHY_LED_OFF:
435                 EFX_POPULATE_WORD_8(word,
436                     LED_TMODE, LED_OFF_DECODE,
437                     LED_SPARE, LED_OFF_DECODE,
438                     LED_MS, LED_OFF_DECODE,
439                     LED_RX, LED_OFF_DECODE,
440                     LED_TX, LED_OFF_DECODE,
441                     LED_SPEED0, LED_OFF_DECODE,
442                     LED_SPEED1, LED_OFF_DECODE,
443                     LED_LINK, LED_OFF_DECODE);
444                 break;

446         case EFX_PHY_LED_ON:
447                 EFX_POPULATE_WORD_8(word,
448                     LED_TMODE, LED_ON_DECODE,
449                     LED_SPARE, LED_ON_DECODE,
450                     LED_MS, LED_ON_DECODE,
451                     LED_RX, LED_ON_DECODE,
452                     LED_TX, LED_ON_DECODE,
453                     LED_SPEED0, LED_ON_DECODE,
454                     LED_SPEED1, LED_ON_DECODE,
455                     LED_LINK, LED_ON_DECODE);
456                 break;
```



```
458         case EFX_PHY_LED_FLASH:
459                 EFX_POPULATE_WORD_8(word,
460                     LED_TMODE, LED_FLASH_DECODE,
461                     LED_SPARE, LED_FLASH_DECODE,
462                     LED_MS, LED_FLASH_DECODE,
463                     LED_RX, LED_FLASH_DECODE,
464                     LED_TX, LED_FLASH_DECODE,
465                     LED_SPEED0, LED_FLASH_DECODE,
466                     LED_SPEED1, LED_FLASH_DECODE,
467                     LED_LINK, LED_FLASH_DECODE);
468                 break;

470         default:
471                 EFSYS_ASSERT(B_FALSE);
472                 break;
473         }

475 #else   /* EFSYS_OPT_PHY_LED_CONTROL */

477         EFX_POPULATE_WORD_8(word,
478             LED_TMODE, LED_NORMAL_DECODE,
479             LED_SPARE, LED_NORMAL_DECODE,
480             LED_MS, LED_NORMAL_DECODE,
481             LED_RX, LED_NORMAL_DECODE,
482             LED_TX, LED_NORMAL_DECODE,
483             LED_SPEED0, LED_NORMAL_DECODE,
484             LED_SPEED1, LED_NORMAL_DECODE,
485             LED_LINK, LED_NORMAL_DECODE);

487 #endif  /* EFSYS_OPT_PHY_LED_CONTROL */

489         if ((rc = falcon_mdio_write(enp, epp->ep_port, PMA_PMD_MMD,
490             PMA_PMD_LED_OVERRIDE_REG, &word)) != 0)
491                 goto fail1;

493         return (0);

495 fail1:
496         EFSYS_PROBE1(fail1, int, rc);

498         return (rc);
499 }

501         __checkReturn   int
502 sft9001_reset(
503         __in            efx_nic_t *enp)
504 {
505         int state;

507         /* Lock I2C bus because sft9001 is sensitive to GPIO3 */
508         EFSYS_LOCK(enp->en_eslp, state);
509         EFSYS_ASSERT(!enp->en_u.falcon.enu_i2c_locked);
510         enp->en_u.falcon.enu_i2c_locked = B_TRUE;
511         EFSYS_UNLOCK(enp->en_eslp, state);

513         /* Pull the external reset line */
514         falcon_nic_phy_reset(enp);

516         /* Unlock I2C */
517         EFSYS_LOCK(enp->en_eslp, state);
518         enp->en_u.falcon.enu_i2c_locked = B_FALSE;
519         EFSYS_UNLOCK(enp->en_eslp, state);

521         return (0);
522 }
```

```
524         __checkReturn   int
525 sft9001_reconfigure(
526         __in            efx_nic_t *enp)
527 {
528         efx_port_t *epp = &(enp->en_port);
529         unsigned int count;
530         int rc;
531
532         /* Wait for the firmware boot to complete */
533         count = 0;
534         do {
535                 efx_word_t word;
536
537                 EFSYS_PROBE1(wait, unsigned int, count);
538
539                 /* Spin for 1 ms */
540                 EFSYS_SPIN(1000);
541
542                 if ((rc = falcon_mdio_read(enp, epp->ep_port, PCS_MMD,
543                     PCS_BOOT_STATUS_REG, &word)) != 0)
544                         goto fail1;
545
546                 if (EFX_WORD_FIELD(word, FATAL_ERR) != 0)
547                         break;  /* no point in continuing */
548
549                 if (EFX_WORD_FIELD(word, BOOT_STATUS) != 0 &&
550                     EFX_WORD_FIELD(word, CODE_DOWNLOAD) != 0 &&
551                     EFX_WORD_FIELD(word, CKSUM_OK) != 0 &&
552                     EFX_WORD_FIELD(word, CODE_STARTED) != 0 &&
553                     EFX_WORD_FIELD(word, BOOT_PROGRESS) == APP_JMP_DECODE)
554                         goto configure;
555
556         } while (++count < 1000);
557
558         rc = ENOTACTIVE;
559         goto fail2;
560
561 configure:
562         if ((rc = xphy_pkg_wait(enp, epp->ep_port, SFT9001_MMD_MASK)) != 0)
563                 goto fail3;
564
565         /* Make sure auto-negotiation is off whilst we configure the PHY */
566         if ((rc = sft9001_an_set(enp, B_FALSE)) != 0)
567                 goto fail4;
568
569         if ((rc = sft9001_gmii_cfg(enp)) != 0)
570                 goto fail5;
571
572         if ((rc = sft9001_clock_cfg(enp)) != 0)
573                 goto fail6;
574
575         if ((rc = sft9001_adv_cap_cfg(enp)) != 0)
576                 goto fail7;
577
578 #if EFSYS_OPT_LOOPBACK
579         if ((rc = sft9001_loopback_cfg(enp)) != 0)
580                 goto fail8;
581 #endif  /* EFSYS_OPT_LOOPBACK */
582
583         if ((rc = sft9001_led_cfg(enp)) != 0)
584                 goto fail9;
585
586         if ((rc = sft9001_robust_set(enp, B_TRUE)) != 0)
587                 goto fail10;
588
589 #if EFSYS_OPT_LOOPBACK
```



```
590         if (epp->ep_loopback_type == EFX_LOOPBACK_OFF) {
591                 if ((rc = sft9001_an_set(enp, B_TRUE)) != 0)
592                         goto fail11;
593         }
594 #else   /* EFSYS_OPT_LOOPBACK */
595         if ((rc = sft9001_an_set(enp, B_TRUE)) != 0)
596                 goto fail11;
597 #endif  /* EFSYS_OPT_LOOPBACK */
598
599         return (0);
600
601 fail11:
602         EFSYS_PROBE(fail11);
603 fail10:
604         EFSYS_PROBE(fail10);
605 fail9:
606         EFSYS_PROBE(fail9);
607
608 #if EFSYS_OPT_LOOPBACK
609 fail8:
610         EFSYS_PROBE(fail8);
611 #endif  /* EFSYS_OPT_LOOPBACK */
612
613 fail7:
614         EFSYS_PROBE(fail7);
615 fail6:
616         EFSYS_PROBE(fail6);
617 fail5:
618         EFSYS_PROBE(fail5);
619 fail4:
620         EFSYS_PROBE(fail4);
621 fail3:
622         EFSYS_PROBE(fail3);
623 fail2:
624         EFSYS_PROBE(fail2);
625 fail1:
626         EFSYS_PROBE1(fail1, int, rc);
627
628         return (rc);
629 }
630
631         __checkReturn   int
632 sft9001_verify(
633         __in            efx_nic_t *enp)
634 {
635         efx_port_t *epp = &(enp->en_port);
636         int rc;
637
638         if ((rc = xphy_pkg_verify(enp, epp->ep_port, SFT9001_MMD_MASK)) != 0)
639                 goto fail1;
640
641         return (0);
642
643 fail1:
644         EFSYS_PROBE1(fail1, int, rc);
645
646         return (rc);
647 }
648
649         __checkReturn   int
650 sft9001_uplink_check(
651         __in            efx_nic_t *enp,
652         __out           boolean_t *upp)
653 {
654         efx_port_t *epp = &(enp->en_port);
655         efx_word_t word;
```

```
656         int rc;

658         if (epp->ep_mac_type != EFX_MAC_FALCON_XMAC) {
659                 rc = ENOTSUP;
660                 goto fail1;
661         }

663         if ((rc = falcon_mdio_read(enp, epp->ep_port, PHY_XS_MMD,
664             PHY_XS_LANE_STATUS_REG, &word)) != 0)
665                 goto fail2;

667         *upp = ((EFX_WORD_FIELD(word, PHY_XS_ALIGNED) != 0) &&
668             (EFX_WORD_FIELD(word, PHY_XS_LANE0_SYNC) != 0) &&
669             (EFX_WORD_FIELD(word, PHY_XS_LANE1_SYNC) != 0) &&
670             (EFX_WORD_FIELD(word, PHY_XS_LANE2_SYNC) != 0) &&
671             (EFX_WORD_FIELD(word, PHY_XS_LANE3_SYNC) != 0));

673         return (0);

675 fail2:
676         EFSYS_PROBE(fail2);
677 fail1:
678         EFSYS_PROBE1(fail1, int, rc);

680         return (rc);
681 }

683 static  __checkReturn   int
684 sft9001_lp_cap_get(
685         __in            efx_nic_t *enp,
686         __out           unsigned int *maskp)
687 {
688         efx_port_t *epp = &(enp->en_port);
689         efx_word_t word;
690         int rc;

692         *maskp = 0;

694         if ((rc = falcon_mdio_read(enp, epp->ep_port, AN_MMD,
695             AN_LP_BP_CAP_REG, &word)) != 0)
696                 goto fail1;

698         if (EFX_WORD_FIELD(word, AN_LP_TA_10BASE_T) != 0)
699                 *maskp |= (1 << EFX_PHY_CAP_10HDX);

701         if (EFX_WORD_FIELD(word, AN_LP_TA_10BASE_T_FDX) != 0)
702                 *maskp |= (1 << EFX_PHY_CAP_10FDX);

704         if (EFX_WORD_FIELD(word, AN_LP_TA_100BASE_TX) != 0)
705                 *maskp |= (1 << EFX_PHY_CAP_100HDX);

707         if (EFX_WORD_FIELD(word, AN_LP_TA_100BASE_TX_FDX) != 0)
708                 *maskp |= (1 << EFX_PHY_CAP_100FDX);

710         if (EFX_WORD_FIELD(word, AN_LP_TA_PAUSE) != 0)
711                 *maskp |= (1 << EFX_PHY_CAP_PAUSE);

713         if (EFX_WORD_FIELD(word, AN_LP_TA_ASM_DIR) != 0)
714                 *maskp |= (1 << EFX_PHY_CAP_ASYM);

716         if ((rc = falcon_mdio_read(enp, epp->ep_port, CL22EXT_MMD,
717             CL22EXT_MS_STATUS_REG, &word)) != 0)
718                 goto fail2;

720         if (EFX_WORD_FIELD(word, CL22EXT_1000BASE_T_LP) != 0)
721                 *maskp |= (1 << EFX_PHY_CAP_1000HDX);
```



```
723         if (EFX_WORD_FIELD(word, CL22EXT_1000BASE_T_FDX_LP) != 0)
724                 *maskp |= (1 << EFX_PHY_CAP_1000FDX);

726         if ((rc = falcon_mdio_read(enp, epp->ep_port, AN_MMD,
727             AN_10G_BASE_T_STATUS_REG, &word)) != 0)
728                 goto fail3;

730         if (EFX_WORD_FIELD(word, AN_10G_BASE_T_LP) != 0)
731                 *maskp |= (1 << EFX_PHY_CAP_10000FDX);

733         return (0);

735 fail3:
736         EFSYS_PROBE(fail3);
737 fail2:
738         EFSYS_PROBE(fail2);
739 fail1:
740         EFSYS_PROBE1(fail1, int, rc);

742         return (rc);
743 }

745         __checkReturn   int
746 sft9001_downlink_check(
747         __in            efx_nic_t *enp,
748         __out           efx_link_mode_t *modep,
749         __out           unsigned int *fcntlp,
750         __out           uint32_t *lp_cap_maskp)
751 {
752         efx_port_t *epp = &(enp->en_port);
753         unsigned int fcntl = epp->ep_fcntl;
754         unsigned int lp_cap_mask = epp->ep_lp_cap_mask;
755         boolean_t up;
756         uint32_t common;
757         int rc;

759 #if EFSYS_OPT_LOOPBACK
760         switch (epp->ep_loopback_type) {
761         case EFX_LOOPBACK_PHY_XS:
762                 rc = xphy_mmd_fault(enp, epp->ep_port, &up);
763                 if (rc != 0)
764                         goto fail1;

766                 *modep = (up) ? EFX_LINK_10000FDX : EFX_LINK_DOWN;
767                 goto done;

769         case EFX_LOOPBACK_PCS:
770         case EFX_LOOPBACK_PMA_PMD:
771                 rc = xphy_mmd_check(enp, epp->ep_port, PHY_XS_MMD, &up);
772                 if (rc != 0)
773                         goto fail1;

775                 *modep = (up) ? EFX_LINK_10000FDX : EFX_LINK_DOWN;
776                 goto done;

778         case EFX_LOOPBACK_GPHY:
779                 *modep = EFX_LINK_1000FDX;
780                 goto done;

782         default:
783                 break;
784         }
785 #endif  /* EFSYS_OPT_LOOPBACK */

787         if ((rc = xphy_mmd_check(enp, epp->ep_port, AN_MMD, &up)) != 0)
```

```
788                 goto fail1;

790         if (!up) {
791                 *modep = EFX_LINK_DOWN;
792                 goto done;
793         }

795         /* Check the link partner capabilities */
796         if ((rc = sft9001_lp_cap_get(enp, &lp_cap_mask)) != 0)
797                 goto fail2;

799         /* Resolve the common capabilities */
800         common = epp->ep_adv_cap_mask & lp_cap_mask;

802         /* The 'best' common link mode should be the one in operation */
803         if (common & (1 << EFX_PHY_CAP_10000FDX)) {
804                 *modep = EFX_LINK_10000FDX;
805         } else if (common & (1 << EFX_PHY_CAP_1000FDX)) {
806                 *modep = EFX_LINK_1000FDX;
807         } else if (common & (1 << EFX_PHY_CAP_100FDX)) {
808                 *modep = EFX_LINK_100FDX;
809         } else if (common & (1 << EFX_PHY_CAP_100HDX)) {
810                 *modep = EFX_LINK_100HDX;
811         } else {
812                 *modep = EFX_LINK_UNKNOWN;
813         }

815         /* Determine negotiated or forced flow control mode */
816         fcntl = 0;
817         if (epp->ep_fcntl_autoneg) {
818                 if (common & (1 << EFX_PHY_CAP_PAUSE))
819                         fcntl = EFX_FCNTL_GENERATE | EFX_FCNTL_RESPOND;
820                 else if (common & (1 << EFX_PHY_CAP_ASYM)) {
821                         if (epp->ep_adv_cap_mask & (1 << EFX_PHY_CAP_PAUSE))
822                                 fcntl = EFX_FCNTL_RESPOND;
823                         else if (lp_cap_mask & (1 << EFX_PHY_CAP_PAUSE))
824                                 fcntl = EFX_FCNTL_GENERATE;
825                 }
826         } else {
827                 if (epp->ep_adv_cap_mask & (1 << EFX_PHY_CAP_PAUSE))
828                         fcntl = EFX_FCNTL_GENERATE | EFX_FCNTL_RESPOND;
829                 if (epp->ep_adv_cap_mask & (1 << EFX_PHY_CAP_ASYM))
830                         fcntl ^= EFX_FCNTL_GENERATE;
831         }

833 done:
834         *fcntlp = fcntl;
835         *lp_cap_maskp = lp_cap_mask;

837         return (0);

839 fail2:
840         EFSYS_PROBE(fail2);
841 fail1:
842         EFSYS_PROBE1(fail1, int, rc);

844         return (rc);
845 }

847         __checkReturn   int
848 sft9001_oui_get(
849         __in            efx_nic_t *enp,
850         __out           uint32_t *ouip)
851 {
852         efx_port_t *epp = &(enp->en_port);
853         int rc;
```



```
855         if ((rc = xphy_mmd_oui_get(enp, epp->ep_port, PMA_PMD_MMD, ouip)) != 0)
856                 goto fail1;

858         return (0);

860 fail1:
861         EFSYS_PROBE1(fail1, int, rc);

863         return (rc);
864 }

866 #define SFT9001_STAT_SET(_stat, _mode, _id, _val)                       \
867         do {                                                            \
868                 (_mode) |= (1 << (_id));                                \
869                 (_stat)[_id] = (uint32_t)(_val);                        \
870         _NOTE(CONSTANTCONDITION)                                        \
871         } while (B_FALSE)

873 static  __checkReturn   int
874 sft9001_rev_get(
875         __in            efx_nic_t *enp,
876         __out           uint8_t *ap,
877         __out           uint8_t *bp,
878         __out           uint8_t *cp,
879         __out           uint8_t *dp)
880 {
881         efx_port_t *epp = &(enp->en_port);
882         efx_word_t word;
883         int rc;

885         if ((rc = falcon_mdio_read(enp, epp->ep_port, PMA_PMD_MMD,
886             PMA_PMD_FW_REV0_REG, &word)) != 0)
887                 goto fail1;

889         *ap = (uint8_t)EFX_WORD_FIELD(word, EFX_BYTE_1);
890         *bp = (uint8_t)EFX_WORD_FIELD(word, EFX_BYTE_0);

892         if ((rc = falcon_mdio_read(enp, epp->ep_port, PMA_PMD_MMD,
893             PMA_PMD_FW_REV1_REG, &word)) != 0)
894                 goto fail2;

896         *cp = (uint8_t)EFX_WORD_FIELD(word, EFX_BYTE_1);
897         *dp = (uint8_t)EFX_WORD_FIELD(word, EFX_BYTE_0);

899         return (0);

901 fail2:
902         EFSYS_PROBE(fail2);
903 fail1:
904         EFSYS_PROBE1(fail1, int, rc);

906         return (rc);
907 }

909 #if EFSYS_OPT_PHY_STATS

911 static  __checkReturn                   int
912 sft9001_pma_pmd_stats_update(
913         __in                            efx_nic_t *enp,
914         __inout                         uint64_t *maskp,
915         __inout_ecount(EFX_PHY_NSTATS)  uint32_t *stat)
916 {
917         efx_port_t *epp = &(enp->en_port);
918         efx_word_t word;
919         uint8_t a;
```

```
920         uint8_t b;
921         uint8_t c;
922         uint8_t d;
923         int rc;

925         if ((rc = falcon_mdio_read(enp, epp->ep_port, PMA_PMD_MMD,
926             PMA_PMD_STATUS1_REG, &word)) != 0)
927                 goto fail1;

929         SFT9001_STAT_SET(stat, *maskp, EFX_PHY_STAT_PMA_PMD_LINK_UP,
930             (EFX_WORD_FIELD(word, PMA_PMD_LINK_UP) != 0) ? 1 : 0);

932         if ((rc = falcon_mdio_read(enp, epp->ep_port, PMA_PMD_MMD,
933             PMA_PMD_STATUS2_REG, &word)) != 0)
934                 goto fail2;

936         SFT9001_STAT_SET(stat, *maskp, EFX_PHY_STAT_PMA_PMD_RX_FAULT,
937             (EFX_WORD_FIELD(word, PMA_PMD_RX_FAULT) != 0) ? 1 : 0);
938         SFT9001_STAT_SET(stat, *maskp, EFX_PHY_STAT_PMA_PMD_TX_FAULT,
939             (EFX_WORD_FIELD(word, PMA_PMD_TX_FAULT) != 0) ? 1 : 0);

941         if ((rc = sft9001_rev_get(enp, &a, &b, &c, &d)) != 0)
942                 goto fail3;

944         SFT9001_STAT_SET(stat, *maskp, EFX_PHY_STAT_PMA_PMD_REV_A, a);
945         SFT9001_STAT_SET(stat, *maskp, EFX_PHY_STAT_PMA_PMD_REV_B, b);
946         SFT9001_STAT_SET(stat, *maskp, EFX_PHY_STAT_PMA_PMD_REV_C, c);
947         SFT9001_STAT_SET(stat, *maskp, EFX_PHY_STAT_PMA_PMD_REV_D, d);

949         if ((rc = falcon_mdio_read(enp, epp->ep_port, PMA_PMD_MMD,
950             PMA_PMD_CHANNELA_SNR_REG, &word)) != 0)
951                 goto fail4;

953         SFT9001_STAT_SET(stat, *maskp, EFX_PHY_STAT_SNR_A,
954             EFX_WORD_FIELD(word, PMA_PMD_SNR));

956         if ((rc = falcon_mdio_read(enp, epp->ep_port, PMA_PMD_MMD,
957             PMA_PMD_CHANNELB_SNR_REG, &word)) != 0)
958                 goto fail5;

960         SFT9001_STAT_SET(stat, *maskp, EFX_PHY_STAT_SNR_B,
961             EFX_WORD_FIELD(word, PMA_PMD_SNR));

963         if ((rc = falcon_mdio_read(enp, epp->ep_port, PMA_PMD_MMD,
964             PMA_PMD_CHANNELC_SNR_REG, &word)) != 0)
965                 goto fail6;

967         SFT9001_STAT_SET(stat, *maskp, EFX_PHY_STAT_SNR_C,
968             EFX_WORD_FIELD(word, PMA_PMD_SNR));

970         if ((rc = falcon_mdio_read(enp, epp->ep_port, PMA_PMD_MMD,
971             PMA_PMD_CHANNELD_SNR_REG, &word)) != 0)
972                 goto fail7;

974         SFT9001_STAT_SET(stat, *maskp, EFX_PHY_STAT_SNR_D,
975             EFX_WORD_FIELD(word, PMA_PMD_SNR));

977         return (0);

979 fail7:
980         EFSYS_PROBE(fail7);
981 fail6:
982         EFSYS_PROBE(fail6);
983 fail5:
984         EFSYS_PROBE(fail5);
985 fail4:
```



```
986         EFSYS_PROBE(fail4);
987 fail3:
988         EFSYS_PROBE(fail3);
989 fail2:
990         EFSYS_PROBE(fail2);
991 fail1:
992         EFSYS_PROBE1(fail1, int, rc);

994         return (rc);
995 }

997 static  __checkReturn                   int
998 sft9001_pcs_stats_update(
999         __in                            efx_nic_t *enp,
1000        __inout                         uint64_t *maskp,
1001        __inout_ecount(EFX_PHY_NSTATS)  uint32_t *stat)
1002 {
1003        efx_port_t *epp = &(enp->en_port);
1004        efx_word_t word;
1005        int rc;

1007        if ((rc = falcon_mdio_read(enp, epp->ep_port, PCS_MMD,
1008            PCS_STATUS1_REG, &word)) != 0)
1009                goto fail1;

1011        SFT9001_STAT_SET(stat, *maskp, EFX_PHY_STAT_PCS_LINK_UP,
1012            (EFX_WORD_FIELD(word, PCS_LINK_UP) != 0) ? 1 : 0);

1014        if ((rc = falcon_mdio_read(enp, epp->ep_port, PCS_MMD,
1015            PCS_STATUS2_REG, &word)) != 0)
1016                goto fail2;

1018        SFT9001_STAT_SET(stat, *maskp, EFX_PHY_STAT_PCS_RX_FAULT,
1019            (EFX_WORD_FIELD(word, PCS_RX_FAULT) != 0) ? 1 : 0);
1020        SFT9001_STAT_SET(stat, *maskp, EFX_PHY_STAT_PCS_TX_FAULT,
1021            (EFX_WORD_FIELD(word, PCS_TX_FAULT) != 0) ? 1 : 0);

1023        if ((rc = falcon_mdio_read(enp, epp->ep_port, PCS_MMD,
1024            PCS_10GBASE_T_STATUS2_REG, &word)) != 0)
1025                goto fail3;

1027        SFT9001_STAT_SET(stat, *maskp, EFX_PHY_STAT_PCS_BER,
1028            EFX_WORD_FIELD(word, PCS_BER));
1029        SFT9001_STAT_SET(stat, *maskp, EFX_PHY_STAT_PCS_BLOCK_ERRORS,
1030            EFX_WORD_FIELD(word, PCS_ERR));

1032        return (0);

1034 fail3:
1035        EFSYS_PROBE(fail3);
1036 fail2:
1037        EFSYS_PROBE(fail2);
1038 fail1:
1039        EFSYS_PROBE1(fail1, int, rc);

1041        return (rc);
1042 }

1044 static  __checkReturn                   int
1045 sft9001_phy_xs_stats_update(
1046        __in                            efx_nic_t *enp,
1047        __inout                         uint64_t *maskp,
1048        __inout_ecount(EFX_PHY_NSTATS)  uint32_t *stat)
1049 {
1050        efx_port_t *epp = &(enp->en_port);
1051        efx_word_t word;
```

```
1052         int rc;

1054         if ((rc = falcon_mdio_read(enp, epp->ep_port, PHY_XS_MMD,
1055             PHY_XS_STATUS1_REG, &word)) != 0)
1056                 goto fail1;

1058         SFT9001_STAT_SET(stat, *maskp, EFX_PHY_STAT_PHY_XS_LINK_UP,
1059             (EFX_WORD_FIELD(word, PHY_XS_LINK_UP) != 0) ? 1 : 0);

1061         if ((rc = falcon_mdio_read(enp, epp->ep_port, PHY_XS_MMD,
1062             PHY_XS_STATUS2_REG, &word)) != 0)
1063                 goto fail2;

1065         SFT9001_STAT_SET(stat, *maskp, EFX_PHY_STAT_PHY_XS_RX_FAULT,
1066             (EFX_WORD_FIELD(word, PHY_XS_RX_FAULT) != 0) ? 1 : 0);
1067         SFT9001_STAT_SET(stat, *maskp, EFX_PHY_STAT_PHY_XS_TX_FAULT,
1068             (EFX_WORD_FIELD(word, PHY_XS_TX_FAULT) != 0) ? 1 : 0);

1070         if ((rc = falcon_mdio_read(enp, epp->ep_port, PHY_XS_MMD,
1071             PHY_XS_LANE_STATUS_REG, &word)) != 0)
1072                 goto fail3;

1074         SFT9001_STAT_SET(stat, *maskp, EFX_PHY_STAT_PHY_XS_ALIGN,
1075             (EFX_WORD_FIELD(word, PHY_XS_ALIGNED) != 0) ? 1 : 0);
1076         SFT9001_STAT_SET(stat, *maskp, EFX_PHY_STAT_PHY_XS_SYNC_A,
1077             (EFX_WORD_FIELD(word, PHY_XS_LANE0_SYNC) != 0) ? 1 : 0);
1078         SFT9001_STAT_SET(stat, *maskp, EFX_PHY_STAT_PHY_XS_SYNC_B,
1079             (EFX_WORD_FIELD(word, PHY_XS_LANE1_SYNC) != 0) ? 1 : 0);
1080         SFT9001_STAT_SET(stat, *maskp, EFX_PHY_STAT_PHY_XS_SYNC_C,
1081             (EFX_WORD_FIELD(word, PHY_XS_LANE2_SYNC) != 0) ? 1 : 0);
1082         SFT9001_STAT_SET(stat, *maskp, EFX_PHY_STAT_PHY_XS_SYNC_D,
1083             (EFX_WORD_FIELD(word, PHY_XS_LANE3_SYNC) != 0) ? 1 : 0);

1085         return (0);

1087 fail3:
1088         EFSYS_PROBE(fail3);
1089 fail2:
1090         EFSYS_PROBE(fail2);
1091 fail1:
1092         EFSYS_PROBE1(fail1, int, rc);

1094         return (rc);
1095 }

1097 static  __checkReturn                   int
1098 sft9001_an_stats_update(
1099         __in                            efx_nic_t *enp,
1100         __inout                         uint64_t *maskp,
1101         __inout_ecount(EFX_PHY_NSTATS)  uint32_t *stat)
1102 {
1103         efx_port_t *epp = &(enp->en_port);
1104         efx_word_t word;
1105         int rc;

1107         if ((rc = falcon_mdio_read(enp, epp->ep_port, AN_MMD,
1108             AN_STATUS1_REG, &word)) != 0)
1109                 goto fail1;

1111         SFT9001_STAT_SET(stat, *maskp, EFX_PHY_STAT_AN_LINK_UP,
1112             (EFX_WORD_FIELD(word, AN_LINK_UP) != 0) ? 1 : 0);

1114         if ((rc = falcon_mdio_read(enp, epp->ep_port, AN_MMD,
1115             AN_10G_BASE_T_STATUS_REG, &word)) != 0)
1116                 goto fail2;
```



```
1118         SFT9001_STAT_SET(stat, *maskp, EFX_PHY_STAT_AN_MASTER,
1119             (EFX_WORD_FIELD(word, AN_MASTER) != 0) ? 1 : 0);
1120         SFT9001_STAT_SET(stat, *maskp, EFX_PHY_STAT_AN_LOCAL_RX_OK,
1121             (EFX_WORD_FIELD(word, AN_LOCAL_RX_OK) != 0) ? 1 : 0);
1122         SFT9001_STAT_SET(stat, *maskp, EFX_PHY_STAT_AN_REMOTE_RX_OK,
1123             (EFX_WORD_FIELD(word, AN_REMOTE_RX_OK) != 0) ? 1 : 0);

1125         return (0);

1127 fail2:
1128         EFSYS_PROBE(fail2);
1129 fail1:
1130         EFSYS_PROBE1(fail1, int, rc);

1132         return (rc);
1133 }

1135 static  __checkReturn                   int
1136 sft9001_cl22ext_stats_update(
1137         __in                            efx_nic_t *enp,
1138         __inout                         uint64_t *maskp,
1139         __inout_ecount(EFX_PHY_NSTATS)  uint32_t *stat)
1140 {
1141         efx_port_t *epp = &(enp->en_port);
1142         efx_word_t word;
1143         int rc;

1145         if ((rc = falcon_mdio_read(enp, epp->ep_port, CL22EXT_MMD,
1146             CL22EXT_STATUS_REG, &word)) != 0)
1147                 goto fail1;

1149         SFT9001_STAT_SET(stat, *maskp, EFX_PHY_STAT_CL22EXT_LINK_UP,
1150             (EFX_WORD_FIELD(word, CL22EXT_LINK_UP) != 0) ? 1 : 0);

1152         return (0);

1154 fail1:
1155         EFSYS_PROBE1(fail1, int, rc);

1157         return (rc);
1158 }


1161         __checkReturn                   int
1162 sft9001_stats_update(
1163         __in                            efx_nic_t *enp,
1164         __in                            efsys_mem_t *esmp,
1165         __out_ecount(EFX_PHY_NSTATS)    uint32_t *stat)
1166 {
1167         efx_port_t *epp = &(enp->en_port);
1168         efx_nic_cfg_t *encp = &(enp->en_nic_cfg);
1169         uint64_t mask = 0;
1170         uint32_t oui;
1171         int rc;

1173         _NOTE(ARGUNUSED(esmp))

1175         if ((rc = xphy_mmd_oui_get(enp, epp->ep_port, PMA_PMD_MMD, &oui)) != 0)
1176                 goto fail1;

1178         SFT9001_STAT_SET(stat, mask, EFX_PHY_STAT_OUI, oui);

1180         if ((rc = sft9001_pma_pmd_stats_update(enp, &mask, stat)) != 0)
1181                 goto fail2;

1183         if ((rc = sft9001_pcs_stats_update(enp, &mask, stat)) != 0)
```

```
1184                 goto fail3;
1185
1186         if ((rc = sft9001_phy_xs_stats_update(enp, &mask, stat)) != 0)
1187                 goto fail4;
1188
1189         if ((rc = sft9001_an_stats_update(enp, &mask, stat)) != 0)
1190                 goto fail5;
1191
1192         if ((rc = sft9001_cl22ext_stats_update(enp, &mask, stat)) != 0)
1193                 goto fail6;
1194
1195         /* Ensure all the supported statistics are up to date */
1196         EFSYS_ASSERT(mask == encp->enc_phy_stat_mask);
1197
1198         return (0);
1199
1200 fail6:
1201         EFSYS_PROBE(fail6);
1202 fail5:
1203         EFSYS_PROBE(fail5);
1204 fail4:
1205         EFSYS_PROBE(fail4);
1206 fail3:
1207         EFSYS_PROBE(fail3);
1208 fail2:
1209         EFSYS_PROBE(fail2);
1210 fail1:
1211         EFSYS_PROBE1(fail1, int, rc);
1212
1213         return (rc);
1214 }
1215 #endif  /* EFSYS_OPT_PHY_STATS */
1216
1217 #if EFSYS_OPT_PHY_PROPS
1218
1219 #if EFSYS_OPT_NAMES
1220 /* START MKCONFIG GENERATED Sft9001PhyPropNamesBlock 575ed5e718aa4657 */
1221 static const char       __cs * __cs __sft9001_prop_name[] = {
1222         "short_reach",
1223         "robust",
1224 };
1225
1226 /* END MKCONFIG GENERATED Sft9001PhyPropNamesBlock */
1227
1228                 const char __cs *
1229 sft9001_prop_name(
1230         __in    efx_nic_t *enp,
1231         __in    unsigned int id)
1232 {
1233         _NOTE(ARGUNUSED(enp))
1234
1235         EFSYS_ASSERT3U(id, <, SFT9001_NPROPS);
1236
1237         return (__sft9001_prop_name[id]);
1238 }
1239 #endif  /* EFSYS_OPT_NAMES */
1240
1241         __checkReturn   int
1242 sft9001_prop_get(
1243         __in            efx_nic_t *enp,
1244         __in            unsigned int id,
1245         __in            uint32_t flags,
1246         __out           uint32_t *valp)
1247 {
1248         uint32_t val;
1249         int rc;
```



```
1251         switch (id) {
1252         case SFT9001_SHORT_REACH: {
1253                 boolean_t on;
1254
1255                 if (flags * EFX_PHY_PROP_DEFAULT) {
1256                         val = 0;
1257                         break;
1258                 }
1259
1260                 if ((rc = sft9001_short_reach_get(enp, &on)) != 0)
1261                         goto fail1;
1262
1263                 val = (on) ? 1 : 0;
1264                 break;
1265         }
1266         case SFT9001_ROBUST: {
1267                 boolean_t on;
1268
1269                 if (flags * EFX_PHY_PROP_DEFAULT) {
1270                         val = 0;
1271                         break;
1272                 }
1273
1274                 if ((rc = sft9001_robust_get(enp, &on)) != 0)
1275                         goto fail1;
1276
1277                 val = (on) ? 1 : 0;
1278                 break;
1279         }
1280         default:
1281                 EFSYS_ASSERT(B_FALSE);
1282
1283                 val = 0;
1284                 break;
1285         }
1286
1287         *valp = val;
1288         return (0);
1289
1290 fail1:
1291         EFSYS_PROBE1(fail1, int, rc);
1292
1293         return (rc);
1294 }
1295
1296         __checkReturn   int
1297 sft9001_prop_set(
1298         __in            efx_nic_t *enp,
1299         __in            unsigned int id,
1300         __in            uint32_t val)
1301 {
1302         efx_port_t *epp = &(enp->en_port);
1303         int rc;
1304
1305         switch (id) {
1306         case SFT9001_SHORT_REACH:
1307                 if ((rc = sft9001_an_set(enp, B_FALSE)) != 0)
1308                         goto fail1;
1309
1310                 if ((rc = sft9001_short_reach_set(enp, (val != 0))) != 0)
1311                         goto fail2;
1312
1313 #if EFSYS_OPT_LOOPBACK
1314                 if (epp->ep_loopback_type == EFX_LOOPBACK_OFF) {
1315                         if ((rc = sft9001_an_set(enp, B_TRUE)) != 0)
```

```
1316                                    goto fail3;
1317                    }
1318 #else   /* EFSYS_OPT_LOOPBACK */
1319                    if ((rc = sft9001_an_set(enp, B_TRUE)) != 0)
1320                            goto fail3;
1321 #endif  /* EFSYS_OPT_LOOPBACK */

1323                    break;

1325            case SFT9001_ROBUST:
1326                    if ((rc = sft9001_an_set(enp, B_FALSE)) != 0)
1327                            goto fail1;

1329                    if ((rc = sft9001_robust_set(enp, (val != 0))) != 0)
1330                            goto fail2;

1332 #if EFSYS_OPT_LOOPBACK
1333                    if (epp->ep_loopback_type == EFX_LOOPBACK_OFF) {
1334                            if ((rc = sft9001_an_set(enp, B_TRUE)) != 0)
1335                                    goto fail3;
1336                    }
1337 #else   /* EFSYS_OPT_LOOPBACK */
1338                    if ((rc = sft9001_an_set(enp, B_TRUE)) != 0)
1339                            goto fail3;
1340 #endif  /* EFSYS_OPT_LOOPBACK */

1342                    break;

1344            default:
1345                    EFSYS_ASSERT(B_FALSE);
1346                    break;
1347            }

1349            return (0);

1351 fail3:
1352            EFSYS_PROBE(fail3);
1353 fail2:
1354            EFSYS_PROBE(fail2);
1355 fail1:
1356            EFSYS_PROBE1(fail1, int, rc);

1358            return (rc);
1359 }
1360 #endif  /* EFSYS_OPT_PHY_PROPS */

1362 #if EFSYS_OPT_NVRAM_SFT9001

1364 static  __checkReturn   int
1365 sft9001_ssr(
1366            __in            efx_nic_t *enp,
1367            __in            boolean_t loader)
1368 {
1369            efx_port_t *epp = &(enp->en_port);
1370            efx_oword_t oword;
1371            efx_word_t word;
1372            int state;
1373            int rc;

1375            EFSYS_LOCK(enp->en_eslp, state);

1377            /* Lock I2C bus and pull GPIO(3) low */
1378            EFSYS_ASSERT(!enp->en_u.falcon.enu_i2c_locked);
1379            enp->en_u.falcon.enu_i2c_locked = B_TRUE;

1381            EFX_BAR_READO(enp, FR_AB_GPIO_CTL_REG, &oword);
```



```
1382            EFX_SET_OWORD_FIELD(oword, FRF_AB_GPIO3_OEN, loader ? 1 : 0);
1383            EFX_SET_OWORD_FIELD(oword, FRF_AB_GPIO3_OUT, 0);
1384            EFX_BAR_WRITEO(enp, FR_AB_GPIO_CTL_REG, &oword);

1386            EFSYS_UNLOCK(enp->en_eslp, state);

1388            /* Special software reset */
1389            if ((rc = falcon_mdio_read(enp, epp->ep_port, PMA_PMD_MMD,
1390                PMA_PMD_XCONTROL_REG, &word)) != 0)
1391                    goto fail1;
1392            EFX_SET_WORD_FIELD(word, SSR, 1);
1393            if ((rc = falcon_mdio_write(enp, epp->ep_port, PMA_PMD_MMD,
1394                PMA_PMD_XCONTROL_REG, &word)) != 0)
1395                    goto fail2;

1397            /* Sleep for 500ms */
1398            EFSYS_SLEEP(500000);

1400            goto out;

1402 fail2:
1403            EFSYS_PROBE(fail2);
1404 fail1:
1405            EFSYS_PROBE1(fail1, int, rc);

1407 out:
1408            /* Unlock the I2C bus and restore GPIO(3) */
1409            EFSYS_LOCK(enp->en_eslp, state);
1410            enp->en_u.falcon.enu_i2c_locked = B_FALSE;

1412            EFX_BAR_READO(enp, FR_AB_GPIO_CTL_REG, &oword);
1413            EFX_SET_OWORD_FIELD(oword, FRF_AB_GPIO3_OEN, 0);
1414            EFX_BAR_WRITEO(enp, FR_AB_GPIO_CTL_REG, &oword);

1416            EFSYS_UNLOCK(enp->en_eslp, state);

1418            return (rc);
1419 }

1421 static  __checkReturn           int
1422 sft9001_loader_wait(
1423            __in                    efx_nic_t *enp)
1424 {
1425            efx_port_t *epp = &(enp->en_port);
1426            efx_word_t word;
1427            unsigned int count;
1428            unsigned int response;
1429            int rc;

1431            /* Wait up to 20s for the command to complete */
1432            for (count = 0; count < 200; count++) {
1433                    EFSYS_SLEEP(100000);    /* 100ms */

1435                    if ((rc = falcon_mdio_read(enp, epp->ep_port, LOADER_MMD,
1436                        LOADER_CMD_RESPONSE_REG, &word)) != 0)
1437                            goto fail1;

1439                    response = EFX_WORD_FIELD(word, EFX_WORD_0);
1440                    if (response == LOADER_RESPONSE_OK)
1441                            return (0);
1442                    if (response != LOADER_RESPONSE_BUSY) {
1443                            rc = EIO;
1444                            goto fail2;
1445                    }
1446            }
```

```
1448         rc = ETIMEDOUT;

1450         EFSYS_PROBE(fail3);
1451 fail2:
1452         EFSYS_PROBE(fail2);
1453 fail1:
1454         EFSYS_PROBE1(fail1, int, rc);

1456         return (rc);
1457 }

1459 static  __checkReturn           int
1460 sft9001_program_loader(
1461         __in                    efx_nic_t *enp,
1462         __in                    unsigned int offset,
1463         __in                    size_t words)
1464 {
1465         efx_port_t *epp = &(enp->en_port);
1466         efx_word_t word;
1467         int rc;

1469         /* Setup address of block transfer */
1470         EFX_POPULATE_WORD_1(word, EFX_WORD_0, offset);
1471         if ((rc = falcon_mdio_write(enp, epp->ep_port, LOADER_MMD,
1472             LOADER_FLASH_ADDR_LOW_REG, &word)) != 0)
1473                 goto fail1;

1475         EFX_POPULATE_WORD_1(word, EFX_WORD_0, (offset >> 16));
1476         if ((rc = falcon_mdio_write(enp, epp->ep_port, LOADER_MMD,
1477             LOADER_FLASH_ADDR_HI_REG, &word)) != 0)
1478                 goto fail2;

1480         EFX_POPULATE_WORD_1(word, EFX_WORD_0, words);
1481         if ((rc = falcon_mdio_write(enp, epp->ep_port, LOADER_MMD,
1482             LOADER_ACTUAL_BUFF_SZ_REG, &word)) != 0)
1483                 goto fail3;

1485         return (0);

1487 fail3:
1488         EFSYS_PROBE(fail3);
1489 fail2:
1490         EFSYS_PROBE(fail2);
1491 fail1:
1492         EFSYS_PROBE1(fail1, int, rc);

1494         return (rc);
1495 }

1497 __checkReturn           int
1498 sft9001_nvram_size(
1499         __in                    efx_nic_t *enp,
1500         __out                   size_t *sizep)
1501 {
1502         _NOTE(ARGUNUSED(enp))
1503         EFSYS_ASSERT(sizep);

1505         *sizep = FIRMWARE_MAX_SIZE;

1507         return (0);
1508 }

1510         __checkReturn           int
1511 sft9001_nvram_get_version(
1512         __in                    efx_nic_t *enp,
1513         __out                   uint32_t *subtypep,
```



```
1514         __out_ecount(4)         uint16_t version[4])
1515 {
1516         efx_nic_cfg_t *encp = &(enp->en_nic_cfg);
1517         uint8_t a, b, c, d;
1518         int rc;

1520         if ((rc = sft9001_rev_get(enp, &a, &b, &c, &d)) != 0)
1521                 goto fail1;

1523         version[0] = a;
1524         version[1] = b;
1525         version[2] = c;
1526         version[3] = d;

1528         switch (encp->enc_phy_type) {
1529         case EFX_PHY_SFT9001A:
1530                 *subtypep = PHY_TYPE_SFT9001A_DECODE;
1531                 break;
1532         case EFX_PHY_SFT9001B:
1533                 *subtypep = PHY_TYPE_SFT9001B_DECODE;
1534                 break;
1535         default:
1536                 EFSYS_ASSERT(0);
1537                 *subtypep = 0;
1538         }

1540         return (0);

1542 fail1:
1543         EFSYS_PROBE1(fail1, int, rc);

1545         return (0);
1546 }

1548         __checkReturn           int
1549 sft9001_nvram_rw_start(
1550         __in                    efx_nic_t *enp,
1551         __out                   size_t *block_sizep)
1552 {
1553         efx_port_t *epp = &(enp->en_port);
1554         sft9001_firmware_header_t header;
1555         unsigned int pos;
1556         efx_word_t word;
1557         int rc;

1559         /* Reboot without starting the firmware */
1560         if ((rc = sft9001_ssr(enp, B_TRUE)) != 0)
1561                 goto fail1;

1563         /* Check that the C166 is idle, and in download mode */
1564         if ((rc = falcon_mdio_read(enp, epp->ep_port, PCS_MMD,
1565                     PCS_BOOT_STATUS_REG, &word)) != 0)
1566                 goto fail2;
1567         if (EFX_WORD_FIELD(word, BOOT_STATUS) == 0 ||
1568             EFX_WORD_FIELD(word, BOOT_PROGRESS) != MDIO_WAIT_DECODE) {
1569                 rc = ETIMEDOUT;
1570                 goto fail3;
1571         }

1573         /* Download loader code */
1574         EFX_ZERO_WORD(word);
1575         if ((rc = falcon_mdio_write(enp, epp->ep_port, PCS_MMD,
1576                                     PCS_LM_RAM_LS_ADDR_REG, &word)) != 0)
1577                 goto fail4;
1578         if ((rc = falcon_mdio_write(enp, epp->ep_port, PCS_MMD,
1579                                     PCS_LM_RAM_MS_ADDR_REG, &word)) != 0)
```

```
1580                goto fail5;
1581        for (pos = 0; pos < sft9001_loader_size / sizeof (uint16_t); pos++) {
1582                /* Firmware is little endian */
1583                word.ew_u8[0] = sft9001_loader[pos];
1584                word.ew_u8[1] = sft9001_loader[pos+1];
1585                if ((rc = falcon_mdio_write(enp, epp->ep_port, PCS_MMD,
1586                            PCS_LM_RAM_DATA_REG, &word)) != 0)
1587                        goto fail6;
1588        }

1590        /* Sleep for 500ms */
1591        EFSYS_SLEEP(500000);

1593        /* Start downloaded code */
1594        if ((rc = falcon_mdio_read(enp, epp->ep_port, PCS_MMD,
1595                            PCS_BOOT_STATUS_REG, &word)) != 0)
1596                goto fail7;
1597        EFX_SET_WORD_FIELD(word, CODE_DOWNLOAD, 1);
1598        if ((rc = falcon_mdio_write(enp, epp->ep_port, PCS_MMD,
1599                            PCS_BOOT_STATUS_REG, &word)) != 0)
1600                goto fail8;

1602        /* Sleep 1s */
1603        EFSYS_SLEEP(1000000);

1605        /* And check it started */
1606        if ((rc = falcon_mdio_read(enp, epp->ep_port, PCS_MMD,
1607                            PCS_BOOT_STATUS_REG, &word)) != 0)
1608                goto fail9;

1610        if (EFX_WORD_FIELD(word, CODE_STARTED) == 0) {
1611                rc = ETIMEDOUT;
1612                goto fail10;
1613        }

1615        /* Verify program block size is appropriate */
1616        if ((rc = falcon_mdio_read(enp, epp->ep_port, LOADER_MMD,
1617                            LOADER_MAX_BUFF_SZ_REG, &word)) != 0)
1618                goto fail11;
1619        if (EFX_WORD_FIELD(word, EFX_WORD_0) < FIRMWARE_BLOCK_SIZE) {
1620                rc = EIO;
1621                goto fail12;
1622        }
1623        if (block_sizep != NULL)
1624                *block_sizep = FIRMWARE_BLOCK_SIZE;

1626        /* Read firmware header */
1627        if ((rc = sft9001_nvram_read_chunk(enp, 0, (void *)&header,
1628            sizeof (header))) != 0)
1629                goto fail13;

1631        /* Verify firmware isn't too large */
1632        if (EFX_DWORD_FIELD(header.code_length, EFX_DWORD_0) +
1633            sizeof (sft9001_firmware_header_t) > FIRMWARE_MAX_SIZE) {
1634                rc = EIO;
1635                goto fail14;
1636        }

1638        return (0);

1640 fail14:
1641        EFSYS_PROBE(fail14);
1642 fail13:
1643        EFSYS_PROBE(fail13);
1644 fail12:
1645        EFSYS_PROBE(fail12);
```



```
1646 fail11:
1647        EFSYS_PROBE(fail11);
1648 fail10:
1649        EFSYS_PROBE(fail10);
1650 fail9:
1651        EFSYS_PROBE(fail9);
1652 fail8:
1653        EFSYS_PROBE(fail8);
1654 fail7:
1655        EFSYS_PROBE(fail7);
1656 fail6:
1657        EFSYS_PROBE(fail6);
1658 fail5:
1659        EFSYS_PROBE(fail5);
1660 fail4:
1661        EFSYS_PROBE(fail4);
1662 fail3:
1663        EFSYS_PROBE(fail3);
1664 fail2:
1665        EFSYS_PROBE(fail2);
1666 fail1:
1667        EFSYS_PROBE1(fail1, int, rc);

1669        /* Reboot the PHY into the main firmware */
1670        (void) sft9001_ssr(enp, B_FALSE);

1672        /* Sleep for 500ms */
1673        EFSYS_SLEEP(500000);

1675        return (rc);
1676 }

1678        __checkReturn           int
1679 sft9001_nvram_read_chunk(
1680        __in                    efx_nic_t *enp,
1681        __in                    unsigned int offset,
1682        __out_bcount(size)      caddr_t data,
1683        __in                    size_t size)
1684 {
1685        efx_port_t *epp = &(enp->en_port);
1686        efx_word_t word;
1687        unsigned int pos;
1688        size_t chunk;
1689        size_t words;
1690        int rc;

1692        while (size > 0) {
1693                chunk = MIN(size, FIRMWARE_BLOCK_SIZE);

1695                /* Read in 2byte words */
1696                EFSYS_ASSERT(!(chunk & 0x1));
1697                words = chunk >> 1;

1699                /* Program address/length */
1700                if ((rc = sft9001_program_loader(enp, offset, words)) != 0)
1701                        goto fail1;

1703                /* Select read mode, and wait for buffer to fill */
1704                EFX_POPULATE_WORD_1(word, EFX_WORD_0, LOADER_CMD_READ_FLASH);
1705                if ((rc = falcon_mdio_write(enp, epp->ep_port, LOADER_MMD,
1706                    LOADER_CMD_RESPONSE_REG, &word)) != 0)
1707                        goto fail2;

1709                if ((rc = sft9001_loader_wait(enp)) != 0)
1710                        goto fail3;
```

```
1712                 if ((rc = falcon_mdio_read(enp, epp->ep_port, LOADER_MMD,
1713                     LOADER_WORDS_READ_REG, &word)) != 0)
1714                         goto fail4;

1716                 if (words != (size_t)EFX_WORD_FIELD(word, EFX_WORD_0))
1717                         goto fail5;

1719                 for (pos = 0; pos < words; ++pos) {
1720                         if ((rc = falcon_mdio_read(enp, epp->ep_port,
1721                             LOADER_MMD, LOADER_DATA_REG, &word)) != 0)
1722                                 goto fail6;

1724                         /* Firmware is little endian */
1725                         data[pos] = word.ew_u8[0];
1726                         data[pos+1] = word.ew_u8[1];
1727                 }

1729                 size -= chunk;
1730                 offset += chunk;
1731                 data += chunk;
1732         }

1734         return (0);

1736 fail6:
1737         EFSYS_PROBE(fail6);
1738 fail5:
1739         EFSYS_PROBE(fail5);
1740 fail4:
1741         EFSYS_PROBE(fail4);
1742 fail3:
1743         EFSYS_PROBE(fail3);
1744 fail2:
1745         EFSYS_PROBE(fail2);
1746 fail1:
1747         EFSYS_PROBE1(fail1, int, rc);

1749         return (rc);
1750 }

1752         __checkReturn           int
1753 sft9001_nvram_erase(
1754         __in                    efx_nic_t *enp)
1755 {
1756         efx_port_t *epp = &(enp->en_port);
1757         efx_word_t word;
1758         int rc;

1760         EFX_POPULATE_WORD_1(word, EFX_BYTE_0, LOADER_CMD_ERASE_FLASH);
1761         if ((rc = falcon_mdio_write(enp, epp->ep_port,
1762             LOADER_MMD, LOADER_CMD_RESPONSE_REG, &word)) != 0)
1763                 goto fail1;

1765         if ((rc = sft9001_loader_wait(enp)) != 0)
1766                 goto fail2;

1768         return (0);

1770 fail2:
1771         EFSYS_PROBE(fail2);
1772 fail1:
1773         EFSYS_PROBE1(fail1, int, rc);

1775         return (rc);
1776 }
```



```
1778         __checkReturn           int
1779 sft9001_nvram_write_chunk(
1780         __in                    efx_nic_t *enp,
1781         __in                    unsigned int offset,
1782         __in_bcount(size)       caddr_t data,
1783         __in                    size_t size)
1784 {
1785         efx_port_t *epp = &(enp->en_port);
1786         efx_word_t word;
1787         unsigned int pos;
1788         size_t chunk;
1789         size_t words;
1790         int rc;

1792         while (size > 0) {
1793                 chunk = MIN(size, FIRMWARE_BLOCK_SIZE);

1795                 /* Write in 2byte words */
1796                 EFSYS_ASSERT(!(chunk & 0x1));
1797                 words = chunk >> 1;

1799                 /* Program address/length */
1800                 if ((rc = sft9001_program_loader(enp, offset, words)) != 0)
1801                         goto fail1;

1803                 /* Select write mode */
1804                 EFX_POPULATE_WORD_1(word, EFX_WORD_0, LOADER_CMD_FILL_BUFFER);
1805                 if ((rc = falcon_mdio_write(enp, epp->ep_port, LOADER_MMD,
1806                     LOADER_CMD_RESPONSE_REG, &word)) != 0)
1807                         goto fail2;

1809                 for (pos = 0; pos < words; ++pos) {
1810                         /* Firmware is little-endian */
1811                         word.ew_u8[0] = data[pos];
1812                         word.ew_u8[1] = data[pos+1];

1814                         if ((rc = falcon_mdio_write(enp, epp->ep_port,
1815                             LOADER_MMD, LOADER_DATA_REG, &word)) != 0)
1816                                 goto fail3;
1817                 }

1819                 EFX_POPULATE_WORD_1(word, EFX_WORD_0,
1820                                     LOADER_CMD_PROGRAM_FLASH);
1821                 if ((rc = falcon_mdio_write(enp, epp->ep_port,
1822                     LOADER_MMD, LOADER_CMD_RESPONSE_REG, &word)) != 0)
1823                         goto fail4;

1825                 if ((rc = sft9001_loader_wait(enp)) != 0)
1826                         goto fail5;

1828                 if ((rc = falcon_mdio_read(enp, epp->ep_port, LOADER_MMD,
1829                     LOADER_WORDS_WRITTEN_REG, &word)) != 0)
1830                         goto fail6;

1832                 if (words != EFX_WORD_FIELD(word, EFX_WORD_0))
1833                         goto fail7;

1835                 size -= chunk;
1836                 offset += chunk;
1837                 data += chunk;
1838         }

1840         return (0);

1842 fail7:
1843         EFSYS_PROBE(fail7);
```

```
fail6:
        EFSYS_PROBE(fail6);
fail5:
        EFSYS_PROBE(fail5);
fail4:
        EFSYS_PROBE(fail4);
fail3:
        EFSYS_PROBE(fail3);
fail2:
        EFSYS_PROBE(fail2);
fail1:
        EFSYS_PROBE1(fail1, int, rc);

        return (rc);
}

                                void
sft9001_nvram_rw_finish(
        __in                    efx_nic_t *enp)
{
        efx_port_t *epp = &(enp->en_port);
        efx_word_t word;
        int rc;

        /* Reboot the PHY into the main firmware */
        if ((rc = sft9001_ssr(enp, B_FALSE)) != 0)
                goto fail1;

        /* Sleep for 500ms */
        EFSYS_SLEEP(500000);

        /* Verify that PHY rebooted */
        if ((rc = falcon_mdio_read(enp, epp->ep_port, PCS_MMD,
            PCS_BOOT_STATUS_REG, &word)) != 0)
                goto fail2;
        if (EFX_WORD_FIELD(word, EFX_WORD_0) != 0x7E)
                goto fail3;

        return;

fail3:
        EFSYS_PROBE(fail3);
fail2:
        EFSYS_PROBE(fail2);
fail1:
        EFSYS_PROBE1(fail1, int, rc);
}

#endif  /* EFSYS_OPT_NVRAM_SFT9001 */

#if EFSYS_OPT_PHY_BIST

        __checkReturn   int
sft9001_bist_start(
        __in            efx_nic_t *enp,
        __in            efx_phy_bist_type_t type)
{
        efx_port_t *epp = &(enp->en_port);
        boolean_t break_link = (type == EFX_PHY_BIST_TYPE_CABLE_LONG);
        efx_word_t word;
        int rc;

        if ((rc = falcon_mdio_read(enp, epp->ep_port, PMA_PMD_MMD,
            PMA_PMD_DIAG_CONTROL_REG, &word)) != 0)
                goto fail1;
```



```
        if (EFX_WORD_FIELD(word, DIAG_RUNNING) != 0) {
                rc = EBUSY;
                goto fail2;
        }

        EFX_POPULATE_WORD_3(word,
                            RUN_DIAG_IMMED, 1,
                            LENGTH_UNIT, LENGTH_M_DECODE,
                            BREAK_LINK, break_link ? 1 : 0);
        if ((rc = falcon_mdio_write(enp, epp->ep_port, PMA_PMD_MMD,
            PMA_PMD_DIAG_CONTROL_REG, &word)) != 0)
                goto fail3;

        return (0);

fail3:
        EFSYS_PROBE(fail3);
fail2:
        EFSYS_PROBE(fail2);
fail1:
        EFSYS_PROBE1(fail1, int, rc);

        return (rc);
}

static                  efx_phy_cable_status_t
sft9001_bist_status(
        __in            uint16_t code)
{
        switch (code) {
        case PAIR_BUSY_DECODE:
                return (EFX_PHY_CABLE_STATUS_BUSY);
        case INTER_PAIR_SHORT_DECODE:
                return (EFX_PHY_CABLE_STATUS_INTERPAIRSHORT);
        case INTRA_PAIR_SHORT_DECODE:
                return (EFX_PHY_CABLE_STATUS_INTRAPAIRSHORT);
        case PAIR_OPEN_DECODE:
                return (EFX_PHY_CABLE_STATUS_OPEN);
        case PAIR_OK_DECODE:
                return (EFX_PHY_CABLE_STATUS_OK);
        default:
                return (EFX_PHY_CABLE_STATUS_INVALID);
        }
}

        __checkReturn   int
sft9001_bist_poll(
        __in                    efx_nic_t *enp,
        __in                    efx_phy_bist_type_t type,
        __out                   efx_phy_bist_result_t *resultp,
        __out_opt               uint32_t *value_maskp,
        __out_ecount_opt(count) unsigned long *valuesp,
        __in                    size_t count)
{
        efx_port_t *epp = &(enp->en_port);
        uint32_t value_mask = 0;
        efx_word_t word;
        int rc;

        _NOTE(ARGUNUSED(type))

        *resultp = EFX_PHY_BIST_RESULT_UNKNOWN;

        if ((rc = falcon_mdio_read(enp, epp->ep_port, PMA_PMD_MMD,
            PMA_PMD_DIAG_CONTROL_REG, &word)) != 0)
                goto fail1;
```

```
1977         if (EFX_WORD_FIELD(word, DIAG_RUNNING)) {
1978                 *resultp = EFX_PHY_BIST_RESULT_RUNNING;
1979                 return (0);
1980         }

1982         if ((rc = falcon_mdio_read(enp, epp->ep_port, PMA_PMD_MMD,
1983             PMA_PMD_DIAG_RESULT_REG, &word)) != 0)
1984                 goto fail2;

1986         *resultp = EFX_PHY_BIST_RESULT_PASSED;

1988         if (count > EFX_PHY_BIST_CABLE_STATUS_A) {
1989                 valuesp[EFX_PHY_BIST_CABLE_STATUS_A] =
1990                         sft9001_bist_status(EFX_WORD_FIELD(word, PAIR_A_CODE));
1991                 value_mask |= (1 << EFX_PHY_BIST_CABLE_STATUS_A);
1992         }
1993         if (count > EFX_PHY_BIST_CABLE_STATUS_B) {
1994                 valuesp[EFX_PHY_BIST_CABLE_STATUS_B] =
1995                         sft9001_bist_status(EFX_WORD_FIELD(word, PAIR_B_CODE));
1996                 value_mask |= (1 << EFX_PHY_BIST_CABLE_STATUS_B);
1997         }
1998         if (count > EFX_PHY_BIST_CABLE_STATUS_C) {
1999                 valuesp[EFX_PHY_BIST_CABLE_STATUS_C] =
2000                         sft9001_bist_status(EFX_WORD_FIELD(word, PAIR_C_CODE));
2001                 value_mask |= (1 << EFX_PHY_BIST_CABLE_STATUS_C);
2002         }
2003         if (count > EFX_PHY_BIST_CABLE_STATUS_D) {
2004                 valuesp[EFX_PHY_BIST_CABLE_STATUS_D] =
2005                         sft9001_bist_status(EFX_WORD_FIELD(word, PAIR_D_CODE));
2006                 value_mask |= (1 << EFX_PHY_BIST_CABLE_STATUS_D);
2007         }

2009         if (count > EFX_PHY_BIST_CABLE_LENGTH_A) {
2010                 if ((rc = falcon_mdio_read(enp, epp->ep_port, PMA_PMD_MMD,
2011                     PMA_PMD_DIAG_A_LENGTH_REG, &word)) != 0)
2012                         goto fail3;
2013                 valuesp[EFX_PHY_BIST_CABLE_LENGTH_A] =
2014                         EFX_WORD_FIELD(word, EFX_WORD_0);
2015                 value_mask |= (1 << EFX_PHY_BIST_CABLE_LENGTH_A);
2016         }
2017         if (count > EFX_PHY_BIST_CABLE_LENGTH_B) {
2018                 if ((rc = falcon_mdio_read(enp, epp->ep_port, PMA_PMD_MMD,
2019                     PMA_PMD_DIAG_B_LENGTH_REG, &word)) != 0)
2020                         goto fail4;
2021                 valuesp[EFX_PHY_BIST_CABLE_LENGTH_B] =
2022                         EFX_WORD_FIELD(word, EFX_WORD_0);
2023                 value_mask |= (1 << EFX_PHY_BIST_CABLE_LENGTH_B);
2024         }
2025         if (count > EFX_PHY_BIST_CABLE_LENGTH_C) {
2026                 if ((rc = falcon_mdio_read(enp, epp->ep_port, PMA_PMD_MMD,
2027                     PMA_PMD_DIAG_C_LENGTH_REG, &word)) != 0)
2028                         goto fail5;
2029                 valuesp[EFX_PHY_BIST_CABLE_LENGTH_C] =
2030                         EFX_WORD_FIELD(word, EFX_WORD_0);
2031                 value_mask |= (1 << EFX_PHY_BIST_CABLE_LENGTH_C);
2032         }
2033         if (count > EFX_PHY_BIST_CABLE_LENGTH_D) {
2034                 if ((rc = falcon_mdio_read(enp, epp->ep_port, PMA_PMD_MMD,
2035                     PMA_PMD_DIAG_D_LENGTH_REG, &word)) != 0)
2036                         goto fail6;
2037                 valuesp[EFX_PHY_BIST_CABLE_LENGTH_D] =
2038                         EFX_WORD_FIELD(word, EFX_WORD_0);
2039                 value_mask |= (1 << EFX_PHY_BIST_CABLE_LENGTH_D);
2040         }
```



```
2042         if (value_maskp != NULL)
2043                 *value_maskp = value_mask;

2045         return (0);

2047 fail6:
2048         EFSYS_PROBE(fail6);
2049 fail5:
2050         EFSYS_PROBE(fail5);
2051 fail4:
2052         EFSYS_PROBE(fail4);
2053 fail3:
2054         EFSYS_PROBE(fail3);
2055 fail2:
2056         EFSYS_PROBE(fail2);
2057 fail1:
2058         EFSYS_PROBE1(fail1, int, rc);

2060         return (rc);
2061 }

2063                         void
2064 sft9001_bist_stop(
2065         __in            efx_nic_t *enp,
2066         __in            efx_phy_bist_type_t type)
2067 {
2068         boolean_t break_link = (type == EFX_PHY_BIST_TYPE_CABLE_LONG);

2070         if (break_link) {
2071                 /* Pull external reset and reconfigure the PHY */
2072                 falcon_nic_phy_reset(enp);

2074                 EFSYS_ASSERT3U(enp->en_reset_flags, &, EFX_RESET_PHY);
2075                 enp->en_reset_flags &= ~EFX_RESET_PHY;

2077                 (void) sft9001_reconfigure(enp);
2078         }
2079 }

2081 #endif  /* EFSYS_OPT_PHY_BIST */

2083 #endif  /* EFSYS_OPT_PHY_SFT9001 */
2084 #endif /* ! codereview */
```

```
**********************************************************
    7214 Thu Aug 22 18:59:26 2013
new/usr/src/uts/common/io/sfxge/sft9001.h
Merged sfxge driver
**********************************************************
```

```
/*-
 * Copyright 2008-2013 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#ifndef _SYS_SFT9001_H
#define _SYS_SFT9001_H

#include "efx.h"

#ifdef  __cplusplus
extern "C" {
#endif

#if EFSYS_OPT_PHY_SFT9001

#define SFT9001_10G_LOOPBACK_MASK                                       \
            ((1 << EFX_LOOPBACK_PHY_XS) |                               \
            (1 << EFX_LOOPBACK_PCS) |                                   \
            (1 << EFX_LOOPBACK_PMA_PMD) |                               \
            FALCON_XMAC_LOOPBACK_MASK)

#define SFT9001_1G_LOOPBACK_MASK                                        \
            ((1 << EFX_LOOPBACK_GPHY) |                                 \
            FALCON_GMAC_LOOPBACK_MASK)

#define SFT9001_LED_MASK                                                \
            ((1 << EFX_PHY_LED_OFF) |                                   \
            (1 << EFX_PHY_LED_ON) |                                     \
            (1 << EFX_PHY_LED_FLASH))

/* START MKCONFIG GENERATED Sft9001PhyHeaderPropsBlock 9b100228a7cfe533 */
typedef enum sft9001_prop_e {
        SFT9001_SHORT_REACH,
        SFT9001_ROBUST,
        SFT9001_NPROPS
} sft9001_prop_t;

/* END MKCONFIG GENERATED Sft9001PhyHeaderPropsBlock */

#define SFT9001_ADV_CAP_MASK                                            \
```



```
            ((1 << EFX_PHY_CAP_AN) |                                    \
            (1 << EFX_PHY_CAP_10000FDX) |                               \
            (1 << EFX_PHY_CAP_1000FDX) |                                \
            (1 << EFX_PHY_CAP_100FDX) |                                 \
            (1 << EFX_PHY_CAP_PAUSE))

#define SFT9001_ADV_CAP_PERM                                            \
            ((1 << EFX_PHY_CAP_10000FDX) |                              \
            (1 << EFX_PHY_CAP_1000FDX) |                                \
            (1 << EFX_PHY_CAP_100FDX) |                                 \
            (1 << EFX_PHY_CAP_100HDX) |                                 \
            (1 << EFX_PHY_CAP_PAUSE) |                                  \
            (1 << EFX_PHY_CAP_ASYM))

#define SFT9001_BIST_MASK                                               \
            ((1 << EFX_PHY_BIST_TYPE_CABLE_SHORT) |                     \
            (1 << EFX_PHY_BIST_TYPE_CABLE_LONG))

extern  __checkReturn   int
sft9001_reset(
        __in            efx_nic_t *enp);

extern  __checkReturn   int
sft9001_reconfigure(
        __in            efx_nic_t *enp);

extern  __checkReturn   int
sft9001_verify(
        __in            efx_nic_t *enp);

extern  __checkReturn   int
sft9001_uplink_check(
        __in            efx_nic_t *enp,
        __out           boolean_t *upp);

extern  __checkReturn   int
sft9001_downlink_check(
        __in            efx_nic_t *enp,
        __out           efx_link_mode_t *modep,
        __out           unsigned int *fcntlp,
        __out           uint32_t *lp_cap_maskp);

extern __checkReturn    int
sft9001_oui_get(
        __in            efx_nic_t *enp,
        __out           uint32_t *ouip);

#if EFSYS_OPT_PHY_STATS

/* START MKCONFIG GENERATED Sft9001PhyHeaderStatsMask 06818b95754126e3 */
#define SFT9001_STAT_MASK \
        (1ULL << EFX_PHY_STAT_OUI) | \
        (1ULL << EFX_PHY_STAT_PMA_PMD_LINK_UP) | \
        (1ULL << EFX_PHY_STAT_PMA_PMD_RX_FAULT) | \
        (1ULL << EFX_PHY_STAT_PMA_PMD_TX_FAULT) | \
        (1ULL << EFX_PHY_STAT_PMA_PMD_REV_A) | \
        (1ULL << EFX_PHY_STAT_PMA_PMD_REV_B) | \
        (1ULL << EFX_PHY_STAT_PMA_PMD_REV_C) | \
        (1ULL << EFX_PHY_STAT_PMA_PMD_REV_D) | \
        (1ULL << EFX_PHY_STAT_PCS_LINK_UP) | \
        (1ULL << EFX_PHY_STAT_PCS_RX_FAULT) | \
        (1ULL << EFX_PHY_STAT_PCS_TX_FAULT) | \
        (1ULL << EFX_PHY_STAT_PCS_BER) | \
        (1ULL << EFX_PHY_STAT_PCS_BLOCK_ERRORS) | \
        (1ULL << EFX_PHY_STAT_PHY_XS_LINK_UP) | \
        (1ULL << EFX_PHY_STAT_PHY_XS_RX_FAULT) | \
```

```
128         (1ULL << EFX_PHY_STAT_PHY_XS_TX_FAULT) | \
129         (1ULL << EFX_PHY_STAT_PHY_XS_ALIGN) | \
130         (1ULL << EFX_PHY_STAT_PHY_XS_SYNC_A) | \
131         (1ULL << EFX_PHY_STAT_PHY_XS_SYNC_B) | \
132         (1ULL << EFX_PHY_STAT_PHY_XS_SYNC_C) | \
133         (1ULL << EFX_PHY_STAT_PHY_XS_SYNC_D) | \
134         (1ULL << EFX_PHY_STAT_AN_LINK_UP) | \
135         (1ULL << EFX_PHY_STAT_AN_MASTER) | \
136         (1ULL << EFX_PHY_STAT_AN_LOCAL_RX_OK) | \
137         (1ULL << EFX_PHY_STAT_AN_REMOTE_RX_OK) | \
138         (1ULL << EFX_PHY_STAT_CL22EXT_LINK_UP) | \
139         (1ULL << EFX_PHY_STAT_SNR_A) | \
140         (1ULL << EFX_PHY_STAT_SNR_B) | \
141         (1ULL << EFX_PHY_STAT_SNR_C) | \
142         (1ULL << EFX_PHY_STAT_SNR_D)

144 /* END MKCONFIG GENERATED Sft9001PhyHeaderStatsMask */

146 extern  __checkReturn                   int
147 sft9001_stats_update(
148         __in                            efx_nic_t *enp,
149         __in                            efsys_mem_t *esmp,
150         __out_ecount(EFX_PHY_NSTATS)    uint32_t *stat);

152 #endif  /* EFSYS_OPT_PHY_STATS */

154 #if EFSYS_OPT_PHY_PROPS

156 #if EFSYS_OPT_NAMES

158 extern          const char __cs *
159 sft9001_prop_name(
160         __in    efx_nic_t *enp,
161         __in    unsigned int id);

163 #endif  /* EFSYS_OPT_NAMES */

165 extern  __checkReturn   int
166 sft9001_prop_get(
167         __in            efx_nic_t *enp,
168         __in            unsigned int id,
169         __in            uint32_t flags,
170         __out           uint32_t *valp);

172 extern  __checkReturn   int
173 sft9001_prop_set(
174         __in            efx_nic_t *enp,
175         __in            unsigned int id,
176         __in            uint32_t val);

178 #endif  /* EFSYS_OPT_PHY_PROPS */

180 #if EFSYS_OPT_NVRAM_SFT9001

182 extern  __checkReturn           int
183 sft9001_nvram_size(
184         __in                    efx_nic_t *enp,
185         __out                   size_t *sizep);

187 extern  __checkReturn           int
188 sft9001_nvram_get_version(
189         __in                    efx_nic_t *enp,
190         __out                   uint32_t *subtypep,
191         __out_ecount(4)         uint16_t version[4]);

193 extern  __checkReturn           int
```



```
194 sft9001_nvram_rw_start(
195         __in                    efx_nic_t *enp,
196         __out                   size_t *block_sizep);

198 extern  __checkReturn           int
199 sft9001_nvram_read_chunk(
200         __in                    efx_nic_t *enp,
201         __in                    unsigned int offset,
202         __out_bcount(size)      caddr_t data,
203         __in                    size_t size);

205 extern  __checkReturn           int
206 sft9001_nvram_erase(
207         __in                    efx_nic_t *enp);

209 extern  __checkReturn           int
210 sft9001_nvram_write_chunk(
211         __in                    efx_nic_t *enp,
212         __in                    unsigned int offset,
213         __in_bcount(size)       caddr_t data,
214         __in                    size_t size);

216 extern                          void
217 sft9001_nvram_rw_finish(
218         __in                    efx_nic_t *enp);

220 extern const uint8_t * const sft9001_loader;
221 extern const size_t sft9001_loader_size;

223 #pragma pack(1)

225 typedef struct sft9001_firmware_header_s {
226         efx_dword_t code_length;
227         efx_dword_t destination_address;
228         efx_word_t code_checksum;
229         efx_word_t boot_config;
230         efx_word_t header_csum;
231         efx_byte_t reserved[18];
232 } sft9001_firmware_header_t;

234 #pragma pack()

236 #endif  /* EFSYS_OPT_NVRAM_SFT9001 */

238 #if EFSYS_OPT_PHY_BIST

240 extern  __checkReturn           int
241 sft9001_bist_start(
242         __in                    efx_nic_t *enp,
243         __in                    efx_phy_bist_type_t type);

245 extern  __checkReturn           int
246 sft9001_bist_poll(
247         __in                    efx_nic_t *enp,
248         __in                    efx_phy_bist_type_t type,
249         __out                   efx_phy_bist_result_t *resultp,
250         __out_opt               uint32_t *value_maskp,
251         __out_ecount_opt(count) unsigned long *valuesp,
252         __in                    size_t count);

254 extern                          void
255 sft9001_bist_stop(
256         __in                    efx_nic_t *enp,
257         __in                    efx_phy_bist_type_t type);

259 #endif  /* EFSYS_OPT_PHY_BIST */
```

```
261 #endif  /* EFSYS_OPT_SFT9001 */

263 #ifdef  __cplusplus
264 }
265 #endif

267 #endif  /* _SYS_SFT9001_H */
268 #endif /* ! codereview */
```

```
**********************************************************
    6959 Thu Aug 22 18:59:26 2013
new/usr/src/uts/common/io/sfxge/sft9001_impl.h
Merged sfxge driver
**********************************************************
```

```
/*-
 * Copyright 2008-2013 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#ifndef _SYS_SFT9001_IMPL_H
#define _SYS_SFT9001_IMPL_H

#ifdef __cplusplus
extern "C" {
#endif

#if EFSYS_OPT_PHY_SFT9001

#define SFT9001_MMD_MASK                                        \
            ((1 << PMA_PMD_MMD) |                               \
            (1 << PCS_MMD) |                                    \
            (1 << PHY_XS_MMD) |                                 \
            (1 << AN_MMD) |                                     \
            (1 << CL22EXT_MMD))

#define PMA_PMD_PWR_BACKOFF_REG 0x83
#define LP_TX_PWR_BACKOFF_LBN 13
#define LP_TX_PWR_BACKOFF_WIDTH 3
#define TX_PWR_BACKOFF_LBN 10
#define TX_PWR_BACKOFF_WIDTH 3
#define SHORT_REACH_LBN 0
#define SHORT_REACH_WIDTH 1

#define PMA_PMD_XCONTROL_REG 0xc000
#define SSR_LBN 15
#define SSR_WIDTH 1
#define ROBUST_LBN 14
#define ROBUST_WIDTH 1
#define CLK312_OUT_SEL_LBN 8
#define CLK312_OUT_SEL_WIDTH 1
#define SEL_312MHZ_DECODE 1
#define SEL_156MHZ_DECODE 0
#define TEST_CLKOUT_SEL_LBN 4
#define TEST_CLKOUT_SEL_WIDTH 4
#define SEL_125MHZ_DECODE 5
```



```
#define TEST_CLKOUT_EN_LBN 3
#define TEST_CLKOUT_EN_WIDTH 1
#define CLK312_OUT_EN_LBN 2
#define CLK312_OUT_EN_WIDTH 1
#define GMII_EN_LBN 1
#define GMII_EN_WIDTH 1

#define PMA_PMD_LED_CONTROL_REG 0xc007
#define LED_ACTIVITY_EN_LBN 3
#define LED_ACTIVITY_EN_WIDTH 1
#define LED_FLASH_PERIOD_LBN 1
#define LED_FLASH_PERIOD_WIDTH 2

#define PMA_PMD_LED_OVERRIDE_REG 0xc009
#define LED_TMODE_LBN 14
#define LED_TMODE_WIDTH 2
#define LED_SPARE_LBN 12
#define LED_SPARE_WIDTH 2
#define LED_MS_LBN 10
#define LED_MS_WIDTH 2
#define LED_RX_LBN 8
#define LED_RX_WIDTH 2
#define LED_TX_LBN 6
#define LED_TX_WIDTH 2
#define LED_SPEED1_LBN 4
#define LED_SPEED1_WIDTH 2
#define LED_SPEED0_LBN 2
#define LED_SPEED0_WIDTH 2
#define LED_LINK_LBN 0
#define LED_LINK_WIDTH 2
#define LED_NORMAL_DECODE 0
#define LED_ON_DECODE 1
#define LED_OFF_DECODE 2
#define LED_FLASH_DECODE 3

#define PMA_PMD_DIAG_RESULT_REG 0xc016
#define PAIR_A_CODE_LBN 12
#define PAIR_A_CODE_WIDTH 4
#define PAIR_B_CODE_LBN 8
#define PAIR_B_CODE_WIDTH 4
#define PAIR_C_CODE_LBN 4
#define PAIR_C_CODE_WIDTH 4
#define PAIR_D_CODE_LBN 0
#define PAIR_D_CODE_WIDTH 4
#define PAIR_BUSY_DECODE 0x9
#define INTER_PAIR_SHORT_DECODE 0x4
#define INTRA_PAIR_SHORT_DECODE 0x3
#define PAIR_OPEN_DECODE 0x2
#define PAIR_OK_DECODE 0x1

#define PMA_PMD_DIAG_A_LENGTH_REG 0xc017
#define PMA_PMD_DIAG_B_LENGTH_REG 0xc018
#define PMA_PMD_DIAG_C_LENGTH_REG 0xc019
#define PMA_PMD_DIAG_D_LENGTH_REG 0xc01a

#define PMA_PMD_FW_REV0_REG 0xc026
#define PMA_PMD_FW_REV1_REG 0xc027

#define PMA_PMD_SPEED_REG 0xc028
#define SPEED_RES_LBN 4
#define SPEED_RES_WIDTH 4
#define SPEED_10BASE_T_DECODE 1
#define SPEED_100BASE_TX_DECODE 2
#define SPEED_1000BASE_T_DECODE 3
#define SPEED_10GBASE_T_DECODE 4
```

```
#define PMA_PMD_DIAG_CONTROL_REG 0xc03d
#define RUN_DIAG_IMMED_LBN 15
#define RUN_DIAG_IMMED_WIDTH 1
#define RUN_DIAG_AUTO_LBN 14
#define RUN_DIAG_AUTO_WIDTH 1
#define INTER_PAIR_CHECK_DIS_LBN 13
#define INTER_PAIR_CHECK_DIS_WIDTH 1
#define BREAK_LINK_LBN 12
#define BREAK_LINK_WIDTH 1
#define DIAG_RUNNING_LBN 11
#define DIAG_RUNNING_WIDTH 1
#define LENGTH_UNIT_LBN 10
#define LENGTH_UNIT_WIDTH 1
#define LENGTH_M_DECODE 1
#define LENGTH_CM_DECODE 0

#define PCS_BOOT_STATUS_REG 0xd000
#define RESET_CAUSE_LBN 8
#define RESET_CAUSE_WIDTH 2
#define HW_RESET_DECODE 0x0
#define SW_RESET_DECODE 0x1
#define WD_RESET_DECODE 0x2
#define SW_WD_RESET_DECODE 0x3
#define UPLOAD_PROGRESS_LBN 7
#define UPLOAD_PROGRESS_WIDTH 1
#define CODE_DOWNLOAD_LBN 6
#define CODE_DOWNLOAD_WIDTH 1
#define CKSUM_OK_LBN 5
#define CKSUM_OK_WIDTH 1
#define CODE_STARTED_LBN 4
#define CODE_STARTED_WIDTH 1
#define BOOT_STATUS_LBN 3
#define BOOT_STATUS_WIDTH 1
#define BOOT_PROGRESS_LBN 1
#define BOOT_PROGRESS_WIDTH 2
#define INIT_DECODE 0x0
#define MDIO_WAIT_DECODE 0x1
#define CKSUM_START_DECODE 0x2
#define APP_JMP_DECODE 0x3
#define FATAL_ERR_LBN 0
#define FATAL_ERR_WIDTH 1

#define PCS_LM_RAM_LS_ADDR_REG 0xd004
#define LM_RAM_LS_ADDR_LBN 0
#define LM_RAM_LS_ADDR_WIDTH 16

#define PCS_LM_RAM_MS_ADDR_REG 0xd005
#define LM_RAM_MS_ADDR_LBN 0
#define LM_RAM_MS_ADDR_WIDTH 3
#define BYTE_ACCESS_LBN 15
#define BYTE_ACCESS_WIDTH 1

#define PCS_LM_RAM_DATA_REG 0xd006
#define LM_RAM_DATA_LBN 0
#define LM_RAM_DATA_WIDTH 16

#define PHY_XS_TEST1_REG 0xc00a
#define PHY_XS_NE_LOOPBACK_LBN 8
#define PHY_XS_NE_LOOPBACK_WIDTH 1

#define CL22EXT_CONTROL_REG 0xc000
#define CL22EXT_NE_LOOPBACK_LBN 14
#define CL22EXT_NE_LOOPBACK_WIDTH 1

#define CL22EXT_STATUS_REG 0xc001
#define CL22EXT_LINK_UP_LBN 2
```



```
#define CL22EXT_LINK_UP_WIDTH 1
#define CL22EXT_JABBER_LBN 1
#define CL22EXT_JABBER_WIDTH 1

#define CL22EXT_MS_CONTROL_REG 0xc009
#define CL22EXT_1000BASE_T_ADV_LBN 8
#define CL22EXT_1000BASE_T_ADV_WIDTH 1
#define CL22EXT_1000BASE_T_FDX_ADV_LBN 9
#define CL22EXT_1000BASE_T_FDX_ADV_WIDTH 1

#define CL22EXT_MS_STATUS_REG 0xc00a
#define CL22EXT_1000BASE_T_LP_LBN 10
#define CL22EXT_1000BASE_T_LP_WIDTH 1
#define CL22EXT_1000BASE_T_FDX_LP_LBN 11
#define CL22EXT_1000BASE_T_FDX_LP_WIDTH 1

#define LOADER_MMD 1
#define LOADER_MAX_BUFF_SZ_REG 49192
#define LOADER_ACTUAL_BUFF_SZ_REG 49193
#define LOADER_CMD_RESPONSE_REG 49194
#define LOADER_CMD_ERASE_FLASH 0x0001
#define LOADER_CMD_FILL_BUFFER 0x0002
#define LOADER_CMD_PROGRAM_FLASH 0x0003
#define LOADER_CMD_READ_FLASH 0x0004
#define LOADER_RESPONSE_OK 0x0100
#define LOADER_RESPONSE_ERROR 0x0200
#define LOADER_RESPONSE_BUSY 0x0300
#define LOADER_WORDS_WRITTEN_REG 49195
#define LOADER_WORDS_READ_REG 49195
#define LOADER_FLASH_ADDR_LOW_REG 49196
#define LOADER_FLASH_ADDR_HI_REG  49197
#define LOADER_DATA_REG 49198

#define FIRMWARE_BLOCK_SIZE 0x4000
#define FIRMWARE_MAX_SIZE 0x30000

#endif  /* EFSYS_OPT_PHY_SFT9001 */

#ifdef  __cplusplus
}
#endif

#endif  /* _SYS_SFT9001_IMPL_H */
#endif /* ! codereview */
```

```
**********************************************************
   38170 Thu Aug 22 18:59:26 2013
new/usr/src/uts/common/io/sfxge/sfx7101.c
Merged sfxge driver
**********************************************************
```

```
/*
 * Copyright 2007-2013 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#include "efsys.h"
#include "efx.h"
#include "efx_types.h"
#include "efx_regs.h"
#include "efx_impl.h"
#include "falcon_nvram.h"
#include "sfx7101.h"
#include "sfx7101_impl.h"
#include "xphy.h"

#if EFSYS_OPT_PHY_SFX7101

static  __checkReturn   int
sfx7101_power_on(
        __in            efx_nic_t *enp,
        __in            boolean_t reflash)
{
        efx_byte_t byte;
        int rc;

        if ((rc = falcon_i2c_check(enp, PCA9539)) != 0)
                goto fail1;

        /* Enable all port 0 outputs on IO expander */
        EFX_ZERO_BYTE(byte);

        if ((rc = falcon_i2c_write(enp, PCA9539, P0_CONFIG, (caddr_t)&byte,
            1)) != 0)
                goto fail2;

        /* Enable necessary port 1 outputs on IO expander */
        EFX_SET_BYTE(byte);
        EFX_SET_BYTE_FIELD(byte, P1_SPARE, 0);

        if ((rc = falcon_i2c_write(enp, PCA9539, P1_CONFIG, (caddr_t)&byte,
            1)) != 0)
```



```
                goto fail3;

        /* Turn off all power rails */
        EFX_SET_BYTE(byte);
        if ((rc = falcon_i2c_write(enp, PCA9539, P0_OUT, (caddr_t)&byte,
            1)) != 0)
                goto fail4;

        /* Sleep for 1 s */
        EFSYS_SLEEP(1000000);

        /* Turn on 1.2V, 2.5V, and 5V power rails */
        EFX_SET_BYTE(byte);
        EFX_SET_BYTE_FIELD(byte, P0_EN_1V2, 0);
        EFX_SET_BYTE_FIELD(byte, P0_EN_2V5, 0);
        EFX_SET_BYTE_FIELD(byte, P0_EN_5V, 0);
        EFX_SET_BYTE_FIELD(byte, P0_X_TRST, 0);

        /* Disable flash configuration */
        if (!reflash)
                EFX_SET_BYTE_FIELD(byte, P0_FLASH_CFG_EN, 0);

        /* Turn off JTAG */
        EFX_SET_BYTE_FIELD(byte, P0_SHORTEN_JTAG, 0);

        if ((rc = falcon_i2c_write(enp, PCA9539, P0_OUT, (caddr_t)&byte,
            1)) != 0)
                goto fail5;

        /* Spin for 10 ms */
        EFSYS_SPIN(10000);

        /* Turn on 1V power rail */
        EFX_SET_BYTE_FIELD(byte, P0_EN_1V0X, 0);

        if ((rc = falcon_i2c_write(enp, PCA9539, P0_OUT, (caddr_t)&byte,
            1)) != 0)
                goto fail6;

        /* Sleep for 1 s */
        EFSYS_SLEEP(1000000);

        enp->en_reset_flags |= EFX_RESET_PHY;

        return (0);

fail6:
        EFSYS_PROBE(fail6);
fail5:
        EFSYS_PROBE(fail5);
fail4:
        EFSYS_PROBE(fail4);

        /* Turn off all power rails */
        EFX_SET_BYTE(byte);

        (void) falcon_i2c_write(enp, PCA9539, P0_OUT, (caddr_t)&byte, 1);

fail3:
        EFSYS_PROBE(fail3);

        /* Disable port 1 outputs on IO expander */
        EFX_SET_BYTE(byte);
        (void) falcon_i2c_write(enp, PCA9539, P1_CONFIG, (caddr_t)&byte, 1);

fail2:
```

```
128         EFSYS_PROBE(fail2);

130         /* Disable port 0 outputs on IO expander */
131         EFX_SET_BYTE(byte);
132         (void) falcon_i2c_write(enp, PCA9539, P0_CONFIG, (caddr_t)&byte, 1);

134 fail1:
135         EFSYS_PROBE1(fail1, int, rc);

137         return (rc);
138 }

140 static  __checkReturn   int
141 sfx7101_power_off(
142         __in            efx_nic_t *enp)
143 {
144         efx_port_t *epp = &(enp->en_port);
145         efx_word_t word;
146         efx_byte_t byte;
147         int rc;

149         /* Power down the LNPGA */
150         EFX_POPULATE_WORD_1(word, LNPGA_POWERDOWN, 1);
151         if ((rc = falcon_mdio_write(enp, epp->ep_port, PMA_PMD_MMD,
152             PMA_PMD_XCONTROL_REG, &word)) != 0)
153                 goto fail1;

155         /* Sleep for 200 ms */
156         EFSYS_SLEEP(200000);

158         /* Turn off all power rails */
159         EFX_SET_BYTE(byte);
160         if ((rc = falcon_i2c_write(enp, PCA9539, P0_OUT, (caddr_t)&byte,
161             1)) != 0)
162                 goto fail2;

164         /* Disable port 1 outputs on IO expander */
165         EFX_SET_BYTE(byte);
166         if ((rc = falcon_i2c_write(enp, PCA9539, P1_CONFIG, (caddr_t)&byte,
167             1)) != 0)
168                 goto fail3;

170         /* Disable port 0 outputs on IO expander */
171         EFX_SET_BYTE(byte);
172         if ((rc = falcon_i2c_write(enp, PCA9539, P0_CONFIG, (caddr_t)&byte,
173             1)) != 0)
174                 goto fail4;

176         return (0);

178 fail4:
179         EFSYS_PROBE(fail4);
180 fail3:
181         EFSYS_PROBE(fail3);
182 fail2:
183         EFSYS_PROBE(fail2);
184 fail1:
185         EFSYS_PROBE1(fail1, int, rc);

187         return (rc);
188 }

190         __checkReturn   int
191 sfx7101_power(
192         __in            efx_nic_t *enp,
193         __in            boolean_t on)
```



```
194 {
195         int rc;

197         if (on) {
198                 if ((rc = sfx7101_power_on(enp, B_FALSE)) != 0)
199                         goto fail1;

201         } else {
202                 if ((rc = sfx7101_power_off(enp)) != 0)
203                         goto fail1;
204         }

206         return (0);

208 fail1:
209         EFSYS_PROBE1(fail1, int, rc);

211         return (rc);
212 }

214         __checkReturn   int
215 sfx7101_reset(
216         __in            efx_nic_t *enp)
217 {
218         efx_port_t *epp = &(enp->en_port);
219         efx_word_t word;
220         int rc;

222         if ((rc = falcon_mdio_read(enp, epp->ep_port, PMA_PMD_MMD,
223             PMA_PMD_XCONTROL_REG, &word)) != 0)
224                 goto fail1;

226         EFX_SET_WORD_FIELD(word, SSR, 1);

228         if ((rc = falcon_mdio_write(enp, epp->ep_port, PMA_PMD_MMD,
229             PMA_PMD_XCONTROL_REG, &word)) != 0)
230                 goto fail2;

232         /* Sleep 500ms */
233         EFSYS_SPIN(500000);

235         enp->en_reset_flags |= EFX_RESET_PHY;

237         return (0);

239 fail2:
240         EFSYS_PROBE(fail2);
241 fail1:
242         EFSYS_PROBE1(fail1, int, rc);

244         return (rc);
245 }

247 static  __checkReturn   int
248 sfx7101_an_set(
249         __in            efx_nic_t *enp,
250         __in            boolean_t on)
251 {
252         efx_port_t *epp = &(enp->en_port);
253         efx_word_t word;
254         int rc;

256         if ((rc = falcon_mdio_read(enp, epp->ep_port, AN_MMD,
257             AN_CONTROL1_REG, &word)) != 0)
258                 goto fail1;
```

```
260         if (on) {
261                 EFX_SET_WORD_FIELD(word, AN_ENABLE, 1);
262                 EFX_SET_WORD_FIELD(word, AN_RESTART, 1);
263         } else {
264                 EFX_SET_WORD_FIELD(word, AN_ENABLE, 0);
265         }

267         if ((rc = falcon_mdio_write(enp, epp->ep_port, AN_MMD,
268             AN_CONTROL1_REG, &word)) != 0)
269                 goto fail2;

271         return (0);

273 fail2:
274         EFSYS_PROBE(fail2);
275 fail1:
276         EFSYS_PROBE1(fail1, int, rc);

278         return (rc);
279 }

281 static  __checkReturn   int
282 sfx7101_led_cfg(
283         __in            efx_nic_t *enp)
284 {
285         efx_port_t *epp = &(enp->en_port);
286         efx_word_t word;
287         int rc;

289         if ((rc = falcon_mdio_read(enp, epp->ep_port, PMA_PMD_MMD,
290             PMA_PMD_LED_CONTROL_REG, &word)) != 0)
291                 goto fail1;

293         EFX_SET_WORD_FIELD(word, LED_ACTIVITY_EN, 1);

295         if ((rc = falcon_mdio_write(enp, epp->ep_port, PMA_PMD_MMD,
296             PMA_PMD_LED_CONTROL_REG, &word)) != 0)
297                 goto fail2;

299 #if EFSYS_OPT_PHY_LED_CONTROL

301         switch (epp->ep_phy_led_mode) {
302         case EFX_PHY_LED_DEFAULT:
303                 EFX_POPULATE_WORD_3(word,
304                     LED_LINK, LED_NORMAL_DECODE,
305                     LED_TX, LED_NORMAL_DECODE,
306                     LED_RX, LED_OFF_DECODE);
307                 break;

309         case EFX_PHY_LED_OFF:
310                 EFX_POPULATE_WORD_3(word,
311                     LED_LINK, LED_OFF_DECODE,
312                     LED_TX, LED_OFF_DECODE,
313                     LED_RX, LED_OFF_DECODE);
314                 break;

316         case EFX_PHY_LED_ON:
317                 EFX_POPULATE_WORD_3(word,
318                     LED_LINK, LED_ON_DECODE,
319                     LED_TX, LED_ON_DECODE,
320                     LED_RX, LED_ON_DECODE);
321                 break;

323         case EFX_PHY_LED_FLASH:
324                 EFX_POPULATE_WORD_3(word,
325                     LED_LINK, LED_FLASH_DECODE,
```



```
326                     LED_TX, LED_FLASH_DECODE,
327                     LED_RX, LED_FLASH_DECODE);
328                 break;

330         default:
331                 EFSYS_ASSERT(B_FALSE);
332                 break;
333         }

335 #else   /* EFSYS_OPT_PHY_LED_CONTROL */

337         EFX_POPULATE_WORD_3(word,
338             LED_LINK, LED_NORMAL_DECODE,
339             LED_TX, LED_NORMAL_DECODE,
340             LED_RX, LED_OFF_DECODE);

342 #endif  /* EFSYS_OPT_PHY_LED_CONTROL */

344         if ((rc = falcon_mdio_write(enp, epp->ep_port, PMA_PMD_MMD,
345             PMA_PMD_LED_OVERRIDE_REG, &word)) != 0)
346                 goto fail3;

348         return (0);

350 fail3:
351         EFSYS_PROBE(fail3);
352 fail2:
353         EFSYS_PROBE(fail2);
354 fail1:
355         EFSYS_PROBE1(fail1, int, rc);

357         return (rc);
358 }

360 static  __checkReturn   int
361 sfx7101_clock_cfg(
362         __in            efx_nic_t *enp,
363         __in            boolean_t enable)
364 {
365         efx_port_t *epp = &(enp->en_port);
366         efx_word_t word;
367         int rc;

369         EFX_POPULATE_WORD_1(word, CLK312_EN, (enable) ? 1 : 0);

371         if ((rc = falcon_mdio_write(enp, epp->ep_port, PCS_MMD,
372             PCS_TEST_SELECT_REG, &word)) != 0)
373                 goto fail1;

375         return (0);

377 fail1:
378         EFSYS_PROBE1(fail1, int, rc);

380         return (rc);
381 }

383 static  __checkReturn   int
384 sfx7101_adv_cap_cfg(
385         __in            efx_nic_t *enp)
386 {
387         efx_port_t *epp = &(enp->en_port);
388         efx_word_t word;
389         int rc;

391         /* Check base page */
```

```
392         if ((rc = falcon_mdio_read(enp, epp->ep_port, AN_MMD,
393             AN_ADV_BP_CAP_REG, &word)) != 0)
394                 goto fail1;

396         EFX_SET_WORD_FIELD(word, AN_ADV_TA_PAUSE,
397             ((epp->ep_adv_cap_mask >> EFX_PHY_CAP_PAUSE) & 0x1));
398         EFX_SET_WORD_FIELD(word, AN_ADV_TA_ASM_DIR,
399             ((epp->ep_adv_cap_mask >> EFX_PHY_CAP_ASYM) & 0x1));

401         if ((rc = falcon_mdio_write(enp, epp->ep_port, AN_MMD,
402             AN_ADV_BP_CAP_REG, &word)) != 0)
403                 goto fail2;

405         return (0);

407 fail2:
408         EFSYS_PROBE(fail2);
409 fail1:
410         EFSYS_PROBE1(fail1, int, rc);

412         return (rc);
413 }

415 #if EFSYS_OPT_LOOPBACK
416 static  __checkReturn   int
417 sfx7101_loopback_cfg(
418         __in            efx_nic_t *enp)
419 {
420         efx_port_t *epp = &(enp->en_port);
421         efx_word_t word;
422         int rc;

424         switch (epp->ep_loopback_type) {
425         case EFX_LOOPBACK_PHY_XS:
426                 if ((rc = falcon_mdio_read(enp, epp->ep_port, PHY_XS_MMD,
427                     PHY_XS_XGXS_TEST_REG, &word)) != 0)
428                         goto fail1;

430                 EFX_SET_WORD_FIELD(word, NE_LOOPBACK, 1);

432                 if ((rc = falcon_mdio_write(enp, epp->ep_port, PHY_XS_MMD,
433                     PHY_XS_XGXS_TEST_REG, &word)) != 0)
434                         goto fail2;

436                 break;

438         case EFX_LOOPBACK_PCS:
439                 if ((rc = xphy_mmd_loopback_set(enp, epp->ep_port, PCS_MMD,
440                     B_TRUE)) != 0)
441                         goto fail1;

443                 break;

445         case EFX_LOOPBACK_PMA_PMD:
446                 if ((rc = xphy_mmd_loopback_set(enp, epp->ep_port, PMA_PMD_MMD,
447                     B_TRUE)) != 0)
448                         goto fail1;

450                 break;

452         default:
453                 break;
454         }

456         return (0);
```



```
458 fail2:
459         EFSYS_PROBE(fail2);
460 fail1:
461         EFSYS_PROBE1(fail1, int, rc);

463         return (rc);
464 }
465 #endif  /* EFSYS_OPT_LOOPBACK */

467         __checkReturn   int
468 sfx7101_reconfigure(
469         __in            efx_nic_t *enp)
470 {
471         efx_port_t *epp = &(enp->en_port);
472         unsigned int count;
473         int rc;

475         /* Wait for the firmware boot to complete */
476         count = 0;
477         do {
478                 efx_word_t word;

480                 EFSYS_PROBE1(wait, unsigned int, count);

482                 /* Spin for 1 ms */
483                 EFSYS_SPIN(1000);

485                 if ((rc = falcon_mdio_read(enp, epp->ep_port, PCS_MMD,
486                     PCS_BOOT_STATUS_REG, &word)) != 0)
487                         goto fail1;

489                 if (EFX_WORD_FIELD(word, FATAL_ERR) != 0)
490                         break;  /* no point in continuing */

492                 if (EFX_WORD_FIELD(word, BOOT_STATUS) != 0 &&
493                     EFX_WORD_FIELD(word, CODE_DOWNLOAD) != 0 &&
494                     EFX_WORD_FIELD(word, CKSUM_OK) != 0 &&
495                     EFX_WORD_FIELD(word, CODE_STARTED) != 0 &&
496                     EFX_WORD_FIELD(word, BOOT_PROGRESS) == APP_JMP_DECODE)
497                         goto configure;

499         } while (++count < 1000);

501         rc = ENOTACTIVE;
502         goto fail2;

504 configure:
505         if ((rc = xphy_pkg_wait(enp, epp->ep_port, SFX7101_MMD_MASK)) != 0)
506                 goto fail3;

508         /* Make sure auto-negotiation is off whilst we configure the PHY */
509         if ((rc = sfx7101_an_set(enp, B_FALSE)) != 0)
510                 goto fail4;

512         if ((rc = sfx7101_clock_cfg(enp, B_TRUE)) != 0)
513                 goto fail5;

515 #if EFSYS_OPT_LOOPBACK
516         if ((rc = sfx7101_loopback_cfg(enp)) != 0)
517                 goto fail6;
518 #endif  /* EFSYS_OPT_LOOPBACK */

520         if ((rc = sfx7101_led_cfg(enp)) != 0)
521                 goto fail7;

523         if ((rc = sfx7101_adv_cap_cfg(enp)) != 0)
```

```
524                 goto fail8;

526 #if EFSYS_OPT_LOOPBACK
527         if (epp->ep_loopback_type == EFX_LOOPBACK_OFF) {
528                 if ((rc = sfx7101_an_set(enp, B_TRUE)) != 0)
529                         goto fail9;
530         }
531 #else   /* EFSYS_OPT_LOOPBACK */
532         if ((rc = sfx7101_an_set(enp, B_TRUE)) != 0)
533                 goto fail9;
534 #endif  /* EFSYS_OPT_LOOPBACK */

536         return (0);

538 fail9:
539         EFSYS_PROBE(fail9);
540 fail8:
541         EFSYS_PROBE(fail8);
542 fail7:
543         EFSYS_PROBE(fail7);

545 #if EFSYS_OPT_LOOPBACK
546 fail6:
547         EFSYS_PROBE(fail6);
548 #endif  /* EFSYS_OPT_LOOPBACK */

550 fail5:
551         EFSYS_PROBE(fail5);
552 fail4:
553         EFSYS_PROBE(fail4);
554 fail3:
555         EFSYS_PROBE(fail3);
556 fail2:
557         EFSYS_PROBE(fail2);
558 fail1:
559         EFSYS_PROBE1(fail1, int, rc);

561         return (rc);
562 }

564         __checkReturn   int
565 sfx7101_verify(
566         __in            efx_nic_t *enp)
567 {
568         efx_port_t *epp = &(enp->en_port);
569         int rc;

571         if ((rc = xphy_pkg_verify(enp, epp->ep_port, SFX7101_MMD_MASK)) != 0)
572                 goto fail1;

574         return (0);

576 fail1:
577         EFSYS_PROBE1(fail1, int, rc);

579         return (rc);
580 }

582         __checkReturn   int
583 sfx7101_uplink_check(
584         __in            efx_nic_t *enp,
585         __out           boolean_t *upp)
586 {
587         efx_port_t *epp = &(enp->en_port);
588         efx_word_t word;
589         int rc;
```



```
591         if (epp->ep_mac_type != EFX_MAC_FALCON_XMAC) {
592                 rc = ENOTSUP;
593                 goto fail1;
594         }

596         if ((rc = falcon_mdio_read(enp, epp->ep_port, PHY_XS_MMD,
597             PHY_XS_LANE_STATUS_REG, &word)) != 0)
598                 goto fail2;

600         *upp = ((EFX_WORD_FIELD(word, PHY_XS_ALIGNED) != 0) &&
601             (EFX_WORD_FIELD(word, PHY_XS_LANE0_SYNC) != 0) &&
602             (EFX_WORD_FIELD(word, PHY_XS_LANE1_SYNC) != 0) &&
603             (EFX_WORD_FIELD(word, PHY_XS_LANE2_SYNC) != 0) &&
604             (EFX_WORD_FIELD(word, PHY_XS_LANE3_SYNC) != 0));

606         return (0);

608 fail2:
609         EFSYS_PROBE(fail2);
610 fail1:
611         EFSYS_PROBE1(fail1, int, rc);

613         return (rc);
614 }

616                 void
617 sfx7101_uplink_reset(
618         __in    efx_nic_t *enp)
619 {
620         efx_port_t *epp = &(enp->en_port);
621         efx_word_t word;
622         int rc;

624         if (epp->ep_mac_type != EFX_MAC_FALCON_XMAC) {
625                 rc = ENOTSUP;
626                 goto fail1;
627         }

629         /* Disable the clock */
630         if ((rc = sfx7101_clock_cfg(enp, B_FALSE)) != 0)
631                 goto fail2;

633         /* Put the XGXS and SERDES into reset */
634         if ((rc = falcon_mdio_read(enp, epp->ep_port, PCS_MMD,
635             PCS_SOFT_RST2_REG, &word)) != 0)
636                 goto fail3;

638         EFX_SET_WORD_FIELD(word, XGXS_RST_N, 0);
639         EFX_SET_WORD_FIELD(word, SERDES_RST_N, 0);

641         if ((rc = falcon_mdio_write(enp, epp->ep_port, PCS_MMD,
642             PCS_SOFT_RST2_REG, &word)) != 0)
643                 goto fail4;

645         /* Put the PLL into reset */
646         if ((rc = falcon_mdio_read(enp, epp->ep_port, PCS_MMD,
647             PCS_CLOCK_CTRL_REG, &word)) != 0)
648                 goto fail5;

650         EFX_SET_WORD_FIELD(word, PLL312_RST_N, 0);

652         if ((rc = falcon_mdio_write(enp, epp->ep_port, PCS_MMD,
653             PCS_CLOCK_CTRL_REG, &word)) != 0)
654                 goto fail6;
```

```
656         EFSYS_SPIN(10);

658         /* Take the PLL out of reset */
659         EFX_SET_WORD_FIELD(word, PLL312_RST_N, 1);

661         if ((rc = falcon_mdio_write(enp, epp->ep_port, PCS_MMD,
662             PCS_CLOCK_CTRL_REG, &word)) != 0)
663                 goto fail7;

665         EFSYS_SPIN(10);

667         /* Take the XGXS and SERDES out of reset */
668         if ((rc = falcon_mdio_read(enp, epp->ep_port, PCS_MMD,
669             PCS_SOFT_RST2_REG, &word)) != 0)
670                 goto fail8;

672         EFX_SET_WORD_FIELD(word, XGXS_RST_N, 1);
673         EFX_SET_WORD_FIELD(word, SERDES_RST_N, 1);

675         if ((rc = falcon_mdio_write(enp, epp->ep_port, PCS_MMD,
676             PCS_SOFT_RST2_REG, &word)) != 0)
677                 goto fail9;

679         /* Enable the clock */
680         if ((rc = sfx7101_clock_cfg(enp, B_TRUE)) != 0)
681                 goto fail10;

683         /* Sleep 200 ms */
684         EFSYS_SLEEP(200000);

686         return;

688 fail10:
689         EFSYS_PROBE(fail10);
690 fail9:
691         EFSYS_PROBE(fail9);
692 fail8:
693         EFSYS_PROBE(fail8);
694 fail7:
695         EFSYS_PROBE(fail7);
696 fail6:
697         EFSYS_PROBE(fail6);
698 fail5:
699         EFSYS_PROBE(fail5);
700 fail4:
701         EFSYS_PROBE(fail4);
702 fail3:
703         EFSYS_PROBE(fail3);
704 fail2:
705         EFSYS_PROBE(fail2);
706 fail1:
707         EFSYS_PROBE1(fail1, int, rc);
708 }

710 static  __checkReturn   int
711 sfx7101_lp_cap_get(
712         __in            efx_nic_t *enp,
713         __out           unsigned int *maskp)
714 {
715         efx_port_t *epp = &(enp->en_port);
716         efx_word_t word;
717         int rc;

719         if ((rc = falcon_mdio_read(enp, epp->ep_port, AN_MMD,
720             AN_LP_BP_CAP_REG, &word)) != 0)
721                 goto fail1;
```



```
723         if (EFX_WORD_FIELD(word, AN_LP_TA_10BASE_T) != 0)
724                 *maskp |= (1 << EFX_PHY_CAP_10HDX);

726         if (EFX_WORD_FIELD(word, AN_LP_TA_10BASE_T_FDX) != 0)
727                 *maskp |= (1 << EFX_PHY_CAP_10FDX);

729         if (EFX_WORD_FIELD(word, AN_LP_TA_100BASE_TX) != 0)
730                 *maskp |= (1 << EFX_PHY_CAP_100HDX);

732         if (EFX_WORD_FIELD(word, AN_LP_TA_100BASE_TX_FDX) != 0)
733                 *maskp |= (1 << EFX_PHY_CAP_100FDX);

735         if (EFX_WORD_FIELD(word, AN_LP_TA_PAUSE) != 0)
736                 *maskp |= (1 << EFX_PHY_CAP_PAUSE);

738         if (EFX_WORD_FIELD(word, AN_LP_TA_ASM_DIR) != 0)
739                 *maskp |= (1 << EFX_PHY_CAP_ASYM);

741         if ((rc = falcon_mdio_read(enp, epp->ep_port, AN_MMD,
742             AN_10G_BASE_T_STATUS_REG, &word)) != 0)
743                 goto fail2;

745         if (EFX_WORD_FIELD(word, AN_10G_BASE_T_LP) != 0)
746                 *maskp |= (1 << EFX_PHY_CAP_10000FDX);

748         return (0);

750 fail2:
751         EFSYS_PROBE(fail2);
752 fail1:
753         EFSYS_PROBE1(fail1, int, rc);

755         return (rc);
756 }

758         __checkReturn   int
759 sfx7101_downlink_check(
760         __in            efx_nic_t *enp,
761         __out           efx_link_mode_t *modep,
762         __out           unsigned int *fcntlp,
763         __out           uint32_t *lp_cap_maskp)
764 {
765         efx_port_t *epp = &(enp->en_port);
766         unsigned int lp_cap_mask = epp->ep_lp_cap_mask;
767         unsigned int fcntl = epp->ep_fcntl;
768         uint32_t common;
769         boolean_t up;
770         int rc;

772 #if EFSYS_OPT_LOOPBACK
773         switch (epp->ep_loopback_type) {
774         case EFX_LOOPBACK_PHY_XS:
775                 rc = xphy_mmd_fault(enp, epp->ep_port, &up);
776                 if (rc != 0)
777                         goto fail1;

779                 *modep = (up) ? EFX_LINK_10000FDX : EFX_LINK_DOWN;
780                 goto done;

782         case EFX_LOOPBACK_PCS:
783         case EFX_LOOPBACK_PMA_PMD:
784                 rc = xphy_mmd_check(enp, epp->ep_port, PHY_XS_MMD, &up);
785                 if (rc != 0)
786                         goto fail1;
```

```
788                 *modep = (up) ? EFX_LINK_10000FDX : EFX_LINK_DOWN;
789                 goto done;

791         default:
792                 break;
793         }

795 #endif  /* EFSYS_OPT_LOOPBACK */
796         rc = xphy_mmd_check(enp, epp->ep_port, AN_MMD, &up);
797         if (rc != 0)
798                 goto fail1;

800         /* Check the link partner capabilities */
801         if ((rc = sfx7101_lp_cap_get(enp, &lp_cap_mask)) != 0)
802                 goto fail2;

804         /* Resolve the common capabilities */
805         common = epp->ep_adv_cap_mask & lp_cap_mask;

807         /* Determine negotiated or forced flow control mode */
808         fcntl = 0;
809         if (epp->ep_fcntl_autoneg) {
810                 if (common & (1 << EFX_PHY_CAP_PAUSE))
811                         fcntl = EFX_FCNTL_GENERATE | EFX_FCNTL_RESPOND;
812                 else if (common & (1 << EFX_PHY_CAP_ASYM)) {
813                         if (epp->ep_adv_cap_mask & (1 << EFX_PHY_CAP_PAUSE))
814                                 fcntl = EFX_FCNTL_RESPOND;
815                         else if (lp_cap_mask & (1 << EFX_PHY_CAP_PAUSE))
816                                 fcntl = EFX_FCNTL_GENERATE;
817                 }
818         } else {
819                 if (epp->ep_adv_cap_mask & (1 << EFX_PHY_CAP_PAUSE))
820                         fcntl = EFX_FCNTL_GENERATE | EFX_FCNTL_RESPOND;
821                 if (epp->ep_adv_cap_mask & (1 << EFX_PHY_CAP_ASYM))
822                         fcntl ^= EFX_FCNTL_GENERATE;
823         }

825         *fcntlp = fcntl;
826         *lp_cap_maskp = lp_cap_mask;
827         *modep = (up) ? EFX_LINK_10000FDX : EFX_LINK_DOWN;

829 #if EFSYS_OPT_LOOPBACK
830 done:
831 #endif
832         *fcntlp = epp->ep_fcntl;
833         *lp_cap_maskp = epp->ep_lp_cap_mask;

835         return (0);

837 fail2:
838         EFSYS_PROBE(fail2);
839 fail1:
840         EFSYS_PROBE1(fail1, int, rc);

842         return (rc);
843 }

845         __checkReturn   int
846 sfx7101_oui_get(
847         __in            efx_nic_t *enp,
848         __out           uint32_t *ouip)
849 {
850         efx_port_t *epp = &(enp->en_port);
851         int rc;

853         if ((rc = xphy_mmd_oui_get(enp, epp->ep_port, PMA_PMD_MMD, ouip)) != 0)
```



```
854                 goto fail1;

856         return (0);

858 fail1:
859         EFSYS_PROBE1(fail1, int, rc);

861         return (rc);
862 }

864 #if EFSYS_OPT_PHY_STATS

866 #define SFX7101_STAT_SET(_stat, _mask, _id, _val)                       \
867         do {                                                            \
868                 (_mask) |= (1ULL << (_id));                             \
869                 (_stat)[_id] = (uint32_t)(_val);                        \
870         _NOTE(CONSTANTCONDITION)                                        \
871         } while (B_FALSE)

873 static  __checkReturn   int
874 sfx7101_rev_get(
875         __in            efx_nic_t *enp,
876         __out           uint16_t *majorp,
877         __out           uint16_t *minorp,
878         __out           uint16_t *microp)
879 {
880         efx_port_t *epp = &(enp->en_port);
881         efx_word_t word[2];
882         int rc;

884         /*
885          * There appear to be two version string formats in use:
886          *
887          * - ('0','major') ('.','minor') with micro == 0
888          * - ('major','.') ('minor','micro')
889          *
890          */

892         if ((rc = falcon_mdio_read(enp, epp->ep_port, PMA_PMD_MMD,
893             PMA_PMD_FW_REV0_REG, &(word[0]))) != 0)
894                 goto fail1;

896         if ((rc = falcon_mdio_read(enp, epp->ep_port, PMA_PMD_MMD,
897             PMA_PMD_FW_REV1_REG, &(word[1]))) != 0)
898                 goto fail2;

900         if (EFX_WORD_FIELD(word[1], EFX_BYTE_0) == '0') {
901                 *majorp = EFX_WORD_FIELD(word[1], EFX_BYTE_1) - '0';

903                 if (EFX_WORD_FIELD(word[0], EFX_BYTE_0) != '.')
904                         goto fail3;

906                 *minorp = EFX_WORD_FIELD(word[0], EFX_BYTE_1) - '0';
907                 *microp = 0;

909         } else {
910                 *majorp = EFX_WORD_FIELD(word[1], EFX_BYTE_0) - '0';

912                 if (EFX_WORD_FIELD(word[1], EFX_BYTE_1) != '.')
913                         goto fail3;

915                 *minorp = EFX_WORD_FIELD(word[0], EFX_BYTE_0) - '0';
916                 *microp = EFX_WORD_FIELD(word[0], EFX_BYTE_1) - '0';
917         }

919         return (0);
```

```
fail3:
        rc = EIO;
        EFSYS_PROBE(fail3);
fail2:
        EFSYS_PROBE(fail2);
fail1:
        EFSYS_PROBE1(fail1, int, rc);

        return (rc);
}

static  __checkReturn                   int
sfx7101_pma_pmd_stats_update(
        __in                            efx_nic_t *enp,
        __inout                         uint64_t *maskp,
        __inout_ecount(EFX_PHY_NSTATS)  uint32_t *stat)
{
        efx_port_t *epp = &(enp->en_port);
        efx_word_t word;
        uint16_t major;
        uint16_t minor;
        uint16_t micro;
        int rc;

        if ((rc = falcon_mdio_read(enp, epp->ep_port, PMA_PMD_MMD,
            PMA_PMD_STATUS1_REG, &word)) != 0)
                goto fail1;

        SFX7101_STAT_SET(stat, *maskp, EFX_PHY_STAT_PMA_PMD_LINK_UP,
            (EFX_WORD_FIELD(word, PMA_PMD_LINK_UP) != 0) ? 1 : 0);

        if ((rc = falcon_mdio_read(enp, epp->ep_port, PMA_PMD_MMD,
            PMA_PMD_STATUS2_REG, &word)) != 0)
                goto fail2;

        SFX7101_STAT_SET(stat, *maskp, EFX_PHY_STAT_PMA_PMD_RX_FAULT,
            (EFX_WORD_FIELD(word, PMA_PMD_RX_FAULT) != 0) ? 1 : 0);
        SFX7101_STAT_SET(stat, *maskp, EFX_PHY_STAT_PMA_PMD_TX_FAULT,
            (EFX_WORD_FIELD(word, PMA_PMD_TX_FAULT) != 0) ? 1 : 0);

        if ((rc = sfx7101_rev_get(enp, &major, &minor, &micro)) != 0)
                goto fail3;

        SFX7101_STAT_SET(stat, *maskp, EFX_PHY_STAT_PMA_PMD_REV_MAJOR, major);
        SFX7101_STAT_SET(stat, *maskp, EFX_PHY_STAT_PMA_PMD_REV_MINOR, minor);
        SFX7101_STAT_SET(stat, *maskp, EFX_PHY_STAT_PMA_PMD_REV_MICRO, micro);

        if ((rc = falcon_mdio_read(enp, epp->ep_port, PMA_PMD_MMD,
            PMA_PMD_CHANNELA_SNR_REG, &word)) != 0)
                goto fail4;

        SFX7101_STAT_SET(stat, *maskp, EFX_PHY_STAT_SNR_A,
            EFX_WORD_FIELD(word, PMA_PMD_SNR));

        if ((rc = falcon_mdio_read(enp, epp->ep_port, PMA_PMD_MMD,
            PMA_PMD_CHANNELB_SNR_REG, &word)) != 0)
                goto fail5;

        SFX7101_STAT_SET(stat, *maskp, EFX_PHY_STAT_SNR_B,
            EFX_WORD_FIELD(word, PMA_PMD_SNR));

        if ((rc = falcon_mdio_read(enp, epp->ep_port, PMA_PMD_MMD,
            PMA_PMD_CHANNELC_SNR_REG, &word)) != 0)
                goto fail6;
```



```
        SFX7101_STAT_SET(stat, *maskp, EFX_PHY_STAT_SNR_C,
            EFX_WORD_FIELD(word, PMA_PMD_SNR));

        if ((rc = falcon_mdio_read(enp, epp->ep_port, PMA_PMD_MMD,
            PMA_PMD_CHANNELD_SNR_REG, &word)) != 0)
                goto fail7;

        SFX7101_STAT_SET(stat, *maskp, EFX_PHY_STAT_SNR_D,
            EFX_WORD_FIELD(word, PMA_PMD_SNR));

        return (0);

fail7:
        EFSYS_PROBE(fail7);
fail6:
        EFSYS_PROBE(fail6);
fail5:
        EFSYS_PROBE(fail5);
fail4:
        EFSYS_PROBE(fail4);
fail3:
        EFSYS_PROBE(fail3);
fail2:
        EFSYS_PROBE(fail2);
fail1:
        EFSYS_PROBE1(fail1, int, rc);

        return (rc);
}

static  __checkReturn                   int
sfx7101_pcs_stats_update(
        __in                            efx_nic_t *enp,
        __inout                         uint64_t *maskp,
        __inout_ecount(EFX_PHY_NSTATS)  uint32_t *stat)
{
        efx_port_t *epp = &(enp->en_port);
        efx_word_t word;
        int rc;

        if ((rc = falcon_mdio_read(enp, epp->ep_port, PCS_MMD,
            PCS_STATUS1_REG, &word)) != 0)
                goto fail1;

        SFX7101_STAT_SET(stat, *maskp, EFX_PHY_STAT_PCS_LINK_UP,
            (EFX_WORD_FIELD(word, PCS_LINK_UP) != 0) ? 1 : 0);

        if ((rc = falcon_mdio_read(enp, epp->ep_port, PCS_MMD,
            PCS_STATUS2_REG, &word)) != 0)
                goto fail2;

        SFX7101_STAT_SET(stat, *maskp, EFX_PHY_STAT_PCS_RX_FAULT,
            (EFX_WORD_FIELD(word, PCS_RX_FAULT) != 0) ? 1 : 0);
        SFX7101_STAT_SET(stat, *maskp, EFX_PHY_STAT_PCS_TX_FAULT,
            (EFX_WORD_FIELD(word, PCS_TX_FAULT) != 0) ? 1 : 0);

        if ((rc = falcon_mdio_read(enp, epp->ep_port, PCS_MMD,
            PCS_10GBASE_R_STATUS2_REG, &word)) != 0)
                goto fail3;

        SFX7101_STAT_SET(stat, *maskp, EFX_PHY_STAT_PCS_BER,
            EFX_WORD_FIELD(word, PCS_BER));
        SFX7101_STAT_SET(stat, *maskp, EFX_PHY_STAT_PCS_BLOCK_ERRORS,
            EFX_WORD_FIELD(word, PCS_ERR));

        return (0);
```

```
fail3:
        EFSYS_PROBE(fail3);
fail2:
        EFSYS_PROBE(fail2);
fail1:
        EFSYS_PROBE1(fail1, int, rc);

        return (rc);
}

static  __checkReturn                   int
sfx7101_phy_xs_stats_update(
        __in                            efx_nic_t *enp,
        __inout                         uint64_t *maskp,
        __inout_ecount(EFX_PHY_NSTATS)  uint32_t *stat)
{
        efx_port_t *epp = &(enp->en_port);
        efx_word_t word;
        int rc;

        if ((rc = falcon_mdio_read(enp, epp->ep_port, PHY_XS_MMD,
            PHY_XS_STATUS1_REG, &word)) != 0)
                goto fail1;

        SFX7101_STAT_SET(stat, *maskp, EFX_PHY_STAT_PHY_XS_LINK_UP,
            (EFX_WORD_FIELD(word, PHY_XS_LINK_UP) != 0) ? 1 : 0);

        if ((rc = falcon_mdio_read(enp, epp->ep_port, PHY_XS_MMD,
            PHY_XS_STATUS2_REG, &word)) != 0)
                goto fail2;

        SFX7101_STAT_SET(stat, *maskp, EFX_PHY_STAT_PHY_XS_RX_FAULT,
            (EFX_WORD_FIELD(word, PHY_XS_RX_FAULT) != 0) ? 1 : 0);
        SFX7101_STAT_SET(stat, *maskp, EFX_PHY_STAT_PHY_XS_TX_FAULT,
            (EFX_WORD_FIELD(word, PHY_XS_TX_FAULT) != 0) ? 1 : 0);

        if ((rc = falcon_mdio_read(enp, epp->ep_port, PHY_XS_MMD,
            PHY_XS_LANE_STATUS_REG, &word)) != 0)
                goto fail3;

        SFX7101_STAT_SET(stat, *maskp, EFX_PHY_STAT_PHY_XS_ALIGN,
            (EFX_WORD_FIELD(word, PHY_XS_ALIGNED) != 0) ? 1 : 0);
        SFX7101_STAT_SET(stat, *maskp, EFX_PHY_STAT_PHY_XS_SYNC_A,
            (EFX_WORD_FIELD(word, PHY_XS_LANE0_SYNC) != 0) ? 1 : 0);
        SFX7101_STAT_SET(stat, *maskp, EFX_PHY_STAT_PHY_XS_SYNC_B,
            (EFX_WORD_FIELD(word, PHY_XS_LANE1_SYNC) != 0) ? 1 : 0);
        SFX7101_STAT_SET(stat, *maskp, EFX_PHY_STAT_PHY_XS_SYNC_C,
            (EFX_WORD_FIELD(word, PHY_XS_LANE2_SYNC) != 0) ? 1 : 0);
        SFX7101_STAT_SET(stat, *maskp, EFX_PHY_STAT_PHY_XS_SYNC_D,
            (EFX_WORD_FIELD(word, PHY_XS_LANE3_SYNC) != 0) ? 1 : 0);

        return (0);

fail3:
        EFSYS_PROBE(fail3);
fail2:
        EFSYS_PROBE(fail2);
fail1:
        EFSYS_PROBE1(fail1, int, rc);

        return (rc);
}

static  __checkReturn                   int
sfx7101_an_stats_update(
```



```
        __in                            efx_nic_t *enp,
        __inout                         uint64_t *maskp,
        __inout_ecount(EFX_PHY_NSTATS)  uint32_t *stat)
{
        efx_port_t *epp = &(enp->en_port);
        efx_word_t word;
        int rc;

        if ((rc = falcon_mdio_read(enp, epp->ep_port, AN_MMD,
            AN_STATUS1_REG, &word)) != 0)
                goto fail1;

        SFX7101_STAT_SET(stat, *maskp, EFX_PHY_STAT_AN_LINK_UP,
            (EFX_WORD_FIELD(word, AN_LINK_UP) != 0) ? 1 : 0);

        if ((rc = falcon_mdio_read(enp, epp->ep_port, AN_MMD,
            AN_10G_BASE_T_STATUS_REG, &word)) != 0)
                goto fail2;

        SFX7101_STAT_SET(stat, *maskp, EFX_PHY_STAT_AN_MASTER,
            (EFX_WORD_FIELD(word, AN_MASTER) != 0) ? 1 : 0);
        SFX7101_STAT_SET(stat, *maskp, EFX_PHY_STAT_AN_LOCAL_RX_OK,
            (EFX_WORD_FIELD(word, AN_LOCAL_RX_OK) != 0) ? 1 : 0);
        SFX7101_STAT_SET(stat, *maskp, EFX_PHY_STAT_AN_REMOTE_RX_OK,
            (EFX_WORD_FIELD(word, AN_REMOTE_RX_OK) != 0) ? 1 : 0);

        return (0);

fail2:
        EFSYS_PROBE(fail2);
fail1:
        EFSYS_PROBE1(fail1, int, rc);

        return (rc);
}

        __checkReturn                   int
sfx7101_stats_update(
        __in                            efx_nic_t *enp,
        __in                            efsys_mem_t *esmp,
        __out_ecount(EFX_PHY_NSTATS)    uint32_t *stat)
{
        efx_port_t *epp = &(enp->en_port);
        efx_nic_cfg_t *encp = &(enp->en_nic_cfg);
        uint64_t mask = 0;
        uint32_t oui;
        int rc;

        _NOTE(ARGUNUSED(esmp))

        if ((rc = xphy_mmd_oui_get(enp, epp->ep_port, PMA_PMD_MMD, &oui)) != 0)
                goto fail1;

        SFX7101_STAT_SET(stat, mask, EFX_PHY_STAT_OUI, oui);

        if ((rc = sfx7101_pma_pmd_stats_update(enp, &mask, stat)) != 0)
                goto fail2;

        if ((rc = sfx7101_pcs_stats_update(enp, &mask, stat)) != 0)
                goto fail3;

        if ((rc = sfx7101_phy_xs_stats_update(enp, &mask, stat)) != 0)
                goto fail4;

        if ((rc = sfx7101_an_stats_update(enp, &mask, stat)) != 0)
                goto fail5;
```

```
1185         /* Ensure all the supported statistics are up to date */
1186         EFSYS_ASSERT(mask == encp->enc_phy_stat_mask);

1188         return (0);

1190 fail5:
1191         EFSYS_PROBE(fail5);
1192 fail4:
1193         EFSYS_PROBE(fail4);
1194 fail3:
1195         EFSYS_PROBE(fail3);
1196 fail2:
1197         EFSYS_PROBE(fail2);
1198 fail1:
1199         EFSYS_PROBE1(fail1, int, rc);

1201         return (rc);
1202 }
1203 #endif  /* EFSYS_OPT_PHY_STATS */

1205 #if EFSYS_OPT_PHY_PROPS

1207 #if EFSYS_OPT_NAMES
1208                 const char __cs *
1209 sfx7101_prop_name(
1210         __in    efx_nic_t *enp,
1211         __in    unsigned int id)
1212 {
1213         _NOTE(ARGUNUSED(enp, id))

1215         EFSYS_ASSERT(B_FALSE);

1217         return (NULL);
1218 }
1219 #endif  /* EFSYS_OPT_NAMES */

1221         __checkReturn   int
1222 sfx7101_prop_get(
1223         __in            efx_nic_t *enp,
1224         __in            unsigned int id,
1225         __in            uint32_t flags,
1226         __out           uint32_t *valp)
1227 {
1228         _NOTE(ARGUNUSED(enp, id, flags, valp))

1230         EFSYS_ASSERT(B_FALSE);

1232         return (ENOTSUP);
1233 }

1235         __checkReturn   int
1236 sfx7101_prop_set(
1237         __in            efx_nic_t *enp,
1238         __in            unsigned int id,
1239         __in            uint32_t val)
1240 {
1241         _NOTE(ARGUNUSED(enp, id, val))

1243         EFSYS_ASSERT(B_FALSE);

1245         return (ENOTSUP);
1246 }

1248 #endif  /* EFSYS_OPT_PHY_PROPS */
```



```
1250 #if EFSYS_OPT_NVRAM_SFX7101

1252 static  __checkReturn           int
1253 sfx7101_loader_wait(
1254         __in                    efx_nic_t *enp)
1255 {
1256         efx_port_t *epp = &(enp->en_port);
1257         efx_word_t word;
1258         unsigned int count;
1259         unsigned int response;
1260         int rc;

1262         /* Wait up to 20s for the command to complete */
1263         for (count = 0; count < 200; count++) {
1264                 EFSYS_SLEEP(100000);    /* 100ms */

1266                 if ((rc = falcon_mdio_read(enp, epp->ep_port, LOADER_MMD,
1267                     LOADER_CMD_RESPONSE_REG, &word)) != 0)
1268                         goto fail1;

1270                 response = EFX_WORD_FIELD(word, EFX_WORD_0);
1271                 if (response == LOADER_RESPONSE_OK)
1272                         return (0);
1273                 if (response != LOADER_RESPONSE_BUSY) {
1274                         rc = EIO;
1275                         goto fail2;
1276                 }
1277         }

1279         rc = ETIMEDOUT;

1281         EFSYS_PROBE(fail3);
1282 fail2:
1283         EFSYS_PROBE(fail2);
1284 fail1:
1285         EFSYS_PROBE1(fail1, int, rc);

1287         return (rc);
1288 }

1290 static  __checkReturn           int
1291 sfx7101_program_loader(
1292         __in                    efx_nic_t *enp,
1293         __in                    unsigned int offset,
1294         __in                    size_t words)
1295 {
1296         efx_port_t *epp = &(enp->en_port);
1297         efx_word_t word;
1298         int rc;

1300         /* Setup address of block transfer */
1301         EFX_POPULATE_WORD_1(word, EFX_WORD_0, offset);
1302         if ((rc = falcon_mdio_write(enp, epp->ep_port, LOADER_MMD,
1303             LOADER_FLASH_ADDR_LOW_REG, &word)) != 0)
1304                 goto fail1;

1306         EFX_POPULATE_WORD_1(word, EFX_WORD_0, (offset >> 16));
1307         if ((rc = falcon_mdio_write(enp, epp->ep_port, LOADER_MMD,
1308             LOADER_FLASH_ADDR_HI_REG, &word)) != 0)
1309                 goto fail2;

1311         EFX_POPULATE_WORD_1(word, EFX_WORD_0, words);
1312         if ((rc = falcon_mdio_write(enp, epp->ep_port, LOADER_MMD,
1313             LOADER_ACTUAL_BUFF_SZ_REG, &word)) != 0)
1314                 goto fail3;
```

```
1316         return (0);

1318 fail3:
1319         EFSYS_PROBE(fail3);
1320 fail2:
1321         EFSYS_PROBE(fail2);
1322 fail1:
1323         EFSYS_PROBE1(fail1, int, rc);

1325         return (rc);
1326 }

1328 __checkReturn           int
1329 sfx7101_nvram_size(
1330         __in                    efx_nic_t *enp,
1331         __out                   size_t *sizep)
1332 {
1333         _NOTE(ARGUNUSED(enp))
1334         EFSYS_ASSERT(sizep);

1336         *sizep = FIRMWARE_MAX_SIZE;

1338         return (0);
1339 }

1341         __checkReturn           int
1342 sfx7101_nvram_get_version(
1343         __in                    efx_nic_t *enp,
1344         __out                   uint32_t *subtypep,
1345         __out_ecount(4)         uint16_t version[4])
1346 {
1347         uint16_t major, minor, micro;
1348         int rc;

1350         if ((rc = sfx7101_rev_get(enp, &major, &minor, &micro)) != 0)
1351                 goto fail1;

1353         version[0] = major;
1354         version[1] = minor;
1355         version[2] = 0;
1356         version[3] = 0;

1358         *subtypep = PHY_TYPE_SFX7101_DECODE;

1360         return (0);

1362 fail1:
1363         EFSYS_PROBE1(fail1, int, rc);

1365         return (0);
1366 }

1368         __checkReturn           int
1369 sfx7101_nvram_rw_start(
1370         __in                    efx_nic_t *enp,
1371         __out                   size_t *block_sizep)
1372 {
1373         efx_port_t *epp = &(enp->en_port);
1374         sfx7101_firmware_header_t header;
1375         efx_word_t word;
1376         unsigned int pos;
1377         int rc;

1379         /* Ensure the PHY is on */
1380         if ((rc = sfx7101_power_on(enp, B_TRUE)) != 0)
1381                 goto fail1;
```



```
1383         /* Special software reset */
1384         if ((rc = falcon_mdio_read(enp, epp->ep_port, PMA_PMD_MMD,
1385                     PMA_PMD_XCONTROL_REG, &word)) != 0)
1386                 goto fail2;
1387         EFX_SET_WORD_FIELD(word, SSR, 1);
1388         if ((rc = falcon_mdio_write(enp, epp->ep_port, PMA_PMD_MMD,
1389                     PMA_PMD_XCONTROL_REG, &word)) != 0)
1390                 goto fail3;

1392         /* Sleep for 500ms */
1393         EFSYS_SLEEP(500000);

1395         /* Check that the C166 is idle, and in download mode */
1396         if ((rc = falcon_mdio_read(enp, epp->ep_port, PCS_MMD,
1397                     PCS_BOOT_STATUS_REG, &word)) != 0)
1398                 goto fail4;
1399         if (EFX_WORD_FIELD(word, BOOT_STATUS) == 0 ||
1400             EFX_WORD_FIELD(word, BOOT_PROGRESS) != MDIO_WAIT_DECODE) {
1401                 rc = ETIMEDOUT;
1402                 goto fail5;
1403         }

1405         /* Download loader code into PHY RAM */
1406         EFX_ZERO_WORD(word);
1407         if ((rc = falcon_mdio_write(enp, epp->ep_port, PCS_MMD,
1408                     PCS_LM_RAM_LS_ADDR_REG, &word)) != 0)
1409                 goto fail6;
1410         if ((rc = falcon_mdio_write(enp, epp->ep_port, PCS_MMD,
1411                     PCS_LM_RAM_MS_ADDR_REG, &word)) != 0)
1412                 goto fail7;

1414         for (pos = 0; pos < sfx7101_loader_size / sizeof (uint16_t); pos++) {
1415                 /* Firmware is little endian */
1416                 word.ew_u8[0] = sfx7101_loader[pos];
1417                 word.ew_u8[1] = sfx7101_loader[pos+1];
1418                 if ((rc = falcon_mdio_write(enp, epp->ep_port, PCS_MMD,
1419                             PCS_LM_RAM_DATA_REG, &word)) != 0)
1420                         goto fail8;
1421         }

