```
**********************************************************
    1862 Thu Jun 12 17:42:13 2014
new/usr/src/cmd/mdb/Makefile.common
Add mpt_sas3 to (k)mdb make list.
**********************************************************
```

```
#
# 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) 1999, 2010, Oracle and/or its affiliates. All rights reserved.
#
# MDB modules used for debugging user processes that every ISA's build
# subdirectory will need to build.
#

COMMON_MODULES_PROC = \
        dof \
        libavl \
        libc \
        libcmdutils \
        libnvpair \
        libproc \
        libpython2.6 \
        libsysevent \
        libtopo \
        libumem \
        libuutil \
        libzpool \
        mdb_ds \
        mdb_test

#
# MDB modules used for debugging user processes which are only 32-bit
#
COMMON_MODULES_PROC_32BIT = \
        svc.configd \
        svc.startd

#
# MDB modules used for debugging kernels.
#
COMMON_MODULES_KVM = \
        arp \
        cpc \
        crypto  \
        dtrace  \
        emlxs \
        fcip \
        fcp \
        fctl \
```



```
        genunix \
        hook \
        neti \
        idm \
        ii \
        ip \
        ipc \
        ipp \
        krtld \
        lofs \
        logindmux \
        mac \
        md \
        mpt_sas \
        mpt_sas3 \
#endif /* ! codereview */
        mr_sas \
        nca \
        nsctl \
        nsmb \
        pmcs \
        ptm \
        qlc \
        random \
        rdc \
        s1394 \
        scsi_vhci \
        sctp \
        sd \
        sdbc \
        smbfs \
        smbsrv \
        sockfs \
        specfs \
        sppp \
        srpt \
        stmf \
        stmf_sbd \
        sv \
        ufs \
        usba \
        zfs

CLOSED_COMMON_MODULES_KVM = \
        mpt \
        nfs

include $(SRC)/Makefile.master
```

```
**********************************************************
   22887 Thu Jun 12 17:42:13 2014
new/usr/src/cmd/mdb/common/modules/mpt_sas/mpt_sas.c
Merge fixes for Illumos issue 4819, fix mpt_sas command timeout handling.
Add rolling buffer for *all* debug messages.
Improve mdb module and seperate out into mpt_sas3.
**********************************************************
______unchanged_portion_omitted_

 152 void
 153 mdi_info(struct mptsas *mp, int target)
 153 mdi_info(struct mptsas m, int target)
 154 {
 155         struct  dev_info        d;
 156         struct  mdi_phci        p;

 158         if (mdb_vread(&d, sizeof (d), (uintptr_t)mp->m_dip) == -1) {
 158         if (mdb_vread(&d, sizeof (d), (uintptr_t)m.m_dip) == -1) {
 159                 mdb_warn("couldn't read m_dip");
 160                 return;
 161         }

 163         if (MDI_PHCI(&d)) {
 164                 if (mdb_vread(&p, sizeof (p), (uintptr_t)d.devi_mdi_xhci)
 165                     == -1) {
 166                         mdb_warn("couldn't read m_dip.devi_mdi_xhci");
 167                         return;
 168                 }
 169                 if (p.ph_path_head)
 170                         mdb_pwalk("mdipi_phci_list", (mdb_walk_cb_t)mdi_info_cb,
 171                             &target, (uintptr_t)p.ph_path_head);
 172                 return;
 173         }
 174 }
______unchanged_portion_omitted_

 303 void
 304 display_targets(struct mptsas *mp, uint_t verbose)
 304 display_targets(struct mptsas *mp)
 305 {
 306         mptsas_target_t *ptgt;
 307         mptsas_smp_t *psmp;
 308         int loop, comma;
 309 #endif /* ! codereview */

 311         mdb_printf("\n");
 312         mdb_printf("slot devhdl      wwn     ncmds throttle   "
 313             "dr_flag  dups\n");
 314         mdb_printf("------------------------------"
 315             "------------------------------\n");
 316         for (ptgt = (mptsas_target_t *)krefhash_first(
 317             (uintptr_t)mp->m_targets);
 308         mdb_printf("The SCSI target information\n");
 309         for (ptgt = (mptsas_target_t *)krefhash_first((uintptr_t)mp->m_targets);
 318             ptgt != NULL;
 319             ptgt = krefhash_next((uintptr_t)mp->m_targets, ptgt)) {
 320                 if (ptgt->m_addr.mta_wwn ||
 321                     ptgt->m_deviceinfo) {
 322                         mdb_printf("%4d ", ptgt->m_slot_num);
 323                         mdb_printf("%4d ", ptgt->m_devhdl);
 324                         if (ptgt->m_addr.mta_wwn)
 312                 mdb_printf("\n");
 313                 mdb_printf("devhdl %x, sasaddress %"PRIx64", phymask %x,"
 314                     "devinfo %x\n", ptgt->m_devhdl, ptgt->m_addr.mta_wwn,
 315                     ptgt->m_addr.mta_phymask, ptgt->m_deviceinfo);
 316                 mdb_printf("throttle %x, dr_flag %x, m_t_ncmds %x, "
```



```
 317                     "enclosure %x, slot_num %x\n", ptgt->m_t_throttle,
 318                     ptgt->m_dr_flag, ptgt->m_t_ncmds, ptgt->m_enclosure,
 319                     ptgt->m_slot_num);
 320         }

 322         mdb_printf("\n");
 323         mdb_printf("The smp child information\n");
 324         for (psmp = (mptsas_smp_t *)krefhash_first(
 325             (uintptr_t)mp->m_smp_targets);
 326             psmp != NULL;
 327             psmp = krefhash_next((uintptr_t)mp->m_smp_targets, psmp)) {
 328                 mdb_printf("\n");
 329                 mdb_printf("devhdl %x, sasaddress %"PRIx64", phymask %x \n",
 330                     psmp->m_devhdl, psmp->m_addr.mta_wwn,
 331                     psmp->m_addr.mta_phymask);
 332         }
 333         mdb_printf("\n");
 334 #if 0
 335         mdb_printf("targ         wwn      ncmds throttle "
 336             "dr_flag  timeout  dups\n");
 337         mdb_printf("-------------------------------"
 338             "-------------------------------\n");
 339         for (i = 0; i < MPTSAS_MAX_TARGETS; i++) {
 340                 if (s->m_target[i].m_addr.mta_wwn ||
 341                     s->m_target[i].m_deviceinfo) {
 342                         mdb_printf("%4d ", i);
 343                         if (s->m_target[i].m_addr.mta_wwn)
 325                                 mdb_printf("%"PRIx64" ",
 326                                     ptgt->m_addr.mta_wwn);
 327                         mdb_printf("%3d", ptgt->m_t_ncmds);
 328                         switch (ptgt->m_t_throttle) {
 345                                     s->m_target[i].m_addr.mta_wwn);
 346                         mdb_printf("%3d", s->m_target[i].m_t_ncmds);
 347                         switch (s->m_target[i].m_t_throttle) {
 329                                 case QFULL_THROTTLE:
 330                                         mdb_printf("   QFULL ");
 331                                         break;
 332                                 case DRAIN_THROTTLE:
 333                                         mdb_printf("   DRAIN ");
 334                                         break;
 335                                 case HOLD_THROTTLE:
 336                                         mdb_printf("    HOLD ");
 337                                         break;
 338                                 case MAX_THROTTLE:
 339                                         mdb_printf("     MAX ");
 340                                         break;
 360                                 case CHOKE_THROTTLE:
 361                                         mdb_printf("   CHOKE ");
 362                                         break;
 341                                 default:
 342                                         mdb_printf("%8d ",
 343                                             ptgt->m_t_throttle);
 365                                             s->m_target[i].m_t_throttle);
 344                         }
 345                         switch (ptgt->m_dr_flag) {
 367                         switch (s->m_target[i].m_dr_flag) {
 346                                 case MPTSAS_DR_INACTIVE:
 347                                         mdb_printf("  INACTIVE ");
 348                                         break;
 349                                 case MPTSAS_DR_INTRANSITION:
 350                                         mdb_printf("TRANSITION ");
 371                                 case MPTSAS_DR_PRE_OFFLINE_TIMEOUT:
 372                                         mdb_printf("   TIMEOUT ");
 373                                         break;
 374                                 case MPTSAS_DR_PRE_OFFLINE_TIMEOUT_NO_CANCEL:
 375                                         mdb_printf("TIMEOUT_NC ");
```

```
376                                     break;
377                             case MPTSAS_DR_OFFLINE_IN_PROGRESS:
378                                     mdb_printf(" OFFLINING ");
379                                     break;
380                             case MPTSAS_DR_ONLINE_IN_PROGRESS:
381                                     mdb_printf("  ONLINING ");
351                                     break;
352                             default:
353                                     mdb_printf("   UNKNOWN ");
354                                     break;
355                             }
356                     mdb_printf("%d\n", ptgt->m_dups);
387                     mdb_printf("%3d/%-3d   %d/%d\n",
388                         s->m_target[i].m_dr_timeout, m.m_offline_delay,
389                         s->m_target[i].m_dr_online_dups,
390                         s->m_target[i].m_dr_offline_dups);

358                     if (verbose) {
359                             mdb_inc_indent(5);
360                             if ((ptgt->m_deviceinfo &
394                             if ((s->m_target[i].m_deviceinfo &
361                                 MPI2_SAS_DEVICE_INFO_MASK_DEVICE_TYPE) ==
362                                 MPI2_SAS_DEVICE_INFO_FANOUT_EXPANDER)
363                                     mdb_printf("Fanout expander: ");
364                             if ((ptgt->m_deviceinfo &
398                             if ((s->m_target[i].m_deviceinfo &
365                                 MPI2_SAS_DEVICE_INFO_MASK_DEVICE_TYPE) ==
366                                 MPI2_SAS_DEVICE_INFO_EDGE_EXPANDER)
367                                     mdb_printf("Edge expander: ");
368                             if ((ptgt->m_deviceinfo &
402                             if ((s->m_target[i].m_deviceinfo &
369                                 MPI2_SAS_DEVICE_INFO_MASK_DEVICE_TYPE) ==
370                                 MPI2_SAS_DEVICE_INFO_END_DEVICE)
371                                     mdb_printf("End device: ");
372                             if ((ptgt->m_deviceinfo &
406                             if ((s->m_target[i].m_deviceinfo &
373                                 MPI2_SAS_DEVICE_INFO_MASK_DEVICE_TYPE) ==
374                                 MPI2_SAS_DEVICE_INFO_NO_DEVICE)
375                                     mdb_printf("No device ");

377                             for (loop = 0, comma = 0;
378                                 loop < (sizeof (devinfo_array) /
379                                 sizeof (devinfo_array[0])); loop++) {
380                                     if (ptgt->m_deviceinfo &
414                                     if (s->m_target[i].m_deviceinfo &
381                                         devinfo_array[loop].value) {
382                                             mdb_printf("%s%s",
383                                                 (comma ? ", " : ""),
384                                                 devinfo_array[loop].text);
385                                             comma++;
386                                     }
387                             }
388                             mdb_printf("\n");
389 #if 0
390                             if (ptgt->m_tgt_dip) {
391                                     char    target_path[PATH_MAX];

424                             if (s->m_target[i].m_tgt_dip) {
392                                     *target_path = 0;
393                                     if (construct_path((uintptr_t)
394                                         ptgt->m_tgt_dip,
427                                         s->m_target[i].m_tgt_dip,
395                                         target_path)
396                                         == DCMD_OK)
397                                             mdb_printf("%s\n", target_path);
398                             }
```



```
399 #endif
400                                     mdi_info(mp, ptgt->m_slot_num);
432                                     mdi_info(m, i);
401                                     mdb_dec_indent(5);
402                             }
403                     }
404             }

406             mdb_printf("\n");
407             mdb_printf("The smp child information\n");
408             for (psmp = (mptsas_smp_t *)krefhash_first(
409                 (uintptr_t)mp->m_smp_targets);
410                 psmp != NULL;
411                 psmp = krefhash_next((uintptr_t)mp->m_smp_targets, psmp)) {
412                     mdb_printf("\n");
413                     mdb_printf("devhdl %x, sasaddress %"PRIx64", phymask %x \n",
414                         psmp->m_devhdl, psmp->m_addr.mta_wwn,
415                         psmp->m_addr.mta_phymask);
416             }
437 #endif
417 }

419 int
420 display_slotinfo(struct mptsas *mp, struct mptsas_slots *s)
441 display_slotinfo()
421 {
443 #if 0
422             int                     i, nslots;
423             struct mptsas_cmd       c, *q, *slots;
424             mptsas_target_t         *ptgt;
425 #endif /* ! codereview */
426             int                     header_output = 0;
427             int                     rv = DCMD_OK;
428             int                     slots_in_use = 0;
429             int                     tcmds = 0;
430             int                     mismatch = 0;
431             int                     wq, dq;
432             int                     ncmds = 0;
433             ulong_t                 saved_indent;

435             nslots = s->m_n_normal;

436             slots = mdb_alloc(sizeof (mptsas_cmd_t) * nslots, UM_SLEEP);

438             for (i = 0; i < nslots; i++)
439                     if (s->m_slot[i]) {
440                             slots_in_use++;
441                             if (mdb_vread(&slots[i], sizeof (mptsas_cmd_t),
442                                 (uintptr_t)s->m_slot[i]) == -1) {
443                                     mdb_warn("couldn't read slot");
444                                     s->m_slot[i] = NULL;
445                             }
446                             if ((slots[i].cmd_flags & CFLAG_CMDIOC) == 0)
447                                     tcmds++;
448                             if (i != slots[i].cmd_slot)
449                                     mismatch++;
450                     }

452             for (q = mp->m_waitq, wq = 0; q; q = c.cmd_linkp, wq++)
463             for (q = m.m_waitq, wq = 0; q; q = c.cmd_linkp, wq++)
453                     if (mdb_vread(&c, sizeof (mptsas_cmd_t), (uintptr_t)q) == -1) {
454                             mdb_warn("couldn't follow m_waitq");
455                             rv = DCMD_ERR;
456                             goto exit;
457                     }
```

```
459         for (q = mp->m_doneq, dq = 0; q; q = c.cmd_linkp, dq++)
470         for (q = m.m_doneq, dq = 0; q; q = c.cmd_linkp, dq++)
460                 if (mdb_vread(&c, sizeof (mptsas_cmd_t), (uintptr_t)q) == -1) {
461                         mdb_warn("couldn't follow m_doneq");
462                         rv = DCMD_ERR;
463                         goto exit;
464                 }

466         for (ptgt = (mptsas_target_t *)krefhash_first(
467             (uintptr_t)mp->m_targets);
468             ptgt != NULL;
469             ptgt = krefhash_next((uintptr_t)mp->m_targets, ptgt)) {
470                 if (ptgt->m_addr.mta_wwn ||
471                     ptgt->m_deviceinfo) {
472                         ncmds += ptgt->m_t_ncmds;
473                 }
474         }
477         for (i = 0; i < MPTSAS_MAX_TARGETS; i++)
478                 ncmds += s->m_target[i].m_t_ncmds;

476         mdb_printf("\n");
477         mdb_printf("   mpt.  slot              mptsas_slots     slot");
478         mdb_printf("\n");
479         mdb_printf("m_ncmds total"
480             " targ throttle m_t_ncmds targ_tot wq dq");
481         mdb_printf("\n");
482         mdb_printf("-----------------------------------------------------");
483         mdb_printf("\n");

485         mdb_printf("%7d ", mp->m_ncmds);
486         mdb_printf("%s", (mp->m_ncmds == slots_in_use ? "  " : "!="));
489         mdb_printf("%7d ", m.m_ncmds);
490         mdb_printf("%s", (m.m_ncmds == slots_in_use ? "  " : "!="));
487         mdb_printf("%3d              total %3d ", slots_in_use, ncmds);
488         mdb_printf("%s", (tcmds == ncmds ? "     " : "   !="));
489         mdb_printf("%3d %2d %2d\n", tcmds, wq, dq);

491         saved_indent = mdb_dec_indent(0);
492         mdb_dec_indent(saved_indent);

494         for (i = 0; i < s->m_n_normal; i++)
495                 if (s->m_slot[i]) {
496                         if (!header_output) {
497                                 mdb_printf("\n");
498                                 mdb_printf("mptsas_cmd          slot cmd_slot "
499                                     "cmd_flags cmd_pkt_flags scsi_pkt      "
500                                     "  targ,lun [ pkt_cdbp ...\n");
501                                 mdb_printf("--------------------------------"
502                                     "--------------------------------------"
503                                     "--------------------------------------"
504                                     "------\n");
505                                 header_output = 1;
506                         }
507                         mdb_printf("%16p %4d %s %4d  %8x      %8x %16p ",
508                             s->m_slot[i], i,
509                             (i == slots[i].cmd_slot?"   ":"BAD"),
510                             slots[i].cmd_slot,
511                             slots[i].cmd_flags,
512                             slots[i].cmd_pkt_flags,
513                             slots[i].cmd_pkt);
514                         (void) print_cdb(&slots[i]);
515                 }

517         /* print the wait queue */

519         for (q = mp->m_waitq; q; q = c.cmd_linkp) {
```



```
520                 if (q == mp->m_waitq)
523         for (q = m.m_waitq; q; q = c.cmd_linkp) {
524                 if (q == m.m_waitq)
521                         mdb_printf("\n");
522                 if (mdb_vread(&c, sizeof (mptsas_cmd_t), (uintptr_t)q)
523                     == -1) {
524                         mdb_warn("couldn't follow m_waitq");
525                         rv = DCMD_ERR;
526                         goto exit;
527                 }
528                 mdb_printf("%16p wait n/a %4d  %8x      %8x %16p ",
529                     q, c.cmd_slot, c.cmd_flags, c.cmd_pkt_flags,
530                     c.cmd_pkt);
531                 print_cdb(&c);
532         }

534         /* print the done queue */

536         for (q = mp->m_doneq; q; q = c.cmd_linkp) {
537                 if (q == mp->m_doneq)
540         for (q = m.m_doneq; q; q = c.cmd_linkp) {
541                 if (q == m.m_doneq)
538                         mdb_printf("\n");
539                 if (mdb_vread(&c, sizeof (mptsas_cmd_t), (uintptr_t)q)
540                     == -1) {
541                         mdb_warn("couldn't follow m_doneq");
542                         rv = DCMD_ERR;
543                         goto exit;
544                 }
545                 mdb_printf("%16p done  n/a %4d  %8x      %8x %16p ",
546                     q, c.cmd_slot, c.cmd_flags, c.cmd_pkt_flags,
547                     c.cmd_pkt);
548                 print_cdb(&c);
549         }

551         mdb_inc_indent(saved_indent);

553         if (mp->m_ncmds != slots_in_use)
557         if (m.m_ncmds != slots_in_use)
554                 mdb_printf("WARNING: mpt.m_ncmds does not match the number of "
555                     "slots in use\n");

557         if (tcmds != ncmds)
558                 mdb_printf("WARNING: the total of m_target[].m_t_ncmds does "
559                     "not match the slots in use\n");

561         if (mismatch)
562                 mdb_printf("WARNING: corruption in slot table, "
563                     "m_slot[].cmd_slot incorrect\n");

565         /* now check for corruptions */

567         for (q = mp->m_waitq; q; q = c.cmd_linkp) {
571         for (q = m.m_waitq; q; q = c.cmd_linkp) {
568                 for (i = 0; i < nslots; i++)
569                         if (s->m_slot[i] == q)
570                                 mdb_printf("WARNING: m_waitq entry"
571                                     "(mptsas_cmd_t) %p is in m_slot[%i]\n",
572                                     q, i);

574                 if (mdb_vread(&c, sizeof (mptsas_cmd_t), (uintptr_t)q) == -1) {
575                         mdb_warn("couldn't follow m_waitq");
576                         rv = DCMD_ERR;
577                         goto exit;
578                 }
579         }
```

```
581         for (q = mp->m_doneq; q; q = c.cmd_linkp) {
585         for (q = m.m_doneq; q; q = c.cmd_linkp) {
582                 for (i = 0; i < nslots; i++)
583                         if (s->m_slot[i] == q)
584                                 mdb_printf("WARNING: m_doneq entry "
585                                 "(mptsas_cmd_t) %p is in m_slot[%i]\n", q, i);

587                 if (mdb_vread(&c, sizeof (mptsas_cmd_t), (uintptr_t)q) == -1) {
588                         mdb_warn("couldn't follow m_doneq");
589                         rv = DCMD_ERR;
590                         goto exit;
591                 }
592                 if ((c.cmd_flags & CFLAG_FINISHED) == 0)
593                         mdb_printf("WARNING: m_doneq entry (mptsas_cmd_t) %p "
594                             "should have CFLAG_FINISHED set\n", q);
595                 if (c.cmd_flags & CFLAG_IN_TRANSPORT)
596                         mdb_printf("WARNING: m_doneq entry (mptsas_cmd_t) %p "
597                             "should not have CFLAG_IN_TRANSPORT set\n", q);
598                 if (c.cmd_flags & CFLAG_CMDARQ)
599                         mdb_printf("WARNING: m_doneq entry (mptsas_cmd_t) %p "
600                             "should not have CFLAG_CMDARQ set\n", q);
601                 if (c.cmd_flags & CFLAG_COMPLETED)
602                         mdb_printf("WARNING: m_doneq entry (mptsas_cmd_t) %p "
603                             "should not have CFLAG_COMPLETED set\n", q);
604         }

606 exit:
607         mdb_free(slots, sizeof (mptsas_cmd_t) * nslots);
608         return (rv);
613 #endif
614         mdb_printf("\n");
615         mdb_printf("The slot information is not implemented yet\n");
616         return (0);
609 }

611 void
612 display_deviceinfo(struct mptsas *mp)
613 {
614         char    device_path[PATH_MAX];

616         *device_path = 0;
617         if (construct_path((uintptr_t)mp->m_dip, device_path) != DCMD_OK) {
618                 strcpy(device_path, "couldn't determine device path");
619         }

621         mdb_printf("\n");
630         mdb_printf("Path in device tree %s\n", device_path);
631 #if 0
622         mdb_printf("base_wwid          phys "
623             " prodid  devid          revid   ssid\n");
633             "mptid prodid  devid        revid   ssid\n");
624         mdb_printf("----------------------------"
625             "---------------------------------\n");
626         mdb_printf("%"PRIx64"     %2d  "
627             "0x%04x 0x%04x ", mp->un.m_base_wwid, mp->m_num_phys,
628             mp->m_productid, mp->m_devid);
629         switch (mp->m_devid) {
630                 case MPI2_MFGPAGE_DEVID_SAS2004:
631                         mdb_printf("(SAS2004) ");
632                         break;
633                 case MPI2_MFGPAGE_DEVID_SAS2008:
634                         mdb_printf("(SAS2008) ");
635                         break;
636                 case MPI2_MFGPAGE_DEVID_SAS2108_1:
637                 case MPI2_MFGPAGE_DEVID_SAS2108_2:
```



```
638                 case MPI2_MFGPAGE_DEVID_SAS2108_3:
639                         mdb_printf("(SAS2108) ");
640                         break;
641                 case MPI2_MFGPAGE_DEVID_SAS2116_1:
642                 case MPI2_MFGPAGE_DEVID_SAS2116_2:
643                         mdb_printf("(SAS2116) ");
644                         break;
645                 case MPI2_MFGPAGE_DEVID_SAS2208_1:
646                 case MPI2_MFGPAGE_DEVID_SAS2208_2:
647                 case MPI2_MFGPAGE_DEVID_SAS2208_3:
648                 case MPI2_MFGPAGE_DEVID_SAS2208_4:
649                 case MPI2_MFGPAGE_DEVID_SAS2208_5:
650                 case MPI2_MFGPAGE_DEVID_SAS2208_6:
651 #if 0
652                 /* Same as 2308_1/2 ?? */
653                 case MPI2_MFGPAGE_DEVID_SAS2208_7:
654                 case MPI2_MFGPAGE_DEVID_SAS2208_8:
655 #endif
656                         mdb_printf("(SAS2208) ");
636         mdb_printf("%"PRIx64"     %2d   %3d "
637             "0x%04x 0x%04x ", m.un.m_base_wwid, m.m_num_phys, m.m_mptid,
638             m.m_productid, m.m_devid);
639         switch (m.m_devid) {
640                 case MPTSAS_909:
641                         mdb_printf("(909)   ");
642                         break;
643                 case MPTSAS_929:
644                         mdb_printf("(929)   ");
645                         break;
646                 case MPTSAS_919:
647                         mdb_printf("(919)   ");
648                         break;
649                 case MPTSAS_1030:
650                         mdb_printf("(1030)  ");
651                         break;
652                 case MPTSAS_1064:
653                         mdb_printf("(1064)  ");
654                         break;
655                 case MPTSAS_1068:
656                         mdb_printf("(1068)  ");
657                         break;
658                 case MPTSAS_1064E:
659                         mdb_printf("(1064E) ");
660                         break;
661                 case MPTSAS_1068E:
662                         mdb_printf("(1068E) ");
657                         break;
658                 default:
659                         mdb_printf("(SAS????) ");
665                         mdb_printf("(?????) ");
660                         break;
661         }
662         mdb_printf("0x%02x 0x%04x\n", mp->m_revid, mp->m_ssid);
668         mdb_printf("0x%02x 0x%04x\n", m.m_revid, m.m_ssid);
663         mdb_printf("%s\n", device_path);

665 #if 0
666 #endif /* ! codereview */
667         for (i = 0; i < MAX_MPI2_PORTS; i++) {
668                 if (i%4 == 0)
669                         mdb_printf("\n");

671                 mdb_printf("%d:", i);

673                 switch (mp->m_port_type[i]) {
671                 switch (m.m_port_type[i]) {
```

```
674                         case MPI2_PORTFACTS_PORTTYPE_INACTIVE:
675                                 mdb_printf("inactive     ",
676                                     mp->m_protocol_flags[i]);
674                                     m.m_protocol_flags[i]);
677                                 break;
678                         case MPI2_PORTFACTS_PORTTYPE_SCSI:
679                                 mdb_printf("SCSI (0x%1x)   ",
680                                     mp->m_protocol_flags[i]);
678                                     m.m_protocol_flags[i]);
681                                 break;
682                         case MPI2_PORTFACTS_PORTTYPE_FC:
683                                 mdb_printf("FC (0x%1x)     ",
684                                     mp->m_protocol_flags[i]);
682                                     m.m_protocol_flags[i]);
685                                 break;
686                         case MPI2_PORTFACTS_PORTTYPE_ISCSI:
687                                 mdb_printf("iSCSI (0x%1x)  ",
688                                     mp->m_protocol_flags[i]);
686                                     m.m_protocol_flags[i]);
689                                 break;
690                         case MPI2_PORTFACTS_PORTTYPE_SAS:
691                                 mdb_printf("SAS (0x%1x)    ",
692                                     mp->m_protocol_flags[i]);
690                                     m.m_protocol_flags[i]);
693                                 break;
694                         default:
695                                 mdb_printf("unknown      ");
696                 }
697         }
698         mdb_printf("\n");
699 #endif /* ! codereview */
700 #endif
701 }

703 void
704 dump_debug_log(struct mptsas *mp)
705 {
706         uint_t  idx;
707         char    *logbuf;
708         int     i;

710         if (mdb_readsym(&idx, sizeof (uint_t),
711                 "mptsas_dbglog_idx") == -1) {
712                 mdb_warn("No debug log buffer present");
713                 return;
714         }
715         logbuf = mdb_alloc(16*256, UM_SLEEP);
716         if (idx == 0) {
717                 mdb_warn("Logging turned off");
718                 return;
719         }
720
721         if (mdb_readsym(logbuf, 16*256,
722                 "mptsas_dbglog_bufs") == -1) {
723                 mdb_warn("No debug log buffer present");
724                 return;
725         }
726 #endif /* ! codereview */
727         mdb_printf("\n");
728         idx &= 0xf;
729         for (i = 0; i < 16; i++) {
730                 mdb_printf("%s\n", &logbuf[idx*256]);
731                 idx = (idx+1) & 0xf;
732         }
733         mdb_free(logbuf, 16*256);
734 #endif /* ! codereview */
```



```
735 }

737 static int
738 mptsas_dcmd(uintptr_t addr, uint_t flags, int argc, const mdb_arg_t *argv)
739 {
740         struct mptsas           m;
741         struct mptsas_slots     *s;

743         int                     nslots;
744         int                     slot_size = 0;
745         uint_t                  verbose = FALSE;
746         uint_t                  target_info = FALSE;
747         uint_t                  slot_info = FALSE;
748         uint_t                  device_info = FALSE;
749         uint_t                  port_info = FALSE;
750         uint_t                  debug_log = FALSE;
751 #endif /* ! codereview */
752         int                     rv = DCMD_OK;
696         void                    *mptsas_state;

754         if (!(flags & DCMD_ADDRSPEC)) {
755                 void            *mptsas_state = NULL;

699                 mptsas_state = NULL;
757                 if (mdb_readvar(&mptsas_state, "mptsas_state") == -1) {
758                         mdb_warn("can't read mptsas_state");
759                         return (DCMD_ERR);
760                 }
761                 if (mdb_pwalk_dcmd("genunix`softstate",
762                         "mpt_sas`mptsas", argc,
704                 if (mdb_pwalk_dcmd("genunix`softstate", "mpt_sas`mptsas", argc,
763                         argv, (uintptr_t)mptsas_state) == -1) {
764                         mdb_warn("mdb_pwalk_dcmd failed");
765                         return (DCMD_ERR);
766                 }
767                 return (DCMD_OK);
768         }

770         if (mdb_getopts(argc, argv,
771             's', MDB_OPT_SETBITS, TRUE, &slot_info,
772             'd', MDB_OPT_SETBITS, TRUE, &device_info,
773             't', MDB_OPT_SETBITS, TRUE, &target_info,
774             'p', MDB_OPT_SETBITS, TRUE, &port_info,
775             'v', MDB_OPT_SETBITS, TRUE, &verbose,
776             'D', MDB_OPT_SETBITS, TRUE, &debug_log,
777 #endif /* ! codereview */
778             NULL) != argc)
779                 return (DCMD_USAGE);


782         if (mdb_vread(&m, sizeof (m), addr) == -1) {
783                 mdb_warn("couldn't read mpt struct at 0x%p", addr);
784                 return (DCMD_ERR);
785         }

787         s = mdb_alloc(sizeof (mptsas_slots_t), UM_SLEEP);

789         if (mdb_vread(s, sizeof (mptsas_slots_t),
790             (uintptr_t)m.m_active) == -1) {
791                 mdb_warn("couldn't read small mptsas_slots_t at 0x%p",
792                     m.m_active);
793                 mdb_free(s, sizeof (mptsas_slots_t));
794                 return (DCMD_ERR);
795         }

797         nslots = s->m_n_normal;
```

```
799         mdb_free(s, sizeof (mptsas_slots_t));

801         slot_size = sizeof (mptsas_slots_t) +
802             (sizeof (mptsas_cmd_t *) * (nslots-1));

804         s = mdb_alloc(slot_size, UM_SLEEP);

806         if (mdb_vread(s, slot_size, (uintptr_t)m.m_active) == -1) {
807                 mdb_warn("couldn't read large mptsas_slots_t at 0x%p",
808                     m.m_active);
809                 mdb_free(s, slot_size);
810                 return (DCMD_ERR);
811         }

813         /* processing completed */

815         if (((flags & DCMD_ADDRSPEC) && !(flags & DCMD_LOOP)) ||
816             (flags & DCMD_LOOPFIRST) || slot_info || device_info ||
817             target_info) {
818                 if ((flags & DCMD_LOOP) && !(flags & DCMD_LOOPFIRST))
819                         mdb_printf("\n");
820                 mdb_printf("        mptsas_t inst ncmds suspend  power");
821                 mdb_printf("\n");
822                 mdb_printf("======================================="
823                     "====================================");
824                 mdb_printf("\n");
825         }

827         mdb_printf("%16p %4d %5d ", addr, m.m_instance, m.m_ncmds);
828         mdb_printf("%7d", m.m_suspended);
829         switch (m.m_power_level) {
830                 case PM_LEVEL_D0:
831                         mdb_printf(" ON=D0 ");
832                         break;
833                 case PM_LEVEL_D1:
834                         mdb_printf("    D1 ");
835                         break;
836                 case PM_LEVEL_D2:
837                         mdb_printf("    D2 ");
838                         break;
839                 case PM_LEVEL_D3:
840                         mdb_printf("OFF=D3 ");
841                         break;
842                 default:
843                         mdb_printf("INVALD ");
844         }
845         mdb_printf("\n");

847         mdb_inc_indent(17);

849         if (target_info)
850                 display_targets(&m, verbose);
718                 display_targets(&m);

852         if (port_info)
853                 display_ports(&m);

855         if (device_info)
856                 display_deviceinfo(&m);

858         if (slot_info)
859                 display_slotinfo(&m, s);

861         if (debug_log)
862                 dump_debug_log(&m);
```



```
727                 display_slotinfo();

864         mdb_dec_indent(17);

866         mdb_free(s, slot_size);

868         return (rv);
869 }

871 void
872 mptsas_help()
873 {
874         mdb_printf("Prints summary information about each mpt_sas instance, "
875             "including warning\nmessages when slot usage doesn't match "
876             "summary information.\n"
877             "Without the address of a \"struct mptsas\", prints every "
878             "instance.\n\n"
879             "Switches:\n"
880             "  -t[v]  includes information about targets, v = be more verbose\n"
745             "  -t   includes information about targets\n"
881             "  -p     includes information about port\n"
882             "  -s     includes information about mpt slots\n"
883             "  -d     includes information about the hardware\n"
884             "  -D     print the mptsas specific debug log\n");
747             "  -d   includes information about the hardware\n");
885 }

887 static const mdb_dcmd_t dcmds[] = {
888         { "mptsas", "?[-tpsdD]", "print mpt_sas information", mptsas_dcmd,
751         { "mptsas", "?[-tpd]", "print mpt_sas information", mptsas_dcmd,
889             mptsas_help}, { NULL }
890 };
```