1423         /* Sleep for 500ms */
1424         EFSYS_SLEEP(500000);

1426         /* Start downloaded code */
1427         if ((rc = falcon_mdio_read(enp, epp->ep_port, PCS_MMD,
1428                     PCS_BOOT_STATUS_REG, &word)) != 0)
1429                 goto fail9;
1430         EFX_SET_WORD_FIELD(word, CODE_DOWNLOAD, 1);
1431         if ((rc = falcon_mdio_write(enp, epp->ep_port, PCS_MMD,
1432                     PCS_BOOT_STATUS_REG, &word)) != 0)
1433                 goto fail10;

1435         /* Sleep 1s */
1436         EFSYS_SLEEP(1000000);

1438         /* And check it started */
1439         if ((rc = falcon_mdio_read(enp, epp->ep_port, PCS_MMD,
1440                     PCS_BOOT_STATUS_REG, &word)) != 0)
1441                 goto fail11;

1443         if (!EFX_WORD_FIELD(word, CODE_STARTED)) {
1444                 rc = ETIMEDOUT;
1445                 goto fail12;
1446         }
```

```
1448         /* Verify program block size is appropriate */
1449         if ((rc = falcon_mdio_read(enp, epp->ep_port, LOADER_MMD,
1450                   LOADER_MAX_BUFF_SZ_REG, &word)) != 0)
1451                 goto fail13;
1452         if (EFX_WORD_FIELD(word, EFX_WORD_0) < FIRMWARE_BLOCK_SIZE) {
1453                 rc = EIO;
1454                 goto fail14;
1455         }
1456         if (block_sizep != NULL)
1457                 *block_sizep = FIRMWARE_BLOCK_SIZE;

1459         /* Read firmware header */
1460         if ((rc = sfx7101_nvram_read_chunk(enp, 0, (void *)&header,
1461             sizeof (header))) != 0)
1462                 goto fail15;

1464         /* Verify firmware isn't too large */
1465         if (EFX_DWORD_FIELD(header.code_length, EFX_DWORD_0) +
1466             sizeof (sfx7101_firmware_header_t) > FIRMWARE_MAX_SIZE) {
1467                 rc = EIO;
1468                 goto fail16;
1469         }

1471         return (0);

1473 fail16:
1474         EFSYS_PROBE(fail16);
1475 fail15:
1476         EFSYS_PROBE(fail15);
1477 fail14:
1478         EFSYS_PROBE(fail14);
1479 fail13:
1480         EFSYS_PROBE(fail13);
1481 fail12:
1482         EFSYS_PROBE(fail12);
1483 fail11:
1484         EFSYS_PROBE(fail11);
1485 fail10:
1486         EFSYS_PROBE(fail10);
1487 fail9:
1488         EFSYS_PROBE(fail9);
1489 fail8:
1490         EFSYS_PROBE(fail8);
1491 fail7:
1492         EFSYS_PROBE(fail7);
1493 fail6:
1494         EFSYS_PROBE(fail6);
1495 fail5:
1496         EFSYS_PROBE(fail5);
1497 fail4:
1498         EFSYS_PROBE(fail4);
1499 fail3:
1500         EFSYS_PROBE(fail3);
1501 fail2:
1502         EFSYS_PROBE(fail2);
1503 fail1:
1504         EFSYS_PROBE1(fail1, int, rc);

1506         /* Restore the original image */
1507         sfx7101_nvram_rw_finish(enp);

1509         return (rc);
1510 }

1512         __checkReturn           int
1513 sfx7101_nvram_read_chunk(
```



```
1514         __in                    efx_nic_t *enp,
1515         __in                    unsigned int offset,
1516         __out_bcount(size)      caddr_t data,
1517         __in                    size_t size)
1518 {
1519         efx_port_t *epp = &(enp->en_port);
1520         efx_word_t word;
1521         unsigned int pos;
1522         size_t chunk;
1523         size_t words;
1524         int rc;

1526         while (size > 0) {
1527                 chunk = MIN(size, FIRMWARE_BLOCK_SIZE);

1529                 /* Read in 2byte words */
1530                 EFSYS_ASSERT(!(chunk & 0x1));
1531                 words = chunk >> 1;

1533                 /* Program address/length */
1534                 if ((rc = sfx7101_program_loader(enp, offset, words)) != 0)
1535                         goto fail1;

1537                 /* Select read mode, and wait for buffer to fill */
1538                 EFX_POPULATE_WORD_1(word, EFX_WORD_0, LOADER_CMD_READ_FLASH);
1539                 if ((rc = falcon_mdio_write(enp, epp->ep_port, LOADER_MMD,
1540                     LOADER_CMD_RESPONSE_REG, &word)) != 0)
1541                         goto fail2;

1543                 if ((rc = sfx7101_loader_wait(enp)) != 0)
1544                         goto fail3;

1546                 if ((rc = falcon_mdio_read(enp, epp->ep_port, LOADER_MMD,
1547                     LOADER_WORDS_READ_REG, &word)) != 0)
1548                         goto fail4;

1550                 if (words != (size_t)EFX_WORD_FIELD(word, EFX_WORD_0))
1551                         goto fail5;

1553                 for (pos = 0; pos < words; ++pos) {
1554                         if ((rc = falcon_mdio_read(enp, epp->ep_port,
1555                             LOADER_MMD, LOADER_DATA_REG, &word)) != 0)
1556                                 goto fail6;

1558                         /* Firmware is little endian */
1559                         data[pos] = word.ew_u8[0];
1560                         data[pos+1] = word.ew_u8[1];
1561                 }

1563                 offset += chunk;
1564                 data += chunk;
1565                 size -= chunk;
1566         }

1568         return (0);

1570 fail6:
1571         EFSYS_PROBE(fail6);
1572 fail5:
1573         EFSYS_PROBE(fail5);
1574 fail4:
1575         EFSYS_PROBE(fail4);
1576 fail3:
1577         EFSYS_PROBE(fail3);
1578 fail2:
1579         EFSYS_PROBE(fail2);
```

```
fail1:
        EFSYS_PROBE1(fail1, int, rc);

        return (rc);
}

        __checkReturn                           int
sfx7101_nvram_erase(
        __in                                    efx_nic_t *enp)
{
        efx_port_t *epp = &(enp->en_port);
        efx_word_t word;
        int rc;

        EFX_POPULATE_WORD_1(word, EFX_BYTE_0, LOADER_CMD_ERASE_FLASH);
        if ((rc = falcon_mdio_write(enp, epp->ep_port,
            LOADER_MMD, LOADER_CMD_RESPONSE_REG, &word)) != 0)
                goto fail1;

        return (0);

fail1:
        EFSYS_PROBE1(fail1, int, rc);

        return (rc);
}

        __checkReturn                           int
sfx7101_nvram_write_chunk(
        __in                                    efx_nic_t *enp,
        __in                                    unsigned int offset,
        __in_bcount(size)                       caddr_t data,
        __in                                    size_t size)
{
        efx_port_t *epp = &(enp->en_port);
        efx_word_t word;
        unsigned int pos;
        size_t chunk;
        size_t words;
        int rc;

        while (size > 0) {
                chunk = MIN(size, FIRMWARE_BLOCK_SIZE);

                /* Write in 2byte words */
                EFSYS_ASSERT(!(chunk & 0x1));
                words = chunk >> 1;

                /* Program address/length */
                if ((rc = sfx7101_program_loader(enp, offset, words)) != 0)
                        goto fail1;

                /* Select write mode */
                EFX_POPULATE_WORD_1(word, EFX_WORD_0, LOADER_CMD_FILL_BUFFER);
                if ((rc = falcon_mdio_write(enp, epp->ep_port, LOADER_MMD,
                    LOADER_CMD_RESPONSE_REG, &word)) != 0)
                        goto fail2;

                for (pos = 0; pos < words; ++pos) {
                        /* Firmware is little-endian */
                        word.ew_u8[0] = data[pos];
                        word.ew_u8[1] = data[pos+1];

                        if ((rc = falcon_mdio_write(enp, epp->ep_port,
                            LOADER_MMD, LOADER_DATA_REG, &word)) != 0)
                                goto fail3;
```



```
                }

                EFX_POPULATE_WORD_1(word, EFX_WORD_0,
                                    LOADER_CMD_PROGRAM_FLASH);
                if ((rc = falcon_mdio_write(enp, epp->ep_port,
                    LOADER_MMD, LOADER_CMD_RESPONSE_REG, &word)) != 0)
                        goto fail4;

                if ((rc = sfx7101_loader_wait(enp)) != 0)
                        goto fail5;

                if ((rc = falcon_mdio_read(enp, epp->ep_port, LOADER_MMD,
                    LOADER_WORDS_WRITTEN_REG, &word)) != 0)
                        goto fail6;

                if (words != EFX_WORD_FIELD(word, EFX_WORD_0))
                        goto fail7;

                size -= chunk;
                offset += chunk;
                data += chunk;
        }

        return (0);

fail7:
        EFSYS_PROBE(fail7);
fail6:
        EFSYS_PROBE(fail6);
fail5:
        EFSYS_PROBE(fail5);
fail4:
        EFSYS_PROBE(fail4);
fail3:
        EFSYS_PROBE(fail3);
fail2:
        EFSYS_PROBE(fail2);
fail1:
        EFSYS_PROBE1(fail1, int, rc);

        return (rc);
}


                                        void
sfx7101_nvram_rw_finish(
        __in                            efx_nic_t *enp)
{
        efx_port_t *epp = &(enp->en_port);
        efx_word_t word;
        efx_byte_t byte;
        int rc;

        EFX_SET_BYTE(byte);
        EFX_SET_BYTE_FIELD(byte, P0_EN_1V2, 0);
        EFX_SET_BYTE_FIELD(byte, P0_EN_2V5, 0);
        EFX_SET_BYTE_FIELD(byte, P0_EN_5V, 0);
        EFX_SET_BYTE_FIELD(byte, P0_X_TRST, 0);
        EFX_SET_BYTE_FIELD(byte, P0_FLASH_CFG_EN, 0);

        if ((rc = falcon_i2c_write(enp, PCA9539, P0_OUT,
                    (caddr_t)&byte, 1)) != 0)
                goto fail1;

        /* Special software reset */
        if ((rc = falcon_mdio_read(enp, epp->ep_port, PMA_PMD_MMD,
```

```
1712                            PMA_PMD_XCONTROL_REG, &word)) != 0)
1713                goto fail2;
1714        EFX_SET_WORD_FIELD(word, SSR, 1);
1715        if ((rc = falcon_mdio_write(enp, epp->ep_port, PMA_PMD_MMD,
1716                            PMA_PMD_XCONTROL_REG, &word)) != 0)
1717                goto fail3;

1719        /* Sleep 1/2 second */
1720        EFSYS_SLEEP(500000);

1722        /* Verify that PHY rebooted */
1723        if ((rc = falcon_mdio_read(enp, epp->ep_port, PCS_MMD,
1724                            PCS_BOOT_STATUS_REG, &word)) != 0)
1725                goto fail4;
1726        if (EFX_WORD_FIELD(word, EFX_WORD_0) != 0x7E)
1727                goto fail5;

1729        return;

1731 fail5:
1732        EFSYS_PROBE(fail4);
1733 fail4:
1734        EFSYS_PROBE(fail4);
1735 fail3:
1736        EFSYS_PROBE(fail3);
1737 fail2:
1738        EFSYS_PROBE(fail2);
1739 fail1:
1740        EFSYS_PROBE1(fail1, int, rc);
1741 }

1743 #endif  /* EFSYS_OPT_NVRAM_SFX7101 */

1745 #endif  /* EFSYS_OPT_PHY_SFX7101 */
1746 #endif /* ! codereview */
```

```
**********************************************************
    6067 Thu Aug 22 18:59:26 2013
new/usr/src/uts/common/io/sfxge/sfx7101.h
Merged sfxge driver
**********************************************************
```

```
/*-
 * Copyright 2007-2013 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#ifndef _SYS_SFX7101_H
#define _SYS_SFX7101_H

#include "efx.h"

#ifdef  __cplusplus
extern "C" {
#endif

#if EFSYS_OPT_PHY_SFX7101

#define SFX7101_LOOPBACK_MASK                                           \
            ((1 << EFX_LOOPBACK_PHY_XS) |                               \
            (1 << EFX_LOOPBACK_PCS) |                                   \
            (1 << EFX_LOOPBACK_PMA_PMD) |                               \
            FALCON_XMAC_LOOPBACK_MASK)

#define SFX7101_LED_MASK                                                \
            ((1 << EFX_PHY_LED_OFF) |                                   \
            (1 << EFX_PHY_LED_ON) |                                     \
            (1 << EFX_PHY_LED_FLASH))

#define SFX7101_NPROPS          0

#define SFX7101_ADV_CAP_MASK                                            \
            ((1 << EFX_PHY_CAP_AN) |                                    \
            (1 << EFX_PHY_CAP_10000FDX) |                               \
            (1 << EFX_PHY_CAP_ASYM) |                                   \
            (1 << EFX_PHY_CAP_PAUSE))

#define SFX7101_ADV_CAP_PERM    0

#define SFX7101_BIST_MASK       0

extern  __checkReturn   int
sfx7101_power(
```



```
        __in            efx_nic_t *enp,
        __in            boolean_t on);

extern  __checkReturn   int
sfx7101_reset(
        __in            efx_nic_t *enp);

extern  __checkReturn   int
sfx7101_reconfigure(
        __in            efx_nic_t *enp);

extern  __checkReturn   int
sfx7101_verify(
        __in            efx_nic_t *enp);

extern  __checkReturn   int
sfx7101_uplink_check(
        __in            efx_nic_t *enp,
        __out           boolean_t *upp);

extern          void
sfx7101_uplink_reset(
        __in    efx_nic_t *enp);

extern  __checkReturn   int
sfx7101_downlink_check(
        __in            efx_nic_t *enp,
        __out           efx_link_mode_t *modep,
        __out           unsigned int *fcntlp,
        __out           uint32_t *lp_cap_maskp);

extern  __checkReturn   int
sfx7101_oui_get(
        __in            efx_nic_t *enp,
        __out           uint32_t *ouip);

#if EFSYS_OPT_PHY_STATS

/* START MKCONFIG GENERATED Sfx7101PhyHeaderStatsMask edaf3cd6dfd8b815 */
#define SFX7101_STAT_MASK \
        (1ULL << EFX_PHY_STAT_OUI) | \
        (1ULL << EFX_PHY_STAT_PMA_PMD_LINK_UP) | \
        (1ULL << EFX_PHY_STAT_PMA_PMD_RX_FAULT) | \
        (1ULL << EFX_PHY_STAT_PMA_PMD_TX_FAULT) | \
        (1ULL << EFX_PHY_STAT_PMA_PMD_REV_MAJOR) | \
        (1ULL << EFX_PHY_STAT_PMA_PMD_REV_MINOR) | \
        (1ULL << EFX_PHY_STAT_PMA_PMD_REV_MICRO) | \
        (1ULL << EFX_PHY_STAT_PCS_LINK_UP) | \
        (1ULL << EFX_PHY_STAT_PCS_RX_FAULT) | \
        (1ULL << EFX_PHY_STAT_PCS_TX_FAULT) | \
        (1ULL << EFX_PHY_STAT_PCS_BER) | \
        (1ULL << EFX_PHY_STAT_PCS_BLOCK_ERRORS) | \
        (1ULL << EFX_PHY_STAT_PHY_XS_LINK_UP) | \
        (1ULL << EFX_PHY_STAT_PHY_XS_RX_FAULT) | \
        (1ULL << EFX_PHY_STAT_PHY_XS_TX_FAULT) | \
        (1ULL << EFX_PHY_STAT_PHY_XS_ALIGN) | \
        (1ULL << EFX_PHY_STAT_PHY_XS_SYNC_A) | \
        (1ULL << EFX_PHY_STAT_PHY_XS_SYNC_B) | \
        (1ULL << EFX_PHY_STAT_PHY_XS_SYNC_C) | \
        (1ULL << EFX_PHY_STAT_PHY_XS_SYNC_D) | \
        (1ULL << EFX_PHY_STAT_AN_LINK_UP) | \
        (1ULL << EFX_PHY_STAT_AN_MASTER) | \
        (1ULL << EFX_PHY_STAT_AN_LOCAL_RX_OK) | \
        (1ULL << EFX_PHY_STAT_AN_REMOTE_RX_OK) | \
        (1ULL << EFX_PHY_STAT_SNR_A) | \
        (1ULL << EFX_PHY_STAT_SNR_B) | \
```

```
128         (1ULL << EFX_PHY_STAT_SNR_C) | \
129         (1ULL << EFX_PHY_STAT_SNR_D)

131 /* END MKCONFIG GENERATED Sfx7101PhyHeaderStatsMask */

133 extern  __checkReturn                  int
134 sfx7101_stats_update(
135         __in                            efx_nic_t *enp,
136         __in                            efsys_mem_t *esmp,
137         __out_ecount(EFX_PHY_NSTATS)    uint32_t *stat);

139 #endif  /* EFSYS_OPT_PHY_STATS */

141 #if EFSYS_OPT_PHY_PROPS

143 #if EFSYS_OPT_NAMES

145 extern          const char __cs *
146 sfx7101_prop_name(
147         __in    efx_nic_t *enp,
148         __in    unsigned int id);

150 #endif

152 extern  __checkReturn   int
153 sfx7101_prop_get(
154         __in            efx_nic_t *enp,
155         __in            unsigned int id,
156         __in            uint32_t flags,
157         __out           uint32_t *valp);

159 extern  __checkReturn   int
160 sfx7101_prop_set(
161         __in            efx_nic_t *enp,
162         __in            unsigned int id,
163         __in            uint32_t val);

165 #endif  /* EFSYS_OPT_PHY_PROPS */

167 #if EFSYS_OPT_NVRAM_SFX7101

169 extern  __checkReturn           int
170 sfx7101_nvram_size(
171         __in                    efx_nic_t *enp,
172         __out                   size_t *sizep);

174 extern  __checkReturn           int
175 sfx7101_nvram_get_version(
176         __in                    efx_nic_t *enp,
177         __out                   uint32_t *subtypep,
178         __out_ecount(4)         uint16_t version[4]);

180 extern  __checkReturn           int
181 sfx7101_nvram_rw_start(
182         __in                    efx_nic_t *enp,
183         __out                   size_t *block_sizep);

185 extern  __checkReturn           int
186 sfx7101_nvram_read_chunk(
187         __in                    efx_nic_t *enp,
188         __in                    unsigned int offset,
189         __out_bcount(size)      caddr_t data,
190         __in                    size_t size);

192 extern  __checkReturn           int
193 sfx7101_nvram_erase(
```



```
194         __in                    efx_nic_t *enp);

196 extern  __checkReturn           int
197 sfx7101_nvram_write_chunk(
198         __in                    efx_nic_t *enp,
199         __in                    unsigned int offset,
200         __in_bcount(size)       caddr_t data,
201         __in                    size_t size);

203 extern                          void
204 sfx7101_nvram_rw_finish(
205         __in                    efx_nic_t *enp);

207 extern const uint8_t * const sfx7101_loader;
208 extern const size_t sfx7101_loader_size;

210 #pragma pack(1)

212 typedef struct sfx7001_firmware_header_s {
213         efx_dword_t code_length;
214         efx_dword_t destination_address;
215         efx_word_t code_checksum;
216         efx_word_t boot_config;
217         efx_word_t header_csum;
218         efx_byte_t reserved[18];
219 } sfx7101_firmware_header_t;

221 #pragma pack()

223 #endif  /* EFSYS_OPT_NVRAM_SFX7101 */

225 #endif  /* EFSYS_OPT_PHY_SFX7101 */

227 #ifdef  __cplusplus
228 }
229 #endif

231 #endif  /* _SYS_SFX7101_H */
232 #endif /* ! codereview */
```

```
**********************************************************
    5759 Thu Aug 22 18:59:27 2013
new/usr/src/uts/common/io/sfxge/sfx7101_impl.h
Merged sfxge driver
**********************************************************
```

```
/*-
 * Copyright 2007-2013 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#ifndef _SYS_SFX7101_IMPL_H
#define _SYS_SFX7101_IMPL_H

#ifdef  __cplusplus
extern "C" {
#endif

#if EFSYS_OPT_PHY_SFX7101

#define SFX7101_MMD_MASK                                        \
            ((1 << PMA_PMD_MMD) |                               \
            (1 << PCS_MMD) |                                    \
            (1 << PHY_XS_MMD) |                                 \
            (1 << AN_MMD))

/* IO expender */
#define PCA9539 0x74

#define P0_IN 0x00
#define P0_OUT 0x02
#define P0_INVERT 0x04
#define P0_CONFIG 0x06

#define P0_EN_1V0X_LBN 0
#define P0_EN_1V0X_WIDTH 1
#define P0_EN_1V2_LBN 1
#define P0_EN_1V2_WIDTH 1
#define P0_EN_2V5_LBN 2
#define P0_EN_2V5_WIDTH 1
#define P0_FLASH_CFG_EN_LBN 3
#define P0_FLASH_CFG_EN_WIDTH 1
#define P0_EN_5V_LBN 4
#define P0_EN_5V_WIDTH 1
#define P0_SHORTEN_JTAG_LBN 5
#define P0_SHORTEN_JTAG_WIDTH 1
#define P0_X_TRST_LBN 6
```



```
#define P0_X_TRST_WIDTH 1
#define P0_DSP_RESET_LBN 7
#define P0_DSP_RESET_WIDTH 1

#define P1_IN 0x01
#define P1_OUT 0x03
#define P1_INVERT 0x05
#define P1_CONFIG 0x07

#define P1_AFE_PWD_LBN 0
#define P1_AFE_PWD_WIDTH 1
#define P1_DSP_PWD25_LBN 1
#define P1_DSP_PWD25_WIDTH 1
#define P1_RESERVED_LBN 2
#define P1_RESERVED_WIDTH 2
#define P1_SPARE_LBN 4
#define P1_SPARE_WIDTH 4

#define PMA_PMD_XCONTROL_REG 0xc000
#define SSR_LBN 15
#define SSR_WIDTH 1
#define LNPGA_POWERDOWN_LBN 8
#define LNPGA_POWERDOWN_WIDTH 1
#define AFE_POWERDOWN_LBN 9
#define AFE_POWERDOWN_WIDTH 1
#define DSP_POWERDOWN_LBN 10
#define DSP_POWERDOWN_WIDTH 1
#define PHY_POWERDOWN_LBN 11
#define PHY_POWERDOWN_WIDTH 1

#define PMA_PMD_XSTATUS_REG 0xc001
#define FLP_RCVD_LBN 12
#define FLP_RCVD_WIDTH 1

#define PMA_PMD_LED_CONTROL_REG 0xc007
#define LED_ACTIVITY_EN_LBN 3
#define LED_ACTIVITY_EN_WIDTH 1
#define LED_FLASH_PERIOD_LBN 0
#define LED_FLASH_PERIOD_WIDTH 3

#define PMA_PMD_LED_OVERRIDE_REG 0xc009
#define LED_LINK_LBN 0 /* Green */
#define LED_LINK_WIDTH 2
#define LED_TX_LBN 4 /* Amber */
#define LED_TX_WIDTH 2
#define LED_RX_LBN 6 /* Red */
#define LED_RX_WIDTH 2
#define LED_NORMAL_DECODE 0x0
#define LED_ON_DECODE 0x1
#define LED_OFF_DECODE 0x2
#define LED_FLASH_DECODE 0x3

#define PMA_PMD_FW_REV0_REG 0xc026
#define PMA_PMD_FW_REV1_REG 0xc027

#define PCS_BOOT_STATUS_REG 0xd000      /* PRM 10.4.1 */
#define RESET_CAUSE_LBN 8
#define RESET_CAUSE_WIDTH 2
#define HW_RESET_DECODE 0x0
#define SW_RESET_DECODE 0x1
#define WD_RESET_DECODE 0x2
#define SW_WD_RESET_DECODE 0x3
#define UPLOAD_PROGRESS_LBN 7
#define UPLOAD_PROGRESS_WIDTH 1
#define CODE_DOWNLOAD_LBN 6
#define CODE_DOWNLOAD_WIDTH 1
```

```
#define CKSUM_OK_LBN 5
#define CKSUM_OK_WIDTH 1
#define CODE_STARTED_LBN 4
#define CODE_STARTED_WIDTH 1
#define BOOT_STATUS_LBN 3
#define BOOT_STATUS_WIDTH 1
#define BOOT_PROGRESS_LBN 1
#define BOOT_PROGRESS_WIDTH 2
#define INIT_DECODE 0x0
#define MDIO_WAIT_DECODE 0x1
#define CKSUM_START_DECODE 0x2
#define APP_JMP_DECODE 0x3
#define FATAL_ERR_LBN 0
#define FATAL_ERR_WIDTH 1

#define PCS_LM_RAM_LS_ADDR_REG 0xd004
#define LM_RAM_LS_ADDR_LBN 0
#define LM_RAM_LS_ADDR_WIDTH 16

#define PCS_LM_RAM_MS_ADDR_REG 0xd005
#define LM_RAM_MS_ADDR_LBN 0
#define LM_RAM_MS_ADDR_WIDTH 3
#define BYTE_ACCESS_LBN 15
#define BYTE_ACCESS_WIDTH 1

#define PCS_LM_RAM_DATA_REG 0xd006
#define LM_RAM_DATA_LBN 0
#define LM_RAM_DATA_WIDTH 16

#define PCS_TEST_SELECT_REG 0xd807      /* PRM 10.5.8 */
#define CLK312_EN_LBN 3
#define CLK312_EN_WIDTH 1

#define PCS_CLOCK_CTRL_REG 0xd801
#define PLL312_RST_N_LBN 2
#define PLL312_RST_N_WIDTH 1

#define PCS_SOFT_RST2_REG 0xd806
#define SERDES_RST_N_LBN 13
#define SERDES_RST_N_WIDTH 1
#define XGXS_RST_N_LBN 12
#define XGXS_RST_N_WIDTH 1

#define PHY_XS_XGXS_TEST_REG 0xc00a
#define SERDES_LOOPBACK_LBN 9
#define SERDES_LOOPBACK_WIDTH 1
#define NE_LOOPBACK_LBN 8
#define NE_LOOPBACK_WIDTH 1

#define LOADER_MMD 1
#define LOADER_MAX_BUFF_SZ_REG 49192
#define LOADER_ACTUAL_BUFF_SZ_REG 49193
#define LOADER_CMD_RESPONSE_REG 49194
#define LOADER_CMD_ERASE_FLASH 0x0001
#define LOADER_CMD_FILL_BUFFER 0x0002
#define LOADER_CMD_PROGRAM_FLASH 0x0003
#define LOADER_CMD_READ_FLASH 0x0004
#define LOADER_RESPONSE_OK 0x0100
#define LOADER_RESPONSE_ERROR 0x0200
#define LOADER_RESPONSE_BUSY 0x0300
#define LOADER_WORDS_WRITTEN_REG 49195
#define LOADER_WORDS_READ_REG 49195
#define LOADER_FLASH_ADDR_LOW_REG 49196
#define LOADER_FLASH_ADDR_HI_REG  49197
#define LOADER_DATA_REG 49198
```



```
#define FIRMWARE_BLOCK_SIZE 0x4000
#define FIRMWARE_MAX_SIZE 0x30000

#endif  /* EFSYS_OPT_PHY_SFX7101 */

#ifdef  __cplusplus
}
#endif

#endif  /* _SYS_SFX7101_IMPL_H */
#endif /* ! codereview */
```

```
**********************************************************
   30512 Thu Aug 22 18:59:27 2013
new/usr/src/uts/common/io/sfxge/sfxge.c
Merged sfxge driver
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2008-2013 Solarflare Communications Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#include <sys/types.h>
#include <sys/ddi.h>
#include <sys/sunddi.h>
#include <sys/modctl.h>
#include <sys/conf.h>
#include <sys/ethernet.h>
#include <sys/pci.h>
#include <sys/stream.h>
#include <sys/strsun.h>
#include <sys/processor.h>
#include <sys/cpuvar.h>
#include <sys/pghw.h>

#include "version.h"

#include "sfxge.h"
#include "efsys.h"
#include "efx.h"

#ifdef  DEBUG
boolean_t sfxge_aask = B_FALSE;
#endif

/* Receive queue TRIM default polling interval (in microseconds) */
#define SFXGE_RX_QPOLL_USEC     (5000000)

/* Broadcast address */
uint8_t sfxge_brdcst[] = { 0xff, 0xff, 0xff, 0xff, 0xff, 0xff };

/* Soft state head */
static void     *sfxge_ss;

/*
 * By default modinfo will display lines truncated to 80 characters and so just
 * show 32 characters of our sfxge_ident string. At the moment CI_VERSION_STRING
```



```
 * is 12 characters. To show the whole string use modinfo -w
 */
#if defined(_USE_GLD_V3_SOL10) && !defined(_USE_GLD_V3_SOL11)
#ifdef DEBUG
/*
 * The (DEBUG) part of this string will not be displayed in modinfo by
 * default. See previous comment.
 */
const char sfxge_ident[] =
    CI_VERSION_STRING" for Sol10u8,u9,u10 (DEBUG)";
#else
const char sfxge_ident[] =
    CI_VERSION_STRING" for Sol10u8,u9,u10";
#endif
#elif defined(_USE_GLD_V3_SOL11)
#ifdef DEBUG
const char sfxge_ident[] = CI_VERSION_STRING" for Sol11 (DEBUG)";
#else
const char sfxge_ident[] = CI_VERSION_STRING" for Sol11";
#endif
#elif defined(_USE_GLD_V3)
#ifdef DEBUG
const char sfxge_ident[] = CI_VERSION_STRING" GLDv3 (DEBUG)";
#else
const char sfxge_ident[] = CI_VERSION_STRING" GLDv3";
#endif
#elif defined(_USE_GLD_V2)
#ifdef DEBUG
const char sfxge_ident[] = CI_VERSION_STRING" GLDv2 (DEBUG)";
#else
const char sfxge_ident[] = CI_VERSION_STRING" GLDv2";
#endif
#else
#error "sfxge_ident undefined"
#endif
const char sfxge_version[] = CI_VERSION_STRING;

static void
sfxge_cfg_build(sfxge_t *sp)
{
        const efx_nic_cfg_t *encp = efx_nic_cfg_get(sp->s_enp);
        (void) snprintf(sp->s_cfg_kstat.buf.sck_mac, 64,
                        "%02X:%02X:%02X:%02X:%02X:%02X",
                        encp->enc_mac_addr[0], encp->enc_mac_addr[1],
                        encp->enc_mac_addr[2], encp->enc_mac_addr[3],
                        encp->enc_mac_addr[4], encp->enc_mac_addr[5]);
}

static int
sfxge_create(dev_info_t *dip, sfxge_t **spp)
{
        int instance = ddi_get_instance(dip);
        sfxge_t *sp;
        efx_nic_t *enp;
        char name[MAXNAMELEN];
        unsigned int rxq_size;
        int rxq_poll_usec;
        int rc;

        /* Allocate the soft state object */
        if (ddi_soft_state_zalloc(sfxge_ss, instance) != DDI_SUCCESS) {
                rc = ENOMEM;
                goto fail1;
        }

        sp = ddi_get_soft_state(sfxge_ss, instance);
```

```
128         ASSERT(sp != NULL);

130         SFXGE_OBJ_CHECK(sp, sfxge_t);

132         sp->s_dip = dip;

134         mutex_init(&(sp->s_state_lock), "", MUTEX_DRIVER, NULL);
135         sp->s_state = SFXGE_UNINITIALIZED;

137         /* Get property values */
138         sp->s_mtu = ddi_prop_get_int(DDI_DEV_T_ANY, sp->s_dip,
139             DDI_PROP_DONTPASS, "mtu", ETHERMTU);

141         sp->s_action_on_hw_err = ddi_prop_get_int(DDI_DEV_T_ANY, sp->s_dip,
142             DDI_PROP_DONTPASS, "action_on_hw_err", SFXGE_RECOVER);

144         rxq_size = ddi_prop_get_int(DDI_DEV_T_ANY, sp->s_dip,
145             DDI_PROP_DONTPASS, "rxq_size", SFXGE_DEFAULT_RXQ_SIZE);
146         if (!(IS_POW2(rxq_size)))
147                 rxq_size = SFXGE_DEFAULT_RXQ_SIZE;
148         rxq_size = min(rxq_size, EFX_RXQ_MAXNDESCS);
149         sp->s_rxq_size = max(rxq_size, EFX_RXQ_MINNDESCS);

151         /* Configure polling interval for queue refill/trim */
152         rxq_poll_usec = ddi_prop_get_int(DDI_DEV_T_ANY, sp->s_dip,
153                 DDI_PROP_DONTPASS, "rxq_poll_usec", SFXGE_RX_QPOLL_USEC);
154         if (rxq_poll_usec <= 0)
155                 rxq_poll_usec = SFXGE_RX_QPOLL_USEC;
156         sp->s_rxq_poll_usec = rxq_poll_usec;

158         /* Create a taskq */
159         (void) snprintf(name, MAXNAMELEN - 1, "%s_tq", ddi_driver_name(dip));
160         sp->s_tqp = ddi_taskq_create(dip, name, 1, TASKQ_DEFAULTPRI, DDI_SLEEP);
161         if (sp->s_tqp == NULL) {
162                 rc = ENOMEM;
163                 goto fail2;
164         }

166         /* Check and initialize PCI configuration space */
167         if ((rc = sfxge_pci_init(sp)) != 0)
168                 goto fail3;

170         /* Map the device registers */
171         if ((rc = sfxge_bar_init(sp)) != 0)
172                 goto fail4;

174         /* Create the NIC object */
175         mutex_init(&(sp->s_nic_lock), NULL, MUTEX_DRIVER, NULL);

177         if ((rc = efx_nic_create(sp->s_family, (efsys_identifier_t *)sp,
178             &(sp->s_bar), &(sp->s_nic_lock), &enp)) != 0)
179                 goto fail5;

181         sp->s_enp = enp;

183         /* Initialize MCDI to talk to the Microcontroller */
184         if ((rc = sfxge_mcdi_init(sp)) != 0)
185                 goto fail6;

187         /* Probe the NIC and build the configuration data area */
188         if ((rc = efx_nic_probe(enp)) != 0)
189                 goto fail7;

191         if (sp->s_family == EFX_FAMILY_SIENA) {
192                 sfxge_pcie_check_link(sp, 8, 2); /* PCI 8x Gen2 */
```



```
194         } else if (sp->s_family == EFX_FAMILY_FALCON) {
195                 sfxge_pcie_check_link(sp, 8, 1); /* PCI 8x Gen1 */

197                 rc = efx_nic_pcie_tune(enp, sp->s_pcie_nlanes);
198                 ASSERT(rc == 0);
199         }

201         if ((rc = efx_nvram_init(enp)) != 0)
202                 goto fail8;

204         if ((rc = efx_vpd_init(enp)) != 0)
205                 goto fail9;

207         if ((rc = efx_nic_reset(enp)) != 0)
208                 goto fail10;

210         sfxge_sram_init(sp);

212         if ((rc = sfxge_intr_init(sp)) != 0)
213                 goto fail11;

215         if ((rc = sfxge_ev_init(sp)) != 0)
216                 goto fail12;

218         if ((rc = sfxge_rx_init(sp)) != 0)
219                 goto fail13;

221         if ((rc = sfxge_tx_init(sp)) != 0)
222                 goto fail14;

224         if ((rc = sfxge_mon_init(sp)) != 0)
225                 goto fail15;

227         if ((rc = sfxge_mac_init(sp)) != 0)
228                 goto fail16;

230         mutex_init(&(sp->s_tx_flush_lock), "", MUTEX_DRIVER,
231             DDI_INTR_PRI(sp->s_intr.si_intr_pri));
232         cv_init(&(sp->s_tx_flush_kv), "", CV_DRIVER, NULL);

234         sp->s_state = SFXGE_INITIALIZED;

236         *spp = sp;
237         return (0);

239 fail16:
240         DTRACE_PROBE(fail16);
241         sfxge_mon_fini(sp);

243 fail15:
244         DTRACE_PROBE(fail15);
245         sfxge_tx_fini(sp);

247 fail14:
248         DTRACE_PROBE(fail14);
249         sfxge_rx_fini(sp);

251 fail13:
252         DTRACE_PROBE(fail13);
253         sfxge_ev_fini(sp);

255 fail12:
256         DTRACE_PROBE(fail12);
257         sfxge_intr_fini(sp);

259 fail11:
```

```
260         DTRACE_PROBE(fail11);
261         sfxge_sram_fini(sp);
262         (void) efx_nic_reset(sp->s_enp);

264 fail10:
265         DTRACE_PROBE(fail10);
266         efx_vpd_fini(enp);

268 fail9:
269         DTRACE_PROBE(fail9);
270         efx_nvram_fini(enp);

272 fail8:
273         DTRACE_PROBE(fail8);
274         efx_nic_unprobe(enp);

276 fail7:
277         DTRACE_PROBE(fail7);
278         sfxge_mcdi_fini(sp);

280 fail6:
281         DTRACE_PROBE(fail6);
282         sp->s_enp = NULL;
283         efx_nic_destroy(enp);

285 fail5:
286         DTRACE_PROBE(fail5);
287         mutex_destroy(&(sp->s_nic_lock));
288         sfxge_bar_fini(sp);

290 fail4:
291         DTRACE_PROBE(fail4);
292         sfxge_pci_fini(sp);

294 fail3:
295         DTRACE_PROBE(fail3);
296         ddi_taskq_destroy(sp->s_tqp);
297         sp->s_tqp = NULL;

299 fail2:
300         DTRACE_PROBE(fail2);

302         /* Clear property values */
303         sp->s_mtu = 0;

305         mutex_destroy(&(sp->s_state_lock));

307         /* Free the soft state */
308         sp->s_dip = NULL;

310         SFXGE_OBJ_CHECK(sp, sfxge_t);
311         ddi_soft_state_free(sfxge_ss, instance);

313 fail1:
314         DTRACE_PROBE1(fail1, int, rc);

316         return (rc);
317 }


320 static int
321 sfxge_start_locked(sfxge_t *sp, boolean_t restart)
322 {
323         int rc;

325         ASSERT(mutex_owned(&(sp->s_state_lock)));
```



```
327         if (sp->s_state == SFXGE_STARTED)
328                 goto done;

330         if (sp->s_state != SFXGE_REGISTERED) {
331                 rc = EINVAL;
332                 goto fail1;
333         }
334         sp->s_state = SFXGE_STARTING;

336         if ((rc = efx_nic_reset(sp->s_enp)) != 0)
337                 goto fail2;

339         if ((rc = efx_nic_init(sp->s_enp)) != 0)
340                 goto fail3;

342         if ((rc = sfxge_sram_start(sp)) != 0)
343                 goto fail4;

345         if ((rc = sfxge_intr_start(sp)) != 0)
346                 goto fail5;

348         if ((rc = sfxge_ev_start(sp)) != 0)
349                 goto fail6;

351         if ((rc = sfxge_rx_start(sp)) != 0)
352                 goto fail7;

354         if ((rc = sfxge_tx_start(sp)) != 0)
355                 goto fail8;

357         if ((rc = sfxge_mon_start(sp)) != 0)
358                 goto fail9;

360         if ((rc = sfxge_mac_start(sp, restart)) != 0)
361                 goto fail10;

363         ASSERT3U(sp->s_state, ==, SFXGE_STARTING);
364         sp->s_state = SFXGE_STARTED;

366         /* Notify any change of MTU */
367         sfxge_gld_mtu_update(sp);

369 done:
370         return (0);

372 fail10:
373         DTRACE_PROBE(fail10);
374         sfxge_mon_stop(sp);

376 fail9:
377         DTRACE_PROBE(fail9);
378         sfxge_tx_stop(sp);

380 fail8:
381         DTRACE_PROBE(fail8);
382         sfxge_rx_stop(sp);

384 fail7:
385         DTRACE_PROBE(fail7);
386         sfxge_ev_stop(sp);

388 fail6:
389         DTRACE_PROBE(fail6);
390         sfxge_intr_stop(sp);
```

```
392 fail5:
393         DTRACE_PROBE(fail5);
394         sfxge_sram_stop(sp);

396 fail4:
397         DTRACE_PROBE(fail4);
398         efx_nic_fini(sp->s_enp);

400 fail3:
401         DTRACE_PROBE(fail3);
402         (void) efx_nic_reset(sp->s_enp);

404 fail2:
405         DTRACE_PROBE(fail2);

407         ASSERT3U(sp->s_state, ==, SFXGE_STARTING);
408         sp->s_state = SFXGE_REGISTERED;

410 fail1:
411         DTRACE_PROBE1(fail1, int, rc);

413         return (rc);
414 }


417 int
418 sfxge_start(sfxge_t *sp, boolean_t restart)
419 {
420         int rc;

422         mutex_enter(&(sp->s_state_lock));
423         rc = sfxge_start_locked(sp, restart);
424         mutex_exit(&(sp->s_state_lock));
425         return (rc);
426 }


429 static void
430 sfxge_stop_locked(sfxge_t *sp)
431 {
432         ASSERT(mutex_owned(&(sp->s_state_lock)));

434         if (sp->s_state != SFXGE_STARTED) {
435                 return;
436         }
437         sp->s_state = SFXGE_STOPPING;

439         sfxge_mac_stop(sp);
440         sfxge_mon_stop(sp);
441         sfxge_tx_stop(sp);
442         sfxge_rx_stop(sp);

444         /* Stop event processing - must be after rx_stop see sfxge_rx_qpoll() */
445         sfxge_ev_stop(sp);
446         sfxge_intr_stop(sp); /* cope with late flush/soft events until here */
447         sfxge_sram_stop(sp);

449         efx_nic_fini(sp->s_enp);
450         efx_nic_reset(sp->s_enp);

452         ASSERT3U(sp->s_state, ==, SFXGE_STOPPING);
453         sp->s_state = SFXGE_REGISTERED;
454 }

456 void
457 sfxge_stop(sfxge_t *sp)
```



```
458 {
459         mutex_enter(&(sp->s_state_lock));
460         sfxge_stop_locked(sp);
461         mutex_exit(&(sp->s_state_lock));
462 }

464 static void
465 _sfxge_restart(void *arg)
466 {
467         sfxge_t *sp = arg;
468         int rc;

470         /* logging on entry is in sfxge_restart_dispatch */
471         mutex_enter(&(sp->s_state_lock));

473         DTRACE_PROBE(_sfxge_restart);
474         if (sp->s_state != SFXGE_STARTED)
475                 goto done;

477         /* inform the OS that the link is down - may trigger IPMP failover */
478         if (sp->s_hw_err && sp->s_action_on_hw_err != SFXGE_INVISIBLE) {
479                 sp->s_mac.sm_link_mode = EFX_LINK_DOWN;
480                 sfxge_gld_link_update(sp);
481         }

483         /* Stop processing */
484         sfxge_stop_locked(sp);

486         if (sp->s_hw_err && sp->s_action_on_hw_err == SFXGE_LEAVE_DEAD) {
487                 cmn_err(CE_WARN, SFXGE_CMN_ERR "[%s%d] NIC error - interface is"
488                     " being left permanently DOWN per driver config",
489                     ddi_driver_name(sp->s_dip), ddi_get_instance(sp->s_dip));
490                 mutex_exit(&(sp->s_state_lock));
491                 return;
492         } else
493                 sp->s_hw_err = SFXGE_HW_OK;

495         /* Start processing */
496         if ((rc = sfxge_start_locked(sp, B_TRUE)) != 0)
497                 goto fail1;

499 done:
500         mutex_exit(&(sp->s_state_lock));
501         cmn_err(CE_WARN, SFXGE_CMN_ERR "[%s%d] NIC restart complete",
502             ddi_driver_name(sp->s_dip), ddi_get_instance(sp->s_dip));
503         return;

505 fail1:
506         DTRACE_PROBE1(fail1, int, rc);
507         cmn_err(CE_WARN,
508             SFXGE_CMN_ERR "[%s%d] FATAL ERROR: NIC restart failed rc=%d",
509             ddi_driver_name(sp->s_dip), ddi_get_instance(sp->s_dip), rc);

511         mutex_exit(&(sp->s_state_lock));
512 }

514 int
515 sfxge_restart_dispatch(sfxge_t *sp, uint_t cflags, sfxge_hw_err_t hw_err,
516     const char *reason, uint32_t errval)
517 {
518         if (hw_err == SFXGE_HW_OK)
519                 sp->s_num_restarts++;
520         else {
521                 sp->s_hw_err = hw_err;
522                 sp->s_num_restarts_hw_err++;
523         }
```

```
525         DTRACE_PROBE2(sfxge_restart_dispatch, sfxge_hw_err_t, hw_err, char *,
526             reason);

528         cmn_err(CE_WARN, SFXGE_CMN_ERR "[%s%d] NIC restart due to %s:%d",
529             ddi_driver_name(sp->s_dip), ddi_get_instance(sp->s_dip), reason,
530             errval);

532         /* If cflags == DDI_SLEEP then guaranteed to succeed */
533         return (ddi_taskq_dispatch(sp->s_tqp, _sfxge_restart, sp, cflags));
534 }


537 static int
538 sfxge_can_destroy(sfxge_t *sp)
539 {
540         sfxge_intr_t *sip = &(sp->s_intr);
541         int index;

543         /*
544          * In SFC bug 19834 it was noted that a mblk passed up to STREAMS
545          * could be reused for transmit and sit in the sfxge_tx_packet_cache.
546          * This call to empty the TX deferred packet list may result in
547          * rx_loaned reducing.
548          */
549         index = sip->si_nalloc;
550         while (--index >= 0) {
551                 sfxge_txq_t *stp = sp->s_stp[index];
552                 sfxge_tx_qdpl_flush(stp);
553         }

555         /* Need to wait for desballoc free_func callback */
556         return (sfxge_rx_loaned(sp));
557 }


560 static int
561 sfxge_destroy(sfxge_t *sp)
562 {
563         dev_info_t *dip = sp->s_dip;
564         int instance = ddi_get_instance(dip);
565         ddi_taskq_t *tqp;
566         efx_nic_t *enp;
567         int rc;

569         ASSERT3U(sp->s_state, ==, SFXGE_INITIALIZED);
570         enp = sp->s_enp;

572         if (sfxge_can_destroy(sp) != 0) {
573                 rc = EBUSY;
574                 goto fail1;
575         }

577         sp->s_state = SFXGE_UNINITIALIZED;

579         cv_destroy(&(sp->s_tx_flush_kv));
580         mutex_destroy(&(sp->s_tx_flush_lock));

582         sfxge_mac_fini(sp);
583         sfxge_mon_fini(sp);
584         sfxge_tx_fini(sp);
585         sfxge_rx_fini(sp);
586         sfxge_ev_fini(sp);
587         sfxge_intr_fini(sp);
588         sfxge_sram_fini(sp);
589         (void) efx_nic_reset(enp);
```



```
591         efx_vpd_fini(enp);
592         efx_nvram_fini(enp);
593         efx_nic_unprobe(enp);
594         sfxge_mcdi_fini(sp);

596         /* Destroy the NIC object */
597         sp->s_enp = NULL;
598         efx_nic_destroy(enp);

600         mutex_destroy(&(sp->s_nic_lock));

602         /* Unmap the device registers */
603         sfxge_bar_fini(sp);

605         /* Tear down PCI configuration space */
606         sfxge_pci_fini(sp);

608         /* Destroy the taskq */
609         tqp = sp->s_tqp;
610         sp->s_tqp = NULL;
611         ddi_taskq_destroy(tqp);

613         mutex_destroy(&(sp->s_state_lock));

615         /* Clear property values */
616         sp->s_mtu = 0;

618         /* Free the soft state */
619         sp->s_dip = NULL;

621         SFXGE_OBJ_CHECK(sp, sfxge_t);
622         ddi_soft_state_free(sfxge_ss, instance);

624         return (0);

626 fail1:
627         DTRACE_PROBE1(fail1, int, rc);

629         return (rc);
630 }

632 void
633 sfxge_ioctl(sfxge_t *sp, queue_t *wq, mblk_t *mp)
634 {
635         struct iocblk *iocp;
636         int rc, taskq_wait = 0;
637         size_t ioclen = 0;

639         /*
640          * single concurrent IOCTL
641          * serialized from sfxge_create, _destroy, _(re)start, _stop
642          */
643         mutex_enter(&(sp->s_state_lock));

645         /*LINTED*/
646         iocp = (struct iocblk *)mp->b_rptr;

648         switch (iocp->ioc_cmd) {
649         case SFXGE_TX_IOC:
650                 ioclen = sizeof (sfxge_tx_ioc_t);
651                 break;
652         case SFXGE_RX_IOC:
653                 ioclen = sizeof (sfxge_rx_ioc_t);
654                 break;
655         case SFXGE_BAR_IOC:
```

```
656                 ioclen = sizeof (sfxge_bar_ioc_t);
657                 break;
658         case SFXGE_PCI_IOC:
659                 ioclen = sizeof (sfxge_pci_ioc_t);
660                 break;
661         case SFXGE_MAC_IOC:
662                 ioclen = sizeof (sfxge_mac_ioc_t);
663                 break;
664         case SFXGE_PHY_IOC:
665                 ioclen = sizeof (sfxge_phy_ioc_t);
666                 break;
667         case SFXGE_PHY_BIST_IOC:
668                 ioclen = sizeof (sfxge_phy_bist_ioc_t);
669                 break;
670         case SFXGE_SRAM_IOC:
671                 ioclen = sizeof (sfxge_sram_ioc_t);
672                 break;
673         case SFXGE_NVRAM_IOC:
674                 ioclen = sizeof (sfxge_nvram_ioc_t);
675                 break;
676         case SFXGE_MCDI_IOC:
677                 ioclen = sizeof (sfxge_mcdi_ioc_t);
678                 break;
679         case SFXGE_VPD_IOC:
680                 ioclen = sizeof (sfxge_vpd_ioc_t);
681                 break;
682         case SFXGE_START_IOC:
683         case SFXGE_STOP_IOC:
684         case SFXGE_NIC_RESET_IOC:
685                 break;
686         default:
687                 rc = ENOTSUP;
688                 goto fail1;
689         }

691         if (iocp->ioc_count != ioclen) {
692                 rc = EINVAL;
693                 goto fail2;
694         }

696         /* if in multiple fragments pull it up to one linear buffer */
697         if ((rc = miocpullup(mp, ioclen)) != 0) {
698                 goto fail3;
699         }

701         switch (iocp->ioc_cmd) {
702         case SFXGE_START_IOC:
703                 if ((rc = sfxge_start_locked(sp, B_TRUE)) != 0)
704                         goto fail4;

706                 break;

708         case SFXGE_STOP_IOC:
709                 sfxge_stop_locked(sp);
710                 break;

712         case SFXGE_TX_IOC: {
713                 sfxge_tx_ioc_t *stip = (sfxge_tx_ioc_t *)mp->b_cont->b_rptr;

715                 if ((rc = sfxge_tx_ioctl(sp, stip)) != 0)
716                         goto fail4;

718                 break;
719         }
720         case SFXGE_RX_IOC: {
721                 sfxge_rx_ioc_t *srip = (sfxge_rx_ioc_t *)mp->b_cont->b_rptr;
```



```
723                 if ((rc = sfxge_rx_ioctl(sp, srip)) != 0)
724                         goto fail4;

726                 break;
727         }
728         case SFXGE_BAR_IOC: {
729                 sfxge_bar_ioc_t *sbip = (sfxge_bar_ioc_t *)mp->b_cont->b_rptr;

731                 if ((rc = sfxge_bar_ioctl(sp, sbip)) != 0)
732                         goto fail4;

734                 break;
735         }
736         case SFXGE_PCI_IOC: {
737                 sfxge_pci_ioc_t *spip = (sfxge_pci_ioc_t *)mp->b_cont->b_rptr;

739                 if ((rc = sfxge_pci_ioctl(sp, spip)) != 0)
740                         goto fail4;

742                 break;
743         }
744         case SFXGE_MAC_IOC: {
745                 sfxge_mac_ioc_t *smip = (sfxge_mac_ioc_t *)mp->b_cont->b_rptr;

747                 if ((rc = sfxge_mac_ioctl(sp, smip)) != 0)
748                         goto fail4;

750                 break;
751         }
752         case SFXGE_PHY_IOC: {
753                 sfxge_phy_ioc_t *spip = (sfxge_phy_ioc_t *)mp->b_cont->b_rptr;

755                 if ((rc = sfxge_phy_ioctl(sp, spip)) != 0)
756                         goto fail4;

758                 break;
759         }
760         case SFXGE_PHY_BIST_IOC: {
761                 sfxge_phy_bist_ioc_t *spbip;

763                 spbip = (sfxge_phy_bist_ioc_t *)mp->b_cont->b_rptr;

765                 if ((rc = sfxge_phy_bist_ioctl(sp, spbip)) != 0)
766                         goto fail4;

768                 break;
769         }
770         case SFXGE_SRAM_IOC: {
771                 sfxge_sram_ioc_t *ssip = (sfxge_sram_ioc_t *)mp->b_cont->b_rptr;

773                 if ((rc = sfxge_sram_ioctl(sp, ssip)) != 0)
774                         goto fail4;

776                 break;
777         }
778         case SFXGE_NVRAM_IOC: {
779                 sfxge_nvram_ioc_t *snip =
780                     (sfxge_nvram_ioc_t *)mp->b_cont->b_rptr;

782                 if ((rc = sfxge_nvram_ioctl(sp, snip)) != 0)
783                         goto fail4;

785                 break;
786         }
787         case SFXGE_MCDI_IOC: {
```

```
788                 sfxge_mcdi_ioc_t *smip = (sfxge_mcdi_ioc_t *)mp->b_cont->b_rptr;

790                 if ((rc = sfxge_mcdi_ioctl(sp, smip)) != 0)
791                         goto fail4;
792                 taskq_wait = 1;

794                 break;
795         }
796         case SFXGE_NIC_RESET_IOC: {
797                 DTRACE_PROBE(nic_reset_ioc);

799                 /* sp->s_state_lock held */
800                 (void) sfxge_restart_dispatch(sp, DDI_SLEEP, SFXGE_HW_OK,
801                     "NIC_RESET_IOC", 0);
802                 taskq_wait = 1;

804                 break;
805         }
806         case SFXGE_VPD_IOC: {
807                 sfxge_vpd_ioc_t *svip = (sfxge_vpd_ioc_t *)mp->b_cont->b_rptr;

809                 if ((rc = sfxge_vpd_ioctl(sp, svip)) != 0)
810                         goto fail4;

812                 break;
813         }
814         default:
815                 ASSERT(0);
816         }

818         mutex_exit(&(sp->s_state_lock));

820         if (taskq_wait) {
821                 /*
822                  * Wait for any tasks that may be accessing GLD functions
823                  * This may end up waiting for multiple nic_resets
824                  * as it needs to be outside of s_state_lock for sfxge_restart()
825                  */
826                 ddi_taskq_wait(sp->s_tqp);
827         }

829         /* The entire structure is the acknowledgement */
830         miocack(wq, mp, iocp->ioc_count, 0);

832         return;

834 fail4:
835         DTRACE_PROBE(fail4);
836 fail3:
837         DTRACE_PROBE(fail3);
838 fail2:
839         DTRACE_PROBE(fail2);
840 fail1:
841         DTRACE_PROBE1(fail1, int, rc);

843         mutex_exit(&(sp->s_state_lock));

845         /* no data returned */
846         miocnak(wq, mp, 0, rc);
847 }

849 static int
850 sfxge_register(sfxge_t *sp)
851 {
852         int rc;
```



```
854         ASSERT3U(sp->s_state, ==, SFXGE_INITIALIZED);

856         if ((rc = sfxge_gld_register(sp)) != 0)
857                 goto fail1;

859         sp->s_state = SFXGE_REGISTERED;

861         return (0);

863 fail1:
864         DTRACE_PROBE1(fail1, int, rc);

866         return (rc);
867 }

869 static int
870 sfxge_unregister(sfxge_t *sp)
871 {
872         int rc;

874         ASSERT3U(sp->s_state, ==, SFXGE_REGISTERED);

876         /* Wait for any tasks that may be accessing GLD functions */
877         ddi_taskq_wait(sp->s_tqp);

879         if ((rc = sfxge_gld_unregister(sp)) != 0)
880                 goto fail1;

882         sp->s_state = SFXGE_INITIALIZED;

884         return (0);

886 fail1:
887         DTRACE_PROBE1(fail1, int, rc);

889         return (rc);
890 }

892 static void
893 _sfxge_vpd_kstat_init(sfxge_t *sp, caddr_t vpd, size_t size, efx_vpd_tag_t tag,
894             const char *keyword, sfxge_vpd_type_t type)
895 {
896         static const char unknown[] = "?";
897         efx_nic_t *enp = sp->s_enp;
898         sfxge_vpd_kstat_t *svkp = &(sp->s_vpd_kstat);
899         kstat_named_t *knp;
900         efx_vpd_value_t *evvp;

902         evvp = svkp->svk_vv + type;
903         evvp->evv_tag = tag;
904         evvp->evv_keyword = EFX_VPD_KEYWORD(keyword[0], keyword[1]);

906         if (efx_vpd_get(enp, vpd, size, evvp) != 0) {
907                 evvp->evv_length = strlen(unknown) + 1;
908                 memcpy(evvp->evv_value, unknown, evvp->evv_length);
909         }

911         knp = &(svkp->svk_stat[type]);

913         kstat_named_init(knp, (char *)keyword, KSTAT_DATA_STRING);
914         kstat_named_setstr(knp, (char *)evvp->evv_value);
915         svkp->svk_ksp->ks_data_size += sizeof (*evvp);
916 }

918 static int
919 sfxge_vpd_kstat_init(sfxge_t *sp)
```

```
{
        efx_nic_t *enp = sp->s_enp;
        sfxge_vpd_kstat_t *svkp = &(sp->s_vpd_kstat);
        dev_info_t *dip = sp->s_dip;
        char name[MAXNAMELEN];
        kstat_t *ksp;
        caddr_t vpd;
        size_t size;
        int rc;

        SFXGE_OBJ_CHECK(svkp, sfxge_vpd_kstat_t);
        (void) snprintf(name, MAXNAMELEN - 1, "%s_vpd", ddi_driver_name(dip));

        /* Get a copy of the VPD space */
        if ((rc = efx_vpd_size(enp, &size)) != 0)
                goto fail1;

        if ((vpd = kmem_zalloc(size, KM_NOSLEEP)) == NULL) {
                rc = ENOMEM;
                goto fail2;
        }

        if ((svkp->svk_vv = kmem_zalloc(sizeof (efx_vpd_value_t) *
            SFXGE_VPD_MAX, KM_NOSLEEP)) == NULL) {
                rc = ENOMEM;
                goto fail3;
        }

        if ((rc = efx_vpd_read(enp, vpd, size)) != 0)
                goto fail4;

        if ((ksp = kstat_create((char *)ddi_driver_name(dip),
            ddi_get_instance(dip), name, "vpd", KSTAT_TYPE_NAMED, SFXGE_VPD_MAX,
            KSTAT_FLAG_VIRTUAL)) == NULL) {
                rc = ENOMEM;
                goto fail5;
        }
        svkp->svk_ksp = ksp;
        ksp->ks_data = &(svkp->svk_stat);

        _sfxge_vpd_kstat_init(sp, vpd, size, EFX_VPD_ID, "ID", SFXGE_VPD_ID);
        _sfxge_vpd_kstat_init(sp, vpd, size, EFX_VPD_RO, "PN", SFXGE_VPD_PN);
        _sfxge_vpd_kstat_init(sp, vpd, size, EFX_VPD_RO, "SN", SFXGE_VPD_SN);
        _sfxge_vpd_kstat_init(sp, vpd, size, EFX_VPD_RO, "EC", SFXGE_VPD_EC);
        _sfxge_vpd_kstat_init(sp, vpd, size, EFX_VPD_RO, "VD", SFXGE_VPD_VD);

        kstat_install(ksp);
        kmem_free(vpd, size);

        return (0);

fail5:
        DTRACE_PROBE(fail5);
fail4:
        DTRACE_PROBE(fail4);
        kmem_free(svkp->svk_vv, sizeof (efx_vpd_value_t) * SFXGE_VPD_MAX);
fail3:
        DTRACE_PROBE(fail3);
        kmem_free(vpd, size);
fail2:
        DTRACE_PROBE(fail2);
fail1:
        DTRACE_PROBE1(fail1, int, rc);
        SFXGE_OBJ_CHECK(svkp, sfxge_vpd_kstat_t);

        return (rc);
```



```
}

static void
sfxge_vpd_kstat_fini(sfxge_t *sp)
{
        sfxge_vpd_kstat_t *svkp = &(sp->s_vpd_kstat);

        /* NOTE: VPD support is optional, so kstats might not be registered */
        if (svkp->svk_ksp != NULL) {

                kstat_delete(svkp->svk_ksp);

                kmem_free(svkp->svk_vv,
                    sizeof (efx_vpd_value_t) * SFXGE_VPD_MAX);

                bzero(svkp->svk_stat,
                    sizeof (kstat_named_t) * SFXGE_VPD_MAX);

                svkp->svk_ksp = NULL;
        }

        SFXGE_OBJ_CHECK(svkp, sfxge_vpd_kstat_t);
}

static int
sfxge_cfg_kstat_init(sfxge_t *sp)
{
        dev_info_t *dip = sp->s_dip;
        char name[MAXNAMELEN];
        kstat_t *ksp;
        sfxge_cfg_kstat_t *sckp;
        int rc;

        sfxge_cfg_build(sp);

        /* Create the set */
        (void) snprintf(name, MAXNAMELEN - 1, "%s_cfg", ddi_driver_name(dip));

        if ((ksp = kstat_create((char *)ddi_driver_name(dip),
            ddi_get_instance(dip), name, "cfg", KSTAT_TYPE_NAMED,
            sizeof (sckp->kstat) / sizeof (kstat_named_t),
            KSTAT_FLAG_VIRTUAL)) == NULL) {
                rc = ENOMEM;
                goto fail1;
        }

        sp->s_cfg_ksp = ksp;

        ksp->ks_data = sckp = &(sp->s_cfg_kstat);

        kstat_named_init(&(sckp->kstat.sck_mac), "mac", KSTAT_DATA_STRING);
        kstat_named_setstr(&(sckp->kstat.sck_mac), sckp->buf.sck_mac);
        ksp->ks_data_size += sizeof (sckp->buf.sck_mac);

        kstat_named_init(&(sckp->kstat.sck_version), "version",
            KSTAT_DATA_STRING);
        kstat_named_setstr(&(sckp->kstat.sck_version), sfxge_version);
        ksp->ks_data_size += sizeof (sfxge_version);

        kstat_install(ksp);
        return (0);

fail1:
        DTRACE_PROBE1(fail1, int, rc);

        return (rc);
```

```
1052 }
1054 static void
1055 sfxge_cfg_kstat_fini(sfxge_t *sp)
1056 {
1057         if (sp->s_cfg_ksp == NULL)
1058                 return;
1060         kstat_delete(sp->s_cfg_ksp);
1061         sp->s_cfg_ksp = NULL;
1063         bzero(&(sp->s_cfg_kstat), sizeof (sfxge_cfg_kstat_t));
1064 }
1066 static int
1067 sfxge_resume(dev_info_t *dip)
1068 {
1069         sfxge_t *sp = ddi_get_soft_state(sfxge_ss, ddi_get_instance(dip));
1070         int rc;
1072         /* Start processing */
1073         if ((rc = sfxge_start(sp, B_FALSE)) != 0)
1074                 goto fail1;
1076         return (DDI_SUCCESS);
1078 fail1:
1079         DTRACE_PROBE1(fail1, int, rc);
1081         return (DDI_FAILURE);
1082 }
1084 static int
1085 sfxge_attach(dev_info_t *dip, ddi_attach_cmd_t cmd)
1086 {
1087         sfxge_t *sp = ddi_get_soft_state(sfxge_ss, ddi_get_instance(dip));
1088         int rc;
1090         switch (cmd) {
1091         case DDI_ATTACH:
1092                 if (sp != NULL) {
1093                         cmn_err(CE_WARN, SFXGE_CMN_ERR
1094                             "[%s%d] ATTACH for attached instance\n",
1095                             ddi_driver_name(sp->s_dip),
1096                             ddi_get_instance(sp->s_dip));
1097                         return (DDI_FAILURE);
1098                 }
1099                 break;
1101         case DDI_RESUME:
1102                 if (sp == NULL) {
1103                         cmn_err(CE_WARN, SFXGE_CMN_ERR
1104                             "[%s%d] RESUME for missing instance\n",
1105                             ddi_driver_name(sp->s_dip),
1106                             ddi_get_instance(sp->s_dip));
1107                         return (DDI_FAILURE);
1108                 }
1109                 return (sfxge_resume(dip));
1111         default:
1112                 return (DDI_FAILURE);
1113         }
1115         /* Create the soft state */
1116         if ((rc = sfxge_create(dip, &sp)) != 0)
1117                 goto fail1;
```



```
1119         /* Create the configuration kstats */
1120         if ((rc = sfxge_cfg_kstat_init(sp)) != 0)
1121                 goto fail2;
1123         /* Create the VPD kstats */
1124         if ((rc = sfxge_vpd_kstat_init(sp)) != 0) {
1125                 if (rc != ENOTSUP)
1126                         goto fail3;
1127         }
1129         /* Register the interface */
1130         if ((rc = sfxge_register(sp)) != 0)
1131                 goto fail4;
1133         /* Announce ourselves in the system log */
1134         ddi_report_dev(dip);
1136         return (DDI_SUCCESS);
1138 fail4:
1139         DTRACE_PROBE(fail4);
1141         /* Destroy the VPD kstats */
1142         sfxge_vpd_kstat_fini(sp);
1144 fail3:
1145         DTRACE_PROBE(fail3);
1147         /* Destroy the configuration kstats */
1148         sfxge_cfg_kstat_fini(sp);
1150 fail2:
1151         DTRACE_PROBE(fail2);
1153         /* Destroy the soft state */
1154         (void) sfxge_destroy(sp);
1156 fail1:
1157         DTRACE_PROBE1(fail1, int, rc);
1159         return (DDI_FAILURE);
1160 }
1162 static int
1163 sfxge_suspend(dev_info_t *dip)
1164 {
1165         sfxge_t *sp = ddi_get_soft_state(sfxge_ss, ddi_get_instance(dip));
1167         /* Stop processing */
1168         sfxge_stop(sp);
1170         return (DDI_SUCCESS);
1171 }
1173 static int
1174 sfxge_detach(dev_info_t *dip, ddi_detach_cmd_t cmd)
1175 {
1176         sfxge_t *sp = ddi_get_soft_state(sfxge_ss, ddi_get_instance(dip));
1177         int rc;
1179         switch (cmd) {
1180         case DDI_DETACH:
1181                 if (sp == NULL) {
1182                         cmn_err(CE_WARN, SFXGE_CMN_ERR
1183                             "[%s%d] DETACH for missing instance\n",
```

```
                            ddi_driver_name(sp->s_dip),
                            ddi_get_instance(sp->s_dip));
                    return (DDI_FAILURE);
            }
            break;

        case DDI_SUSPEND:
            if (sp == NULL) {
                    cmn_err(CE_WARN, SFXGE_CMN_ERR
                        "[%s%d] SUSPEND for missing instance\n",
                        ddi_driver_name(sp->s_dip),
                        ddi_get_instance(sp->s_dip));
                    return (DDI_FAILURE);
            }
            return (sfxge_suspend(dip));

        default:
            return (DDI_FAILURE);
        }

        ASSERT(sp != NULL);

        /* Wait for any pending restarts to complete */
        ddi_taskq_wait(sp->s_tqp);

        /*
         * IOCTLs from utilites can cause GLD mc_start() (SFXGE_STARTED state)
         * And mc_stop() may not occur until detach time and race. SFC bug 19855
         * Holding the lock seems to be enough - the log message is not seen
         */
        mutex_enter(&(sp->s_state_lock));
        if (sp->s_state == SFXGE_STARTED) {
            cmn_err(CE_WARN, SFXGE_CMN_ERR
                        "[%s%d] STREAMS detach when STARTED\n",
                        ddi_driver_name(sp->s_dip),
                        ddi_get_instance(sp->s_dip));
            sfxge_stop_locked(sp);
            ASSERT3U(sp->s_state, ==, SFXGE_REGISTERED);
        }
        mutex_exit(&(sp->s_state_lock));

        ASSERT(sp->s_state == SFXGE_REGISTERED ||
            sp->s_state == SFXGE_INITIALIZED);

        if (sp->s_state != SFXGE_REGISTERED)
            goto destroy;

        /* Unregister the interface */
        if ((rc = sfxge_unregister(sp)) != 0)
            goto fail1;

destroy:
        /* Destroy the VPD kstats */
        sfxge_vpd_kstat_fini(sp);

        /* Destroy the configuration kstats */
        sfxge_cfg_kstat_fini(sp);

        /*
         * Destroy the soft state - this might fail until rx_loaned packets that
         * have been passed up the STREAMS stack are returned
         */
        if ((rc = sfxge_destroy(sp)) != 0)
            goto fail2;

        return (DDI_SUCCESS);
```



```
fail2:
        DTRACE_PROBE(fail2);
fail1:
        DTRACE_PROBE1(fail1, int, rc);

        return (DDI_FAILURE);
}

#ifdef _USE_GLD_V3
#ifndef _USE_GLD_V3_SOL10
static int
sfxge_quiesce(dev_info_t *dip)
{
        sfxge_t *sp = ddi_get_soft_state(sfxge_ss, ddi_get_instance(dip));
        int rc;

        /* Reset the hardware */
        if ((rc = sfxge_reset(sp, B_FALSE)) != 0)
                goto fail1;

        return (DDI_SUCCESS);

fail1:
        DTRACE_PROBE1(fail1, int, rc);

        return (DDI_FAILURE);
}
#endif
#endif

/*
 * modlinkage
 */
DDI_DEFINE_STREAM_OPS(sfxge_dev_ops, nulldev, nulldev, sfxge_attach,
    sfxge_detach, nulldev, NULL, D_MP, NULL, NULL);

#ifdef _USE_GLD_V2

static struct module_info       sfxge_module_info = {
        0,
        SFXGE_DRIVER_NAME,
        0,
        INFPSZ,
        1,
        0
};

static struct qinit             sfxge_rqinit = {
        NULL,
        gld_rsrv,
        gld_open,
        gld_close,
        NULL,
        &sfxge_module_info,
        NULL
};

static struct qinit             sfxge_wqinit = {
        gld_wput,
        gld_wsrv,
        NULL,
        NULL,
        NULL,
        &sfxge_module_info,
        NULL
```

```
};

static struct streamtab         sfxge_streamtab = {
        &sfxge_rqinit,
        &sfxge_wqinit,
        NULL,
        NULL
};

static struct cb_ops            sfxge_cb_ops = {
        nulldev,                /* cb_open */
        nulldev,                /* cb_close */
        nodev,                  /* cb_strategy */
        nodev,                  /* cb_print */
        nodev,                  /* cb_dump */
        nodev,                  /* cb_read */
        nodev,                  /* cb_write */
        nodev,                  /* cb_ioctl */
        nodev,                  /* cb_devmap */
        nodev,                  /* cb_mmap */
        nodev,                  /* cb_segmap */
        nochpoll,               /* cb_chpoll */
        ddi_prop_op,            /* cb_prop_op */
        &sfxge_streamtab,       /* cb_stream */
        D_MP,                   /* cb_flag */
        CB_REV,                 /* cb_rev */
        nodev,                  /* cb_aread */
        nodev,                  /* cb_awrite */
};

static struct dev_ops           sfxge_dev_ops = {
        DEVO_REV,               /* devo_rev */
        0,                      /* devo_refcnt */
        NULL,                   /* devo_getinfo */
        nulldev,                /* devo_identify */
        nulldev,                /* devo_probe */
        sfxge_attach,           /* devo_attach */
        sfxge_detach,           /* devo_detach */
        nulldev,                /* devo_reset */
        &sfxge_cb_ops,          /* devo_cb_ops */
        (struct bus_ops *)NULL, /* devo_bus_ops */
        NULL                    /* devo_power */
};

#endif

static struct modldrv           sfxge_modldrv = {
        &mod_driverops,
        (char *)sfxge_ident,
        &sfxge_dev_ops,
};

static struct modlinkage        sfxge_modlinkage = {
        MODREV_1,
        { &sfxge_modldrv, NULL }
};

kmutex_t        sfxge_global_lock;
unsigned int    *sfxge_cpu;
#ifdef  _USE_CPU_PHYSID
unsigned int    *sfxge_core;
unsigned int    *sfxge_cache;
unsigned int    *sfxge_chip;
#endif

int
```



```
_init(void)
{
        int rc;

        mutex_init(&sfxge_global_lock, NULL, MUTEX_DRIVER, NULL);

        /* Create tables for CPU, core, cache and chip counts */
        sfxge_cpu = kmem_zalloc(sizeof (unsigned int) * NCPU, KM_SLEEP);
#ifdef  _USE_CPU_PHYSID
        sfxge_core = kmem_zalloc(sizeof (unsigned int) * NCPU, KM_SLEEP);
        sfxge_cache = kmem_zalloc(sizeof (unsigned int) * NCPU, KM_SLEEP);
        sfxge_chip = kmem_zalloc(sizeof (unsigned int) * NCPU, KM_SLEEP);
#endif

        if ((rc = ddi_soft_state_init(&sfxge_ss, sizeof (sfxge_t), 0)) != 0)
                goto fail1;

#ifdef _USE_GLD_V3
        mac_init_ops(&sfxge_dev_ops, SFXGE_DRIVER_NAME);
#endif

        if ((rc = mod_install(&sfxge_modlinkage)) != 0)
                goto fail2;

        cmn_err(CE_NOTE, SFXGE_CMN_ERR
            "LOAD: Solarflare Ethernet Driver (%s) %s",
            SFXGE_DRIVER_NAME, sfxge_ident);

        return (0);

fail2:
        DTRACE_PROBE(fail2);

#ifdef _USE_GLD_V3
//      mac_fini_ops(&sfxge_dev_ops);
#endif

        ddi_soft_state_fini(&sfxge_ss);

fail1:
        DTRACE_PROBE1(fail1, int, rc);

        return (rc);
}

int
_fini(void)
{
        int rc;

        if ((rc = mod_remove(&sfxge_modlinkage)) != 0)
                return (rc);

        cmn_err(CE_NOTE, SFXGE_CMN_ERR
            "UNLOAD: Solarflare Ethernet Driver (%s) %s",
            SFXGE_DRIVER_NAME, sfxge_ident);

#ifdef _USE_GLD_V3
        mac_fini_ops(&sfxge_dev_ops);
#endif

        ddi_soft_state_fini(&sfxge_ss);

        /* Destroy tables */
#ifdef  _USE_CPU_PHYSID
        kmem_free(sfxge_chip, sizeof (unsigned int) * NCPU);
```

```
1448         kmem_free(sfxge_cache, sizeof (unsigned int) * NCPU);
1449         kmem_free(sfxge_core, sizeof (unsigned int) * NCPU);
1450 #endif
1451         kmem_free(sfxge_cpu, sizeof (unsigned int) * NCPU);

1453         mutex_destroy(&sfxge_global_lock);

1455         return (0);
1456 }

1458 int
1459 _info(struct modinfo *mip)
1460 {
1461         return (mod_info(&sfxge_modlinkage, mip));
1462 }
1463 #endif /* ! codereview */
```

```
**********************************************************
   11214 Thu Aug 22 18:59:27 2013
new/usr/src/uts/common/io/sfxge/sfxge.conf
Merged sfxge driver
**********************************************************
#### This file contains configuration parameters for the Solarflare
#### network adapter driver "sfxge"

################################################################################
#### HOW TO MAKE CHANGES TO THIS FILE TAKE EFFECT
################################################################################
#### In order for changes made to this file to be used by the driver either:
#### i) reboot
#### OR
#### ii) Unplumb all sfxge inet,inet6 interfaces or link aggregations: e.g.
####       $ ifconfig sfxge0 unplumb
####       $ ifconfig sfxge0 inet6 unplumb
####     Delete any link aggregations that setup (see dladm show-aggr):
####       $ dladm delete-aggr <key>
####     Update the driver config
####       $ update_drv sfxge
####     Note update_drv will unload then reload the driver, otherwise it warns:
####       "Cannot unload module: sfxge"
####     Then plumb interfaces and reconfigure IP addresses as required
####
################################################################################

################################################################################
#### PER PORT SETTINGS
################################################################################
#### Note that is possible to have different settings per network port
#### First using the file "/etc/path_to_inst" you can identify some information
####
####  $ grep sfxge /etc/path_to_inst
####   # "/pci@3a,0/pci8086,340c@5/pci1924,6201@0"   4 "sfxge"
####   # "/pci@3a,0/pci8086,340c@5/pci1924,6201@0,1" 5 "sfxge"
####   #  ^^^^^^^^^^^^^^^^^^^^^^^^             ^^^
####   #         parent                  unit-address
####
#### Using the information gathered above to create entries in this file
#### based on the example below; this tries to allocate 4 MSI-X interrupts
#### to port0 and 8 MSI-X interrupts to port 1
####
####  name="sfxge" parent="/pci@3a,0/pci8086,340c@5" unit-address="0"
####               rx_scale_count=4;
####  name="sfxge" parent="/pci@3a,0/pci8086,340c@5" unit-address="0,1"
####               rx_scale_count=8;
################################################################################

################################################################################
#### RECEIVE SIDE SCALING
################################################################################
#### Maximum number of RSS-channels to use per port (default 0, global max 32)
#### This spreads RX flows across CPUs using multiple MSI-X interrupts
####
#### - A value of 0 tries to allocate the same number of RSS-channels as logical
####   CPUs in order to provide the best spreading accross the entire system.
####
#### - actual number may be lower due to availability of MSI-X interrupts
####
#### - to successfully use >2 MSI-X interrupts per port you also need to use
####  "set ddi_msix_alloc_limit=8" (maximum) in /etc/system and reboot
####   OR temporarily adjust this limit using mdb and then reload the driver
####     # mdb -kw
####    > ddi_msix_alloc_limit/W 8
####    ddi_msix_alloc_limit:          0x2            =       0x8
```



```
####    > ddi_msix_alloc_limit/X
####    ddi_msix_alloc_limit:
####    ddi_msix_alloc_limit:          8
####
#### - Note that the total MSI-X interrupt vector assigned to all network
####     drivers (in IPL 6; interrupt priority level) is 32
####   See the "INTERUPT PRIORITY" section for a way overcome this
####
####  -  You can see the current allocation with this run as root:
####   $ echo "::interrupts -d" | mdb -k | egrep 'IRQ| 6 '
####     IRQ  Vector IPL Bus   Type  CPU Share APIC/INT# Driver Name(s)
####     93   0x60   6         MSI-X 2   1     -         sfxge#0
####     94   0x61   6         MSI-X 3   1     -         sfxge#0
####                ^^^
####
#### - If the sysadmin does not take care to partition the interupts between
####     the network ports then later ports could use a single legacy interrupt
####     i.e. no RSS, significantly reducing performance
####   See PER PORT SETTINGS ABOVE for how to apply rx_scale_count per port
####
#### - rx_scale_count can be manipulated dynamically via ndd but the number
####   of interrupts will stay the same; though only a subset will fire for RX.
####   All interrupts will still see ~10 interrupts/sec
####
#### - note that the number of RSS-channels can be set to be greater than the
####   number of logical CPUs. This can be useful to prevent some types of RX
####   drops

rx_scale_count=0;

################################################################################
#### USING NON_DEFAULT INTERRUPT PRIORITY LEVEL (IPL)
################################################################################
#### To overcome the Solaris limitation of 32 MSI-X vectors per IPL
#### it is possible to set this driver to allocate interrupts in a different IPL
#### Please only do this if necessary
####
#### See http://download.oracle.com/docs/cd/E20815_01/html/E20816/gjtag.html
#### Verify with:
####   $ echo "::interrupts -d" | mdb -k | egrep 'IRQ|sfxge'
####     IRQ  Vector IPL Bus   Type  CPU Share APIC/INT# Driver Name(s)
####     93   0x60   5         MSI-X 2   1     -         sfxge#0
####     94   0x61   5         MSI-X 3   1     -         sfxge#0
####                ^^^
####
#### (default IPL 6; the document referenced recommends 5 or 4)

#interrupt-priorities=5;

################################################################################
#### RX DMA RING SIZE
################################################################################
#### Number of entries in the per receive queue hardware DMA ring
####
#### Increase this value if you see increasing values of "rx_nodesc_drop_cnt"
#### as reported by "kstat -m sfxge -c mac -s rx_nodesc_drop_cnt" and you think
#### the traffic is bursty
####
#### Increasing this value can increase driver memory usage and reduce the
#### cache-hit rates seen for RX buffers so the largest value is not
#### necessarily the best
####
#### (minimum 512)
#### (default 1024)
#### (maximum 4096)
#### The value must be a power of two else it will be ignored.
```

```
#rxq_size=1024;

################################################################################
#### LARGE RECEIVE OFFLOAD
################################################################################
#### Coalesce RX packets (aka Large Receive Offload)
####  0 => off (default)
####  1 => on
####  2 => on, respecting TCP PSH boundaries

#rx_coalesce_mode=1;

################################################################################
#### INTERRUPT MODERATION
################################################################################
#### Interrupt moderation in us (default 30, minimum 0)
#### Decreasing this reduces latency but increases interrupt rate and therefore
#### CPU usage which will decrease maximum bandwidth achievable
####
#### Decrease this value if you see increasing values of "rx_nodesc_drop_cnt"
#### as reported by "kstat -m sfxge -c mac -s rx_nodesc_drop_cnt" so that
#### the driver has more opportunities to refill the hardware RX descriptor
#### ring
####
#### (maximum 20000 us)
#### (default 30 us)

#intr_moderation=10;

################################################################################
#### MAXIMUM MTU
################################################################################
#### Maximum MTU of an sfxge interface (in bytes - excludes ethernet framing)
#### Increase this value to support Jumbo frames
####
#### NB increasing this value consumes more memory for all RX buffers
#### even for network interfaces not configured with high MTUs
####
#### MTUs over 3968 mean that >1 contiguous 4k page of memory are allocated
#### for all receive buffers. Contiguous pages of memory can be harder for
#### the OS to allocate when under memory pressue.
####
#### In Solaris 11 the MTU can be dynamically changed so you should not
#### need to alter the setting below. A non-default MTU can be persistently
#### configured depending on your network configuration:
#### 1) For manual mode or using NWAM with the DefaultFixed profile:
####    dladm set-linkprop -p mtu=<MTU> sfxge<n>
#### 2) If using NWAM using a custom profile please set the property link-mtu:
####    netcfg "select ncp myprofile; select ncu phys sfxge0; set link-mtu=9000"
####
#### (maximum 9000)
#### (default 1500)

#mtu=1500;

################################################################################
#### PER PORT MEMORY LIMIT FOR RX PACKET BUFFERS
################################################################################
#### Limit max memory for RX packets allocated per instance (port) in bytes
####
#### RX packets are passed up from the driver to the kernel TCP/IP stack
#### and freed after the data is delivered to the application socket
#### buffers. If the OS falls behind this allocation can grow.
####
#### This parameter can help make behaviour reasonable when approaching
```



```
#### an overload condition. Other ways to avoid overload would be RSS
####
#### The size of each bufer, number in use can be seen with
####    echo "::kmastat" | mdb -k
#### OR kstat -c kmem_cache -n sfxgeX_rx_packet_cache # replace X
####
#### (default unlimited when unset)
####
#### Note In Solarflare driver package versions 3.1.x.xxxx and earlier set
#### the RX buffering limit to 10773741824 (1GB) per interface in this file.
####
#### 64MB of RX buffering per interface is still oversized for nearly all
#### applications but provides a hard limit.

rx_pkt_mem_max=67108864;

################################################################################
#### PREALLOCATION OF RX PACKET BUFFERS
################################################################################
#### Number of rx packet buffers to allocate at start of an rxq and keep a
#### free packet pool of atleast this many.
####
#### Keeping a free packet pool of rx packet buffers means we do not need to
#### repeatedly allocate and map dma buffers.
####
#### (minimum 0 => off)
#### (default 0 => off)
#### (maximum is limited by available memory)

#rx_prealloc_pkt_buffers=512;

################################################################################
#### ACTION ON HARDWARE ERRORS
################################################################################
#### This parameter controls the action taken on a hardware error
#### which may be a PCIe error or the driver detecting unexpected behaviour
#### from the hardware
####
#### Currently this driver does not reports error into the Solaris fault
#### management architecture (but the PCIe root-port may do so)
####
#### 0 => recover the server adapter to a working state
#### 1 => do not advertise to the kernel that the link is down during the reset
#### 2 => reset the hardware, but do not attempt to use it again.
####      this is useful if you have a failover mechanism, and want to ensure
####      that this server adapter does not become the active link again
####      note that the interface will stay plumbed but will not pass traffic
####
#### All of the above log a message and increment a kstat counter viewable with:
####    kstat -m sfxge -c mon -s num_restarts_hw_err
####
#### (default 0 => recover)

action_on_hw_err=0;

#endif /* ! codereview */
```

```
**********************************************************
   29948 Thu Aug 22 18:59:27 2013
new/usr/src/uts/common/io/sfxge/sfxge.h
Merged sfxge driver
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2008-2013 Solarflare Communications Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * Solaris 10 9/10 (U9) is required to build this driver, as earlier Solaris
 * releases do not ship with the required header files for GLDv3.  The driver
 * will run on Solaris 10 10/09 (U8) and later versions.
 */


#ifndef _SYS_SFXGE_H
#define _SYS_SFXGE_H

#ifdef __cplusplus
extern "C" {
#endif

#include <sys/types.h>
#include <sys/ddi.h>
#include <sys/sunddi.h>
#include <sys/stream.h>
#include <sys/ethernet.h>
#include <sys/cpuvar.h>

#ifdef _USE_GLD_V3
#include <sys/mac.h>
#include <sys/mac_ether.h>
#include <sys/mac_provider.h>
#endif

#ifdef _USE_GLD_V2
#include <sys/gld.h>
#endif

#include "sfxge_ioc.h"
#include "sfxge_debug.h"

#include "efx.h"
```



```
#include "efx_regs.h"

#ifdef _USE_GLD_V3_SOL10
#include "compat.h"
#endif

#if defined(_USE_MAC_PRIV_PROP) && !defined(_USE_GLD_V3_PROPS)
#error "The _USE_MAC_PRIV_PROP build option is dependent on _USE_GLD_V3_PROPS"
#endif

#ifdef  _KERNEL

#define SFXGE_DRIVER_NAME "sfxge"

#define SFXGE_CPU_CACHE_SIZE    64

#define IS_POW2(x)  ((x) && ! ((x) & ((x) - 1)))

typedef struct sfxge_s  sfxge_t;

typedef enum sfxge_intr_state_e {
        SFXGE_INTR_UNINITIALIZED = 0,
        SFXGE_INTR_INITIALIZED,
        SFXGE_INTR_TESTING,
        SFXGE_INTR_STARTED
} sfxge_intr_state_t;

typedef struct sfxge_intr_s {
        ddi_intr_handle_t       *si_table;
        int                     si_table_size;
        int                     si_nalloc;
        int                     si_type;
        int                     si_cap;
        efsys_mem_t             si_mem;
        uint64_t                si_mask;
        sfxge_intr_state_t      si_state;
        uint32_t                si_zero_count;
        int                     si_intr_pri;
} sfxge_intr_t;

typedef enum sfxge_promisc_type_e {
        SFXGE_PROMISC_OFF = 0,
        SFXGE_PROMISC_ALL_MULTI,
        SFXGE_PROMISC_ALL_PHYS
} sfxge_promisc_type_t;

typedef enum sfxge_link_duplex_e {
        SFXGE_LINK_DUPLEX_UNKNOWN = 0,
        SFXGE_LINK_DUPLEX_HALF,
        SFXGE_LINK_DUPLEX_FULL
} sfxge_link_duplex_t;

typedef enum sfxge_unicst_type_e {
        SFXGE_UNICST_BIA = 0,
        SFXGE_UNICST_LAA,
        SFXGE_UNICST_NTYPES
} sfxge_unicst_type_t;

typedef struct sfxge_phy_s {
                kstat_t                 *sp_ksp;
                kstat_named_t           *sp_stat;
                uint32_t                *sp_statbuf;
                efsys_mem_t             sp_mem;
} sfxge_phy_t;

typedef enum sfxge_mac_state_e {
```

```
128         SFXGE_MAC_UNINITIALIZED = 0,
129         SFXGE_MAC_INITIALIZED,
130         SFXGE_MAC_STARTED
131 } sfxge_mac_state_t;

133 typedef struct sfxge_mac_s {
134         sfxge_t                 *sm_sp;
135         efsys_mem_t             sm_mem;
136         kstat_t                 *sm_ksp;
137         kstat_named_t           *sm_stat;
138         uint8_t                 sm_bia[ETHERADDRL];
139         uint8_t                 sm_laa[ETHERADDRL];
140         boolean_t               sm_laa_valid;
141         unsigned int            sm_fcntl;
142         sfxge_promisc_type_t    sm_promisc;
143         unsigned int            sm_bucket[EFX_MAC_HASH_BITS];
144         clock_t                 sm_lbolt;
145         kmutex_t                sm_lock;
146         efx_link_mode_t         sm_link_mode;
147         unsigned int            sm_link_speed;
148         sfxge_link_duplex_t     sm_link_duplex;
149         boolean_t               sm_link_up;
150         boolean_t               sm_link_poll_reqd;
151         kcondvar_t              sm_link_poll_kv;
152         boolean_t               sm_mac_stats_timer_reqd;
153         boolean_t               sm_mac_stats_pend;
154         ddi_taskq_t             *sm_tqp;
155         sfxge_mac_state_t       sm_state;
156         sfxge_phy_t             sm_phy;
157         uint32_t                sm_phy_cap_to_set;
158         uint32_t                sm_phy_cap_to_unset;
159 } sfxge_mac_t;

161 typedef enum sfxge_mon_state_e {
162         SFXGE_MON_UNINITIALIZED = 0,
163         SFXGE_MON_INITIALIZED,
164         SFXGE_MON_STARTED
165 } sfxge_mon_state_t;

167 typedef struct sfxge_mon_s {
168         sfxge_t                 *sm_sp;
169         efx_mon_type_t          sm_type;
170         unsigned int            sm_devid;
171         kstat_t                 *sm_ksp;
172         kstat_named_t           *sm_stat;
173         efx_mon_stat_value_t    *sm_statbuf;
174         kmutex_t                sm_lock;
175         sfxge_mon_state_t       sm_state;
176         efsys_mem_t             sm_mem;
177 } sfxge_mon_t;

179 typedef enum sfxge_sram_state_e {
180         SFXGE_SRAM_UNINITIALIZED = 0,
181         SFXGE_SRAM_INITIALIZED,
182         SFXGE_SRAM_STARTED
183 } sfxge_sram_state_t;

185 typedef struct sfxge_sram_s {
186         sfxge_t                 *ss_sp;
187         kmutex_t                ss_lock;
188         vmem_t                  *ss_buf_tbl;
189         unsigned int            ss_count;
190         sfxge_sram_state_t      ss_state;
191 } sfxge_sram_t;

193 typedef enum sfxge_mcdi_state_e {
```



```
194         SFXGE_MCDI_UNINITIALIZED = 0,
195         SFXGE_MCDI_INITIALIZED,
196         SFXGE_MCDI_BUSY,
197         SFXGE_MCDI_COMPLETED
198 } sfxge_mcdi_state_t;

200 typedef struct sfxge_mcdi_s {
201         sfxge_t                 *sm_sp;
202         kmutex_t                sm_lock;
203         sfxge_mcdi_state_t      sm_state;
204         efx_mcdi_transport_t    sm_emt;
205         kcondvar_t              sm_kv;          /* MCDI poll complete */
206 } sfxge_mcdi_t;

208 #define SFXGE_NEVS      4096
209 #define SFXGE_NDESCS    1024
210 #define SFXGE_TX_NDESCS SFXGE_NDESCS
211 #define SFXGE_DEFAULT_RXQ_SIZE 1024

213 #define SFXGE_DEFAULT_MODERATION        30

215 typedef enum sfxge_evq_state_e {
216         SFXGE_EVQ_UNINITIALIZED = 0,
217         SFXGE_EVQ_INITIALIZED,
218         SFXGE_EVQ_STARTING,
219         SFXGE_EVQ_STARTED
220 } sfxge_evq_state_t;

222 #define SFXGE_EV_BATCH  (SFXGE_NEVS / 4)

224 typedef struct sfxge_txq_s      sfxge_txq_t;

226 typedef struct sfxge_evq_s {
227         union {
228                 struct {
229                         sfxge_t                 *__se_sp;
230                         unsigned int            __se_index;
231                         efsys_mem_t             __se_mem;
232                         unsigned int            __se_id;
233                         kstat_t                 *__se_ksp;
234                         kstat_named_t           *__se_stat;
235                         efx_ev_callbacks_t      __se_eec;
236                         sfxge_evq_state_t       __se_state;
237                         boolean_t               __se_exception;
238                 } __se_s1;
239                 uint8_t __se_pad[SFXGE_CPU_CACHE_SIZE * 4];
240         } __se_u1;
241         union {
242                 struct {
243                         kmutex_t                __se_lock;
244                         kcondvar_t              __se_init_kv;
245                         efx_evq_t               *__se_eep;
246                         unsigned int            __se_count;
247                         unsigned int            __se_rx;
248                         unsigned int            __se_tx;
249                         sfxge_txq_t             *__se_stp;
250                         sfxge_txq_t             **__se_stpp;
251                         processorid_t           __se_cpu_id;
252 #ifdef  _USE_CPU_PHYSID
253                         id_t                    __se_chip_id;
254                         id_t                    __se_core_id;
255                         id_t                    __se_cache_id;
256 #endif
257                         uint16_t                __se_ev_batch;
258                 } __se_s2;
259                 uint8_t __se_pad[SFXGE_CPU_CACHE_SIZE];
```

```
260         } __se_u2;
261 } sfxge_evq_t;

263 #define se_sp           __se_u1.__se_s1.__se_sp
264 #define se_index        __se_u1.__se_s1.__se_index
265 #define se_mem          __se_u1.__se_s1.__se_mem
266 #define se_id           __se_u1.__se_s1.__se_id
267 #define se_ksp          __se_u1.__se_s1.__se_ksp
268 #define se_stat         __se_u1.__se_s1.__se_stat
269 #define se_eec          __se_u1.__se_s1.__se_eec
270 #define se_state        __se_u1.__se_s1.__se_state
271 #define se_exception    __se_u1.__se_s1.__se_exception

273 #define se_lock         __se_u2.__se_s2.__se_lock
274 #define se_init_kv      __se_u2.__se_s2.__se_init_kv
275 #define se_eep          __se_u2.__se_s2.__se_eep
276 #define se_count        __se_u2.__se_s2.__se_count
277 #define se_rx           __se_u2.__se_s2.__se_rx
278 #define se_tx           __se_u2.__se_s2.__se_tx
279 #define se_stp          __se_u2.__se_s2.__se_stp
280 #define se_stpp         __se_u2.__se_s2.__se_stpp
281 #define se_cpu_id       __se_u2.__se_s2.__se_cpu_id
282 #ifdef  _USE_CPU_PHYSID
283 #define se_chip_id      __se_u2.__se_s2.__se_chip_id
284 #define se_core_id      __se_u2.__se_s2.__se_core_id
285 #define se_cache_id     __se_u2.__se_s2.__se_cache_id
286 #endif
287 #define se_ev_batch     __se_u2.__se_s2.__se_ev_batch

289 #define SFXGE_MAGIC_RESERVED    0x8000

291 #define SFXGE_MAGIC_DMAQ_LABEL_WIDTH  5
292 #define SFXGE_MAGIC_DMAQ_LABEL_MASK   ((1 << SFXGE_MAGIC_DMAQ_LABEL_WIDTH) - 1)

294 #define SFXGE_MAGIC_RX_QFLUSH_DONE                                      \
295         (SFXGE_MAGIC_RESERVED | (1 << SFXGE_MAGIC_DMAQ_LABEL_WIDTH))

297 #define SFXGE_MAGIC_RX_QFLUSH_FAILED                                    \
298         (SFXGE_MAGIC_RESERVED | (2 << SFXGE_MAGIC_DMAQ_LABEL_WIDTH))

300 #define SFXGE_MAGIC_RX_QFPP_TRIM                                        \
301         (SFXGE_MAGIC_RESERVED | (3 << SFXGE_MAGIC_DMAQ_LABEL_WIDTH))

303 #define SFXGE_MAGIC_TX_QFLUSH_DONE                                      \
304         (SFXGE_MAGIC_RESERVED | (4 << SFXGE_MAGIC_DMAQ_LABEL_WIDTH))

306 typedef struct sfxge_rxq_s              sfxge_rxq_t;

308 #define SFXGE_IP_ALIGN  2

310 #define SFXGE_ETHERTYPE_LOOPBACK        0x9000  /* Xerox loopback */

312 typedef struct sfxge_rx_packet_s        sfxge_rx_packet_t;

314 struct sfxge_rx_packet_s {
315         union {
316                 struct {
317                         frtn_t                  __srp_free;
318                         uint16_t                __srp_flags;
319                         uint16_t                __srp_size;
320                         mblk_t                  *__srp_mp;
321                         struct ether_header     *__srp_etherhp;
322                         struct ip               *__srp_iphp;
323                         struct tcphdr           *__srp_thp;
324                         size_t                  __srp_off;
325                 } __srp_s1;
```



```
326                 uint8_t __srp_pad[SFXGE_CPU_CACHE_SIZE];
327         } __srp_u1;
328         union {
329                 struct {
330                         sfxge_rxq_t             *__srp_srp;
331                         ddi_dma_handle_t        __srp_dma_handle;
332                         ddi_acc_handle_t        __srp_acc_handle;
333                         unsigned char           *__srp_base;
334                         size_t                  __srp_mblksize;
335                         uint64_t                __srp_addr;
336                         boolean_t               __srp_recycle;
337                         caddr_t                 __srp_putp;
338                 } __srp_s2;
339                 uint8_t __srp_pad[SFXGE_CPU_CACHE_SIZE * 2];
340         } __srp_u2;
341 };

343 #define srp_free        __srp_u1.__srp_s1.__srp_free
344 #define srp_flags       __srp_u1.__srp_s1.__srp_flags
345 #define srp_size        __srp_u1.__srp_s1.__srp_size
346 #define srp_mp          __srp_u1.__srp_s1.__srp_mp
347 #define srp_etherhp     __srp_u1.__srp_s1.__srp_etherhp
348 #define srp_iphp        __srp_u1.__srp_s1.__srp_iphp
349 #define srp_thp         __srp_u1.__srp_s1.__srp_thp
350 #define srp_off         __srp_u1.__srp_s1.__srp_off

352 #define srp_srp         __srp_u2.__srp_s2.__srp_srp
353 #define srp_dma_handle  __srp_u2.__srp_s2.__srp_dma_handle
354 #define srp_acc_handle  __srp_u2.__srp_s2.__srp_acc_handle
355 #define srp_base        __srp_u2.__srp_s2.__srp_base
356 #define srp_mblksize    __srp_u2.__srp_s2.__srp_mblksize
357 #define srp_addr        __srp_u2.__srp_s2.__srp_addr
358 #define srp_recycle     __srp_u2.__srp_s2.__srp_recycle
359 #define srp_putp        __srp_u2.__srp_s2.__srp_putp

361 #define SFXGE_RX_FPP_NSLOTS     8
362 #define SFXGE_RX_FPP_MASK       (SFXGE_RX_FPP_NSLOTS - 1)

364 /* Free packet pool putlist (dynamically allocated) */
365 typedef struct sfxge_rx_fpp_putlist_s {
366         kmutex_t                srfpl_lock;
367         unsigned int            srfpl_count;
368         mblk_t                  *srfpl_putp;
369         mblk_t                  **srfpl_putpp;
370 } sfxge_rx_fpp_putlist_t;

372 /* Free packet pool */
373 typedef struct sfxge_rx_fpp_s {
374         caddr_t         srfpp_putp;
375         unsigned int    srfpp_loaned;
376         mblk_t          *srfpp_get;
377         unsigned int    srfpp_count;
378         unsigned int    srfpp_min;
379         /* Low water mark: Don't trim to below this */
380         unsigned int    srfpp_lowat;
381 } sfxge_rx_fpp_t;

383 typedef struct sfxge_rx_flow_s  sfxge_rx_flow_t;

385 struct sfxge_rx_flow_s {
386         uint32_t                srf_tag;
387         /* in-order segment count */
388         unsigned int            srf_count;
389         uint16_t                srf_tci;
390         uint32_t                srf_saddr;
391         uint32_t                srf_daddr;
```

```
392         uint16_t                srf_sport;
393         uint16_t                srf_dport;
394         /* sequence number */
395         uint32_t                srf_seq;
396         clock_t                 srf_lbolt;
397         mblk_t                  *srf_mp;
398         mblk_t                  **srf_mpp;
399         struct ether_header     *srf_etherhp;
400         struct ip               *srf_iphp;
401         struct tcphdr           *srf_first_thp;
402         struct tcphdr           *srf_last_thp;
403         size_t                  srf_len;
404         sfxge_rx_flow_t         *srf_next;
405 };

407 #define SFXGE_MAX_FLOW          1024
408 #define SFXGE_SLOW_START        20

410 typedef enum sfxge_flush_state_e {
411         SFXGE_FLUSH_INACTIVE = 0,
412         SFXGE_FLUSH_DONE,
413         SFXGE_FLUSH_PENDING,
414         SFXGE_FLUSH_FAILED
415 } sfxge_flush_state_t;

417 typedef enum sfxge_rxq_state_e {
418         SFXGE_RXQ_UNINITIALIZED = 0,
419         SFXGE_RXQ_INITIALIZED,
420         SFXGE_RXQ_STARTED
421 } sfxge_rxq_state_t;


424 #define SFXGE_RX_BATCH  128
425 #define SFXGE_RX_NSTATS 8 /* note that *esballoc share one kstat */

427 struct sfxge_rxq_s {
428         union {
429                 struct {
430                         sfxge_t                         *__sr_sp;
431                         unsigned int                    __sr_index;
432                         efsys_mem_t                     __sr_mem;
433                         unsigned int                    __sr_id;
434                         unsigned int                    __sr_lowat;
435                         unsigned int                    __sr_hiwat;
436                         volatile timeout_id_t           __sr_tid;
437                         sfxge_rxq_state_t               __sr_state;
438                 } __sr_s1;
439                 uint8_t __sr_pad[SFXGE_CPU_CACHE_SIZE * 2];
440         } __sr_u1;
441         union {
442                 struct {
443                         sfxge_rx_packet_t               **__sr_srpp;
444                         unsigned int                    __sr_added;
445                         unsigned int                    __sr_pending;
446                         unsigned int                    __sr_completed;
447                         unsigned int                    __sr_loopback;
448                         mblk_t                          *__sr_mp;
449                         mblk_t                          **__sr_mpp;
450                         sfxge_rx_flow_t                 *__sr_flow;
451                         sfxge_rx_flow_t                 *__sr_srfp;
452                         sfxge_rx_flow_t                 **__sr_srfpp;
453                         clock_t                         __sr_rto;
454                 } __sr_s2;
455                 uint8_t __sr_pad[SFXGE_CPU_CACHE_SIZE * 2];
456         } __sr_u2;
457         union {
```



```
458                 struct {
459                         sfxge_rx_fpp_t                  __sr_fpp;
460                         efx_rxq_t                       *__sr_erp;
461                         volatile sfxge_flush_state_t    __sr_flush;
462                         kcondvar_t                      __sr_flush_kv;
463                         kstat_t                         *__sr_ksp;
464                 } __sr_s3;
465                 uint8_t __sr_pad[SFXGE_CPU_CACHE_SIZE];
466         } __sr_u3;
467         struct {
468                 /* NB must match SFXGE_RX_NSTATS */
469                 uint32_t    srk_rx_pkt_mem_limit;
470                 uint32_t    srk_kcache_alloc_nomem;
471                 uint32_t    srk_dma_alloc_nomem;
472                 uint32_t    srk_dma_alloc_fail;
473                 uint32_t    srk_dma_bind_nomem;
474                 uint32_t    srk_dma_bind_fail;
475                 /* Following two are mutually exclusive */
476 #ifdef _USE_XESBALLOC
477                 uint32_t    srk_xesballoc_fail;
478 #endif
479 #ifdef _USE_DESBALLOC
480                 uint32_t    srk_desballoc_fail;
481 #endif
482                 uint32_t    srk_rxq_empty_discard;
483         } sr_kstat;
484 };

486 #define sr_sp           __sr_u1.__sr_s1.__sr_sp
487 #define sr_index        __sr_u1.__sr_s1.__sr_index
488 #define sr_mem          __sr_u1.__sr_s1.__sr_mem
489 #define sr_id           __sr_u1.__sr_s1.__sr_id
490 #define sr_mrh          __sr_u1.__sr_s1.__sr_mrh
491 #define sr_lowat        __sr_u1.__sr_s1.__sr_lowat
492 #define sr_hiwat        __sr_u1.__sr_s1.__sr_hiwat
493 #define sr_tid          __sr_u1.__sr_s1.__sr_tid
494 #define sr_state        __sr_u1.__sr_s1.__sr_state

496 #define sr_srpp         __sr_u2.__sr_s2.__sr_srpp
497 #define sr_added        __sr_u2.__sr_s2.__sr_added
498 #define sr_pending      __sr_u2.__sr_s2.__sr_pending
499 #define sr_completed    __sr_u2.__sr_s2.__sr_completed
500 #define sr_loopback     __sr_u2.__sr_s2.__sr_loopback
501 #define sr_mp           __sr_u2.__sr_s2.__sr_mp
502 #define sr_mpp          __sr_u2.__sr_s2.__sr_mpp
503 #define sr_flow         __sr_u2.__sr_s2.__sr_flow
504 #define sr_srfp         __sr_u2.__sr_s2.__sr_srfp
505 #define sr_srfpp        __sr_u2.__sr_s2.__sr_srfpp
506 #define sr_rto          __sr_u2.__sr_s2.__sr_rto

508 #define sr_fpp          __sr_u3.__sr_s3.__sr_fpp
509 #define sr_erp          __sr_u3.__sr_s3.__sr_erp
510 #define sr_flush        __sr_u3.__sr_s3.__sr_flush
511 #define sr_flush_kv     __sr_u3.__sr_s3.__sr_flush_kv
512 #define sr_ksp          __sr_u3.__sr_s3.__sr_ksp

514 typedef struct sfxge_tx_packet_s        sfxge_tx_packet_t;

516 struct sfxge_tx_packet_s {
517         sfxge_tx_packet_t       *stp_next;
518         mblk_t                  *stp_mp;
519         struct ether_header     *stp_etherhp;
520         struct ip               *stp_iphp;
521         struct tcphdr           *stp_thp;
522         size_t                  stp_off;
523         size_t                  stp_size;
```

```
524         size_t                  stp_mss;
525         uint32_t                stp_dpl_put_len;
526 };

528 #define SFXGE_TX_FPP_MAX        64

530 typedef struct sfxge_tx_fpp_s {
531         sfxge_tx_packet_t       *stf_stpp;
532         unsigned int            stf_count;
533 } sfxge_tx_fpp_t;

535 typedef struct sfxge_tx_mapping_s       sfxge_tx_mapping_t;

537 #define SFXGE_TX_MAPPING_NADDR  (((1 << 16) >> 12) + 2)

539 struct sfxge_tx_mapping_s {
540         sfxge_tx_mapping_t      *stm_next;
541         sfxge_t                 *stm_sp;
542         mblk_t                  *stm_mp;
543         ddi_dma_handle_t        stm_dma_handle;
544         caddr_t                 stm_base;
545         size_t                  stm_size;
546         size_t                  stm_off;
547         uint64_t                stm_addr[SFXGE_TX_MAPPING_NADDR];
548 };

550 typedef struct sfxge_tx_fmp_s {
551         sfxge_tx_mapping_t      *stf_stmp;
552         unsigned int            stf_count;
553 } sfxge_tx_fmp_t;

555 typedef struct sfxge_tx_buffer_s        sfxge_tx_buffer_t;

557 struct sfxge_tx_buffer_s {
558         sfxge_tx_buffer_t       *stb_next;
559         size_t                  stb_off;
560         efsys_mem_t             stb_esm;
561 };

563 #define SFXGE_TX_BUFFER_SIZE    0x400
564 #define SFXGE_TX_HEADER_SIZE    0x100
565 #define SFXGE_TX_COPY_THRESHOLD 0x200

567 typedef struct sfxge_tx_fbp_s {
568         sfxge_tx_buffer_t       *stf_stbp;
569         unsigned int            stf_count;
570 } sfxge_tx_fbp_t;

572 typedef struct sfxge_tx_dpl_s {
573         uintptr_t               std_put;
574         sfxge_tx_packet_t       *std_get;
575         sfxge_tx_packet_t       **std_getp;
576         unsigned int            std_count; /* only get list count */
577         unsigned int            get_pkt_limit;
578         unsigned int            put_pkt_limit;
579         unsigned int            get_full_count;
580         unsigned int            put_full_count;
581 } sfxge_tx_dpl_t;

583 typedef enum sfxge_txq_state_e {
584         SFXGE_TXQ_UNINITIALIZED = 0,
585         SFXGE_TXQ_INITIALIZED,
586         SFXGE_TXQ_STARTED
587 } sfxge_txq_state_t;

589 typedef enum sfxge_txq_type_e {
```



```
590         SFXGE_TXQ_NON_CKSUM = 0,
591         SFXGE_TXQ_IP_CKSUM,
592         SFXGE_TXQ_IP_TCP_UDP_CKSUM,
593         SFXGE_TXQ_NTYPES
594 } sfxge_txq_type_t;

596 #define SFXGE_TXQ_UNBLOCK_LEVEL1        (EFX_TXQ_LIMIT(SFXGE_NDESCS) / 4)
597 #define SFXGE_TXQ_UNBLOCK_LEVEL2        0
598 #define SFXGE_TXQ_NOT_BLOCKED           -1

600 #define SFXGE_TX_BATCH  64

602 struct sfxge_txq_s {
603         union {
604                 struct {
605                         sfxge_t                         *__st_sp;
606                         unsigned int                    __st_index;
607                         sfxge_txq_type_t                __st_type;
608                         unsigned int                    __st_evq;
609                         efsys_mem_t                     __st_mem;
610                         unsigned int                    __st_id;
611                         kstat_t                         *__st_ksp;
612                         kstat_named_t                   *__st_stat;
613                         sfxge_txq_state_t               __st_state;
614                 } __st_s1;
615                 uint8_t __st_pad[SFXGE_CPU_CACHE_SIZE * 2];
616         } __st_u1;
617         union {
618                 struct {
619                         sfxge_tx_dpl_t                  __st_dpl;
620                 } __st_s2;
621                 uint8_t __st_pad[SFXGE_CPU_CACHE_SIZE];
622         } __st_u2;
623         union {
624                 struct {
625                         kmutex_t                        __st_lock;
626                         /* mapping pool - sfxge_tx_mapping_t */
627                         sfxge_tx_fmp_t                  __st_fmp;
628                         /* buffer pool - sfxge_tx_buffer_t */
629                         sfxge_tx_fbp_t                  __st_fbp;
630                         /* packet pool - sfxge_tx_packet_t */
631                         sfxge_tx_fpp_t                  __st_fpp;
632                         efx_buffer_t                    *__st_eb;
633                         unsigned int                    __st_n;
634                         efx_txq_t                       *__st_etp;
635                         sfxge_tx_mapping_t              **__st_stmp;
636                         sfxge_tx_buffer_t               **__st_stbp;
637                         mblk_t                          **__st_mp;
638                         unsigned int                    __st_added;
639                         unsigned int                    __st_reaped;
640                         int                             __st_unblock;
641                 } __st_s3;
642                 uint8_t __st_pad[SFXGE_CPU_CACHE_SIZE * 3];
643         } __st_u3;
644         union {
645                 struct {
646                         sfxge_txq_t                     *__st_next;
647                         unsigned int                    __st_pending;
648                         unsigned int                    __st_completed;
649                         volatile sfxge_flush_state_t    __st_flush;

651                 } __st_s4;
652                 uint8_t __st_pad[SFXGE_CPU_CACHE_SIZE];
653         } __st_u4;
654 };
```

```
656 #define st_sp           __st_u1.__st_s1.__st_sp
657 #define st_index        __st_u1.__st_s1.__st_index
658 #define st_type         __st_u1.__st_s1.__st_type
659 #define st_evq          __st_u1.__st_s1.__st_evq
660 #define st_mem          __st_u1.__st_s1.__st_mem
661 #define st_id           __st_u1.__st_s1.__st_id
662 #define st_ksp          __st_u1.__st_s1.__st_ksp
663 #define st_stat         __st_u1.__st_s1.__st_stat
664 #define st_state        __st_u1.__st_s1.__st_state

666 #define st_dpl          __st_u2.__st_s2.__st_dpl

668 #define st_lock         __st_u3.__st_s3.__st_lock
669 #define st_fmp          __st_u3.__st_s3.__st_fmp
670 #define st_fbp          __st_u3.__st_s3.__st_fbp
671 #define st_fpp          __st_u3.__st_s3.__st_fpp
672 #define st_eb           __st_u3.__st_s3.__st_eb
673 #define st_n            __st_u3.__st_s3.__st_n
674 #define st_etp          __st_u3.__st_s3.__st_etp
675 #define st_stmp         __st_u3.__st_s3.__st_stmp
676 #define st_stbp         __st_u3.__st_s3.__st_stbp
677 #define st_mp           __st_u3.__st_s3.__st_mp
678 #define st_added        __st_u3.__st_s3.__st_added
679 #define st_reaped       __st_u3.__st_s3.__st_reaped
680 #define st_unblock      __st_u3.__st_s3.__st_unblock

682 #define st_next         __st_u4.__st_s4.__st_next
683 #define st_pending      __st_u4.__st_s4.__st_pending
684 #define st_completed    __st_u4.__st_s4.__st_completed
685 #define st_flush        __st_u4.__st_s4.__st_flush

687 typedef enum sfxge_rx_scale_state_e {
688         SFXGE_RX_SCALE_UNINITIALIZED = 0,
689         SFXGE_RX_SCALE_INITIALIZED,
690         SFXGE_RX_SCALE_STARTED
691 } sfxge_rx_scale_state_t;

693 #define SFXGE_RX_SCALE_MAX      EFX_RSS_TBL_SIZE

695 typedef struct sfxge_rx_scale_s {
696         kmutex_t                srs_lock;
697         unsigned int            *srs_cpu;
698 #ifdef  _USE_CPU_PHYSID
699         unsigned int            *srs_core;
700         unsigned int            *srs_cache;
701         unsigned int            *srs_chip;
702 #endif
703         unsigned int            srs_tbl[SFXGE_RX_SCALE_MAX];
704         unsigned int            srs_count;
705         kstat_t                 *srs_ksp;
706         sfxge_rx_scale_state_t  srs_state;
707 } sfxge_rx_scale_t;


710 #if defined(_USE_GLD_V2) || defined(_USE_GLD_V3_SOL10)
711 typedef struct sfxge_ndd_param_s {
712         sfxge_t         *snp_sp;
713         unsigned int    snp_id;
714         const char      *snp_name;
715         int             (*snp_get)(queue_t *, mblk_t *, caddr_t, cred_t *);
716         int             (*snp_set)(queue_t *, mblk_t *, char *, caddr_t,
717             cred_t *);
718 } sfxge_ndd_param_t;

720 #endif
721 typedef enum sfxge_rx_coalesce_mode_e {
```



```
722         SFXGE_RX_COALESCE_OFF = 0,
723         SFXGE_RX_COALESCE_DISALLOW_PUSH = 1,
724         SFXGE_RX_COALESCE_ALLOW_PUSH = 2
725 } sfxge_rx_coalesce_mode_t;

727 typedef enum sfxge_vpd_type_e {
728         SFXGE_VPD_ID = 0,
729         SFXGE_VPD_PN = 1,
730         SFXGE_VPD_SN = 2,
731         SFXGE_VPD_EC = 3,
732         SFXGE_VPD_MN = 4,
733         SFXGE_VPD_VD = 5,
734         SFXGE_VPD_MAX = 6,
735 } sfxge_vpd_type_t;

737 typedef struct sfxge_vpd_kstat_s {
738         kstat_t         *svk_ksp;
739         kstat_named_t   svk_stat[SFXGE_VPD_MAX];
740         efx_vpd_value_t *svk_vv;
741 } sfxge_vpd_kstat_t;

743 typedef struct sfxge_cfg_kstat_s {
744         struct {
745                 kstat_named_t   sck_mac;
746                 kstat_named_t   sck_version;
747         } kstat;
748         struct {
749                 char            sck_mac[64 + 1];
750         } buf;
751 } sfxge_cfg_kstat_t;

753 typedef enum sfxge_state_e {
754         SFXGE_UNINITIALIZED = 0,
755         SFXGE_INITIALIZED,
756         SFXGE_REGISTERED,
757         SFXGE_STARTING,
758         SFXGE_STARTED,
759         SFXGE_STOPPING
760 } sfxge_state_t;

762 typedef enum sfxge_hw_err_e {
763         SFXGE_HW_OK = 0,
764         SFXGE_HW_ERR,
765 } sfxge_hw_err_t;

767 typedef enum sfxge_action_on_hw_err_e {
768         SFXGE_RECOVER = 0,
769         SFXGE_INVISIBLE = 1,
770         SFXGE_LEAVE_DEAD = 2,
771 } sfxge_action_on_hw_err_t;

773 typedef char *sfxge_mac_priv_prop_t;

775 struct sfxge_s {
776         kmutex_t                        s_state_lock;
777         sfxge_state_t                   s_state;
778         dev_info_t                      *s_dip;
779         ddi_taskq_t                     *s_tqp;
780         ddi_acc_handle_t                s_pci_handle;
781         uint16_t                        s_pci_venid;
782         uint16_t                        s_pci_devid;
783         efx_family_t                    s_family;
784         unsigned int                    s_pcie_nlanes;
785         unsigned int                    s_pcie_linkspeed;
786         kmutex_t                        s_nic_lock;
787         efsys_bar_t                     s_bar;
```

```
788         sfxge_intr_t                    s_intr;
789         sfxge_mac_t                     s_mac;
790         sfxge_mon_t                     s_mon;
791         sfxge_sram_t                    s_sram;
792         sfxge_mcdi_t                    s_mcdi;
793         kmem_cache_t                    *s_eq0c; /* eventQ 0 */
794         kmem_cache_t                    *s_eqXc; /* all other eventQs */
795         sfxge_evq_t                     *s_sep[SFXGE_RX_SCALE_MAX];
796         unsigned int                    s_ev_moderation;
797         kmem_cache_t                    *s_rqc;
798         sfxge_rxq_t                     *s_srp[SFXGE_RX_SCALE_MAX];
799         sfxge_rx_scale_t                s_rx_scale;
800         size_t                          s_rx_prefix_size;
801         size_t                          s_rx_buffer_size;
802         size_t                          s_rx_buffer_align;
803         sfxge_rx_coalesce_mode_t        s_rx_coalesce_mode;
804         int64_t                         s_rx_pkt_mem_max;
805         volatile uint64_t               s_rx_pkt_mem_alloc;
806         kmem_cache_t                    *s_rpc;
807         kmem_cache_t                    *s_tqc;
808         int                             s_tx_qcount;
809         sfxge_txq_t                     *s_stp[SFXGE_TXQ_NTYPES +
810             SFXGE_RX_SCALE_MAX];
811         kmem_cache_t                    *s_tpc;
812         int                             s_tx_flush_pending;
813         kmutex_t                        s_tx_flush_lock;
814         kcondvar_t                      s_tx_flush_kv;
815         kmem_cache_t                    *s_tbc;
816         kmem_cache_t                    *s_tmc;
817         efx_nic_t                       *s_enp;
818         sfxge_vpd_kstat_t               s_vpd_kstat;
819         sfxge_cfg_kstat_t               s_cfg_kstat;
820         kstat_t                         *s_cfg_ksp;
821         size_t                          s_mtu;
822         int                             s_rxq_poll_usec;
823 #ifdef _USE_GLD_V3
824         mac_callbacks_t                 s_mc;
825         mac_handle_t                    s_mh;
826 #ifdef _USE_MAC_PRIV_PROP
827         sfxge_mac_priv_prop_t           *s_mac_priv_props;
828         int                             s_mac_priv_props_alloc;
829 #endif
830 #endif
831 #if defined(_USE_GLD_V2) || defined(_USE_GLD_V3)
832         sfxge_ndd_param_t               *s_ndp;
833         kstat_named_t                   *s_nd_stat;
834         caddr_t                         s_ndh;
835         kstat_t                         *s_nd_ksp;
836 #endif
837 #ifdef _USE_GLD_V2
838         gld_mac_info_t                  *s_gmip;
839         kmutex_t                        s_rx_lock;
840         mblk_t                          *s_mp;
841         mblk_t                          **s_mpp;
842 #endif
843         uint32_t                        s_num_restarts;
844         uint32_t                        s_num_restarts_hw_err;
845         sfxge_hw_err_t                  s_hw_err;
846         sfxge_action_on_hw_err_t        s_action_on_hw_err;
847         uint16_t                        s_rxq_size;
848         uint16_t                        s_evq0_size;
849         uint16_t                        s_evqX_size;
850 };

852 typedef struct sfxge_dma_buffer_attr_s {
853         dev_info_t              *sdba_dip;
```



```
854         ddi_dma_attr_t          *sdba_dattrp;
855         int                     (*sdba_callback) (caddr_t);
856         size_t                  sdba_length;
857         uint_t                  sdba_memflags;
858         ddi_device_acc_attr_t   *sdba_devaccp;
859         uint_t                  sdba_bindflags;
860         int                     sdba_maxcookies;
861         boolean_t               sdba_zeroinit;
862 } sfxge_dma_buffer_attr_t;

864 extern const char               sfxge_ident[];
865 extern uint8_t                  sfxge_brdcst[];

867 extern kmutex_t                 sfxge_global_lock;

869 extern unsigned int             *sfxge_cpu;
870 #ifdef  _USE_CPU_PHYSID
871 extern unsigned int             *sfxge_core;
872 extern unsigned int             *sfxge_cache;
873 extern unsigned int             *sfxge_chip;
874 #endif

876 extern int                      sfxge_start(sfxge_t *, boolean_t);
877 extern void                     sfxge_stop(sfxge_t *);
878 extern void                     sfxge_ioctl(sfxge_t *, queue_t *, mblk_t *);
879 extern int                      sfxge_restart_dispatch(sfxge_t *, uint_t,
880     sfxge_hw_err_t, const char *, uint32_t);

882 extern void                     sfxge_gld_link_update(sfxge_t *);
883 extern void                     sfxge_gld_mtu_update(sfxge_t *);
884 extern void                     sfxge_gld_rx_post(sfxge_t *, unsigned int,
885     mblk_t *);
886 extern void                     sfxge_gld_rx_push(sfxge_t *);
887 extern int                      sfxge_gld_register(sfxge_t *);
888 extern int                      sfxge_gld_unregister(sfxge_t *);

890 extern int                      sfxge_gld_nd_register(sfxge_t *);
891 extern void                     sfxge_gld_nd_unregister(sfxge_t *);
892 #ifdef _USE_MAC_PRIV_PROP
893 extern void                     sfxge_gld_priv_prop_rename(sfxge_t *);
894 #endif

896 extern int                      sfxge_dma_buffer_create(efsys_mem_t *,
897     const sfxge_dma_buffer_attr_t *);
898 extern void                     sfxge_dma_buffer_destroy(efsys_mem_t *);

900 extern int                      sfxge_intr_init(sfxge_t *);
901 extern int                      sfxge_intr_start(sfxge_t *);
902 extern void                     sfxge_intr_stop(sfxge_t *);
903 extern void                     sfxge_intr_fini(sfxge_t *);
904 extern void                     sfxge_intr_fatal(sfxge_t *);

906 extern int                      sfxge_ev_init(sfxge_t *);
907 extern int                      sfxge_ev_start(sfxge_t *);
908 extern void                     sfxge_ev_moderation_get(sfxge_t *,
909     unsigned int *);
910 extern int                      sfxge_ev_moderation_set(sfxge_t *,
911     unsigned int);
912 extern int                      sfxge_ev_qmoderate(sfxge_t *, unsigned int,
913     unsigned int);
914 extern int                      sfxge_ev_qpoll(sfxge_t *, unsigned int);
915 extern int                      sfxge_ev_qprime(sfxge_t *, unsigned int);
916 extern void                     sfxge_ev_stop(sfxge_t *);
917 extern void                     sfxge_ev_fini(sfxge_t *);

919 extern int                      sfxge_mon_init(sfxge_t *);
```

```
extern int                      sfxge_mon_start(sfxge_t *);
extern void                     sfxge_mon_stop(sfxge_t *);
extern void                     sfxge_mon_fini(sfxge_t *);

extern int                      sfxge_mac_init(sfxge_t *);
extern int                      sfxge_mac_start(sfxge_t *, boolean_t);
extern void                     sfxge_mac_stat_get(sfxge_t *, unsigned int,
    uint64_t *);
extern void                     sfxge_mac_link_check(sfxge_t *, boolean_t *);
extern void                     sfxge_mac_link_speed_get(sfxge_t *,
    unsigned int *);
extern void                     sfxge_mac_link_duplex_get(sfxge_t *,
    sfxge_link_duplex_t *);
extern void                     sfxge_mac_fcntl_get(sfxge_t *, unsigned int *);
extern int                      sfxge_mac_fcntl_set(sfxge_t *, unsigned int);
extern int                      sfxge_mac_unicst_get(sfxge_t *,
    sfxge_unicst_type_t, uint8_t *);
extern int                      sfxge_mac_unicst_set(sfxge_t *,
    uint8_t *);
extern int                      sfxge_mac_promisc_set(sfxge_t *,
    sfxge_promisc_type_t);
extern int                      sfxge_mac_multicst_add(sfxge_t *,
    uint8_t *);
extern int                      sfxge_mac_multicst_remove(sfxge_t *,
    uint8_t *);
extern int                      sfxge_mac_ioctl(sfxge_t *, sfxge_mac_ioc_t *);
extern void                     sfxge_mac_stop(sfxge_t *);
extern void                     sfxge_mac_fini(sfxge_t *);
extern void                     sfxge_mac_link_update(sfxge_t *sp,
    efx_link_mode_t mode);

extern int                      sfxge_mcdi_init(sfxge_t *sp);
extern void                     sfxge_mcdi_fini(sfxge_t *sp);
extern int                      sfxge_mcdi_ioctl(sfxge_t *sp,
    sfxge_mcdi_ioc_t *smip);

extern int                      sfxge_phy_init(sfxge_t *);
extern void                     sfxge_phy_link_mode_get(sfxge_t *,
    efx_link_mode_t *);
extern int                      sfxge_phy_ioctl(sfxge_t *, sfxge_phy_ioc_t *);
extern int                      sfxge_phy_bist_ioctl(sfxge_t *,
    sfxge_phy_bist_ioc_t *);
extern void                     sfxge_phy_fini(sfxge_t *);
extern int                      sfxge_phy_kstat_init(sfxge_t *sp);
extern void                     sfxge_phy_kstat_fini(sfxge_t *sp);
extern uint8_t                  sfxge_phy_lp_cap_test(sfxge_t *sp,
    uint32_t field);
extern int                      sfxge_phy_cap_apply(sfxge_t *sp,
    boolean_t use_default);
extern uint8_t                  sfxge_phy_cap_test(sfxge_t *sp, uint32_t flags,
    uint32_t field, boolean_t *mutablep);
extern int                      sfxge_phy_cap_set(sfxge_t *sp, uint32_t field,
    int set);
#ifdef _USE_MAC_PRIV_PROP
extern int                      sfxge_phy_prop_get(sfxge_t *sp, unsigned int id,
    uint32_t flags, uint32_t *valp);
extern int                      sfxge_phy_prop_set(sfxge_t *sp, unsigned int id,
    uint32_t val);
#endif

extern int                      sfxge_rx_init(sfxge_t *);
extern int                      sfxge_rx_start(sfxge_t *);
extern void                     sfxge_rx_coalesce_mode_get(sfxge_t *,
    sfxge_rx_coalesce_mode_t *);
extern int                      sfxge_rx_coalesce_mode_set(sfxge_t *,
    sfxge_rx_coalesce_mode_t);
```



```
extern unsigned int             sfxge_rx_scale_prop_get(sfxge_t *);
extern void                     sfxge_rx_scale_update(void *);
extern int                      sfxge_rx_scale_count_get(sfxge_t *,
    unsigned int *);
extern int                      sfxge_rx_scale_count_set(sfxge_t *,
    unsigned int);
extern void                     sfxge_rx_qcomplete(sfxge_rxq_t *, boolean_t);
extern void                     sfxge_rx_qflush_done(sfxge_rxq_t *);
extern void                     sfxge_rx_qflush_failed(sfxge_rxq_t *);
extern void                     sfxge_rx_qfpp_trim(sfxge_rxq_t *);
extern int                      sfxge_rx_ioctl(sfxge_t *, sfxge_rx_ioc_t *);
extern void                     sfxge_rx_stop(sfxge_t *);
extern unsigned int             sfxge_rx_loaned(sfxge_t *);
extern void                     sfxge_rx_fini(sfxge_t *);

extern int                      sfxge_tx_init(sfxge_t *);
extern int                      sfxge_tx_start(sfxge_t *);
extern int                      sfxge_tx_packet_add(sfxge_t *, mblk_t *);
extern void                     sfxge_tx_qcomplete(sfxge_txq_t *);
extern void                     sfxge_tx_qflush_done(sfxge_txq_t *);
extern int                      sfxge_tx_ioctl(sfxge_t *, sfxge_tx_ioc_t *);
extern void                     sfxge_tx_stop(sfxge_t *);
extern void                     sfxge_tx_fini(sfxge_t *);
extern void                     sfxge_tx_qdpl_flush(sfxge_txq_t *stp);

extern void                     sfxge_sram_init(sfxge_t *);
extern int                      sfxge_sram_buf_tbl_alloc(sfxge_t *, size_t,
    uint32_t *);
extern int                      sfxge_sram_start(sfxge_t *);
extern int                      sfxge_sram_buf_tbl_set(sfxge_t *, uint32_t,
    efsys_mem_t *, size_t);
extern void                     sfxge_sram_buf_tbl_clear(sfxge_t *, uint32_t,
    size_t);
extern void                     sfxge_sram_stop(sfxge_t *);
extern void                     sfxge_sram_buf_tbl_free(sfxge_t *, uint32_t,
    size_t);
extern int                      sfxge_sram_ioctl(sfxge_t *, sfxge_sram_ioc_t *);
extern void                     sfxge_sram_fini(sfxge_t *);

extern void                     sfxge_tcp_parse(mblk_t *,
    struct ether_header **, struct ip **, struct tcphdr **, size_t *, size_t *);

#define SFXGE_TCP_HASH(_raddr, _rport, _laddr, _lport, _hash)           \
        do {                                                            \
                uint32_t raddr = (uint32_t)(_raddr);                    \
                uint32_t rport = (uint32_t)(_rport);                    \
                uint32_t laddr = (uint32_t)(_laddr);                    \
                uint32_t lport = (uint32_t)(_lport);                    \
                uint32_t key;                                           \
                uint32_t lfsr;                                          \
                unsigned int index;                                     \
                                                                        \
                key = laddr ^                                           \
                    ((lport << 16) | (raddr >> 16)) ^                   \
                    ((raddr << 16) | rport);                            \
                                                                        \
                lfsr = 0xffffffff;                                      \
                for (index = 0; index < 32; index++) {                  \
                        uint32_t input;                                 \
                        uint32_t key_bit32;                             \
                        uint32_t lfsr_bit16;                            \
                        uint32_t lfsr_bit3;                             \
                                                                        \
                        key_bit32 = key >> 31;                          \
                        key <<= 1;                                      \
                                                                        \
```

```
1052                        lfsr_bit16 = (lfsr >> 15) & 1;                  \
1053                        lfsr_bit3 = (lfsr >> 2) & 1;                    \
1054                                                                        \
1055                        input = (lfsr_bit16 ^ lfsr_bit3) ^ key_bit32;   \
1056                        ASSERT((input & ~1) == 0);                      \
1057                                                                        \
1058                        lfsr = (lfsr << 1) | input;                     \
1059                }                                                       \
1060                                                                        \
1061                (_hash) = (uint16_t)lfsr;                               \
1062        _NOTE(CONSTANTCONDITION)                                        \
1063        } while (B_FALSE)

1065 extern int             sfxge_nvram_ioctl(sfxge_t *, sfxge_nvram_ioc_t *);

1067 extern int             sfxge_pci_init(sfxge_t *);
1068 extern void            sfxge_pcie_check_link(sfxge_t *, unsigned int,
1069     unsigned int);
1070 extern int             sfxge_pci_ioctl(sfxge_t *, sfxge_pci_ioc_t *);
1071 extern void            sfxge_pci_fini(sfxge_t *);

1073 extern int             sfxge_bar_init(sfxge_t *);
1074 extern int             sfxge_bar_ioctl(sfxge_t *, sfxge_bar_ioc_t *);
1075 extern void            sfxge_bar_fini(sfxge_t *);

1077 extern int             sfxge_vpd_ioctl(sfxge_t *, sfxge_vpd_ioc_t *);

1079 #endif /* _KERNEL */

1081 #ifdef  __cplusplus
1082 }
1083 #endif

1085 #endif  /* _SYS_SFXGE_H */
1086 #endif /* ! codereview */
```

```
**********************************************************
    2827 Thu Aug 22 18:59:27 2013
new/usr/src/uts/common/io/sfxge/sfxge_bar.c
Merged sfxge driver
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2008-2013 Solarflare Communications Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#include <sys/types.h>
#include <sys/ddi.h>
#include <sys/sunddi.h>
#include <sys/stream.h>
#include <sys/dlpi.h>

#include "sfxge.h"

int
sfxge_bar_init(sfxge_t *sp)
{
        efsys_bar_t *esbp = &(sp->s_bar);
        ddi_device_acc_attr_t devacc;
        int rc;

        devacc.devacc_attr_version = DDI_DEVICE_ATTR_V0;
        devacc.devacc_attr_endian_flags = DDI_NEVERSWAP_ACC;
        devacc.devacc_attr_dataorder = DDI_STRICTORDER_ACC;

        if (ddi_regs_map_setup(sp->s_dip, EFX_MEM_BAR, &(esbp->esb_base), 0, 0,
            &devacc, &(esbp->esb_handle)) != DDI_SUCCESS) {
                rc = ENODEV;
                goto fail1;
        }

        mutex_init(&(esbp->esb_lock), NULL, MUTEX_DRIVER, NULL);

        return (0);

fail1:
        DTRACE_PROBE1(fail1, int, rc);

        return (rc);
}
```



```
int
sfxge_bar_ioctl(sfxge_t *sp, sfxge_bar_ioc_t *sbip)
{
        efsys_bar_t *esbp = &(sp->s_bar);
        efx_oword_t oword;
        int rc;

        if (!IS_P2ALIGNED(sbip->sbi_addr, sizeof (efx_oword_t))) {
                rc = EINVAL;
                goto fail1;
        }

        switch (sbip->sbi_op) {
        case SFXGE_BAR_OP_READ:
                EFSYS_BAR_READO(esbp, sbip->sbi_addr, &oword, B_TRUE);

                sbip->sbi_data[0] = EFX_OWORD_FIELD(oword, EFX_DWORD_0);
                sbip->sbi_data[1] = EFX_OWORD_FIELD(oword, EFX_DWORD_1);
                sbip->sbi_data[2] = EFX_OWORD_FIELD(oword, EFX_DWORD_2);
                sbip->sbi_data[3] = EFX_OWORD_FIELD(oword, EFX_DWORD_3);

                break;

        case SFXGE_BAR_OP_WRITE:
                EFX_POPULATE_OWORD_4(oword,
                    EFX_DWORD_0, sbip->sbi_data[0],
                    EFX_DWORD_1, sbip->sbi_data[1],
                    EFX_DWORD_2, sbip->sbi_data[2],
                    EFX_DWORD_3, sbip->sbi_data[3]);

                EFSYS_BAR_WRITEO(esbp, sbip->sbi_addr, &oword, B_TRUE);
                break;

        default:
                rc = ENOTSUP;
                goto fail2;
        }

        return (0);

fail2:
        DTRACE_PROBE(fail2);
fail1:
        DTRACE_PROBE1(fail1, int, rc);

        return (rc);
}

void
sfxge_bar_fini(sfxge_t *sp)
{
        efsys_bar_t *esbp = &(sp->s_bar);

        ddi_regs_map_free(&(esbp->esb_handle));

        mutex_destroy(&(esbp->esb_lock));

        esbp->esb_base = NULL;
        esbp->esb_handle = NULL;
}
#endif /* ! codereview */
```

```
**********************************************************
    1996 Thu Aug 22 18:59:27 2013
new/usr/src/uts/common/io/sfxge/sfxge_debug.h
Merged sfxge driver
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2008-2013 Solarflare Communications Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#ifndef _SYS_SFXGE_DEBUG_H
#define _SYS_SFXGE_DEBUG_H

#include <sys/types.h>

#ifdef  __cplusplus
extern "C" {
#endif

#ifdef  DEBUG

extern boolean_t sfxge_aask;

#define SFXGE_OBJ_CHECK(_objp, _type)                                   \
        do {                                                            \
                uint8_t *p = (uint8_t *)(_objp);                        \
                size_t off;                                             \
                                                                        \
                for (off = 0; off < sizeof (_type); off++) {            \
                        char buf[MAXNAMELEN];                           \
                                                                        \
                        if (*p++ == 0)                                  \
                                continue;                               \
                                                                        \
                        (void) snprintf(buf, MAXNAMELEN - 1,            \
                            "%s[%d]: non-zero byte found in %s "        \
                            "at 0x%p+%lx", __FILE__, __LINE__, #_type,  \
                            (void *)(_objp), off);                      \
                                                                        \
                        if (sfxge_aask)                                 \
                                debug_enter(buf);                       \
                        break;                                          \
                }                                                       \
        _NOTE(CONSTANTCONDITION)                                        \
        } while (B_FALSE)
```



```
/* Log cmn_err(9F) messages to console and system log */
#define SFXGE_CMN_ERR   ""

#else   /* DEBUG */

#define SFXGE_OBJ_CHECK(_objp, _type)

/* Log cmn_err(9F) messages to system log only */
#define SFXGE_CMN_ERR   "!"