_____*unchanged_portion_omitted_*

```
**********************************************************
   23880 Thu Jun 12 17:42:14 2014
new/usr/src/cmd/mdb/common/modules/mpt_sas3/mpt_sas3.c
hg changesets 607a5b46a793..b706c96317c3
Fix ncpus for early boot config
Purge the ack to the interrupt before exiting mptsas_intr()
Changes from code review
Enable Fast Path for capable devices.
Merge fixes for Illumos issue 4819, fix mpt_sas command timeout handling.
Improve the tx_waitq code path.
Major rework of mutexes.
During normal operation do not grab m_mutex during interrupt.
Use reply post queues instead.
Add rolling buffer for *all* debug messages.
Improve mdb module and seperate out into mpt_sas3.
**********************************************************
```

```
/*
 * 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.
 */

/*
 * Copyright 2014 Joyent, Inc.  All rights reserved.
 * Copyright (c) 2014, Tegile Systems Inc. All rights reserved.
 */

#include <limits.h>
#include <sys/mdb_modapi.h>
#include <sys/sysinfo.h>
#include <sys/sunmdi.h>
#include <sys/list.h>
#include <sys/scsi/scsi.h>

#pragma pack(1)
#include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_type.h>
#include <sys/scsi/adapters/mpt_sas3/mpi/mpi2.h>
#include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_cnfg.h>
#include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_init.h>
#include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_ioc.h>
#include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_sas.h>
#include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_raid.h>
#include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_tool.h>
#pragma pack()

#include <sys/scsi/adapters/mpt_sas3/mptsas3_var.h>
#include <sys/scsi/adapters/mpt_sas3/mptsas3_hash.h>
```



```
struct {
        int     value;
        char    *text;
} devinfo_array[] = {
        { MPI2_SAS_DEVICE_INFO_SEP,             "SEP" },
        { MPI2_SAS_DEVICE_INFO_ATAPI_DEVICE,    "ATAPI device" },
        { MPI2_SAS_DEVICE_INFO_LSI_DEVICE,      "LSI device" },
        { MPI2_SAS_DEVICE_INFO_DIRECT_ATTACH,   "direct attach" },
        { MPI2_SAS_DEVICE_INFO_SSP_TARGET,      "SSP tgt" },
        { MPI2_SAS_DEVICE_INFO_STP_TARGET,      "STP tgt" },
        { MPI2_SAS_DEVICE_INFO_SMP_TARGET,      "SMP tgt" },
        { MPI2_SAS_DEVICE_INFO_SATA_DEVICE,     "SATA dev" },
        { MPI2_SAS_DEVICE_INFO_SSP_INITIATOR,   "SSP init" },
        { MPI2_SAS_DEVICE_INFO_STP_INITIATOR,   "STP init" },
        { MPI2_SAS_DEVICE_INFO_SMP_INITIATOR,   "SMP init" },
        { MPI2_SAS_DEVICE_INFO_SATA_HOST,       "SATA host" }
};

int
construct_path(uintptr_t addr, char *result)
{
        struct  dev_info        d;
        char    devi_node[PATH_MAX];
        char    devi_addr[PATH_MAX];

        if (mdb_vread(&d, sizeof (d), addr) == -1) {
                mdb_warn("couldn't read dev_info");
                return (DCMD_ERR);
        }

        if (d.devi_parent) {
                construct_path((uintptr_t)d.devi_parent, result);
                mdb_readstr(devi_node, sizeof (devi_node),
                    (uintptr_t)d.devi_node_name);
                mdb_readstr(devi_addr, sizeof (devi_addr),
                    (uintptr_t)d.devi_addr);
                mdb_snprintf(result+strlen(result),
                    PATH_MAX-strlen(result),
                    "/%s%s%s", devi_node, (*devi_addr ? "@" : ""),
                    devi_addr);
        }
        return (DCMD_OK);
}

/* ARGSUSED */
int
mdi_info_cb(uintptr_t addr, const void *data, void *cbdata)
{
        struct  mdi_pathinfo    pi;
        struct  mdi_client      c;
        char    dev_path[PATH_MAX];
        char    string[PATH_MAX];
        int     mdi_target = 0, mdi_lun = 0;
        int     target = *(int *)cbdata;

        if (mdb_vread(&pi, sizeof (pi), addr) == -1) {
                mdb_warn("couldn't read mdi_pathinfo");
                return (DCMD_ERR);
        }
        mdb_readstr(string, sizeof (string), (uintptr_t)pi.pi_addr);
        mdi_target = (int)mdb_strtoull(string);
        mdi_lun = (int)mdb_strtoull(strchr(string, ',') + 1);
        if (target != mdi_target)
                return (0);
```

```
117         if (mdb_vread(&c, sizeof (c), (uintptr_t)pi.pi_client) == -1) {
118                 mdb_warn("couldn't read mdi_client");
119                 return (-1);
120         }
121
122         *dev_path = NULL;
123         if (construct_path((uintptr_t)c.ct_dip, dev_path) != DCMD_OK)
124                 strcpy(dev_path, "unknown");
125
126         mdb_printf("LUN %d: %s\n", mdi_lun, dev_path);
127         mdb_printf("       dip: %p %s path", c.ct_dip,
128             (pi.pi_preferred ? "preferred" : ""));
129         switch (pi.pi_state & MDI_PATHINFO_STATE_MASK) {
130                 case MDI_PATHINFO_STATE_INIT:
131                         mdb_printf(" initializing");
132                         break;
133                 case MDI_PATHINFO_STATE_ONLINE:
134                         mdb_printf(" online");
135                         break;
136                 case MDI_PATHINFO_STATE_STANDBY:
137                         mdb_printf(" standby");
138                         break;
139                 case MDI_PATHINFO_STATE_FAULT:
140                         mdb_printf(" fault");
141                         break;
142                 case MDI_PATHINFO_STATE_OFFLINE:
143                         mdb_printf(" offline");
144                         break;
145                 default:
146                         mdb_printf(" invalid state");
147                         break;
148         }
149         mdb_printf("\n");
150         return (0);
151 }
152
153 void
154 mdi_info(struct mptsas *mp, int target)
155 {
156         struct  dev_info        d;
157         struct  mdi_phci        p;
158
159         if (mdb_vread(&d, sizeof (d), (uintptr_t)mp->m_dip) == -1) {
160                 mdb_warn("couldn't read m_dip");
161                 return;
162         }
163
164         if (MDI_PHCI(&d)) {
165                 if (mdb_vread(&p, sizeof (p), (uintptr_t)d.devi_mdi_xhci)
166                     == -1) {
167                         mdb_warn("couldn't read m_dip.devi_mdi_xhci");
168                         return;
169                 }
170                 if (p.ph_path_head)
171                         mdb_pwalk("mdipi_phci_list", (mdb_walk_cb_t)mdi_info_cb,
172                             &target, (uintptr_t)p.ph_path_head);
173                 return;
174         }
175 }
176
177 void
178 print_cdb(mptsas_cmd_t *m)
179 {
180         struct  scsi_pkt        pkt;
181         uchar_t cdb[512];       /* an arbitrarily large number */
182         int     j;
```



```
183
184         if (mdb_vread(&pkt, sizeof (pkt), (uintptr_t)m->cmd_pkt) == -1) {
185                 mdb_warn("couldn't read cmd_pkt");
186                 return;
187         }
188
189         /*
190          * We use cmd_cdblen here because 5.10 doesn't
191          * have the cdb length in the pkt
192          */
193         if (mdb_vread(&cdb, m->cmd_cdblen, (uintptr_t)pkt.pkt_cdbp) == -1) {
194                 mdb_warn("couldn't read pkt_cdbp");
195                 return;
196         }
197
198         mdb_printf("%3d,%-3d [ ",
199             pkt.pkt_address.a_target, pkt.pkt_address.a_lun);
200
201         for (j = 0; j < m->cmd_cdblen; j++)
202                 mdb_printf("%02x ", cdb[j]);
203
204         mdb_printf("]\n");
205 }
206
207
208 void
209 display_ports(struct mptsas *mp)
210 {
211         int i;
212         mdb_printf("\n");
213         mdb_printf("phy number and port mapping table\n");
214         for (i = 0; i < MPTSAS_MAX_PHYS; i++) {
215                 if (mp->m_phy_info[i].attached_devhdl) {
216                         mdb_printf("phy %x --> port %x, phymask %x,"
217                         "attached_devhdl %x\n", i, mp->m_phy_info[i].port_num,
218                             mp->m_phy_info[i].phy_mask,
219                             mp->m_phy_info[i].attached_devhdl);
220                 }
221         }
222         mdb_printf("\n");
223 }
224
225 static uintptr_t
226 klist_head(list_t *lp, uintptr_t klp)
227 {
228         if ((uintptr_t)lp->list_head.list_next ==
229             klp + offsetof(struct list, list_head))
230                 return (NULL);
231
232         return ((uintptr_t)(((char *)lp->list_head.list_next) -
233             lp->list_offset));
234 }
235
236 static uintptr_t
237 klist_next(list_t *lp, uintptr_t klp, void *op)
238 {
239         /* LINTED E_BAD_PTR_CAST_ALIG */
240         struct list_node *np = (struct list_node *)(((char *)op) +
241             lp->list_offset);
242
243         if ((uintptr_t)np->list_next == klp + offsetof(struct list, list_head))
244                 return (NULL);
245
246         return (((uintptr_t)(np->list_next)) - lp->list_offset);
247 }
```

```
249 static void *
250 krefhash_first(uintptr_t khp, uintptr_t *addr)
251 {
252         refhash_t mh;
253         uintptr_t klp;
254         uintptr_t kop;
255         void *rp;

257         mdb_vread(&mh, sizeof (mh), khp);
258         klp = klist_head(&mh.rh_objs, khp + offsetof(refhash_t, rh_objs));
259         if (klp == 0)
260                 return (NULL);

262         kop = klp - mh.rh_link_off;
263         if (addr)
264                 *addr = kop;
265         rp = mdb_alloc(mh.rh_obj_size, UM_SLEEP);
266         mdb_vread(rp, mh.rh_obj_size, kop);

268         return (rp);
269 }

271 static void *
272 krefhash_next(uintptr_t khp, void *op, uintptr_t *addr)
273 {
274         refhash_t mh;
275         void *prev = op;
276         refhash_link_t *lp;
277         uintptr_t klp;
278         uintptr_t kop;
279         refhash_link_t ml;
280         void *rp;

282         mdb_vread(&mh, sizeof (mh), khp);
283         /* LINTED E_BAD_PTR_CAST_ALIG */
284         lp = (refhash_link_t *)(((char *)(op)) + mh.rh_link_off);
285         ml = *lp;
286         while ((klp = klist_next(&mh.rh_objs,
287             khp + offsetof(refhash_t, rh_objs), &ml)) != NULL) {
288                 mdb_vread(&ml, sizeof (ml), klp);
289                 if (!(ml.rhl_flags & RHL_F_DEAD))
290                         break;
291         }

293         if (klp == 0) {
294                 mdb_free(prev, mh.rh_obj_size);
295                 return (NULL);
296         }

298         kop = klp - mh.rh_link_off;
299         if (addr)
300                 *addr = kop;
301         rp = mdb_alloc(mh.rh_obj_size, UM_SLEEP);
302         mdb_vread(rp, mh.rh_obj_size, kop);

304         mdb_free(prev, mh.rh_obj_size);
305         return (rp);
306 }

308 void
309 display_targets(struct mptsas *mp, uint_t verbose)
310 {
311         mptsas_target_t *ptgt;
312         mptsas_smp_t *psmp;
313         int loop, comma;
314         uintptr_t p_addr;
```



```
316         mdb_printf("\n");
317         mdb_printf(" mptsas_target_t slot devhdl      wwn     ncmds throttle   "
318             "dr_flag dups\n");
319         mdb_printf("----------------------------------------"
320             "----------------------------\n");
321         for (ptgt = (mptsas_target_t *)krefhash_first(
322                 (uintptr_t)mp->m_targets, &p_addr);
323             ptgt != NULL;
324             ptgt = krefhash_next((uintptr_t)mp->m_targets, ptgt, &p_addr)) {
325                 if (ptgt->m_addr.mta_wwn ||
326                     ptgt->m_deviceinfo) {
327                         mdb_printf("%16p ", p_addr);
328                         mdb_printf("%4d ", ptgt->m_slot_num);
329                         mdb_printf("%4d ", ptgt->m_devhdl);
330                         if (ptgt->m_addr.mta_wwn)
331                                 mdb_printf("%"PRIx64" ",
332                                     ptgt->m_addr.mta_wwn);
333                         mdb_printf("%3d", ptgt->m_t_ncmds);
334                         switch (ptgt->m_t_throttle) {
335                                 case QFULL_THROTTLE:
336                                         mdb_printf("   QFULL ");
337                                         break;
338                                 case DRAIN_THROTTLE:
339                                         mdb_printf("   DRAIN ");
340                                         break;
341                                 case HOLD_THROTTLE:
342                                         mdb_printf("    HOLD ");
343                                         break;
344                                 case MAX_THROTTLE:
345                                         mdb_printf("     MAX ");
346                                         break;
347                                 default:
348                                         mdb_printf("%8d ",
349                                             ptgt->m_t_throttle);
350                         }
351                         switch (ptgt->m_dr_flag) {
352                                 case MPTSAS_DR_INACTIVE:
353                                         mdb_printf("  INACTIVE ");
354                                         break;
355                                 case MPTSAS_DR_INTRANSITION:
356                                         mdb_printf("TRANSITION ");
357                                         break;
358                                 default:
359                                         mdb_printf("   UNKNOWN ");
360                                         break;
361                                 }
362                         mdb_printf("%d\n",
363                             ptgt->m_dups);

365                         if (verbose) {
366                                 mdb_inc_indent(5);
367                                 if ((ptgt->m_deviceinfo &
368                                     MPI2_SAS_DEVICE_INFO_MASK_DEVICE_TYPE) ==
369                                     MPI2_SAS_DEVICE_INFO_FANOUT_EXPANDER)
370                                         mdb_printf("Fanout expander: ");
371                                 if ((ptgt->m_deviceinfo &
372                                     MPI2_SAS_DEVICE_INFO_MASK_DEVICE_TYPE) ==
373                                     MPI2_SAS_DEVICE_INFO_EDGE_EXPANDER)
374                                         mdb_printf("Edge expander: ");
375                                 if ((ptgt->m_deviceinfo &
376                                     MPI2_SAS_DEVICE_INFO_MASK_DEVICE_TYPE) ==
377                                     MPI2_SAS_DEVICE_INFO_END_DEVICE)
378                                         mdb_printf("End device: ");
379                                 if ((ptgt->m_deviceinfo &
380                                     MPI2_SAS_DEVICE_INFO_MASK_DEVICE_TYPE) ==
```

```
381                                 MPI2_SAS_DEVICE_INFO_NO_DEVICE)
382                                     mdb_printf("No device ");

384                                 for (loop = 0, comma = 0;
385                                     loop < (sizeof (devinfo_array) /
386                                     sizeof (devinfo_array[0])); loop++) {
387                                         if (ptgt->m_deviceinfo &
388                                             devinfo_array[loop].value) {
389                                                 mdb_printf("%s%s",
390                                                     (comma ? ", " : ""),
391                                                     devinfo_array[loop].text);
392                                                 comma++;
393                                         }
394                                 }
395                                 mdb_printf("\n");
396                                 mdi_info(mp, ptgt->m_slot_num);
397                                 mdb_dec_indent(5);
398                         }
399                 }
400         }

402         mdb_printf("\n");
403         mdb_printf("    mptsas_smp_t devhdl      wwn          phymask\n");
404         mdb_printf("--------------------------------------"
405             "-----------------\n");
406         for (psmp = (mptsas_smp_t *)krefhash_first(
407             (uintptr_t)mp->m_smp_targets, &p_addr);
408             psmp != NULL;
409             psmp = krefhash_next((uintptr_t)mp->m_smp_targets, psmp,
410             &p_addr)) {
411                 mdb_printf("%16p   ", p_addr);
412                 mdb_printf("%4d  %"PRIx64"    %04x\n",
413                     psmp->m_devhdl, psmp->m_addr.mta_wwn,
414                     psmp->m_addr.mta_phymask);
415                 if (verbose) {
416                         mdb_inc_indent(5);
417                         if ((psmp->m_deviceinfo &
418                                 MPI2_SAS_DEVICE_INFO_MASK_DEVICE_TYPE) ==
419                             MPI2_SAS_DEVICE_INFO_FANOUT_EXPANDER)
420                                 mdb_printf("Fanout expander: ");
421                         if ((psmp->m_deviceinfo &
422                                 MPI2_SAS_DEVICE_INFO_MASK_DEVICE_TYPE) ==
423                             MPI2_SAS_DEVICE_INFO_EDGE_EXPANDER)
424                                 mdb_printf("Edge expander: ");
425                         if ((psmp->m_deviceinfo &
426                                 MPI2_SAS_DEVICE_INFO_MASK_DEVICE_TYPE) ==
427                             MPI2_SAS_DEVICE_INFO_END_DEVICE)
428                                 mdb_printf("End device: ");
429                         if ((psmp->m_deviceinfo &
430                                 MPI2_SAS_DEVICE_INFO_MASK_DEVICE_TYPE) ==
431                             MPI2_SAS_DEVICE_INFO_NO_DEVICE)
432                                 mdb_printf("No device ");

434                         for (loop = 0, comma = 0;
435                              loop < (sizeof (devinfo_array) /
436                                sizeof (devinfo_array[0])); loop++) {
437                                 if (psmp->m_deviceinfo &
438                                     devinfo_array[loop].value) {
439                                         mdb_printf("%s%s",
440                                             (comma ? ", " : ""),
441                                             devinfo_array[loop].text);
442                                         comma++;
443                                 }
444                         }
445                         mdb_printf("\n");
446                         mdb_dec_indent(5);
```



```
447                 }
448         }
449 }

451 int
452 display_slotinfo(struct mptsas *mp, struct mptsas_slots *s)
453 {
454         int                     i, nslots;
455         struct mptsas_cmd       c, *q, *slots;
456         mptsas_target_t         *ptgt;
457         int                     header_output = 0;
458         int                     rv = DCMD_OK;
459         int                     slots_in_use = 0;
460         int                     tcmds = 0;
461         int                     mismatch = 0;
462         int                     wq, dq;
463         int                     ncmds = 0;
464         ulong_t                 saved_indent;

466         nslots = s->m_n_normal;
467         slots = mdb_alloc(sizeof (mptsas_cmd_t) * nslots, UM_SLEEP);

469         for (i = 0; i < nslots; i++)
470                 if (s->m_slot[i]) {
471                         slots_in_use++;
472                         if (mdb_vread(&slots[i], sizeof (mptsas_cmd_t),
473                             (uintptr_t)s->m_slot[i]) == -1) {
474                                 mdb_warn("couldn't read slot");
475                                 s->m_slot[i] = NULL;
476                         }
477                         if ((slots[i].cmd_flags & CFLAG_CMDIOC) == 0)
478                                 tcmds++;
479                         if (i != slots[i].cmd_slot)
480                                 mismatch++;
481                 }

483         for (q = mp->m_waitq, wq = 0; q; q = c.cmd_linkp, wq++)
484                 if (mdb_vread(&c, sizeof (mptsas_cmd_t), (uintptr_t)q) == -1) {
485                         mdb_warn("couldn't follow m_waitq");
486                         rv = DCMD_ERR;
487                         goto exit;
488                 }

490         for (q = mp->m_dlist.dl_q, dq = 0; q; q = c.cmd_linkp, dq++)
491                 if (mdb_vread(&c, sizeof (mptsas_cmd_t), (uintptr_t)q) == -1) {
492                         mdb_warn("couldn't follow m_doneq");
493                         rv = DCMD_ERR;
494                         goto exit;
495                 }

497         for (ptgt = (mptsas_target_t *)krefhash_first(
498                 (uintptr_t)mp->m_targets, NULL);
499             ptgt != NULL;
500              ptgt = krefhash_next((uintptr_t)mp->m_targets, ptgt, NULL)) {
501                 if (ptgt->m_addr.mta_wwn ||
502                     ptgt->m_deviceinfo) {
503                         ncmds += ptgt->m_t_ncmds;
504                 }
505         }

507         mdb_printf("\n");
508         mdb_printf("   mpt.  slot              mptsas_slots     slot");
509         mdb_printf("\n");
510         mdb_printf("m_ncmds total"
511             " targ throttle m_t_ncmds targ_tot wq dq");
512         mdb_printf("\n");
```

```
513         mdb_printf("-----------------------------------------------------");
514         mdb_printf("\n");

516         mdb_printf("%7d ", mp->m_ncmds);
517         mdb_printf("%s", (mp->m_ncmds == slots_in_use ? "  " : "!="));
518         mdb_printf("%3d              total %3d ", slots_in_use, ncmds);
519         mdb_printf("%s", (tcmds == ncmds ? "     " : "   !="));
520         mdb_printf("%3d %2d %2d\n", tcmds, wq, dq);

522         saved_indent = mdb_dec_indent(0);
523         mdb_dec_indent(saved_indent);

525         for (i = 0; i < s->m_n_normal; i++)
526                 if (s->m_slot[i]) {
527                         if (!header_output) {
528                                 mdb_printf("\n");
529                                 mdb_printf("mptsas_cmd          slot cmd_slot "
530                                     "cmd_flags cmd_pkt_flags scsi_pkt      "
531                                     "  targ,lun [ pkt_cdbp ...\n");
532                                 mdb_printf("------------------------------"
533                                     "--------------------------------------"
534                                     "--------------------------------------"
535                                     "------\n");
536                                 header_output = 1;
537                         }
538                         mdb_printf("%16p %4d %s %4d  %8x      %8x %16p ",
539                             s->m_slot[i], i,
540                             (i == slots[i].cmd_slot?"   ":"BAD"),
541                             slots[i].cmd_slot,
542                             slots[i].cmd_flags,
543                             slots[i].cmd_pkt_flags,
544                             slots[i].cmd_pkt);
545                         (void) print_cdb(&slots[i]);
546                 }

548         /* print the wait queue */

550         for (q = mp->m_waitq; q; q = c.cmd_linkp) {
551                 if (q == mp->m_waitq)
552                         mdb_printf("\n");
553                 if (mdb_vread(&c, sizeof (mptsas_cmd_t), (uintptr_t)q)
554                     == -1) {
555                         mdb_warn("couldn't follow m_waitq");
556                         rv = DCMD_ERR;
557                         goto exit;
558                 }
559                 mdb_printf("%16p wait n/a %4d  %8x      %8x %16p ",
560                     q, c.cmd_slot, c.cmd_flags, c.cmd_pkt_flags,
561                     c.cmd_pkt);
562                 print_cdb(&c);
563         }

565         /* print the done queue */

567         for (q = mp->m_dlist.dl_q; q; q = c.cmd_linkp) {
568                 if (q == mp->m_dlist.dl_q)
569                         mdb_printf("\n");
570                 if (mdb_vread(&c, sizeof (mptsas_cmd_t), (uintptr_t)q)
571                     == -1) {
572                         mdb_warn("couldn't follow m_doneq");
573                         rv = DCMD_ERR;
574                         goto exit;
575                 }
576                 mdb_printf("%16p done  n/a %4d  %8x      %8x %16p ",
577                     q, c.cmd_slot, c.cmd_flags, c.cmd_pkt_flags,
578                     c.cmd_pkt);
```



```
579                 print_cdb(&c);
580         }

582         mdb_inc_indent(saved_indent);

584         if (mp->m_ncmds != slots_in_use)
585                 mdb_printf("WARNING: mpt.m_ncmds does not match the number of "
586                     "slots in use\n");

588         if (tcmds != ncmds)
589                 mdb_printf("WARNING: the total of m_target[].m_t_ncmds does "
590                     "not match the slots in use\n");

592         if (mismatch)
593                 mdb_printf("WARNING: corruption in slot table, "
594                     "m_slot[].cmd_slot incorrect\n");

596         /* now check for corruptions */

598         for (q = mp->m_waitq; q; q = c.cmd_linkp) {
599                 for (i = 0; i < nslots; i++)
600                         if (s->m_slot[i] == q)
601                                 mdb_printf("WARNING: m_waitq entry"
602                                     "(mptsas_cmd_t) %p is in m_slot[%i]\n",
603                                     q, i);

605                 if (mdb_vread(&c, sizeof (mptsas_cmd_t), (uintptr_t)q) == -1) {
606                         mdb_warn("couldn't follow m_waitq");
607                         rv = DCMD_ERR;
608                         goto exit;
609                 }
610         }

612         for (q = mp->m_dlist.dl_q; q; q = c.cmd_linkp) {
613                 for (i = 0; i < nslots; i++)
614                         if (s->m_slot[i] == q)
615                                 mdb_printf("WARNING: m_doneq entry "
616                                 "(mptsas_cmd_t) %p is in m_slot[%i]\n", q, i);

618                 if (mdb_vread(&c, sizeof (mptsas_cmd_t), (uintptr_t)q) == -1) {
619                         mdb_warn("couldn't follow m_doneq");
620                         rv = DCMD_ERR;
621                         goto exit;
622                 }
623                 if ((c.cmd_flags & CFLAG_FINISHED) == 0)
624                         mdb_printf("WARNING: m_doneq entry (mptsas_cmd_t) %p "
625                             "should have CFLAG_FINISHED set\n", q);
626                 if (c.cmd_flags & CFLAG_IN_TRANSPORT)
627                         mdb_printf("WARNING: m_doneq entry (mptsas_cmd_t) %p "
628                             "should not have CFLAG_IN_TRANSPORT set\n", q);
629                 if (c.cmd_flags & CFLAG_CMDARQ)
630                         mdb_printf("WARNING: m_doneq entry (mptsas_cmd_t) %p "
631                             "should not have CFLAG_CMDARQ set\n", q);
632                 if (c.cmd_flags & CFLAG_COMPLETED)
633                         mdb_printf("WARNING: m_doneq entry (mptsas_cmd_t) %p "
634                             "should not have CFLAG_COMPLETED set\n", q);
635         }

637 exit:
638         mdb_free(slots, sizeof (mptsas_cmd_t) * nslots);
639         return (rv);
640 }

642 void
643 display_deviceinfo(struct mptsas *mp)
644 {
```

```
645         char    device_path[PATH_MAX];

647         *device_path = 0;
648         if (construct_path((uintptr_t)mp->m_dip, device_path) != DCMD_OK) {
649                 strcpy(device_path, "couldn't determine device path");
650         }

652         mdb_printf("\n");
653         mdb_printf("base_wwid          phys "
654             " prodid  devid          revid   ssid\n");
655         mdb_printf("----------------------------"
656             "---------------------------------\n");
657         mdb_printf("%"PRIx64"     %2d  "
658             "0x%04x 0x%04x ", mp->un.m_base_wwid, mp->m_num_phys,
659             mp->m_productid, mp->m_devid);
660         switch (mp->m_devid) {
661                 case MPI2_MFGPAGE_DEVID_SAS2004:
662                         mdb_printf("(SAS2004) ");
663                         break;
664                 case MPI2_MFGPAGE_DEVID_SAS2008:
665                         mdb_printf("(SAS2008) ");
666                         break;
667                 case MPI2_MFGPAGE_DEVID_SAS2108_1:
668                 case MPI2_MFGPAGE_DEVID_SAS2108_2:
669                 case MPI2_MFGPAGE_DEVID_SAS2108_3:
670                         mdb_printf("(SAS2108) ");
671                         break;
672                 case MPI2_MFGPAGE_DEVID_SAS2116_1:
673                 case MPI2_MFGPAGE_DEVID_SAS2116_2:
674                         mdb_printf("(SAS2116) ");
675                         break;
676                 case MPI2_MFGPAGE_DEVID_SSS6200:
677                         mdb_printf("(SSS6200) ");
678                         break;
679                 case MPI2_MFGPAGE_DEVID_SAS2208_1:
680                 case MPI2_MFGPAGE_DEVID_SAS2208_2:
681                 case MPI2_MFGPAGE_DEVID_SAS2208_3:
682                 case MPI2_MFGPAGE_DEVID_SAS2208_4:
683                 case MPI2_MFGPAGE_DEVID_SAS2208_5:
684                 case MPI2_MFGPAGE_DEVID_SAS2208_6:
685 #if 0
686                 /* Same as 2308_1/2 ?? */
687                 case MPI2_MFGPAGE_DEVID_SAS2208_7:
688                 case MPI2_MFGPAGE_DEVID_SAS2208_8:
689 #endif
690                         mdb_printf("(SAS2208) ");
691                         break;
692                 case MPI2_MFGPAGE_DEVID_SAS2308_1:
693                 case MPI2_MFGPAGE_DEVID_SAS2308_2:
694                 case MPI2_MFGPAGE_DEVID_SAS2308_3:
695                         mdb_printf("(SAS2308) ");
696                         break;
697                 case MPI25_MFGPAGE_DEVID_SAS3004:
698                         mdb_printf("(SAS3004) ");
699                         break;
700                 case MPI25_MFGPAGE_DEVID_SAS3008:
701                         mdb_printf("(SAS3008) ");
702                         break;
703                 case MPI25_MFGPAGE_DEVID_SAS3108_1:
704                 case MPI25_MFGPAGE_DEVID_SAS3108_2:
705                 case MPI25_MFGPAGE_DEVID_SAS3108_3:
706                 case MPI25_MFGPAGE_DEVID_SAS3108_4:
707                 case MPI25_MFGPAGE_DEVID_SAS3108_5:
708                 case MPI25_MFGPAGE_DEVID_SAS3108_6:
709                         mdb_printf("(SAS3108) ");
710                         break;
```



```
711                 default:
712                         mdb_printf("(SAS????) ");
713                         break;
714         }
715         mdb_printf("0x%02x 0x%04x\n", mp->m_revid, mp->m_ssid);
716         mdb_printf("%s\n", device_path);
717 }

719 void
720 dump_debug_log(struct mptsas *mp)
721 {
722         uint_t  idx;
723         char    *logbuf;
724         int     i;

726         if (mdb_readsym(&idx, sizeof (uint_t),
727                 "mptsas_dbglog_idx") == -1) {
728                 mdb_warn("No debug log buffer present");
729                 return;
730         }
731         logbuf = mdb_alloc(32*256, UM_SLEEP);
732         if (idx == 0) {
733                 mdb_warn("Logging turned off");
734                 return;
735         }

737         if (mdb_readsym(logbuf, 32*256,
738                 "mptsas_dbglog_bufs") == -1) {
739                 mdb_warn("No debug log buffer present");
740                 return;
741         }
742         mdb_printf("\n");
743         idx &= 0x1f;
744         for (i = 0; i < 32; i++) {
745                 mdb_printf("%s\n", &logbuf[idx*256]);
746                 idx = (idx+1) & 0x1f;
747         }
748         mdb_free(logbuf, 32*256);
749 }

751 static int
752 mptsas_dcmd(uintptr_t addr, uint_t flags, int argc, const mdb_arg_t *argv)
753 {
754         struct mptsas           m;
755         struct mptsas_slots     *s;

757         int                     nslots;
758         int                     slot_size = 0;
759         uint_t                  verbose = FALSE;
760         uint_t                  target_info = FALSE;
761         uint_t                  slot_info = FALSE;
762         uint_t                  device_info = FALSE;
763         uint_t                  port_info = FALSE;
764         uint_t                  debug_log = FALSE;
765         int                     rv = DCMD_OK;

767         if (!(flags & DCMD_ADDRSPEC)) {
768                 void            *mptsas3_state = NULL;

770                 if (mdb_readvar(&mptsas3_state, "mptsas3_state") == -1) {
771                         mdb_warn("can't read mptsas3_state");
772                         return (DCMD_ERR);
773                 }
774                 if (mdb_pwalk_dcmd("genunix`softstate",
775                         "mpt_sas3`mptsas3", argc,
776                         argv, (uintptr_t)mptsas3_state) == -1) {
```

```
777                         mdb_warn("mdb_pwalk_dcmd failed");
778                         return (DCMD_ERR);
779                 }
780                 return (DCMD_OK);
781         }

783         if (mdb_getopts(argc, argv,
784             's', MDB_OPT_SETBITS, TRUE, &slot_info,
785             'd', MDB_OPT_SETBITS, TRUE, &device_info,
786             't', MDB_OPT_SETBITS, TRUE, &target_info,
787             'p', MDB_OPT_SETBITS, TRUE, &port_info,
788             'v', MDB_OPT_SETBITS, TRUE, &verbose,
789             'D', MDB_OPT_SETBITS, TRUE, &debug_log,
790             NULL) != argc)
791                 return (DCMD_USAGE);


794         if (mdb_vread(&m, sizeof (m), addr) == -1) {
795                 mdb_warn("couldn't read mpt struct at 0x%p", addr);
796                 return (DCMD_ERR);
797         }

799         s = mdb_alloc(sizeof (mptsas_slots_t), UM_SLEEP);

801         if (mdb_vread(s, sizeof (mptsas_slots_t),
802             (uintptr_t)m.m_active) == -1) {
803                 mdb_warn("couldn't read small mptsas_slots_t at 0x%p",
804                     m.m_active);
805                 mdb_free(s, sizeof (mptsas_slots_t));
806                 return (DCMD_ERR);
807         }

809         nslots = s->m_n_normal;

811         mdb_free(s, sizeof (mptsas_slots_t));

813         slot_size = sizeof (mptsas_slots_t) +
814             (sizeof (mptsas_cmd_t *) * (nslots-1));

816         s = mdb_alloc(slot_size, UM_SLEEP);

818         if (mdb_vread(s, slot_size, (uintptr_t)m.m_active) == -1) {
819                 mdb_warn("couldn't read large mptsas_slots_t at 0x%p",
820                     m.m_active);
821                 mdb_free(s, slot_size);
822                 return (DCMD_ERR);
823         }

825         /* processing completed */

827         if (((flags & DCMD_ADDRSPEC) && !(flags & DCMD_LOOP)) ||
828             (flags & DCMD_LOOPFIRST) || slot_info || device_info ||
829             target_info) {
830                 if ((flags & DCMD_LOOP) && !(flags & DCMD_LOOPFIRST))
831                         mdb_printf("\n");
832                 mdb_printf("       mptsas_t inst ncmds suspend  power");
833                 mdb_printf("\n");
834                 mdb_printf("======================================="
835                     "=======================================");
836                 mdb_printf("\n");
837         }

839         mdb_printf("%16p %4d %5d ", addr, m.m_instance, m.m_ncmds);
840         mdb_printf("%7d", m.m_suspended);
841         switch (m.m_power_level) {
842                 case PM_LEVEL_D0:
```



```
843                         mdb_printf(" ON=D0 ");
844                         break;
845                 case PM_LEVEL_D1:
846                         mdb_printf("    D1 ");
847                         break;
848                 case PM_LEVEL_D2:
849                         mdb_printf("    D2 ");
850                         break;
851                 case PM_LEVEL_D3:
852                         mdb_printf("OFF=D3 ");
853                         break;
854                 default:
855                         mdb_printf("INVALD ");
856         }
857         mdb_printf("\n");

859         mdb_inc_indent(17);

861         if (target_info)
862                 display_targets(&m, verbose);

864         if (port_info)
865                 display_ports(&m);

867         if (device_info)
868                 display_deviceinfo(&m);

870         if (slot_info)
871                 display_slotinfo(&m, s);

873         if (debug_log)
874                 dump_debug_log(&m);

876         mdb_dec_indent(17);

878         mdb_free(s, slot_size);

880         return (rv);
881 }

883 void
884 mptsas_help()
885 {
886         mdb_printf("Prints summary information about each mpt_sas3 instance, "
887             "including warning\nmessages when slot usage doesn't match "
888             "summary information.\n"
889             "Without the address of a \"struct mptsas\", prints every "
890             "instance.\n\n"
891             "Switches:\n"
892             "  -t[v]  includes information about targets, v = be more verbose\n"
893             "  -p     includes information about port\n"
894             "  -s     includes information about mpt slots\n"
895             "  -d     includes information about the hardware\n"
896             "  -D     print the mptsas specific debug log\n");
897 }

899 static const mdb_dcmd_t dcmds[] = {
900         { "mptsas3", "?[-tpsdD]", "print mpt_sas3 information", mptsas_dcmd,
901             mptsas_help}, { NULL }
902 };

904 static const mdb_modinfo_t modinfo = {
905         MDB_API_VERSION, dcmds, NULL
906 };

908 const mdb_modinfo_t *
```

```
909 _mdb_init(void)
910 {
911         return (&modinfo);
912 }
913 #endif /* ! codereview */
```

```
**********************************************************
    1154 Thu Jun 12 17:42:14 2014
new/usr/src/cmd/mdb/intel/amd64/mpt_sas3/Makefile
Add rolling buffer for *all* debug messages.
Improve mdb module and seperate out into mpt_sas3.
**********************************************************
```

```
#
# 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.

MODULE = mpt_sas3.so
MDBTGT = kvm

MODSRCS = mpt_sas3.c

include ../../../../Makefile.cmd
include ../../../../Makefile.cmd.64
include ../../Makefile.amd64
include ../../../Makefile.module

CPPFLAGS += -I$(SRC)/uts/common

CERRWARN += -_gcc=-Wno-trigraphs
#endif /* ! codereview */
```

```
**********************************************************
    1118 Thu Jun 12 17:42:14 2014
new/usr/src/cmd/mdb/intel/ia32/mpt_sas3/Makefile
Add rolling buffer for *all* debug messages.
Improve mdb module and seperate out into mpt_sas3.
**********************************************************
```

```
#
# 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.


MODULE = mpt_sas3.so
MDBTGT = kvm

MODSRCS = mpt_sas3.c

include ../../../../Makefile.cmd
include ../../Makefile.ia32
include ../../../Makefile.module

CPPFLAGS += -I$(SRC)/uts/common

CERRWARN += -_gcc=-Wno-trigraphs
#endif /* ! codereview */
```

```
**********************************************************
    1156 Thu Jun 12 17:42:14 2014
new/usr/src/cmd/mdb/sparc/v9/mpt_sas3/Makefile
Add rolling buffer for *all* debug messages.
Improve mdb module and seperate out into mpt_sas3.
**********************************************************
```

```
#
# 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.

MODULE = mpt_sas3.so
MDBTGT = kvm

MODSRCS = mpt_sas3.c

include ../../../../Makefile.cmd
include ../../../../Makefile.cmd.64
include ../../Makefile.sparcv9
include ../../../Makefile.module

CPPFLAGS += -I$(SRC)/uts/common

CERRWARN += -_gcc=-Wno-trigraphs
#endif /* ! codereview */
```

```
**********************************************************
    4539 Thu Jun 12 17:42:14 2014
new/usr/src/man/man7d/Makefile
Added man page.
**********************************************************
#
# This file and its contents are supplied under the terms of the
# Common Development and Distribution License ("CDDL"), version 1.0.
# You may only use this file in accordance with the terms of version
# 1.0 of the CDDL.
#
# A full copy of the text of the CDDL should have accompanied this
# source.  A copy of the CDDL is also available via the Internet
# at http://www.illumos.org/license/CDDL.
#

#
# Copyright 2011, Richard Lowe
# Copyright 2013 Nexenta Systems, Inc.  All rights reserved.
# Copyright 2013 Garrett D'Amore <garrett@damore.org>
#

include         $(SRC)/Makefile.master

MANSECT=        7d

_MANFILES=      aac.7d          \
                afe.7d          \
                audio.7d        \
                audio1575.7d    \
                audioens.7d     \
                audiols.7d      \
                audiop16x.7d    \
                audiopci.7d     \
                audiots.7d      \
                av1394.7d       \
                bge.7d          \
                bscv.7d         \
                chxge.7d        \
                console.7d      \
                cpuid.7d        \
                dca.7d          \
                dcam1394.7d     \
                devinfo.7d      \
                dmfe.7d         \
                dtrace.7d       \
                ehci.7d         \
                fasttrap.7d     \
                fbt.7d          \
                fcip.7d         \
                fcoe.7d         \
                fcoei.7d        \
                fcoet.7d        \
                fcp.7d          \
                fctl.7d         \
                fd.7d           \
                fp.7d           \
                gld.7d          \
                hci1394.7d      \
                hermon.7d       \
                hid.7d          \
                hme.7d          \
                hubd.7d         \
                hwahc.7d        \
                hwarc.7d        \
                hxge.7d         \
```



```
                ib.7d           \
                ibcm.7d         \
                ibd.7d          \
                ibdm.7d         \
                ibdma.7d        \
                ibtl.7d         \
                ieee1394.7d     \
                igb.7d          \
                ii.7d           \
                ipnet.7d        \
                iscsi.7d        \
                iser.7d         \
                ixgbe.7d        \
                kmdb.7d         \
                kstat.7d        \
                ksyms.7d        \
                llc1.7d         \
                lockstat.7d     \
                lofi.7d         \
                log.7d          \
                md.7d           \
                mediator.7d     \
                mem.7d          \
                mpt_sas.7d      \
                mpt_sas3.7d     \
#endif /* ! codereview */
                mr_sas.7d       \
                msglog.7d       \
                mt.7d           \
                mxfe.7d         \
                myri10ge.7d     \
                null.7d         \
                nulldriver.7d   \
                nxge.7d         \
                ohci.7d         \
                openprom.7d     \
                pcic.7d         \
                pcmcia.7d       \
                physmem.7d      \
                pm.7d           \
                poll.7d         \
                profile.7d      \
                ptm.7d          \
                pts.7d          \
                pty.7d          \
                qlc.7d          \
                ramdisk.7d      \
                random.7d       \
                rge.7d          \
                sad.7d          \
                sata.7d         \
                scsa1394.7d     \
                scsa2usb.7d     \
                sd.7d           \
                sdp.7d          \
                sdt.7d          \
                ses.7d          \
                sfe.7d          \
                sgen.7d         \
                srpt.7d         \
                st.7d           \
                sv.7d           \
                sysmsg.7d       \
                systrace.7d     \
                ticlts.7d       \
                tty.7d          \
```

```
128                 ttymux.7d       \
129                 tzmon.7d        \
130                 ugen.7d         \
131                 uhci.7d         \
132                 usb_ac.7d       \
133                 usb_as.7d       \
134                 usb_ia.7d       \
135                 usb_mid.7d      \
136                 usba.7d         \
137                 usbftdi.7d      \
138                 usbprn.7d       \
139                 usbsacm.7d      \
140                 usbsksp.7d      \
141                 usbsprl.7d      \
142                 usbvc.7d        \
143                 uwba.7d         \
144                 virtualkm.7d    \
145                 vni.7d          \
146                 vr.7d           \
147                 wscons.7d       \
148                 wusb_ca.7d      \
149                 wusb_df.7d      \
150                 xge.7d          \
151                 yge.7d          \
152                 zcons.7d        \
153                 zero.7d

155 sparc_MANFILES= audiocs.7d      \
156                 bbc_beep.7d     \
157                 ctsmc.7d        \
158                 cvc.7d          \
159                 cvcredir.7d     \
160                 dad.7d          \
161                 dm2s.7d         \
162                 dr.7d           \
163                 eri.7d          \
164                 fas.7d          \
165                 gpio_87317.7d   \
166                 grbeep.7d       \
167                 idn.7d          \
168                 mc-opl.7d       \
169                 n2rng.7d        \
170                 ncp.7d          \
171                 ntwdt.7d        \
172                 oplkmdrv.7d     \
173                 oplmsu.7d       \
174                 oplpanel.7d     \
175                 pcicmu.7d       \
176                 pcipsy.7d       \
177                 pcisch.7d       \
178                 schpc.7d        \
179                 sf.7d           \
180                 smbus.7d        \
181                 socal.7d        \
182                 ssd.7d          \
183                 su.7d           \
184                 todopl.7d       \
185                 tsalarm.7d      \
186                 zs.7d           \
187                 zsh.7d

189 i386_MANFILES=  ahci.7d         \
190                 amd8111s.7d     \
191                 amr.7d          \
192                 arcmsr.7d       \
193                 arn.7d          \
```



```
194                 asy.7d          \
195                 ata.7d          \
196                 atge.7d         \
197                 ath.7d          \
198                 atu.7d          \
199                 audio810.7d     \
200                 audiocmi.7d     \
201                 audiocmihd.7d   \
202                 audioemu10k.7d  \
203                 audiohd.7d      \
204                 audioixp.7d     \
205                 audiosolo.7d    \
206                 audiovia823x.7d \
207                 audiovia97.7d   \
208                 bcm_sata.7d     \
209                 bfe.7d          \
210                 cmdk.7d         \
211                 cpqary3.7d      \
212                 dnet.7d         \
213                 ecpp.7d         \
214                 heci.7d         \
215                 i915.7d         \
216                 ipmi.7d         \
217                 ipw.7d          \
218                 iwh.7d          \
219                 iwi.7d          \
220                 mega_sas.7d     \
221                 npe.7d          \
222                 ntxn.7d         \
223                 nv_sata.7d      \
224                 pcn.7d          \
225                 radeon.7d       \
226                 ral.7d          \
227                 rtw.7d          \
228                 rum.7d          \
229                 rwd.7d          \
230                 rwn.7d          \
231                 sda.7d          \
232                 sdcard.7d       \
233                 sdhost.7d       \
234                 si3124.7d       \
235                 smbios.7d       \
236                 uath.7d         \
237                 ural.7d         \
238                 urtw.7d         \
239                 wpi.7d          \
240                 zyd.7d

242 _MANLINKS=      1394.7d         \
243                 allkmem.7d      \
244                 bscbus.7d       \
245                 fdc.7d          \
246                 firewire.7d     \
247                 hwa1480_fw.7d   \
248                 i2bsc.7d        \
249                 kmem.7d         \
250                 lo0.7d          \
251                 ticots.7d       \
252                 ticotsord.7d    \
253                 urandom.7d      \
254                 usb.7d          \
255                 uwb.7d

257 sparc_MANLINKS= drmach.7d       \
258                 ngdr.7d         \
259                 ngdrmach.7d
```

```
261 MANFILES =      $(_MANFILES) $($(MACH)_MANFILES)
262 MANLINKS =      $(_MANLINKS) $($(MACH)_MANLINKS)

264 bscbus.7d       := LINKSRC = bscv.7d
265 i2bsc.7d        := LINKSRC = bscv.7d

267 drmach.7d       := LINKSRC = dr.7d
268 ngdr.7d         := LINKSRC = dr.7d
269 ngdrmach.7d     := LINKSRC = dr.7d

271 fdc.7d          := LINKSRC = fd.7d

273 1394.7d         := LINKSRC = ieee1394.7d
274 firewire.7d     := LINKSRC = ieee1394.7d

276 lo0.7d          := LINKSRC = ipnet.7d

278 allkmem.7d      := LINKSRC = mem.7d
279 kmem.7d         := LINKSRC = mem.7d

281 urandom.7d      := LINKSRC = random.7d

283 ticots.7d       := LINKSRC = ticlts.7d
284 ticotsord.7d    := LINKSRC = ticlts.7d

286 usb.7d          := LINKSRC = usba.7d

288 uwb.7d          := LINKSRC = uwba.7d

290 hwa1480_fw.7d   := LINKSRC = wusb_df.7d

292 .KEEP_STATE:

294 include         $(SRC)/man/Makefile.man

296 install:        $(ROOTMANFILES) $(ROOTMANLINKS)
```

```
**********************************************************
    3679 Thu Jun 12 17:42:14 2014
new/usr/src/man/man7d/mpt_sas3.7d
Added man page.
**********************************************************
'\" te
.\" Copyright (c) 2009, Sun Microsystems, Inc. All Rights Reserved
.\" The contents of this file are subject to the terms of the Common Development
.\" or http://www.opensolaris.org/os/licensing.  See the License for the specifi
.\" the following below this CDDL HEADER, with the fields enclosed by brackets "
.TH MPT_SAS3 7D "Jun 2, 2014"
.SH NAME
mpt_sas3 \- SAS-3 host bus adapter driver
.SH SYNOPSIS
.sp
.in +2
.nf
scsi@unit-address
.fi
.in -2

.SH DESCRIPTION
.sp
.LP
The \fBmpt_sas3\fR host bus adapter driver is a nexus driver that supports the
LSI SAS300x/3108 series of chips. These chips support SAS/SATA interfaces,
including tagged and untagged queuing, multiple MSI-X interrupts, LSI
fastpath and SATA 3G/6G/12G.
.SS "Configuration"
.sp
.LP
The \fBmpt_sas3\fR driver is configured by defining properties in
\fBmpt_sas3.conf\fR. These properties override the global SCSI settings. The
\fBmpt_sas3\fR driver supports one modifiable property:
.sp
.ne 2
.na
\fB\fBmpxio-disable\fR\fR
.ad
.sp .6
.RS 4n
Solaris I/O multipathing is enabled or disabled on SAS devices with the
\fBmpxio-disable\fR property. Specifying \fBmpxio-disable="no"\fR activates I/O
multipathing, while \fBmpxio-disable="yes"\fR disables I/O multipathing.
.sp
Solaris I/O multipathing can be enabled or disabled on a per port basis. Per
port settings override the global setting for the specified ports.
.sp
The following example shows how to disable multipathing on port 0 whose parent
is \fB/pci@0,0/pci8086,2940@1c/pci1000,90@0\fR:
.sp
.in +2
.nf
name="mpt_sas3"    parent="/pci@0,0/pci8086,2940@1c/pci1000,90@0"
mpxio-disable="yes";
.fi
.in -2

The \fBmpt_sas3\fR host bus adapter driver is also capable of driving older
SAS2 controllers cards such as 2008 - 2308. Replacing the older mpt_sas driver
with mpt_sas3 will automatically enhance the performance where the card
is capable.

.RE

.SH EXAMPLES
```



```
.LP
\fBExample 1 \fRUsing the \fBmpt_sas3\fR Configuration File to Disable MPXIO
.sp
.LP
Create a file called \fB/kernel/drv/mpt_sas3.conf\fR and add the following line:

.sp
.in +2
.nf
name="mpt_sas3"    parent="/pci@0,0/pci8086,2940@1c/pci1000,90@0"
mpxio-disable="yes";
.fi
.in -2

.SH FILES
.sp
.ne 2
.na
\fB\fB/kernel/drv/mpt_sas3\fR\fR
.ad
.sp .6
.RS 4n
32-bit ELF kernel module
.RE

.sp
.ne 2
.na
\fB\fB/kernel/drv/sparcv9/mpt_sas3\fR\fR
.ad
.sp .6
.RS 4n
64-bit SPARC ELF kernel module
.RE

.sp
.ne 2
.na
\fB\fB/kernel/drv/amd64/mpt_sas3\fR\fR
.ad
.sp .6
.RS 4n
64-bit x86 ELF kernel module
.RE

.sp
.ne 2
.na
\fB\fB/kernel/drv/mpt_sas3.conf\fR\fR
.ad
.sp .6
.RS 4n
Optional configuration file
.RE

.SH ATTRIBUTES
.sp
.LP
See \fBattributes\fR(5) for a description of the following attributes:
.sp

.sp
.TS
box;
l | l
l | l .
```

```
128 ATTRIBUTE TYPE  ATTRIBUTE VALUE
129 _
130 Architecture    SPARC, x86
131 .TE

133 .SH SEE ALSO
134 .sp
135 .LP
136 \fBprtconf\fR(1M), \fBdriver.conf\fR(4), \fBpci\fR(4), \fBattributes\fR(5),
137 \fBscsi_abort\fR(9F), \fBscsi_device\fR(9S), \fBscsi_extended_sense\fR(9S),
138 \fBscsi_inquiry\fR(9S), \fBscsi_hba_attach_setup\fR(9F), \fBupdate_drv\fR(1M),
139 \fBscsi_ifgetcap\fR(9F), \fBscsi_ifsetcap\fR(9F), \fBscsi_pkt\fR(9S),
140 \fBscsi_reset\fR(9F), \fBscsi_sync_pkt\fR(9F), \fBscsi_transport\fR(9F),
141 #endif /* ! codereview */
```

```
**********************************************************
   21467 Thu Jun 12 17:42:14 2014
new/usr/src/pkg/manifests/developer-debug-mdb.mf
Add rolling buffer for *all* debug messages.
Improve mdb module and seperate out into mpt_sas3.
**********************************************************
```

```
   1 #
   2 # CDDL HEADER START
   3 #
   4 # The contents of this file are subject to the terms of the
   5 # Common Development and Distribution License (the "License").
   6 # You may not use this file except in compliance with the License.
   7 #
   8 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9 # or http://www.opensolaris.org/os/licensing.
  10 # See the License for the specific language governing permissions
  11 # and limitations under the License.
  12 #
  13 # When distributing Covered Code, include this CDDL HEADER in each
  14 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15 # If applicable, add the following below this CDDL HEADER, with the
  16 # fields enclosed by brackets "[]" replaced with your own identifying
  17 # information: Portions Copyright [yyyy] [name of copyright owner]
  18 #
  19 # CDDL HEADER END
  20 #

  22 #
  23 # Copyright (c) 2010, Oracle and/or its affiliates. All rights reserved.
  24 #

  26 set name=pkg.fmri value=pkg:/developer/debug/mdb@$(PKGVERS)
  27 set name=pkg.description value="Modular Debugger (MDB)"
  28 set name=pkg.summary value="Modular Debugger"
  29 set name=info.classification \
  30     value=org.opensolaris.category.2008:Development/System
  31 set name=variant.arch value=$(ARCH)
  32 dir path=kernel group=sys
  33 dir path=kernel/kmdb group=sys
  34 dir path=kernel/kmdb/$(ARCH64) group=sys
  35 dir path=platform group=sys variant.opensolaris.zone=global