#endif  /* DEBUG */

#ifdef  __cplusplus
}
#endif

#endif  /* _SYS_SFXGE_DEBUG_H */
#endif /* ! codereview */
```

```
**********************************************************
    3854 Thu Aug 22 18:59:27 2013
new/usr/src/uts/common/io/sfxge/sfxge_dma.c
Merged sfxge driver
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2008-2013 Solarflare Communications Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#include <sys/ddi.h>

#include "sfxge.h"
#include "efx.h"

static int
sfxge_dma_buffer_unbind_handle(efsys_mem_t *esmp)
{
        int rc;

        esmp->esm_addr = 0;
        rc = ddi_dma_unbind_handle(esmp->esm_dma_handle);
        if (rc != DDI_SUCCESS)
                goto fail1;

        return (0);

fail1:
        DTRACE_PROBE1(fail1, int, rc);

        return (rc);
}

static void
sfxge_dma_buffer_mem_free(efsys_mem_t *esmp)
{
        esmp->esm_base = NULL;
        ddi_dma_mem_free(&(esmp->esm_acc_handle));
        esmp->esm_acc_handle = NULL;
}

static void
sfxge_dma_buffer_handle_free(ddi_dma_handle_t *dhandlep)
{
        ddi_dma_free_handle(dhandlep);
```



```
        *dhandlep = NULL;
}

int
sfxge_dma_buffer_create(efsys_mem_t *esmp, const sfxge_dma_buffer_attr_t *sdbap)
{
        int err;
        int rc;
        size_t unit;
        ddi_dma_cookie_t dmac;
        unsigned int ncookies;

        /* Allocate a DMA handle */
        err = ddi_dma_alloc_handle(sdbap->sdba_dip, sdbap->sdba_dattrp,
            sdbap->sdba_callback, NULL, &(esmp->esm_dma_handle));
        switch (err) {
        case DDI_SUCCESS:
                break;

        case DDI_DMA_BADATTR:
                rc = EINVAL;
                goto fail1;

        case DDI_DMA_NORESOURCES:
                rc = ENOMEM;
                goto fail1;

        default:
                rc = EFAULT;
                goto fail1;
        }

        /* Allocate some DMA memory */
        err = ddi_dma_mem_alloc(esmp->esm_dma_handle, sdbap->sdba_length,
            sdbap->sdba_devaccp, sdbap->sdba_memflags,
            sdbap->sdba_callback, NULL,
            &(esmp->esm_base), &unit, &(esmp->esm_acc_handle));
        switch (err) {
        case DDI_SUCCESS:
                break;

        case DDI_FAILURE:
                /*FALLTHRU*/
        default:
                rc = EFAULT;
                goto fail2;
        }

        if (sdbap->sdba_zeroinit)
                bzero(esmp->esm_base, sdbap->sdba_length);

        /* Bind the DMA memory to the DMA handle */
        /* We aren't handling partial mappings */
        ASSERT3U(sdbap->sdba_bindflags & DDI_DMA_PARTIAL, !=, DDI_DMA_PARTIAL);
        err = ddi_dma_addr_bind_handle(esmp->esm_dma_handle, NULL,
            esmp->esm_base, sdbap->sdba_length, sdbap->sdba_bindflags,
            sdbap->sdba_callback, NULL, &dmac, &ncookies);
        switch (err) {
        case DDI_DMA_MAPPED:
                break;

        case DDI_DMA_INUSE:
                rc = EEXIST;
                goto fail3;

        case DDI_DMA_NORESOURCES:
```

```
128                 rc = ENOMEM;
129                 goto fail3;

131         case DDI_DMA_NOMAPPING:
132                 rc = ENOTSUP;
133                 goto fail3;

135         case DDI_DMA_TOOBIG:
136                 rc = EFBIG;
137                 goto fail3;

139         default:
140                 rc = EFAULT;
141                 goto fail3;
142         }
143         ASSERT3U(ncookies, >=, 1);
144         ASSERT3U(ncookies, <=, sdbap->sdba_maxcookies);

146         esmp->esm_addr = dmac.dmac_laddress;
147         DTRACE_PROBE1(addr, efsys_dma_addr_t, esmp->esm_addr);

149         return (0);

151 fail3:
152         DTRACE_PROBE(fail3);

154         sfxge_dma_buffer_mem_free(esmp);

156 fail2:
157         DTRACE_PROBE(fail2);

159         sfxge_dma_buffer_handle_free(&(esmp->esm_dma_handle));
160         esmp->esm_dma_handle = NULL;

162 fail1:
163         DTRACE_PROBE1(fail1, int, rc);

165         return (-1);
166 }

168 void
169 sfxge_dma_buffer_destroy(efsys_mem_t *esmp)
170 {
171         int rc;

173         rc = sfxge_dma_buffer_unbind_handle(esmp);
174         if (rc != 0) {
175                 cmn_err(CE_WARN, SFXGE_CMN_ERR "ERROR: DMA Unbind failed rc=%d",
176                     rc);
177         }
178         sfxge_dma_buffer_mem_free(esmp);
179         sfxge_dma_buffer_handle_free(&(esmp->esm_dma_handle));
180 }
181 #endif /* ! codereview */
```

```
**********************************************************
    2225 Thu Aug 22 18:59:27 2013
new/usr/src/uts/common/io/sfxge/sfxge_err.c
Merged sfxge driver
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2008-2013 Solarflare Communications Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#include <sys/types.h>
#include <sys/ddi.h>
#include <sys/sunddi.h>

#include "sfxge.h"

#include "efx.h"

static const char *__sfxge_err[] = {
        "",
        "SRAM out-of-bounds",
        "Buffer ID out-of-bounds",
        "Internal memory parity",
        "Receive buffer ownership",
        "Transmit buffer ownership",
        "Receive descriptor ownership",
        "Transmit descriptor ownership",
        "Event queue ownership",
        "Event queue FIFO overflow",
        "Illegal address",
        "SRAM parity"
};

void
sfxge_err(efsys_identifier_t *arg, unsigned int code, uint32_t dword0,
    uint32_t dword1)
{
        sfxge_t *sp = (sfxge_t *)arg;
        dev_info_t *dip = sp->s_dip;

        ASSERT3U(code, <, EFX_ERR_NCODES);

        cmn_err(CE_WARN, SFXGE_CMN_ERR "[%s%d] FATAL ERROR: %s (0x%08x%08x)",
            ddi_driver_name(dip), ddi_get_instance(dip), __sfxge_err[code],
            dword1, dword0);
```



```
}

void
sfxge_intr_fatal(sfxge_t *sp)
{
        efx_nic_t *enp = sp->s_enp;
        int err;

        efx_intr_disable(enp);
        efx_intr_fatal(enp);

        err = sfxge_restart_dispatch(sp, DDI_NOSLEEP, SFXGE_HW_ERR,
            "Fatal Interrupt", 0);
        if (err != 0) {
                cmn_err(CE_WARN, SFXGE_CMN_ERR
                                "[%s%d] UNRECOVERABLE ERROR:"
                                " Could not schedule driver restart."
                                " err=%d\n",
                                ddi_driver_name(sp->s_dip),
                                ddi_get_instance(sp->s_dip),
                                err);
                ASSERT(B_FALSE);
        }
}
#endif /* ! codereview */
```

```
**********************************************************
   26468 Thu Aug 22 18:59:27 2013
new/usr/src/uts/common/io/sfxge/sfxge_ev.c
Merged sfxge driver
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2008-2013 Solarflare Communications Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#include <sys/types.h>
#include <sys/sysmacros.h>
#include <sys/ddi.h>
#include <sys/sunddi.h>
#include <sys/stream.h>
#include <sys/strsun.h>
#include <sys/strsubr.h>
#include <sys/cpu.h>
#include <sys/pghw.h>

#include "sfxge.h"

#include "efx.h"


/* Event queue DMA attributes */
static ddi_device_acc_attr_t sfxge_evq_devacc = {

        DDI_DEVICE_ATTR_V0,     /* devacc_attr_version */
        DDI_NEVERSWAP_ACC,      /* devacc_attr_endian_flags */
        DDI_STRICTORDER_ACC     /* devacc_attr_dataorder */
};

static ddi_dma_attr_t sfxge_evq_dma_attr = {
        DMA_ATTR_V0,            /* dma_attr_version     */
        0,                      /* dma_attr_addr_lo     */
        0xffffffffffffffffull,  /* dma_attr_addr_hi     */
        0xffffffffffffffffull,  /* dma_attr_count_max   */
        EFX_BUF_SIZE,           /* dma_attr_align       */
        0xffffffff,             /* dma_attr_burstsizes  */
        1,                      /* dma_attr_minxfer     */
        0xffffffffffffffffull,  /* dma_attr_maxxfer     */
        0xffffffffffffffffull,  /* dma_attr_seg         */
        1,                      /* dma_attr_sgllen      */
        1,                      /* dma_attr_granular    */
```



```
        0                       /* dma_attr_flags       */
};

static int
_sfxge_ev_qctor(sfxge_t *sp, sfxge_evq_t *sep, int kmflags, uint16_t evq_size)
{
        efsys_mem_t *esmp = &(sep->se_mem);
        sfxge_dma_buffer_attr_t dma_attr;
        int rc;

        /* Compile-time structure layout checks */
        EFX_STATIC_ASSERT(sizeof (sep->__se_u1.__se_s1) <=
            sizeof (sep->__se_u1.__se_pad));
        EFX_STATIC_ASSERT(sizeof (sep->__se_u2.__se_s2) <=
            sizeof (sep->__se_u2.__se_pad));

        bzero(sep, sizeof (sfxge_evq_t));

        sep->se_sp = sp;

        dma_attr.sdba_dip        = sp->s_dip;
        dma_attr.sdba_dattrp     = &sfxge_evq_dma_attr;
        dma_attr.sdba_callback   = (kmflags == KM_SLEEP) ?
            DDI_DMA_SLEEP : DDI_DMA_DONTWAIT;
        dma_attr.sdba_length     = EFX_EVQ_SIZE(evq_size);
        dma_attr.sdba_memflags   = DDI_DMA_CONSISTENT;
        dma_attr.sdba_devaccp    = &sfxge_evq_devacc;
        dma_attr.sdba_bindflags  = DDI_DMA_READ | DDI_DMA_CONSISTENT;
        dma_attr.sdba_maxcookies = 1;
        dma_attr.sdba_zeroinit   = B_FALSE;

        if ((rc = sfxge_dma_buffer_create(esmp, &dma_attr)) != 0)
                goto fail1;

        /* Allocate some buffer table entries */
        if ((rc = sfxge_sram_buf_tbl_alloc(sp, EFX_EVQ_NBUFS(evq_size),
            &(sep->se_id))) != 0)
                goto fail2;

        sep->se_stpp = &(sep->se_stp);

        return (0);

fail2:
        DTRACE_PROBE(fail2);

        /* Tear down DMA setup */
        esmp->esm_addr = 0;
        sfxge_dma_buffer_destroy(esmp);

fail1:
        DTRACE_PROBE1(fail1, int, rc);

        sep->se_sp = NULL;

        SFXGE_OBJ_CHECK(sep, sfxge_evq_t);

        return (-1);
}

static int
sfxge_ev_q0ctor(void *buf, void *arg, int kmflags)
{
        sfxge_evq_t *sep = buf;
        sfxge_t *sp = arg;
        return (_sfxge_ev_qctor(sp, sep, kmflags, sp->s_evq0_size));
```

```
}

static int
sfxge_ev_qXctor(void *buf, void *arg, int kmflags)
{
        sfxge_evq_t *sep = buf;
        sfxge_t *sp = arg;
        return (_sfxge_ev_qctor(sp, sep, kmflags, sp->s_evqX_size));
}
static void
_sfxge_ev_qdtor(sfxge_t *sp, sfxge_evq_t *sep, uint16_t evq_size)
{
        efsys_mem_t *esmp = &(sep->se_mem);
        ASSERT3P(sep->se_sp, ==, sp);
        ASSERT3P(sep->se_stpp, ==, &(sep->se_stp));
        sep->se_stpp = NULL;

        /* Free the buffer table entries */
        sfxge_sram_buf_tbl_free(sp, sep->se_id, EFX_EVQ_NBUFS(evq_size));
        sep->se_id = 0;

        /* Tear down DMA setup */
        sfxge_dma_buffer_destroy(esmp);

        sep->se_sp = NULL;

        SFXGE_OBJ_CHECK(sep, sfxge_evq_t);
}

static void
sfxge_ev_q0dtor(void *buf, void *arg)
{
        sfxge_evq_t *sep = buf;
        sfxge_t *sp = arg;
        _sfxge_ev_qdtor(sp, sep, sp->s_evq0_size);
}

static void
sfxge_ev_qXdtor(void *buf, void *arg)
{
        sfxge_evq_t *sep = buf;
        sfxge_t *sp = arg;
        _sfxge_ev_qdtor(sp, sep, sp->s_evqX_size);
}

static boolean_t
sfxge_ev_initialized(void *arg)
{
        sfxge_evq_t *sep = arg;

        ASSERT(mutex_owned(&(sep->se_lock)));
        ASSERT3U(sep->se_state, ==, SFXGE_EVQ_STARTING);
        sep->se_state = SFXGE_EVQ_STARTED;

        cv_broadcast(&(sep->se_init_kv));

        return (B_FALSE);
}

static void
sfxge_ev_qcomplete(sfxge_evq_t *sep, boolean_t eop)
{
        sfxge_t *sp = sep->se_sp;
        unsigned int index = sep->se_index;
        sfxge_rxq_t *srp = sp->s_srp[index];
        sfxge_txq_t *stp;
```



```
        if ((stp = sep->se_stp) != NULL) {
                sep->se_stp = NULL;
                sep->se_stpp = &(sep->se_stp);

                do {
                        sfxge_txq_t *next;

                        next = stp->st_next;
                        stp->st_next = NULL;

                        ASSERT3U(stp->st_evq, ==, index);

                        if (stp->st_pending != stp->st_completed)
                                sfxge_tx_qcomplete(stp);

                        stp = next;
                } while (stp != NULL);
        }

        if (srp != NULL) {
                if (srp->sr_pending != srp->sr_completed)
                        sfxge_rx_qcomplete(srp, eop);
        }
}

static boolean_t
sfxge_ev_rx(void *arg, uint32_t label, uint32_t id, uint32_t size,
    uint16_t flags)
{
        sfxge_evq_t *sep = arg;
        sfxge_t *sp = sep->se_sp;
        sfxge_rxq_t *srp;
        sfxge_rx_packet_t *srpp;
        unsigned int expected;
        unsigned int prefetch;

        ASSERT(mutex_owned(&(sep->se_lock)));

        if (sep->se_exception)
                goto done;

        srp = sp->s_srp[label];
        if (srp == NULL)
                goto done;

        ASSERT3U(sep->se_index, ==, srp->sr_index);
        ASSERT3U(id, <, sp->s_rxq_size);

        /*
         * Note that in sfxge_stop() EVQ stopped after RXQ, and will be reset
         * So the return missing srp->sr_pending++ is safe
         */
        if (srp->sr_state != SFXGE_RXQ_STARTED)
                goto done;

        expected = srp->sr_pending++ & (sp->s_rxq_size - 1);
        if (id != expected) {
                sep->se_exception = B_TRUE;

                DTRACE_PROBE(restart_ev_rx_id);
                /* sfxge_evq_t->se_lock held */
                (void) sfxge_restart_dispatch(sp, DDI_SLEEP, SFXGE_HW_ERR,
                    "Out of order RX event", id - expected);

                goto done;
```

```
260         }

262         prefetch = (id + 4) & (sp->s_rxq_size - 1);
263         if ((srpp = srp->sr_srpp[prefetch]) != NULL)
264                 prefetch_read_many(srpp);

266         srpp = srp->sr_srpp[id];
267         ASSERT(srpp != NULL);

269         ASSERT3U(srpp->srp_flags, ==, EFX_DISCARD);
270         srpp->srp_flags = flags;

272         ASSERT3U(size, <, (1 << 16));
273         srpp->srp_size = (uint16_t)size;

275         prefetch_read_many(srpp->srp_mp);

277         sep->se_rx++;

279         DTRACE_PROBE2(qlevel, unsigned int, srp->sr_index,
280             unsigned int, srp->sr_added - srp->sr_pending);

282         if (srp->sr_pending - srp->sr_completed >= SFXGE_RX_BATCH)
283                 sfxge_ev_qcomplete(sep, B_FALSE);

285 done:
286         /* returning B_TRUE makes efx_ev_qpoll() stop processing events */
287         return (sep->se_rx >= sep->se_ev_batch);
288 }

290 static boolean_t
291 sfxge_ev_exception(void *arg, uint32_t code, uint32_t data)
292 {
293         sfxge_evq_t *sep = arg;
294         sfxge_t *sp = sep->se_sp;

296         _NOTE(ARGUNUSED(code))
297         _NOTE(ARGUNUSED(data))

299         ASSERT(mutex_owned(&(sep->se_lock)));
300         sep->se_exception = B_TRUE;

302         if (code != EFX_EXCEPTION_UNKNOWN_SENSOREVT) {

304                 DTRACE_PROBE(restart_ev_exception);

306                 /* sfxge_evq_t->se_lock held */
307                 (void) sfxge_restart_dispatch(sp, DDI_SLEEP, SFXGE_HW_ERR,
308                     "Unknown EV", code);
309         }

311         return (B_FALSE);
312 }

314 static boolean_t
315 sfxge_ev_rxq_flush_done(void *arg, uint32_t label)
316 {
317         sfxge_evq_t *sep_targetq, *sep = arg;
318         sfxge_t *sp = sep->se_sp;
319         sfxge_rxq_t *srp;
320         unsigned int index;
321         uint16_t magic;

323         ASSERT(mutex_owned(&(sep->se_lock)));

325         /* Ensure RXQ exists, as events may arrive after RXQ was destroyed */
```



```
326         srp = sp->s_srp[label];
327         if (srp == NULL)
328                 goto done;

330         /* Resend a software event on the correct queue */
331         index = srp->sr_index;
332         sep_targetq = sp->s_sep[index];

334         if (sep_targetq->se_state != SFXGE_EVQ_STARTED)
335                 goto done; /* TBD: state test not under the lock */

337         ASSERT((label & SFXGE_MAGIC_DMAQ_LABEL_MASK) == label);
338         magic = SFXGE_MAGIC_RX_QFLUSH_DONE | label;

340         efx_ev_qpost(sep_targetq->se_eep, magic);

342 done:
343         return (B_FALSE);
344 }

346 static boolean_t
347 sfxge_ev_rxq_flush_failed(void *arg, uint32_t label)
348 {
349         sfxge_evq_t *sep_targetq, *sep = arg;
350         sfxge_t *sp = sep->se_sp;
351         sfxge_rxq_t *srp;
352         unsigned int index;
353         uint16_t magic;

355         ASSERT(mutex_owned(&(sep->se_lock)));

357         /* Ensure RXQ exists, as events may arrive after RXQ was destroyed */
358         srp = sp->s_srp[label];
359         if (srp == NULL)
360                 goto done;

362         /* Resend a software event on the correct queue */
363         index = srp->sr_index;
364         sep_targetq = sp->s_sep[index];

366         ASSERT((label & SFXGE_MAGIC_DMAQ_LABEL_MASK) == label);
367         magic = SFXGE_MAGIC_RX_QFLUSH_FAILED | label;

369         if (sep_targetq->se_state != SFXGE_EVQ_STARTED)
370                 goto done; /* TBD: state test not under the lock */

372         efx_ev_qpost(sep_targetq->se_eep, magic);

374 done:
375         return (B_FALSE);
376 }

378 static boolean_t
379 sfxge_ev_tx(void *arg, uint32_t label, uint32_t id)
380 {
381         sfxge_evq_t *sep = arg;
382         sfxge_t *sp = sep->se_sp;
383         sfxge_txq_t *stp;
384         unsigned int stop;
385         unsigned int delta;

387         ASSERT(mutex_owned(&(sep->se_lock)));

389         stp = sp->s_stp[label];
390         if (stp == NULL)
391                 goto done;
```

```
393         if (stp->st_state != SFXGE_TXQ_STARTED)
394                 goto done;

396         ASSERT3U(sep->se_index, ==, stp->st_evq);

398         stop = (id + 1) & (SFXGE_NDESCS - 1);
399         id = stp->st_pending & (SFXGE_NDESCS - 1);

401         delta = (stop >= id) ? (stop - id) : (SFXGE_NDESCS - id + stop);
402         stp->st_pending += delta;

404         sep->se_tx++;

406         if (stp->st_next == NULL &&
407             sep->se_stpp != &(stp->st_next)) {
408                 *(sep->se_stpp) = stp;
409                 sep->se_stpp = &(stp->st_next);
410         }

412         DTRACE_PROBE2(qlevel, unsigned int, stp->st_index,
413             unsigned int, stp->st_added - stp->st_pending);

415         if (stp->st_pending - stp->st_completed >= SFXGE_TX_BATCH)
416                 sfxge_tx_qcomplete(stp);

418 done:
419         /* returning B_TRUE makes efx_ev_qpoll() stop processing events */
420         return (sep->se_tx >= sep->se_ev_batch);
421 }

423 static boolean_t
424 sfxge_ev_txq_flush_done(void *arg, uint32_t label)
425 {
426         sfxge_evq_t *sep = arg;
427         sfxge_t *sp = sep->se_sp;
428         sfxge_txq_t *stp;
429         unsigned int evq;
430         uint16_t magic;

432         ASSERT(mutex_owned(&(sep->se_lock)));

434         /* Ensure TXQ exists, as events may arrive after TXQ was destroyed */
435         stp = sp->s_stp[label];
436         if (stp == NULL)
437                 goto done;

439         ASSERT3U(stp->st_state, ==, SFXGE_TXQ_INITIALIZED);

441         /* Resend a software event on the correct queue */
442         evq = stp->st_evq;
443         sep = sp->s_sep[evq];

445         ASSERT((label & SFXGE_MAGIC_DMAQ_LABEL_MASK) == label);
446         magic = SFXGE_MAGIC_TX_QFLUSH_DONE | label;

448         ASSERT3U(sep->se_state, ==, SFXGE_EVQ_STARTED);
449         efx_ev_qpost(sep->se_eep, magic);

451 done:
452         return (B_FALSE);
453 }

455 static boolean_t
456 sfxge_ev_software(void *arg, uint16_t magic)
457 {
```



```
458         sfxge_evq_t *sep = arg;
459         sfxge_t *sp = sep->se_sp;
460         dev_info_t *dip = sp->s_dip;
461         unsigned int label;

463         ASSERT(mutex_owned(&(sep->se_lock)));

465         EFX_STATIC_ASSERT(SFXGE_MAGIC_DMAQ_LABEL_WIDTH ==
466             FSF_AZ_RX_EV_Q_LABEL_WIDTH);
467         EFX_STATIC_ASSERT(SFXGE_MAGIC_DMAQ_LABEL_WIDTH ==
468             FSF_AZ_TX_EV_Q_LABEL_WIDTH);

470         label = magic & SFXGE_MAGIC_DMAQ_LABEL_MASK;
471         magic &= ~SFXGE_MAGIC_DMAQ_LABEL_MASK;

473         switch (magic) {
474         case SFXGE_MAGIC_RX_QFLUSH_DONE: {
475                 sfxge_rxq_t *srp = sp->s_srp[label];

477                 if (srp != NULL) {
478                         ASSERT3U(sep->se_index, ==, srp->sr_index);

480                         sfxge_rx_qflush_done(srp);
481                 }
482                 break;
483         }
484         case SFXGE_MAGIC_RX_QFLUSH_FAILED: {
485                 sfxge_rxq_t *srp = sp->s_srp[label];

487                 if (srp != NULL) {
488                         ASSERT3U(sep->se_index, ==, srp->sr_index);

490                         sfxge_rx_qflush_failed(srp);
491                 }
492                 break;
493         }
494         case SFXGE_MAGIC_RX_QFPP_TRIM: {
495                 sfxge_rxq_t *srp = sp->s_srp[label];

497                 if (srp != NULL) {
498                         ASSERT3U(sep->se_index, ==, srp->sr_index);

500                         sfxge_rx_qfpp_trim(srp);
501                 }
502                 break;
503         }
504         case SFXGE_MAGIC_TX_QFLUSH_DONE: {
505                 sfxge_txq_t *stp = sp->s_stp[label];

507                 if (stp != NULL) {
508                         ASSERT3U(sep->se_index, ==, stp->st_evq);

510                         sfxge_tx_qflush_done(stp);
511                 }
512                 break;
513         }
514         default:
515                 cmn_err(CE_NOTE,
516                         SFXGE_CMN_ERR "[%s%d] unknown software event 0x%x",
517                         ddi_driver_name(dip), ddi_get_instance(dip), magic);
518                 break;
519         }

521         return (B_FALSE);
522 }
```

```
524 static boolean_t
525 sfxge_ev_sram(void *arg, uint32_t code)
526 {
527         _NOTE(ARGUNUSED(arg))

529         switch (code) {
530         case EFX_SRAM_UPDATE:
531                 DTRACE_PROBE(sram_update);
532                 break;

534         case EFX_SRAM_CLEAR:
535                 DTRACE_PROBE(sram_clear);
536                 break;

538         case EFX_SRAM_ILLEGAL_CLEAR:
539                 DTRACE_PROBE(sram_illegal_clear);
540                 break;

542         default:
543                 ASSERT(B_FALSE);
544                 break;
545         }

547         return (B_FALSE);
548 }

550 static boolean_t
551 sfxge_ev_timer(void *arg, uint32_t index)
552 {
553         _NOTE(ARGUNUSED(arg, index))

555         return (B_FALSE);
556 }

558 static boolean_t
559 sfxge_ev_wake_up(void *arg, uint32_t index)
560 {
561         _NOTE(ARGUNUSED(arg, index))

563         return (B_FALSE);
564 }

566 static boolean_t
567 sfxge_ev_monitor(void *arg, efx_mon_stat_t id, efx_mon_stat_value_t value)
568 {
569         _NOTE(ARGUNUSED(arg, id, value))

571         return (B_FALSE);
572 }

574 static boolean_t
575 sfxge_ev_link_change(void *arg, efx_link_mode_t link_mode)
576 {
577         sfxge_evq_t *sep = arg;
578         sfxge_t *sp = sep->se_sp;

580         sfxge_mac_link_update(sp, link_mode);

582         return (B_FALSE);
583 }

585 static int
586 sfxge_ev_kstat_update(kstat_t *ksp, int rw)
587 {
588         sfxge_evq_t *sep = ksp->ks_private;
589         kstat_named_t *knp;
```



```
590         int rc;

592         if (rw != KSTAT_READ) {
593                 rc = EACCES;
594                 goto fail1;
595         }

597         ASSERT(mutex_owned(&(sep->se_lock)));

599         if (sep->se_state != SFXGE_EVQ_STARTED)
600                 goto done;

602         efx_ev_qstats_update(sep->se_eep, sep->se_stat);

604         knp = ksp->ks_data;
605         knp += EV_NQSTATS;

607         knp->value.ui64 = sep->se_cpu_id;

609 done:
610         return (0);

612 fail1:
613         DTRACE_PROBE1(fail1, int, rc);

615         return (rc);
616 }

618 static int
619 sfxge_ev_kstat_init(sfxge_evq_t *sep)
620 {
621         sfxge_t *sp = sep->se_sp;
622         unsigned int index = sep->se_index;
623         dev_info_t *dip = sp->s_dip;
624         kstat_t *ksp;
625         kstat_named_t *knp;
626         char name[MAXNAMELEN];
627         unsigned int id;
628         int rc;

630         /* Determine the name */
631         (void) snprintf(name, MAXNAMELEN - 1, "%s_evq%04d",
632             ddi_driver_name(dip), index);

634         /* Create the set */
635         if ((ksp = kstat_create((char *)ddi_driver_name(dip),
636             ddi_get_instance(dip), name, "queue", KSTAT_TYPE_NAMED,
637             EV_NQSTATS + 1, 0)) == NULL) {
638                 rc = ENOMEM;
639                 goto fail1;
640         }

642         sep->se_ksp = ksp;

644         ksp->ks_update = sfxge_ev_kstat_update;
645         ksp->ks_private = sep;
646         ksp->ks_lock = &(sep->se_lock);

648         /* Initialise the named stats */
649         sep->se_stat = knp = ksp->ks_data;
650         for (id = 0; id < EV_NQSTATS; id++) {
651                 kstat_named_init(knp, (char *)efx_ev_qstat_name(sp->s_enp, id),
652                     KSTAT_DATA_UINT64);
653                 knp++;
654         }
```

```
656         kstat_named_init(knp, "cpu", KSTAT_DATA_UINT64);

658         kstat_install(ksp);
659         return (0);

661 fail1:
662         DTRACE_PROBE1(fail1, int, rc);

664         return (rc);
665 }

667 static void
668 sfxge_ev_kstat_fini(sfxge_evq_t *sep)
669 {
670         /* Destroy the set */
671         kstat_delete(sep->se_ksp);
672         sep->se_ksp = NULL;
673         sep->se_stat = NULL;
674 }

676 inline unsigned pow2_ge(unsigned int n) {
677         unsigned int order = 0;
678         ASSERT3U(n, >, 0);
679         while ((1ul << order) < n) ++order;
680         return (1ul << (order));
681 }

683 static int
684 sfxge_ev_qinit(sfxge_t *sp, unsigned int index, unsigned int ev_batch)
685 {
686         sfxge_evq_t *sep;
687         int rc;

689         ASSERT3U(index, <, SFXGE_RX_SCALE_MAX);

691         sep = kmem_cache_alloc(index ? sp->s_eqXc : sp->s_eq0c, KM_SLEEP);
692         ASSERT3U(sep->se_state, ==, SFXGE_EVQ_UNINITIALIZED);

694         sep->se_index = index;

696         mutex_init(&(sep->se_lock), NULL,
697             MUTEX_DRIVER, DDI_INTR_PRI(sp->s_intr.si_intr_pri));

699         cv_init(&(sep->se_init_kv), NULL, CV_DRIVER, NULL);

701         /* Initialize the statistics */
702         if ((rc = sfxge_ev_kstat_init(sep)) != 0)
703                 goto fail1;

705         sep->se_state = SFXGE_EVQ_INITIALIZED;
706         sep->se_ev_batch = ev_batch;
707         sp->s_sep[index] = sep;

709         return (0);

711 fail1:
712         DTRACE_PROBE1(fail1, int, rc);

714         sep->se_index = 0;

716         cv_destroy(&(sep->se_init_kv));
717         mutex_destroy(&(sep->se_lock));

719         kmem_cache_free(index ? sp->s_eqXc : sp->s_eq0c, sep);

721         return (rc);
```



```
722 }

724 static int
725 sfxge_ev_qstart(sfxge_t *sp, unsigned int index)
726 {
727         sfxge_evq_t *sep = sp->s_sep[index];
728         sfxge_intr_t *sip = &(sp->s_intr);
729         efx_nic_t *enp = sp->s_enp;
730         efx_ev_callbacks_t *eecp;
731         efsys_mem_t *esmp;
732         clock_t timeout;
733         int rc;
734         uint16_t evq_size = index ? sp->s_evqX_size : sp->s_evq0_size;

736         mutex_enter(&(sep->se_lock));
737         esmp = &(sep->se_mem);

739         ASSERT3U(sep->se_state, ==, SFXGE_EVQ_INITIALIZED);

741         /* Set the memory to all ones */
742         (void) memset(esmp->esm_base, 0xff, EFX_EVQ_SIZE(evq_size));

744         /* Program the buffer table */
745         if ((rc = sfxge_sram_buf_tbl_set(sp, sep->se_id, esmp,
746             EFX_EVQ_NBUFS(evq_size))) != 0)
747                 goto fail1;

749         /* Set up the event callbacks */
750         eecp = &(sep->se_eec);
751         eecp->eec_initialized = sfxge_ev_initialized;
752         eecp->eec_rx = sfxge_ev_rx;
753         eecp->eec_tx = sfxge_ev_tx;
754         eecp->eec_exception = sfxge_ev_exception;
755         eecp->eec_rxq_flush_done = sfxge_ev_rxq_flush_done;
756         eecp->eec_rxq_flush_failed = sfxge_ev_rxq_flush_failed;
757         eecp->eec_txq_flush_done = sfxge_ev_txq_flush_done;
758         eecp->eec_software = sfxge_ev_software;
759         eecp->eec_sram = sfxge_ev_sram;
760         eecp->eec_wake_up = sfxge_ev_wake_up;
761         eecp->eec_timer = sfxge_ev_timer;
762         eecp->eec_link_change = sfxge_ev_link_change;
763         eecp->eec_monitor = sfxge_ev_monitor;

765         /* Create the event queue */
766         if ((rc = efx_ev_qcreate(enp, index, esmp, evq_size, sep->se_id,
767             &(sep->se_eep))) != 0)
768                 goto fail2;

770         /* Set the default moderation */
771         if ((rc = efx_ev_qmoderate(sep->se_eep, sp->s_ev_moderation)) != 0)
772                 goto fail3;

774         /* Check that interrupts are enabled at the NIC */
775         if (sip->si_state != SFXGE_INTR_STARTED) {
776                 rc = EINVAL;
777                 goto fail4;
778         }

780         sep->se_state = SFXGE_EVQ_STARTING;

782         /* Prime the event queue for interrupts */
783         if ((rc = efx_ev_qprime(sep->se_eep, sep->se_count)) != 0)
784                 goto fail5;

786         /* Wait for the initialization event */
787         timeout = ddi_get_lbolt() + drv_usectohz(2000000);
```

```
788         while (sep->se_state != SFXGE_EVQ_STARTED) {
789                 if (cv_timedwait(&(sep->se_init_kv), &(sep->se_lock),
790                         timeout) < 0) {
791                         /* Timeout waiting for initialization */
792                         dev_info_t *dip = sp->s_dip;

794                         DTRACE_PROBE(timeout);
795                         cmn_err(CE_NOTE,
796                             SFXGE_CMN_ERR "[%s%d] ev qstart timeout",
797                             ddi_driver_name(dip), ddi_get_instance(dip));
798                         rc = ETIMEDOUT;
799                         goto fail6;
800                 }
801         }

803         mutex_exit(&(sep->se_lock));
804         return (0);

806 fail6:
807         DTRACE_PROBE(fail6);

809 fail5:
810         DTRACE_PROBE(fail5);

812         sep->se_state = SFXGE_EVQ_INITIALIZED;

814 fail4:
815         DTRACE_PROBE(fail4);

817 fail3:
818         DTRACE_PROBE(fail3);

820         /* Destroy the event queue */
821         efx_ev_qdestroy(sep->se_eep);
822         sep->se_eep = NULL;

824 fail2:
825         DTRACE_PROBE(fail2);

827         /* Zero out the event handlers */
828         bzero(&(sep->se_eec), sizeof (efx_ev_callbacks_t));

830         /* Clear entries from the buffer table */
831         sfxge_sram_buf_tbl_clear(sp, sep->se_id, EFX_EVQ_NBUFS(evq_size));

833 fail1:
834         DTRACE_PROBE1(fail1, int, rc);

836         mutex_exit(&(sep->se_lock));

838         return (rc);
839 }

841 int
842 sfxge_ev_qpoll(sfxge_t *sp, unsigned int index)
843 {
844         sfxge_evq_t *sep = sp->s_sep[index];
845         processorid_t cpu_id;
846         int rc;
847         uint16_t evq_size = index ? sp->s_evqX_size : sp->s_evq0_size;

849         mutex_enter(&(sep->se_lock));

851         if (sep->se_state != SFXGE_EVQ_STARTING &&
852             sep->se_state != SFXGE_EVQ_STARTED) {
853                 rc = EINVAL;
```



```
854                 goto fail1;
855         }

857         /* Make sure the CPU information is up to date */
858         cpu_id = CPU->cpu_id;

860         if (cpu_id != sep->se_cpu_id) {
861 #ifdef  _USE_CPU_PHYSID
862                 cpu_physid_t *cpp = CPU->cpu_physid;
863 #endif

865                 sep->se_cpu_id = cpu_id;

867 #ifdef  _USE_CPU_PHYSID
868                 sep->se_core_id = cpp->cpu_coreid;
869                 sep->se_cache_id = cpp->cpu_cacheid;
870                 sep->se_chip_id = cpp->cpu_chipid;
871 #endif

873                 /* sfxge_evq_t->se_lock held */
874                 (void) ddi_taskq_dispatch(sp->s_tqp, sfxge_rx_scale_update, sp,
875                     DDI_NOSLEEP);
876         }

878         /* Synchronize the DMA memory for reading */
879         (void) ddi_dma_sync(sep->se_mem.esm_dma_handle,
880             0,
881             EFX_EVQ_SIZE(evq_size),
882             DDI_DMA_SYNC_FORKERNEL);

884         ASSERT3U(sep->se_rx, ==, 0);
885         ASSERT3U(sep->se_tx, ==, 0);
886         ASSERT3P(sep->se_stp, ==, NULL);
887         ASSERT3P(sep->se_stpp, ==, &(sep->se_stp));

889         /* Poll the queue */
890         efx_ev_qpoll(sep->se_eep, &(sep->se_count), &(sep->se_eec),
891             sep);

893         sep->se_rx = 0;
894         sep->se_tx = 0;

896         /* Perform any pending completion processing */
897         sfxge_ev_qcomplete(sep, B_TRUE);

899         /* Re-prime the event queue for interrupts */
900         if ((rc = efx_ev_qprime(sep->se_eep, sep->se_count)) != 0)
901                 goto fail2;

903         mutex_exit(&(sep->se_lock));

905         return (0);

907 fail2:
908         DTRACE_PROBE(fail2);
909 fail1:
910         DTRACE_PROBE1(fail1, int, rc);

912         mutex_exit(&(sep->se_lock));

914         return (rc);
915 }

917 int
918 sfxge_ev_qprime(sfxge_t *sp, unsigned int index)
919 {
```

```
920         sfxge_evq_t *sep = sp->s_sep[index];
921         int rc;

923         mutex_enter(&(sep->se_lock));

925         if (sep->se_state != SFXGE_EVQ_STARTING &&
926             sep->se_state != SFXGE_EVQ_STARTED) {
927                 rc = EINVAL;
928                 goto fail1;
929         }

931         if ((rc = efx_ev_qprime(sep->se_eep, sep->se_count)) != 0)
932                 goto fail2;

934         mutex_exit(&(sep->se_lock));

936         return (0);

938 fail2:
939         DTRACE_PROBE(fail2);
940 fail1:
941         DTRACE_PROBE1(fail1, int, rc);

943         mutex_exit(&(sep->se_lock));

945         return (rc);
946 }


949 int
950 sfxge_ev_qmoderate(sfxge_t *sp, unsigned int index, unsigned int us)
951 {
952         sfxge_evq_t *sep = sp->s_sep[index];
953         efx_evq_t *eep = sep->se_eep;

955         ASSERT3U(sep->se_state, ==, SFXGE_EVQ_STARTED);

957         return (efx_ev_qmoderate(eep, us));
958 }

960 static void
961 sfxge_ev_qstop(sfxge_t *sp, unsigned int index)
962 {
963         sfxge_evq_t *sep = sp->s_sep[index];
964         uint16_t evq_size;

966         mutex_enter(&(sep->se_lock));
967         ASSERT3U(sep->se_state, ==, SFXGE_EVQ_STARTED);
968         sep->se_state = SFXGE_EVQ_INITIALIZED;
969         evq_size = index ? sp->s_evqX_size : sp->s_evq0_size;

971         /* Clear the CPU information */
972 #ifdef  _USE_CPU_PHYSID
973         sep->se_cache_id = 0;
974         sep->se_core_id = 0;
975         sep->se_chip_id = 0;
976 #endif

978         sep->se_cpu_id = 0;

980         /* Clear the event count */
981         sep->se_count = 0;

983         /* Reset the exception flag */
984         sep->se_exception = B_FALSE;
```



```
986         /* Destroy the event queue */
987         efx_ev_qdestroy(sep->se_eep);
988         sep->se_eep = NULL;

990         mutex_exit(&(sep->se_lock));

992         /* Zero out the event handlers */
993         bzero(&(sep->se_eec), sizeof (efx_ev_callbacks_t));

995         /* Clear entries from the buffer table */
996         sfxge_sram_buf_tbl_clear(sp, sep->se_id, EFX_EVQ_NBUFS(evq_size));
997 }

999 static void
1000 sfxge_ev_qfini(sfxge_t *sp, unsigned int index)
1001 {
1002         sfxge_evq_t *sep = sp->s_sep[index];

1004         ASSERT3U(sep->se_state, ==, SFXGE_EVQ_INITIALIZED);

1006         sp->s_sep[index] = NULL;
1007         sep->se_state = SFXGE_EVQ_UNINITIALIZED;

1009         /* Tear down the statistics */
1010         sfxge_ev_kstat_fini(sep);

1012         cv_destroy(&(sep->se_init_kv));
1013         mutex_destroy(&(sep->se_lock));

1015         sep->se_index = 0;

1017         kmem_cache_free(index ? sp->s_eqXc : sp->s_eq0c, sep);
1018 }


1021 static  kmem_cache_t *
1022 sfxge_ev_kmem_cache_create(sfxge_t *sp, const char *qname,
1023     int (*ctor)(void *, void *, int), void (*dtor)(void *, void *))
1024 {
1025         char name[MAXNAMELEN];
1026         kmem_cache_t *eqc;

1028         (void) snprintf(name, MAXNAMELEN - 1, "%s%d_%s_cache",
1029             ddi_driver_name(sp->s_dip), ddi_get_instance(sp->s_dip), qname);

1031         eqc = kmem_cache_create(name, sizeof (sfxge_evq_t),
1032             SFXGE_CPU_CACHE_SIZE, ctor, dtor, NULL, sp, NULL, 0);
1033         ASSERT(eqc != NULL);
1034         return (eqc);
1035 }

1037 int
1038 sfxge_ev_init(sfxge_t *sp)
1039 {
1040         sfxge_intr_t *sip = &(sp->s_intr);
1041         unsigned int evq0_size;
1042         unsigned int evqX_size;
1043         unsigned int ev_batch;
1044         int index;
1045         int rc;

1047         ASSERT3U(sip->si_state, ==, SFXGE_INTR_INITIALIZED);

1049         /*
1050          * Must account for RXQ, TXQ(s); MCDI not event completed at present
1051          * Note that common code does not completely fill descriptor queues
```

```
1052          */
1053         evqX_size = sp->s_rxq_size + SFXGE_TX_NDESCS;
1054         evq0_size = evqX_size + SFXGE_TX_NDESCS; /* only IP checksum TXQ */
1055         evq0_size += SFXGE_TX_NDESCS; /* no checksums */

1057         ASSERT3U(evqX_size, >=, EFX_EVQ_MINNEVS);
1058         ASSERT3U(evq0_size, >, evqX_size);

1060         if (evq0_size > EFX_EVQ_MAXNEVS) {
1061                 rc = EINVAL;
1062                 goto fail1;
1063         }

1065         sp->s_evq0_size = pow2_ge(evq0_size);
1066         sp->s_evqX_size = pow2_ge(evqX_size);

1068         /* Read driver parameters */
1069         sp->s_ev_moderation = ddi_prop_get_int(DDI_DEV_T_ANY, sp->s_dip,
1070             DDI_PROP_DONTPASS, "intr_moderation", SFXGE_DEFAULT_MODERATION);

1072         ev_batch = ddi_prop_get_int(DDI_DEV_T_ANY, sp->s_dip,
1073             DDI_PROP_DONTPASS, "ev_batch", SFXGE_EV_BATCH);

1075         /*
1076          * It is slightly peverse to have a cache for one item. But it allows
1077          * for simple alignment control without increasing the allocation size
1078          */
1079         sp->s_eq0c = sfxge_ev_kmem_cache_create(sp, "evq0", sfxge_ev_q0ctor,
1080             sfxge_ev_q0dtor);
1081         sp->s_eqXc = sfxge_ev_kmem_cache_create(sp, "evqX", sfxge_ev_qXctor,
1082             sfxge_ev_qXdtor);

1084         /* Initialize the event queue(s) */
1085         for (index = 0; index < sip->si_nalloc; index++) {
1086                 if ((rc = sfxge_ev_qinit(sp, index, ev_batch)) != 0)
1087                         goto fail2;
1088         }

1090         return (0);

1092 fail2:
1093         DTRACE_PROBE(fail2);

1095         while (--index >= 0)
1096                 sfxge_ev_qfini(sp, index);
1097         sp->s_ev_moderation = 0;

1099 fail1:
1100         DTRACE_PROBE1(fail1, int, rc);

1102         kmem_cache_destroy(sp->s_eqXc);
1103         kmem_cache_destroy(sp->s_eq0c);
1104         sp->s_eqXc = NULL;
1105         sp->s_eq0c = NULL;

1107         return (rc);
1108 }

1110 int
1111 sfxge_ev_start(sfxge_t *sp)
1112 {
1113         sfxge_intr_t *sip = &(sp->s_intr);
1114         efx_nic_t *enp = sp->s_enp;
1115         int index;
1116         int rc;
```



```
1118         ASSERT3U(sip->si_state, ==, SFXGE_INTR_STARTED);

1120         /* Initialize the event module */
1121         if ((rc = efx_ev_init(enp)) != 0)
1122                 goto fail1;

1124         /* Start the event queues */
1125         for (index = 0; index < sip->si_nalloc; index++) {
1126                 if ((rc = sfxge_ev_qstart(sp, index)) != 0)
1127                         goto fail2;
1128         }

1130         return (0);

1132 fail2:
1133         DTRACE_PROBE(fail2);

1135         /* Stop the event queue(s) */
1136         while (--index >= 0)
1137                 sfxge_ev_qstop(sp, index);

1139         /* Tear down the event module */
1140         efx_ev_fini(enp);

1142 fail1:
1143         DTRACE_PROBE1(fail1, int, rc);

1145         return (rc);
1146 }

1148 void
1149 sfxge_ev_moderation_get(sfxge_t *sp, unsigned int *usp)
1150 {
1151         *usp = sp->s_ev_moderation;
1152 }

1154 int
1155 sfxge_ev_moderation_set(sfxge_t *sp, unsigned int us)
1156 {
1157         sfxge_intr_t *sip = &(sp->s_intr);
1158         int index;
1159         int rc;

1161         if (sip->si_state != SFXGE_INTR_STARTED)
1162                 return (ENODEV);

1164         for (index = 0; index < sip->si_nalloc; index++) {
1165                 if ((rc = sfxge_ev_qmoderate(sp, index, us)) != 0)
1166                         goto fail1;
1167         }

1169         sp->s_ev_moderation = us;
1170         return (0);

1172 fail1:
1173         DTRACE_PROBE1(fail1, int, rc);

1175         /*  The only error path is if the value to set to is invalid. */
1176         ASSERT3U(index, ==, 0);

1178         return (rc);
1179 }

1181 void
1182 sfxge_ev_stop(sfxge_t *sp)
1183 {
```

```
1184         sfxge_intr_t *sip = &(sp->s_intr);
1185         efx_nic_t *enp = sp->s_enp;
1186         int index;

1188         ASSERT3U(sip->si_state, ==, SFXGE_INTR_STARTED);

1190         /* Stop the event queue(s) */
1191         index = sip->si_nalloc;
1192         while (--index >= 0)
1193                 sfxge_ev_qstop(sp, index);

1195         /* Tear down the event module */
1196         efx_ev_fini(enp);
1197 }

1199 void
1200 sfxge_ev_fini(sfxge_t *sp)
1201 {
1202         sfxge_intr_t *sip = &(sp->s_intr);
1203         int index;

1205         ASSERT3U(sip->si_state, ==, SFXGE_INTR_INITIALIZED);

1207         sp->s_ev_moderation = 0;

1209         /* Tear down the event queue(s) */
1210         index = sip->si_nalloc;
1211         while (--index >= 0)
1212                 sfxge_ev_qfini(sp, index);

1214         kmem_cache_destroy(sp->s_eqXc);
1215         kmem_cache_destroy(sp->s_eq0c);
1216         sp->s_eqXc = NULL;
1217         sp->s_eq0c = NULL;
1218 }
1219 #endif /* ! codereview */
```

```
**********************************************************
   20755 Thu Aug 22 18:59:27 2013
new/usr/src/uts/common/io/sfxge/sfxge_gld_ndd.c
Merged sfxge driver
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2008-2013 Solarflare Communications Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#include <sys/types.h>
#include <sys/ddi.h>
#include <sys/sunddi.h>
#include <sys/gld.h>

#include <inet/nd.h>
#include <inet/mi.h>

#include "sfxge.h"


typedef enum sfxge_prop_e {
        SFXGE_RX_COALESCE_MODE = 0,
        SFXGE_RX_SCALE_COUNT,
        SFXGE_FCNTL_RESPOND,
        SFXGE_FCNTL_GENERATE,
        SFXGE_INTR_MODERATION,
        SFXGE_ADV_AUTONEG,
        SFXGE_ADV_10GFDX,
        SFXGE_ADV_1000FDX,
        SFXGE_ADV_1000HDX,
        SFXGE_ADV_100FDX,
        SFXGE_ADV_100HDX,
        SFXGE_ADV_10FDX,
        SFXGE_ADV_10HDX,
        SFXGE_ADV_PAUSE,
        SFXGE_ADV_ASM_PAUSE,
        SFXGE_LP_AUTONEG,
        SFXGE_LP_10GFDX,
        SFXGE_LP_1000FDX,
        SFXGE_LP_1000HDX,
        SFXGE_LP_100FDX,
        SFXGE_LP_100HDX,
        SFXGE_LP_10FDX,
        SFXGE_LP_10HDX,
```



```
        SFXGE_LP_PAUSE,
        SFXGE_LP_ASM_PAUSE,
        SFXGE_CAP_AUTONEG,
        SFXGE_CAP_10GFDX,
        SFXGE_CAP_1000FDX,
        SFXGE_CAP_1000HDX,
        SFXGE_CAP_100FDX,
        SFXGE_CAP_100HDX,
        SFXGE_CAP_10FDX,
        SFXGE_CAP_10HDX,
        SFXGE_CAP_PAUSE,
        SFXGE_CAP_ASM_PAUSE,
        SFXGE_NPROPS
} sfxge_prop_t;


static int
sfxge_gld_nd_get(sfxge_t *sp, unsigned int id, uint32_t *valp)
{
        efx_nic_t *enp = sp->s_enp;
        unsigned int nprops;
        int rc;

        nprops = efx_nic_cfg_get(enp)->enc_phy_nprops;

        if (sp->s_mac.sm_state != SFXGE_MAC_STARTED) {
                rc = ENODEV;
                goto fail1;
        }

        ASSERT3U(id, <, nprops + SFXGE_NPROPS);

        if (id < nprops) {
                if ((rc = efx_phy_prop_get(enp, id, 0, valp)) != 0)
                        goto fail2;
        } else {
                id -= nprops;

                switch (id) {
                case SFXGE_RX_COALESCE_MODE: {
                        sfxge_rx_coalesce_mode_t mode;

                        sfxge_rx_coalesce_mode_get(sp, &mode);

                        *valp = mode;
                        break;
                }
                case SFXGE_RX_SCALE_COUNT: {
                        unsigned int count;

                        if (sfxge_rx_scale_count_get(sp, &count) != 0)
                                count = 0;

                        *valp = count;
                        break;
                }
                case SFXGE_FCNTL_RESPOND: {
                        unsigned int fcntl;

                        sfxge_mac_fcntl_get(sp, &fcntl);

                        *valp = (fcntl & EFX_FCNTL_RESPOND) ? 1 : 0;
                        break;
                }
                case SFXGE_FCNTL_GENERATE: {
                        unsigned int fcntl;
```

```
129                     sfxge_mac_fcntl_get(sp, &fcntl);

131                     *valp = (fcntl & EFX_FCNTL_GENERATE) ? 1 : 0;
132                     break;
133             }
134             case SFXGE_INTR_MODERATION: {
135                     unsigned int us;

137                     sfxge_ev_moderation_get(sp, &us);

139                     *valp = (long)us;
140                     break;
141             }
142             case SFXGE_ADV_AUTONEG: {
143                     uint32_t mask;

145                     efx_phy_adv_cap_get(enp, EFX_PHY_CAP_CURRENT, &mask);

147                     *valp = (mask & (1 << EFX_PHY_CAP_AN)) ? 1 : 0;
148                     break;
149             }
150             case SFXGE_ADV_10GFDX: {
151                     uint32_t mask;

153                     efx_phy_adv_cap_get(enp, EFX_PHY_CAP_CURRENT, &mask);

155                     *valp = (mask & (1 << EFX_PHY_CAP_10000FDX)) ? 1 : 0;
156                     break;
157             }
158             case SFXGE_ADV_1000FDX: {
159                     uint32_t mask;

161                     efx_phy_adv_cap_get(enp, EFX_PHY_CAP_CURRENT, &mask);

163                     *valp = (mask & (1 << EFX_PHY_CAP_1000FDX)) ? 1 : 0;
164                     break;
165             }
166             case SFXGE_ADV_1000HDX: {
167                     uint32_t mask;

169                     efx_phy_adv_cap_get(enp, EFX_PHY_CAP_CURRENT, &mask);

171                     *valp = (mask & (1 << EFX_PHY_CAP_1000HDX)) ? 1 : 0;
172                     break;
173             }
174             case SFXGE_ADV_100FDX: {
175                     uint32_t mask;

177                     efx_phy_adv_cap_get(enp, EFX_PHY_CAP_CURRENT, &mask);

179                     *valp = (mask & (1 << EFX_PHY_CAP_100FDX)) ? 1 : 0;
180                     break;
181             }
182             case SFXGE_ADV_100HDX: {
183                     uint32_t mask;

185                     efx_phy_adv_cap_get(enp, EFX_PHY_CAP_CURRENT, &mask);

187                     *valp = (mask & (1 << EFX_PHY_CAP_100HDX)) ? 1 : 0;
188                     break;
189             }
190             case SFXGE_ADV_10FDX: {
191                     uint32_t mask;

193                     efx_phy_adv_cap_get(enp, EFX_PHY_CAP_CURRENT, &mask);
```



```
195                     *valp = (mask & (1 << EFX_PHY_CAP_10FDX)) ? 1 : 0;
196                     break;
197             }
198             case SFXGE_ADV_10HDX: {
199                     uint32_t mask;

201                     efx_phy_adv_cap_get(enp, EFX_PHY_CAP_CURRENT, &mask);

203                     *valp = (mask & (1 << EFX_PHY_CAP_10HDX)) ? 1 : 0;
204                     break;
205             }
206             case SFXGE_ADV_PAUSE: {
207                     uint32_t mask;

209                     efx_phy_adv_cap_get(enp, EFX_PHY_CAP_CURRENT, &mask);

211                     *valp = (mask & (1 << EFX_PHY_CAP_PAUSE)) ? 1 : 0;
212                     break;
213             }
214             case SFXGE_ADV_ASM_PAUSE: {
215                     uint32_t mask;

217                     efx_phy_adv_cap_get(enp, EFX_PHY_CAP_CURRENT, &mask);

219                     *valp = (mask & (1 << EFX_PHY_CAP_ASYM)) ? 1 : 0;
220                     break;
221             }
222             case SFXGE_LP_AUTONEG: {
223                     uint32_t mask;

225                     efx_phy_adv_cap_get(enp, EFX_PHY_CAP_CURRENT, &mask);

227                     *valp = (mask & (1 << EFX_PHY_CAP_AN)) ? 1 : 0;
228                     break;
229             }
230             case SFXGE_LP_10GFDX: {
231                     uint32_t mask;

233                     efx_phy_lp_cap_get(enp, &mask);

235                     *valp = (mask & (1 << EFX_PHY_CAP_10000FDX)) ? 1 : 0;
236                     break;
237             }
238             case SFXGE_LP_1000FDX: {
239                     uint32_t mask;

241                     efx_phy_lp_cap_get(enp, &mask);

243                     *valp = (mask & (1 << EFX_PHY_CAP_1000FDX)) ? 1 : 0;
244                     break;
245             }
246             case SFXGE_LP_1000HDX: {
247                     uint32_t mask;

249                     efx_phy_lp_cap_get(enp, &mask);

251                     *valp = (mask & (1 << EFX_PHY_CAP_1000HDX)) ? 1 : 0;
252                     break;
253             }
254             case SFXGE_LP_100FDX: {
255                     uint32_t mask;

257                     efx_phy_lp_cap_get(enp, &mask);

259                     *valp = (mask & (1 << EFX_PHY_CAP_100FDX)) ? 1 : 0;
```

```
260                     break;
261             }
262             case SFXGE_LP_100HDX: {
263                     uint32_t mask;

265                     efx_phy_lp_cap_get(enp, &mask);

267                     *valp = (mask & (1 << EFX_PHY_CAP_100HDX)) ? 1 : 0;
268                     break;
269             }
270             case SFXGE_LP_10FDX: {
271                     uint32_t mask;

273                     efx_phy_lp_cap_get(enp, &mask);

275                     *valp = (mask & (1 << EFX_PHY_CAP_10FDX)) ? 1 : 0;
276                     break;
277             }
278             case SFXGE_LP_10HDX: {
279                     uint32_t mask;

281                     efx_phy_lp_cap_get(enp, &mask);

283                     *valp = (mask & (1 << EFX_PHY_CAP_10HDX)) ? 1 : 0;
284                     break;
285             }
286             case SFXGE_LP_PAUSE: {
287                     uint32_t mask;

289                     efx_phy_lp_cap_get(enp, &mask);

291                     *valp = (mask & (1 << EFX_PHY_CAP_PAUSE)) ? 1 : 0;
292                     break;
293             }
294             case SFXGE_LP_ASM_PAUSE: {
295                     uint32_t mask;

297                     efx_phy_lp_cap_get(enp, &mask);

299                     *valp = (mask & (1 << EFX_PHY_CAP_ASYM)) ? 1 : 0;
300                     break;
301             }
302             case SFXGE_CAP_AUTONEG: {
303                     uint32_t mask;

305                     efx_phy_adv_cap_get(enp, EFX_PHY_CAP_DEFAULT, &mask);

307                     *valp = (mask & (1 << EFX_PHY_CAP_AN)) ? 1 : 0;
308                     break;
309             }
310             case SFXGE_CAP_10GFDX: {
311                     uint32_t mask;

313                     efx_phy_adv_cap_get(enp, EFX_PHY_CAP_DEFAULT, &mask);

315                     *valp = (mask & (1 << EFX_PHY_CAP_10000FDX)) ? 1 : 0;
316                     break;
317             }
318             case SFXGE_CAP_1000FDX: {
319                     uint32_t mask;

321                     efx_phy_adv_cap_get(enp, EFX_PHY_CAP_DEFAULT, &mask);

323                     *valp = (mask & (1 << EFX_PHY_CAP_1000FDX)) ? 1 : 0;
324                     break;
325             }
```



```
326             case SFXGE_CAP_1000HDX: {
327                     uint32_t mask;

329                     efx_phy_adv_cap_get(enp, EFX_PHY_CAP_DEFAULT, &mask);

331                     *valp = (mask & (1 << EFX_PHY_CAP_1000HDX)) ? 1 : 0;
332                     break;
333             }
334             case SFXGE_CAP_100FDX: {
335                     uint32_t mask;

337                     efx_phy_adv_cap_get(enp, EFX_PHY_CAP_DEFAULT, &mask);

339                     *valp = (mask & (1 << EFX_PHY_CAP_100FDX)) ? 1 : 0;
340                     break;
341             }
342             case SFXGE_CAP_100HDX: {
343                     uint32_t mask;

345                     efx_phy_adv_cap_get(enp, EFX_PHY_CAP_DEFAULT, &mask);

347                     *valp = (mask & (1 << EFX_PHY_CAP_100HDX)) ? 1 : 0;
348                     break;
349             }
350             case SFXGE_CAP_10FDX: {
351                     uint32_t mask;

353                     efx_phy_adv_cap_get(enp, EFX_PHY_CAP_DEFAULT, &mask);

355                     *valp = (mask & (1 << EFX_PHY_CAP_10FDX)) ? 1 : 0;
356                     break;
357             }
358             case SFXGE_CAP_10HDX: {
359                     uint32_t mask;

361                     efx_phy_adv_cap_get(enp, EFX_PHY_CAP_DEFAULT, &mask);

363                     *valp = (mask & (1 << EFX_PHY_CAP_10HDX)) ? 1 : 0;
364                     break;
365             }
366             case SFXGE_CAP_PAUSE: {
367                     uint32_t mask;

369                     efx_phy_adv_cap_get(enp, EFX_PHY_CAP_DEFAULT, &mask);

371                     *valp = (mask & (1 << EFX_PHY_CAP_PAUSE)) ? 1 : 0;
372                     break;
373             }
374             case SFXGE_CAP_ASM_PAUSE: {
375                     uint32_t mask;

377                     efx_phy_adv_cap_get(enp, EFX_PHY_CAP_DEFAULT, &mask);

379                     *valp = (mask & (1 << EFX_PHY_CAP_ASYM)) ? 1 : 0;
380                     break;
381             }
382             default:
383                     ASSERT(B_FALSE);
384                     break;
385             }
386     }

388     return (0);
389 fail2:
390     DTRACE_PROBE(fail2);
391 fail1:
```

```
392         DTRACE_PROBE1(fail1, int, rc);

394         return (rc);
395 }

397 #ifdef _USE_NDD_PROPS
398 static int
399 sfxge_gld_nd_get_ioctl(queue_t *q, mblk_t *mp, caddr_t arg, cred_t *credp)
400 {
401         sfxge_ndd_param_t *snpp = (sfxge_ndd_param_t *)arg;
402         sfxge_t *sp = snpp->snp_sp;
403         unsigned int id = snpp->snp_id;
404         uint32_t val;
405         int rc;

407         _NOTE(ARGUNUSED(q, credp))

409         if ((rc = sfxge_gld_nd_get(sp, id, &val)) != 0)
410                 goto fail1;

412         (void) mi_mpprintf(mp, "%d", val);

414         return (0);

416 fail1:
417         DTRACE_PROBE1(fail1, int, rc);

419         return (rc);
420 }
421 #else
422 static int
423 sfxge_gld_nd_get_ioctl(queue_t *q, mblk_t *mp, caddr_t arg, cred_t *credp)
424 {
425         ASSERT(B_FALSE);
426         return (-ENODEV);
427 }
428 #endif


431 static int
432 sfxge_gld_nd_set(sfxge_t *sp, unsigned int id, uint32_t val)
433 {
434         efx_nic_t *enp = sp->s_enp;
435         unsigned int nprops;
436         int rc;

438         nprops = efx_nic_cfg_get(enp)->enc_phy_nprops;

440         ASSERT3U(id, <, nprops + SFXGE_NPROPS);

442         if (id < nprops) {
443                 if ((rc = efx_phy_prop_set(enp, id, val)) != 0)
444                         goto fail1;
445         } else {
446                 id -= nprops;

448                 switch (id) {
449                 case SFXGE_RX_COALESCE_MODE: {
450                         sfxge_rx_coalesce_mode_t mode =
451                             (sfxge_rx_coalesce_mode_t)val;

453                         if ((rc = sfxge_rx_coalesce_mode_set(sp, mode)) != 0)
454                                 goto fail1;

456                         break;
457                 }
```



```
458                 case SFXGE_RX_SCALE_COUNT: {
459                         unsigned int count = (unsigned int)val;

461                         if ((rc = sfxge_rx_scale_count_set(sp, count)) != 0)
462                                 goto fail1;

464                         break;
465                 }
466                 case SFXGE_FCNTL_RESPOND: {
467                         unsigned int fcntl;

469                         sfxge_mac_fcntl_get(sp, &fcntl);

471                         if (val != 0)
472                                 fcntl |= EFX_FCNTL_RESPOND;
473                         else
474                                 fcntl &= ~EFX_FCNTL_RESPOND;

476                         if ((rc = sfxge_mac_fcntl_set(sp, fcntl)) != 0)
477                                 goto fail1;

479                         break;
480                 }
481                 case SFXGE_FCNTL_GENERATE: {
482                         unsigned int fcntl;

484                         sfxge_mac_fcntl_get(sp, &fcntl);

486                         if (val != 0)
487                                 fcntl |= EFX_FCNTL_GENERATE;
488                         else
489                                 fcntl &= ~EFX_FCNTL_GENERATE;

491                         if ((rc = sfxge_mac_fcntl_set(sp, fcntl)) != 0)
492                                 goto fail1;

494                         break;
495                 }
496                 case SFXGE_INTR_MODERATION: {
497                         unsigned int us = (unsigned int)val;

499                         if ((rc = sfxge_ev_moderation_set(sp, us)) != 0)
500                                 goto fail1;
501                         break;
502                 }
503                 case SFXGE_ADV_AUTONEG: {
504                         uint32_t mask;

506                         efx_phy_adv_cap_get(enp, EFX_PHY_CAP_CURRENT, &mask);

508                         if (val != 0)
509                                 mask |= (1 << EFX_PHY_CAP_AN);
510                         else
511                                 mask &= ~(1 << EFX_PHY_CAP_AN);

513                         if ((rc = efx_phy_adv_cap_set(enp, mask)) != 0)
514                                 goto fail1;

516                         break;
517                 }
518                 case SFXGE_ADV_10GFDX: {
519                         uint32_t mask;

521                         efx_phy_adv_cap_get(enp, EFX_PHY_CAP_CURRENT, &mask);

523                         if (val != 0)
```

```
524                             mask |= (1 << EFX_PHY_CAP_10000FDX);
525                     else
526                             mask &= ~(1 << EFX_PHY_CAP_10000FDX);

528                     if ((rc = efx_phy_adv_cap_set(enp, mask)) != 0)
529                             goto fail1;

531                     break;
532             }
533             case SFXGE_ADV_1000FDX: {
534                     uint32_t mask;

536                     efx_phy_adv_cap_get(enp, EFX_PHY_CAP_CURRENT, &mask);

538                     if (val != 0)
539                             mask |= (1 << EFX_PHY_CAP_1000FDX);
540                     else
541                             mask &= ~(1 << EFX_PHY_CAP_1000FDX);

543                     if ((rc = efx_phy_adv_cap_set(enp, mask)) != 0)
544                             goto fail1;

546                     break;
547             }
548             case SFXGE_ADV_1000HDX: {
549                     uint32_t mask;

551                     efx_phy_adv_cap_get(enp, EFX_PHY_CAP_CURRENT, &mask);

553                     if (val != 0)
554                             mask |= (1 << EFX_PHY_CAP_1000HDX);
555                     else
556                             mask &= ~(1 << EFX_PHY_CAP_1000HDX);

558                     if ((rc = efx_phy_adv_cap_set(enp, mask)) != 0)
559                             goto fail1;

561                     break;
562             }
563             case SFXGE_ADV_100FDX: {
564                     uint32_t mask;

566                     efx_phy_adv_cap_get(enp, EFX_PHY_CAP_CURRENT, &mask);

568                     if (val != 0)
569                             mask |= (1 << EFX_PHY_CAP_100FDX);
570                     else
571                             mask &= ~(1 << EFX_PHY_CAP_100FDX);

573                     if ((rc = efx_phy_adv_cap_set(enp, mask)) != 0)
574                             goto fail1;

576                     break;
577             }
578             case SFXGE_ADV_100HDX: {
579                     uint32_t mask;

581                     efx_phy_adv_cap_get(enp, EFX_PHY_CAP_CURRENT, &mask);

583                     if (val != 0)
584                             mask |= (1 << EFX_PHY_CAP_100HDX);
585                     else
586                             mask &= ~(1 << EFX_PHY_CAP_100HDX);

588                     if ((rc = efx_phy_adv_cap_set(enp, mask)) != 0)
589                             goto fail1;
```



```
591                     break;
592             }
593             case SFXGE_ADV_10FDX: {
594                     uint32_t mask;

596                     efx_phy_adv_cap_get(enp, EFX_PHY_CAP_CURRENT, &mask);

598                     if (val != 0)
599                             mask |= (1 << EFX_PHY_CAP_10FDX);
600                     else
601                             mask &= ~(1 << EFX_PHY_CAP_10FDX);

603                     if ((rc = efx_phy_adv_cap_set(enp, mask)) != 0)
604                             goto fail1;

606                     break;
607             }
608             case SFXGE_ADV_10HDX: {
609                     uint32_t mask;

611                     efx_phy_adv_cap_get(enp, EFX_PHY_CAP_CURRENT, &mask);

613                     if (val != 0)
614                             mask |= (1 << EFX_PHY_CAP_10HDX);
615                     else
616                             mask &= ~(1 << EFX_PHY_CAP_10HDX);

618                     if ((rc = efx_phy_adv_cap_set(enp, mask)) != 0)
619                             goto fail1;

621                     break;
622             }
623             case SFXGE_ADV_PAUSE: {
624                     uint32_t mask;

626                     efx_phy_adv_cap_get(enp, EFX_PHY_CAP_CURRENT, &mask);

628                     if (val != 0)
629                             mask |= (1 << EFX_PHY_CAP_PAUSE);
630                     else
631                             mask &= ~(1 << EFX_PHY_CAP_PAUSE);

633                     if ((rc = efx_phy_adv_cap_set(enp, mask)) != 0)
634                             goto fail1;

636                     break;
637             }
638             case SFXGE_ADV_ASM_PAUSE: {
639                     uint32_t mask;

641                     efx_phy_adv_cap_get(enp, EFX_PHY_CAP_CURRENT, &mask);

643                     if (val != 0)
644                             mask |= (1 << EFX_PHY_CAP_ASYM);
645                     else
646                             mask &= ~(1 << EFX_PHY_CAP_ASYM);

648                     if ((rc = efx_phy_adv_cap_set(enp, mask)) != 0)
649                             goto fail1;

651                     break;
652             }
653             /* Ignore other kstat writes. Might be for the link partner */
654             default:
655                     DTRACE_PROBE1(ignore_kstat_write, int, id);
```

```
656                 }
657         }

659         return (0);

661 fail1:
662         DTRACE_PROBE1(fail1, int, rc);

664         return (rc);
665 }


668 #ifdef _USE_NDD_PROPS
669 static int
670 sfxge_gld_nd_set_ioctl(queue_t *q, mblk_t *mp, char *valp, caddr_t arg,
671     cred_t *credp)
672 {
673         sfxge_ndd_param_t *snpp = (sfxge_ndd_param_t *)arg;
674         sfxge_t *sp = snpp->snp_sp;
675         unsigned int id = snpp->snp_id;
676         long val;
677         int rc;

679         _NOTE(ARGUNUSED(q, mp, credp))

681         (void) ddi_strtol(valp, (char **)NULL, 0, &val);

683         if ((rc = sfxge_gld_nd_set(sp, id, (uint32_t)val)) != 0)
684                 goto fail1;

686         return (0);

688 fail1:
689         DTRACE_PROBE1(fail1, int, rc);

691         return (rc);
692 }
693 #else
694 static int
695 sfxge_gld_nd_set_ioctl(queue_t *q, mblk_t *mp, char *valp, caddr_t arg,
696     cred_t *credp)
697 {
698         ASSERT(B_FALSE);
699         return (-ENODEV);
700 }
701 #endif


704 static int
705 sfxge_gld_nd_update(kstat_t *ksp, int rw)
706 {
707         sfxge_t *sp = ksp->ks_private;
708         efx_nic_t *enp = sp->s_enp;
709         unsigned int nprops;
710         unsigned int id;
711         int rc = 0;

713         nprops = efx_nic_cfg_get(enp)->enc_phy_nprops;

715         for (id = 0; id < nprops + SFXGE_NPROPS; id++) {
716                 kstat_named_t *knp = &(sp->s_nd_stat[id]);

718                 if (rw == KSTAT_READ)
719                         rc = sfxge_gld_nd_get(sp, id, &(knp->value.ui32));
720                 else if (rw == KSTAT_WRITE)
721                         rc = sfxge_gld_nd_set(sp, id, knp->value.ui32);
```



```
722                 else
723                         rc = EACCES;

725                 if (rc != 0)
726                         goto fail1;
727         }

729         return (0);

731 fail1:
732         DTRACE_PROBE1(fail1, int, rc);
733         return (rc);
734 }


737 static sfxge_ndd_param_t        sfxge_ndd_param[] = {
738         {
739                 NULL,
740                 SFXGE_RX_COALESCE_MODE,
741                 "rx_coalesce_mode",
742                 sfxge_gld_nd_get_ioctl,
743                 sfxge_gld_nd_set_ioctl
744         },
745         {
746                 NULL,
747                 SFXGE_RX_SCALE_COUNT,
748                 "rx_scale_count",
749                 sfxge_gld_nd_get_ioctl,
750                 sfxge_gld_nd_set_ioctl
751         },
752         {
753                 NULL,
754                 SFXGE_FCNTL_RESPOND,
755                 "fcntl_respond",
756                 sfxge_gld_nd_get_ioctl,
757                 sfxge_gld_nd_set_ioctl
758         },
759         {
760                 NULL,
761                 SFXGE_FCNTL_GENERATE,
762                 "fcntl_generate",
763                 sfxge_gld_nd_get_ioctl,
764                 sfxge_gld_nd_set_ioctl
765         },
766         {
767                 NULL,
768                 SFXGE_INTR_MODERATION,
769                 "intr_moderation",
770                 sfxge_gld_nd_get_ioctl,
771                 sfxge_gld_nd_set_ioctl
772         },
773         {
774                 NULL,
775                 SFXGE_ADV_AUTONEG,
776                 "adv_cap_autoneg",
777                 sfxge_gld_nd_get_ioctl,
778                 sfxge_gld_nd_set_ioctl
779         },
780         {
781                 NULL,
782                 SFXGE_ADV_10GFDX,
783                 "adv_cap_10gfdx",
784                 sfxge_gld_nd_get_ioctl,
785                 sfxge_gld_nd_set_ioctl
786         },
787         {
```

```
788                 NULL,
789                 SFXGE_ADV_1000FDX,
790                 "adv_cap_1000fdx",
791                 sfxge_gld_nd_get_ioctl,
792                 sfxge_gld_nd_set_ioctl
793         },
794         {
795                 NULL,
796                 SFXGE_ADV_1000HDX,
797                 "adv_cap_1000hdx",
798                 sfxge_gld_nd_get_ioctl,
799                 sfxge_gld_nd_set_ioctl
800         },
801         {
802                 NULL,
803                 SFXGE_ADV_100FDX,
804                 "adv_cap_100fdx",
805                 sfxge_gld_nd_get_ioctl,
806                 sfxge_gld_nd_set_ioctl
807         },
808         {
809                 NULL,
810                 SFXGE_ADV_100HDX,
811                 "adv_cap_100hdx",
812                 sfxge_gld_nd_get_ioctl,
813                 sfxge_gld_nd_set_ioctl
814         },
815         {
816                 NULL,
817                 SFXGE_ADV_10FDX,
818                 "adv_cap_10fdx",
819                 sfxge_gld_nd_get_ioctl,
820                 sfxge_gld_nd_set_ioctl
821         },
822         {
823                 NULL,
824                 SFXGE_ADV_10HDX,
825                 "adv_cap_10hdx",
826                 sfxge_gld_nd_get_ioctl,
827                 sfxge_gld_nd_set_ioctl
828         },
829         {
830                 NULL,
831                 SFXGE_ADV_PAUSE,
832                 "adv_cap_pause",
833                 sfxge_gld_nd_get_ioctl,
834                 sfxge_gld_nd_set_ioctl
835         },
836         {
837                 NULL,
838                 SFXGE_ADV_ASM_PAUSE,
839                 "adv_cap_asm_pause",
840                 sfxge_gld_nd_get_ioctl,
841                 sfxge_gld_nd_set_ioctl
842         },
843         {
844                 NULL,
845                 SFXGE_LP_AUTONEG,
846                 "lp_cap_autoneg",
847                 sfxge_gld_nd_get_ioctl,
848                 NULL
849         },
850         {
851                 NULL,
852                 SFXGE_LP_10GFDX,
853                 "lp_cap_10gfdx",
```



```
854                 sfxge_gld_nd_get_ioctl,
855                 NULL
856         },
857         {
858                 NULL,
859                 SFXGE_LP_1000FDX,
860                 "lp_cap_1000fdx",
861                 sfxge_gld_nd_get_ioctl,
862                 NULL
863         },
864         {
865                 NULL,
866                 SFXGE_LP_1000HDX,
867                 "lp_cap_1000hdx",
868                 sfxge_gld_nd_get_ioctl,
869                 NULL
870         },
871         {
872                 NULL,
873                 SFXGE_LP_100FDX,
874                 "lp_cap_100fdx",
875                 sfxge_gld_nd_get_ioctl,
876                 NULL
877         },
878         {
879                 NULL,
880                 SFXGE_LP_100HDX,
881                 "lp_cap_100hdx",
882                 sfxge_gld_nd_get_ioctl,
883                 NULL
884         },
885         {
886                 NULL,
887                 SFXGE_LP_10FDX,
888                 "lp_cap_10fdx",
889                 sfxge_gld_nd_get_ioctl,
890                 NULL
891         },
892         {
893                 NULL,
894                 SFXGE_LP_10HDX,
895                 "lp_cap_10hdx",
896                 sfxge_gld_nd_get_ioctl,
897                 NULL
898         },
899         {
900                 NULL,
901                 SFXGE_LP_PAUSE,
902                 "lp_cap_pause",
903                 sfxge_gld_nd_get_ioctl,
904                 NULL
905         },
906         {
907                 NULL,
908                 SFXGE_LP_ASM_PAUSE,
909                 "lp_cap_asm_pause",
910                 sfxge_gld_nd_get_ioctl,
911                 NULL
912         },
913         {
914                 NULL,
915                 SFXGE_CAP_AUTONEG,
916                 "cap_autoneg",
917                 sfxge_gld_nd_get_ioctl,
918                 NULL
919         },
```

```
920         {
921                 NULL,
922                 SFXGE_CAP_10GFDX,
923                 "cap_10gfdx",
924                 sfxge_gld_nd_get_ioctl,
925                 NULL
926         },
927         {
928                 NULL,
929                 SFXGE_CAP_1000FDX,
930                 "cap_1000fdx",
931                 sfxge_gld_nd_get_ioctl,
932                 NULL
933         },
934         {
935                 NULL,
936                 SFXGE_CAP_1000HDX,
937                 "cap_1000hdx",
938                 sfxge_gld_nd_get_ioctl,
939                 NULL
940         },
941         {
942                 NULL,
943                 SFXGE_CAP_100FDX,
944                 "cap_100fdx",
945                 sfxge_gld_nd_get_ioctl,
946                 NULL
947         },
948         {
949                 NULL,
950                 SFXGE_CAP_100HDX,
951                 "cap_100hdx",
952                 sfxge_gld_nd_get_ioctl,
953                 NULL
954         },
955         {
956                 NULL,
957                 SFXGE_CAP_10FDX,
958                 "cap_10fdx",
959                 sfxge_gld_nd_get_ioctl,
960                 NULL
961         },
962         {
963                 NULL,
964                 SFXGE_CAP_10HDX,
965                 "cap_10hdx",
966                 sfxge_gld_nd_get_ioctl,
967                 NULL
968         },
969         {
970                 NULL,
971                 SFXGE_CAP_PAUSE,
972                 "cap_pause",
973                 sfxge_gld_nd_get_ioctl,
974                 NULL
975         },
976         {
977                 NULL,
978                 SFXGE_CAP_ASM_PAUSE,
979                 "cap_asm_pause",
980                 sfxge_gld_nd_get_ioctl,
981                 NULL
982         }
983 };
```



```
986 int
987 sfxge_gld_nd_register(sfxge_t *sp)
988 {
989 #ifdef _USE_NDD_PROPS
990         caddr_t *ndhp = &(sp->s_ndh);
991 #endif
992         efx_nic_t *enp = sp->s_enp;
993         unsigned int nprops;
994         unsigned int id;
995         char name[MAXNAMELEN];
996         kstat_t *ksp;
997         int rc;

999         ASSERT3P(sp->s_ndh, ==, NULL);

1001        nprops = efx_nic_cfg_get(enp)->enc_phy_nprops;

1003 #ifdef _USE_NDD_PROPS
1004        /* Register with the NDD framework */
1005        if ((sp->s_ndp = kmem_zalloc(sizeof (sfxge_ndd_param_t) *
1006            (nprops + SFXGE_NPROPS), KM_NOSLEEP)) == NULL) {
1007                rc = ENOMEM;
1008                goto fail1;
1009        }

1011        for (id = 0; id < nprops; id++) {
1012                sfxge_ndd_param_t *snpp = &(sp->s_ndp[id]);

1014                snpp->snp_sp = sp;
1015                snpp->snp_id = id;
1016                snpp->snp_name = efx_phy_prop_name(enp, id);
1017                snpp->snp_get = sfxge_gld_nd_get_ioctl;
1018                snpp->snp_set = sfxge_gld_nd_set_ioctl;

1020                ASSERT(snpp->snp_name != NULL);

1022                (void) nd_load(ndhp, (char *)(snpp->snp_name),
1023                    snpp->snp_get, snpp->snp_set, (caddr_t)snpp);
1024        }

1026        for (id = 0; id < SFXGE_NPROPS; id++) {
1027                sfxge_ndd_param_t *snpp = &(sp->s_ndp[id + nprops]);

1029                *snpp = sfxge_ndd_param[id];
1030                ASSERT3U(snpp->snp_id, ==, id);

1032                snpp->snp_sp = sp;
1033                snpp->snp_id += nprops;

1035                (void) nd_load(ndhp, (char *)(snpp->snp_name),
1036                    snpp->snp_get, snpp->snp_set, (caddr_t)snpp);
1037        }
1038 #endif

1040        /* Also create a kstat set */
1041        (void) snprintf(name, MAXNAMELEN - 1, "%s_ndd",
1042            ddi_driver_name(sp->s_dip));

1044        if ((ksp = kstat_create((char *)ddi_driver_name(sp->s_dip),
1045            ddi_get_instance(sp->s_dip), name, "ndd", KSTAT_TYPE_NAMED,
1046            (nprops + SFXGE_NPROPS), KSTAT_FLAG_WRITABLE)) == NULL) {
1047                rc = ENOMEM;
1048                goto fail2;
1049        }

1051        sp->s_nd_ksp = ksp;
```

```
1053         ksp->ks_update = sfxge_gld_nd_update;
1054         ksp->ks_private = sp;

1056         sp->s_nd_stat = ksp->ks_data;

1058         for (id = 0; id < nprops; id++) {
1059                 kstat_named_t *knp = &(sp->s_nd_stat[id]);

1061                 kstat_named_init(knp, (char *)efx_phy_prop_name(enp, id),
1062                     KSTAT_DATA_UINT32);
1063         }

1065         for (id = 0; id < SFXGE_NPROPS; id++) {
1066                 kstat_named_t *knp = &(sp->s_nd_stat[id + nprops]);
1067                 sfxge_ndd_param_t *snpp = &sfxge_ndd_param[id];

1069                 kstat_named_init(knp, (char *)(snpp->snp_name),
1070                     KSTAT_DATA_UINT32);
1071         }

1073         kstat_install(ksp);

1075         return (0);

1077 fail2:
1078         DTRACE_PROBE(fail2);

1080 #ifdef _USE_NDD_PROPS
1081         nd_free(ndhp);
1082         sp->s_ndh = NULL;

1084         kmem_free(sp->s_ndp, sizeof (sfxge_ndd_param_t) *
1085             (nprops + SFXGE_NPROPS));
1086         sp->s_ndp = NULL;

1088 fail1:
1089         DTRACE_PROBE1(fail1, int, rc);
1090 #endif

1092         return (rc);
1093 }


1096 void
1097 sfxge_gld_nd_unregister(sfxge_t *sp)
1098 {
1099 #ifdef _USE_NDD_PROPS
1100         caddr_t *ndhp = &(sp->s_ndh);
1101 #endif
1102         efx_nic_t *enp = sp->s_enp;
1103         unsigned int nprops;

1105         nprops = efx_nic_cfg_get(enp)->enc_phy_nprops;

1107         /* Destroy the kstat set */
1108         kstat_delete(sp->s_nd_ksp);
1109         sp->s_nd_ksp = NULL;
1110         sp->s_nd_stat = NULL;

1112         /* Unregister from the NDD framework */
1113 #ifdef _USE_NDD_PROPS
1114         nd_free(ndhp);

1116         sp->s_ndh = NULL;
```



```
1118         kmem_free(sp->s_ndp, sizeof (sfxge_ndd_param_t) *
1119             (nprops + SFXGE_NPROPS));
1120         sp->s_ndp = NULL;
1121 #endif
1122 }
1123 #endif /* ! codereview */
```

```
**********************************************************
    8665 Thu Aug 22 18:59:27 2013
new/usr/src/uts/common/io/sfxge/sfxge_gld_v2.c
Merged sfxge driver
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2008-2013 Solarflare Communications Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#include <sys/types.h>
#include <sys/ddi.h>
#include <sys/sunddi.h>
#include <sys/gld.h>
#include <sys/stream.h>
#include <sys/strsun.h>
#include <sys/strsubr.h>
#include <sys/dlpi.h>
#include <sys/pattr.h>
#include <sys/ksynch.h>
#include <sys/spl.h>

#include <inet/nd.h>
#include <inet/mi.h>

#include "sfxge.h"

#ifdef _USE_GLD_V2

void
sfxge_gld_link_update(sfxge_t *sp)
{
        sfxge_mac_t *smp = &(sp->s_mac);
        int32_t link;

        switch (smp->sm_link_mode) {
        case EFX_LINK_UNKNOWN:
                link = GLD_LINKSTATE_UNKNOWN;
                break;
        case EFX_LINK_DOWN:
                link = GLD_LINKSTATE_DOWN;
                break;
        default:
                link = GLD_LINKSTATE_UP;
        }
```



```
        gld_linkstate(sp->s_gmip, link);
}

void
sfxge_gld_mtu_update(sfxge_t *sp)
{
}

void
sfxge_gld_rx_post(sfxge_t *sp, unsigned int index, mblk_t *mp)
{
        mblk_t **mpp;

        _NOTE(ARGUNUSED(index))

        mutex_enter(&(sp->s_rx_lock));
        *(sp->s_mpp) = mp;

        mpp = &mp;
        while ((mp = *mpp) != NULL)
                mpp = &(mp->b_next);

        sp->s_mpp = mpp;
        mutex_exit(&(sp->s_rx_lock));
}

void
sfxge_gld_rx_push(sfxge_t *sp)
{
        mblk_t *mp;

        mutex_enter(&(sp->s_rx_lock));
        mp = sp->s_mp;
        sp->s_mp = NULL;
        sp->s_mpp = &(sp->s_mp);
        mutex_exit(&(sp->s_rx_lock));

        while (mp != NULL) {
                mblk_t *next;

                next = mp->b_next;
                mp->b_next = NULL;

                /* The stack behaves badly with multi-fragment messages */
                if (mp->b_cont != NULL) {
                        uint16_t flags = DB_CKSUMFLAGS(mp);

                        (void) pullupmsg(mp, -1);

                        DB_CKSUMFLAGS(mp) = flags;
                }

                gld_recv(sp->s_gmip, mp);

                mp = next;
        }
}

static int
sfxge_gld_get_stats(gld_mac_info_t *gmip, struct gld_stats *gsp)
{
        sfxge_t *sp = (sfxge_t *)(gmip->gldm_private);
        unsigned int speed;
        sfxge_link_duplex_t duplex;
        uint64_t val;
```

```
129         sfxge_mac_stat_get(sp, EFX_MAC_TX_ERRORS, &val);
130         gsp->glds_errxmt = (uint32_t)val;

132         sfxge_mac_stat_get(sp, EFX_MAC_RX_ERRORS, &val);
133         gsp->glds_errrcv = (uint32_t)val;

135         sfxge_mac_stat_get(sp, EFX_MAC_RX_FCS_ERRORS, &val);
136         gsp->glds_crc = (uint32_t)val;

138         sfxge_mac_stat_get(sp, EFX_MAC_RX_DROP_EVENTS, &val);
139         gsp->glds_norcvbuf = (uint32_t)val;

141         sfxge_mac_link_speed_get(sp, &speed);
142         gsp->glds_speed = (uint64_t)speed * 1000000ull;

144         sfxge_mac_link_duplex_get(sp, &duplex);

146         switch (duplex) {
147         case SFXGE_LINK_DUPLEX_UNKNOWN:
148                 gsp->glds_duplex = GLD_DUPLEX_UNKNOWN;
149                 break;

151         case SFXGE_LINK_DUPLEX_HALF:
152                 gsp->glds_duplex = GLD_DUPLEX_HALF;
153                 break;

155         case SFXGE_LINK_DUPLEX_FULL:
156                 gsp->glds_duplex = GLD_DUPLEX_FULL;
157                 break;

159         default:
160                 ASSERT(B_FALSE);
161                 break;
162         }

164         return (GLD_SUCCESS);
165 }

167 static int
168 sfxge_gld_reset(gld_mac_info_t *gmip)
169 {
170         /*
171          * The driver already has hardware resets at appropriate times
172          * This is only ever called before gld_start()
173          */
174         return (GLD_SUCCESS);
175 }

177 static int
178 sfxge_gld_start(gld_mac_info_t *gmip)
179 {
180         sfxge_t *sp = (sfxge_t *)(gmip->gldm_private);
181         int rc;

183         mutex_enter(&(sp->s_rx_lock));
184         sp->s_mpp = &(sp->s_mp);
185         mutex_exit(&(sp->s_rx_lock));

187         if ((rc = sfxge_start(sp, B_FALSE)) != 0)
188                 goto fail1;

190         return (GLD_SUCCESS);

192 fail1:
193         DTRACE_PROBE1(fail1, int, rc);
```



```
195         mutex_enter(&(sp->s_rx_lock));
196         ASSERT3P(sp->s_mp, ==, NULL);
197         sp->s_mpp = NULL;
198         mutex_exit(&(sp->s_rx_lock));

200         return (GLD_FAILURE);
201 }

203 static int
204 sfxge_gld_stop(gld_mac_info_t *gmip)
205 {
206         sfxge_t *sp = (sfxge_t *)(gmip->gldm_private);

208         sfxge_stop(sp);

210         mutex_enter(&(sp->s_rx_lock));
211         ASSERT3P(sp->s_mp, ==, NULL);
212         sp->s_mpp = NULL;
213         mutex_exit(&(sp->s_rx_lock));

215         return (GLD_SUCCESS);
216 }

218 static int
219 sfxge_gld_set_promiscuous(gld_mac_info_t *gmip, int flags)
220 {
221         sfxge_t *sp = (sfxge_t *)(gmip->gldm_private);
222         int rc;

224         switch (flags) {
225         case GLD_MAC_PROMISC_NONE:
226                 if ((rc = sfxge_mac_promisc_set(sp, SFXGE_PROMISC_OFF)) != 0)
227                         goto fail1;
228                 break;
229         case GLD_MAC_PROMISC_MULTI:
230                 if ((rc = sfxge_mac_promisc_set(sp, SFXGE_PROMISC_ALL_MULTI))
231                     != 0)
232                         goto fail2;
233                 break;
234         default:
235                 if ((rc = sfxge_mac_promisc_set(sp, SFXGE_PROMISC_ALL_PHYS))
236                     != 0)
237                         goto fail3;
238                 break;
239         }

241         return (GLD_SUCCESS);

243 fail3:
244         DTRACE_PROBE(fail3);
245 fail2:
246         DTRACE_PROBE(fail2);
247 fail1:
248         DTRACE_PROBE1(fail1, int, rc);
249         return (GLD_FAILURE);
250 }

252 static int
253 sfxge_gld_set_multicast(gld_mac_info_t *gmip, unsigned char *addr, int flag)
254 {
255         sfxge_t *sp = (sfxge_t *)(gmip->gldm_private);
256         int rc;

258         switch (flag) {
259         case GLD_MULTI_ENABLE:
```

```
260                 if ((rc = sfxge_mac_multicst_add(sp, addr)) != 0)
261                         goto fail1;
262                 break;
263         case GLD_MULTI_DISABLE:
264                 if ((rc = sfxge_mac_multicst_remove(sp, addr)) != 0)
265                         goto fail2;
266                 break;
267         default:
268                 ASSERT(B_FALSE);
269                 break;
270         }

272         return (GLD_SUCCESS);

274 fail2:
275         DTRACE_PROBE(fail2);
276 fail1:
277         DTRACE_PROBE1(fail1, int, rc);
278         return (GLD_FAILURE);
279 }

281 static int
282 sfxge_gld_set_mac_addr(gld_mac_info_t *gmip, unsigned char *addr)
283 {
284         sfxge_t *sp = (sfxge_t *)(gmip->gldm_private);
285         int rc;

287         if ((rc = sfxge_mac_unicst_set(sp, addr)) != 0)
288                 goto fail1;

290         return (GLD_SUCCESS);

292 fail1:
293         DTRACE_PROBE1(fail1, int, rc);

295         return (GLD_BADARG);
296 }

298 int
299 sfxge_gld_send(gld_mac_info_t *gmip, mblk_t *mp)
300 {
301         sfxge_t *sp = (sfxge_t *)(gmip->gldm_private);

303         (void) sfxge_tx_packet_add(sp, mp);

305         /*
306          * This gives no TX backpressure, which can cause unbounded TX DPL
307          * size. See bug 18984. This feature is implemented for GLDv3
308          */
309         return (GLD_SUCCESS);
310 }

312 int
313 sfxge_gld_ioctl(gld_mac_info_t *gmip, queue_t *wq, mblk_t *mp)
314 {
315         sfxge_t *sp = (sfxge_t *)(gmip->gldm_private);
316         struct iocblk *iocp;

318         iocp = (struct iocblk *)mp->b_rptr;

320         switch (iocp->ioc_cmd) {
321         case ND_GET:
322         case ND_SET:
323                 if (!nd_getset(wq, sp->s_ndh, mp))
324                         miocnak(wq, mp, 0, EINVAL);
325                 else
```



```
326                         qreply(wq, mp);
327                 break;

329         default:
330                 sfxge_ioctl(sp, wq, mp);
331                 break;
332         }

334         return (GLD_SUCCESS);
335 }


338 int
339 sfxge_gld_register(sfxge_t *sp)
340 {
341         gld_mac_info_t *gmip;
342         unsigned int pri;
343         int rc;

345         mutex_init(&(sp->s_rx_lock), NULL, MUTEX_DRIVER,
346             DDI_INTR_PRI(sp->s_intr.si_intr_pri));

348         if ((rc = sfxge_gld_nd_register(sp)) != 0)
349                 goto fail1;

351         gmip = gld_mac_alloc(sp->s_dip);

353         gmip->gldm_private = (caddr_t)sp;

355         gmip->gldm_reset = sfxge_gld_reset;
356         gmip->gldm_start = sfxge_gld_start;
357         gmip->gldm_stop = sfxge_gld_stop;
358         gmip->gldm_set_mac_addr = sfxge_gld_set_mac_addr;
359         gmip->gldm_set_multicast = sfxge_gld_set_multicast;
360         gmip->gldm_set_promiscuous = sfxge_gld_set_promiscuous;
361         gmip->gldm_send = sfxge_gld_send;
362         gmip->gldm_get_stats = sfxge_gld_get_stats;
363         gmip->gldm_ioctl = sfxge_gld_ioctl;

365         gmip->gldm_ident = (char *)sfxge_ident;
366         gmip->gldm_type = DL_ETHER;
367         gmip->gldm_minpkt = 0;
368         gmip->gldm_maxpkt = sp->s_mtu;
369         gmip->gldm_addrlen = ETHERADDRL;
370         gmip->gldm_saplen = -2;
371         gmip->gldm_broadcast_addr = sfxge_brdcst;

373         gmip->gldm_vendor_addr = kmem_alloc(ETHERADDRL, KM_SLEEP);

375         if ((rc = sfxge_mac_unicst_get(sp, SFXGE_UNICST_BIA,
376             gmip->gldm_vendor_addr)) != 0)
377                 goto fail2;

379         gmip->gldm_devinfo = sp->s_dip;
380         gmip->gldm_ppa = ddi_get_instance(sp->s_dip);

382         gmip->gldm_cookie = spltoipl(0);

384         gmip->gldm_capabilities =
385             GLD_CAP_LINKSTATE |
386             GLD_CAP_CKSUM_IPHDR |
387             GLD_CAP_CKSUM_FULL_V4 |
388             GLD_CAP_ZEROCOPY;

390         if ((rc = gld_register(sp->s_dip, (char *)ddi_driver_name(sp->s_dip),
391             gmip)) != 0)
```

```
392                 goto fail3;
394         sp->s_gmip = gmip;
395         return (0);
397 fail3:
398         DTRACE_PROBE(fail3);
399 fail2:
400         DTRACE_PROBE(fail2);
402         kmem_free(gmip->gldm_vendor_addr, ETHERADDRL);
403         gld_mac_free(gmip);
405         sfxge_gld_nd_unregister(sp);
407 fail1:
408         DTRACE_PROBE1(fail1, int, rc);
409         mutex_destroy(&(sp->s_rx_lock));
411         return (rc);
412 }
414 int
415 sfxge_gld_unregister(sfxge_t *sp)
416 {
417         gld_mac_info_t *gmip = sp->s_gmip;
418         int rc;
420         if ((rc = gld_unregister(gmip)) != 0)
421                 goto fail1;
423         sp->s_gmip = NULL;
425         kmem_free(gmip->gldm_vendor_addr, ETHERADDRL);
426         gld_mac_free(gmip);
428         sfxge_gld_nd_unregister(sp);
430         mutex_destroy(&(sp->s_rx_lock));
432         return (0);
434 fail1:
435         DTRACE_PROBE1(fail1, int, rc);
437         return (rc);
438 }
439 #endif  /* _USE_GLD_V2 */
440 #endif /* ! codereview */
```

```
**********************************************************
   31560 Thu Aug 22 18:59:28 2013
new/usr/src/uts/common/io/sfxge/sfxge_gld_v3.c
Merged sfxge driver
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2008-2013 Solarflare Communications Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#ifdef _USE_GLD_V3

#include <sys/types.h>
#include <sys/ddi.h>
#include <sys/sunddi.h>
#include <sys/stream.h>
#include <sys/strsun.h>
#include <sys/strsubr.h>
#include <sys/dlpi.h>
#ifndef _USE_GLD_V3_SOL10
#include <sys/dld.h>
#endif
#include <sys/ksynch.h>
#include <sys/cpuvar.h>
#include <sys/cpu.h>

#include <inet/tcp.h>

#include "sfxge.h"

#ifndef MAC_VERSION_V1
/* GLDv3 interface version for sol10 (u8/u9) */
#define MAC_VERSION_V1 MAC_VERSION
#endif

/* A vlan tag is 4 bytes */
#define SFXGE_VLAN_TAGSZ 4

void
sfxge_gld_link_update(sfxge_t *sp)
{
        sfxge_mac_t *smp = &(sp->s_mac);
        link_state_t link;

        switch (smp->sm_link_mode) {
```



```
        case EFX_LINK_UNKNOWN:
                link = LINK_STATE_UNKNOWN;
                break;
        case EFX_LINK_DOWN:
                link = LINK_STATE_DOWN;
                break;
        default:
                link = LINK_STATE_UP;
        }

        mac_link_update(sp->s_mh, link);
}

void
sfxge_gld_mtu_update(sfxge_t *sp)
{
#ifdef _USE_MTU_UPDATE
        (void) mac_maxsdu_update(sp->s_mh, sp->s_mtu);
#endif
}

void
sfxge_gld_rx_post(sfxge_t *sp, unsigned int index, mblk_t *mp)
{
        _NOTE(ARGUNUSED(index))

        mac_rx(sp->s_mh, NULL, mp);
}


void
sfxge_gld_rx_push(sfxge_t *sp)
{
        _NOTE(ARGUNUSED(sp))
}


static uint64_t
sfxge_phy_dfl_cap_test64(sfxge_t *sp, uint32_t field)
{
        return (sfxge_phy_cap_test(sp, EFX_PHY_CAP_DEFAULT, field, NULL) ?
            1ull : 0ull);
}


static uint64_t
sfxge_phy_cur_cap_test64(sfxge_t *sp, uint32_t field)
{
        return (sfxge_phy_cap_test(sp, EFX_PHY_CAP_CURRENT, field, NULL) ?
            1ull : 0ull);
}

static uint64_t
sfxge_phy_lp_cap_test64(sfxge_t *sp, uint32_t field)
{
        return (sfxge_phy_lp_cap_test(sp, field) ? 1ull : 0ull);
}

static int
sfxge_gld_getstat(void *arg, unsigned int id, uint64_t *valp)
{
        sfxge_t *sp = arg;
        efx_nic_t *enp = sp->s_enp;
        int rc;

        if (sp->s_mac.sm_state != SFXGE_MAC_STARTED) {
```

```
128                 rc = ENODEV;
129                 goto fail1;
130         }

132         switch (id) {
133         case MAC_STAT_IFSPEED: {
134                 unsigned int speed;

136                 sfxge_mac_link_speed_get(sp, &speed);

138                 *valp = (uint64_t)speed * 1000000ull;
139                 break;
140         }
141         case ETHER_STAT_LINK_DUPLEX: {
142                 sfxge_link_duplex_t duplex;

144                 sfxge_mac_link_duplex_get(sp, &duplex);

146                 switch (duplex) {
147                 case SFXGE_LINK_DUPLEX_UNKNOWN:
148                         *valp = LINK_DUPLEX_UNKNOWN;
149                         break;

151                 case SFXGE_LINK_DUPLEX_HALF:
152                         *valp = LINK_DUPLEX_HALF;
153                         break;

155                 case SFXGE_LINK_DUPLEX_FULL:
156                         *valp = LINK_DUPLEX_FULL;
157                         break;

159                 default:
160                         ASSERT(B_FALSE);
161                         break;
162                 }
163                 break;
164         }

166 #ifdef ETHER_STAT_CAP_10GFDX
167         case ETHER_STAT_CAP_10GFDX:
168                 *valp = sfxge_phy_dfl_cap_test64(sp, EFX_PHY_CAP_10000FDX);
169                 break;
170 #endif
171         case ETHER_STAT_CAP_1000FDX:
172                 *valp = sfxge_phy_dfl_cap_test64(sp, EFX_PHY_CAP_1000FDX);
173                 break;
174         case ETHER_STAT_CAP_1000HDX:
175                 *valp = sfxge_phy_dfl_cap_test64(sp, EFX_PHY_CAP_1000HDX);
176                 break;
177         case ETHER_STAT_CAP_100FDX:
178                 *valp = sfxge_phy_dfl_cap_test64(sp, EFX_PHY_CAP_100FDX);
179                 break;
180         case ETHER_STAT_CAP_100HDX:
181                 *valp = sfxge_phy_dfl_cap_test64(sp, EFX_PHY_CAP_100HDX);
182                 break;
183         case ETHER_STAT_CAP_10FDX:
184                 *valp = sfxge_phy_dfl_cap_test64(sp, EFX_PHY_CAP_10FDX);
185                 break;
186         case ETHER_STAT_CAP_10HDX:
187                 *valp = sfxge_phy_dfl_cap_test64(sp, EFX_PHY_CAP_10HDX);
188                 break;
189         case ETHER_STAT_CAP_ASMPAUSE:
190                 *valp = sfxge_phy_dfl_cap_test64(sp, EFX_PHY_CAP_ASYM);
191                 break;
192         case ETHER_STAT_CAP_PAUSE:
193                 *valp = sfxge_phy_dfl_cap_test64(sp, EFX_PHY_CAP_PAUSE);
```



```
194                 break;
195         case ETHER_STAT_CAP_AUTONEG:
196                 *valp = sfxge_phy_dfl_cap_test64(sp, EFX_PHY_CAP_AN);
197                 break;

199 #ifdef ETHER_STAT_ADV_CAP_10GFDX
200         case ETHER_STAT_ADV_CAP_10GFDX:
201                 *valp = sfxge_phy_cur_cap_test64(sp, EFX_PHY_CAP_10000FDX);
202                 break;
203 #endif
204         case ETHER_STAT_ADV_CAP_1000FDX:
205                 *valp = sfxge_phy_cur_cap_test64(sp, EFX_PHY_CAP_1000FDX);
206                 break;
207         case ETHER_STAT_ADV_CAP_1000HDX:
208                 *valp = sfxge_phy_cur_cap_test64(sp, EFX_PHY_CAP_1000HDX);
209                 break;
210         case ETHER_STAT_ADV_CAP_100FDX:
211                 *valp = sfxge_phy_cur_cap_test64(sp, EFX_PHY_CAP_100FDX);
212                 break;
213         case ETHER_STAT_ADV_CAP_100HDX:
214                 *valp = sfxge_phy_cur_cap_test64(sp, EFX_PHY_CAP_100HDX);
215                 break;
216         case ETHER_STAT_ADV_CAP_10FDX:
217                 *valp = sfxge_phy_cur_cap_test64(sp, EFX_PHY_CAP_10FDX);
218                 break;
219         case ETHER_STAT_ADV_CAP_10HDX:
220                 *valp = sfxge_phy_cur_cap_test64(sp, EFX_PHY_CAP_10HDX);
221                 break;
222         case ETHER_STAT_ADV_CAP_ASMPAUSE:
223                 *valp = sfxge_phy_cur_cap_test64(sp, EFX_PHY_CAP_ASYM);
224                 break;
225         case ETHER_STAT_ADV_CAP_PAUSE:
226                 *valp = sfxge_phy_cur_cap_test64(sp, EFX_PHY_CAP_PAUSE);
227                 break;
228         case ETHER_STAT_ADV_CAP_AUTONEG:
229                 *valp = sfxge_phy_cur_cap_test64(sp, EFX_PHY_CAP_AN);
230                 break;

232 #ifdef ETHER_STAT_LP_CAP_10GFDX
233         case ETHER_STAT_LP_CAP_10GFDX:
234                 *valp = sfxge_phy_lp_cap_test64(sp, EFX_PHY_CAP_10000FDX);
235                 break;
236 #endif
237         case ETHER_STAT_LP_CAP_1000FDX:
238                 *valp = sfxge_phy_lp_cap_test64(sp, EFX_PHY_CAP_1000FDX);
239                 break;
240         case ETHER_STAT_LP_CAP_1000HDX:
241                 *valp = sfxge_phy_lp_cap_test64(sp, EFX_PHY_CAP_1000HDX);
242                 break;
243         case ETHER_STAT_LP_CAP_100FDX:
244                 *valp = sfxge_phy_lp_cap_test64(sp, EFX_PHY_CAP_100FDX);
245                 break;
246         case ETHER_STAT_LP_CAP_100HDX:
247                 *valp = sfxge_phy_lp_cap_test64(sp, EFX_PHY_CAP_100HDX);
248                 break;
249         case ETHER_STAT_LP_CAP_10FDX:
250                 *valp = sfxge_phy_lp_cap_test64(sp, EFX_PHY_CAP_10FDX);
251                 break;
252         case ETHER_STAT_LP_CAP_10HDX:
253                 *valp = sfxge_phy_lp_cap_test64(sp, EFX_PHY_CAP_10HDX);
254                 break;
255         case ETHER_STAT_LP_CAP_ASMPAUSE:
256                 *valp = sfxge_phy_lp_cap_test64(sp, EFX_PHY_CAP_ASYM);
257                 break;
258         case ETHER_STAT_LP_CAP_PAUSE:
259                 *valp = sfxge_phy_lp_cap_test64(sp, EFX_PHY_CAP_PAUSE);
```

```
260                 break;
261         case ETHER_STAT_LP_CAP_AUTONEG:
262                 *valp = sfxge_phy_lp_cap_test64(sp, EFX_PHY_CAP_AN);
263                 break;

265         case ETHER_STAT_XCVR_ADDR: {
266                 const efx_nic_cfg_t *encp = efx_nic_cfg_get(enp);
267                 *valp = encp->enc_port;
268                 break;
269         }
270         case ETHER_STAT_XCVR_ID: {
271                 uint32_t oui;

273                 if ((rc = efx_phy_oui_get(sp->s_enp, &oui)) != 0)
274                         goto fail2;
275                 *valp = oui;
276                 break;
277         }
278         case MAC_STAT_MULTIRCV:
279                 sfxge_mac_stat_get(sp, EFX_MAC_RX_MULTICST_PKTS, valp);
280                 break;

282         case MAC_STAT_BRDCSTRCV:
283                 sfxge_mac_stat_get(sp, EFX_MAC_RX_BRDCST_PKTS, valp);
284                 break;

286         case MAC_STAT_MULTIXMT:
287                 sfxge_mac_stat_get(sp, EFX_MAC_TX_MULTICST_PKTS, valp);
288                 break;

290         case MAC_STAT_BRDCSTXMT:
291                 sfxge_mac_stat_get(sp, EFX_MAC_TX_BRDCST_PKTS, valp);
292                 break;

294         case MAC_STAT_IERRORS:
295                 sfxge_mac_stat_get(sp, EFX_MAC_RX_ERRORS, valp);
296                 break;

298         case MAC_STAT_OERRORS:
299                 sfxge_mac_stat_get(sp, EFX_MAC_TX_ERRORS, valp);
300                 break;

302         case MAC_STAT_RBYTES:
303                 sfxge_mac_stat_get(sp, EFX_MAC_RX_OCTETS, valp);
304                 break;

306         case MAC_STAT_IPACKETS:
307                 sfxge_mac_stat_get(sp, EFX_MAC_RX_PKTS, valp);
308                 break;

310         case MAC_STAT_OBYTES:
311                 sfxge_mac_stat_get(sp, EFX_MAC_TX_OCTETS, valp);
312                 break;

314         case MAC_STAT_OPACKETS:
315                 sfxge_mac_stat_get(sp, EFX_MAC_TX_PKTS, valp);
316                 break;

318         case MAC_STAT_NORCVBUF:
319                 sfxge_mac_stat_get(sp, EFX_MAC_RX_DROP_EVENTS, valp);
320                 break;

322         case ETHER_STAT_FCS_ERRORS:
323                 sfxge_mac_stat_get(sp, EFX_MAC_RX_FCS_ERRORS, valp);
324                 break;
```



```
326         default:
327                 rc = ENOTSUP;
328                 goto fail3;
329         }

331         return (0);
332 fail3:
333         DTRACE_PROBE(fail3);
334 fail2:
335         DTRACE_PROBE(fail2);
336 fail1:
337         DTRACE_PROBE1(fail1, int, rc);

339         return (rc);
340 }

342 static int
343 sfxge_gld_start(void *arg)
344 {
345         sfxge_t *sp = arg;
346         int rc;

348         if ((rc = sfxge_start(sp, B_FALSE)) != 0)
349                 goto fail1;

351         return (0);

353 fail1:
354         DTRACE_PROBE1(fail1, int, rc);

356         return (rc);
357 }

359 static void
360 sfxge_gld_stop(void *arg)
361 {
362         sfxge_t *sp = arg;

364         sfxge_stop(sp);
365 }

367 static int
368 sfxge_gld_setpromisc(void *arg, boolean_t on)
369 {
370         sfxge_t *sp = arg;

372         return sfxge_mac_promisc_set(sp,
373             (on) ? SFXGE_PROMISC_ALL_PHYS : SFXGE_PROMISC_OFF);
374 }

376 static int
377 sfxge_gld_multicst(void *arg, boolean_t add, const uint8_t *addr)
378 {
379         sfxge_t *sp = arg;
380         int rc;

382         if (add) {
383                 if ((rc = sfxge_mac_multicst_add(sp, (uint8_t *)addr)) != 0)
384                         goto fail1;
385         } else {
386                 if ((rc = sfxge_mac_multicst_remove(sp, (uint8_t *)addr)) != 0)
387                         goto fail2;
388         }

390         return (0);
```

```
fail2:
        DTRACE_PROBE(fail2);
fail1:
        DTRACE_PROBE1(fail1, int, rc);
        return (rc);
}

static int
sfxge_gld_unicst(void *arg, const uint8_t *addr)
{
        sfxge_t *sp = arg;
        int rc;

        if ((rc = sfxge_mac_unicst_set(sp, (uint8_t *)addr)) != 0)
                goto fail1;

        return (0);

fail1:
        DTRACE_PROBE1(fail1, int, rc);

        return (rc);
}

static void
sfxge_gld_ioctl(void *arg, queue_t *wq, mblk_t *mp)
{
        sfxge_t *sp = arg;
        struct iocblk *iocp = (struct iocblk *)mp->b_rptr;

        switch (iocp->ioc_cmd) {
#ifdef _USE_NDD_PROPS
        case ND_GET:
        case ND_SET:
                if (!nd_getset(wq, sp->s_ndh, mp))
                        miocnak(wq, mp, 0, EINVAL);
                else
                        qreply(wq, mp);
                break;
#endif

        default:
                sfxge_ioctl(sp, wq, mp);
                break;
        }
}


static mblk_t *
sfxge_gld_tx(void *arg, mblk_t *mp)
{
        sfxge_t *sp = arg;
        mblk_t *next;

        /* Walk the packet chain */
        do {
                /* Break the packet out of the chain */
                next = mp->b_next;
                mp->b_next = NULL;

                if (next != NULL)
                        prefetch_read_many(next);

                /* Post the packet in the appropriate transmit queue */
                if (sfxge_tx_packet_add(sp, mp) == ENOSPC) {
                        mp->b_next = next;
```



```
                        return (mp);
                }

                mp = next;
        } while (mp != NULL);

        return (NULL);
}

static boolean_t        sfxge_lso = B_TRUE;

static boolean_t
sfxge_gld_getcapab(void *arg, mac_capab_t cap, void *cap_arg)
{
        int rc;

        _NOTE(ARGUNUSED(arg))

        switch (cap) {
        case MAC_CAPAB_LSO: {
                mac_capab_lso_t *lsop = cap_arg;

                /* Check whether LSO is disabled */
                if (!sfxge_lso) {
                        rc = ENOTSUP;
                        goto fail1;
                }

                DTRACE_PROBE(lso);

                lsop->lso_flags = LSO_TX_BASIC_TCP_IPV4;
                lsop->lso_basic_tcp_ipv4.lso_max = TCP_MAX_LSO_LENGTH;
                break;
        }
        case MAC_CAPAB_HCKSUM: {
                uint32_t *hcksump = cap_arg;

                DTRACE_PROBE(cksum);

                *hcksump = HCKSUM_INET_FULL_V4 | HCKSUM_IPHDRCKSUM;
                break;
        }
        default:
                rc = ENOTSUP;
                goto fail1;
        }

        return (B_TRUE);

fail1:
        DTRACE_PROBE1(fail1, int, rc);

        return (B_FALSE);
}

#ifdef _USE_MAC_PRIV_PROP

/*
 * GLDv3 driver-private property names must be preceded by an underscore - see
 * mc_getprop(9E).
 */
#define SFXGE_PRIV_PROP_NAME(s) ("_" #s)

/* Return the index of the named phy property. Return -1 if not found. */
static int
sfxge_gld_priv_prop_phy_find(sfxge_t *sp, const char *name)
```

```
{
        efx_nic_t *enp = sp->s_enp;
        sfxge_mac_priv_prop_t *mac_priv_props;
        unsigned int id;
        unsigned int nprops;

        mac_priv_props = sp->s_mac_priv_props;
        nprops = efx_nic_cfg_get(enp)->enc_phy_nprops;

        for (id = 0; id < nprops; id++) {
                if (strncmp(name, *mac_priv_props, MAXLINKPROPNAME) == 0)
                        return (id);
                mac_priv_props++;
        }
        return (-1);
}

#define SFXGE_XSTR(s) SFXGE_STR(s)
#define SFXGE_STR(s) #s

static void
sfxge_gld_priv_prop_info(sfxge_t *sp, const char *name,
mac_prop_info_handle_t handle)
{
        /*
         * Using mac_prop_info_set_default_str rather than the the corresponding
         * mac_prop_info_set_default_uint32 etc as it gives readable output in
         * "dladm show-linkprop" commands for private properties. Note this does
         * not break "dladm reset-linkprop" as might have been expected.
         */
        /* Treat all the phy properties the same */
        if (sfxge_gld_priv_prop_phy_find(sp, name) > 0) {
                mac_prop_info_set_default_str(handle, "0");
                mac_prop_info_set_perm(handle, MAC_PROP_PERM_RW);
                return;
        }

        if (strcmp(name, SFXGE_PRIV_PROP_NAME(rx_coalesce_mode)) == 0) {
                mac_prop_info_set_default_str(handle,
                    SFXGE_XSTR(SFXGE_RX_COALESCE_OFF));
                mac_prop_info_set_perm(handle, MAC_PROP_PERM_RW);
                return;
        }

        if (strcmp(name, SFXGE_PRIV_PROP_NAME(rx_scale_count)) == 0) {
                mac_prop_info_set_default_str(handle,
                    SFXGE_XSTR(SFXGE_RX_SCALE_MAX));
                mac_prop_info_set_perm(handle, MAC_PROP_PERM_RW);
                return;
        }

        if (strcmp(name, SFXGE_PRIV_PROP_NAME(intr_moderation)) == 0) {
                mac_prop_info_set_default_str(handle,
                    SFXGE_XSTR(SFXGE_DEFAULT_MODERATION));
                mac_prop_info_set_perm(handle, MAC_PROP_PERM_RW);
                return;
        }
        DTRACE_PROBE(unknown_priv_prop);
}


static int
sfxge_gld_priv_prop_get(sfxge_t *sp, const char *name,
    unsigned int size, void *valp)
{
        int id;
```



```
        long val;
        int rc;

        if ((id = sfxge_gld_priv_prop_phy_find(sp, name)) > 0) {
                if ((rc = sfxge_phy_prop_get(sp, id, 0, (uint32_t *)&val)) != 0)
                        goto fail1;
                goto done;
        }

        if (strcmp(name, SFXGE_PRIV_PROP_NAME(rx_coalesce_mode)) == 0) {
                sfxge_rx_coalesce_mode_t mode;

                sfxge_rx_coalesce_mode_get(sp, &mode);

                val = (long)mode;
                goto done;
        }

        if (strcmp(name, SFXGE_PRIV_PROP_NAME(rx_scale_count)) == 0) {
                unsigned int count;

                if (sfxge_rx_scale_count_get(sp, &count) != 0)
                        count = 0;

                val = (long)count;
                goto done;
        }

        if (strcmp(name, SFXGE_PRIV_PROP_NAME(intr_moderation)) == 0) {
                unsigned int us;

                sfxge_ev_moderation_get(sp, &us);

                val = (long)us;
                goto done;
        }

        rc = ENOTSUP;
        goto fail2;

done:
        (void) snprintf(valp, size, "%ld", val);

        return (0);

fail2:
        DTRACE_PROBE(fail2);

fail1:
        DTRACE_PROBE1(fail1, int, rc);

        return (rc);
}


static int
sfxge_gld_priv_prop_set(sfxge_t *sp, const char *name, unsigned int size,
    const void *valp)
{
        int id;
        long val;
        int rc = 0;

        _NOTE(ARGUNUSED(size))

        (void) ddi_strtol(valp, (char **)NULL, 0, &val);
```

```
657         if ((id = sfxge_gld_priv_prop_phy_find(sp, name)) > 0) {
658                 if ((rc = sfxge_phy_prop_set(sp, id, (uint32_t)val)) != 0)
659                         goto fail1;
660                 goto done;
661         }


664         if (strcmp(name, SFXGE_PRIV_PROP_NAME(rx_coalesce_mode)) == 0) {
665                 if ((rc = sfxge_rx_coalesce_mode_set(sp,
666                         (sfxge_rx_coalesce_mode_t)val)) != 0)
667                         goto fail1;

669                 goto done;
670         }

672         if (strcmp(name, SFXGE_PRIV_PROP_NAME(rx_scale_count)) == 0) {
673                 if ((rc = sfxge_rx_scale_count_set(sp, (unsigned int)val)) != 0)
674                         goto fail1;

676                 goto done;
677         }

679         if (strcmp(name, SFXGE_PRIV_PROP_NAME(intr_moderation)) == 0) {
680                 if ((rc = sfxge_ev_moderation_set(sp, (unsigned int) val) != 0))
681                         goto fail1;

683                 goto done;
684         }

686         rc = ENOTSUP;
687         goto fail1;

689 done:
690         return (0);

692 fail1:
693         DTRACE_PROBE1(fail1, int, rc);

695         return (rc);
696 }

698 /*
699  * The renaming of properties is necessary as efx_phy_prop_name needs to be
700  * called after efx_port_init().  See bug 18074 and sfxge_phy.c for notes on the
701  * locking strategy.
702  */
703 void
704 sfxge_gld_priv_prop_rename(sfxge_t *sp)
705 {
706         sfxge_mac_t *smp = &(sp->s_mac);
707         sfxge_mac_priv_prop_t *mac_priv_props = sp->s_mac_priv_props;
708         efx_nic_t *enp = sp->s_enp;
709         unsigned int nprops;
710         int id;

712         ASSERT(mutex_owned(&(smp->sm_lock)));

714         nprops = efx_nic_cfg_get(enp)->enc_phy_nprops;

716         for (id = 0; id < nprops; id++) {
717                 (void) snprintf(*mac_priv_props, MAXLINKPROPNAME - 1, "_%s",
718                                 efx_phy_prop_name(enp, id));
719                 mac_priv_props++;
720         }
721 }
```



```


724 static void
725 sfxge_gld_priv_prop_init(sfxge_t *sp)
726 {
727         efx_nic_t *enp = sp->s_enp;
728         sfxge_mac_priv_prop_t *mac_priv_props;
729         unsigned int nprops;
730         unsigned int id;
731         int nnamed_props = 3;

733         nprops = efx_nic_cfg_get(enp)->enc_phy_nprops;

735         /*
736          * We have nprops phy properties, nnamed_props (3) named properties and
737          * the structure must be finished by a NULL pointer.
738          */
739         sp->s_mac_priv_props_alloc = nprops + nnamed_props + 1;
740         sp->s_mac_priv_props = kmem_zalloc(sizeof (sfxge_mac_priv_prop_t) *
741             sp->s_mac_priv_props_alloc,
742             KM_SLEEP);

744         /*
745          * Driver-private property names start with an underscore - see
746          * mc_getprop(9E).  Phy property names are only available later - see
747          * bug 18074. Siena does not have these phy properties.
748          */

750         mac_priv_props = sp->s_mac_priv_props;
751         for (id = 0; id < nprops; id++) {
752                 *mac_priv_props = kmem_zalloc(MAXLINKPROPNAME, KM_SLEEP);
753                 (void) snprintf(*mac_priv_props, MAXLINKPROPNAME - 1,
754                     SFXGE_PRIV_PROP_NAME(phyprop%d), id);
755                 mac_priv_props++;
756         }

758         *mac_priv_props = kmem_zalloc(MAXLINKPROPNAME, KM_SLEEP);
759         (void) snprintf(*mac_priv_props, MAXLINKPROPNAME - 1,
760             SFXGE_PRIV_PROP_NAME(rx_coalesce_mode));
761         mac_priv_props++;
762         nprops++;

764         *mac_priv_props = kmem_zalloc(MAXLINKPROPNAME, KM_SLEEP);
765         (void) snprintf(*mac_priv_props, MAXLINKPROPNAME - 1,
766             SFXGE_PRIV_PROP_NAME(rx_scale_count));
767         mac_priv_props++;
768         nprops++;

770         *mac_priv_props = kmem_zalloc(MAXLINKPROPNAME, KM_SLEEP);
771         (void) snprintf(*mac_priv_props, MAXLINKPROPNAME - 1,
772             SFXGE_PRIV_PROP_NAME(intr_moderation));
773         mac_priv_props++;
774         nprops++;

776         ASSERT3U((nprops + 1), ==, sp->s_mac_priv_props_alloc);

778         /* Terminated by a NULL pointer */
779         *mac_priv_props = NULL;
780 }


783 static void
784 sfxge_gld_priv_prop_fini(sfxge_t *sp)
785 {
786         efx_nic_t *enp = sp->s_enp;
787         unsigned int nprops;
```

```
788         char **mac_priv_props;
789         unsigned int id;

791         nprops = efx_nic_cfg_get(enp)->enc_phy_nprops;
792         mac_priv_props = sp->s_mac_priv_props;

794         for (id = 0; id < nprops + 3; id++) {
795                 kmem_free(*mac_priv_props, MAXLINKPROPNAME);
796                 mac_priv_props++;
797         }

799         kmem_free(sp->s_mac_priv_props, sizeof (sfxge_mac_priv_prop_t) *
800             sp->s_mac_priv_props_alloc);
801         sp->s_mac_priv_props = NULL;
802 }
803 #endif /* _USE_MAC_PRIV_PROP */


806 #ifdef _USE_GLD_V3_PROPS
807 static int
808 sfxge_gld_getprop(void *arg, const char *name, mac_prop_id_t id,
809     unsigned int size, void *valp)
810 {
811         sfxge_t *sp = arg;
812         uint32_t flag = EFX_PHY_CAP_CURRENT;
813         uint8_t *v8 = ((uint8_t *)valp);
814         int rc;

816         /* check size */
817         switch (id) {
818         case MAC_PROP_DUPLEX:
819                 if (size < sizeof (link_duplex_t)) {
820                         rc = EINVAL;
821                         goto fail1;
822                 }
823                 break;
824         case MAC_PROP_FLOWCTRL:
825                 if (size < sizeof (link_flowctrl_t)) {
826                         rc = EINVAL;
827                         goto fail1;
828                 }
829                 break;
830         case MAC_PROP_SPEED:
831         case MAC_PROP_STATUS:
832                 if (size < sizeof (uint64_t)) {
833                         rc = EINVAL;
834                         goto fail1;
835                 }
836                 break;
837         case MAC_PROP_MTU:
838                 if (size < sizeof (uint32_t)) {
839                         rc = EINVAL;
840                         goto fail1;
841                 }
842                 break;
843         case MAC_PROP_EN_AUTONEG:
844         case MAC_PROP_AUTONEG:
845         case MAC_PROP_EN_10GFDX_CAP:
846         case MAC_PROP_ADV_10GFDX_CAP:
847         case MAC_PROP_EN_1000FDX_CAP:
848         case MAC_PROP_ADV_1000FDX_CAP:
849         case MAC_PROP_EN_1000HDX_CAP:
850         case MAC_PROP_ADV_1000HDX_CAP:
851         case MAC_PROP_EN_100FDX_CAP:
852         case MAC_PROP_ADV_100FDX_CAP:
853         case MAC_PROP_EN_100HDX_CAP:
```



```
854         case MAC_PROP_ADV_100HDX_CAP:
855         case MAC_PROP_EN_10FDX_CAP:
856         case MAC_PROP_ADV_10FDX_CAP:
857         case MAC_PROP_EN_10HDX_CAP:
858         case MAC_PROP_ADV_10HDX_CAP:
859                 if (size < sizeof (uint8_t)) {
860                         rc = EINVAL;
861                         goto fail1;
862                 }
863                 break;
864 #ifdef _USE_MAC_PRIV_PROP
865         case MAC_PROP_PRIVATE:
866                 /* sfxge_gld_priv_prop_get should do any size checking */
867                 break;
868 #endif
869         default:
870                 rc = ENOTSUP;
871                 goto fail1;
872                 break;
873         }

875         switch (id) {
876         case MAC_PROP_DUPLEX: {
877                 sfxge_link_duplex_t duplex;

879                 sfxge_mac_link_duplex_get(sp, &duplex);

881                 switch (duplex) {
882                 case SFXGE_LINK_DUPLEX_UNKNOWN:
883                         *((link_duplex_t *)valp) = LINK_DUPLEX_UNKNOWN;
884                         break;

886                 case SFXGE_LINK_DUPLEX_HALF:
887                         *((link_duplex_t *)valp) = LINK_DUPLEX_HALF;
888                         break;

890                 case SFXGE_LINK_DUPLEX_FULL:
891                         *((link_duplex_t *)valp) = LINK_DUPLEX_FULL;
892                         break;

894                 default:
895                         ASSERT(B_FALSE);
896                         break;
897                 }

899                 break;
900         }
901         case MAC_PROP_SPEED: {
902                 unsigned int speed;

904                 sfxge_mac_link_speed_get(sp, &speed);

906                 *((uint64_t *)valp) = (uint64_t)speed * 1000000ull;

908                 break;
909         }
910         case MAC_PROP_STATUS: {
911                 boolean_t up;

913                 sfxge_mac_link_check(sp, &up);

915                 *((link_state_t *)valp) = (up) ?
916                     LINK_STATE_UP : LINK_STATE_DOWN;

918                 break;
919         }
```

```
920         case MAC_PROP_EN_AUTONEG:
921         case MAC_PROP_AUTONEG:
922                 *v8 = sfxge_phy_cap_test(sp, flag, EFX_PHY_CAP_AN, NULL);
923                 break;
924         case MAC_PROP_EN_10GFDX_CAP:
925         case MAC_PROP_ADV_10GFDX_CAP:
926                 *v8 = sfxge_phy_cap_test(sp, flag, EFX_PHY_CAP_10000FDX, NULL);
927                 break;
928         case MAC_PROP_EN_1000FDX_CAP:
929         case MAC_PROP_ADV_1000FDX_CAP:
930                 *v8 = sfxge_phy_cap_test(sp, flag, EFX_PHY_CAP_1000FDX, NULL);
931                 break;
932         case MAC_PROP_EN_1000HDX_CAP:
933         case MAC_PROP_ADV_1000HDX_CAP:
934                 *v8 = sfxge_phy_cap_test(sp, flag, EFX_PHY_CAP_1000HDX, NULL);
935                 break;
936         case MAC_PROP_EN_100FDX_CAP:
937         case MAC_PROP_ADV_100FDX_CAP:
938                 *v8 = sfxge_phy_cap_test(sp, flag, EFX_PHY_CAP_100FDX, NULL);
939                 break;
940         case MAC_PROP_EN_100HDX_CAP:
941         case MAC_PROP_ADV_100HDX_CAP:
942                 *v8 = sfxge_phy_cap_test(sp, flag, EFX_PHY_CAP_100HDX, NULL);
943                 break;
944         case MAC_PROP_EN_10FDX_CAP:
945         case MAC_PROP_ADV_10FDX_CAP:
946                 *v8 = sfxge_phy_cap_test(sp, flag, EFX_PHY_CAP_10FDX, NULL);
947                 break;
948         case MAC_PROP_EN_10HDX_CAP:
949         case MAC_PROP_ADV_10HDX_CAP:
950                 *v8 = sfxge_phy_cap_test(sp, flag, EFX_PHY_CAP_10HDX, NULL);
951                 break;
952         case MAC_PROP_MTU:
953                 *((uint32_t *)valp) = (uint32_t)(sp->s_mtu);
954                 break;

956         case MAC_PROP_FLOWCTRL: {
957                 unsigned int fcntl;

959                 sfxge_mac_fcntl_get(sp, &fcntl);

961                 switch (fcntl) {
962                 case 0:
963                         *((link_flowctrl_t *)valp) = LINK_FLOWCTRL_NONE;
964                         break;

966                 case EFX_FCNTL_GENERATE:
967                         *((link_flowctrl_t *)valp) = LINK_FLOWCTRL_RX;
968                         break;

970                 case EFX_FCNTL_RESPOND:
971                         *((link_flowctrl_t *)valp) = LINK_FLOWCTRL_TX;
972                         break;

974                 case (EFX_FCNTL_GENERATE | EFX_FCNTL_RESPOND):
975                         *((link_flowctrl_t *)valp) = LINK_FLOWCTRL_BI;
976                         break;

978                 default:
979                         ASSERT(B_FALSE);
980                         break;
981                 }
982                 break;
983         }
984 #ifdef _USE_MAC_PRIV_PROP
985         case MAC_PROP_PRIVATE:
```



```
986                 if ((rc = sfxge_gld_priv_prop_get(sp, name, size, valp)) != 0)
987                         goto fail2;
988                 break;
989 #endif
990         default:
991                 rc = ENOTSUP;
992                 goto fail3;
993                 break;
994         }

996         return (0);

998 fail3:
999         DTRACE_PROBE(fail3);

1001 #ifdef _USE_MAC_PRIV_PROP
1002 fail2:
1003         DTRACE_PROBE(fail2);
1004 #endif

1006 fail1:
1007         DTRACE_PROBE1(fail1, int, rc);

1009         return (rc);
1010 }
1011 #endif


1014 #ifdef _USE_GLD_V3_PROPS
1015 static int
1016 sfxge_gld_setprop(void *arg, const char *name, mac_prop_id_t id,
1017     unsigned int size, const void *valp)
1018 {
1019         sfxge_t *sp = arg;
1020         int v8 =  *(uint8_t *)valp;
1021         int rc;

1023         /* get size checks out fo the way */
1024         switch (id) {
1025         /*
1026          * On Sol11 (no updates) dladm seems to be using MAC_PROP_AUTONEG to set
1027          * the autoneg parameter. This does not match the scheme suggested in
1028          * mac(9E) but as they both map to the same think in the driver and in
1029          * dladm it doesn't matter.
1030          */
1031         case MAC_PROP_AUTONEG:
1032         case MAC_PROP_EN_AUTONEG:
1033         case MAC_PROP_EN_10GFDX_CAP:
1034         case MAC_PROP_EN_1000FDX_CAP:
1035         case MAC_PROP_EN_1000HDX_CAP:
1036         case MAC_PROP_EN_100FDX_CAP:
1037         case MAC_PROP_EN_100HDX_CAP:
1038         case MAC_PROP_EN_10FDX_CAP:
1039         case MAC_PROP_EN_10HDX_CAP:
1040                 if (size < sizeof (uint8_t)) {
1041                         rc = EINVAL;
1042                         goto fail1;
1043                 }
1044                 break;
1045         case MAC_PROP_MTU:
1046                 if (size < sizeof (uint32_t)) {
1047                         rc = EINVAL;
1048                         goto fail1;
1049                 }
1050                 break;
1051         case MAC_PROP_FLOWCTRL:
```

```
1052                 if (size < sizeof (link_flowctrl_t)) {
1053                         rc = EINVAL;
1054                         goto fail1;
1055                 }
1056                 break;
1057 #ifdef _USE_MAC_PRIV_PROP
1058         case MAC_PROP_PRIVATE:
1059                 /* sfxge_gld_priv_prop_set should do any size checking */
1060                 break;
1061 #endif
1062         default:
1063                 rc = ENOTSUP;
1064                 goto fail1;
1065                 break;
1066         }