  36 $(i386_ONLY)dir path=platform/i86pc group=sys variant.opensolaris.zone=global
  37 $(i386_ONLY)dir path=platform/i86pc/kernel group=sys \
  38     variant.opensolaris.zone=global
  39 $(i386_ONLY)dir path=platform/i86pc/kernel/kmdb group=sys
  40 $(i386_ONLY)dir path=platform/i86pc/kernel/kmdb/$(ARCH64) group=sys
  41 $(i386_ONLY)dir path=platform/i86xpv group=sys variant.opensolaris.zone=global
  42 $(i386_ONLY)dir path=platform/i86xpv/kernel group=sys \
  43     variant.opensolaris.zone=global
  44 $(i386_ONLY)dir path=platform/i86xpv/kernel/kmdb group=sys
  45 $(i386_ONLY)dir path=platform/i86xpv/kernel/kmdb/$(ARCH64) group=sys
  46 $(sparc_ONLY)dir path=platform/sun4u group=sys variant.opensolaris.zone=global
  47 $(sparc_ONLY)dir path=platform/sun4u/kernel group=sys \
  48     variant.opensolaris.zone=global
  49 $(sparc_ONLY)dir path=platform/sun4u/kernel/kmdb group=sys
  50 $(sparc_ONLY)dir path=platform/sun4u/kernel/kmdb/$(ARCH64) group=sys
  51 $(sparc_ONLY)dir path=platform/sun4v group=sys variant.opensolaris.zone=global
  52 $(sparc_ONLY)dir path=platform/sun4v/kernel group=sys \
  53     variant.opensolaris.zone=global
  54 $(sparc_ONLY)dir path=platform/sun4v/kernel/kmdb group=sys
  55 $(sparc_ONLY)dir path=platform/sun4v/kernel/kmdb/$(ARCH64) group=sys
  56 dir path=usr group=sys
  57 dir path=usr/bin
  58 dir path=usr/bin/$(ARCH32)
  59 dir path=usr/bin/$(ARCH64)
  60 dir path=usr/include
```



```
  61 dir path=usr/include/sys
  62 dir path=usr/lib
  63 dir path=usr/lib/mdb group=sys
  64 dir path=usr/lib/mdb/kvm group=sys
  65 dir path=usr/lib/mdb/kvm/$(ARCH64) group=sys
  66 dir path=usr/lib/mdb/proc group=sys
  67 $(sparc_ONLY)dir path=usr/lib/mdb/proc/$(ARCH64) group=sys
  68 $(i386_ONLY)dir path=usr/lib/mdb/proc/$(ARCH64)
  69 dir path=usr/lib/mdb/raw group=sys
  70 dir path=usr/platform group=sys
  71 $(i386_ONLY)dir path=usr/platform/i86pc group=sys
  72 $(i386_ONLY)dir path=usr/platform/i86pc/lib
  73 $(i386_ONLY)dir path=usr/platform/i86pc/lib/mdb group=sys
  74 $(i386_ONLY)dir path=usr/platform/i86pc/lib/mdb/kvm group=sys
  75 $(i386_ONLY)dir path=usr/platform/i86pc/lib/mdb/kvm/$(ARCH64) group=sys
  76 $(i386_ONLY)dir path=usr/platform/i86xpv group=sys
  77 $(i386_ONLY)dir path=usr/platform/i86xpv/lib
  78 $(i386_ONLY)dir path=usr/platform/i86xpv/lib/mdb group=sys
  79 $(i386_ONLY)dir path=usr/platform/i86xpv/lib/mdb/kvm group=sys
  80 $(i386_ONLY)dir path=usr/platform/i86xpv/lib/mdb/kvm/$(ARCH64) group=sys
  81 $(sparc_ONLY)dir path=usr/platform/sun4u group=sys
  82 $(sparc_ONLY)dir path=usr/platform/sun4u/lib
  83 $(sparc_ONLY)dir path=usr/platform/sun4u/lib/mdb group=sys
  84 $(sparc_ONLY)dir path=usr/platform/sun4u/lib/mdb/kvm group=sys
  85 $(sparc_ONLY)dir path=usr/platform/sun4u/lib/mdb/kvm/$(ARCH64) group=sys
  86 $(sparc_ONLY)dir path=usr/platform/sun4v group=sys
  87 $(sparc_ONLY)dir path=usr/platform/sun4v/lib
  88 $(sparc_ONLY)dir path=usr/platform/sun4v/lib/mdb group=sys
  89 $(sparc_ONLY)dir path=usr/platform/sun4v/lib/mdb/kvm group=sys
  90 $(sparc_ONLY)dir path=usr/platform/sun4v/lib/mdb/kvm/$(ARCH64) group=sys
  91 dir path=usr/share/man
  92 dir path=usr/share/man/man1
  93 file path=kernel/kmdb/$(ARCH64)/arp group=sys mode=0555
  94 file path=kernel/kmdb/$(ARCH64)/cpc group=sys mode=0555
  95 $(i386_ONLY)file path=kernel/kmdb/$(ARCH64)/cpu.generic group=sys mode=0555
  96 $(i386_ONLY)file path=kernel/kmdb/$(ARCH64)/cpu_ms.AuthenticAMD.15 group=sys \
  97     mode=0555
  98 file path=kernel/kmdb/$(ARCH64)/crypto group=sys mode=0555
  99 file path=kernel/kmdb/$(ARCH64)/genunix group=sys mode=0555
 100 file path=kernel/kmdb/$(ARCH64)/hook group=sys mode=0555
 101 $(sparc_ONLY)file path=kernel/kmdb/$(ARCH64)/intr group=sys mode=0555
 102 file path=kernel/kmdb/$(ARCH64)/ip group=sys mode=0555
 103 file path=kernel/kmdb/$(ARCH64)/ipc group=sys mode=0555
 104 file path=kernel/kmdb/$(ARCH64)/ipp group=sys mode=0555
 105 file path=kernel/kmdb/$(ARCH64)/krtld group=sys mode=0555
 106 file path=kernel/kmdb/$(ARCH64)/lofs group=sys mode=0555
 107 file path=kernel/kmdb/$(ARCH64)/logindmux group=sys mode=0555
 108 file path=kernel/kmdb/$(ARCH64)/mac group=sys mode=0555
 109 file path=kernel/kmdb/$(ARCH64)/md group=sys mode=0555
 110 file path=kernel/kmdb/$(ARCH64)/mdb_ds group=sys mode=0555
 111 file path=kernel/kmdb/$(ARCH64)/mpt group=sys mode=0555
 112 file path=kernel/kmdb/$(ARCH64)/mpt_sas group=sys mode=0555
 113 file path=kernel/kmdb/$(ARCH64)/mpt_sas3 group=sys mode=0555
 114 #endif /* ! codereview */
 115 file path=kernel/kmdb/$(ARCH64)/mr_sas group=sys mode=0555
 116 file path=kernel/kmdb/$(ARCH64)/nca group=sys mode=0555
 117 file path=kernel/kmdb/$(ARCH64)/neti group=sys mode=0555
 118 file path=kernel/kmdb/$(ARCH64)/nfs group=sys mode=0555
 119 file path=kernel/kmdb/$(ARCH64)/ptm group=sys mode=0555
 120 file path=kernel/kmdb/$(ARCH64)/random group=sys mode=0555
 121 file path=kernel/kmdb/$(ARCH64)/s1394 group=sys mode=0555
 122 $(i386_ONLY)file path=kernel/kmdb/$(ARCH64)/sata group=sys mode=0555
 123 file path=kernel/kmdb/$(ARCH64)/scsi_vhci group=sys mode=0555
 124 file path=kernel/kmdb/$(ARCH64)/sctp group=sys mode=0555
 125 file path=kernel/kmdb/$(ARCH64)/sd group=sys mode=0555
 126 file path=kernel/kmdb/$(ARCH64)/sockfs group=sys mode=0555
```

```
file path=kernel/kmdb/$(ARCH64)/specfs group=sys mode=0555
file path=kernel/kmdb/$(ARCH64)/sppp group=sys mode=0555
$(sparc_ONLY)file path=kernel/kmdb/$(ARCH64)/ssd group=sys mode=0555
file path=kernel/kmdb/$(ARCH64)/ufs group=sys mode=0555
$(i386_ONLY)file path=kernel/kmdb/$(ARCH64)/uhci group=sys mode=0555
file path=kernel/kmdb/$(ARCH64)/usba group=sys mode=0555
$(i386_ONLY)file path=kernel/kmdb/arp group=sys mode=0555
$(i386_ONLY)file path=kernel/kmdb/cpc group=sys mode=0555
$(i386_ONLY)file path=kernel/kmdb/cpu.generic group=sys mode=0555
$(i386_ONLY)file path=kernel/kmdb/cpu_ms.AuthenticAMD.15 group=sys mode=0555
$(i386_ONLY)file path=kernel/kmdb/crypto group=sys mode=0555
$(i386_ONLY)file path=kernel/kmdb/genunix group=sys mode=0555
$(i386_ONLY)file path=kernel/kmdb/hook group=sys mode=0555
$(i386_ONLY)file path=kernel/kmdb/ip group=sys mode=0555
$(i386_ONLY)file path=kernel/kmdb/ipc group=sys mode=0555
$(i386_ONLY)file path=kernel/kmdb/ipp group=sys mode=0555
$(i386_ONLY)file path=kernel/kmdb/krtld group=sys mode=0555
$(i386_ONLY)file path=kernel/kmdb/lofs group=sys mode=0555
$(i386_ONLY)file path=kernel/kmdb/logindmux group=sys mode=0555
$(i386_ONLY)file path=kernel/kmdb/mac group=sys mode=0555
$(i386_ONLY)file path=kernel/kmdb/md group=sys mode=0555
$(i386_ONLY)file path=kernel/kmdb/mdb_ds group=sys mode=0555
$(i386_ONLY)file path=kernel/kmdb/mpt group=sys mode=0555
$(i386_ONLY)file path=kernel/kmdb/mpt_sas group=sys mode=0555
$(i386_ONLY)file path=kernel/kmdb/mpt_sas3 group=sys mode=0555
#endif /* ! codereview */
$(i386_ONLY)file path=kernel/kmdb/mr_sas group=sys mode=0555
$(i386_ONLY)file path=kernel/kmdb/nca group=sys mode=0555
$(i386_ONLY)file path=kernel/kmdb/neti group=sys mode=0555
$(i386_ONLY)file path=kernel/kmdb/nfs group=sys mode=0555
$(i386_ONLY)file path=kernel/kmdb/ptm group=sys mode=0555
$(i386_ONLY)file path=kernel/kmdb/random group=sys mode=0555
$(i386_ONLY)file path=kernel/kmdb/s1394 group=sys mode=0555
$(i386_ONLY)file path=kernel/kmdb/sata group=sys mode=0555
$(i386_ONLY)file path=kernel/kmdb/scsi_vhci group=sys mode=0555
$(i386_ONLY)file path=kernel/kmdb/sctp group=sys mode=0555
$(i386_ONLY)file path=kernel/kmdb/sd group=sys mode=0555
$(i386_ONLY)file path=kernel/kmdb/sockfs group=sys mode=0555
$(i386_ONLY)file path=kernel/kmdb/specfs group=sys mode=0555
$(i386_ONLY)file path=kernel/kmdb/sppp group=sys mode=0555
$(i386_ONLY)file path=kernel/kmdb/ufs group=sys mode=0555
$(i386_ONLY)file path=kernel/kmdb/uhci group=sys mode=0555
$(i386_ONLY)file path=kernel/kmdb/usba group=sys mode=0555
$(i386_ONLY)file path=platform/i86pc/kernel/kmdb/$(ARCH64)/apix group=sys \
    mode=0555
$(i386_ONLY)file path=platform/i86pc/kernel/kmdb/$(ARCH64)/pcplusmp group=sys \
    mode=0555
$(i386_ONLY)file path=platform/i86pc/kernel/kmdb/$(ARCH64)/unix group=sys \
    mode=0555
$(i386_ONLY)file path=platform/i86pc/kernel/kmdb/$(ARCH64)/uppc group=sys \
    mode=0555
$(i386_ONLY)file path=platform/i86pc/kernel/kmdb/apix group=sys mode=0555
$(i386_ONLY)file path=platform/i86pc/kernel/kmdb/pcplusmp group=sys mode=0555
$(i386_ONLY)file path=platform/i86pc/kernel/kmdb/unix group=sys mode=0555
$(i386_ONLY)file path=platform/i86pc/kernel/kmdb/uppc group=sys mode=0555
$(i386_ONLY)file path=platform/i86xpv/kernel/kmdb/$(ARCH64)/unix group=sys \
    mode=0555
$(i386_ONLY)file path=platform/i86xpv/kernel/kmdb/$(ARCH64)/xpv_psm group=sys \
    mode=0555
$(i386_ONLY)file path=platform/i86xpv/kernel/kmdb/$(ARCH64)/xpv_uppc group=sys \
    mode=0555
$(i386_ONLY)file path=platform/i86xpv/kernel/kmdb/unix group=sys mode=0555
$(i386_ONLY)file path=platform/i86xpv/kernel/kmdb/xpv_psm group=sys mode=0555
$(i386_ONLY)file path=platform/i86xpv/kernel/kmdb/xpv_uppc group=sys mode=0555
$(sparc_ONLY)file path=platform/sun4u/kernel/kmdb/$(ARCH64)/oplhwd group=sys \
    mode=0555
```



```
$(sparc_ONLY)file path=platform/sun4u/kernel/kmdb/$(ARCH64)/sgenv group=sys \
    mode=0555
$(sparc_ONLY)file path=platform/sun4u/kernel/kmdb/$(ARCH64)/sgsbbc group=sys \
    mode=0555
$(sparc_ONLY)file path=platform/sun4u/kernel/kmdb/$(ARCH64)/unix group=sys \
    mode=0555
$(sparc_ONLY)file path=platform/sun4v/kernel/kmdb/$(ARCH64)/errh group=sys \
    mode=0555
$(sparc_ONLY)file path=platform/sun4v/kernel/kmdb/$(ARCH64)/ldc group=sys \
    mode=0555
$(sparc_ONLY)file path=platform/sun4v/kernel/kmdb/$(ARCH64)/mdesc group=sys \
    mode=0555
$(sparc_ONLY)file path=platform/sun4v/kernel/kmdb/$(ARCH64)/unix group=sys \
    mode=0555
$(sparc_ONLY)file path=platform/sun4v/kernel/kmdb/$(ARCH64)/vdsk group=sys \
    mode=0555
file path=usr/bin/$(ARCH32)/mdb mode=0555
file path=usr/bin/$(ARCH64)/mdb mode=0555
file path=usr/include/sys/mdb_modapi.h
file path=usr/lib/mdb/kvm/$(ARCH64)/arp.so group=sys mode=0555
file path=usr/lib/mdb/kvm/$(ARCH64)/cpc.so group=sys mode=0555
$(i386_ONLY)file path=usr/lib/mdb/kvm/$(ARCH64)/cpu.generic.so group=sys \
    mode=0555
$(i386_ONLY)file path=usr/lib/mdb/kvm/$(ARCH64)/cpu_ms.AuthenticAMD.15.so \
    group=sys mode=0555
file path=usr/lib/mdb/kvm/$(ARCH64)/crypto.so group=sys mode=0555
file path=usr/lib/mdb/kvm/$(ARCH64)/genunix.so group=sys mode=0555
file path=usr/lib/mdb/kvm/$(ARCH64)/hook.so group=sys mode=0555
$(sparc_ONLY)file path=usr/lib/mdb/kvm/$(ARCH64)/intr.so group=sys mode=0555
file path=usr/lib/mdb/kvm/$(ARCH64)/ip.so group=sys mode=0555
file path=usr/lib/mdb/kvm/$(ARCH64)/ipc.so group=sys mode=0555
file path=usr/lib/mdb/kvm/$(ARCH64)/ipp.so group=sys mode=0555
file path=usr/lib/mdb/kvm/$(ARCH64)/krtld.so group=sys mode=0555
file path=usr/lib/mdb/kvm/$(ARCH64)/lofs.so group=sys mode=0555
file path=usr/lib/mdb/kvm/$(ARCH64)/logindmux.so group=sys mode=0555
file path=usr/lib/mdb/kvm/$(ARCH64)/mac.so group=sys mode=0555
file path=usr/lib/mdb/kvm/$(ARCH64)/md.so group=sys mode=0555
$(i386_ONLY)file path=usr/lib/mdb/kvm/$(ARCH64)/mdb_kb.so group=sys mode=0555
file path=usr/lib/mdb/kvm/$(ARCH64)/mdb_ks.so group=sys mode=0555
file path=usr/lib/mdb/kvm/$(ARCH64)/mpt.so group=sys mode=0555
file path=usr/lib/mdb/kvm/$(ARCH64)/mpt_sas.so group=sys mode=0555
file path=usr/lib/mdb/kvm/$(ARCH64)/mpt_sas3.so group=sys mode=0555
#endif /* ! codereview */
file path=usr/lib/mdb/kvm/$(ARCH64)/mr_sas.so group=sys mode=0555
file path=usr/lib/mdb/kvm/$(ARCH64)/nca.so group=sys mode=0555
file path=usr/lib/mdb/kvm/$(ARCH64)/neti.so group=sys mode=0555
file path=usr/lib/mdb/kvm/$(ARCH64)/nfs.so group=sys mode=0555
file path=usr/lib/mdb/kvm/$(ARCH64)/ptm.so group=sys mode=0555
file path=usr/lib/mdb/kvm/$(ARCH64)/random.so group=sys mode=0555
file path=usr/lib/mdb/kvm/$(ARCH64)/s1394.so group=sys mode=0555
$(i386_ONLY)file path=usr/lib/mdb/kvm/$(ARCH64)/sata.so group=sys mode=0555
file path=usr/lib/mdb/kvm/$(ARCH64)/scsi_vhci.so group=sys mode=0555
file path=usr/lib/mdb/kvm/$(ARCH64)/sctp.so group=sys mode=0555
file path=usr/lib/mdb/kvm/$(ARCH64)/sd.so group=sys mode=0555
file path=usr/lib/mdb/kvm/$(ARCH64)/sockfs.so group=sys mode=0555
file path=usr/lib/mdb/kvm/$(ARCH64)/specfs.so group=sys mode=0555
file path=usr/lib/mdb/kvm/$(ARCH64)/sppp.so group=sys mode=0555
$(sparc_ONLY)file path=usr/lib/mdb/kvm/$(ARCH64)/ssd.so group=sys mode=0555
file path=usr/lib/mdb/kvm/$(ARCH64)/ufs.so group=sys mode=0555
$(i386_ONLY)file path=usr/lib/mdb/kvm/$(ARCH64)/uhci.so group=sys mode=0555
file path=usr/lib/mdb/kvm/$(ARCH64)/usba.so group=sys mode=0555
$(i386_ONLY)file path=usr/lib/mdb/kvm/arp.so group=sys mode=0555
$(i386_ONLY)file path=usr/lib/mdb/kvm/cpc.so group=sys mode=0555
$(i386_ONLY)file path=usr/lib/mdb/kvm/cpu.generic.so group=sys mode=0555
$(i386_ONLY)file path=usr/lib/mdb/kvm/cpu_ms.AuthenticAMD.15.so group=sys \
    mode=0555
```

```
$(i386_ONLY)file path=usr/lib/mdb/kvm/crypto.so group=sys mode=0555
$(i386_ONLY)file path=usr/lib/mdb/kvm/genunix.so group=sys mode=0555
$(i386_ONLY)file path=usr/lib/mdb/kvm/hook.so group=sys mode=0555
$(i386_ONLY)file path=usr/lib/mdb/kvm/ip.so group=sys mode=0555
$(i386_ONLY)file path=usr/lib/mdb/kvm/ipc.so group=sys mode=0555
$(i386_ONLY)file path=usr/lib/mdb/kvm/ipp.so group=sys mode=0555
$(i386_ONLY)file path=usr/lib/mdb/kvm/krtld.so group=sys mode=0555
$(i386_ONLY)file path=usr/lib/mdb/kvm/lofs.so group=sys mode=0555
$(i386_ONLY)file path=usr/lib/mdb/kvm/logindmux.so group=sys mode=0555
$(i386_ONLY)file path=usr/lib/mdb/kvm/mac.so group=sys mode=0555
$(i386_ONLY)file path=usr/lib/mdb/kvm/md.so group=sys mode=0555
$(i386_ONLY)file path=usr/lib/mdb/kvm/mdb_kb.so group=sys mode=0555
$(i386_ONLY)file path=usr/lib/mdb/kvm/mdb_ks.so group=sys mode=0555
$(i386_ONLY)file path=usr/lib/mdb/kvm/mpt.so group=sys mode=0555
$(i386_ONLY)file path=usr/lib/mdb/kvm/mpt_sas.so group=sys mode=0555
$(i386_ONLY)file path=usr/lib/mdb/kvm/mpt_sas3.so group=sys mode=0555
#endif /* ! codereview */
$(i386_ONLY)file path=usr/lib/mdb/kvm/mr_sas.so group=sys mode=0555
$(i386_ONLY)file path=usr/lib/mdb/kvm/nca.so group=sys mode=0555
$(i386_ONLY)file path=usr/lib/mdb/kvm/neti.so group=sys mode=0555
$(i386_ONLY)file path=usr/lib/mdb/kvm/nfs.so group=sys mode=0555
$(i386_ONLY)file path=usr/lib/mdb/kvm/ptm.so group=sys mode=0555
$(i386_ONLY)file path=usr/lib/mdb/kvm/random.so group=sys mode=0555
$(i386_ONLY)file path=usr/lib/mdb/kvm/s1394.so group=sys mode=0555
$(i386_ONLY)file path=usr/lib/mdb/kvm/sata.so group=sys mode=0555
$(i386_ONLY)file path=usr/lib/mdb/kvm/scsi_vhci.so group=sys mode=0555
$(i386_ONLY)file path=usr/lib/mdb/kvm/sctp.so group=sys mode=0555
$(i386_ONLY)file path=usr/lib/mdb/kvm/sd.so group=sys mode=0555
$(i386_ONLY)file path=usr/lib/mdb/kvm/sockfs.so group=sys mode=0555
$(i386_ONLY)file path=usr/lib/mdb/kvm/specfs.so group=sys mode=0555
$(i386_ONLY)file path=usr/lib/mdb/kvm/sppp.so group=sys mode=0555
$(i386_ONLY)file path=usr/lib/mdb/kvm/ufs.so group=sys mode=0555
$(i386_ONLY)file path=usr/lib/mdb/kvm/uhci.so group=sys mode=0555
$(i386_ONLY)file path=usr/lib/mdb/kvm/usba.so group=sys mode=0555
file path=usr/lib/mdb/proc/$(ARCH64)/ld.so group=sys mode=0555
file path=usr/lib/mdb/proc/$(ARCH64)/libavl.so group=sys mode=0555
file path=usr/lib/mdb/proc/$(ARCH64)/libc.so group=sys mode=0555
file path=usr/lib/mdb/proc/$(ARCH64)/libcmdutils.so group=sys mode=0555
file path=usr/lib/mdb/proc/$(ARCH64)/libnvpair.so group=sys mode=0555
file path=usr/lib/mdb/proc/$(ARCH64)/libproc.so group=sys mode=0555
file path=usr/lib/mdb/proc/$(ARCH64)/libpython2.6.so group=sys mode=0555
file path=usr/lib/mdb/proc/$(ARCH64)/libsysevent.so group=sys mode=0555
file path=usr/lib/mdb/proc/$(ARCH64)/libtopo.so group=sys mode=0555
file path=usr/lib/mdb/proc/$(ARCH64)/libumem.so group=sys mode=0555
file path=usr/lib/mdb/proc/$(ARCH64)/libuutil.so group=sys mode=0555
file path=usr/lib/mdb/proc/$(ARCH64)/mdb_ds.so group=sys mode=0555
$(i386_ONLY)file path=usr/lib/mdb/proc/$(ARCH64)/mdb_test.so group=sys \
    mode=0555
file path=usr/lib/mdb/proc/ld.so group=sys mode=0555
file path=usr/lib/mdb/proc/libavl.so group=sys mode=0555
file path=usr/lib/mdb/proc/libc.so group=sys mode=0555
file path=usr/lib/mdb/proc/libcmdutils.so group=sys mode=0555
file path=usr/lib/mdb/proc/libnvpair.so group=sys mode=0555
file path=usr/lib/mdb/proc/libproc.so group=sys mode=0555
file path=usr/lib/mdb/proc/libpython2.6.so group=sys mode=0555
file path=usr/lib/mdb/proc/libsysevent.so group=sys mode=0555
file path=usr/lib/mdb/proc/libtopo.so group=sys mode=0555
file path=usr/lib/mdb/proc/libumem.so group=sys mode=0555
file path=usr/lib/mdb/proc/libuutil.so group=sys mode=0555
file path=usr/lib/mdb/proc/mdb_ds.so group=sys mode=0555
file path=usr/lib/mdb/proc/svc.configd.so group=sys mode=0555
file path=usr/lib/mdb/proc/svc.startd.so group=sys mode=0555
$(i386_ONLY)file path=usr/platform/i86pc/lib/mdb/kvm/$(ARCH64)/apix.so \
    group=sys mode=0555
$(i386_ONLY)file path=usr/platform/i86pc/lib/mdb/kvm/$(ARCH64)/pcplusmp.so \
    group=sys mode=0555
```



```
$(i386_ONLY)file path=usr/platform/i86pc/lib/mdb/kvm/$(ARCH64)/unix.so \
    group=sys mode=0555
$(i386_ONLY)file path=usr/platform/i86pc/lib/mdb/kvm/$(ARCH64)/uppc.so \
    group=sys mode=0555
$(i386_ONLY)file path=usr/platform/i86pc/lib/mdb/kvm/apix.so group=sys \
    mode=0555
$(i386_ONLY)file path=usr/platform/i86pc/lib/mdb/kvm/pcplusmp.so group=sys \
    mode=0555
$(i386_ONLY)file path=usr/platform/i86pc/lib/mdb/kvm/unix.so group=sys \
    mode=0555
$(i386_ONLY)file path=usr/platform/i86pc/lib/mdb/kvm/uppc.so group=sys \
    mode=0555
$(i386_ONLY)file path=usr/platform/i86xpv/lib/mdb/kvm/$(ARCH64)/unix.so \
    group=sys mode=0555
$(i386_ONLY)file path=usr/platform/i86xpv/lib/mdb/kvm/$(ARCH64)/xpv.so \
    group=sys mode=0555
$(i386_ONLY)file path=usr/platform/i86xpv/lib/mdb/kvm/$(ARCH64)/xpv_psm.so \
    group=sys mode=0555
$(i386_ONLY)file path=usr/platform/i86xpv/lib/mdb/kvm/$(ARCH64)/xpv_uppc.so \
    group=sys mode=0555
$(i386_ONLY)file path=usr/platform/i86xpv/lib/mdb/kvm/unix.so group=sys \
    mode=0555
$(i386_ONLY)file path=usr/platform/i86xpv/lib/mdb/kvm/xpv.so group=sys \
    mode=0555
$(i386_ONLY)file path=usr/platform/i86xpv/lib/mdb/kvm/xpv_psm.so group=sys \
    mode=0555
$(i386_ONLY)file path=usr/platform/i86xpv/lib/mdb/kvm/xpv_uppc.so group=sys \
    mode=0555
$(sparc_ONLY)file path=usr/platform/sun4u/lib/mdb/kvm/$(ARCH64)/oplhwd.so \
    group=sys mode=0555
$(sparc_ONLY)file path=usr/platform/sun4u/lib/mdb/kvm/$(ARCH64)/sgenv.so \
    group=sys mode=0555
$(sparc_ONLY)file path=usr/platform/sun4u/lib/mdb/kvm/$(ARCH64)/sgsbbc.so \
    group=sys mode=0555
$(sparc_ONLY)file path=usr/platform/sun4u/lib/mdb/kvm/$(ARCH64)/unix.so \
    group=sys mode=0555
$(sparc_ONLY)file path=usr/platform/sun4v/lib/mdb/kvm/$(ARCH64)/errh.so \
    group=sys mode=0555
$(sparc_ONLY)file path=usr/platform/sun4v/lib/mdb/kvm/$(ARCH64)/ldc.so \
    group=sys mode=0555
$(sparc_ONLY)file path=usr/platform/sun4v/lib/mdb/kvm/$(ARCH64)/mdesc.so \
    group=sys mode=0555
$(sparc_ONLY)file path=usr/platform/sun4v/lib/mdb/kvm/$(ARCH64)/unix.so \
    group=sys mode=0555
$(sparc_ONLY)file path=usr/platform/sun4v/lib/mdb/kvm/$(ARCH64)/vdsk.so \
    group=sys mode=0555
file path=usr/share/man/man1/adb.1
file path=usr/share/man/man1/kmdb.1
file path=usr/share/man/man1/mdb.1
hardlink path=usr/bin/$(ARCH32)/adb target=../../../usr/bin/$(ARCH32)/mdb
hardlink path=usr/bin/$(ARCH64)/adb target=../../../usr/bin/$(ARCH64)/mdb
hardlink path=usr/bin/adb target=../../usr/lib/isaexec
hardlink path=usr/bin/mdb target=../../usr/lib/isaexec
legacy pkg=SUNWmdb desc="Modular Debugger (MDB)" name="Modular Debugger"
legacy pkg=SUNWmdbr desc="Modular Debugger (MDB) (Root)" \
    name="Modular Debugger (Root)"
license cr_Sun license=cr_Sun
license lic_CDDL license=lic_CDDL
license usr/src/common/bzip2/LICENSE license=usr/src/common/bzip2/LICENSE
license usr/src/uts/common/io/mr_sas/THIRDPARTYLICENSE \
    license=usr/src/uts/common/io/mr_sas/THIRDPARTYLICENSE
license usr/src/uts/common/zmod/THIRDPARTYLICENSE \
    license=usr/src/uts/common/zmod/THIRDPARTYLICENSE
$(sparc_ONLY)link path=kernel/kmdb/$(ARCH64)/niumx target=intr
$(sparc_ONLY)link path=kernel/kmdb/$(ARCH64)/pcipsy target=intr
$(sparc_ONLY)link path=kernel/kmdb/$(ARCH64)/pcisch target=intr
```

```
391 $(sparc_ONLY)link path=kernel/kmdb/$(ARCH64)/px target=intr
392 $(sparc_ONLY)link path=usr/lib/mdb/kvm/$(ARCH64)/niumx.so target=intr.so
393 $(sparc_ONLY)link path=usr/lib/mdb/kvm/$(ARCH64)/pcipsy.so target=intr.so
394 $(sparc_ONLY)link path=usr/lib/mdb/kvm/$(ARCH64)/pcisch.so target=intr.so
395 $(sparc_ONLY)link path=usr/lib/mdb/kvm/$(ARCH64)/px.so target=intr.so
```

```
**********************************************************
    1924 Thu Jun 12 17:42:14 2014
new/usr/src/pkg/manifests/driver-storage-mpt_sas3.mf
Changes to enable driver to compile.
Header paths, object lists, etc.
**********************************************************
```

```
   1 #
   2 # CDDL HEADER START
   3 #
   4 # The contents of this file are subject to the terms of the
   5 # Common Development and Distribution License (the "License").
   6 # You may not use this file except in compliance with the License.
   7 #
   8 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9 # or http://www.opensolaris.org/os/licensing.
  10 # See the License for the specific language governing permissions
  11 # and limitations under the License.
  12 #
  13 # When distributing Covered Code, include this CDDL HEADER in each
  14 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15 # If applicable, add the following below this CDDL HEADER, with the
  16 # fields enclosed by brackets "[]" replaced with your own identifying
  17 # information: Portions Copyright [yyyy] [name of copyright owner]
  18 #
  19 # CDDL HEADER END
  20 #

  22 #
  23 # Copyright (c) 2010, Oracle and/or its affiliates. All rights reserved.
  24 #

  26 #
  27 # The default for payload-bearing actions in this package is to appear in the
  28 # global zone only.  See the include file for greater detail, as well as
  29 # information about overriding the defaults.
  30 #
  31 <include global_zone_only_component>
  32 set name=pkg.fmri value=pkg:/driver/storage/mpt_sas3@$(PKGVERS)
  33 set name=pkg.description value="LSI MPT SAS 3.0 Controller HBA Driver"
  34 set name=pkg.summary value="LSI MPT SAS 3.0 Controller HBA Driver"
  35 set name=info.classification \
  36     value=org.opensolaris.category.2008:Drivers/Storage
  37 set name=variant.arch value=$(ARCH)
  38 dir path=kernel group=sys
  39 dir path=kernel/drv group=sys
  40 dir path=kernel/drv/$(ARCH64) group=sys
  41 dir path=usr/share/man
  42 dir path=usr/share/man/man7d
  43 driver name=mpt_sas3 class=scsi-self-identifying \
  44     alias=pciex1000,90 \
  45     alias=pciex1000,97
  46 file path=kernel/drv/$(ARCH64)/mpt_sas3 group=sys
  47 $(i386_ONLY)file path=kernel/drv/mpt_sas3 group=sys
  48 file path=kernel/drv/mpt_sas3.conf group=sys preserve=true
  49 file path=usr/share/man/man7d/mpt_sas3.7d
  50 license lic_CDDL license=lic_CDDL
  51 #endif /* ! codereview */
```

```
**********************************************************
   44257 Thu Jun 12 17:42:14 2014
new/usr/src/uts/common/Makefile.files
Changes to enable driver to compile.
Header paths, object lists, etc.
**********************************************************
#
# 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) 2013 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         \
```

```
127                 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
```

```
UNISTAT_OBJS += spuni.o \
                spcs_s_k.o

NSKERN_OBJS +=  nsc_ddi.o \
                nsc_proc.o \
                nsc_raw.o \
                nsc_thread.o \
                nskernd.o

SV_OBJS +=      sv.o

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

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

#
#       Build up defines and paths.

ST_OBJS +=      st.o    st_conf.o

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

EMLXS_FW_OBJS +=        emlxs_fw.o

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

FCT_OBJS += discovery.o fct.o

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

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

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

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

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

ISCSIT_SHARED_OBJS += \
                iscsit_common.o

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

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

STMF_OBJS += lun_map.o stmf.o

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

SYSMSG_OBJS +=  sysmsg.o

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

TNF_OBJS +=     tnf_buf.o       tnf_trace.o     tnf_writer.o    trace_init.o \
```



```
                trace_funcs.o   tnf_probe.o     tnf.o

LOGINDMUX_OBJS += logindmux.o

DEVINFO_OBJS += devinfo.o

DEVPOLL_OBJS += devpoll.o

DEVPOOL_OBJS += devpool.o

I8042_OBJS +=   i8042.o

KB8042_OBJS +=  \
                at_keyprocess.o \
                kb8042.o        \
                kb8042_keytables.o

MOUSE8042_OBJS += mouse8042.o

FDC_OBJS +=     fdc.o

ASY_OBJS +=     asy.o

ECPP_OBJS +=    ecpp.o

VUIDM3P_OBJS += vuidmice.o vuidm3p.o

VUIDM4P_OBJS += vuidmice.o vuidm4p.o

VUIDM5P_OBJS += vuidmice.o vuidm5p.o

VUIDPS2_OBJS += vuidmice.o vuidps2.o

HPCSVC_OBJS += hpcsvc.o

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

PCIHPNEXUS_OBJS += pcihp.o

OPENEEPR_OBJS += openprom.o

RANDOM_OBJS += random.o

PSHOT_OBJS += pshot.o

GEN_DRV_OBJS += gen_drv.o

TCLIENT_OBJS += tclient.o

TPHCI_OBJS += tphci.o

TVHCI_OBJS += tvhci.o

EMUL64_OBJS += emul64.o emul64_bsd.o

FCP_OBJS += fcp.o

FCIP_OBJS += fcip.o

FCSM_OBJS += fcsm.o

FCTL_OBJS += fctl.o

FP_OBJS += fp.o

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

1053 QLC_FW_2200_OBJS += ql_fw_2200.o

1055 QLC_FW_2300_OBJS += ql_fw_2300.o

1057 QLC_FW_2400_OBJS += ql_fw_2400.o

1059 QLC_FW_2500_OBJS += ql_fw_2500.o

1061 QLC_FW_6322_OBJS += ql_fw_6322.o

1063 QLC_FW_8100_OBJS += ql_fw_8100.o

1065 QLGE_OBJS += qlge.o qlge_dbg.o qlge_flash.o qlge_fm.o qlge_gld.o qlge_mpi.o

1067 ZCONS_OBJS += zcons.o

1069 NV_SATA_OBJS += nv_sata.o

1071 SI3124_OBJS += si3124.o

1073 AHCI_OBJS += ahci.o

1075 PCIIDE_OBJS += pci-ide.o

1077 PCEPP_OBJS += pcepp.o

1079 CPC_OBJS += cpc.o

1081 CPUID_OBJS += cpuid_drv.o

1083 SYSEVENT_OBJS += sysevent.o

1085 BL_OBJS += bl.o

1087 DRM_OBJS += drm_sunmod.o drm_kstat.o drm_agpsupport.o \
1088             drm_auth.o drm_bufs.o drm_context.o drm_dma.o \
1089             drm_drawable.o drm_drv.o drm_fops.o drm_ioctl.o drm_irq.o \
1090             drm_lock.o drm_memory.o drm_msg.o drm_pci.o drm_scatter.o \
1091             drm_cache.o drm_gem.o drm_mm.o ati_pcigart.o

1093 FM_OBJS += devfm.o devfm_machdep.o

1095 RTLS_OBJS +=    rtls.o

1097 #
1098 #                       exec modules
1099 #
1100 AOUTEXEC_OBJS +=aout.o

1102 ELFEXEC_OBJS += elf.o elf_notes.o old_notes.o

1104 INTPEXEC_OBJS +=intp.o

1106 SHBINEXEC_OBJS +=shbin.o

1108 JAVAEXEC_OBJS +=java.o

1110 #
1111 #                       file system modules
1112 #
1113 AUTOFS_OBJS +=  auto_vfsops.o auto_vnops.o auto_subr.o auto_xdr.o auto_sys.o

1115 CACHEFS_OBJS += cachefs_cnode.o         cachefs_cod.o \
1116                 cachefs_dir.o           cachefs_dlog.o  cachefs_filegrp.o \
```



```
1117                 cachefs_fscache.o       cachefs_ioctl.o cachefs_log.o \
1118                 cachefs_module.o \
1119                 cachefs_noopc.o         cachefs_resource.o \
1120                 cachefs_strict.o \
1121                 cachefs_subr.o          cachefs_vfsops.o \
1122                 cachefs_vnops.o

1124 DCFS_OBJS +=    dc_vnops.o

1126 DEVFS_OBJS +=   devfs_subr.o    devfs_vfsops.o  devfs_vnops.o

1128 DEV_OBJS  +=    sdev_subr.o     sdev_vfsops.o   sdev_vnops.o    \
1129                 sdev_ptsops.o   sdev_zvolops.o  sdev_comm.o     \
1130                 sdev_profile.o  sdev_ncache.o   sdev_netops.o   \
1131                 sdev_ipnetops.o \
1132                 sdev_vtops.o

1134 CTFS_OBJS +=    ctfs_all.o ctfs_cdir.o ctfs_ctl.o ctfs_event.o \
1135                 ctfs_latest.o ctfs_root.o ctfs_sym.o ctfs_tdir.o ctfs_tmpl.o

1137 OBJFS_OBJS +=   objfs_vfs.o     objfs_root.o    objfs_common.o \
1138                 objfs_odir.o    objfs_data.o

1140 FDFS_OBJS +=    fdops.o

1142 FIFO_OBJS +=    fifosubr.o      fifovnops.o

1144 PIPE_OBJS +=    pipe.o

1146 HSFS_OBJS +=    hsfs_node.o     hsfs_subr.o     hsfs_vfsops.o   hsfs_vnops.o \
1147                 hsfs_susp.o     hsfs_rrip.o     hsfs_susp_subr.o

1149 LOFS_OBJS +=    lofs_subr.o     lofs_vfsops.o   lofs_vnops.o

1151 NAMEFS_OBJS +=  namevfs.o       namevno.o

1153 NFS_OBJS +=     nfs_client.o    nfs_common.o    nfs_dump.o \
1154                 nfs_subr.o      nfs_vfsops.o    nfs_vnops.o \
1155                 nfs_xdr.o       nfs_sys.o       nfs_strerror.o \
1156                 nfs3_vfsops.o   nfs3_vnops.o    nfs3_xdr.o \
1157                 nfs_acl_vnops.o nfs_acl_xdr.o   nfs4_vfsops.o \
1158                 nfs4_vnops.o    nfs4_xdr.o      nfs4_idmap.o \
1159                 nfs4_shadow.o   nfs4_subr.o \
1160                 nfs4_attr.o     nfs4_rnode.o    nfs4_client.o \
1161                 nfs4_acache.o   nfs4_common.o   nfs4_client_state.o \
1162                 nfs4_callback.o nfs4_recovery.o nfs4_client_secinfo.o \
1163                 nfs4_client_debug.o     nfs_stats.o \
1164                 nfs4_acl.o      nfs4_stub_vnops.o       nfs_cmd.o

1166 NFSSRV_OBJS +=  nfs_server.o    nfs_srv.o       nfs3_srv.o \
1167                 nfs_acl_srv.o   nfs_auth.o      nfs_auth_xdr.o \
1168                 nfs_export.o    nfs_log.o       nfs_log_xdr.o \
1169                 nfs4_srv.o      nfs4_state.o    nfs4_srv_attr.o \
1170                 nfs4_srv_ns.o   nfs4_db.o       nfs4_srv_deleg.o \
1171                 nfs4_deleg_ops.o nfs4_srv_readdir.o nfs4_dispatch.o

1173 SMBSRV_SHARED_OBJS += \
1174                 smb_inet.o \
1175                 smb_match.o \
1176                 smb_msgbuf.o \
1177                 smb_oem.o \
1178                 smb_string.o \
1179                 smb_utf8.o \
1180                 smb_door_legacy.o \
1181                 smb_xdr.o \
1182                 smb_token.o \
```

```
1183                    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_bookmark.o         \
1347         dsl_dir.o              \
1348         dsl_dataset.o          \
1349         dsl_deadlist.o         \
1350         dsl_destroy.o          \
1351         dsl_pool.o             \
1352         dsl_synctask.o         \
1353         dsl_userhold.o         \
1354         dmu_zfetch.o           \
1355         dsl_deleg.o            \
1356         dsl_prop.o             \
1357         dsl_scan.o             \
1358         zfeature.o             \
1359         gzip.o                 \
1360         lz4.o                  \
1361         lzjb.o                 \
1362         metaslab.o             \
1363         range_tree.o           \
1364         refcount.o             \
1365         rrwlock.o              \
1366         sa.o                   \
1367         sha256.o               \
1368         spa.o                  \
1369         spa_config.o           \
1370         spa_errlog.o           \
1371         spa_history.o          \
1372         spa_misc.o             \
1373         space_map.o            \
1374         space_reftree.o        \
1375         txg.o                  \
1376         uberblock.o            \
1377         unique.o               \
1378         vdev.o                 \
1379         vdev_cache.o           \
1380         vdev_file.o            \
```



```
1381         vdev_label.o           \
1382         vdev_mirror.o          \
1383         vdev_missing.o         \
1384         vdev_queue.o           \
1385         vdev_raidz.o           \
1386         vdev_root.o            \
1387         zap.o                  \
1388         zap_leaf.o             \
1389         zap_micro.o            \
1390         zfs_byteswap.o         \
1391         zfs_debug.o            \
1392         zfs_fm.o               \
1393         zfs_fuid.o             \
1394         zfs_sa.o               \
1395         zfs_znode.o            \
1396         zil.o                  \
1397         zio.o                  \
1398         zio_checksum.o         \
1399         zio_compress.o         \
1400         zio_inject.o           \
1401         zle.o                  \
1402         zrlock.o

1404 ZFS_SHARED_OBJS +=             \
1405         zfeature_common.o      \
1406         zfs_comutil.o          \
1407         zfs_deleg.o            \
1408         zfs_fletcher.o         \
1409         zfs_namecheck.o        \
1410         zfs_prop.o             \
1411         zpool_prop.o           \
1412         zprop_common.o

1414 ZFS_OBJS +=                    \
1415         $(ZFS_COMMON_OBJS)     \
1416         $(ZFS_SHARED_OBJS)     \
1417         vdev_disk.o            \
1418         zfs_acl.o              \
1419         zfs_ctldir.o           \
1420         zfs_dir.o              \
1421         zfs_ioctl.o            \
1422         zfs_log.o              \
1423         zfs_onexit.o           \
1424         zfs_replay.o           \
1425         zfs_rlock.o            \
1426         zfs_vfsops.o           \
1427         zfs_vnops.o            \
1428         zvol.o

1430 ZUT_OBJS +=                    \
1431         zut.o

1433 #
1434 #                       streams modules
1435 #
1436 BUFMOD_OBJS     +=      bufmod.o

1438 CONNLD_OBJS +=  connld.o

1440 DEDUMP_OBJS +=  dedump.o

1442 DRCOMPAT_OBJS +=        drcompat.o

1444 LDLINUX_OBJS += ldlinux.o

1446 LDTERM_OBJS +=  ldterm.o uwidth.o
```

```
1448 PCKT_OBJS +=    pckt.o

1450 PFMOD_OBJS +=   pfmod.o

1452 PTEM_OBJS +=    ptem.o

1454 REDIRMOD_OBJS += strredirm.o

1456 TIMOD_OBJS +=   timod.o

1458 TIRDWR_OBJS +=  tirdwr.o

1460 TTCOMPAT_OBJS +=ttcompat.o

1462 LOG_OBJS +=     log.o

1464 PIPEMOD_OBJS += pipemod.o

1466 RPCMOD_OBJS +=  rpcmod.o        clnt_cots.o     clnt_clts.o \
1467                 clnt_gen.o      clnt_perr.o     mt_rpcinit.o    rpc_calmsg.o \
1468                 rpc_prot.o      rpc_sztypes.o   rpc_subr.o      rpcb_prot.o \
1469                 svc.o           svc_clts.o      svc_gen.o       svc_cots.o \
1470                 rpcsys.o        xdr_sizeof.o    clnt_rdma.o     svc_rdma.o \
1471                 xdr_rdma.o      rdma_subr.o     xdrrdma_sizeof.o

1473 KLMMOD_OBJS +=  klmmod.o \
1474                 nlm_impl.o \
1475                 nlm_rpc_handle.o \
1476                 nlm_dispatch.o \
1477                 nlm_rpc_svc.o \
1478                 nlm_client.o \
1479                 nlm_service.o \
1480                 nlm_prot_clnt.o \
1481                 nlm_prot_xdr.o \
1482                 nlm_rpc_clnt.o \
1483                 nsm_addr_clnt.o \
1484                 nsm_addr_xdr.o \
1485                 sm_inter_clnt.o \
1486                 sm_inter_xdr.o

1488 KLMOPS_OBJS +=  klmops.o

1490 TLIMOD_OBJS +=  tlimod.o        t_kalloc.o      t_kbind.o       t_kclose.o \
1491                 t_kconnect.o    t_kfree.o       t_kgtstate.o    t_kopen.o \
1492                 t_krcvudat.o    t_ksndudat.o    t_kspoll.o      t_kunbind.o \
1493                 t_kutil.o

1495 RLMOD_OBJS += rlmod.o

1497 TELMOD_OBJS += telmod.o

1499 CRYPTMOD_OBJS += cryptmod.o

1501 KB_OBJS +=      kbd.o           keytables.o

1503 #
1504 #                       ID mapping module
1505 #
1506 IDMAP_OBJS +=   idmap_mod.o     idmap_kapi.o    idmap_xdr.o     idmap_cache.o

1508 #
1509 #                       scheduling class modules
1510 #
1511 SDC_OBJS +=             sysdc.o
```



```
1513 RT_OBJS +=              rt.o
1514 RT_DPTBL_OBJS +=        rt_dptbl.o

1516 TS_OBJS +=              ts.o
1517 TS_DPTBL_OBJS +=        ts_dptbl.o

1519 IA_OBJS +=              ia.o

1521 FSS_OBJS +=             fss.o

1523 FX_OBJS +=              fx.o
1524 FX_DPTBL_OBJS +=        fx_dptbl.o

1526 #
1527 #                       Inter-Process Communication (IPC) modules
1528 #
1529 IPC_OBJS +=     ipc.o

1531 IPCMSG_OBJS +=  msg.o

1533 IPCSEM_OBJS +=  sem.o

1535 IPCSHM_OBJS +=  shm.o

1537 #
1538 #                       bignum module
1539 #
1540 COMMON_BIGNUM_OBJS += bignum_mod.o bignumimpl.o

1542 BIGNUM_OBJS += $(COMMON_BIGNUM_OBJS) $(BIGNUM_PSR_OBJS)

1544 #
1545 #                       kernel cryptographic framework
1546 #
1547 KCF_OBJS +=     kcf.o kcf_callprov.o kcf_cbufcall.o kcf_cipher.o kcf_crypto.o \
1548                 kcf_cryptoadm.o kcf_ctxops.o kcf_digest.o kcf_dual.o \
1549                 kcf_keys.o kcf_mac.o kcf_mech_tabs.o kcf_miscapi.o \
1550                 kcf_object.o kcf_policy.o kcf_prov_lib.o kcf_prov_tabs.o \
1551                 kcf_sched.o kcf_session.o kcf_sign.o kcf_spi.o kcf_verify.o \
1552                 kcf_random.o modes.o ecb.o cbc.o ctr.o ccm.o gcm.o \
1553                 fips_random.o

1555 CRYPTOADM_OBJS += cryptoadm.o

1557 CRYPTO_OBJS +=  crypto.o

1559 DPROV_OBJS +=   dprov.o

1561 DCA_OBJS +=     dca.o dca_3des.o dca_debug.o dca_dsa.o dca_kstat.o dca_rng.o \
1562                 dca_rsa.o

1564 AESPROV_OBJS += aes.o aes_impl.o aes_modes.o

1566 ARCFOURPROV_OBJS += arcfour.o arcfour_crypt.o

1568 BLOWFISHPROV_OBJS += blowfish.o blowfish_impl.o

1570 ECCPROV_OBJS += ecc.o ec.o ec2_163.o ec2_mont.o ecdecode.o ecl_mult.o \
1571                 ecp_384.o ecp_jac.o ec2_193.o ecl.o ecp_192.o ecp_521.o \
1572                 ecp_jm.o ec2_233.o ecl_curve.o ecp_224.o ecp_aff.o \
1573                 ecp_mont.o ec2_aff.o ec_naf.o ecl_gf.o ecp_256.o mp_gf2m.o \
1574                 mpi.o mplogic.o mpmontg.o mpprime.o oid.o \
1575                 secitem.o ec2_test.o ecp_test.o

1577 RSAPROV_OBJS += rsa.o rsa_impl.o pkcs1.o
```

```
1579 SWRANDPROV_OBJS += swrand.o

1581 #
1582 #                       kernel SSL
1583 #
1584 KSSL_OBJS +=    kssl.o kssloctl.o

1586 KSSL_SOCKFIL_MOD_OBJS += ksslfilter.o ksslapi.o ksslrec.o

1588 #
1589 #                       misc. modules
1590 #

1592 C2AUDIT_OBJS += adr.o audit.o audit_event.o audit_io.o \
1593                 audit_path.o audit_start.o audit_syscalls.o audit_token.o \
1594                 audit_mem.o

1596 PCIC_OBJS +=    pcic.o

1598 RPCSEC_OBJS +=  secmod.o        sec_clnt.o      sec_svc.o       sec_gen.o \
1599                 auth_des.o      auth_kern.o     auth_none.o     auth_loopb.o\
1600                 authdesprt.o    authdesubr.o    authu_prot.o \
1601                 key_call.o      key_prot.o      svc_authu.o     svcauthdes.o

1603 RPCSEC_GSS_OBJS +=      rpcsec_gssmod.o rpcsec_gss.o rpcsec_gss_misc.o \
1604                 rpcsec_gss_utils.o svc_rpcsec_gss.o

1606 CONSCONFIG_OBJS += consconfig.o

1608 CONSCONFIG_DACF_OBJS  += consconfig_dacf.o consplat.o

1610 TEM_OBJS += tem.o tem_safe.o 6x10.o 7x14.o 12x22.o

1612 KBTRANS_OBJS +=                        \
1613                 kbtrans.o               \
1614                 kbtrans_keytables.o     \
1615                 kbtrans_polled.o        \
1616                 kbtrans_streams.o       \
1617                 usb_keytables.o

1619 KGSSD_OBJS +=   gssd_clnt_stubs.o gssd_handle.o gssd_prot.o \
1620                 gss_display_name.o gss_release_name.o gss_import_name.o \
1621                 gss_release_buffer.o gss_release_oid_set.o gen_oids.o gssdmod.o

1623 KGSSD_DERIVED_OBJS = gssd_xdr.o

1625 KGSS_DUMMY_OBJS += dmech.o

1627 KSOCKET_OBJS += ksocket.o ksocket_mod.o

1629 CRYPTO= cksumtypes.o decrypt.o encrypt.o encrypt_length.o etypes.o \
1630         nfold.o verify_checksum.o prng.o block_size.o make_checksum.o\
1631         checksum_length.o hmac.o default_state.o mandatory_sumtype.o

1633 # crypto/des
1634 CRYPTO_DES= f_cbc.o f_cksum.o f_parity.o weak_key.o d3_cbc.o ef_crypto.o

1636 CRYPTO_DK= checksum.o derive.o dk_decrypt.o dk_encrypt.o

1638 CRYPTO_ARCFOUR= k5_arcfour.o

1640 # crypto/enc_provider
1641 CRYPTO_ENC= des.o des3.o arcfour_provider.o aes_provider.o

1643 # crypto/hash_provider
1644 CRYPTO_HASH= hash_kef_generic.o hash_kmd5.o hash_crc32.o hash_ksha1.o
```



```
1646 # crypto/keyhash_provider
1647 CRYPTO_KEYHASH= descbc.o k5_kmd5des.o k_hmac_md5.o

1649 # crypto/crc32
1650 CRYPTO_CRC32= crc32.o

1652 # crypto/old
1653 CRYPTO_OLD= old_decrypt.o old_encrypt.o

1655 # crypto/raw
1656 CRYPTO_RAW= raw_decrypt.o raw_encrypt.o

1658 K5_KRB= kfree.o copy_key.o \
1659         parse.o init_ctx.o \
1660         ser_adata.o ser_addr.o \
1661         ser_auth.o ser_cksum.o \
1662         ser_key.o ser_princ.o \
1663         serialize.o unparse.o \
1664         ser_actx.o

1666 K5_OS=  timeofday.o toffset.o \
1667         init_os_ctx.o c_ustime.o

1669 SEAL=   seal.o unseal.o

1671 MECH=   delete_sec_context.o \
1672         import_sec_context.o \
1673         gssapi_krb5.o \
1674         k5seal.o k5unseal.o k5sealv3.o \
1675         ser_sctx.o \
1676         sign.o \
1677         util_crypt.o  \
1678         util_validate.o  util_ordering.o  \
1679         util_seqnum.o util_set.o util_seed.o \
1680         wrap_size_limit.o verify.o



1684 MECH_GEN= util_token.o


1687 KGSS_KRB5_OBJS += krb5mech.o \
1688         $(MECH) $(SEAL) $(MECH_GEN) \
1689         $(CRYPTO) $(CRYPTO_DES) $(CRYPTO_DK) $(CRYPTO_ARCFOUR) \
1690         $(CRYPTO_ENC) $(CRYPTO_HASH) \
1691         $(CRYPTO_KEYHASH) $(CRYPTO_CRC32) \
1692         $(CRYPTO_OLD) \
1693         $(CRYPTO_RAW) $(K5_KRB) $(K5_OS)

1695 DES_OBJS +=     des_crypt.o des_impl.o des_ks.o des_soft.o

1697 DLBOOT_OBJS +=  bootparam_xdr.o nfs_dlinet.o scan.o

1699 KRTLD_OBJS +=   kobj_bootflags.o getoptstr.o \
1700                 kobj.o kobj_kdi.o kobj_lm.o kobj_subr.o

1702 MOD_OBJS +=     modctl.o modsubr.o modsysfile.o modconf.o modhash.o

1704 STRPLUMB_OBJS += strplumb.o

1706 CPR_OBJS +=     cpr_driver.o cpr_dump.o \
1707                 cpr_main.o cpr_misc.o cpr_mod.o cpr_stat.o \
1708                 cpr_uthread.o

1710 PROF_OBJS +=    prf.o
```

```
1712 SE_OBJS += se_driver.o

1714 SYSACCT_OBJS += acct.o

1716 ACCTCTL_OBJS += acctctl.o

1718 EXACCTSYS_OBJS += exacctsys.o

1720 KAIO_OBJS += aio.o

1722 PCMCIA_OBJS += pcmcia.o cs.o cis.o cis_callout.o cis_handlers.o cis_params.o

1724 BUSRA_OBJS += busra.o

1726 PCS_OBJS += pcs.o

1728 PSET_OBJS +=    pset.o

1730 OHCI_OBJS += ohci.o ohci_hub.o ohci_polled.o

1732 UHCI_OBJS += uhci.o uhciutil.o uhcitgt.o uhcihub.o uhcipolled.o

1734 EHCI_OBJS += ehci.o ehci_hub.o ehci_xfer.o ehci_intr.o ehci_util.o ehci_polled.o

1736 HUBD_OBJS += hubd.o

1738 USB_MID_OBJS += usb_mid.o

1740 USB_IA_OBJS += usb_ia.o

1742 UWBA_OBJS += uwba.o uwbai.o

1744 SCSA2USB_OBJS += scsa2usb.o usb_ms_bulkonly.o usb_ms_cbi.o

1746 HWAHC_OBJS += hwahc.o hwahc_util.o

1748 WUSB_DF_OBJS += wusb_df.o
1749 WUSB_FWMOD_OBJS += wusb_fwmod.o

1751 IPF_OBJS += ip_fil_solaris.o fil.o solaris.o ip_state.o ip_frag.o ip_nat.o \
1752             ip_proxy.o ip_auth.o ip_pool.o ip_htable.o ip_lookup.o \
1753             ip_log.o misc.o ip_compat.o ip_nat6.o drand48.o

1755 IPD_OBJS += ipd.o

1757 IBD_OBJS +=     ibd.o ibd_cm.o

1759 EIBNX_OBJS +=   enx_main.o enx_hdlrs.o enx_ibt.o enx_log.o enx_fip.o \
1760                 enx_misc.o enx_q.o enx_ctl.o

1762 EOIB_OBJS +=    eib_adm.o eib_chan.o eib_cmn.o eib_ctl.o eib_data.o \
1763                 eib_fip.o eib_ibt.o eib_log.o eib_mac.o eib_main.o \
1764                 eib_rsrc.o eib_svc.o eib_vnic.o

1766 DLPISTUB_OBJS += dlpistub.o

1768 SDP_OBJS +=     sdpddi.o

1770 TRILL_OBJS +=   trill.o

1772 CTF_OBJS += ctf_create.o ctf_decl.o ctf_error.o ctf_hash.o ctf_labels.o \
1773         ctf_lookup.o ctf_open.o ctf_types.o ctf_util.o ctf_subr.o ctf_mod.o

1775 SMBIOS_OBJS += smb_error.o smb_info.o smb_open.o smb_subr.o smb_dev.o
```



```
1777 RPCIB_OBJS += rpcib.o

1779 KMDB_OBJS += kdrv.o

1781 AFE_OBJS += afe.o

1783 BGE_OBJS += bge_main2.o bge_chip2.o bge_kstats.o bge_log.o bge_ndd.o \
1784                 bge_atomic.o bge_mii.o bge_send.o bge_recv2.o bge_mii_5906.o

1786 DMFE_OBJS += dmfe_log.o dmfe_main.o dmfe_mii.o

1788 EFE_OBJS += efe.o

1790 ELXL_OBJS += elxl.o

1792 HME_OBJS += hme.o

1794 IXGB_OBJS += ixgb.o ixgb_atomic.o ixgb_chip.o ixgb_gld.o ixgb_kstats.o \
1795                 ixgb_log.o ixgb_ndd.o ixgb_rx.o ixgb_tx.o ixgb_xmii.o

1797 NGE_OBJS += nge_main.o nge_atomic.o nge_chip.o nge_ndd.o nge_kstats.o \
1798                 nge_log.o nge_rx.o nge_tx.o nge_xmii.o

1800 PCN_OBJS += pcn.o

1802 RGE_OBJS += rge_main.o rge_chip.o rge_ndd.o rge_kstats.o rge_log.o rge_rxtx.o

1804 URTW_OBJS += urtw.o

1806 ARN_OBJS += arn_hw.o arn_eeprom.o arn_mac.o arn_calib.o arn_ani.o arn_phy.o arn_
1807                 arn_main.o arn_recv.o arn_xmit.o arn_rc.o

1809 ATH_OBJS += ath_aux.o ath_main.o ath_osdep.o ath_rate.o

1811 ATU_OBJS += atu.o

1813 IPW_OBJS += ipw2100_hw.o ipw2100.o

1815 IWI_OBJS += ipw2200_hw.o ipw2200.o

1817 IWH_OBJS += iwh.o

1819 IWK_OBJS += iwk2.o

1821 IWP_OBJS += iwp.o

1823 MWL_OBJS += mwl.o

1825 MWLFW_OBJS += mwlfw_mode.o

1827 WPI_OBJS += wpi.o

1829 RAL_OBJS += rt2560.o ral_rate.o

1831 RUM_OBJS += rum.o

1833 RWD_OBJS += rt2661.o

1835 RWN_OBJS += rt2860.o

1837 UATH_OBJS += uath.o

1839 UATHFW_OBJS += uathfw_mod.o

1841 URAL_OBJS += ural.o
```

```
1843 RTW_OBJS += rtw.o smc93cx6.o rtwphy.o rtwphyio.o

1845 ZYD_OBJS += zyd.o zyd_usb.o zyd_hw.o zyd_fw.o

1847 MXFE_OBJS += mxfe.o

1849 MPTSAS_OBJS += mptsas.o mptsas_hash.o mptsas_impl.o mptsas_init.o \
1850                 mptsas_raid.o mptsas_smhba.o

1852 MPTSAS3_OBJS += mptsas3.o mptsas3_hash.o mptsas3_impl.o mptsas3_init.o \
1853                 mptsas3_raid.o mptsas3_smhba.o

1855 #endif /* ! codereview */
1856 SFE_OBJS += sfe.o sfe_util.o

1858 BFE_OBJS += bfe.o

1860 BRIDGE_OBJS += bridge.o

1862 IDM_SHARED_OBJS += base64.o

1864 IDM_OBJS +=     $(IDM_SHARED_OBJS) \
1865                 idm.o idm_impl.o idm_text.o idm_conn_sm.o idm_so.o

1867 VR_OBJS += vr.o

1869 ATGE_OBJS += atge_main.o atge_l1e.o atge_mii.o atge_l1.o atge_l1c.o

1871 YGE_OBJS = yge.o

1873 #
1874 #       Build up defines and paths.
1875 #
1876 LINT_DEFS       += -Dunix

1878 #
1879 #       This duality can be removed when the native and target compilers
1880 #       are the same (or at least recognize the same command line syntax!)
1881 #       It is a bug in the current compilation system that the assember
1882 #       can't process the -Y I, flag.
1883 #
1884 NATIVE_INC_PATH += $(INC_PATH) $(CCYFLAG)$(UTSBASE)/common
1885 AS_INC_PATH     += $(INC_PATH) -I$(UTSBASE)/common
1886 INCLUDE_PATH    += $(INC_PATH) $(CCYFLAG)$(UTSBASE)/common

1888 PCIEB_OBJS += pcieb.o

1890 #       Chelsio N110 10G NIC driver module
1891 #
1892 CH_OBJS = ch.o glue.o pe.o sge.o

1894 CH_COM_OBJS =   ch_mac.o ch_subr.o cspi.o espi.o ixf1010.o mc3.o mc4.o mc5.o \
1895                 mv88e1xxx.o mv88x201x.o my3126.o pm3393.o tp.o ulp.o \
1896                 vsc7321.o vsc7326.o xpak.o

1898 #
1899 #       Chelsio Terminator 4 10G NIC nexus driver module
1900 #
1901 CXGBE_FW_OBJS  =        t4_fw.o t4_cfg.o
1902 CXGBE_COM_OBJS =        t4_hw.o common.o
1903 CXGBE_NEX_OBJS =        t4_nexus.o t4_sge.o t4_mac.o t4_ioctl.o shared.o \
1904                         t4_l2t.o adapter.o osdep.o

1906 #
1907 #       Chelsio Terminator 4 10G NIC driver module
1908 #
```



```
1909 CXGBE_OBJS =    cxgbe.o

1911 #
1912 #       PCI strings file
1913 #
1914 PCI_STRING_OBJS = pci_strings.o

1916 NET_DACF_OBJS += net_dacf.o

1918 #
1919 #       Xframe 10G NIC driver module
1920 #
1921 XGE_OBJS = xge.o xgell.o

1923 XGE_HAL_OBJS =  xgehal-channel.o xgehal-fifo.o xgehal-ring.o  xgehal-config.o \
1924                 xgehal-driver.o  xgehal-mm.o xgehal-stats.o  xgehal-device.o \
1925                 xge-queue.o  xgehal-mgmt.o xgehal-mgmtaux.o

1927 #
1928 #       e1000/igb common objs
1929 #
1930 #       Historically e1000g and igb had separate copies of all of the common
1931 #       code. At this time while they are now sharing the same copy of it, they
1932 #       are building it into their own modules which is due to the differences
1933 #       in the osdep and debug portions of their code.
1934 #
1935 E1000API_OBJS += e1000_80003es2lan.o e1000_82540.o e1000_82541.o e1000_82542.o \
1936                 e1000_82543.o e1000_82571.o e1000_api.o e1000_ich8lan.o \
1937                 e1000_mac.o e1000_manage.o e1000_nvm.o e1000_phy.o \
1938                 e1000_82575.o e1000_i210.o e1000_mbx.o e1000_vf.o

1940 #
1941 #       e1000g module
1942 #
1943 E1000G_OBJS +=  e1000g_debug.o e1000g_main.o e1000g_alloc.o \
1944                 e1000g_tx.o e1000g_rx.o e1000g_stat.o \
1945                 e1000g_osdep.o e1000g_workarounds.o
1946

1948 #
1949 #       Intel 82575 1G NIC driver module
1950 #
1951 IGB_OBJS =      igb_buf.o igb_debug.o igb_gld.o igb_log.o igb_main.o \
1952                 igb_rx.o igb_stat.o igb_tx.o igb_osdep.o

1954 #
1955 #       Intel Pro/100 NIC driver module
1956 #
1957 IPRB_OBJS =     iprb.o

1959 #
1960 #       Intel 10GbE PCIE NIC driver module
1961 #
1962 IXGBE_OBJS =    ixgbe_82598.o ixgbe_82599.o ixgbe_api.o         \
1963                 ixgbe_common.o ixgbe_phy.o                      \
1964                 ixgbe_buf.o ixgbe_debug.o ixgbe_gld.o           \
1965                 ixgbe_log.o ixgbe_main.o                        \
1966                 ixgbe_osdep.o ixgbe_rx.o ixgbe_stat.o           \
1967                 ixgbe_tx.o  ixgbe_x540.o ixgbe_mbx.o

1969 #
1970 #       NIU 10G/1G driver module
1971 #
1972 NXGE_OBJS =     nxge_mac.o nxge_ipp.o nxge_rxdma.o              \
1973                 nxge_txdma.o nxge_txc.o nxge_main.o             \
1974                 nxge_hw.o nxge_fzc.o nxge_virtual.o             \
```

```
1975                nxge_send.o nxge_classify.o nxge_fflp.o         \
1976                nxge_fflp_hash.o nxge_ndd.o nxge_kstats.o       \
1977                nxge_zcp.o nxge_fm.o nxge_espc.o nxge_hv.o      \
1978                nxge_hio.o nxge_hio_guest.o nxge_intr.o

1980 NXGE_NPI_OBJS = \
1981                npi.o npi_mac.o npi_ipp.o                       \
1982                npi_txdma.o npi_rxdma.o npi_txc.o               \
1983                npi_zcp.o npi_espc.o npi_fflp.o                 \
1984                npi_vir.o

1986 NXGE_HCALL_OBJS =       \
1987                nxge_hcall.o

1989 #
1990 # Virtio modules
1991 #

1993 # Virtio core
1994 VIRTIO_OBJS = virtio.o

1996 # Virtio block driver
1997 VIOBLK_OBJS = vioblk.o

1999 #
2000 #       kiconv modules
2001 #
2002 KICONV_EMEA_OBJS += kiconv_emea.o

2004 KICONV_JA_OBJS += kiconv_ja.o

2006 KICONV_KO_OBJS += kiconv_cck_common.o kiconv_ko.o

2008 KICONV_SC_OBJS += kiconv_cck_common.o kiconv_sc.o

2010 KICONV_TC_OBJS += kiconv_cck_common.o kiconv_tc.o

2012 #
2013 #       AAC module
2014 #
2015 AAC_OBJS = aac.o aac_ioctl.o

2017 #
2018 #       sdcard modules
2019 #
2020 SDA_OBJS =      sda_cmd.o sda_host.o sda_init.o sda_mem.o sda_mod.o sda_slot.o
2021 SDHOST_OBJS =   sdhost.o

2023 #
2024 #       hxge 10G driver module
2025 #
2026 HXGE_OBJS =     hxge_main.o hxge_vmac.o hxge_send.o              \
2027                hxge_txdma.o hxge_rxdma.o hxge_virtual.o         \
2028                hxge_fm.o hxge_fzc.o hxge_hw.o hxge_kstats.o     \
2029                hxge_ndd.o hxge_pfc.o                            \
2030                hpi.o hpi_vmac.o hpi_rxdma.o hpi_txdma.o         \
2031                hpi_vir.o hpi_pfc.o

2033 #
2034 #       MEGARAID_SAS module
2035 #
2036 MEGA_SAS_OBJS = megaraid_sas.o

2038 #
2039 #       MR_SAS module
2040 #
```



```
2041 MR_SAS_OBJS = ld_pd_map.o mr_sas.o mr_sas_tbolt.o mr_sas_list.o

2043 #
2044 #       CPQARY3 module
2045 #
2046 CPQARY3_OBJS =  cpqary3.o cpqary3_noe.o cpqary3_talk2ctlr.o     \
2047                cpqary3_isr.o cpqary3_transport.o cpqary3_mem.o \
2048                cpqary3_scsi.o cpqary3_util.o cpqary3_ioctl.o   \
2049                cpqary3_bd.o

2051 #
2052 #       ISCSI_INITIATOR module
2053 #
2054 ISCSI_INITIATOR_OBJS =  chap.o iscsi_io.o iscsi_thread.o        \
2055                         iscsi_ioctl.o iscsid.o iscsi.o          \
2056                         iscsi_login.o isns_client.o iscsiAuthClient.o   \
2057                         iscsi_lun.o iscsiAuthClientGlue.o       \
2058                         iscsi_net.o nvfile.o iscsi_cmd.o        \
2059                         iscsi_queue.o persistent.o iscsi_conn.o \
2060                         iscsi_sess.o radius_auth.o iscsi_crc.o  \
2061                         iscsi_stats.o radius_packet.o iscsi_doorclt.o   \
2062                         iscsi_targetparam.o utils.o kifconf.o

2064 #
2065 #       ntxn 10Gb/1Gb NIC driver module
2066 #
2067 NTXN_OBJS =     unm_nic_init.o unm_gem.o unm_nic_hw.o unm_ndd.o \
2068                         unm_nic_main.o unm_nic_isr.o unm_nic_ctx.o niu.o

2070 #
2071 #       Myricom 10Gb NIC driver module
2072 #
2073 MYRI10GE_OBJS = myri10ge.o myri10ge_lro.o

2075 #       nulldriver module
2076 #
2077 NULLDRIVER_OBJS =       nulldriver.o

2079 TPM_OBJS =      tpm.o tpm_hcall.o
```

```
**********************************************************
   73286 Thu Jun 12 17:42:15 2014
new/usr/src/uts/common/Makefile.rules
Use MSI-X interrupts, just one for now.
Pre-allocate array for request sense buffers, similar to command frames.
No more messing about with scsi_alloc_consistent_buf().
Changes to enable driver to compile.
Header paths, object lists, etc.
**********************************************************
```

```
#
# 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 2013 Garrett D'Amore <garrett@damore.org>
#

#
# 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 $@ $<
```

```
124         $(CTFCONVERT_O)

126 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/ctf/%.c
127         $(COMPILE.c) -o $@ $<
128         $(CTFCONVERT_O)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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



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

194 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/exec/shbin/%.c
195         $(COMPILE.c) -o $@ $<
196         $(CTFCONVERT_O)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
256         $(CTFCONVERT_O)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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



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

326 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/vscan/%.c
327         $(COMPILE.c) -o $@ $<
328         $(CTFCONVERT_O)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

387 $(OBJS_DIR)/%.o:                $(KMECHKRB5_BASE)/crypto/arcfour/%.c
```