1068         switch (id) {
1069         /*
1070          * It is unclear which of MAC_PROP_AUTONEG and MAC_PROP_EN_AUTONEG is
1071          * used.  Try both.
1072          */
1073         case MAC_PROP_AUTONEG:
1074         case MAC_PROP_EN_AUTONEG:
1075                 if ((rc = sfxge_phy_cap_set(sp, EFX_PHY_CAP_AN, v8)) != 0)
1076                         goto fail2;
1077                 break;
1078         case MAC_PROP_EN_10GFDX_CAP: {
1079                 if ((rc = sfxge_phy_cap_set(sp, EFX_PHY_CAP_10000FDX, v8)) != 0)
1080                         goto fail2;
1081                 break;
1082         }
1083         case MAC_PROP_EN_1000FDX_CAP: {
1084                 if ((rc = sfxge_phy_cap_set(sp, EFX_PHY_CAP_1000FDX, v8)) != 0)
1085                         goto fail2;
1086                 break;
1087         }
1088         case MAC_PROP_EN_1000HDX_CAP: {
1089                 if ((rc = sfxge_phy_cap_set(sp, EFX_PHY_CAP_1000HDX, v8)) != 0)
1090                         goto fail2;
1091                 break;
1092         }
1093         case MAC_PROP_EN_100FDX_CAP: {
1094                 if ((rc = sfxge_phy_cap_set(sp, EFX_PHY_CAP_100FDX, v8)) != 0)
1095                         goto fail2;
1096                 break;
1097         }
1098         case MAC_PROP_EN_100HDX_CAP: {
1099                 if ((rc = sfxge_phy_cap_set(sp, EFX_PHY_CAP_100HDX, v8)) != 0)
1100                         goto fail2;
1101                 break;
1102         }
1103         case MAC_PROP_EN_10FDX_CAP: {
1104                 if ((rc = sfxge_phy_cap_set(sp, EFX_PHY_CAP_10FDX, v8)) != 0)
1105                         goto fail2;
1106                 break;
1107         }
1108         case MAC_PROP_EN_10HDX_CAP: {
1109                 if ((rc = sfxge_phy_cap_set(sp, EFX_PHY_CAP_10HDX, v8)) != 0)
1110                         goto fail2;
1111                 break;
1112         }
1113         case MAC_PROP_MTU: {
1114                 size_t mtu = (size_t)(*((uint32_t *)valp));

1116                 if (mtu > EFX_MAC_SDU_MAX) {
1117                         rc = EINVAL;
```



```
1118                         goto fail2;
1119                 }

1121                 sp->s_mtu = mtu;

1123                 DTRACE_PROBE(restart_mtu);
1124                 (void) sfxge_restart_dispatch(sp, DDI_SLEEP, SFXGE_HW_OK,
1125                     "MTU changing", (uint32_t)mtu);

1127                 break;
1128         }
1129         case MAC_PROP_FLOWCTRL: {
1130                 unsigned int fcntl = 0;

1132                 switch (*((link_flowctrl_t *)valp)) {
1133                 case LINK_FLOWCTRL_NONE:
1134                         fcntl = 0;
1135                         break;

1137                 case LINK_FLOWCTRL_RX:
1138                         fcntl = EFX_FCNTL_GENERATE;
1139                         break;

1141                 case LINK_FLOWCTRL_TX:
1142                         fcntl = EFX_FCNTL_RESPOND;
1143                         break;

1145                 case LINK_FLOWCTRL_BI:
1146                         fcntl = EFX_FCNTL_GENERATE | EFX_FCNTL_RESPOND;
1147                         break;

1149                 default:
1150                         rc = EINVAL;
1151                         goto fail2;
1152                         break;
1153                 }

1155                 if ((rc = sfxge_mac_fcntl_set(sp, fcntl)) != 0)
1156                         goto fail3;

1158                 break;
1159         }
1160 #ifdef _USE_MAC_PRIV_PROP
1161         case MAC_PROP_PRIVATE:
1162                 if ((rc = sfxge_gld_priv_prop_set(sp, name, size, valp)) != 0)
1163                         goto fail4;

1165                 break;
1166 #endif
1167         default:
1168                 rc = ENOTSUP;
1169                 goto fail5;
1170         }

1172         return (0);

1174 fail5:
1175         DTRACE_PROBE(fail5);

1177 #ifdef _USE_MAC_PRIV_PROP
1178 fail4:
1179         DTRACE_PROBE(fail4);
1180 #endif

1182 fail3:
1183         DTRACE_PROBE(fail3);
```

```
1185 fail2:
1186         DTRACE_PROBE(fail2);

1188 fail1:
1189         DTRACE_PROBE1(fail1, int, rc);

1191         return (rc);
1192 }
1193 #endif

1195 #ifdef _USE_GLD_V3_PROPS
1196 static void
1197 sfxge_gld_propinfo(void *arg, const char *name, mac_prop_id_t id,
1198     mac_prop_info_handle_t handle)
1199 {
1200         sfxge_t *sp = arg;
1201         efx_phy_cap_type_t phy_cap = EFX_PHY_CAP_INVALID;
1202         switch (id) {
1203                 case MAC_PROP_DUPLEX:
1204                         mac_prop_info_set_perm(handle, MAC_PROP_PERM_READ);
1205                         return;
1206                 case MAC_PROP_FLOWCTRL:
1207                         mac_prop_info_set_perm(handle, MAC_PROP_PERM_RW);
1208                         mac_prop_info_set_default_link_flowctrl(handle,
1209                             LINK_FLOWCTRL_BI);
1210                         return;
1211                 case MAC_PROP_SPEED:
1212                         mac_prop_info_set_perm(handle, MAC_PROP_PERM_READ);
1213                         return;
1214                 case MAC_PROP_STATUS:
1215                         mac_prop_info_set_perm(handle, MAC_PROP_PERM_READ);
1216                         return;
1217                 case MAC_PROP_MTU: {
1218                         uint32_t mtu_default;
1219                         mac_prop_info_set_perm(handle, MAC_PROP_PERM_RW);
1220                         mtu_default = ddi_prop_get_int(DDI_DEV_T_ANY,
1221                             sp->s_dip, DDI_PROP_DONTPASS, "mtu", ETHERMTU);
1222                         mac_prop_info_set_default_uint32(handle, mtu_default);
1223                         return;
1224                         }
1225 #ifdef _USE_MAC_PRIV_PROP
1226                 case MAC_PROP_PRIVATE:
1227                         sfxge_gld_priv_prop_info(sp, name, handle);
1228                         return;
1229 #endif
1230                 case MAC_PROP_EN_AUTONEG:
1231                 case MAC_PROP_AUTONEG:
1232                         phy_cap = EFX_PHY_CAP_AN;
1233                         break;
1234                 case MAC_PROP_EN_10GFDX_CAP:
1235                 case MAC_PROP_ADV_10GFDX_CAP:
1236                         phy_cap = EFX_PHY_CAP_10000FDX;
1237                         break;
1238                 case MAC_PROP_EN_1000FDX_CAP:
1239                 case MAC_PROP_ADV_1000FDX_CAP:
1240                         phy_cap = EFX_PHY_CAP_1000FDX;
1241                         break;
1242                 case MAC_PROP_EN_1000HDX_CAP:
1243                 case MAC_PROP_ADV_1000HDX_CAP:
1244                         phy_cap = EFX_PHY_CAP_1000HDX;
1245                         break;
1246                 case MAC_PROP_EN_100FDX_CAP:
1247                 case MAC_PROP_ADV_100FDX_CAP:
1248                         phy_cap = EFX_PHY_CAP_100FDX;
1249                         break;
```



```
1250                 case MAC_PROP_EN_100HDX_CAP:
1251                 case MAC_PROP_ADV_100HDX_CAP:
1252                         phy_cap = EFX_PHY_CAP_100HDX;
1253                         break;
1254                 case MAC_PROP_EN_10FDX_CAP:
1255                 case MAC_PROP_ADV_10FDX_CAP:
1256                         phy_cap = EFX_PHY_CAP_10FDX;
1257                         break;
1258                 case MAC_PROP_EN_10HDX_CAP:
1259                 case MAC_PROP_ADV_10HDX_CAP:
1260                         phy_cap = EFX_PHY_CAP_10HDX;
1261                         break;
1262                 default:
1263                         DTRACE_PROBE(unknown_prop);
1264                         return;
1265         }
1266         if (phy_cap != EFX_PHY_CAP_INVALID) {
1267                 boolean_t rw;
1268                 uint8_t cap_default;
1269                 cap_default = sfxge_phy_cap_test(sp, EFX_PHY_CAP_DEFAULT,
1270                     phy_cap, &rw);
1271                 if (rw == B_TRUE)
1272                         mac_prop_info_set_perm(handle, MAC_PROP_PERM_RW);
1273                 else
1274                         mac_prop_info_set_perm(handle, MAC_PROP_PERM_READ);
1275                 mac_prop_info_set_default_uint8(handle, cap_default);
1276         }
1277 }
1278 #endif

1280 int
1281 sfxge_gld_register(sfxge_t *sp)
1282 {
1283         mac_callbacks_t *mcp;
1284         mac_register_t *mrp;
1285         mac_handle_t mh;
1286         int rc;

1288 #ifdef _USE_GLD_V3_SOL10
1289         if ((rc = sfxge_gld_nd_register(sp)) != 0)
1290                 goto fail0;
1291 #endif

1293         /*
1294          * NOTE: mac_register_t has additional fields in later kernels,
1295          * so the buffer returned by mac_alloc(9F) changes size. This
1296          * is not a problem for forward compatibility (a driver binary
1297          * built with older headers/libs running on a newer kernel).
1298          *
1299          * For Solaris 10, we build the sfxge driver on s10u9 to run on
1300          * s10u8, and later. This requries backward compatibility and
1301          * causes a problem. The mac_register_t in s10u8 is smaller than
1302          * the version in s10u9, so writing to the mc_margin field causes
1303          * a buffer overflow (and a hard-to-debug panic).
1304          *
1305          * Work around this problem by allocating mac_register_t using
1306          * kmem_alloc(9F) so it has the size expected by the driver. The
1307          * running kernel ignores the additional fields.
1308          *
1309          * Replace mac_alloc() with kmem_zalloc() and then set m_version.
1310          * Replace mac_free() with kmem_free().
1311          *
1312          * See bug 33189 and bug33213 for details.
1313          */
1314         if ((mrp = kmem_zalloc(sizeof (mac_register_t), KM_SLEEP)) == NULL) {
1315                 rc = ENOMEM;
```

```
1316                goto fail1;
1317        }
1318        mrp->m_version = MAC_VERSION;

1320        mrp->m_type_ident = MAC_PLUGIN_IDENT_ETHER;
1321        mrp->m_driver = sp;
1322        mrp->m_dip = sp->s_dip;

1324        /* Set up the callbacks */
1325        mcp = &(sp->s_mc);
1326        bzero(mcp, sizeof (mac_callbacks_t));

1328        mcp->mc_getstat = sfxge_gld_getstat;
1329        mcp->mc_start = sfxge_gld_start;
1330        mcp->mc_stop = sfxge_gld_stop;
1331        mcp->mc_setpromisc = sfxge_gld_setpromisc;
1332        mcp->mc_multicst = sfxge_gld_multicst;
1333        mcp->mc_unicst = sfxge_gld_unicst;
1334        mcp->mc_tx = sfxge_gld_tx;

1336        mcp->mc_callbacks |= MC_IOCTL;
1337        mcp->mc_ioctl = sfxge_gld_ioctl;

1339        mcp->mc_callbacks |= MC_GETCAPAB;
1340        mcp->mc_getcapab = sfxge_gld_getcapab;

1342 #ifdef _USE_GLD_V3_PROPS
1343        /* NOTE: mc_setprop, mc_getprop, mc_propinfo added in s10u9 */
1344        mcp->mc_callbacks |= MC_SETPROP;
1345        mcp->mc_setprop = sfxge_gld_setprop;

1347        mcp->mc_callbacks |= MC_GETPROP;
1348        mcp->mc_getprop = sfxge_gld_getprop;

1350        mcp->mc_callbacks |= MC_PROPINFO;
1351        mcp->mc_propinfo = sfxge_gld_propinfo;
1352 #endif

1354        mrp->m_callbacks = mcp;

1356        mrp->m_src_addr = kmem_alloc(ETHERADDRL, KM_SLEEP);

1358        if ((rc = sfxge_mac_unicst_get(sp, SFXGE_UNICST_BIA,
1359            mrp->m_src_addr)) != 0)
1360                goto fail2;

1362        mrp->m_min_sdu = 0;
1363        mrp->m_max_sdu = sp->s_mtu;

1365        /* NOTE: m_margin added in s10u9 */
1366        mrp->m_margin = SFXGE_VLAN_TAGSZ;

1368        /* Set up the private properties */
1369 #ifdef _USE_MAC_PRIV_PROP
1370        /* NOTE: m_priv_props added in s10u9 */
1371        mrp->m_priv_props = sp->s_mac_priv_props;
1372        sfxge_gld_priv_prop_init(sp);
1373 #endif

1375        /* NOTE: m_flags added in s11.0 */
1376        /* NOTE: m_multicast_sdu added in s11.0 */

1378        /* Register the interface */
1379        if ((rc = mac_register(mrp, &mh)) != 0)
1380                goto fail3;
```



```
1382        kmem_free(mrp->m_src_addr, ETHERADDRL);

1384        /* Free the stack registration object */
1385        kmem_free(mrp, sizeof (mac_register_t));

1387        sp->s_mh = mh;

1389        return (0);
1390 fail3:
1391        DTRACE_PROBE(fail3);
1392 fail2:
1393        DTRACE_PROBE(fail2);

1395        kmem_free(mrp->m_src_addr, ETHERADDRL);

1397        /* Free the stack registration object */
1398        kmem_free(mrp, sizeof (mac_register_t));

1400 #ifdef _USE_MAC_PRIV_PROP
1401        /* Tear down the private properties */
1402        sfxge_gld_priv_prop_fini(sp);
1403 #endif

1405        /* Clear the callbacks */
1406        bzero(&(sp->s_mc), sizeof (mac_callbacks_t));

1408 fail1:
1409        DTRACE_PROBE1(fail1, int, rc);
1410 #ifdef _USE_GLD_V3_SOL10
1411        sfxge_gld_nd_unregister(sp);

1413 fail0:
1414        DTRACE_PROBE1(fail0, int, rc);
1415 #endif

1417        return (rc);
1418 }

1420 int
1421 sfxge_gld_unregister(sfxge_t *sp)
1422 {
1423        mac_handle_t mh = sp->s_mh;
1424        int rc;

1426        if ((rc = mac_unregister(mh)) != 0)
1427                goto fail1;

1429        sp->s_mh = NULL;

1431 #ifdef _USE_MAC_PRIV_PROP
1432        /* Tear down the private properties */
1433        sfxge_gld_priv_prop_fini(sp);
1434 #endif
1435 #ifdef _USE_GLD_V3_SOL10
1436        sfxge_gld_nd_unregister(sp);
1437 #endif

1439        /* Clear the callbacks */
1440        bzero(&(sp->s_mc), sizeof (mac_callbacks_t));

1442        return (0);

1444 fail1:
1445        DTRACE_PROBE1(fail1, int, rc);

1447        return (rc);
```

```
1448 }
1449 #endif  /* _USE_GLD_V3 */
1450 #endif /* ! codereview */
```

```
**********************************************************
   19066 Thu Aug 22 18:59:28 2013
new/usr/src/uts/common/io/sfxge/sfxge_intr.c
Merged sfxge driver
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2008-2013 Solarflare Communications Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#include <sys/types.h>
#include <sys/ddi.h>
#include <sys/sunddi.h>
#include <sys/atomic.h>
#include <sys/modctl.h>
#include <sys/conf.h>
#include <sys/ethernet.h>
#include <sys/pci.h>
#include <sys/pcie.h>

#include "sfxge.h"

#include "efx.h"


/* Interrupt table DMA attributes */
static ddi_device_acc_attr_t sfxge_intr_devacc = {

        DDI_DEVICE_ATTR_V0,     /* devacc_attr_version */
        DDI_NEVERSWAP_ACC,      /* devacc_attr_endian_flags */
        DDI_STRICTORDER_ACC     /* devacc_attr_dataorder */
};

static ddi_dma_attr_t sfxge_intr_dma_attr = {
        DMA_ATTR_V0,            /* dma_attr_version     */
        0,                      /* dma_attr_addr_lo     */
        0xffffffffffffffffull,  /* dma_attr_addr_hi     */
        0xffffffffffffffffull,  /* dma_attr_count_max   */
        EFX_INTR_SIZE,          /* dma_attr_align       */
        0xffffffff,             /* dma_attr_burstsizes  */
        1,                      /* dma_attr_minxfer     */
        0xffffffffffffffffull,  /* dma_attr_maxxfer     */
        0xffffffffffffffffull,  /* dma_attr_seg         */
        1,                      /* dma_attr_sgllen      */
        1,                      /* dma_attr_granular    */
```



```
        0                       /* dma_attr_flags       */
};

static unsigned int
sfxge_intr_line(caddr_t arg1, caddr_t arg2)
{
        sfxge_t *sp = (void *)arg1;
        efx_nic_t *enp = sp->s_enp;
        sfxge_intr_t *sip = &(sp->s_intr);
        unsigned int index;
        boolean_t fatal;
        uint32_t qmask;
        int rc;

        _NOTE(ARGUNUSED(arg2))

        ASSERT3U(sip->si_type, ==, EFX_INTR_LINE);

        if (sip->si_state != SFXGE_INTR_STARTED &&
            sip->si_state != SFXGE_INTR_TESTING) {
                rc = DDI_INTR_UNCLAIMED;
                goto done;
        }

        if (sip->si_state == SFXGE_INTR_TESTING) {
                sip->si_mask |= 1;      /* only one interrupt */
                rc = DDI_INTR_CLAIMED;
                goto done;
        }

        efx_intr_status_line(enp, &fatal, &qmask);

        if (fatal) {
                sfxge_intr_fatal(sp);

                rc = DDI_INTR_CLAIMED;
                goto done;
        }

        if (qmask != 0) {
                for (index = 0; index < EFX_INTR_NEVQS; index++) {
                        if (qmask & (1 << index))
                                (void) sfxge_ev_qpoll(sp, index);
                }

                sip->si_zero_count = 0;
                sfxge_gld_rx_push(sp);
                rc = DDI_INTR_CLAIMED;
                goto done;
        }

        /*
         * bug15671/bug17203 workaround. Return CLAIMED for the first ISR=0
         * interrupt, and poll all evqs for work. For subsequent ISR=0
         * interrupts (the line must be shared in this case), just rearm the
         * event queues to ensure we don't miss an interrupt.
         */
        if (sip->si_zero_count++ == 0) {
                for (index = 0; index < EFX_INTR_NEVQS; index++) {
                        if (sp->s_sep[index] != NULL)
                                (void) sfxge_ev_qpoll(sp, index);
                }

                rc = DDI_INTR_CLAIMED;
        } else {
                for (index = 0; index < EFX_INTR_NEVQS; index++) {
```

```
128                         if (sp->s_sep[index] != NULL)
129                                 (void) sfxge_ev_qprime(sp, index);
130                 }
131
132                 rc = DDI_INTR_UNCLAIMED;
133         }
134
135 done:
136         return (rc);
137 }
138
139 static unsigned int
140 sfxge_intr_message(caddr_t arg1, caddr_t arg2)
141 {
142         sfxge_t *sp = (void *)arg1;
143         efx_nic_t *enp = sp->s_enp;
144         sfxge_intr_t *sip = &(sp->s_intr);
145         unsigned int index = (unsigned int)(uintptr_t)arg2;
146         boolean_t fatal;
147         int rc;
148
149         ASSERT3U(sip->si_type, ==, EFX_INTR_MESSAGE);
150
151         if (sip->si_state != SFXGE_INTR_STARTED &&
152             sip->si_state != SFXGE_INTR_TESTING) {
153                 rc = DDI_INTR_UNCLAIMED;
154                 goto done;
155         }
156
157         if (sip->si_state == SFXGE_INTR_TESTING) {
158                 uint64_t mask;
159
160                 do {
161                         mask = sip->si_mask;
162                 } while (atomic_cas_64(&(sip->si_mask), mask,
163                     mask | (1 << index)) != mask);
164
165                 rc = DDI_INTR_CLAIMED;
166                 goto done;
167         }
168
169         efx_intr_status_message(enp, index, &fatal);
170
171         if (fatal) {
172                 sfxge_intr_fatal(sp);
173
174                 rc = DDI_INTR_CLAIMED;
175                 goto done;
176         }
177
178         (void) sfxge_ev_qpoll(sp, index);
179
180         sfxge_gld_rx_push(sp);
181         rc = DDI_INTR_CLAIMED;
182
183 done:
184         return (rc);
185 }
186
187 static int
188 sfxge_intr_bus_enable(sfxge_t *sp)
189 {
190         sfxge_intr_t *sip = &(sp->s_intr);
191         ddi_intr_handler_t *handler;
192         int add_index;
193         int en_index;
```



```
194         int err;
195         int rc;
196
197         /* Serialise all instances to avoid problems seen in bug31184. */
198         mutex_enter(&sfxge_global_lock);
199
200         switch (sip->si_type) {
201         case EFX_INTR_MESSAGE:
202                 handler = sfxge_intr_message;
203                 break;
204
205         case EFX_INTR_LINE:
206                 handler = sfxge_intr_line;
207                 break;
208
209         default:
210                 cmn_err(CE_WARN, SFXGE_CMN_ERR
211                     "[%s%d] bus_enable: unknown intr type"
212                     " (si_type=%d nalloc=%d)\n",
213                     ddi_driver_name(sp->s_dip),
214                     ddi_get_instance(sp->s_dip),
215                     sip->si_type, sip->si_nalloc);
216                 ASSERT(B_FALSE);
217                 rc = EINVAL;
218                 goto fail1;
219         }
220
221         /* Try to add the handlers */
222         for (add_index = 0; add_index < sip->si_nalloc; add_index++) {
223                 unsigned int pri;
224
225                 (void) ddi_intr_get_pri(sip->si_table[add_index], &pri);
226                 DTRACE_PROBE2(pri, unsigned int, add_index, unsigned int, pri);
227
228                 err = ddi_intr_add_handler(sip->si_table[add_index], handler,
229                     (caddr_t)sp, (caddr_t)(uintptr_t)add_index);
230                 if (err != DDI_SUCCESS) {
231                         cmn_err(CE_WARN, SFXGE_CMN_ERR
232                             "[%s%d] bus_enable: ddi_intr_add_handler failed"
233                             " err=%d (h=%p idx=%d nalloc=%d)\n",
234                             ddi_driver_name(sp->s_dip),
235                             ddi_get_instance(sp->s_dip),
236                             err, sip->si_table[add_index], add_index,
237                             sip->si_nalloc);
238
239                         rc = (err == DDI_EINVAL) ? EINVAL : EFAULT;
240                         goto fail2;
241                 }
242         }
243
244         /* Get interrupt capabilities */
245         err = ddi_intr_get_cap(sip->si_table[0], &(sip->si_cap));
246         if (err != DDI_SUCCESS) {
247                 cmn_err(CE_WARN, SFXGE_CMN_ERR
248                     "[%s%d] bus_enable: ddi_intr_get_cap failed"
249                     " err=%d (h=%p idx=%d nalloc=%d)\n",
250                     ddi_driver_name(sp->s_dip),
251                     ddi_get_instance(sp->s_dip),
252                     err, sip->si_table[0], 0, sip->si_nalloc);
253
254                 if (err == DDI_EINVAL)
255                         rc = EINVAL;
256                 else if (err == DDI_ENOTSUP)
257                         rc = ENOTSUP;
258                 else
259                         rc = EFAULT;
```

```
261                 goto fail3;
262         }
263
264         /* Enable interrupts at the bus  */
265         if (sip->si_cap & DDI_INTR_FLAG_BLOCK) {
266                 en_index = 0; /* Silence gcc */
267                 err = ddi_intr_block_enable(sip->si_table, sip->si_nalloc);
268                 if (err != DDI_SUCCESS) {
269                         cmn_err(CE_WARN, SFXGE_CMN_ERR
270                             "[%s%d] bus_enable: ddi_intr_block_enable failed"
271                             " err=%d (table=%p nalloc=%d)\n",
272                             ddi_driver_name(sp->s_dip),
273                             ddi_get_instance(sp->s_dip),
274                             err, sip->si_table, sip->si_nalloc);
275
276                         rc = (err == DDI_EINVAL) ? EINVAL : EFAULT;
277                         goto fail4;
278                 }
279         } else {
280                 for (en_index = 0; en_index < sip->si_nalloc; en_index++) {
281                         err = ddi_intr_enable(sip->si_table[en_index]);
282                         if (err != DDI_SUCCESS) {
283                                 cmn_err(CE_WARN, SFXGE_CMN_ERR
284                                     "[%s%d] bus_enable: ddi_intr_enable failed"
285                                     " err=%d (h=%p idx=%d nalloc=%d)\n",
286                                     ddi_driver_name(sp->s_dip),
287                                     ddi_get_instance(sp->s_dip),
288                                     err, sip->si_table[en_index], en_index,
289                                     sip->si_nalloc);
290
291                                 rc = (err == DDI_EINVAL) ? EINVAL : EFAULT;
292                                 goto fail4;
293                         }
294                 }
295         }
296
297         mutex_exit(&sfxge_global_lock);
298         return (0);
299
300 fail4:
301         DTRACE_PROBE(fail4);
302
303         /* Disable the enabled handlers */
304         if (!(sip->si_cap & DDI_INTR_FLAG_BLOCK)) {
305                 while (--en_index >= 0) {
306                         err = ddi_intr_disable(sip->si_table[en_index]);
307                         if (err != DDI_SUCCESS) {
308                                 cmn_err(CE_WARN, SFXGE_CMN_ERR
309                                     "[%s%d] bus_enable: ddi_intr_disable"
310                                     " failed err=%d (h=%p idx=%d nalloc=%d)\n",
311                                     ddi_driver_name(sp->s_dip),
312                                     ddi_get_instance(sp->s_dip),
313                                     err, sip->si_table[en_index], en_index,
314                                     sip->si_nalloc);
315                         }
316                 }
317         }
318
319 fail3:
320         DTRACE_PROBE(fail3);
321
322         /* Remove all handlers */
323         add_index = sip->si_nalloc;
324
325 fail2:
```



```
326         DTRACE_PROBE(fail2);
327
328         /* Remove remaining handlers */
329         while (--add_index >= 0) {
330                 err = ddi_intr_remove_handler(sip->si_table[add_index]);
331                 if (err != DDI_SUCCESS) {
332                         cmn_err(CE_WARN, SFXGE_CMN_ERR
333                             "[%s%d] bus_enable: ddi_intr_remove_handler"
334                             " failed err=%d (h=%p idx=%d nalloc=%d)\n",
335                             ddi_driver_name(sp->s_dip),
336                             ddi_get_instance(sp->s_dip),
337                             err, sip->si_table[add_index], add_index,
338                             sip->si_nalloc);
339                 }
340         }
341
342 fail1:
343         DTRACE_PROBE1(fail1, int, rc);
344
345         mutex_exit(&sfxge_global_lock);
346         return (rc);
347 }
348
349 static void
350 sfxge_intr_bus_disable(sfxge_t *sp)
351 {
352         sfxge_intr_t *sip = &(sp->s_intr);
353         int index;
354         int err;
355
356         /* Serialise all instances to avoid problems seen in bug31184. */
357         mutex_enter(&sfxge_global_lock);
358
359         /* Disable interrupts at the bus */
360         if (sip->si_cap & DDI_INTR_FLAG_BLOCK) {
361                 err = ddi_intr_block_disable(sip->si_table, sip->si_nalloc);
362                 if (err != DDI_SUCCESS) {
363                         cmn_err(CE_WARN, SFXGE_CMN_ERR
364                             "[%s%d] bus_disable: ddi_intr_block_disable"
365                             " failed err=%d (table=%p nalloc=%d)\n",
366                             ddi_driver_name(sp->s_dip),
367                             ddi_get_instance(sp->s_dip),
368                             err, sip->si_table, sip->si_nalloc);
369                 }
370         } else {
371                 index = sip->si_nalloc;
372                 while (--index >= 0) {
373                         err = ddi_intr_disable(sip->si_table[index]);
374                         if (err != DDI_SUCCESS) {
375                                 cmn_err(CE_WARN, SFXGE_CMN_ERR
376                                     "[%s%d] bus_disable: ddi_intr_disable"
377                                     " failed err=%d (h=%p idx=%d nalloc=%d)\n",
378                                     ddi_driver_name(sp->s_dip),
379                                     ddi_get_instance(sp->s_dip),
380                                     err, sip->si_table[index], index,
381                                     sip->si_nalloc);
382                         }
383                 }
384         }
385
386         sip->si_cap = 0;
387
388         /* Remove all handlers */
389         index = sip->si_nalloc;
390         while (--index >= 0) {
391                 err = ddi_intr_remove_handler(sip->si_table[index]);
```

```
392                 if (err != DDI_SUCCESS) {
393                         cmn_err(CE_WARN, SFXGE_CMN_ERR
394                             "[%s%d] bus_disable: ddi_intr_remove_handler"
395                             " failed err=%d (h=%p idx=%d nalloc=%d)\n",
396                             ddi_driver_name(sp->s_dip),
397                             ddi_get_instance(sp->s_dip),
398                             err, sip->si_table[index], index,
399                             sip->si_nalloc);
400                 }
401         }

403         mutex_exit(&sfxge_global_lock);
404 }

406 static int
407 sfxge_intr_nic_enable(sfxge_t *sp)
408 {
409         sfxge_intr_t *sip = &(sp->s_intr);
410         efsys_mem_t *esmp = &(sip->si_mem);
411         efx_nic_t *enp = sp->s_enp;
412         unsigned int index;
413         uint64_t mask;
414         unsigned int count;
415         int rc;

417         /* Zero the memory */
418         (void) memset(esmp->esm_base, 0, EFX_INTR_SIZE);

420         /* Enable interrupts at the NIC */
421         if ((rc = efx_intr_init(enp, sip->si_type, esmp)) != 0)
422                 goto fail1;

424         efx_intr_enable(enp);

426         /* Test the interrupts */
427         mask = 0;
428         for (index = 0; index < sip->si_nalloc; index++) {
429                 mask |= (1 << index);

431                 rc = efx_intr_trigger(enp, index);
432                 ASSERT3U(rc, ==, 0);
433         }

435         /* Wait for the tests to complete */
436         count = 0;
437         do {
438                 DTRACE_PROBE1(wait, unsigned int, count);

440                 /* Spin for 1 ms */
441                 drv_usecwait(1000);

443                 /*
444                  * Check to see that all the test interrupts have been
445                  * processed.
446                  */
447                 if ((mask & sip->si_mask) == mask)
448                         goto done;

450         } while (++count < 20);

452         rc = ETIMEDOUT;
453         goto fail2;

455 done:
456         return (0);
```



```
458 fail2:
459         DTRACE_PROBE(fail2);

461         cmn_err(CE_WARN, SFXGE_CMN_ERR
462             "[%s%d] Interrupt test failed (mask=%"PRIx64" got=%"
463             PRIx64"). NIC is disabled\n",
464             ddi_driver_name(sp->s_dip), ddi_get_instance(sp->s_dip),
465             mask, sip->si_mask);

467         DTRACE_PROBE2(int_test_fail, uint64_t, mask, uint64_t, sip->si_mask);

469         sip->si_mask = 0;

471         /* Disable interrupts at the NIC */
472         efx_intr_disable(enp);
473         efx_intr_fini(enp);

475 fail1:
476         DTRACE_PROBE1(fail1, int, rc);

478         return (rc);
479 }

481 static void
482 sfxge_intr_nic_disable(sfxge_t *sp)
483 {
484         sfxge_intr_t *sip = &(sp->s_intr);
485         efx_nic_t *enp = sp->s_enp;

487         sip->si_mask = 0;

489         /* Disable interrupts at the NIC */
490         efx_intr_disable(enp);
491         efx_intr_fini(enp);
492 }

494 inline unsigned pow2_le(unsigned long n) {
495         unsigned int order = 1;
496         ASSERT3U(n, >, 0);
497         while ((1ul << order) <= n) ++order;
498         return (1ul << (order - 1));
499 }

501 int
502 sfxge_intr_init(sfxge_t *sp)
503 {
504         dev_info_t *dip = sp->s_dip;
505         sfxge_intr_t *sip = &(sp->s_intr);
506         efsys_mem_t *esmp = &(sip->si_mem);
507         sfxge_dma_buffer_attr_t dma_attr;
508         int err;
509         int rc;
510         int types;
511         int type;
512         int index;
513         unsigned int nalloc;
514         int navail;

516         SFXGE_OBJ_CHECK(sip, sfxge_intr_t);

518         ASSERT3U(sip->si_state, ==, SFXGE_INTR_UNINITIALIZED);

520 #ifdef __sparc
521         /* PSARC 2007/453 */
522         (void) ddi_prop_create(DDI_DEV_T_NONE, dip, DDI_PROP_CANSLEEP,
523             "#msix-request", NULL, 0);
```

```
#endif

        /* Get the map of supported interrupt types */
        err = ddi_intr_get_supported_types(dip, &types);
        if (err != DDI_SUCCESS) {
                cmn_err(CE_WARN, SFXGE_CMN_ERR
                    "[%s%d] intr_init: ddi_intr_get_supported_types"
                    " failed err=%d\n",
                    ddi_driver_name(sp->s_dip),
                    ddi_get_instance(sp->s_dip),
                    err);

                if (err == DDI_EINVAL)
                        rc = EINVAL;
                else if (err == DDI_INTR_NOTFOUND)
                        rc = ENOENT;
                else
                        rc = EFAULT;

                goto fail1;
        }

        /* Choose most favourable type */
        if (types & DDI_INTR_TYPE_MSIX) {
                DTRACE_PROBE(msix);

                type = DDI_INTR_TYPE_MSIX;
                sip->si_type = EFX_INTR_MESSAGE;
        } else {
                DTRACE_PROBE(fixed);

                ASSERT(types & DDI_INTR_TYPE_FIXED);

                type = DDI_INTR_TYPE_FIXED;
                sip->si_type = EFX_INTR_LINE;
        }

        /* Get the number of available interrupts */
        navail = 0;
        err = ddi_intr_get_navail(dip, type, &navail);
        if (err != DDI_SUCCESS) {
                cmn_err(CE_WARN, SFXGE_CMN_ERR
                    "[%s%d] intr_init: ddi_intr_get_navail failed err=%d\n",
                    ddi_driver_name(sp->s_dip), ddi_get_instance(sp->s_dip),
                    err);

                if (err == DDI_EINVAL)
                        rc = EINVAL;
                else if (err == DDI_INTR_NOTFOUND)
                        rc = ENOENT;
                else
                        rc = EFAULT;

                goto fail2;
        }

        /* Double-check */
        if (navail == 0) {
                rc = ENOENT;
                goto fail2;
        }

        /*
         * Allow greater number of MSI-X interrupts than CPUs.
         * This can be useful to prevent RX no desc drops; See task 32179.
         * Limit non MSI-X interrupts to a single instance.
```



```
         */
        if (type != DDI_INTR_TYPE_MSIX)
                navail = 1;
        else
                navail = min(navail, sfxge_rx_scale_prop_get(sp));

        DTRACE_PROBE1(navail, unsigned int, navail);

        /* Allocate a handle table */
        sip->si_table_size = navail * sizeof (ddi_intr_handle_t);
        sip->si_table = kmem_zalloc(sip->si_table_size, KM_SLEEP);

        /*
         * Allocate interrupt handles.
         * Serialise all device instances to avoid problems seen in bug31184.
         */
        mutex_enter(&sfxge_global_lock);

        err = ddi_intr_alloc(dip, sip->si_table, type, 0,
            navail, &(sip->si_nalloc), DDI_INTR_ALLOC_NORMAL);

        mutex_exit(&sfxge_global_lock);

        if (err != DDI_SUCCESS) {
                cmn_err(CE_WARN, SFXGE_CMN_ERR
                    "[%s%d] intr_init: ddi_intr_alloc failed err=%d"
                    " (navail=%d nalloc=%d)\n",
                    ddi_driver_name(sp->s_dip), ddi_get_instance(sp->s_dip),
                    err, navail, sip->si_nalloc);

                if (err == DDI_EINVAL)
                        rc = EINVAL;
                else if (err == DDI_EAGAIN)
                        rc = EAGAIN;
                else if (err == DDI_INTR_NOTFOUND)
                        rc = ENOENT;
                else
                        rc = EFAULT;

                goto fail3;
        }

        /* Double-check */
        if (sip->si_nalloc == 0) {
                rc = ENOENT;
                goto fail3;
        }

        /* Round down to a power of 2 */
        nalloc = pow2_le(sip->si_nalloc);

        /* Free off any excess handles */
        mutex_enter(&sfxge_global_lock);

        index = sip->si_nalloc;
        while (--index >= nalloc) {
                (void) ddi_intr_free(sip->si_table[index]);
                sip->si_table[index] = NULL;
        }

        mutex_exit(&sfxge_global_lock);

        sip->si_nalloc = nalloc;
        DTRACE_PROBE1(nalloc, unsigned int, sip->si_nalloc);

        dma_attr.sdba_dip       = sp->s_dip;
```

```
656         dma_attr.sdba_dattrp     = &sfxge_intr_dma_attr;
657         dma_attr.sdba_callback   = DDI_DMA_SLEEP;
658         dma_attr.sdba_length     = EFX_INTR_SIZE;
659         dma_attr.sdba_memflags   = DDI_DMA_CONSISTENT;
660         dma_attr.sdba_devaccp    = &sfxge_intr_devacc;
661         dma_attr.sdba_bindflags  = DDI_DMA_RDWR | DDI_DMA_CONSISTENT;
662         dma_attr.sdba_maxcookies = 1;
663         dma_attr.sdba_zeroinit   = B_TRUE;

665         if ((rc = sfxge_dma_buffer_create(esmp, &dma_attr)) != 0)
666                 goto fail4;

668         /* Store the highest priority for convenience */
669         sip->si_intr_pri = 0;
670         for (index = 0; index < sip->si_nalloc; index++) {
671                 int pri;
672                 if ((rc = ddi_intr_get_pri(sip->si_table[index], &pri)) !=  0)
673                         goto fail5;
674                 if (pri > sip->si_intr_pri)
675                         sip->si_intr_pri = pri;
676         }

678         sip->si_state = SFXGE_INTR_INITIALIZED;
679         return (0);

681 fail5:
682         DTRACE_PROBE(fail5);

684 fail4:
685         DTRACE_PROBE(fail4);

687         /* Free interrupt handles */
688         mutex_exit(&sfxge_global_lock);

690         index = sip->si_nalloc;
691         while (--index >= 0) {
692                 err = ddi_intr_free(sip->si_table[index]);
693                 if (err != DDI_SUCCESS) {
694                         cmn_err(CE_WARN, SFXGE_CMN_ERR
695                             "[%s%d] intr_init: ddi_intr_free failed err=%d"
696                             " (h=%p idx=%d nalloc=%d)\n",
697                             ddi_driver_name(sp->s_dip),
698                             ddi_get_instance(sp->s_dip),
699                             err, sip->si_table[index], index, sip->si_nalloc);
700                 }
701                 sip->si_table[index] = NULL;
702         }
703         sip->si_nalloc = 0;

705         mutex_exit(&sfxge_global_lock);

707 fail3:
708         DTRACE_PROBE(fail3);

710         /* Free the handle table */
711         kmem_free(sip->si_table, sip->si_table_size);
712         sip->si_table = NULL;
713         sip->si_table_size = 0;

715 fail2:
716         DTRACE_PROBE(fail2);

718         /* Clear the interrupt type */
719         sip->si_type = EFX_INTR_INVALID;

721 fail1:
```



```
722         DTRACE_PROBE1(fail1, int, rc);

724         SFXGE_OBJ_CHECK(sip, sfxge_intr_t);

726         return (rc);
727 }

729 int
730 sfxge_intr_start(sfxge_t *sp)
731 {
732         sfxge_intr_t *sip = &(sp->s_intr);
733         int rc;

735         ASSERT3U(sip->si_state, ==, SFXGE_INTR_INITIALIZED);

737         /* Enable interrupts at the bus */
738         if ((rc = sfxge_intr_bus_enable(sp)) != 0)
739                 goto fail1;

741         sip->si_state = SFXGE_INTR_TESTING;

743         /* Enable interrupts at the NIC */
744         if ((rc = sfxge_intr_nic_enable(sp)) != 0)
745                 goto fail2;

747         sip->si_state = SFXGE_INTR_STARTED;

749         return (0);

751 fail2:
752         DTRACE_PROBE(fail2);

754         /* Disable interrupts at the bus */
755         sfxge_intr_bus_disable(sp);

757 fail1:
758         DTRACE_PROBE1(fail1, int, rc);

760         sip->si_state = SFXGE_INTR_INITIALIZED;

762         return (rc);
763 }

765 void
766 sfxge_intr_stop(sfxge_t *sp)
767 {
768         sfxge_intr_t *sip = &(sp->s_intr);

770         ASSERT3U(sip->si_state, ==, SFXGE_INTR_STARTED);

772         sip->si_state = SFXGE_INTR_INITIALIZED;

774         /* Disable interrupts at the NIC */
775         sfxge_intr_nic_disable(sp);

777         /* Disable interrupts at the bus */
778         sfxge_intr_bus_disable(sp);
779 }

781 void
782 sfxge_intr_fini(sfxge_t *sp)
783 {
784         sfxge_intr_t *sip = &(sp->s_intr);
785         efsys_mem_t *esmp = &(sip->si_mem);
786         int index;
787         int err;
```

```
789         ASSERT3U(sip->si_state, ==, SFXGE_INTR_INITIALIZED);
791         sip->si_state = SFXGE_INTR_UNINITIALIZED;
793         /* Tear down dma setup */
794         sfxge_dma_buffer_destroy(esmp);
797         /* Free interrupt handles */
798         mutex_enter(&sfxge_global_lock);
800         index = sip->si_nalloc;
801         while (--index >= 0) {
802                 err = ddi_intr_free(sip->si_table[index]);
803                 if (err != DDI_SUCCESS) {
804                         cmn_err(CE_WARN, SFXGE_CMN_ERR
805                             "[%s%d] intr_fini: ddi_intr_free failed err=%d"
806                             " (h=%p idx=%d nalloc=%d)\n",
807                             ddi_driver_name(sp->s_dip),
808                             ddi_get_instance(sp->s_dip),
809                             err, sip->si_table[index], index, sip->si_nalloc);
810                 }
811                 sip->si_table[index] = NULL;
812         }
813         sip->si_nalloc = 0;
815         mutex_exit(&sfxge_global_lock);
817         /* Free the handle table */
818         kmem_free(sip->si_table, sip->si_table_size);
819         sip->si_table = NULL;
820         sip->si_table_size = 0;
822         /* Clear the interrupt type */
823         sip->si_type = EFX_INTR_INVALID;
825         SFXGE_OBJ_CHECK(sip, sfxge_intr_t);
826 }
827 #endif /* ! codereview */
```

```
**********************************************************
    5185 Thu Aug 22 18:59:28 2013
new/usr/src/uts/common/io/sfxge/sfxge_ioc.h
Merged sfxge driver
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2008-2013 Solarflare Communications Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#ifndef _SYS_SFXGE_IOC_H
#define _SYS_SFXGE_IOC_H

#ifdef  __cplusplus
extern "C" {
#endif

#include <sys/types.h>

/* Ensure no ambiguity over structure layouts */
#pragma pack(1)

#define SFXGE_IOC       ('S' << 24 | 'F' << 16 | 'C' << 8)

#define SFXGE_STOP_IOC  (SFXGE_IOC | 0x01)
#define SFXGE_START_IOC (SFXGE_IOC | 0x02)

/* MDIO was SFXGE_IOC 0x03 */

/* I2C was SFXGE_IOC 0x04 */

/* SPI was SFXGE_IOC 0x05 */

/* BAR */

#define SFXGE_BAR_IOC   (SFXGE_IOC | 0x06)

typedef struct sfxge_bar_ioc_s {
        uint32_t        sbi_op;
        uint32_t        sbi_addr;
        uint32_t        sbi_data[4];
} sfxge_bar_ioc_t;

#define SFXGE_BAR_OP_READ       0x00000001
#define SFXGE_BAR_OP_WRITE      0x00000002
```



```
/* PCI */

#define SFXGE_PCI_IOC   (SFXGE_IOC | 0x07)

typedef struct sfxge_pci_ioc_s {
        uint32_t        spi_op;
        uint8_t         spi_addr;
        uint8_t         spi_data;
} sfxge_pci_ioc_t;

#define SFXGE_PCI_OP_READ       0x00000001
#define SFXGE_PCI_OP_WRITE      0x00000002

/* MAC */

#define SFXGE_MAC_IOC   (SFXGE_IOC | 0x08)

typedef struct sfxge_mac_ioc_s {
        uint32_t        smi_op;
        uint32_t        smi_data;
} sfxge_mac_ioc_t;

#define SFXGE_MAC_OP_LOOPBACK   0x00000001

/* PHY */

#define SFXGE_PHY_IOC   (SFXGE_IOC | 0x09)

typedef struct sfxge_phy_ioc_s {
        uint32_t        spi_op;
        uint32_t        spi_data;
} sfxge_phy_ioc_t;

#define SFXGE_PHY_OP_LOOPBACK   0x00000001
#define SFXGE_PHY_OP_LINK       0x00000002
#define SFXGE_PHY_OP_LED        0x00000003

/* SRAM */

#define SFXGE_SRAM_IOC  (SFXGE_IOC | 0x0a)

typedef struct sfxge_sram_ioc_s {
        uint32_t        ssi_op;
        uint32_t        ssi_data;
} sfxge_sram_ioc_t;

#define SFXGE_SRAM_OP_TEST      0x00000001

/* TX */

#define SFXGE_TX_IOC    (SFXGE_IOC | 0x0b)

typedef struct sfxge_tx_ioc_s {
        uint32_t        sti_op;
        uint32_t        sti_data;
} sfxge_tx_ioc_t;

#define SFXGE_TX_OP_LOOPBACK    0x00000001

/* RX */

#define SFXGE_RX_IOC    (SFXGE_IOC | 0x0c)

typedef struct sfxge_rx_ioc_s {
        uint32_t        sri_op;
```

```
128         uint32_t        sri_data;
129 } sfxge_rx_ioc_t;

131 #define SFXGE_RX_OP_LOOPBACK    0x00000001

133 /* NVRAM */

135 #define SFXGE_NVRAM_IOC (SFXGE_IOC | 0x0d)

137 typedef struct sfxge_nvram_ioc_s {
138         uint32_t        sni_op;
139         uint32_t        sni_type;
140         uint32_t        sni_offset;
141         uint32_t        sni_size;
142         uint32_t        sni_subtype;
143         uint16_t        sni_version[4];         /* get/set_ver */
144         /*
145          * Streams STRMSGSZ limit (default 64kb)
146          * See write(2) and I_STR in streamio(7i)
147          */
148         uint8_t         sni_data[32*1024];      /* read/write */
149 } sfxge_nvram_ioc_t;

151 #define SFXGE_NVRAM_OP_SIZE             0x00000001
152 #define SFXGE_NVRAM_OP_READ             0x00000002
153 #define SFXGE_NVRAM_OP_WRITE            0x00000003
154 #define SFXGE_NVRAM_OP_ERASE            0x00000004
155 #define SFXGE_NVRAM_OP_GET_VER          0x00000005
156 #define SFXGE_NVRAM_OP_SET_VER          0x00000006

158 #define SFXGE_NVRAM_TYPE_BOOTROM        0x00000001
159 #define SFXGE_NVRAM_TYPE_BOOTROM_CFG    0x00000002
160 #define SFXGE_NVRAM_TYPE_MC             0x00000003
161 #define SFXGE_NVRAM_TYPE_MC_GOLDEN      0x00000004
162 #define SFXGE_NVRAM_TYPE_PHY            0x00000005
163 #define SFXGE_NVRAM_TYPE_NULL_PHY       0x00000006
164 #define SFXGE_NVRAM_TYPE_FPGA           0x00000007

166 /* PHY BIST */

168 #define SFXGE_PHY_BIST_IOC      (SFXGE_IOC | 0x0e)

170 typedef struct sfxge_phy_bist_ioc_s {
171         boolean_t       spbi_break_link;
172         uint8_t         spbi_status_a;
173         uint8_t         spbi_status_b;
174         uint8_t         spbi_status_c;
175         uint8_t         spbi_status_d;
176         uint16_t        spbi_length_ind_a;
177         uint16_t        spbi_length_ind_b;
178         uint16_t        spbi_length_ind_c;
179         uint16_t        spbi_length_ind_d;
180 } sfxge_phy_bist_ioc_t;

182 #define SFXGE_PHY_BIST_CABLE_OK                 0
183 #define SFXGE_PHY_BIST_CABLE_INVALID            1
184 #define SFXGE_PHY_BIST_CABLE_OPEN               2
185 #define SFXGE_PHY_BIST_CABLE_INTRAPAIRSHORT     3
186 #define SFXGE_PHY_BIST_CABLE_INTERPAIRSHORT     4
187 #define SFXGE_PHY_BIST_CABLE_BUSY               5
188 #define SFXGE_PHY_BIST_CABLE_UNKNOWN            6

190 /* MCDI */

192 #define SFXGE_MCDI_IOC  (SFXGE_IOC | 0x0f)
```



```
194 typedef struct sfxge_mcdi_ioc_s {
195         uint8_t         smi_payload[256];
196         uint8_t         smi_cmd;
197         uint8_t         smi_len; /* In and out */
198         uint8_t         smi_rc;
199 } sfxge_mcdi_ioc_t;

201 /* Reset the NIC */

203 #define SFXGE_NIC_RESET_IOC     (SFXGE_IOC | 0x10)

205 /* VPD */

207 #define SFXGE_VPD_IOC   (SFXGE_IOC | 0x11)

209 #define SFXGE_VPD_MAX_PAYLOAD 0x100

211 typedef struct sfxge_vpd_ioc_s {
212         uint8_t         svi_op;
213         uint8_t         svi_tag;
214         uint16_t        svi_keyword;
215         uint8_t         svi_len; /* In or out */
216         uint8_t         svi_payload[SFXGE_VPD_MAX_PAYLOAD]; /* In or out */
217 } sfxge_vpd_ioc_t;

219 #define SFXGE_VPD_OP_GET_KEYWORD        0x00000001
220 #define SFXGE_VPD_OP_SET_KEYWORD        0x00000002

222 #pragma pack()

224 #ifdef  __cplusplus
225 }
226 #endif

228 #endif  /* _SYS_SFXGE_IOC_H */
229 #endif /* ! codereview */
```

```
**********************************************************
   23621 Thu Aug 22 18:59:28 2013
new/usr/src/uts/common/io/sfxge/sfxge_mac.c
Merged sfxge driver
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2008-2013 Solarflare Communications Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * All efx_mac_*() must be after efx_port_init()
 * LOCKING STRATEGY: Aquire sm_lock and test sm_state==SFXGE_MAC_STARTED
 * to serialise against sfxge_restart()
 */

#include <sys/types.h>
#include <sys/sysmacros.h>
#include <sys/ddi.h>
#include <sys/sunddi.h>

#include "sfxge.h"
#include "efx.h"

#define SFXGE_MAC_POLL_PERIOD_MS 1000

static void sfxge_mac_link_update_locked(sfxge_t *sp, efx_link_mode_t mode);


/* MAC DMA attributes */
static ddi_device_acc_attr_t sfxge_mac_devacc = {

        DDI_DEVICE_ATTR_V0,     /* devacc_attr_version */
        DDI_NEVERSWAP_ACC,      /* devacc_attr_endian_flags */
        DDI_STRICTORDER_ACC     /* devacc_attr_dataorder */
};

static ddi_dma_attr_t sfxge_mac_dma_attr = {
        DMA_ATTR_V0,            /* dma_attr_version     */
        0,                      /* dma_attr_addr_lo     */
        0xffffffffffffffffull,  /* dma_attr_addr_hi     */
        0xffffffffffffffffull,  /* dma_attr_count_max   */
        0x1000,                 /* dma_attr_align       */
        0xffffffff,             /* dma_attr_burstsizes  */
        1,                      /* dma_attr_minxfer     */
```



```
        0xffffffffffffffffull,  /* dma_attr_maxxfer     */
        0xffffffffffffffffull,  /* dma_attr_seg         */
        1,                      /* dma_attr_sgllen      */
        1,                      /* dma_attr_granular    */
        0                       /* dma_attr_flags       */
};


static void
_sfxge_mac_stat_update(sfxge_mac_t *smp, int tries, int delay_usec)
{
        sfxge_t *sp = smp->sm_sp;
        efsys_mem_t *esmp = &(smp->sm_mem);
        int rc, i;

        ASSERT(mutex_owned(&(smp->sm_lock)));
        ASSERT3U(smp->sm_state, !=, SFXGE_MAC_UNINITIALIZED);

        /* if no stats pending then they are already freshly updated */
        if (smp->sm_mac_stats_timer_reqd && !smp->sm_mac_stats_pend)
                return;

        for (i = 0; i < tries; i++) {
                /* Synchronize the DMA memory for reading */
                (void) ddi_dma_sync(smp->sm_mem.esm_dma_handle,
                    0,
                    EFX_MAC_STATS_SIZE,
                    DDI_DMA_SYNC_FORKERNEL);

                /* Try to update the cached counters */
                if ((rc = efx_mac_stats_update(sp->s_enp, esmp, smp->sm_stat,
                    NULL)) != EAGAIN)
                        goto done;

                drv_usecwait(delay_usec);
        }

        DTRACE_PROBE(mac_stat_timeout);
        cmn_err(CE_NOTE, SFXGE_CMN_ERR "[%s%d] MAC stats timeout",
            ddi_driver_name(sp->s_dip), ddi_get_instance(sp->s_dip));
        return;

done:
        smp->sm_mac_stats_pend = B_FALSE;
        smp->sm_lbolt = ddi_get_lbolt();
}

static void
sfxge_mac_stat_update_quick(sfxge_mac_t *smp)
{
        /*
         * Update the statistics from the most recent DMA. This might race
         * with an inflight dma, so retry once. Otherwise get mac stat
         * values from the last mac_poll() or MC periodic stats.
         */
        _sfxge_mac_stat_update(smp, 2, 50);
}

static void
sfxge_mac_stat_update_wait(sfxge_mac_t *smp)
{
        /* Wait a max of 20 * 500us = 10ms */
        _sfxge_mac_stat_update(smp, 20, 500);
}

static int
```

```
128 sfxge_mac_kstat_update(kstat_t *ksp, int rw)
129 {
130         sfxge_mac_t *smp = ksp->ks_private;
131         kstat_named_t *knp;
132         int rc;

134         if (rw != KSTAT_READ) {
135                 rc = EACCES;
136                 goto fail1;
137         }

139         ASSERT(mutex_owned(&(smp->sm_lock)));

141         if (smp->sm_state != SFXGE_MAC_STARTED)
142                 goto done;

144         sfxge_mac_stat_update_quick(smp);

146         knp = smp->sm_stat;
147         knp += EFX_MAC_NSTATS;

149         knp->value.ui64 = (smp->sm_link_up) ? 1 : 0;
150         knp++;

152         knp->value.ui64 = smp->sm_link_speed;
153         knp++;

155         knp->value.ui64 = smp->sm_link_duplex;
156         knp++;

158 done:
159         return (0);

161 fail1:
162         DTRACE_PROBE1(fail1, int, rc);

164         return (rc);
165 }

167 static int
168 sfxge_mac_kstat_init(sfxge_t *sp)
169 {
170         sfxge_mac_t *smp = &(sp->s_mac);
171         dev_info_t *dip = sp->s_dip;
172         char name[MAXNAMELEN];
173         kstat_t *ksp;
174         kstat_named_t *knp;
175         unsigned int id;
176         int rc;

178         /* Create the set */
179         (void) snprintf(name, MAXNAMELEN - 1, "%s_mac", ddi_driver_name(dip));

181         if ((ksp = kstat_create((char *)ddi_driver_name(dip),
182             ddi_get_instance(dip), name, "mac", KSTAT_TYPE_NAMED,
183             EFX_MAC_NSTATS + 4, 0)) == NULL) {
184                 rc = ENOMEM;
185                 goto fail1;
186         }

188         smp->sm_ksp = ksp;

190         ksp->ks_update = sfxge_mac_kstat_update;
191         ksp->ks_private = smp;
192         ksp->ks_lock = &(smp->sm_lock);
```



```
194         /* Initialise the named stats */
195         smp->sm_stat = knp = ksp->ks_data;
196         for (id = 0; id < EFX_MAC_NSTATS; id++) {
197                 kstat_named_init(knp, (char *)efx_mac_stat_name(sp->s_enp, id),
198                     KSTAT_DATA_UINT64);
199                 knp++;
200         }

202         kstat_named_init(knp++, "link_up", KSTAT_DATA_UINT64);
203         kstat_named_init(knp++, "link_speed", KSTAT_DATA_UINT64);
204         kstat_named_init(knp++, "link_duplex", KSTAT_DATA_UINT64);

206         kstat_install(ksp);

208         return (0);

210 fail1:
211         DTRACE_PROBE1(fail1, int, rc);

213         return (rc);
214 }

216 static void
217 sfxge_mac_kstat_fini(sfxge_t *sp)
218 {
219         sfxge_mac_t *smp = &(sp->s_mac);

221         /* Destroy the set */
222         kstat_delete(smp->sm_ksp);
223         smp->sm_ksp = NULL;
224         smp->sm_stat = NULL;
225 }

227 void
228 sfxge_mac_stat_get(sfxge_t *sp, unsigned int id, uint64_t *valp)
229 {
230         sfxge_mac_t *smp = &(sp->s_mac);

232         /* Make sure the cached counter values are recent */
233         mutex_enter(&(smp->sm_lock));

235         if (smp->sm_state != SFXGE_MAC_STARTED)
236                 goto done;

238         sfxge_mac_stat_update_quick(smp);

240         *valp = smp->sm_stat[id].value.ui64;

242 done:
243         mutex_exit(&(smp->sm_lock));
244 }

246 static void
247 sfxge_mac_poll(void *arg)
248 {
249         sfxge_t *sp = arg;
250         efx_nic_t *enp = sp->s_enp;
251         sfxge_mac_t *smp = &(sp->s_mac);
252         efsys_mem_t *esmp = &(smp->sm_mem);
253         efx_link_mode_t mode;
254         clock_t timeout;

256         mutex_enter(&(smp->sm_lock));
257         while (smp->sm_state == SFXGE_MAC_STARTED) {

259                 /* clears smp->sm_mac_stats_pend if appropriate */
```

```
260                 if (smp->sm_mac_stats_pend)
261                         sfxge_mac_stat_update_wait(smp);
263                 /* This may sleep waiting for MCDI completion */
264                 mode = EFX_LINK_UNKNOWN;
265                 if (efx_port_poll(enp, &mode) == 0)
266                         sfxge_mac_link_update_locked(sp, mode);
268                 if ((smp->sm_link_poll_reqd == B_FALSE) &&
269                     (smp->sm_mac_stats_timer_reqd == B_FALSE))
270                         goto done;
272                 /* Zero the memory */
273                 (void) memset(esmp->esm_base, 0, EFX_MAC_STATS_SIZE);
275                 /* Trigger upload the MAC statistics counters */
276                 if (smp->sm_link_up &&
277                     efx_mac_stats_upload(sp->s_enp, esmp) == 0)
278                         smp->sm_mac_stats_pend = B_TRUE;
280                 /* Wait for timeout or end of polling */
281                 timeout = ddi_get_lbolt() + drv_usectohz(1000 *
282                     SFXGE_MAC_POLL_PERIOD_MS);
283                 while (smp->sm_state == SFXGE_MAC_STARTED) {
284                         if (cv_timedwait(&(smp->sm_link_poll_kv),
285                                 &(smp->sm_lock), timeout) < 0) {
286                                 /* Timeout - poll if polling still enabled */
287                                 break;
288                         }
289                 }
290         }
291 done:
292         mutex_exit(&(smp->sm_lock));
294 }
296 static void
297 sfxge_mac_poll_start(sfxge_t *sp)
298 {
299         sfxge_mac_t *smp = &(sp->s_mac);
301         ASSERT(mutex_owned(&(smp->sm_lock)));
302         ASSERT3U(smp->sm_state, ==, SFXGE_MAC_STARTED);
304         /* Schedule a poll */
305         (void) ddi_taskq_dispatch(smp->sm_tqp, sfxge_mac_poll, sp, DDI_SLEEP);
306 }
308 static void
309 sfxge_mac_poll_stop(sfxge_t *sp)
310 {
311         sfxge_mac_t *smp = &(sp->s_mac);
313         ASSERT(mutex_owned(&(smp->sm_lock)));
314         ASSERT3U(smp->sm_state, ==, SFXGE_MAC_INITIALIZED);
316         cv_broadcast(&(smp->sm_link_poll_kv));
318         /* Wait for link polling to cease */
319         mutex_exit(&(smp->sm_lock));
320         ddi_taskq_wait(smp->sm_tqp);
321         mutex_enter(&(smp->sm_lock));
323         /* Wait for any pending DMAed stats to complete */
324         sfxge_mac_stat_update_wait(smp);
325 }
```



```
327 int
328 sfxge_mac_init(sfxge_t *sp)
329 {
330         sfxge_mac_t *smp = &(sp->s_mac);
331         efsys_mem_t *esmp = &(smp->sm_mem);
332         dev_info_t *dip = sp->s_dip;
333         sfxge_dma_buffer_attr_t dma_attr;
334         const efx_nic_cfg_t *encp;
335         unsigned char *bytes;
336         char buf[8];    /* sufficient for "true" or "false" plus NULL */
337         char name[MAXNAMELEN];
338         int *ints;
339         unsigned int n;
340         int err, rc;
342         SFXGE_OBJ_CHECK(smp, sfxge_mac_t);
344         ASSERT3U(smp->sm_state, ==, SFXGE_MAC_UNINITIALIZED);
346         smp->sm_sp = sp;
347         encp = efx_nic_cfg_get(sp->s_enp);
348         smp->sm_link_poll_reqd = (~encp->enc_features &
349                                   EFX_FEATURE_LINK_EVENTS);
350         smp->sm_mac_stats_timer_reqd = (~encp->enc_features &
351                                         EFX_FEATURE_PERIODIC_MAC_STATS);
353         mutex_init(&(smp->sm_lock), NULL, MUTEX_DRIVER,
354             DDI_INTR_PRI(sp->s_intr.si_intr_pri));
355         cv_init(&(smp->sm_link_poll_kv), NULL, CV_DRIVER, NULL);
357         /* Create link poll taskq */
358         (void) snprintf(name, MAXNAMELEN - 1, "%s_mac_tq",
359             ddi_driver_name(dip));
360         smp->sm_tqp = ddi_taskq_create(dip, name, 1, TASKQ_DEFAULTPRI,
361             DDI_SLEEP);
362         if (smp->sm_tqp == NULL) {
363                 rc = ENOMEM;
364                 goto fail1;
365         }
367         if ((rc = sfxge_phy_init(sp)) != 0)
368                 goto fail2;
370         dma_attr.sdba_dip        = dip;
371         dma_attr.sdba_dattrp     = &sfxge_mac_dma_attr;
372         dma_attr.sdba_callback   = DDI_DMA_SLEEP;
373         dma_attr.sdba_length     = EFX_MAC_STATS_SIZE;
374         dma_attr.sdba_memflags   = DDI_DMA_CONSISTENT;
375         dma_attr.sdba_devaccp    = &sfxge_mac_devacc;
376         dma_attr.sdba_bindflags  = DDI_DMA_READ | DDI_DMA_CONSISTENT;
377         dma_attr.sdba_maxcookies = 1;
378         dma_attr.sdba_zeroinit   = B_TRUE;
380         if ((rc = sfxge_dma_buffer_create(esmp, &dma_attr)) != 0)
381                 goto fail3;
383         /*
384          * Set the initial group hash to allow reception of only broadcast
385          * packets.
386          */
387         smp->sm_bucket[0xff] = 1;
389         /* Set the initial flow control values */
390         smp->sm_fcntl = EFX_FCNTL_RESPOND | EFX_FCNTL_GENERATE;
```

```
392         /*
393          * Determine the 'burnt-in' MAC address:
394          *
395          * A: if the "mac-address" property is set on our device node use that.
396          * B: otherwise, if the system property "local-mac-address?" is set to
397          *    "false" then we use the system MAC address.
398          * C: otherwise, if the "local-mac-address" property is set on our
399          *    device node use that.
400          * D: otherwise, use the value from NVRAM.
401          */
402
403         /* A */
404         err = ddi_prop_lookup_byte_array(DDI_DEV_T_ANY, dip, DDI_PROP_DONTPASS,
405             "mac-address", &bytes, &n);
406         switch (err) {
407         case DDI_PROP_SUCCESS:
408                 if (n == ETHERADDRL) {
409                         bcopy(bytes, smp->sm_bia, ETHERADDRL);
410                         goto done;
411                 }
412
413                 ddi_prop_free(bytes);
414                 break;
415
416         default:
417                 break;
418         }
419
420         /* B */
421         n = sizeof (buf);
422         bzero(buf, n--);
423         (void) ddi_getlongprop_buf(DDI_DEV_T_ANY, dip, DDI_PROP_CANSLEEP,
424             "local-mac-address?", buf, (int *)&n);
425
426         if (strcmp(buf, "false") == 0) {
427                 struct ether_addr addr;
428
429                 if (localetheraddr(NULL, &addr) != 0) {
430                         bcopy((uint8_t *)&addr, smp->sm_bia, ETHERADDRL);
431                         goto done;
432                 }
433         }
434
435         /*
436          * C
437          *
438          * NOTE: "local-mac_address" maybe coded as an integer or byte array.
439          */
440         err = ddi_prop_lookup_int_array(DDI_DEV_T_ANY, dip, DDI_PROP_DONTPASS,
441             "local-mac-address", &ints, &n);
442         switch (err) {
443         case DDI_PROP_SUCCESS:
444                 if (n == ETHERADDRL) {
445                         while (n-- != 0)
446                                 smp->sm_bia[n] = ints[n] & 0xff;
447
448                         goto done;
449                 }
450
451                 ddi_prop_free(ints);
452                 break;
453
454         default:
455                 break;
456         }
```



```
458         err = ddi_prop_lookup_byte_array(DDI_DEV_T_ANY, dip, DDI_PROP_DONTPASS,
459             "local-mac-address", &bytes, &n);
460         switch (err) {
461         case DDI_PROP_SUCCESS:
462                 if (n == ETHERADDRL) {
463                         bcopy(bytes, smp->sm_bia, ETHERADDRL);
464                         goto done;
465                 }
466
467                 ddi_prop_free(bytes);
468                 break;
469
470         default:
471                 break;
472         }
473
474         /* D */
475         bcopy(encp->enc_mac_addr, smp->sm_bia, ETHERADDRL);
476
477 done:
478         /* Initialize the statistics */
479         if ((rc = sfxge_mac_kstat_init(sp)) != 0)
480                 goto fail4;
481
482         if ((rc = sfxge_phy_kstat_init(sp)) != 0)
483                 goto fail5;
484
485         smp->sm_state = SFXGE_MAC_INITIALIZED;
486
487         return (0);
488
489 fail5:
490         DTRACE_PROBE(fail5);
491
492         sfxge_mac_kstat_fini(sp);
493 fail4:
494         DTRACE_PROBE(fail4);
495
496         /* Tear down DMA setup */
497         sfxge_dma_buffer_destroy(esmp);
498 fail3:
499         DTRACE_PROBE(fail3);
500
501         sfxge_phy_fini(sp);
502 fail2:
503         DTRACE_PROBE(fail2);
504
505         /* Destroy the link poll taskq */
506         ddi_taskq_destroy(smp->sm_tqp);
507         smp->sm_tqp = NULL;
508
509 fail1:
510         DTRACE_PROBE1(fail1, int, rc);
511
512         cv_destroy(&(smp->sm_link_poll_kv));
513
514         mutex_destroy(&(smp->sm_lock));
515
516         smp->sm_sp = NULL;
517
518         SFXGE_OBJ_CHECK(smp, sfxge_mac_t);
519
520         return (rc);
521 }
522
523 int
```

```
524 sfxge_mac_start(sfxge_t *sp, boolean_t restart)
525 {
526         sfxge_mac_t *smp = &(sp->s_mac);
527         efsys_mem_t *esmp = &(smp->sm_mem);
528         efx_nic_t *enp = sp->s_enp;
529         size_t pdu;
530         int rc;

532         mutex_enter(&(smp->sm_lock));

534         ASSERT3U(smp->sm_state, ==, SFXGE_MAC_INITIALIZED);

536         if ((rc = efx_port_init(enp)) != 0)
537                 goto fail1;

539         /*
540          * Set up the advertised capabilities that may have been asked for
541          * before the call to efx_port_init().
542          */
543         if ((rc = sfxge_phy_cap_apply(sp, !restart)) != 0)
544                 goto fail2;

546         /* Set the SDU */
547         pdu = EFX_MAC_PDU(sp->s_mtu);
548         if ((rc = efx_mac_pdu_set(enp, pdu)) != 0)
549                 goto fail3;

551         if ((rc = efx_mac_fcntl_set(enp, smp->sm_fcntl, B_TRUE)) != 0)
552                 goto fail4;

554         /* Set the unicast address */
555         if ((rc = efx_mac_addr_set(enp, (smp->sm_laa_valid) ?
556             smp->sm_laa : smp->sm_bia)) != 0)
557                 goto fail5;

559         /* Set the unicast filter */
560         if ((rc = efx_mac_filter_set(enp,
561             (smp->sm_promisc == SFXGE_PROMISC_ALL_PHYS), B_TRUE)) != 0) {
562                 goto fail6;
563         };

565         /* Set the group hash */
566         if (smp->sm_promisc >= SFXGE_PROMISC_ALL_MULTI) {
567                 unsigned int bucket[EFX_MAC_HASH_BITS];
568                 unsigned int index;

570                 for (index = 0; index < EFX_MAC_HASH_BITS; index++)
571                         bucket[index] = 1;

573                 if ((rc = efx_mac_hash_set(enp, bucket)) != 0)
574                         goto fail7;
575         } else {
576                 if ((rc = efx_mac_hash_set(enp, smp->sm_bucket)) != 0)
577                         goto fail8;
578         }

580         if (!smp->sm_mac_stats_timer_reqd) {
581                 if ((rc = efx_mac_stats_periodic(enp, esmp,
582                     SFXGE_MAC_POLL_PERIOD_MS, B_FALSE)) != 0)
583                         goto fail9;
584         }

586         if ((rc = efx_mac_drain(enp, B_FALSE)) != 0)
587                 goto fail10;

589         smp->sm_state = SFXGE_MAC_STARTED;
```



```
591 #ifdef _USE_MAC_PRIV_PROP
592         sfxge_gld_priv_prop_rename(sp);
593 #endif

595         /*
596          * Start link state polling. For hardware that reports link change
597          * events we still poll once to update the initial link state.
598          */
599         sfxge_mac_poll_start(sp);

601         mutex_exit(&(smp->sm_lock));
602         return (0);

604 fail10:
605         DTRACE_PROBE(fail10);
606         (void) efx_mac_stats_periodic(enp, esmp, 0, B_FALSE);
607 fail9:
608         DTRACE_PROBE(fail9);
609 fail8:
610         DTRACE_PROBE(fail8);
611 fail7:
612         DTRACE_PROBE(fail7);
613 fail6:
614         DTRACE_PROBE(fail6);
615 fail5:
616         DTRACE_PROBE(fail5);
617 fail4:
618         DTRACE_PROBE(fail4);
619 fail3:
620         DTRACE_PROBE(fail3);
621 fail2:
622         DTRACE_PROBE(fail2);
623         efx_port_fini(enp);
624 fail1:
625         DTRACE_PROBE1(fail1, int, rc);

627         mutex_exit(&(smp->sm_lock));

629         return (rc);
630 }


633 static void
634 sfxge_mac_link_update_locked(sfxge_t *sp, efx_link_mode_t mode)
635 {
636         sfxge_mac_t *smp = &(sp->s_mac);
637         const char *change, *duplex;
638         char info[sizeof (": now 10000Mbps FULL duplex")];

640         ASSERT(mutex_owned(&(smp->sm_lock)));
641         if (smp->sm_state != SFXGE_MAC_STARTED)
642                 return;

644         if (smp->sm_link_mode == mode)
645                 return;

647         smp->sm_link_mode = mode;
648         smp->sm_link_up = B_TRUE;

650         switch (smp->sm_link_mode) {
651         case EFX_LINK_UNKNOWN:
652         case EFX_LINK_DOWN:
653                 smp->sm_link_speed = 0;
654                 smp->sm_link_duplex = SFXGE_LINK_DUPLEX_UNKNOWN;
655                 smp->sm_link_up = B_FALSE;
```

```
656                 break;

658         case EFX_LINK_10HDX:
659         case EFX_LINK_10FDX:
660                 smp->sm_link_speed = 10;
661                 smp->sm_link_duplex = (smp->sm_link_mode == EFX_LINK_10HDX) ?
662                     SFXGE_LINK_DUPLEX_HALF : SFXGE_LINK_DUPLEX_FULL;
663                 break;

665         case EFX_LINK_100HDX:
666         case EFX_LINK_100FDX:
667                 smp->sm_link_speed = 100;
668                 smp->sm_link_duplex = (smp->sm_link_mode == EFX_LINK_100HDX) ?
669                     SFXGE_LINK_DUPLEX_HALF : SFXGE_LINK_DUPLEX_FULL;
670                 break;

672         case EFX_LINK_1000HDX:
673         case EFX_LINK_1000FDX:
674                 smp->sm_link_speed = 1000;
675                 smp->sm_link_duplex = (smp->sm_link_mode == EFX_LINK_1000HDX) ?
676                     SFXGE_LINK_DUPLEX_HALF : SFXGE_LINK_DUPLEX_FULL;
677                 break;

679         case EFX_LINK_10000FDX:
680                 smp->sm_link_speed = 10000;
681                 smp->sm_link_duplex = SFXGE_LINK_DUPLEX_FULL;
682                 break;

684         default:
685                 ASSERT(B_FALSE);
686                 break;
687         }

689         duplex = (smp->sm_link_duplex == SFXGE_LINK_DUPLEX_FULL) ?
690             "full" : "half";
691         change = (smp->sm_link_up) ? "UP" : "DOWN";
692         snprintf(info, sizeof (info), ": now %dMbps %s duplex",
693             smp->sm_link_speed, duplex);

695         cmn_err(CE_NOTE, SFXGE_CMN_ERR "[%s%d] Link %s%s",
696             ddi_driver_name(sp->s_dip), ddi_get_instance(sp->s_dip),
697             change, smp->sm_link_up ? info : "");

699         /* Push link state update to the OS */
700         sfxge_gld_link_update(sp);
701 }

703 void
704 sfxge_mac_link_update(sfxge_t *sp, efx_link_mode_t mode)
705 {
706         sfxge_mac_t *smp = &(sp->s_mac);

708         mutex_enter(&(smp->sm_lock));
709         sfxge_mac_link_update_locked(sp, mode);
710         mutex_exit(&(smp->sm_lock));
711 }

713 void
714 sfxge_mac_link_check(sfxge_t *sp, boolean_t *upp)
715 {
716         sfxge_mac_t *smp = &(sp->s_mac);

718         mutex_enter(&(smp->sm_lock));
719         *upp = smp->sm_link_up;
720         mutex_exit(&(smp->sm_lock));
721 }
```



```
723 void
724 sfxge_mac_link_speed_get(sfxge_t *sp, unsigned int *speedp)
725 {
726         sfxge_mac_t *smp = &(sp->s_mac);

728         mutex_enter(&(smp->sm_lock));
729         *speedp = smp->sm_link_speed;
730         mutex_exit(&(smp->sm_lock));
731 }

733 void
734 sfxge_mac_link_duplex_get(sfxge_t *sp, sfxge_link_duplex_t *duplexp)
735 {
736         sfxge_mac_t *smp = &(sp->s_mac);

738         mutex_enter(&(smp->sm_lock));
739         *duplexp = smp->sm_link_duplex;
740         mutex_exit(&(smp->sm_lock));
741 }

743 void
744 sfxge_mac_fcntl_get(sfxge_t *sp, unsigned int *fcntlp)
745 {
746         sfxge_mac_t *smp = &(sp->s_mac);

748         mutex_enter(&(smp->sm_lock));
749         *fcntlp = smp->sm_fcntl;
750         mutex_exit(&(smp->sm_lock));
751 }

753 int
754 sfxge_mac_fcntl_set(sfxge_t *sp, unsigned int fcntl)
755 {
756         sfxge_mac_t *smp = &(sp->s_mac);
757         int rc;

759         mutex_enter(&(smp->sm_lock));

761         if (smp->sm_fcntl == fcntl)
762                 goto done;

764         smp->sm_fcntl = fcntl;

766         if (smp->sm_state != SFXGE_MAC_STARTED)
767                 goto done;

769         if ((rc = efx_mac_fcntl_set(sp->s_enp, smp->sm_fcntl, B_TRUE)) != 0)
770                 goto fail1;

772 done:
773         mutex_exit(&(smp->sm_lock));

775         return (0);

777 fail1:
778         DTRACE_PROBE1(fail1, int, rc);

780         mutex_exit(&(smp->sm_lock));

782         return (rc);
783 }

785 int
786 sfxge_mac_unicst_get(sfxge_t *sp, sfxge_unicst_type_t type, uint8_t *addr)
787 {
```

```
788         sfxge_mac_t *smp = &(sp->s_mac);
789         int rc;

791         if (type >= SFXGE_UNICST_NTYPES) {
792                 rc = EINVAL;
793                 goto fail1;
794         }

796         mutex_enter(&(smp->sm_lock));

798         if (smp->sm_state != SFXGE_MAC_INITIALIZED &&
799             smp->sm_state != SFXGE_MAC_STARTED) {
800                 rc = EFAULT;
801                 goto fail2;
802         }

804         switch (type) {
805         case SFXGE_UNICST_BIA:
806                 bcopy(smp->sm_bia, addr, ETHERADDRL);
807                 break;

809         case SFXGE_UNICST_LAA:
810                 if (!(smp->sm_laa_valid)) {
811                         rc = ENOENT;
812                         goto fail3;
813                 }

815                 bcopy(smp->sm_laa, addr, ETHERADDRL);
816                 break;

818         default:
819                 ASSERT(B_FALSE);
820                 break;
821         }

823         mutex_exit(&(smp->sm_lock));

825         return (0);


828 fail3:
829         DTRACE_PROBE(fail3);
830 fail2:
831         DTRACE_PROBE(fail2);

833         mutex_exit(&(smp->sm_lock));

835 fail1:
836         DTRACE_PROBE1(fail1, int, rc);

838         return (rc);
839 }

841 int
842 sfxge_mac_unicst_set(sfxge_t *sp, uint8_t *addr)
843 {
844         sfxge_mac_t *smp = &(sp->s_mac);
845         efx_nic_t *enp = sp->s_enp;
846         int rc;

848         mutex_enter(&(smp->sm_lock));

850         bcopy(addr, smp->sm_laa, ETHERADDRL);
851         smp->sm_laa_valid = B_TRUE;

853         if (smp->sm_state != SFXGE_MAC_STARTED)
```



```
854                 goto done;

856         if ((rc = efx_mac_addr_set(enp, smp->sm_laa)) != 0)
857                 goto fail1;

859 done:
860         mutex_exit(&(smp->sm_lock));

862         return (0);

864 fail1:
865         DTRACE_PROBE1(fail1, int, rc);

867         mutex_exit(&(smp->sm_lock));

869         return (rc);
870 }

872 int
873 sfxge_mac_promisc_set(sfxge_t *sp, sfxge_promisc_type_t promisc)
874 {
875         sfxge_mac_t *smp = &(sp->s_mac);
876         efx_nic_t *enp = sp->s_enp;
877         int rc;

879         mutex_enter(&(smp->sm_lock));

881         if (smp->sm_promisc == promisc)
882                 goto done;

884         smp->sm_promisc = promisc;

886         if (smp->sm_state != SFXGE_MAC_STARTED)
887                 goto done;

889         if ((rc = efx_mac_filter_set(enp, (promisc == SFXGE_PROMISC_ALL_PHYS),
890             B_TRUE)) != 0)
891                 goto fail1;

893         if (promisc >= SFXGE_PROMISC_ALL_MULTI) {
894                 unsigned int bucket[EFX_MAC_HASH_BITS];
895                 unsigned int index;

897                 for (index = 0; index < EFX_MAC_HASH_BITS; index++)
898                         bucket[index] = 1;

900                 if ((rc = efx_mac_hash_set(enp, bucket)) != 0)
901                         goto fail2;
902         } else {
903                 if ((rc = efx_mac_hash_set(enp, smp->sm_bucket)) != 0)
904                         goto fail3;
905         }

907 done:
908         mutex_exit(&(smp->sm_lock));
909         return (0);

911 fail3:
912         DTRACE_PROBE(fail3);
913 fail2:
914         DTRACE_PROBE(fail2);
915 fail1:
916         DTRACE_PROBE1(fail1, int, rc);
917         mutex_exit(&(smp->sm_lock));

919         return (rc);
```

```
}

int
sfxge_mac_multicst_add(sfxge_t *sp, uint8_t *addr)
{
        sfxge_mac_t *smp = &(sp->s_mac);
        efx_nic_t *enp = sp->s_enp;
        uint32_t crc;
        int rc;

        mutex_enter(&(smp->sm_lock));

        CRC32(crc, addr, ETHERADDRL, 0xffffffff, crc32_table);
        smp->sm_bucket[crc % EFX_MAC_HASH_BITS]++;

        if (smp->sm_state != SFXGE_MAC_STARTED)
                goto done;

        if (smp->sm_promisc >= SFXGE_PROMISC_ALL_MULTI)
                goto done;

        if ((rc = efx_mac_hash_set(enp, smp->sm_bucket)) != 0)
                goto fail1;

done:
        mutex_exit(&(smp->sm_lock));
        return (0);

fail1:
        DTRACE_PROBE1(fail1, int, rc);
        mutex_exit(&(smp->sm_lock));

        return (rc);
}

int
sfxge_mac_multicst_remove(sfxge_t *sp, uint8_t *addr)
{
        sfxge_mac_t *smp = &(sp->s_mac);
        efx_nic_t *enp = sp->s_enp;
        uint32_t crc;
        int rc;

        mutex_enter(&(smp->sm_lock));

        CRC32(crc, addr, ETHERADDRL, 0xffffffff, crc32_table);
        ASSERT(smp->sm_bucket[crc % EFX_MAC_HASH_BITS] != 0);
        smp->sm_bucket[crc % EFX_MAC_HASH_BITS]--;

        if (smp->sm_state != SFXGE_MAC_STARTED)
                goto done;

        if (smp->sm_promisc >= SFXGE_PROMISC_ALL_MULTI)
                goto done;

        if ((rc = efx_mac_hash_set(enp, smp->sm_bucket)) != 0)
                goto fail1;

done:
        mutex_exit(&(smp->sm_lock));
        return (0);

fail1:
        DTRACE_PROBE1(fail1, int, rc);
        mutex_exit(&(smp->sm_lock));
```



```
        return (rc);
}

static int
sfxge_mac_loopback_set(sfxge_t *sp, efx_loopback_type_t type)
{
        sfxge_mac_t *smp = &(sp->s_mac);
        efx_nic_t *enp = sp->s_enp;
        int rc;

        mutex_enter(&(smp->sm_lock));
        ASSERT3U(smp->sm_state, ==, SFXGE_MAC_STARTED);

        if ((rc = efx_port_loopback_set(enp, EFX_LINK_UNKNOWN, type)) != 0)
                goto fail1;

        mutex_exit(&(smp->sm_lock));
        return (0);

fail1:
        DTRACE_PROBE1(fail1, int, rc);
        mutex_exit(&(smp->sm_lock));

        return (rc);
}

int
sfxge_mac_ioctl(sfxge_t *sp, sfxge_mac_ioc_t *smip)
{
        int rc;

        switch (smip->smi_op) {
        case SFXGE_MAC_OP_LOOPBACK: {
                efx_loopback_type_t type = smip->smi_data;

                if ((rc = sfxge_mac_loopback_set(sp, type)) != 0)
                        goto fail1;

                break;
        }
        default:
                rc = ENOTSUP;
                goto fail1;
        }

        return (0);

fail1:
        DTRACE_PROBE1(fail1, int, rc);

        return (rc);
}

void
sfxge_mac_stop(sfxge_t *sp)
{
        sfxge_mac_t *smp = &(sp->s_mac);
        efx_nic_t *enp = sp->s_enp;
        efsys_mem_t *esmp = &(smp->sm_mem);

        mutex_enter(&(smp->sm_lock));

        ASSERT3U(smp->sm_state, ==, SFXGE_MAC_STARTED);
        ASSERT3P(smp->sm_sp, ==, sp);
        smp->sm_state = SFXGE_MAC_INITIALIZED;
```

```
1052         /* If stopping in response to an MC reboot this may fail */
1053         if (!smp->sm_mac_stats_timer_reqd)
1054                 (void) efx_mac_stats_periodic(enp, esmp, 0, B_FALSE);

1056         sfxge_mac_poll_stop(sp);

1058         smp->sm_lbolt = 0;

1060         smp->sm_link_up = B_FALSE;
1061         smp->sm_link_speed = 0;
1062         smp->sm_link_duplex = SFXGE_LINK_DUPLEX_UNKNOWN;

1064         /* This may call MCDI */
1065         (void) efx_mac_drain(enp, B_TRUE);

1067         smp->sm_link_mode = EFX_LINK_UNKNOWN;


1070         efx_port_fini(enp);

1072         mutex_exit(&(smp->sm_lock));
1073 }

1075 void
1076 sfxge_mac_fini(sfxge_t *sp)
1077 {
1078         sfxge_mac_t *smp = &(sp->s_mac);
1079         efsys_mem_t *esmp = &(smp->sm_mem);
1080         unsigned int index;

1082         ASSERT3U(smp->sm_state, ==, SFXGE_MAC_INITIALIZED);
1083         ASSERT3P(smp->sm_sp, ==, sp);

1085         /* Tear down the statistics */
1086         sfxge_phy_kstat_fini(sp);
1087         sfxge_mac_kstat_fini(sp);

1089         smp->sm_state = SFXGE_MAC_UNINITIALIZED;
1090         smp->sm_link_mode = EFX_LINK_UNKNOWN;
1091         smp->sm_promisc = SFXGE_PROMISC_OFF;

1093         /* Clear the group hash */
1094         for (index = 0; index < EFX_MAC_HASH_BITS; index++)
1095                 smp->sm_bucket[index] = 0;

1097         bzero(smp->sm_laa, ETHERADDRL);
1098         smp->sm_laa_valid = B_FALSE;

1100         bzero(smp->sm_bia, ETHERADDRL);

1102         smp->sm_fcntl = 0;

1104         /* Finish with PHY DMA memory */
1105         sfxge_phy_fini(sp);

1107         /* Teardown the DMA */
1108         sfxge_dma_buffer_destroy(esmp);

1110         /* Destroy the link poll taskq */
1111         ddi_taskq_destroy(smp->sm_tqp);
1112         smp->sm_tqp = NULL;

1114         mutex_destroy(&(smp->sm_lock));

1116         smp->sm_sp = NULL;
```



```
1118         SFXGE_OBJ_CHECK(smp, sfxge_mac_t);
1119 }
1120 #endif /* ! codereview */
```

```
**********************************************************
    7502 Thu Aug 22 18:59:28 2013
new/usr/src/uts/common/io/sfxge/sfxge_mcdi.c
Merged sfxge driver
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2009 Solarflare Communications Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#include <sys/types.h>
#include <sys/ddi.h>
#include <sys/sunddi.h>
#include <sys/stream.h>
#include <sys/dlpi.h>

#include "sfxge.h"
#include "efsys.h"
#include "efx.h"
#include "efx_mcdi.h"
#include "efx_regs_mcdi.h"


/*
 * Notes on MCDI operation:
 * ------------------------
 * MCDI requests can be made in arbitrary thread context, and as a synchronous
 * API must therefore block until the response is available from the MC, or
 * a watchdog timeout occurs.
 *
 * This interacts badly with the limited number of worker threads (2 per CPU)
 * used by the Solaris callout subsystem to invoke timeout handlers. If both
 * worker threads are blocked (e.g. waiting for a condvar or mutex) then timeout
 * processing is deadlocked on that CPU, causing system failure.
 *
 * For this reason the driver does not use event based MCDI completion, as this
 * leads to numerous paths involving timeouts and reentrant GLDv3 entrypoints
 * that result in a deadlocked system.
 */
#define SFXGE_MCDI_POLL_INTERVAL        10              /* 10us in 1us units */
#define SFXGE_MCDI_WATCHDOG_INTERVAL    10000000        /* 10s in 1us units */


/* Acquire exclusive access to MCDI for the duration of a request */
static void
```



```
sfxge_mcdi_acquire(sfxge_mcdi_t *smp)
{
        mutex_enter(&(smp->sm_lock));
        ASSERT3U(smp->sm_state, !=, SFXGE_MCDI_UNINITIALIZED);

        while (smp->sm_state != SFXGE_MCDI_INITIALIZED) {
                (void) cv_wait_sig(&(smp->sm_kv), &(smp->sm_lock));
        }
        smp->sm_state = SFXGE_MCDI_BUSY;

        mutex_exit(&(smp->sm_lock));
}


/* Release ownership of MCDI on request completion */
static void
sfxge_mcdi_release(sfxge_mcdi_t *smp)
{
        mutex_enter(&(smp->sm_lock));
        ASSERT((smp->sm_state == SFXGE_MCDI_BUSY) ||
            (smp->sm_state == SFXGE_MCDI_COMPLETED));

        smp->sm_state = SFXGE_MCDI_INITIALIZED;
        cv_broadcast(&(smp->sm_kv));

        mutex_exit(&(smp->sm_lock));
}


static void
sfxge_mcdi_timeout(sfxge_t *sp)
{
        dev_info_t *dip = sp->s_dip;

        cmn_err(CE_WARN, SFXGE_CMN_ERR "[%s%d] MC_TIMEOUT",
            ddi_driver_name(dip), ddi_get_instance(dip));

        DTRACE_PROBE(mcdi_timeout);
        (void) sfxge_restart_dispatch(sp, DDI_SLEEP, SFXGE_HW_ERR,
            "MCDI timeout", 0);
}


static void
sfxge_mcdi_poll(sfxge_t *sp)
{
        efx_nic_t *enp = sp->s_enp;
        clock_t timeout;
        boolean_t aborted;

        /* Poll until request completes or timeout */
        timeout = ddi_get_lbolt() + drv_usectohz(SFXGE_MCDI_WATCHDOG_INTERVAL);
        while (efx_mcdi_request_poll(enp) == B_FALSE) {

                /* No response received yet */
                if (ddi_get_lbolt() > timeout) {
                        /* Timeout expired */
                        goto fail;
                }

                /* Short delay to avoid excessive PCIe traffic */
                drv_usecwait(SFXGE_MCDI_POLL_INTERVAL);
        }

        /* Request completed (or polling failed) */
        return;
```

```
129 fail:
130         /* Timeout before request completion */
131         DTRACE_PROBE(fail);
132         aborted = efx_mcdi_request_abort(enp);
133         ASSERT(aborted);
134         sfxge_mcdi_timeout(sp);
135 }


138 static void
139 sfxge_mcdi_execute(void *arg, efx_mcdi_req_t *emrp)
140 {
141         sfxge_t *sp = (sfxge_t *)arg;
142         sfxge_mcdi_t *smp = &(sp->s_mcdi);

144         sfxge_mcdi_acquire(smp);

146         /* Issue request and poll for completion */
147         efx_mcdi_request_start(sp->s_enp, emrp, B_FALSE);
148         sfxge_mcdi_poll(sp);

150         sfxge_mcdi_release(smp);
151 }


154 static void
155 sfxge_mcdi_ev_cpl(void *arg)
156 {
157         sfxge_t *sp = (sfxge_t *)arg;
158         sfxge_mcdi_t *smp = &(sp->s_mcdi);

160         mutex_enter(&(smp->sm_lock));
161         ASSERT(smp->sm_state == SFXGE_MCDI_BUSY);
162         smp->sm_state = SFXGE_MCDI_COMPLETED;
163         cv_broadcast(&(smp->sm_kv));
164         mutex_exit(&(smp->sm_lock));
165 }


168 static void
169 sfxge_mcdi_exception(void *arg, efx_mcdi_exception_t eme)
170 {
171         sfxge_t *sp = (sfxge_t *)arg;
172         const char *reason;

174         if (eme == EFX_MCDI_EXCEPTION_MC_REBOOT)
175                 reason = "MC_REBOOT";
176         else if (eme == EFX_MCDI_EXCEPTION_MC_BADASSERT)
177                 reason = "MC_BADASSERT";
178         else
179                 reason = "MC_UNKNOWN";

181         DTRACE_PROBE(mcdi_exception);
182         /* sfxge_evq_t->se_lock held */
183         (void) sfxge_restart_dispatch(sp, DDI_SLEEP, SFXGE_HW_ERR, reason, 0);
184 }


187 int
188 sfxge_mcdi_init(sfxge_t *sp)
189 {
190         efx_nic_t *enp = sp->s_enp;
191         sfxge_mcdi_t *smp = &(sp->s_mcdi);
192         efx_mcdi_transport_t *emtp = &(smp->sm_emt);
193         int rc;
```



```
195         ASSERT3U(smp->sm_state, ==, SFXGE_MCDI_UNINITIALIZED);

197         mutex_init(&(smp->sm_lock), NULL, MUTEX_DRIVER, NULL);

199         smp->sm_state = SFXGE_MCDI_INITIALIZED;

201         emtp->emt_context = sp;
202         emtp->emt_execute = sfxge_mcdi_execute;
203         emtp->emt_ev_cpl = sfxge_mcdi_ev_cpl;
204         emtp->emt_exception = sfxge_mcdi_exception;

206         cv_init(&(smp->sm_kv), NULL, CV_DRIVER, NULL);

208         if ((rc = efx_mcdi_init(enp, emtp)) != 0)
209                 goto fail1;

211         return (0);

213 fail1:
214         DTRACE_PROBE1(fail1, int, rc);

216         cv_destroy(&(smp->sm_kv));
217         mutex_destroy(&(smp->sm_lock));

219         smp->sm_state = SFXGE_MCDI_UNINITIALIZED;
220         smp->sm_sp = NULL;
221         SFXGE_OBJ_CHECK(smp, sfxge_mcdi_t);

223         return (rc);
224 }


227 void
228 sfxge_mcdi_fini(sfxge_t *sp)
229 {
230         efx_nic_t *enp = sp->s_enp;
231         sfxge_mcdi_t *smp = &(sp->s_mcdi);
232         efx_mcdi_transport_t *emtp;

234         mutex_enter(&(smp->sm_lock));
235         ASSERT3U(smp->sm_state, ==, SFXGE_MCDI_INITIALIZED);

237         efx_mcdi_fini(enp);
238         emtp = &(smp->sm_emt);
239         bzero(emtp, sizeof (*emtp));

241         smp->sm_sp = NULL;

243         cv_destroy(&(smp->sm_kv));
244         mutex_exit(&(smp->sm_lock));

246         mutex_destroy(&(smp->sm_lock));

248         smp->sm_state = SFXGE_MCDI_UNINITIALIZED;
249         SFXGE_OBJ_CHECK(smp, sfxge_mcdi_t);
250 }


253 int
254 sfxge_mcdi_ioctl(sfxge_t *sp, sfxge_mcdi_ioc_t *smip)
255 {
256         const efx_nic_cfg_t *encp = efx_nic_cfg_get(sp->s_enp);
257         sfxge_mcdi_t *smp = &(sp->s_mcdi);
258         efx_mcdi_req_t emr;
259         uint8_t *out;
```

```
260         int rc;

262         if (smp->sm_state == SFXGE_MCDI_UNINITIALIZED) {
263                 rc = ENODEV;
264                 goto fail1;
265         }

267         if (!(encp->enc_features & EFX_FEATURE_MCDI)) {
268                 rc = ENOTSUP;
269                 goto fail2;
270         }

272         if ((out = kmem_zalloc(sizeof (smip->smi_payload), KM_SLEEP)) == NULL) {
273                 rc = ENOMEM;
274                 goto fail3;
275         }

277         emr.emr_cmd = smip->smi_cmd;
278         emr.emr_in_buf = smip->smi_payload;
279         emr.emr_in_length = smip->smi_len;

281         emr.emr_out_buf = out;
282         emr.emr_out_length = sizeof (smip->smi_payload);

284         sfxge_mcdi_execute(sp, &emr);

286         smip->smi_rc = (uint8_t)emr.emr_rc;
287         smip->smi_cmd = (uint8_t)emr.emr_cmd;
288         smip->smi_len = (uint8_t)emr.emr_out_length_used;
289         memcpy(smip->smi_payload, out, smip->smi_len);

291         /*
292          * Helpfully trigger a device reset in response to an MCDI_CMD_REBOOT
293          * Both ports will see ->emt_exception callbacks on the next MCDI poll
294          */
295         if (smip->smi_cmd == MC_CMD_REBOOT) {

297                 DTRACE_PROBE(mcdi_ioctl_mc_reboot);
298                 /* sfxge_t->s_state_lock held */
299                 (void) sfxge_restart_dispatch(sp, DDI_SLEEP, SFXGE_HW_OK,
300                     "MC_REBOOT triggering restart", 0);
301         }

303         kmem_free(out, sizeof (smip->smi_payload));

305         return (0);

307 fail3:
308         DTRACE_PROBE(fail3);
309 fail2:
310         DTRACE_PROBE(fail2);
311 fail1:
312         DTRACE_PROBE1(fail1, int, rc);
313         return (rc);
314 }
315 #endif /* ! codereview */
```

```
**********************************************************
    6846 Thu Aug 22 18:59:28 2013
new/usr/src/uts/common/io/sfxge/sfxge_mon.c
Merged sfxge driver
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2008-2013 Solarflare Communications Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#include <sys/types.h>
#include <sys/sysmacros.h>
#include <sys/ddi.h>
#include <sys/sunddi.h>
#include <sys/cyclic.h>

#include "sfxge.h"

#include "efx.h"

/* Monitor DMA attributes */
static ddi_device_acc_attr_t sfxge_mon_devacc = {

        DDI_DEVICE_ATTR_V0,     /* devacc_attr_version */
        DDI_NEVERSWAP_ACC,      /* devacc_attr_endian_flags */
        DDI_STRICTORDER_ACC     /* devacc_attr_dataorder */
};

static ddi_dma_attr_t sfxge_mon_dma_attr = {
        DMA_ATTR_V0,            /* dma_attr_version     */
        0,                      /* dma_attr_addr_lo     */
        0xffffffffffffffffull,  /* dma_attr_addr_hi     */
        0xffffffffffffffffull,  /* dma_attr_count_max   */
        0x1000,                 /* dma_attr_align       */
        0xffffffff,             /* dma_attr_burstsizes  */
        1,                      /* dma_attr_minxfer     */
        0xffffffffffffffffull,  /* dma_attr_maxxfer     */
        0xffffffffffffffffull,  /* dma_attr_seg         */
        1,                      /* dma_attr_sgllen      */
        1,                      /* dma_attr_granular    */
        0                       /* dma_attr_flags       */
};


static int
```



```
sfxge_mon_kstat_update(kstat_t *ksp, int rw)
{
        sfxge_t *sp = ksp->ks_private;
        sfxge_mon_t *smp = &(sp->s_mon);
        efsys_mem_t *esmp = &(smp->sm_mem);
        efx_nic_t *enp = sp->s_enp;
        kstat_named_t *knp;
        int rc, sn;

        if (rw != KSTAT_READ) {
                rc = EACCES;
                goto fail1;
        }

        ASSERT(mutex_owned(&(smp->sm_lock)));

        if (smp->sm_state != SFXGE_MON_STARTED)
                goto done;

        /* Synchronize the DMA memory for reading */
        (void) ddi_dma_sync(smp->sm_mem.esm_dma_handle,
            0,
            EFX_MON_STATS_SIZE,
            DDI_DMA_SYNC_FORKERNEL);

        if ((rc = efx_mon_stats_update(enp, esmp, smp->sm_statbuf)) != 0)
                goto fail2;

        knp = smp->sm_stat;
        for (sn = 0; sn < EFX_MON_NSTATS; sn++) {
                knp->value.ui64 = smp->sm_statbuf[sn].emsv_value;
                knp++;
        }

        knp->value.ui32 = sp->s_num_restarts;
        knp++;
        knp->value.ui32 = sp->s_num_restarts_hw_err;
        knp++;

done:
        return (0);

fail2:
        DTRACE_PROBE(fail2);
fail1:
        DTRACE_PROBE1(fail1, int, rc);

        return (rc);
}

static int
sfxge_mon_kstat_init(sfxge_t *sp)
{
        sfxge_mon_t *smp = &(sp->s_mon);
        dev_info_t *dip = sp->s_dip;
        efx_nic_t *enp = sp->s_enp;
        kstat_t *ksp;
        kstat_named_t *knp;
        char name[MAXNAMELEN];
        unsigned int id;
        int rc;

        if ((smp->sm_statbuf = kmem_zalloc(sizeof (uint32_t) * EFX_MON_NSTATS,
            KM_NOSLEEP)) == NULL) {
                rc = ENOMEM;
                goto fail1;
```

```
128         }
129
130         (void) snprintf(name, MAXNAMELEN - 1, "%s_%s", ddi_driver_name(dip),
131             efx_mon_name(enp));
132
133         /* Create the set */
134         if ((ksp = kstat_create((char *)ddi_driver_name(dip),
135             ddi_get_instance(dip), name, "mon", KSTAT_TYPE_NAMED,
136             EFX_MON_NSTATS+2, 0)) == NULL) {
137                 rc = ENOMEM;
138                 goto fail2;
139         }
140
141         smp->sm_ksp = ksp;
142
143         ksp->ks_update = sfxge_mon_kstat_update;
144         ksp->ks_private = sp;
145         ksp->ks_lock = &(smp->sm_lock);
146
147         /* Initialise the named stats */
148         smp->sm_stat = knp = ksp->ks_data;
149         for (id = 0; id < EFX_MON_NSTATS; id++) {
150                 kstat_named_init(knp, (char *)efx_mon_stat_name(enp, id),
151                     KSTAT_DATA_UINT64);
152                 knp++;
153         }
154         kstat_named_init(knp, "num_restarts", KSTAT_DATA_UINT32);
155         knp++;
156         kstat_named_init(knp, "num_restarts_hw_err", KSTAT_DATA_UINT32);
157         knp++;
158
159         kstat_install(ksp);
160
161         return (0);
162
163 fail2:
164         DTRACE_PROBE(fail2);
165         kmem_free(smp->sm_statbuf, sizeof (uint32_t) * EFX_MON_NSTATS);
166 fail1:
167         DTRACE_PROBE1(fail1, int, rc);
168
169         return (rc);
170 }
171
172 static void
173 sfxge_mon_kstat_fini(sfxge_t *sp)
174 {
175         sfxge_mon_t *smp = &(sp->s_mon);
176
177         /* Destroy the set */
178         kstat_delete(smp->sm_ksp);
179         smp->sm_ksp = NULL;
180         smp->sm_stat = NULL;
181
182         kmem_free(smp->sm_statbuf, sizeof (uint32_t) * EFX_MON_NSTATS);
183 }
184
185 int
186 sfxge_mon_init(sfxge_t *sp)
187 {
188         sfxge_mon_t *smp = &(sp->s_mon);
189         efx_nic_t *enp = sp->s_enp;
190         efsys_mem_t *esmp = &(smp->sm_mem);
191         sfxge_dma_buffer_attr_t dma_attr;
192         const efx_nic_cfg_t *encp;
193         int rc;
```



```
194
195         SFXGE_OBJ_CHECK(smp, sfxge_mon_t);
196
197         ASSERT3U(smp->sm_state, ==, SFXGE_MON_UNINITIALIZED);
198
199         smp->sm_sp = sp;
200
201         mutex_init(&(smp->sm_lock), NULL, MUTEX_DRIVER, NULL);
202
203         dma_attr.sdba_dip        = sp->s_dip;
204         dma_attr.sdba_dattrp     = &sfxge_mon_dma_attr;
205         dma_attr.sdba_callback   = DDI_DMA_SLEEP;
206         dma_attr.sdba_length     = EFX_MON_STATS_SIZE;
207         dma_attr.sdba_memflags   = DDI_DMA_CONSISTENT;
208         dma_attr.sdba_devaccp    = &sfxge_mon_devacc;
209         dma_attr.sdba_bindflags  = DDI_DMA_READ | DDI_DMA_CONSISTENT;
210         dma_attr.sdba_maxcookies = 1;
211         dma_attr.sdba_zeroinit   = B_TRUE;
212
213         if ((rc = sfxge_dma_buffer_create(esmp, &dma_attr)) != 0)
214                 goto fail1;
215
216         encp = efx_nic_cfg_get(enp);
217         smp->sm_type = encp->enc_mon_type;
218
219         DTRACE_PROBE1(mon, efx_mon_type_t, smp->sm_type);
220
221         smp->sm_state = SFXGE_MON_INITIALIZED;
222
223         /* Initialize the statistics */
224         if ((rc = sfxge_mon_kstat_init(sp)) != 0)
225                 goto fail2;
226
227         return (0);
228
229 fail2:
230         DTRACE_PROBE(fail2);
231
232         /* Tear down DMA setup */
233         sfxge_dma_buffer_destroy(esmp);
234
235 fail1:
236         DTRACE_PROBE1(fail1, int, rc);
237         mutex_destroy(&(smp->sm_lock));
238
239         smp->sm_sp = NULL;
240
241         SFXGE_OBJ_CHECK(smp, sfxge_mac_t);
242
243         return (rc);
244 }
245
246 int
247 sfxge_mon_start(sfxge_t *sp)
248 {
249         sfxge_mon_t *smp = &(sp->s_mon);
250         int rc;
251
252         mutex_enter(&(smp->sm_lock));
253         ASSERT3U(smp->sm_state, ==, SFXGE_MON_INITIALIZED);
254
255         /* Initialize the MON module */
256         if ((rc = efx_mon_init(sp->s_enp)) != 0)
257                 goto fail1;
258
259         smp->sm_state = SFXGE_MON_STARTED;
```

```
261         mutex_exit(&(smp->sm_lock));

263         return (0);

265 fail1:
266         DTRACE_PROBE1(fail1, int, rc);

268         mutex_exit(&(smp->sm_lock));

270         return (rc);
271 }

273 void
274 sfxge_mon_stop(sfxge_t *sp)
275 {
276         sfxge_mon_t *smp = &(sp->s_mon);

278         mutex_enter(&(smp->sm_lock));

280         ASSERT3U(smp->sm_state, ==, SFXGE_MON_STARTED);
281         smp->sm_state = SFXGE_MON_INITIALIZED;

283         /* Tear down the MON module */
284         efx_mon_fini(sp->s_enp);

286         mutex_exit(&(smp->sm_lock));
287 }

289 void
290 sfxge_mon_fini(sfxge_t *sp)
291 {
292         sfxge_mon_t *smp = &(sp->s_mon);
293         efsys_mem_t *esmp = &(smp->sm_mem);

295         ASSERT3U(smp->sm_state, ==, SFXGE_MON_INITIALIZED);

297         /* Tear down the statistics */
298         sfxge_mon_kstat_fini(sp);

300         smp->sm_state = SFXGE_MON_UNINITIALIZED;
301         mutex_destroy(&(smp->sm_lock));

303         smp->sm_sp = NULL;
304         smp->sm_type = EFX_MON_INVALID;

306         /* Tear down DMA setup */
307         sfxge_dma_buffer_destroy(esmp);

309         SFXGE_OBJ_CHECK(smp, sfxge_mon_t);
310 }
311 #endif /* ! codereview */
```

```
**********************************************************
    4331 Thu Aug 22 18:59:28 2013
new/usr/src/uts/common/io/sfxge/sfxge_nvram.c
Merged sfxge driver
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2009 Solarflare Communications Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#include <sys/types.h>
#include <sys/ddi.h>
#include <sys/sunddi.h>
#include <sys/stream.h>
#include <sys/dlpi.h>

#include "sfxge.h"

int
sfxge_nvram_rw(sfxge_t *sp, sfxge_nvram_ioc_t *snip, efx_nvram_type_t type,
    boolean_t write)
{
        int (*op)(efx_nic_t *, efx_nvram_type_t, unsigned int, caddr_t, size_t);
        efx_nic_t *enp = sp->s_enp;
        size_t chunk_size;
        off_t off;
        int rc;

        op = (write) ? efx_nvram_write_chunk : efx_nvram_read_chunk;

        if ((rc = efx_nvram_rw_start(enp, type, &chunk_size)) != 0)
                goto fail1;

        off = 0;
        while (snip->sni_size) {
                size_t len = MIN(chunk_size, snip->sni_size);
                caddr_t buf = (caddr_t)(&snip->sni_data[off]);

                if ((rc = op(enp, type, snip->sni_offset + off, buf, len)) != 0)
                        goto fail2;

                snip->sni_size -= len;
                off += len;
        }
```



```
        efx_nvram_rw_finish(enp, type);
        return (0);

fail2:
        DTRACE_PROBE(fail2);
        efx_nvram_rw_finish(enp, type);
fail1:
        DTRACE_PROBE1(fail1, int, rc);
        return (rc);
}


int
sfxge_nvram_erase(sfxge_t *sp, sfxge_nvram_ioc_t *snip, efx_nvram_type_t type)
{
        efx_nic_t *enp = sp->s_enp;
        size_t chunk_size;
        int rc;

        if ((rc = efx_nvram_rw_start(enp, type, &chunk_size)) != 0)
                goto fail1;

        if ((rc = efx_nvram_erase(enp, type)) != 0)
                goto fail2;

        efx_nvram_rw_finish(enp, type);
        return (0);

fail2:
        DTRACE_PROBE(fail2);
        efx_nvram_rw_finish(enp, type);
fail1:
        DTRACE_PROBE1(fail1, int, rc);
        return (rc);
}

int
sfxge_nvram_ioctl(sfxge_t *sp, sfxge_nvram_ioc_t *snip)
{
        efx_nic_t *enp = sp->s_enp;
        efx_nvram_type_t type;
        int rc;

        if (snip->sni_type == SFXGE_NVRAM_TYPE_MC_GOLDEN &&
            (snip->sni_op == SFXGE_NVRAM_OP_WRITE ||
            snip->sni_op == SFXGE_NVRAM_OP_ERASE ||
            snip->sni_op == SFXGE_NVRAM_OP_SET_VER)) {
                rc = ENOTSUP;
                goto fail4;
        }

        switch (snip->sni_type) {
        case SFXGE_NVRAM_TYPE_BOOTROM:
                type = EFX_NVRAM_BOOTROM;
                break;
        case SFXGE_NVRAM_TYPE_BOOTROM_CFG:
                type = EFX_NVRAM_BOOTROM_CFG;
                break;
        case SFXGE_NVRAM_TYPE_MC:
                type = EFX_NVRAM_MC_FIRMWARE;
                break;
        case SFXGE_NVRAM_TYPE_MC_GOLDEN:
                type = EFX_NVRAM_MC_GOLDEN;
                break;
        case SFXGE_NVRAM_TYPE_PHY:
                type = EFX_NVRAM_PHY;
```

```
128                 break;
129         case SFXGE_NVRAM_TYPE_NULL_PHY:
130                 type = EFX_NVRAM_NULLPHY;
131                 break;
132         case SFXGE_NVRAM_TYPE_FPGA: /* PTP timestamping FPGA */
133                 type = EFX_NVRAM_FPGA;
134                 break;
135         default:
136                 rc = EINVAL;
137                 goto fail1;
138         }

140         if (snip->sni_size > sizeof (snip->sni_data)) {
141                 rc = ENOSPC;
142                 goto fail2;
143         }

145         switch (snip->sni_op) {
146         case SFXGE_NVRAM_OP_SIZE:
147         {
148                 size_t size;
149                 if ((rc = efx_nvram_size(enp, type, &size)) != 0)
150                         goto fail3;
151                 snip->sni_size = size;
152                 break;
153         }
154         case SFXGE_NVRAM_OP_READ:
155                 if ((rc = sfxge_nvram_rw(sp, snip, type, B_FALSE)) != 0)
156                         goto fail3;
157                 break;
158         case SFXGE_NVRAM_OP_WRITE:
159                 if ((rc = sfxge_nvram_rw(sp, snip, type, B_TRUE)) != 0)
160                         goto fail3;
161                 break;
162         case SFXGE_NVRAM_OP_ERASE:
163                 if ((rc = sfxge_nvram_erase(sp, snip, type)) != 0)
164                         goto fail3;
165                 break;
166         case SFXGE_NVRAM_OP_GET_VER:
167                 if ((rc = efx_nvram_get_version(enp, type, &snip->sni_subtype,
168                     &snip->sni_version[0])) != 0)
169                         goto fail3;
170                 break;
171         case SFXGE_NVRAM_OP_SET_VER:
172                 if ((rc = efx_nvram_set_version(enp, type,
173                     &snip->sni_version[0])) != 0)
174                         goto fail3;
175                 break;
176         default:
177                 rc = ENOTSUP;
178                 goto fail4;
179         }

181         return (0);

183 fail4:
184         DTRACE_PROBE(fail4);
185 fail3:
186         DTRACE_PROBE(fail3);
187 fail2:
188         DTRACE_PROBE(fail2);
189 fail1:
190         DTRACE_PROBE1(fail1, int, rc);

192         return (rc);
193 }
```



```
194 #endif /* ! codereview */
```

```
**********************************************************
    5105 Thu Aug 22 18:59:28 2013
new/usr/src/uts/common/io/sfxge/sfxge_pci.c
Merged sfxge driver
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2008-2013 Solarflare Communications Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#include <sys/types.h>
#include <sys/ddi.h>
#include <sys/sunddi.h>
#include <sys/pci.h>
#include <sys/pcie.h>

/* PCIe 2.0 link speeds */
#ifndef PCIE_LINKCAP_MAX_SPEED_5_0
#define PCIE_LINKCAP_MAX_SPEED_5_0      0x2
#endif
#ifndef PCIE_LINKSTS_SPEED_5_0
#define PCIE_LINKSTS_SPEED_5_0          0x2
#endif

#include "sfxge.h"

int
sfxge_pci_cap_find(sfxge_t *sp, uint8_t cap_id, off_t *offp)
{
        off_t off;
        uint16_t stat;
        int rc;

        stat = pci_config_get16(sp->s_pci_handle, PCI_CONF_STAT);

        if (!(stat & PCI_STAT_CAP)) {
                rc = ENOTSUP;
                goto fail1;
        }

        for (off = pci_config_get8(sp->s_pci_handle, PCI_CONF_CAP_PTR);
            off != PCI_CAP_NEXT_PTR_NULL;
            off = pci_config_get8(sp->s_pci_handle, off + PCI_CAP_NEXT_PTR)) {
                if (cap_id == pci_config_get8(sp->s_pci_handle,
                    off + PCI_CAP_ID))
```



```
                        goto done;
        }

        rc = ENOENT;
        goto fail2;

done:
        *offp = off;
        return (0);

fail2:
        DTRACE_PROBE(fail2);
fail1:
        DTRACE_PROBE1(fail1, int, rc);

        return (rc);
}

int
sfxge_pci_init(sfxge_t *sp)
{
        off_t off;
        uint16_t pciecap;
        uint16_t devctl;
        uint16_t linksts;
        uint16_t max_payload_size;
        uint16_t max_read_request;
        int rc;

        if (pci_config_setup(sp->s_dip, &(sp->s_pci_handle)) != DDI_SUCCESS) {
                rc = ENODEV;
                goto fail1;
        }

        sp->s_pci_venid = pci_config_get16(sp->s_pci_handle, PCI_CONF_VENID);
        sp->s_pci_devid = pci_config_get16(sp->s_pci_handle, PCI_CONF_DEVID);
        if ((rc = efx_family(sp->s_pci_venid, sp->s_pci_devid,
            &sp->s_family)) != 0)
                goto fail2;

        if ((rc = sfxge_pci_cap_find(sp, PCI_CAP_ID_PCI_E, &off)) != 0)
                goto fail3;

        pciecap = pci_config_get16(sp->s_pci_handle, off + PCIE_PCIECAP);
        ASSERT3U((pciecap & PCIE_PCIECAP_VER_MASK), >=, PCIE_PCIECAP_VER_1_0);

        linksts = pci_config_get16(sp->s_pci_handle, off + PCIE_LINKSTS);
        switch (linksts & PCIE_LINKSTS_NEG_WIDTH_MASK) {
        case PCIE_LINKSTS_NEG_WIDTH_X1:
                sp->s_pcie_nlanes = 1;
                break;

        case PCIE_LINKSTS_NEG_WIDTH_X2:
                sp->s_pcie_nlanes = 2;
                break;

        case PCIE_LINKSTS_NEG_WIDTH_X4:
                sp->s_pcie_nlanes = 4;
                break;

        case PCIE_LINKSTS_NEG_WIDTH_X8:
                sp->s_pcie_nlanes = 8;
                break;

        default:
                ASSERT(B_FALSE);
```

```
128                 break;
129         }

131         switch (linksts & PCIE_LINKSTS_SPEED_MASK) {
132         case PCIE_LINKSTS_SPEED_2_5:
133                 sp->s_pcie_linkspeed = 1;
134                 break;

136         case PCIE_LINKSTS_SPEED_5_0:
137                 sp->s_pcie_linkspeed = 2;
138                 break;

140         default:
141                 ASSERT(B_FALSE);
142                 break;
143         }

145         devctl = pci_config_get16(sp->s_pci_handle, off + PCIE_DEVCTL);

147         max_payload_size = (devctl & PCIE_DEVCTL_MAX_PAYLOAD_MASK)
148             >> PCIE_DEVCTL_MAX_PAYLOAD_SHIFT;

150         max_read_request = (devctl & PCIE_DEVCTL_MAX_READ_REQ_MASK)
151             >> PCIE_DEVCTL_MAX_READ_REQ_SHIFT;

153         cmn_err(CE_NOTE,
154             SFXGE_CMN_ERR "PCIe MRR: %d TLP: %d Link: %s Lanes: x%d",
155             128 << max_read_request,
156             128 << max_payload_size,
157             (sp->s_pcie_linkspeed == 1) ? "2.5G" :
158             (sp->s_pcie_linkspeed == 2) ? "5.0G" :
159             "UNKNOWN",
160             sp->s_pcie_nlanes);

162         return (0);

164 fail3:
165         DTRACE_PROBE(fail3);
166 fail2:
167         DTRACE_PROBE(fail2);

169         pci_config_teardown(&(sp->s_pci_handle));
170         sp->s_pci_handle = NULL;

172 fail1:
173         DTRACE_PROBE1(fail1, int, rc);

175         return (rc);
176 }

178 void
179 sfxge_pcie_check_link(sfxge_t *sp, unsigned int full_nlanes,
180         unsigned int full_speed)
181 {
182         if ((sp->s_pcie_linkspeed < full_speed) ||
183             (sp->s_pcie_nlanes    < full_nlanes))
184                 cmn_err(CE_NOTE,
185                     SFXGE_CMN_ERR "The %s%d device requires %d PCIe lanes "
186                     "at %s link speed to reach full bandwidth.",
187                     ddi_driver_name(sp->s_dip),
188                     ddi_get_instance(sp->s_dip),
189                     full_nlanes,
190                     (full_speed == 1) ? "2.5G" :
191                     (full_speed == 2) ? "5.0G" :
192                     "UNKNOWN");
193 }
```



```
195 int
196 sfxge_pci_ioctl(sfxge_t *sp, sfxge_pci_ioc_t *spip)
197 {
198         int rc;

200         switch (spip->spi_op) {
201         case SFXGE_PCI_OP_READ:
202                 spip->spi_data = pci_config_get8(sp->s_pci_handle,
203                     spip->spi_addr);
204                 break;

206         case SFXGE_PCI_OP_WRITE:
207                 pci_config_put8(sp->s_pci_handle,
208                     spip->spi_addr, spip->spi_data);
209                 break;

211         default:
212                 rc = ENOTSUP;
213                 goto fail1;
214         }

216         return (0);

218 fail1:
219         DTRACE_PROBE1(fail1, int, rc);

221         return (0);
222 }

224 void
225 sfxge_pci_fini(sfxge_t *sp)
226 {
227         sp->s_pcie_nlanes = 0;
228         sp->s_pcie_linkspeed = 0;

230         pci_config_teardown(&(sp->s_pci_handle));
231         sp->s_pci_handle = NULL;
232 }
233 #endif /* ! codereview */
```

```
**********************************************************
   13747 Thu Aug 22 18:59:28 2013
new/usr/src/uts/common/io/sfxge/sfxge_phy.c
Merged sfxge driver
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2008-2013 Solarflare Communications Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#include <sys/types.h>
#include <sys/sysmacros.h>
#include <sys/ddi.h>
#include <sys/sunddi.h>
#include <sys/cyclic.h>

#include "sfxge.h"
#include "efx.h"

/*
 * All efx_phy_*() must be after efx_port_init()
 *
 * LOCKING STRATEGY: Aquire sm_lock and test sm_state==SFXGE_MAC_STARTED
 * to serialise against sfxge_restart()
 *
 * Note that there is no seperate PHY lock
 * Everything is driven from MAC code and the MAC lock is used
 */

/* PHY DMA attributes */
static ddi_device_acc_attr_t sfxge_phy_devacc = {

        DDI_DEVICE_ATTR_V0,     /* devacc_attr_version */
        DDI_NEVERSWAP_ACC,      /* devacc_attr_endian_flags */
        DDI_STRICTORDER_ACC     /* devacc_attr_dataorder */
};

static ddi_dma_attr_t sfxge_phy_dma_attr = {
        DMA_ATTR_V0,            /* dma_attr_version     */
        0,                      /* dma_attr_addr_lo     */
        0xffffffffffffffffull,  /* dma_attr_addr_hi     */
        0xffffffffffffffffull,  /* dma_attr_count_max   */
        0x1000,                 /* dma_attr_align       */
        0xffffffff,             /* dma_attr_burstsizes  */
        1,                      /* dma_attr_minxfer     */
```



```
        0xffffffffffffffffull,  /* dma_attr_maxxfer     */
        0xffffffffffffffffull,  /* dma_attr_seg         */
        1,                      /* dma_attr_sgllen      */
        1,                      /* dma_attr_granular    */
        0                       /* dma_attr_flags       */
};


/* The common code requires the loader *without* the REFLASH_HEADER */
#if EFSYS_OPT_FALCON && EFSYS_OPT_NVRAM_SFT9001
static const uint8_t SFT9001_FULL_LOADER[] = {
#include "firmware/image.h"
#include "firmware/SFT9001A_LOADER.c"
};
const uint8_t * const sft9001_loader = SFT9001_FULL_LOADER +
    sizeof (image_header_t);
const size_t sft9001_loader_size = sizeof (SFT9001_FULL_LOADER) -
    sizeof (image_header_t);
#endif

#if EFSYS_OPT_FALCON && EFSYS_OPT_NVRAM_SFX7101
static const uint8_t SFX7101_FULL_LOADER[] = {
#include "firmware/image.h"
#include "firmware/SFX7101B_LOADER.c"
};
const uint8_t * const sfx7101_loader = SFX7101_FULL_LOADER +
    sizeof (image_header_t);
const size_t sfx7101_loader_size = sizeof (SFX7101_FULL_LOADER) -
    sizeof (image_header_t);
#endif


static int
sfxge_phy_kstat_update(kstat_t *ksp, int rw)
{
        sfxge_t *sp = ksp->ks_private;
        sfxge_mac_t *smp = &(sp->s_mac);
        sfxge_phy_t *spp = &(smp->sm_phy);
        efx_nic_t *enp = sp->s_enp;
        kstat_named_t *knp;
        const efx_nic_cfg_t *encp;
        int rc, sn;

        if (rw != KSTAT_READ) {
                rc = EACCES;
                goto fail1;
        }

        ASSERT(mutex_owned(&(smp->sm_lock)));

        if (smp->sm_state != SFXGE_MAC_STARTED)
                goto done;

        /* Synchronize the DMA memory for reading */
        (void) ddi_dma_sync(spp->sp_mem.esm_dma_handle,
            0,
            EFX_PHY_STATS_SIZE,
            DDI_DMA_SYNC_FORKERNEL);

        if ((rc = efx_phy_stats_update(enp, &spp->sp_mem, spp->sp_statbuf))
            != 0)
                goto fail2;

        knp = spp->sp_stat;
        for (sn = 0; sn < EFX_PHY_NSTATS; sn++) {
                knp->value.ui64 = spp->sp_statbuf[sn];
```

```
128                 knp++;
129         }
130
131         encp = efx_nic_cfg_get(enp);
132         knp->value.ui64 = encp->enc_port;
133
134 done:
135         return (0);
136
137 fail2:
138         DTRACE_PROBE(fail2);
139 fail1:
140         DTRACE_PROBE1(fail1, int, rc);
141
142         return (rc);
143 }
144
145 int
146 sfxge_phy_kstat_init(sfxge_t *sp)
147 {
148         dev_info_t *dip = sp->s_dip;
149         sfxge_phy_t *spp = &(sp->s_mac.sm_phy);
150         efx_nic_t *enp = sp->s_enp;
151         kstat_t *ksp;
152         kstat_named_t *knp;
153         const efx_nic_cfg_t *encp;
154         unsigned int id;
155         char name[MAXNAMELEN];
156         int rc;
157
158         if ((spp->sp_statbuf = kmem_zalloc(sizeof (uint32_t) * EFX_PHY_NSTATS,
159             KM_NOSLEEP)) == NULL) {
160                 rc = ENOMEM;
161                 goto fail1;
162         }
163
164         encp = efx_nic_cfg_get(enp);
165
166         (void) snprintf(name, MAXNAMELEN - 1, "%s_%s", ddi_driver_name(dip),
167             encp->enc_phy_name);
168
169         /* Create the set */
170         if ((ksp = kstat_create((char *)ddi_driver_name(dip),
171             ddi_get_instance(dip), name, "phy", KSTAT_TYPE_NAMED,
172             EFX_PHY_NSTATS + 1, 0)) == NULL) {
173                 rc = ENOMEM;
174                 goto fail2;
175         }
176
177         spp->sp_ksp = ksp;
178
179         ksp->ks_update = sfxge_phy_kstat_update;
180         ksp->ks_private = sp;
181         ksp->ks_lock = &(sp->s_mac.sm_lock);
182
183         /* Initialise the named stats */
184         spp->sp_stat = knp = ksp->ks_data;
185         for (id = 0; id < EFX_PHY_NSTATS; id++) {
186                 kstat_named_init(knp, (char *)efx_phy_stat_name(enp, id),
187                     KSTAT_DATA_UINT64);
188                 knp++;
189         }
190
191         kstat_named_init(knp, "port", KSTAT_DATA_UINT64);
192         kstat_install(ksp);
```



```
194         return (0);
195
196 fail2:
197         DTRACE_PROBE(fail2)
198         kmem_free(spp->sp_statbuf, sizeof (uint32_t) * EFX_PHY_NSTATS);
199
200 fail1:
201         DTRACE_PROBE1(fail1, int, rc);
202
203         return (rc);
204 }
205
206 void
207 sfxge_phy_kstat_fini(sfxge_t *sp)
208 {
209         sfxge_phy_t *spp = &(sp->s_mac.sm_phy);
210
211         /* Destroy the set */
212         kstat_delete(spp->sp_ksp);
213         spp->sp_ksp = NULL;
214         spp->sp_stat = NULL;
215
216         kmem_free(spp->sp_statbuf, sizeof (uint32_t) * EFX_PHY_NSTATS);
217 }
218
219
220 int
221 sfxge_phy_init(sfxge_t *sp)
222 {
223         sfxge_phy_t *spp = &(sp->s_mac.sm_phy);
224         efsys_mem_t *esmp = &(spp->sp_mem);
225         sfxge_dma_buffer_attr_t dma_attr;
226         int rc;
227
228         dma_attr.sdba_dip        = sp->s_dip;
229         dma_attr.sdba_dattrp     = &sfxge_phy_dma_attr;
230         dma_attr.sdba_callback   = DDI_DMA_SLEEP;
231         dma_attr.sdba_length     = EFX_PHY_STATS_SIZE;
232         dma_attr.sdba_memflags   = DDI_DMA_CONSISTENT;
233         dma_attr.sdba_devaccp    = &sfxge_phy_devacc;
234         dma_attr.sdba_bindflags  = DDI_DMA_READ | DDI_DMA_CONSISTENT;
235         dma_attr.sdba_maxcookies = 1;
236         dma_attr.sdba_zeroinit   = B_TRUE;
237
238         if ((rc = sfxge_dma_buffer_create(esmp, &dma_attr)) != 0)
239                 goto fail1;
240
241         return (0);
242
243 fail1:
244         DTRACE_PROBE1(fail1, int, rc);
245         SFXGE_OBJ_CHECK(spp, sfxge_phy_t);
246
247         return (rc);
248 }
249
250 static int
251 sfxge_phy_led_mode_set(sfxge_t *sp, efx_phy_led_mode_t mode)
252 {
253         sfxge_mac_t *smp = &(sp->s_mac);
254         int rc;
255
256         ASSERT3U(smp->sm_state, ==, SFXGE_MAC_STARTED);
257
258         if (mode >= EFX_PHY_LED_NMODES) {
259                 rc = EINVAL;
```

```
260                 goto fail1;
261         }

263         mutex_enter(&(smp->sm_lock));

265         if ((rc = efx_phy_led_set(sp->s_enp, mode)) != 0)
266                 goto fail2;

268         mutex_exit(&(smp->sm_lock));

270         return (0);

272 fail2:
273         DTRACE_PROBE(fail2);

275         mutex_exit(&(smp->sm_lock));

277 fail1:
278         DTRACE_PROBE1(fail1, int, rc);

280         return (rc);
281 }


284 int
285 sfxge_phy_ioctl(sfxge_t *sp, sfxge_phy_ioc_t *spip)
286 {
287         int rc;

289         switch (spip->spi_op) {
290         case SFXGE_PHY_OP_LINK: {
291                 break;
292         }
293         case SFXGE_PHY_OP_LED: {
294                 efx_phy_led_mode_t mode = spip->spi_data;

296                 if ((rc = sfxge_phy_led_mode_set(sp, mode)) != 0)
297                         goto fail1;

299                 break;
300         }
301         default:
302                 rc = ENOTSUP;
303                 goto fail1;
304         }

306         return (0);

308 fail1:
309         DTRACE_PROBE1(fail1, int, rc);

311         return (rc);
312 }


315 static uint8_t
316 bist_status(unsigned long value)
317 {
318         uint8_t ret;

320         efx_phy_cable_status_t status = (efx_phy_cable_status_t)value;
321         switch (status) {
322         case EFX_PHY_CABLE_STATUS_OK:
323                 ret = SFXGE_PHY_BIST_CABLE_OK;
324                 break;
325         case EFX_PHY_CABLE_STATUS_INVALID:
```



```
326                 ret =  SFXGE_PHY_BIST_CABLE_INVALID;
327                 break;
328         case EFX_PHY_CABLE_STATUS_OPEN:
329                 ret = SFXGE_PHY_BIST_CABLE_OPEN;
330                 break;
331         case EFX_PHY_CABLE_STATUS_INTRAPAIRSHORT:
332                 ret = SFXGE_PHY_BIST_CABLE_INTRAPAIRSHORT;
333                 break;
334         case EFX_PHY_CABLE_STATUS_INTERPAIRSHORT:
335                 ret = SFXGE_PHY_BIST_CABLE_INTERPAIRSHORT;
336                 break;
337         case EFX_PHY_CABLE_STATUS_BUSY:
338                 ret = SFXGE_PHY_BIST_CABLE_BUSY;
339                 break;
340         default:
341                 ret = SFXGE_PHY_BIST_CABLE_UNKNOWN;
342         }

344         return (ret);
345 }


348 int
349 sfxge_phy_bist_ioctl(sfxge_t *sp, sfxge_phy_bist_ioc_t *spbip)
350 {
351         sfxge_mac_t *smp = &(sp->s_mac);
352         efx_phy_bist_type_t bist_type;
353         efx_nic_t *enp;
354         const efx_nic_cfg_t *encp;
355         efx_phy_bist_result_t result;
356         unsigned long values[EFX_PHY_BIST_NVALUES];
357         uint32_t mask;
358         int rc;

360         mutex_enter(&(smp->sm_lock));
361         enp = sp->s_enp;
362         encp = efx_nic_cfg_get(enp);

364         bist_type = spbip->spbi_break_link ? EFX_PHY_BIST_TYPE_CABLE_LONG :
365             EFX_PHY_BIST_TYPE_CABLE_SHORT;

367         if (~encp->enc_bist_mask & (1 << bist_type)) {
368                 rc = ENOTSUP;
369                 goto fail1;
370         }

372         if ((rc = efx_phy_bist_start(enp, bist_type)) != 0)
373                 goto fail2;

375         do {
376                 /* Spin for 1 ms */
377                 drv_usecwait(1000);

379                 if ((rc = efx_phy_bist_poll(enp, bist_type, &result, &mask,
380                     values, sizeof (values) / sizeof (values[0]))) != 0)
381                         goto fail3;

383                 ASSERT3U(result, !=, EFX_PHY_BIST_RESULT_UNKNOWN);

385         } while (result == EFX_PHY_BIST_RESULT_RUNNING);

387         ASSERT3U(mask, ==, ((1 << EFX_PHY_BIST_CABLE_LENGTH_A) |
388                             (1 << EFX_PHY_BIST_CABLE_LENGTH_B) |
389                             (1 << EFX_PHY_BIST_CABLE_LENGTH_C) |
390                             (1 << EFX_PHY_BIST_CABLE_LENGTH_D) |
391                             (1 << EFX_PHY_BIST_CABLE_STATUS_A) |
```

```
392                             (1 << EFX_PHY_BIST_CABLE_STATUS_B) |
393                             (1 << EFX_PHY_BIST_CABLE_STATUS_C) |
394                             (1 << EFX_PHY_BIST_CABLE_STATUS_D)));

396         spbip->spbi_status_a = bist_status(values[EFX_PHY_BIST_CABLE_STATUS_A]);
397         spbip->spbi_status_b = bist_status(values[EFX_PHY_BIST_CABLE_STATUS_B]);
398         spbip->spbi_status_c = bist_status(values[EFX_PHY_BIST_CABLE_STATUS_C]);
399         spbip->spbi_status_d = bist_status(values[EFX_PHY_BIST_CABLE_STATUS_D]);

401         spbip->spbi_length_ind_a =
402             (uint16_t)values[EFX_PHY_BIST_CABLE_LENGTH_A];
403         spbip->spbi_length_ind_b =
404             (uint16_t)values[EFX_PHY_BIST_CABLE_LENGTH_B];
405         spbip->spbi_length_ind_c =
406             (uint16_t)values[EFX_PHY_BIST_CABLE_LENGTH_C];
407         spbip->spbi_length_ind_d =
408             (uint16_t)values[EFX_PHY_BIST_CABLE_LENGTH_D];

410         /* Bring the PHY back to life */
411         efx_phy_bist_stop(enp, bist_type);

413         mutex_exit(&(smp->sm_lock));

415         return (0);

417 fail3:
418         DTRACE_PROBE(fail3);
419         efx_phy_bist_stop(enp, bist_type);
420 fail2:
421         DTRACE_PROBE(fail2);
422 fail1:
423         mutex_exit(&(smp->sm_lock));
424         DTRACE_PROBE1(fail1, int, rc);

426         return (rc);
427 }


430 uint8_t
431 sfxge_phy_lp_cap_test(sfxge_t *sp, uint32_t field)
432 {
433         sfxge_mac_t *smp = &(sp->s_mac);
434         uint32_t cap = 0;

436         mutex_enter(&(smp->sm_lock));

438         if (smp->sm_state != SFXGE_MAC_STARTED)
439                 goto done;

441         efx_phy_lp_cap_get(sp->s_enp, &cap);

443 done:
444         mutex_exit(&(smp->sm_lock));

446         return (cap & (1 << field));
447 }

449 /*
450  * Set up the advertised capabilities that may have been asked for
451  * when the mac was not in the state SFXGE_MAC_STARTED.
452  * Must be called after efx_port_init().
453  */
454 int
455 sfxge_phy_cap_apply(sfxge_t *sp, boolean_t use_default)
456 {
457         sfxge_mac_t *smp = &(sp->s_mac);
```



```
458         efx_nic_t *enp;
459         uint32_t adv_cap;
460         int rc;
461         int err;

463         ASSERT(mutex_owned(&(smp->sm_lock)));

465         enp = sp->s_enp;

467         if (use_default)
468                 efx_phy_adv_cap_get(enp, EFX_PHY_CAP_DEFAULT, &adv_cap);
469         else
470                 efx_phy_adv_cap_get(enp, EFX_PHY_CAP_CURRENT, &adv_cap);

472         adv_cap |= smp->sm_phy_cap_to_set;
473         smp->sm_phy_cap_to_set = 0;
474         adv_cap &= ~(smp->sm_phy_cap_to_unset);
475         smp->sm_phy_cap_to_unset = 0;
476         if ((err = efx_phy_adv_cap_set(enp, adv_cap)) != 0) {
477                 if (err == EINVAL) {
478                         /*
479                          * The configuation wasn't accepted, so set to
480                          * defaults.
481                          */
482                         uint32_t requested = adv_cap;
483                         uint32_t supported;
484                         efx_phy_adv_cap_get(enp, EFX_PHY_CAP_PERM, &supported);
485                         efx_phy_adv_cap_get(enp, EFX_PHY_CAP_DEFAULT, &adv_cap);
486                         if ((rc = efx_phy_adv_cap_set(enp, adv_cap)) != 0)
487                                 goto fail1;
488                         cmn_err(CE_WARN, SFXGE_CMN_ERR
489                             "[%s%d] Setting of advertised link "
490                             "capabilities failed. "
491                             "Using default settings. "
492                             "(Requested 0x%x Given 0x%x Supported 0x%x)",
493                             ddi_driver_name(sp->s_dip),
494                             ddi_get_instance(sp->s_dip),
495                             requested,
496                             adv_cap,
497                             supported);
498                 } else {
499                         rc = err;
500                         goto fail2;
501                 }
502         }

504         return (0);

506 fail2:
507         DTRACE_PROBE(fail2);

509 fail1:
510         DTRACE_PROBE1(fail1, int, rc);

512         return (rc);
513 }

515 uint8_t
516 sfxge_phy_cap_test(sfxge_t *sp, uint32_t flag, uint32_t field,
517         boolean_t *mutablep)
518 {
519         sfxge_mac_t *smp = &(sp->s_mac);
520         efx_nic_t *enp;
521         uint32_t cap = 0;
522         uint32_t perm = 0;
```

```
524         mutex_enter(&(smp->sm_lock));
525         enp = sp->s_enp;

527         if (smp->sm_state != SFXGE_MAC_STARTED)
528                 goto done;

530         efx_phy_adv_cap_get(enp, flag, &cap);
531         efx_phy_adv_cap_get(enp, EFX_PHY_CAP_PERM, &perm);

533 done:
534         mutex_exit(&(smp->sm_lock));

536         if (mutablep)
537                 *mutablep = (perm & (1 << field)) ? B_TRUE : B_FALSE;

539         return ((cap & (1 << field)) ? 1 : 0);
540 }


543 int
544 sfxge_phy_cap_set(sfxge_t *sp, uint32_t field, int set)
545 {
546         sfxge_mac_t *smp = &(sp->s_mac);
547         efx_nic_t *enp = sp->s_enp;
548         uint32_t cap;
549         int rc = 0;

551         mutex_enter(&(smp->sm_lock));

553         if (smp->sm_state != SFXGE_MAC_STARTED) {
554                 /* Store the request for when the mac is started */
555                 if (set)
556                         smp->sm_phy_cap_to_set |= (1 << field);
557                 else
558                         smp->sm_phy_cap_to_unset |= (1 << field);
559                 goto done;
560         }

562         efx_phy_adv_cap_get(enp, EFX_PHY_CAP_CURRENT, &cap);

564         if (set)
565                 cap |= (1 << field);
566         else
567                 cap &= ~(1 << field);

569         rc = efx_phy_adv_cap_set(enp, cap);
570 done:
571         mutex_exit(&(smp->sm_lock));

573         return (rc);
574 }


577 int
578 sfxge_phy_prop_get(sfxge_t *sp, unsigned int id, uint32_t flags, uint32_t *valp)
579 {
580         sfxge_mac_t *smp = &(sp->s_mac);
581         efx_nic_t *enp = sp->s_enp;
582         int rc = 0;

584         mutex_enter(&(smp->sm_lock));

586         if (smp->sm_state != SFXGE_MAC_STARTED)
587                 goto done;

589         rc = efx_phy_prop_get(enp, id, flags, valp);
```



```
591 done:
592         mutex_exit(&(smp->sm_lock));

594         return (rc);
595 }


598 int
599 sfxge_phy_prop_set(sfxge_t *sp, unsigned int id, uint32_t val)
600 {
601         sfxge_mac_t *smp = &(sp->s_mac);
602         efx_nic_t *enp = sp->s_enp;
603         int rc = 0;

605         mutex_enter(&(smp->sm_lock));

607         if (smp->sm_state != SFXGE_MAC_STARTED)
608                 goto done;

610         rc = efx_phy_prop_set(enp, id, val);

612 done:
613         mutex_exit(&(smp->sm_lock));

615         return (rc);
616 }


619 void
620 sfxge_phy_fini(sfxge_t *sp)
621 {
622         sfxge_phy_t *spp = &(sp->s_mac.sm_phy);
623         efsys_mem_t *esmp = &(spp->sp_mem);

625         sfxge_dma_buffer_destroy(esmp);
626 }
627 #endif /* ! codereview */
```

```
**********************************************************
   67038 Thu Aug 22 18:59:28 2013
new/usr/src/uts/common/io/sfxge/sfxge_rx.c
Merged sfxge driver
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2008-2013 Solarflare Communications Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#include <sys/types.h>
#include <sys/sysmacros.h>
#include <sys/ddi.h>
#include <sys/sunddi.h>
#include <sys/atomic.h>
#include <sys/stream.h>
#include <sys/strsun.h>
#include <sys/strsubr.h>
#include <sys/strft.h>
#include <sys/ksynch.h>
#include <sys/ethernet.h>
#include <sys/crc32.h>
#include <sys/pattr.h>
#include <sys/cpu.h>

#include <sys/ethernet.h>
#include <inet/ip.h>

#include <netinet/in.h>
#include <netinet/ip.h>
#include <netinet/tcp.h>

#include "sfxge.h"

#include "efx.h"

/* RXQ flush response timeout (in microseconds) */
#define SFXGE_RX_QFLUSH_USEC    (2000000)

/* RXQ default packet buffer preallocation (number of packet buffers) */
#define SFXGE_RX_QPREALLOC      (0)

/* Receive packet DMA attributes */
static ddi_device_acc_attr_t sfxge_rx_packet_devacc = {
```



```
        DDI_DEVICE_ATTR_V0,     /* devacc_attr_version */
        DDI_NEVERSWAP_ACC,      /* devacc_attr_endian_flags */
        DDI_STRICTORDER_ACC     /* devacc_attr_dataorder */
};

static ddi_dma_attr_t sfxge_rx_packet_dma_attr = {
        DMA_ATTR_V0,            /* dma_attr_version     */
        0,                      /* dma_attr_addr_lo     */
        0xffffffffffffffffull,  /* dma_attr_addr_hi     */
        0xffffffffffffffffull,  /* dma_attr_count_max   */
        SFXGE_CPU_CACHE_SIZE,   /* dma_attr_align       */
        0xffffffff,             /* dma_attr_burstsizes  */
        1,                      /* dma_attr_minxfer     */
        0xffffffffffffffffull,  /* dma_attr_maxxfer     */
        0xffffffffffffffffull,  /* dma_attr_seg         */
        1,                      /* dma_attr_sgllen      */
        1,                      /* dma_attr_granular    */
        0                       /* dma_attr_flags       */
};

/* Receive queue DMA attributes */
static ddi_device_acc_attr_t sfxge_rxq_devacc = {

        DDI_DEVICE_ATTR_V0,     /* devacc_attr_version */
        DDI_NEVERSWAP_ACC,      /* devacc_attr_endian_flags */
        DDI_STRICTORDER_ACC     /* devacc_attr_dataorder */
};

static ddi_dma_attr_t sfxge_rxq_dma_attr = {
        DMA_ATTR_V0,            /* dma_attr_version     */
        0,                      /* dma_attr_addr_lo     */
        0xffffffffffffffffull,  /* dma_attr_addr_hi     */
        0xffffffffffffffffull,  /* dma_attr_count_max   */
        EFX_BUF_SIZE,           /* dma_attr_align       */
        0xffffffff,             /* dma_attr_burstsizes  */
        1,                      /* dma_attr_minxfer     */
        0xffffffffffffffffull,  /* dma_attr_maxxfer     */
        0xffffffffffffffffull,  /* dma_attr_seg         */
        1,                      /* dma_attr_sgllen      */
        1,                      /* dma_attr_granular    */
        0                       /* dma_attr_flags       */
};

/* Forward declaration */
static int
sfxge_rx_qpreallocate(sfxge_rxq_t *srp, int nprealloc);

static int
sfxge_rx_packet_ctor(void *buf, void *arg, int kmflags)
{
        sfxge_rx_packet_t *srpp = buf;
        sfxge_t *sp = arg;
        dev_info_t *dip = sp->s_dip;
        int err;

        ASSERT3U(sizeof (srpp->__srp_u1.__srp_s1), <=,
            sizeof (srpp->__srp_u1.__srp_pad));
        ASSERT3U(sizeof (srpp->__srp_u2.__srp_s2), <=,
            sizeof (srpp->__srp_u2.__srp_pad));

        bzero(buf, sizeof (sfxge_rx_packet_t));

        /* Allocate a DMA handle */
        err = ddi_dma_alloc_handle(dip, &sfxge_rx_packet_dma_attr,
            (kmflags == KM_SLEEP) ? DDI_DMA_SLEEP : DDI_DMA_DONTWAIT,
            NULL, &(srpp->srp_dma_handle));
```

```
128         if (err != DDI_SUCCESS)
129                 goto fail1;

131         return (0);

133 fail1:
134         DTRACE_PROBE1(fail1, int, err);

136         SFXGE_OBJ_CHECK(srpp, sfxge_rx_packet_t);

138         return (-1);
139 }

141 static void
142 sfxge_rx_packet_dtor(void *buf, void *arg)
143 {
144         sfxge_rx_packet_t *srpp = buf;

146         _NOTE(ARGUNUSED(arg))

148         /* Free the DMA handle */
149         ddi_dma_free_handle(&(srpp->srp_dma_handle));
150         srpp->srp_dma_handle = NULL;

152         SFXGE_OBJ_CHECK(srpp, sfxge_rx_packet_t);
153 }

155 static int
156 sfxge_rx_qctor(void *buf, void *arg, int kmflags)
157 {
158         sfxge_rxq_t *srp = buf;
159         efsys_mem_t *esmp = &(srp->sr_mem);
160         sfxge_t *sp = arg;
161         sfxge_dma_buffer_attr_t dma_attr;
162         sfxge_rx_fpp_t *srfppp;
163         int nprealloc;
164         unsigned int id;
165         int rc;

167         /* Compile-time structure layout checks */
168         EFX_STATIC_ASSERT(sizeof (srp->__sr_u1.__sr_s1) <=
169             sizeof (srp->__sr_u1.__sr_pad));
170         EFX_STATIC_ASSERT(sizeof (srp->__sr_u2.__sr_s2) <=
171             sizeof (srp->__sr_u2.__sr_pad));
172         EFX_STATIC_ASSERT(sizeof (srp->__sr_u3.__sr_s3) <=
173             sizeof (srp->__sr_u3.__sr_pad));

175         bzero(buf, sizeof (sfxge_rxq_t));

177         srp->sr_sp = sp;

179         dma_attr.sdba_dip        = sp->s_dip;
180         dma_attr.sdba_dattrp     = &sfxge_rxq_dma_attr;
181         dma_attr.sdba_callback   = DDI_DMA_SLEEP;
182         dma_attr.sdba_length     = EFX_RXQ_SIZE(sp->s_rxq_size);
183         dma_attr.sdba_memflags   = DDI_DMA_CONSISTENT;
184         dma_attr.sdba_devaccp    = &sfxge_rxq_devacc;
185         dma_attr.sdba_bindflags  = DDI_DMA_READ | DDI_DMA_CONSISTENT;
186         dma_attr.sdba_maxcookies = 1;
187         dma_attr.sdba_zeroinit   = B_FALSE;

189         if ((rc = sfxge_dma_buffer_create(esmp, &dma_attr)) != 0)
190                 goto fail1;

192         /* Allocate some buffer table entries */
193         if ((rc = sfxge_sram_buf_tbl_alloc(sp, EFX_RXQ_NBUFS(sp->s_rxq_size),
```



```
194             &(srp->sr_id))) != 0)
195                 goto fail2;

197         /* Allocate the context array */
198         if ((srp->sr_srpp = kmem_zalloc(sizeof (sfxge_rx_packet_t *) *
199             sp->s_rxq_size, kmflags)) == NULL) {
200                 rc = ENOMEM;
201                 goto fail3;
202         }

204         /* Allocate the flow table */
205         if ((srp->sr_flow = kmem_zalloc(sizeof (sfxge_rx_flow_t) *
206             SFXGE_MAX_FLOW, kmflags)) == NULL) {
207                 rc = ENOMEM;
208                 goto fail4;
209         }

211         srp->sr_srfpp = &(srp->sr_srfp);
212         srp->sr_rto = drv_usectohz(200000);

214         srp->sr_mpp = &(srp->sr_mp);

216         /* Initialize the free packet pool */
217         srfppp = &(srp->sr_fpp);
218         if ((srfppp->srfpp_putp = kmem_zalloc(SFXGE_CPU_CACHE_SIZE *
219                 SFXGE_RX_FPP_NSLOTS, kmflags)) == NULL) {
220                 rc = ENOMEM;
221                 goto fail5;
222         }
223         for (id = 0; id < SFXGE_RX_FPP_NSLOTS; id++) {
224                 sfxge_rx_fpp_putlist_t *putp;
225                 size_t off;

227                 off = id * SFXGE_CPU_CACHE_SIZE;
228                 putp = (void *)(srfppp->srfpp_putp + off);

230                 putp->srfpl_putp = NULL;
231                 putp->srfpl_putpp = &(putp->srfpl_putp);
232                 mutex_init(&(putp->srfpl_lock), NULL, MUTEX_DRIVER,
233                     DDI_INTR_PRI(sp->s_intr.si_intr_pri));
234         }

236         cv_init(&(srp->sr_flush_kv), NULL, CV_DRIVER, NULL);

238         /* Preallocate some packets on the free packet pool */
239         nprealloc = ddi_prop_get_int(DDI_DEV_T_ANY, sp->s_dip,
240             DDI_PROP_DONTPASS, "rx_prealloc_pkt_buffers", SFXGE_RX_QPREALLOC);
241         sfxge_rx_qpreallocate(srp, nprealloc);


244         return (0);

246 fail5:
247         DTRACE_PROBE(fail5);

249         srp->sr_mpp = NULL;

251         srp->sr_rto = 0;
252         srp->sr_srfpp = NULL;

254         /* Free the flow table */
255         kmem_free(srp->sr_flow, sizeof (sfxge_rx_flow_t) *
256             SFXGE_MAX_FLOW);
257         srp->sr_flow = NULL;

259 fail4:
```

```
260         DTRACE_PROBE(fail4);

262         /* Free the context array */
263         kmem_free(srp->sr_srpp, sizeof (sfxge_rx_packet_t *) *
264             sp->s_rxq_size);
265         srp->sr_srpp = NULL;

267 fail3:
268         DTRACE_PROBE(fail3);

270         /* Free the buffer table entries */
271         sfxge_sram_buf_tbl_free(sp, srp->sr_id,
272             EFX_RXQ_NBUFS(sp->s_rxq_size));
273         srp->sr_id = 0;

275 fail2:
276         DTRACE_PROBE(fail2);
277         /* Remove dma setup */
278         sfxge_dma_buffer_destroy(esmp);

280 fail1:
281         DTRACE_PROBE1(fail1, int, rc);

283         srp->sr_sp = NULL;

285         SFXGE_OBJ_CHECK(srp, sfxge_rxq_t);

287         return (-1);
288 }

290 static void
291 sfxge_rx_qdtor(void *buf, void *arg)
292 {
293         sfxge_rxq_t *srp = buf;
294         efsys_mem_t *esmp = &(srp->sr_mem);
295         sfxge_t *sp = srp->sr_sp;
296         sfxge_rx_fpp_t *srfppp = &(srp->sr_fpp);
297         unsigned int id;

299         _NOTE(ARGUNUSED(arg))

301         cv_destroy(&(srp->sr_flush_kv));

303         /* Tear down the free packet pool */
304         for (id = 0; id < SFXGE_RX_FPP_NSLOTS; id++) {
305                 sfxge_rx_fpp_putlist_t *putp;
306                 size_t off;

308                 off = id * SFXGE_CPU_CACHE_SIZE;
309                 putp = (void *)(srfppp->srfpp_putp + off);

311                 putp->srfpl_putpp = NULL;
312                 mutex_destroy(&(putp->srfpl_lock));

314                 SFXGE_OBJ_CHECK(putp, sfxge_rx_fpp_putlist_t);
315         }
316         kmem_free(srfppp->srfpp_putp, SFXGE_CPU_CACHE_SIZE *
317             SFXGE_RX_FPP_NSLOTS);
318         srfppp->srfpp_putp = NULL;

320         srp->sr_mpp = NULL;

322         srp->sr_rto = 0;
323         srp->sr_srfpp = NULL;

325         /* Free the flow table */
```



```
326         kmem_free(srp->sr_flow, sizeof (sfxge_rx_flow_t) *
327             SFXGE_MAX_FLOW);
328         srp->sr_flow = NULL;

330         /* Free the context array */
331         kmem_free(srp->sr_srpp, sizeof (sfxge_rx_packet_t *) *
332             sp->s_rxq_size);
333         srp->sr_srpp = NULL;

335         /* Free the buffer table entries */
336         sfxge_sram_buf_tbl_free(sp, srp->sr_id,
337             EFX_RXQ_NBUFS(sp->s_rxq_size));
338         srp->sr_id = 0;

340         /* Tear down dma setup */
341         sfxge_dma_buffer_destroy(esmp);

343         SFXGE_OBJ_CHECK(srp, sfxge_rxq_t);
344 }

346 /* Note: This function takes ownership of *srpp. */
347 static inline void
348 sfxge_rx_qfpp_put(sfxge_rxq_t *srp, sfxge_rx_packet_t *srpp)
349 {
350         sfxge_rx_fpp_t *srfppp = &(srp->sr_fpp);
351         mblk_t *mp = srpp->srp_mp;
352         unsigned int id;
353         size_t off;
354         sfxge_rx_fpp_putlist_t *putp;

356         ASSERT3P(mp->b_next, ==, NULL);
357         ASSERT3P(mp->b_prev, ==, NULL);

359         id = CPU->cpu_seqid & SFXGE_RX_FPP_MASK;
360         off = id * SFXGE_CPU_CACHE_SIZE;

362         ASSERT3P(srpp->srp_putp, ==, srfppp->srfpp_putp);
363         putp = (void *)(srpp->srp_putp + off);

365         mutex_enter(&(putp->srfpl_lock));
366         putp->srfpl_count++;
367         *putp->srfpl_putpp = mp;
368         putp->srfpl_putpp = &(mp->b_next);
369         mutex_exit(&(putp->srfpl_lock));
370 }

372 static unsigned int
373 sfxge_rx_qfpp_swizzle(sfxge_rxq_t *srp)
374 {
375         sfxge_t *sp = srp->sr_sp;
376         unsigned int index = srp->sr_index;
377         sfxge_evq_t *sep = sp->s_sep[index];
378         sfxge_rx_fpp_t *srfppp = &(srp->sr_fpp);
379         unsigned int start;
380         unsigned int id;
381         mblk_t *p;
382         mblk_t **pp;
383         unsigned int count;
384         unsigned int loaned;

386         ASSERT(mutex_owned(&(sep->se_lock)));

388         /* We want to access the put list for the current CPU last */
389         id = start = (CPU->cpu_seqid + 1) & SFXGE_RX_FPP_MASK;

391         do {
```

```
392                 sfxge_rx_fpp_putlist_t *putp;
393                 size_t off;

395                 off = id * SFXGE_CPU_CACHE_SIZE;
396                 id  = (id + 1) & SFXGE_RX_FPP_MASK;

398                 putp = (void *)(srfppp->srfpp_putp + off);

400                 /* Acquire the put list */
401                 mutex_enter(&(putp->srfpl_lock));

403                 p = putp->srfpl_putp;
404                 pp = putp->srfpl_putpp;
405                 count = putp->srfpl_count;

407                 putp->srfpl_putp = NULL;
408                 putp->srfpl_putpp = &(putp->srfpl_putp);
409                 putp->srfpl_count = 0;

411                 mutex_exit(&(putp->srfpl_lock));

413                 if (p == NULL)
414                         continue;

416                 /* Add the list to the head of the get list */
417                 *pp = srfppp->srfpp_get;
418                 srfppp->srfpp_get = p;

420                 /* Adjust the counters */
421                 ASSERT3U(srfppp->srfpp_loaned, >=, count);
422                 srfppp->srfpp_loaned -= count;
423                 srfppp->srfpp_count += count;

425 #if 0
426                 /* NOTE: this probe is disabled because it is expensive!! */
427                 DTRACE_PROBE2(count,
428                     unsigned int, (id - 1) & SFXGE_RX_FPP_MASK,
429                     unsigned int, count);
430 #endif

432         } while (id != start);

434         /* Return the number of packets yet to appear in the put list */
435         loaned = srfppp->srfpp_loaned;


438         return (loaned);
439 }


442 #define DB_FRTNP(mp)    ((mp)->b_datap->db_frtnp)

444 static void
445 sfxge_rx_qfpp_empty(sfxge_rxq_t *srp)
446 {
447         sfxge_t *sp = srp->sr_sp;
448         unsigned int index = srp->sr_index;
449         sfxge_evq_t *sep = sp->s_sep[index];
450         sfxge_rx_fpp_t *srfppp;
451         mblk_t *mp;

453         mutex_enter(&(sep->se_lock));
454         srfppp = &(srp->sr_fpp);

456         /* Swizzle put list to get list */
457         (void) sfxge_rx_qfpp_swizzle(srp);
```



```
458         ASSERT3U(srfppp->srfpp_loaned, ==, 0);

460         mp = srfppp->srfpp_get;
461         srfppp->srfpp_get = NULL;

463         /* Free the remainder */
464         while (mp != NULL) {
465                 mblk_t *next;
466                 frtn_t *freep;
467                 sfxge_rx_packet_t *srpp;

469                 next = mp->b_next;
470                 mp->b_next = NULL;

472                 ASSERT3U(srfppp->srfpp_count, >, 0);
473                 srfppp->srfpp_count--;

475                 freep = DB_FRTNP(mp);
476                 /*
477                  * ASSERT3P(freep->free_func, ==, sfxge_rx_qpacket_free);
478                  *   is implied by srpp test below
479                  */
480                 /*LINTED*/
481                 srpp = (sfxge_rx_packet_t *)(freep->free_arg);
482                 ASSERT3P(srpp->srp_mp, ==, mp);
483                 ASSERT3P(mp->b_cont, ==, NULL);
484                 srpp->srp_recycle = B_FALSE;

486                 freeb(mp);

488                 mp = next;
489         }
490         ASSERT3U(srfppp->srfpp_count, ==, 0);

492         srfppp->srfpp_min = 0;

494         mutex_exit(&(sep->se_lock));
495 }

497 /*
498  * This is an estimate of all memory consumed per RX packet
499  * it can be inaccurate but but sp->s_rx_pkt_mem_alloc mustn't drift
500  */
501 static uint64_t
502 sfxge_rx_pkt_mem_approx(const sfxge_rx_packet_t *srpp)
503 {
504         return (srpp->srp_mblksize + sizeof (mblk_t) + sizeof (dblk_t) +
505             sizeof (sfxge_rx_packet_t));
506 }

508 static void
509 sfxge_rx_qpacket_destroy(sfxge_rxq_t *srp, sfxge_rx_packet_t *srpp)
510 {
511         sfxge_t *sp = srp->sr_sp;
512         int64_t delta = sfxge_rx_pkt_mem_approx(srpp);

514         ASSERT(!(srpp->srp_recycle));
515         ASSERT3P(srpp->srp_mp, ==, NULL);

517         srpp->srp_off = 0;
518         srpp->srp_thp = NULL;
519         srpp->srp_iphp = NULL;
520         srpp->srp_etherhp = NULL;
521         srpp->srp_size = 0;
522         srpp->srp_flags = 0;
```

```
524         bzero(&(srpp->srp_free), sizeof (frtn_t));

526         srpp->srp_mblksize = 0;
527         srpp->srp_base = NULL;

529         /* Unbind the DMA memory from the DMA handle */
530         srpp->srp_addr = 0;
531         (void) ddi_dma_unbind_handle(srpp->srp_dma_handle);

533         /* Free the DMA memory */
534         srpp->srp_base = NULL;
535         ddi_dma_mem_free(&(srpp->srp_acc_handle));
536         srpp->srp_acc_handle = NULL;

538         srpp->srp_putp = NULL;
539         srpp->srp_srp = NULL;

541         kmem_cache_free(sp->s_rpc, srpp);
542         if (sp->s_rx_pkt_mem_max)
543                 atomic_add_64(&sp->s_rx_pkt_mem_alloc, -delta);
544 }

546 #ifdef _USE_XESBALLOC
547 static void
548 sfxge_rx_qpacket_free(void *arg, mblk_t *mp, boolean_t *recyclep)
549 {
550         sfxge_rx_packet_t *srpp = arg;
551         sfxge_rxq_t *srp = srpp->srp_srp;

553         /*
554          * WARNING "man -s 9f esballoc" states:
555          * => runs async in a background context
556          * => must not sleep, or access data structures that could be freed
557          */
558         ASSERT3P(DB_BASE(mp), ==, srpp->srp_base);
559         ASSERT3P(MBLKSIZE(mp), ==, srpp->srp_mblksize);

561         /* Check whether we want to recycle the receive packets */
562         if (srpp->srp_recycle) {
563                 ASSERT3P(DB_FRTNP(mp), ==, &(srpp->srp_free));

565                 srpp->srp_mp = mp;

567                 /* NORMAL recycled case */
568                 sfxge_rx_qfpp_put(srp, srpp);
569                 *recyclep = B_TRUE;
570                 return;
571         }

573         srpp->srp_mp = NULL;

575         sfxge_rx_qpacket_destroy(srp, srpp);
576         *recyclep = B_FALSE;
577 }
578 #endif  /* _USE_XESBALLOC */

580 #ifdef _USE_DESBALLOC
581 static void
582 sfxge_rx_qpacket_free(void *arg)
583 {
584         sfxge_rx_packet_t *srpp = arg;
585         sfxge_rxq_t *srp = srpp->srp_srp;

587         /*
588          * WARNING "man -s 9f esballoc"  states:
589          * => runs sync from the thread calling freeb()
```



```
590          * => must not sleep, or access data structures that could be freed
591          */

593         /* Check whether we want to recycle the receive packets */
594         if (srpp->srp_recycle) {
595                 frtn_t *freep;
596                 mblk_t *mp;
597                 size_t size;

599                 freep = &(srpp->srp_free);
600                 ASSERT3P(freep->free_func, ==, sfxge_rx_qpacket_free);
601                 ASSERT3P(freep->free_arg, ==, (caddr_t)srpp);

603                 /*
604                  * Allocate a matching mblk_t before the current one is
605                  * freed.
606                  */
607                 size = srpp->srp_mblksize;

609                 if ((mp = desballoc(srpp->srp_base, size, BPRI_HI,
610                     freep)) != NULL) {
611                         srpp->srp_mp = mp;

613                         /* NORMAL recycled case */
614                         sfxge_rx_qfpp_put(srp, srpp);
615                         return;
616                 }
617         }

619         srpp->srp_mp = NULL;

621         sfxge_rx_qpacket_destroy(srp, srpp);
622 }
623 #endif  /* _USE_DESBALLOC */

625 static sfxge_rx_packet_t *
626 sfxge_rx_qpacket_create(sfxge_rxq_t *srp)
627 {
628         sfxge_t *sp = srp->sr_sp;
629         sfxge_rx_fpp_t *srfppp = &(srp->sr_fpp);
630         sfxge_rx_packet_t *srpp;
631         size_t size;
632         caddr_t base;
633         size_t unit;
634         ddi_dma_cookie_t dmac;
635         unsigned int ncookies;
636         frtn_t *freep;
637         mblk_t *mp;
638         int err;
639         int rc;

641         size = sp->s_rx_buffer_size;

643         if (sp->s_rx_pkt_mem_max &&
644             (sp->s_rx_pkt_mem_alloc + size >= sp->s_rx_pkt_mem_max)) {
645                 DTRACE_PROBE(rx_pkt_mem_max);
646                 srp->sr_kstat.srk_rx_pkt_mem_limit++;
647                 return (NULL);
648         }

650         /* Allocate a new packet */
651         if ((srpp = kmem_cache_alloc(sp->s_rpc, KM_NOSLEEP)) == NULL) {
652                 srp->sr_kstat.srk_kcache_alloc_nomem++;
653                 rc = ENOMEM;
654                 goto fail1;
655         }
```

```
657         srpp->srp_srp = srp;
658         srpp->srp_putp = srfppp->srfpp_putp;
659 
660         /* Allocate some DMA memory */
661         err = ddi_dma_mem_alloc(srpp->srp_dma_handle, size,
662             &sfxge_rx_packet_devacc, DDI_DMA_STREAMING, DDI_DMA_DONTWAIT,
663             NULL, &base, &unit, &(srpp->srp_acc_handle));
664         switch (err) {
665         case DDI_SUCCESS:
666                 break;
667 
668         case DDI_FAILURE:
669                 srp->sr_kstat.srk_dma_alloc_nomem++;
670                 rc = ENOMEM;
671                 goto fail2;
672 
673         default:
674                 srp->sr_kstat.srk_dma_alloc_fail++;
675                 rc = EFAULT;
676                 goto fail2;
677         }
678 
679         /* Adjust the buffer to align the start of the DMA area correctly */
680         base += sp->s_rx_buffer_align;
681         size -= sp->s_rx_buffer_align;
682 
683         /* Bind the DMA memory to the DMA handle */
684         err = ddi_dma_addr_bind_handle(srpp->srp_dma_handle, NULL,
685             base, size, DDI_DMA_READ | DDI_DMA_STREAMING,
686             DDI_DMA_DONTWAIT, NULL, &dmac, &ncookies);
687         switch (err) {
688         case DDI_DMA_MAPPED:
689                 break;
690 
691         case DDI_DMA_INUSE:
692                 srp->sr_kstat.srk_dma_bind_fail++;
693                 rc = EEXIST;
694                 goto fail3;
695 
696         case DDI_DMA_NORESOURCES:
697                 srp->sr_kstat.srk_dma_bind_nomem++;
698                 rc = ENOMEM;
699                 goto fail3;
700 
701         case DDI_DMA_NOMAPPING:
702                 srp->sr_kstat.srk_dma_bind_fail++;
703                 rc = ENOTSUP;
704                 goto fail3;
705 
706         case DDI_DMA_TOOBIG:
707                 srp->sr_kstat.srk_dma_bind_fail++;
708                 rc = EFBIG;
709                 goto fail3;
710 
711         default:
712                 srp->sr_kstat.srk_dma_bind_fail++;
713                 rc = EFAULT;
714                 goto fail3;
715         }
716         ASSERT3U(ncookies, ==, 1);
717 
718         srpp->srp_addr = dmac.dmac_laddress;
719 
720         srpp->srp_base = (unsigned char *)base;
721         srpp->srp_mblksize = size;
```



```
722 
723         /*
724          * Allocate a STREAMS block: We use size 1 so that the allocator will
725          * use the first (and smallest) dblk cache.
726          */
727         freep = &(srpp->srp_free);
728         freep->free_func = sfxge_rx_qpacket_free;
729         freep->free_arg  = (caddr_t)srpp;
730 
731 #ifdef _USE_XESBALLOC
732         if ((mp = xesballoc(srpp->srp_base, size, BPRI_HI, freep)) == NULL) {
733                 srp->sr_kstat.srk_xesballoc_fail++;
734                 rc = ENOMEM;
735                 goto fail4;
736         }
737 #endif  /* _USE_XESBALLOC */
738 
739 #ifdef _USE_DESBALLOC
740         if ((mp = desballoc(srpp->srp_base, size, BPRI_HI, freep)) == NULL) {
741                 srp->sr_kstat.srk_desballoc_fail++;
742                 rc = ENOMEM;
743                 goto fail4;
744         }
745 #endif  /* _USE_DESBALLOC */
746 
747         srpp->srp_mp = mp;
748         srpp->srp_recycle = B_TRUE;
749 
750         if (sp->s_rx_pkt_mem_max) {
751                 int64_t delta = sfxge_rx_pkt_mem_approx(srpp);
752                 atomic_add_64(&sp->s_rx_pkt_mem_alloc, delta);
753         }
754 
755         return (srpp);
756 
757 fail4:
758         DTRACE_PROBE(fail4);
759 
760         bzero(&(srpp->srp_free), sizeof (frtn_t));
761 
762         srpp->srp_mblksize = 0;
763         srpp->srp_base = NULL;
764 
765         /* Unbind the DMA memory from the DMA handle */
766         srpp->srp_addr = 0;
767         (void) ddi_dma_unbind_handle(srpp->srp_dma_handle);
768 
769 fail3:
770         DTRACE_PROBE(fail3);
771 
772         /* Free the DMA memory */
773         ddi_dma_mem_free(&(srpp->srp_acc_handle));
774         srpp->srp_acc_handle = NULL;
775 
776 fail2:
777         DTRACE_PROBE(fail2);
778 
779         srpp->srp_putp = NULL;
780         srpp->srp_srp = NULL;
781 
782         kmem_cache_free(sp->s_rpc, srpp);
783 
784 fail1:
785         DTRACE_PROBE1(fail1, int, rc);
786 
787         return (NULL);
```

```
788 }

790 #define SFXGE_REFILL_BATCH  64

792 /* Try to refill the RX descriptor ring from the associated free pkt pool */
793 static void
794 sfxge_rx_qrefill(sfxge_rxq_t *srp, unsigned int target)
795 {
796         sfxge_t *sp = srp->sr_sp;
797         sfxge_rx_fpp_t *srfppp = &(srp->sr_fpp);
798         unsigned int index = srp->sr_index;
799         sfxge_evq_t *sep = sp->s_sep[index];
800         efsys_dma_addr_t addr[SFXGE_REFILL_BATCH];
801         mblk_t *mp;
802         int ntodo;
803         unsigned int count;
804         unsigned int batch;
805         unsigned int rxfill;
806         unsigned int mblksize;

808         prefetch_read_many(sp->s_enp);
809         prefetch_read_many(srp->sr_erp);

811         ASSERT(mutex_owned(&(sep->se_lock)));

813         if (srp->sr_state != SFXGE_RXQ_STARTED)
814                 return;

816         rxfill = srp->sr_added - srp->sr_completed;
817         ASSERT3U(rxfill, <=, EFX_RXQ_LIMIT(sp->s_rxq_size));
818         ntodo = min(EFX_RXQ_LIMIT(sp->s_rxq_size) - rxfill, target);
819         ASSERT3U(ntodo, <=, EFX_RXQ_LIMIT(sp->s_rxq_size));

821         if (ntodo == 0)
822                 goto out;

824         (void) sfxge_rx_qfpp_swizzle(srp);

826         mp = srfppp->srfpp_get;
827         count = srfppp->srfpp_count;
828         mblksize = sp->s_rx_buffer_size - sp->s_rx_buffer_align;

830         batch = 0;
831         while (ntodo-- > 0) {
832                 mblk_t *next;
833                 frtn_t *freep;
834                 sfxge_rx_packet_t *srpp;
835                 unsigned int id;

837                 if (mp == NULL)
838                         break;

840                 next = mp->b_next;
841                 mp->b_next = NULL;

843                 if (next != NULL)
844                         prefetch_read_many(next);

846                 freep = DB_FRTNP(mp);
847                 /*LINTED*/
848                 srpp = (sfxge_rx_packet_t *)(freep->free_arg);
849                 ASSERT3P(srpp->srp_mp, ==, mp);

851                 /* The MTU may have changed since the packet was allocated */
852                 if (MBLKSIZE(mp) != mblksize) {
853                         srpp->srp_recycle = B_FALSE;
```



```
855                         freeb(mp);

857                         --count;
858                         mp = next;
859                         continue;
860                 }

862                 srpp->srp_off = 0;
863                 srpp->srp_thp = NULL;
864                 srpp->srp_iphp = NULL;
865                 srpp->srp_etherhp = NULL;
866                 srpp->srp_size = 0;
867                 srpp->srp_flags = EFX_DISCARD;

869                 id = (srp->sr_added + batch) & (sp->s_rxq_size - 1);
870                 ASSERT(srp->sr_srpp[id] == NULL);
871                 srp->sr_srpp[id] = srpp;

873                 addr[batch++] = srpp->srp_addr;
874                 if (batch == SFXGE_REFILL_BATCH) {
875                         efx_rx_qpost(srp->sr_erp, addr, mblksize, batch,
876                             srp->sr_completed, srp->sr_added);
877                         srp->sr_added += batch;
878                         batch = 0;
879                 }

881                 --count;
882                 mp = next;
883         }

885         srfppp->srfpp_get = mp;
886         srfppp->srfpp_count = count;

888         if (batch != 0) {
889                 efx_rx_qpost(srp->sr_erp, addr, mblksize, batch,
890                     srp->sr_completed, srp->sr_added);
891                 srp->sr_added += batch;
892         }

894         /* Make the descriptors visible to the hardware */
895         (void) ddi_dma_sync(srp->sr_mem.esm_dma_handle,
896             0,
897             EFX_RXQ_SIZE(sp->s_rxq_size),
898             DDI_DMA_SYNC_FORDEV);

900         efx_rx_qpush(srp->sr_erp, srp->sr_added);

902 out:
903         if (srfppp->srfpp_count < srfppp->srfpp_min)
904                 srfppp->srfpp_min = srfppp->srfpp_count;
905 }

907 /* Preallocate packets and put them in the free packet pool */
908 static int
909 sfxge_rx_qpreallocate(sfxge_rxq_t *srp, int nprealloc)
910 {
911         sfxge_rx_fpp_t *srfppp = &((srp)->sr_fpp);
912         srfppp->srfpp_lowat = nprealloc;
913         while (nprealloc-- > 0) {
914                 sfxge_rx_packet_t *srpp;

916                 if ((srpp = sfxge_rx_qpacket_create(srp)) == NULL)
917                         break;
918                 sfxge_rx_qfpp_put(srp, srpp);
919         }
```

```
920         return (0);
921 }

923 /* Try to refill the RX descriptor ring by allocating new packets */
924 static void
925 sfxge_rx_qfill(sfxge_rxq_t *srp, unsigned int target)
926 {
927         sfxge_t *sp = srp->sr_sp;
928         unsigned int index = srp->sr_index;
929         sfxge_evq_t *sep = sp->s_sep[index];
930         unsigned int batch;
931         unsigned int rxfill;
932         unsigned int mblksize;
933         int ntodo;
934         efsys_dma_addr_t addr[SFXGE_REFILL_BATCH];
935         mblk_t *mp = NULL;

937         prefetch_read_many(sp->s_enp);
938         prefetch_read_many(srp->sr_erp);

940         ASSERT(mutex_owned(&(sep->se_lock)));

942         if (srp->sr_state != SFXGE_RXQ_STARTED)
943                 return;

945         rxfill = srp->sr_added - srp->sr_completed;
946         ASSERT3U(rxfill, <=, EFX_RXQ_LIMIT(sp->s_rxq_size));
947         ntodo = min(EFX_RXQ_LIMIT(sp->s_rxq_size) - rxfill, target);
948         ASSERT3U(ntodo, <=, EFX_RXQ_LIMIT(sp->s_rxq_size));

950         if (ntodo == 0)
951                 return;

953         mblksize = sp->s_rx_buffer_size - sp->s_rx_buffer_align;

955         batch = 0;
956         while (ntodo-- > 0) {
957                 sfxge_rx_packet_t *srpp;
958                 unsigned int id;

960                 if ((srpp = sfxge_rx_qpacket_create(srp)) == NULL)
961                         break;

963                 mp = srpp->srp_mp;

965                 ASSERT3U(MBLKSIZE(mp), ==, mblksize);

967                 ASSERT3U(srpp->srp_off, ==, 0);
968                 ASSERT3P(srpp->srp_thp, ==, NULL);
969                 ASSERT3P(srpp->srp_iphp, ==, NULL);
970                 ASSERT3P(srpp->srp_etherhp, ==, NULL);
971                 ASSERT3U(srpp->srp_size, ==, 0);

973                 srpp->srp_flags = EFX_DISCARD;

975                 id = (srp->sr_added + batch) & (sp->s_rxq_size - 1);
976                 ASSERT(srp->sr_srpp[id] == NULL);
977                 srp->sr_srpp[id] = srpp;

979                 addr[batch++] = srpp->srp_addr;
980                 if (batch == SFXGE_REFILL_BATCH) {
981                         efx_rx_qpost(srp->sr_erp, addr, mblksize, batch,
982                             srp->sr_completed, srp->sr_added);
983                         srp->sr_added += batch;
984                         batch = 0;
985                 }
```



```
986         }

988         if (batch != 0) {
989                 efx_rx_qpost(srp->sr_erp, addr, mblksize, batch,
990                     srp->sr_completed, srp->sr_added);
991                 srp->sr_added += batch;
992         }

994         /* Make the descriptors visible to the hardware */
995         (void) ddi_dma_sync(srp->sr_mem.esm_dma_handle,
996             0,
997             EFX_RXQ_SIZE(sp->s_rxq_size),
998             DDI_DMA_SYNC_FORDEV);

1000        efx_rx_qpush(srp->sr_erp, srp->sr_added);
1001 }

1003 void
1004 sfxge_rx_qfpp_trim(sfxge_rxq_t *srp)
1005 {
1006         sfxge_rx_fpp_t *srfppp = &(srp->sr_fpp);
1007         sfxge_t *sp = srp->sr_sp;
1008         unsigned int index = srp->sr_index;
1009         sfxge_evq_t *sep = sp->s_sep[index];
1010         mblk_t *p;
1011         mblk_t **pp;
1012         int count;

1014         ASSERT(mutex_owned(&(sep->se_lock)));

1016         if (srp->sr_state != SFXGE_RXQ_STARTED)
1017                 goto done;

1019         /* Make sure the queue is full */
1020         sfxge_rx_qrefill(srp, EFX_RXQ_LIMIT(sp->s_rxq_size));

1022         /* The refill may have emptied the pool */
1023         if (srfppp->srfpp_min == 0)
1024                 goto done;

1026         /* Don't trim below the pool's low water mark */
1027         if (srfppp->srfpp_count <= srfppp->srfpp_lowat)
1028                 goto done;

1030         ASSERT(srfppp->srfpp_min <= srfppp->srfpp_count);

1032         /* Trim to the largest of srfppp->srfpp_min and srfpp->srfpp_lowat */
1033         if (srfppp->srfpp_lowat > srfppp->srfpp_min)
1034                 count = srfppp->srfpp_count - srfppp->srfpp_lowat;
1035         else
1036                 count = srfppp->srfpp_count - srfppp->srfpp_min;

1038         /* Walk the get list */
1039         pp = &(srfppp->srfpp_get);
1040         while (--count >= 0) {
1041                 ASSERT(pp);
1042                 p = *pp;
1043                 ASSERT(p != NULL);

1045                 pp = &(p->b_next);
1046         }
1047         ASSERT(pp);
1048         p = *pp;

1050         /* Truncate the get list */
1051         *pp = NULL;
```

```
1053         /* Free the remainder */
1054         while (p != NULL) {
1055                 mblk_t *next;
1056                 frtn_t *freep;
1057                 sfxge_rx_packet_t *srpp;

1059                 next = p->b_next;
1060                 p->b_next = NULL;

1062                 ASSERT3U(srfppp->srfpp_min, >, 0);
1063                 srfppp->srfpp_min--;
1064                 srfppp->srfpp_count--;

1066                 freep = DB_FRTNP(p);
1067                 /*LINTED*/
1068                 srpp = (sfxge_rx_packet_t *)(freep->free_arg);
1069                 ASSERT3P(srpp->srp_mp, ==, p);

1071                 srpp->srp_recycle = B_FALSE;

1073                 freeb(p);

1075                 p = next;
1076         }

1078 done:
1079         srfppp->srfpp_min = srfppp->srfpp_count;
1080 }

1082 static void
1083 sfxge_rx_qpoll(void *arg)
1084 {
1085         sfxge_rxq_t *srp = arg;
1086         sfxge_t *sp = srp->sr_sp;
1087         unsigned int index = srp->sr_index;
1088         sfxge_evq_t *sep = sp->s_sep[index];
1089         uint16_t magic;

1091         /*
1092          * man timeout(9f) states that this code should adhere to the
1093          * same requirements as a softirq handler - DO NOT BLOCK
1094          */

1096         /*
1097          * Post an event to the event queue to cause the free packet pool to be
1098          * trimmed if it is oversize.
1099          */
1100         magic = SFXGE_MAGIC_RX_QFPP_TRIM | index;

1102 #if defined(DEBUG)
1103         /* This is guaranteed due to the start/stop order of rx and ev */
1104         ASSERT3U(sep->se_state, ==, SFXGE_EVQ_STARTED);
1105         ASSERT3U(srp->sr_state, ==, SFXGE_RXQ_STARTED);
1106 #else
1107         /*
1108          * Bug22691 WORKAROUND:
1109          * This handler has been observed in the field to be invoked for a
1110          * queue in the INITIALIZED state, which should never happen.
1111          * Until the mechanism for this is properly understood, add defensive
1112          * checks.
1113          */
1114         if ((sep->se_state != SFXGE_EVQ_STARTED) ||
1115             (srp->sr_state != SFXGE_RXQ_STARTED) ||
1116             (!sep->se_eep)) {
1117                 cmn_err(CE_WARN, SFXGE_CMN_ERR
```



```
1118                         "[%s%d] RXQ[%d] bad state in sfxge_rx_qpoll %d %d %p",
1119                         ddi_driver_name(sp->s_dip), ddi_get_instance(sp->s_dip),
1120                         index, sep->se_state, srp->sr_state, sep->se_eep);
1121                 return;
1122         }
1123 #endif
1124         efx_ev_qpost(sep->se_eep, magic);

1126         srp->sr_tid = timeout(sfxge_rx_qpoll, srp,
1127                 drv_usectohz(sp->s_rxq_poll_usec));
1128 }

1130 static void
1131 sfxge_rx_qpoll_start(sfxge_rxq_t *srp)
1132 {
1133         sfxge_t *sp = srp->sr_sp;
1134         unsigned int index = srp->sr_index;
1135         sfxge_evq_t *sep = sp->s_sep[index];

1137         ASSERT(mutex_owned(&(sep->se_lock)));
1138         ASSERT3U(srp->sr_state, ==, SFXGE_RXQ_STARTED);

1140         /* Schedule a poll */
1141         ASSERT3P(srp->sr_tid, ==, 0);
1142         srp->sr_tid = timeout(sfxge_rx_qpoll, srp, 0);
1143 }

1145 static void
1146 sfxge_rx_qpoll_stop(sfxge_rxq_t *srp)
1147 {
1148         sfxge_t *sp = srp->sr_sp;
1149         unsigned int index = srp->sr_index;
1150         sfxge_evq_t *sep = sp->s_sep[index];
1151         timeout_id_t tid;

1153         ASSERT(mutex_owned(&(sep->se_lock)));
1154         ASSERT3U(srp->sr_state, ==, SFXGE_RXQ_STARTED);

1156         /*
1157          * Cancel the qpoll timer. Care is needed as this function
1158          * can race with sfxge_rx_qpoll() for timeout id updates.
1159          *
1160          * Do not hold locks used by any timeout(9f) handlers across
1161          * calls to untimeout(9f) as this will deadlock.
1162          */
1163         tid = 0;
1164         while ((srp->sr_tid != 0) && (srp->sr_tid != tid)) {
1165                 tid = srp->sr_tid;
1166                 (void) untimeout(tid);
1167         }
1168         srp->sr_tid = 0;
1169 }

1171 static int
1172 sfxge_rx_kstat_update(kstat_t *ksp, int rw)
1173 {
1174         sfxge_rxq_t *srp = ksp->ks_private;
1175         sfxge_t *sp = srp->sr_sp;
1176         unsigned int index = srp->sr_index;
1177         sfxge_evq_t *sep = sp->s_sep[index];
1178         kstat_named_t *knp;
1179         int rc;

1181         if (rw != KSTAT_READ) {
1182                 rc = EACCES;
1183                 goto fail1;
```

```
1184         }

1186         ASSERT(mutex_owned(&(sep->se_lock)));
1187         if (srp->sr_state != SFXGE_RXQ_STARTED)
1188                 goto done;

1190         knp = ksp->ks_data;
1191         /* NB pointer post-increment below */
1192         knp++->value.ui32 = srp->sr_kstat.srk_rx_pkt_mem_limit;
1193         knp++->value.ui32 = srp->sr_kstat.srk_kcache_alloc_nomem;
1194         knp++->value.ui32 = srp->sr_kstat.srk_dma_alloc_nomem;
1195         knp++->value.ui32 = srp->sr_kstat.srk_dma_alloc_fail;
1196         knp++->value.ui32 = srp->sr_kstat.srk_dma_bind_nomem;
1197         knp++->value.ui32 = srp->sr_kstat.srk_dma_bind_fail;
1198 #ifdef _USE_XESBALLOC
1199         knp++->value.ui32 = srp->sr_kstat.srk_xesballoc_fail;
1200 #endif
1201 #ifdef _USE_DESBALLOC
1202         knp++->value.ui32 = srp->sr_kstat.srk_desballoc_fail;
1203 #endif
1204         knp++->value.ui32 = srp->sr_kstat.srk_rxq_empty_discard;

1206 done:
1207         return (0);

1209 fail1:
1210         DTRACE_PROBE1(fail1, int, rc);

1212         return (rc);
1213 }

1215 static int
1216 sfxge_rx_kstat_init(sfxge_rxq_t *srp)
1217 {
1218         sfxge_t *sp = srp->sr_sp;
1219         unsigned int index = srp->sr_index;
1220         sfxge_evq_t *sep = sp->s_sep[index];
1221         dev_info_t *dip = sp->s_dip;
1222         char name[MAXNAMELEN];
1223         kstat_t *ksp;
1224         kstat_named_t *knp;
1225         int rc;

1227         /* Create the set */
1228         (void) snprintf(name, MAXNAMELEN - 1, "%s_rxq%04d",
1229             ddi_driver_name(dip), index);

1231         if ((ksp = kstat_create((char *)ddi_driver_name(dip),
1232             ddi_get_instance(dip), name, "rxq", KSTAT_TYPE_NAMED,
1233             SFXGE_RX_NSTATS, 0)) == NULL) {
1234                 rc = ENOMEM;
1235                 goto fail1;
1236         }

1238         srp->sr_ksp = ksp;

1240         ksp->ks_update = sfxge_rx_kstat_update;
1241         ksp->ks_private = srp;
1242         ksp->ks_lock = &(sep->se_lock);

1244         /* Initialise the named stats */
1245         knp = ksp->ks_data;
1246         kstat_named_init(knp, "rx_pkt_mem_limit", KSTAT_DATA_UINT32);
1247         knp++;
1248         kstat_named_init(knp, "kcache_alloc_nomem", KSTAT_DATA_UINT32);
1249         knp++;
```



```
1250         kstat_named_init(knp, "dma_alloc_nomem", KSTAT_DATA_UINT32);
1251         knp++;
1252         kstat_named_init(knp, "dma_alloc_fail", KSTAT_DATA_UINT32);
1253         knp++;
1254         kstat_named_init(knp, "dma_bind_nomem", KSTAT_DATA_UINT32);
1255         knp++;
1256         kstat_named_init(knp, "dma_bind_fail", KSTAT_DATA_UINT32);
1257         knp++;
1258 #ifdef _USE_XESBALLOC
1259         kstat_named_init(knp, "xesballoc_fail", KSTAT_DATA_UINT32);
1260 #endif
1261 #ifdef _USE_DESBALLOC
1262         kstat_named_init(knp, "desballoc_fail", KSTAT_DATA_UINT32);
1263 #endif
1264         kstat_named_init(knp, "rxq_empty_discard", KSTAT_DATA_UINT32);

1266         kstat_install(ksp);
1267         return (0);

1269 fail1:
1270         DTRACE_PROBE1(fail1, int, rc);

1272         return (rc);
1273 }

1275 static int
1276 sfxge_rx_qinit(sfxge_t *sp, unsigned int index)
1277 {
1278         sfxge_rxq_t *srp;
1279         int rc;

1281         ASSERT3U(index, <, SFXGE_RX_SCALE_MAX);

1283         srp = kmem_cache_alloc(sp->s_rqc, KM_SLEEP);

1285         ASSERT3U(srp->sr_state, ==, SFXGE_RXQ_UNINITIALIZED);

1287         srp->sr_index = index;
1288         sp->s_srp[index] = srp;

1290         if ((rc = sfxge_rx_kstat_init(srp)) != 0)
1291                 goto fail1;

1293         srp->sr_state = SFXGE_RXQ_INITIALIZED;

1295         return (0);
1296 fail1:
1297         DTRACE_PROBE1(fail1, int, rc);
1298         kmem_cache_free(sp->s_rqc, srp);

1300         return (rc);
1301 }

1303 static int
1304 sfxge_rx_qstart(sfxge_t *sp, unsigned int index)
1305 {
1306         sfxge_evq_t *sep = sp->s_sep[index];
1307         sfxge_rxq_t *srp;
1308         efsys_mem_t *esmp;
1309         efx_nic_t *enp;
1310         unsigned int level;
1311         int rc;

1313         mutex_enter(&(sep->se_lock));
1314         srp = sp->s_srp[index];
1315         enp = sp->s_enp;
```

```
1316         esmp = &(srp->sr_mem);

1318         ASSERT3U(srp->sr_state, ==, SFXGE_RXQ_INITIALIZED);
1319         ASSERT3U(sep->se_state, ==, SFXGE_EVQ_STARTED);

1321         /* Zero the memory */
1322         (void) memset(esmp->esm_base, 0, EFX_RXQ_SIZE(sp->s_rxq_size));

1324         /* Program the buffer table */
1325         if ((rc = sfxge_sram_buf_tbl_set(sp, srp->sr_id, esmp,
1326             EFX_RXQ_NBUFS(sp->s_rxq_size))) != 0)
1327                 goto fail1;

1329         /* Create the receive queue */
1330         if ((rc = efx_rx_qcreate(enp, index, index, EFX_RXQ_TYPE_DEFAULT,
1331             esmp, sp->s_rxq_size, srp->sr_id, sep->se_eep, &(srp->sr_erp)))
1332             != 0)
1333                 goto fail2;

1335         /* Enable the receive queue */
1336         efx_rx_qenable(srp->sr_erp);

1338         /* Set the water marks */
1339         srp->sr_hiwat = EFX_RXQ_LIMIT(sp->s_rxq_size) * 9 / 10;
1340         srp->sr_lowat = srp->sr_hiwat / 2;

1342         srp->sr_state = SFXGE_RXQ_STARTED;

1344         sfxge_rx_qpoll_start(srp);

1346         /* Try to fill the queue from the pool */
1347         sfxge_rx_qrefill(srp, EFX_RXQ_LIMIT(sp->s_rxq_size));

1349         /*
1350          * If there were insufficient buffers in the pool to reach the at
1351          * least a batch then allocate some.
1352          */
1353         level = srp->sr_added - srp->sr_completed;
1354         if (level < SFXGE_RX_BATCH)
1355                 sfxge_rx_qfill(srp, SFXGE_RX_BATCH);

1357         mutex_exit(&(sep->se_lock));

1359         return (0);

1361 fail2:
1362         DTRACE_PROBE(fail2);

1364         /* Clear entries from the buffer table */
1365         sfxge_sram_buf_tbl_clear(sp, srp->sr_id,
1366             EFX_RXQ_NBUFS(sp->s_rxq_size));

1368 fail1:
1369         DTRACE_PROBE1(fail1, int, rc);

1371         mutex_exit(&(sep->se_lock));

1373         return (rc);
1374 }

1376 static void
1377 sfxge_rx_qflow_complete(sfxge_rxq_t *srp, sfxge_rx_flow_t *srfp)
1378 {
1379         mblk_t *mp;
1380         struct ether_header *etherhp;
1381         struct ip *iphp;
```



```
1382         struct tcphdr *thp;

1384         if (srfp->srf_mp == NULL)
1385                 return;

1387         mp = srfp->srf_mp;
1388         etherhp = srfp->srf_etherhp;
1389         iphp = srfp->srf_iphp;
1390         thp = srfp->srf_last_thp;

1392         ASSERT3U(((etherhp->ether_type == htons(ETHERTYPE_VLAN)) ?
1393             sizeof (struct ether_vlan_header) :
1394             sizeof (struct ether_header)) +
1395             srfp->srf_len, ==, msgdsize(mp));

1397         ASSERT3U(srfp->srf_len & 0xffff, ==, srfp->srf_len);
1398         iphp->ip_len = htons(srfp->srf_len);

1400         srfp->srf_first_thp->th_ack = thp->th_ack;
1401         srfp->srf_first_thp->th_win = thp->th_win;
1402         srfp->srf_first_thp->th_flags = thp->th_flags;

1404         DTRACE_PROBE2(flow_complete, uint32_t, srfp->srf_tag,
1405             size_t, srfp->srf_len);

1407         srfp->srf_mp = NULL;
1408         srfp->srf_len = 0;

1410         ASSERT(mp->b_next == NULL);
1411         *(srp->sr_mpp) = mp;
1412         srp->sr_mpp = &(mp->b_next);
1413 }

1415 static boolean_t
1416 sfxge_rx_qflow_add(sfxge_rxq_t *srp, sfxge_rx_flow_t *srfp,
1417     sfxge_rx_packet_t *srpp, clock_t now)
1418 {
1419         sfxge_t *sp = srp->sr_sp;
1420         struct ether_header *etherhp = srpp->srp_etherhp;
1421         struct ip *iphp = srpp->srp_iphp;
1422         struct tcphdr *thp = srpp->srp_thp;
1423         size_t off = srpp->srp_off;
1424         size_t size = (size_t)(srpp->srp_size);
1425         mblk_t *mp = srpp->srp_mp;
1426         uint32_t seq;
1427         unsigned int shift;

1429         ASSERT3U(MBLKL(mp), ==, off + size);
1430         ASSERT3U(DB_CKSUMFLAGS(mp), ==,
1431             HCK_FULLCKSUM | HCK_FULLCKSUM_OK | HCK_IPV4_HDRCKSUM);

1433         seq = htonl(thp->th_seq);

1435         /*
1436          * If the time between this segment and the last is greater than RTO
1437          * then consider this a new flow.
1438          */
1439         if (now - srfp->srf_lbolt > srp->sr_rto) {
1440                 srfp->srf_count = 1;
1441                 srfp->srf_seq = seq + size;

1443                 goto fail1;
1444         }

1446         if (seq != srfp->srf_seq) {
1447                 if (srfp->srf_count > SFXGE_SLOW_START)
```

```
1448                        srfp->srf_count = SFXGE_SLOW_START;

1450                srfp->srf_count >>= 1;

1452                srfp->srf_count++;
1453                srfp->srf_seq = seq + size;

1455                goto fail2;
1456        }

1458        /* Update the in-order segment count and sequence number */
1459        srfp->srf_count++;
1460        srfp->srf_seq = seq + size;

1462        /* Don't merge across pure ACK, URG, SYN or RST segments */
1463        if (size == 0 || thp->th_flags & (TH_URG | TH_SYN | TH_RST) ||
1464            thp->th_urp != 0)
1465                goto fail3;

1467        /*
1468         * If the in-order segment count has not yet reached the slow-start
1469         * threshold then we cannot coalesce.
1470         */
1471        if (srfp->srf_count < SFXGE_SLOW_START)
1472                goto fail4;

1474        /* Scale up the packet size from 4k (the maximum being 64k) */
1475        ASSERT3U(srfp->srf_count, >=, SFXGE_SLOW_START);
1476        shift = MIN(srfp->srf_count - SFXGE_SLOW_START + 12, 16);
1477        if (srfp->srf_len + size >= (1 << shift))
1478                sfxge_rx_qflow_complete(srp, srfp);

1480        ASSERT(mp->b_cont == NULL);

1482 #ifdef _USE_GLD_V3_SOL10
1483        /*
1484         * The IP and UDP layers in Solaris 10 have slow paths for
1485         * handling mblks with more than 2 fragments.
1486         * UDP: see OpenSolaris CR 6305037
1487         * IP: see <http://www.mail-archive.com/networking-discuss@
1488         *   opensolaris.org/msg07366.html>
1489         */
1490        if (srfp->srf_mp && srfp->srf_mp->b_cont) {
1491                sfxge_rx_qflow_complete(srp, srfp);
1492        }
1493 #endif

1495        if (srfp->srf_mp == NULL) {
1496                /* First packet in this flow */
1497                srfp->srf_etherhp = etherhp;
1498                srfp->srf_iphp = iphp;
1499                srfp->srf_first_thp = srfp->srf_last_thp = thp;

1501                ASSERT3P(mp->b_cont, ==, NULL);
1502                srfp->srf_mp = mp;
1503                srfp->srf_mpp = &(mp->b_cont);

1505                srfp->srf_len = ntohs(iphp->ip_len);

1507                /*
1508                 * If the flow is not already in the list of occupied flows then
1509                 * add it.
1510                 */
1511                if (srfp->srf_next == NULL &&
1512                    srp->sr_srfpp != &(srfp->srf_next)) {
1513                        *(srp->sr_srfpp) = srfp;
```



```
1514                        srp->sr_srfpp = &(srfp->srf_next);
1515                }
1516        } else {
1517                /* Later packet in this flow - skip TCP header */
1518                srfp->srf_last_thp = thp;

1520                mp->b_rptr += off;
1521                ASSERT3U(MBLKL(mp), ==, size);

1523                ASSERT3P(mp->b_cont, ==, NULL);
1524                *(srfp->srf_mpp) = mp;
1525                srfp->srf_mpp = &(mp->b_cont);

1527                srfp->srf_len += size;

1529                ASSERT(srfp->srf_next != NULL ||
1530                    srp->sr_srfpp == &(srfp->srf_next));
1531        }

1533        DTRACE_PROBE2(flow_add, uint32_t, srfp->srf_tag, size_t, size);

1535        /*
1536         * Try to align coalesced segments on push boundaries, unless they
1537         * are too frequent.
1538         */
1539        if (sp->s_rx_coalesce_mode == SFXGE_RX_COALESCE_ALLOW_PUSH &&
1540            thp->th_flags & TH_PUSH)
1541                sfxge_rx_qflow_complete(srp, srfp);

1543        srfp->srf_lbolt = now;
1544        return (B_TRUE);

1546 fail4:
1547 fail3:
1548 fail2:
1549 fail1:
1550        sfxge_rx_qflow_complete(srp, srfp);

1552        srfp->srf_lbolt = now;
1553        return (B_FALSE);
1554 }

1556 void
1557 sfxge_rx_qpacket_coalesce(sfxge_rxq_t *srp)
1558 {
1559        sfxge_t *sp = srp->sr_sp;
1560        clock_t now;
1561        mblk_t *mp;
1562        sfxge_rx_flow_t *srfp;

1564        ASSERT(sp->s_rx_coalesce_mode != SFXGE_RX_COALESCE_OFF);

1566        now = ddi_get_lbolt();

1568        mp = srp->sr_mp;

1570        srp->sr_mp = NULL;
1571        srp->sr_mpp = &(srp->sr_mp);

1573        /* Start with the last flow to be appended to */
1574        srfp = *(srp->sr_srfpp);

1576        while (mp != NULL) {
1577                frtn_t *freep;
1578                sfxge_rx_packet_t *srpp;
1579                struct ether_header *etherhp;
```

```
1580                struct ip *iphp;
1581                struct tcphdr *thp;
1582                size_t off;
1583                size_t size;
1584                uint16_t ether_tci;
1585                uint16_t hash;
1586                uint32_t tag;
1587                mblk_t *next;

1589                next = mp->b_next;
1590                mp->b_next = NULL;

1592                if (next != NULL)
1593                        prefetch_read_many(next);

1595                freep = DB_FRTNP(mp);
1596                /*LINTED*/
1597                srpp = (sfxge_rx_packet_t *)(freep->free_arg);
1598                ASSERT3P(srpp->srp_mp, ==, mp);

1600                /* If the packet is not TCP then we cannot coalesce it */
1601                if (~(srpp->srp_flags) & EFX_PKT_TCP)
1602                        goto reject;

1604                /*
1605                 * If the packet is not fully checksummed then we cannot
1606                 * coalesce it.
1607                 */
1608                if (~(srpp->srp_flags) & (EFX_CKSUM_TCPUDP | EFX_CKSUM_IPV4))
1609                        goto reject;

1611                /* Parse the TCP header */
1612                sfxge_tcp_parse(mp, &etherhp, &iphp, &thp, &off,
1613                    &size);
1614                ASSERT(etherhp != NULL);
1615                ASSERT(iphp != NULL);
1616                ASSERT(thp != NULL);
1617                ASSERT(off != 0);

1619                if ((iphp->ip_off & ~htons(IP_DF)) != 0)
1620                        goto reject;

1622                if (etherhp->ether_type == htons(ETHERTYPE_VLAN)) {
1623                        struct ether_vlan_header *ethervhp;

1625                        ethervhp = (struct ether_vlan_header *)etherhp;
1626                        ether_tci = ethervhp->ether_tci;
1627                } else {
1628                        ether_tci = 0;
1629                }

1631                /*
1632                 * Make sure any minimum length padding is stripped
1633                 * before we try to add the packet to a flow.
1634                 */
1635                ASSERT3U(sp->s_rx_prefix_size + MBLKL(mp), ==,
1636                    (size_t)(srpp->srp_size));
1637                ASSERT3U(sp->s_rx_prefix_size + off + size, <=,
1638                    (size_t)(srpp->srp_size));

1640                if (sp->s_rx_prefix_size + off + size <
1641                    (size_t)(srpp->srp_size))
1642                        mp->b_wptr = mp->b_rptr + off + size;

1644                /*
1645                 * If there is no current flow, or the segment does not match
```



```
1646                 * the current flow then we must attempt to look up the
1647                 * correct flow in the table.
1648                 */
1649                if (srfp == NULL)
1650                        goto lookup;

1652                if (srfp->srf_saddr != iphp->ip_src.s_addr ||
1653                    srfp->srf_daddr != iphp->ip_dst.s_addr)
1654                        goto lookup;

1656                if (srfp->srf_sport != thp->th_sport ||
1657                    srfp->srf_dport != thp->th_dport)
1658                        goto lookup;

1660                if (srfp->srf_tci != ether_tci)
1661                        goto lookup;

1663        add:
1664                ASSERT(srfp != NULL);

1666                srpp->srp_etherhp = etherhp;
1667                srpp->srp_iphp = iphp;
1668                srpp->srp_thp = thp;
1669                srpp->srp_off = off;

1671                ASSERT3U(size, <, (1 << 16));
1672                srpp->srp_size = (uint16_t)size;

1674                /* Try to append the packet to the flow */
1675                if (!sfxge_rx_qflow_add(srp, srfp, srpp, now))
1676                        goto reject;

1678                mp = next;
1679                continue;

1681        lookup:
1682                /*
1683                 * If there is a prefix area then read the hash from that,
1684                 * otherwise calculate it.
1685                 */
1686                if (sp->s_rx_prefix_size != 0) {
1687                        hash = EFX_RX_HASH_VALUE(EFX_RX_HASHALG_LFSR,
1688                            DB_BASE(mp));
1689                } else {
1690                        SFXGE_TCP_HASH(
1691                            ntohl(iphp->ip_src.s_addr),
1692                            ntohs(thp->th_sport),
1693                            ntohl(iphp->ip_dst.s_addr),
1694                            ntohs(thp->th_dport),
1695                            hash);
1696                }

1698                srfp = &(srp->sr_flow[(hash >> 6) % SFXGE_MAX_FLOW]);
1699                tag = (uint32_t)hash + 1; /* Make sure it's not zero */

1701                /*
1702                 * If the flow we have found does not match the hash then
1703                 * it may be an unused flow, or it may be stale.
1704                 */
1705                if (tag != srfp->srf_tag) {
1706                        if (srfp->srf_count != 0) {
1707                                if (now - srfp->srf_lbolt <= srp->sr_rto)
1708                                        goto reject;
1709                        }

1711                        if (srfp->srf_mp != NULL)
```

```
1712                                goto reject;

1714                        /* Start a new flow */
1715                        ASSERT(srfp->srf_next == NULL);

1717                        srfp->srf_tag = tag;

1719                        srfp->srf_saddr = iphp->ip_src.s_addr;
1720                        srfp->srf_daddr = iphp->ip_dst.s_addr;
1721                        srfp->srf_sport = thp->th_sport;
1722                        srfp->srf_dport = thp->th_dport;
1723                        srfp->srf_tci = ether_tci;

1725                        srfp->srf_count = 0;
1726                        srfp->srf_seq = ntohl(thp->th_seq);

1728                        srfp->srf_lbolt = now;
1729                        goto add;
1730                }

1732                /*
1733                 * If the flow we have found does match the hash then it could
1734                 * still be an alias.
1735                 */
1736                if (srfp->srf_saddr != iphp->ip_src.s_addr ||
1737                    srfp->srf_daddr != iphp->ip_dst.s_addr)
1738                        goto reject;

1740                if (srfp->srf_sport != thp->th_sport ||
1741                    srfp->srf_dport != thp->th_dport)
1742                        goto reject;

1744                if (srfp->srf_tci != ether_tci)
1745                        goto reject;

1747                goto add;

1749 reject:
1750                *(srp->sr_mpp) = mp;
1751                srp->sr_mpp = &(mp->b_next);

1753                mp = next;
1754        }
1755 }

1757 void
1758 sfxge_rx_qcomplete(sfxge_rxq_t *srp, boolean_t eop)
1759 {
1760        sfxge_t *sp = srp->sr_sp;
1761        unsigned int index = srp->sr_index;
1762        sfxge_evq_t *sep = sp->s_sep[index];
1763        unsigned int completed;
1764        sfxge_rx_fpp_t *srfppp = &(srp->sr_fpp);
1765        unsigned int level;

1767        ASSERT(mutex_owned(&(sep->se_lock)));

1769        ASSERT(srp->sr_mp == NULL);
1770        ASSERT(srp->sr_mpp == &(srp->sr_mp));

1772        completed = srp->sr_completed;
1773        while (completed != srp->sr_pending) {
1774                unsigned int id;
1775                sfxge_rx_packet_t *srpp;
1776                mblk_t *mp;
1777                size_t size;
```



```
1778                uint16_t flags;

1780                id = completed++ & (sp->s_rxq_size - 1);

1782                if (srp->sr_pending - completed >= 4) {
1783                        unsigned int prefetch;

1785                        prefetch = (id + 4) & (sp->s_rxq_size - 1);

1787                        srpp = srp->sr_srpp[prefetch];
1788                        ASSERT(srpp != NULL);

1790                        mp = srpp->srp_mp;
1791                        prefetch_read_many(mp->b_datap);
1792                } else if (completed == srp->sr_pending) {
1793                        prefetch_read_many(srp->sr_mp);
1794                }

1796                srpp = srp->sr_srpp[id];
1797                ASSERT(srpp != NULL);

1799                srp->sr_srpp[id] = NULL;

1801                mp = srpp->srp_mp;
1802                ASSERT(mp->b_cont == NULL);

1804                /* when called from sfxge_rx_qstop() */
1805                if (srp->sr_state != SFXGE_RXQ_STARTED)
1806                        goto discard;

1808                if (srpp->srp_flags & (EFX_ADDR_MISMATCH | EFX_DISCARD))
1809                        goto discard;

1811                /* Set up the packet length */
1812                ASSERT3P(mp->b_rptr, ==, DB_BASE(mp));
1813                mp->b_rptr += sp->s_rx_prefix_size;

1815                prefetch_read_many(mp->b_rptr);

1817                ASSERT3P(mp->b_wptr, ==, DB_BASE(mp));
1818                mp->b_wptr += (size_t)(srpp->srp_size);
1819                ASSERT3P(mp->b_wptr, <=, DB_LIM(mp));

1821                /* Calculate the maximum packet size */
1822                size = sp->s_mtu;
1823                size += (srpp->srp_flags & EFX_PKT_VLAN_TAGGED) ?
1824                    sizeof (struct ether_vlan_header) :
1825                    sizeof (struct ether_header);

1827                if (MBLKL(mp) > size)
1828                        goto discard;

1830                /* Make the data visible to the kernel */
1831                (void) ddi_dma_sync(srpp->srp_dma_handle, 0,
1832                    (size_t)(srpp->srp_size), DDI_DMA_SYNC_FORKERNEL);

1834                /* Check for loopback packets */
1835                if (!(srpp->srp_flags & EFX_PKT_IPV4) &&
1836                    !(srpp->srp_flags & EFX_PKT_IPV6)) {
1837                        struct ether_header *etherhp;

1839                        /*LINTED*/
1840                        etherhp = (struct ether_header *)(mp->b_rptr);

1842                        if (etherhp->ether_type ==
1843                            htons(SFXGE_ETHERTYPE_LOOPBACK)) {
```

```
1844                                DTRACE_PROBE(loopback);

1846                                srp->sr_loopback++;
1847                                goto discard;
1848                        }
1849                }

1851                /* Set up the checksum information */
1852                flags = 0;

1854                if (srpp->srp_flags & EFX_CKSUM_IPV4) {
1855                        ASSERT(srpp->srp_flags & EFX_PKT_IPV4);
1856                        flags |= HCK_IPV4_HDRCKSUM;
1857                }

1859                if (srpp->srp_flags & EFX_CKSUM_TCPUDP) {
1860                        ASSERT(srpp->srp_flags & EFX_PKT_TCP ||
1861                            srpp->srp_flags & EFX_PKT_UDP);
1862                        flags |= HCK_FULLCKSUM | HCK_FULLCKSUM_OK;
1863                }

1865                DB_CKSUMSTART(mp) = 0;
1866                DB_CKSUMSTUFF(mp) = 0;
1867                DB_CKSUMEND(mp) = 0;
1868                DB_CKSUMFLAGS(mp) = flags;
1869                DB_CKSUM16(mp) = 0;

1871                /* Add the packet to the tail of the chain */
1872                srfppp->srfpp_loaned++;

1874                ASSERT(mp->b_next == NULL);
1875                *(srp->sr_mpp) = mp;
1876                srp->sr_mpp = &(mp->b_next);

1878                continue;

1880 discard:
1881                /* Return the packet to the pool */
1882                srfppp->srfpp_loaned++;
1883                freeb(mp); /* Equivalent to freemsg() as b_cont==0 */
1884        }
1885        srp->sr_completed = completed;

1887        /* Attempt to coalesce any TCP packets */
1888        if (sp->s_rx_coalesce_mode != SFXGE_RX_COALESCE_OFF)
1889                sfxge_rx_qpacket_coalesce(srp);

1891        /*
1892         * If there are any pending flows and this is the end of the
1893         * poll then they must be completed.
1894         */
1895        if (srp->sr_srfp != NULL && eop) {
1896                sfxge_rx_flow_t *srfp;

1898                srfp = srp->sr_srfp;

1900                srp->sr_srfp = NULL;
1901                srp->sr_srfpp = &(srp->sr_srfp);

1903                do {
1904                        sfxge_rx_flow_t *next;

1906                        next = srfp->srf_next;
1907                        srfp->srf_next = NULL;

1909                        sfxge_rx_qflow_complete(srp, srfp);
```



```
1911                        srfp = next;
1912                } while (srfp != NULL);
1913        }

1915        level = srp->sr_added - srp->sr_completed;

1917        /* If there are any packets then pass them up the stack */
1918        if (srp->sr_mp != NULL) {
1919                mblk_t *mp;

1921                mp = srp->sr_mp;

1923                srp->sr_mp = NULL;
1924                srp->sr_mpp = &(srp->sr_mp);

1926                if (level == 0) {
1927                        /* Try to refill ASAP */
1928                        sfxge_rx_qrefill(srp, EFX_RXQ_LIMIT(sp->s_rxq_size));
1929                        level = srp->sr_added - srp->sr_completed;
1930                }

1932                /*
1933                 * If the RXQ is still empty, discard and recycle the
1934                 * current entry to ensure that the ring always
1935                 * contains at least one descriptor. This ensures that
1936                 * the next hardware RX will trigger an event
1937                 * (possibly delayed by interrupt moderation) and
1938                 * trigger another refill/fill attempt.
1939                 *
1940                 * Note this drops a complete LRO fragment from the
1941                 * start of the batch.
1942                 *
1943                 * Note also that copymsgchain() does not help with
1944                 * resource starvation here, unless we are short of DMA
1945                 * mappings.
1946                 */
1947                if (level == 0) {
1948                        mblk_t *nmp;

1950                        srp->sr_kstat.srk_rxq_empty_discard++;
1951                        DTRACE_PROBE1(rxq_empty_discard, int, index);
1952                        nmp = mp->b_next;
1953                        if (nmp)
1954                                sfxge_gld_rx_post(sp, index, nmp);
1955                        /* as level==0 will swizzle,rxpost below */
1956                        freemsg(mp);
1957                } else {
1958                        sfxge_gld_rx_post(sp, index, mp);
1959                }
1960        }

1962        /* Top up the queue if necessary */
1963        if (level < srp->sr_hiwat) {
1964                sfxge_rx_qrefill(srp, EFX_RXQ_LIMIT(sp->s_rxq_size));

1966                level = srp->sr_added - srp->sr_completed;
1967                if (level < srp->sr_lowat)
1968                        sfxge_rx_qfill(srp, EFX_RXQ_LIMIT(sp->s_rxq_size));
1969        }
1970 }

1972 static unsigned int
1973 sfxge_rx_qloopback(sfxge_t *sp, unsigned int index)
1974 {
1975        sfxge_evq_t *sep = sp->s_sep[index];
```

```
1976         sfxge_rxq_t *srp;
1977         unsigned int count;

1979         mutex_enter(&(sep->se_lock));
1980         srp = sp->s_srp[index];
1981         count = srp->sr_loopback;
1982         srp->sr_loopback = 0;
1983         mutex_exit(&(sep->se_lock));

1985         return (count);
1986 }

1988 void
1989 sfxge_rx_qflush_done(sfxge_rxq_t *srp)
1990 {
1991         sfxge_t *sp = srp->sr_sp;
1992         unsigned int index = srp->sr_index;
1993         sfxge_evq_t *sep = sp->s_sep[index];

1995         ASSERT(mutex_owned(&(sep->se_lock)));

1997         /* SFCbug22989: events may be delayed. EVQs are stopped after RXQs */
1998         if ((srp->sr_state != SFXGE_RXQ_INITIALIZED) ||
1999             (srp->sr_flush == SFXGE_FLUSH_DONE))
2000                 return;

2002         /* Flush successful: wakeup sfxge_rx_qstop() */
2003         srp->sr_flush = SFXGE_FLUSH_DONE;
2004         cv_broadcast(&(srp->sr_flush_kv));
2005 }

2007 void
2008 sfxge_rx_qflush_failed(sfxge_rxq_t *srp)
2009 {
2010         sfxge_t *sp = srp->sr_sp;
2011         unsigned int index = srp->sr_index;
2012         sfxge_evq_t *sep = sp->s_sep[index];

2014         ASSERT(mutex_owned(&(sep->se_lock)));

2016         /* SFCbug22989: events may be delayed. EVQs are stopped after RXQs */
2017         if ((srp->sr_state != SFXGE_RXQ_INITIALIZED) ||
2018             (srp->sr_flush == SFXGE_FLUSH_DONE))
2019                 return;

2021         /* SFCbug22989: events may be delayed. EVQs are stopped after RXQs */
2022         if (srp->sr_state != SFXGE_RXQ_STARTED)
2023                 return;

2025         /* Flush failed, so retry until timeout in sfxge_rx_qstop() */
2026         srp->sr_flush = SFXGE_FLUSH_FAILED;
2027         efx_rx_qflush(srp->sr_erp);
2028 }

2030 static void
2031 sfxge_rx_qstop(sfxge_t *sp, unsigned int index)
2032 {
2033         sfxge_evq_t *sep = sp->s_sep[index];
2034         sfxge_rxq_t *srp;
2035         clock_t timeout;

2037         mutex_enter(&(sep->se_lock));

2039         srp = sp->s_srp[index];
2040         ASSERT3U(srp->sr_state, ==, SFXGE_RXQ_STARTED);
```



```
2042         sfxge_rx_qpoll_stop(srp);

2044         srp->sr_state = SFXGE_RXQ_INITIALIZED;

2046         if (sp->s_hw_err == SFXGE_HW_OK) {
2047                 /* Wait upto 2sec for queue flushing to complete */
2048                 srp->sr_flush = SFXGE_FLUSH_PENDING;
2049                 efx_rx_qflush(srp->sr_erp);
2050         } else {
2051                 /* Do not attempt flush if indication of H/W failure */
2052                 srp->sr_flush = SFXGE_FLUSH_DONE;
2053         }

2055         timeout = ddi_get_lbolt() + drv_usectohz(SFXGE_RX_QFLUSH_USEC);

2057         while (srp->sr_flush != SFXGE_FLUSH_DONE) {
2058                 if (cv_timedwait(&(srp->sr_flush_kv), &(sep->se_lock),
2059                         timeout) < 0) {
2060                         /* Timeout waiting for successful flush */
2061                         dev_info_t *dip = sp->s_dip;

2063                                 ddi_driver_name(sp->s_dip),
2064                         cmn_err(CE_NOTE,
2065                             SFXGE_CMN_ERR "[%s%d] rxq[%d] flush timeout",
2066                             ddi_driver_name(dip), ddi_get_instance(dip), index);
2067                         break;
2068                 }
2069         }

2071         DTRACE_PROBE1(flush, sfxge_flush_state_t, srp->sr_flush);
2072         srp->sr_flush = SFXGE_FLUSH_DONE;

2074         /* Destroy the receive queue */
2075         efx_rx_qdestroy(srp->sr_erp);
2076         srp->sr_erp = NULL;

2078         /* Clear entries from the buffer table */
2079         sfxge_sram_buf_tbl_clear(sp, srp->sr_id,
2080             EFX_RXQ_NBUFS(sp->s_rxq_size));

2082         /*
2083          * Free any unused RX packets which had descriptors on the RXQ
2084          * Packets will be discard as state != STARTED
2085          */
2086         srp->sr_pending = srp->sr_added;
2087         sfxge_rx_qcomplete(srp, B_TRUE);

2089         ASSERT3U(srp->sr_completed, ==, srp->sr_pending);

2091         srp->sr_added = 0;
2092         srp->sr_pending = 0;
2093         srp->sr_completed = 0;
2094         srp->sr_loopback = 0;

2096         srp->sr_lowat = 0;
2097         srp->sr_hiwat = 0;

2099         mutex_exit(&(sep->se_lock));
2100 }

2102 static void
2103 sfxge_rx_kstat_fini(sfxge_rxq_t *srp)
2104 {
2105         kstat_delete(srp->sr_ksp);
2106         srp->sr_ksp = NULL;
2107 }
```

```
2109 static void
2110 sfxge_rx_qfini(sfxge_t *sp, unsigned int index)
2111 {
2112         sfxge_rxq_t *srp = sp->s_srp[index];

2114         ASSERT3U(srp->sr_state, ==, SFXGE_RXQ_INITIALIZED);

2116         sp->s_srp[index] = NULL;
2117         srp->sr_state = SFXGE_RXQ_UNINITIALIZED;

2119         sfxge_rx_kstat_fini(srp);

2121         /* Empty the pool */
2122         sfxge_rx_qfpp_empty(srp);

2124         srp->sr_index = 0;

2126         kmem_cache_free(sp->s_rqc, srp);
2127 }

2129 static int
2130 sfxge_rx_scale_kstat_update(kstat_t *ksp, int rw)
2131 {
2132         sfxge_t *sp = ksp->ks_private;
2133         sfxge_rx_scale_t *srsp = &(sp->s_rx_scale);
2134         sfxge_intr_t *sip = &(sp->s_intr);
2135         kstat_named_t *knp;
2136         unsigned int index;
2137         unsigned int entry;
2138         unsigned int *freq;
2139         int rc;

2141         ASSERT(mutex_owned(&(srsp->srs_lock)));

2143         if (rw != KSTAT_READ) {
2144                 rc = EACCES;
2145                 goto fail1;
2146         }

2148         if ((freq = kmem_zalloc(sizeof (unsigned int) * sip->si_nalloc,
2149                                 KM_NOSLEEP)) == NULL) {
2150                 rc = ENOMEM;
2151                 goto fail2;
2152         }

2154         for (index = 0; index < sip->si_nalloc; index++)
2155                 freq[index] = 0;

2157         for (entry = 0; entry < SFXGE_RX_SCALE_MAX; entry++) {
2158                 index = srsp->srs_tbl[entry];

2160                 freq[index]++;
2161         }

2163         knp = ksp->ks_data;
2164         for (index = 0; index < sip->si_nalloc; index++) {
2165                 knp->value.ui64 = freq[index];
2166                 knp++;
2167         }

2169         knp->value.ui64 = srsp->srs_count;

2171         kmem_free(freq, sizeof (unsigned int) * sip->si_nalloc);

2173         return (0);
```



```
2175 fail2:
2176         DTRACE_PROBE(fail2);
2177 fail1:
2178         DTRACE_PROBE1(fail1, int, rc);
2179         return (rc);
2180 }

2182 static int
2183 sfxge_rx_scale_kstat_init(sfxge_t *sp)
2184 {
2185         dev_info_t *dip = sp->s_dip;
2186         sfxge_rx_scale_t *srsp = &(sp->s_rx_scale);
2187         sfxge_intr_t *sip = &(sp->s_intr);
2188         char name[MAXNAMELEN];
2189         kstat_t *ksp;
2190         kstat_named_t *knp;
2191         unsigned int index;
2192         int rc;

2194         /* Create the set */
2195         (void) snprintf(name, MAXNAMELEN - 1, "%s_rss", ddi_driver_name(dip));

2197         if ((ksp = kstat_create((char *)ddi_driver_name(dip),
2198             ddi_get_instance(dip), name, "rss", KSTAT_TYPE_NAMED,
2199             sip->si_nalloc + 1, 0)) == NULL) {
2200                 rc = ENOMEM;
2201                 goto fail1;
2202         }

2204         srsp->srs_ksp = ksp;

2206         ksp->ks_update = sfxge_rx_scale_kstat_update;
2207         ksp->ks_private = sp;
2208         ksp->ks_lock = &(srsp->srs_lock);

2210         /* Initialise the named stats */
2211         knp = ksp->ks_data;
2212         for (index = 0; index < sip->si_nalloc; index++) {
2213                 char name[MAXNAMELEN];

2215                 (void) snprintf(name, MAXNAMELEN - 1, "evq%04d_count", index);
2216                 kstat_named_init(knp, name, KSTAT_DATA_UINT64);
2217                 knp++;
2218         }

2220         kstat_named_init(knp, "scale", KSTAT_DATA_UINT64);

2222         kstat_install(ksp);
2223         return (0);

2225 fail1:
2226         DTRACE_PROBE1(fail1, int, rc);

2228         return (rc);
2229 }

2231 static void
2232 sfxge_rx_scale_kstat_fini(sfxge_t *sp)
2233 {
2234         sfxge_rx_scale_t *srsp = &(sp->s_rx_scale);

2236         /* Destroy the set */
2237         kstat_delete(srsp->srs_ksp);
2238         srsp->srs_ksp = NULL;
2239 }
```

```
2242 unsigned int
2243 sfxge_rx_scale_prop_get(sfxge_t *sp)
2244 {
2245         int rx_scale;

2247         rx_scale = ddi_prop_get_int(DDI_DEV_T_ANY, sp->s_dip,
2248                                     DDI_PROP_DONTPASS, "rx_scale_count",
2249                                     SFXGE_RX_SCALE_MAX);
2250         /* 0 and all -ve numbers sets to number of logical CPUs */
2251         if (rx_scale <= 0)
2252                 rx_scale = ncpus;

2254         return (rx_scale);
2255 }


2258 static int
2259 sfxge_rx_scale_init(sfxge_t *sp)
2260 {
2261         sfxge_rx_scale_t *srsp = &(sp->s_rx_scale);
2262         sfxge_intr_t *sip = &(sp->s_intr);
2263         int rc;

2265         ASSERT3U(srsp->srs_state, ==, SFXGE_RX_SCALE_UNINITIALIZED);

2267         /* Create tables for CPU, core, cache and chip counts */
2268         srsp->srs_cpu = kmem_zalloc(sizeof (unsigned int) * NCPU, KM_SLEEP);
2269 #ifdef  _USE_CPU_PHYSID
2270         srsp->srs_core = kmem_zalloc(sizeof (unsigned int) * NCPU, KM_SLEEP);
2271         srsp->srs_cache = kmem_zalloc(sizeof (unsigned int) * NCPU, KM_SLEEP);
2272         srsp->srs_chip = kmem_zalloc(sizeof (unsigned int) * NCPU, KM_SLEEP);
2273 #endif

2275         mutex_init(&(srsp->srs_lock), NULL, MUTEX_DRIVER, NULL);

2277         /* We need at least one event queue */
2278         srsp->srs_count = sfxge_rx_scale_prop_get(sp);
2279         if (srsp->srs_count > sip->si_nalloc)
2280                 srsp->srs_count = sip->si_nalloc;
2281         if (srsp->srs_count < 1)
2282                 srsp->srs_count = 1;

2284         /* Set up the kstats */
2285         if ((rc = sfxge_rx_scale_kstat_init(sp)) != 0)
2286                 goto fail1;

2288         srsp->srs_state = SFXGE_RX_SCALE_INITIALIZED;

2290         return (0);

2292 fail1:
2293         DTRACE_PROBE1(fail1, int, rc);
2294         mutex_destroy(&(srsp->srs_lock));

2296         return (rc);
2297 }

2299 void
2300 sfxge_rx_scale_update(void *arg)
2301 {
2302         sfxge_t *sp = arg;
2303         sfxge_rx_scale_t *srsp = &(sp->s_rx_scale);
2304         sfxge_intr_t *sip;
2305         processorid_t id;
```



```
2306         unsigned int count;
2307         unsigned int *tbl;
2308         unsigned int *rating;
2309         unsigned int entry;
2310         int rc;

2312         mutex_enter(&(srsp->srs_lock));

2314         if (srsp->srs_state != SFXGE_RX_SCALE_STARTED) {
2315                 rc = EFAULT;
2316                 goto fail1;
2317         }

2319         if ((tbl =  kmem_zalloc(sizeof (unsigned int) * SFXGE_RX_SCALE_MAX,
2320                                 KM_NOSLEEP)) == NULL) {
2321                 rc = ENOMEM;
2322                 goto fail2;
2323         }

2325         sip = &(sp->s_intr);
2326         if ((rating = kmem_zalloc(sizeof (unsigned int) * sip->si_nalloc,
2327                                   KM_NOSLEEP)) == NULL) {
2328                 rc = ENOMEM;
2329                 goto fail3;
2330         }

2332         mutex_enter(&cpu_lock);

2334         /*
2335          * Substract any current CPU, core, cache and chip usage from the
2336          * global contention tables.
2337          */
2338         for (id = 0; id < NCPU; id++) {
2339                 ASSERT3U(sfxge_cpu[id], >=, srsp->srs_cpu[id]);
2340                 sfxge_cpu[id] -= srsp->srs_cpu[id];
2341                 srsp->srs_cpu[id] = 0;

2343 #ifdef  _USE_CPU_PHYSID
2344                 ASSERT3U(sfxge_core[id], >=, srsp->srs_core[id]);
2345                 sfxge_core[id] -= srsp->srs_core[id];
2346                 srsp->srs_core[id] = 0;

2348                 ASSERT3U(sfxge_cache[id], >=, srsp->srs_cache[id]);
2349                 sfxge_cache[id] -= srsp->srs_cache[id];
2350                 srsp->srs_cache[id] = 0;

2352                 ASSERT3U(sfxge_chip[id], >=, srsp->srs_chip[id]);
2353                 sfxge_chip[id] -= srsp->srs_chip[id];
2354                 srsp->srs_chip[id] = 0;
2355 #endif
2356         }

2358         ASSERT(srsp->srs_count != 0);

2360         /* Choose as many event queues as we need */
2361         for (count = 0; count < srsp->srs_count; count++) {
2362                 unsigned int index;
2363                 sfxge_evq_t *sep;
2364                 unsigned int choice;
2365                 unsigned int choice_rating;

2367                 bzero(rating, sizeof (unsigned int) * sip->si_nalloc);

2369                 /*
2370                  * Rate each event queue on its global level of CPU
2371                  * contention.
```

```
2372                 */
2373                for (index = 0; index < sip->si_nalloc; index++) {
2374                        sep = sp->s_sep[index];

2376                        id = sep->se_cpu_id;
2377                        rating[index] += sfxge_cpu[id];

2379 #ifdef  _USE_CPU_PHYSID
2380                        id = sep->se_core_id;
2381                        rating[index] += sfxge_core[id];

2383                        id = sep->se_cache_id;
2384                        rating[index] += sfxge_cache[id];

2386                        id = sep->se_chip_id;
2387                        rating[index] += sfxge_chip[id];
2388 #endif
2389                }

2391                /* Choose the queue with the lowest CPU contention */
2392                choice = 0;
2393                choice_rating = rating[0];

2395                for (index = 1; index < sip->si_nalloc; index++) {
2396                        if (rating[index] < choice_rating) {
2397                                choice = index;
2398                                choice_rating = rating[index];
2399                        }
2400                }

2402                /* Add our choice to the condensed RSS table */
2403                tbl[count] = choice;

2405                /* Add information to the global contention tables */
2406                sep = sp->s_sep[choice];

2408                id = sep->se_cpu_id;
2409                srsp->srs_cpu[id]++;
2410                sfxge_cpu[id]++;

2412 #ifdef  _USE_CPU_PHYSID
2413                id = sep->se_core_id;
2414                srsp->srs_core[id]++;
2415                sfxge_core[id]++;

2417                id = sep->se_cache_id;
2418                srsp->srs_cache[id]++;
2419                sfxge_cache[id]++;

2421                id = sep->se_chip_id;
2422                srsp->srs_chip[id]++;
2423                sfxge_chip[id]++;
2424 #endif
2425        }

2427        mutex_exit(&cpu_lock);

2429        /* Build the expanded RSS table */
2430        count = 0;
2431        for (entry = 0; entry < SFXGE_RX_SCALE_MAX; entry++) {
2432                unsigned int index;

2434                index = tbl[count];
2435                count = (count + 1) % srsp->srs_count;

2437                srsp->srs_tbl[entry] = index;
```



```
2438        }

2440        /* Program the expanded RSS table into the hardware */
2441        (void) efx_rx_scale_tbl_set(sp->s_enp, srsp->srs_tbl,
2442            SFXGE_RX_SCALE_MAX);

2444        mutex_exit(&(srsp->srs_lock));
2445        kmem_free(rating, sizeof (unsigned int) * sip->si_nalloc);
2446        kmem_free(tbl, sizeof (unsigned int) * SFXGE_RX_SCALE_MAX);
2447        return;

2449 fail3:
2450        DTRACE_PROBE(fail3);
2451        kmem_free(tbl, sizeof (unsigned int) * SFXGE_RX_SCALE_MAX);
2452 fail2:
2453        DTRACE_PROBE(fail2);
2454 fail1:
2455        DTRACE_PROBE1(fail1, int, rc);

2457        mutex_exit(&(srsp->srs_lock));
2458 }

2460 static int
2461 sfxge_rx_scale_start(sfxge_t *sp)
2462 {
2463        sfxge_rx_scale_t *srsp = &(sp->s_rx_scale);
2464        const efx_nic_cfg_t *encp;
2465        int rc;

2467        mutex_enter(&(srsp->srs_lock));

2469        ASSERT3U(srsp->srs_state, ==, SFXGE_RX_SCALE_INITIALIZED);

2471        /* Clear down the RSS table */
2472        bzero(srsp->srs_tbl, sizeof (unsigned int) * SFXGE_RX_SCALE_MAX);

2474        (void) efx_rx_scale_tbl_set(sp->s_enp, srsp->srs_tbl,
2475            SFXGE_RX_SCALE_MAX);

2477        /* Make sure the LFSR hash is selected */
2478        encp = efx_nic_cfg_get(sp->s_enp);
2479        if ((rc = efx_rx_scale_mode_set(sp->s_enp, EFX_RX_HASHALG_LFSR, 0,
2480            (encp->enc_features & EFX_FEATURE_LFSR_HASH_INSERT))) != 0)
2481                goto fail1;

2483        srsp->srs_state = SFXGE_RX_SCALE_STARTED;

2485        mutex_exit(&(srsp->srs_lock));

2487        /* sfxge_t->s_state_lock held */
2488        (void) ddi_taskq_dispatch(sp->s_tqp, sfxge_rx_scale_update, sp,
2489            DDI_SLEEP);

2491        return (0);

2493 fail1:
2494        DTRACE_PROBE1(fail1, int, rc);

2496        mutex_exit(&(srsp->srs_lock));

2498        return (rc);
2499 }

2501 int
2502 sfxge_rx_scale_count_get(sfxge_t *sp, unsigned int *countp)
2503 {
```

```
2504         sfxge_rx_scale_t *srsp = &(sp->s_rx_scale);
2505         int rc;

2507         mutex_enter(&(srsp->srs_lock));

2509         if (srsp->srs_state != SFXGE_RX_SCALE_INITIALIZED &&
2510             srsp->srs_state != SFXGE_RX_SCALE_STARTED) {
2511                 rc = ENOTSUP;
2512                 goto fail1;
2513         }

2515         *countp = srsp->srs_count;

2517         mutex_exit(&(srsp->srs_lock));

2519         return (0);

2521 fail1:
2522         DTRACE_PROBE1(fail1, int, rc);

2524         mutex_exit(&(srsp->srs_lock));

2526         return (rc);
2527 }

2529 int
2530 sfxge_rx_scale_count_set(sfxge_t *sp, unsigned int count)
2531 {
2532         sfxge_rx_scale_t *srsp = &(sp->s_rx_scale);
2533         sfxge_intr_t *sip = &(sp->s_intr);
2534         int dispatch = 1;
2535         int rc;

2537         if (count < 1 || count > sip->si_nalloc) {
2538                 rc = EINVAL;
2539                 goto fail1;
2540         }

2542         mutex_enter(&(srsp->srs_lock));

2544         if (srsp->srs_state != SFXGE_RX_SCALE_INITIALIZED &&
2545             srsp->srs_state != SFXGE_RX_SCALE_STARTED) {
2546                 rc = ENOTSUP;
2547                 goto fail2;
2548         }

2550         srsp->srs_count = count;

2552         if (srsp->srs_state != SFXGE_RX_SCALE_STARTED)
2553                 dispatch = 0;

2555         mutex_exit(&(srsp->srs_lock));

2557         if (dispatch)
2558                 /* no locks held */
2559                 (void) ddi_taskq_dispatch(sp->s_tqp, sfxge_rx_scale_update, sp,
2560                     DDI_SLEEP);

2562         return (0);

2564 fail2:
2565         DTRACE_PROBE(fail2);

2567         mutex_exit(&(srsp->srs_lock));

2569 fail1:
```



```
2570         DTRACE_PROBE1(fail1, int, rc);

2572         return (rc);
2573 }

2575 static void
2576 sfxge_rx_scale_stop(sfxge_t *sp)
2577 {
2578         sfxge_rx_scale_t *srsp = &(sp->s_rx_scale);
2579         processorid_t id;

2581         mutex_enter(&(srsp->srs_lock));

2583         ASSERT3U(srsp->srs_state, ==, SFXGE_RX_SCALE_STARTED);

2585         srsp->srs_state = SFXGE_RX_SCALE_INITIALIZED;

2587         mutex_enter(&cpu_lock);

2589         /*
2590          * Substract any current CPU, core, cache and chip usage from the
2591          * global contention tables.
2592          */
2593         for (id = 0; id < NCPU; id++) {
2594                 ASSERT3U(sfxge_cpu[id], >=, srsp->srs_cpu[id]);
2595                 sfxge_cpu[id] -= srsp->srs_cpu[id];
2596                 srsp->srs_cpu[id] = 0;

2598 #ifdef  _USE_CPU_PHYSID
2599                 ASSERT3U(sfxge_core[id], >=, srsp->srs_core[id]);
2600                 sfxge_core[id] -= srsp->srs_core[id];
2601                 srsp->srs_core[id] = 0;

2603                 ASSERT3U(sfxge_cache[id], >=, srsp->srs_cache[id]);
2604                 sfxge_cache[id] -= srsp->srs_cache[id];
2605                 srsp->srs_cache[id] = 0;

2607                 ASSERT3U(sfxge_chip[id], >=, srsp->srs_chip[id]);
2608                 sfxge_chip[id] -= srsp->srs_chip[id];
2609                 srsp->srs_chip[id] = 0;
2610 #endif
2611         }

2613         mutex_exit(&cpu_lock);

2615         /* Clear down the RSS table */
2616         bzero(srsp->srs_tbl, sizeof (unsigned int) * SFXGE_RX_SCALE_MAX);

2618         (void) efx_rx_scale_tbl_set(sp->s_enp, srsp->srs_tbl,
2619             SFXGE_RX_SCALE_MAX);

2621         mutex_exit(&(srsp->srs_lock));
2622 }

2624 static void
2625 sfxge_rx_scale_fini(sfxge_t *sp)
2626 {
2627         sfxge_rx_scale_t *srsp = &(sp->s_rx_scale);

2629         ASSERT3U(srsp->srs_state, ==, SFXGE_RX_SCALE_INITIALIZED);

2631         srsp->srs_state = SFXGE_RX_SCALE_UNINITIALIZED;

2633         /* Tear down the kstats */
2634         sfxge_rx_scale_kstat_fini(sp);
```

```
2636         srsp->srs_count = 0;

2638         mutex_destroy(&(srsp->srs_lock));

2640         /* Destroy tables */
2641 #ifdef  _USE_CPU_PHYSID
2642         kmem_free(srsp->srs_chip, sizeof (unsigned int) * NCPU);
2643         srsp->srs_chip = NULL;

2645         kmem_free(srsp->srs_cache, sizeof (unsigned int) * NCPU);
2646         srsp->srs_cache = NULL;

2648         kmem_free(srsp->srs_core, sizeof (unsigned int) * NCPU);
2649         srsp->srs_core = NULL;
2650 #endif
2651         kmem_free(srsp->srs_cpu, sizeof (unsigned int) * NCPU);
2652         srsp->srs_cpu = NULL;
2653 }

2655 int
2656 sfxge_rx_init(sfxge_t *sp)
2657 {
2658         sfxge_intr_t *sip = &(sp->s_intr);
2659         const efx_nic_cfg_t *encp;
2660         char name[MAXNAMELEN];
2661         int index;
2662         int rc;

2664         if (sip->si_state == SFXGE_INTR_UNINITIALIZED) {
2665                 rc = EINVAL;
2666                 goto fail1;
2667         }

2669         encp = efx_nic_cfg_get(sp->s_enp);
2670         if ((rc = sfxge_rx_scale_init(sp)) != 0)
2671                 goto fail2;

2673         (void) snprintf(name, MAXNAMELEN - 1, "%s%d_rx_packet_cache",
2674             ddi_driver_name(sp->s_dip), ddi_get_instance(sp->s_dip));

2676         sp->s_rpc = kmem_cache_create(name, sizeof (sfxge_rx_packet_t),
2677             SFXGE_CPU_CACHE_SIZE, sfxge_rx_packet_ctor, sfxge_rx_packet_dtor,
2678             NULL, sp, NULL, 0);
2679         ASSERT(sp->s_rpc != NULL);

2681         (void) snprintf(name, MAXNAMELEN - 1, "%s%d_rxq_cache",
2682             ddi_driver_name(sp->s_dip), ddi_get_instance(sp->s_dip));

2684         sp->s_rqc = kmem_cache_create(name, sizeof (sfxge_rxq_t),
2685             SFXGE_CPU_CACHE_SIZE, sfxge_rx_qctor, sfxge_rx_qdtor, NULL, sp,
2686             NULL, 0);
2687         ASSERT(sp->s_rqc != NULL);

2689         sp->s_rx_pkt_mem_max = ddi_prop_get_int64(DDI_DEV_T_ANY, sp->s_dip,
2690             DDI_PROP_DONTPASS, "rx_pkt_mem_max", 0); /* disabled */

2692         /* Initialize the receive queue(s) */
2693         for (index = 0; index < sip->si_nalloc; index++) {
2694                 if ((rc = sfxge_rx_qinit(sp, index)) != 0)
2695                         goto fail3;
2696         }

2698         sp->s_rx_coalesce_mode = ddi_prop_get_int(DDI_DEV_T_ANY, sp->s_dip,
2699             DDI_PROP_DONTPASS, "rx_coalesce_mode", SFXGE_RX_COALESCE_OFF);

2701         return (0);
```



```
2703 fail3:
2704         DTRACE_PROBE(fail3);

2706         /* Tear down the receive queue(s) */
2707         while (--index >= 0)
2708                 sfxge_rx_qfini(sp, index);

2710         kmem_cache_destroy(sp->s_rqc);
2711         sp->s_rqc = NULL;

2713         kmem_cache_destroy(sp->s_rpc);
2714         sp->s_rpc = NULL;

2716         sfxge_rx_scale_fini(sp);

2718 fail2:
2719         DTRACE_PROBE(fail2);
2720 fail1:
2721         DTRACE_PROBE1(fail1, int, rc);

2723         return (rc);
2724 }

2726 int
2727 sfxge_rx_start(sfxge_t *sp)
2728 {
2729         sfxge_mac_t *smp = &(sp->s_mac);
2730         sfxge_intr_t *sip;
2731         const efx_nic_cfg_t *encp;
2732         int index;
2733         int rc;

2735         mutex_enter(&(smp->sm_lock));

2737         /* Calculate the receive packet buffer size and alignment */
2738         sp->s_rx_buffer_size = EFX_MAC_PDU(sp->s_mtu);

2740         encp = efx_nic_cfg_get(sp->s_enp);
2741         if (encp->enc_features & EFX_FEATURE_LFSR_HASH_INSERT) {
2742                 size_t align;

2744                 sp->s_rx_prefix_size = EFX_RX_PREFIX_SIZE;

2746                 /*
2747                  * Place the start of the buffer a prefix length minus 2
2748                  * before the start of a cache line. This ensures that the
2749                  * last two bytes of the prefix (which is where the LFSR hash
2750                  * is located) are in the same cache line as the headers, and
2751                  * the IP header is 32-bit aligned.
2752                  */
2753                 align = SFXGE_CPU_CACHE_SIZE + SFXGE_IP_ALIGN -
2754                     EFX_RX_PREFIX_SIZE;

2756                 sp->s_rx_buffer_align = align;
2757                 sp->s_rx_buffer_size += align;
2758         } else {
2759                 sp->s_rx_prefix_size = 0;

2761                 /*
2762                  * Place the start of the buffer 2 bytes after a cache line
2763                  * boundary so that the headers fit into the cache line and
2764                  * the IP header is 32-bit aligned.
2765                  */

2767                 sp->s_rx_buffer_align = SFXGE_IP_ALIGN;
```

```
2768                    sp->s_rx_buffer_size += SFXGE_IP_ALIGN;
2769            }

2771            /* Initialize the receive module */
2772            if ((rc = efx_rx_init(sp->s_enp)) != 0)
2773                    goto fail1;

2775            mutex_exit(&(smp->sm_lock));

2777            if ((rc = sfxge_rx_scale_start(sp)) != 0)
2778                    goto fail2;

2780            /* Start the receive queue(s) */
2781            sip = &(sp->s_intr);
2782            for (index = 0; index < sip->si_nalloc; index++) {
2783                    if ((rc = sfxge_rx_qstart(sp, index)) != 0)
2784                            goto fail3;
2785            }

2787            return (0);

2789 fail3:
2790            DTRACE_PROBE(fail3);

2792            /* Stop the receive queue(s) */
2793            while (--index >= 0)
2794                    sfxge_rx_qstop(sp, index);

2796            sfxge_rx_scale_stop(sp);

2798 fail2:
2799            DTRACE_PROBE(fail2);

2801            mutex_enter(&(smp->sm_lock));

2803            /* Tear down the receive module */
2804            efx_rx_fini(sp->s_enp);

2806 fail1:
2807            DTRACE_PROBE1(fail1, int, rc);

2809            mutex_exit(&(smp->sm_lock));

2811            return (rc);
2812 }

2814 void
2815 sfxge_rx_coalesce_mode_get(sfxge_t *sp, sfxge_rx_coalesce_mode_t *modep)
2816 {
2817            *modep = sp->s_rx_coalesce_mode;
2818 }

2820 int
2821 sfxge_rx_coalesce_mode_set(sfxge_t *sp, sfxge_rx_coalesce_mode_t mode)
2822 {
2823            int rc;

2825            switch (mode) {
2826            case SFXGE_RX_COALESCE_OFF:
2827            case SFXGE_RX_COALESCE_DISALLOW_PUSH:
2828            case SFXGE_RX_COALESCE_ALLOW_PUSH:
2829                    break;

2831            default:
2832                    rc = EINVAL;
2833                    goto fail1;
```



```
2834            }

2836            sp->s_rx_coalesce_mode = mode;

2838            return (0);

2840 fail1:
2841            DTRACE_PROBE1(fail1, int, rc);

2843            return (rc);
2844 }

2846 void
2847 sfxge_rx_loopback(sfxge_t *sp, unsigned int *countp)
2848 {
2849            sfxge_intr_t *sip = &(sp->s_intr);
2850            int index;

2852            *countp = 0;
2853            for (index = 0; index < sip->si_nalloc; index++)
2854                    *countp += sfxge_rx_qloopback(sp, index);
2855 }

2857 int
2858 sfxge_rx_ioctl(sfxge_t *sp, sfxge_rx_ioc_t *srip)
2859 {
2860            int rc;

2862            switch (srip->sri_op) {
2863            case SFXGE_RX_OP_LOOPBACK: {
2864                    unsigned int count;

2866                    sfxge_rx_loopback(sp, &count);

2868                    srip->sri_data = count;

2870                    break;
2871            }
2872            default:
2873                    rc = ENOTSUP;
2874                    goto fail1;
2875            }

2877            return (0);

2879 fail1:
2880            DTRACE_PROBE1(fail1, int, rc);

2882            return (rc);
2883 }

2885 void
2886 sfxge_rx_stop(sfxge_t *sp)
2887 {
2888            sfxge_mac_t *smp = &(sp->s_mac);
2889            sfxge_intr_t *sip = &(sp->s_intr);
2890            efx_nic_t *enp = sp->s_enp;
2891            const efx_nic_cfg_t *encp;
2892            int index;

2894            /* Stop the receive queue(s) */
2895            index = sip->si_nalloc;
2896            while (--index >= 0)
2897                    /* TBD: Flush RXQs in parallel; HW has limit + may need retry */
2898                    sfxge_rx_qstop(sp, index);
```

```
2900         encp = efx_nic_cfg_get(sp->s_enp);
2901         sfxge_rx_scale_stop(sp);

2903         mutex_enter(&(smp->sm_lock));

2905         /* Tear down the receive module */
2906         efx_rx_fini(enp);

2908         sp->s_rx_buffer_align = 0;
2909         sp->s_rx_prefix_size = 0;
2910         sp->s_rx_buffer_size = 0;

2912         mutex_exit(&(smp->sm_lock));
2913 }

2915 unsigned int
2916 sfxge_rx_loaned(sfxge_t *sp)
2917 {
2918         sfxge_intr_t *sip = &(sp->s_intr);
2919         int index;
2920         unsigned int loaned;

2922         ASSERT3U(sip->si_state, ==, SFXGE_INTR_INITIALIZED);

2924         loaned = 0;
2925         for (index = 0; index < sip->si_nalloc; index++) {
2926                 sfxge_rxq_t *srp = sp->s_srp[index];
2927                 sfxge_evq_t *sep = sp->s_sep[srp->sr_index];

2929                 mutex_enter(&(sep->se_lock));

2931                 loaned += sfxge_rx_qfpp_swizzle(srp);

2933                 mutex_exit(&(sep->se_lock));
2934         }

2936         return (loaned);
2937 }

2939 void
2940 sfxge_rx_fini(sfxge_t *sp)
2941 {
2942         sfxge_intr_t *sip = &(sp->s_intr);
2943         const efx_nic_cfg_t *encp;
2944         int index;

2946         ASSERT3U(sip->si_state, ==, SFXGE_INTR_INITIALIZED);

2948         sp->s_rx_coalesce_mode = SFXGE_RX_COALESCE_OFF;

2950         /* Tear down the receive queue(s) */
2951         index = sip->si_nalloc;
2952         while (--index >= 0)
2953                 sfxge_rx_qfini(sp, index);

2955         ASSERT3U(sp->s_rx_pkt_mem_alloc, ==, 0);

2957         kmem_cache_destroy(sp->s_rqc);
2958         sp->s_rqc = NULL;

2960         kmem_cache_destroy(sp->s_rpc);
2961         sp->s_rpc = NULL;

2963         encp = efx_nic_cfg_get(sp->s_enp);
2964         sfxge_rx_scale_fini(sp);
2965 }
```



```
2966 #endif /* ! codereview */
```

```
**********************************************************
    5046 Thu Aug 22 18:59:29 2013
new/usr/src/uts/common/io/sfxge/sfxge_sram.c
Merged sfxge driver
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2008-2013 Solarflare Communications Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#include <sys/types.h>
#include <sys/ddi.h>
#include <sys/sunddi.h>
#include <sys/stream.h>
#include <sys/dlpi.h>

#include "sfxge.h"

void
sfxge_sram_init(sfxge_t *sp)
{
        sfxge_sram_t *ssp = &(sp->s_sram);
        dev_info_t *dip = sp->s_dip;
        char name[MAXNAMELEN];

        ASSERT3U(ssp->ss_state, ==, SFXGE_SRAM_UNINITIALIZED);

        mutex_init(&(ssp->ss_lock), NULL, MUTEX_DRIVER, NULL);

        /*
         * Create a VMEM arena for the buffer table
         * Note that these are not in the DDI/DKI.
         * See "man rmalloc" for a possible alternative
         */
        (void) snprintf(name, MAXNAMELEN - 1, "%s%d_sram", ddi_driver_name(dip),
            ddi_get_instance(dip));
        ssp->ss_buf_tbl = vmem_create(name, (void *)1, EFX_BUF_TBL_SIZE, 1,
            NULL, NULL, NULL, 1, VM_SLEEP | VMC_IDENTIFIER);

        ssp->ss_state = SFXGE_SRAM_INITIALIZED;
}

int
sfxge_sram_buf_tbl_alloc(sfxge_t *sp, size_t n, uint32_t *idp)
{
```



```
        sfxge_sram_t *ssp = &(sp->s_sram);
        void *base;
        int rc;

        mutex_enter(&(ssp->ss_lock));

        ASSERT(ssp->ss_state != SFXGE_SRAM_UNINITIALIZED);

        if ((base = vmem_alloc(ssp->ss_buf_tbl, n, VM_NOSLEEP)) == NULL) {
                rc = ENOSPC;
                goto fail1;
        }

        *idp = (uint32_t)((uintptr_t)base - 1);

        mutex_exit(&(ssp->ss_lock));

        return (0);

fail1:
        DTRACE_PROBE1(fail1, int, rc);

        mutex_exit(&(ssp->ss_lock));

        return (rc);
}

int
sfxge_sram_start(sfxge_t *sp)
{
        sfxge_sram_t *ssp = &(sp->s_sram);

        mutex_enter(&(ssp->ss_lock));

        ASSERT3U(ssp->ss_state, ==, SFXGE_SRAM_INITIALIZED);
        ASSERT3U(ssp->ss_count, ==, 0);

        ssp->ss_state = SFXGE_SRAM_STARTED;

        mutex_exit(&(ssp->ss_lock));

        return (0);
}

int
sfxge_sram_buf_tbl_set(sfxge_t *sp, uint32_t id, efsys_mem_t *esmp,
    size_t n)
{
        sfxge_sram_t *ssp = &(sp->s_sram);
        int rc;

        mutex_enter(&(ssp->ss_lock));

        ASSERT3U(ssp->ss_state, ==, SFXGE_SRAM_STARTED);

        if ((rc = efx_sram_buf_tbl_set(sp->s_enp, id, esmp, n)) != 0)
                goto fail1;

        ssp->ss_count += n;

        mutex_exit(&(ssp->ss_lock));

        return (0);

fail1:
        DTRACE_PROBE1(fail1, int, rc);
```

```
129         mutex_exit(&(ssp->ss_lock));
131         return (rc);
132 }
134 void
135 sfxge_sram_buf_tbl_clear(sfxge_t *sp, uint32_t id, size_t n)
136 {
137         sfxge_sram_t *ssp = &(sp->s_sram);
139         mutex_enter(&(ssp->ss_lock));
141         ASSERT3U(ssp->ss_state, ==, SFXGE_SRAM_STARTED);
143         ASSERT3U(ssp->ss_count, >=, n);
144         ssp->ss_count -= n;
146         efx_sram_buf_tbl_clear(sp->s_enp, id, n);
148         mutex_exit(&(ssp->ss_lock));
149 }
151 void
152 sfxge_sram_stop(sfxge_t *sp)
153 {
154         sfxge_sram_t *ssp = &(sp->s_sram);
156         mutex_enter(&(ssp->ss_lock));
158         ASSERT3U(ssp->ss_state, ==, SFXGE_SRAM_STARTED);
159         ASSERT3U(ssp->ss_count, ==, 0);
161         ssp->ss_state = SFXGE_SRAM_INITIALIZED;
163         mutex_exit(&(ssp->ss_lock));
164 }
166 void
167 sfxge_sram_buf_tbl_free(sfxge_t *sp, uint32_t id, size_t n)
168 {
169         sfxge_sram_t *ssp = &(sp->s_sram);
170         void *base;
172         mutex_enter(&(ssp->ss_lock));
174         ASSERT(ssp->ss_state != SFXGE_SRAM_UNINITIALIZED);
176         base = (void *)((uintptr_t)id + 1);
177         vmem_free(ssp->ss_buf_tbl, base, n);
179         mutex_exit(&(ssp->ss_lock));
180 }
182 static int
183 sfxge_sram_test(sfxge_t *sp, efx_pattern_type_t type)
184 {
185         sfxge_sram_t *ssp = &(sp->s_sram);
186         int rc;
188         if (ssp->ss_state != SFXGE_SRAM_INITIALIZED) {
189                 rc = EFAULT;
190                 goto fail1;
191         }
193         if (type >= EFX_PATTERN_NTYPES) {
```



```
194                 rc = EINVAL;
195                 goto fail2;
196         }
198         mutex_enter(&(ssp->ss_lock));
200         if ((rc = efx_sram_test(sp->s_enp, type)) != 0)
201                 goto fail3;
203         mutex_exit(&(ssp->ss_lock));
205         return (0);
207 fail3:
208         DTRACE_PROBE(fail3);
209         mutex_exit(&(ssp->ss_lock));
210 fail2:
211         DTRACE_PROBE(fail2);
212 fail1:
213         DTRACE_PROBE1(fail1, int, rc);
215         return (rc);
216 }
218 int
219 sfxge_sram_ioctl(sfxge_t *sp, sfxge_sram_ioc_t *ssip)
220 {
221         int rc;
223         switch (ssip->ssi_op) {
224         case SFXGE_SRAM_OP_TEST: {
225                 efx_pattern_type_t type = ssip->ssi_data;
227                 if ((rc = sfxge_sram_test(sp, type)) != 0)
228                         goto fail1;
230                 break;
231         }
232         default:
233                 rc = ENOTSUP;
234                 goto fail1;
235         }
237         return (0);
239 fail1:
240         DTRACE_PROBE1(fail1, int, rc);
242         return (rc);
243 }
245 void
246 sfxge_sram_fini(sfxge_t *sp)
247 {
248         sfxge_sram_t *ssp = &(sp->s_sram);
250         ASSERT3U(ssp->ss_state, ==, SFXGE_SRAM_INITIALIZED);
252         /* Destroy the VMEM arena */
253         vmem_destroy(ssp->ss_buf_tbl);
254         ssp->ss_buf_tbl = NULL;
256         mutex_destroy(&(ssp->ss_lock));
258         ssp->ss_state = SFXGE_SRAM_UNINITIALIZED;
259 }
```

```
260 #endif /* ! codereview */
```

```
**********************************************************
    3311 Thu Aug 22 18:59:29 2013
new/usr/src/uts/common/io/sfxge/sfxge_tcp.c
Merged sfxge driver
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2008-2013 Solarflare Communications Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#include <sys/types.h>
#include <sys/sysmacros.h>
#include <sys/ddi.h>
#include <sys/sunddi.h>
#include <sys/stream.h>
#include <sys/strsun.h>
#include <sys/strsubr.h>
#include <sys/pattr.h>

#include <sys/ethernet.h>
#include <inet/ip.h>

#include <netinet/in.h>
#include <netinet/ip.h>
#include <netinet/tcp.h>

#include "sfxge.h"

#include "efx.h"

void
sfxge_tcp_parse(mblk_t *mp, struct ether_header **etherhpp, struct ip **iphpp,
    struct tcphdr **thpp, size_t *offp, size_t *sizep)
{
        struct ether_header *etherhp;
        uint16_t ether_type;
        size_t etherhs;
        struct ip *iphp;
        size_t iphs;
        struct tcphdr *thp;
        size_t ths;
        size_t off;
        size_t size;

        etherhp = NULL;
```



```
        iphp = NULL;
        thp = NULL;
        off = 0;
        size = 0;

        /* Grab the MAC header */
        if (MBLKL(mp) < sizeof (struct ether_header))
                if (!pullupmsg(mp, sizeof (struct ether_header)))
                        goto done;

        /*LINTED*/
        etherhp = (struct ether_header *)(mp->b_rptr);
        ether_type = etherhp->ether_type;
        etherhs = sizeof (struct ether_header);

        if (ether_type == htons(ETHERTYPE_VLAN)) {
                struct ether_vlan_header *ethervhp;

                if (MBLKL(mp) < sizeof (struct ether_vlan_header))
                        if (!pullupmsg(mp, sizeof (struct ether_vlan_header)))
                                goto done;

                /*LINTED*/
                ethervhp = (struct ether_vlan_header *)(mp->b_rptr);
                ether_type = ethervhp->ether_type;
                etherhs = sizeof (struct ether_vlan_header);
        }

        if (ether_type != htons(ETHERTYPE_IP))
                goto done;

        /* Skip over the MAC header */
        off += etherhs;

        /* Grab the IP header */
        if (MBLKL(mp) < off + sizeof (struct ip))
                if (!pullupmsg(mp, off + sizeof (struct ip)))
                        goto done;

        /*LINTED*/
        iphp = (struct ip *)(mp->b_rptr + off);
        iphs = iphp->ip_hl * 4;

        if (iphp->ip_v != IPV4_VERSION)
                goto done;

        /* Get the size of the packet */
        size = ntohs(iphp->ip_len);

        ASSERT3U(etherhs + size, <=, msgdsize(mp));

        if (iphp->ip_p != IPPROTO_TCP)
                goto done;

        /* Skip over the IP header */
        off += iphs;
        size -= iphs;

        /* Grab the TCP header */
        if (MBLKL(mp) < off + sizeof (struct tcphdr))
                if (!pullupmsg(mp, off + sizeof (struct tcphdr)))
                        goto done;

        /*LINTED*/
        thp = (struct tcphdr *)(mp->b_rptr + off);
        ths = thp->th_off * 4;
```

```
129         /* Skip over the TCP header */
130         off += ths;
131         size -= ths;
132 
133         if (MBLKL(mp) < off)
134                 (void) pullupmsg(mp, off);
135 
136 done:
137         *etherhpp = etherhp;
138         *iphpp = iphp;
139         *thpp = thp;
140         *offp = off;
141         *sizep = size;
142 }
143 #endif /* ! codereview */
```

```
**********************************************************
   65959 Thu Aug 22 18:59:29 2013
new/usr/src/uts/common/io/sfxge/sfxge_tx.c
Merged sfxge driver
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2008-2013 Solarflare Communications Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#include <sys/types.h>
#include <sys/sysmacros.h>
#include <sys/ddi.h>
#include <sys/sunddi.h>
#include <sys/atomic.h>
#include <sys/stream.h>
#include <sys/strsun.h>
#include <sys/strsubr.h>
#include <sys/pattr.h>
#include <sys/cpu.h>

#include <sys/ethernet.h>
#include <inet/ip.h>

#include <netinet/in.h>
#include <netinet/ip.h>
#include <netinet/tcp.h>

#include "sfxge.h"

#include "efx.h"

/* TXQ flush response timeout (in microseconds) */
#define SFXGE_TX_QFLUSH_USEC    (2000000)
#define EVQ_0 0

/* See sfxge.conf.private for descriptions */
#define SFXGE_TX_DPL_GET_PKT_LIMIT_DEFAULT 4096
#define SFXGE_TX_DPL_PUT_PKT_LIMIT_DEFAULT 256


/* Transmit buffer DMA attributes */
static ddi_device_acc_attr_t sfxge_tx_buffer_devacc = {

        DDI_DEVICE_ATTR_V0,     /* devacc_attr_version */
```



```
        DDI_NEVERSWAP_ACC,      /* devacc_attr_endian_flags */
        DDI_STRICTORDER_ACC     /* devacc_attr_dataorder */
};

static ddi_dma_attr_t sfxge_tx_buffer_dma_attr = {
        DMA_ATTR_V0,            /* dma_attr_version     */
        0,                      /* dma_attr_addr_lo     */
        0xffffffffffffffffull,  /* dma_attr_addr_hi     */
        0xffffffffffffffffull,  /* dma_attr_count_max   */
        SFXGE_TX_BUFFER_SIZE,   /* dma_attr_align       */
        0xffffffff,             /* dma_attr_burstsizes  */
        1,                      /* dma_attr_minxfer     */
        0xffffffffffffffffull,  /* dma_attr_maxxfer     */
        0xffffffffffffffffull,  /* dma_attr_seg         */
        1,                      /* dma_attr_sgllen      */
        1,                      /* dma_attr_granular    */
        0                       /* dma_attr_flags       */
};

/* Transmit mapping DMA attributes */
static ddi_dma_attr_t sfxge_tx_mapping_dma_attr = {
        DMA_ATTR_V0,            /* dma_attr_version     */
        0,                      /* dma_attr_addr_lo     */
        0xffffffffffffffffull,  /* dma_attr_addr_hi     */
        0xffffffffffffffffull,  /* dma_attr_count_max   */
        1,                      /* dma_attr_align       */
        0xffffffff,             /* dma_attr_burstsizes  */
        1,                      /* dma_attr_minxfer     */
        0xffffffffffffffffull,  /* dma_attr_maxxfer     */
        0xffffffffffffffffull,  /* dma_attr_seg         */
        0x7fffffff,             /* dma_attr_sgllen      */
        1,                      /* dma_attr_granular    */
        0                       /* dma_attr_flags       */
};

/* Transmit queue DMA attributes */
static ddi_device_acc_attr_t sfxge_txq_devacc = {

        DDI_DEVICE_ATTR_V0,     /* devacc_attr_version */
        DDI_NEVERSWAP_ACC,      /* devacc_attr_endian_flags */
        DDI_STRICTORDER_ACC     /* devacc_attr_dataorder */
};

static ddi_dma_attr_t sfxge_txq_dma_attr = {
        DMA_ATTR_V0,            /* dma_attr_version     */
        0,                      /* dma_attr_addr_lo     */
        0xffffffffffffffffull,  /* dma_attr_addr_hi     */
        0xffffffffffffffffull,  /* dma_attr_count_max   */
        EFX_BUF_SIZE,           /* dma_attr_align       */
        0xffffffff,             /* dma_attr_burstsizes  */
        1,                      /* dma_attr_minxfer     */
        0xffffffffffffffffull,  /* dma_attr_maxxfer     */
        0xffffffffffffffffull,  /* dma_attr_seg         */
        1,                      /* dma_attr_sgllen      */
        1,                      /* dma_attr_granular    */
        0                       /* dma_attr_flags       */
};


/*
 * A sfxge_tx_qdpl_swizzle() can happen when the DPL get list is one packet
 * under the limit, and must move all packets from the DPL put->get list
 * Hence this is the real maximum length of the TX DPL get list.
 */
static int
sfxge_tx_dpl_get_pkt_max(sfxge_txq_t *stp)
```

```
128 {
129         sfxge_tx_dpl_t *stdp = &(stp->st_dpl);
130         return (stdp->get_pkt_limit + stdp->put_pkt_limit - 1);
131 }


134 static int
135 sfxge_tx_packet_ctor(void *buf, void *arg, int kmflags)
136 {
137         _NOTE(ARGUNUSED(arg, kmflags))

139         bzero(buf, sizeof (sfxge_tx_packet_t));

141         return (0);
142 }

144 static void
145 sfxge_tx_packet_dtor(void *buf, void *arg)
146 {
147         sfxge_tx_packet_t *stpp = buf;

149         _NOTE(ARGUNUSED(arg))

151         SFXGE_OBJ_CHECK(stpp, sfxge_tx_packet_t);
152 }

154 static int
155 sfxge_tx_buffer_ctor(void *buf, void *arg, int kmflags)
156 {
157         sfxge_tx_buffer_t *stbp = buf;
158         sfxge_t *sp = arg;
159         sfxge_dma_buffer_attr_t dma_attr;
160         int rc;

162         bzero(buf, sizeof (sfxge_tx_buffer_t));

164         dma_attr.sdba_dip        = sp->s_dip;
165         dma_attr.sdba_dattrp     = &sfxge_tx_buffer_dma_attr;
166         dma_attr.sdba_callback   = ((kmflags == KM_SLEEP) ?
167             DDI_DMA_SLEEP : DDI_DMA_DONTWAIT);
168         dma_attr.sdba_length     = SFXGE_TX_BUFFER_SIZE;
169         dma_attr.sdba_memflags   = DDI_DMA_STREAMING;
170         dma_attr.sdba_devaccp    = &sfxge_tx_buffer_devacc;
171         dma_attr.sdba_bindflags  = DDI_DMA_WRITE | DDI_DMA_STREAMING;
172         dma_attr.sdba_maxcookies = 1;
173         dma_attr.sdba_zeroinit   = B_FALSE;

175         if ((rc = sfxge_dma_buffer_create(&(stbp->stb_esm), &dma_attr)) != 0)
176                 goto fail1;

178         return (0);

180 fail1:
181         DTRACE_PROBE1(fail1, int, rc);

183         SFXGE_OBJ_CHECK(stbp, sfxge_tx_buffer_t);

185         return (-1);
186 }

188 static void
189 sfxge_tx_buffer_dtor(void *buf, void *arg)
190 {
191         sfxge_tx_buffer_t *stbp = buf;

193         _NOTE(ARGUNUSED(arg))
```



```
195         sfxge_dma_buffer_destroy(&(stbp->stb_esm));

197         SFXGE_OBJ_CHECK(stbp, sfxge_tx_buffer_t);
198 }

200 static int
201 sfxge_tx_mapping_ctor(void *buf, void *arg, int kmflags)
202 {
203         sfxge_tx_mapping_t *stmp = buf;
204         sfxge_t *sp = arg;
205         dev_info_t *dip = sp->s_dip;
206         int rc;

208         bzero(buf, sizeof (sfxge_tx_mapping_t));

210         stmp->stm_sp = sp;

212         /* Allocate DMA handle */
213         rc = ddi_dma_alloc_handle(dip, &sfxge_tx_mapping_dma_attr,
214             (kmflags == KM_SLEEP) ? DDI_DMA_SLEEP : DDI_DMA_DONTWAIT,
215             NULL, &(stmp->stm_dma_handle));
216         if (rc != DDI_SUCCESS)
217                 goto fail1;

219         return (0);

221 fail1:
222         DTRACE_PROBE1(fail1, int, rc);

224         stmp->stm_sp = NULL;

226         SFXGE_OBJ_CHECK(stmp, sfxge_tx_mapping_t);

228         return (-1);
229 }

231 static void
232 sfxge_tx_mapping_dtor(void *buf, void *arg)
233 {
234         sfxge_tx_mapping_t *stmp = buf;

236         _NOTE(ARGUNUSED(arg))

238         ASSERT3P(stmp->stm_sp, ==, arg);

240         /* Free the DMA handle */
241         ddi_dma_free_handle(&(stmp->stm_dma_handle));
242         stmp->stm_dma_handle = NULL;

244         stmp->stm_sp = NULL;

246         SFXGE_OBJ_CHECK(stmp, sfxge_tx_mapping_t);
247 }

249 static int
250 sfxge_tx_qctor(void *buf, void *arg, int kmflags)
251 {
252         sfxge_txq_t *stp = buf;
253         efsys_mem_t *esmp = &(stp->st_mem);
254         sfxge_t *sp = arg;
255         sfxge_dma_buffer_attr_t dma_attr;
256         sfxge_tx_dpl_t *stdp;
257         int rc;

259         /* Compile-time structure layout checks */
```

```
260         EFX_STATIC_ASSERT(sizeof (stp->__st_u1.__st_s1) <=
261             sizeof (stp->__st_u1.__st_pad));
262         EFX_STATIC_ASSERT(sizeof (stp->__st_u2.__st_s2) <=
263             sizeof (stp->__st_u2.__st_pad));
264         EFX_STATIC_ASSERT(sizeof (stp->__st_u3.__st_s3) <=
265             sizeof (stp->__st_u3.__st_pad));
266         EFX_STATIC_ASSERT(sizeof (stp->__st_u4.__st_s4) <=
267             sizeof (stp->__st_u4.__st_pad));

269         bzero(buf, sizeof (sfxge_txq_t));

271         stp->st_sp = sp;

273         dma_attr.sdba_dip        = sp->s_dip;
274         dma_attr.sdba_dattrp     = &sfxge_txq_dma_attr;
275         dma_attr.sdba_callback   = DDI_DMA_SLEEP;
276         dma_attr.sdba_length     = EFX_TXQ_SIZE(SFXGE_TX_NDESCS);
277         dma_attr.sdba_memflags   = DDI_DMA_CONSISTENT;
278         dma_attr.sdba_devaccp    = &sfxge_txq_devacc;
279         dma_attr.sdba_bindflags  = DDI_DMA_READ | DDI_DMA_CONSISTENT;
280         dma_attr.sdba_maxcookies = EFX_TXQ_NBUFS(SFXGE_TX_NDESCS);
281         dma_attr.sdba_zeroinit   = B_FALSE;

283         if ((rc = sfxge_dma_buffer_create(esmp, &dma_attr)) != 0)
284                 goto fail1;

286         /* Allocate some buffer table entries */
287         if ((rc = sfxge_sram_buf_tbl_alloc(sp, EFX_TXQ_NBUFS(SFXGE_TX_NDESCS),
288             &(stp->st_id))) != 0)
289                 goto fail2;

291         /* Allocate the descriptor array */
292         if ((stp->st_eb = kmem_zalloc(sizeof (efx_buffer_t) *
293             EFX_TXQ_LIMIT(SFXGE_TX_NDESCS), kmflags)) == NULL) {
294                 rc = ENOMEM;
295                 goto fail3;
296         }

298         /* Allocate the context arrays */
299         if ((stp->st_stmp = kmem_zalloc(sizeof (sfxge_tx_mapping_t *) *
300             SFXGE_TX_NDESCS, kmflags)) == NULL) {
301                 rc = ENOMEM;
302                 goto fail4;
303         }

305         if ((stp->st_stbp = kmem_zalloc(sizeof (sfxge_tx_buffer_t *) *
306             SFXGE_TX_NDESCS, kmflags)) == NULL) {
307                 rc = ENOMEM;
308                 goto fail5;
309         }

311         if ((stp->st_mp = kmem_zalloc(sizeof (mblk_t *) *
312             SFXGE_TX_NDESCS, kmflags)) == NULL) {
313                 rc = ENOMEM;
314                 goto fail6;
315         }

317         /* Initialize the deferred packet list */
318         stdp = &(stp->st_dpl);
319         stdp->std_getp = &(stdp->std_get);

321         stp->st_unblock = SFXGE_TXQ_NOT_BLOCKED;

323         return (0);

325 fail6:
```



```
326         DTRACE_PROBE(fail6);

328         kmem_free(stp->st_stbp, sizeof (sfxge_tx_buffer_t *) * SFXGE_TX_NDESCS);
329         stp->st_stbp = NULL;

331 fail5:
332         DTRACE_PROBE(fail5);

334         kmem_free(stp->st_stmp,
335             sizeof (sfxge_tx_mapping_t *) * SFXGE_TX_NDESCS);
336         stp->st_stmp = NULL;

338 fail4:
339         DTRACE_PROBE(fail4);