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

391 $(OBJS_DIR)/%.o:                $(KMECHKRB5_BASE)/crypto/dk/%.c
392         $(COMPILE.c) $(KGSSDFLAGS) -o $@ $<
393         $(CTFCONVERT_O)

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

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

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

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

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

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

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

423 $(OBJS_DIR)/ser_sctx.o := CPPFLAGS += -DPROVIDE_KERNEL_IMPORT=1

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

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

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

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

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

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

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

453 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/avs/ns/solaris/%.c
```



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

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

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

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

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

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

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

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

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

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

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

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

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

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

509 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/inet/ipd/%.c
510         $(COMPILE.c) -o $@ $<
511         $(CTFCONVERT_O)

513 $(OBJS_DIR)/%.o:                $(COMMONBASE)/net/patricia/%.c
514         $(COMPILE.c) -o $@ $<
515         $(CTFCONVERT_O)

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

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

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

529 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/inet/dlpistub/%.c
530         $(COMPILE.c) -o $@ $<
531         $(CTFCONVERT_O)

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

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

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

545 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/1394/targets/av1394/%.c
546         $(COMPILE.c) -o $@ $<
547         $(CTFCONVERT_O)

549 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/1394/targets/dcam1394/%.c
550         $(COMPILE.c) -o $@ $<
551         $(CTFCONVERT_O)

553 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/1394/targets/scsa1394/%.c
554         $(COMPILE.c) -o $@ $<
555         $(CTFCONVERT_O)

557 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/sbp2/%.c
558         $(COMPILE.c) -o $@ $<
559         $(CTFCONVERT_O)

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

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

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

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

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

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

585 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/audio/impl/%.c
```



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

589 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/audio/ac97/%.c
590         $(COMPILE.c) -o $@ $<
591         $(CTFCONVERT_O)

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

597 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/audio/drv/audioemu10k/%.c
598         $(COMPILE.c) -o $@ $<
599         $(CTFCONVERT_O)

601 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/audio/drv/audio1575/%.c
602         $(COMPILE.c) -o $@ $<
603         $(CTFCONVERT_O)

605 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/audio/drv/audio810/%.c
606         $(COMPILE.c) -o $@ $<
607         $(CTFCONVERT_O)

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

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

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

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

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

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

633 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/audio/drv/audiop16x/%.c
634         $(COMPILE.c) -o $@ $<
635         $(CTFCONVERT_O)

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

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

645 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/audio/drv/audiovia823x/%.c
646         $(COMPILE.c) -o $@ $<
647         $(CTFCONVERT_O)

649 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/audio/drv/audiovia97/%.c
650         $(COMPILE.c) -o $@ $<
651         $(CTFCONVERT_O)
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/cpqary3/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

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

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



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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

789 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/ib/clients/of/sol_umad/%.c
790         $(COMPILE.c) -o $@ $<
791         $(CTFCONVERT_O)

793 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/ib/clients/of/sol_uverbs/%.
794         $(COMPILE.c) -o $@ $<
795         $(CTFCONVERT_O)

797 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/ib/clients/sdp/%.c
798         $(COMPILE.c) -o $@ $<
799         $(CTFCONVERT_O)

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

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

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

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

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

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

825 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/ib/adapters/tavor/%.c
826         $(COMPILE.c) -o $@ $<
827         $(CTFCONVERT_O)

829 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/ib/adapters/hermon/%.c
830         $(COMPILE.c) -o $@ $<
831         $(CTFCONVERT_O)

833 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/ib/clients/daplt/%.c
834         $(COMPILE.c) -o $@ $<
835         $(CTFCONVERT_O)

837 $(OBJS_DIR)/%.o:                $(COMMONBASE)/iscsi/%.c
838         $(COMPILE.c) -o $@ $<
839         $(CTFCONVERT_O)

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

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

849 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/iwh/%.c
```



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

853 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/iwi/%.c
854         $(COMPILE.c) -o $@ $<
855         $(CTFCONVERT_O)

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

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

865 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/kb8042/%.c
866         $(COMPILE.c) -o $@ $<
867         $(CTFCONVERT_O)

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

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

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

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

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

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

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

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

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

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

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

913 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/lvm/trans/%.c
914         $(COMPILE.c) -o $@ $<
915         $(CTFCONVERT_O)
```

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

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

$(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/mega_sas/%.c
        $(COMPILE.c) -o $@ $<
        $(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/scsi/adapters/mpt_sas3/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

#endif /* ! codereview */
$(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/pcn/%.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/spppptun/%.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
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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



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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/usb/clients/usbinput/usbwcm
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/usb/scsa2usb/%.c
```



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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/e1000api/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

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

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

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

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

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

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

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

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

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

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

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

$(OBJS_DIR)/%.o:                $(UTSBASE)/common/ipp/meters/%.c
```



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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

#
# krtld must refer to its own bzero/bcopy until the kernel is fully linked
#
```

```
$(OBJS_DIR)/bootrd.o            := CPPFLAGS += -DKOBJ_OVERRIDES
$(OBJS_DIR)/doreloc.o           := CPPFLAGS += -DKOBJ_OVERRIDES
$(OBJS_DIR)/kobj.o              := CPPFLAGS += -DKOBJ_OVERRIDES
$(OBJS_DIR)/kobj_boot.o         := CPPFLAGS += -DKOBJ_OVERRIDES
$(OBJS_DIR)/kobj_bootflags.o    := CPPFLAGS += -DKOBJ_OVERRIDES
$(OBJS_DIR)/kobj_convrelstr.o   := CPPFLAGS += -DKOBJ_OVERRIDES
$(OBJS_DIR)/kobj_isa.o          := CPPFLAGS += -DKOBJ_OVERRIDES
$(OBJS_DIR)/kobj_kdi.o          := CPPFLAGS += -DKOBJ_OVERRIDES
$(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)
```

```
1577 BZ2LINT = -erroff=%all -I$(UTSBASE)/common/bzip2

1579 $(LINTS_DIR)/bz2%.ln:           $(COMMONBASE)/bzip2/%.c
1580          @($(LHEAD) $(LINT.c) -C $(LINTS_DIR)/`basename $@ .ln` $(BZ2LINT) $< $(

1582 #
1583 #       Section 1b:     Lint `objects'
1584 #
1585 $(LINTS_DIR)/%.ln:              $(COMMONBASE)/crypto/aes/%.c
1586         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1588 $(LINTS_DIR)/%.ln:              $(COMMONBASE)/crypto/arcfour/%.c
1589         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1591 $(LINTS_DIR)/%.ln:              $(COMMONBASE)/crypto/blowfish/%.c
1592         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1594 $(LINTS_DIR)/%.ln:              $(COMMONBASE)/crypto/ecc/%.c
1595         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1597 $(LINTS_DIR)/%.ln:              $(COMMONBASE)/crypto/modes/%.c
1598         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1600 $(LINTS_DIR)/%.ln:              $(COMMONBASE)/crypto/padding/%.c
1601         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1603 $(LINTS_DIR)/%.ln:              $(COMMONBASE)/crypto/rng/%.c
1604         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1606 $(LINTS_DIR)/%.ln:              $(COMMONBASE)/crypto/rsa/%.c
1607         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1609 $(LINTS_DIR)/%.ln:              $(COMMONBASE)/bignum/%.c
1610         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1612 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/bignum/%.c
1613         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1615 $(LINTS_DIR)/%.ln:              $(COMMONBASE)/mpi/%.c
1616         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1618 $(LINTS_DIR)/%.ln:              $(COMMONBASE)/acl/%.c
1619         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1621 $(LINTS_DIR)/%.ln:              $(COMMONBASE)/avl/%.c
1622         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1624 $(LINTS_DIR)/%.ln:              $(COMMONBASE)/ucode/%.c
1625         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1627 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/brand/sn1/%.c
1628         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1630 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/brand/solaris10/%.c
1631         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1633 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/c2/%.c
1634         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1636 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/conf/%.c
1637         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1639 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/contract/%.c
1640         @($(LHEAD) $(LINT.c) $< $(LTAIL))
```



```
1642 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/cpr/%.c
1643         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1645 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/ctf/%.c
1646         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1648 $(LINTS_DIR)/%.ln:              $(COMMONBASE)/ctf/%.c
1649         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1651 $(LINTS_DIR)/%.ln:              $(COMMONBASE)/pci/%.c
1652         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1654 $(LINTS_DIR)/%.ln:              $(COMMONBASE)/devid/%.c
1655         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1657 $(LINTS_DIR)/%.ln:              $(COMMONBASE)/crypto/des/%.c
1658         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1660 $(LINTS_DIR)/%.ln:              $(COMMONBASE)/smbios/%.c
1661         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1663 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/avs/ncall/%.c
1664         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1666 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/avs/ns/dsw/%.c
1667         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1669 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/avs/ns/nsctl/%.c
1670         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1672 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/avs/ns/rdc/%.c
1673         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1675 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/avs/ns/sdbc/%.c
1676         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1678 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/avs/ns/solaris/%.c
1679         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1681 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/avs/ns/sv/%.c
1682         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1684 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/avs/ns/unistat/%.c
1685         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1687 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/des/%.c
1688         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1690 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/crypto/api/%.c
1691         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1693 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/crypto/core/%.c
1694         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1696 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/crypto/io/%.c
1697         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1699 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/crypto/spi/%.c
1700         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1702 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/disp/%.c
1703         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1705 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/dtrace/%.c
1706         @($(LHEAD) $(LINT.c) $< $(LTAIL))
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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



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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/fs/zut/%.c
        @($(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:              $(UTSBASE)/common/inet/ipd/%.c
        @($(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))

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

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

```
1972 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/atu/%.c
1973         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1975 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/audio/impl/%.c
1976         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1978 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/audio/ac97/%.c
1979         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1981 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/audio/drv/audio1575/%.c
1982         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1984 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/audio/drv/audio810/%.c
1985         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1987 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/audio/drv/audiocmi/%.c
1988         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1990 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/audio/drv/audiocmihd/%.c
1991         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1993 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/audio/drv/audioens/%.c
1994         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1996 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/audio/drv/audioemu10k/%.c
1997         @($(LHEAD) $(LINT.c) $< $(LTAIL))

1999 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/audio/drv/audiohd/%.c
2000         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2002 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/audio/drv/audioixp/%.c
2003         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2005 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/audio/drv/audiols/%.c
2006         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2008 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/audio/drv/audiopci/%.c
2009         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2011 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/audio/drv/audiop16x/%.c
2012         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2014 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/audio/drv/audiosolo/%.c
2015         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2017 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/audio/drv/audiots/%.c
2018         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2020 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/audio/drv/audiovia823x/%.c
2021         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2023 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/audio/drv/audiovia97/%.c
2024         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2026 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/bfe/%.c
2027         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2029 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/bpf/%.c
2030         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2032 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/bge/%.c
2033         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2035 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/blkdev/%.c
2036         @($(LHEAD) $(LINT.c) $< $(LTAIL))
```



```
2038 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/cardbus/%.c
2039         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2041 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/comstar/lu/stmf_sbd/%.c
2042         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2044 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/comstar/port/fct/%.c
2045         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2047 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/comstar/port/qlt/%.c
2048         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2050 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/comstar/port/srpt/%.c
2051         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2053 $(LINTS_DIR)/%.ln:              $(COMMONBASE)/iscsit/%.c
2054         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2056 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/comstar/port/fcoet/%.c
2057         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2059 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/comstar/port/iscsit/%.c
2060         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2062 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/comstar/port/pppt/%.c
2063         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2065 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/comstar/stmf/%.c
2066         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2068 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/cpqary3/%.c
2069         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2071 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/dld/%.c
2072         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2074 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/dls/%.c
2075         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2077 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/dmfe/%.c
2078         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2080 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/drm/%.c
2081         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2083 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/efe/%.c
2084         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2086 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/elxl/%.c
2087         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2089 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/fcoe/%.c
2090         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2092 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/hme/%.c
2093         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2095 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/pciex/%.c
2096         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2098 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/hotplug/hpcsvc/%.c
2099         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2101 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/pciex/hotplug/%.c
2102         @($(LHEAD) $(LINT.c) $< $(LTAIL))
```

```
2104 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/hotplug/pcihp/%.c
2105         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2107 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/ib/clients/rds/%.c
2108         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2110 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/ib/clients/rdsv3/%.c
2111         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2113 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/ib/clients/iser/%.c
2114         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2116 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/ib/clients/ibd/%.c
2117         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2119 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/ib/clients/eoib/%.c
2120         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2122 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/ib/clients/of/sol_ofs/%.c
2123         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2125 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/ib/clients/of/sol_ucma/%.c
2126         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2128 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/ib/clients/of/sol_umad/%.c
2129         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2131 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/ib/clients/of/sol_uverbs/%.
2132         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2134 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/ib/clients/sdp/%.c
2135         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2137 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/ib/mgt/ibcm/%.c
2138         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2140 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/ib/mgt/ibdm/%.c
2141         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2143 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/ib/mgt/ibdma/%.c
2144         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2146 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/ib/mgt/ibmf/%.c
2147         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2149 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/ib/ibnex/%.c
2150         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2152 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/ib/ibtl/%.c
2153         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2155 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/ib/adapters/tavor/%.c
2156         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2158 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/ib/adapters/hermon/%.c
2159         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2161 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/ib/clients/daplt/%.c
2162         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2164 $(LINTS_DIR)/%.ln:              $(COMMONBASE)/iscsi/%.c
2165         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2167 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/idm/%.c
2168         @($(LHEAD) $(LINT.c) $< $(LTAIL))
```



```
2170 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/ipw/%.c
2171         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2173 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/iwh/%.c
2174         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2176 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/iwi/%.c
2177         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2179 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/iwk/%.c
2180         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2182 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/iwp/%.c
2183         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2185 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/kb8042/%.c
2186         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2188 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/kbtrans/%.c
2189         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2191 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/ksocket/%.c
2192         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2194 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/aggr/%.c
2195         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2197 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/lp/%.c
2198         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2200 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/lvm/hotspares/%.c
2201         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2203 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/lvm/md/%.c
2204         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2206 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/lvm/mirror/%.c
2207         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2209 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/lvm/raid/%.c
2210         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2212 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/lvm/softpart/%.c
2213         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2215 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/lvm/stripe/%.c
2216         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2218 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/lvm/notify/%.c
2219         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2221 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/lvm/trans/%.c
2222         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2224 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/mac/%.c
2225         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2227 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/mac/plugins/%.c
2228         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2230 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/mega_sas/%.c
2231         @($(LHEAD) $(LINT.c) $< $(LTAIL))

2233 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/mii/%.c
2234         @($(LHEAD) $(LINT.c) $< $(LTAIL))
```

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

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

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/scsi/adapters/mpt_sas3/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

#endif /* ! codereview */
$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/mxfe/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

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

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

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/net80211/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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



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

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

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

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

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/sata/adapters/si3124/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

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

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

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/scsi/adapters/blk2scsa/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

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

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

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

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

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

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/fibre-channel/fca/qlc/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/fibre-channel/fca/qlge/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/fibre-channel/fca/emlxs/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/fibre-channel/fca/oce/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/fibre-channel/fca/fcoei/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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



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

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

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

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

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

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

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/usb/clients/hwa1480_fw/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

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

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

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

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

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

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

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/usb/scsa2usb/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

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

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

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

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/usb/usba10/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/uwb/uwba/%.c
        @($(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:              $(UTSBASE)/common/io/e1000api/%.c
        @($(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:              $(UTSBASE)/common/ipp/%.c
        @($(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:              $(UTSBASE)/common/klm/%.c
        @($(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))
```

```
**********************************************************
  468547 Thu Jun 12 17:42:15 2014
new/usr/src/uts/common/io/scsi/adapters/mpt_sas3/mptsas3.c
Code reconciliation with other base.
hg changesets 607a5b46a793..b706c96317c3
Fix ncpus for early boot config
Purge the ack to the interrupt before exiting mptsas_intr()
Changes from code review
Optimise slot alocation through rotor and the use
of per target mutex in start path.
Update tx waitq's code.
Create 2 threads, divide the workflow and deliver
to the hardware from the threads.
Optimise mutex's and code paths.
Split out offline target code.
Test timeouts code
Add support for more than 8 MSI-X interrupts.
Tidy up interrupt assignement and card ID messages.
Enable Fast Path for capable devices.
Merge fixes for Illumos issue 4819, fix mpt_sas command timeout handling.
Tweeks debug flags.
Default to process done commands all in threads if only 1 interrupt.
Lint and cstyle fixes.
Fix problem with running against 64bit msgaddr attributes for DMA.
Default is now to run like this.
Don't take tx_waiq_mutex if draining isn't set.
Fixes for Illumos issue 4682.
Fix hang bug to do with tx_wq.
Re-arrange mptsas_poll() to disable interrupts before issuing the
command.
Improve the tx_waitq code path.
Major rework of mutexes.
During normal operation do not grab m_mutex during interrupt.
Use reply post queues instead.
Make a few variable non static so you can change in /etc/system.
Fixes to some address arithmetic using 32bit values.
Distribute command done processing around the threads.
Improved auto-request sense memory usage.
Fix for Nexenta commit 36c74113a21
OS-91 mptsas does inquiry without setting pkt_time
Add comment about testing.
Test firmware version of 2008 controllers for MSI-X Compatibility.
Re-arrange mptsas_intr() to reduce number of spurious interrupts.
Fix bug in mptsas_free_post_queue().
Change mptsas_doneq_mv() to not loop.
Should not need m_in_callback flag. It prevents concurrent
command completion processing.
Added code to support using MSI-X interrupts across multiple
reply queues. Not tested with anything other than 3008 yet.
Change output "mptsas%d" -> "mptsas3%d".
Add SAS3 specific messages (12.0Gb).
Allow over-ride for interrupt type.
Restrict pre MPI2.5 to MSI interrupts.
Allow watchdog timout to work for mptsas_smhba_setup() in attach().
Merge fixes for "4403 mpt_sas panic when pulling a drive", commit f7d0d869a9ae78
Use MSI-X interrupts, just one for now.
Pre-allocate array for request sense buffers, similar to command frames.
No more messing about with scsi_alloc_consistent_buf().
Add rolling buffer for *all* debug messages.
Improve mdb module and seperate out into mpt_sas3.
Initial modifications using the code changes present between
the LSI source code for FreeBSD drivers. Specifically the changes
between from mpslsi-source-17.00.00.00 -> mpslsi-source-03.00.00.00.
This mainly involves using a different scatter/gather element in
frame setup.
Change some obvious references sas -> sas3.
Changes to enable driver to compile.
Header paths, object lists, etc.
**********************************************************
```



```
 1 /*
 2  * CDDL HEADER START
 3  *
 4  * The contents of this file are subject to the terms of the
 5  * Common Development and Distribution License (the "License").
 6  * You may not use this file except in compliance with the License.
 7  *
 8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 9  * or http://www.opensolaris.org/os/licensing.
10  * See the License for the specific language governing permissions
11  * and limitations under the License.
12  *
13  * When distributing Covered Code, include this CDDL HEADER in each
14  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15  * If applicable, add the following below this CDDL HEADER, with the
16  * fields enclosed by brackets "[]" replaced with your own identifying
17  * information: Portions Copyright [yyyy] [name of copyright owner]
18  *
19  * CDDL HEADER END
20  */

22 /*
23  * Copyright (c) 2009, 2010, Oracle and/or its affiliates. All rights reserved.
24  * Copyright 2014 Nexenta Systems, Inc. All rights reserved.
24  * Copyright 2012 Nexenta Systems, Inc. All rights reserved.
25  * Copyright (c) 2014, Joyent, Inc. All rights reserved.
26  * Copyright (c) 2014, Tegile Systems Inc. All rights reserved.
27  * Copyright 2014 OmniTI Computer Consulting, Inc. All rights reserved.
28 #endif /* ! codereview */
29  */

31 /*
32  * Copyright (c) 2000 to 2010, LSI Corporation.
33  * All rights reserved.
34  *
35  * Redistribution and use in source and binary forms of all code within
36  * this file that is exclusively owned by LSI, with or without
37  * modification, is permitted provided that, in addition to the CDDL 1.0
38  * License requirements, the following conditions are met:
39  *
40  *    Neither the name of the author nor the names of its contributors may be
41  *    used to endorse or promote products derived from this software without
42  *    specific prior written permission.
43  *
44  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
45  * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
46  * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
47  * FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
48  * COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
49  * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
50  * BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
51  * OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
52  * AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
53  * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
54  * OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
55  * DAMAGE.
56  */

58 /*
59  * mptsas3 - This is a driver based on LSI Logic's MPT2.0/2.5 interface.
26  * mptsas - This is a driver based on LSI Logic's MPT2.0 interface.
60  *
61  */

63 #if defined(lint) || defined(DEBUG)
64 #define MPTSAS_DEBUG
```

```
 65 #endif

 67 /*
 68  * standard header files.
 69  */
 70 #include <sys/note.h>
 71 #include <sys/scsi/scsi.h>
 72 #include <sys/pci.h>
 73 #include <sys/file.h>
 74 #include <sys/policy.h>
 75 #include <sys/model.h>
 76 #include <sys/sysevent.h>
 77 #include <sys/sysevent/eventdefs.h>
 78 #include <sys/sysevent/dr.h>
 79 #include <sys/sata/sata_defs.h>
 80 #include <sys/scsi/generic/sas.h>
 81 #include <sys/scsi/impl/scsi_sas.h>

 83 #pragma pack(1)
 84 #include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_type.h>
 85 #include <sys/scsi/adapters/mpt_sas3/mpi/mpi2.h>
 86 #include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_cnfg.h>
 87 #include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_init.h>
 88 #include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_ioc.h>
 89 #include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_sas.h>
 90 #include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_tool.h>
 91 #include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_raid.h>
 51 #include <sys/scsi/adapters/mpt_sas/mpi/mpi2_type.h>
 52 #include <sys/scsi/adapters/mpt_sas/mpi/mpi2.h>
 53 #include <sys/scsi/adapters/mpt_sas/mpi/mpi2_cnfg.h>
 54 #include <sys/scsi/adapters/mpt_sas/mpi/mpi2_init.h>
 55 #include <sys/scsi/adapters/mpt_sas/mpi/mpi2_ioc.h>
 56 #include <sys/scsi/adapters/mpt_sas/mpi/mpi2_sas.h>
 57 #include <sys/scsi/adapters/mpt_sas/mpi/mpi2_tool.h>
 58 #include <sys/scsi/adapters/mpt_sas/mpi/mpi2_raid.h>
 92 #pragma pack()

 94 /*
 95  * private header files.
 96  *
 97  */
 98 #include <sys/scsi/impl/scsi_reset_notify.h>
 99 #include <sys/scsi/adapters/mpt_sas3/mptsas3_var.h>
100 #include <sys/scsi/adapters/mpt_sas3/mptsas3_ioctl.h>
101 #include <sys/scsi/adapters/mpt_sas3/mptsas3_smhba.h>
102 #include <sys/scsi/adapters/mpt_sas3/mptsas3_hash.h>
 66 #include <sys/scsi/adapters/mpt_sas/mptsas_var.h>
 67 #include <sys/scsi/adapters/mpt_sas/mptsas_ioctl.h>
 68 #include <sys/scsi/adapters/mpt_sas/mptsas_smhba.h>
 69 #include <sys/scsi/adapters/mpt_sas/mptsas_hash.h>
103 #include <sys/raidioctl.h>

105 #include <sys/fs/dv_node.h>     /* devfs_clean */

107 /*
108  * FMA header files
109  */
110 #include <sys/ddifm.h>
111 #include <sys/fm/protocol.h>
112 #include <sys/fm/util.h>
113 #include <sys/fm/io/ddi.h>

115 /*
116  * autoconfiguration data and routines.
117  */
118 static int mptsas_attach(dev_info_t *dip, ddi_attach_cmd_t cmd);
```



```
119 static int mptsas_detach(dev_info_t *devi, ddi_detach_cmd_t cmd);
120 static int mptsas_power(dev_info_t *dip, int component, int level);

122 /*
123  * cb_ops function
124  */
125 static int mptsas_ioctl(dev_t dev, int cmd, intptr_t data, int mode,
126         cred_t *credp, int *rval);
127 #ifdef __sparc
128 static int mptsas_reset(dev_info_t *devi, ddi_reset_cmd_t cmd);
129 #else  /* __sparc */
130 static int mptsas_quiesce(dev_info_t *devi);
131 #endif  /* __sparc */

133 /*
134  * Resource initilaization for hardware
135  */
136 static void mptsas_setup_cmd_reg(mptsas_t *mpt);
137 static void mptsas_disable_bus_master(mptsas_t *mpt);
138 static void mptsas_hba_fini(mptsas_t *mpt);
139 static void mptsas_cfg_fini(mptsas_t *mptsas_blkp);
140 static int mptsas_hba_setup(mptsas_t *mpt);
141 static void mptsas_hba_teardown(mptsas_t *mpt);
142 static int mptsas_config_space_init(mptsas_t *mpt);
143 static void mptsas_config_space_fini(mptsas_t *mpt);
144 static void mptsas_iport_register(mptsas_t *mpt);
145 static int mptsas_smp_setup(mptsas_t *mpt);
146 static void mptsas_smp_teardown(mptsas_t *mpt);
147 static int mptsas_cache_create(mptsas_t *mpt);
148 static void mptsas_cache_destroy(mptsas_t *mpt);
149 static int mptsas_alloc_request_frames(mptsas_t *mpt);
150 static int mptsas_alloc_sense_bufs(mptsas_t *mpt);
151 #endif /* ! codereview */
152 static int mptsas_alloc_reply_frames(mptsas_t *mpt);
153 static int mptsas_alloc_free_queue(mptsas_t *mpt);
154 static int mptsas_alloc_post_queue(mptsas_t *mpt);
155 static void mptsas_free_post_queue(mptsas_t *mpt);
156 #endif /* ! codereview */
157 static void mptsas_alloc_reply_args(mptsas_t *mpt);
158 static int mptsas_alloc_extra_sgl_frame(mptsas_t *mpt, mptsas_cmd_t *cmd);
159 static void mptsas_free_extra_sgl_frame(mptsas_t *mpt, mptsas_cmd_t *cmd);
160 static int mptsas_init_chip(mptsas_t *mpt, int first_time);

162 /*
163  * SCSA function prototypes
164  */
165 static int mptsas_scsi_start(struct scsi_address *ap, struct scsi_pkt *pkt);
166 static int mptsas_scsi_reset(struct scsi_address *ap, int level);
167 static int mptsas_scsi_abort(struct scsi_address *ap, struct scsi_pkt *pkt);
168 static int mptsas_scsi_getcap(struct scsi_address *ap, char *cap, int tgtonly);
169 static int mptsas_scsi_setcap(struct scsi_address *ap, char *cap, int value,
170     int tgtonly);
171 static void mptsas_scsi_dmafree(struct scsi_address *ap, struct scsi_pkt *pkt);
172 static struct scsi_pkt *mptsas_scsi_init_pkt(struct scsi_address *ap,
173     struct scsi_pkt *pkt, struct buf *bp, int cmdlen, int statuslen,
174         int tgtlen, int flags, int (*callback)(), caddr_t arg);
175 static void mptsas_scsi_sync_pkt(struct scsi_address *ap, struct scsi_pkt *pkt);
176 static void mptsas_scsi_destroy_pkt(struct scsi_address *ap,
177     struct scsi_pkt *pkt);
178 static int mptsas_scsi_tgt_init(dev_info_t *hba_dip, dev_info_t *tgt_dip,
179     scsi_hba_tran_t *hba_tran, struct scsi_device *sd);
180 static void mptsas_scsi_tgt_free(dev_info_t *hba_dip, dev_info_t *tgt_dip,
181     scsi_hba_tran_t *hba_tran, struct scsi_device *sd);
182 static int mptsas_scsi_reset_notify(struct scsi_address *ap, int flag,
183     void (*callback)(caddr_t), caddr_t arg);
184 static int mptsas_get_name(struct scsi_device *sd, char *name, int len);
```

```
185 static int mptsas_get_bus_addr(struct scsi_device *sd, char *name, int len);
186 static int mptsas_scsi_quiesce(dev_info_t *dip);
187 static int mptsas_scsi_unquiesce(dev_info_t *dip);
188 static int mptsas_bus_config(dev_info_t *pdip, uint_t flags,
189     ddi_bus_config_op_t op, void *arg, dev_info_t **childp);

191 /*
192  * SMP functions
193  */
194 static int mptsas_smp_start(struct smp_pkt *smp_pkt);

196 /*
197  * internal function prototypes.
198  */
199 static void mptsas_list_add(mptsas_t *mpt);
200 static void mptsas_list_del(mptsas_t *mpt);

202 static int mptsas_quiesce_bus(mptsas_t *mpt);
203 static int mptsas_unquiesce_bus(mptsas_t *mpt);

205 static int mptsas_alloc_handshake_msg(mptsas_t *mpt, size_t alloc_size);
206 static void mptsas_free_handshake_msg(mptsas_t *mpt);

208 static void mptsas_ncmds_checkdrain(void *arg);

210 static int mptsas_prepare_pkt(mptsas_cmd_t *cmd);
211 static void mptsas_retry_pkt(mptsas_t *mpt, mptsas_cmd_t *sp);
212 static int mptsas_save_cmd_to_slot(mptsas_t *mpt, mptsas_cmd_t *cmd);
213 static int mptsas_accept_pkt(mptsas_t *mpt, mptsas_cmd_t *sp,
214     int *tran_rval);
215 static void mptsas_accept_tx_waitqs(mptsas_t *mpt);
216 static void mptsas_unblock_tx_waitqs(mptsas_t *mpt);
217 static void mptsas_drain_tx_waitq(mptsas_t *mpt, mptsas_tx_waitqueue_t *txwq);
218 static int mptsas_check_targ_intxtion(mptsas_target_t *ptgt, int cmd_pkt_flags);
117 static int mptsas_accept_pkt(mptsas_t *mpt, mptsas_cmd_t *sp);
118 static int mptsas_accept_txwq_and_pkt(mptsas_t *mpt, mptsas_cmd_t *sp);
119 static void mptsas_accept_tx_waitq(mptsas_t *mpt);

220 static int mptsas_do_detach(dev_info_t *dev);
221 static int mptsas_do_scsi_reset(mptsas_t *mpt, uint16_t devhdl);
222 static int mptsas_do_scsi_abort(mptsas_t *mpt, int target, int lun,
223     struct scsi_pkt *pkt);
224 static int mptsas_scsi_capchk(char *cap, int tgtonly, int *cidxp);

226 static void mptsas_handle_qfull(mptsas_t *mpt, mptsas_cmd_t *cmd);
227 static void mptsas_handle_event(void *args);
228 static int mptsas_handle_event_sync(void *args);
229 static void mptsas_handle_dr(void *args);
230 static void mptsas_handle_topo_change(mptsas_topo_change_list_t *topo_node,
231     dev_info_t *pdip);

233 static void mptsas_restart_cmd(void *);

235 static void mptsas_flush_hba(mptsas_t *mpt);
236 static void mptsas_flush_target(mptsas_t *mpt, ushort_t target, int lun,
237         uint8_t tasktype);
238 static void mptsas_set_pkt_reason(mptsas_t *mpt, mptsas_cmd_t *cmd,
239     uchar_t reason, uint_t stat);

241 static uint_t mptsas_intr(caddr_t arg1, caddr_t arg2);
242 static void mptsas_process_intr(mptsas_t *mpt, mptsas_reply_pqueue_t *rpqp,
143 static void mptsas_process_intr(mptsas_t *mpt,
243     pMpi2ReplyDescriptorsUnion_t reply_desc_union);
244 static void mptsas_handle_scsi_io_success(mptsas_t *mpt,
245     mptsas_reply_pqueue_t *rpqp, pMpi2ReplyDescriptorsUnion_t reply_desc);
146     pMpi2ReplyDescriptorsUnion_t reply_desc);
```



```
246 static void mptsas_handle_address_reply(mptsas_t *mpt,
247     pMpi2ReplyDescriptorsUnion_t reply_desc);
248 static int mptsas_wait_intr(mptsas_t *mpt, int polltime);
249 static void mptsas_sge_setup(mptsas_t *mpt, mptsas_cmd_t *cmd,
250     uint32_t *control, pMpi2SCSIIORequest_t frame, ddi_acc_handle_t acc_hdl);

252 static void mptsas_watch(void *arg);
253 static void mptsas_watchsubr(mptsas_t *mpt);
254 static void mptsas_cmd_timeout(mptsas_t *mpt, mptsas_target_t *ptgt);
155 static void mptsas_cmd_timeout(mptsas_t *mpt, uint16_t devhdl);

256 static void mptsas_start_passthru(mptsas_t *mpt, mptsas_cmd_t *cmd);
257 static int mptsas_do_passthru(mptsas_t *mpt, uint8_t *request, uint8_t *reply,
258     uint8_t *data, uint32_t request_size, uint32_t reply_size,
259     uint32_t data_size, uint8_t direction, uint8_t *dataout,
160     uint32_t data_size, uint32_t direction, uint8_t *dataout,
260     uint32_t dataout_size, short timeout, int mode);
261 static int mptsas_free_devhdl(mptsas_t *mpt, uint16_t devhdl);

263 static uint8_t mptsas_get_fw_diag_buffer_number(mptsas_t *mpt,
264     uint32_t unique_id);
265 static void mptsas_start_diag(mptsas_t *mpt, mptsas_cmd_t *cmd);
266 static int mptsas_post_fw_diag_buffer(mptsas_t *mpt,
267     mptsas_fw_diagnostic_buffer_t *pBuffer, uint32_t *return_code);
268 static int mptsas_release_fw_diag_buffer(mptsas_t *mpt,
269     mptsas_fw_diagnostic_buffer_t *pBuffer, uint32_t *return_code,
270     uint32_t diag_type);
271 static int mptsas_diag_register(mptsas_t *mpt,
272     mptsas_fw_diag_register_t *diag_register, uint32_t *return_code);
273 static int mptsas_diag_unregister(mptsas_t *mpt,
274     mptsas_fw_diag_unregister_t *diag_unregister, uint32_t *return_code);
275 static int mptsas_diag_query(mptsas_t *mpt, mptsas_fw_diag_query_t *diag_query,
276     uint32_t *return_code);
277 static int mptsas_diag_read_buffer(mptsas_t *mpt,
278     mptsas_diag_read_buffer_t *diag_read_buffer, uint8_t *ioctl_buf,
279     uint32_t *return_code, int ioctl_mode);
280 static int mptsas_diag_release(mptsas_t *mpt,
281     mptsas_fw_diag_release_t *diag_release, uint32_t *return_code);
282 static int mptsas_do_diag_action(mptsas_t *mpt, uint32_t action,
283     uint8_t *diag_action, uint32_t length, uint32_t *return_code,
284     int ioctl_mode);
285 static int mptsas_diag_action(mptsas_t *mpt, mptsas_diag_action_t *data,
286     int mode);

288 static int mptsas_pkt_alloc_extern(mptsas_t *mpt, mptsas_cmd_t *cmd,
289     int cmdlen, int tgtlen, int statuslen, int kf);
290 static void mptsas_pkt_destroy_extern(mptsas_t *mpt, mptsas_cmd_t *cmd);

292 static int mptsas_kmem_cache_constructor(void *buf, void *cdrarg, int kmflags);
293 static void mptsas_kmem_cache_destructor(void *buf, void *cdrarg);

295 static int mptsas_cache_frames_constructor(void *buf, void *cdrarg,
296     int kmflags);
297 static void mptsas_cache_frames_destructor(void *buf, void *cdrarg);

299 static void mptsas_check_scsi_io_error(mptsas_t *mpt, pMpi2SCSIIOReply_t reply,
300     mptsas_cmd_t *cmd);
301 static void mptsas_check_task_mgt(mptsas_t *mpt,
302     pMpi2SCSIManagementReply_t reply, mptsas_cmd_t *cmd);
303 static int mptsas_send_scsi_cmd(mptsas_t *mpt, struct scsi_address *ap,
304     mptsas_target_t *ptgt, uchar_t *cdb, int cdblen, struct buf *data_bp,
305     int *resid);

307 static int mptsas_alloc_active_slots(mptsas_t *mpt, int flag);
308 static void mptsas_free_active_slots(mptsas_t *mpt);
309 static int mptsas_start_cmd(mptsas_t *mpt, mptsas_cmd_t *cmd);
```

```
311 static void mptsas_restart_hba(mptsas_t *mpt);
312 static void mptsas_restart_waitq(mptsas_t *mpt);

314 static void mptsas_deliver_doneq_thread(mptsas_t *mpt,
315     mptsas_done_list_t *dlist);
215 static void mptsas_deliver_doneq_thread(mptsas_t *mpt);
316 static void mptsas_doneq_add(mptsas_t *mpt, mptsas_cmd_t *cmd);
317 static void mptsas_rpdoneq_add(mptsas_t *mpt, mptsas_reply_pqueue_t *rpqp,
318     mptsas_cmd_t *cmd);
319 static void mptsas_doneq_mv(mptsas_done_list_t *from,
320     mptsas_doneq_thread_list_t *item);
217 static void mptsas_doneq_mv(mptsas_t *mpt, uint64_t t);

322 static mptsas_cmd_t *mptsas_doneq_thread_rm(mptsas_t *mpt, uint64_t t);
323 static void mptsas_doneq_empty(mptsas_t *mpt);
324 static void mptsas_rpdoneq_empty(mptsas_reply_pqueue_t *rpqp);
325 static void mptsas_doneq_thread(mptsas_thread_arg_t *arg);
326 static void mptsas_tx_waitq_thread(mptsas_thread_arg_t *arg);
221 static void mptsas_doneq_thread(mptsas_doneq_thread_arg_t *arg);

328 static mptsas_cmd_t *mptsas_waitq_rm(mptsas_t *mpt);
329 static void mptsas_waitq_delete(mptsas_t *mpt, mptsas_cmd_t *cmd);
225 static mptsas_cmd_t *mptsas_tx_waitq_rm(mptsas_t *mpt);
226 static void mptsas_tx_waitq_delete(mptsas_t *mpt, mptsas_cmd_t *cmd);


331 static void mptsas_start_watch_reset_delay();
332 static void mptsas_setup_bus_reset_delay(mptsas_t *mpt);
333 static void mptsas_watch_reset_delay(void *arg);
334 static int mptsas_watch_reset_delay_subr(mptsas_t *mpt);
335 static void mptsas_set_throttle(struct mptsas *mpt, mptsas_target_t *ptgt,
336     int what);
337 static void mptsas_set_throttle_mtx(struct mptsas *mpt, mptsas_target_t *ptgt,
338     int what);
339 static void mptsas_remove_cmd_nomtx(mptsas_t *mpt, mptsas_cmd_t *cmd);
340 #endif /* ! codereview */

342 /*
343  * helper functions
344  */
345 static void mptsas_dump_cmd(mptsas_t *mpt, mptsas_cmd_t *cmd);

347 static dev_info_t *mptsas_find_child(dev_info_t *pdip, char *name);
348 static dev_info_t *mptsas_find_child_phy(dev_info_t *pdip, uint8_t phy);
349 static dev_info_t *mptsas_find_child_addr(dev_info_t *pdip, uint64_t sasaddr,
350     int lun);
351 static mdi_pathinfo_t *mptsas_find_path_addr(dev_info_t *pdip, uint64_t sasaddr,
352     int lun);
353 static mdi_pathinfo_t *mptsas_find_path_phy(dev_info_t *pdip, uint8_t phy);
354 static dev_info_t *mptsas_find_smp_child(dev_info_t *pdip, char *str_wwn);

356 static int mptsas_parse_address(char *name, uint64_t *wwid, uint8_t *phy,
357     int *lun);
358 static int mptsas_parse_smp_name(char *name, uint64_t *wwn);

360 static mptsas_target_t *mptsas_phy_to_tgt(mptsas_t *mpt,
361     mptsas_phymask_t phymask, uint8_t phy);
362 static mptsas_target_t *mptsas_wwid_to_ptgt(mptsas_t *mpt,
363     mptsas_phymask_t phymask, uint64_t wwid);
364 static mptsas_smp_t *mptsas_wwid_to_psmp(mptsas_t *mpt,
365     mptsas_phymask_t phymask, uint64_t wwid);

367 static int mptsas_inquiry(mptsas_t *mpt, mptsas_target_t *ptgt, int lun,
368     uchar_t page, unsigned char *buf, int len, int *rlen, uchar_t evpd);
```



```
370 static int mptsas_get_target_device_info(mptsas_t *mpt, uint32_t page_address,
371     uint16_t *handle, mptsas_target_t **pptgt);
372 static void mptsas_update_phymask(mptsas_t *mpt);

374 static int mptsas_send_sep(mptsas_t *mpt, mptsas_target_t *ptgt,
375     uint32_t *status, uint8_t cmd);
376 static dev_info_t *mptsas_get_dip_from_dev(dev_t dev,
377     mptsas_phymask_t *phymask);
378 static mptsas_target_t *mptsas_addr_to_ptgt(mptsas_t *mpt, char *addr,
379     mptsas_phymask_t phymask);
380 static int mptsas_flush_led_status(mptsas_t *mpt, mptsas_target_t *ptgt);


383 /*
384  * Enumeration / DR functions
385  */
386 static void mptsas_config_all(dev_info_t *pdip);
387 static int mptsas_config_one_addr(dev_info_t *pdip, uint64_t sasaddr, int lun,
388     dev_info_t **lundip);
389 static int mptsas_config_one_phy(dev_info_t *pdip, uint8_t phy, int lun,
390     dev_info_t **lundip);

392 static int mptsas_config_target(dev_info_t *pdip, mptsas_target_t *ptgt);
393 static int mptsas_offline_targetdev(dev_info_t *pdip, char *name);
394 static void mptsas_offline_target(mptsas_t *mpt, mptsas_target_t *ptgt,
395     uint8_t topo_flags, dev_info_t *parent);
233 static int mptsas_offline_target(dev_info_t *pdip, char *name);

397 static int mptsas_config_raid(dev_info_t *pdip, uint16_t target,
398     dev_info_t **dip);

400 static int mptsas_config_luns(dev_info_t *pdip, mptsas_target_t *ptgt);
401 static int mptsas_probe_lun(dev_info_t *pdip, int lun,
402     dev_info_t **dip, mptsas_target_t *ptgt);

404 static int mptsas_create_lun(dev_info_t *pdip, struct scsi_inquiry *sd_inq,
405     dev_info_t **dip, mptsas_target_t *ptgt, int lun);

407 static int mptsas_create_phys_lun(dev_info_t *pdip, struct scsi_inquiry *sd,
408     char *guid, dev_info_t **dip, mptsas_target_t *ptgt, int lun);
409 static int mptsas_create_virt_lun(dev_info_t *pdip, struct scsi_inquiry *sd,
410     char *guid, dev_info_t **dip, mdi_pathinfo_t **pip, mptsas_target_t *ptgt,
411     int lun);

413 static void mptsas_offline_missed_luns(dev_info_t *pdip,
414     uint16_t *repluns, int lun_cnt, mptsas_target_t *ptgt);
415 static int mptsas_offline_lun(dev_info_t *pdip, dev_info_t *rdip,
416     mdi_pathinfo_t *rpip, uint_t flags);

418 static int mptsas_config_smp(dev_info_t *pdip, uint64_t sas_wwn,
419     dev_info_t **smp_dip);
420 static int mptsas_offline_smp(dev_info_t *pdip, mptsas_smp_t *smp_node,
421     uint_t flags);

423 static int mptsas_event_query(mptsas_t *mpt, mptsas_event_query_t *data,
424     int mode, int *rval);
425 static int mptsas_event_enable(mptsas_t *mpt, mptsas_event_enable_t *data,
426     int mode, int *rval);
427 static int mptsas_event_report(mptsas_t *mpt, mptsas_event_report_t *data,
428     int mode, int *rval);
429 static void mptsas_record_event(void *args);
430 static int mptsas_reg_access(mptsas_t *mpt, mptsas_reg_access_t *data,
431     int mode);

433 mptsas_target_t *mptsas_tgt_alloc(mptsas_t *, uint16_t, uint64_t,
434     uint32_t, mptsas_phymask_t, uint8_t);
```

```
435 static mptsas_smp_t *mptsas_smp_alloc(mptsas_t *, mptsas_smp_t *);
436 static int mptsas_online_smp(dev_info_t *pdip, mptsas_smp_t *smp_node,
437     dev_info_t **smp_dip);

439 /*
440  * Power management functions
441  */
442 static int mptsas_get_pci_cap(mptsas_t *mpt);
443 static int mptsas_init_pm(mptsas_t *mpt);

445 /*
446  * MPT MSI tunable:
447  *
448  * By default MSI is enabled on all supported platforms.
449  */
450 boolean_t mptsas_enable_msi = B_TRUE;
451 boolean_t mptsas_enable_msix = B_TRUE;
452 #endif /* ! codereview */
453 boolean_t mptsas_physical_bind_failed_page_83 = B_FALSE;

455 /*
456  * Global switch for use of MPI2.5 FAST PATH.
457  */
458 boolean_t mptsas3_use_fastpath = B_TRUE;

460 #endif /* ! codereview */
461 static int mptsas_register_intrs(mptsas_t *);
462 static void mptsas_unregister_intrs(mptsas_t *);
463 static int mptsas_add_intrs(mptsas_t *, int);
464 static void mptsas_rem_intrs(mptsas_t *);

466 /*
467  * FMA Prototypes
468  */
469 static void mptsas_fm_init(mptsas_t *mpt);
470 static void mptsas_fm_fini(mptsas_t *mpt);
471 static int mptsas_fm_error_cb(dev_info_t *, ddi_fm_error_t *, const void *);