341         /* Free the descriptor array */
342         kmem_free(stp->st_eb, sizeof (efx_buffer_t) *
343             EFX_TXQ_LIMIT(SFXGE_TX_NDESCS));
344         stp->st_eb = NULL;

346 fail3:
347         DTRACE_PROBE(fail3);

349         /* Free the buffer table entries */
350         sfxge_sram_buf_tbl_free(sp, stp->st_id, EFX_TXQ_NBUFS(SFXGE_TX_NDESCS));
351         stp->st_id = 0;

353 fail2:
354         DTRACE_PROBE(fail2);

356         /* Tear down DMA setup */
357         sfxge_dma_buffer_destroy(esmp);

359 fail1:
360         DTRACE_PROBE1(fail1, int, rc);

362         stp->st_sp = NULL;

364         SFXGE_OBJ_CHECK(stp, sfxge_txq_t);

366         return (-1);
367 }

369 static void
370 sfxge_tx_qdtor(void *buf, void *arg)
371 {
372         sfxge_txq_t *stp = buf;
373         efsys_mem_t *esmp = &(stp->st_mem);
374         sfxge_t *sp = stp->st_sp;
375         sfxge_tx_dpl_t *stdp;

377         _NOTE(ARGUNUSED(arg))

379         stp->st_unblock = 0;

381         /* Tear down the deferred packet list */
382         stdp = &(stp->st_dpl);
383         ASSERT3P(stdp->std_getp, ==, &(stdp->std_get));
384         stdp->std_getp = NULL;

386         /* Free the context arrays */
387         kmem_free(stp->st_mp, sizeof (mblk_t *) * SFXGE_TX_NDESCS);
388         stp->st_mp = NULL;

390         kmem_free(stp->st_stbp, sizeof (sfxge_tx_buffer_t *) * SFXGE_TX_NDESCS);
391         stp->st_stbp = NULL;
```

```
393         kmem_free(stp->st_stmp,
394             sizeof (sfxge_tx_mapping_t *) * SFXGE_TX_NDESCS);
395         stp->st_stmp = NULL;

397         /* Free the descriptor array */
398         kmem_free(stp->st_eb, sizeof (efx_buffer_t) *
399             EFX_TXQ_LIMIT(SFXGE_TX_NDESCS));
400         stp->st_eb = NULL;

402         /* Free the buffer table entries */
403         sfxge_sram_buf_tbl_free(sp, stp->st_id, EFX_TXQ_NBUFS(SFXGE_TX_NDESCS));
404         stp->st_id = 0;

406         /* Tear down dma setup */
407         sfxge_dma_buffer_destroy(esmp);

409         stp->st_sp = NULL;

411         SFXGE_OBJ_CHECK(stp, sfxge_txq_t);
412 }

414 static void
415 sfxge_tx_packet_destroy(sfxge_t *sp, sfxge_tx_packet_t *stpp)
416 {
417         kmem_cache_free(sp->s_tpc, stpp);
418 }

420 static sfxge_tx_packet_t *
421 sfxge_tx_packet_create(sfxge_t *sp)
422 {
423         sfxge_tx_packet_t *stpp;

425         stpp = kmem_cache_alloc(sp->s_tpc, KM_NOSLEEP);

427         return (stpp);
428 }

430 static inline int
431 sfxge_tx_qfpp_put(sfxge_txq_t *stp, sfxge_tx_packet_t *stpp)
432 {
433         sfxge_tx_fpp_t *stfp = &(stp->st_fpp);

435         ASSERT(mutex_owned(&(stp->st_lock)));

437         ASSERT3P(stpp->stp_next, ==, NULL);
438         ASSERT3P(stpp->stp_mp, ==, NULL);
439         ASSERT3P(stpp->stp_etherhp, ==, NULL);
440         ASSERT3P(stpp->stp_iphp, ==, NULL);
441         ASSERT3P(stpp->stp_thp, ==, NULL);
442         ASSERT3U(stpp->stp_off, ==, 0);
443         ASSERT3U(stpp->stp_size, ==, 0);
444         ASSERT3U(stpp->stp_mss, ==, 0);
445         ASSERT3U(stpp->stp_dpl_put_len, ==, 0);

447         if (stfp->stf_count < SFXGE_TX_FPP_MAX) {
448                 /* Add to the start of the list */
449                 stpp->stp_next = stfp->stf_stpp;
450                 stfp->stf_stpp = stpp;
451                 stfp->stf_count++;

453                 return (0);
454         }

456         DTRACE_PROBE(fpp_full);
457         return (ENOSPC);
```



```
458 }

460 static inline sfxge_tx_packet_t *
461 sfxge_tx_qfpp_get(sfxge_txq_t *stp)
462 {
463         sfxge_tx_packet_t *stpp;
464         sfxge_tx_fpp_t *stfp = &(stp->st_fpp);

466         ASSERT(mutex_owned(&(stp->st_lock)));

468         stpp = stfp->stf_stpp;
469         if (stpp == NULL) {
470                 ASSERT3U(stfp->stf_count, ==, 0);
471                 return (NULL);
472         }

474         /* Remove item from the head of the list */
475         stfp->stf_stpp = stpp->stp_next;
476         stpp->stp_next = NULL;

478         ASSERT3U(stfp->stf_count, >, 0);
479         stfp->stf_count--;

481         if (stfp->stf_count != 0) {
482                 ASSERT(stfp->stf_stpp != NULL);
483                 prefetch_read_many(stfp->stf_stpp);
484         }
485         return (stpp);
486 }

488 static void
489 sfxge_tx_qfpp_empty(sfxge_txq_t *stp)
490 {
491         sfxge_t *sp = stp->st_sp;
492         sfxge_tx_fpp_t *stfp = &(stp->st_fpp);
493         sfxge_tx_packet_t *stpp;

495         mutex_enter(&(stp->st_lock));

497         stpp = stfp->stf_stpp;
498         stfp->stf_stpp = NULL;

500         while (stpp != NULL) {
501                 sfxge_tx_packet_t *next;

503                 next = stpp->stp_next;
504                 stpp->stp_next = NULL;

506                 ASSERT3U(stfp->stf_count, >, 0);
507                 stfp->stf_count--;

509                 sfxge_tx_packet_destroy(sp, stpp);

511                 stpp = next;
512         }
513         ASSERT3U(stfp->stf_count, ==, 0);

515         mutex_exit(&(stp->st_lock));
516 }

518 static inline void
519 sfxge_tx_qfbp_put(sfxge_txq_t *stp, sfxge_tx_buffer_t *stbp)
520 {
521         sfxge_tx_fbp_t *stfp = &(stp->st_fbp);

523         ASSERT3P(stbp->stb_next, ==, NULL);
```

```
524         ASSERT3U(stbp->stb_off, ==, 0);
525         ASSERT3U(stbp->stb_esm.esm_size, ==, 0);

527         stbp->stb_next = stfp->stf_stbp;
528         stfp->stf_stbp = stbp;
529         stfp->stf_count++;
530 }


533 static inline sfxge_tx_buffer_t *
534 sfxge_tx_qfbp_get(sfxge_txq_t *stp)
535 {
536         sfxge_tx_buffer_t *stbp;
537         sfxge_tx_fbp_t *stfp = &(stp->st_fbp);

539         stbp = stfp->stf_stbp;
540         if (stbp == NULL) {
541                 ASSERT3U(stfp->stf_count, ==, 0);
542                 return (NULL);
543         }

545         stfp->stf_stbp = stbp->stb_next;
546         stbp->stb_next = NULL;

548         ASSERT3U(stfp->stf_count, >, 0);
549         stfp->stf_count--;

551         if (stfp->stf_count != 0) {
552                 ASSERT(stfp->stf_stbp != NULL);
553                 prefetch_read_many(stfp->stf_stbp);
554         }

556         return (stbp);
557 }

559 static void
560 sfxge_tx_qfbp_empty(sfxge_txq_t *stp)
561 {
562         sfxge_t *sp = stp->st_sp;
563         sfxge_tx_fbp_t *stfp = &(stp->st_fbp);
564         sfxge_tx_buffer_t *stbp;

566         mutex_enter(&(stp->st_lock));

568         stbp = stfp->stf_stbp;
569         stfp->stf_stbp = NULL;

571         while (stbp != NULL) {
572                 sfxge_tx_buffer_t *next;

574                 next = stbp->stb_next;
575                 stbp->stb_next = NULL;

577                 ASSERT3U(stfp->stf_count, >, 0);
578                 stfp->stf_count--;

580                 kmem_cache_free(sp->s_tbc, stbp);

582                 stbp = next;
583         }
584         ASSERT3U(stfp->stf_count, ==, 0);

586         mutex_exit(&(stp->st_lock));
587 }

589 static inline void
```



```
590 sfxge_tx_qfmp_put(sfxge_txq_t *stp, sfxge_tx_mapping_t *stmp)
591 {
592         sfxge_tx_fmp_t *stfp = &(stp->st_fmp);

594         ASSERT3P(stmp->stm_next, ==, NULL);
595         ASSERT3P(stmp->stm_mp, ==, NULL);
596         ASSERT3P(stmp->stm_base, ==, NULL);
597         ASSERT3U(stmp->stm_off, ==, 0);
598         ASSERT3U(stmp->stm_size, ==, 0);

600         stmp->stm_next = stfp->stf_stmp;
601         stfp->stf_stmp = stmp;
602         stfp->stf_count++;
603 }

605 static inline sfxge_tx_mapping_t *
606 sfxge_tx_qfmp_get(sfxge_txq_t *stp)
607 {
608         sfxge_tx_mapping_t *stmp;
609         sfxge_tx_fmp_t *stfp = &(stp->st_fmp);

611         stmp = stfp->stf_stmp;
612         if (stmp == NULL) {
613                 ASSERT3U(stfp->stf_count, ==, 0);
614                 return (NULL);
615         }

617         stfp->stf_stmp = stmp->stm_next;
618         stmp->stm_next = NULL;

620         ASSERT3U(stfp->stf_count, >, 0);
621         stfp->stf_count--;

623         if (stfp->stf_count != 0) {
624                 ASSERT(stfp->stf_stmp != NULL);
625                 prefetch_read_many(stfp->stf_stmp);
626         }
627         return (stmp);
628 }

630 static void
631 sfxge_tx_qfmp_empty(sfxge_txq_t *stp)
632 {
633         sfxge_t *sp = stp->st_sp;
634         sfxge_tx_fmp_t *stfp = &(stp->st_fmp);
635         sfxge_tx_mapping_t *stmp;

637         mutex_enter(&(stp->st_lock));

639         stmp = stfp->stf_stmp;
640         stfp->stf_stmp = NULL;

642         while (stmp != NULL) {
643                 sfxge_tx_mapping_t *next;

645                 next = stmp->stm_next;
646                 stmp->stm_next = NULL;

648                 ASSERT3U(stfp->stf_count, >, 0);
649                 stfp->stf_count--;

651                 kmem_cache_free(sp->s_tmc, stmp);

653                 stmp = next;
654         }
655         ASSERT3U(stfp->stf_count, ==, 0);
```

```
657         mutex_exit(&(stp->st_lock));
658 }

660 static void
661 sfxge_tx_msgb_unbind(sfxge_tx_mapping_t *stmp)
662 {
663         bzero(stmp->stm_addr, sizeof (uint64_t) * SFXGE_TX_MAPPING_NADDR);
664         stmp->stm_off = 0;

666         (void) ddi_dma_unbind_handle(stmp->stm_dma_handle);

668         stmp->stm_size = 0;
669         stmp->stm_base = NULL;

671         stmp->stm_mp = NULL;
672 }

674 #define SFXGE_TX_DESCSHIFT      12
675 #define SFXGE_TX_DESCSIZE       (1 << 12)

677 #define SFXGE_TX_DESCOFFSET     (SFXGE_TX_DESCSIZE - 1)
678 #define SFXGE_TX_DESCMASK       (~SFXGE_TX_DESCOFFSET)

680 static int
681 sfxge_tx_msgb_bind(mblk_t *mp, sfxge_tx_mapping_t *stmp)
682 {
683         ddi_dma_cookie_t dmac;
684         unsigned int ncookies;
685         size_t size;
686         unsigned int n;
687         int rc;

689         ASSERT(mp != NULL);
690         ASSERT3U(DB_TYPE(mp), ==, M_DATA);

692         ASSERT(stmp->stm_mp == NULL);
693         stmp->stm_mp = mp;

695         stmp->stm_base = (caddr_t)(mp->b_rptr);
696         stmp->stm_size = MBLKL(mp);

698         /* Bind the STREAMS block to the mapping */
699         rc = ddi_dma_addr_bind_handle(stmp->stm_dma_handle, NULL,
700             stmp->stm_base, stmp->stm_size, DDI_DMA_WRITE | DDI_DMA_STREAMING,
701             DDI_DMA_DONTWAIT, NULL, &dmac, &ncookies);
702         if (rc != DDI_DMA_MAPPED)
703                 goto fail1;

705         ASSERT3U(ncookies, <=, SFXGE_TX_MAPPING_NADDR);

707         /*
708          * Construct an array of addresses and an initial
709          * offset.
710          */
711         n = 0;
712         stmp->stm_addr[n++] = dmac.dmac_laddress & SFXGE_TX_DESCMASK;
713         DTRACE_PROBE1(addr, uint64_t, dmac.dmac_laddress & SFXGE_TX_DESCMASK);

715         stmp->stm_off = dmac.dmac_laddress & SFXGE_TX_DESCOFFSET;

717         size = MIN(SFXGE_TX_DESCSIZE - stmp->stm_off, dmac.dmac_size);
718         dmac.dmac_laddress += size;
719         dmac.dmac_size -= size;

721         for (;;) {
```



```
722                 ASSERT3U(n, <, SFXGE_TX_MAPPING_NADDR);

724                 if (dmac.dmac_size == 0) {
725                         if (--ncookies == 0)
726                                 break;

728                         ddi_dma_nextcookie(stmp->stm_dma_handle, &dmac);
729                 }

731                 ASSERT((dmac.dmac_laddress & SFXGE_TX_DESCMASK) != 0);
732                 ASSERT((dmac.dmac_laddress & SFXGE_TX_DESCOFFSET) == 0);
733                 stmp->stm_addr[n++] = dmac.dmac_laddress;
734                 DTRACE_PROBE1(addr, uint64_t, dmac.dmac_laddress);

736                 size = MIN(SFXGE_TX_DESCSIZE, dmac.dmac_size);
737                 dmac.dmac_laddress += size;
738                 dmac.dmac_size -= size;
739         }
740         ASSERT3U(n, <=, SFXGE_TX_MAPPING_NADDR);

742         return (0);

744 fail1:
745         DTRACE_PROBE1(fail1, int, rc);

747         stmp->stm_size = 0;
748         stmp->stm_base = NULL;

750         stmp->stm_mp = NULL;

752         return (-1);
753 }

755 static void
756 sfxge_tx_qreap(sfxge_txq_t *stp)
757 {
758         unsigned int reaped;

760         ASSERT(mutex_owned(&(stp->st_lock)));

762         reaped = stp->st_reaped;
763         while (reaped != stp->st_completed) {
764                 unsigned int id;
765                 sfxge_tx_mapping_t *stmp;
766                 sfxge_tx_buffer_t *stbp;

768                 id = reaped++ & (SFXGE_TX_NDESCS - 1);

770                 ASSERT3P(stp->st_mp[id], ==, NULL);

772                 if ((stmp = stp->st_stmp[id]) != NULL) {
773                         stp->st_stmp[id] = NULL;

775                         /* Free all the mappings */
776                         do {
777                                 sfxge_tx_mapping_t *next;

779                                 next = stmp->stm_next;
780                                 stmp->stm_next = NULL;

782                                 sfxge_tx_qfmp_put(stp, stmp);

784                                 stmp = next;
785                         } while (stmp != NULL);
786                 }
```

```
788                 if ((stbp = stp->st_stbp[id]) != NULL) {
789                         stp->st_stbp[id] = NULL;
790
791                         /* Free all the buffers */
792                         do {
793                                 sfxge_tx_buffer_t *next;
794
795                                 next = stbp->stb_next;
796                                 stbp->stb_next = NULL;
797
798                                 stbp->stb_esm.esm_size = 0;
799                                 stbp->stb_off = 0;
800
801                                 sfxge_tx_qfbp_put(stp, stbp);
802
803                                 stbp = next;
804                         } while (stbp != NULL);
805                 }
806         }
807         stp->st_reaped = reaped;
808 }
809
810 static void
811 sfxge_tx_qlist_abort(sfxge_txq_t *stp)
812 {
813         unsigned int id;
814         sfxge_tx_mapping_t *stmp;
815         sfxge_tx_buffer_t *stbp;
816         mblk_t *mp;
817
818         ASSERT(mutex_owned(&(stp->st_lock)));
819
820         id = stp->st_added & (SFXGE_TX_NDESCS - 1);
821
822         /* Clear the completion information */
823         stmp = stp->st_stmp[id];
824         stp->st_stmp[id] = NULL;
825
826         /* Free any mappings that were used */
827         while (stmp != NULL) {
828                 sfxge_tx_mapping_t *next;
829
830                 next = stmp->stm_next;
831                 stmp->stm_next = NULL;
832
833                 if (stmp->stm_mp != NULL)
834                         sfxge_tx_msgb_unbind(stmp);
835
836                 sfxge_tx_qfmp_put(stp, stmp);
837
838                 stmp = next;
839         }
840
841         stbp = stp->st_stbp[id];
842         stp->st_stbp[id] = NULL;
843
844         /* Free any buffers that were used */
845         while (stbp != NULL) {
846                 sfxge_tx_buffer_t *next;
847
848                 next = stbp->stb_next;
849                 stbp->stb_next = NULL;
850
851                 stbp->stb_off = 0;
852                 stbp->stb_esm.esm_size = 0;
```



```
853
854                 sfxge_tx_qfbp_put(stp, stbp);
855
856                 stbp = next;
857         }
858
859         mp = stp->st_mp[id];
860         stp->st_mp[id] = NULL;
861
862         if (mp != NULL)
863                 freemsg(mp);
864
865         /* Clear the fragment list */
866         stp->st_n = 0;
867 }
868
869 /* Push descriptors to the TX ring setting blocked if no space */
870 static void
871 sfxge_tx_qlist_post(sfxge_txq_t *stp)
872 {
873         unsigned int id;
874         unsigned int level;
875         unsigned int available;
876         int rc;
877
878         ASSERT(mutex_owned(&(stp->st_lock)));
879
880         ASSERT(stp->st_n != 0);
881
882 again:
883         level = stp->st_added - stp->st_reaped;
884         available = EFX_TXQ_LIMIT(SFXGE_TX_NDESCS) - level;
885
886         id = stp->st_added & (SFXGE_TX_NDESCS - 1);
887
888         if (available < stp->st_n) {
889                 rc = ENOSPC;
890                 goto fail1;
891         }
892
893         ASSERT3U(available, >=, stp->st_n);
894
895         /* Post the fragment list */
896         if ((rc = efx_tx_qpost(stp->st_etp, stp->st_eb, stp->st_n,
897             stp->st_reaped, &(stp->st_added))) != 0)
898                 goto fail2;
899
900         /*
901          * If the list took more than a single descriptor then we need to
902          * to move the completion information so it is referenced by the last
903          * descriptor.
904          */
905         if (((stp->st_added - 1) & (SFXGE_TX_NDESCS - 1)) != id) {
906                 sfxge_tx_mapping_t *stmp;
907                 sfxge_tx_buffer_t *stbp;
908                 mblk_t *mp;
909
910                 stmp = stp->st_stmp[id];
911                 stp->st_stmp[id] = NULL;
912
913                 stbp = stp->st_stbp[id];
914                 stp->st_stbp[id] = NULL;
915
916                 mp = stp->st_mp[id];
917                 stp->st_mp[id] = NULL;
918
919                 id = (stp->st_added - 1) & (SFXGE_TX_NDESCS - 1);
```

```
921                 ASSERT(stp->st_stmp[id] == NULL);
922                 stp->st_stmp[id] = stmp;
923
924                 ASSERT(stp->st_stbp[id] == NULL);
925                 stp->st_stbp[id] = stbp;
926
927                 ASSERT(stp->st_mp[id] == NULL);
928                 stp->st_mp[id] = mp;
929         }
930
931         /* Make the descriptors visible to the hardware */
932         (void) ddi_dma_sync(stp->st_mem.esm_dma_handle,
933             0,
934             EFX_TXQ_SIZE(SFXGE_TX_NDESCS),
935             DDI_DMA_SYNC_FORDEV);
936
937         /* Clear the list */
938         stp->st_n = 0;
939
940         ASSERT3U(stp->st_unblock, ==, SFXGE_TXQ_NOT_BLOCKED);
941         return;
942
943 fail2:
944         DTRACE_PROBE(fail2);
945 fail1:
946         DTRACE_PROBE1(fail1, int, rc);
947
948         ASSERT(rc == ENOSPC);
949
950         level = stp->st_added - stp->st_completed;
951         available = EFX_TXQ_LIMIT(SFXGE_TX_NDESCS) - level;
952
953         /*
954          * If there would be enough space after we've reaped any completed
955          * mappings and buffers, and we gain sufficient queue space by doing
956          * so, then reap now and try posting again.
957          */
958         if (stp->st_n <= available &&
959             stp->st_completed - stp->st_reaped >= SFXGE_TX_BATCH) {
960                 sfxge_tx_qreap(stp);
961
962                 goto again;
963         }
964
965         /* Set the unblock level */
966         if (stp->st_unblock == SFXGE_TXQ_NOT_BLOCKED) {
967                 stp->st_unblock = SFXGE_TXQ_UNBLOCK_LEVEL1;
968         } else {
969                 ASSERT(stp->st_unblock == SFXGE_TXQ_UNBLOCK_LEVEL1);
970
971                 stp->st_unblock = SFXGE_TXQ_UNBLOCK_LEVEL2;
972         }
973
974         /*
975          * Avoid a race with completion interrupt handling that could leave the
976          * queue blocked.
977          *
978          * NOTE: The use of st_pending rather than st_completed is intentional
979          *       as st_pending is updated per-event rather than per-batch and
980          *       therefore avoids needless deferring.
981          */
982         if (stp->st_pending == stp->st_added) {
983                 sfxge_tx_qreap(stp);
984
985                 stp->st_unblock = SFXGE_TXQ_NOT_BLOCKED;
```



```
986                 goto again;
987         }
988
989         ASSERT(stp->st_unblock != SFXGE_TXQ_NOT_BLOCKED);
990 }
991
992 static int
993 sfxge_tx_kstat_update(kstat_t *ksp, int rw)
994 {
995         sfxge_txq_t *stp = ksp->ks_private;
996         sfxge_tx_dpl_t *stdp = &(stp->st_dpl);
997         kstat_named_t *knp;
998         int rc;
999
1000        ASSERT(mutex_owned(&(stp->st_lock)));
1001
1002        if (rw != KSTAT_READ) {
1003                rc = EACCES;
1004                goto fail1;
1005        }
1006
1007        if (stp->st_state != SFXGE_TXQ_STARTED)
1008                goto done;
1009
1010        efx_tx_qstats_update(stp->st_etp, stp->st_stat);
1011        knp = (kstat_named_t *)ksp->ks_data + TX_NQSTATS;
1012        knp->value.ui64 = stdp->get_pkt_limit;
1013        knp++;
1014        knp->value.ui64 = stdp->put_pkt_limit;
1015        knp++;
1016        knp->value.ui64 = stdp->get_full_count;
1017        knp++;
1018        knp->value.ui64 = stdp->put_full_count;
1019
1020 done:
1021        return (0);
1022
1023 fail1:
1024        DTRACE_PROBE1(fail1, int, rc);
1025
1026        return (rc);
1027 }
1028
1029 static int
1030 sfxge_tx_kstat_init(sfxge_txq_t *stp)
1031 {
1032        sfxge_t *sp = stp->st_sp;
1033        unsigned int index = stp->st_index;
1034        dev_info_t *dip = sp->s_dip;
1035        kstat_t *ksp;
1036        kstat_named_t *knp;
1037        char name[MAXNAMELEN];
1038        unsigned int id;
1039        int rc;
1040
1041        /* Create the set */
1042        (void) snprintf(name, MAXNAMELEN - 1, "%s_txq%04d",
1043            ddi_driver_name(dip), index);
1044
1045        if ((ksp = kstat_create((char *)ddi_driver_name(dip),
1046            ddi_get_instance(dip), name, "queue", KSTAT_TYPE_NAMED,
1047            TX_NQSTATS + 4, 0)) == NULL) {
1048                rc = ENOMEM;
1049                goto fail1;
1050        }
```

```
1052         stp->st_ksp = ksp;

1054         ksp->ks_update = sfxge_tx_kstat_update;
1055         ksp->ks_private = stp;
1056         ksp->ks_lock = &(stp->st_lock);

1058         /* Initialise the named stats */
1059         stp->st_stat = knp = ksp->ks_data;
1060         for (id = 0; id < TX_NQSTATS; id++) {
1061                 kstat_named_init(knp, (char *)efx_tx_qstat_name(sp->s_enp, id),
1062                     KSTAT_DATA_UINT64);
1063                 knp++;
1064         }
1065         kstat_named_init(knp, "dpl_get_pkt_limit", KSTAT_DATA_UINT64);
1066         knp++;
1067         kstat_named_init(knp, "dpl_put_pkt_limit", KSTAT_DATA_UINT64);
1068         knp++;
1069         kstat_named_init(knp, "dpl_get_full_count", KSTAT_DATA_UINT64);
1070         knp++;
1071         kstat_named_init(knp, "dpl_put_full_count", KSTAT_DATA_UINT64);

1073         kstat_install(ksp);
1074         return (0);

1076 fail1:
1077         DTRACE_PROBE1(fail1, int, rc);

1079         return (rc);
1080 }

1082 static void
1083 sfxge_tx_kstat_fini(sfxge_txq_t *stp)
1084 {
1085         /* Destroy the set */
1086         kstat_delete(stp->st_ksp);
1087         stp->st_ksp = NULL;
1088         stp->st_stat = NULL;
1089 }

1091 static int
1092 sfxge_tx_qinit(sfxge_t *sp, unsigned int index, sfxge_txq_type_t type,
1093     unsigned int evq)
1094 {
1095         sfxge_txq_t *stp;
1096         sfxge_tx_dpl_t *stdp;
1097         int rc;

1099         ASSERT3U(index, <, SFXGE_TXQ_NTYPES + SFXGE_RX_SCALE_MAX);
1100         ASSERT3U(type, <, SFXGE_TXQ_NTYPES);
1101         ASSERT3U(evq, <, SFXGE_RX_SCALE_MAX);

1103         stp = kmem_cache_alloc(sp->s_tqc, KM_SLEEP);
1104         stdp = &(stp->st_dpl);

1106         ASSERT3U(stp->st_state, ==, SFXGE_TXQ_UNINITIALIZED);

1108         stp->st_index = index;
1109         stp->st_type = type;
1110         stp->st_evq = evq;

1112         mutex_init(&(stp->st_lock), NULL, MUTEX_DRIVER,
1113             DDI_INTR_PRI(sp->s_intr.si_intr_pri));

1115         /* Initialize the statistics */
1116         if ((rc = sfxge_tx_kstat_init(stp)) != 0)
1117                 goto fail1;
```



```
1119         stdp->get_pkt_limit = ddi_prop_get_int(DDI_DEV_T_ANY, sp->s_dip,
1120             DDI_PROP_DONTPASS, "tx_dpl_get_pkt_limit",
1121             SFXGE_TX_DPL_GET_PKT_LIMIT_DEFAULT);

1123         stdp->put_pkt_limit = ddi_prop_get_int(DDI_DEV_T_ANY, sp->s_dip,
1124             DDI_PROP_DONTPASS, "tx_dpl_put_pkt_limit",
1125             SFXGE_TX_DPL_PUT_PKT_LIMIT_DEFAULT);

1127         stp->st_state = SFXGE_TXQ_INITIALIZED;

1129         /* Attach the TXQ to the driver */
1130         ASSERT3P(sp->s_stp[index], ==, NULL);
1131         sp->s_stp[index] = stp;
1132         sp->s_tx_qcount++;

1134         return (0);

1136 fail1:
1137         DTRACE_PROBE1(fail1, int, rc);

1139         stp->st_evq = 0;
1140         stp->st_type = 0;
1141         stp->st_index = 0;

1143         mutex_destroy(&(stp->st_lock));

1145         kmem_cache_free(sp->s_tqc, stp);

1147         return (rc);
1148 }

1150 static int
1151 sfxge_tx_qstart(sfxge_t *sp, unsigned int index)
1152 {
1153         sfxge_txq_t *stp = sp->s_stp[index];
1154         efx_nic_t *enp = sp->s_enp;
1155         efsys_mem_t *esmp;
1156         sfxge_evq_t *sep;
1157         unsigned int evq;
1158         unsigned int flags;
1159         int rc;

1161         mutex_enter(&(stp->st_lock));

1163         esmp = &(stp->st_mem);
1164         evq = stp->st_evq;
1165         sep = sp->s_sep[evq];

1167         ASSERT3U(stp->st_state, ==, SFXGE_TXQ_INITIALIZED);
1168         ASSERT3U(sep->se_state, ==, SFXGE_EVQ_STARTED);

1170         /* Zero the memory */
1171         (void) memset(esmp->esm_base, 0, EFX_TXQ_SIZE(SFXGE_TX_NDESCS));

1173         /* Program the buffer table */
1174         if ((rc = sfxge_sram_buf_tbl_set(sp, stp->st_id, esmp,
1175             EFX_TXQ_NBUFS(SFXGE_TX_NDESCS))) != 0)
1176                 goto fail1;

1178         switch (stp->st_type) {
1179         case SFXGE_TXQ_NON_CKSUM:
1180                 flags = 0;
1181                 break;

1183         case SFXGE_TXQ_IP_CKSUM:
```

```
1184                 flags = EFX_CKSUM_IPV4;
1185                 break;

1187         case SFXGE_TXQ_IP_TCP_UDP_CKSUM:
1188                 flags = EFX_CKSUM_IPV4 | EFX_CKSUM_TCPUDP;
1189                 break;

1191         default:
1192                 ASSERT(B_FALSE);

1194                 flags = 0;
1195                 break;
1196         }

1198         /* Create the transmit queue */
1199         if ((rc = efx_tx_qcreate(enp, index, index, esmp, SFXGE_TX_NDESCS,
1200             stp->st_id, flags, sep->se_eep, &(stp->st_etp))) != 0)
1201                 goto fail2;

1203         /* Enable the transmit queue */
1204         efx_tx_qenable(stp->st_etp);

1206         stp->st_state = SFXGE_TXQ_STARTED;

1208         mutex_exit(&(stp->st_lock));

1210         return (0);

1212 fail2:
1213         DTRACE_PROBE(fail2);

1215         /* Clear entries from the buffer table */
1216         sfxge_sram_buf_tbl_clear(sp, stp->st_id,
1217             EFX_TXQ_NBUFS(SFXGE_TX_NDESCS));

1219 fail1:
1220         DTRACE_PROBE1(fail1, int, rc);

1222         mutex_exit(&(stp->st_lock));

1224         return (rc);
1225 }

1227 static inline int
1228 sfxge_tx_qmapping_add(sfxge_txq_t *stp, sfxge_tx_mapping_t *stmp,
1229 size_t *offp, size_t *limitp)
1230 {
1231         mblk_t *mp;
1232         size_t mapping_off;
1233         size_t mapping_size;
1234         int rc;

1236         ASSERT3U(*offp, <, stmp->stm_size);
1237         ASSERT(*limitp != 0);

1239         mp = stmp->stm_mp;

1241         ASSERT3P(stmp->stm_base, ==, mp->b_rptr);
1242         ASSERT3U(stmp->stm_size, ==, MBLKL(mp));

1244         mapping_off = stmp->stm_off + *offp;
1245         mapping_size = stmp->stm_size - *offp;

1247         while (mapping_size != 0 && *limitp != 0) {
1248                 size_t page =
1249                     mapping_off >> SFXGE_TX_DESCSHIFT;
```



```
1250                 size_t page_off =
1251                     mapping_off & SFXGE_TX_DESCOFFSET;
1252                 size_t page_size =
1253                     SFXGE_TX_DESCSIZE - page_off;
1254                 efx_buffer_t *ebp;

1256                 ASSERT3U(page, <, SFXGE_TX_MAPPING_NADDR);
1257                 ASSERT((stmp->stm_addr[page] &
1258                         SFXGE_TX_DESCMASK) != 0);

1260                 page_size = MIN(page_size, mapping_size);
1261                 page_size = MIN(page_size, *limitp);

1263                 ASSERT3U(stp->st_n, <=,
1264                     EFX_TXQ_LIMIT(SFXGE_TX_NDESCS));
1265                 if (stp->st_n ==
1266                     EFX_TXQ_LIMIT(SFXGE_TX_NDESCS)) {
1267                         rc = ENOSPC;
1268                         goto fail1;
1269                 }

1271                 ebp = &(stp->st_eb[stp->st_n++]);
1272                 ebp->eb_addr = stmp->stm_addr[page] +
1273                     page_off;
1274                 ebp->eb_size = page_size;

1276                 *offp += page_size;
1277                 *limitp -= page_size;

1279                 mapping_off += page_size;
1280                 mapping_size -= page_size;

1282                 ebp->eb_eop = (*limitp == 0 ||
1283                     (mapping_size == 0 && mp->b_cont == NULL));

1285                 DTRACE_PROBE5(tx_mapping_add,
1286                     unsigned int, stp->st_index,
1287                     unsigned int, stp->st_n - 1,
1288                     uint64_t, ebp->eb_addr,
1289                     size_t, ebp->eb_size,
1290                     boolean_t, ebp->eb_eop);
1291         }

1293         ASSERT3U(*offp, <=, stmp->stm_size);

1295         return (0);

1297 fail1:
1298         DTRACE_PROBE1(fail1, int, rc);

1300         return (rc);
1301 }

1303 static inline int
1304 sfxge_tx_qbuffer_add(sfxge_txq_t *stp, sfxge_tx_buffer_t *stbp, boolean_t eop)
1305 {
1306         efx_buffer_t *ebp;
1307         int rc;

1309         ASSERT3U(stp->st_n, <=,
1310             EFX_TXQ_LIMIT(SFXGE_TX_NDESCS));
1311         if (stp->st_n == EFX_TXQ_LIMIT(SFXGE_TX_NDESCS)) {
1312                 rc = ENOSPC;
1313                 goto fail1;
1314         }
```

```
1316         ebp = &(stp->st_eb[stp->st_n++]);
1317         ebp->eb_addr = stbp->stb_esm.esm_addr + stbp->stb_off;
1318         ebp->eb_size = stbp->stb_esm.esm_size - stbp->stb_off;
1319         ebp->eb_eop = eop;

1321         (void) ddi_dma_sync(stbp->stb_esm.esm_dma_handle,
1322             stbp->stb_off, ebp->eb_size,
1323             DDI_DMA_SYNC_FORDEV);

1325         stbp->stb_off = stbp->stb_esm.esm_size;

1327         DTRACE_PROBE5(tx_buffer_add,
1328             unsigned int, stp->st_index,
1329             unsigned int, stp->st_n - 1,
1330             uint64_t, ebp->eb_addr, size_t, ebp->eb_size,
1331             boolean_t, ebp->eb_eop);

1333         return (0);

1335 fail1:
1336         DTRACE_PROBE1(fail1, int, rc);

1338         return (rc);
1339 }

1341 static inline boolean_t
1342 sfxge_tx_msgb_copy(mblk_t *mp, sfxge_tx_buffer_t *stbp, size_t *offp,
1343     size_t *limitp)
1344 {
1345         size_t data_off;
1346         size_t data_size;
1347         size_t copy_off;
1348         size_t copy_size;
1349         boolean_t eop;

1351         ASSERT3U(*offp, <=, MBLKL(mp));
1352         ASSERT(*limitp != 0);

1354         data_off = *offp;
1355         data_size = MBLKL(mp) - *offp;

1357         copy_off = stbp->stb_esm.esm_size;
1358         copy_size = SFXGE_TX_BUFFER_SIZE - copy_off;

1360         copy_size = MIN(copy_size, data_size);
1361         copy_size = MIN(copy_size, *limitp);

1363         bcopy(mp->b_rptr + data_off,
1364             stbp->stb_esm.esm_base + copy_off, copy_size);

1366         stbp->stb_esm.esm_size += copy_size;
1367         ASSERT3U(stbp->stb_esm.esm_size, <=,
1368             SFXGE_TX_BUFFER_SIZE);

1370         *offp += copy_size;
1371         *limitp -= copy_size;

1373         data_off += copy_size;
1374         data_size -= copy_size;

1376         eop = (*limitp == 0 ||
1377             (data_size == 0 && mp->b_cont == NULL));

1379         ASSERT3U(*offp, <=, MBLKL(mp));

1381         return (eop);
```



```
1382 }

1384 static int
1385 sfxge_tx_qpayload_fragment(sfxge_txq_t *stp, unsigned int id, mblk_t **mpp,
1386     size_t *offp, size_t size, boolean_t copy)
1387 {
1388         sfxge_t *sp = stp->st_sp;
1389         mblk_t *mp = *mpp;
1390         size_t off = *offp;
1391         sfxge_tx_buffer_t *stbp;
1392         sfxge_tx_mapping_t *stmp;
1393         int rc;

1395         stbp = stp->st_stbp[id];
1396         ASSERT(stbp == NULL || (stbp->stb_esm.esm_size == stbp->stb_off));

1398         stmp = stp->st_stmp[id];

1400         while (size != 0) {
1401                 boolean_t eop;

1403                 ASSERT(mp != NULL);

1405                 if (mp->b_cont != NULL)
1406                         prefetch_read_many(mp->b_cont);

1408                 ASSERT3U(off, <, MBLKL(mp));

1410                 if (copy)
1411                         goto copy;

1413                 /*
1414                  * Check whether we have already mapped this data block for
1415                  * DMA.
1416                  */
1417                 if (stmp == NULL || stmp->stm_mp != mp) {
1418                         /*
1419                          * If we are part way through copying a data block then
1420                          * there's no point in trying to map it for DMA.
1421                          */
1422                         if (off != 0)
1423                                 goto copy;

1425                         /*
1426                          * If the data block is too short then the cost of
1427                          * mapping it for DMA would outweigh the cost of
1428                          * copying it.
1429                          */
1430                         if (MBLKL(mp) < SFXGE_TX_COPY_THRESHOLD)
1431                                 goto copy;

1433                         /* Try to grab a transmit mapping from the pool */
1434                         stmp = sfxge_tx_qfmp_get(stp);
1435                         if (stmp == NULL) {
1436                                 /*
1437                                  * The pool was empty so allocate a new
1438                                  * mapping.
1439                                  */
1440                                 if ((stmp = kmem_cache_alloc(sp->s_tmc,
1441                                     KM_NOSLEEP)) == NULL)
1442                                         goto copy;
1443                         }

1445                         /* Add the DMA mapping to the list */
1446                         stmp->stm_next = stp->st_stmp[id];
1447                         stp->st_stmp[id] = stmp;
```

```
1449                        /* Try to bind the data block to the mapping */
1450                        if (sfxge_tx_msgb_bind(mp, stmp) != 0)
1451                                goto copy;
1452                }
1453                ASSERT3P(stmp->stm_mp, ==, mp);

1455                /*
1456                 * If we have a partially filled buffer then we must add it to
1457                 * the fragment list before adding the mapping.
1458                 */
1459                if (stbp != NULL && (stbp->stb_esm.esm_size > stbp->stb_off)) {
1460                        rc = sfxge_tx_qbuffer_add(stp, stbp, B_FALSE);
1461                        if (rc != 0)
1462                                goto fail1;
1463                }

1465                /* Add the mapping to the fragment list */
1466                rc = sfxge_tx_qmapping_add(stp, stmp, &off, &size);
1467                if (rc != 0)
1468                        goto fail2;

1470                ASSERT(off == MBLKL(mp) || size == 0);

1472                /*
1473                 * If the data block has been exhausted then Skip over the
1474                 * control block and advance to the next data block.
1475                 */
1476                if (off == MBLKL(mp)) {
1477                        mp = mp->b_cont;
1478                        off = 0;
1479                }

1481                continue;

1483        copy:
1484                if (stbp == NULL ||
1485                    stbp->stb_esm.esm_size == SFXGE_TX_BUFFER_SIZE) {
1486                        /* Try to grab a buffer from the pool */
1487                        stbp = sfxge_tx_qfbp_get(stp);
1488                        if (stbp == NULL) {
1489                                /*
1490                                 * The pool was empty so allocate a new
1491                                 * buffer.
1492                                 */
1493                                if ((stbp = kmem_cache_alloc(sp->s_tbc,
1494                                    KM_NOSLEEP)) == NULL) {
1495                                        rc = ENOMEM;
1496                                        goto fail3;
1497                                }
1498                        }

1500                        /* Add it to the list */
1501                        stbp->stb_next = stp->st_stbp[id];
1502                        stp->st_stbp[id] = stbp;
1503                }

1505                /* Copy as much of the data block as we can into the buffer */
1506                eop = sfxge_tx_msgb_copy(mp, stbp, &off, &size);

1508                ASSERT(off == MBLKL(mp) || size == 0 ||
1509                    stbp->stb_esm.esm_size == SFXGE_TX_BUFFER_SIZE);

1511                /*
1512                 * If we have reached the end of the packet, or the buffer is
1513                 * full, then add the buffer to the fragment list.
```



```
1514                 */
1515                if (stbp->stb_esm.esm_size == SFXGE_TX_BUFFER_SIZE || eop) {
1516                        rc = sfxge_tx_qbuffer_add(stp, stbp, eop);
1517                        if (rc != 0)
1518                                goto fail4;
1519                }

1521                /*
1522                 * If the data block has been exhaused then advance to the next
1523                 * one.
1524                 */
1525                if (off == MBLKL(mp)) {
1526                        mp = mp->b_cont;
1527                        off = 0;
1528                }
1529        }

1531        *mpp = mp;
1532        *offp = off;

1534        return (0);

1536 fail4:
1537        DTRACE_PROBE(fail4);
1538 fail3:
1539        DTRACE_PROBE(fail3);
1540 fail2:
1541        DTRACE_PROBE(fail2);
1542 fail1:
1543        DTRACE_PROBE1(fail1, int, rc);

1545        return (rc);
1546 }

1548 static int
1549 sfxge_tx_qlso_fragment(sfxge_txq_t *stp, sfxge_tx_packet_t *stpp,
1550     boolean_t copy)
1551 {
1552        sfxge_t *sp = stp->st_sp;
1553        mblk_t *mp = stpp->stp_mp;
1554        struct ether_header *etherhp = stpp->stp_etherhp;
1555        struct ip *iphp = stpp->stp_iphp;
1556        struct tcphdr *thp = stpp->stp_thp;
1557        size_t size = stpp->stp_size;
1558        size_t off = stpp->stp_off;
1559        size_t mss = stpp->stp_mss;
1560        unsigned int id;
1561        caddr_t hp;
1562        size_t ehs, hs;
1563        uint16_t start_len;
1564        uint16_t start_id;
1565        uint16_t ip_id;
1566        uint8_t start_flags;
1567        uint32_t start_seq;
1568        uint32_t th_seq;
1569        size_t lss;
1570        sfxge_tx_buffer_t *stbp;
1571        int rc;

1573        ASSERT(mutex_owned(&(stp->st_lock)));

1575        if ((DB_LSOFLAGS(mp) & HW_LSO) == 0) {
1576                rc = EINVAL;
1577                goto fail1;
1578        }
```

```
1580         id = stp->st_added & (SFXGE_TX_NDESCS - 1);

1582         ASSERT(stp->st_n == 0);
1583         ASSERT(stp->st_stbp[id] == NULL);
1584         ASSERT(stp->st_stmp[id] == NULL);

1586         ehs = (etherhp->ether_type == htons(ETHERTYPE_VLAN)) ?
1587             sizeof (struct ether_vlan_header) :
1588             sizeof (struct ether_header);
1589         if (msgdsize(mp) != ehs + ntohs(iphp->ip_len)) {
1590                 rc = EINVAL;
1591                 goto fail2;
1592         }

1594         /* The payload offset is equivalent to the size of the headers */
1595         hp = (caddr_t)(mp->b_rptr);
1596         hs = off;

1598         /*
1599          * If the initial data block only contains the headers then advance
1600          * to the next one.
1601          */
1602         if (hs > MBLKL(mp)) {
1603                 rc = EINVAL;
1604                 goto fail3;
1605         }
1606         mp->b_rptr += hs;

1608         if (MBLKL(mp) == 0)
1609                 mp = mp->b_cont;

1611         off = 0;

1613         /* Check IP and TCP headers are suitable for LSO */
1614         if (((iphp->ip_off & ~htons(IP_DF)) != 0) ||
1615             ((thp->th_flags & (TH_URG | TH_SYN)) != 0) ||
1616             (thp->th_urp != 0)) {
1617                 rc = EINVAL;
1618                 goto fail4;
1619         }

1621         if (size + (thp->th_off << 2) + (iphp->ip_hl << 2) !=
1622             ntohs(iphp->ip_len)) {
1623                 rc = EINVAL;
1624                 goto fail4;
1625         }

1627         /*
1628          * Get the base IP id, The stack leaves enough of a gap in id space
1629          * for us to increment this for each segment we send out.
1630          */
1631         start_len = ntohs(iphp->ip_len);
1632         start_id = ip_id = ntohs(iphp->ip_id);

1634         /* Get the base TCP sequence number and flags */
1635         start_flags = thp->th_flags;
1636         start_seq = th_seq = ntohl(thp->th_seq);

1638         /* Adjust the header for interim segments */
1639         iphp->ip_len = htons((iphp->ip_hl << 2) + (thp->th_off << 2) + mss);
1640         thp->th_flags = start_flags & ~(TH_PUSH | TH_FIN);

1642         lss = size;
1643         if ((lss / mss) >= (EFX_TXQ_LIMIT(SFXGE_TX_NDESCS) / 2)) {
1644                 rc = EINVAL;
1645                 goto fail5;
```



```
1646         }

1648         stbp = NULL;
1649         while (lss != 0) {
1650                 size_t ss = MIN(lss, mss);
1651                 boolean_t eol = (ss == lss);

1653                 /* Adjust the header for this segment */
1654                 iphp->ip_id = htons(ip_id);
1655                 ip_id++;

1657                 thp->th_seq = htonl(th_seq);
1658                 th_seq += ss;

1660                 /* If this is the final segment then do some extra adjustment */
1661                 if (eol) {
1662                         iphp->ip_len = htons((iphp->ip_hl << 2) +
1663                             (thp->th_off << 2) + ss);
1664                         thp->th_flags = start_flags;
1665                 }

1667                 if (stbp == NULL ||
1668                     stbp->stb_esm.esm_size + hs > SFXGE_TX_BUFFER_SIZE) {
1669                         /* Try to grab a buffer from the pool */
1670                         stbp = sfxge_tx_qfbp_get(stp);
1671                         if (stbp == NULL) {
1672                                 /*
1673                                  * The pool was empty so allocate a new
1674                                  * buffer.
1675                                  */
1676                                 if ((stbp = kmem_cache_alloc(sp->s_tbc,
1677                                     KM_NOSLEEP)) == NULL) {
1678                                         rc = ENOMEM;
1679                                         goto fail6;
1680                                 }
1681                         }

1683                         /* Add it to the list */
1684                         stbp->stb_next = stp->st_stbp[id];
1685                         stp->st_stbp[id] = stbp;
1686                 }

1688                 /* Copy in the headers */
1689                 ASSERT3U(stbp->stb_off, ==, stbp->stb_esm.esm_size);
1690                 bcopy(hp, stbp->stb_esm.esm_base + stbp->stb_off, hs);
1691                 stbp->stb_esm.esm_size += hs;

1693                 /* Add the buffer to the fragment list */
1694                 rc = sfxge_tx_qbuffer_add(stp, stbp, B_FALSE);
1695                 if (rc != 0)
1696                         goto fail7;

1698                 /* Add the payload to the fragment list */
1699                 if ((rc = sfxge_tx_qpayload_fragment(stp, id, &mp, &off,
1700                     ss, copy)) != 0)
1701                         goto fail8;

1703                 lss -= ss;
1704         }
1705         ASSERT3U(off, ==, 0);
1706         ASSERT3P(mp, ==, NULL);

1708         ASSERT3U(th_seq - start_seq, ==, size);

1710         /*
1711          * If no part of the packet has been mapped for DMA then we can free
```

```
1712          * it now, otherwise it can only be freed on completion.
1713          */
1714         if (stp->st_stmp[id] == NULL)
1715                 freemsg(stpp->stp_mp);
1716         else
1717                 stp->st_mp[id] = stpp->stp_mp;

1719         stpp->stp_mp = NULL;

1721         return (0);

1723 fail8:
1724         DTRACE_PROBE(fail8);
1725 fail7:
1726         DTRACE_PROBE(fail7);
1727 fail6:
1728         DTRACE_PROBE(fail6);
1729 fail5:
1730         DTRACE_PROBE(fail5);

1732         /* Restore the header */
1733         thp->th_seq = htonl(start_seq);
1734         thp->th_flags = start_flags;

1736         iphp->ip_len = htons(start_len);
1737         iphp->ip_id = htons(start_id);

1739 fail4:
1740         DTRACE_PROBE(fail4);

1742         mp = stpp->stp_mp;
1743         mp->b_rptr -= hs;

1745         ASSERT3U(((etherhp->ether_type == htons(ETHERTYPE_VLAN)) ?
1746             sizeof (struct ether_vlan_header) :
1747             sizeof (struct ether_header)) +
1748             ntohs(iphp->ip_len), ==, msgdsize(mp));

1750         ASSERT(stp->st_mp[id] == NULL);

1752 fail3:
1753         DTRACE_PROBE(fail3);
1754 fail2:
1755         DTRACE_PROBE(fail2);
1756 fail1:
1757         DTRACE_PROBE1(fail1, int, rc);

1759         return (rc);
1760 }

1762 static int
1763 sfxge_tx_qpacket_fragment(sfxge_txq_t *stp, sfxge_tx_packet_t *stpp,
1764     boolean_t copy)
1765 {
1766         sfxge_t *sp = stp->st_sp;
1767         mblk_t *mp = stpp->stp_mp;
1768         unsigned int id;
1769         size_t off;
1770         size_t size;
1771         sfxge_tx_mapping_t *stmp;
1772         sfxge_tx_buffer_t *stbp;
1773         int rc;

1775         ASSERT(mutex_owned(&(stp->st_lock)));

1777         ASSERT(stp->st_n == 0);
```



```
1779         id = stp->st_added & (SFXGE_TX_NDESCS - 1);

1781         ASSERT(stp->st_stbp[id] == NULL);
1782         ASSERT(stp->st_stmp[id] == NULL);

1784         off = 0;
1785         size = LONG_MAX;        /* must be larger than the packet */

1787         stbp = NULL;
1788         stmp = NULL;

1790         while (mp != NULL) {
1791                 boolean_t eop;

1793                 ASSERT(mp != NULL);

1795                 if (mp->b_cont != NULL)
1796                         prefetch_read_many(mp->b_cont);

1798                 ASSERT(stmp == NULL || stmp->stm_mp != mp);

1800                 if (copy)
1801                         goto copy;

1803                 /*
1804                  * If we are part way through copying a data block then there's
1805                  * no point in trying to map it for DMA.
1806                  */
1807                 if (off != 0)
1808                         goto copy;

1810                 /*
1811                  * If the data block is too short then the cost of mapping it
1812                  * for DMA would outweigh the cost of copying it.
1813                  *
1814                  * TX copy break
1815                  */
1816                 if (MBLKL(mp) < SFXGE_TX_COPY_THRESHOLD)
1817                         goto copy;

1819                 /* Try to grab a transmit mapping from the pool */
1820                 stmp = sfxge_tx_qfmp_get(stp);
1821                 if (stmp == NULL) {
1822                         /*
1823                          * The pool was empty so allocate a new
1824                          * mapping.
1825                          */
1826                         if ((stmp = kmem_cache_alloc(sp->s_tmc,
1827                             KM_NOSLEEP)) == NULL)
1828                                 goto copy;
1829                 }

1831                 /* Add the DMA mapping to the list */
1832                 stmp->stm_next = stp->st_stmp[id];
1833                 stp->st_stmp[id] = stmp;

1835                 /* Try to bind the data block to the mapping */
1836                 if (sfxge_tx_msgb_bind(mp, stmp) != 0)
1837                         goto copy;

1839                 /*
1840                  * If we have a partially filled buffer then we must add it to
1841                  * the fragment list before adding the mapping.
1842                  */
1843                 if (stbp != NULL && (stbp->stb_esm.esm_size > stbp->stb_off)) {
```

```
1844                        rc = sfxge_tx_qbuffer_add(stp, stbp, B_FALSE);
1845                        if (rc != 0)
1846                                goto fail1;
1847                }

1849                /* Add the mapping to the fragment list */
1850                rc = sfxge_tx_qmapping_add(stp, stmp, &off, &size);
1851                if (rc != 0)
1852                        goto fail2;

1854                ASSERT3U(off, ==, MBLKL(mp));

1856                /* Advance to the next data block */
1857                mp = mp->b_cont;
1858                off = 0;
1859                continue;

1861 copy:
1862                if (stbp == NULL ||
1863                    stbp->stb_esm.esm_size == SFXGE_TX_BUFFER_SIZE) {
1864                        /* Try to grab a buffer from the pool */
1865                        stbp = sfxge_tx_qfbp_get(stp);
1866                        if (stbp == NULL) {
1867                                /*
1868                                 * The pool was empty so allocate a new
1869                                 * buffer.
1870                                 */
1871                                if ((stbp = kmem_cache_alloc(sp->s_tbc,
1872                                    KM_NOSLEEP)) == NULL) {
1873                                        rc = ENOMEM;
1874                                        goto fail3;
1875                                }
1876                        }

1878                        /* Add it to the list */
1879                        stbp->stb_next = stp->st_stbp[id];
1880                        stp->st_stbp[id] = stbp;
1881                }

1883                /* Copy as much of the data block as we can into the buffer */
1884                eop = sfxge_tx_msgb_copy(mp, stbp, &off, &size);

1886                ASSERT(off == MBLKL(mp) ||
1887                    stbp->stb_esm.esm_size == SFXGE_TX_BUFFER_SIZE);

1889                /*
1890                 * If we have reached the end of the packet, or the buffer is
1891                 * full, then add the buffer to the fragment list.
1892                 */
1893                if (stbp->stb_esm.esm_size == SFXGE_TX_BUFFER_SIZE || eop) {
1894                        rc = sfxge_tx_qbuffer_add(stp, stbp, eop);
1895                        if (rc != 0)
1896                                goto fail4;
1897                }

1899                /*
1900                 * If the data block has been exhaused then advance to the next
1901                 * one.
1902                 */
1903                if (off == MBLKL(mp)) {
1904                        mp = mp->b_cont;
1905                        off = 0;
1906                }
1907        }
1908        ASSERT3U(off, ==, 0);
1909        ASSERT3P(mp, ==, NULL);
```



```
1910        ASSERT3U(size, !=, 0);

1912        /*
1913         * If no part of the packet has been mapped for DMA then we can free
1914         * it now, otherwise it can only be freed on completion.
1915         */
1916        if (stp->st_stmp[id] == NULL)
1917                freemsg(stpp->stp_mp);
1918        else
1919                stp->st_mp[id] = stpp->stp_mp;

1921        stpp->stp_mp = NULL;

1923        return (0);

1925 fail4:
1926        DTRACE_PROBE(fail4);
1927 fail3:
1928        DTRACE_PROBE(fail3);
1929 fail2:
1930        DTRACE_PROBE(fail2);
1931 fail1:
1932        DTRACE_PROBE1(fail1, int, rc);

1934        ASSERT(stp->st_stmp[id] == NULL);

1936        return (rc);
1937 }


1940 #define SFXGE_TX_QDPL_PUT_PENDING(_stp)                                 \
1941        ((_stp)->st_dpl.std_put != 0)

1943 static void
1944 sfxge_tx_qdpl_swizzle(sfxge_txq_t *stp)
1945 {
1946        sfxge_tx_dpl_t *stdp = &(stp->st_dpl);
1947        volatile uintptr_t *putp;
1948        uintptr_t put;
1949        sfxge_tx_packet_t *stpp;
1950        sfxge_tx_packet_t *p;
1951        sfxge_tx_packet_t **pp;
1952        unsigned int count;

1954        ASSERT(mutex_owned(&(stp->st_lock)));

1956        /*
1957         * Guaranteed that in flight TX packets will cause more TX completions
1958         * hence more swizzles must happen
1959         */
1960        ASSERT3U(stdp->std_count, <=, sfxge_tx_dpl_get_pkt_max(stp));
1961        if (stdp->std_count >= stdp->get_pkt_limit)
1962                return;

1964        /* Acquire the put list - replacing with an empty list */
1965        putp = &(stdp->std_put);
1966        put = atomic_swap_ulong(putp, 0);
1967        stpp = (void *)put;

1969        if (stpp == NULL)
1970                return;

1972        /* Reverse the list */
1973        pp = &(stpp->stp_next);
1974        p = NULL;
```

```
1976         count = 0;
1977         do {
1978                 sfxge_tx_packet_t *next;

1980                 next = stpp->stp_next;

1982                 stpp->stp_next = p;
1983                 p = stpp;

1985                 count++;
1986                 stpp = next;
1987         } while (stpp != NULL);

1989         /* Add it to the tail of the get list */
1990         ASSERT3P(*pp, ==, NULL);

1992         *(stdp->std_getp) = p;
1993         stdp->std_getp = pp;
1994         stdp->std_count += count;
1995         ASSERT3U(stdp->std_count, <=, sfxge_tx_dpl_get_pkt_max(stp));

1997         DTRACE_PROBE2(dpl_counts, int, stdp->std_count, int, count);
1998 }


2001 /*
2002  * If TXQ locked, add the RX DPL put list and this packet to the TX DPL get list
2003  * If TXQ unlocked, atomically add this packet to TX DPL put list
2004  *
2005  * The only possible error is ENOSPC (used for TX backpressure)
2006  * For the TX DPL put or get list becoming full, in both cases there must be
2007  * future TX completions (as represented by the packets on the DPL get lists).
2008  *
2009  * This ensures that in the future mac_tx_update() will be called from
2010  * sfxge_tx_qcomplete()
2011  */
2012 static inline int
2013 sfxge_tx_qdpl_add(sfxge_txq_t *stp, sfxge_tx_packet_t *stpp, int locked)
2014 {
2015         sfxge_tx_dpl_t *stdp = &stp->st_dpl;

2017         ASSERT3P(stpp->stp_next, ==, NULL);

2019         if (locked) {
2020                 ASSERT(mutex_owned(&stp->st_lock));

2022                 if (stdp->std_count >= stdp->get_pkt_limit) {
2023                         stdp->get_full_count++;
2024                         return (ENOSPC);
2025                 }

2027                 /* Reverse the put list onto the get list */
2028                 sfxge_tx_qdpl_swizzle(stp);

2030                 /* Add to the tail of the get list */
2031                 *(stdp->std_getp) = stpp;
2032                 stdp->std_getp = &stpp->stp_next;
2033                 stdp->std_count++;
2034                 ASSERT3U(stdp->std_count, <=, sfxge_tx_dpl_get_pkt_max(stp));

2036         } else {
2037                 volatile uintptr_t *putp;
2038                 uintptr_t old;
2039                 uintptr_t new;
2040                 sfxge_tx_packet_t *old_pkt;
```



```
2042                 putp = &(stdp->std_put);
2043                 new = (uintptr_t)stpp;

2045                 /* Add to the head of the put list, keeping a list length */
2046                 do {
2047                         old = *putp;
2048                         old_pkt =  (sfxge_tx_packet_t *)old;

2050                         stpp->stp_dpl_put_len = old ?
2051                             old_pkt->stp_dpl_put_len + 1 : 1;

2053                         if (stpp->stp_dpl_put_len >= stdp->put_pkt_limit) {
2054                                 stpp->stp_next = 0;
2055                                 stpp->stp_dpl_put_len = 0;
2056                                 stdp->put_full_count++;
2057                                 return (ENOSPC);
2058                         }

2060                         stpp->stp_next = (void *)old;
2061                 } while (atomic_cas_ulong(putp, old, new) != old);
2062         }
2063         return (0);
2064 }


2067 /* Take all packets from DPL get list and try to send to HW */
2068 static void
2069 sfxge_tx_qdpl_drain(sfxge_txq_t *stp)
2070 {
2071         sfxge_t *sp = stp->st_sp;
2072         sfxge_tx_dpl_t *stdp = &(stp->st_dpl);
2073         unsigned int pushed = stp->st_added;
2074         sfxge_tx_packet_t *stpp;
2075         unsigned int count;

2077         ASSERT(mutex_owned(&(stp->st_lock)));

2079         prefetch_read_many(sp->s_enp);
2080         prefetch_read_many(stp->st_etp);

2082         stpp = stdp->std_get;
2083         count = stdp->std_count;

2085         while (count != 0) {
2086                 sfxge_tx_packet_t *next;
2087                 boolean_t copy;
2088                 int rc;

2090                 ASSERT(stpp != NULL);

2092                 /* Split stpp off */
2093                 next = stpp->stp_next;
2094                 stpp->stp_next = NULL;

2096                 if (next != NULL)
2097                         prefetch_read_many(next);

2099                 if (stp->st_state != SFXGE_TXQ_STARTED)
2100                         goto reject;

2102                 copy = B_FALSE;

2104 again:
2105                 /* Fragment the packet */
2106                 if (stpp->stp_mss != 0) {
2107                         rc = sfxge_tx_qlso_fragment(stp, stpp, copy);
```

```
2108                } else {
2109                        rc = sfxge_tx_qpacket_fragment(stp, stpp, copy);
2110                }

2112                switch (rc) {
2113                case 0:
2114                        break;

2116                case ENOSPC:
2117                        if (!copy)
2118                                goto copy;

2120                /*FALLTHRU*/
2121                default:
2122                        goto reject;
2123                }

2125                /* Free the packet structure */
2126                stpp->stp_etherhp = NULL;
2127                stpp->stp_iphp = NULL;
2128                stpp->stp_thp = NULL;
2129                stpp->stp_off = 0;
2130                stpp->stp_size = 0;
2131                stpp->stp_mss = 0;
2132                stpp->stp_dpl_put_len = 0;

2134                ASSERT3P(stpp->stp_mp, ==, NULL);

2136                if (sfxge_tx_qfpp_put(stp, stpp) != 0) {
2137                        sfxge_tx_packet_destroy(sp, stpp);
2138                        stpp = NULL;
2139                }

2141                --count;
2142                stpp = next;

2144                /* Post the packet */
2145                sfxge_tx_qlist_post(stp);

2147                if (stp->st_unblock != SFXGE_TXQ_NOT_BLOCKED)
2148                        goto defer;

2150                if (stp->st_added - pushed >= SFXGE_TX_BATCH) {
2151                        efx_tx_qpush(stp->st_etp, stp->st_added);
2152                        pushed = stp->st_added;
2153                }

2155                continue;

2157 copy:
2158                /* Abort the current fragment list */
2159                sfxge_tx_qlist_abort(stp);

2161                /* Try copying the packet to flatten it */
2162                ASSERT(!copy);
2163                copy = B_TRUE;

2165                goto again;

2167 reject:
2168                /* Abort the current fragment list */
2169                sfxge_tx_qlist_abort(stp);

2171                /* Discard the packet */
2172                freemsg(stpp->stp_mp);
2173                stpp->stp_mp = NULL;
```



```
2175                /* Free the packet structure */
2176                stpp->stp_etherhp = NULL;
2177                stpp->stp_iphp = NULL;
2178                stpp->stp_thp = NULL;
2179                stpp->stp_off = 0;
2180                stpp->stp_size = 0;
2181                stpp->stp_mss = 0;
2182                stpp->stp_dpl_put_len = 0;

2184                if (sfxge_tx_qfpp_put(stp, stpp) != 0) {
2185                        sfxge_tx_packet_destroy(sp, stpp);
2186                        stpp = NULL;
2187                }

2189                --count;
2190                stpp = next;
2191                continue;
2192 defer:
2193                DTRACE_PROBE1(defer, unsigned int, stp->st_index);
2194                break;
2195        }

2197        if (count == 0) {
2198                /* New empty get list */
2199                ASSERT3P(stpp, ==, NULL);
2200                stdp->std_get = NULL;
2201                stdp->std_count = 0;

2203                stdp->std_getp = &(stdp->std_get);
2204        } else {
2205                /* shorten the list by moving the head */
2206                stdp->std_get = stpp;
2207                stdp->std_count = count;
2208                ASSERT3U(stdp->std_count, <=, sfxge_tx_dpl_get_pkt_max(stp));
2209        }

2211        if (stp->st_added != pushed)
2212                efx_tx_qpush(stp->st_etp, stp->st_added);

2214        ASSERT(stp->st_unblock != SFXGE_TXQ_NOT_BLOCKED ||
2215            stdp->std_count == 0);
2216 }

2218 /* Swizzle deferred packet list, try and push to HW */
2219 static inline void
2220 sfxge_tx_qdpl_service(sfxge_txq_t *stp)
2221 {
2222        do {
2223                ASSERT(mutex_owned(&(stp->st_lock)));

2225                if (SFXGE_TX_QDPL_PUT_PENDING(stp))
2226                        sfxge_tx_qdpl_swizzle(stp);

2228                if (stp->st_unblock == SFXGE_TXQ_NOT_BLOCKED)
2229                        sfxge_tx_qdpl_drain(stp);

2231                mutex_exit(&(stp->st_lock));

2233                if (!SFXGE_TX_QDPL_PUT_PENDING(stp))
2234                        break;
2235        } while (mutex_tryenter(&(stp->st_lock)));
2236 }

2238 static void
2239 sfxge_tx_qdpl_flush_locked(sfxge_txq_t *stp)
```

```
2240 {
2241         sfxge_t *sp = stp->st_sp;
2242         sfxge_tx_dpl_t *stdp = &(stp->st_dpl);
2243         sfxge_tx_packet_t *stpp;
2244         unsigned int count;

2246         ASSERT(mutex_owned(&(stp->st_lock)));

2248         /* Swizzle put list to the get list */
2249         sfxge_tx_qdpl_swizzle(stp);

2251         stpp = stdp->std_get;
2252         count = stdp->std_count;

2254         while (count != 0) {
2255                 sfxge_tx_packet_t *next;

2257                 next = stpp->stp_next;
2258                 stpp->stp_next = NULL;

2260                 /* Discard the packet */
2261                 freemsg(stpp->stp_mp);
2262                 stpp->stp_mp = NULL;

2264                 /* Free the packet structure */
2265                 stpp->stp_etherhp = NULL;
2266                 stpp->stp_iphp = NULL;
2267                 stpp->stp_thp = NULL;
2268                 stpp->stp_off = 0;
2269                 stpp->stp_size = 0;
2270                 stpp->stp_mss = 0;
2271                 stpp->stp_dpl_put_len = 0;

2273                 sfxge_tx_packet_destroy(sp, stpp);

2275                 --count;
2276                 stpp = next;
2277         }

2279         ASSERT3P(stpp, ==, NULL);

2281         /* Empty list */
2282         stdp->std_get = NULL;
2283         stdp->std_count = 0;
2284         stdp->std_getp = &(stdp->std_get);
2285 }


2288 void
2289 sfxge_tx_qdpl_flush(sfxge_txq_t *stp)
2290 {
2291         mutex_enter(&(stp->st_lock));
2292         sfxge_tx_qdpl_flush_locked(stp);
2293         mutex_exit(&(stp->st_lock));
2294 }


2297 static void
2298 sfxge_tx_qunblock(sfxge_txq_t *stp)
2299 {
2300         sfxge_t *sp = stp->st_sp;
2301         unsigned int evq = stp->st_evq;
2302         sfxge_evq_t *sep = sp->s_sep[evq];

2304         ASSERT(mutex_owned(&(sep->se_lock)));
```



```
2306         if (stp->st_state != SFXGE_TXQ_STARTED)
2307                 return;

2309         mutex_enter(&(stp->st_lock));

2311         if (stp->st_unblock != SFXGE_TXQ_NOT_BLOCKED) {
2312                 unsigned int level;

2314                 level = stp->st_added - stp->st_completed;
2315                 if (level <= stp->st_unblock) {
2316                         stp->st_unblock = SFXGE_TXQ_NOT_BLOCKED;
2317                         sfxge_tx_qlist_post(stp);
2318                 }
2319         }

2321         sfxge_tx_qdpl_service(stp);
2322         /* lock has been dropped */
2323 }

2325 void
2326 sfxge_tx_qcomplete(sfxge_txq_t *stp)
2327 {
2328         sfxge_t *sp = stp->st_sp;
2329         sfxge_tx_dpl_t *stdp = &(stp->st_dpl);
2330         unsigned int evq = stp->st_evq;
2331         sfxge_evq_t *sep = sp->s_sep[evq];
2332         unsigned int completed;

2334         ASSERT(mutex_owned(&(sep->se_lock)));

2336         completed = stp->st_completed;
2337         while (completed != stp->st_pending) {
2338                 unsigned int id;
2339                 sfxge_tx_mapping_t *stmp;

2341                 id = completed++ & (SFXGE_TX_NDESCS - 1);

2343                 if ((stmp = stp->st_stmp[id]) != NULL) {
2344                         mblk_t *mp;

2346                         /* Unbind all the mappings */
2347                         do {
2348                                 ASSERT(stmp->stm_mp != NULL);
2349                                 sfxge_tx_msgb_unbind(stmp);

2351                                 stmp = stmp->stm_next;
2352                         } while (stmp != NULL);

2354                         /*
2355                          * Now that the packet is no longer mapped for DMA it
2356                          * can be freed.
2357                          */
2358                         mp = stp->st_mp[id];
2359                         stp->st_mp[id] = NULL;

2361                         ASSERT(mp != NULL);
2362                         freemsg(mp);
2363                 }
2364         }
2365         stp->st_completed = completed;

2367         /* Check whether we need to unblock the queue */
2368         if (stp->st_unblock != SFXGE_TXQ_NOT_BLOCKED) {
2369                 unsigned int level;

2371                 level = stp->st_added - stp->st_completed;
```

```
2372                 if (level <= stp->st_unblock)
2373                         sfxge_tx_qunblock(stp);
2374         }

2376         /* Release TX backpressure from the TX DPL put/get list being full */
2377         if (stdp->std_count < stdp->get_pkt_limit)
2378                 mac_tx_update(sp->s_mh);
2379 }

2381 void
2382 sfxge_tx_qflush_done(sfxge_txq_t *stp)
2383 {
2384         sfxge_t *sp = stp->st_sp;

2386         ASSERT(mutex_owned(&(sp->s_sep[stp->st_evq]->se_lock)));

2388         mutex_enter(&(stp->st_lock));

2390         if (stp->st_flush == SFXGE_FLUSH_PENDING)
2391                 stp->st_flush = SFXGE_FLUSH_DONE;

2393         mutex_exit(&(stp->st_lock));

2395         mutex_enter(&(sp->s_tx_flush_lock));
2396         sp->s_tx_flush_pending--;
2397         if (sp->s_tx_flush_pending <= 0) {
2398                 /* All queues flushed: wakeup sfxge_tx_stop() */
2399                 cv_signal(&(sp->s_tx_flush_kv));
2400         }
2401         mutex_exit(&(sp->s_tx_flush_lock));
2402 }

2404 static void
2405 sfxge_tx_qflush(sfxge_t *sp, unsigned int index, boolean_t do_flush)
2406 {
2407         sfxge_txq_t *stp = sp->s_stp[index];

2409         ASSERT(mutex_owned(&(sp->s_state_lock)));

2411         mutex_enter(&(stp->st_lock));

2413         /* Prepare to flush and stop the queue */
2414         if (stp->st_state == SFXGE_TXQ_STARTED)
2415                 stp->st_state = SFXGE_TXQ_INITIALIZED;
2416         else
2417                 do_flush = B_FALSE; /* No hardware ring, so don't flush */

2419         if (do_flush)
2420                 stp->st_flush = SFXGE_FLUSH_PENDING;
2421         else
2422                 stp->st_flush = SFXGE_FLUSH_INACTIVE;

2424         mutex_exit(&(stp->st_lock));

2426         /* Flush the transmit queue */
2427         if (do_flush)
2428                 efx_tx_qflush(stp->st_etp);
2429 }

2431 static void
2432 sfxge_tx_qstop(sfxge_t *sp, unsigned int index)
2433 {
2434         sfxge_txq_t *stp = sp->s_stp[index];
2435         unsigned int evq = stp->st_evq;
2436         sfxge_evq_t *sep = sp->s_sep[evq];
```



```
2438         mutex_enter(&(sep->se_lock));
2439         mutex_enter(&(stp->st_lock));
2440         ASSERT3U(stp->st_state, ==, SFXGE_TXQ_INITIALIZED);

2442         /* All queues should have been flushed */
2443         ASSERT3S(stp->st_sp->s_tx_flush_pending, ==, 0);
2444         ASSERT(stp->st_flush != SFXGE_FLUSH_FAILED);

2446         /* in case of TX flush timeout */
2447         stp->st_flush = SFXGE_FLUSH_DONE;

2449         /* Destroy the transmit queue */
2450         efx_tx_qdestroy(stp->st_etp);
2451         stp->st_etp = NULL;

2453         /* Clear entries from the buffer table */
2454         sfxge_sram_buf_tbl_clear(sp, stp->st_id,
2455             EFX_TXQ_NBUFS(SFXGE_TX_NDESCS));

2457         sfxge_tx_qlist_abort(stp);
2458         ASSERT3U(stp->st_n, ==, 0);

2460         stp->st_unblock = SFXGE_TXQ_NOT_BLOCKED;

2462         stp->st_pending = stp->st_added;

2464         sfxge_tx_qcomplete(stp);
2465         ASSERT3U(stp->st_completed, ==, stp->st_pending);

2467         sfxge_tx_qreap(stp);
2468         ASSERT3U(stp->st_reaped, ==, stp->st_completed);

2470         /*
2471          * Ensure the deferred packet list is cleared
2472          * Can race with sfxge_tx_packet_add() adding to the put list
2473          */
2474         sfxge_tx_qdpl_flush_locked(stp);

2476         stp->st_added = 0;
2477         stp->st_pending = 0;
2478         stp->st_completed = 0;
2479         stp->st_reaped = 0;

2481         mutex_exit(&(stp->st_lock));
2482         mutex_exit(&(sep->se_lock));
2483 }

2485 static void
2486 sfxge_tx_qfini(sfxge_t *sp, unsigned int index)
2487 {
2488         sfxge_txq_t *stp = sp->s_stp[index];
2489         sfxge_tx_dpl_t *stdp = &(stp->st_dpl);

2491         /* Detach the TXQ from the driver */
2492         sp->s_stp[index] = NULL;
2493         ASSERT(sp->s_tx_qcount > 0);
2494         sp->s_tx_qcount--;

2496         ASSERT3U(stp->st_state, ==, SFXGE_TXQ_INITIALIZED);
2497         stp->st_state = SFXGE_TXQ_UNINITIALIZED;

2499         /* Tear down the statistics */
2500         sfxge_tx_kstat_fini(stp);

2502         /* Ensure the deferred packet list is empty */
2503         ASSERT3U(stdp->std_count, ==, 0);
```

```
2504         ASSERT3P(stdp->std_get, ==, NULL);
2505         ASSERT3U(stdp->std_put, ==, 0);

2507         /* Clear the free buffer pool */
2508         sfxge_tx_qfbp_empty(stp);

2510         /* Clear the free mapping pool */
2511         sfxge_tx_qfmp_empty(stp);

2513         /* Clear the free packet pool */
2514         sfxge_tx_qfpp_empty(stp);

2516         mutex_destroy(&(stp->st_lock));

2518         stp->st_evq = 0;
2519         stp->st_type = 0;
2520         stp->st_index = 0;

2522         kmem_cache_free(sp->s_tqc, stp);
2523 }

2525 int
2526 sfxge_tx_init(sfxge_t *sp)
2527 {
2528         sfxge_intr_t *sip = &(sp->s_intr);
2529         char name[MAXNAMELEN];
2530         int index;
2531         int rc;

2533         (void) snprintf(name, MAXNAMELEN - 1, "%s%d_tx_packet_cache",
2534             ddi_driver_name(sp->s_dip), ddi_get_instance(sp->s_dip));

2536         sp->s_tpc = kmem_cache_create(name, sizeof (sfxge_tx_packet_t),
2537             SFXGE_CPU_CACHE_SIZE, sfxge_tx_packet_ctor, sfxge_tx_packet_dtor,
2538             NULL, sp, NULL, 0);
2539         ASSERT(sp->s_tpc != NULL);

2541         (void) snprintf(name, MAXNAMELEN - 1, "%s%d_tx_buffer_cache",
2542             ddi_driver_name(sp->s_dip), ddi_get_instance(sp->s_dip));

2544         sp->s_tbc = kmem_cache_create(name, sizeof (sfxge_tx_buffer_t),
2545             SFXGE_CPU_CACHE_SIZE, sfxge_tx_buffer_ctor, sfxge_tx_buffer_dtor,
2546             NULL, sp, NULL, 0);
2547         ASSERT(sp->s_tbc != NULL);

2549         (void) snprintf(name, MAXNAMELEN - 1, "%s%d_tx_mapping_cache",
2550             ddi_driver_name(sp->s_dip), ddi_get_instance(sp->s_dip));

2552         sp->s_tmc = kmem_cache_create(name, sizeof (sfxge_tx_mapping_t),
2553             SFXGE_CPU_CACHE_SIZE, sfxge_tx_mapping_ctor, sfxge_tx_mapping_dtor,
2554             NULL, sp, NULL, 0);
2555         ASSERT(sp->s_tmc != NULL);

2557         (void) snprintf(name, MAXNAMELEN - 1, "%s%d_txq_cache",
2558             ddi_driver_name(sp->s_dip), ddi_get_instance(sp->s_dip));

2560         sp->s_tqc = kmem_cache_create(name, sizeof (sfxge_txq_t),
2561             SFXGE_CPU_CACHE_SIZE, sfxge_tx_qctor, sfxge_tx_qdtor, NULL, sp,
2562             NULL, 0);
2563         ASSERT(sp->s_tqc != NULL);

2565         /* Initialize the special non-checksummed transmit queues */

2567         /* NB sfxge_ev_qinit() is sensitive to using EVQ_0 */
2568         if ((rc = sfxge_tx_qinit(sp, SFXGE_TXQ_NON_CKSUM,
2569                     SFXGE_TXQ_NON_CKSUM, EVQ_0)) != 0)
```



```
2570                 goto fail1;

2572         /* NB sfxge_ev_qinit() is sensitive to using EVQ_0 */
2573         if ((rc = sfxge_tx_qinit(sp, SFXGE_TXQ_IP_CKSUM,
2574             SFXGE_TXQ_IP_CKSUM, EVQ_0)) != 0)
2575                 goto fail2;

2577         /* Initialize the normal transmit queues */
2578         for (index = 0; index < sip->si_nalloc; index++) {
2579                 if ((rc = sfxge_tx_qinit(sp, SFXGE_TXQ_IP_TCP_UDP_CKSUM + index,
2580                     SFXGE_TXQ_IP_TCP_UDP_CKSUM, index)) != 0)
2581                         goto fail3;
2582         }

2584         return (0);

2586 fail3:
2587         DTRACE_PROBE(fail3);

2589         while (--index >= 0)
2590                 sfxge_tx_qfini(sp, SFXGE_TXQ_IP_TCP_UDP_CKSUM + index);

2592         sfxge_tx_qfini(sp, SFXGE_TXQ_IP_CKSUM);

2594 fail2:
2595         DTRACE_PROBE(fail2);

2597 fail1:
2598         DTRACE_PROBE1(fail1, int, rc);

2600         sfxge_tx_qfini(sp, SFXGE_TXQ_NON_CKSUM);

2602         kmem_cache_destroy(sp->s_tqc);
2603         sp->s_tqc = NULL;

2605         kmem_cache_destroy(sp->s_tmc);
2606         sp->s_tmc = NULL;

2608         kmem_cache_destroy(sp->s_tbc);
2609         sp->s_tbc = NULL;

2611         kmem_cache_destroy(sp->s_tpc);
2612         sp->s_tpc = NULL;

2614         return (rc);
2615 }

2617 int
2618 sfxge_tx_start(sfxge_t *sp)
2619 {
2620         efx_nic_t *enp = sp->s_enp;
2621         int index;
2622         int rc;

2624         /* Initialize the transmit module */
2625         if ((rc = efx_tx_init(enp)) != 0)
2626                 goto fail1;

2628         for (index = 0; index < sp->s_tx_qcount; index++) {
2629                 if ((rc = sfxge_tx_qstart(sp, index)) != 0)
2630                         goto fail2;
2631         }

2633         return (0);

2635 fail2:
```

```
2636         DTRACE_PROBE(fail2);

2638         while (--index >= 0)
2639                 sfxge_tx_qstop(sp, index);

2641         /* Tear down the transmit module */
2642         efx_tx_fini(enp);

2644 fail1:
2645         DTRACE_PROBE1(fail1, int, rc);

2647         return (rc);
2648 }


2651 /*
2652  * Add a packet to the TX Deferred Packet List and if the TX queue lock
2653  * can be acquired then call sfxge_tx_qdpl_service() to fragment and push
2654  * to the H/W transmit descriptor ring
2655  *
2656  * If ENOSPC is returned then the DPL is full or the packet create failed, but
2657  * the mblk isn't freed so that the caller can return this mblk from mc_tx() to
2658  * back-pressure the OS stack.
2659  *
2660  * For all other errors the mblk is freed
2661  */
2662 int
2663 sfxge_tx_packet_add(sfxge_t *sp, mblk_t *mp)
2664 {
2665         struct ether_header *etherhp;
2666         struct ip *iphp;
2667         struct tcphdr *thp;
2668         size_t off;
2669         size_t size;
2670         size_t mss;
2671         sfxge_txq_t *stp;
2672         boolean_t locked;
2673         sfxge_tx_packet_t *stpp;
2674         int rc = 0;

2676         ASSERT3P(mp->b_next, ==, NULL);
2677         ASSERT(!(DB_CKSUMFLAGS(mp) & HCK_PARTIALCKSUM));

2679         /*
2680          * Do not enqueue packets during startup/shutdown;
2681          *
2682          * NOTE: This access to the state is NOT protected by the state lock. It
2683          * is an imperfect test and anything further getting onto the get/put
2684          * deferred packet lists is cleaned up in (possibly repeated) calls to
2685          * sfxge_can_destroy().
2686          */
2687         if (sp->s_state != SFXGE_STARTED) {
2688                 rc = EINVAL;
2689                 goto fail1;
2690         }

2692         etherhp = NULL;
2693         iphp = NULL;
2694         thp = NULL;
2695         off = 0;
2696         size = 0;
2697         mss = 0;

2699         /* Check whether we need the header pointers for LSO segmentation */
2700         if (DB_LSOFLAGS(mp) & HW_LSO) {
2701                 /* LSO segmentation relies on hardware checksum offload */
```



```
2702                 DB_CKSUMFLAGS(mp) |= HCK_FULLCKSUM;

2704                 if ((mss = DB_LSOMSS(mp)) == 0) {
2705                         rc = EINVAL;
2706                         goto fail1;
2707                 }

2709                 sfxge_tcp_parse(mp, &etherhp, &iphp, &thp, &off, &size);

2711                 if (etherhp == NULL ||
2712                     iphp == NULL ||
2713                     thp == NULL ||
2714                     off == 0) {
2715                         rc = EINVAL;
2716                         goto fail2;
2717                 }
2718         }

2720         /* Choose the appropriate transit queue */
2721         if (DB_CKSUMFLAGS(mp) & HCK_FULLCKSUM) {
2722                 sfxge_rx_scale_t *srsp = &(sp->s_rx_scale);

2724                 if (srsp->srs_state == SFXGE_RX_SCALE_STARTED) {
2725                         uint16_t hash;
2726                         int index;

2728                         if (srsp->srs_count > 1) {
2729                                 /*
2730                                  * If we have not already parsed the headers
2731                                  * for LSO segmentation then we need to do it
2732                                  * now so we can calculate the hash.
2733                                  */
2734                                 if (thp == NULL)
2735                                         sfxge_tcp_parse(mp, &etherhp, &iphp,
2736                                             &thp, &off, &size);

2738                                 if (thp != NULL) {
2739                                         SFXGE_TCP_HASH(
2740                                             ntohl(iphp->ip_dst.s_addr),
2741                                             ntohs(thp->th_dport),
2742                                             ntohl(iphp->ip_src.s_addr),
2743                                             ntohs(thp->th_sport), hash);

2745                                         index = srsp->srs_tbl[hash %
2746                                             SFXGE_RX_SCALE_MAX];
2747                                 } else {
2748                                         /*
2749                                          * Non-TCP traffix always goes to the
2750                                          * the queue in the zero-th entry of
2751                                          * the RSS table.
2752                                          */
2753                                         index = srsp->srs_tbl[0];
2754                                 }
2755                         } else {
2756                                 /*
2757                                  * It does not matter what the hash is
2758                                  * because all the RSS table entries will be
2759                                  * the same.
2760                                  */
2761                                 index = srsp->srs_tbl[0];
2762                         }

2764                         /*
2765                          * Find the event queue corresponding to the hash in
2766                          * the RSS table.
2767                          */
```

```
2768                        stp = sp->s_stp[SFXGE_TXQ_IP_TCP_UDP_CKSUM + index];
2769                        ASSERT3U(stp->st_evq, ==, index);
2770                } else {
2771                        stp = sp->s_stp[SFXGE_TXQ_IP_TCP_UDP_CKSUM];
2772                }
2773        } else if (DB_CKSUMFLAGS(mp) & HCK_IPV4_HDRCKSUM) {
2774                stp = sp->s_stp[SFXGE_TXQ_IP_CKSUM];
2775        } else {
2776                if ((stp = sp->s_stp[SFXGE_TXQ_NON_CKSUM]) == NULL)
2777                        stp = sp->s_stp[SFXGE_TXQ_IP_CKSUM];
2778        }
2779        ASSERT(stp != NULL);

2781        ASSERT(mss == 0 || (DB_LSOFLAGS(mp) & HW_LSO));

2783        /* Try to grab the lock */
2784        locked = mutex_tryenter(&(stp->st_lock));

2786        if (locked) {
2787                /* Try to grab a packet from the pool */
2788                stpp = sfxge_tx_qfpp_get(stp);
2789        } else {
2790                stpp = NULL;
2791        }

2793        if (stpp == NULL) {
2794                /*
2795                 * Either the pool was empty or we don't have the lock so
2796                 * allocate a new packet.
2797                 */
2798                if ((stpp = sfxge_tx_packet_create(sp)) == NULL) {
2799                        rc = ENOSPC;
2800                        goto fail3;
2801                }
2802        }

2804        stpp->stp_mp = mp;
2805        stpp->stp_etherhp = etherhp;
2806        stpp->stp_iphp = iphp;
2807        stpp->stp_thp = thp;
2808        stpp->stp_off = off;
2809        stpp->stp_size = size;
2810        stpp->stp_mss = mss;
2811        stpp->stp_dpl_put_len = 0;

2813        rc = sfxge_tx_qdpl_add(stp, stpp, locked);
2814        if (rc != 0) {
2815                /* ENOSPC can happen for DPL get or put list is full */
2816                ASSERT3U(rc, ==, ENOSPC);

2818                /*
2819                 * Note; if this is the unlocked DPL put list full case there is
2820                 * no need to worry about a race with locked
2821                 * sfxge_tx_qdpl_swizzle() as we know that the TX DPL put list
2822                 * was full and would have been swizzle'd to the TX DPL get
2823                 * list; hence guaranteeing future TX completions and calls
2824                 * to mac_tx_update() via sfxge_tx_qcomplete()
2825                 */
2826                goto fail4;
2827        }

2829        /* Try to grab the lock again */
2830        if (!locked)
2831                locked = mutex_tryenter(&(stp->st_lock));

2833        if (locked) {
```



```
2834                /* Try to service the list */
2835                sfxge_tx_qdpl_service(stp);
2836                /* lock has been dropped */
2837        }

2839        return (0);

2841 fail4:
2842        DTRACE_PROBE(fail4);
2843        sfxge_tx_packet_destroy(sp, stpp);
2844 fail3:
2845        DTRACE_PROBE(fail3);
2846        if (locked)
2847                mutex_exit(&(stp->st_lock));
2848 fail2:
2849        DTRACE_PROBE(fail2);
2850 fail1:
2851        DTRACE_PROBE1(fail1, int, rc);

2853        if (rc != ENOSPC)
2854                freemsg(mp);
2855        return (rc);
2856 }

2858 int
2859 sfxge_tx_loopback(sfxge_t *sp, unsigned int count)
2860 {
2861        uint8_t unicst[ETHERADDRL];
2862        size_t mtu;
2863        mblk_t *mp;
2864        struct ether_header *etherhp;
2865        unsigned int byte;
2866        int rc;

2868        if (count == 0) {
2869                rc = EINVAL;
2870                goto fail1;
2871        }

2873        rc = sfxge_mac_unicst_get(sp, SFXGE_UNICST_LAA, unicst);

2875        if (rc == ENOENT)
2876                rc = sfxge_mac_unicst_get(sp, SFXGE_UNICST_BIA, unicst);

2878        if (rc != 0)
2879                goto fail2;

2881        mtu = sp->s_mtu;

2883        if ((mp = allocb(sizeof (struct ether_header) + mtu,
2884            BPRI_HI)) == NULL) {
2885                rc = ENOMEM;
2886                goto fail3;
2887        }

2889        mp->b_wptr = mp->b_rptr + sizeof (struct ether_header);
2890        bzero(mp->b_rptr, MBLKL(mp));

2892        /*LINTED*/
2893        etherhp = (struct ether_header *)(mp->b_rptr);
2894        bcopy(sfxge_brdcst, &(etherhp->ether_dhost), ETHERADDRL);
2895        bcopy(unicst, &(etherhp->ether_shost), ETHERADDRL);
2896        etherhp->ether_type = htons(SFXGE_ETHERTYPE_LOOPBACK);

2898        for (byte = 0; byte < 30; byte++)
2899                *(mp->b_wptr++) = (byte & 1) ? 0xaa : 0x55;
```

```
2901         do {
2902                 mblk_t *nmp;

2904                 if ((nmp = dupb(mp)) == NULL) {
2905                         rc = ENOMEM;
2906                         goto fail4;
2907                 }

2909                 rc = sfxge_tx_packet_add(sp, nmp);
2910                 if (rc != 0) {
2911                         freeb(nmp);
2912                         goto fail5;
2913                 }

2915         } while (--count != 0);

2917         freeb(mp);
2918         return (0);

2920 fail5:
2921         DTRACE_PROBE(fail5);
2922 fail4:
2923         DTRACE_PROBE(fail4);

2925         freeb(mp);

2927 fail3:
2928         DTRACE_PROBE(fail3);
2929 fail2:
2930         DTRACE_PROBE(fail2);
2931 fail1:
2932         DTRACE_PROBE1(fail1, int, rc);

2934         return (rc);
2935 }

2937 int
2938 sfxge_tx_ioctl(sfxge_t *sp, sfxge_tx_ioc_t *stip)
2939 {
2940         int rc;

2942         switch (stip->sti_op) {
2943         case SFXGE_TX_OP_LOOPBACK: {
2944                 unsigned int count = stip->sti_data;

2946                 if ((rc = sfxge_tx_loopback(sp, count)) != 0)
2947                         goto fail1;

2949                 break;
2950         }
2951         default:
2952                 rc = ENOTSUP;
2953                 goto fail1;
2954         }

2956         return (0);

2958 fail1:
2959         DTRACE_PROBE1(fail1, int, rc);

2961         return (rc);
2962 }

2964 void
2965 sfxge_tx_stop(sfxge_t *sp)
```



```
2966 {
2967         efx_nic_t *enp = sp->s_enp;
2968         clock_t timeout;
2969         boolean_t do_flush;
2970         int index;

2972         ASSERT(mutex_owned(&(sp->s_state_lock)));

2974         mutex_enter(&(sp->s_tx_flush_lock));

2976         /* Flush all the queues */
2977         if (sp->s_hw_err == SFXGE_HW_OK) {
2978                 sp->s_tx_flush_pending = sp->s_tx_qcount;
2979                 do_flush = B_TRUE;
2980         } else {
2981                 sp->s_tx_flush_pending = 0;
2982                 do_flush = B_FALSE;
2983         }

2985         /* Prepare queues to stop and flush the hardware ring */
2986         for (index = 0; index < sp->s_tx_qcount; index++)
2987                 sfxge_tx_qflush(sp, index, do_flush);

2989         if (do_flush == B_FALSE)
2990                 goto flush_done;

2992         /* Wait upto 2sec for queue flushing to complete */
2993         timeout = ddi_get_lbolt() + drv_usectohz(SFXGE_TX_QFLUSH_USEC);

2995         while (sp->s_tx_flush_pending > 0) {
2996                 if (cv_timedwait(&(sp->s_tx_flush_kv), &(sp->s_tx_flush_lock),
2997                         timeout) < 0) {
2998                         /* Timeout waiting for queues to flush */
2999                         dev_info_t *dip = sp->s_dip;

3001                         DTRACE_PROBE(timeout);
3002                         cmn_err(CE_NOTE,
3003                             SFXGE_CMN_ERR "[%s%d] tx qflush timeout",
3004                             ddi_driver_name(dip), ddi_get_instance(dip));
3005                         break;
3006                 }
3007         }
3008         sp->s_tx_flush_pending = 0;

3010 flush_done:
3011         mutex_exit(&(sp->s_tx_flush_lock));

3013         /* Stop all the queues */
3014         for (index = 0; index < sp->s_tx_qcount; index++)
3015                 sfxge_tx_qstop(sp, index);

3017         /* Tear down the transmit module */
3018         efx_tx_fini(enp);
3019 }

3021 void
3022 sfxge_tx_fini(sfxge_t *sp)
3023 {
3024         int index;

3026         index = sp->s_tx_qcount;
3027         while (--index >= 0)
3028                 sfxge_tx_qfini(sp, index);

3030         kmem_cache_destroy(sp->s_tqc);
3031         sp->s_tqc = NULL;
```

```
3033         kmem_cache_destroy(sp->s_tmc);
3034         sp->s_tmc = NULL;

3036         kmem_cache_destroy(sp->s_tbc);
3037         sp->s_tbc = NULL;

3039         kmem_cache_destroy(sp->s_tpc);
3040         sp->s_tpc = NULL;
3041 }
3042 #endif /* ! codereview */
```

```
**********************************************************
    3727 Thu Aug 22 18:59:29 2013
new/usr/src/uts/common/io/sfxge/sfxge_vpd.c
Merged sfxge driver
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2009 Solarflare Communications Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#include <sys/types.h>
#include <sys/kmem.h>
#include "sfxge.h"


int
sfxge_vpd_get_keyword(sfxge_t *sp, sfxge_vpd_ioc_t *svip)
{
        efx_nic_t *enp = sp->s_enp;
        efx_vpd_value_t vpd;
        size_t size;
        void *buf;
        int rc;

        if ((rc = efx_vpd_size(enp, &size)) != 0)
                goto fail1;

        buf = kmem_zalloc(size, KM_SLEEP);
        ASSERT(buf);

        if ((rc = efx_vpd_read(enp, buf, size)) != 0)
                goto fail2;

        if ((rc = efx_vpd_verify(enp, buf, size)) != 0)
                goto fail3;

        vpd.evv_tag = svip->svi_tag;
        vpd.evv_keyword = svip->svi_keyword;

        if ((rc = efx_vpd_get(enp, buf, size, &vpd)) != 0)
                goto fail4;

        svip->svi_len = vpd.evv_length;
        EFX_STATIC_ASSERT(sizeof (svip->svi_payload) == sizeof (vpd.evv_value));
        memcpy(svip->svi_payload, &vpd.evv_value[0],
```



```
            sizeof (svip->svi_payload));

        kmem_free(buf, size);

        return (0);

fail4:
        DTRACE_PROBE(fail4);
fail3:
        DTRACE_PROBE(fail3);
fail2:
        DTRACE_PROBE(fail2);
        kmem_free(buf, size);
fail1:
        DTRACE_PROBE1(fail1, int, rc);

        return (rc);
}


int
sfxge_vpd_set_keyword(sfxge_t *sp, sfxge_vpd_ioc_t *svip)
{
        efx_nic_t *enp = sp->s_enp;
        efx_vpd_value_t vpd;
        size_t size;
        void *buf;
        int rc;

        /* restriction on writable tags is in efx_vpd_hunk_set() */

        if ((rc = efx_vpd_size(enp, &size)) != 0)
                goto fail1;

        buf = kmem_zalloc(size, KM_SLEEP);
        ASSERT(buf);

        if ((rc = efx_vpd_read(enp, buf, size)) != 0)
                goto fail2;

        if ((rc = efx_vpd_verify(enp, buf, size)) != 0) {
                if ((rc = efx_vpd_reinit(enp, buf, size)) != 0)
                        goto fail3;
                if ((rc = efx_vpd_verify(enp, buf, size)) != 0)
                        goto fail4;
        }

        vpd.evv_tag = svip->svi_tag;
        vpd.evv_keyword = svip->svi_keyword;
        vpd.evv_length = svip->svi_len;

        EFX_STATIC_ASSERT(sizeof (svip->svi_payload) == sizeof (vpd.evv_value));
        memcpy(&vpd.evv_value[0], svip->svi_payload,
            sizeof (svip->svi_payload));

        if ((rc = efx_vpd_set(enp, buf, size, &vpd)) != 0)
                goto fail5;

        if ((rc = efx_vpd_verify(enp, buf, size)) != 0)
                        goto fail6;

        /* And write the VPD back to the hardware */
        if ((rc = efx_vpd_write(enp, buf, size)) != 0)
                goto fail7;

        kmem_free(buf, size);
```

```
129         return (0);

131 fail7:
132         DTRACE_PROBE(fail7);
133 fail6:
134         DTRACE_PROBE(fail6);
135 fail5:
136         DTRACE_PROBE(fail5);
137 fail4:
138         DTRACE_PROBE(fail4);
139 fail3:
140         DTRACE_PROBE(fail3);
141 fail2:
142         DTRACE_PROBE(fail2);
143         kmem_free(buf, size);
144 fail1:
145         DTRACE_PROBE1(fail1, int, rc);

147         return (rc);
148 }


151 int
152 sfxge_vpd_ioctl(sfxge_t *sp, sfxge_vpd_ioc_t *svip)
153 {
154         int rc;

156         switch (svip->svi_op) {
157         case SFXGE_VPD_OP_GET_KEYWORD:
158                 if ((rc = sfxge_vpd_get_keyword(sp, svip)) != 0)
159                         goto fail1;
160                 break;
161         case SFXGE_VPD_OP_SET_KEYWORD:
162                 if ((rc = sfxge_vpd_set_keyword(sp, svip)) != 0)
163                         goto fail1;
164                 break;
165         default:
166                 rc = EINVAL;
167                 goto fail2;
168         }

170         return (0);

172 fail2:
173         DTRACE_PROBE(fail2);
174 fail1:
175         DTRACE_PROBE1(fail1, int, rc);

177         return (rc);
178 }
179 #endif /* ! codereview */
```

```
**********************************************************
    5388 Thu Aug 22 18:59:29 2013
new/usr/src/uts/common/io/sfxge/siena_flash.h
Merged sfxge driver
**********************************************************
```

```
/*-
 * Copyright 2007-2013 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */


#ifndef _SYS_SIENA_FLASH_H
#define _SYS_SIENA_FLASH_H

#pragma pack(1)

/* Fixed locations near the start of flash (which may be in the internal PHY
 * firmware header) point to the boot header.
 *
 * - parsed by MC boot ROM and firmware
 * - reserved (but not parsed) by PHY firmware
 * - opaque to driver
 */

#define SIENA_MC_BOOT_PHY_FW_HDR_LEN (0x20)

#define SIENA_MC_BOOT_PTR_LOCATION (0x18)      /* First thing we try to boot */
#define SIENA_MC_BOOT_ALT_PTR_LOCATION (0x1c)  /* Alternative if that fails */

#define SIENA_MC_BOOT_HDR_LEN (0x200)

#define SIENA_MC_BOOT_MAGIC (0x51E4A001)
#define SIENA_MC_BOOT_VERSION (1)

typedef struct siena_mc_boot_hdr_s {            /* GENERATED BY scripts/genfwdef
        efx_dword_t     magic;                  /* = SIENA_MC_BOOT_MAGIC */
        efx_word_t      hdr_version;            /* this structure definition is
        efx_byte_t      board_type;
        efx_byte_t      firmware_version_a;
        efx_byte_t      firmware_version_b;
        efx_byte_t      firmware_version_c;
        efx_word_t      checksum;               /* of whole header area + firmwa
        efx_word_t      firmware_version_d;
        efx_word_t      reserved_a[1];          /* (set to 0) */
        efx_dword_t     firmware_text_offset;   /* offset to firmware .text */
        efx_dword_t     firmware_text_size;     /* length of firmware .text, in
```



```
        efx_dword_t     firmware_data_offset;   /* offset to firmware .data */
        efx_dword_t     firmware_data_size;     /* length of firmware .data, in
        efx_dword_t     reserved_b[8];          /* (set to 0) */
} siena_mc_boot_hdr_t;

#define SIENA_MC_STATIC_CONFIG_MAGIC (0xBDCF5555)
#define SIENA_MC_STATIC_CONFIG_VERSION (0)

typedef struct siena_mc_static_config_hdr_s {   /* GENERATED BY scripts/genfwdef
        efx_dword_t     magic;                  /* = SIENA_MC_STATIC_CONFIG_MAGI
        efx_word_t      length;                 /* of header area (i.e. not incl
        efx_byte_t      version;
        efx_byte_t      csum;                   /* over header area (i.e. not in
        efx_dword_t     static_vpd_offset;
        efx_dword_t     static_vpd_length;
        efx_dword_t     capabilities;
        efx_byte_t      mac_addr_base[6];
        efx_byte_t      green_mode_cal;         /* Green mode calibration result
        efx_byte_t      green_mode_valid;       /* Whether cal holds a valid val
        efx_word_t      mac_addr_count;
        efx_word_t      mac_addr_stride;
        efx_word_t      calibrated_vref;
        efx_word_t      adc_vref;
        efx_dword_t     reserved2[1];           /* (write as zero) */
        efx_dword_t     num_dbi_items;
        struct {
                efx_word_t      addr;
                efx_word_t      byte_enables;
                efx_dword_t     value;
        } dbi[];
} siena_mc_static_config_hdr_t;

#define SIENA_MC_DYNAMIC_CONFIG_MAGIC (0xBDCFDDDD)
#define SIENA_MC_DYNAMIC_CONFIG_VERSION (0)

typedef struct siena_mc_fw_version_s {          /* GENERATED BY scripts/genfwdef
        efx_dword_t     fw_subtype;
        efx_word_t      version_w;
        efx_word_t      version_x;
        efx_word_t      version_y;
        efx_word_t      version_z;
} siena_mc_fw_version_t;

typedef struct siena_mc_dynamic_config_hdr_s {  /* GENERATED BY scripts/genfwdef
        efx_dword_t     magic;                  /* = SIENA_MC_DYNAMIC_CONFIG_MAG
        efx_word_t      length;                 /* of header area (i.e. not incl
        efx_byte_t      version;
        efx_byte_t      csum;                   /* over header area (i.e. not in
        efx_dword_t     dynamic_vpd_offset;
        efx_dword_t     dynamic_vpd_length;
        efx_dword_t     num_fw_version_items;
        siena_mc_fw_version_t   fw_version[];
} siena_mc_dynamic_config_hdr_t;

#define SIENA_MC_EXPROM_SINGLE_MAGIC (0xAA55)  /* little-endian uint16_t */

#define SIENA_MC_EXPROM_COMBO_MAGIC (0xB0070102)  /* little-endian uint32_t */

typedef struct siena_mc_combo_rom_hdr_s {       /* GENERATED BY scripts/genfwdef
        efx_dword_t     magic;                  /* = SIENA_MC_EXPROM_COMBO_MAGIC
        efx_dword_t     len1;                   /* length of first image */
        efx_dword_t     len2;                   /* length of second image */
        efx_dword_t     off1;                   /* offset of first byte to edit
        efx_dword_t     off2;                   /* offset of second byte to edit
        efx_word_t      infoblk0_off;           /* infoblk offset */
        efx_word_t      infoblk1_off;           /* infoblk offset */
```

```
128         efx_byte_t      infoblk_len;            /* length of space reserved for
129         efx_byte_t      reserved[7];            /* (set to 0) */
130 } siena_mc_combo_rom_hdr_t;

132 #pragma pack()

134 #endif  /* _SYS_SIENA_FLASH_H */
135 #endif /* ! codereview */
```

```
**********************************************************
   10500 Thu Aug 22 18:59:29 2013
new/usr/src/uts/common/io/sfxge/siena_impl.h
Merged sfxge driver
**********************************************************
```

```
/*-
 * Copyright 2009 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#ifndef _SYS_SIENA_IMPL_H
#define _SYS_SIENA_IMPL_H

#include "efx.h"
#include "efx_regs.h"
#include "efx_mcdi.h"
#include "siena_flash.h"

#ifdef  __cplusplus
extern "C" {
#endif

#if EFSYS_OPT_PHY_PROPS

/* START MKCONFIG GENERATED SienaPhyHeaderPropsBlock a8db1f8eb5106efd */
typedef enum siena_phy_prop_e {
        SIENA_PHY_NPROPS
} siena_phy_prop_t;

/* END MKCONFIG GENERATED SienaPhyHeaderPropsBlock */

#endif  /* EFSYS_OPT_PHY_PROPS */

#define SIENA_NVRAM_CHUNK 0x80

extern  __checkReturn   int
siena_nic_probe(
        __in            efx_nic_t *enp);

#if EFSYS_OPT_PCIE_TUNE

extern  __checkReturn   int
siena_nic_pcie_extended_sync(
        __in            efx_nic_t *enp);

#endif
```



```
extern  __checkReturn   int
siena_nic_reset(
        __in            efx_nic_t *enp);

extern  __checkReturn   int
siena_nic_init(
        __in            efx_nic_t *enp);

#if EFSYS_OPT_DIAG

extern  __checkReturn   int
siena_nic_register_test(
        __in            efx_nic_t *enp);

#endif  /* EFSYS_OPT_DIAG */

extern                  void
siena_nic_fini(
        __in            efx_nic_t *enp);

extern                  void
siena_nic_unprobe(
        __in            efx_nic_t *enp);

#define SIENA_SRAM_ROWS 0x12000

extern                  void
siena_sram_init(
        __in            efx_nic_t *enp);

#if EFSYS_OPT_DIAG

extern  __checkReturn   int
siena_sram_test(
        __in            efx_nic_t *enp,
        __in            efx_sram_pattern_fn_t func);

#endif  /* EFSYS_OPT_DIAG */


#if EFSYS_OPT_NVRAM || EFSYS_OPT_VPD

extern  __checkReturn           int
siena_nvram_partn_size(
        __in                    efx_nic_t *enp,
        __in                    unsigned int partn,
        __out                   size_t *sizep);

extern  __checkReturn           int
siena_nvram_partn_lock(
        __in                    efx_nic_t *enp,
        __in                    unsigned int partn);

extern  __checkReturn           int
siena_nvram_partn_read(
        __in                    efx_nic_t *enp,
        __in                    unsigned int partn,
        __in                    unsigned int offset,
        __out_bcount(size)      caddr_t data,
        __in                    size_t size);

extern  __checkReturn           int
siena_nvram_partn_erase(
        __in                    efx_nic_t *enp,
        __in                    unsigned int partn,
```

```
128         __in                    unsigned int offset,
129         __in                    size_t size);

131 extern  __checkReturn            int
132 siena_nvram_partn_write(
133         __in                    efx_nic_t *enp,
134         __in                    unsigned int partn,
135         __in                    unsigned int offset,
136         __out_bcount(size)      caddr_t data,
137         __in                    size_t size);

139 extern                          void
140 siena_nvram_partn_unlock(
141         __in                    efx_nic_t *enp,
142         __in                    unsigned int partn);

144 extern  __checkReturn           int
145 siena_nvram_get_dynamic_cfg(
146         __in                    efx_nic_t *enp,
147         __in                    unsigned int index,
148         __in                    boolean_t vpd,
149         __out                   siena_mc_dynamic_config_hdr_t **dcfgp,
150         __out                   size_t *sizep);

152 #endif  /* EFSYS_OPT_VPD || EFSYS_OPT_NVRAM */

154 #if EFSYS_OPT_NVRAM

156 #if EFSYS_OPT_DIAG

158 extern  __checkReturn           int
159 siena_nvram_test(
160         __in                    efx_nic_t *enp);

162 #endif  /* EFSYS_OPT_DIAG */

164 extern  __checkReturn           int
165 siena_nvram_size(
166         __in                    efx_nic_t *enp,
167         __in                    efx_nvram_type_t type,
168         __out                   size_t *sizep);

170 extern  __checkReturn           int
171 siena_nvram_get_version(
172         __in                    efx_nic_t *enp,
173         __in                    efx_nvram_type_t type,
174         __out                   uint32_t *subtypep,
175         __out_ecount(4)         uint16_t version[4]);

177 extern  __checkReturn           int
178 siena_nvram_rw_start(
179         __in                    efx_nic_t *enp,
180         __in                    efx_nvram_type_t type,
181         __out                   size_t *pref_chunkp);

183 extern  __checkReturn           int
184 siena_nvram_read_chunk(
185         __in                    efx_nic_t *enp,
186         __in                    efx_nvram_type_t type,
187         __in                    unsigned int offset,
188         __out_bcount(size)      caddr_t data,
189         __in                    size_t size);

191 extern   __checkReturn          int
192 siena_nvram_erase(
193         __in                    efx_nic_t *enp,
```



```
194         __in                    efx_nvram_type_t type);

196 extern  __checkReturn           int
197 siena_nvram_write_chunk(
198         __in                    efx_nic_t *enp,
199         __in                    efx_nvram_type_t type,
200         __in                    unsigned int offset,
201         __in_bcount(size)       caddr_t data,
202         __in                    size_t size);

204 extern                          void
205 siena_nvram_rw_finish(
206         __in                    efx_nic_t *enp,
207         __in                    efx_nvram_type_t type);

209 extern  __checkReturn           int
210 siena_nvram_set_version(
211         __in                    efx_nic_t *enp,
212         __in                    efx_nvram_type_t type,
213         __out                   uint16_t version[4]);

215 #endif  /* EFSYS_OPT_NVRAM */

217 #if EFSYS_OPT_VPD

219 extern  __checkReturn           int
220 siena_vpd_init(
221         __in                    efx_nic_t *enp);

223 extern  __checkReturn           int
224 siena_vpd_size(
225         __in                    efx_nic_t *enp,
226         __out                   size_t *sizep);

228 extern  __checkReturn           int
229 siena_vpd_read(
230         __in                    efx_nic_t *enp,
231         __out_bcount(size)      caddr_t data,
232         __in                    size_t size);

234 extern  __checkReturn           int
235 siena_vpd_verify(
236         __in                    efx_nic_t *enp,
237         __in_bcount(size)       caddr_t data,
238         __in                    size_t size);

240 extern  __checkReturn           int
241 siena_vpd_reinit(
242         __in                    efx_nic_t *enp,
243         __in_bcount(size)       caddr_t data,
244         __in                    size_t size);

246 extern  __checkReturn           int
247 siena_vpd_get(
248         __in                    efx_nic_t *enp,
249         __in_bcount(size)       caddr_t data,
250         __in                    size_t size,
251         __inout                 efx_vpd_value_t *evvp);

253 extern  __checkReturn           int
254 siena_vpd_set(
255         __in                    efx_nic_t *enp,
256         __in_bcount(size)       caddr_t data,
257         __in                    size_t size,
258         __in                    efx_vpd_value_t *evvp);
```

```
260 extern  __checkReturn           int
261 siena_vpd_next(
262         __in                    efx_nic_t *enp,
263         __in_bcount(size)       caddr_t data,
264         __in                    size_t size,
265         __out                   efx_vpd_value_t *evvp,
266         __inout                 unsigned int *contp);

268 extern __checkReturn            int
269 siena_vpd_write(
270         __in                    efx_nic_t *enp,
271         __in_bcount(size)       caddr_t data,
272         __in                    size_t size);

274 extern                          void
275 siena_vpd_fini(
276         __in                    efx_nic_t *enp);

278 #endif  /* EFSYS_OPT_VPD */

280 typedef struct siena_link_state_s {
281         uint32_t                sls_adv_cap_mask;
282         uint32_t                sls_lp_cap_mask;
283         unsigned int            sls_fcntl;
284         efx_link_mode_t         sls_link_mode;
285 #if EFSYS_OPT_LOOPBACK
286         efx_loopback_type_t     sls_loopback;
287 #endif
288         boolean_t               sls_mac_up;
289 } siena_link_state_t;

291 extern                  void
292 siena_phy_link_ev(
293         __in            efx_nic_t *enp,
294         __in            efx_qword_t *eqp,
295         __out           efx_link_mode_t *link_modep);

297 extern  __checkReturn   int
298 siena_phy_get_link(
299         __in            efx_nic_t *enp,
300         __out           siena_link_state_t *slsp);

302 extern  __checkReturn   int
303 siena_phy_power(
304         __in            efx_nic_t *enp,
305         __in            boolean_t on);

307 extern  __checkReturn   int
308 siena_phy_reconfigure(
309         __in            efx_nic_t *enp);

311 extern  __checkReturn   int
312 siena_phy_verify(
313         __in            efx_nic_t *enp);

315 extern  __checkReturn   int
316 siena_phy_oui_get(
317         __in            efx_nic_t *enp,
318         __out           uint32_t *ouip);

320 #if EFSYS_OPT_PHY_STATS

322 extern                                  void
323 siena_phy_decode_stats(
324         __in                            efx_nic_t *enp,
325         __in                            uint32_t vmask,
```



```
326         __in_opt                        efsys_mem_t *esmp,
327         __out_opt                       uint64_t *smaskp,
328         __out_ecount_opt(EFX_PHY_NSTATS)        uint32_t *stat);

330 extern  __checkReturn                   int
331 siena_phy_stats_update(
332         __in                            efx_nic_t *enp,
333         __in                            efsys_mem_t *esmp,
334         __out_ecount(EFX_PHY_NSTATS)    uint32_t *stat);

336 #endif  /* EFSYS_OPT_PHY_STATS */

338 #if EFSYS_OPT_PHY_PROPS

340 #if EFSYS_OPT_NAMES

342 extern          const char __cs *
343 siena_phy_prop_name(
344         __in    efx_nic_t *enp,
345         __in    unsigned int id);

347 #endif  /* EFSYS_OPT_NAMES */

349 extern  __checkReturn   int
350 siena_phy_prop_get(
351         __in            efx_nic_t *enp,
352         __in            unsigned int id,
353         __in            uint32_t flags,
354         __out           uint32_t *valp);

356 extern  __checkReturn   int
357 siena_phy_prop_set(
358         __in            efx_nic_t *enp,
359         __in            unsigned int id,
360         __in            uint32_t val);

362 #endif  /* EFSYS_OPT_PHY_PROPS */

364 #if EFSYS_OPT_PHY_BIST

366 extern  __checkReturn           int
367 siena_phy_bist_start(
368         __in                    efx_nic_t *enp,
369         __in                    efx_phy_bist_type_t type);

371 extern  __checkReturn           int
372 siena_phy_bist_poll(
373         __in                    efx_nic_t *enp,
374         __in                    efx_phy_bist_type_t type,
375         __out                   efx_phy_bist_result_t *resultp,
376         __out_opt __drv_when(count > 0, __notnull)
377         uint32_t        *value_maskp,
378         __out_ecount_opt(count) __drv_when(count > 0, __notnull)
379         unsigned long   *valuesp,
380         __in                    size_t count);

382 extern                          void
383 siena_phy_bist_stop(
384         __in                    efx_nic_t *enp,
385         __in                    efx_phy_bist_type_t type);

387 #endif  /* EFSYS_OPT_PHY_BIST */

389 extern  __checkReturn   int
390 siena_mac_poll(
391         __in            efx_nic_t *enp,
```

```
392         __out           efx_link_mode_t *link_modep);

394 extern  __checkReturn   int
395 siena_mac_up(
396         __in            efx_nic_t *enp,
397         __out           boolean_t *mac_upp);

399 extern  __checkReturn   int
400 siena_mac_reconfigure(
401         __in    efx_nic_t *enp);

403 #if EFSYS_OPT_LOOPBACK

405 extern  __checkReturn   int
406 siena_mac_loopback_set(
407         __in            efx_nic_t *enp,
408         __in            efx_link_mode_t link_mode,
409         __in            efx_loopback_type_t loopback_type);

411 #endif  /* EFSYS_OPT_LOOPBACK */

413 #if EFSYS_OPT_MAC_STATS

415 extern  __checkReturn                   int
416 siena_mac_stats_clear(
417         __in                            efx_nic_t *enp);

419 extern  __checkReturn                   int
420 siena_mac_stats_upload(
421         __in                            efx_nic_t *enp,
422         __in                            efsys_mem_t *esmp);

424 extern  __checkReturn                   int
425 siena_mac_stats_periodic(
426         __in                            efx_nic_t *enp,
427         __in                            efsys_mem_t *esmp,
428         __in                            uint16_t period_ms,
429         __in                            boolean_t events);

431 extern  __checkReturn                   int
432 siena_mac_stats_update(
433         __in                            efx_nic_t *enp,
434         __in                            efsys_mem_t *esmp,
435         __out_ecount(EFX_MAC_NSTATS)    efsys_stat_t *stat,
436         __out_opt                       uint32_t *generationp);

438 #endif  /* EFSYS_OPT_MAC_STATS */

440 extern  __checkReturn   int
441 siena_mon_reset(
442         __in            efx_nic_t *enp);

444 extern  __checkReturn   int
445 siena_mon_reconfigure(
446         __in            efx_nic_t *enp);

448 #if EFSYS_OPT_MON_STATS

450 extern                                  void
451 siena_mon_decode_stats(
452         __in                            efx_nic_t *enp,
453         __in                            uint32_t dmask,
454         __in_opt                        efsys_mem_t *esmp,
455         __out_opt                               uint32_t *vmaskp,
456         __out_ecount_opt(EFX_MON_NSTATS)        efx_mon_stat_value_t *value);
```



```
458 extern  __checkReturn                   int
459 siena_mon_ev(
460         __in                            efx_nic_t *enp,
461         __in                            efx_qword_t *eqp,
462         __out                           efx_mon_stat_t *idp,
463         __out                           efx_mon_stat_value_t *valuep);

465 extern  __checkReturn                   int
466 siena_mon_stats_update(
467         __in                            efx_nic_t *enp,
468         __in                            efsys_mem_t *esmp,
469         __out_ecount(EFX_MON_NSTATS)    efx_mon_stat_value_t *values);

471 #endif  /* EFSYS_OPT_MON_STATS */

473 #ifdef  __cplusplus
474 }
475 #endif

477 #endif  /* _SYS_SIENA_IMPL_H */
478 #endif /* ! codereview */
```

```
**********************************************************
   16649 Thu Aug 22 18:59:29 2013
new/usr/src/uts/common/io/sfxge/siena_mac.c
Merged sfxge driver
**********************************************************
```

```
/*
 * Copyright 2009 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */
#include "efsys.h"
#include "efx.h"
#include "efx_impl.h"

#if EFSYS_OPT_SIENA

        __checkReturn   int
siena_mac_poll(
        __in            efx_nic_t *enp,
        __out           efx_link_mode_t *link_modep)
{
        efx_port_t *epp = &(enp->en_port);
        siena_link_state_t sls;
        int rc;

        if ((rc = siena_phy_get_link(enp, &sls)) != 0)
                goto fail1;

        epp->ep_adv_cap_mask = sls.sls_adv_cap_mask;
        epp->ep_fcntl = sls.sls_fcntl;

        *link_modep = sls.sls_link_mode;

        return (0);

fail1:
        EFSYS_PROBE1(fail1, int, rc);

        *link_modep = EFX_LINK_UNKNOWN;

        return (rc);
}

        __checkReturn   int
siena_mac_up(
        __in            efx_nic_t *enp,
        __out           boolean_t *mac_upp)
```



```
{
        siena_link_state_t sls;
        int rc;

        /*
         * Because Siena doesn't *require* polling, we can't rely on
         * siena_mac_poll() being executed to populate epp->ep_mac_up.
         */
        if ((rc = siena_phy_get_link(enp, &sls)) != 0)
                goto fail1;

        *mac_upp = sls.sls_mac_up;

        return (0);

fail1:
        EFSYS_PROBE1(fail1, int, rc);

        return (rc);
}

        __checkReturn   int
siena_mac_reconfigure(
        __in            efx_nic_t *enp)
{
        efx_port_t *epp = &(enp->en_port);
        uint8_t payload[MAX(MC_CMD_SET_MAC_IN_LEN,
                            MC_CMD_SET_MCAST_HASH_IN_LEN)];
        efx_mcdi_req_t req;
        unsigned int fcntl;
        int rc;

        req.emr_cmd = MC_CMD_SET_MAC;
        req.emr_in_buf = payload;
        req.emr_in_length = MC_CMD_SET_MAC_IN_LEN;
        EFX_STATIC_ASSERT(MC_CMD_SET_MAC_OUT_LEN == 0);
        req.emr_out_buf = NULL;
        req.emr_out_length = 0;

        MCDI_IN_SET_DWORD(req, SET_MAC_IN_MTU, epp->ep_mac_pdu);
        MCDI_IN_SET_DWORD(req, SET_MAC_IN_DRAIN, epp->ep_mac_drain ? 1 : 0);
        EFX_MAC_ADDR_COPY(MCDI_IN2(req, uint8_t, SET_MAC_IN_ADDR),
                          epp->ep_mac_addr);
        MCDI_IN_POPULATE_DWORD_2(req, SET_MAC_IN_REJECT,
                                 SET_MAC_IN_REJECT_UNCST, !epp->ep_unicst,
                                 SET_MAC_IN_REJECT_BRDCST, !epp->ep_brdcst);

        if (epp->ep_fcntl_autoneg)
                /* efx_fcntl_set() has already set the phy capabilities */
                fcntl = MC_CMD_FCNTL_AUTO;
        else if (epp->ep_fcntl & EFX_FCNTL_RESPOND)
                fcntl = (epp->ep_fcntl & EFX_FCNTL_GENERATE)
                        ? MC_CMD_FCNTL_BIDIR
                        : MC_CMD_FCNTL_RESPOND;
        else
                fcntl = MC_CMD_FCNTL_OFF;

        MCDI_IN_SET_DWORD(req, SET_MAC_IN_FCNTL, fcntl);

        efx_mcdi_execute(enp, &req);

        if (req.emr_rc != 0) {
                rc = req.emr_rc;
                goto fail1;
        }
```

```
128         /* Push multicast hash. Set the broadcast bit (0xff) appropriately */
129         req.emr_cmd = MC_CMD_SET_MCAST_HASH;
130         req.emr_in_buf = payload;
131         req.emr_in_length = MC_CMD_SET_MCAST_HASH_IN_LEN;
132         EFX_STATIC_ASSERT(MC_CMD_SET_MCAST_HASH_OUT_LEN == 0);
133         req.emr_out_buf = NULL;
134         req.emr_out_length = 0;

136         memcpy(MCDI_IN2(req, uint8_t, SET_MCAST_HASH_IN_HASH0),
137             epp->ep_multicst_hash, sizeof (epp->ep_multicst_hash));
138         if (epp->ep_brdcst)
139                 EFX_SET_OWORD_BIT(*MCDI_IN2(req, efx_oword_t,
140                     SET_MCAST_HASH_IN_HASH1), 0x7f);

142         efx_mcdi_execute(enp, &req);

144         if (req.emr_rc != 0) {
145                 rc = req.emr_rc;
146                 goto fail2;
147         }

149         return (0);

151 fail2:
152         EFSYS_PROBE(fail2);
153 fail1:
154         EFSYS_PROBE1(fail1, int, rc);

156         return (rc);
157 }

159 #if EFSYS_OPT_LOOPBACK

161         __checkReturn   int
162 siena_mac_loopback_set(
163         __in            efx_nic_t *enp,
164         __in            efx_link_mode_t link_mode,
165         __in            efx_loopback_type_t loopback_type)
166 {
167         efx_port_t *epp = &(enp->en_port);
168         efx_phy_ops_t *epop = epp->ep_epop;
169         efx_loopback_type_t old_loopback_type;
170         efx_link_mode_t old_loopback_link_mode;
171         int rc;

173         /* The PHY object handles this on Siena */
174         old_loopback_type = epp->ep_loopback_type;
175         old_loopback_link_mode = epp->ep_loopback_link_mode;
176         epp->ep_loopback_type = loopback_type;
177         epp->ep_loopback_link_mode = link_mode;

179         if ((rc = epop->epo_reconfigure(enp)) != 0)
180                 goto fail1;

182         return (0);

184 fail1:
185         EFSYS_PROBE(fail2);

187         epp->ep_loopback_type = old_loopback_type;
188         epp->ep_loopback_link_mode = old_loopback_link_mode;

190         return (rc);
191 }

193 #endif  /* EFSYS_OPT_LOOPBACK */
```



```
195 #if EFSYS_OPT_MAC_STATS

197         __checkReturn                   int
198 siena_mac_stats_clear(
199         __in                            efx_nic_t *enp)
200 {
201         uint8_t payload[MC_CMD_MAC_STATS_IN_LEN];
202         efx_mcdi_req_t req;
203         int rc;

205         req.emr_cmd = MC_CMD_MAC_STATS;
206         req.emr_in_buf = payload;
207         req.emr_in_length = sizeof (payload);
208         EFX_STATIC_ASSERT(MC_CMD_MAC_STATS_OUT_DMA_LEN == 0);
209         req.emr_out_buf = NULL;
210         req.emr_out_length = 0;

212         MCDI_IN_POPULATE_DWORD_3(req, MAC_STATS_IN_CMD,
213                                  MAC_STATS_IN_DMA, 0,
214                                  MAC_STATS_IN_CLEAR, 1,
215                                  MAC_STATS_IN_PERIODIC_CHANGE, 0);

217         efx_mcdi_execute(enp, &req);

219         if (req.emr_rc != 0) {
220                 rc = req.emr_rc;
221                 goto fail1;
222         }

224         return (0);

226 fail1:
227         EFSYS_PROBE1(fail1, int, rc);

229         return (rc);
230 }

232         __checkReturn                   int
233 siena_mac_stats_upload(
234         __in                            efx_nic_t *enp,
235         __in                            efsys_mem_t *esmp)
236 {
237         uint8_t payload[MC_CMD_MAC_STATS_IN_LEN];
238         efx_mcdi_req_t req;
239         size_t bytes;
240         int rc;

242         EFX_STATIC_ASSERT(MC_CMD_MAC_NSTATS * sizeof (uint64_t) <=
243             EFX_MAC_STATS_SIZE);

245         bytes = MC_CMD_MAC_NSTATS * sizeof (uint64_t);

247         req.emr_cmd = MC_CMD_MAC_STATS;
248         req.emr_in_buf = payload;
249         req.emr_in_length = sizeof (payload);
250         EFX_STATIC_ASSERT(MC_CMD_MAC_STATS_OUT_DMA_LEN == 0);
251         req.emr_out_buf = NULL;
252         req.emr_out_length = 0;

254         MCDI_IN_SET_DWORD(req, MAC_STATS_IN_DMA_ADDR_LO,
255                           EFSYS_MEM_ADDR(esmp) & 0xffffffff);
256         MCDI_IN_SET_DWORD(req, MAC_STATS_IN_DMA_ADDR_HI,
257                           EFSYS_MEM_ADDR(esmp) >> 32);
258         MCDI_IN_SET_DWORD(req, MAC_STATS_IN_DMA_LEN, bytes);
```

```
260         /*
261          * The MC DMAs aggregate statistics for our convinience, so we can
262          * avoid having to pull the statistics buffer into the cache to
263          * maintain cumulative statistics.
264          */
265         MCDI_IN_POPULATE_DWORD_3(req, MAC_STATS_IN_CMD,
266                                  MAC_STATS_IN_DMA, 1,
267                                  MAC_STATS_IN_CLEAR, 0,
268                                  MAC_STATS_IN_PERIODIC_CHANGE, 0);

270         efx_mcdi_execute(enp, &req);

272         if (req.emr_rc != 0) {
273                 rc = req.emr_rc;
274                 goto fail1;
275         }

277         return (0);

279 fail1:
280         EFSYS_PROBE1(fail1, int, rc);

282         return (rc);
283 }

285         __checkReturn                   int
286 siena_mac_stats_periodic(
287         __in                            efx_nic_t *enp,
288         __in                            efsys_mem_t *esmp,
289         __in                            uint16_t period,
290         __in                            boolean_t events)
291 {
292         uint8_t payload[MC_CMD_MAC_STATS_IN_LEN];
293         efx_mcdi_req_t req;
294         size_t bytes;
295         int rc;

297         bytes = MC_CMD_MAC_NSTATS * sizeof (uint64_t);

299         req.emr_cmd = MC_CMD_MAC_STATS;
300         req.emr_in_buf = payload;
301         req.emr_in_length = sizeof (payload);
302         EFX_STATIC_ASSERT(MC_CMD_MAC_STATS_OUT_DMA_LEN == 0);
303         req.emr_out_buf = NULL;
304         req.emr_out_length = 0;

306         MCDI_IN_SET_DWORD(req, MAC_STATS_IN_DMA_ADDR_LO,
307                           EFSYS_MEM_ADDR(esmp) & 0xffffffff);
308         MCDI_IN_SET_DWORD(req, MAC_STATS_IN_DMA_ADDR_HI,
309                           EFSYS_MEM_ADDR(esmp) >> 32);
310         MCDI_IN_SET_DWORD(req, MAC_STATS_IN_DMA_LEN, bytes);

312         /*
313          * The MC DMAs aggregate statistics for our convinience, so we can
314          * avoid having to pull the statistics buffer into the cache to
315          * maintain cumulative statistics.
316          */
317         MCDI_IN_POPULATE_DWORD_6(req, MAC_STATS_IN_CMD,
318                                  MAC_STATS_IN_DMA, 0,
319                                  MAC_STATS_IN_CLEAR, 0,
320                                  MAC_STATS_IN_PERIODIC_CHANGE, 1,
321                                  MAC_STATS_IN_PERIODIC_ENABLE, period ? 1 : 0,
322                                  MAC_STATS_IN_PERIODIC_NOEVENT, events ? 0 : 1,
323                                  MAC_STATS_IN_PERIOD_MS, period);

325         efx_mcdi_execute(enp, &req);
```



```
327         if (req.emr_rc != 0) {
328                 rc = req.emr_rc;
329                 goto fail1;
330         }

332         return (0);

334 fail1:
335         EFSYS_PROBE1(fail1, int, rc);

337         return (rc);
338 }


341 #define SIENA_MAC_STAT_READ(_esmp, _field, _eqp)                        \
342         EFSYS_MEM_READQ((_esmp), (_field) * sizeof (efx_qword_t), _eqp)

344         __checkReturn                   int
345 siena_mac_stats_update(
346         __in                            efx_nic_t *enp,
347         __in                            efsys_mem_t *esmp,
348         __out_ecount(EFX_MAC_NSTATS)    efsys_stat_t *stat,
349         __out_opt                       uint32_t *generationp)
350 {
351         efx_qword_t rx_pkts;
352         efx_qword_t value;
353         efx_qword_t generation_start;
354         efx_qword_t generation_end;

356         _NOTE(ARGUNUSED(enp))

358         /* Read END first so we don't race with the MC */
359         SIENA_MAC_STAT_READ(esmp, MC_CMD_MAC_GENERATION_END,
360                             &generation_end);
361         EFSYS_MEM_READ_BARRIER();

363         /* TX */
364         SIENA_MAC_STAT_READ(esmp, MC_CMD_MAC_TX_PKTS, &value);
365         EFSYS_STAT_SET_QWORD(&(stat[EFX_MAC_TX_PKTS]), &value);
366         SIENA_MAC_STAT_READ(esmp, MC_CMD_MAC_TX_CONTROL_PKTS, &value);
367         EFSYS_STAT_SUBR_QWORD(&(stat[EFX_MAC_TX_PKTS]), &value);

369         SIENA_MAC_STAT_READ(esmp, MC_CMD_MAC_TX_PAUSE_PKTS, &value);
370         EFSYS_STAT_SET_QWORD(&(stat[EFX_MAC_TX_PAUSE_PKTS]), &value);

372         SIENA_MAC_STAT_READ(esmp, MC_CMD_MAC_TX_UNICAST_PKTS, &value);
373         EFSYS_STAT_SET_QWORD(&(stat[EFX_MAC_TX_UNICST_PKTS]), &value);

375         SIENA_MAC_STAT_READ(esmp, MC_CMD_MAC_TX_MULTICAST_PKTS, &value);
376         EFSYS_STAT_SET_QWORD(&(stat[EFX_MAC_TX_MULTICST_PKTS]), &value);

378         SIENA_MAC_STAT_READ(esmp, MC_CMD_MAC_TX_BROADCAST_PKTS, &value);
379         EFSYS_STAT_SET_QWORD(&(stat[EFX_MAC_TX_BRDCST_PKTS]), &value);

381         SIENA_MAC_STAT_READ(esmp, MC_CMD_MAC_TX_BYTES, &value);
382         EFSYS_STAT_SET_QWORD(&(stat[EFX_MAC_TX_OCTETS]), &value);

384         SIENA_MAC_STAT_READ(esmp, MC_CMD_MAC_TX_LT64_PKTS, &value);
385         EFSYS_STAT_SET_QWORD(&(stat[EFX_MAC_TX_LE_64_PKTS]), &value);
386         SIENA_MAC_STAT_READ(esmp, MC_CMD_MAC_TX_64_PKTS, &value);
387         EFSYS_STAT_SET_QWORD(&(stat[EFX_MAC_TX_LE_64_PKTS]), &value);

389         SIENA_MAC_STAT_READ(esmp, MC_CMD_MAC_TX_65_TO_127_PKTS, &value);
390         EFSYS_STAT_SET_QWORD(&(stat[EFX_MAC_TX_65_TO_127_PKTS]), &value);
```