473 extern pri_t minclsyspri, maxclsyspri;
474 /*
475  * NCPUS is used to determine some optimal configurations for number
476  * of threads created to perform specific jobs. If we are invoked because
477  * a disk is part of the root file system ncpus may still be 1 so check
478  * boot_ncpus as well.
479  */
480 extern int ncpus, boot_ncpus;
481 #define NCPUS   max(ncpus, boot_ncpus)
482 #endif /* ! codereview */

484 /*
485  * This device is created by the SCSI pseudo nexus driver (SCSI vHCI).  It is
486  * under this device that the paths to a physical device are created when
487  * MPxIO is used.
488  */
489 extern dev_info_t       *scsi_vhci_dip;

491 /*
492  * Tunable timeout value for Inquiry VPD page 0x83
493  * By default the value is 30 seconds.
494  */
495 int mptsas_inq83_retry_timeout = 30;

497 /*
498  * Tunable for default SCSI pkt timeout. Defaults to 5 seconds, which should
499  * be plenty for INQUIRY and REPORT_LUNS, which are the only commands currently
500  * issued by mptsas directly.
```



```
501  */
502 int mptsas_scsi_pkt_time = 5;

504 /*
505 #endif /* ! codereview */
506  * This is used to allocate memory for message frame storage, not for
507  * data I/O DMA. All message frames must be stored in the first 4G of
508  * physical memory.
509  */
510 ddi_dma_attr_t mptsas_dma_attrs = {
511         DMA_ATTR_V0,    /* attribute layout version             */
512         0x0ull,         /* address low - should be 0 (longlong) */
513         0xffffffffull,  /* address high - 32-bit max range      */
514         0x00ffffffull,  /* count max - max DMA object size      */
515         4,              /* allocation alignment requirements    */
516         0x78,           /* burstsizes - binary encoded values   */
517         1,              /* minxfer - gran. of DMA engine        */
518         0x00ffffffull,  /* maxxfer - gran. of DMA engine        */
519         0xffffffffull,  /* max segment size (DMA boundary)      */
520         MPTSAS_MAX_DMA_SEGS, /* scatter/gather list length      */
521         512,            /* granularity - device transfer size   */
522         0               /* flags, set to 0                      */
523 };

525 /*
526  * This is used for data I/O DMA memory allocation. (full 64-bit DMA
527  * physical addresses are supported.)
528  */
529 ddi_dma_attr_t mptsas_dma_attrs64 = {
530         DMA_ATTR_V0,    /* attribute layout version             */
531         0x0ull,         /* address low - should be 0 (longlong) */
532         0xffffffffffffffffull,  /* address high - 64-bit max    */
533         0x00ffffffull,  /* count max - max DMA object size      */
534         4,              /* allocation alignment requirements    */
535         0x78,           /* burstsizes - binary encoded values   */
536         1,              /* minxfer - gran. of DMA engine        */
537         0x00ffffffull,  /* maxxfer - gran. of DMA engine        */
538         0xffffffffull,  /* max segment size (DMA boundary)      */
539         MPTSAS_MAX_DMA_SEGS, /* scatter/gather list length      */
540         512,            /* granularity - device transfer size   */
541         0               /* flags, set to 0 */
289         DDI_DMA_RELAXED_ORDERING        /* flags, enable relaxed ordering */
542 };
```

_____*unchanged_portion_omitted_*

```
596 #define MPTSAS_MOD_STRING "MPTSAS3 HBA Driver 00.00.01"
344 #define MPTSAS_MOD_STRING "MPTSAS HBA Driver 00.00.00.24"

598 static struct modldrv modldrv = {
599         &mod_driverops, /* Type of module. This one is a driver */
600         MPTSAS_MOD_STRING, /* Name of the module. */
601         &mptsas_ops,    /* driver ops */
602 };
```

_____*unchanged_portion_omitted_*

```
607 #define TARGET_PROP     "target"
608 #define LUN_PROP        "lun"
609 #define LUN64_PROP      "lun64"
610 #define SAS_PROP        "sas-mpt"
611 #define MDI_GUID        "wwn"
612 #define NDI_GUID        "guid"
613 #define MPTSAS_DEV_GONE "mptsas_dev_gone"

615 /*
616  * Local static data
617  */
```

```
#if defined(MPTSAS_DEBUG)
uint32_t mptsas_debug_flags = 0x0;
/*
 * Flags to ignore these messages in local debug ring buffer.
 * Default is to ignore the watchsubr() output which normally happens
 * every second.
 */
uint32_t mptsas_dbglog_imask = 0x40000000;
uint32_t mptsas_test_timeout = 0;
uint32_t mptsas_debug_flags = 0;
#endif  /* defined(MPTSAS_DEBUG) */
uint32_t mptsas_debug_resets = 0;

static kmutex_t         mptsas_global_mutex;
static void             *mptsas3_state;         /* soft state ptr */
static void             *mptsas_state;          /* soft state ptr */
static krwlock_t        mptsas_global_rwlock;

static kmutex_t         mptsas_log_mutex;
static char             mptsas_log_buf[256];
_NOTE(MUTEX_PROTECTS_DATA(mptsas_log_mutex, mptsas_log_buf))

static mptsas_t *mptsas_head, *mptsas_tail;
static clock_t mptsas_scsi_watchdog_tick;
static clock_t mptsas_tick;
static timeout_id_t mptsas_reset_watch;
static timeout_id_t mptsas_timeout_id;
static int mptsas_timeouts_enabled = 0;

/*
 * Maximum number of MSI-X interrupts any instance of mptsas3 can use.
 * Note that if you want to increase this you may have to also bump the
 * value of ddi_msix_alloc_limit which defaults to 8.
 * Set to zero to fall back to other interrupt types.
 */
int mptsas3_max_msix_intrs = 8;

/*
 * Default length for extended auto request sense buffers.
 * All sense buffers need to be under the same alloc because there
 * is only one common top 32bits (of 64bits) address register.
 * Most requests only require 32 bytes, but some request >256.
 * We use rmalloc()/rmfree() on this additional memory to manage the
 * "extended" requests.
 */
int mptsas_extreq_sense_bufsize = 256*64;

/*
 * Believe that all software resrictions of having to run with DMA
 * attributes to limit allocation to the first 4G are removed.
 * However, this flag remains to enable quick switchback should suspicious
 * problems emerge.
 * Note that scsi_alloc_consistent_buf() does still adhering to allocating
 * 32 bit addressable memory, but we can cope if that is changed now.
 */
int mptsas_use_64bit_msgaddr = 1;

#endif /* ! codereview */
/*
 * warlock directives
 */
_NOTE(SCHEME_PROTECTS_DATA("unique per pkt", scsi_pkt \
        mptsas_cmd NcrTableIndirect buf scsi_cdb scsi_status))
_NOTE(SCHEME_PROTECTS_DATA("unique per pkt", smp_pkt))
_NOTE(SCHEME_PROTECTS_DATA("stable data", scsi_device scsi_address))
_NOTE(SCHEME_PROTECTS_DATA("No Mutex Needed", mptsas_tgt_private))
```



```
_NOTE(SCHEME_PROTECTS_DATA("No Mutex Needed", scsi_hba_tran::tran_tgt_private))

/*
 * SM - HBA statics
 */
char    *mptsas_driver_rev = MPTSAS_MOD_STRING;

#ifdef MPTSAS_DEBUG
void debug_enter(char *);
#endif

/*
 * Notes:
 *      - scsi_hba_init(9F) initializes SCSI HBA modules
 *      - must call scsi_hba_fini(9F) if modload() fails
 */
int
_init(void)
{
        int status;
        /* CONSTCOND */
        ASSERT(NO_COMPETING_THREADS);

        NDBG0(("_init"));

        status = ddi_soft_state_init(&mptsas3_state, MPTSAS_SIZE,
        status = ddi_soft_state_init(&mptsas_state, MPTSAS_SIZE,
            MPTSAS_INITIAL_SOFT_SPACE);
        if (status != 0) {
                return (status);
        }

        if ((status = scsi_hba_init(&modlinkage)) != 0) {
                ddi_soft_state_fini(&mptsas3_state);
                ddi_soft_state_fini(&mptsas_state);
                return (status);
        }

        mutex_init(&mptsas_global_mutex, NULL, MUTEX_DRIVER, NULL);
        rw_init(&mptsas_global_rwlock, NULL, RW_DRIVER, NULL);
        mutex_init(&mptsas_log_mutex, NULL, MUTEX_DRIVER, NULL);

        if ((status = mod_install(&modlinkage)) != 0) {
                mutex_destroy(&mptsas_log_mutex);
                rw_destroy(&mptsas_global_rwlock);
                mutex_destroy(&mptsas_global_mutex);
                ddi_soft_state_fini(&mptsas3_state);
                ddi_soft_state_fini(&mptsas_state);
                scsi_hba_fini(&modlinkage);
        }

        return (status);
}

/*
 * Notes:
 *      - scsi_hba_fini(9F) uninitializes SCSI HBA modules
 */
int
_fini(void)
{
        int     status;
        /* CONSTCOND */
        ASSERT(NO_COMPETING_THREADS);

        NDBG0(("_fini"));
```

```
746         if ((status = mod_remove(&modlinkage)) == 0) {
747                 ddi_soft_state_fini(&mptsas3_state);
425                 ddi_soft_state_fini(&mptsas_state);
748                 scsi_hba_fini(&modlinkage);
749                 mutex_destroy(&mptsas_global_mutex);
750                 rw_destroy(&mptsas_global_rwlock);
751                 mutex_destroy(&mptsas_log_mutex);
752         }
753         return (status);
754 }
_____unchanged_portion_omitted_

845 static void
846 mptsas_destroy_hashes(mptsas_t *mpt)
847 {
848         mptsas_target_t *tp;
849         mptsas_smp_t *sp;

851         for (tp = refhash_first(mpt->m_targets); tp != NULL;
852             tp = refhash_next(mpt->m_targets, tp)) {
853                 mutex_destroy(&tp->m_t_mutex);
854 #endif /* ! codereview */
855                 refhash_remove(mpt->m_targets, tp);
856         }
857         for (sp = refhash_first(mpt->m_smp_targets); sp != NULL;
858             sp = refhash_next(mpt->m_smp_targets, sp)) {
859                 refhash_remove(mpt->m_smp_targets, sp);
860         }
861         refhash_destroy(mpt->m_targets);
862         refhash_destroy(mpt->m_smp_targets);
863         mpt->m_targets = NULL;
864         mpt->m_smp_targets = NULL;
865 }

867 static int
868 mptsas_iport_attach(dev_info_t *dip, ddi_attach_cmd_t cmd)
869 {
870         dev_info_t              *pdip;
871         mptsas_t                *mpt;
872         scsi_hba_tran_t         *hba_tran;
873         char                    *iport = NULL;
874         char                    phymask[MPTSAS_MAX_PHYS];
875         mptsas_phymask_t        phy_mask = 0;
876         int                     dynamic_port = 0;
877         uint32_t                page_address;
878         char                    initiator_wwnstr[MPTSAS_WWN_STRLEN];
879         int                     rval = DDI_FAILURE;
880         int                     i = 0;
881         uint8_t                 numphys = 0;
882         uint8_t                 phy_id;
883         uint8_t                 phy_port = 0;
884         uint16_t                attached_devhdl = 0;
885         uint32_t                dev_info;
886         uint64_t                attached_sas_wwn;
887         uint16_t                dev_hdl;
888         uint16_t                pdev_hdl;
889         uint16_t                bay_num, enclosure, io_flags;
531         uint16_t                bay_num, enclosure;
890         char                    attached_wwnstr[MPTSAS_WWN_STRLEN];

892         /* CONSTCOND */
893         ASSERT(NO_COMPETING_THREADS);

895         switch (cmd) {
896         case DDI_ATTACH:
```



```
897                 break;

899         case DDI_RESUME:
900                 /*
901                  * If this a scsi-iport node, nothing to do here.
902                  */
903                 return (DDI_SUCCESS);

905         default:
906                 return (DDI_FAILURE);
907         }

909         pdip = ddi_get_parent(dip);

911         if ((hba_tran = ndi_flavorv_get(pdip, SCSA_FLAVOR_SCSI_DEVICE)) ==
912             NULL) {
913                 cmn_err(CE_WARN, "Failed attach iport because fail to "
914                     "get tran vector for the HBA node");
915                 return (DDI_FAILURE);
916         }

918         mpt = TRAN2MPT(hba_tran);
919         ASSERT(mpt != NULL);
920         if (mpt == NULL)
921                 return (DDI_FAILURE);

923         if ((hba_tran = ndi_flavorv_get(dip, SCSA_FLAVOR_SCSI_DEVICE)) ==
924             NULL) {
925                 mptsas_log(mpt, CE_WARN, "Failed attach iport because fail to "
926                     "get tran vector for the iport node");
927                 return (DDI_FAILURE);
928         }

930         /*
931          * Overwrite parent's tran_hba_private to iport's tran vector
932          */
933         hba_tran->tran_hba_private = mpt;

935         ddi_report_dev(dip);

937         /*
938          * Get SAS address for initiator port according dev_handle
939          */
940         iport = ddi_get_name_addr(dip);
941         if (iport && strncmp(iport, "v0", 2) == 0) {
942                 if (ddi_prop_update_int(DDI_DEV_T_NONE, dip,
943                     MPTSAS_VIRTUAL_PORT, 1) !=
944                     DDI_PROP_SUCCESS) {
945                         (void) ddi_prop_remove(DDI_DEV_T_NONE, dip,
946                             MPTSAS_VIRTUAL_PORT);
947                         mptsas_log(mpt, CE_WARN, "mptsas virtual port "
948                             "prop update failed");
949                         return (DDI_FAILURE);
950                 }
951                 return (DDI_SUCCESS);
952         }

954         mutex_enter(&mpt->m_mutex);
955         for (i = 0; i < MPTSAS_MAX_PHYS; i++) {
956                 bzero(phymask, sizeof (phymask));
957                 (void) sprintf(phymask,
958                     "%x", mpt->m_phy_info[i].phy_mask);
959                 if (strcmp(phymask, iport) == 0) {
960                         break;
961                 }
962         }
```

```
 964         if (i == MPTSAS_MAX_PHYS) {
 965                 mptsas_log(mpt, CE_WARN, "Failed attach port %s because port"
 966                     "seems not exist", iport);
 967                 mutex_exit(&mpt->m_mutex);
 968                 return (DDI_FAILURE);
 969         }

 971         phy_mask = mpt->m_phy_info[i].phy_mask;

 973         if (mpt->m_phy_info[i].port_flags & AUTO_PORT_CONFIGURATION)
 974                 dynamic_port = 1;
 975         else
 976                 dynamic_port = 0;

 978         /*
 979          * Update PHY info for smhba
 980          */
 981         if (mptsas_smhba_phy_init(mpt)) {
 982                 mutex_exit(&mpt->m_mutex);
 983                 mptsas_log(mpt, CE_WARN, "mptsas phy update "
 984                     "failed");
 985                 return (DDI_FAILURE);
 986         }

 988         mutex_exit(&mpt->m_mutex);

 990         numphys = 0;
 991         for (i = 0; i < MPTSAS_MAX_PHYS; i++) {
 992                 if ((phy_mask >> i) & 0x01) {
 993                         numphys++;
 994                 }
 995         }

 997         bzero(initiator_wwnstr, sizeof (initiator_wwnstr));
 998         (void) sprintf(initiator_wwnstr, "w%016"PRIx64,
 999             mpt->un.m_base_wwid);

1001         if (ddi_prop_update_string(DDI_DEV_T_NONE, dip,
1002             SCSI_ADDR_PROP_INITIATOR_PORT, initiator_wwnstr) !=
1003             DDI_PROP_SUCCESS) {
1004                 (void) ddi_prop_remove(DDI_DEV_T_NONE,
1005                     dip, SCSI_ADDR_PROP_INITIATOR_PORT);
1006                 mptsas_log(mpt, CE_WARN, "mptsas Initiator port "
1007                     "prop update failed");
1008                 return (DDI_FAILURE);
1009         }
1010         if (ddi_prop_update_int(DDI_DEV_T_NONE, dip,
1011             MPTSAS_NUM_PHYS, numphys) !=
1012             DDI_PROP_SUCCESS) {
1013                 (void) ddi_prop_remove(DDI_DEV_T_NONE, dip, MPTSAS_NUM_PHYS);
1014                 return (DDI_FAILURE);
1015         }

1017         if (ddi_prop_update_int(DDI_DEV_T_NONE, dip,
1018             "phymask", phy_mask) !=
1019             DDI_PROP_SUCCESS) {
1020                 (void) ddi_prop_remove(DDI_DEV_T_NONE, dip, "phymask");
1021                 mptsas_log(mpt, CE_WARN, "mptsas phy mask "
1022                     "prop update failed");
1023                 return (DDI_FAILURE);
1024         }

1026         if (ddi_prop_update_int(DDI_DEV_T_NONE, dip,
1027             "dynamic-port", dynamic_port) !=
1028             DDI_PROP_SUCCESS) {
```



```
1029                 (void) ddi_prop_remove(DDI_DEV_T_NONE, dip, "dynamic-port");
1030                 mptsas_log(mpt, CE_WARN, "mptsas dynamic port "
1031                     "prop update failed");
1032                 return (DDI_FAILURE);
1033         }
1034         if (ddi_prop_update_int(DDI_DEV_T_NONE, dip,
1035             MPTSAS_VIRTUAL_PORT, 0) !=
1036             DDI_PROP_SUCCESS) {
1037                 (void) ddi_prop_remove(DDI_DEV_T_NONE, dip,
1038                     MPTSAS_VIRTUAL_PORT);
1039                 mptsas_log(mpt, CE_WARN, "mptsas virtual port "
1040                     "prop update failed");
1041                 return (DDI_FAILURE);
1042         }
1043         mptsas_smhba_set_all_phy_props(mpt, dip, numphys, phy_mask,
1044             &attached_devhdl);

1046         mutex_enter(&mpt->m_mutex);
1047         page_address = (MPI2_SAS_DEVICE_PGAD_FORM_HANDLE &
1048             MPI2_SAS_DEVICE_PGAD_FORM_MASK) | (uint32_t)attached_devhdl;
1049         rval = mptsas_get_sas_device_page0(mpt, page_address, &dev_hdl,
1050             &attached_sas_wwn, &dev_info, &phy_port, &phy_id,
1051             &pdev_hdl, &bay_num, &enclosure, &io_flags);
 693             &pdev_hdl, &bay_num, &enclosure);
1052         if (rval != DDI_SUCCESS) {
1053                 mptsas_log(mpt, CE_WARN,
1054                     "Failed to get device page0 for handle:%d",
1055                     attached_devhdl);
1056                 mutex_exit(&mpt->m_mutex);
1057                 return (DDI_FAILURE);
1058         }

1060         for (i = 0; i < MPTSAS_MAX_PHYS; i++) {
1061                 bzero(phymask, sizeof (phymask));
1062                 (void) sprintf(phymask, "%x", mpt->m_phy_info[i].phy_mask);
1063                 if (strcmp(phymask, iport) == 0) {
1064                         (void) sprintf(&mpt->m_phy_info[i].smhba_info.path[0],
1065                             "%x",
1066                             mpt->m_phy_info[i].phy_mask);
1067                 }
1068         }
1069         mutex_exit(&mpt->m_mutex);

1071         bzero(attached_wwnstr, sizeof (attached_wwnstr));
1072         (void) sprintf(attached_wwnstr, "w%016"PRIx64,
1073             attached_sas_wwn);
1074         if (ddi_prop_update_string(DDI_DEV_T_NONE, dip,
1075             SCSI_ADDR_PROP_ATTACHED_PORT, attached_wwnstr) !=
1076             DDI_PROP_SUCCESS) {
1077                 (void) ddi_prop_remove(DDI_DEV_T_NONE,
1078                     dip, SCSI_ADDR_PROP_ATTACHED_PORT);
1079                 return (DDI_FAILURE);
1080         }

1082         /* Create kstats for each phy on this iport */

1084         mptsas_create_phy_stats(mpt, iport, dip);

1086         /*
1087          * register sas hba iport with mdi (MPxIO/vhci)
1088          */
1089         if (mdi_phci_register(MDI_HCI_CLASS_SCSI,
1090             dip, 0) == MDI_SUCCESS) {
1091                 mpt->m_mpxio_enable = TRUE;
1092         }
1093         return (DDI_SUCCESS);
```

```
1094 }

1096 /*
1097  * Notes:
1098  *      Set up all device state and allocate data structures,
1099  *      mutexes, condition variables, etc. for device operation.
1100  *      Add interrupts needed.
1101  *      Return DDI_SUCCESS if device is ready, else return DDI_FAILURE.
1102  */
1103 static int
1104 mptsas_attach(dev_info_t *dip, ddi_attach_cmd_t cmd)
1105 {
1106         mptsas_t                *mpt = NULL;
1107         int                     instance, i, j;
1108         int                     q_thread_num;
 750         int                     doneq_thread_num;
 751         char                    intr_added = 0;
1109         char                    map_setup = 0;
1110         char                    config_setup = 0;
1111         char                    hba_attach_setup = 0;
1112         char                    smp_attach_setup = 0;
1113         char                    mutex_init_done = 0;
1114         char                    event_taskq_create = 0;
1115         char                    dr_taskq_create = 0;
1116         char                    doneq_thread_create = 0;
1117         char                    txwq_thread_create = 0;
1118         char                    added_watchdog = 0;
1119 #endif /* ! codereview */
1120         scsi_hba_tran_t         *hba_tran;
1121         uint_t                  mem_bar = MEM_SPACE;
1122         int                     rval = DDI_FAILURE;

1124         /* CONSTCOND */
1125         ASSERT(NO_COMPETING_THREADS);

1127         if (scsi_hba_iport_unit_address(dip)) {
1128                 return (mptsas_iport_attach(dip, cmd));
1129         }

1131         switch (cmd) {
1132         case DDI_ATTACH:
1133                 break;

1135         case DDI_RESUME:
1136                 if ((hba_tran = ddi_get_driver_private(dip)) == NULL)
1137                         return (DDI_FAILURE);

1139                 mpt = TRAN2MPT(hba_tran);

1141                 if (!mpt) {
1142                         return (DDI_FAILURE);
1143                 }

1145                 /*
1146                  * Reset hardware and softc to "no outstanding commands"
1147                  * Note that a check condition can result on first command
1148                  * to a target.
1149                  */
1150                 mutex_enter(&mpt->m_mutex);

1152                 /*
1153                  * raise power.
1154                  */
1155                 if (mpt->m_options & MPTSAS_OPT_PM) {
1156                         mutex_exit(&mpt->m_mutex);
1157                         (void) pm_busy_component(dip, 0);
```



```
1158                         rval = pm_power_has_changed(dip, 0, PM_LEVEL_D0);
1159                         if (rval == DDI_SUCCESS) {
1160                                 mutex_enter(&mpt->m_mutex);
1161                         } else {
1162                                 /*
1163                                  * The pm_raise_power() call above failed,
1164                                  * and that can only occur if we were unable
1165                                  * to reset the hardware.  This is probably
1166                                  * due to unhealty hardware, and because
1167                                  * important filesystems(such as the root
1168                                  * filesystem) could be on the attached disks,
1169                                  * it would not be a good idea to continue,
1170                                  * as we won't be entirely certain we are
1171                                  * writing correct data.  So we panic() here
1172                                  * to not only prevent possible data corruption,
1173                                  * but to give developers or end users a hope
1174                                  * of identifying and correcting any problems.
1175                                  */
1176                                 fm_panic("mptsas could not reset hardware "
1177                                     "during resume");
1178                         }
1179                 }

1181                 mpt->m_suspended = 0;

1183                 /*
1184                  * Reinitialize ioc
1185                  */
1186                 mpt->m_softstate |= MPTSAS_SS_MSG_UNIT_RESET;
1187                 if (mptsas_init_chip(mpt, FALSE) == DDI_FAILURE) {
1188                         mutex_exit(&mpt->m_mutex);
1189                         if (mpt->m_options & MPTSAS_OPT_PM) {
1190                                 (void) pm_idle_component(dip, 0);
1191                         }
1192                         fm_panic("mptsas init chip fail during resume");
1193                 }
1194                 /*
1195                  * mptsas_update_driver_data needs interrupts so enable them
1196                  * first.
1197                  */
1198                 MPTSAS_ENABLE_INTR(mpt);
1199                 mptsas_update_driver_data(mpt);

1201                 /* start requests, if possible */
1202                 mptsas_restart_hba(mpt);

1204                 mutex_exit(&mpt->m_mutex);

1206                 /*
1207                  * Restart watch thread
1208                  */
1209                 mutex_enter(&mptsas_global_mutex);
1210                 if (mptsas_timeout_id == 0) {
1211                         mptsas_timeout_id = timeout(mptsas_watch, NULL,
1212                             mptsas_tick);
1213                         mptsas_timeouts_enabled = 1;
1214                 }
1215                 mutex_exit(&mptsas_global_mutex);

1217                 /* report idle status to pm framework */
1218                 if (mpt->m_options & MPTSAS_OPT_PM) {
1219                         (void) pm_idle_component(dip, 0);
1220                 }

1222                 return (DDI_SUCCESS);
```

```
1224         default:
1225                 return (DDI_FAILURE);

1227         }

1229         instance = ddi_get_instance(dip);

1231         /*
1232          * Allocate softc information.
1233          */
1234         if (ddi_soft_state_zalloc(mptsas3_state, instance) != DDI_SUCCESS) {
 760         if (ddi_soft_state_zalloc(mptsas_state, instance) != DDI_SUCCESS) {
1235                 mptsas_log(NULL, CE_WARN,
1236                     "mptsas%d: cannot allocate soft state", instance);
1237                 goto fail;
1238         }

1240         mpt = ddi_get_soft_state(mptsas3_state, instance);
 766         mpt = ddi_get_soft_state(mptsas_state, instance);

1242         if (mpt == NULL) {
1243                 mptsas_log(NULL, CE_WARN,
1244                     "mptsas%d: cannot get soft state", instance);
1245                 goto fail;
1246         }

1248         /* Indicate that we are 'sizeof (scsi_*(9S))' clean. */
1249         scsi_size_clean(dip);

1251         mpt->m_dip = dip;
1252         mpt->m_instance = instance;

1254         /* Make a per-instance copy of the structures */
1255         mpt->m_io_dma_attr = mptsas_dma_attrs64;
1256         if (mptsas_use_64bit_msgaddr) {
1257                 mpt->m_msg_dma_attr = mptsas_dma_attrs64;
1258         } else {
1259 #endif /* ! codereview */
1260                 mpt->m_msg_dma_attr = mptsas_dma_attrs;
1261         }
1262 #endif /* ! codereview */
1263         mpt->m_reg_acc_attr = mptsas_dev_attr;
1264         mpt->m_dev_acc_attr = mptsas_dev_attr;

1266         /*
1267          * Round down the arq sense buffer size to nearest 16 bytes.
1268          */
1269         mpt->m_req_sense_size = EXTCMDS_STATUS_SIZE;

1271         /*
1272 #endif /* ! codereview */
1273          * Initialize FMA
1274          */
1275         mpt->m_fm_capabilities = ddi_getprop(DDI_DEV_T_ANY, mpt->m_dip,
1276             DDI_PROP_CANSLEEP | DDI_PROP_DONTPASS, "fm-capable",
1277             DDI_FM_EREPORT_CAPABLE | DDI_FM_ACCCHK_CAPABLE |
1278             DDI_FM_DMACHK_CAPABLE | DDI_FM_ERRCB_CAPABLE);

1280         mptsas_fm_init(mpt);

1282         if (mptsas_alloc_handshake_msg(mpt,
1283             sizeof (Mpi2SCSITaskManagementRequest_t)) == DDI_FAILURE) {
1284                 mptsas_log(mpt, CE_WARN, "cannot initialize handshake msg.");
1285                 goto fail;
1286         }
```



```
1288         /*
1289          * Setup configuration space
1290          */
1291         if (mptsas_config_space_init(mpt) == FALSE) {
1292                 mptsas_log(mpt, CE_WARN, "mptsas_config_space_init failed");
1293                 goto fail;
1294         }
1295         config_setup++;

1297         if (ddi_regs_map_setup(dip, mem_bar, (caddr_t *)&mpt->m_reg,
1298             0, 0, &mpt->m_reg_acc_attr, &mpt->m_datap) != DDI_SUCCESS) {
1299                 mptsas_log(mpt, CE_WARN, "map setup failed");
1300                 goto fail;
1301         }
1302         map_setup++;

1304         /*
1305          * A taskq is created for dealing with the event handler
1306          */
1307         if ((mpt->m_event_taskq = ddi_taskq_create(dip, "mptsas_event_taskq",
1308             1, TASKQ_DEFAULTPRI, 0)) == NULL) {
1309                 mptsas_log(mpt, CE_NOTE, "ddi_taskq_create failed");
1310                 goto fail;
1311         }
1312         event_taskq_create++;

1314         /*
1315          * A taskq is created for dealing with dr events
1316          */
1317         if ((mpt->m_dr_taskq = ddi_taskq_create(dip,
1318             "mptsas_dr_taskq",
1319             1, TASKQ_DEFAULTPRI, 0)) == NULL) {
1320                 mptsas_log(mpt, CE_NOTE, "ddi_taskq_create for discovery "
1321                     "failed");
1322                 goto fail;
1323         }
1324         dr_taskq_create++;

1326         cv_init(&mpt->m_qthread_cv, NULL, CV_DRIVER, NULL);
1327         mutex_init(&mpt->m_qthread_mutex, NULL, MUTEX_DRIVER, NULL);

1329         i = ddi_prop_get_int(DDI_DEV_T_ANY, dip,
1330             0, "mptsas_enable_txwq_prop", NCPUS > 1);
1331         if (i) {
1332                 mpt->m_txwq_thread_n = NUM_TX_WAITQ;
1333                 mpt->m_txwq_enabled = FALSE;
1334                 if (ddi_prop_get_int(DDI_DEV_T_ANY, dip,
1335                     0, "mptsas_allow_txwq_jumping", 0)) {
1336                         mpt->m_txwq_allow_q_jumping = TRUE;
1337                 }
1338                 i = ddi_prop_get_int(DDI_DEV_T_ANY, dip,
1339                     0, "mptsas_txwq_threashold_prop", 80000);
1340                 mpt->m_txwq_thread_threshold = (uint16_t)i;
1341         } else {
1342                 mpt->m_txwq_thread_n = 0;
1343                 mpt->m_txwq_enabled = FALSE;
1344         }

1346         if (mpt->m_txwq_thread_n) {
1347                 mutex_enter(&mpt->m_qthread_mutex);
1348                 for (j = 0; j < NUM_TX_WAITQ; j++) {
1349                         mutex_init(&mpt->m_tx_waitq[j].txwq_mutex, NULL,
1350                             MUTEX_DRIVER,
1351                             NULL);
1352                         cv_init(&mpt->m_tx_waitq[j].txwq_cv, NULL, CV_DRIVER,
1353                             NULL);
```

```
                        cv_init(&mpt->m_tx_waitq[j].txwq_drain_cv, NULL,
                            CV_DRIVER, NULL);
                        mpt->m_tx_waitq[j].txwq_active = TRUE;
                        mpt->m_tx_waitq[j].txwq_draining = FALSE;
                        mpt->m_tx_waitq[j].txwq_cmdq = NULL;
                        mpt->m_tx_waitq[j].txwq_qtail =
                            &mpt->m_tx_waitq[j].txwq_cmdq;
                        mutex_enter(&mpt->m_tx_waitq[j].txwq_mutex);
                        mpt->m_tx_waitq[j].arg.mpt = mpt;
                        mpt->m_tx_waitq[j].arg.t = j;
                        mpt->m_tx_waitq[j].txwq_threadp =
                            thread_create(NULL, 0, mptsas_tx_waitq_thread,
                            &mpt->m_tx_waitq[j].arg,
                            0, &p0, TS_RUN, maxclsyspri - 10);
                        mutex_exit(&mpt->m_tx_waitq[j].txwq_mutex);
                }
                mutex_exit(&mpt->m_qthread_mutex);
                txwq_thread_create++;
        }

#endif /* ! codereview */
        mpt->m_doneq_thread_threshold = ddi_prop_get_int(DDI_DEV_T_ANY, dip,
            0, "mptsas_doneq_thread_threshold_prop", 10);
        mpt->m_doneq_length_threshold = ddi_prop_get_int(DDI_DEV_T_ANY, dip,
            0, "mptsas_doneq_length_threshold_prop", 8);
        mpt->m_doneq_thread_n = ddi_prop_get_int(DDI_DEV_T_ANY, dip,
            0, "mptsas_doneq_thread_n_prop", min(NCPUS, 8));
            0, "mptsas_doneq_thread_n_prop", 8);

        if (mpt->m_doneq_thread_n) {
                mutex_enter(&mpt->m_qthread_mutex);
                cv_init(&mpt->m_doneq_thread_cv, NULL, CV_DRIVER, NULL);
                mutex_init(&mpt->m_doneq_mutex, NULL, MUTEX_DRIVER, NULL);

                mutex_enter(&mpt->m_doneq_mutex);
                mpt->m_doneq_thread_id =
                    kmem_zalloc(sizeof (mptsas_doneq_thread_list_t)
                    * mpt->m_doneq_thread_n, KM_SLEEP);

                for (j = 0; j < mpt->m_doneq_thread_n; j++) {
                        cv_init(&mpt->m_doneq_thread_id[j].cv, NULL,
                            CV_DRIVER, NULL);
                        mutex_init(&mpt->m_doneq_thread_id[j].mutex, NULL,
                            MUTEX_DRIVER, NULL);
                        mutex_enter(&mpt->m_doneq_thread_id[j].mutex);
                        mpt->m_doneq_thread_id[j].flag |=
                            MPTSAS_DONEQ_THREAD_ACTIVE;
                        mpt->m_doneq_thread_id[j].arg.mpt = mpt;
                        mpt->m_doneq_thread_id[j].arg.t = j;
                        mpt->m_doneq_thread_id[j].threadp =
                            thread_create(NULL, 0, mptsas_doneq_thread,
                            &mpt->m_doneq_thread_id[j].arg,
                            0, &p0, TS_RUN, maxclsyspri - 10);
                        mpt->m_doneq_thread_id[j].dlist.dl_tail =
                            &mpt->m_doneq_thread_id[j].dlist.dl_q;
                            0, &p0, TS_RUN, minclsyspri);
                        mpt->m_doneq_thread_id[j].donetail =
                            &mpt->m_doneq_thread_id[j].doneq;
                        mutex_exit(&mpt->m_doneq_thread_id[j].mutex);
                }
                mutex_exit(&mpt->m_qthread_mutex);
                mutex_exit(&mpt->m_doneq_mutex);
                doneq_thread_create++;
        }

        /*
```



```
         * Disable hardware interrupt since we're not ready to
         * handle it yet.
         */
        MPTSAS_DISABLE_INTR(mpt);

        /*
         * Initialize mutex used in interrupt handler.
         * We don't support hi-level so the mutex's are all adaptive
         * and we don't want to register the interrupts until we get
         * the chip type information from _init_chip() below.
         * Otherwise we would use DDI_INTR_PRI(mpt->m_intr_pri)
         * rather than NULL in the mutex_init() calls.
         */
        mutex_init(&mpt->m_mutex, NULL, MUTEX_DRIVER, NULL);
        /* Initialize mutex used in interrupt handler */
        mutex_init(&mpt->m_mutex, NULL, MUTEX_DRIVER,
            DDI_INTR_PRI(mpt->m_intr_pri));
        mutex_init(&mpt->m_passthru_mutex, NULL, MUTEX_DRIVER, NULL);
        mutex_init(&mpt->m_tx_waitq_mutex, NULL, MUTEX_DRIVER,
            DDI_INTR_PRI(mpt->m_intr_pri));
        for (i = 0; i < MPTSAS_MAX_PHYS; i++) {
                mutex_init(&mpt->m_phy_info[i].smhba_info.phy_mutex,
                    NULL, MUTEX_DRIVER, NULL);
                    NULL, MUTEX_DRIVER,
                    DDI_INTR_PRI(mpt->m_intr_pri));
        }

        cv_init(&mpt->m_cv, NULL, CV_DRIVER, NULL);
        cv_init(&mpt->m_passthru_cv, NULL, CV_DRIVER, NULL);
        cv_init(&mpt->m_fw_cv, NULL, CV_DRIVER, NULL);
        cv_init(&mpt->m_config_cv, NULL, CV_DRIVER, NULL);
        cv_init(&mpt->m_fw_diag_cv, NULL, CV_DRIVER, NULL);
        mutex_init_done++;

        /*
         * Disable hardware interrupt since we're not ready to
         * handle it yet.
         */
        MPTSAS_DISABLE_INTR(mpt);
        if (mptsas_register_intrs(mpt) == FALSE)
                goto fail;
        intr_added++;

        mutex_enter(&mpt->m_mutex);
        /*
         * Initialize power management component
         */
        if (mpt->m_options & MPTSAS_OPT_PM) {
                if (mptsas_init_pm(mpt)) {
                        mutex_exit(&mpt->m_mutex);
                        mptsas_log(mpt, CE_WARN, "mptsas pm initialization "
                            "failed");
                        goto fail;
                }
        }

        /*
         * Initialize chip using Message Unit Reset, if allowed
         */
        mpt->m_softstate |= MPTSAS_SS_MSG_UNIT_RESET;
        if (mptsas_init_chip(mpt, TRUE) == DDI_FAILURE) {
                mutex_exit(&mpt->m_mutex);
                mptsas_log(mpt, CE_WARN, "mptsas chip initialization failed");
                goto fail;
        }
```

```
1461         /*
1462          * Fill in the phy_info structure and get the base WWID
1463          */
1464         if (mptsas_get_manufacture_page5(mpt) == DDI_FAILURE) {
1465                 mptsas_log(mpt, CE_WARN,
1466                     "mptsas_get_manufacture_page5 failed!");
1467                 goto fail;
1468         }

1470         if (mptsas_get_sas_io_unit_page_hndshk(mpt)) {
1471                 mptsas_log(mpt, CE_WARN,
1472                     "mptsas_get_sas_io_unit_page_hndshk failed!");
1473                 goto fail;
1474         }

1476         if (mptsas_get_manufacture_page0(mpt) == DDI_FAILURE) {
1477                 mptsas_log(mpt, CE_WARN,
1478                     "mptsas_get_manufacture_page0 failed!");
1479                 goto fail;
1480         }

1482         /*
1483          * If we only have one interrupt the default for doneq_thread_threshold
1484          * should be 0 so that all completion processing goes to the threads.
1485          * Only change it if it wasn't set from .conf file.
1486          */
1487         if (mpt->m_doneq_thread_n != 0 &&
1488             ddi_prop_exists(DDI_DEV_T_ANY, dip,
1489             0, "mptsas_doneq_length_threshold_prop") == 0 &&
1490             mpt->m_intr_cnt == 1) {
1491                 mpt->m_doneq_length_threshold = 0;
1492         }


1495 #endif /* ! codereview */
1496         mutex_exit(&mpt->m_mutex);

1498         /*
1499          * Register the iport for multiple port HBA
1500          */
1501         mptsas_iport_register(mpt);

1503         /*
1504          * initialize SCSI HBA transport structure
1505          */
1506         if (mptsas_hba_setup(mpt) == FALSE)
1507                 goto fail;
1508         hba_attach_setup++;

1510         if (mptsas_smp_setup(mpt) == FALSE)
1511                 goto fail;
1512         smp_attach_setup++;

1514         if (mptsas_cache_create(mpt) == FALSE)
1515                 goto fail;

1517         mpt->m_scsi_reset_delay = ddi_prop_get_int(DDI_DEV_T_ANY,
1518             dip, 0, "scsi-reset-delay", SCSI_DEFAULT_RESET_DELAY);
1519         if (mpt->m_scsi_reset_delay == 0) {
1520                 mptsas_log(mpt, CE_NOTE,
1521                     "scsi_reset_delay of 0 is not recommended,"
1522                     " resetting to SCSI_DEFAULT_RESET_DELAY\n");
1523                 mpt->m_scsi_reset_delay = SCSI_DEFAULT_RESET_DELAY;
1524         }

1526         /*
```



```
1527          * Initialize the wait and done FIFO queue
1528          */
1529         mpt->m_dlist.dl_tail = &mpt->m_dlist.dl_q;
 887         mpt->m_donetail = &mpt->m_doneq;
1530         mpt->m_waitqtail = &mpt->m_waitq;
 889         mpt->m_tx_waitqtail = &mpt->m_tx_waitq;
 890         mpt->m_tx_draining = 0;

1532         /*
1533          * ioc cmd queue initialize
1534          */
1535         mpt->m_ioc_event_cmdtail = &mpt->m_ioc_event_cmdq;
1536         mpt->m_dev_handle = 0xFFFF;

1538         MPTSAS_ENABLE_INTR(mpt);

1540         /*
1541          * enable event notification
1542          */
1543         mutex_enter(&mpt->m_mutex);
1544         if (mptsas_ioc_enable_event_notification(mpt)) {
1545                 mutex_exit(&mpt->m_mutex);
1546                 goto fail;
1547         }
1548         mutex_exit(&mpt->m_mutex);

1550         /*
1551          * used for mptsas_watch
1552          */
1553         mptsas_list_add(mpt);

1555         mutex_enter(&mptsas_global_mutex);
1556         if (mptsas_timeouts_enabled == 0) {
1557                 mptsas_scsi_watchdog_tick = ddi_prop_get_int(DDI_DEV_T_ANY,
1558                     dip, 0, "scsi-watchdog-tick", DEFAULT_WD_TICK);

1560                 mptsas_tick = mptsas_scsi_watchdog_tick *
1561                     drv_usectohz((clock_t)1000000);

1563                 mptsas_timeout_id = timeout(mptsas_watch, NULL, mptsas_tick);
1564                 mptsas_timeouts_enabled = 1;
1565         }
1566         mutex_exit(&mptsas_global_mutex);
1567         added_watchdog++;

1569         /*
1570          * Initialize PHY info for smhba.
1571          * This requires watchdog to be enabled otherwise if interrupts
1572          * don't work the system will hang.
 911          * Initialize PHY info for smhba
1573          */
1574         if (mptsas_smhba_setup(mpt)) {
1575                 mptsas_log(mpt, CE_WARN, "mptsas phy initialization "
1576                     "failed");
1577                 goto fail;
1578         }

1580         /* Check all dma handles allocated in attach */
1581         if ((mptsas_check_dma_handle(mpt->m_dma_req_frame_hdl)
1582             != DDI_SUCCESS) ||
1583             (mptsas_check_dma_handle(mpt->m_dma_req_sense_hdl)
1584             != DDI_SUCCESS) ||
1585 #endif /* ! codereview */
1586             (mptsas_check_dma_handle(mpt->m_dma_reply_frame_hdl)
1587             != DDI_SUCCESS) ||
1588             (mptsas_check_dma_handle(mpt->m_dma_free_queue_hdl)
```

```
            != DDI_SUCCESS) ||
            (mptsas_check_dma_handle(mpt->m_dma_post_queue_hdl)
            != DDI_SUCCESS) ||
            (mptsas_check_dma_handle(mpt->m_hshk_dma_hdl)
            != DDI_SUCCESS)) {
                goto fail;
        }

        /* Check all acc handles allocated in attach */
        if ((mptsas_check_acc_handle(mpt->m_datap) != DDI_SUCCESS) ||
            (mptsas_check_acc_handle(mpt->m_acc_req_frame_hdl)
            != DDI_SUCCESS) ||
            (mptsas_check_acc_handle(mpt->m_acc_req_sense_hdl)
            != DDI_SUCCESS) ||
#endif /* ! codereview */
            (mptsas_check_acc_handle(mpt->m_acc_reply_frame_hdl)
            != DDI_SUCCESS) ||
            (mptsas_check_acc_handle(mpt->m_acc_free_queue_hdl)
            != DDI_SUCCESS) ||
            (mptsas_check_acc_handle(mpt->m_acc_post_queue_hdl)
            != DDI_SUCCESS) ||
            (mptsas_check_acc_handle(mpt->m_hshk_acc_hdl)
            != DDI_SUCCESS) ||
            (mptsas_check_acc_handle(mpt->m_config_handle)
            != DDI_SUCCESS)) {
                goto fail;
        }

        /*
         * After this point, we are not going to fail the attach.
         */
        /*
         * used for mptsas_watch
         */
        mptsas_list_add(mpt);

        mutex_enter(&mptsas_global_mutex);
        if (mptsas_timeouts_enabled == 0) {
                mptsas_scsi_watchdog_tick = ddi_prop_get_int(DDI_DEV_T_ANY,
                    dip, 0, "scsi-watchdog-tick", DEFAULT_WD_TICK);

                mptsas_tick = mptsas_scsi_watchdog_tick *
                    drv_usectohz((clock_t)1000000);

                mptsas_timeout_id = timeout(mptsas_watch, NULL, mptsas_tick);
                mptsas_timeouts_enabled = 1;
        }
        mutex_exit(&mptsas_global_mutex);

        /* Print message of HBA present */
        ddi_report_dev(dip);

        /* report idle status to pm framework */
        if (mpt->m_options & MPTSAS_OPT_PM) {
                (void) pm_idle_component(dip, 0);
        }

        return (DDI_SUCCESS);

fail:
        mptsas_log(mpt, CE_WARN, "attach failed");
        mptsas_fm_ereport(mpt, DDI_FM_DEVICE_NO_RESPONSE);
        ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_LOST);
        if (mpt) {
                /* deallocate in reverse order */
                if (added_watchdog) {
```



```
                        mptsas_list_del(mpt);
#endif /* ! codereview */
                        mutex_enter(&mptsas_global_mutex);

                        if (mptsas_timeout_id && (mptsas_head == NULL)) {
                                timeout_id_t tid = mptsas_timeout_id;
                                mptsas_timeouts_enabled = 0;
                                mptsas_timeout_id = 0;
                                mutex_exit(&mptsas_global_mutex);
                                (void) untimeout(tid);
                                mutex_enter(&mptsas_global_mutex);
                        }
                        mutex_exit(&mptsas_global_mutex);
                }

                /* deallocate in reverse order */
                mptsas_cache_destroy(mpt);

                if (smp_attach_setup) {
                        mptsas_smp_teardown(mpt);
                }
                if (hba_attach_setup) {
                        mptsas_hba_teardown(mpt);
                }

                if (mpt->m_targets)
                        refhash_destroy(mpt->m_targets);
                if (mpt->m_smp_targets)
                        refhash_destroy(mpt->m_smp_targets);

                if (mpt->m_active) {
                        mptsas_free_active_slots(mpt);
                }
                if (mpt->m_intr_cnt) {
                if (intr_added) {
                        mptsas_unregister_intrs(mpt);
                }

                if (doneq_thread_create) {
                        mutex_enter(&mpt->m_qthread_mutex);
                        q_thread_num = mpt->m_doneq_thread_n;
                        for (j = 0; j < q_thread_num; j++) {
                        mutex_enter(&mpt->m_doneq_mutex);
                        doneq_thread_num = mpt->m_doneq_thread_n;
                        for (j = 0; j < mpt->m_doneq_thread_n; j++) {
                                mutex_enter(&mpt->m_doneq_thread_id[j].mutex);
                                mpt->m_doneq_thread_id[j].flag &=
                                    (~MPTSAS_DONEQ_THREAD_ACTIVE);
                                cv_signal(&mpt->m_doneq_thread_id[j].cv);
                                mutex_exit(&mpt->m_doneq_thread_id[j].mutex);
                        }
                        while (mpt->m_doneq_thread_n) {
                                cv_wait(&mpt->m_qthread_cv,
                                    &mpt->m_qthread_mutex);
                                cv_wait(&mpt->m_doneq_thread_cv,
                                    &mpt->m_doneq_mutex);
                        }
                        for (j = 0; j < q_thread_num; j++) {
                        for (j = 0; j < doneq_thread_num; j++) {
                                cv_destroy(&mpt->m_doneq_thread_id[j].cv);
                                mutex_destroy(&mpt->m_doneq_thread_id[j].mutex);
                        }
                        kmem_free(mpt->m_doneq_thread_id,
                            sizeof (mptsas_doneq_thread_list_t)
                            * q_thread_num);
                        mutex_exit(&mpt->m_qthread_mutex);
```

```
1696                 }
1697                 if (txwq_thread_create) {
1698                         mutex_enter(&mpt->m_qthread_mutex);
1699                         q_thread_num = mpt->m_txwq_thread_n;
1700                         for (j = 0; j < q_thread_num; j++) {
1701                                 mutex_enter(&mpt->m_tx_waitq[j].txwq_mutex);
1702                                 mpt->m_tx_waitq[j].txwq_active = FALSE;
1703                                 cv_signal(&mpt->m_tx_waitq[j].txwq_cv);
1704                                 mutex_exit(&mpt->m_tx_waitq[j].txwq_mutex);
1705                         }
1706                         while (mpt->m_txwq_thread_n) {
1707                                 cv_wait(&mpt->m_qthread_cv,
1708                                     &mpt->m_qthread_mutex);
1709                         }
1710                         for (j = 0; j < q_thread_num; j++) {
1711                                 cv_destroy(&mpt->m_tx_waitq[j].txwq_cv);
1712                                 cv_destroy(&mpt->m_tx_waitq[j].txwq_drain_cv);
1713                                 mutex_destroy(&mpt->m_tx_waitq[j].txwq_mutex);
1714                         }
 997                         * doneq_thread_num);
 998                         mutex_exit(&mpt->m_doneq_mutex);
 999                         cv_destroy(&mpt->m_doneq_thread_cv);
1000                         mutex_destroy(&mpt->m_doneq_mutex);
1715                 }
1716                 if (event_taskq_create) {
1717                         ddi_taskq_destroy(mpt->m_event_taskq);
1718                 }
1719                 if (dr_taskq_create) {
1720                         ddi_taskq_destroy(mpt->m_dr_taskq);
1721                 }
1722                 if (mutex_init_done) {
1723                         mutex_destroy(&mpt->m_qthread_mutex);
1009                         mutex_destroy(&mpt->m_tx_waitq_mutex);
1724                         mutex_destroy(&mpt->m_passthru_mutex);
1725                         mutex_destroy(&mpt->m_mutex);
1726                         for (i = 0; i < MPTSAS_MAX_PHYS; i++) {
1727                                 mutex_destroy(
1728                                     &mpt->m_phy_info[i].smhba_info.phy_mutex);
1729                         }
1730                         cv_destroy(&mpt->m_qthread_cv);
1731 #endif /* ! codereview */
1732                         cv_destroy(&mpt->m_cv);
1733                         cv_destroy(&mpt->m_passthru_cv);
1734                         cv_destroy(&mpt->m_fw_cv);
1735                         cv_destroy(&mpt->m_config_cv);
1736                         cv_destroy(&mpt->m_fw_diag_cv);
1737                 }

1739                 if (map_setup) {
1740                         mptsas_cfg_fini(mpt);
1741                 }
1742                 if (config_setup) {
1743                         mptsas_config_space_fini(mpt);
1744                 }
1745                 mptsas_free_handshake_msg(mpt);
1746                 mptsas_hba_fini(mpt);

1748                 mptsas_fm_fini(mpt);
1749                 ddi_soft_state_free(mptsas3_state, instance);
1016                 ddi_soft_state_free(mptsas_state, instance);
1750                 ddi_prop_remove_all(dip);
1751         }
1752         return (DDI_FAILURE);
1753 }
_____unchanged_portion_omitted_
```



```
1954 static int
1955 mptsas_do_detach(dev_info_t *dip)
1956 {
1957         mptsas_t        *mpt;
1958         scsi_hba_tran_t *tran;
1959         int             circ = 0;
1960         int             circ1 = 0;
1961         mdi_pathinfo_t  *pip = NULL;
1962         int             i;
1963         int             q_thread_num = 0;
1230         int             doneq_thread_num = 0;

1965         NDBG0(("mptsas_do_detach: dip=0x%p", (void *)dip));

1967         if ((tran = ndi_flavorv_get(dip, SCSA_FLAVOR_SCSI_DEVICE)) == NULL)
1968                 return (DDI_FAILURE);

1970         mpt = TRAN2MPT(tran);
1971         if (!mpt) {
1972                 return (DDI_FAILURE);
1973         }
1974         /*
1975          * Still have pathinfo child, should not detach mpt driver
1976          */
1977         if (scsi_hba_iport_unit_address(dip)) {
1978                 if (mpt->m_mpxio_enable) {
1979                         /*
1980                          * MPxIO enabled for the iport
1981                          */
1982                         ndi_devi_enter(scsi_vhci_dip, &circ1);
1983                         ndi_devi_enter(dip, &circ);
1984                         while (pip = mdi_get_next_client_path(dip, NULL)) {
1985                                 if (mdi_pi_free(pip, 0) == MDI_SUCCESS) {
1986                                         continue;
1987                                 }
1988                                 ndi_devi_exit(dip, circ);
1989                                 ndi_devi_exit(scsi_vhci_dip, circ1);
1990                                 NDBG12(("detach failed because of "
1991                                     "outstanding path info"));
1992                                 return (DDI_FAILURE);
1993                         }
1994                         ndi_devi_exit(dip, circ);
1995                         ndi_devi_exit(scsi_vhci_dip, circ1);
1996                         (void) mdi_phci_unregister(dip, 0);
1997                 }

1999                 ddi_prop_remove_all(dip);

2001                 return (DDI_SUCCESS);
2002         }

2004         /* Make sure power level is D0 before accessing registers */
2005         if (mpt->m_options & MPTSAS_OPT_PM) {
2006                 (void) pm_busy_component(dip, 0);
2007                 if (mpt->m_power_level != PM_LEVEL_D0) {
2008                         if (pm_raise_power(dip, 0, PM_LEVEL_D0) !=
2009                             DDI_SUCCESS) {
2010                                 mptsas_log(mpt, CE_WARN,
2011                                     "mptsas3%d: Raise power request failed.",
1278                                     "mptsas%d: Raise power request failed.",
2012                                     mpt->m_instance);
2013                                 (void) pm_idle_component(dip, 0);
2014                                 return (DDI_FAILURE);
2015                         }
2016                 }
2017         }
```

```
2019         /*
2020          * Send RAID action system shutdown to sync IR.  After action, send a
2021          * Message Unit Reset. Since after that DMA resource will be freed,
2022          * set ioc to READY state will avoid HBA initiated DMA operation.
2023          */
2024         mutex_enter(&mpt->m_mutex);
2025         MPTSAS_DISABLE_INTR(mpt);
2026         mptsas_raid_action_system_shutdown(mpt);
2027         mpt->m_softstate |= MPTSAS_SS_MSG_UNIT_RESET;
2028         (void) mptsas_ioc_reset(mpt, FALSE);
2029         mutex_exit(&mpt->m_mutex);
2030         mptsas_rem_intrs(mpt);
2031         ddi_taskq_destroy(mpt->m_event_taskq);
2032         ddi_taskq_destroy(mpt->m_dr_taskq);

2034         if (mpt->m_doneq_thread_n) {
2035                 mutex_enter(&mpt->m_qthread_mutex);
2036                 q_thread_num = mpt->m_doneq_thread_n;
1302                 mutex_enter(&mpt->m_doneq_mutex);
1303                 doneq_thread_num = mpt->m_doneq_thread_n;
2037                 for (i = 0; i < mpt->m_doneq_thread_n; i++) {
2038                         mutex_enter(&mpt->m_doneq_thread_id[i].mutex);
2039                         mpt->m_doneq_thread_id[i].flag &=
2040                             (~MPTSAS_DONEQ_THREAD_ACTIVE);
2041                         cv_signal(&mpt->m_doneq_thread_id[i].cv);
2042                         mutex_exit(&mpt->m_doneq_thread_id[i].mutex);
2043                 }
2044                 while (mpt->m_doneq_thread_n) {
2045                         cv_wait(&mpt->m_qthread_cv,
2046                             &mpt->m_qthread_mutex);
1312                         cv_wait(&mpt->m_doneq_thread_cv,
1313                             &mpt->m_doneq_mutex);
2047                 }
2048                 for (i = 0;  i < q_thread_num; i++) {
1315                 for (i = 0;  i < doneq_thread_num; i++) {
2049                         cv_destroy(&mpt->m_doneq_thread_id[i].cv);
2050                         mutex_destroy(&mpt->m_doneq_thread_id[i].mutex);
2051                 }
2052                 kmem_free(mpt->m_doneq_thread_id,
2053                     sizeof (mptsas_doneq_thread_list_t)
2054                     * q_thread_num);
2055                 mutex_exit(&mpt->m_qthread_mutex);
2056         }
2057         if (mpt->m_txwq_thread_n) {
2058                 mutex_enter(&mpt->m_qthread_mutex);
2059                 q_thread_num = mpt->m_txwq_thread_n;
2060                 for (i = 0; i < q_thread_num; i++) {
2061                         mutex_enter(&mpt->m_tx_waitq[i].txwq_mutex);
2062                         mpt->m_tx_waitq[i].txwq_active = FALSE;
2063                         cv_signal(&mpt->m_tx_waitq[i].txwq_cv);
2064                         mutex_exit(&mpt->m_tx_waitq[i].txwq_mutex);
2065                 }
2066                 while (mpt->m_txwq_thread_n) {
2067                         cv_wait(&mpt->m_qthread_cv,
2068                             &mpt->m_qthread_mutex);
2069                 }
2070                 for (i = 0; i < q_thread_num; i++) {
2071                         cv_destroy(&mpt->m_tx_waitq[i].txwq_cv);
2072                         cv_destroy(&mpt->m_tx_waitq[i].txwq_drain_cv);
2073                         mutex_destroy(&mpt->m_tx_waitq[i].txwq_mutex);
2074                 }
1321                     * doneq_thread_num);
1322                 mutex_exit(&mpt->m_doneq_mutex);
1323                 cv_destroy(&mpt->m_doneq_thread_cv);
1324                 mutex_destroy(&mpt->m_doneq_mutex);
```



```
2075         }

2077         scsi_hba_reset_notify_tear_down(mpt->m_reset_notify_listf);

2079         mptsas_list_del(mpt);

2081         /*
2082          * Cancel timeout threads for this mpt
2083          */
2084         mutex_enter(&mpt->m_mutex);
2085         if (mpt->m_quiesce_timeid) {
2086                 timeout_id_t tid = mpt->m_quiesce_timeid;
2087                 mpt->m_quiesce_timeid = 0;
2088                 mutex_exit(&mpt->m_mutex);
2089                 (void) untimeout(tid);
2090                 mutex_enter(&mpt->m_mutex);
2091         }

2093         if (mpt->m_restart_cmd_timeid) {
2094                 timeout_id_t tid = mpt->m_restart_cmd_timeid;
2095                 mpt->m_restart_cmd_timeid = 0;
2096                 mutex_exit(&mpt->m_mutex);
2097                 (void) untimeout(tid);
2098                 mutex_enter(&mpt->m_mutex);
2099         }

2101         mutex_exit(&mpt->m_mutex);

2103         /*
2104          * last mpt? ... if active, CANCEL watch threads.
2105          */
2106         mutex_enter(&mptsas_global_mutex);
2107         if (mptsas_head == NULL) {
2108                 timeout_id_t tid;
2109                 /*
2110                  * Clear mptsas_timeouts_enable so that the watch thread
2111                  * gets restarted on DDI_ATTACH
2112                  */
2113                 mptsas_timeouts_enabled = 0;
2114                 if (mptsas_timeout_id) {
2115                         tid = mptsas_timeout_id;
2116                         mptsas_timeout_id = 0;
2117                         mutex_exit(&mptsas_global_mutex);
2118                         (void) untimeout(tid);
2119                         mutex_enter(&mptsas_global_mutex);
2120                 }
2121                 if (mptsas_reset_watch) {
2122                         tid = mptsas_reset_watch;
2123                         mptsas_reset_watch = 0;
2124                         mutex_exit(&mptsas_global_mutex);
2125                         (void) untimeout(tid);
2126                         mutex_enter(&mptsas_global_mutex);
2127                 }
2128         }
2129         mutex_exit(&mptsas_global_mutex);

2131         /*
2132          * Delete Phy stats
2133          */
2134         mptsas_destroy_phy_stats(mpt);

2136         mptsas_destroy_hashes(mpt);

2138         /*
2139          * Delete nt_active.
2140          */
```

```
2141         mutex_enter(&mpt->m_mutex);
2142         mptsas_free_active_slots(mpt);
2143         mutex_exit(&mpt->m_mutex);

2145         /* deallocate everything that was allocated in mptsas_attach */
2146         mptsas_cache_destroy(mpt);

2148         mptsas_hba_fini(mpt);
2149         mptsas_cfg_fini(mpt);

2151         /* Lower the power informing PM Framework */
2152         if (mpt->m_options & MPTSAS_OPT_PM) {
2153                 if (pm_lower_power(dip, 0, PM_LEVEL_D3) != DDI_SUCCESS)
2154                         mptsas_log(mpt, CE_WARN,
2155                             "!mptsas3%d: Lower power request failed "
1405                         "!mptsas%d: Lower power request failed "
2156                             "during detach, ignoring.",
2157                             mpt->m_instance);
2158         }

2160         mutex_destroy(&mpt->m_qthread_mutex);
1410         mutex_destroy(&mpt->m_tx_waitq_mutex);
2161         mutex_destroy(&mpt->m_passthru_mutex);
2162         mutex_destroy(&mpt->m_mutex);
2163         for (i = 0; i < MPTSAS_MAX_PHYS; i++) {
2164                 mutex_destroy(&mpt->m_phy_info[i].smhba_info.phy_mutex);
2165         }
2166         cv_destroy(&mpt->m_qthread_cv);
2167 #endif /* ! codereview */
2168         cv_destroy(&mpt->m_cv);
2169         cv_destroy(&mpt->m_passthru_cv);
2170         cv_destroy(&mpt->m_fw_cv);
2171         cv_destroy(&mpt->m_config_cv);
2172         cv_destroy(&mpt->m_fw_diag_cv);


2175         mptsas_smp_teardown(mpt);
2176         mptsas_hba_teardown(mpt);

2178         mptsas_config_space_fini(mpt);

2180         mptsas_free_handshake_msg(mpt);

2182         mptsas_fm_fini(mpt);
2183         ddi_soft_state_free(mptsas3_state, ddi_get_instance(dip));
1416         ddi_soft_state_free(mptsas_state, ddi_get_instance(dip));
2184         ddi_prop_remove_all(dip);

2186         return (DDI_SUCCESS);
2187 }
```

*unchanged_portion_omitted_*

```
2231 static int
2232 mptsas_alloc_handshake_msg(mptsas_t *mpt, size_t alloc_size)
2233 {
2234         ddi_dma_attr_t  task_dma_attrs;

2236         mpt->m_hshk_dma_size = 0;
2237 #endif /* ! codereview */
2238         task_dma_attrs = mpt->m_msg_dma_attr;
2239         task_dma_attrs.dma_attr_sgllen = 1;
2240         task_dma_attrs.dma_attr_granular = (uint32_t)(alloc_size);

2242         /* allocate Task Management ddi_dma resources */
2243         if (mptsas_dma_addr_create(mpt, task_dma_attrs,
2244             &mpt->m_hshk_dma_hdl, &mpt->m_hshk_acc_hdl, &mpt->m_hshk_memp,
```



```
2245             alloc_size, NULL) == FALSE) {
2246                 return (DDI_FAILURE);
2247         }
2248         mpt->m_hshk_dma_size = alloc_size;

2250         return (DDI_SUCCESS);
2251 }

2253 static void
2254 mptsas_free_handshake_msg(mptsas_t *mpt)
2255 {
2256         if (mpt->m_hshk_dma_size == 0)
2257                 return;
2258 #endif /* ! codereview */
2259         mptsas_dma_addr_destroy(&mpt->m_hshk_dma_hdl, &mpt->m_hshk_acc_hdl);
2260         mpt->m_hshk_dma_size = 0;
2261 }

2263 static int
2264 mptsas_hba_setup(mptsas_t *mpt)
2265 {
2266         scsi_hba_tran_t         *hba_tran;
2267         int                     tran_flags;

2269         /* Allocate a transport structure */
2270         hba_tran = mpt->m_tran = scsi_hba_tran_alloc(mpt->m_dip,
2271             SCSI_HBA_CANSLEEP);
2272         ASSERT(mpt->m_tran != NULL);

2274         hba_tran->tran_hba_private      = mpt;
2275         hba_tran->tran_tgt_private      = NULL;

2277         hba_tran->tran_tgt_init         = mptsas_scsi_tgt_init;
2278         hba_tran->tran_tgt_free         = mptsas_scsi_tgt_free;

2280         hba_tran->tran_start            = mptsas_scsi_start;
2281         hba_tran->tran_reset            = mptsas_scsi_reset;
2282         hba_tran->tran_abort            = mptsas_scsi_abort;
2283         hba_tran->tran_getcap           = mptsas_scsi_getcap;
2284         hba_tran->tran_setcap           = mptsas_scsi_setcap;
2285         hba_tran->tran_init_pkt         = mptsas_scsi_init_pkt;
2286         hba_tran->tran_destroy_pkt      = mptsas_scsi_destroy_pkt;

2288         hba_tran->tran_dmafree          = mptsas_scsi_dmafree;
2289         hba_tran->tran_sync_pkt         = mptsas_scsi_sync_pkt;
2290         hba_tran->tran_reset_notify     = mptsas_scsi_reset_notify;

2292         hba_tran->tran_get_bus_addr     = mptsas_get_bus_addr;
2293         hba_tran->tran_get_name         = mptsas_get_name;

2295         hba_tran->tran_quiesce          = mptsas_scsi_quiesce;
2296         hba_tran->tran_unquiesce        = mptsas_scsi_unquiesce;
2297         hba_tran->tran_bus_reset        = NULL;

2299         hba_tran->tran_add_eventcall    = NULL;
2300         hba_tran->tran_get_eventcookie  = NULL;
2301         hba_tran->tran_post_event       = NULL;
2302         hba_tran->tran_remove_eventcall = NULL;

2304         hba_tran->tran_bus_config       = mptsas_bus_config;

2306         hba_tran->tran_interconnect_type = INTERCONNECT_SAS;

2308         /*
2309          * All children of the HBA are iports. We need tran was cloned.
2310          * So we pass the flags to SCSA. SCSI_HBA_TRAN_CLONE will be
```

```
2311          * inherited to iport's tran vector.
2312          */
2313         tran_flags = (SCSI_HBA_HBA | SCSI_HBA_TRAN_CLONE);

2315         if (scsi_hba_attach_setup(mpt->m_dip, &mpt->m_msg_dma_attr,
2316             hba_tran, tran_flags) != DDI_SUCCESS) {
2317                 mptsas_log(mpt, CE_WARN, "hba attach setup failed");
2318                 scsi_hba_tran_free(hba_tran);
2319                 mpt->m_tran = NULL;
2320                 return (FALSE);
2321         }
2322         return (TRUE);
2323 }

2325 static void
2326 mptsas_hba_teardown(mptsas_t *mpt)
2327 {
2328         (void) scsi_hba_detach(mpt->m_dip);
2329         if (mpt->m_tran != NULL) {
2330                 scsi_hba_tran_free(mpt->m_tran);
2331                 mpt->m_tran = NULL;
2332         }
2333 }

2335 static void
2336 mptsas_iport_register(mptsas_t *mpt)
2337 {
2338         int i, j;
2339         mptsas_phymask_t        mask = 0x0;
2340         /*
2341          * initial value of mask is 0
2342          */
2343         mutex_enter(&mpt->m_mutex);
2344         for (i = 0; i < mpt->m_num_phys; i++) {
2345                 mptsas_phymask_t phy_mask = 0x0;
2346                 char phy_mask_name[MPTSAS_MAX_PHYS];
2347                 uint8_t current_port;

2349                 if (mpt->m_phy_info[i].attached_devhdl == 0)
2350                         continue;

2352                 bzero(phy_mask_name, sizeof (phy_mask_name));

2354                 current_port = mpt->m_phy_info[i].port_num;

2356                 if ((mask & (1 << i)) != 0)
2357                         continue;

2359                 for (j = 0; j < mpt->m_num_phys; j++) {
2360                         if (mpt->m_phy_info[j].attached_devhdl &&
2361                             (mpt->m_phy_info[j].port_num == current_port)) {
2362                                 phy_mask |= (1 << j);
2363                         }
2364                 }
2365                 mask = mask | phy_mask;

2367                 for (j = 0; j < mpt->m_num_phys; j++) {
2368                         if ((phy_mask >> j) & 0x01) {
2369                                 mpt->m_phy_info[j].phy_mask = phy_mask;
2370                         }
2371                 }

2373                 (void) sprintf(phy_mask_name, "%x", phy_mask);

2375                 mutex_exit(&mpt->m_mutex);
2376                 /*
```



```
2377                  * register a iport
2378                  */
2379                 (void) scsi_hba_iport_register(mpt->m_dip, phy_mask_name);
2380                 mutex_enter(&mpt->m_mutex);
2381         }
2382         mutex_exit(&mpt->m_mutex);
2383         /*
2384          * register a virtual port for RAID volume always
2385          */
2386         (void) scsi_hba_iport_register(mpt->m_dip, "v0");

2388 }

2390 static int
2391 mptsas_smp_setup(mptsas_t *mpt)
2392 {
2393         mpt->m_smptran = smp_hba_tran_alloc(mpt->m_dip);
2394         ASSERT(mpt->m_smptran != NULL);
2395         mpt->m_smptran->smp_tran_hba_private = mpt;
2396         mpt->m_smptran->smp_tran_start = mptsas_smp_start;
2397         if (smp_hba_attach_setup(mpt->m_dip, mpt->m_smptran) != DDI_SUCCESS) {
2398                 mptsas_log(mpt, CE_WARN, "smp attach setup failed");
2399                 smp_hba_tran_free(mpt->m_smptran);
2400                 mpt->m_smptran = NULL;
2401                 return (FALSE);
2402         }
2403         /*
2404          * Initialize smp hash table
2405          */
2406         mpt->m_smp_targets = refhash_create(MPTSAS_SMP_BUCKET_COUNT,
2407             mptsas_target_addr_hash, mptsas_target_addr_cmp,
2408             mptsas_smp_free, sizeof (mptsas_smp_t),
2409             offsetof(mptsas_smp_t, m_link), offsetof(mptsas_smp_t, m_addr),
2410             KM_SLEEP);
2411         mpt->m_smp_devhdl = 0xFFFF;

2413         return (TRUE);
2414 }

2416 static void
2417 mptsas_smp_teardown(mptsas_t *mpt)
2418 {
2419         (void) smp_hba_detach(mpt->m_dip);
2420         if (mpt->m_smptran != NULL) {
2421                 smp_hba_tran_free(mpt->m_smptran);
2422                 mpt->m_smptran = NULL;
2423         }
2424         mpt->m_smp_devhdl = 0;
2425 }

2427 static int
2428 mptsas_cache_create(mptsas_t *mpt)
2429 {
2430         int instance = mpt->m_instance;
2431         char buf[64];

2433         /*
2434          * create kmem cache for packets
2435          */
2436         (void) sprintf(buf, "mptsas3%d_cache", instance);
1469         (void) sprintf(buf, "mptsas%d_cache", instance);
2437         mpt->m_kmem_cache = kmem_cache_create(buf,
2438             sizeof (struct mptsas_cmd) + scsi_pkt_size(), 16,
1471             sizeof (struct mptsas_cmd) + scsi_pkt_size(), 8,
2439             mptsas_kmem_cache_constructor, mptsas_kmem_cache_destructor,
2440             NULL, (void *)mpt, NULL, 0);
```

```
2442         if (mpt->m_kmem_cache == NULL) {
2443                 mptsas_log(mpt, CE_WARN, "creating kmem cache failed");
2444                 return (FALSE);
2445         }

2447         /*
2448          * create kmem cache for extra SGL frames if SGL cannot
2449          * be accomodated into main request frame.
2450          */
2451         (void) sprintf(buf, "mptsas3%d_cache_frames", instance);
1484         (void) sprintf(buf, "mptsas%d_cache_frames", instance);
2452         mpt->m_cache_frames = kmem_cache_create(buf,
2453             sizeof (mptsas_cache_frames_t), 16,
1486             sizeof (mptsas_cache_frames_t), 8,
2454             mptsas_cache_frames_constructor, mptsas_cache_frames_destructor,
2455             NULL, (void *)mpt, NULL, 0);

2457         if (mpt->m_cache_frames == NULL) {
2458                 mptsas_log(mpt, CE_WARN, "creating cache for frames failed");
2459                 return (FALSE);
2460         }

2462         return (TRUE);
2463 }
______unchanged_portion_omitted_

2479 static int
2480 mptsas_power(dev_info_t *dip, int component, int level)
2481 {
2482 #ifndef __lock_lint
2483         _NOTE(ARGUNUSED(component))
2484 #endif
2485         mptsas_t        *mpt;
2486         int             rval = DDI_SUCCESS;
2487         int             polls = 0;
2488         uint32_t        ioc_status;

2490         if (scsi_hba_iport_unit_address(dip) != 0)
2491                 return (DDI_SUCCESS);

2493         mpt = ddi_get_soft_state(mptsas3_state, ddi_get_instance(dip));
1526         mpt = ddi_get_soft_state(mptsas_state, ddi_get_instance(dip));
2494         if (mpt == NULL) {
2495                 return (DDI_FAILURE);
2496         }

2498         mutex_enter(&mpt->m_mutex);

2500         /*
2501          * If the device is busy, don't lower its power level
2502          */
2503         if (mpt->m_busy && (mpt->m_power_level > level)) {
2504                 mutex_exit(&mpt->m_mutex);
2505                 return (DDI_FAILURE);
2506         }
2507         switch (level) {
2508         case PM_LEVEL_D0:
2509                 NDBG11(("mptsas3%d: turning power ON.", mpt->m_instance));
1542                 NDBG11(("mptsas%d: turning power ON.", mpt->m_instance));
2510                 MPTSAS_POWER_ON(mpt);
2511                 /*
2512                  * Wait up to 30 seconds for IOC to come out of reset.
2513                  */
2514                 while (((ioc_status = ddi_get32(mpt->m_datap,
2515                     &mpt->m_reg->Doorbell)) &
```



```
2516                     MPI2_IOC_STATE_MASK) == MPI2_IOC_STATE_RESET) {
2517                         if (polls++ > 3000) {
2518                                 break;
2519                         }
2520                         delay(drv_usectohz(10000));
2521                 }
2522                 /*
2523                  * If IOC is not in operational state, try to hard reset it.
2524                  */
2525                 if ((ioc_status & MPI2_IOC_STATE_MASK) !=
2526                     MPI2_IOC_STATE_OPERATIONAL) {
2527                         mpt->m_softstate &= ~MPTSAS_SS_MSG_UNIT_RESET;
2528                         if (mptsas_restart_ioc(mpt) == DDI_FAILURE) {
2529                                 mptsas_log(mpt, CE_WARN,
2530                                     "mptsas_power: hard reset failed");
2531                                 mutex_exit(&mpt->m_mutex);
2532                                 return (DDI_FAILURE);
2533                         }
2534                 }
2535                 mpt->m_power_level = PM_LEVEL_D0;
2536                 break;
2537         case PM_LEVEL_D3:
2538                 NDBG11(("mptsas3%d: turning power OFF.", mpt->m_instance));
1571                 NDBG11(("mptsas%d: turning power OFF.", mpt->m_instance));
2539                 MPTSAS_POWER_OFF(mpt);
2540                 break;
2541         default:
2542                 mptsas_log(mpt, CE_WARN, "mptsas3%d: unknown power level <%x>.",
1575                 mptsas_log(mpt, CE_WARN, "mptsas%d: unknown power level <%x>.",
2543                     mpt->m_instance, level);
2544                 rval = DDI_FAILURE;
2545                 break;
2546         }
2547         mutex_exit(&mpt->m_mutex);
2548         return (rval);
2549 }
______unchanged_portion_omitted_

2709 static int
2710 mptsas_alloc_request_frames(mptsas_t *mpt)
2711 {
2712         ddi_dma_attr_t          frame_dma_attrs;
2713         caddr_t                 memp;
2714         ddi_dma_cookie_t        cookie;
2715         size_t                  mem_size;

2717         /*
2718          * re-alloc when it has already alloced
2719          */
2720         if (mpt->m_dma_flags & MPTSAS_REQ_FRAME) {
2721 #endif /* ! codereview */
2722                 mptsas_dma_addr_destroy(&mpt->m_dma_req_frame_hdl,
2723                     &mpt->m_acc_req_frame_hdl);
2724                 mpt->m_dma_flags &= ~MPTSAS_REQ_FRAME;
2725         }
2726 #endif /* ! codereview */

2728         /*
2729          * The size of the request frame pool is:
2730          *   Number of Request Frames * Request Frame Size
2731          */
2732         mem_size = mpt->m_max_requests * mpt->m_req_frame_size;

2734         /*
2735          * set the DMA attributes.  System Request Message Frames must be
2736          * aligned on a 16-byte boundry.
```

```
2737          */
2738         frame_dma_attrs = mpt->m_msg_dma_attr;
2739         frame_dma_attrs.dma_attr_align = 16;
2740         frame_dma_attrs.dma_attr_sgllen = 1;

2742         /*
2743          * allocate the request frame pool.
2744          */
2745         if (mptsas_dma_addr_create(mpt, frame_dma_attrs,
2746             &mpt->m_dma_req_frame_hdl, &mpt->m_acc_req_frame_hdl, &memp,
2747             mem_size, &cookie) == FALSE) {
2748                 return (DDI_FAILURE);
2749         }

2751         /*
2752          * Store the request frame memory address.  This chip uses this
2753          * address to dma to and from the driver's frame.  The second
2754          * address is the address mpt uses to fill in the frame.
2755          */
2756         mpt->m_req_frame_dma_addr = cookie.dmac_laddress;
2757         mpt->m_req_frame = memp;

2759         /*
2760          * Clear the request frame pool.
2761          */
2762         bzero(mpt->m_req_frame, mem_size);

2764         mpt->m_dma_flags |= MPTSAS_REQ_FRAME;
2765         return (DDI_SUCCESS);
2766 }

2768 static int
2769 mptsas_alloc_sense_bufs(mptsas_t *mpt)
2770 {
2771         ddi_dma_attr_t          sense_dma_attrs;
2772         caddr_t                 memp;
2773         ddi_dma_cookie_t        cookie;
2774         size_t                  mem_size;
2775         int                     num_extrqsense_bufs;

2777         /*
2778          * re-alloc when it has already alloced
2779          */
2780         if (mpt->m_dma_flags & MPTSAS_REQ_SENSE) {
2781                 rmfreemap(mpt->m_erqsense_map);
2782                 mptsas_dma_addr_destroy(&mpt->m_dma_req_sense_hdl,
2783                     &mpt->m_acc_req_sense_hdl);
2784                 mpt->m_dma_flags &= ~MPTSAS_REQ_SENSE;
2785         }

2787         /*
2788          * The size of the request sense pool is:
2789          *   (Number of Request Frames - 2 ) * Request Sense Size +
2790          *   extra memory for extended sense requests.
2791          */
2792         mem_size = ((mpt->m_max_requests - 2) * mpt->m_req_sense_size) +
2793             mptsas_extreq_sense_bufsize;

2795         /*
2796          * set the DMA attributes.  ARQ buffers
2797          * aligned on a 16-byte boundry.
2798          */
2799         sense_dma_attrs = mpt->m_msg_dma_attr;
2800         sense_dma_attrs.dma_attr_align = 16;
2801         sense_dma_attrs.dma_attr_sgllen = 1;
```



```
2803         /*
2804          * allocate the request sense buffer pool.
2805          */
2806         if (mptsas_dma_addr_create(mpt, sense_dma_attrs,
2807             &mpt->m_dma_req_sense_hdl, &mpt->m_acc_req_sense_hdl, &memp,
2808             mem_size, &cookie) == FALSE) {
2809                 return (DDI_FAILURE);
2810         }

2812         /*
2813          * Store the request sense base memory address.  This chip uses this
2814          * address to dma the request sense data.  The second
2815          * address is the address mpt uses to access the data.
2816          * The third is the base for the extended rqsense buffers.
2817          */
2818         mpt->m_req_sense_dma_addr = cookie.dmac_laddress;
2819         mpt->m_req_sense = memp;
2820         memp += (mpt->m_max_requests - 2) * mpt->m_req_sense_size;
2821         mpt->m_extreq_sense = memp;

2823         /*
2824          * The extra memory is divided up into multiples of the base
2825          * buffer size in order to allocate via rmalloc().
2826          * Note that the rmallocmap cannot start at zero!
2827          */
2828         num_extrqsense_bufs = mptsas_extreq_sense_bufsize /
2829             mpt->m_req_sense_size;
2830         mpt->m_erqsense_map = rmallocmap_wait(num_extrqsense_bufs);
2831         rmfree(mpt->m_erqsense_map, num_extrqsense_bufs, 1);

2833         /*
2834          * Clear the pool.
2835          */
2836         bzero(mpt->m_req_sense, mem_size);

2838         mpt->m_dma_flags |= MPTSAS_REQ_SENSE;
2839 #endif /* ! codereview */
2840         return (DDI_SUCCESS);
2841 }

2843 static int
2844 mptsas_alloc_reply_frames(mptsas_t *mpt)
2845 {
2846         ddi_dma_attr_t          frame_dma_attrs;
2847         caddr_t                 memp;
2848         ddi_dma_cookie_t        cookie;
2849         size_t                  mem_size;

2851         /*
2852          * re-alloc when it has already alloced
2853          */
2854         if (mpt->m_dma_flags & MPTSAS_REPLY_FRAME) {
2855 #endif /* ! codereview */
2856                 mptsas_dma_addr_destroy(&mpt->m_dma_reply_frame_hdl,
2857                     &mpt->m_acc_reply_frame_hdl);
2858                 mpt->m_dma_flags &= ~MPTSAS_REPLY_FRAME;
2859         }
2860 #endif /* ! codereview */

2862         /*
2863          * The size of the reply frame pool is:
2864          *   Number of Reply Frames * Reply Frame Size
2865          */
2866         mem_size = mpt->m_max_replies * mpt->m_reply_frame_size;

2868         /*
```

```
2869          * set the DMA attributes.   System Reply Message Frames must be
2870          * aligned on a 4-byte boundry.  This is the default.
2871          */
2872         frame_dma_attrs = mpt->m_msg_dma_attr;
2873         frame_dma_attrs.dma_attr_sgllen = 1;