```
392         SIENA_MAC_STAT_READ(esmp, MC_CMD_MAC_TX_128_TO_255_PKTS, &value);
393         EFSYS_STAT_SET_QWORD(&(stat[EFX_MAC_TX_128_TO_255_PKTS]), &value);
394
395         SIENA_MAC_STAT_READ(esmp, MC_CMD_MAC_TX_256_TO_511_PKTS, &value);
396         EFSYS_STAT_SET_QWORD(&(stat[EFX_MAC_TX_256_TO_511_PKTS]), &value);
397
398         SIENA_MAC_STAT_READ(esmp, MC_CMD_MAC_TX_512_TO_1023_PKTS, &value);
399         EFSYS_STAT_SET_QWORD(&(stat[EFX_MAC_TX_512_TO_1023_PKTS]), &value);
400
401         SIENA_MAC_STAT_READ(esmp, MC_CMD_MAC_TX_1024_TO_15XX_PKTS, &value);
402         EFSYS_STAT_SET_QWORD(&(stat[EFX_MAC_TX_1024_TO_15XX_PKTS]), &value);
403
404         SIENA_MAC_STAT_READ(esmp, MC_CMD_MAC_TX_15XX_TO_JUMBO_PKTS, &value);
405         EFSYS_STAT_SET_QWORD(&(stat[EFX_MAC_TX_GE_15XX_PKTS]), &value);
406         SIENA_MAC_STAT_READ(esmp, MC_CMD_MAC_TX_GTJUMBO_PKTS, &value);
407         EFSYS_STAT_INCR_QWORD(&(stat[EFX_MAC_TX_GE_15XX_PKTS]), &value);
408
409         SIENA_MAC_STAT_READ(esmp, MC_CMD_MAC_TX_BAD_FCS_PKTS, &value);
410         EFSYS_STAT_SET_QWORD(&(stat[EFX_MAC_TX_ERRORS]), &value);
411
412         SIENA_MAC_STAT_READ(esmp, MC_CMD_MAC_TX_SINGLE_COLLISION_PKTS, &value);
413         EFSYS_STAT_SET_QWORD(&(stat[EFX_MAC_TX_SGL_COL_PKTS]), &value);
414
415         SIENA_MAC_STAT_READ(esmp, MC_CMD_MAC_TX_MULTIPLE_COLLISION_PKTS,
416                             &value);
417         EFSYS_STAT_SET_QWORD(&(stat[EFX_MAC_TX_MULT_COL_PKTS]), &value);
418
419         SIENA_MAC_STAT_READ(esmp, MC_CMD_MAC_TX_EXCESSIVE_COLLISION_PKTS,
420                             &value);
421         EFSYS_STAT_SET_QWORD(&(stat[EFX_MAC_TX_EX_COL_PKTS]), &value);
422
423         SIENA_MAC_STAT_READ(esmp, MC_CMD_MAC_TX_LATE_COLLISION_PKTS, &value);
424         EFSYS_STAT_SET_QWORD(&(stat[EFX_MAC_TX_LATE_COL_PKTS]), &value);
425
426         SIENA_MAC_STAT_READ(esmp, MC_CMD_MAC_TX_DEFERRED_PKTS, &value);
427         EFSYS_STAT_SET_QWORD(&(stat[EFX_MAC_TX_DEF_PKTS]), &value);
428
429         SIENA_MAC_STAT_READ(esmp, MC_CMD_MAC_TX_EXCESSIVE_DEFERRED_PKTS,
430             &value);
431         EFSYS_STAT_SET_QWORD(&(stat[EFX_MAC_TX_EX_DEF_PKTS]), &value);
432
433         /* RX */
434         SIENA_MAC_STAT_READ(esmp, MC_CMD_MAC_RX_BYTES, &rx_pkts);
435         EFSYS_STAT_SET_QWORD(&(stat[EFX_MAC_RX_OCTETS]), &rx_pkts);
436
437         SIENA_MAC_STAT_READ(esmp, MC_CMD_MAC_RX_PKTS, &value);
438         EFSYS_STAT_SET_QWORD(&(stat[EFX_MAC_RX_PKTS]), &value);
439
440         SIENA_MAC_STAT_READ(esmp, MC_CMD_MAC_RX_UNICAST_PKTS, &value);
441         EFSYS_STAT_SET_QWORD(&(stat[EFX_MAC_RX_UNICST_PKTS]), &value);
442
443         SIENA_MAC_STAT_READ(esmp, MC_CMD_MAC_RX_MULTICAST_PKTS, &value);
444         EFSYS_STAT_SET_QWORD(&(stat[EFX_MAC_RX_MULTICST_PKTS]), &value);
445
446         SIENA_MAC_STAT_READ(esmp, MC_CMD_MAC_RX_BROADCAST_PKTS, &value);
447         EFSYS_STAT_SET_QWORD(&(stat[EFX_MAC_RX_BRDCST_PKTS]), &value);
448
449         SIENA_MAC_STAT_READ(esmp, MC_CMD_MAC_RX_PAUSE_PKTS, &value);
450         EFSYS_STAT_SET_QWORD(&(stat[EFX_MAC_RX_PAUSE_PKTS]), &value);
451
452         SIENA_MAC_STAT_READ(esmp, MC_CMD_MAC_RX_UNDERSIZE_PKTS, &value);
453         EFSYS_STAT_SET_QWORD(&(stat[EFX_MAC_RX_LE_64_PKTS]), &value);
454         SIENA_MAC_STAT_READ(esmp, MC_CMD_MAC_RX_64_PKTS, &value);
455         EFSYS_STAT_INCR_QWORD(&(stat[EFX_MAC_RX_LE_64_PKTS]), &value);
456
457         SIENA_MAC_STAT_READ(esmp, MC_CMD_MAC_RX_65_TO_127_PKTS, &value);
```



```
458         EFSYS_STAT_SET_QWORD(&(stat[EFX_MAC_RX_65_TO_127_PKTS]), &value);
459
460         SIENA_MAC_STAT_READ(esmp, MC_CMD_MAC_RX_128_TO_255_PKTS, &value);
461         EFSYS_STAT_SET_QWORD(&(stat[EFX_MAC_RX_128_TO_255_PKTS]), &value);
462
463         SIENA_MAC_STAT_READ(esmp, MC_CMD_MAC_RX_256_TO_511_PKTS, &value);
464         EFSYS_STAT_SET_QWORD(&(stat[EFX_MAC_RX_256_TO_511_PKTS]), &value);
465
466         SIENA_MAC_STAT_READ(esmp, MC_CMD_MAC_RX_512_TO_1023_PKTS, &value);
467         EFSYS_STAT_SET_QWORD(&(stat[EFX_MAC_RX_512_TO_1023_PKTS]), &value);
468
469         SIENA_MAC_STAT_READ(esmp, MC_CMD_MAC_RX_1024_TO_15XX_PKTS, &value);
470         EFSYS_STAT_SET_QWORD(&(stat[EFX_MAC_RX_1024_TO_15XX_PKTS]), &value);
471
472         SIENA_MAC_STAT_READ(esmp, MC_CMD_MAC_RX_15XX_TO_JUMBO_PKTS, &value);
473         EFSYS_STAT_SET_QWORD(&(stat[EFX_MAC_RX_GE_15XX_PKTS]), &value);
474         SIENA_MAC_STAT_READ(esmp, MC_CMD_MAC_RX_GTJUMBO_PKTS, &value);
475         EFSYS_STAT_INCR_QWORD(&(stat[EFX_MAC_RX_GE_15XX_PKTS]), &value);
476
477         SIENA_MAC_STAT_READ(esmp, MC_CMD_MAC_RX_BAD_FCS_PKTS, &value);
478         EFSYS_STAT_SET_QWORD(&(stat[EFX_MAC_RX_FCS_ERRORS]), &value);
479
480         SIENA_MAC_STAT_READ(esmp, MC_CMD_MAC_RX_OVERFLOW_PKTS, &value);
481         EFSYS_STAT_SET_QWORD(&(stat[EFX_MAC_RX_DROP_EVENTS]), &value);
482
483         SIENA_MAC_STAT_READ(esmp, MC_CMD_MAC_RX_FALSE_CARRIER_PKTS, &value);
484         EFSYS_STAT_SET_QWORD(&(stat[EFX_MAC_RX_FALSE_CARRIER_ERRORS]), &value);
485
486         SIENA_MAC_STAT_READ(esmp, MC_CMD_MAC_RX_SYMBOL_ERROR_PKTS, &value);
487         EFSYS_STAT_SET_QWORD(&(stat[EFX_MAC_RX_SYMBOL_ERRORS]), &value);
488
489         SIENA_MAC_STAT_READ(esmp, MC_CMD_MAC_RX_ALIGN_ERROR_PKTS, &value);
490         EFSYS_STAT_SET_QWORD(&(stat[EFX_MAC_RX_ALIGN_ERRORS]), &value);
491
492         SIENA_MAC_STAT_READ(esmp, MC_CMD_MAC_RX_INTERNAL_ERROR_PKTS, &value);
493         EFSYS_STAT_SET_QWORD(&(stat[EFX_MAC_RX_INTERNAL_ERRORS]), &value);
494
495         SIENA_MAC_STAT_READ(esmp, MC_CMD_MAC_RX_JABBER_PKTS, &value);
496         EFSYS_STAT_SET_QWORD(&(stat[EFX_MAC_RX_JABBER_PKTS]), &value);
497
498         SIENA_MAC_STAT_READ(esmp, MC_CMD_MAC_RX_LANES01_CHAR_ERR, &value);
499         EFSYS_STAT_SET_DWORD(&(stat[EFX_MAC_RX_LANE0_CHAR_ERR]),
500                              &(value.eq_dword[0]));
501         EFSYS_STAT_SET_DWORD(&(stat[EFX_MAC_RX_LANE1_CHAR_ERR]),
502                              &(value.eq_dword[1]));
503
504         SIENA_MAC_STAT_READ(esmp, MC_CMD_MAC_RX_LANES23_CHAR_ERR, &value);
505         EFSYS_STAT_SET_DWORD(&(stat[EFX_MAC_RX_LANE2_CHAR_ERR]),
506                              &(value.eq_dword[0]));
507         EFSYS_STAT_SET_DWORD(&(stat[EFX_MAC_RX_LANE3_CHAR_ERR]),
508                              &(value.eq_dword[1]));
509
510         SIENA_MAC_STAT_READ(esmp, MC_CMD_MAC_RX_LANES01_DISP_ERR, &value);
511         EFSYS_STAT_SET_DWORD(&(stat[EFX_MAC_RX_LANE0_DISP_ERR]),
512                              &(value.eq_dword[0]));
513         EFSYS_STAT_SET_DWORD(&(stat[EFX_MAC_RX_LANE1_DISP_ERR]),
514                              &(value.eq_dword[1]));
515
516         SIENA_MAC_STAT_READ(esmp, MC_CMD_MAC_RX_LANES23_DISP_ERR, &value);
517         EFSYS_STAT_SET_DWORD(&(stat[EFX_MAC_RX_LANE2_DISP_ERR]),
518                              &(value.eq_dword[0]));
519         EFSYS_STAT_SET_DWORD(&(stat[EFX_MAC_RX_LANE3_DISP_ERR]),
520                              &(value.eq_dword[1]));
521
522         SIENA_MAC_STAT_READ(esmp, MC_CMD_MAC_RX_MATCH_FAULT, &value);
523         EFSYS_STAT_SET_QWORD(&(stat[EFX_MAC_RX_MATCH_FAULT]), &value);
```

```
525         SIENA_MAC_STAT_READ(esmp, MC_CMD_MAC_RX_NODESC_DROPS, &value);
526         EFSYS_STAT_SET_QWORD(&(stat[EFX_MAC_RX_NODESC_DROP_CNT]), &value);

528         EFSYS_MEM_READ_BARRIER();
529         SIENA_MAC_STAT_READ(esmp, MC_CMD_MAC_GENERATION_START,
530                             &generation_start);

532         /* Check that we didn't read the stats in the middle of a DMA */
533         /* Not a good enough check ? */
534         if (memcmp(&generation_start, &generation_end,
535             sizeof (generation_start)))
536                 return (EAGAIN);

538         if (generationp)
539                 *generationp = EFX_QWORD_FIELD(generation_start, EFX_DWORD_0);

541         return (0);
542 }

544 #endif  /* EFSYS_OPT_MAC_STATS */

546 #endif  /* EFSYS_OPT_SIENA */
547 #endif /* ! codereview */
```

```
**********************************************************
    7635 Thu Aug 22 18:59:29 2013
new/usr/src/uts/common/io/sfxge/siena_mon.c
Merged sfxge driver
**********************************************************
```

```
/*
 * Copyright 2009 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */
#include "efsys.h"
#include "efx.h"
#include "efx_impl.h"

#if EFSYS_OPT_MON_SIENA

        __checkReturn   int
siena_mon_reset(
        __in            efx_nic_t *enp)
{
        _NOTE(ARGUNUSED(enp))
        EFSYS_ASSERT(enp->en_family == EFX_FAMILY_SIENA);

        return (0);
}

        __checkReturn   int
siena_mon_reconfigure(
        __in            efx_nic_t *enp)
{
        _NOTE(ARGUNUSED(enp))
        EFSYS_ASSERT(enp->en_family == EFX_FAMILY_SIENA);

        return (0);
}

#if EFSYS_OPT_MON_STATS

#define SIENA_MON_WRONG_PORT (uint16_t)0xffff

static __cs uint16_t __siena_mon_port0_map[] = {
        EFX_MON_STAT_INT_TEMP,          /* MC_CMD_SENSOR_CONTROLLER_TEMP */
        EFX_MON_STAT_EXT_TEMP,          /* MC_CMD_SENSOR_PHY_COMMON_TEMP */
        EFX_MON_STAT_INT_COOLING,       /* MC_CMD_SENSOR_CONTROLLER_COOLING */
        EFX_MON_STAT_EXT_TEMP,          /* MC_CMD_SENSOR_PHY0_TEMP */
        EFX_MON_STAT_EXT_COOLING,       /* MC_CMD_SENSOR_PHY0_COOLING */
        SIENA_MON_WRONG_PORT,           /* MC_CMD_SENSOR_PHY1_TEMP */
```



```
        SIENA_MON_WRONG_PORT,           /* MC_CMD_SENSOR_PHY1_COOLING */
        EFX_MON_STAT_1V,                /* MC_CMD_SENSOR_IN_1V0 */
        EFX_MON_STAT_1_2V,              /* MC_CMD_SENSOR_IN_1V2 */
        EFX_MON_STAT_1_8V,              /* MC_CMD_SENSOR_IN_1V8 */
        EFX_MON_STAT_2_5V,              /* MC_CMD_SENSOR_IN_2V5 */
        EFX_MON_STAT_3_3V,              /* MC_CMD_SENSOR_IN_3V3 */
        EFX_MON_STAT_12V,               /* MC_CMD_SENSOR_IN_12V0 */
        EFX_MON_STAT_1_2VA,             /* MC_CMD_SENSOR_IN_1V2A */
        EFX_MON_STAT_VREF,              /* MC_CMD_SENSOR_IN_VREF */
};

static __cs uint16_t __siena_mon_port1_map[] = {
        EFX_MON_STAT_INT_TEMP,          /* MC_CMD_SENSOR_CONTROLLER_TEMP */
        EFX_MON_STAT_EXT_TEMP,          /* MC_CMD_SENSOR_PHY_COMMON_TEMP */
        EFX_MON_STAT_INT_COOLING,       /* MC_CMD_SENSOR_CONTROLLER_COOLING */
        SIENA_MON_WRONG_PORT,           /* MC_CMD_SENSOR_PHY0_TEMP */
        SIENA_MON_WRONG_PORT,           /* MC_CMD_SENSOR_PHY0_COOLING */
        EFX_MON_STAT_EXT_TEMP,          /* MC_CMD_SENSOR_PHY1_TEMP */
        EFX_MON_STAT_EXT_COOLING,       /* MC_CMD_SENSOR_PHY1_COOLING */
        EFX_MON_STAT_1V,                /* MC_CMD_SENSOR_IN_1V0 */
        EFX_MON_STAT_1_2V,              /* MC_CMD_SENSOR_IN_1V2 */
        EFX_MON_STAT_1_8V,              /* MC_CMD_SENSOR_IN_1V8 */
        EFX_MON_STAT_2_5V,              /* MC_CMD_SENSOR_IN_2V5 */
        EFX_MON_STAT_3_3V,              /* MC_CMD_SENSOR_IN_3V3 */
        EFX_MON_STAT_12V,               /* MC_CMD_SENSOR_IN_12V0 */
        EFX_MON_STAT_1_2VA,             /* MC_CMD_SENSOR_IN_1V2A */
        EFX_MON_STAT_VREF,              /* MC_CMD_SENSOR_IN_VREF */
};

#define SIENA_STATIC_SENSOR_ASSERT(_field)                              \
        EFX_STATIC_ASSERT(MC_CMD_SENSOR_STATE_ ## _field                \
                          == EFX_MON_STAT_STATE_ ## _field)

                                        void
siena_mon_decode_stats(
        __in                            efx_nic_t *enp,
        __in                            uint32_t dmask,
        __in_opt                        efsys_mem_t *esmp,
        __out_opt                       uint32_t *vmaskp,
        __out_ecount_opt(EFX_MON_NSTATS)        efx_mon_stat_value_t *value)
{
        efx_mcdi_iface_t *emip = &(enp->en_u.siena.enu_mip);
        uint16_t *sensor_map;
        uint16_t mc_sensor;
        size_t mc_sensor_max;
        uint32_t vmask = 0;
        uint32_t idx = 0;

        /* Assert the MC_CMD_SENSOR and EFX_MON_STATE namespaces agree */
        SIENA_STATIC_SENSOR_ASSERT(OK);
        SIENA_STATIC_SENSOR_ASSERT(WARNING);
        SIENA_STATIC_SENSOR_ASSERT(FATAL);
        SIENA_STATIC_SENSOR_ASSERT(BROKEN);

        EFX_STATIC_ASSERT(sizeof (__siena_mon_port1_map)
                          == sizeof (__siena_mon_port0_map));
        mc_sensor_max = EFX_ARRAY_SIZE(__siena_mon_port0_map);
        sensor_map = (emip->emi_port == 1)
                ? __siena_mon_port0_map
                : __siena_mon_port1_map;

        /*
         * dmask may legitimately contain sensors not understood by the driver
         */
        for (mc_sensor = 0; mc_sensor < mc_sensor_max; ++mc_sensor) {
                uint16_t efx_sensor = sensor_map[mc_sensor];
```

```
129                 if (~dmask & (1 << mc_sensor))
130                         continue;
131                 idx++;
132
133                 if (efx_sensor == SIENA_MON_WRONG_PORT)
134                         continue;
135                 EFSYS_ASSERT(efx_sensor < EFX_MON_NSTATS);
136
137                 vmask |= (1 << efx_sensor);
138                 if (value != NULL && esmp != NULL && !EFSYS_MEM_IS_NULL(esmp)) {
139                         efx_mon_stat_value_t *emsvp = value + efx_sensor;
140                         efx_dword_t dword;
141                         EFSYS_MEM_READD(esmp, 4 * (idx - 1), &dword);
142                         emsvp->emsv_value =
143                             (uint16_t)EFX_DWORD_FIELD(
144                                 dword,
145                                 MC_CMD_SENSOR_VALUE_ENTRY_TYPEDEF_VALUE);
146                         emsvp->emsv_state =
147                             (uint16_t)EFX_DWORD_FIELD(
148                                 dword,
149                                 MC_CMD_SENSOR_VALUE_ENTRY_TYPEDEF_STATE);
150                 }
151         }
152
153         if (vmaskp != NULL)
154                 *vmaskp = vmask;
155 }
156
157         __checkReturn                   int
158 siena_mon_ev(
159         __in                            efx_nic_t *enp,
160         __in                            efx_qword_t *eqp,
161         __out                           efx_mon_stat_t *idp,
162         __out                           efx_mon_stat_value_t *valuep)
163 {
164         efx_mcdi_iface_t *emip = &(enp->en_u.siena.enu_mip);
165         efx_nic_cfg_t *encp = &(enp->en_nic_cfg);
166         uint16_t ev_monitor;
167         uint16_t ev_state;
168         uint16_t ev_value;
169         uint16_t *sensor_map;
170         efx_mon_stat_t id;
171         int rc;
172
173         sensor_map = (emip->emi_port == 1)
174                 ? __siena_mon_port0_map
175                 : __siena_mon_port1_map;
176
177         ev_monitor = (uint16_t)MCDI_EV_FIELD(eqp, SENSOREVT_MONITOR);
178         ev_state = (uint16_t)MCDI_EV_FIELD(eqp, SENSOREVT_STATE);
179         ev_value = (uint16_t)MCDI_EV_FIELD(eqp, SENSOREVT_VALUE);
180
181         /* Hardware must support this statistic */
182         EFSYS_ASSERT((1 << ev_monitor) & encp->enc_siena_mon_stat_mask);
183
184         /* But we don't have to understand it */
185         if (ev_monitor >= EFX_ARRAY_SIZE(__siena_mon_port0_map)) {
186                 rc = ENOTSUP;
187                 goto fail1;
188         }
189
190         id = sensor_map[ev_monitor];
191         if (id == SIENA_MON_WRONG_PORT)
192                 return (ENODEV);
193         EFSYS_ASSERT(id < EFX_MON_NSTATS);
```



```
194
195         *idp = id;
196         valuep->emsv_value = ev_value;
197         valuep->emsv_state = ev_state;
198
199         return (0);
200
201 fail1:
202         EFSYS_PROBE1(fail1, int, rc);
203
204         return (rc);
205 }
206
207         __checkReturn                   int
208 siena_mon_stats_update(
209         __in                            efx_nic_t *enp,
210         __in                            efsys_mem_t *esmp,
211         __out_ecount(EFX_MON_NSTATS)    efx_mon_stat_value_t *values)
212 {
213         efx_nic_cfg_t *encp = &(enp->en_nic_cfg);
214         uint32_t dmask = encp->enc_siena_mon_stat_mask;
215         uint32_t vmask;
216         uint8_t payload[MC_CMD_READ_SENSORS_IN_LEN];
217         efx_mcdi_req_t req;
218         int rc;
219
220         EFSYS_ASSERT(enp->en_family == EFX_FAMILY_SIENA);
221
222         req.emr_cmd = MC_CMD_READ_SENSORS;
223         req.emr_in_buf = payload;
224         req.emr_in_length = sizeof (payload);
225         EFX_STATIC_ASSERT(MC_CMD_READ_SENSORS_OUT_LEN == 0);
226         req.emr_out_buf = NULL;
227         req.emr_out_length = 0;
228
229         MCDI_IN_SET_DWORD(req, READ_SENSORS_IN_DMA_ADDR_LO,
230                           EFSYS_MEM_ADDR(esmp) & 0xffffffff);
231         MCDI_IN_SET_DWORD(req, READ_SENSORS_IN_DMA_ADDR_HI,
232                           EFSYS_MEM_ADDR(esmp) >> 32);
233
234         efx_mcdi_execute(enp, &req);
235
236         if (req.emr_rc != 0) {
237                 rc = req.emr_rc;
238                 goto fail1;
239         }
240
241         siena_mon_decode_stats(enp, dmask, esmp, &vmask, values);
242         EFSYS_ASSERT(vmask == encp->enc_mon_stat_mask);
243
244         return (0);
245
246 fail1:
247         EFSYS_PROBE1(fail1, int, rc);
248
249         return (rc);
250 }
251
252 #endif  /* EFSYS_OPT_MON_STATS */
253
254 #endif  /* EFSYS_OPT_MON_SIENA */
255 #endif /* ! codereview */
```

```
**********************************************************
   24708 Thu Aug 22 18:59:29 2013
new/usr/src/uts/common/io/sfxge/siena_nic.c
Merged sfxge driver
**********************************************************
```

```
/*
 * Copyright 2009 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */
#include "efsys.h"
#include "efx.h"
#include "efx_impl.h"

#if EFSYS_OPT_SIENA

static  __checkReturn           int
siena_nic_get_partn_mask(
        __in                    efx_nic_t *enp,
        __out                   unsigned int *maskp)
{
        efx_mcdi_req_t req;
        uint8_t outbuf[MC_CMD_NVRAM_TYPES_OUT_LEN];
        int rc;

        req.emr_cmd = MC_CMD_NVRAM_TYPES;
        EFX_STATIC_ASSERT(MC_CMD_NVRAM_TYPES_IN_LEN == 0);
        req.emr_in_buf = NULL;
        req.emr_in_length = 0;
        req.emr_out_buf = outbuf;
        req.emr_out_length = sizeof (outbuf);

        efx_mcdi_execute(enp, &req);

        if (req.emr_rc != 0) {
                rc = req.emr_rc;
                goto fail1;
        }

        if (req.emr_out_length_used < MC_CMD_NVRAM_TYPES_OUT_LEN) {
                rc = EMSGSIZE;
                goto fail2;
        }

        *maskp = MCDI_OUT_DWORD(req, NVRAM_TYPES_OUT_TYPES);

        return (0);
```



```
fail2:
        EFSYS_PROBE(fail2);
fail1:
        EFSYS_PROBE1(fail1, int, rc);

        return (rc);
}

static  __checkReturn   int
siena_nic_exit_assertion_handler(
        __in            efx_nic_t *enp)
{
        efx_mcdi_req_t req;
        uint8_t payload[MC_CMD_REBOOT_IN_LEN];
        int rc;

        req.emr_cmd = MC_CMD_REBOOT;
        req.emr_in_buf = payload;
        req.emr_in_length = MC_CMD_REBOOT_IN_LEN;
        EFX_STATIC_ASSERT(MC_CMD_REBOOT_OUT_LEN == 0);
        req.emr_out_buf = NULL;
        req.emr_out_length = 0;

        MCDI_IN_SET_DWORD(req, REBOOT_IN_FLAGS,
                          MC_CMD_REBOOT_FLAGS_AFTER_ASSERTION);

        efx_mcdi_execute(enp, &req);

        if (req.emr_rc != 0 && req.emr_rc != EIO) {
                rc = req.emr_rc;
                goto fail1;
        }

        return (0);

fail1:
        EFSYS_PROBE1(fail1, int, rc);

        return (rc);
}

static  __checkReturn   int
siena_nic_read_assertion(
        __in            efx_nic_t *enp)
{
        efx_mcdi_req_t req;
        uint8_t payload[MAX(MC_CMD_GET_ASSERTS_IN_LEN,
                            MC_CMD_GET_ASSERTS_OUT_LEN)];
        const char *reason;
        unsigned int flags;
        unsigned int index;
        unsigned int ofst;
        int retry;
        int rc;

        /*
         * Before we attempt to chat to the MC, we should verify that the MC
         * isn't in it's assertion handler, either due to a previous reboot,
         * or because we're reinitializing due to an eec_exception().
         *
         * Use GET_ASSERTS to read any assertion state that may be present.
         * Retry this command twice. Once because a boot-time assertion failure
         * might cause the 1st MCDI request to fail. And once again because
         * we might race with siena_nic_exit_assertion_handler() running on the
         * other port.
```

```
128          */
129         retry = 2;
130         do {
131                 req.emr_cmd = MC_CMD_GET_ASSERTS;
132                 req.emr_in_buf = payload;
133                 req.emr_in_length = MC_CMD_GET_ASSERTS_IN_LEN;
134                 req.emr_out_buf = payload;
135                 req.emr_out_length = MC_CMD_GET_ASSERTS_OUT_LEN;

137                 MCDI_IN_SET_DWORD(req, GET_ASSERTS_IN_CLEAR, 1);
138                 efx_mcdi_execute(enp, &req);

140         } while ((req.emr_rc == EINTR || req.emr_rc == EIO) && retry-- > 0);

142         if (req.emr_rc != 0) {
143                 rc = req.emr_rc;
144                 goto fail1;
145         }

147         if (req.emr_out_length_used < MC_CMD_GET_ASSERTS_OUT_LEN) {
148                 rc = EMSGSIZE;
149                 goto fail2;
150         }

152         /* Print out any assertion state recorded */
153         flags = MCDI_OUT_DWORD(req, GET_ASSERTS_OUT_GLOBAL_FLAGS);
154         if (flags == MC_CMD_GET_ASSERTS_FLAGS_NO_FAILS)
155                 return (0);

157         reason = (flags == MC_CMD_GET_ASSERTS_FLAGS_SYS_FAIL)
158                 ? "system-level assertion"
159                 : (flags == MC_CMD_GET_ASSERTS_FLAGS_THR_FAIL)
160                 ? "thread-level assertion"
161                 : (flags == MC_CMD_GET_ASSERTS_FLAGS_WDOG_FIRED)
162                 ? "watchdog reset"
163                 : "unknown assertion";
164         EFSYS_PROBE3(mcpu_assertion,
165             const char *, reason, unsigned int,
166             MCDI_OUT_DWORD(req, GET_ASSERTS_OUT_SAVED_PC_OFFS),
167             unsigned int,
168             MCDI_OUT_DWORD(req, GET_ASSERTS_OUT_THREAD_OFFS));

170         /* Print out the registers */
171         ofst = MC_CMD_GET_ASSERTS_OUT_GP_REGS_OFFS_OFST;
172         for (index = 1; index < 32; index++) {
173                 EFSYS_PROBE2(mcpu_register, unsigned int, index, unsigned int,
174                              EFX_DWORD_FIELD(*MCDI_OUT(req, efx_dword_t, ofst),
175                                              EFX_DWORD_0));
176                 ofst += sizeof (efx_dword_t);
177         }
178         EFSYS_ASSERT(ofst <= MC_CMD_GET_ASSERTS_OUT_LEN);

180         return (0);

182 fail2:
183         EFSYS_PROBE(fail2);
184 fail1:
185         EFSYS_PROBE1(fail1, int, rc);

187         return (rc);
188 }

190 static  __checkReturn   int
191 siena_nic_attach(
192         __in            efx_nic_t *enp,
193         __in            boolean_t attach)
```



```
194 {
195         efx_mcdi_req_t req;
196         uint8_t payload[MC_CMD_DRV_ATTACH_IN_LEN];
197         int rc;

199         req.emr_cmd = MC_CMD_DRV_ATTACH;
200         req.emr_in_buf = payload;
201         req.emr_in_length = MC_CMD_DRV_ATTACH_IN_LEN;
202         req.emr_out_buf = NULL;
203         req.emr_out_length = 0;

205         MCDI_IN_SET_DWORD(req, DRV_ATTACH_IN_NEW_STATE, attach ? 1 : 0);
206         MCDI_IN_SET_DWORD(req, DRV_ATTACH_IN_UPDATE, 1);

208         efx_mcdi_execute(enp, &req);

210         if (req.emr_rc != 0) {
211                 rc = req.emr_rc;
212                 goto fail1;
213         }

215         if (req.emr_out_length_used < MC_CMD_DRV_ATTACH_OUT_LEN) {
216                 rc = EMSGSIZE;
217                 goto fail2;
218         }

220         return (0);

222 fail2:
223         EFSYS_PROBE(fail2);
224 fail1:
225         EFSYS_PROBE1(fail1, int, rc);

227         return (rc);
228 }

230 #if EFSYS_OPT_PCIE_TUNE

232         __checkReturn   int
233 siena_nic_pcie_extended_sync(
234         __in            efx_nic_t *enp)
235 {
236         uint8_t inbuf[MC_CMD_WORKAROUND_IN_LEN];
237         efx_mcdi_req_t req;
238         int rc;

240         EFSYS_ASSERT3U(enp->en_family, ==, EFX_FAMILY_SIENA);

242         req.emr_cmd = MC_CMD_WORKAROUND;
243         req.emr_in_buf = inbuf;
244         req.emr_in_length = sizeof (inbuf);
245         EFX_STATIC_ASSERT(MC_CMD_WORKAROUND_OUT_LEN == 0);
246         req.emr_out_buf = NULL;
247         req.emr_out_length = 0;

249         MCDI_IN_SET_DWORD(req, WORKAROUND_IN_TYPE, MC_CMD_WORKAROUND_BUG17230);
250         MCDI_IN_SET_DWORD(req, WORKAROUND_IN_ENABLED, 1);

252         efx_mcdi_execute(enp, &req);

254         if (req.emr_rc != 0) {
255                 rc = req.emr_rc;
256                 goto fail1;
257         }

259         return (0);
```

```
fail1:
        EFSYS_PROBE1(fail1, int, rc);

        return (rc);
}

#endif  /* EFSYS_OPT_PCIE_TUNE */

static  __checkReturn   int
siena_board_cfg(
        __in            efx_nic_t *enp)
{
        efx_nic_cfg_t *encp = &(enp->en_nic_cfg);
        efx_mcdi_iface_t *emip = &(enp->en_u.siena.enu_mip);
        uint8_t outbuf[MAX(MC_CMD_GET_BOARD_CFG_OUT_LENMIN,
                    MC_CMD_GET_RESOURCE_LIMITS_OUT_LEN)];
        efx_mcdi_req_t req;
        uint8_t *mac_addr;
        efx_dword_t *capabilities;
        int rc;

        /* Board configuration */
        req.emr_cmd = MC_CMD_GET_BOARD_CFG;
        EFX_STATIC_ASSERT(MC_CMD_GET_BOARD_CFG_IN_LEN == 0);
        req.emr_in_buf = NULL;
        req.emr_in_length = 0;
        req.emr_out_buf = outbuf;
        req.emr_out_length = MC_CMD_GET_BOARD_CFG_OUT_LENMIN;

        efx_mcdi_execute(enp, &req);

        if (req.emr_rc != 0) {
                rc = req.emr_rc;
                goto fail1;
        }

        if (req.emr_out_length_used < MC_CMD_GET_BOARD_CFG_OUT_LENMIN) {
                rc = EMSGSIZE;
                goto fail2;
        }

        if (emip->emi_port == 1) {
                mac_addr = MCDI_OUT2(req, uint8_t,
                            GET_BOARD_CFG_OUT_MAC_ADDR_BASE_PORT0);
                capabilities = MCDI_OUT2(req, efx_dword_t,
                            GET_BOARD_CFG_OUT_CAPABILITIES_PORT0);
        } else {
                mac_addr = MCDI_OUT2(req, uint8_t,
                            GET_BOARD_CFG_OUT_MAC_ADDR_BASE_PORT1);
                capabilities = MCDI_OUT2(req, efx_dword_t,
                            GET_BOARD_CFG_OUT_CAPABILITIES_PORT1);
        }
        EFX_MAC_ADDR_COPY(encp->enc_mac_addr, mac_addr);

        encp->enc_board_type = MCDI_OUT_DWORD(req,
                                    GET_BOARD_CFG_OUT_BOARD_TYPE);

        /* Additional capabilities */
        encp->enc_clk_mult = 1;
        if (MCDI_CMD_DWORD_FIELD(capabilities, CAPABILITIES_TURBO)) {
                enp->en_features |= EFX_FEATURE_TURBO;

                if (MCDI_CMD_DWORD_FIELD(capabilities,
                    CAPABILITIES_TURBO_ACTIVE))
                        encp->enc_clk_mult = 2;
```



```
        }

        encp->enc_evq_timer_quantum_ns =
                EFX_EVQ_SIENA_TIMER_QUANTUM_NS / encp->enc_clk_mult;
        encp->enc_evq_timer_max_us = (encp->enc_evq_timer_quantum_ns <<
                FRF_CZ_TC_TIMER_VAL_WIDTH) / 1000;

        /* Resource limits */
        req.emr_cmd = MC_CMD_GET_RESOURCE_LIMITS;
        EFX_STATIC_ASSERT(MC_CMD_GET_RESOURCE_LIMITS_IN_LEN == 0);
        req.emr_in_buf = NULL;
        req.emr_in_length = 0;
        req.emr_out_buf = outbuf;
        req.emr_out_length = MC_CMD_GET_RESOURCE_LIMITS_OUT_LEN;

        efx_mcdi_execute(enp, &req);

        if (req.emr_rc == 0) {
                if (req.emr_out_length_used <
                    MC_CMD_GET_RESOURCE_LIMITS_OUT_LEN) {
                        rc = EMSGSIZE;
                        goto fail3;
                }

                encp->enc_evq_limit = MCDI_OUT_DWORD(req,
                    GET_RESOURCE_LIMITS_OUT_EVQ);
                encp->enc_txq_limit = MIN(EFX_TXQ_LIMIT_TARGET,
                    MCDI_OUT_DWORD(req, GET_RESOURCE_LIMITS_OUT_TXQ));
                encp->enc_rxq_limit = MIN(EFX_RXQ_LIMIT_TARGET,
                    MCDI_OUT_DWORD(req, GET_RESOURCE_LIMITS_OUT_RXQ));
        } else if (req.emr_rc == ENOTSUP) {
                encp->enc_evq_limit = 1024;
                encp->enc_txq_limit = EFX_TXQ_LIMIT_TARGET;
                encp->enc_rxq_limit = EFX_RXQ_LIMIT_TARGET;
        } else {
                rc = req.emr_rc;
                goto fail4;
        }

        encp->enc_buftbl_limit = SIENA_SRAM_ROWS -
            (encp->enc_txq_limit * EFX_TXQ_DC_NDESCS(EFX_TXQ_DC_SIZE)) -
            (encp->enc_rxq_limit * EFX_RXQ_DC_NDESCS(EFX_RXQ_DC_SIZE));

        return (0);

fail4:
        EFSYS_PROBE(fail4);
fail3:
        EFSYS_PROBE(fail3);
fail2:
        EFSYS_PROBE(fail2);
fail1:
        EFSYS_PROBE1(fail1, int, rc);

        return (rc);
}

static  __checkReturn   int
siena_phy_cfg(
        __in            efx_nic_t *enp)
{
        efx_port_t *epp = &(enp->en_port);
        efx_nic_cfg_t *encp = &(enp->en_nic_cfg);
        efx_mcdi_req_t req;
        uint8_t outbuf[MC_CMD_GET_PHY_CFG_OUT_LEN];
        int rc;
```

```
393         req.emr_cmd = MC_CMD_GET_PHY_CFG;
394         EFX_STATIC_ASSERT(MC_CMD_GET_PHY_CFG_IN_LEN == 0);
395         req.emr_in_buf = NULL;
396         req.emr_in_length = 0;
397         req.emr_out_buf = outbuf;
398         req.emr_out_length = sizeof (outbuf);

400         efx_mcdi_execute(enp, &req);

402         if (req.emr_rc != 0) {
403                 rc = req.emr_rc;
404                 goto fail1;
405         }

407         if (req.emr_out_length_used < MC_CMD_GET_PHY_CFG_OUT_LEN) {
408                 rc = EMSGSIZE;
409                 goto fail2;
410         }

412         encp->enc_phy_type = MCDI_OUT_DWORD(req, GET_PHY_CFG_OUT_TYPE);
413 #if EFSYS_OPT_NAMES
414         (void) strncpy(encp->enc_phy_name,
415                 MCDI_OUT2(req, char, GET_PHY_CFG_OUT_NAME),
416                 MIN(sizeof (encp->enc_phy_name) - 1,
417                     MC_CMD_GET_PHY_CFG_OUT_NAME_LEN));
418 #endif  /* EFSYS_OPT_NAMES */
419         (void) memset(encp->enc_phy_revision, 0,
420             sizeof (encp->enc_phy_revision));
421         memcpy(encp->enc_phy_revision,
422                 MCDI_OUT2(req, char, GET_PHY_CFG_OUT_REVISION),
423                 MIN(sizeof (encp->enc_phy_revision) - 1,
424                     MC_CMD_GET_PHY_CFG_OUT_REVISION_LEN));
425 #if EFSYS_OPT_PHY_LED_CONTROL
426         encp->enc_led_mask = ((1 << EFX_PHY_LED_DEFAULT) |
427                               (1 << EFX_PHY_LED_OFF) |
428                               (1 << EFX_PHY_LED_ON));
429 #endif  /* EFSYS_OPT_PHY_LED_CONTROL */

431 #if EFSYS_OPT_PHY_PROPS
432         encp->enc_phy_nprops  = 0;
433 #endif  /* EFSYS_OPT_PHY_PROPS */

435         /* Get the media type of the fixed port, if recognised. */
436         EFX_STATIC_ASSERT(MC_CMD_MEDIA_XAUI == EFX_PHY_MEDIA_XAUI);
437         EFX_STATIC_ASSERT(MC_CMD_MEDIA_CX4 == EFX_PHY_MEDIA_CX4);
438         EFX_STATIC_ASSERT(MC_CMD_MEDIA_KX4 == EFX_PHY_MEDIA_KX4);
439         EFX_STATIC_ASSERT(MC_CMD_MEDIA_XFP == EFX_PHY_MEDIA_XFP);
440         EFX_STATIC_ASSERT(MC_CMD_MEDIA_SFP_PLUS == EFX_PHY_MEDIA_SFP_PLUS);
441         EFX_STATIC_ASSERT(MC_CMD_MEDIA_BASE_T == EFX_PHY_MEDIA_BASE_T);
442         epp->ep_fixed_port_type =
443                 MCDI_OUT_DWORD(req, GET_PHY_CFG_OUT_MEDIA_TYPE);
444         if (epp->ep_fixed_port_type >= EFX_PHY_MEDIA_NTYPES)
445                 epp->ep_fixed_port_type = EFX_PHY_MEDIA_INVALID;

447         epp->ep_phy_cap_mask =
448                 MCDI_OUT_DWORD(req, GET_PHY_CFG_OUT_SUPPORTED_CAP);
449 #if EFSYS_OPT_PHY_FLAGS
450         encp->enc_phy_flags_mask = MCDI_OUT_DWORD(req, GET_PHY_CFG_OUT_FLAGS);
451 #endif  /* EFSYS_OPT_PHY_FLAGS */

453         encp->enc_port = (uint8_t)MCDI_OUT_DWORD(req, GET_PHY_CFG_OUT_PRT);

455         /* Populate internal state */
456         encp->enc_siena_channel =
457                 (uint8_t)MCDI_OUT_DWORD(req, GET_PHY_CFG_OUT_CHANNEL);
```



```
459 #if EFSYS_OPT_PHY_STATS
460         encp->enc_siena_phy_stat_mask =
461                 MCDI_OUT_DWORD(req, GET_PHY_CFG_OUT_STATS_MASK);

463         /* Convert the MCDI statistic mask into the EFX_PHY_STAT mask */
464         siena_phy_decode_stats(enp, encp->enc_siena_phy_stat_mask,
465                                NULL, &encp->enc_phy_stat_mask, NULL);
466 #endif  /* EFSYS_OPT_PHY_STATS */

468 #if EFSYS_OPT_PHY_BIST
469         encp->enc_bist_mask = 0;
470         if (MCDI_OUT_DWORD_FIELD(req, GET_PHY_CFG_OUT_FLAGS,
471             GET_PHY_CFG_OUT_BIST_CABLE_SHORT))
472                 encp->enc_bist_mask |= (1 << EFX_PHY_BIST_TYPE_CABLE_SHORT);
473         if (MCDI_OUT_DWORD_FIELD(req, GET_PHY_CFG_OUT_FLAGS,
474             GET_PHY_CFG_OUT_BIST_CABLE_LONG))
475                 encp->enc_bist_mask |= (1 << EFX_PHY_BIST_TYPE_CABLE_LONG);
476         if (MCDI_OUT_DWORD_FIELD(req, GET_PHY_CFG_OUT_FLAGS,
477             GET_PHY_CFG_OUT_BIST))
478                 encp->enc_bist_mask |= (1 << EFX_PHY_BIST_TYPE_NORMAL);
479 #endif  /* EFSYS_OPT_PHY_BIST */

481         return (0);

483 fail2:
484         EFSYS_PROBE(fail2);
485 fail1:
486         EFSYS_PROBE1(fail1, int, rc);

488         return (rc);
489 }

491 #if EFSYS_OPT_LOOPBACK

493 static  __checkReturn   int
494 siena_loopback_cfg(
495         __in            efx_nic_t *enp)
496 {
497         efx_nic_cfg_t *encp = &(enp->en_nic_cfg);
498         efx_mcdi_req_t req;
499         uint8_t outbuf[MC_CMD_GET_LOOPBACK_MODES_OUT_LEN];
500         int rc;

502         req.emr_cmd = MC_CMD_GET_LOOPBACK_MODES;
503         EFX_STATIC_ASSERT(MC_CMD_GET_LOOPBACK_MODES_IN_LEN == 0);
504         req.emr_in_buf = NULL;
505         req.emr_in_length = 0;
506         req.emr_out_buf = outbuf;
507         req.emr_out_length = sizeof (outbuf);

509         efx_mcdi_execute(enp, &req);

511         if (req.emr_rc != 0) {
512                 rc = req.emr_rc;
513                 goto fail1;
514         }

516         if (req.emr_out_length_used < MC_CMD_GET_LOOPBACK_MODES_OUT_LEN) {
517                 rc = EMSGSIZE;
518                 goto fail2;
519         }

521         /*
522          * We assert the MC_CMD_LOOPBACK and EFX_LOOPBACK namespaces agree
523          * in siena_phy.c:siena_phy_get_link()
```

```
524          */
525         encp->enc_loopback_types[EFX_LINK_100FDX] = EFX_LOOPBACK_MASK &
526             MCDI_OUT_DWORD(req, GET_LOOPBACK_MODES_OUT_100M) &
527             MCDI_OUT_DWORD(req, GET_LOOPBACK_MODES_OUT_SUGGESTED);
528         encp->enc_loopback_types[EFX_LINK_1000FDX] = EFX_LOOPBACK_MASK &
529             MCDI_OUT_DWORD(req, GET_LOOPBACK_MODES_OUT_1G) &
530             MCDI_OUT_DWORD(req, GET_LOOPBACK_MODES_OUT_SUGGESTED);
531         encp->enc_loopback_types[EFX_LINK_10000FDX] = EFX_LOOPBACK_MASK &
532             MCDI_OUT_DWORD(req, GET_LOOPBACK_MODES_OUT_10G) &
533             MCDI_OUT_DWORD(req, GET_LOOPBACK_MODES_OUT_SUGGESTED);
534         encp->enc_loopback_types[EFX_LINK_UNKNOWN] =
535             (1 << EFX_LOOPBACK_OFF) |
536             encp->enc_loopback_types[EFX_LINK_100FDX] |
537             encp->enc_loopback_types[EFX_LINK_1000FDX] |
538             encp->enc_loopback_types[EFX_LINK_10000FDX];

540         return (0);

542 fail2:
543         EFSYS_PROBE(fail2);
544 fail1:
545         EFSYS_PROBE1(fail1, int, rc);

547         return (rc);
548 }

550 #endif  /* EFSYS_OPT_LOOPBACK */

552 #if EFSYS_OPT_MON_STATS

554 static  __checkReturn   int
555 siena_monitor_cfg(
556         __in            efx_nic_t *enp)
557 {
558         efx_nic_cfg_t *encp = &(enp->en_nic_cfg);
559         efx_mcdi_req_t req;
560         uint8_t outbuf[MCDI_CTL_SDU_LEN_MAX];
561         int rc;

563         req.emr_cmd = MC_CMD_SENSOR_INFO;
564         EFX_STATIC_ASSERT(MC_CMD_SENSOR_INFO_IN_LEN == 0);
565         req.emr_in_buf = NULL;
566         req.emr_in_length = 0;
567         req.emr_out_buf = outbuf;
568         req.emr_out_length = sizeof (outbuf);

570         efx_mcdi_execute(enp, &req);

572         if (req.emr_rc != 0) {
573                 rc = req.emr_rc;
574                 goto fail1;
575         }

577         if (req.emr_out_length_used < MC_CMD_SENSOR_INFO_OUT_MASK_OFST + 4) {
578                 rc = EMSGSIZE;
579                 goto fail2;
580         }

582         encp->enc_siena_mon_stat_mask =
583                 MCDI_OUT_DWORD(req, SENSOR_INFO_OUT_MASK);
584         encp->enc_mon_type = EFX_MON_SFC90X0;

586         siena_mon_decode_stats(enp, encp->enc_siena_mon_stat_mask,
587                                NULL, &(encp->enc_mon_stat_mask), NULL);

589         return (0);
```



```
591 fail2:
592         EFSYS_PROBE(fail2);
593 fail1:
594         EFSYS_PROBE1(fail1, int, rc);

596         return (rc);
597 }

599 #endif  /* EFSYS_OPT_MON_STATS */

601         __checkReturn   int
602 siena_nic_probe(
603         __in            efx_nic_t *enp)
604 {
605         efx_port_t *epp = &(enp->en_port);
606         efx_nic_cfg_t *encp = &(enp->en_nic_cfg);
607         siena_link_state_t sls;
608         unsigned int mask;
609         int rc;

611         EFSYS_ASSERT3U(enp->en_family, ==, EFX_FAMILY_SIENA);

613         /* Read clear any assertion state */
614         if ((rc = siena_nic_read_assertion(enp)) != 0)
615                 goto fail1;

617         /* Exit the assertion handler */
618         if ((rc = siena_nic_exit_assertion_handler(enp)) != 0)
619                 goto fail2;

621         /* Wrestle control from the BMC */
622         if ((rc = siena_nic_attach(enp, B_TRUE)) != 0)
623                 goto fail3;

625         if ((rc = siena_board_cfg(enp)) != 0)
626                 goto fail4;

628         if ((rc = siena_phy_cfg(enp)) != 0)
629                 goto fail5;

631         /* Obtain the default PHY advertised capabilities */
632         if ((rc = siena_nic_reset(enp)) != 0)
633                 goto fail6;
634         if ((rc = siena_phy_get_link(enp, &sls)) != 0)
635                 goto fail7;
636         epp->ep_default_adv_cap_mask = sls.sls_adv_cap_mask;
637         epp->ep_adv_cap_mask = sls.sls_adv_cap_mask;

639 #if EFSYS_OPT_VPD || EFSYS_OPT_NVRAM
640         if ((rc = siena_nic_get_partn_mask(enp, &mask)) != 0)
641                 goto fail8;
642         enp->en_u.siena.enu_partn_mask = mask;
643 #endif

645 #if EFSYS_OPT_MAC_STATS
646         /* Wipe the MAC statistics */
647         if ((rc = siena_mac_stats_clear(enp)) != 0)
648                 goto fail9;
649 #endif

651 #if EFSYS_OPT_LOOPBACK
652         if ((rc = siena_loopback_cfg(enp)) != 0)
653                 goto fail10;
654 #endif
```

```
#if EFSYS_OPT_MON_STATS
        if ((rc = siena_monitor_cfg(enp)) != 0)
                goto fail11;
#endif

        encp->enc_features = enp->en_features;

        return (0);

#if EFSYS_OPT_MON_STATS
fail11:
        EFSYS_PROBE(fail11);
#endif
#if EFSYS_OPT_LOOPBACK
fail10:
        EFSYS_PROBE(fail10);
#endif
#if EFSYS_OPT_MAC_STATS
fail9:
        EFSYS_PROBE(fail9);
#endif
#if EFSYS_OPT_VPD || EFSYS_OPT_NVRAM
fail8:
        EFSYS_PROBE(fail8);
#endif
fail7:
        EFSYS_PROBE(fail7);
fail6:
        EFSYS_PROBE(fail6);
fail5:
        EFSYS_PROBE(fail5);
fail4:
        EFSYS_PROBE(fail4);
fail3:
        EFSYS_PROBE(fail3);
fail2:
        EFSYS_PROBE(fail2);
fail1:
        EFSYS_PROBE1(fail1, int, rc);

        return (rc);
}

        __checkReturn   int
siena_nic_reset(
        __in            efx_nic_t *enp)
{
        efx_mcdi_req_t req;
        int rc;

        EFSYS_ASSERT3U(enp->en_family, ==, EFX_FAMILY_SIENA);

        /* siena_nic_reset() is called to recover from BADASSERT failures. */
        if ((rc = siena_nic_read_assertion(enp)) != 0)
                goto fail1;
        if ((rc = siena_nic_exit_assertion_handler(enp)) != 0)
                goto fail2;

        req.emr_cmd = MC_CMD_PORT_RESET;
        EFX_STATIC_ASSERT(MC_CMD_PORT_RESET_IN_LEN == 0);
        req.emr_in_buf = NULL;
        req.emr_in_length = 0;
        EFX_STATIC_ASSERT(MC_CMD_PORT_RESET_OUT_LEN == 0);
        req.emr_out_buf = NULL;
        req.emr_out_length = 0;
```



```
        efx_mcdi_execute(enp, &req);

        if (req.emr_rc != 0) {
                rc = req.emr_rc;
                goto fail3;
        }

        return (0);

fail3:
        EFSYS_PROBE(fail3);
fail2:
        EFSYS_PROBE(fail2);
fail1:
        EFSYS_PROBE1(fail1, int, rc);

        return (0);
}

static  __checkReturn   int
siena_nic_logging(
        __in            efx_nic_t *enp)
{
        efx_mcdi_req_t req;
        uint8_t payload[MC_CMD_LOG_CTRL_IN_LEN];
        int rc;

        req.emr_cmd = MC_CMD_LOG_CTRL;
        req.emr_in_buf = payload;
        req.emr_in_length = MC_CMD_LOG_CTRL_IN_LEN;
        EFX_STATIC_ASSERT(MC_CMD_LOG_CTRL_OUT_LEN == 0);
        req.emr_out_buf = NULL;
        req.emr_out_length = 0;

        MCDI_IN_SET_DWORD(req, LOG_CTRL_IN_LOG_DEST,
                          MC_CMD_LOG_CTRL_IN_LOG_DEST_EVQ);
        MCDI_IN_SET_DWORD(req, LOG_CTRL_IN_LOG_DEST_EVQ, 0);

        efx_mcdi_execute(enp, &req);

        if (req.emr_rc != 0) {
                rc = req.emr_rc;
                goto fail1;
        }

        return (0);

fail1:
        EFSYS_PROBE1(fail1, int, rc);

        return (rc);
}

static                  void
siena_nic_rx_cfg(
        __in            efx_nic_t *enp)
{
        efx_oword_t oword;

        /*
         * RX_INGR_EN is always enabled on Siena, because we rely on
         * the RX parser to be resiliant to missing SOP/EOP.
         */
        EFX_BAR_READO(enp, FR_AZ_RX_CFG_REG, &oword);
        EFX_SET_OWORD_FIELD(oword, FRF_BZ_RX_INGR_EN, 1);
        EFX_BAR_WRITEO(enp, FR_AZ_RX_CFG_REG, &oword);
```

```
789         /* Disable parsing of additional 802.1Q in Q packets */
790         EFX_BAR_READO(enp, FR_AZ_RX_FILTER_CTL_REG, &oword);
791         EFX_SET_OWORD_FIELD(oword, FRF_CZ_RX_FILTER_ALL_VLAN_ETHERTYPES, 0);
792         EFX_BAR_WRITEO(enp, FR_AZ_RX_FILTER_CTL_REG, &oword);
793 }

795 static                  void
796 siena_nic_usrev_dis(
797         __in            efx_nic_t *enp)
798 {
799         efx_oword_t     oword;

801         EFX_POPULATE_OWORD_1(oword, FRF_CZ_USREV_DIS, 1);
802         EFX_BAR_WRITEO(enp, FR_CZ_USR_EV_CFG, &oword);
803 }

805         __checkReturn   int
806 siena_nic_init(
807         __in            efx_nic_t *enp)
808 {
809         int rc;

811         EFSYS_ASSERT3U(enp->en_family, ==, EFX_FAMILY_SIENA);

813         if ((rc = siena_nic_logging(enp)) != 0)
814                 goto fail1;

816         siena_sram_init(enp);

818         /* Configure Siena's RX block */
819         siena_nic_rx_cfg(enp);

821         /* Disable USR_EVents for now */
822         siena_nic_usrev_dis(enp);

824         /* bug17057: Ensure set_link is called */
825         if ((rc = siena_phy_reconfigure(enp)) != 0)
826                 goto fail2;

828         return (0);

830 fail2:
831         EFSYS_PROBE(fail2);
832 fail1:
833         EFSYS_PROBE1(fail1, int, rc);

835         return (rc);
836 }

838                         void
839 siena_nic_fini(
840         __in            efx_nic_t *enp)
841 {
842         _NOTE(ARGUNUSED(enp))
843 }

845                         void
846 siena_nic_unprobe(
847         __in            efx_nic_t *enp)
848 {
849         (void) siena_nic_attach(enp, B_FALSE);
850 }

852 #if EFSYS_OPT_DIAG
```



```
854 static efx_register_set_t __cs  __siena_registers[] = {
855         { FR_AZ_ADR_REGION_REG_OFST, 0, 1 },
856         { FR_CZ_USR_EV_CFG_OFST, 0, 1 },
857         { FR_AZ_RX_CFG_REG_OFST, 0, 1 },
858         { FR_AZ_TX_CFG_REG_OFST, 0, 1 },
859         { FR_AZ_TX_RESERVED_REG_OFST, 0, 1 },
860         { FR_AZ_SRM_TX_DC_CFG_REG_OFST, 0, 1 },
861         { FR_AZ_RX_DC_CFG_REG_OFST, 0, 1 },
862         { FR_AZ_RX_DC_PF_WM_REG_OFST, 0, 1 },
863         { FR_AZ_DP_CTRL_REG_OFST, 0, 1 },
864         { FR_BZ_RX_RSS_TKEY_REG_OFST, 0, 1},
865         { FR_CZ_RX_RSS_IPV6_REG1_OFST, 0, 1},
866         { FR_CZ_RX_RSS_IPV6_REG2_OFST, 0, 1},
867         { FR_CZ_RX_RSS_IPV6_REG3_OFST, 0, 1}
868 };

870 static const uint32_t __cs      __siena_register_masks[] = {
871         0x0003FFFF, 0x0003FFFF, 0x0003FFFF, 0x0003FFFF,
872         0x000103FF, 0x00000000, 0x00000000, 0x00000000,
873         0xFFFFFFFE, 0xFFFFFFFF, 0x0003FFFF, 0x00000000,
874         0x7FFF0037, 0xFFFF8000, 0xFFFFFFFF, 0x03FFFFFF,
875         0xFFFEFE80, 0x1FFFFFFF, 0x020000FE, 0x007FFFFF,
876         0x001FFFFF, 0x00000000, 0x00000000, 0x00000000,
877         0x00000003, 0x00000000, 0x00000000, 0x00000000,
878         0x000003FF, 0x00000000, 0x00000000, 0x00000000,
879         0x00000FFF, 0x00000000, 0x00000000, 0x00000000,
880         0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF,
881         0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF,
882         0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF,
883         0xFFFFFFFF, 0xFFFFFFFF, 0x00000007, 0x00000000
884 };

886 static efx_register_set_t __cs  __siena_tables[] = {
887         { FR_AZ_RX_FILTER_TBL0_OFST, FR_AZ_RX_FILTER_TBL0_STEP,
888             FR_AZ_RX_FILTER_TBL0_ROWS },
889         { FR_CZ_RX_MAC_FILTER_TBL0_OFST, FR_CZ_RX_MAC_FILTER_TBL0_STEP,
890             FR_CZ_RX_MAC_FILTER_TBL0_ROWS },
891         { FR_AZ_RX_DESC_PTR_TBL_OFST,
892             FR_AZ_RX_DESC_PTR_TBL_STEP, FR_CZ_RX_DESC_PTR_TBL_ROWS },
893         { FR_AZ_TX_DESC_PTR_TBL_OFST,
894             FR_AZ_TX_DESC_PTR_TBL_STEP, FR_CZ_TX_DESC_PTR_TBL_ROWS },
895         { FR_AZ_TIMER_TBL_OFST, FR_AZ_TIMER_TBL_STEP, FR_CZ_TIMER_TBL_ROWS },
896         { FR_CZ_TX_FILTER_TBL0_OFST,
897             FR_CZ_TX_FILTER_TBL0_STEP, FR_CZ_TX_FILTER_TBL0_ROWS },
898         { FR_CZ_TX_MAC_FILTER_TBL0_OFST,
899             FR_CZ_TX_MAC_FILTER_TBL0_STEP, FR_CZ_TX_MAC_FILTER_TBL0_ROWS }
900 };

902 static const uint32_t __cs      __siena_table_masks[] = {
903         0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0x000003FF,
904         0xFFFF0FFF, 0xFFFFFFFF, 0x00000E7F, 0x00000000,
905         0xFFFFFFFE, 0x0FFFFFFF, 0x01800000, 0x00000000,
906         0xFFFFFFFE, 0x0FFFFFFF, 0x0C000000, 0x00000000,
907         0x3FFFFFFF, 0x00000000, 0x00000000, 0x00000000,
908         0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0x000013FF,
909         0xFFFF07FF, 0xFFFFFFFF, 0x0000007F, 0x00000000,
910 };

912         __checkReturn   int
913 siena_nic_register_test(
914         __in            efx_nic_t *enp)
915 {
916         efx_register_set_t *rsp;
917         const uint32_t *dwordp;
918         unsigned int nitems;
919         unsigned int count;
```

```
920         int rc;
921
922         /* Fill out the register mask entries */
923         EFX_STATIC_ASSERT(EFX_ARRAY_SIZE(__siena_register_masks)
924                           == EFX_ARRAY_SIZE(__siena_registers) * 4);
925
926         nitems = EFX_ARRAY_SIZE(__siena_registers);
927         dwordp = __siena_register_masks;
928         for (count = 0; count < nitems; ++count) {
929                 rsp = __siena_registers + count;
930                 rsp->mask.eo_u32[0] = *dwordp++;
931                 rsp->mask.eo_u32[1] = *dwordp++;
932                 rsp->mask.eo_u32[2] = *dwordp++;
933                 rsp->mask.eo_u32[3] = *dwordp++;
934         }
935
936         /* Fill out the register table entries */
937         EFX_STATIC_ASSERT(EFX_ARRAY_SIZE(__siena_table_masks)
938                           == EFX_ARRAY_SIZE(__siena_tables) * 4);
939
940         nitems = EFX_ARRAY_SIZE(__siena_tables);
941         dwordp = __siena_table_masks;
942         for (count = 0; count < nitems; ++count) {
943                 rsp = __siena_tables + count;
944                 rsp->mask.eo_u32[0] = *dwordp++;
945                 rsp->mask.eo_u32[1] = *dwordp++;
946                 rsp->mask.eo_u32[2] = *dwordp++;
947                 rsp->mask.eo_u32[3] = *dwordp++;
948         }
949
950         if ((rc = efx_nic_test_registers(enp, __siena_registers,
951             EFX_ARRAY_SIZE(__siena_registers))) != 0)
952                 goto fail1;
953
954         if ((rc = efx_nic_test_tables(enp, __siena_tables,
955             EFX_PATTERN_BYTE_ALTERNATE,
956             EFX_ARRAY_SIZE(__siena_tables))) != 0)
957                 goto fail2;
958
959         if ((rc = efx_nic_test_tables(enp, __siena_tables,
960             EFX_PATTERN_BYTE_CHANGING,
961             EFX_ARRAY_SIZE(__siena_tables))) != 0)
962                 goto fail3;
963
964         if ((rc = efx_nic_test_tables(enp, __siena_tables,
965             EFX_PATTERN_BIT_SWEEP, EFX_ARRAY_SIZE(__siena_tables))) != 0)
966                 goto fail4;
967
968         return (0);
969
970 fail4:
971         EFSYS_PROBE(fail4);
972 fail3:
973         EFSYS_PROBE(fail3);
974 fail2:
975         EFSYS_PROBE(fail2);
976 fail1:
977         EFSYS_PROBE1(fail1, int, rc);
978
979         return (rc);
980 }
981
982 #endif  /* EFSYS_OPT_DIAG */
983
984 #endif  /* EFSYS_OPT_SIENA */
985 #endif /* ! codereview */
```

```
**********************************************************
   23241 Thu Aug 22 18:59:29 2013
new/usr/src/uts/common/io/sfxge/siena_nvram.c
Merged sfxge driver
**********************************************************
```

```
/*
 * Copyright 2009 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#include "efsys.h"
#include "efx.h"
#include "efx_types.h"
#include "efx_regs.h"
#include "efx_impl.h"

#if EFSYS_OPT_SIENA

#if EFSYS_OPT_VPD || EFSYS_OPT_NVRAM

        __checkReturn           int
siena_nvram_partn_size(
        __in                    efx_nic_t *enp,
        __in                    unsigned int partn,
        __out                   size_t *sizep)
{
        efx_mcdi_req_t req;
        uint8_t payload[MAX(MC_CMD_NVRAM_INFO_IN_LEN,
                            MC_CMD_NVRAM_INFO_OUT_LEN)];
        int rc;

        if ((1 << partn) & ~enp->en_u.siena.enu_partn_mask) {
                rc = ENOTSUP;
                goto fail1;
        }

        req.emr_cmd = MC_CMD_NVRAM_INFO;
        req.emr_in_buf = payload;
        req.emr_in_length = MC_CMD_NVRAM_INFO_IN_LEN;
        req.emr_out_buf = payload;
        req.emr_out_length = MC_CMD_NVRAM_INFO_OUT_LEN;

        MCDI_IN_SET_DWORD(req, NVRAM_INFO_IN_TYPE, partn);

        efx_mcdi_execute(enp, &req);
```



```
        if (req.emr_rc != 0) {
                rc = req.emr_rc;
                goto fail2;
        }

        if (req.emr_out_length_used < MC_CMD_NVRAM_INFO_OUT_LEN) {
                rc = EMSGSIZE;
                goto fail3;
        }

        *sizep = MCDI_OUT_DWORD(req, NVRAM_INFO_OUT_SIZE);

        return (0);

fail3:
        EFSYS_PROBE(fail3);
fail2:
        EFSYS_PROBE(fail2);
fail1:
        EFSYS_PROBE1(fail1, int, rc);

        return (rc);
}

        __checkReturn           int
siena_nvram_partn_lock(
        __in                    efx_nic_t *enp,
        __in                    unsigned int partn)
{
        efx_mcdi_req_t req;
        uint8_t payload[MC_CMD_NVRAM_UPDATE_START_IN_LEN];
        int rc;

        req.emr_cmd = MC_CMD_NVRAM_UPDATE_START;
        req.emr_in_buf = payload;
        req.emr_in_length = MC_CMD_NVRAM_UPDATE_START_IN_LEN;
        EFX_STATIC_ASSERT(MC_CMD_NVRAM_UPDATE_START_OUT_LEN == 0);
        req.emr_out_buf = NULL;
        req.emr_out_length = 0;

        MCDI_IN_SET_DWORD(req, NVRAM_UPDATE_START_IN_TYPE, partn);

        efx_mcdi_execute(enp, &req);

        if (req.emr_rc != 0) {
                rc = req.emr_rc;
                goto fail1;
        }

        return (0);

fail1:
        EFSYS_PROBE1(fail1, int, rc);

        return (rc);
}

        __checkReturn           int
siena_nvram_partn_read(
        __in                    efx_nic_t *enp,
        __in                    unsigned int partn,
        __in                    unsigned int offset,
        __out_bcount(size)      caddr_t data,
        __in                    size_t size)
{
        efx_mcdi_req_t req;
```

```
128         uint8_t payload[MAX(MC_CMD_NVRAM_READ_IN_LEN,
129                             MC_CMD_NVRAM_READ_OUT_LEN(SIENA_NVRAM_CHUNK))];
130         size_t chunk;
131         int rc;
132
133         while (size > 0) {
134                 chunk = MIN(size, SIENA_NVRAM_CHUNK);
135
136                 req.emr_cmd = MC_CMD_NVRAM_READ;
137                 req.emr_in_buf = payload;
138                 req.emr_in_length = MC_CMD_NVRAM_READ_IN_LEN;
139                 req.emr_out_buf = payload;
140                 req.emr_out_length =
141                         MC_CMD_NVRAM_READ_OUT_LEN(SIENA_NVRAM_CHUNK);
142
143                 MCDI_IN_SET_DWORD(req, NVRAM_READ_IN_TYPE, partn);
144                 MCDI_IN_SET_DWORD(req, NVRAM_READ_IN_OFFSET, offset);
145                 MCDI_IN_SET_DWORD(req, NVRAM_READ_IN_LENGTH, chunk);
146
147                 efx_mcdi_execute(enp, &req);
148
149                 if (req.emr_rc != 0) {
150                         rc = req.emr_rc;
151                         goto fail1;
152                 }
153
154                 if (req.emr_out_length_used <
155                     MC_CMD_NVRAM_READ_OUT_LEN(chunk)) {
156                         rc = EMSGSIZE;
157                         goto fail2;
158                 }
159
160                 memcpy(data,
161                     MCDI_OUT2(req, uint8_t, NVRAM_READ_OUT_READ_BUFFER),
162                     chunk);
163
164                 size -= chunk;
165                 data += chunk;
166                 offset += chunk;
167         }
168
169         return (0);
170
171 fail2:
172         EFSYS_PROBE(fail2);
173 fail1:
174         EFSYS_PROBE1(fail1, int, rc);
175
176         return (rc);
177 }
178
179         __checkReturn           int
180 siena_nvram_partn_erase(
181         __in                    efx_nic_t *enp,
182         __in                    unsigned int partn,
183         __in                    unsigned int offset,
184         __in                    size_t size)
185 {
186         efx_mcdi_req_t req;
187         uint8_t payload[MC_CMD_NVRAM_ERASE_IN_LEN];
188         int rc;
189
190         req.emr_cmd = MC_CMD_NVRAM_ERASE;
191         req.emr_in_buf = payload;
192         req.emr_in_length = MC_CMD_NVRAM_ERASE_IN_LEN;
193         EFX_STATIC_ASSERT(MC_CMD_NVRAM_ERASE_OUT_LEN == 0);
```



```
194         req.emr_out_buf = NULL;
195         req.emr_out_length = 0;
196
197         MCDI_IN_SET_DWORD(req, NVRAM_ERASE_IN_TYPE, partn);
198         MCDI_IN_SET_DWORD(req, NVRAM_ERASE_IN_OFFSET, offset);
199         MCDI_IN_SET_DWORD(req, NVRAM_ERASE_IN_LENGTH, size);
200
201         efx_mcdi_execute(enp, &req);
202
203         if (req.emr_rc != 0) {
204                 rc = req.emr_rc;
205                 goto fail1;
206         }
207
208         return (0);
209
210 fail1:
211         EFSYS_PROBE1(fail1, int, rc);
212
213         return (rc);
214 }
215
216         __checkReturn           int
217 siena_nvram_partn_write(
218         __in                    efx_nic_t *enp,
219         __in                    unsigned int partn,
220         __in                    unsigned int offset,
221         __out_bcount(size)      caddr_t data,
222         __in                    size_t size)
223 {
224         efx_mcdi_req_t req;
225         uint8_t payload[MC_CMD_NVRAM_WRITE_IN_LEN(SIENA_NVRAM_CHUNK)];
226         size_t chunk;
227         int rc;
228
229         while (size > 0) {
230                 chunk = MIN(size, SIENA_NVRAM_CHUNK);
231
232                 req.emr_cmd = MC_CMD_NVRAM_WRITE;
233                 req.emr_in_buf = payload;
234                 req.emr_in_length = MC_CMD_NVRAM_WRITE_IN_LEN(chunk);
235                 EFX_STATIC_ASSERT(MC_CMD_NVRAM_WRITE_OUT_LEN == 0);
236                 req.emr_out_buf = NULL;
237                 req.emr_out_length = 0;
238
239                 MCDI_IN_SET_DWORD(req, NVRAM_WRITE_IN_TYPE, partn);
240                 MCDI_IN_SET_DWORD(req, NVRAM_WRITE_IN_OFFSET, offset);
241                 MCDI_IN_SET_DWORD(req, NVRAM_WRITE_IN_LENGTH, chunk);
242
243                 memcpy(MCDI_IN2(req, uint8_t, NVRAM_WRITE_IN_WRITE_BUFFER),
244                     data, chunk);
245
246                 efx_mcdi_execute(enp, &req);
247
248                 if (req.emr_rc != 0) {
249                         rc = req.emr_rc;
250                         goto fail1;
251                 }
252
253                 size -= chunk;
254                 data += chunk;
255                 offset += chunk;
256         }
257
258         return (0);
```

```
260 fail1:
261         EFSYS_PROBE1(fail1, int, rc);

263         return (rc);
264 }

266                                 void
267 siena_nvram_partn_unlock(
268         __in                    efx_nic_t *enp,
269         __in                    unsigned int partn)
270 {
271         efx_mcdi_req_t req;
272         uint8_t payload[MC_CMD_NVRAM_UPDATE_FINISH_IN_LEN];
273         uint32_t reboot;
274         int rc;

276         req.emr_cmd = MC_CMD_NVRAM_UPDATE_FINISH;
277         req.emr_in_buf = payload;
278         req.emr_in_length = MC_CMD_NVRAM_UPDATE_FINISH_IN_LEN;
279         EFX_STATIC_ASSERT(MC_CMD_NVRAM_UPDATE_FINISH_OUT_LEN == 0);
280         req.emr_out_buf = NULL;
281         req.emr_out_length = 0;

283         /*
284          * Reboot into the new image only for PHYs. The driver has to
285          * explicitly cope with an MC reboot after a firmware update.
286          */
287         reboot = (partn == MC_CMD_NVRAM_TYPE_PHY_PORT0 ||
288                   partn == MC_CMD_NVRAM_TYPE_PHY_PORT1 ||
289                   partn == MC_CMD_NVRAM_TYPE_DISABLED_CALLISTO);

291         MCDI_IN_SET_DWORD(req, NVRAM_UPDATE_FINISH_IN_TYPE, partn);
292         MCDI_IN_SET_DWORD(req, NVRAM_UPDATE_FINISH_IN_REBOOT, reboot);

294         efx_mcdi_execute(enp, &req);

296         if (req.emr_rc != 0) {
297                 rc = req.emr_rc;
298                 goto fail1;
299         }

301         return;

303 fail1:
304         EFSYS_PROBE1(fail1, int, rc);
305 }

307 #endif  /* EFSYS_OPT_VPD || EFSYS_OPT_NVRAM */

309 #if EFSYS_OPT_NVRAM

311 typedef struct siena_parttbl_entry_s {
312         unsigned int            partn;
313         unsigned int            port;
314         efx_nvram_type_t        nvtype;
315 } siena_parttbl_entry_t;

317 static siena_parttbl_entry_t siena_parttbl[] = {
318         {MC_CMD_NVRAM_TYPE_DISABLED_CALLISTO,   1, EFX_NVRAM_NULLPHY},
319         {MC_CMD_NVRAM_TYPE_DISABLED_CALLISTO,   2, EFX_NVRAM_NULLPHY},
320         {MC_CMD_NVRAM_TYPE_MC_FW,               1, EFX_NVRAM_MC_FIRMWARE},
321         {MC_CMD_NVRAM_TYPE_MC_FW,               2, EFX_NVRAM_MC_FIRMWARE},
322         {MC_CMD_NVRAM_TYPE_MC_FW_BACKUP,        1, EFX_NVRAM_MC_GOLDEN},
323         {MC_CMD_NVRAM_TYPE_MC_FW_BACKUP,        2, EFX_NVRAM_MC_GOLDEN},
324         {MC_CMD_NVRAM_TYPE_EXP_ROM,             1, EFX_NVRAM_BOOTROM},
325         {MC_CMD_NVRAM_TYPE_EXP_ROM,             2, EFX_NVRAM_BOOTROM},
```



```
326         {MC_CMD_NVRAM_TYPE_EXP_ROM_CFG_PORT0,   1, EFX_NVRAM_BOOTROM_CFG},
327         {MC_CMD_NVRAM_TYPE_EXP_ROM_CFG_PORT1,   2, EFX_NVRAM_BOOTROM_CFG},
328         {MC_CMD_NVRAM_TYPE_PHY_PORT0,           1, EFX_NVRAM_PHY},
329         {MC_CMD_NVRAM_TYPE_PHY_PORT1,           2, EFX_NVRAM_PHY},
330         {MC_CMD_NVRAM_TYPE_FPGA,                1, EFX_NVRAM_FPGA},
331         {MC_CMD_NVRAM_TYPE_FPGA,                2, EFX_NVRAM_FPGA},
332         {0, 0, 0},
333 };

335 static  __checkReturn           siena_parttbl_entry_t *
336 siena_parttbl_entry(
337         __in                    efx_nic_t *enp,
338         __in                    efx_nvram_type_t type)
339 {
340         efx_mcdi_iface_t *emip = &(enp->en_u.siena.enu_mip);
341         siena_parttbl_entry_t *entry;

343         EFSYS_ASSERT3U(type, <, EFX_NVRAM_NTYPES);

345         for (entry = siena_parttbl; entry->port > 0; ++entry) {
346                 if (entry->port == emip->emi_port && entry->nvtype == type)
347                         return (entry);
348         }

350         return (NULL);
351 }

353 #if EFSYS_OPT_DIAG

355         __checkReturn           int
356 siena_nvram_test(
357         __in                    efx_nic_t *enp)
358 {
359         efx_mcdi_iface_t *emip = &(enp->en_u.siena.enu_mip);
360         siena_parttbl_entry_t *entry;
361         efx_mcdi_req_t req;
362         uint8_t payload[MAX(MC_CMD_NVRAM_TEST_IN_LEN,
363                             MC_CMD_NVRAM_TEST_OUT_LEN)];
364         int result;
365         int rc;

367         req.emr_cmd = MC_CMD_NVRAM_TEST;
368         req.emr_in_buf = payload;
369         req.emr_in_length = MC_CMD_NVRAM_TEST_IN_LEN;
370         req.emr_out_buf = payload;
371         req.emr_out_length = MC_CMD_NVRAM_TEST_OUT_LEN;

373         /*
374          * Iterate over the list of supported partition types
375          * applicable to *this* port
376          */
377         for (entry = siena_parttbl; entry->port > 0; ++entry) {
378                 if (entry->port != emip->emi_port ||
379                     !(enp->en_u.siena.enu_partn_mask & (1 << entry->partn)))
380                         continue;

382                 MCDI_IN_SET_DWORD(req, NVRAM_TEST_IN_TYPE, entry->partn);

384                 efx_mcdi_execute(enp, &req);

386                 if (req.emr_rc != 0) {
387                         rc = req.emr_rc;
388                         goto fail1;
389                 }

391                 if (req.emr_out_length_used < MC_CMD_NVRAM_TEST_OUT_LEN) {
```

```
392                         rc = EMSGSIZE;
393                         goto fail2;
394                 }

396                 result = MCDI_OUT_DWORD(req, NVRAM_TEST_OUT_RESULT);
397                 if (result == MC_CMD_NVRAM_TEST_FAIL) {

399                         EFSYS_PROBE1(nvram_test_failure, int, entry->partn);

401                         rc = (EINVAL);
402                         goto fail3;
403                 }
404         }

406         return (0);

408 fail3:
409         EFSYS_PROBE(fail3);
410 fail2:
411         EFSYS_PROBE(fail2);
412 fail1:
413         EFSYS_PROBE1(fail1, int, rc);

415         return (rc);
416 }

418 #endif  /* EFSYS_OPT_DIAG */

420         __checkReturn           int
421 siena_nvram_size(
422         __in                    efx_nic_t *enp,
423         __in                    efx_nvram_type_t type,
424         __out                   size_t *sizep)
425 {
426         siena_parttbl_entry_t *entry;
427         int rc;

429         if ((entry = siena_parttbl_entry(enp, type)) == NULL) {
430                 rc = ENOTSUP;
431                 goto fail1;
432         }

434         if ((rc = siena_nvram_partn_size(enp, entry->partn, sizep)) != 0)
435                 goto fail2;

437         return (0);

439 fail2:
440         EFSYS_PROBE(fail2);
441 fail1:
442         EFSYS_PROBE1(fail1, int, rc);

444         *sizep = 0;

446         return (rc);
447 }

449 #define SIENA_DYNAMIC_CFG_SIZE(_nitems)                                 \
450         (sizeof (siena_mc_dynamic_config_hdr_t) + ((_nitems) *          \
451         sizeof (((siena_mc_dynamic_config_hdr_t *)NULL)->fw_version[0])))

453         __checkReturn           int
454 siena_nvram_get_dynamic_cfg(
455         __in                    efx_nic_t *enp,
456         __in                    unsigned int partn,
457         __in                    boolean_t vpd,
```



```
458         __out                   siena_mc_dynamic_config_hdr_t **dcfgp,
459         __out                   size_t *sizep)
460 {
461         siena_mc_dynamic_config_hdr_t *dcfg;
462         size_t size;
463         uint8_t cksum;
464         unsigned int vpd_offset;
465         unsigned int vpd_length;
466         unsigned int hdr_length;
467         unsigned int nversions;
468         unsigned int pos;
469         unsigned int region;
470         int rc;

472         EFSYS_ASSERT(partn == MC_CMD_NVRAM_TYPE_DYNAMIC_CFG_PORT0 ||
473                     partn == MC_CMD_NVRAM_TYPE_DYNAMIC_CFG_PORT1);

475         /*
476          * Allocate sufficient memory for the entire dynamiccfg area, even
477          * if we're not actually going to read in the VPD.
478          */
479         if ((rc = siena_nvram_partn_size(enp, partn, &size)) != 0)
480                 goto fail1;

482         EFSYS_KMEM_ALLOC(enp->en_esip, size, dcfg);
483         if (dcfg == NULL) {
484                 rc = ENOMEM;
485                 goto fail2;
486         }

488         if ((rc = siena_nvram_partn_read(enp, partn, 0,
489             (caddr_t)dcfg, SIENA_NVRAM_CHUNK)) != 0)
490                 goto fail3;

492         /* Verify the magic */
493         if (EFX_DWORD_FIELD(dcfg->magic, EFX_DWORD_0)
494             != SIENA_MC_DYNAMIC_CONFIG_MAGIC)
495                 goto invalid1;

497         /* All future versions of the structure must be backwards compatable */
498         EFX_STATIC_ASSERT(SIENA_MC_DYNAMIC_CONFIG_VERSION == 0);

500         hdr_length = EFX_WORD_FIELD(dcfg->length, EFX_WORD_0);
501         nversions = EFX_DWORD_FIELD(dcfg->num_fw_version_items, EFX_DWORD_0);
502         vpd_offset = EFX_DWORD_FIELD(dcfg->dynamic_vpd_offset, EFX_DWORD_0);
503         vpd_length = EFX_DWORD_FIELD(dcfg->dynamic_vpd_length, EFX_DWORD_0);

505         /* Verify the hdr doesn't overflow the partn size */
506         if (hdr_length > size || vpd_offset > size || vpd_length > size ||
507             vpd_length + vpd_offset > size)
508                 goto invalid2;

510         /* Verify the header has room for all it's versions */
511         if (hdr_length < SIENA_DYNAMIC_CFG_SIZE(0) ||
512             hdr_length < SIENA_DYNAMIC_CFG_SIZE(nversions))
513                 goto invalid3;

515         /*
516          * Read the remaining portion of the dcfg, either including
517          * the whole of VPD (there is no vpd length in this structure,
518          * so we have to parse each tag), or just the dcfg header itself
519          */
520         region = vpd ? vpd_offset + vpd_length : hdr_length;
521         if (region > SIENA_NVRAM_CHUNK) {
522                 if ((rc = siena_nvram_partn_read(enp, partn, SIENA_NVRAM_CHUNK,
523                     (caddr_t)dcfg + SIENA_NVRAM_CHUNK,
```

```
524                     region - SIENA_NVRAM_CHUNK)) != 0)
525                         goto fail4;
526         }

528         /* Verify checksum */
529         cksum = 0;
530         for (pos = 0; pos < hdr_length; pos++)
531                 cksum += ((uint8_t *)dcfg)[pos];
532         if (cksum != 0)
533                 goto invalid4;

535         goto done;

537 invalid4:
538         EFSYS_PROBE(invalid4);
539 invalid3:
540         EFSYS_PROBE(invalid3);
541 invalid2:
542         EFSYS_PROBE(invalid2);
543 invalid1:
544         EFSYS_PROBE(invalid1);

546         /*
547          * Construct a new "null" dcfg, with an empty version vector,
548          * and an empty VPD chunk trailing. This has the neat side effect
549          * of testing the exception paths in the write path.
550          */
551         EFX_POPULATE_DWORD_1(dcfg->magic,
552                              EFX_DWORD_0, SIENA_MC_DYNAMIC_CONFIG_MAGIC);
553         EFX_POPULATE_WORD_1(dcfg->length, EFX_WORD_0, sizeof (*dcfg));
554         EFX_POPULATE_BYTE_1(dcfg->version, EFX_BYTE_0,
555                             SIENA_MC_DYNAMIC_CONFIG_VERSION);
556         EFX_POPULATE_DWORD_1(dcfg->dynamic_vpd_offset,
557                              EFX_DWORD_0, sizeof (*dcfg));
558         EFX_POPULATE_DWORD_1(dcfg->dynamic_vpd_length, EFX_DWORD_0, 0);
559         EFX_POPULATE_DWORD_1(dcfg->num_fw_version_items, EFX_DWORD_0, 0);

561 done:
562         *dcfgp = dcfg;
563         *sizep = size;

565         return (0);

567 fail4:
568         EFSYS_PROBE(fail4);
569 fail3:
570         EFSYS_PROBE(fail3);
571 fail2:
572         EFSYS_PROBE(fail2);

574         EFSYS_KMEM_FREE(enp->en_esip, size, dcfg);

576 fail1:
577         EFSYS_PROBE1(fail1, int, rc);

579         return (rc);
580 }

582 static  __checkReturn           int
583 siena_nvram_get_subtype(
584         __in                    efx_nic_t *enp,
585         __in                    unsigned int partn,
586         __out                   uint32_t *subtypep)
587 {
588         efx_mcdi_req_t req;
589         uint8_t outbuf[MC_CMD_GET_BOARD_CFG_OUT_LENMAX];
```



```
590         efx_word_t *fw_list;
591         int rc;

593         req.emr_cmd = MC_CMD_GET_BOARD_CFG;
594         EFX_STATIC_ASSERT(MC_CMD_GET_BOARD_CFG_IN_LEN == 0);
595         req.emr_in_buf = NULL;
596         req.emr_in_length = 0;
597         req.emr_out_buf = outbuf;
598         req.emr_out_length = sizeof (outbuf);

600         efx_mcdi_execute(enp, &req);

602         if (req.emr_rc != 0) {
603                 rc = req.emr_rc;
604                 goto fail1;
605         }

607         if (req.emr_out_length_used < MC_CMD_GET_BOARD_CFG_OUT_LENMIN) {
608                 rc = EMSGSIZE;
609                 goto fail2;
610         }

612         if (req.emr_out_length_used <
613             MC_CMD_GET_BOARD_CFG_OUT_FW_SUBTYPE_LIST_OFST +
614             (partn + 1) * sizeof (efx_word_t)) {
615                 rc = ENOENT;
616                 goto fail3;
617         }

619         fw_list = MCDI_OUT2(req, efx_word_t,
620                             GET_BOARD_CFG_OUT_FW_SUBTYPE_LIST);
621         *subtypep = EFX_WORD_FIELD(fw_list[partn], EFX_WORD_0);

623         return (0);

625 fail3:
626         EFSYS_PROBE(fail3);
627 fail2:
628         EFSYS_PROBE(fail2);
629 fail1:
630         EFSYS_PROBE1(fail1, int, rc);

632         return (rc);
633 }

635         __checkReturn           int
636 siena_nvram_get_version(
637         __in                    efx_nic_t *enp,
638         __in                    efx_nvram_type_t type,
639         __out                   uint32_t *subtypep,
640         __out_ecount(4)         uint16_t version[4])
641 {
642         siena_mc_dynamic_config_hdr_t *dcfg;
643         siena_parttbl_entry_t *entry;
644         unsigned int dcfg_partn;
645         unsigned int partn;
646         int rc;

648         if ((entry = siena_parttbl_entry(enp, type)) == NULL) {
649                 rc = ENOTSUP;
650                 goto fail1;
651         }
652         partn = entry->partn;

654         if ((1 << partn) & ~enp->en_u.siena.enu_partn_mask) {
655                 rc = ENOTSUP;
```

```
656                 goto fail2;
657         }

659         if ((rc = siena_nvram_get_subtype(enp, partn, subtypep)) != 0)
660                 goto fail3;

662         /*
663          * Some partitions are accessible from both ports (for instance BOOTROM)
664          * Find the highest version reported by all dcfg structures on ports
665          * that have access to this partition.
666          */
667         version[0] = version[1] = version[2] = version[3] = 0;
668         for (entry = siena_parttbl; entry->port > 0; ++entry) {
669                 unsigned int nitems;
670                 uint16_t temp[4];
671                 size_t length;

673                 if (entry->partn != partn)
674                         continue;

676                 dcfg_partn = (entry->port == 1)
677                         ? MC_CMD_NVRAM_TYPE_DYNAMIC_CFG_PORT0
678                         : MC_CMD_NVRAM_TYPE_DYNAMIC_CFG_PORT1;
679                 /*
680                  * Ingore missing partitions on port 2, assuming they're due
681                  * to to running on a single port part.
682                  */
683                 if ((1 << dcfg_partn) &  ~enp->en_u.siena.enu_partn_mask) {
684                         if (entry->port == 2)
685                                 continue;
686                 }

688                 if ((rc = siena_nvram_get_dynamic_cfg(enp, dcfg_partn,
689                     B_FALSE, &dcfg, &length)) != 0)
690                         goto fail4;

692                 nitems = EFX_DWORD_FIELD(dcfg->num_fw_version_items,
693                                          EFX_DWORD_0);
694                 if (nitems < entry->partn)
695                         goto done;

697                 temp[0] = EFX_WORD_FIELD(dcfg->fw_version[partn].version_w,
698                                          EFX_WORD_0);
699                 temp[1] = EFX_WORD_FIELD(dcfg->fw_version[partn].version_x,
700                                          EFX_WORD_0);
701                 temp[2] = EFX_WORD_FIELD(dcfg->fw_version[partn].version_y,
702                                          EFX_WORD_0);
703                 temp[3] = EFX_WORD_FIELD(dcfg->fw_version[partn].version_z,
704                                          EFX_WORD_0);
705                 if (memcmp(version, temp, sizeof (temp)) < 0)
706                         memcpy(version, temp, sizeof (temp));

708         done:
709                 EFSYS_KMEM_FREE(enp->en_esip, length, dcfg);
710         }

712         return (0);

714 fail4:
715         EFSYS_PROBE(fail4);
716 fail3:
717         EFSYS_PROBE(fail3);
718 fail2:
719         EFSYS_PROBE(fail2);
720 fail1:
721         EFSYS_PROBE1(fail1, int, rc);
```



```
723         return (rc);
724 }

726         __checkReturn           int
727 siena_nvram_rw_start(
728         __in                    efx_nic_t *enp,
729         __in                    efx_nvram_type_t type,
730         __out                   size_t *chunk_sizep)
731 {
732         siena_parttbl_entry_t *entry;
733         int rc;

735         if ((entry = siena_parttbl_entry(enp, type)) == NULL) {
736                 rc = ENOTSUP;
737                 goto fail1;
738         }

740         if ((rc = siena_nvram_partn_lock(enp, entry->partn)) != 0)
741                 goto fail2;

743         if (chunk_sizep != NULL)
744                 *chunk_sizep = SIENA_NVRAM_CHUNK;

746         return (0);

748 fail2:
749         EFSYS_PROBE(fail2);
750 fail1:
751         EFSYS_PROBE1(fail1, int, rc);

753         return (rc);
754 }

756         __checkReturn           int
757 siena_nvram_read_chunk(
758         __in                    efx_nic_t *enp,
759         __in                    efx_nvram_type_t type,
760         __in                    unsigned int offset,
761         __out_bcount(size)      caddr_t data,
762         __in                    size_t size)
763 {
764         siena_parttbl_entry_t *entry;
765         int rc;

767         if ((entry = siena_parttbl_entry(enp, type)) == NULL) {
768                 rc = ENOTSUP;
769                 goto fail1;
770         }

772         if ((rc = siena_nvram_partn_read(enp, entry->partn,
773             offset, data, size)) != 0)
774                 goto fail2;

776         return (0);

778 fail2:
779         EFSYS_PROBE(fail2);
780 fail1:
781         EFSYS_PROBE1(fail1, int, rc);

783         return (rc);
784 }

786         __checkReturn           int
787 siena_nvram_erase(
```

```
788         __in                    efx_nic_t *enp,
789         __in                    efx_nvram_type_t type)
790 {
791         siena_parttbl_entry_t *entry;
792         size_t size;
793         int rc;

795         if ((entry = siena_parttbl_entry(enp, type)) == NULL) {
796                 rc = ENOTSUP;
797                 goto fail1;
798         }

800         if ((rc = siena_nvram_partn_size(enp, entry->partn, &size)) != 0)
801                 goto fail2;

803         if ((rc = siena_nvram_partn_erase(enp, entry->partn, 0, size)) != 0)
804                 goto fail3;

806         return (0);

808 fail3:
809         EFSYS_PROBE(fail3);
810 fail2:
811         EFSYS_PROBE(fail2);
812 fail1:
813         EFSYS_PROBE1(fail1, int, rc);

815         return (rc);
816 }

818         __checkReturn           int
819 siena_nvram_write_chunk(
820         __in                    efx_nic_t *enp,
821         __in                    efx_nvram_type_t type,
822         __in                    unsigned int offset,
823         __in_bcount(size)       caddr_t data,
824         __in                    size_t size)
825 {
826         siena_parttbl_entry_t *entry;
827         int rc;

829         if ((entry = siena_parttbl_entry(enp, type)) == NULL) {
830                 rc = ENOTSUP;
831                 goto fail1;
832         }

834         if ((rc = siena_nvram_partn_write(enp, entry->partn,
835             offset, data, size)) != 0)
836                 goto fail2;

838         return (0);

840 fail2:
841         EFSYS_PROBE(fail2);
842 fail1:
843         EFSYS_PROBE1(fail1, int, rc);

845         return (rc);
846 }

848                                 void
849 siena_nvram_rw_finish(
850         __in                    efx_nic_t *enp,
851         __in                    efx_nvram_type_t type)
852 {
853         siena_parttbl_entry_t *entry;
```



```
855         if ((entry = siena_parttbl_entry(enp, type)) != NULL)
856                 siena_nvram_partn_unlock(enp, entry->partn);
857 }

859         __checkReturn           int
860 siena_nvram_set_version(
861         __in                    efx_nic_t *enp,
862         __in                    efx_nvram_type_t type,
863         __out                   uint16_t version[4])
864 {
865         siena_mc_dynamic_config_hdr_t *dcfg = NULL;
866         siena_parttbl_entry_t *entry;
867         unsigned int dcfg_partn;
868         size_t partn_size;
869         unsigned int hdr_length;
870         unsigned int vpd_length;
871         unsigned int vpd_offset;
872         unsigned int nitems;
873         unsigned int required_hdr_length;
874         unsigned int pos;
875         uint8_t cksum;
876         uint32_t subtype;
877         size_t length;
878         int rc;

880         if ((entry = siena_parttbl_entry(enp, type)) == NULL) {
881                 rc = ENOTSUP;
882                 goto fail1;
883         }

885         dcfg_partn = (entry->port == 1)
886                 ? MC_CMD_NVRAM_TYPE_DYNAMIC_CFG_PORT0
887                 : MC_CMD_NVRAM_TYPE_DYNAMIC_CFG_PORT1;

889         if ((rc = siena_nvram_partn_size(enp, dcfg_partn, &partn_size)) != 0)
890                 goto fail2;

892         if ((rc = siena_nvram_partn_lock(enp, dcfg_partn)) != 0)
893                 goto fail2;

895         if ((rc = siena_nvram_get_dynamic_cfg(enp, dcfg_partn,
896             B_TRUE, &dcfg, &length)) != 0)
897                 goto fail3;

899         hdr_length = EFX_WORD_FIELD(dcfg->length, EFX_WORD_0);
900         nitems = EFX_DWORD_FIELD(dcfg->num_fw_version_items, EFX_DWORD_0);
901         vpd_length = EFX_DWORD_FIELD(dcfg->dynamic_vpd_length, EFX_DWORD_0);
902         vpd_offset = EFX_DWORD_FIELD(dcfg->dynamic_vpd_offset, EFX_DWORD_0);

904         /*
905          * NOTE: This function will blatt any fields trailing the version
906          * vector, or the VPD chunk.
907          */
908         required_hdr_length = SIENA_DYNAMIC_CFG_SIZE(entry->partn + 1);
909         if (required_hdr_length + vpd_length > length) {
910                 rc = ENOSPC;
911                 goto fail4;
912         }

914         if (vpd_offset < required_hdr_length) {
915                 (void) memmove((caddr_t)dcfg + required_hdr_length,
916                                (caddr_t)dcfg + vpd_offset, vpd_length);
917                 vpd_offset = required_hdr_length;
918                 EFX_POPULATE_DWORD_1(dcfg->dynamic_vpd_offset,
919                                      EFX_DWORD_0, vpd_offset);
```

```
920         }

922         if (hdr_length < required_hdr_length) {
923                 (void) memset((caddr_t)dcfg + hdr_length, 0,
924                         required_hdr_length - hdr_length);
925                 hdr_length = required_hdr_length;
926                 EFX_POPULATE_WORD_1(dcfg->length,
927                                     EFX_WORD_0, hdr_length);
928         }

930         /* Get the subtype to insert into the fw_subtype array */
931         if ((rc = siena_nvram_get_subtype(enp, entry->partn, &subtype)) != 0)
932                 goto fail5;

934         /* Fill out the new version */
935         EFX_POPULATE_DWORD_1(dcfg->fw_version[entry->partn].fw_subtype,
936                              EFX_DWORD_0, subtype);
937         EFX_POPULATE_WORD_1(dcfg->fw_version[entry->partn].version_w,
938                             EFX_WORD_0, version[0]);
939         EFX_POPULATE_WORD_1(dcfg->fw_version[entry->partn].version_x,
940                             EFX_WORD_0, version[1]);
941         EFX_POPULATE_WORD_1(dcfg->fw_version[entry->partn].version_y,
942                             EFX_WORD_0, version[2]);
943         EFX_POPULATE_WORD_1(dcfg->fw_version[entry->partn].version_z,
944                             EFX_WORD_0, version[3]);

946         /* Update the version count */
947         if (nitems < entry->partn + 1) {
948                 nitems = entry->partn + 1;
949                 EFX_POPULATE_DWORD_1(dcfg->num_fw_version_items,
950                                      EFX_DWORD_0, nitems);
951         }

953         /* Update the checksum */
954         cksum = 0;
955         for (pos = 0; pos < hdr_length; pos++)
956                 cksum += ((uint8_t *)dcfg)[pos];
957         dcfg->csum.eb_u8[0] -= cksum;

959         /* Erase and write the new partition */
960         if ((rc = siena_nvram_partn_erase(enp, dcfg_partn, 0, partn_size)) != 0)
961                 goto fail6;

963         /* Write out the new structure to nvram */
964         if ((rc = siena_nvram_partn_write(enp, dcfg_partn, 0,
965             (caddr_t)dcfg, vpd_offset + vpd_length)) != 0)
966                 goto fail7;

968         EFSYS_KMEM_FREE(enp->en_esip, length, dcfg);

970         siena_nvram_partn_unlock(enp, dcfg_partn);

972         return (0);

974 fail7:
975         EFSYS_PROBE(fail7);
976 fail6:
977         EFSYS_PROBE(fail6);
978 fail5:
979         EFSYS_PROBE(fail5);
980 fail4:
981         EFSYS_PROBE(fail4);

983         EFSYS_KMEM_FREE(enp->en_esip, length, dcfg);
984 fail3:
985         EFSYS_PROBE(fail3);
```



```
986 fail2:
987         EFSYS_PROBE(fail2);
988 fail1:
989         EFSYS_PROBE1(fail1, int, rc);

991         return (rc);
992 }

994 #endif  /* EFSYS_OPT_NVRAM */

996 #endif  /* EFSYS_OPT_SIENA */
997 #endif /* ! codereview */
```

```
**********************************************************
   23487 Thu Aug 22 18:59:29 2013
new/usr/src/uts/common/io/sfxge/siena_phy.c
Merged sfxge driver
**********************************************************
```

```
/*
 * Copyright 2009 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */
#include "efsys.h"
#include "efx.h"
#include "efx_impl.h"

#if EFSYS_OPT_SIENA

static                  void
siena_phy_decode_cap(
        __in            uint32_t mcdi_cap,
        __out           uint32_t *maskp)
{
        uint32_t mask;

        mask = 0;
        if (mcdi_cap & (1 << MC_CMD_PHY_CAP_10HDX_LBN))
                mask |= (1 << EFX_PHY_CAP_10HDX);
        if (mcdi_cap & (1 << MC_CMD_PHY_CAP_10FDX_LBN))
                mask |= (1 << EFX_PHY_CAP_10FDX);
        if (mcdi_cap & (1 << MC_CMD_PHY_CAP_100HDX_LBN))
                mask |= (1 << EFX_PHY_CAP_100HDX);
        if (mcdi_cap & (1 << MC_CMD_PHY_CAP_100FDX_LBN))
                mask |= (1 << EFX_PHY_CAP_100FDX);
        if (mcdi_cap & (1 << MC_CMD_PHY_CAP_1000HDX_LBN))
                mask |= (1 << EFX_PHY_CAP_1000HDX);
        if (mcdi_cap & (1 << MC_CMD_PHY_CAP_1000FDX_LBN))
                mask |= (1 << EFX_PHY_CAP_1000FDX);
        if (mcdi_cap & (1 << MC_CMD_PHY_CAP_10000FDX_LBN))
                mask |= (1 << EFX_PHY_CAP_10000FDX);
        if (mcdi_cap & (1 << MC_CMD_PHY_CAP_PAUSE_LBN))
                mask |= (1 << EFX_PHY_CAP_PAUSE);
        if (mcdi_cap & (1 << MC_CMD_PHY_CAP_ASYM_LBN))
                mask |= (1 << EFX_PHY_CAP_ASYM);
        if (mcdi_cap & (1 << MC_CMD_PHY_CAP_AN_LBN))
                mask |= (1 << EFX_PHY_CAP_AN);

        *maskp = mask;
}
```



```
static                  void
siena_phy_decode_link_mode(
        __in            efx_nic_t *enp,
        __in            uint32_t link_flags,
        __in            unsigned int speed,
        __in            unsigned int fcntl,
        __out           efx_link_mode_t *link_modep,
        __out           unsigned int *fcntlp)
{
        boolean_t fd = !!(link_flags &
                    (1 << MC_CMD_GET_LINK_OUT_FULL_DUPLEX_LBN));
        boolean_t up = !!(link_flags &
                    (1 << MC_CMD_GET_LINK_OUT_LINK_UP_LBN));

        _NOTE(ARGUNUSED(enp))

        if (!up)
                *link_modep = EFX_LINK_DOWN;
        else if (speed == 10000 && fd)
                *link_modep = EFX_LINK_10000FDX;
        else if (speed == 1000)
                *link_modep = fd ? EFX_LINK_1000FDX : EFX_LINK_1000HDX;
        else if (speed == 100)
                *link_modep = fd ? EFX_LINK_100FDX : EFX_LINK_100HDX;
        else if (speed == 10)
                *link_modep = fd ? EFX_LINK_10FDX : EFX_LINK_10HDX;
        else
                *link_modep = EFX_LINK_UNKNOWN;

        if (fcntl == MC_CMD_FCNTL_OFF)
                *fcntlp = 0;
        else if (fcntl == MC_CMD_FCNTL_RESPOND)
                *fcntlp = EFX_FCNTL_RESPOND;
        else if (fcntl == MC_CMD_FCNTL_BIDIR)
                *fcntlp = EFX_FCNTL_RESPOND | EFX_FCNTL_GENERATE;
        else {
                EFSYS_PROBE1(mc_pcol_error, int, fcntl);
                *fcntlp = 0;
        }
}

                        void
siena_phy_link_ev(
        __in            efx_nic_t *enp,
        __in            efx_qword_t *eqp,
        __out           efx_link_mode_t *link_modep)
{
        efx_port_t *epp = &(enp->en_port);
        unsigned int link_flags;
        unsigned int speed;
        unsigned int fcntl;
        efx_link_mode_t link_mode;
        uint32_t lp_cap_mask;

        /*
         * Convert the LINKCHANGE speed enumeration into mbit/s, in the
         * same way as GET_LINK encodes the speed
         */
        switch (MCDI_EV_FIELD(eqp, LINKCHANGE_SPEED)) {
        case MCDI_EVENT_LINKCHANGE_SPEED_100M:
                speed = 100;
                break;
        case MCDI_EVENT_LINKCHANGE_SPEED_1G:
                speed = 1000;
                break;
```

```
128         case MCDI_EVENT_LINKCHANGE_SPEED_10G:
129                 speed = 10000;
130                 break;
131         default:
132                 speed = 0;
133                 break;
134         }

136         link_flags = MCDI_EV_FIELD(eqp, LINKCHANGE_LINK_FLAGS);
137         siena_phy_decode_link_mode(enp, link_flags, speed,
138                                    MCDI_EV_FIELD(eqp, LINKCHANGE_FCNTL),
139                                    &link_mode, &fcntl);
140         siena_phy_decode_cap(MCDI_EV_FIELD(eqp, LINKCHANGE_LP_CAP),
141                              &lp_cap_mask);

143         /*
144          * It's safe to update ep_lp_cap_mask without the driver's port lock
145          * because presumably any concurrently running efx_port_poll() is
146          * only going to arrive at the same value.
147          *
148          * ep_fcntl has two meanings. It's either the link common fcntl
149          * (if the PHY supports AN), or it's the forced link state. If
150          * the former, it's safe to update the value for the same reason as
151          * for ep_lp_cap_mask. If the latter, then just ignore the value,
152          * because we can race with efx_mac_fcntl_set().
153          */
154         epp->ep_lp_cap_mask = lp_cap_mask;
155         if (epp->ep_phy_cap_mask & (1 << EFX_PHY_CAP_AN))
156                 epp->ep_fcntl = fcntl;

158         *link_modep = link_mode;
159 }

161         __checkReturn   int
162 siena_phy_power(
163         __in            efx_nic_t *enp,
164         __in            boolean_t power)
165 {
166         int rc;

168         if (!power)
169                 return (0);

171         /* Check if the PHY is a zombie */
172         if ((rc = siena_phy_verify(enp)) != 0)
173                 goto fail1;

175         enp->en_reset_flags |= EFX_RESET_PHY;

177         return (0);

179 fail1:
180         EFSYS_PROBE1(fail1, int, rc);

182         return (rc);
183 }

185         __checkReturn   int
186 siena_phy_get_link(
187         __in            efx_nic_t *enp,
188         __out           siena_link_state_t *slsp)
189 {
190         efx_mcdi_req_t req;
191         uint8_t outbuf[MC_CMD_GET_LINK_OUT_LEN];
192         int rc;
```



```
194         req.emr_cmd = MC_CMD_GET_LINK;
195         EFX_STATIC_ASSERT(MC_CMD_GET_LINK_IN_LEN == 0);
196         req.emr_in_buf = NULL;
197         req.emr_in_length = 0;
198         req.emr_out_buf = outbuf;
199         req.emr_out_length = sizeof (outbuf);

201         efx_mcdi_execute(enp, &req);

203         if (req.emr_rc != 0) {
204                 rc = req.emr_rc;
205                 goto fail1;
206         }

208         if (req.emr_out_length_used < MC_CMD_GET_LINK_OUT_LEN) {
209                 rc = EMSGSIZE;
210                 goto fail2;
211         }

213         siena_phy_decode_cap(MCDI_OUT_DWORD(req, GET_LINK_OUT_CAP),
214                              &slsp->sls_adv_cap_mask);
215         siena_phy_decode_cap(MCDI_OUT_DWORD(req, GET_LINK_OUT_LP_CAP),
216                              &slsp->sls_lp_cap_mask);

218         siena_phy_decode_link_mode(enp, MCDI_OUT_DWORD(req, GET_LINK_OUT_FLAGS),
219                                    MCDI_OUT_DWORD(req, GET_LINK_OUT_LINK_SPEED),
220                                    MCDI_OUT_DWORD(req, GET_LINK_OUT_FCNTL),
221                                    &slsp->sls_link_mode, &slsp->sls_fcntl);

223 #if EFSYS_OPT_LOOPBACK
224         /* Assert the MC_CMD_LOOPBACK and EFX_LOOPBACK namespace agree */
225         EFX_STATIC_ASSERT(MC_CMD_LOOPBACK_NONE == EFX_LOOPBACK_OFF);
226         EFX_STATIC_ASSERT(MC_CMD_LOOPBACK_DATA == EFX_LOOPBACK_DATA);
227         EFX_STATIC_ASSERT(MC_CMD_LOOPBACK_GMAC == EFX_LOOPBACK_GMAC);
228         EFX_STATIC_ASSERT(MC_CMD_LOOPBACK_XGMII == EFX_LOOPBACK_XGMII);
229         EFX_STATIC_ASSERT(MC_CMD_LOOPBACK_XGXS == EFX_LOOPBACK_XGXS);
230         EFX_STATIC_ASSERT(MC_CMD_LOOPBACK_XAUI == EFX_LOOPBACK_XAUI);
231         EFX_STATIC_ASSERT(MC_CMD_LOOPBACK_GMII == EFX_LOOPBACK_GMII);
232         EFX_STATIC_ASSERT(MC_CMD_LOOPBACK_SGMII == EFX_LOOPBACK_SGMII);
233         EFX_STATIC_ASSERT(MC_CMD_LOOPBACK_XGBR == EFX_LOOPBACK_XGBR);
234         EFX_STATIC_ASSERT(MC_CMD_LOOPBACK_XFI == EFX_LOOPBACK_XFI);
235         EFX_STATIC_ASSERT(MC_CMD_LOOPBACK_XAUI_FAR == EFX_LOOPBACK_XAUI_FAR);
236         EFX_STATIC_ASSERT(MC_CMD_LOOPBACK_GMII_FAR == EFX_LOOPBACK_GMII_FAR);
237         EFX_STATIC_ASSERT(MC_CMD_LOOPBACK_SGMII_FAR == EFX_LOOPBACK_SGMII_FAR);
238         EFX_STATIC_ASSERT(MC_CMD_LOOPBACK_XFI_FAR == EFX_LOOPBACK_XFI_FAR);
239         EFX_STATIC_ASSERT(MC_CMD_LOOPBACK_GPHY == EFX_LOOPBACK_GPHY);
240         EFX_STATIC_ASSERT(MC_CMD_LOOPBACK_PHYXS == EFX_LOOPBACK_PHY_XS);
241         EFX_STATIC_ASSERT(MC_CMD_LOOPBACK_PCS == EFX_LOOPBACK_PCS);
242         EFX_STATIC_ASSERT(MC_CMD_LOOPBACK_PMAPMD == EFX_LOOPBACK_PMA_PMD);

244         slsp->sls_loopback = MCDI_OUT_DWORD(req, GET_LINK_OUT_LOOPBACK_MODE);
245 #endif  /* EFSYS_OPT_LOOPBACK */

247         slsp->sls_mac_up = MCDI_OUT_DWORD(req, GET_LINK_OUT_MAC_FAULT) == 0;

249         return (0);

251 fail2:
252         EFSYS_PROBE(fail2);
253 fail1:
254         EFSYS_PROBE1(fail1, int, rc);

256         return (rc);
257 }

259         __checkReturn   int
```

```
siena_phy_reconfigure(
        __in            efx_nic_t *enp)
{
        efx_port_t *epp = &(enp->en_port);
        efx_mcdi_req_t req;
        uint8_t payload[MAX(MC_CMD_SET_ID_LED_IN_LEN,
                            MC_CMD_SET_LINK_IN_LEN)];
        uint32_t cap_mask;
        unsigned int led_mode;
        unsigned int speed;
        int rc;

        req.emr_cmd = MC_CMD_SET_LINK;
        req.emr_in_buf = payload;
        req.emr_in_length = MC_CMD_SET_LINK_IN_LEN;
        EFX_STATIC_ASSERT(MC_CMD_SET_LINK_OUT_LEN == 0);
        req.emr_out_buf = NULL;
        req.emr_out_length = 0;

        cap_mask = epp->ep_adv_cap_mask;
        MCDI_IN_POPULATE_DWORD_10(req, SET_LINK_IN_CAP,
                PHY_CAP_10HDX, (cap_mask >> EFX_PHY_CAP_10HDX) & 0x1,
                PHY_CAP_10FDX, (cap_mask >> EFX_PHY_CAP_10FDX) & 0x1,
                PHY_CAP_100HDX, (cap_mask >> EFX_PHY_CAP_100HDX) & 0x1,
                PHY_CAP_100FDX, (cap_mask >> EFX_PHY_CAP_100FDX) & 0x1,
                PHY_CAP_1000HDX, (cap_mask >> EFX_PHY_CAP_1000HDX) & 0x1,
                PHY_CAP_1000FDX, (cap_mask >> EFX_PHY_CAP_1000FDX) & 0x1,
                PHY_CAP_10000FDX, (cap_mask >> EFX_PHY_CAP_10000FDX) & 0x1,
                PHY_CAP_PAUSE, (cap_mask >> EFX_PHY_CAP_PAUSE) & 0x1,
                PHY_CAP_ASYM, (cap_mask >> EFX_PHY_CAP_ASYM) & 0x1,
                PHY_CAP_AN, (cap_mask >> EFX_PHY_CAP_AN) & 0x1);

#if EFSYS_OPT_LOOPBACK
        MCDI_IN_SET_DWORD(req, SET_LINK_IN_LOOPBACK_MODE,
                          epp->ep_loopback_type);
        switch (epp->ep_loopback_link_mode) {
        case EFX_LINK_100FDX:
                speed = 100;
                break;
        case EFX_LINK_1000FDX:
                speed = 1000;
                break;
        case EFX_LINK_10000FDX:
                speed = 10000;
                break;
        default:
                speed = 0;
        }
#else
        MCDI_IN_SET_DWORD(req, SET_LINK_IN_LOOPBACK_MODE, MC_CMD_LOOPBACK_NONE);
        speed = 0;
#endif  /* EFSYS_OPT_LOOPBACK */
        MCDI_IN_SET_DWORD(req, SET_LINK_IN_LOOPBACK_SPEED, speed);

#if EFSYS_OPT_PHY_FLAGS
        MCDI_IN_SET_DWORD(req, SET_LINK_IN_FLAGS, epp->ep_phy_flags);
#else
        MCDI_IN_SET_DWORD(req, SET_LINK_IN_FLAGS, 0);
#endif  /* EFSYS_OPT_PHY_FLAGS */

        efx_mcdi_execute(enp, &req);

        if (req.emr_rc != 0) {
                rc = req.emr_rc;
                goto fail1;
        }
```



```
        /* And set the blink mode */
        req.emr_cmd = MC_CMD_SET_ID_LED;
        req.emr_in_buf = payload;
        req.emr_in_length = MC_CMD_SET_ID_LED_IN_LEN;
        EFX_STATIC_ASSERT(MC_CMD_SET_ID_LED_OUT_LEN == 0);
        req.emr_out_buf = NULL;
        req.emr_out_length = 0;

#if EFSYS_OPT_PHY_LED_CONTROL
        switch (epp->ep_phy_led_mode) {
        case EFX_PHY_LED_DEFAULT:
                led_mode = MC_CMD_LED_DEFAULT;
                break;
        case EFX_PHY_LED_OFF:
                led_mode = MC_CMD_LED_OFF;
                break;
        case EFX_PHY_LED_ON:
                led_mode = MC_CMD_LED_ON;
                break;
        default:
                EFSYS_ASSERT(0);
                led_mode = MC_CMD_LED_DEFAULT;
        }

        MCDI_IN_SET_DWORD(req, SET_ID_LED_IN_STATE, led_mode);
#else
        MCDI_IN_SET_DWORD(req, SET_ID_LED_IN_STATE, MC_CMD_LED_DEFAULT);
#endif  /* EFSYS_OPT_PHY_LED_CONTROL */

        efx_mcdi_execute(enp, &req);

        if (req.emr_rc != 0) {
                rc = req.emr_rc;
                goto fail2;
        }

        return (0);

fail2:
        EFSYS_PROBE(fail2);
fail1:
        EFSYS_PROBE1(fail1, int, rc);

        return (rc);
}

        __checkReturn   int
siena_phy_verify(
        __in            efx_nic_t *enp)
{
        efx_mcdi_req_t req;
        uint8_t outbuf[MC_CMD_GET_PHY_STATE_OUT_LEN];
        uint32_t state;
        int rc;

        req.emr_cmd = MC_CMD_GET_PHY_STATE;
        EFX_STATIC_ASSERT(MC_CMD_GET_PHY_STATE_IN_LEN == 0);
        req.emr_in_buf = NULL;
        req.emr_in_length = 0;
        req.emr_out_buf = outbuf;
        req.emr_out_length = sizeof (outbuf);

        efx_mcdi_execute(enp, &req);

        if (req.emr_rc != 0) {
```

```
392                 rc = req.emr_rc;
393                 goto fail1;
394         }

396         if (req.emr_out_length_used < MC_CMD_GET_PHY_STATE_OUT_LEN) {
397                 rc = EMSGSIZE;
398                 goto fail2;
399         }

401         state = MCDI_OUT_DWORD(req, GET_PHY_STATE_OUT_STATE);
402         if (state != MC_CMD_PHY_STATE_OK) {
403                 if (state != MC_CMD_PHY_STATE_ZOMBIE)
404                         EFSYS_PROBE1(mc_pcol_error, int, state);
405                 rc = ENOTACTIVE;
406                 goto fail3;
407         }

409         return (0);

411 fail3:
412         EFSYS_PROBE(fail3);
413 fail2:
414         EFSYS_PROBE(fail2);
415 fail1:
416         EFSYS_PROBE1(fail1, int, rc);

418         return (rc);
419 }

421         __checkReturn   int
422 siena_phy_oui_get(
423         __in            efx_nic_t *enp,
424         __out           uint32_t *ouip)
425 {
426         _NOTE(ARGUNUSED(enp, ouip))

428         return (ENOTSUP);
429 }

431 #if EFSYS_OPT_PHY_STATS

433 #define SIENA_SIMPLE_STAT_SET(_vmask, _esmp, _smask, _stat,             \
434                               _mc_record, _efx_record)                  \
435         if ((_vmask) & (1ULL << (_mc_record))) {                        \
436                 (_smask) |= (1ULL << (_efx_record));                    \
437                 if ((_stat) != NULL && !EFSYS_MEM_IS_NULL(_esmp)) {     \
438                         efx_dword_t dword;                              \
439                         EFSYS_MEM_READD(_esmp, (_mc_record) * 4, &dword);\
440                         (_stat)[_efx_record] =                          \
441                                 EFX_DWORD_FIELD(dword, EFX_DWORD_0);    \
442                 }                                                       \
443         }

445 #define SIENA_SIMPLE_STAT_SET2(_vmask, _esmp, _smask, _stat, _record)   \
446         SIENA_SIMPLE_STAT_SET(_vmask, _esmp, _smask, _stat,             \
447                               MC_CMD_ ## _record,                       \
448                               EFX_PHY_STAT_ ## _record)

450                                         void
451 siena_phy_decode_stats(
452         __in                            efx_nic_t *enp,
453         __in                            uint32_t vmask,
454         __in_opt                        efsys_mem_t *esmp,
455         __out_opt                       uint64_t *smaskp,
456         __out_ecount_opt(EFX_PHY_NSTATS)        uint32_t *stat)
457 {
```



```
458         uint64_t smask = 0;

460         _NOTE(ARGUNUSED(enp))

462         SIENA_SIMPLE_STAT_SET2(vmask, esmp, smask, stat, OUI);
463         SIENA_SIMPLE_STAT_SET2(vmask, esmp, smask, stat, PMA_PMD_LINK_UP);
464         SIENA_SIMPLE_STAT_SET2(vmask, esmp, smask, stat, PMA_PMD_RX_FAULT);
465         SIENA_SIMPLE_STAT_SET2(vmask, esmp, smask, stat, PMA_PMD_TX_FAULT);

467         if (vmask & (1 << MC_CMD_PMA_PMD_SIGNAL)) {
468                 smask |=   ((1ULL << EFX_PHY_STAT_PMA_PMD_SIGNAL_A) |
469                             (1ULL << EFX_PHY_STAT_PMA_PMD_SIGNAL_B) |
470                             (1ULL << EFX_PHY_STAT_PMA_PMD_SIGNAL_C) |
471                             (1ULL << EFX_PHY_STAT_PMA_PMD_SIGNAL_D));
472                 if (stat != NULL && esmp != NULL && !EFSYS_MEM_IS_NULL(esmp)) {
473                         efx_dword_t dword;
474                         uint32_t sig;
475                         EFSYS_MEM_READD(esmp, 4 * MC_CMD_PMA_PMD_SIGNAL,
476                                         &dword);
477                         sig = EFX_DWORD_FIELD(dword, EFX_DWORD_0);
478                         stat[EFX_PHY_STAT_PMA_PMD_SIGNAL_A] = (sig >> 1) & 1;
479                         stat[EFX_PHY_STAT_PMA_PMD_SIGNAL_B] = (sig >> 2) & 1;
480                         stat[EFX_PHY_STAT_PMA_PMD_SIGNAL_C] = (sig >> 3) & 1;
481                         stat[EFX_PHY_STAT_PMA_PMD_SIGNAL_D] = (sig >> 4) & 1;
482                 }
483         }

485         SIENA_SIMPLE_STAT_SET(vmask, esmp, smask, stat, MC_CMD_PMA_PMD_SNR_A,
486                               EFX_PHY_STAT_SNR_A);
487         SIENA_SIMPLE_STAT_SET(vmask, esmp, smask, stat, MC_CMD_PMA_PMD_SNR_B,
488                               EFX_PHY_STAT_SNR_B);
489         SIENA_SIMPLE_STAT_SET(vmask, esmp, smask, stat, MC_CMD_PMA_PMD_SNR_C,
490                               EFX_PHY_STAT_SNR_C);
491         SIENA_SIMPLE_STAT_SET(vmask, esmp, smask, stat, MC_CMD_PMA_PMD_SNR_D,
492                               EFX_PHY_STAT_SNR_D);

494         SIENA_SIMPLE_STAT_SET2(vmask, esmp, smask, stat, PCS_LINK_UP);
495         SIENA_SIMPLE_STAT_SET2(vmask, esmp, smask, stat, PCS_RX_FAULT);
496         SIENA_SIMPLE_STAT_SET2(vmask, esmp, smask, stat, PCS_TX_FAULT);
497         SIENA_SIMPLE_STAT_SET2(vmask, esmp, smask, stat, PCS_BER);
498         SIENA_SIMPLE_STAT_SET2(vmask, esmp, smask, stat, PCS_BLOCK_ERRORS);

500         SIENA_SIMPLE_STAT_SET(vmask, esmp, smask, stat, MC_CMD_PHYXS_LINK_UP,
501                               EFX_PHY_STAT_PHY_XS_LINK_UP);
502         SIENA_SIMPLE_STAT_SET(vmask, esmp, smask, stat, MC_CMD_PHYXS_RX_FAULT,
503                               EFX_PHY_STAT_PHY_XS_RX_FAULT);
504         SIENA_SIMPLE_STAT_SET(vmask, esmp, smask, stat, MC_CMD_PHYXS_TX_FAULT,
505                               EFX_PHY_STAT_PHY_XS_TX_FAULT);
506         SIENA_SIMPLE_STAT_SET(vmask, esmp, smask, stat, MC_CMD_PHYXS_ALIGN,
507                               EFX_PHY_STAT_PHY_XS_ALIGN);

509         if (vmask & (1 << MC_CMD_PHYXS_SYNC)) {
510                 smask |=   ((1 << EFX_PHY_STAT_PHY_XS_SYNC_A) |
511                             (1 << EFX_PHY_STAT_PHY_XS_SYNC_B) |
512                             (1 << EFX_PHY_STAT_PHY_XS_SYNC_C) |
513                             (1 << EFX_PHY_STAT_PHY_XS_SYNC_D));
514                 if (stat != NULL && !EFSYS_MEM_IS_NULL(esmp)) {
515                         efx_dword_t dword;
516                         uint32_t sync;
517                         EFSYS_MEM_READD(esmp, 4 * MC_CMD_PHYXS_SYNC, &dword);
518                         sync = EFX_DWORD_FIELD(dword, EFX_DWORD_0);
519                         stat[EFX_PHY_STAT_PHY_XS_SYNC_A] = (sync >> 0) & 1;
520                         stat[EFX_PHY_STAT_PHY_XS_SYNC_B] = (sync >> 1) & 1;
521                         stat[EFX_PHY_STAT_PHY_XS_SYNC_C] = (sync >> 2) & 1;
522                         stat[EFX_PHY_STAT_PHY_XS_SYNC_D] = (sync >> 3) & 1;
523                 }
```

```
524         }
526         SIENA_SIMPLE_STAT_SET2(vmask, esmp, smask, stat, AN_LINK_UP);
527         SIENA_SIMPLE_STAT_SET2(vmask, esmp, smask, stat, AN_COMPLETE);
529         SIENA_SIMPLE_STAT_SET(vmask, esmp, smask, stat, MC_CMD_CL22_LINK_UP,
530                               EFX_PHY_STAT_CL22EXT_LINK_UP);
532         if (smaskp != NULL)
533                 *smaskp = smask;
534 }
536         __checkReturn                           int
537 siena_phy_stats_update(
538         __in                                    efx_nic_t *enp,
539         __in                                    efsys_mem_t *esmp,
540         __out_ecount(EFX_PHY_NSTATS)            uint32_t *stat)
541 {
542         efx_nic_cfg_t *encp = &(enp->en_nic_cfg);
543         uint32_t vmask = encp->enc_siena_phy_stat_mask;
544         uint8_t payload[MC_CMD_PHY_STATS_IN_LEN];
545         uint64_t smask;
546         efx_mcdi_req_t req;
547         int rc;
549         req.emr_cmd = MC_CMD_PHY_STATS;
550         req.emr_in_buf = payload;
551         req.emr_in_length = sizeof (payload);
552         EFX_STATIC_ASSERT(MC_CMD_PHY_STATS_OUT_DMA_LEN == 0);
553         req.emr_out_buf = NULL;
554         req.emr_out_length = 0;
556         MCDI_IN_SET_DWORD(req, PHY_STATS_IN_DMA_ADDR_LO,
557                           EFSYS_MEM_ADDR(esmp) & 0xffffffff);
558         MCDI_IN_SET_DWORD(req, PHY_STATS_IN_DMA_ADDR_HI,
559                           EFSYS_MEM_ADDR(esmp) >> 32);
561         efx_mcdi_execute(enp, &req);
563         if (req.emr_rc != 0) {
564                 rc = req.emr_rc;
565                 goto fail1;
566         }
567         EFSYS_ASSERT3U(req.emr_out_length, ==, MC_CMD_PHY_STATS_OUT_DMA_LEN);
569         siena_phy_decode_stats(enp, vmask, esmp, &smask, stat);
570         EFSYS_ASSERT(smask == encp->enc_phy_stat_mask);
572         return (0);
574 fail1:
575         EFSYS_PROBE1(fail1, int, rc);
577         return (0);
578 }
580 #endif  /* EFSYS_OPT_PHY_STATS */
582 #if EFSYS_OPT_PHY_PROPS
584 #if EFSYS_OPT_NAMES
586 extern          const char __cs *
587 siena_phy_prop_name(
588         __in    efx_nic_t *enp,
589         __in    unsigned int id)
```



```
590 {
591         _NOTE(ARGUNUSED(enp, id))
593         return (NULL);
594 }
596 #endif  /* EFSYS_OPT_NAMES */
598 extern  __checkReturn   int
599 siena_phy_prop_get(
600         __in            efx_nic_t *enp,
601         __in            unsigned int id,
602         __in            uint32_t flags,
603         __out           uint32_t *valp)
604 {
605         _NOTE(ARGUNUSED(enp, id, flags, valp))
607         return (ENOTSUP);
608 }
610 extern  __checkReturn   int
611 siena_phy_prop_set(
612         __in            efx_nic_t *enp,
613         __in            unsigned int id,
614         __in            uint32_t val)
615 {
616         _NOTE(ARGUNUSED(enp, id, val))
618         return (ENOTSUP);
619 }
621 #endif  /* EFSYS_OPT_PHY_PROPS */
623 #if EFSYS_OPT_PHY_BIST
625         __checkReturn           int
626 siena_phy_bist_start(
627         __in                    efx_nic_t *enp,
628         __in                    efx_phy_bist_type_t type)
629 {
630         uint8_t payload[MC_CMD_START_BIST_IN_LEN];
631         efx_mcdi_req_t req;
632         int rc;
634         req.emr_cmd = MC_CMD_START_BIST;
635         req.emr_in_buf = payload;
636         req.emr_in_length = sizeof (payload);
637         EFX_STATIC_ASSERT(MC_CMD_START_BIST_OUT_LEN == 0);
638         req.emr_out_buf = NULL;
639         req.emr_out_length = 0;
641         switch (type) {
642         case EFX_PHY_BIST_TYPE_NORMAL:
643                 MCDI_IN_SET_DWORD(req, START_BIST_IN_TYPE, MC_CMD_PHY_BIST);
644                 break;
645         case EFX_PHY_BIST_TYPE_CABLE_SHORT:
646                 MCDI_IN_SET_DWORD(req, START_BIST_IN_TYPE,
647                     MC_CMD_PHY_BIST_CABLE_SHORT);
648                 break;
649         case EFX_PHY_BIST_TYPE_CABLE_LONG:
650                 MCDI_IN_SET_DWORD(req, START_BIST_IN_TYPE,
651                     MC_CMD_PHY_BIST_CABLE_LONG);
652                 break;
653         default:
654                 EFSYS_ASSERT(0);
655         }
```

```
657         efx_mcdi_execute(enp, &req);

659         if (req.emr_rc != 0) {
660                 rc = req.emr_rc;
661                 goto fail1;
662         }

664         return (0);

666 fail1:
667         EFSYS_PROBE1(fail1, int, rc);

669         return (rc);
670 }

672 static  __checkReturn           unsigned long
673 siena_phy_sft9001_bist_status(
674         __in                    uint16_t code)
675 {
676         switch (code) {
677         case MC_CMD_POLL_BIST_SFT9001_PAIR_BUSY:
678                 return (EFX_PHY_CABLE_STATUS_BUSY);
679         case MC_CMD_POLL_BIST_SFT9001_INTER_PAIR_SHORT:
680                 return (EFX_PHY_CABLE_STATUS_INTERPAIRSHORT);
681         case MC_CMD_POLL_BIST_SFT9001_INTRA_PAIR_SHORT:
682                 return (EFX_PHY_CABLE_STATUS_INTRAPAIRSHORT);
683         case MC_CMD_POLL_BIST_SFT9001_PAIR_OPEN:
684                 return (EFX_PHY_CABLE_STATUS_OPEN);
685         case MC_CMD_POLL_BIST_SFT9001_PAIR_OK:
686                 return (EFX_PHY_CABLE_STATUS_OK);
687         default:
688                 return (EFX_PHY_CABLE_STATUS_INVALID);
689         }
690 }

692         __checkReturn           int
693 siena_phy_bist_poll(
694         __in                    efx_nic_t *enp,
695         __in                    efx_phy_bist_type_t type,
696         __out                   efx_phy_bist_result_t *resultp,
697         __out_opt __drv_when(count > 0, __notnull)
698         uint32_t *value_maskp,
699         __out_ecount_opt(count) __drv_when(count > 0, __notnull)
700         unsigned long *valuesp,
701         __in                    size_t count)
702 {
703         efx_nic_cfg_t *encp = &(enp->en_nic_cfg);
704         uint8_t payload[MCDI_CTL_SDU_LEN_MAX];
705         uint32_t value_mask = 0;
706         efx_mcdi_req_t req;
707         uint32_t result;
708         int rc;

710         req.emr_cmd = MC_CMD_POLL_BIST;
711         _NOTE(CONSTANTCONDITION)
712         EFSYS_ASSERT(MC_CMD_POLL_BIST_IN_LEN == 0);
713         req.emr_in_buf = NULL;
714         req.emr_in_length = 0;
715         req.emr_out_buf = payload;
716         req.emr_out_length = sizeof (payload);

718         efx_mcdi_execute(enp, &req);

720         if (req.emr_rc != 0) {
721                 rc = req.emr_rc;
```



```
722                 goto fail1;
723         }

725         if (req.emr_out_length_used < MC_CMD_POLL_BIST_OUT_RESULT_OFST + 4) {
726                 rc = EMSGSIZE;
727                 goto fail2;
728         }

730         if (count > 0)
731                 (void) memset(valuesp, '\0', count * sizeof (unsigned long));

733         result = MCDI_OUT_DWORD(req, POLL_BIST_OUT_RESULT);

735         /* Extract PHY specific results */
736         if (result == MC_CMD_POLL_BIST_PASSED &&
737             encp->enc_phy_type == EFX_PHY_SFT9001B &&
738             req.emr_out_length_used >= MC_CMD_POLL_BIST_OUT_SFT9001_LEN &&
739             (type == EFX_PHY_BIST_TYPE_CABLE_SHORT ||
740             type == EFX_PHY_BIST_TYPE_CABLE_LONG)) {
741                 uint16_t word;

743                 if (count > EFX_PHY_BIST_CABLE_LENGTH_A) {
744                         if (valuesp != NULL)
745                                 valuesp[EFX_PHY_BIST_CABLE_LENGTH_A] =
746                                     MCDI_OUT_DWORD(req,
747                                     POLL_BIST_OUT_SFT9001_CABLE_LENGTH_A);
748                         value_mask |= (1 << EFX_PHY_BIST_CABLE_LENGTH_A);
749                 }

751                 if (count > EFX_PHY_BIST_CABLE_LENGTH_B) {
752                         if (valuesp != NULL)
753                                 valuesp[EFX_PHY_BIST_CABLE_LENGTH_B] =
754                                     MCDI_OUT_DWORD(req,
755                                     POLL_BIST_OUT_SFT9001_CABLE_LENGTH_B);
756                         value_mask |= (1 << EFX_PHY_BIST_CABLE_LENGTH_B);
757                 }

759                 if (count > EFX_PHY_BIST_CABLE_LENGTH_C) {
760                         if (valuesp != NULL)
761                                 valuesp[EFX_PHY_BIST_CABLE_LENGTH_C] =
762                                     MCDI_OUT_DWORD(req,
763                                     POLL_BIST_OUT_SFT9001_CABLE_LENGTH_C);
764                         value_mask |= (1 << EFX_PHY_BIST_CABLE_LENGTH_C);
765                 }

767                 if (count > EFX_PHY_BIST_CABLE_LENGTH_D) {
768                         if (valuesp != NULL)
769                                 valuesp[EFX_PHY_BIST_CABLE_LENGTH_D] =
770                                     MCDI_OUT_DWORD(req,
771                                     POLL_BIST_OUT_SFT9001_CABLE_LENGTH_D);
772                         value_mask |= (1 << EFX_PHY_BIST_CABLE_LENGTH_D);
773                 }

775                 if (count > EFX_PHY_BIST_CABLE_STATUS_A) {
776                         if (valuesp != NULL) {
777                                 word = MCDI_OUT_WORD(req,
778                                     POLL_BIST_OUT_SFT9001_CABLE_STATUS_A);
779                                 valuesp[EFX_PHY_BIST_CABLE_STATUS_A] =
780                                     siena_phy_sft9001_bist_status(word);
781                         }
782                         value_mask |= (1 << EFX_PHY_BIST_CABLE_STATUS_A);
783                 }

785                 if (count > EFX_PHY_BIST_CABLE_STATUS_B) {
786                         if (valuesp != NULL) {
787                                 word = MCDI_OUT_WORD(req,
```

```
788                                 POLL_BIST_OUT_SFT9001_CABLE_STATUS_B);
789                             valuesp[EFX_PHY_BIST_CABLE_STATUS_B] =
790                                 siena_phy_sft9001_bist_status(word);
791                     }
792                     value_mask |= (1 << EFX_PHY_BIST_CABLE_STATUS_B);
793             }

795             if (count > EFX_PHY_BIST_CABLE_STATUS_C) {
796                     if (valuesp != NULL) {
797                             word = MCDI_OUT_WORD(req,
798                                 POLL_BIST_OUT_SFT9001_CABLE_STATUS_C);
799                             valuesp[EFX_PHY_BIST_CABLE_STATUS_C] =
800                                 siena_phy_sft9001_bist_status(word);
801                     }
802                     value_mask |= (1 << EFX_PHY_BIST_CABLE_STATUS_C);
803             }

805             if (count > EFX_PHY_BIST_CABLE_STATUS_D) {
806                     if (valuesp != NULL) {
807                             word = MCDI_OUT_WORD(req,
808                                 POLL_BIST_OUT_SFT9001_CABLE_STATUS_D);
809                             valuesp[EFX_PHY_BIST_CABLE_STATUS_D] =
810                                 siena_phy_sft9001_bist_status(word);
811                     }
812                     value_mask |= (1 << EFX_PHY_BIST_CABLE_STATUS_D);
813             }

815         } else if (result == MC_CMD_POLL_BIST_FAILED &&
816                    encp->enc_phy_type == EFX_PHY_QLX111V &&
817                    req.emr_out_length >= MC_CMD_POLL_BIST_OUT_MRSFP_LEN &&
818                    count > EFX_PHY_BIST_FAULT_CODE) {
819             if (valuesp != NULL)
820                     valuesp[EFX_PHY_BIST_FAULT_CODE] =
821                         MCDI_OUT_DWORD(req, POLL_BIST_OUT_MRSFP_TEST);
822             value_mask |= 1 << EFX_PHY_BIST_FAULT_CODE;
823         }

825         if (value_maskp != NULL)
826                 *value_maskp = value_mask;

828         EFSYS_ASSERT(resultp != NULL);
829         if (result == MC_CMD_POLL_BIST_RUNNING)
830                 *resultp = EFX_PHY_BIST_RESULT_RUNNING;
831         else if (result == MC_CMD_POLL_BIST_PASSED)
832                 *resultp = EFX_PHY_BIST_RESULT_PASSED;
833         else
834                 *resultp = EFX_PHY_BIST_RESULT_FAILED;

836         return (0);

838 fail2:
839         EFSYS_PROBE(fail2);
840 fail1:
841         EFSYS_PROBE1(fail1, int, rc);

843         return (rc);
844 }

846                         void
847 siena_phy_bist_stop(
848         __in            efx_nic_t *enp,
849         __in            efx_phy_bist_type_t type)
850 {
851         /* There is no way to stop BIST on Siena */
852         _NOTE(ARGUNUSED(enp, type))
853 }
```



```
855 #endif  /* EFSYS_OPT_PHY_BIST */

857 #endif  /* EFSYS_OPT_SIENA */
858 #endif /* ! codereview */
```

****
5240 Thu Aug 22 18:59:30 2013
new/usr/src/uts/common/io/sfxge/siena_sram.c
Merged sfxge driver
****

```
/*
 * Copyright 2009 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */
#include "efsys.h"
#include "efx.h"
#include "efx_impl.h"

#if EFSYS_OPT_SIENA

                        void
siena_sram_init(
        __in            efx_nic_t *enp)
{
        efx_nic_cfg_t *encp = &(enp->en_nic_cfg);
        efx_oword_t oword;
        uint32_t rx_base, tx_base;

        EFSYS_ASSERT3U(enp->en_magic, ==, EFX_NIC_MAGIC);
        EFSYS_ASSERT(enp->en_family == EFX_FAMILY_SIENA);

        rx_base = encp->enc_buftbl_limit;
        tx_base = rx_base + (encp->enc_rxq_limit *
            EFX_RXQ_DC_NDESCS(EFX_RXQ_DC_SIZE));

        /* Initialize the transmit descriptor cache */
        EFX_POPULATE_OWORD_1(oword, FRF_AZ_SRM_TX_DC_BASE_ADR, tx_base);
        EFX_BAR_WRITEO(enp, FR_AZ_SRM_TX_DC_CFG_REG, &oword);

        EFX_POPULATE_OWORD_1(oword, FRF_AZ_TX_DC_SIZE, EFX_TXQ_DC_SIZE);
        EFX_BAR_WRITEO(enp, FR_AZ_TX_DC_CFG_REG, &oword);

        /* Initialize the receive descriptor cache */
        EFX_POPULATE_OWORD_1(oword, FRF_AZ_SRM_RX_DC_BASE_ADR, rx_base);
        EFX_BAR_WRITEO(enp, FR_AZ_SRM_RX_DC_CFG_REG, &oword);

        EFX_POPULATE_OWORD_1(oword, FRF_AZ_RX_DC_SIZE, EFX_RXQ_DC_SIZE);
        EFX_BAR_WRITEO(enp, FR_AZ_RX_DC_CFG_REG, &oword);

        /* Set receive descriptor pre-fetch low water mark */
        EFX_POPULATE_OWORD_1(oword, FRF_AZ_RX_DC_PF_LWM, 56);
```



```
        EFX_BAR_WRITEO(enp, FR_AZ_RX_DC_PF_WM_REG, &oword);

        /* Set the event queue to use for SRAM updates */
        EFX_POPULATE_OWORD_1(oword, FRF_AZ_SRM_UPD_EVQ_ID, 0);
        EFX_BAR_WRITEO(enp, FR_AZ_SRM_UPD_EVQ_REG, &oword);
}

#if EFSYS_OPT_DIAG

        __checkReturn   int
siena_sram_test(
        __in            efx_nic_t *enp,
        __in            efx_sram_pattern_fn_t func)
{
        efx_oword_t oword;
        efx_qword_t qword;
        efx_qword_t verify;
        size_t rows;
        unsigned int wptr;
        unsigned int rptr;
        int rc;

        EFSYS_ASSERT(enp->en_family == EFX_FAMILY_SIENA);

        /* Reconfigure into HALF buffer table mode */
        EFX_POPULATE_OWORD_1(oword, FRF_AZ_BUF_TBL_MODE, 0);
        EFX_BAR_WRITEO(enp, FR_AZ_BUF_TBL_CFG_REG, &oword);

        /*
         * Move the descriptor caches up to the top of SRAM, and test
         * all of SRAM below them. We only miss out one row here.
         */
        rows = SIENA_SRAM_ROWS - 1;
        EFX_POPULATE_OWORD_1(oword, FRF_AZ_SRM_RX_DC_BASE_ADR, rows);
        EFX_BAR_WRITEO(enp, FR_AZ_SRM_RX_DC_CFG_REG, &oword);

        EFX_POPULATE_OWORD_1(oword, FRF_AZ_SRM_TX_DC_BASE_ADR, rows + 1);
        EFX_BAR_WRITEO(enp, FR_AZ_SRM_TX_DC_CFG_REG, &oword);

        /*
         * Write the pattern through BUF_HALF_TBL. Write
         * in 64 entry batches, waiting 1us in between each batch
         * to guarantee not to overflow the SRAM fifo
         */
        for (wptr = 0, rptr = 0; wptr < rows; ++wptr) {
                func(wptr, B_FALSE, &qword);
                EFX_BAR_TBL_WRITEQ(enp, FR_AZ_BUF_HALF_TBL, wptr, &qword);

                if ((wptr - rptr) < 64 && wptr < rows - 1)
                        continue;

                EFSYS_SPIN(1);

                for (; rptr <= wptr; ++rptr) {
                        func(rptr, B_FALSE, &qword);
                        EFX_BAR_TBL_READQ(enp, FR_AZ_BUF_HALF_TBL, rptr,
                            &verify);

                        if (!EFX_QWORD_IS_EQUAL(verify, qword)) {
                                rc = EFAULT;
                                goto fail1;
                        }
                }
        }

        /* And do the same negated */
```

```
128         for (wptr = 0, rptr = 0; wptr < rows; ++wptr) {
129                 func(wptr, B_TRUE, &qword);
130                 EFX_BAR_TBL_WRITEQ(enp, FR_AZ_BUF_HALF_TBL, wptr, &qword);

132                 if ((wptr - rptr) < 64 && wptr < rows - 1)
133                         continue;

135                 EFSYS_SPIN(1);

137                 for (; rptr <= wptr; ++rptr) {
138                         func(rptr, B_TRUE, &qword);
139                         EFX_BAR_TBL_READQ(enp, FR_AZ_BUF_HALF_TBL, rptr,
140                             &verify);

142                         if (!EFX_QWORD_IS_EQUAL(verify, qword)) {
143                                 rc = EFAULT;
144                                 goto fail2;
145                         }
146                 }
147         }

149         /* Restore back to FULL buffer table mode */
150         EFX_POPULATE_OWORD_1(oword, FRF_AZ_BUF_TBL_MODE, 1);
151         EFX_BAR_WRITEO(enp, FR_AZ_BUF_TBL_CFG_REG, &oword);

153         /*
154          * We don't need to reconfigure SRAM again because the API
155          * requires efx_nic_fini() to be called after an sram test.
156          */
157         return (0);

159 fail2:
160         EFSYS_PROBE(fail2);
161 fail1:
162         EFSYS_PROBE1(fail1, int, rc);

164         /* Restore back to FULL buffer table mode */
165         EFX_POPULATE_OWORD_1(oword, FRF_AZ_BUF_TBL_MODE, 1);
166         EFX_BAR_WRITEO(enp, FR_AZ_BUF_TBL_CFG_REG, &oword);

168         return (rc);
169 }

171 #endif  /* EFSYS_OPT_DIAG */

173 #endif  /* EFSYS_OPT_SIENA */
174 #endif /* ! codereview */
```

```
**********************************************************
   14083 Thu Aug 22 18:59:30 2013
new/usr/src/uts/common/io/sfxge/siena_vpd.c
Merged sfxge driver
**********************************************************
```

```
/*
 * Copyright 2009 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ‘‘AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#include "efsys.h"
#include "efx.h"
#include "efx_types.h"
#include "efx_regs.h"
#include "efx_impl.h"

#if EFSYS_OPT_VPD

#if EFSYS_OPT_SIENA

static  __checkReturn                   int
siena_vpd_get_static(
        __in                            efx_nic_t *enp,
        __in                            unsigned int partn,
        __deref_out_bcount_opt(*sizep)  caddr_t *svpdp,
        __out                           size_t *sizep)
{
        siena_mc_static_config_hdr_t *scfg;
        caddr_t svpd;
        size_t size;
        uint8_t cksum;
        unsigned int vpd_offset;
        unsigned int vpd_length;
        unsigned int hdr_length;
        unsigned int pos;
        unsigned int region;
        int rc;

        EFSYS_ASSERT(partn == MC_CMD_NVRAM_TYPE_STATIC_CFG_PORT0 ||
                     partn == MC_CMD_NVRAM_TYPE_STATIC_CFG_PORT1);

        /* Allocate sufficient memory for the entire static cfg area */
        if ((rc = siena_nvram_partn_size(enp, partn, &size)) != 0)
                goto fail1;

        EFSYS_KMEM_ALLOC(enp->en_esip, size, scfg);
```



```
        if (scfg == NULL) {
                rc = ENOMEM;
                goto fail2;
        }

        if ((rc = siena_nvram_partn_read(enp, partn, 0,
            (caddr_t)scfg, SIENA_NVRAM_CHUNK)) != 0)
                goto fail3;

        /* Verify the magic number */
        if (EFX_DWORD_FIELD(scfg->magic, EFX_DWORD_0) !=
            SIENA_MC_STATIC_CONFIG_MAGIC) {
                rc = EINVAL;
                goto fail4;
        }

        /* All future versions of the structure must be backwards compatable */
        EFX_STATIC_ASSERT(SIENA_MC_STATIC_CONFIG_VERSION == 0);

        hdr_length = EFX_WORD_FIELD(scfg->length, EFX_WORD_0);
        vpd_offset = EFX_DWORD_FIELD(scfg->static_vpd_offset, EFX_DWORD_0);
        vpd_length = EFX_DWORD_FIELD(scfg->static_vpd_length, EFX_DWORD_0);

        /* Verify the hdr doesn’t overflow the sector size */
        if (hdr_length > size || vpd_offset > size || vpd_length > size ||
            vpd_length + vpd_offset > size) {
                rc = EINVAL;
                goto fail5;
        }

        /* Read the remainder of scfg + static vpd */
        region = vpd_offset + vpd_length;
        if (region > SIENA_NVRAM_CHUNK) {
                if ((rc = siena_nvram_partn_read(enp, partn, SIENA_NVRAM_CHUNK,
                    (caddr_t)scfg + SIENA_NVRAM_CHUNK,
                    region - SIENA_NVRAM_CHUNK)) != 0)
                        goto fail6;
        }

        /* Verify checksum */
        cksum = 0;
        for (pos = 0; pos < hdr_length; pos++)
                cksum += ((uint8_t *)scfg)[pos];
        if (cksum != 0) {
                rc = EINVAL;
                goto fail7;
        }

        if (vpd_length == 0)
                svpd = NULL;
        else {
                /* Copy the vpd data out */
                EFSYS_KMEM_ALLOC(enp->en_esip, vpd_length, svpd);
                if (svpd == NULL) {
                        rc = ENOMEM;
                        goto fail8;
                }
                memcpy(svpd, (caddr_t)scfg + vpd_offset, vpd_length);
        }

        EFSYS_KMEM_FREE(enp->en_esip, size, scfg);

        *svpdp = svpd;
        *sizep = vpd_length;

        return (0);
```

```
129 fail8:
130         EFSYS_PROBE(fail8);
131 fail7:
132         EFSYS_PROBE(fail7);
133 fail6:
134         EFSYS_PROBE(fail6);
135 fail5:
136         EFSYS_PROBE(fail5);
137 fail4:
138         EFSYS_PROBE(fail4);
139 fail3:
140         EFSYS_PROBE(fail3);
141 fail2:
142         EFSYS_PROBE(fail2);

144         EFSYS_KMEM_FREE(enp->en_esip, size, scfg);

146 fail1:
147         EFSYS_PROBE1(fail1, int, rc);

149         return (rc);
150 }

152         __checkReturn           int
153 siena_vpd_init(
154         __in                    efx_nic_t *enp)
155 {
156         efx_mcdi_iface_t *emip = &(enp->en_u.siena.enu_mip);
157         caddr_t svpd = NULL;
158         unsigned partn;
159         size_t size = 0;
160         int rc;

162         EFSYS_ASSERT(enp->en_family == EFX_FAMILY_SIENA);

164         partn = (emip->emi_port == 1)
165                 ? MC_CMD_NVRAM_TYPE_STATIC_CFG_PORT0
166                 : MC_CMD_NVRAM_TYPE_STATIC_CFG_PORT1;

168         /*
169          * We need the static VPD sector to present a unified static+dynamic
170          * VPD, that is, basically on every read, write, verify cycle. Since
171          * it should *never* change we can just cache it here.
172          */
173         if ((rc = siena_vpd_get_static(enp, partn, &svpd, &size)) != 0)
174                 goto fail1;

176         if (svpd != NULL && size > 0) {
177                 if ((rc = efx_vpd_hunk_verify(svpd, size, NULL)) != 0)
178                         goto fail2;
179         }

181         enp->en_u.siena.enu_svpd = svpd;
182         enp->en_u.siena.enu_svpd_length = size;

184         return (0);

186 fail2:
187         EFSYS_PROBE(fail2);

189         EFSYS_KMEM_FREE(enp->en_esip, size, svpd);
190 fail1:
191         EFSYS_PROBE1(fail1, int, rc);

193         return (rc);
```



```
194 }

196         __checkReturn           int
197 siena_vpd_size(
198         __in                    efx_nic_t *enp,
199         __out                   size_t *sizep)
200 {
201         efx_mcdi_iface_t *emip = &(enp->en_u.siena.enu_mip);
202         unsigned int partn;
203         int rc;

205         EFSYS_ASSERT(enp->en_family == EFX_FAMILY_SIENA);

207         /*
208          * This function returns the total size the user should allocate
209          * for all VPD operations. We've already cached the static vpd,
210          * so we just need to return an upper bound on the dynamic vpd.
211          * Since the dynamic_config structure can change under our feet,
212          * (as version numbers are inserted), just be safe and return the
213          * total size of the dynamic_config *sector*
214          */
215         partn = (emip->emi_port == 1)
216                 ? MC_CMD_NVRAM_TYPE_DYNAMIC_CFG_PORT0
217                 : MC_CMD_NVRAM_TYPE_DYNAMIC_CFG_PORT1;

219         if ((rc = siena_nvram_partn_size(enp, partn, sizep)) != 0)
220                 goto fail1;

222         return (0);

224 fail1:
225         EFSYS_PROBE1(fail1, int, rc);

227         return (rc);
228 }

230         __checkReturn           int
231 siena_vpd_read(
232         __in                    efx_nic_t *enp,
233         __out_bcount(size)      caddr_t data,
234         __in                    size_t size)
235 {
236         efx_mcdi_iface_t *emip = &(enp->en_u.siena.enu_mip);
237         siena_mc_dynamic_config_hdr_t *dcfg;
238         unsigned int vpd_length;
239         unsigned int vpd_offset;
240         unsigned int dcfg_partn;
241         size_t dcfg_size;
242         int rc;

244         EFSYS_ASSERT(enp->en_family == EFX_FAMILY_SIENA);

246         dcfg_partn = (emip->emi_port == 1)
247                 ? MC_CMD_NVRAM_TYPE_DYNAMIC_CFG_PORT0
248                 : MC_CMD_NVRAM_TYPE_DYNAMIC_CFG_PORT1;

250         if ((rc = siena_nvram_get_dynamic_cfg(enp, dcfg_partn,
251             B_TRUE, &dcfg, &dcfg_size)) != 0)
252                 goto fail1;

254         vpd_length = EFX_DWORD_FIELD(dcfg->dynamic_vpd_length, EFX_DWORD_0);
255         vpd_offset = EFX_DWORD_FIELD(dcfg->dynamic_vpd_offset, EFX_DWORD_0);

257         if (vpd_length > size) {
258                 rc = EFAULT;    /* Invalid dcfg: header bigger than sector */
259                 goto fail2;
```

```
260         }
262         EFSYS_ASSERT3U(vpd_length, <=, size);
263         memcpy(data, (caddr_t)dcfg + vpd_offset, vpd_length);
265         /* Pad data with all-1s, consistent with update operations */
266         memset(data + vpd_length, 0xff, size - vpd_length);
268         EFSYS_KMEM_FREE(enp->en_esip, dcfg_size, dcfg);
270         return (0);
272 fail2:
273         EFSYS_PROBE(fail2);
275         EFSYS_KMEM_FREE(enp->en_esip, dcfg_size, dcfg);
276 fail1:
277         EFSYS_PROBE1(fail1, int, rc);
279         return (rc);
280 }
282         __checkReturn           int
283 siena_vpd_verify(
284         __in                    efx_nic_t *enp,
285         __in_bcount(size)       caddr_t data,
286         __in                    size_t size)
287 {
288         efx_vpd_tag_t stag;
289         efx_vpd_tag_t dtag;
290         efx_vpd_keyword_t skey;
291         efx_vpd_keyword_t dkey;
292         unsigned int scont;
293         unsigned int dcont;
295         int rc;
297         EFSYS_ASSERT(enp->en_family == EFX_FAMILY_SIENA);
299         /*
300          * Strictly you could take the view that dynamic vpd is optional.
301          * Instead, to conform more closely to the read/verify/reinit()
302          * paradigm, we require dynamic vpd. siena_vpd_reinit() will
303          * reinitialize it as required.
304          */
305         if ((rc = efx_vpd_hunk_verify(data, size, NULL)) != 0)
306                 goto fail1;
308         /*
309          * Verify that there is no duplication between the static and
310          * dynamic cfg sectors.
311          */
312         if (enp->en_u.siena.enu_svpd_length == 0)
313                 goto done;
315         dcont = 0;
316         _NOTE(CONSTANTCONDITION)
317         while (1) {
318                 if ((rc = efx_vpd_hunk_next(data, size, &dtag,
319                     &dkey, NULL, NULL, &dcont)) != 0)
320                         goto fail2;
321                 if (dcont == 0)
322                         break;
324                 scont = 0;
325                 _NOTE(CONSTANTCONDITION)
```



```
326                 while (1) {
327                         if ((rc = efx_vpd_hunk_next(
328                             enp->en_u.siena.enu_svpd,
329                             enp->en_u.siena.enu_svpd_length, &stag, &skey,
330                             NULL, NULL, &scont)) != 0)
331                                 goto fail3;
332                         if (scont == 0)
333                                 break;
335                         if (stag == dtag && skey == dkey) {
336                                 rc = EEXIST;
337                                 goto fail4;
338                         }
339                 }
340         }
342 done:
343         return (0);
345 fail4:
346         EFSYS_PROBE(fail4);
347 fail3:
348         EFSYS_PROBE(fail3);
349 fail2:
350         EFSYS_PROBE(fail2);
351 fail1:
352         EFSYS_PROBE1(fail1, int, rc);
354         return (rc);
355 }
357         __checkReturn           int
358 siena_vpd_reinit(
359         __in                    efx_nic_t *enp,
360         __in_bcount(size)       caddr_t data,
361         __in                    size_t size)
362 {
363         boolean_t wantpid;
364         int rc;
366         /*
367          * Only create a PID if the dynamic cfg doesn't have one
368          */
369         if (enp->en_u.siena.enu_svpd_length == 0)
370                 wantpid = B_TRUE;
371         else {
372                 unsigned int offset;
373                 uint8_t length;
375                 rc = efx_vpd_hunk_get(enp->en_u.siena.enu_svpd,
376                                       enp->en_u.siena.enu_svpd_length,
377                                       EFX_VPD_ID, 0, &offset, &length);
378                 if (rc == 0)
379                         wantpid = B_FALSE;
380                 else if (rc == ENOENT)
381                         wantpid = B_TRUE;
382                 else
383                         goto fail1;
384         }
386         if ((rc = efx_vpd_hunk_reinit(data, size, wantpid)) != 0)
387                 goto fail2;
389         return (0);
391 fail2:
```

```
392         EFSYS_PROBE(fail2);
393 fail1:
394         EFSYS_PROBE1(fail1, int, rc);

396         return (rc);
397 }

399         __checkReturn           int
400 siena_vpd_get(
401         __in                    efx_nic_t *enp,
402         __in_bcount(size)       caddr_t data,
403         __in                    size_t size,
404         __inout                 efx_vpd_value_t *evvp)
405 {
406         unsigned int offset;
407         uint8_t length;
408         int rc;

410         EFSYS_ASSERT(enp->en_family == EFX_FAMILY_SIENA);

412         /* Attempt to satisfy the request from svpd first */
413         if (enp->en_u.siena.enu_svpd_length > 0) {
414                 if ((rc = efx_vpd_hunk_get(enp->en_u.siena.enu_svpd,
415                     enp->en_u.siena.enu_svpd_length, evvp->evv_tag,
416                     evvp->evv_keyword, &offset, &length)) == 0) {
417                         evvp->evv_length = length;
418                         memcpy(evvp->evv_value,
419                             enp->en_u.siena.enu_svpd + offset, length);
420                         return (0);
421                 } else if (rc != ENOENT)
422                         goto fail1;
423         }

425         /* And then from the provided data buffer */
426         if ((rc = efx_vpd_hunk_get(data, size, evvp->evv_tag,
427             evvp->evv_keyword, &offset, &length)) != 0)
428                 goto fail2;

430         evvp->evv_length = length;
431         memcpy(evvp->evv_value, data + offset, length);

433         return (0);

435 fail2:
436         EFSYS_PROBE(fail2);
437 fail1:
438         EFSYS_PROBE1(fail1, int, rc);

440         return (rc);
441 }

443         __checkReturn           int
444 siena_vpd_set(
445         __in                    efx_nic_t *enp,
446         __in_bcount(size)       caddr_t data,
447         __in                    size_t size,
448         __in                    efx_vpd_value_t *evvp)
449 {
450         int rc;

452         EFSYS_ASSERT(enp->en_family == EFX_FAMILY_SIENA);

454         /* If the provided (tag,keyword) exists in svpd, then it is readonly */
455         if (enp->en_u.siena.enu_svpd_length > 0) {
456                 unsigned int offset;
457                 uint8_t length;
```



```
459                 if ((rc = efx_vpd_hunk_get(enp->en_u.siena.enu_svpd,
460                     enp->en_u.siena.enu_svpd_length, evvp->evv_tag,
461                     evvp->evv_keyword, &offset, &length)) == 0) {
462                         rc = EACCES;
463                         goto fail1;
464                 }
465         }

467         if ((rc = efx_vpd_hunk_set(data, size, evvp)) != 0)
468                 goto fail2;

470         return (0);

472 fail2:
473         EFSYS_PROBE(fail2);
474 fail1:
475         EFSYS_PROBE1(fail1, int, rc);

477         return (rc);
478 }

480         __checkReturn           int
481 siena_vpd_next(
482         __in                    efx_nic_t *enp,
483         __in_bcount(size)       caddr_t data,
484         __in                    size_t size,
485         __out                   efx_vpd_value_t *evvp,
486         __inout                 unsigned int *contp)
487 {
488         _NOTE(ARGUNUSED(enp, data, size, evvp, contp))

490         return (ENOTSUP);
491 }

493         __checkReturn           int
494 siena_vpd_write(
495         __in                    efx_nic_t *enp,
496         __in_bcount(size)       caddr_t data,
497         __in                    size_t size)
498 {
499         efx_mcdi_iface_t *emip = &(enp->en_u.siena.enu_mip);
500         siena_mc_dynamic_config_hdr_t *dcfg;
501         unsigned int vpd_offset;
502         unsigned int dcfg_partn;
503         unsigned int hdr_length;
504         unsigned int pos;
505         uint8_t cksum;
506         size_t partn_size, dcfg_size;
507         size_t vpd_length;
508         int rc;

510         EFSYS_ASSERT(enp->en_family == EFX_FAMILY_SIENA);

512         /* Determine total length of all tags */
513         if ((rc = efx_vpd_hunk_length(data, size, &vpd_length)) != 0)
514                 goto fail1;

516         /* Lock dynamic config sector for write, and read structure only */
517         dcfg_partn = (emip->emi_port == 1)
518                 ? MC_CMD_NVRAM_TYPE_DYNAMIC_CFG_PORT0
519                 : MC_CMD_NVRAM_TYPE_DYNAMIC_CFG_PORT1;

521         if ((rc = siena_nvram_partn_size(enp, dcfg_partn, &partn_size)) != 0)
522                 goto fail2;
```

```
524         if ((rc = siena_nvram_partn_lock(enp, dcfg_partn)) != 0)
525                 goto fail2;

527         if ((rc = siena_nvram_get_dynamic_cfg(enp, dcfg_partn,
528             B_FALSE, &dcfg, &dcfg_size)) != 0)
529                 goto fail3;

531         hdr_length = EFX_WORD_FIELD(dcfg->length, EFX_WORD_0);

533         /* Allocated memory should have room for the new VPD */
534         if (hdr_length + vpd_length > dcfg_size) {
535                 rc = ENOSPC;
536                 goto fail3;
537         }

539         /* Copy in new vpd and update header */
540         vpd_offset = dcfg_size - vpd_length;
541         EFX_POPULATE_DWORD_1(dcfg->dynamic_vpd_offset, EFX_DWORD_0, vpd_offset);
542         memcpy((caddr_t)dcfg + vpd_offset, data, vpd_length);
543         EFX_POPULATE_DWORD_1(dcfg->dynamic_vpd_length,
544                              EFX_DWORD_0, vpd_length);

546         /* Update the checksum */
547         cksum = 0;
548         for (pos = 0; pos < hdr_length; pos++)
549                 cksum += ((uint8_t *)dcfg)[pos];
550         dcfg->csum.eb_u8[0] -= cksum;

552         /* Erase and write the new sector */
553         if ((rc = siena_nvram_partn_erase(enp, dcfg_partn, 0, partn_size)) != 0)
554                 goto fail4;

556         /* Write out the new structure to nvram */
557         if ((rc = siena_nvram_partn_write(enp, dcfg_partn, 0, (caddr_t)dcfg,
558             vpd_offset + vpd_length)) != 0)
559                 goto fail5;

561         EFSYS_KMEM_FREE(enp->en_esip, dcfg_size, dcfg);

563         siena_nvram_partn_unlock(enp, dcfg_partn);

565         return (0);

567 fail5:
568         EFSYS_PROBE(fail5);
569 fail4:
570         EFSYS_PROBE(fail4);
571 fail3:
572         EFSYS_PROBE(fail3);

574         EFSYS_KMEM_FREE(enp->en_esip, dcfg_size, dcfg);
575 fail2:
576         EFSYS_PROBE(fail2);

578         siena_nvram_partn_unlock(enp, dcfg_partn);
579 fail1:
580         EFSYS_PROBE1(fail1, int, rc);

582         return (rc);
583 }

585                                 void
586 siena_vpd_fini(
587         __in                    efx_nic_t *enp)
588 {
589         EFSYS_ASSERT(enp->en_family == EFX_FAMILY_SIENA);
```



```
591         if (enp->en_u.siena.enu_svpd_length > 0) {
592                 EFSYS_KMEM_FREE(enp->en_esip, enp->en_u.siena.enu_svpd_length,
593                                 enp->en_u.siena.enu_svpd);

595                 enp->en_u.siena.enu_svpd = NULL;
596                 enp->en_u.siena.enu_svpd_length = 0;
597         }
598 }

600 #endif  /* EFSYS_OPT_SIENA */

602 #endif  /* EFSYS_OPT_VPD */
603 #endif /* ! codereview */
```

```
**********************************************************
   21782 Thu Aug 22 18:59:30 2013
new/usr/src/uts/common/io/sfxge/txc43128.c
Merged sfxge driver
**********************************************************
```

```
/*
 * Copyright 2007-2013 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#include "efsys.h"
#include "efx.h"
#include "efx_types.h"
#include "efx_regs.h"
#include "efx_impl.h"
#include "txc43128.h"
#include "txc43128_impl.h"
#include "xphy.h"

#if EFSYS_OPT_PHY_TXC43128

static  __checkReturn   int
txc43128_bist_run(
        __in            efx_nic_t *enp)
{
        efx_port_t *epp = &(enp->en_port);
        efx_word_t word;
        unsigned int count;
        int rc;

        /* Set the BIST type */
        EFX_POPULATE_WORD_1(word, BTYPE, TSDET_DECODE);
        if ((rc = falcon_mdio_write(enp, epp->ep_port, PCS_MMD, BCTL,
            &word)) != 0)
                goto fail1;

        /* Enable BIST */
        EFX_SET_WORD_FIELD(word, BSTEN, 1);
        if ((rc = falcon_mdio_write(enp, epp->ep_port, PCS_MMD, BCTL,
            &word)) != 0)
                goto fail2;

        /* Start BIST */
        EFX_SET_WORD_FIELD(word, BSTRT, 1);
        if ((rc = falcon_mdio_write(enp, epp->ep_port, PCS_MMD, BCTL,
            &word)) != 0)
```



```
                goto fail3;

        EFX_SET_WORD_FIELD(word, BSTRT, 0);

        /* Spin for 100 us */
        EFSYS_SPIN(100);

        /* Stop BIST */
        EFX_SET_WORD_FIELD(word, BSTOP, 1);
        if ((rc = falcon_mdio_write(enp, epp->ep_port, PCS_MMD, BCTL,
            &word)) != 0)
                goto fail4;

        /* Wait until BIST has stopped */
        count = 0;
        do {
                EFSYS_PROBE1(wait, unsigned int, count);

                /* Spin for 10 us */
                EFSYS_SPIN(10);

                /* Check whether the reset bit has cleared */
                if ((rc = falcon_mdio_read(enp, epp->ep_port, PCS_MMD,
                    BCTL, &word)) != 0)
                        goto fail5;

                if (EFX_WORD_FIELD(word, BSTOP) == 0)
                        goto done;

        } while (++count < 20);

        rc = ETIMEDOUT;
        goto fail6;

done:
        /* Disable BIST */
        EFX_ZERO_WORD(word);
        if ((rc = falcon_mdio_write(enp, epp->ep_port, PCS_MMD, BCTL,
            &word)) != 0)
                goto fail7;

        return (0);

fail7:
        EFSYS_PROBE(fail7);
fail6:
        EFSYS_PROBE(fail6);
fail5:
        EFSYS_PROBE(fail5);
fail4:
        EFSYS_PROBE(fail4);
fail3:
        EFSYS_PROBE(fail3);
fail2:
        EFSYS_PROBE(fail2);
fail1:
        EFSYS_PROBE1(fail1, int, rc);

        return (rc);
}

static  __checkReturn   int
txc43128_bist_check(
        __in            efx_nic_t *enp)
{
        efx_port_t *epp = &(enp->en_port);
```

```
128         efx_word_t word;
129         int rc;
130
131         /* Check lane 0 frame count */
132         if ((rc = falcon_mdio_read(enp, epp->ep_port, PCS_MMD, BRX0FRMCNT,
133             &word)) != 0)
134                 goto fail1;
135
136         if (EFX_WORD_FIELD(word, EFX_WORD_0) == 0)
137                 goto fail2;
138
139         /* Check lane 0 error count */
140         if ((rc = falcon_mdio_read(enp, epp->ep_port, PCS_MMD, BRX0ERRCNT,
141             &word)) != 0)
142                 goto fail3;
143
144         if (EFX_WORD_FIELD(word, EFX_WORD_0) != 0)
145                 goto fail4;
146
147         /* Check lane 1 frame count */
148         if ((rc = falcon_mdio_read(enp, epp->ep_port, PCS_MMD, BRX1FRMCNT,
149             &word)) != 0)
150                 goto fail5;
151
152         if (EFX_WORD_FIELD(word, EFX_WORD_0) == 0)
153                 goto fail6;
154
155         /* Check lane 1 error count */
156         if ((rc = falcon_mdio_read(enp, epp->ep_port, PCS_MMD, BRX1ERRCNT,
157             &word)) != 0)
158                 goto fail7;
159
160         if (EFX_WORD_FIELD(word, EFX_WORD_0) != 0)
161                 goto fail8;
162
163         /* Check lane 2 frame count */
164         if ((rc = falcon_mdio_read(enp, epp->ep_port, PCS_MMD, BRX2FRMCNT,
165             &word)) != 0)
166                 goto fail9;
167
168         if (EFX_WORD_FIELD(word, EFX_WORD_0) == 0)
169                 goto fail10;
170
171         /* Check lane 2 error count */
172         if ((rc = falcon_mdio_read(enp, epp->ep_port, PCS_MMD, BRX2ERRCNT,
173             &word)) != 0)
174                 goto fail11;
175
176         if (EFX_WORD_FIELD(word, EFX_WORD_0) != 0)
177                 goto fail12;
178
179         /* Check lane 3 frame count */
180         if ((rc = falcon_mdio_read(enp, epp->ep_port, PCS_MMD, BRX3FRMCNT,
181             &word)) != 0)
182                 goto fail13;
183
184         if (EFX_WORD_FIELD(word, EFX_WORD_0) == 0)
185                 goto fail14;
186
187         /* Check lane 3 error count */
188         if ((rc = falcon_mdio_read(enp, epp->ep_port, PCS_MMD, BRX3ERRCNT,
189             &word)) != 0)
190                 goto fail15;
191
192         if (EFX_WORD_FIELD(word, EFX_WORD_0) != 0)
193                 goto fail16;
```



```
194
195         return (0);
196
197 fail16:
198         EFSYS_PROBE(fail16);
199 fail15:
200         EFSYS_PROBE(fail15);
201 fail14:
202         EFSYS_PROBE(fail14);
203 fail13:
204         EFSYS_PROBE(fail13);
205 fail12:
206         EFSYS_PROBE(fail12);
207 fail11:
208         EFSYS_PROBE(fail11);
209 fail10:
210         EFSYS_PROBE(fail10);
211 fail9:
212         EFSYS_PROBE(fail9);
213 fail8:
214         EFSYS_PROBE(fail8);
215 fail7:
216         EFSYS_PROBE(fail7);
217 fail6:
218         EFSYS_PROBE(fail6);
219 fail5:
220         EFSYS_PROBE(fail5);
221 fail4:
222         EFSYS_PROBE(fail4);
223 fail3:
224         EFSYS_PROBE(fail3);
225 fail2:
226         EFSYS_PROBE(fail2);
227 fail1:
228         EFSYS_PROBE1(fail1, int, rc);
229
230         return (rc);
231 }
232
233 static  __checkReturn   int
234 txc43128_bist(
235         __in            efx_nic_t *enp)
236 {
237         efx_port_t *epp = &(enp->en_port);
238         efx_word_t word;
239         int rc;
240
241         /* Set PMA into loopback */
242         if ((rc = falcon_mdio_read(enp, epp->ep_port, PCS_MMD, MNDBLCFG,
243             &word)) != 0)
244                 goto fail1;
245
246         EFX_SET_WORD_FIELD(word, LNALPBK, 1);
247
248         if ((rc = falcon_mdio_write(enp, epp->ep_port, PCS_MMD, MNDBLCFG,
249             &word)) != 0)
250                 goto fail2;
251
252         /* Run BIST */
253         if ((rc = txc43128_bist_run(enp)) != 0)
254                 goto fail3;
255
256         /* Check BIST */
257         if ((rc = txc43128_bist_check(enp)) != 0)
258                 goto fail4;
```

```
260         /* Turn off loopback */
261         EFX_SET_WORD_FIELD(word, LNALPBK, 0);

263         if ((rc = falcon_mdio_write(enp, epp->ep_port, PCS_MMD, MNDBLCFG,
264             &word)) != 0)
265                 goto fail5;

267         return (0);

269 fail5:
270         EFSYS_PROBE(fail5);
271 fail4:
272         EFSYS_PROBE(fail4);
273 fail3:
274         EFSYS_PROBE(fail3);
275 fail2:
276         EFSYS_PROBE(fail2);
277 fail1:
278         EFSYS_PROBE1(fail1, int, rc);

280         return (rc);
281 }

283 static  __checkReturn   int
284 txc43128_led_cfg(
285         __in            efx_nic_t *enp)
286 {
287         efx_port_t *epp = &(enp->en_port);
288         efx_word_t word;
289         int rc;

291         /* PIO11 allows us to control the red LED */

293         EFX_POPULATE_WORD_1(word, PIO11FNC, PIOFNC_GPIO_DECODE);

295         if ((rc = falcon_mdio_write(enp, epp->ep_port, PHY_XS_MMD,
296             PIOCFG, &word)) != 0)
297                 goto fail1;

299         EFX_POPULATE_WORD_1(word, PIO11DIR, PIODIR_OUT_DECODE);

301         if ((rc = falcon_mdio_write(enp, epp->ep_port, PHY_XS_MMD,
302             PIODIR, &word)) != 0)
303                 goto fail2;


306 #if EFSYS_OPT_PHY_LED_CONTROL

308         switch (epp->ep_phy_led_mode) {
309         case EFX_PHY_LED_DEFAULT:
310         case EFX_PHY_LED_OFF:
311                 EFX_POPULATE_WORD_1(word, PIO11OUT, 0);
312                 break;

314         case EFX_PHY_LED_ON:
315                 EFX_POPULATE_WORD_1(word, PIO11OUT, 1);
316                 break;

318         default:
319                 EFSYS_ASSERT(B_FALSE);
320                 break;
321         }

323 #else   /* EFSYS_OPT_PHY_LED_CONTROL */

325         EFX_POPULATE_WORD_1(word, PIO11OUT, 0);
```



```
327 #endif  /* EFSYS_OPT_PHY_LED_CONTROL */

329         if ((rc = falcon_mdio_write(enp, epp->ep_port, PHY_XS_MMD,
330             PIODO, &word)) != 0)
331                 goto fail3;

333         return (0);

335 fail3:
336         EFSYS_PROBE(fail3);
337 fail2:
338         EFSYS_PROBE(fail2);
339 fail1:
340         EFSYS_PROBE1(fail1, int, rc);

342         return (rc);
343 }

345 static  __checkReturn   int
346 txc43128_preemphasis_cfg(
347         __in            efx_nic_t *enp)
348 {
349         efx_port_t *epp = &(enp->en_port);
350         efx_word_t word;
351         int rc;

353         /* XAUI settings */
354         EFX_POPULATE_WORD_2(word, TXPRE02, 0, TXPRE13, 0);

356         if ((rc = falcon_mdio_write(enp, epp->ep_port, PHY_XS_MMD,
357             ATXPRE0, &word)) != 0)
358                 goto fail1;

360         if ((rc = falcon_mdio_write(enp, epp->ep_port, PHY_XS_MMD,
361             ATXPRE1, &word)) != 0)
362                 goto fail2;

364         /* Line settings */
365         EFX_POPULATE_WORD_2(word, TXPRE02, 2, TXPRE13, 2);

367         if ((rc = falcon_mdio_write(enp, epp->ep_port, PMA_PMD_MMD,
368             ATXPRE0, &word)) != 0)
369                 goto fail3;

371         if ((rc = falcon_mdio_write(enp, epp->ep_port, PMA_PMD_MMD,
372             ATXPRE1, &word)) != 0)
373                 goto fail4;

375         return (0);

377 fail4:
378         EFSYS_PROBE(fail4);
379 fail3:
380         EFSYS_PROBE(fail3);
381 fail2:
382         EFSYS_PROBE(fail2);
383 fail1:
384         EFSYS_PROBE1(fail1, int, rc);

386         return (rc);
387 }

389 static  __checkReturn   int
390 txc43128_amplitude_cfg(
391         __in            efx_nic_t *enp)
```

```
{
        efx_port_t *epp = &(enp->en_port);
        efx_word_t word;
        int rc;

        /* XAUI settings */
        EFX_POPULATE_WORD_2(word, TXAMP02, 25, TXAMP13, 25);

        if ((rc = falcon_mdio_write(enp, epp->ep_port, PHY_XS_MMD,
            ATXAMP0, &word)) != 0)
                goto fail1;

        if ((rc = falcon_mdio_write(enp, epp->ep_port, PHY_XS_MMD,
            ATXAMP1, &word)) != 0)
                goto fail2;

        /* Line settings */
        EFX_POPULATE_WORD_2(word, TXAMP02, 12, TXAMP13, 12);

        if ((rc = falcon_mdio_write(enp, epp->ep_port, PMA_PMD_MMD,
            ATXAMP0, &word)) != 0)
                goto fail3;

        if ((rc = falcon_mdio_write(enp, epp->ep_port, PMA_PMD_MMD,
            ATXAMP1, &word)) != 0)
                goto fail4;

        return (0);

fail4:
        EFSYS_PROBE(fail4);
fail3:
        EFSYS_PROBE(fail3);
fail2:
        EFSYS_PROBE(fail2);
fail1:
        EFSYS_PROBE1(fail1, int, rc);

        return (rc);
}

#if EFSYS_OPT_LOOPBACK
static  __checkReturn   int
txc43128_loopback_cfg(
        __in            efx_nic_t *enp)
{
        efx_port_t *epp = &(enp->en_port);
        efx_word_t word;
        int rc;

        switch (epp->ep_loopback_type) {
        case EFX_LOOPBACK_PHY_XS:
                if ((rc = falcon_mdio_read(enp, epp->ep_port, PHY_XS_MMD,
                    MNDBLCFG, &word)) != 0)
                        goto fail1;

                EFX_SET_WORD_FIELD(word, PXS8BLPBK, 1);

                if ((rc = falcon_mdio_write(enp, epp->ep_port, PHY_XS_MMD,
                    MNDBLCFG, &word)) != 0)
                        goto fail2;

                break;

        case EFX_LOOPBACK_PCS:
                if ((rc = xphy_mmd_loopback_set(enp, epp->ep_port, PCS_MMD,
```



```
                    B_TRUE)) != 0)
                        goto fail1;

                break;

        case EFX_LOOPBACK_PMA_PMD:
                if ((rc = xphy_mmd_loopback_set(enp, epp->ep_port, PMA_PMD_MMD,
                    B_TRUE)) != 0)
                        goto fail1;

                break;

        default:
                break;
        }

        return (0);

fail2:
        EFSYS_PROBE(fail2);
fail1:
        EFSYS_PROBE1(fail1, int, rc);

        return (rc);
}
#endif  /* EFSYS_PHY_LOOPBACK */

static  __checkReturn   int
txc43128_logic_reset(
        __in            efx_nic_t *enp)
{
        efx_port_t *epp = &(enp->en_port);
        efx_word_t word;
        unsigned int count;
        int rc;

        EFSYS_PROBE(logic_reset);

        /* Set the reset bit in the global command register */
        if ((rc = falcon_mdio_read(enp, epp->ep_port, PCS_MMD, GLCMD,
            &word)) != 0)
                goto fail1;

        EFX_SET_WORD_FIELD(word, LMTSWRST, 1);

        if ((rc = falcon_mdio_write(enp, epp->ep_port, PCS_MMD, GLCMD,
            &word)) != 0)
                goto fail2;

        count = 0;
        do {
                EFSYS_PROBE1(wait, unsigned int, count);

                /* Spin for 10 us */
                EFSYS_SPIN(10);

                /* Check whether the reset bit has cleared */
                if ((rc = falcon_mdio_read(enp, epp->ep_port, PCS_MMD,
                    GLCMD, &word)) != 0)
                        goto fail3;

                if (EFX_WORD_FIELD(word, LMTSWRST) == 0)
                        goto done;

        } while (++count < 1000);
```

```
524         rc = ETIMEDOUT;
525         goto fail4;

527 done:
528         return (0);

530 fail4:
531         EFSYS_PROBE(fail4);
532 fail3:
533         EFSYS_PROBE(fail3);
534 fail2:
535         EFSYS_PROBE(fail2);
536 fail1:
537         EFSYS_PROBE1(fail1, int, rc);

539         return (rc);
540 }

542         __checkReturn   int
543 txc43128_reset(
544         __in            efx_nic_t *enp)
545 {
546         efx_port_t *epp = &(enp->en_port);
547         efx_word_t word;
548         unsigned int count;
549         int rc;

551         /* Set the reset bit in the main control register */
552         if ((rc = falcon_mdio_read(enp, epp->ep_port, PHY_XS_MMD, MNCTL,
553             &word)) != 0)
554                 goto fail1;

556         EFX_SET_WORD_FIELD(word, MRST, 1);

558         if ((rc = falcon_mdio_write(enp, epp->ep_port, PHY_XS_MMD, MNCTL,
559             &word)) != 0)
560                 goto fail2;

562         count = 0;
563         do {
564                 EFSYS_PROBE1(wait, unsigned int, count);

566                 /* Spin for 10 us */
567                 EFSYS_SPIN(10);

569                 /* Check whether the reset bit has cleared */
570                 if ((rc = falcon_mdio_read(enp, epp->ep_port, PHY_XS_MMD,
571                     MNCTL, &word)) != 0)
572                         goto fail3;

574                 if (EFX_WORD_FIELD(word, MRST) == 0)
575                         goto done;

577         } while (++count < 1000);

579         rc = ETIMEDOUT;
580         goto fail4;

582 done:
583         /*
584          * Despite the global reset having completed, we will still be unable
585          * to get sensible values out of the 802.3ae registers for some
586          * unknown time. 250 ms seems to cover it but this is empirically
587          * determined.
588          */
589         EFSYS_SLEEP(250000);
```



```
591         enp->en_reset_flags |= EFX_RESET_PHY;

593         return (0);

595 fail4:
596         EFSYS_PROBE(fail4);
597 fail3:
598         EFSYS_PROBE(fail3);
599 fail2:
600         EFSYS_PROBE(fail2);
601 fail1:
602         EFSYS_PROBE1(fail1, int, rc);

604         return (rc);
605 }

607         __checkReturn   int
608 txc43128_reconfigure(
609         __in            efx_nic_t *enp)
610 {
611         efx_port_t *epp = &(enp->en_port);
612         int rc;

614         if ((rc = xphy_pkg_wait(enp, epp->ep_port, TXC43128_MMD_MASK)) != 0)
615                 goto fail1;

617         if ((rc = txc43128_bist(enp)) != 0)
618                 goto fail2;

620         if ((rc = txc43128_led_cfg(enp)) != 0)
621                 goto fail3;

623         if ((rc = txc43128_preemphasis_cfg(enp)) != 0)
624                 goto fail4;

626         if ((rc = txc43128_amplitude_cfg(enp)) != 0)
627                 goto fail5;

629         EFSYS_ASSERT3U(epp->ep_adv_cap_mask, ==, TXC43128_ADV_CAP_MASK);

631 #if EFSYS_OPT_LOOPBACK
632         if ((rc = txc43128_loopback_cfg(enp)) != 0)
633                 goto fail6;
634 #endif  /* EFSYS_OPT_LOOPBACK */

636         if ((rc = txc43128_logic_reset(enp)) != 0)
637                 goto fail7;

639         return (0);

641 fail7:
642         EFSYS_PROBE(fail7);

644 #if EFSYS_OPT_LOOPBACK
645 fail6:
646         EFSYS_PROBE(fail6);
647 #endif  /* EFSYS_OPT_LOOPBACK */

649 fail5:
650         EFSYS_PROBE(fail5);
651 fail4:
652         EFSYS_PROBE(fail4);
653 fail3:
654         EFSYS_PROBE(fail3);
655 fail2:
```

```
656         EFSYS_PROBE(fail2);
657 fail1:
658         EFSYS_PROBE1(fail1, int, rc);

660         return (rc);
661 }

663         __checkReturn   int
664 txc43128_verify(
665         __in            efx_nic_t *enp)
666 {
667         efx_port_t *epp = &(enp->en_port);
668         int rc;

670         if ((rc = xphy_pkg_verify(enp, epp->ep_port, TXC43128_MMD_MASK)) != 0)
671                 goto fail1;

673         return (0);

675 fail1:
676         EFSYS_PROBE1(fail1, int, rc);

678         return (rc);
679 }

681         __checkReturn   int
682 txc43128_uplink_check(
683         __in            efx_nic_t *enp,
684         __out           boolean_t *upp)
685 {
686         efx_port_t *epp = &(enp->en_port);
687         efx_word_t word;
688         int rc;

690         if (epp->ep_mac_type != EFX_MAC_FALCON_XMAC) {
691                 rc = ENOTSUP;
692                 goto fail1;
693         }

695         if ((rc = falcon_mdio_read(enp, epp->ep_port, PHY_XS_MMD,
696             PHY_XS_LANE_STATUS_REG, &word)) != 0)
697                 goto fail2;

699         *upp = ((EFX_WORD_FIELD(word, PHY_XS_ALIGNED) != 0) &&
700             ((EFX_WORD_FIELD(word, PHY_XS_LANE0_SYNC) != 0) ||
701             (EFX_WORD_FIELD(word, PHY_XS_LANE1_SYNC) != 0) ||
702             (EFX_WORD_FIELD(word, PHY_XS_LANE2_SYNC) != 0) ||
703             (EFX_WORD_FIELD(word, PHY_XS_LANE3_SYNC) != 0)));

705         return (0);

707 fail2:
708         EFSYS_PROBE(fail2);
709 fail1:
710         EFSYS_PROBE1(fail1, int, rc);

712         return (rc);
713 }

715         __checkReturn   int
716 txc43128_downlink_check(
717         __in            efx_nic_t *enp,
718         __out           efx_link_mode_t *modep,
719         __out           unsigned int *fcntlp,
720         __out           uint32_t *lp_cap_maskp)
721 {
```



```
722         efx_port_t *epp = &(enp->en_port);
723         boolean_t up;
724         int rc;

726 #if EFSYS_OPT_LOOPBACK
727         switch (epp->ep_loopback_type) {
728         case EFX_LOOPBACK_PHY_XS:
729                 rc = xphy_mmd_fault(enp, epp->ep_port, &up);
730                 if (rc != 0)
731                         goto fail1;

733                 *modep = (up) ? EFX_LINK_10000FDX : EFX_LINK_DOWN;
734                 goto done;

736         case EFX_LOOPBACK_PCS:
737                 rc = xphy_mmd_check(enp, epp->ep_port, PHY_XS_MMD, &up);
738                 if (rc != 0)
739                         goto fail1;

741                 *modep = (up) ? EFX_LINK_10000FDX : EFX_LINK_DOWN;
742                 goto done;

744         default:
745                 break;
746         }
747 #endif /* EFSYS_OPT_LOOPBACK */

749         if ((rc = xphy_mmd_check(enp, epp->ep_port, PCS_MMD, &up)) != 0)
750                 goto fail1;

752         /* bug10934: Reset the controller if the PCS block is stuck down */
753         if (up)
754                 epp->ep_txc43128.bug10934_count = 0;
755         else if (++(epp->ep_txc43128.bug10934_count) > 10) {
756                 (void) txc43128_logic_reset(enp);
757                 epp->ep_txc43128.bug10934_count = 0;
758         }

760         *modep = (up) ? EFX_LINK_10000FDX : EFX_LINK_DOWN;

762 #if EFSYS_OPT_LOOPBACK
763 done:
764 #endif
765         *fcntlp = epp->ep_fcntl;
766         *lp_cap_maskp = epp->ep_lp_cap_mask;

768         return (0);

770 fail1:
771         EFSYS_PROBE1(fail1, int, rc);

773         return (rc);
774 }

776         __checkReturn   int
777 txc43128_oui_get(
778         __in            efx_nic_t *enp,
779         __out           uint32_t *ouip)
780 {
781         efx_port_t *epp = &(enp->en_port);
782         int rc;

784         if ((rc = xphy_mmd_oui_get(enp, epp->ep_port, PMA_PMD_MMD,
785             ouip)) != 0)
786                 goto fail1;
```

```
788         return (0);

790 fail1:
791         EFSYS_PROBE1(fail1, int, rc);

793         return (rc);
794 }

796 #if EFSYS_OPT_PHY_STATS

798 #define TXC43128_STAT_SET(_stat, _mask, _id, _val)                  \
799         do {                                                        \
800                 (_mask) |= (1ULL << (_id));                         \
801                 (_stat)[_id] = (uint32_t)(_val);                    \
802         _NOTE(CONSTANTCONDITION)                                    \
803         } while (B_FALSE)

805 static  __checkReturn                   int
806 txc43128_pma_pmd_stats_update(
807         __in                            efx_nic_t *enp,
808         __inout                         uint64_t *maskp,
809         __inout_ecount(EFX_PHY_NSTATS)  uint32_t *stat)
810 {
811         efx_port_t *epp = &(enp->en_port);
812         efx_word_t word;
813         int rc;

815         if ((rc = falcon_mdio_read(enp, epp->ep_port, PMA_PMD_MMD,
816             PMA_PMD_STATUS1_REG, &word)) != 0)
817                 goto fail1;

819         TXC43128_STAT_SET(stat, *maskp, EFX_PHY_STAT_PMA_PMD_LINK_UP,
820             (EFX_WORD_FIELD(word, PMA_PMD_LINK_UP) != 0) ? 1 : 0);

822         if ((rc = falcon_mdio_read(enp, epp->ep_port, PMA_PMD_MMD,
823             PMA_PMD_STATUS2_REG, &word)) != 0)
824                 goto fail2;

826         TXC43128_STAT_SET(stat, *maskp, EFX_PHY_STAT_PMA_PMD_RX_FAULT,
827             (EFX_WORD_FIELD(word, PMA_PMD_RX_FAULT) != 0) ? 1 : 0);
828         TXC43128_STAT_SET(stat, *maskp, EFX_PHY_STAT_PMA_PMD_TX_FAULT,
829             (EFX_WORD_FIELD(word, PMA_PMD_TX_FAULT) != 0) ? 1 : 0);

831         if ((rc = falcon_mdio_read(enp, epp->ep_port, PMA_PMD_MMD, SIGDET,
832             &word)) != 0)
833                 goto fail3;

835         TXC43128_STAT_SET(stat, *maskp, EFX_PHY_STAT_PMA_PMD_SIGNAL_A,
836             (EFX_WORD_FIELD(word, RX0SIGDET) != 0));
837         TXC43128_STAT_SET(stat, *maskp, EFX_PHY_STAT_PMA_PMD_SIGNAL_B,
838             (EFX_WORD_FIELD(word, RX1SIGDET) != 0));
839         TXC43128_STAT_SET(stat, *maskp, EFX_PHY_STAT_PMA_PMD_SIGNAL_C,
840             (EFX_WORD_FIELD(word, RX2SIGDET) != 0));
841         TXC43128_STAT_SET(stat, *maskp, EFX_PHY_STAT_PMA_PMD_SIGNAL_D,
842             (EFX_WORD_FIELD(word, RX3SIGDET) != 0));

844         return (0);

846 fail3:
847         EFSYS_PROBE(fail3);
848 fail2:
849         EFSYS_PROBE(fail2);
850 fail1:
851         EFSYS_PROBE1(fail1, int, rc);

853         return (rc);
```



```
854 }

856 static  __checkReturn                   int
857 txc43128_pcs_stats_update(
858         __in                            efx_nic_t *enp,
859         __inout                         uint64_t *maskp,
860         __inout_ecount(EFX_PHY_NSTATS)  uint32_t *stat)
861 {
862         efx_port_t *epp = &(enp->en_port);
863         efx_word_t word;
864         int rc;

866         if ((rc = falcon_mdio_read(enp, epp->ep_port, PCS_MMD,
867             PCS_STATUS1_REG, &word)) != 0)
868                 goto fail1;

870         TXC43128_STAT_SET(stat, *maskp, EFX_PHY_STAT_PCS_LINK_UP,
871             (EFX_WORD_FIELD(word, PCS_LINK_UP) != 0) ? 1 : 0);

873         if ((rc = falcon_mdio_read(enp, epp->ep_port, PCS_MMD,
874             PCS_STATUS2_REG, &word)) != 0)
875                 goto fail2;

877         TXC43128_STAT_SET(stat, *maskp, EFX_PHY_STAT_PCS_RX_FAULT,
878             (EFX_WORD_FIELD(word, PCS_RX_FAULT) != 0) ? 1 : 0);
879         TXC43128_STAT_SET(stat, *maskp, EFX_PHY_STAT_PCS_TX_FAULT,
880             (EFX_WORD_FIELD(word, PCS_TX_FAULT) != 0) ? 1 : 0);

882         return (0);

884 fail2:
885         EFSYS_PROBE(fail2);
886 fail1:
887         EFSYS_PROBE1(fail1, int, rc);

889         return (rc);
890 }

892 static  __checkReturn                   int
893 txc43128_phy_xs_stats_update(
894         __in                            efx_nic_t *enp,
895         __inout                         uint64_t *maskp,
896         __inout_ecount(EFX_PHY_NSTATS)  uint32_t *stat)
897 {
898         efx_port_t *epp = &(enp->en_port);
899         efx_word_t word;
900         int rc;

902         if ((rc = falcon_mdio_read(enp, epp->ep_port, PHY_XS_MMD,
903             PHY_XS_STATUS1_REG, &word)) != 0)
904                 goto fail1;

906         TXC43128_STAT_SET(stat, *maskp, EFX_PHY_STAT_PHY_XS_LINK_UP,
907             (EFX_WORD_FIELD(word, PHY_XS_LINK_UP) != 0) ? 1 : 0);

909         if ((rc = falcon_mdio_read(enp, epp->ep_port, PHY_XS_MMD,
910             PHY_XS_STATUS2_REG, &word)) != 0)
911                 goto fail2;

913         TXC43128_STAT_SET(stat, *maskp, EFX_PHY_STAT_PHY_XS_RX_FAULT,
914             (EFX_WORD_FIELD(word, PHY_XS_RX_FAULT) != 0) ? 1 : 0);
915         TXC43128_STAT_SET(stat, *maskp, EFX_PHY_STAT_PHY_XS_TX_FAULT,
916             (EFX_WORD_FIELD(word, PHY_XS_TX_FAULT) != 0) ? 1 : 0);

918         if ((rc = falcon_mdio_read(enp, epp->ep_port, PHY_XS_MMD,
919             PHY_XS_LANE_STATUS_REG, &word)) != 0)
```

```
920                 goto fail3;

922         TXC43128_STAT_SET(stat, *maskp, EFX_PHY_STAT_PHY_XS_ALIGN,
923             (EFX_WORD_FIELD(word, PHY_XS_ALIGNED) != 0) ? 1 : 0);
924         TXC43128_STAT_SET(stat, *maskp, EFX_PHY_STAT_PHY_XS_SYNC_A,
925             (EFX_WORD_FIELD(word, PHY_XS_LANE0_SYNC) != 0) ? 1 : 0);
926         TXC43128_STAT_SET(stat, *maskp, EFX_PHY_STAT_PHY_XS_SYNC_B,
927             (EFX_WORD_FIELD(word, PHY_XS_LANE1_SYNC) != 0) ? 1 : 0);
928         TXC43128_STAT_SET(stat, *maskp, EFX_PHY_STAT_PHY_XS_SYNC_C,
929             (EFX_WORD_FIELD(word, PHY_XS_LANE2_SYNC) != 0) ? 1 : 0);
930         TXC43128_STAT_SET(stat, *maskp, EFX_PHY_STAT_PHY_XS_SYNC_D,
931             (EFX_WORD_FIELD(word, PHY_XS_LANE3_SYNC) != 0) ? 1 : 0);

933         return (0);

935 fail3:
936         EFSYS_PROBE(fail3);
937 fail2:
938         EFSYS_PROBE(fail2);
939 fail1:
940         EFSYS_PROBE1(fail1, int, rc);

942         return (rc);
943 }

945         __checkReturn                   int
946 txc43128_stats_update(
947         __in                            efx_nic_t *enp,
948         __in                            efsys_mem_t *esmp,
949         __out_ecount(EFX_PHY_NSTATS)    uint32_t *stat)
950 {
951         efx_port_t *epp = &(enp->en_port);
952         efx_nic_cfg_t *encp = &(enp->en_nic_cfg);
953         uint64_t mask = 0;
954         uint32_t oui;
955         int rc;

957         _NOTE(ARGUNUSED(esmp))

959         if ((rc = xphy_mmd_oui_get(enp, epp->ep_port, PMA_PMD_MMD, &oui)) != 0)
960                 goto fail1;

962         TXC43128_STAT_SET(stat, mask, EFX_PHY_STAT_OUI, oui);

964         if ((rc = txc43128_pma_pmd_stats_update(enp, &mask, stat)) != 0)
965                 goto fail2;

967         if ((rc = txc43128_pcs_stats_update(enp, &mask, stat)) != 0)
968                 goto fail3;

970         if ((rc = txc43128_phy_xs_stats_update(enp, &mask, stat)) != 0)
971                 goto fail4;

973         /* Ensure all the supported statistics are up to date */
974         EFSYS_ASSERT(mask == encp->enc_phy_stat_mask);

976         return (0);

978 fail4:
979         EFSYS_PROBE(fail4);
980 fail3:
981         EFSYS_PROBE(fail3);
982 fail2:
983         EFSYS_PROBE(fail2);
984 fail1:
985         EFSYS_PROBE1(fail1, int, rc);
```



```
987         return (rc);
988 }
989 #endif  /* EFSYS_OPT_PHY_STATS */

991 #if EFSYS_OPT_PHY_PROPS

993 #if EFSYS_OPT_NAMES
994                 const char __cs *
995 txc43128_prop_name(
996         __in    efx_nic_t *enp,
997         __in    unsigned int id)
998 {
999         _NOTE(ARGUNUSED(enp, id))

1001        EFSYS_ASSERT(B_FALSE);

1003        return (NULL);
1004 }
1005 #endif  /* EFSYS_OPT_NAMES */

1007        __checkReturn   int
1008 txc43128_prop_get(
1009        __in            efx_nic_t *enp,
1010        __in            unsigned int id,
1011        __in            uint32_t flags,
1012        __out           uint32_t *valp)
1013 {
1014        _NOTE(ARGUNUSED(enp, id, flags, valp))

1016        EFSYS_ASSERT(B_FALSE);

1018        return (ENOTSUP);
1019 }

1021        __checkReturn   int
1022 txc43128_prop_set(
1023        __in            efx_nic_t *enp,
1024        __in            unsigned int id,
1025        __in            uint32_t val)
1026 {
1027        _NOTE(ARGUNUSED(enp, id, val))

1029        EFSYS_ASSERT(B_FALSE);

1031        return (ENOTSUP);
1032 }
1033 #endif  /* EFSYS_OPT_PHY_PROPS */

1035 #endif  /* EFSYS_OPT_PHY_TXC43128 */
1036 #endif /* ! codereview */
```

```
**********************************************************
    4262 Thu Aug 22 18:59:30 2013
new/usr/src/uts/common/io/sfxge/txc43128.h
Merged sfxge driver
**********************************************************
```

```
/*-
 * Copyright 2007-2013 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#ifndef _SYS_TXC43128_H
#define _SYS_TXC43128_H

#include "efx.h"

#ifdef  __cplusplus
extern "C" {
#endif

#if EFSYS_OPT_PHY_TXC43128

#define TXC43128_LOOPBACK_MASK                                          \
            ((1 << EFX_LOOPBACK_PHY_XS) |                               \
            (1 << EFX_LOOPBACK_PCS) |                                   \
            (1 << EFX_LOOPBACK_PMA_PMD) |                               \
            FALCON_XMAC_LOOPBACK_MASK)

#define TXC43128_LED_MASK                                               \
            ((1 << EFX_PHY_LED_OFF) |                                   \
            (1 << EFX_PHY_LED_ON))

#define TXC43128_NPROPS         0

#define TXC43128_ADV_CAP_MASK                                           \
            ((1 << EFX_PHY_CAP_10000FDX) |                              \
            (1 << EFX_PHY_CAP_PAUSE))

#define TXC43128_ADV_CAP_PERM   0

#define TXC43128_BIST_MASK      0

extern  __checkReturn   int
txc43128_reset(
        __in            efx_nic_t *enp);

extern  __checkReturn   int
```



```
txc43128_reconfigure(
        __in            efx_nic_t *enp);

extern  __checkReturn   int
txc43128_verify(
        __in            efx_nic_t *enp);

extern  __checkReturn   int
txc43128_uplink_check(
        __in            efx_nic_t *enp,
        __out           boolean_t *upp);

extern  __checkReturn   int
txc43128_downlink_check(
        __in            efx_nic_t *enp,
        __out           efx_link_mode_t *modep,
        __out           unsigned int *fcntlp,
        __out           uint32_t *lp_cap_maskp);

extern  __checkReturn   int
txc43128_oui_get(
        __in            efx_nic_t *enp,
        __out           uint32_t *ouip);

#if EFSYS_OPT_PHY_STATS

/* START MKCONFIG GENERATED Txc43128PhyHeaderStatsMask 0bd778c521babfec */
#define TXC43128_STAT_MASK \
        (1ULL << EFX_PHY_STAT_OUI) | \
        (1ULL << EFX_PHY_STAT_PMA_PMD_LINK_UP) | \
        (1ULL << EFX_PHY_STAT_PMA_PMD_RX_FAULT) | \
        (1ULL << EFX_PHY_STAT_PMA_PMD_TX_FAULT) | \
        (1ULL << EFX_PHY_STAT_PMA_PMD_SIGNAL_A) | \
        (1ULL << EFX_PHY_STAT_PMA_PMD_SIGNAL_B) | \
        (1ULL << EFX_PHY_STAT_PMA_PMD_SIGNAL_C) | \
        (1ULL << EFX_PHY_STAT_PMA_PMD_SIGNAL_D) | \
        (1ULL << EFX_PHY_STAT_PCS_LINK_UP) | \
        (1ULL << EFX_PHY_STAT_PCS_RX_FAULT) | \
        (1ULL << EFX_PHY_STAT_PCS_TX_FAULT) | \
        (1ULL << EFX_PHY_STAT_PHY_XS_LINK_UP) | \
        (1ULL << EFX_PHY_STAT_PHY_XS_RX_FAULT) | \
        (1ULL << EFX_PHY_STAT_PHY_XS_TX_FAULT) | \
        (1ULL << EFX_PHY_STAT_PHY_XS_ALIGN) | \
        (1ULL << EFX_PHY_STAT_PHY_XS_SYNC_A) | \
        (1ULL << EFX_PHY_STAT_PHY_XS_SYNC_B) | \
        (1ULL << EFX_PHY_STAT_PHY_XS_SYNC_C) | \
        (1ULL << EFX_PHY_STAT_PHY_XS_SYNC_D)

/* END MKCONFIG GENERATED Txc43128PhyHeaderStatsMask */

extern  __checkReturn                   int
txc43128_stats_update(
        __in                            efx_nic_t *enp,
        __in                            efsys_mem_t *esmp,
        __out_ecount(EFX_PHY_NSTATS)    uint32_t *stat);

#endif  /* EFSYS_OPT_PHY_STATS */

#if EFSYS_OPT_PHY_PROPS

#if EFSYS_OPT_NAMES

extern          const char __cs *
txc43128_prop_name(
        __in    efx_nic_t *enp,
        __in    unsigned int id);
```

```
129 #endif

131 extern  __checkReturn   int
132 txc43128_prop_get(
133         __in            efx_nic_t *enp,
134         __in            unsigned int id,
135         __in            uint32_t flags,
136         __out           uint32_t *valp);

138 extern  __checkReturn   int
139 txc43128_prop_set(
140         __in            efx_nic_t *enp,
141         __in            unsigned int id,
142         __in            uint32_t val);

144 #endif  /* EFSYS_OPT_PHY_PROPS */

146 #endif  /* EFSYS_OPT_PHY_TXC43128 */

148 #ifdef  __cplusplus
149 }
150 #endif

152 #endif  /* _SYS_TXC43128_H */
153 #endif /* ! codereview */
```

```
**********************************************************
    4472 Thu Aug 22 18:59:30 2013
new/usr/src/uts/common/io/sfxge/txc43128_impl.h
Merged sfxge driver
**********************************************************
```

```
/*-
 * Copyright 2007-2013 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#ifndef _SYS_TXC43128_IMPL_H
#define _SYS_TXC43128_IMPL_H

#ifdef  __cplusplus
extern "C" {
#endif

#if EFSYS_OPT_PHY_TXC43128

#define TXC43128_MMD_MASK                                               \
        ((1 << PMA_PMD_MMD) |                                           \
        (1 << PCS_MMD) |                                                \
        (1 << PHY_XS_MMD))

#define SIGDET  0x000a
#define RX0SIGDET_LBN 1
#define RX0SIGDET_WIDTH 1
#define RX1SIGDET_LBN 2
#define RX1SIGDET_WIDTH 1
#define RX2SIGDET_LBN 3
#define RX2SIGDET_WIDTH 1
#define RX3SIGDET_LBN 4
#define RX3SIGDET_WIDTH 1

#define GLCMD 0xc004
#define LMTSWRST_LBN 14
#define LMTSWRST_WIDTH 1

#define ATXPRE0 0xc043
#define ATXPRE1 0xc044
#define TXPRE02_LBN 3
#define TXPRE02_WIDTH 5
#define TXPRE13_LBN 11
#define TXPRE13_WIDTH 5

#define ATXAMP0 0xc041
```



```
#define ATXAMP1 0xc042
#define TXAMP02_LBN 3
#define TXAMP02_WIDTH 5
#define TXAMP13_LBN 11
#define TXAMP13_WIDTH 5

#define BCTL 0xc280
#define BSTRT_LBN 15
#define BSTRT_WIDTH 1
#define BSTOP_LBN 14
#define BSTOP_WIDTH 1
#define BSTEN_LBN 13
#define BSTEN_WIDTH 1
#define B10EN_LBN 12
#define B10EN_WIDTH 1
#define BTYPE_LBN 10
#define BTYPE_WIDTH 2
#define TSDET_DECODE 0
#define CRPAT_DECODE 1
#define CJPAT_DECODE 2
#define TSRND_DECODE 3

#define BTXFRMCNT 0xc281
#define BRX0FRMCNT 0xc282
#define BRX1FRMCNT 0xc283
#define BRX2FRMCNT 0xc284
#define BRX3FRMCNT 0xc285
#define BRX0ERRCNT 0xc286
#define BRX1ERRCNT 0xc287
#define BRX2ERRCNT 0xc288
#define BRX3ERRCNT 0xc289

#define MNCTL 0xc340
#define MRST_LBN 15
#define MRST_WIDTH 1
#define ALG2TXALED_LBN 14
#define ALG2TXALED_WIDTH 1
#define ALG2RXALED_LBN 13
#define ALG2RXALED_WIDTH 1

#define PIOCFG 0xc345
#define PIO15FNC_LBN 15
#define PIO15FNC_WIDTH 1
#define PIO14FNC_LBN 14
#define PIO14FNC_WIDTH 1
#define PIO13FNC_LBN 13
#define PIO13FNC_WIDTH 1
#define PIO12FNC_LBN 12
#define PIO12FNC_WIDTH 1
#define PIO11FNC_LBN 11
#define PIO11FNC_WIDTH 1
#define PIO10FNC_LBN 10
#define PIO10FNC_WIDTH 1
#define PIO9FNC_LBN 9
#define PIO9FNC_WIDTH 1
#define PIO8FNC_LBN 8
#define PIO8FNC_WIDTH 1
#define PIOFNC_LED_DECODE 0
#define PIOFNC_GPIO_DECODE 1

#define PIODO 0xc346
#define PIO15OUT_LBN 15
#define PIO15OUT_WIDTH 1
#define PIO14OUT_LBN 14
#define PIO14OUT_WIDTH 1
#define PIO13OUT_LBN 13
```

```
#define PIO13OUT_WIDTH 1
#define PIO12OUT_LBN 12
#define PIO12OUT_WIDTH 1
#define PIO11OUT_LBN 11
#define PIO11OUT_WIDTH 1
#define PIO10OUT_LBN 10
#define PIO10OUT_WIDTH 1
#define PIO9OUT_LBN 9
#define PIO9OUT_WIDTH 1
#define PIO8OUT_LBN 8
#define PIO8OUT_WIDTH 1

#define PIODIR 0xc348
#define PIO15DIR_LBN 15
#define PIO15DIR_WIDTH 1
#define PIO14DIR_LBN 14
#define PIO14DIR_WIDTH 1
#define PIO13DIR_LBN 13
#define PIO13DIR_WIDTH 1
#define PIO12DIR_LBN 12
#define PIO12DIR_WIDTH 1
#define PIO11DIR_LBN 11
#define PIO11DIR_WIDTH 1
#define PIO10DIR_LBN 10
#define PIO10DIR_WIDTH 1
#define PIO9DIR_LBN 9
#define PIO9DIR_WIDTH 1
#define PIO8DIR_LBN 8
#define PIO8DIR_WIDTH 1
#define PIODIR_IN_DECODE 0
#define PIODIR_OUT_DECODE 1

#define MNDBLCFG 0xc34f
#define PXS8BLPBK_LBN 11
#define PXS8BLPBK_WIDTH 1
#define LNALPBK_LBN 10
#define LNALPBK_WIDTH 1

#endif  /* EFSYS_OPT_PHY_TXC43128 */

#ifdef  __cplusplus
}
#endif

#endif  /* _SYS_TXC43128_IMPL_H */
#endif /* ! codereview */
```

****
75 Thu Aug 22 18:59:30 2013
new/usr/src/uts/common/io/sfxge/version.h
Merged sfxge driver
****

```
1 #ifndef CI_VERSION_STRING
2 #define CI_VERSION_STRING "v3.2.4.6137-c"
3 #endif
4 #endif /* ! codereview */
```

```
**********************************************************
    7180 Thu Aug 22 18:59:30 2013
new/usr/src/uts/common/io/sfxge/xphy.c
Merged sfxge driver
**********************************************************
```

```
/*
 * Copyright 2006-2013 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#include "efsys.h"
#include "efx.h"
#include "efx_types.h"
#include "efx_regs.h"
#include "efx_impl.h"
#include "xphy.h"

#if EFSYS_OPT_FALCON

static  __checkReturn   int
xphy_mmd_reset_wait(
        __in            efx_nic_t *enp,
        __in            uint8_t port,
        __in            uint8_t mmd)
{
        unsigned int count;
        int rc;

        /* The Clause 22 extenion MMD does not implement IEEE registers */
        if (mmd == CL22EXT_MMD) {
                rc = ENOTSUP;
                goto fail1;
        }

        count = 0;
        do {
                efx_word_t word;

                EFSYS_PROBE1(wait, unsigned int, count);

                /* Sleep for 100 ms */
                EFSYS_SLEEP(100000);

                /* Read control 1 register and check reset state */
                if ((rc = falcon_mdio_read(enp, port, mmd,
                    MMD_CONTROL1_REG, &word)) != 0)
                        goto fail2;

                if (EFX_WORD_FIELD(word, MMD_RESET) == 0)
                        goto done;

        } while (++count < 100);

        rc = ETIMEDOUT;
        goto fail3;

done:
        return (0);

fail3:
        EFSYS_PROBE(fail3);
fail2:
        EFSYS_PROBE(fail2);
fail1:
        EFSYS_PROBE1(fail1, int, rc);

        return (rc);
}

        __checkReturn   int
xphy_pkg_wait(
        __in            efx_nic_t *enp,
        __in            uint8_t port,
        __in            uint32_t mask)
{
        uint8_t mmd;
        int rc;

        for (mmd = 0; mmd <= MAXMMD; mmd++) {
                /* Only check MMDs in the mask */
                if (((1 << mmd) & mask) == 0)
                        continue;

                /*
                 * The Clause 22 extenion MMD does not implement IEEE
                 * registers.
                 */
                if (mmd == CL22EXT_MMD)
                        continue;

                /* Wait for the MMD to come out of reset */
                if ((rc = xphy_mmd_reset_wait(enp, port, mmd)) != 0)
                        goto fail1;
        }

        return (0);

fail1:
        EFSYS_PROBE1(fail1, int, rc);

        return (rc);
}

        __checkReturn   int
xphy_pkg_verify(
        __in            efx_nic_t *enp,
        __in            uint8_t port,
        __in            uint32_t mask)
{
        uint8_t mmd;
        efx_word_t word;
        int rc;
```

```
129         /* Find the first MMD */
130         for (mmd = 0; mmd <= MAXMMD; mmd++) {
131                 uint32_t devices;
132
133                 if (((1 << mmd) & mask) == 0)
134                         continue;
135
136                 /*
137                  * The Clause 22 extenion MMD does not implement IEEE
138                  * registers.
139                  */
140                 if (mmd == CL22EXT_MMD)
141                         continue;
142
143                 if ((rc = falcon_mdio_read(enp, port, mmd,
144                     MMD_DEVICES2_REG, &word)) != 0)
145                         goto fail1;
146
147                 devices = (uint32_t)EFX_WORD_FIELD(word, EFX_WORD_0) << 16;
148
149                 if ((rc = falcon_mdio_read(enp, port, mmd,
150                     MMD_DEVICES1_REG, &word)) != 0)
151                         goto fail2;
152
153                 devices |= (uint32_t)EFX_WORD_FIELD(word, EFX_WORD_0);
154
155                 EFSYS_PROBE1(devices, uint32_t, devices);
156
157                 /* Check that all the specified MMDs are present */
158                 if ((devices & mask) != mask) {
159                         rc = EFAULT;
160                         goto fail3;
161                 }
162
163                 goto done;
164         }
165
166         rc = ENODEV;
167         goto fail4;
168
169 done:
170         for (mmd = 0; mmd <= MAXMMD; mmd++) {
171                 /* Only check MMDs in the mask */
172                 if (((1 << mmd) & mask) == 0)
173                         continue;
174
175                 /*
176                  * The Clause 22 extenion MMD does not implement IEEE
177                  * registers.
178                  */
179                 if (mmd == CL22EXT_MMD)
180                         continue;
181
182                 /* Check the MMD is responding */
183                 if ((rc = falcon_mdio_read(enp, port, mmd,
184                     MMD_STATUS2_REG, &word)) != 0)
185                         goto fail5;
186
187                 if (EFX_WORD_FIELD(word, MMD_RESPONDING) !=
188                     MMD_RESPONDING_DECODE)
189                         goto fail6;
190
191         }
192
193         return (0);
```



```
194
195 fail6:
196         EFSYS_PROBE(fail6);
197 fail5:
198         EFSYS_PROBE(fail5);
199 fail4:
200         EFSYS_PROBE(fail4);
201 fail3:
202         EFSYS_PROBE(fail3);
203 fail2:
204         EFSYS_PROBE(fail2);
205 fail1:
206         EFSYS_PROBE1(fail1, int, rc);
207
208         return (rc);
209 }
210
211         __checkReturn   int
212 xphy_mmd_oui_get(
213         __in            efx_nic_t *enp,
214         __in            uint8_t port,
215         __in            uint8_t mmd,
216         __out           uint32_t *ouip)
217 {
218         efx_word_t word;
219         int rc;
220
221         /* The Clause 22 extenion MMD does not implement IEEE registers */
222         if (mmd == CL22EXT_MMD) {
223                 rc = ENOTSUP;
224                 goto fail1;
225         }
226
227         if ((rc = falcon_mdio_read(enp, port, mmd, MMD_IDH_REG,
228             &word)) != 0)
229                 goto fail2;
230
231         *ouip = (uint32_t)EFX_WORD_FIELD(word, MMD_IDH) << 16;
232
233         if ((rc = falcon_mdio_read(enp, port, mmd, MMD_IDL_REG,
234             &word)) != 0)
235                 goto fail3;
236
237         *ouip |= (uint32_t)EFX_WORD_FIELD(word, MMD_IDL) & 0x0000ffff;
238
239         return (0);
240
241 fail3:
242         EFSYS_PROBE(fail3);
243 fail2:
244         EFSYS_PROBE(fail2);
245 fail1:
246         EFSYS_PROBE1(fail1, int, rc);
247
248         return (rc);
249 }
250
251         __checkReturn   int
252 xphy_mmd_check(
253         __in            efx_nic_t *enp,
254         __in            uint8_t port,
255         __in            uint8_t mmd,
256         __out           boolean_t *upp)
257 {
258         efx_word_t word;
259         int rc;
```

```
261         /* Reset the latched status and fault flags */
262         if ((rc = falcon_mdio_read(enp, port, mmd, MMD_STATUS1_REG,
263             &word)) != 0)
264                 goto fail1;
265
266         if (mmd == PMA_PMD_MMD || mmd == PCS_MMD || mmd == PHY_XS_MMD) {
267                 if ((rc = falcon_mdio_read(enp, port, mmd,
268                     MMD_STATUS2_REG, &word)) != 0)
269                         goto fail2;
270         }
271
272         /* Check the current status and fault flags */
273         if ((rc = falcon_mdio_read(enp, port, mmd, MMD_STATUS1_REG,
274             &word)) != 0)
275                 goto fail3;
276
277         *upp = (EFX_WORD_FIELD(word, MMD_LINK_UP) != 0 &&
278                 EFX_WORD_FIELD(word, MMD_FAULT) == 0);
279
280         return (0);
281
282 fail3:
283         EFSYS_PROBE(fail3);
284 fail2:
285         EFSYS_PROBE(fail2);
286 fail1:
287         EFSYS_PROBE1(fail1, int, rc);
288
289         return (rc);
290 }
291
292         __checkReturn   int
293 xphy_mmd_fault(
294         __in            efx_nic_t *enp,
295         __in            uint8_t port,
296         __out           boolean_t *upp)
297 {
298         efx_word_t word;
299         int rc;
300
301         if ((rc = falcon_mdio_read(enp, port, PHY_XS_MMD,
302             MMD_STATUS2_REG, &word)) != 0)
303                 goto fail1;
304
305         *upp = (EFX_WORD_FIELD(word, MMD_RX_FAULT) == 0);
306
307         return (0);
308
309 fail1:
310         EFSYS_PROBE1(fail1, int, rc);
311
312         return (rc);
313 }
314
315         __checkReturn   int
316 xphy_mmd_loopback_set(
317         __in            efx_nic_t *enp,
318         __in            uint8_t port,
319         __in            uint8_t mmd,
320         __in            boolean_t on)
321 {
322         efx_word_t word;
323         int rc;
324
325         /* The Clause 22 extenion MMD does not implement IEEE registers */
```



```
326         if (mmd == CL22EXT_MMD) {
327                 rc = ENOTSUP;
328                 goto fail1;
329         }
330
331         if ((rc = falcon_mdio_read(enp, port, mmd, MMD_CONTROL1_REG,
332             &word)) != 0)
333                 goto fail2;
334
335         if (mmd == PMA_PMD_MMD)
336                 EFX_SET_WORD_FIELD(word, MMD_PMA_LOOPBACK, (on) ? 1 : 0);
337         else
338                 EFX_SET_WORD_FIELD(word, MMD_LOOPBACK, (on) ? 1 : 0);
339
340         if ((rc = falcon_mdio_write(enp, port, mmd, MMD_CONTROL1_REG,
341             &word)) != 0)
342                 goto fail3;
343
344         return (0);
345
346 fail3:
347         EFSYS_PROBE(fail3);
348 fail2:
349         EFSYS_PROBE(fail2);
350 fail1:
351         EFSYS_PROBE1(fail1, int, rc);
352
353         return (rc);
354 }
355
356 #endif  /* EFSYS_OPT_FALCON */
357 #endif /* ! codereview */
```

```
**********************************************************
   14234 Thu Aug 22 18:59:30 2013
new/usr/src/uts/common/io/sfxge/xphy.h
Merged sfxge driver
**********************************************************
```

```
/*-
 * Copyright 2006-2013 Solarflare Communications Inc.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#ifndef _SYS_XPHY_H
#define _SYS_XPHY_H

#include "efx.h"

#ifdef  __cplusplus
extern "C" {
#endif

/*
 * Common
 */

/* control register 1 */
#define MMD_CONTROL1_REG 0x00
#define MMD_PMA_LOOPBACK_LBN 0
#define MMD_PMA_LOOPBACK_WIDTH 1
#define MMD_LOOPBACK_LBN 14
#define MMD_LOOPBACK_WIDTH 1
#define MMD_RESET_LBN 15
#define MMD_RESET_WIDTH 1

/* status register 1 */
#define MMD_STATUS1_REG 0x01
#define MMD_LINK_UP_LBN 2
#define MMD_LINK_UP_WIDTH 1
#define MMD_FAULT_LBN 7
#define MMD_FAULT_WIDTH 1

/* identifier registers */
#define MMD_IDH_REG 0x02
#define MMD_IDH_LBN 0
#define MMD_IDH_WIDTH 16

#define MMD_IDL_REG 0x03
#define MMD_IDL_LBN 0
```



```
#define MMD_IDL_WIDTH 16

/* devices in package registers */
#define MMD_DEVICES1_REG 0x05
#define PMA_PMD_PRESENT_LBN 1
#define PMA_PMD_PRESENT_WIDTH 1
#define WIS_PRESENT_LBN 2
#define WIS_PRESENT_WIDTH 1
#define PCS_PRESENT_LBN 3
#define PCS_PRESENT_WIDTH 1
#define PHY_XS_PRESENT_LBN 4
#define PHY_XS_PRESENT_WIDTH 1
#define DTE_XS_PRESENT_LBN 5
#define DTE_XS_PRESENT_WIDTH 1
#define TC_PRESENT_LBN 6
#define TC_PRESENT_WIDTH 1
#define AN_PRESENT_LBN 7
#define AN_PRESENT_WIDTH 1

#define MMD_DEVICES2_REG 0x06
#define CL22EXT_PRESENT_LBN 13
#define CL22EXT_PRESENT_WIDTH 1
#define VENDOR_DEVICE1_PRESENT_LBN 14
#define VENDOR_DEVICE1_PRESENT_WIDTH 1
#define VENDOR_DEVICE2_PRESENT_LBN 15
#define VENDOR_DEVICE2_PRESENT_WIDTH 1

#define MMD_STATUS2_REG 0x08
#define MMD_RX_FAULT_LBN 10
#define MMD_RX_FAULT_WIDTH 1
#define MMD_TX_FAULT_LBN 11
#define MMD_TX_FAULT_WIDTH 1
#define MMD_RESPONDING_LBN 14
#define MMD_RESPONDING_WIDTH 2
#define MMD_RESPONDING_DECODE 0x2       /* 10 */

/*
 * PMA/PMD
 */

/* control register 1 */
#define PMA_PMD_CONTROL1_REG 0x00
#define PMA_PMD_LOOPBACK_EN_LBN 0
#define PMA_PMD_LOOPBACK_EN_WIDTH 1
#define PMA_PMD_LOW_POWER_EN_LBN 11
#define PMA_PMD_LOW_POWER_EN_WIDTH 1
#define PMA_PMD_RESET_LBN 15
#define PMA_PMD_RESET_WIDTH 1

/* status register 1 */
#define PMA_PMD_STATUS1_REG 0x01
#define PMA_PMD_POWER_CAP_LBN 1
#define PMA_PMD_POWER_CAP_WIDTH 1
#define PMA_PMD_LINK_UP_LBN 2
#define PMA_PMD_LINK_UP_WIDTH 1
#define PMA_PMD_FAULT_LBN 7
#define PMA_PMD_FAULT_WIDTH 1

/* control register 2 */
#define PMA_PMD_CONTROL2_REG 0x07
#define PMA_PMD_TYPE_LBN 0
#define PMA_PMD_TYPE_WIDTH 4
#define PMA_PMD_10BASE_T_DECODE 0xf     /* 1111 */
#define PMA_PMD_100BASE_TX_DECODE 0xe   /* 1110 */
#define PMA_PMD_1000BASE_KX_DECODE 0xd  /* 1101 */
#define PMA_PMD_1000BASE_T_DECODE 0xc   /* 1100 */
```

```
#define PMA_PMD_10GBASE_KR_DECODE 0xb   /* 1011 */
#define PMA_PMD_10GBASE_KX4_DECODE 0xa  /* 1010 */
#define PMA_PMD_10GBASE_T_DECODE 0x9    /* 1001 */
#define PMA_PMD_10GBASE_LRM_DECODE 0x8  /* 1001 */
#define PMA_PMD_10GBASE_SR_DECODE 0x7   /* 0111 */
#define PMA_PMD_10GBASE_LR_DECODE 0x6   /* 0110 */
#define PMA_PMD_10GBASE_ER_DECODE 0x5   /* 0101 */
#define PMA_PMD_10GBASE_SW_DECODE 0x3   /* 0011 */
#define PMA_PMD_10GBASE_LW_DECODE 0x2   /* 0010 */
#define PMA_PMD_10GBASE_EW_DECODE 0x1   /* 0001 */

/* status register 2 */
#define PMA_PMD_STATUS2_REG 0x08
#define PMA_PMD_LOOPBACK_CAP_LBN 0
#define PMA_PMD_LOOPBACK_CAP_WIDTH 1
#define PMA_PMD_10GBASE_EW_CAP_LBN 1
#define PMA_PMD_10GBASE_EW_CAP_WIDTH 1
#define PMA_PMD_10GBASE_LW_CAP_LBN 2
#define PMA_PMD_10GBASE_LW_CAP_WIDTH 1
#define PMA_PMD_10GBASE_SW_CAP_LBN 3
#define PMA_PMD_10GBASE_SW_CAP_WIDTH 1
#define PMA_PMD_10GBASE_ER_CAP_LBN 5
#define PMA_PMD_10GBASE_ER_CAP_WIDTH 1
#define PMA_PMD_10GBASE_LR_CAP_LBN 6
#define PMA_PMD_10GBASE_LR_CAP_WIDTH 1
#define PMA_PMD_10GBASE_SR_CAP_LBN 7
#define PMA_PMD_10GBASE_SR_CAP_WIDTH 1
#define PMA_PMD_TX_DISABLE_CAP_LBN 8
#define PMA_PMD_TX_DISABLE_CAP_WIDTH 1
#define PMA_PMD_EXT_CAP_LBN 9
#define PMA_PMD_EXT_CAP_WIDTH 1
#define PMA_PMD_RX_FAULT_LBN 10
#define PMA_PMD_RX_FAULT_WIDTH 1
#define PMA_PMD_TX_FAULT_LBN 11
#define PMA_PMD_TX_FAULT_WIDTH 1
#define PMA_PMD_RX_FAULT_CAP_LBN 12
#define PMA_PMD_RX_FAULT_CAP_WIDTH 1
#define PMA_PMD_TX_FAULT_CAP_LBN 13
#define PMA_PMD_TX_FAULT_CAP_WIDTH 1
#define PMA_PMD_RESPONDING_LBN 14
#define PMA_PMD_RESPONDING_WIDTH 2
#define PMA_PMD_RESPONDING_DECODE 0x2   /* 10 */

/* extended capabilities register */
#define PMA_PMD_EXT_CAP_REG 0x0b
#define PMA_PMD_10GBASE_CX4_CAP_LBN 0
#define PMA_PMD_10GBASE_CX4_CAP_WIDTH 1
#define PMA_PMD_10GBASE_LRM_CAP_LBN 1
#define PMA_PMD_10GBASE_LRM_CAP_WIDTH 1
#define PMA_PMD_10GBASE_T_CAP_LBN 2
#define PMA_PMD_10GBASE_T_CAP_WIDTH 1
#define PMA_PMD_10GBASE_KX4_CAP_LBN 3
#define PMA_PMD_10GBASE_KX4_CAP_WIDTH 1
#define PMA_PMD_10GBASE_KR_CAP_LBN 4
#define PMA_PMD_10GBASE_KR_CAP_WIDTH 1
#define PMA_PMD_1000BASE_T_CAP_LBN 5
#define PMA_PMD_1000BASE_T_CAP_WIDTH 1
#define PMA_PMD_1000BASE_KX_CAP_LBN 6
#define PMA_PMD_1000BASE_KX_CAP_WIDTH 1
#define PMA_PMD_100BASE_TX_CAP_LBN 7
#define PMA_PMD_100BASE_TX_CAP_WIDTH 1
#define PMA_PMD_10BASE_T_CAP_LBN 8
#define PMA_PMD_10BASE_T_CAP_WIDTH 1

/* channel A-D signal/noise ratio registers */
#define PMA_PMD_CHANNELA_SNR_REG 0x85
```



```
#define PMA_PMD_CHANNELB_SNR_REG 0x86
#define PMA_PMD_CHANNELC_SNR_REG 0x87
#define PMA_PMD_CHANNELD_SNR_REG 0x88
#define PMA_PMD_CHANNELA_MIN_SNR_REG 0x89
#define PMA_PMD_CHANNELB_MIN_SNR_REG 0x8a
#define PMA_PMD_CHANNELC_MIN_SNR_REG 0x8b
#define PMA_PMD_CHANNELD_MIN_SNR_REG 0x8c
#define PMA_PMD_SNR_LBN 0
#define PMA_PMD_SNR_WIDTH 16

/* LASI control register */
#define PMA_PMD_LASI_CONTROL_REG 0x9002
#define PMA_PMD_LS_ALARM_EN_LBN 0
#define PMA_PMD_LS_ALARM_EN_WIDTH 1
#define PMA_PMD_TX_ALARM_EN_LBN 1
#define PMA_PMD_TX_ALARM_EN_WIDTH 1
#define PMA_PMD_RX_ALARM_EN_LBN 2
#define PMA_PMD_RX_ALARM_EN_WIDTH 1

/* LASI status register */
#define PMA_PMD_LASI_STATUS_REG 0x9005
#define PMA_PMD_LS_ALARM_LBN 0
#define PMA_PMD_LS_ALARM_WIDTH 1
#define PMA_PMD_TX_ALARM_LBN 1
#define PMA_PMD_TX_ALARM_WIDTH 1
#define PMA_PMD_RX_ALARM_LBN 2
#define PMA_PMD_RX_ALARM_WIDTH 1

/*
 * PCS
 */

/* control register 1 */
#define PCS_CONTROL1_REG 0x00
#define PCS_LOW_POWER_EN_LBN 11
#define PCS_LOW_POWER_EN_WIDTH 1
#define PCS_LOOPBACK_EN_LBN 14
#define PCS_LOOPBACK_EN_WIDTH 1
#define PCS_RESET_LBN 15
#define PCS_RESET_WIDTH 1

/* status register 1 */
#define PCS_STATUS1_REG 0x01
#define PCS_POWER_CAP_LBN 1
#define PCS_POWER_CAP_WIDTH 1
#define PCS_LINK_UP_LBN 2
#define PCS_LINK_UP_WIDTH 1
#define PCS_FAULT_LBN 7
#define PCS_FAULT_WIDTH 1

/* control register 2 */
#define PCS_CONTROL2_REG 0x07
#define PCS_TYPE_LBN 0
#define PCS_TYPE_WIDTH 2
#define PCS_10GBASE_T_DECODE 0x3        /* 11 */
#define PCS_10GBASE_W_DECODE 0x2        /* 10 */
#define PCS_10GBASE_X_DECODE 0x1        /* 01 */
#define PCS_10GBASE_R_DECODE 0x0        /* 00 */

/* 10G status register 2 */
#define PCS_STATUS2_REG 0x08
#define PCS_10GBASE_R_CAP_LBN 0
#define PCS_10GBASE_R_CAP_WIDTH 1
#define PCS_10GBASE_X_CAP_LBN 1
#define PCS_10GBASE_X_CAP_WIDTH 1
#define PCS_10GBASE_W_CAP_LBN 2
```

```
260 #define PCS_10GBASE_W_CAP_WIDTH 1
261 #define PCS_10GBASE_T_CAP_LBN 3
262 #define PCS_10GBASE_T_CAP_WIDTH 1
263 #define PCS_RX_FAULT_LBN 10
264 #define PCS_RX_FAULT_WIDTH 1
265 #define PCS_TX_FAULT_LBN 11
266 #define PCS_TX_FAULT_WIDTH 1
267 #define PCS_RESPONDING_LBN 14
268 #define PCS_RESPONDING_WIDTH 2
269 #define PCS_RESPONDING_DECODE 0x2       /* 10 */

271 /* 10G-BASE-T/R status register 2 */
272 #define PCS_10GBASE_T_STATUS2_REG 0x21
273 #define PCS_10GBASE_R_STATUS2_REG 0x21
274 #define PCS_ERR_LBN 0
275 #define PCS_ERR_WIDTH 7
276 #define PCS_BER_LBN 8
277 #define PCS_BER_WIDTH 6
278 #define PCS_HIGH_BER_LBN 14
279 #define PCS_HIGH_BER_WIDTH 1
280 #define PCS_BLOCK_LOCK_LBN 15
281 #define PCS_BLOCK_LOCK_WIDTH 1

283 /*
284  * PHY_XS
285  */

287 /* control register 1 */
288 #define PHY_XS_CONTROL1_REG 0x00
289 #define PHY_XS_LOW_POWER_EN_LBN 11
290 #define PHY_XS_LOW_POWER_EN_WIDTH 1
291 #define PHY_XS_LOOPBACK_EN_LBN 14
292 #define PHY_XS_LOOPBACK_EN_WIDTH 1
293 #define PHY_XS_RESET_LBN 15
294 #define PHY_XS_RESET_WIDTH 1

296 /* status register 1 */
297 #define PHY_XS_STATUS1_REG 0x01
298 #define PHY_XS_POWER_CAP_LBN 1
299 #define PHY_XS_POWER_CAP_WIDTH 1
300 #define PHY_XS_LINK_UP_LBN 2
301 #define PHY_XS_LINK_UP_WIDTH 1
302 #define PHY_XS_FAULT_LBN 7
303 #define PHY_XS_FAULT_WIDTH 1

305 /* no control register 2 */

307 /* status register 2 */
308 #define PHY_XS_STATUS2_REG 0x08
309 #define PHY_XS_RX_FAULT_LBN 10
310 #define PHY_XS_RX_FAULT_WIDTH 1
311 #define PHY_XS_TX_FAULT_LBN 11
312 #define PHY_XS_TX_FAULT_WIDTH 1
313 #define PHY_XS_RESPONDING_LBN 14
314 #define PHY_XS_RESPONDING_WIDTH 2
315 #define PHY_XS_RESPONDING_DECODE 0x2    /* 10 */

317 /* lane status register */
318 #define PHY_XS_LANE_STATUS_REG 0x18
319 #define PHY_XS_ALIGNED_LBN 12
320 #define PHY_XS_ALIGNED_WIDTH 1
321 #define PHY_XS_LANE0_SYNC_LBN 0
322 #define PHY_XS_LANE0_SYNC_WIDTH 1
323 #define PHY_XS_LANE1_SYNC_LBN 1
324 #define PHY_XS_LANE1_SYNC_WIDTH 1
325 #define PHY_XS_LANE2_SYNC_LBN 2
```



```
326 #define PHY_XS_LANE2_SYNC_WIDTH 1
327 #define PHY_XS_LANE3_SYNC_LBN 3
328 #define PHY_XS_LANE3_SYNC_WIDTH 1

330 /*
331  * DTE_XS
332  */

334 /* control register 1 */
335 #define DTE_XS_CONTROL1_REG 0x00
336 #define DTE_XS_LOW_POWER_EN_LBN 11
337 #define DTE_XS_LOW_POWER_EN_WIDTH 1
338 #define DTE_XS_LOOPBACK_EN_LBN 14
339 #define DTE_XS_LOOPBACK_EN_WIDTH 1
340 #define DTE_XS_RESET_LBN 15
341 #define DTE_XS_RESET_WIDTH 1

343 /* status register 1 */
344 #define DTE_XS_STATUS1_REG 0x01
345 #define DTE_XS_POWER_CAP_LBN 1
346 #define DTE_XS_POWER_CAP_WIDTH 1
347 #define DTE_XS_LINK_UP_LBN 2
348 #define DTE_XS_LINK_UP_WIDTH 1
349 #define DTE_XS_FAULT_LBN 7
350 #define DTE_XS_FAULT_WIDTH 1

352 /* no control register 2 */

354 /* status register 2 */
355 #define DTE_XS_STATUS2_REG 0x08
356 #define DTE_XS_RX_FAULT_LBN 10
357 #define DTE_XS_RX_FAULT_WIDTH 1
358 #define DTE_XS_TX_FAULT_LBN 11
359 #define DTE_XS_TX_FAULT_WIDTH 1
360 #define DTE_XS_RESPONDING_LBN 14
361 #define DTE_XS_RESPONDING_WIDTH 2
362 #define DTE_XS_RESPONDING_DECODE 0x2    /* 10 */

364 /* lane status register */
365 #define DTE_XS_LANE_STATUS_REG 0x18
366 #define DTE_XS_ALIGNED_LBN 12
367 #define DTE_XS_ALIGNED_WIDTH 1
368 #define DTE_XS_LANE0_SYNC_LBN 0
369 #define DTE_XS_LANE0_SYNC_WIDTH 1
370 #define DTE_XS_LANE1_SYNC_LBN 1
371 #define DTE_XS_LANE1_SYNC_WIDTH 1
372 #define DTE_XS_LANE2_SYNC_LBN 2
373 #define DTE_XS_LANE2_SYNC_WIDTH 1
374 #define DTE_XS_LANE3_SYNC_LBN 3
375 #define DTE_XS_LANE3_SYNC_WIDTH 1

377 /*
378  * AN
379  */

381 /* control register 1 */
382 #define AN_CONTROL1_REG 0x00
383 #define AN_RESTART_LBN 9
384 #define AN_RESTART_WIDTH 1
385 #define AN_ENABLE_LBN 12
386 #define AN_ENABLE_WIDTH 1
387 #define AN_RESET_LBN 15
388 #define AN_RESET_WIDTH 1

390 /* status register 1 */
391 #define AN_STATUS1_REG 0x01
```

```
392 #define AN_LP_CAP_LBN 0
393 #define AN_LP_CAP_WIDTH 1
394 #define AN_LINK_UP_LBN 2
395 #define AN_LINK_UP_WIDTH 1
396 #define AN_FAULT_LBN 4
397 #define AN_FAULT_WIDTH 1
398 #define AN_COMPLETE_LBN 5
399 #define AN_COMPLETE_WIDTH 1
400 #define AN_PAGE_RCVD_LBN 6
401 #define AN_PAGE_RCVD_WIDTH 1
402 #define AN_XNP_CAP_LBN 7
403 #define AN_XNP_CAP_WIDTH 1

405 /* advertised capabilities register */
406 #define AN_ADV_BP_CAP_REG 0x10
407 #define AN_ADV_TA_10BASE_T_LBN 5
408 #define AN_ADV_TA_10BASE_T_WIDTH 1
409 #define AN_ADV_TA_10BASE_T_FDX_LBN 6
410 #define AN_ADV_TA_10BASE_T_FDX_WIDTH 1
411 #define AN_ADV_TA_100BASE_TX_LBN 7
412 #define AN_ADV_TA_100BASE_TX_WIDTH 1
413 #define AN_ADV_TA_100BASE_TX_FDX_LBN 8
414 #define AN_ADV_TA_100BASE_TX_FDX_WIDTH 1
415 #define AN_ADV_TA_100BASE_T4_LBN 9
416 #define AN_ADV_TA_100BASE_T4_WIDTH 1
417 #define AN_ADV_TA_PAUSE_LBN 10
418 #define AN_ADV_TA_PAUSE_WIDTH 1
419 #define AN_ADV_TA_ASM_DIR_LBN 11
420 #define AN_ADV_TA_ASM_DIR_WIDTH 1

422 /* link partner base page capabilities register */
423 #define AN_LP_BP_CAP_REG 0x13
424 #define AN_LP_TA_10BASE_T_LBN 5
425 #define AN_LP_TA_10BASE_T_WIDTH 1
426 #define AN_LP_TA_10BASE_T_FDX_LBN 6
427 #define AN_LP_TA_10BASE_T_FDX_WIDTH 1
428 #define AN_LP_TA_100BASE_TX_LBN 7
429 #define AN_LP_TA_100BASE_TX_WIDTH 1
430 #define AN_LP_TA_100BASE_TX_FDX_LBN 8
431 #define AN_LP_TA_100BASE_TX_FDX_WIDTH 1
432 #define AN_LP_TA_100BASE_T4_LBN 9
433 #define AN_LP_TA_100BASE_T4_WIDTH 1
434 #define AN_LP_TA_PAUSE_LBN 10
435 #define AN_LP_TA_PAUSE_WIDTH 1
436 #define AN_LP_TA_ASM_DIR_LBN 11
437 #define AN_LP_TA_ASM_DIR_WIDTH 1

439 /* 10GBASE-T control register */
440 #define AN_10G_BASE_T_CONTROL_REG 0x20
441 #define AN_10G_BASE_T_ADV_LBN 12
442 #define AN_10G_BASE_T_ADV_WIDTH 1

444 /* 10GBASE-T status register */
445 #define AN_10G_BASE_T_STATUS_REG 0x21
446 #define AN_10G_BASE_T_LP_LBN 11
447 #define AN_10G_BASE_T_LP_WIDTH 1
448 #define AN_CONFIG_FAULT_LBN 15
449 #define AN_CONFIG_FAULT_WIDTH 1
450 #define AN_MASTER_LBN 14
451 #define AN_MASTER_WIDTH 1
452 #define AN_LOCAL_RX_OK_LBN 13
453 #define AN_LOCAL_RX_OK_WIDTH 1
454 #define AN_REMOTE_RX_OK_LBN 12
455 #define AN_REMOTE_RX_OK_WIDTH 1

457 extern  __checkReturn   int
```



```
458 xphy_pkg_verify(
459         __in            efx_nic_t *enp,
460         __in            uint8_t port,
461         __in            uint32_t mask);

463 extern  __checkReturn   int
464 xphy_pkg_wait(
465         __in            efx_nic_t *enp,
466         __in            uint8_t port,
467         __in            uint32_t mask);

469 extern  __checkReturn   int
470 xphy_mmd_oui_get(
471         __in            efx_nic_t *enp,
472         __in            uint8_t port,
473         __in            uint8_t mmd,
474         __out           uint32_t *ouip);

476 extern  __checkReturn   int
477 xphy_mmd_check(
478         __in            efx_nic_t *enp,
479         __in            uint8_t port,
480         __in            uint8_t mmd,
481         __out           boolean_t *upp);

483 extern  __checkReturn   int
484 xphy_mmd_fault(
485         __in            efx_nic_t *enp,
486         __in            uint8_t port,
487         __out           boolean_t *upp);

489 extern  __checkReturn   int
490 xphy_mmd_loopback_set(
491         __in            efx_nic_t *enp,
492         __in            uint8_t port,
493         __in            uint8_t mmd,
494         __in            boolean_t on);

496 #ifdef  __cplusplus
497 }
498 #endif

500 #endif /* _SYS_XPHY_H */
501 #endif /* ! codereview */
```

```
**********************************************************
   16959 Thu Aug 22 18:59:30 2013
new/usr/src/uts/intel/Makefile.intel.shared
Merged sfxge driver
**********************************************************
```

```
   1 # CDDL HEADER START
   2 #
   3 # The contents of this file are subject to the terms of the
   4 # Common Development and Distribution License (the "License").
   5 # You may not use this file except in compliance with the License.
   6 #
   7 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   8 # or http://www.opensolaris.org/os/licensing.
   9 # See the License for the specific language governing permissions
  10 # and limitations under the License.
  11 #
  12 # When distributing Covered Code, include this CDDL HEADER in each
  13 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  14 # If applicable, add the following below this CDDL HEADER, with the
  15 # fields enclosed by brackets "[]" replaced with your own identifying
  16 # information: Portions Copyright [yyyy] [name of copyright owner]
  17 #
  18 # CDDL HEADER END
  19 #

  21 # Copyright (c) 2005, 2010, Oracle and/or its affiliates. All rights reserved.
  22 # Copyright (c) 2012 Nexenta Systems, Inc. All rights reserved.

  24 #
  25 #       This makefile contains the common definitions for all intel
  26 #       implementation architecture independent modules.
  27 #

  29 #
  30 #       Machine type (implementation architecture):
  31 #
  32 PLATFORM         = i86pc

  34 #
  35 #       Everybody needs to know how to build modstubs.o and to locate unix.o.
  36 #       Note that unix.o must currently be selected from among the possible
  37 #       "implementation architectures". Note further, that unix.o is only
  38 #       used as an optional error check for undefines so (theoretically)
  39 #       any "implementation architectures" could be used. We choose i86pc
  40 #       because it is the reference port.
  41 #
  42 UNIX_DIR         = $(UTSBASE)/i86pc/unix
  43 GENLIB_DIR       = $(UTSBASE)/intel/genunix
  44 IPDRV_DIR        = $(UTSBASE)/intel/ip
  45 MODSTUBS_DIR     = $(UNIX_DIR)
  46 DSF_DIR          = $(UTSBASE)/$(PLATFORM)/genassym
  47 LINTS_DIR       = $(OBJS_DIR)
  48 LINT_LIB_DIR     = $(UTSBASE)/intel/lint-libs/$(OBJS_DIR)

  50 UNIX_O           = $(UNIX_DIR)/$(OBJS_DIR)/unix.o
  51 GENLIB           = $(GENLIB_DIR)/$(OBJS_DIR)/libgenunix.so
  52 MODSTUBS_O       = $(MODSTUBS_DIR)/$(OBJS_DIR)/modstubs.o
  53 LINT_LIB         = $(UTSBASE)/i86pc/lint-libs/$(OBJS_DIR)/llib-lunix.ln
  54 GEN_LINT_LIB     = $(UTSBASE)/intel/lint-libs/$(OBJS_DIR)/llib-lgenunix.ln

  56 #
  57 #       Include the makefiles which define build rule templates, the
  58 #       collection of files per module, and a few specific flags. Note
  59 #       that order is significant, just as with an include path. The
  60 #       first build rule template which matches the files name will be
  61 #       used. By including these in order from most machine dependent
```



```
  62 #       to most machine independent, we allow a machine dependent file
  63 #       to be used in preference over a machine independent version
  64 #       (Such as a machine specific optimization, which preserves the
  65 #       interfaces.)
  66 #
  67 include $(UTSTREE)/intel/Makefile.files
  68 include $(UTSTREE)/common/Makefile.files

  70 #
  71 # ----- TRANSITIONAL SECTION --------------------------------------------------
  72 #

  74 #
  75 #       Not everything which *should* be a module is a module yet. The
  76 #       following is a list of such objects which are currently part of
  77 #       genunix but which might someday become kmods.  This must be
  78 #       defined before we include Makefile.uts, or else genunix's build
  79 #       won't be as parallel as we might like.
  80 #
  81 NOT_YET_KMODS    = $(OLDPTY_OBJS) $(PTY_OBJS) $(VCONS_CONF_OBJS) $(MOD_OBJS)

  83 #
  84 # ----- END OF TRANSITIONAL SECTION -------------------------------------------
  85 #
  86 #       Include machine independent rules. Note that this does not imply
  87 #       that the resulting module from rules in Makefile.uts is machine
  88 #       independent. Only that the build rules are machine independent.
  89 #
  90 include $(UTSBASE)/Makefile.uts

  92 #
  93 #       The following must be defined for all implementations:
  94 #
  95 MODSTUBS               = $(UTSBASE)/intel/ia32/ml/modstubs.s

  97 #
  98 #       Define supported builds
  99 #
 100 DEF_BUILDS             = $(DEF_BUILDS64) $(DEF_BUILDS32)
 101 ALL_BUILDS             = $(ALL_BUILDS64) $(ALL_BUILDS32)

 103 #
 104 #       x86 or amd64 inline templates
 105 #
 106 INLINES_32             = $(UTSBASE)/intel/ia32/ml/ia32.il
 107 INLINES_64             = $(UTSBASE)/intel/amd64/ml/amd64.il
 108 INLINES                += $(INLINES_$(CLASS))

 110 #
 111 #       kernel-specific optimizations; override default in Makefile.master
 112 #

 114 CFLAGS_XARCH_32         = $(i386_CFLAGS)
 115 CFLAGS_XARCH_64         = $(amd64_CFLAGS)
 116 CFLAGS_XARCH            = $(CFLAGS_XARCH_$(CLASS))

 118 COPTFLAG_32             = $(COPTFLAG)
 119 COPTFLAG_64             = $(COPTFLAG64)
 120 COPTIMIZE               = $(COPTFLAG_$(CLASS))

 122 CFLAGS                  = $(CFLAGS_XARCH)
 123 CFLAGS                  += $(COPTIMIZE)
 124 CFLAGS                  += $(INLINES) -D_ASM_INLINES
 125 CFLAGS                  += $(CCMODE)
 126 CFLAGS                  += $(SPACEFLAG)
 127 CFLAGS                  += $(CCUNBOUND)
```

```
128 CFLAGS                  += $(CFLAGS_uts)
129 CFLAGS                  += -xstrconst

131 ASFLAGS_XARCH_32        = $(i386_ASFLAGS)
132 ASFLAGS_XARCH_64        = $(amd64_ASFLAGS)
133 ASFLAGS_XARCH           = $(ASFLAGS_XARCH_$(CLASS))

135 ASFLAGS                 += $(ASFLAGS_XARCH)

137 #
138 #       Define the base directory for installation.
139 #
140 BASE_INS_DIR    = $(ROOT)

142 #
143 #       Debugging level
144 #
145 #       Special knowledge of which special debugging options affect which
146 #       file is used to optimize the build if these flags are changed.
147 #
148 DEBUG_DEFS_OBJ32        =
149 DEBUG_DEFS_DBG32        = -DDEBUG
150 DEBUG_DEFS_OBJ64        =
151 DEBUG_DEFS_DBG64        = -DDEBUG
152 DEBUG_DEFS              = $(DEBUG_DEFS_$(BUILD_TYPE))

154 DEBUG_COND_OBJ32        :sh = echo \\043
155 DEBUG_COND_DBG32        =
156 DEBUG_COND_OBJ64        :sh = echo \\043
157 DEBUG_COND_DBG64        =
158 IF_DEBUG_OBJ            = $(DEBUG_COND_$(BUILD_TYPE))$(OBJS_DIR)/

160 $(IF_DEBUG_OBJ)syscall.o        :=      DEBUG_DEFS      += -DSYSCALLTRACE
161 $(IF_DEBUG_OBJ)clock.o          :=      DEBUG_DEFS      += -DKSLICE=1

163 #
164 #       Collect the preprocessor definitions to be associated with *all*
165 #       files.
166 #
167 ALL_DEFS         = $(DEBUG_DEFS) $(OPTION_DEFS)

169 #
170 #       The kernels modules which are "implementation architecture"
171 #       specific for this machine are enumerated below. Note that most
172 #       of these modules must exist (in one form or another) for each
173 #       architecture.
174 #
175 #       Common Drivers (usually pseudo drivers) (/kernel/drv)
176 #       DRV_KMODS are built both 32-bit and 64-bit
177 #       DRV_KMODS_32 are built only 32-bit
178 #       DRV_KMODS_64 are built only 64-bit
179 #
180 DRV_KMODS       += aac
181 DRV_KMODS       += aggr
182 DRV_KMODS       += ahci
183 DRV_KMODS       += amd64_gart
184 DRV_KMODS       += amr
185 DRV_KMODS       += agpgart
186 DRV_KMODS       += srn
187 DRV_KMODS       += agptarget
188 DRV_KMODS       += arn
189 DRV_KMODS       += arp
190 DRV_KMODS       += asy
191 DRV_KMODS       += ata
192 DRV_KMODS       += ath
193 DRV_KMODS       += atu
```



```
194 DRV_KMODS       += audio
195 DRV_KMODS       += audio1575
196 DRV_KMODS       += audio810
197 DRV_KMODS       += audiocmi
198 DRV_KMODS       += audiocmihd
199 DRV_KMODS       += audioemu10k
200 DRV_KMODS       += audioens
201 DRV_KMODS       += audiohd
202 DRV_KMODS       += audioixp
203 DRV_KMODS       += audiols
204 DRV_KMODS       += audiop16x
205 DRV_KMODS       += audiopci
206 DRV_KMODS       += audiosolo
207 DRV_KMODS       += audiots
208 DRV_KMODS       += audiovia823x
209 DRV_KMODS_32    += audiovia97
210 DRV_KMODS       += bl
211 DRV_KMODS       += blkdev
212 DRV_KMODS       += bge
213 DRV_KMODS       += bofi
214 DRV_KMODS       += bpf
215 DRV_KMODS       += bridge
216 DRV_KMODS       += bscbus
217 DRV_KMODS       += bscv
218 DRV_KMODS       += chxge
219 DRV_KMODS       += cxgbe
220 DRV_KMODS       += ntxn
221 DRV_KMODS       += myri10ge
222 DRV_KMODS       += clone
223 DRV_KMODS       += cmdk
224 DRV_KMODS       += cn
225 DRV_KMODS       += conskbd
226 DRV_KMODS       += consms
227 DRV_KMODS       += cpuid
228 DRV_KMODS       += cpunex
229 DRV_KMODS       += crypto
230 DRV_KMODS       += cryptoadm
231 DRV_KMODS       += dca
232 DRV_KMODS       += devinfo
233 DRV_KMODS       += dld
234 DRV_KMODS       += dlpistub
235 DRV_KMODS_32    += dnet
236 DRV_KMODS       += dump
237 DRV_KMODS       += ecpp
238 DRV_KMODS       += emlxs
239 DRV_KMODS       += fd
240 DRV_KMODS       += fdc
241 DRV_KMODS       += fm
242 DRV_KMODS       += fssnap
243 DRV_KMODS       += hxge
244 DRV_KMODS       += i8042
245 DRV_KMODS       += i915
246 DRV_KMODS       += icmp
247 DRV_KMODS       += icmp6
248 DRV_KMODS       += intel_nb5000
249 DRV_KMODS       += intel_nhm
250 DRV_KMODS       += ip
251 DRV_KMODS       += ip6
252 DRV_KMODS       += ipf
253 DRV_KMODS       += ipnet
254 DRV_KMODS       += ippctl
255 DRV_KMODS       += ipsecah
256 DRV_KMODS       += ipsecesp
257 DRV_KMODS       += ipw
258 DRV_KMODS       += iwh
259 DRV_KMODS       += iwi
```

```
260 DRV_KMODS       += iwk
261 DRV_KMODS       += iwp
262 DRV_KMODS       += iwscn
263 DRV_KMODS       += kb8042
264 DRV_KMODS       += keysock
265 DRV_KMODS       += kssl
266 DRV_KMODS       += kstat
267 DRV_KMODS       += ksyms
268 DRV_KMODS       += kmdb
269 DRV_KMODS       += llc1
270 DRV_KMODS       += lofi
271 DRV_KMODS       += log
272 DRV_KMODS       += logindmux
273 DRV_KMODS       += mega_sas
274 DRV_KMODS       += mc-amd
275 DRV_KMODS       += mm
276 DRV_KMODS       += mouse8042
277 DRV_KMODS       += mpt_sas
278 DRV_KMODS       += mr_sas
279 DRV_KMODS       += mwl
280 DRV_KMODS       += nca
281 DRV_KMODS       += nsmb
282 DRV_KMODS       += nulldriver
283 DRV_KMODS       += nv_sata
284 DRV_KMODS       += nxge
285 DRV_KMODS       += oce
286 DRV_KMODS       += openeepr
287 DRV_KMODS       += pci_pci
288 DRV_KMODS       += pcic
289 DRV_KMODS       += pcieb
290 DRV_KMODS       += physmem
291 DRV_KMODS       += pcan
292 DRV_KMODS       += pcwl
293 DRV_KMODS       += pit_beep
294 DRV_KMODS       += pm
295 DRV_KMODS       += poll
296 DRV_KMODS       += pool
297 DRV_KMODS       += power
298 DRV_KMODS       += pseudo
299 DRV_KMODS       += ptc
300 DRV_KMODS       += ptm
301 DRV_KMODS       += pts
302 DRV_KMODS       += ptsl
303 DRV_KMODS       += qlge
304 DRV_KMODS       += radeon
305 DRV_KMODS       += ral
306 DRV_KMODS       += ramdisk
307 DRV_KMODS       += random
308 DRV_KMODS       += rds
309 DRV_KMODS       += rdsv3
310 DRV_KMODS       += rpcib
311 DRV_KMODS       += rsm
312 DRV_KMODS       += rts
313 DRV_KMODS       += rtw
314 DRV_KMODS       += rum
315 DRV_KMODS       += rwd
316 DRV_KMODS       += rwn
317 DRV_KMODS       += sad
318 DRV_KMODS       += sd
319 DRV_KMODS       += sdhost
320 DRV_KMODS       += sgen
321 DRV_KMODS       += si3124
322 DRV_KMODS       += smbios
323 DRV_KMODS       += softmac
324 DRV_KMODS       += spdsock
325 DRV_KMODS       += smbsrv
```



```
326 DRV_KMODS       += smp
327 DRV_KMODS       += sppp
328 DRV_KMODS       += sppptun
329 DRV_KMODS       += srpt
330 DRV_KMODS       += st
331 DRV_KMODS       += sy
332 DRV_KMODS       += sysevent
333 DRV_KMODS       += sysmsg
334 DRV_KMODS       += tcp
335 DRV_KMODS       += tcp6
336 DRV_KMODS       += tl
337 DRV_KMODS       += tnf
338 DRV_KMODS       += tpm
339 DRV_KMODS       += trill
340 DRV_KMODS       += udp
341 DRV_KMODS       += udp6
342 DRV_KMODS       += ucode
343 DRV_KMODS       += ural
344 DRV_KMODS       += uath
345 DRV_KMODS       += urtw
346 DRV_KMODS       += vgatext
347 DRV_KMODS       += heci
348 DRV_KMODS       += vnic
349 DRV_KMODS       += vscan
350 DRV_KMODS       += wc
351 DRV_KMODS       += winlock
352 DRV_KMODS       += wpi
353 DRV_KMODS       += xge
354 DRV_KMODS       += yge
355 DRV_KMODS       += zcons
356 DRV_KMODS       += zyd
357 DRV_KMODS       += simnet
358 DRV_KMODS       += stmf
359 DRV_KMODS       += stmf_sbd
360 DRV_KMODS       += fct
361 DRV_KMODS       += fcoe
362 DRV_KMODS       += fcoet
363 DRV_KMODS       += fcoei
364 DRV_KMODS       += qlt
365 DRV_KMODS       += iscsit
366 DRV_KMODS       += pppt
367 DRV_KMODS       += ncall nsctl sdbc nskern sv
368 DRV_KMODS       += ii rdc rdcsrv rdcstub
369 DRV_KMODS       += iptun

371 $(CLOSED_BUILD)CLOSED_DRV_KMODS         += bmc
372 $(CLOSED_BUILD)CLOSED_DRV_KMODS         += glm
373 $(CLOSED_BUILD)CLOSED_DRV_KMODS         += intel_nhmex
374 $(CLOSED_BUILD)CLOSED_DRV_KMODS         += cpqary3
375 $(CLOSED_BUILD)CLOSED_DRV_KMODS         += marvell88sx
376 $(CLOSED_BUILD)CLOSED_DRV_KMODS         += bcm_sata
377 $(CLOSED_BUILD)CLOSED_DRV_KMODS         += memtest
378 $(CLOSED_BUILD)CLOSED_DRV_KMODS         += mpt
379 $(CLOSED_BUILD)CLOSED_DRV_KMODS         += atiatom
380 $(CLOSED_BUILD)CLOSED_DRV_KMODS         += acpi_toshiba

382 #
383 # Common code drivers
384 #

386 DRV_KMODS       += afe
387 DRV_KMODS       += atge
388 DRV_KMODS       += bfe
389 DRV_KMODS       += dmfe
390 DRV_KMODS       += e1000g
391 DRV_KMODS       += efe
```

```
392 DRV_KMODS       += elxl
393 DRV_KMODS       += hme
394 DRV_KMODS       += mxfe
395 DRV_KMODS       += nge
396 DRV_KMODS       += pcn
397 DRV_KMODS       += rge
398 DRV_KMODS       += rtls
399 DRV_KMODS       += sfe
400 DRV_KMODS       += amd8111s
401 DRV_KMODS       += igb
402 DRV_KMODS       += ipmi
403 DRV_KMODS       += iprb
404 DRV_KMODS       += ixgbe
405 DRV_KMODS       += sfxge
406 #endif /* ! codereview */
407 DRV_KMODS       += vr
408 $(CLOSED_BUILD)CLOSED_DRV_KMODS += ixgb

410 #
411 # Virtio drivers
412 #

414 # Virtio core
415 DRV_KMODS       += virtio

417 # Virtio block driver
418 DRV_KMODS       += vioblk

420 #
421 #       DTrace and DTrace Providers
422 #
423 DRV_KMODS       += dtrace
424 DRV_KMODS       += fbt
425 DRV_KMODS       += lockstat
426 DRV_KMODS       += profile
427 DRV_KMODS       += sdt
428 DRV_KMODS       += systrace
429 DRV_KMODS       += fasttrap
430 DRV_KMODS       += dcpc

432 #
433 #       I/O framework test drivers
434 #
435 DRV_KMODS       += pshot
436 DRV_KMODS       += gen_drv
437 DRV_KMODS       += tvhci tphci tclient
438 DRV_KMODS       += emul64

440 #
441 #       Machine Specific Driver Modules (/kernel/drv):
442 #
443 DRV_KMODS       += options
444 DRV_KMODS       += scsi_vhci
445 DRV_KMODS       += pmcs
446 DRV_KMODS       += pmcs8001fw
447 DRV_KMODS       += arcmsr
448 DRV_KMODS       += fcp
449 DRV_KMODS       += fcip
450 DRV_KMODS       += fcsm
451 DRV_KMODS       += fp
452 DRV_KMODS       += qlc
453 DRV_KMODS       += iscsi

455 #
456 #       PCMCIA specific module(s)
457 #
```



```
458 DRV_KMODS       += pcs
459 DRV_KMODS       += pcata
460 MISC_KMODS      += cardbus
461 $(CLOSED_BUILD)CLOSED_DRV_KMODS += pcser

463 #
464 #       SCSI Enclosure Services driver
465 #
466 DRV_KMODS       += ses

468 #
469 #       USB specific modules
470 #
471 DRV_KMODS       += hid
472 DRV_KMODS       += hwarc hwahc
473 DRV_KMODS       += hubd
474 DRV_KMODS       += uhci
475 DRV_KMODS       += ehci
476 DRV_KMODS       += ohci
477 DRV_KMODS       += usb_mid
478 DRV_KMODS       += usb_ia
479 DRV_KMODS       += scsa2usb
480 DRV_KMODS       += usbprn
481 DRV_KMODS       += ugen
482 DRV_KMODS       += usbser
483 DRV_KMODS       += usbsacm
484 DRV_KMODS       += usbsksp
485 DRV_KMODS       += usbsprl
486 DRV_KMODS       += usb_ac
487 DRV_KMODS       += usb_as
488 DRV_KMODS       += usbskel
489 DRV_KMODS       += usbvc
490 DRV_KMODS       += usbftdi
491 DRV_KMODS       += wusb_df
492 DRV_KMODS       += wusb_ca
493 DRV_KMODS       += usbecm

495 $(CLOSED_BUILD)CLOSED_DRV_KMODS += usbser_edge

497 #
498 #       1394 modules
499 #
500 MISC_KMODS      += s1394 sbp2
501 DRV_KMODS       += hci1394 scsa1394
502 DRV_KMODS       += av1394
503 DRV_KMODS       += dcam1394

505 #
506 #       InfiniBand pseudo drivers
507 #
508 DRV_KMODS       += ib ibp eibnx eoib rdsib sdp iser daplt hermon tavor sol_ucma
509 DRV_KMODS       += sol_umad

511 #
512 #       LVM modules
513 #
514 DRV_KMODS       += md
515 MISC_KMODS      += md_stripe md_hotspares md_mirror md_raid md_trans md_notify
516 MISC_KMODS      += md_sp

518 #
519 #       Brand modules
520 #
521 BRAND_KMODS     += sn1_brand s10_brand

523 #
```

```
#       Exec Class Modules (/kernel/exec):
#
EXEC_KMODS      += elfexec intpexec shbinexec javaexec

#
#       Scheduling Class Modules (/kernel/sched):
#
SCHED_KMODS     += IA RT TS RT_DPTBL TS_DPTBL FSS FX FX_DPTBL SDC

#
#       File System Modules (/kernel/fs):
#
FS_KMODS        += autofs cachefs ctfs dcfs dev devfs fdfs fifofs hsfs lofs
FS_KMODS        += mntfs namefs nfs objfs zfs zut
FS_KMODS        += pcfs procfs sockfs specfs tmpfs udfs ufs sharefs
FS_KMODS        += smbfs

#
#       Streams Modules (/kernel/strmod):
#
STRMOD_KMODS    += bufmod connld dedump ldterm pckt pfmod pipemod
STRMOD_KMODS    += ptem redirmod rpcmod rlmod telmod timod
STRMOD_KMODS    += spppasyn spppcomp
STRMOD_KMODS    += tirdwr ttcompat
STRMOD_KMODS    += usbkbm
STRMOD_KMODS    += usbms
STRMOD_KMODS    += usbwcm
STRMOD_KMODS    += usb_ah
STRMOD_KMODS    += drcompat
STRMOD_KMODS    += cryptmod
STRMOD_KMODS    += vuid2ps2
STRMOD_KMODS    += vuid3ps2
STRMOD_KMODS    += vuidm3p
STRMOD_KMODS    += vuidm4p
STRMOD_KMODS    += vuidm5p

#
#       'System' Modules (/kernel/sys):
#
SYS_KMODS       += c2audit
SYS_KMODS       += doorfs
SYS_KMODS       += exacctsys
SYS_KMODS       += inst_sync
SYS_KMODS       += kaio
SYS_KMODS       += msgsys
SYS_KMODS       += pipe
SYS_KMODS       += portfs
SYS_KMODS       += pset
SYS_KMODS       += semsys
SYS_KMODS       += shmsys
SYS_KMODS       += sysacct
SYS_KMODS       += acctctl

#
#       'Misc' Modules (/kernel/misc)
#       MISC_KMODS are built both 32-bit and 64-bit
#       MISC_KMODS_32 are built only 32-bit
#       MISC_KMODS_64 are built only 64-bit
#
MISC_KMODS      += ac97
MISC_KMODS      += acpica
MISC_KMODS      += agpmaster
MISC_KMODS      += bignum
MISC_KMODS      += bootdev
MISC_KMODS      += busra
MISC_KMODS      += cmlb
```



```
MISC_KMODS      += consconfig
MISC_KMODS      += ctf
MISC_KMODS      += dadk
MISC_KMODS      += dcopy
MISC_KMODS      += dls
MISC_KMODS      += drm
MISC_KMODS      += fssnap_if
MISC_KMODS      += gda
MISC_KMODS      += gld
MISC_KMODS      += hidparser
MISC_KMODS      += hook
MISC_KMODS      += hpcsvc
MISC_KMODS      += ibcm
MISC_KMODS      += ibdm
MISC_KMODS      += ibdma
MISC_KMODS      += ibmf
MISC_KMODS      += ibtl
MISC_KMODS      += idm
MISC_KMODS      += idmap
MISC_KMODS      += iommulib
MISC_KMODS      += ipc
MISC_KMODS      += kbtrans
MISC_KMODS      += kcf
MISC_KMODS      += kgssapi
MISC_KMODS      += kmech_dummy
MISC_KMODS      += kmech_krb5
MISC_KMODS      += ksocket
MISC_KMODS      += mac
MISC_KMODS      += mii
MISC_KMODS      += mwlfw
MISC_KMODS      += net80211
MISC_KMODS      += nfs_dlboot
MISC_KMODS      += nfssrv
MISC_KMODS      += neti
MISC_KMODS      += pci_autoconfig
MISC_KMODS      += pcicfg
MISC_KMODS      += pcihp
MISC_KMODS      += pcmcia
MISC_KMODS      += rpcsec
MISC_KMODS      += rpcsec_gss
MISC_KMODS      += rsmops
MISC_KMODS      += sata
MISC_KMODS      += scsi
MISC_KMODS      += sda
MISC_KMODS      += sol_ofs
MISC_KMODS      += spuni
MISC_KMODS      += strategy
MISC_KMODS      += strplumb
MISC_KMODS      += tem
MISC_KMODS      += tlimod
MISC_KMODS      += usba usba10 usbs49_fw
MISC_KMODS      += scsi_vhci_f_sym_hds
MISC_KMODS      += scsi_vhci_f_sym
MISC_KMODS      += scsi_vhci_f_tpgs
MISC_KMODS      += scsi_vhci_f_asym_sun
MISC_KMODS      += scsi_vhci_f_tape
MISC_KMODS      += scsi_vhci_f_tpgs_tape
MISC_KMODS      += fctl
MISC_KMODS      += emlxs_fw
MISC_KMODS      += qlc_fw_2200
MISC_KMODS      += qlc_fw_2300
MISC_KMODS      += qlc_fw_2400
MISC_KMODS      += qlc_fw_2500
MISC_KMODS      += qlc_fw_6322
MISC_KMODS      += qlc_fw_8100
MISC_KMODS      += hwa1480_fw
```

```
MISC_KMODS      += uathfw
MISC_KMODS      += uwba

$(CLOSED_BUILD)CLOSED_MISC_KMODS        += klmmod klmops
$(CLOSED_BUILD)CLOSED_MISC_KMODS        += scsi_vhci_f_asym_lsi
$(CLOSED_BUILD)CLOSED_MISC_KMODS        += scsi_vhci_f_asym_emc
$(CLOSED_BUILD)CLOSED_MISC_KMODS        += scsi_vhci_f_sym_emc

#
#       Software Cryptographic Providers (/kernel/crypto):
#
CRYPTO_KMODS    += aes
CRYPTO_KMODS    += arcfour
CRYPTO_KMODS    += blowfish
CRYPTO_KMODS    += des
CRYPTO_KMODS    += ecc
CRYPTO_KMODS    += md4
CRYPTO_KMODS    += md5
CRYPTO_KMODS    += rsa
CRYPTO_KMODS    += sha1
CRYPTO_KMODS    += sha2
CRYPTO_KMODS    += swrand

#
#       IP Policy Modules (/kernel/ipp)
#
IPP_KMODS       += dlcosmk
IPP_KMODS       += flowacct
IPP_KMODS       += ipgpc
IPP_KMODS       += dscpmk
IPP_KMODS       += tokenmt
IPP_KMODS       += tswtclmt

#
#       generic-unix module (/kernel/genunix):
#
GENUNIX_KMODS   += genunix

#
#       SVVS Testing Modules (/kernel/strmod):
#
#       These are streams and driver modules which are not to be
#       delivered with a released system. However, during development
#       it is convenient to build and install the SVVS kernel modules.
#
SVVS_KMODS      += lmodb lmode lmodr lmodt svvslo tidg tivc tmux

$(CLOSED_BUILD)SVVS             += svvs

#
#       Modules eXcluded from the product:
#
$(CLOSED_BUILD)CLOSED_XMODS =           \
        adpu320         \
        bnx             \
        bnxe            \
        lsimega         \
        sdpib


#
#       'Dacf' Modules (/kernel/dacf):
#

#
#       Performance Counter BackEnd modules (/usr/kernel/pcbe)
```



```
#
PCBE_KMODS      += p123_pcbe p4_pcbe opteron_pcbe core_pcbe

#
#       MAC-Type Plugin Modules (/kernel/mac)
#
MAC_KMODS       += mac_6to4
MAC_KMODS       += mac_ether
MAC_KMODS       += mac_ipv4
MAC_KMODS       += mac_ipv6
MAC_KMODS       += mac_wifi
MAC_KMODS       += mac_ib

#
# socketmod (kernel/socketmod)
#
SOCKET_KMODS    += sockpfp
SOCKET_KMODS    += socksctp
SOCKET_KMODS    += socksdp
SOCKET_KMODS    += sockrds
SOCKET_KMODS    += ksslf

#
#       kiconv modules (/kernel/kiconv):
#
KICONV_KMODS    += kiconv_emea kiconv_ja kiconv_ko kiconv_sc kiconv_tc

#
#       'Dacf' Modules (/kernel/dacf):
#
DACF_KMODS      += net_dacf
```

```
**********************************************************
    2418 Thu Aug 22 18:59:30 2013
new/usr/src/uts/intel/sfxge/Makefile
Merged sfxge driver
**********************************************************
```

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
#
# Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
# uts/intel/sfxge/Makefile
#
#       This makefile drives the production of the sfxge
#       network driver kernel module.
#
#       intel architecture dependent
#


#
#       Paths to the base of the uts directory trees
#
UTSBASE = ../..

#
#       Define the module and object file sets.
#
MODULE          = sfxge
OBJECTS         = $(SFXGE_OBJS:%=$(OBJS_DIR)/%)
LINTS           = $(SFXGE_OBJS:%.o=$(LINTS_DIR)/%.ln)
ROOTMODULE      = $(ROOT_DRV_DIR)/$(MODULE)
CONF_SRCDIR     = $(UTSBASE)/common/io/sfxge

#
#       Include common rules.
#
include $(UTSBASE)/intel/Makefile.intel

CERRWARN        += -_gcc=-Wno-parentheses
CERRWARN        += -_gcc=-Wno-switch
CERRWARN        += -_gcc=-Wno-uninitialized

#
#       Define targets
#
ALL_TARGET      = $(BINARY) $(CONFMOD)
LINT_TARGET     = $(MODULE).lint
INSTALL_TARGET  = $(BINARY) $(ROOTMODULE) $(ROOT_CONFFILE)
```



```
#
# Driver depends on MAC and IP
#
LDFLAGS         += -dy -N misc/mac -N drv/ip

#
# CFLAGS for Sol11-based target
#
CFLAGS    += -D_USE_DESBALLOC
CFLAGS    += -U_USE_XESBALLOC
CFLAGS    += -U_USE_CPU_PHYSID
CFLAGS    += -U_USE_GLD_V2
CFLAGS    += -D_USE_GLD_V3
CFLAGS    += -D_USE_GLD_V3_SOL10
CFLAGS    += -D_USE_GLD_V3_SOL11
CFLAGS    += -D_USE_MAC_PRIV_PROP
CFLAGS    += -D_USE_GLD_V3_PROPS
CFLAGS    += -U_USE_MTU_UPDATE
CFLAGS    += -D_USE_NDD_PROPS


#
#       Default build targets.
#
.KEEP_STATE:

def:            $(DEF_DEPS)

all:            $(ALL_DEPS)

clean:          $(CLEAN_DEPS)

clobber:        $(CLOBBER_DEPS)

lint:           $(LINT_DEPS)

modlintlib:     $(MODLINTLIB_DEPS)

clean.lint:     $(CLEAN_LINT_DEPS)

install:        $(INSTALL_DEPS)

#
#       Include common targets.
#
include $(UTSBASE)/intel/Makefile.targ
#endif /* ! codereview */
```