2875         /*
2876          * allocate the reply frame pool
2877          */
2878         if (mptsas_dma_addr_create(mpt, frame_dma_attrs,
2879             &mpt->m_dma_reply_frame_hdl, &mpt->m_acc_reply_frame_hdl, &memp,
2880             mem_size, &cookie) == FALSE) {
2881                 return (DDI_FAILURE);
2882         }

2884         /*
2885          * Store the reply frame memory address.  This chip uses this
2886          * address to dma to and from the driver's frame.  The second
2887          * address is the address mpt uses to process the frame.
2888          */
2889         mpt->m_reply_frame_dma_addr = cookie.dmac_laddress;
2890         mpt->m_reply_frame = memp;

2892         /*
2893          * Clear the reply frame pool.
2894          */
2895         bzero(mpt->m_reply_frame, mem_size);

2897         mpt->m_dma_flags |= MPTSAS_REPLY_FRAME;
2898 #endif /* ! codereview */
2899         return (DDI_SUCCESS);
2900 }

2902 static int
2903 mptsas_alloc_free_queue(mptsas_t *mpt)
2904 {
2905         ddi_dma_attr_t          frame_dma_attrs;
2906         caddr_t                 memp;
2907         ddi_dma_cookie_t        cookie;
2908         size_t                  mem_size;

2910         /*
2911          * re-alloc when it has already alloced
2912          */
2913         if (mpt->m_dma_flags & MPTSAS_FREE_QUEUE) {
2914 #endif /* ! codereview */
2915                 mptsas_dma_addr_destroy(&mpt->m_dma_free_queue_hdl,
2916                     &mpt->m_acc_free_queue_hdl);
2917                 mpt->m_dma_flags &= ~MPTSAS_FREE_QUEUE;
2918         }
2919 #endif /* ! codereview */

2921         /*
2922          * The reply free queue size is:
2923          *   Reply Free Queue Depth * 4
2924          * The "4" is the size of one 32 bit address (low part of 64-bit
2925          *   address)
2926          */
2927         mem_size = mpt->m_free_queue_depth * 4;

2929         /*
2930          * set the DMA attributes  The Reply Free Queue must be aligned on a
2931          * 16-byte boundry.
2932          */
2933         frame_dma_attrs = mpt->m_msg_dma_attr;
2934         frame_dma_attrs.dma_attr_align = 16;
```



```
2935         frame_dma_attrs.dma_attr_sgllen = 1;

2937         /*
2938          * allocate the reply free queue
2939          */
2940         if (mptsas_dma_addr_create(mpt, frame_dma_attrs,
2941             &mpt->m_dma_free_queue_hdl, &mpt->m_acc_free_queue_hdl, &memp,
2942             mem_size, &cookie) == FALSE) {
2943                 return (DDI_FAILURE);
2944         }

2946         /*
2947          * Store the reply free queue memory address.  This chip uses this
2948          * address to read from the reply free queue.  The second address
2949          * is the address mpt uses to manage the queue.
2950          */
2951         mpt->m_free_queue_dma_addr = cookie.dmac_laddress;
2952         mpt->m_free_queue = memp;

2954         /*
2955          * Clear the reply free queue memory.
2956          */
2957         bzero(mpt->m_free_queue, mem_size);

2959         mpt->m_dma_flags |= MPTSAS_FREE_QUEUE;
2960 #endif /* ! codereview */
2961         return (DDI_SUCCESS);
2962 }

2964 static void
2965 mptsas_free_post_queue(mptsas_t *mpt)
2966 {
2967         mptsas_reply_pqueue_t   *rpqp;
2968         int                     i;

2970         if (mpt->m_dma_flags & MPTSAS_POST_QUEUE) {
2971                 mptsas_dma_addr_destroy(&mpt->m_dma_post_queue_hdl,
2972                     &mpt->m_acc_post_queue_hdl);
2973                 rpqp = mpt->m_rep_post_queues;
2974                 for (i = 0; i < mpt->m_post_reply_qcount; i++) {
2975                         mutex_destroy(&rpqp->rpq_mutex);
2976                         rpqp++;
2977                 }
2978                 kmem_free(mpt->m_rep_post_queues,
2979                     sizeof (mptsas_reply_pqueue_t) *
2980                     mpt->m_post_reply_qcount);
2981                 mpt->m_dma_flags &= ~MPTSAS_POST_QUEUE;
2982         }
2983 }

2985 #endif /* ! codereview */
2986 static int
2987 mptsas_alloc_post_queue(mptsas_t *mpt)
2988 {
2989         ddi_dma_attr_t          frame_dma_attrs;
2990         caddr_t                 memp;
2991         ddi_dma_cookie_t        cookie;
2992         size_t                  mem_size;
2993         mptsas_reply_pqueue_t   *rpqp;
2994         int                     i;
2995 #endif /* ! codereview */

2997         /*
2998          * re-alloc when it has already alloced
2999          */
3000         mptsas_free_post_queue(mpt);
```

```
1753         mptsas_dma_addr_destroy(&mpt->m_dma_post_queue_hdl,
1754             &mpt->m_acc_post_queue_hdl);

3002         /*
3003          * The reply descriptor post queue size is:
3004          *   Reply Descriptor Post Queue Depth * 8
3005          * The "8" is the size of each descriptor (8 bytes or 64 bits).
3006          */
3007         mpt->m_post_reply_qcount = mpt->m_intr_cnt;
3008         mem_size = mpt->m_post_queue_depth * 8 * mpt->m_post_reply_qcount;
1761         mem_size = mpt->m_post_queue_depth * 8;

3010         /*
3011          * set the DMA attributes.  The Reply Descriptor Post Queue must be
3012          * aligned on a 16-byte boundry.
3013          */
3014         frame_dma_attrs = mpt->m_msg_dma_attr;
3015         frame_dma_attrs.dma_attr_align = 16;
3016         frame_dma_attrs.dma_attr_sgllen = 1;

3018         /*
3019          * Allocate the reply post queue(s).
3020          * MPI2.5 introduces a method to allocate multiple queues
3021          * using a redirect table. For now stick to one contiguous
3022          * chunck. This can get as big as 1Mbyte for 16 queues.
3023          * The spec gives no indication that the queue size can be
3024          * reduced if you have many of them.
1772          * allocate the reply post queue
3025          */
3026         if (mptsas_dma_addr_create(mpt, frame_dma_attrs,
3027             &mpt->m_dma_post_queue_hdl, &mpt->m_acc_post_queue_hdl, &memp,
3028             mem_size, &cookie) == FALSE) {
3029                 return (DDI_FAILURE);
3030         }

3032         /*
3033          * Store the reply descriptor post queue memory address.  This chip
3034          * uses this address to write to the reply descriptor post queue.  The
3035          * second address is the address mpt uses to manage the queue.
3036          */
3037         mpt->m_post_queue_dma_addr = cookie.dmac_laddress;
3038         mpt->m_post_queue = memp;

3040         mpt->m_rep_post_queues = kmem_zalloc(sizeof (mptsas_reply_pqueue_t) *
3041             mpt->m_post_reply_qcount, KM_SLEEP);
3042         rpqp = mpt->m_rep_post_queues;
3043         for (i = 0; i < mpt->m_post_reply_qcount; i++) {
3044                 rpqp->rpq_queue = memp;
3045                 mutex_init(&rpqp->rpq_mutex, NULL, MUTEX_DRIVER, NULL);
3046                 rpqp->rpq_dlist.dl_tail = &rpqp->rpq_dlist.dl_q;
3047                 rpqp->rpq_num = (uint8_t)i;
3048                 memp += (mpt->m_post_queue_depth * 8);
3049                 rpqp++;
3050         }

3052 #endif /* ! codereview */
3053         /*
3054          * Clear the reply post queue memory.
3055          */
3056         bzero(mpt->m_post_queue, mem_size);

3058         mpt->m_dma_flags |= MPTSAS_POST_QUEUE;
3059 #endif /* ! codereview */
3060         return (DDI_SUCCESS);
3061 }
```



```
3063 static void
3064 mptsas_alloc_reply_args(mptsas_t *mpt)
3065 {
3066         if (mpt->m_replyh_args == NULL) {
3067                 mpt->m_replyh_args = kmem_zalloc(sizeof (m_replyh_arg_t) *
3068                     mpt->m_max_replies, KM_SLEEP);
3069         } else {
3070                 bzero(mpt->m_replyh_args, sizeof (m_replyh_arg_t) *
3071                     mpt->m_max_replies);
3072 #endif /* ! codereview */
3073         }
3074 }

3076 static int
3077 mptsas_alloc_extra_sgl_frame(mptsas_t *mpt, mptsas_cmd_t *cmd)
3078 {
3079         mptsas_cache_frames_t   *frames = NULL;
3080         if (cmd->cmd_extra_frames == NULL) {
3081                 frames = kmem_cache_alloc(mpt->m_cache_frames, KM_NOSLEEP);
3082                 if (frames == NULL) {
3083                         return (DDI_FAILURE);
3084                 }
3085                 cmd->cmd_extra_frames = frames;
3086         }
3087         return (DDI_SUCCESS);
3088 }

3090 static void
3091 mptsas_free_extra_sgl_frame(mptsas_t *mpt, mptsas_cmd_t *cmd)
3092 {
3093         if (cmd->cmd_extra_frames) {
3094                 kmem_cache_free(mpt->m_cache_frames,
3095                     (void *)cmd->cmd_extra_frames);
3096                 cmd->cmd_extra_frames = NULL;
3097         }
3098 }

3100 static void
3101 mptsas_cfg_fini(mptsas_t *mpt)
3102 {
3103         NDBG0(("mptsas_cfg_fini"));
3104         ddi_regs_map_free(&mpt->m_datap);
3105 }

3107 static void
3108 mptsas_hba_fini(mptsas_t *mpt)
3109 {
3110         NDBG0(("mptsas_hba_fini"));

3112         /*
3113          * Free up any allocated memory
3114          */
3115         if (mpt->m_dma_flags & MPTSAS_REQ_FRAME) {
3116 #endif /* ! codereview */
3117                 mptsas_dma_addr_destroy(&mpt->m_dma_req_frame_hdl,
3118                     &mpt->m_acc_req_frame_hdl);
3119         }

3121         if (mpt->m_dma_flags & MPTSAS_REQ_SENSE) {
3122                 rmfreemap(mpt->m_erqsense_map);
3123                 mptsas_dma_addr_destroy(&mpt->m_dma_req_sense_hdl,
3124                     &mpt->m_acc_req_sense_hdl);
3125         }
3126 #endif /* ! codereview */

3128         if (mpt->m_dma_flags & MPTSAS_REPLY_FRAME) {
```

```
3129 #endif /* ! codereview */
3130                 mptsas_dma_addr_destroy(&mpt->m_dma_reply_frame_hdl,
3131                     &mpt->m_acc_reply_frame_hdl);
3132         }
3133 #endif /* ! codereview */

3135         if (mpt->m_dma_flags & MPTSAS_FREE_QUEUE) {
3136 #endif /* ! codereview */
3137                 mptsas_dma_addr_destroy(&mpt->m_dma_free_queue_hdl,
3138                     &mpt->m_acc_free_queue_hdl);
3139         }
3140 #endif /* ! codereview */

3142         mptsas_free_post_queue(mpt);
1788         mptsas_dma_addr_destroy(&mpt->m_dma_post_queue_hdl,
1789             &mpt->m_acc_post_queue_hdl);

3144         if (mpt->m_replyh_args != NULL) {
3145                 kmem_free(mpt->m_replyh_args, sizeof (m_replyh_arg_t)
3146                     * mpt->m_max_replies);
3147         }
3148 }
______unchanged_portion_omitted_

3400 /*
3401  * scsi_pkt handling
3402  *
3403  * Visible to the external world via the transport structure.
3404  */

3406 /*
3407  * Notes:
3408  *      - transport the command to the addressed SCSI target/lun device
3409  *      - normal operation is to schedule the command to be transported,
3410  *        and return TRAN_ACCEPT if this is successful.
3411  *      - if NO_INTR, tran_start must poll device for command completion
3412  */
3413 static int
3414 mptsas_scsi_start(struct scsi_address *ap, struct scsi_pkt *pkt)
3415 {
3416 #ifndef __lock_lint
3417         _NOTE(ARGUNUSED(ap))
3418 #endif
3419         mptsas_t        *mpt = PKT2MPT(pkt);
3420         mptsas_cmd_t    *cmd = PKT2CMD(pkt);
3421         int             rval, start;
3422         uint8_t         pref;
2068         int             rval;
3423         mptsas_target_t *ptgt = cmd->cmd_tgt_addr;
3424         mptsas_tx_waitqueue_t *txwq;
3425 #endif /* ! codereview */

3427         NDBG1(("mptsas_scsi_start: pkt=0x%p", (void *)pkt));
3428         ASSERT(ptgt);
3429         if (ptgt == NULL)
3430                 return (TRAN_FATAL_ERROR);

3432         /*
3433          * prepare the pkt before taking mutex.
3434          */
3435         rval = mptsas_prepare_pkt(cmd);
3436         if (rval != TRAN_ACCEPT) {
3437                 return (rval);
3438         }

3440         /*
```



```
3441          * Send the command to target/lun, however your HBA requires it.
3442          * If busy, return TRAN_BUSY; if there's some other formatting error
3443          * in the packet, return TRAN_BADPKT; otherwise, fall through to the
3444          * return of TRAN_ACCEPT.
3445          *
3446          * Remember that access to shared resources, including the mptsas_t
3447          * data structure and the HBA hardware registers, must be protected
3448          * with mutexes, here and everywhere.
3449          *
3450          * Also remember that at interrupt time, you'll get an argument
3451          * to the interrupt handler which is a pointer to your mptsas_t
3452          * structure; you'll have to remember which commands are outstanding
3453          * and which scsi_pkt is the currently-running command so the
3454          * interrupt handler can refer to the pkt to set completion
3455          * status, call the target driver back through pkt_comp, etc.
3456          *
3457          * If the instance lock is held by other thread, don't spin to wait
3458          * for it. Instead, queue the cmd and next time when the instance lock
3459          * is not held, accept all the queued cmd. A extra tx_waitq is
3460          * introduced to protect the queue.
3461          *
3462          * The polled cmd will not be queud and accepted as usual.
3463          *
3464          * Under the tx_waitq mutex, record whether a thread is draining
3465          * the tx_waitq.  An IO requesting thread that finds the instance
3466          * mutex contended appends to the tx_waitq and while holding the
3467          * tx_wait mutex, if the draining flag is not set, sets it and then
3468          * proceeds to spin for the instance mutex. This scheme ensures that
3469          * the last cmd in a burst be processed.
3470          *
3471          * we enable this feature only when the helper threads are enabled,
3472          * at which we think the loads are heavy.
3473          *
3474          * per instance, per queue mutex m_tx_waitq[i].txwq_mutex is
3475          * introduced to protect the txwq_qtail, txwq_cmdq, txwq_len
2070          * per instance mutex m_tx_waitq_mutex is introduced to protect the
2071          * m_tx_waitqtail, m_tx_waitq, m_tx_draining.
3476          */

3478         if (mpt->m_txwq_enabled == TRUE) {
3479                 int gotmtx = 0;

3481                 if (mpt->m_txwq_allow_q_jumping) {
3482                         gotmtx = mutex_tryenter(&mpt->m_mutex);
3483                 }
3484                 if (gotmtx == 0) {
3485                         /* We didn't get the mutex or didn't try */
3486                         if (cmd->cmd_pkt_flags & FLAG_NOINTR) {
2074         if (mpt->m_doneq_thread_n) {
2075                 if (mutex_tryenter(&mpt->m_mutex) != 0) {
2076                         rval = mptsas_accept_txwq_and_pkt(mpt, cmd);
2077                         mutex_exit(&mpt->m_mutex);
2078                 } else if (cmd->cmd_pkt_flags & FLAG_NOINTR) {
3487                                 mutex_enter(&mpt->m_mutex);
3488                                 /* Polled commands queue jump */
3489                                 mptsas_accept_tx_waitqs(mpt);
2080                         rval = mptsas_accept_txwq_and_pkt(mpt, cmd);
2081                         mutex_exit(&mpt->m_mutex);
3490                         } else {
3491                                 rval = mptsas_check_targ_intxtion(
3492                                     cmd->cmd_tgt_addr,
3493                                     cmd->cmd_pkt_flags);
3494                                 if (rval != TRAN_ACCEPT) {
3495                                         return (rval);
3496                                 }
```

```
3498                                    cmd->cmd_flags |= CFLAG_TXQ;
3499                                    pref = mpt->m_pref_tx_waitq;
3500                                    txwq = &mpt->m_tx_waitq[pref];

3502                                    if (mutex_tryenter(&txwq->txwq_mutex) == 0) {
3503                                            txwq = &mpt->m_tx_waitq[pref^1];
3504                                            mutex_enter(&txwq->txwq_mutex);
2083                            mutex_enter(&mpt->m_tx_waitq_mutex);
2084                            /*
2085                             * ptgt->m_dr_flag is protected by m_mutex or
2086                             * m_tx_waitq_mutex. In this case, m_tx_waitq_mutex
2087                             * is acquired.
2088                             */
2089                            if (ptgt->m_dr_flag == MPTSAS_DR_INTRANSITION) {
2090                                    if (cmd->cmd_pkt_flags & FLAG_NOQUEUE) {
2091                                            /*
2092                                             * The command should be allowed to
2093                                             * retry by returning TRAN_BUSY to
2094                                             * to stall the I/O's which come from
2095                                             * scsi_vhci since the device/path is
2096                                             * in unstable state now.
2097                                             */
2098                                            mutex_exit(&mpt->m_tx_waitq_mutex);
2099                                            return (TRAN_BUSY);
3505                                    } else {
3506                                            pref ^= 1;
3507                                            mpt->m_pref_tx_waitq = pref;
2101                                            /*
2102                                             * The device is offline, just fail the
2103                                             * command by returning
2104                                             * TRAN_FATAL_ERROR.
2105                                             */
2106                                            mutex_exit(&mpt->m_tx_waitq_mutex);
2107                                            return (TRAN_FATAL_ERROR);
3508                                    }

3510                                    *txwq->txwq_qtail = cmd;
3511                                    txwq->txwq_qtail = &cmd->cmd_linkp;
3512                                    txwq->txwq_len++;
3513                                    if (!txwq->txwq_draining) {
3514                                            cv_signal(&txwq->txwq_cv);
3515 #endif /* ! codereview */
3516                                    }
3517                                    mutex_exit(&txwq->txwq_mutex);
3518                                    return (rval);
2109                            if (mpt->m_tx_draining) {
2110                                    cmd->cmd_flags |= CFLAG_TXQ;
2111                                    *mpt->m_tx_waitqtail = cmd;
2112                                    mpt->m_tx_waitqtail = &cmd->cmd_linkp;
2113                                    mutex_exit(&mpt->m_tx_waitq_mutex);
2114                            } else { /* drain the queue */
2115                                    mpt->m_tx_draining = 1;
2116                                    mutex_exit(&mpt->m_tx_waitq_mutex);
2117                                    mutex_enter(&mpt->m_mutex);
2118                                    rval = mptsas_accept_txwq_and_pkt(mpt, cmd);
2119                                    mutex_exit(&mpt->m_mutex);
3519                            }
3520                    }
3521            } else {
3522                    mutex_enter(&mpt->m_mutex);
3523            }
3524            rval = mptsas_check_targ_intxtion(cmd->cmd_tgt_addr,
3525                cmd->cmd_pkt_flags);
3526            if (rval != TRAN_ACCEPT) {
2124                    /*
2125                     * ptgt->m_dr_flag is protected by m_mutex or m_tx_waitq_mutex
```



```
2126                     * in this case, m_mutex is acquired.
2127                     */
2128                    if (ptgt->m_dr_flag == MPTSAS_DR_INTRANSITION) {
2129                            if (cmd->cmd_pkt_flags & FLAG_NOQUEUE) {
2130                                    /*
2131                                     * commands should be allowed to retry by
2132                                     * returning TRAN_BUSY to stall the I/O's
2133                                     * which come from scsi_vhci since the device/
2134                                     * path is in unstable state now.
2135                                     */
3527                    mutex_exit(&mpt->m_mutex);
3528                    return (rval);
2137                                    return (TRAN_BUSY);
2138                            } else {
2139                                    /*
2140                                     * The device is offline, just fail the
2141                                     * command by returning TRAN_FATAL_ERROR.
2142                                     */
2143                                    mutex_exit(&mpt->m_mutex);
2144                                    return (TRAN_FATAL_ERROR);
3529            }

3531            start = mptsas_accept_pkt(mpt, cmd, &rval);
2146                    }
2147                    rval = mptsas_accept_pkt(mpt, cmd);
3532            mutex_exit(&mpt->m_mutex);
3533            if (start) {
3534                    (void) mptsas_start_cmd(mpt, cmd);
3535 #endif /* ! codereview */
3536            }

3538            return (rval);
3539 }

2149 /*
2150  * Accept all the queued cmds(if any) before accept the current one.
2151  */
3541 static int
3542 mptsas_check_targ_intxtion(mptsas_target_t *ptgt, int cmd_pkt_flags)
2153 mptsas_accept_txwq_and_pkt(mptsas_t *mpt, mptsas_cmd_t *cmd)
3543 {
2155            int rval;
2156            mptsas_target_t *ptgt = cmd->cmd_tgt_addr;

2158            ASSERT(mutex_owned(&mpt->m_mutex));
3544            /*
3545             * ptgt->m_dr_flag is a variable that is only ever changed by
3546             * direct write under the main m_mutex.
3547             * It doesn't need a mutex hold to protect this read.
2160             * The call to mptsas_accept_tx_waitq() must always be performed
2161             * because that is where mpt->m_tx_draining is cleared.
2162             */
2163            mutex_enter(&mpt->m_tx_waitq_mutex);
2164            mptsas_accept_tx_waitq(mpt);
2165            mutex_exit(&mpt->m_tx_waitq_mutex);
2166            /*
2167             * ptgt->m_dr_flag is protected by m_mutex or m_tx_waitq_mutex
2168             * in this case, m_mutex is acquired.
3548             */

3550 #endif /* ! codereview */
3551            if (ptgt->m_dr_flag == MPTSAS_DR_INTRANSITION) {
3552                    if (cmd_pkt_flags & FLAG_NOQUEUE) {
2170                    if (cmd->cmd_pkt_flags & FLAG_NOQUEUE) {
3553                            /*
3554                             * The command should be allowed to retry by returning
```

```
3555                         * TRAN_BUSY to stall the I/O's which come from
3556                         * scsi_vhci since the device/path is in unstable state
3557                         * now.
3558                         */
3559                        return (TRAN_BUSY);
3560                } else {
3561                        /*
3562                         * The device is offline, just fail the command by
3563                         * return TRAN_FATAL_ERROR.
3564                         */
3565                        return (TRAN_FATAL_ERROR);
3566                }
3567        }
3568        return (TRAN_ACCEPT);
3569 }

3571 /*
3572  * Note that this function has a side effect of releasing the
3573  * per target mutex.
3574  */
3575 static void
3576 mptsas_offline_target_direct(mptsas_t *mpt, mptsas_target_t *ptgt)
3577 {
3578        char                            phy_mask_name[MPTSAS_MAX_PHYS];
3579        mptsas_phymask_t                phymask = ptgt->m_addr.mta_phymask;
3580        dev_info_t                      *parent;

3582        ASSERT(mutex_owned(&mpt->m_mutex));

3584        ptgt->m_dr_flag = MPTSAS_DR_INTRANSITION;
3585        bzero(phy_mask_name, MPTSAS_MAX_PHYS);
3586        (void) sprintf(phy_mask_name, "%x", phymask);
3587        parent = scsi_hba_iport_find(mpt->m_dip, phy_mask_name);
2186        rval = mptsas_accept_pkt(mpt, cmd);

3589        if (parent != NULL) {
3590                mptsas_offline_target(mpt, ptgt,
3591                    ptgt->m_deviceinfo & DEVINFO_DIRECT_ATTACHED ?
3592                    MPTSAS_TOPO_FLAG_DIRECT_ATTACHED_DEVICE :
3593                    MPTSAS_TOPO_FLAG_EXPANDER_ATTACHED_DEVICE,
3594                    parent);
3595        } else {
3596                mptsas_log(mpt, CE_WARN, "Failed to find an "
3597                    "iport for \"%s\", should not happen!",
3598                    phy_mask_name);
3599        }
2188        return (rval);
3600 }

3602 /*
3603  * In order to be efficient with the m_mutex (which can be dropped before
3604  * calling mptsas_start_cmd()) indicate if start_cmd should be called via the
3605  * returned value (FALSE or TRUE). Caller is then responsible for doing the
3606  * right thing with the m_mutex.
3607  */
3608 #endif /* ! codereview */
3609 static int
3610 mptsas_accept_pkt(mptsas_t *mpt, mptsas_cmd_t *cmd, int *tran_rval)
2191 mptsas_accept_pkt(mptsas_t *mpt, mptsas_cmd_t *cmd)
3611 {
3612        int             rval = TRAN_ACCEPT;
3613        mptsas_target_t *ptgt = cmd->cmd_tgt_addr;

3615        NDBG1(("mptsas_accept_pkt: cmd=0x%p", (void *)cmd));

3617        ASSERT(mutex_owned(&mpt->m_mutex));
```



```
3619        if ((cmd->cmd_flags & CFLAG_PREPARED) == 0) {
3620                rval = mptsas_prepare_pkt(cmd);
3621                if (rval != TRAN_ACCEPT) {
3622                        cmd->cmd_flags &= ~CFLAG_TRANFLAG;
3623                        goto set_tranrval;
2204                        return (rval);
3624                }
3625        }

3627        /*
3628         * If the command came from the tx wait q it may have slipped
3629         * by the check for dr_flag before being added to the queue.
3630         * Fail here with abort status.
2209         * reset the throttle if we were draining
3631         */
3632        if (cmd->cmd_flags & CFLAG_TXQ) {
3633                rval = mptsas_check_targ_intxtion(cmd->cmd_tgt_addr,
3634                    cmd->cmd_pkt_flags);
3635                if (rval != TRAN_ACCEPT) {
3636                        mptsas_set_pkt_reason(mpt, cmd, CMD_ABORTED,
3637                            STAT_ABORTED);
3638                        mptsas_doneq_add(mpt, cmd);
3639                        mptsas_doneq_empty(mpt);
3640                        goto set_tranrval;
3641                }
2211        if ((ptgt->m_t_ncmds == 0) &&
2212            (ptgt->m_t_throttle == DRAIN_THROTTLE)) {
2213                NDBG23(("reset throttle"));
2214                ASSERT(ptgt->m_reset_delay == 0);
2215                mptsas_set_throttle(mpt, ptgt, MAX_THROTTLE);
3642        }

3643        /*
3644         * If HBA is being reset, the DevHandles are being re-initialized,
3645         * which means that they could be invalid even if the target is still
3646         * attached.  Check if being reset and if DevHandle is being
3647         * re-initialized.  If this is the case, return BUSY so the I/O can be
3648         * retried later.
3649         */
3650        if ((ptgt->m_devhdl == MPTSAS_INVALID_DEVHDL) && mpt->m_in_reset) {
3651                mptsas_set_pkt_reason(mpt, cmd, CMD_RESET, STAT_BUS_RESET);
3652                if (cmd->cmd_flags & CFLAG_TXQ) {
3653                        mptsas_doneq_add(mpt, cmd);
3654                        mptsas_doneq_empty(mpt);
2230                        return (rval);
3655                } else {
3656                        rval = TRAN_BUSY;
2232                        return (TRAN_BUSY);
3657                }
3658                goto set_tranrval;
3659        }

3661        mutex_enter(&ptgt->m_t_mutex);
3662        /*
3663         * reset the throttle if we were draining
3664         */
3665        if ((ptgt->m_t_ncmds == 0) &&
3666            (ptgt->m_t_throttle == DRAIN_THROTTLE)) {
3667                NDBG23(("reset throttle"));
3668                ASSERT(ptgt->m_reset_delay == 0);
3669                mptsas_set_throttle(mpt, ptgt, MAX_THROTTLE);
3670 #endif /* ! codereview */
3671        }

3673        /*
```

```
3674          * If device handle has already been invalidated, just
3675          * fail the command. In theory, for a command from scsi_vhci
3676          * client it's impossible to receive a command with an invalid
2234          * fail the command. In theory, command from scsi_vhci
2235          * client is impossible send down command with invalid
3677          * devhdl since devhdl is set after path offline, target
3678          * driver is not supposed to select an offlined path.
2237          * driver is not suppose to select a offlined path.
3679          */
3680         if (ptgt->m_devhdl == MPTSAS_INVALID_DEVHDL) {
3681                 NDBG3(("rejecting command, it might because invalid devhdl "
2240                 NDBG20(("rejecting command, it might because invalid devhdl "
3682                     "request."));
3683                 mutex_exit(&ptgt->m_t_mutex);
3684 #endif /* ! codereview */
3685                 mptsas_set_pkt_reason(mpt, cmd, CMD_DEV_GONE, STAT_TERMINATED);
3686                 if (cmd->cmd_flags & CFLAG_TXQ) {
3687                         mptsas_doneq_add(mpt, cmd);
3688                         mptsas_doneq_empty(mpt);
2242                         return (rval);
3689                 } else {
3690                         rval = TRAN_FATAL_ERROR;
2244                         return (TRAN_FATAL_ERROR);
3691                 }
3692                 goto set_tranrval;
3693 #endif /* ! codereview */
3694         }
3695         /*
3696          * The first case is the normal case.  mpt gets a command from the
3697          * target driver and starts it.
3698          * Since SMID 0 is reserved and the TM slot is reserved, the actual max
3699          * commands is m_max_requests - 2.
3700          */
3701         if ((mpt->m_ncmds <= (mpt->m_max_requests - 2)) &&
3702             (ptgt->m_t_throttle > HOLD_THROTTLE) &&
3703             (ptgt->m_t_ncmds < ptgt->m_t_throttle) &&
3704             (ptgt->m_reset_delay == 0) && (mpt->m_polled_intr == 0) &&
2246             (ptgt->m_reset_delay == 0) &&
3705             (ptgt->m_t_nwait == 0) &&
3706             ((cmd->cmd_pkt_flags & FLAG_NOINTR) == 0)) {
3707                 ASSERT((cmd->cmd_flags & CFLAG_CMDIOC) == 0);
3708                 if (mptsas_save_cmd_to_slot(mpt, cmd) == TRUE) {
3709                         ptgt->m_t_ncmds++;
3710                         mutex_exit(&ptgt->m_t_mutex);
3711                         cmd->cmd_active_expiration = 0;
3712                         *tran_rval = rval;
3713                         return (TRUE);
2249                 if (mptsas_save_cmd(mpt, cmd) == TRUE) {
2250                         (void) mptsas_start_cmd(mpt, cmd);
3714                 } else {
3715                         mutex_exit(&ptgt->m_t_mutex);
3716 #endif /* ! codereview */
3717                         mptsas_waitq_add(mpt, cmd);
3718                 }
3719         } else {
3720                 mutex_exit(&ptgt->m_t_mutex);
3721 #endif /* ! codereview */
3722                 /*
3723                  * Add this pkt to the work queue
3724                  */
3725                 mptsas_waitq_add(mpt, cmd);

3727                 if (cmd->cmd_pkt_flags & FLAG_NOINTR) {
3728                         (void) mptsas_poll(mpt, cmd, MPTSAS_POLL_TIME);

3730                         /*
```



```
3731                          * Only flush the doneq if this is not a TM
3732                          * cmd.  For TM cmds the flushing of the
3733                          * doneq will be done in those routines.
3734                          */
3735                         if ((cmd->cmd_flags & CFLAG_TM_CMD) == 0) {
3736                                 mptsas_doneq_empty(mpt);
3737                         }
3738                 }
3739         }
3740 set_tranrval:
3741         *tran_rval = rval;
3742         return (FALSE);
3743 }

3745 static void
3746 mptsas_retry_pkt(mptsas_t *mpt, mptsas_cmd_t *cmd)
3747 {
3748         int             rval;

3750         cmd->cmd_pkt_flags |= FLAG_HEAD;
3751         cmd->cmd_flags |= CFLAG_RETRY;
3752         cmd->cmd_flags &= ~CFLAG_TXQ;
3753         if (mptsas_accept_pkt(mpt, cmd, &rval)) {
3754                 (void) mptsas_start_cmd(mpt, cmd);
3755         }

3757         /*
3758          * If there was a problem clear the retry flag so that the
3759          * command will be completed with error rather than get lost!
3760          */
3761         if (rval != TRAN_ACCEPT)
3762                 cmd->cmd_flags &= ~CFLAG_RETRY;
2252         return (rval);
3763 }

3765 static int
3766 mptsas_save_cmd_to_slot(mptsas_t *mpt, mptsas_cmd_t *cmd)
2255 int
2256 mptsas_save_cmd(mptsas_t *mpt, mptsas_cmd_t *cmd)
3767 {
3768         mptsas_slots_t *slots = mpt->m_active;
3769         uint_t slot, start_rotor, rotor, n_normal;
2259         uint_t slot, start_rotor;
2260         mptsas_target_t *ptgt = cmd->cmd_tgt_addr;

2262         ASSERT(MUTEX_HELD(&mpt->m_mutex));

3771         /*
3772          * Account for reserved TM request slot and reserved SMID of 0.
3773          */
3774         ASSERT(slots->m_n_normal == (mpt->m_max_requests - 2));

3776         /*
3777          * Find the next available slot, beginning at m_rotor.  If no slot is
3778          * available, we'll return FALSE to indicate that.  This mechanism
3779          * considers only the normal slots, not the reserved slot 0 nor the
3780          * task management slot m_n_normal + 1.  The rotor is left to point to
3781          * the normal slot after the one we select, unless we select the last
3782          * normal slot in which case it returns to slot 1.
3783          */
3784         start_rotor = rotor = slots->m_rotor;
3785         n_normal = slots->m_n_normal;
2277         start_rotor = slots->m_rotor;
3786         do {
3787                 slot = rotor++;
3788                 if (rotor > n_normal)
```

```
3789                        rotor = 1;
2279                slot = slots->m_rotor++;
2280                if (slots->m_rotor > slots->m_n_normal)
2281                        slots->m_rotor = 1;

3791                if (rotor == start_rotor)
2283                if (slots->m_rotor == start_rotor)
3792                        break;
3793        } while (slots->m_slot[slot] != NULL);
3794        slots->m_rotor = rotor;
3795 #endif /* ! codereview */

3797        if (slots->m_slot[slot] != NULL)
3798                return (FALSE);

3800        ASSERT(slot != 0 && slot <= slots->m_n_normal);

3802        cmd->cmd_slot = slot;
3803        slots->m_slot[slot] = cmd;
3804        atomic_inc_32(&mpt->m_ncmds);

3806        /*
3807         * Distribute the commands amongst the reply queues (Interrupt vectors).
3808         * Stick to 0 for polled.
3809         */
3810        if (!(cmd->cmd_pkt_flags & FLAG_NOINTR) &&
3811            !(cmd->cmd_flags & (CFLAG_PASSTHRU|CFLAG_CONFIG|CFLAG_FW_DIAG)) &&
3812            (mpt->m_post_reply_qcount > 1)) {
3813                cmd->cmd_rpqidx = slot % mpt->m_post_reply_qcount;
3814        }
3815        atomic_inc_32(&mpt->m_rep_post_queues[cmd->cmd_rpqidx].rpq_ncmds);
3816        return (TRUE);
3817 }

3819 int
3820 mptsas_save_cmd(mptsas_t *mpt, mptsas_cmd_t *cmd)
3821 {
3822        mptsas_target_t *ptgt = cmd->cmd_tgt_addr;

3824        ASSERT(MUTEX_HELD(&mpt->m_mutex));

3826        if (!mptsas_save_cmd_to_slot(mpt, cmd)) {
3827                return (FALSE);
3828        }
2286        mpt->m_ncmds++;

3830        /*
3831         * only increment per target ncmds if this is not a
3832         * command that has no target associated with it (i.e. a
3833         * event acknoledgement)
2291         * event acknoledgment)
3834         */
3835        if ((cmd->cmd_flags & CFLAG_CMDIOC) == 0) {
3836                /*
3837                 * Expiration time is set in mptsas_start_cmd
3838                 */
3839                mutex_enter(&ptgt->m_t_mutex);
3840 #endif /* ! codereview */
3841                ptgt->m_t_ncmds++;
3842                mutex_exit(&ptgt->m_t_mutex);
3843                cmd->cmd_active_expiration = 0;
3844        } else {
2294        }
2295        cmd->cmd_active_timeout = cmd->cmd_pkt->pkt_time;

3845                /*
```



```
3846                 * Initialize expiration time for passthrough commands,
2298         * If initial timout is less than or equal to one tick, bump
2299         * the timeout by a tick so that command doesn't timeout before
2300         * its allotted time.
3847                 */
3848                cmd->cmd_active_expiration = gethrtime() +
3849                    (hrtime_t)cmd->cmd_pkt->pkt_time * NANOSEC;
2302        if (cmd->cmd_active_timeout <= mptsas_scsi_watchdog_tick) {
2303                cmd->cmd_active_timeout += mptsas_scsi_watchdog_tick;
3850        }
3851        return (TRUE);
3852 }
______unchanged_portion_omitted_

3903 /*
3904  * tran_init_pkt(9E) - allocate scsi_pkt(9S) for command
3905  *
3906  * One of three possibilities:
3907  *      - allocate scsi_pkt
3908  *      - allocate scsi_pkt and DMA resources
3909  *      - allocate DMA resources to an already-allocated pkt
3910  */
3911 static struct scsi_pkt *
3912 mptsas_scsi_init_pkt(struct scsi_address *ap, struct scsi_pkt *pkt,
3913     struct buf *bp, int cmdlen, int statuslen, int tgtlen, int flags,
3914     int (*callback)(), caddr_t arg)
3915 {
3916        mptsas_cmd_t            *cmd, *new_cmd;
3917        mptsas_t                *mpt = ADDR2MPT(ap);
3918        int                     failure = 1;
3919        uint_t                  oldcookiec;
3920        mptsas_target_t         *ptgt = NULL;
3921        int                     rval;
3922        mptsas_tgt_private_t    *tgt_private;
3923        int                     kf;

3925        kf = (callback == SLEEP_FUNC)? KM_SLEEP: KM_NOSLEEP;

3927        tgt_private = (mptsas_tgt_private_t *)ap->a_hba_tran->
3928            tran_tgt_private;
3929        ASSERT(tgt_private != NULL);
3930        if (tgt_private == NULL) {
3931                return (NULL);
3932        }
3933        ptgt = tgt_private->t_private;
3934        ASSERT(ptgt != NULL);
3935        if (ptgt == NULL)
3936                return (NULL);
3937        ap->a_target = ptgt->m_devhdl;
3938        ap->a_lun = tgt_private->t_lun;

3940        ASSERT(callback == NULL_FUNC || callback == SLEEP_FUNC);
3941 #ifdef MPTSAS_TEST_EXTRN_ALLOC
3942        statuslen *= 100; tgtlen *= 4;
3943 #endif
3944        NDBG3(("mptsas_scsi_init_pkt:\n"
3945            "\ttgt=%d in=0x%p bp=0x%p clen=%d slen=%d tlen=%d flags=%x",
3946            ap->a_target, (void *)pkt, (void *)bp,
3947            cmdlen, statuslen, tgtlen, flags));

3949        /*
3950         * Allocate the new packet.
3951         */
3952        if (pkt == NULL) {
3953                ddi_dma_handle_t        save_dma_handle;
2408                ddi_dma_handle_t        save_arq_dma_handle;
```

```
2409                struct buf              *save_arq_bp;
2410                ddi_dma_cookie_t        save_arqcookie;

3955                cmd = kmem_cache_alloc(mpt->m_kmem_cache, kf);

3957                if (cmd) {
3958                        save_dma_handle = cmd->cmd_dmahandle;
2416                        save_arq_dma_handle = cmd->cmd_arqhandle;
2417                        save_arq_bp = cmd->cmd_arq_buf;
2418                        save_arqcookie = cmd->cmd_arqcookie;
3959                        bzero(cmd, sizeof (*cmd) + scsi_pkt_size());
3960                        cmd->cmd_dmahandle = save_dma_handle;
2421                        cmd->cmd_arqhandle = save_arq_dma_handle;
2422                        cmd->cmd_arq_buf = save_arq_bp;
2423                        cmd->cmd_arqcookie = save_arqcookie;

3962                        pkt = (void *)((uchar_t *)cmd +
3963                            sizeof (struct mptsas_cmd));
3964                        pkt->pkt_ha_private = (opaque_t)cmd;
3965                        pkt->pkt_address = *ap;
3966                        pkt->pkt_private = (opaque_t)cmd->cmd_pkt_private;
3967                        pkt->pkt_scbp = (opaque_t)&cmd->cmd_scb;
3968                        pkt->pkt_cdbp = (opaque_t)&cmd->cmd_cdb;
3969                        cmd->cmd_pkt = (struct scsi_pkt *)pkt;
3970                        cmd->cmd_cdblen = (uchar_t)cmdlen;
3971                        cmd->cmd_scblen = statuslen;
3972                        cmd->cmd_rqslen = SENSE_LENGTH;
3973                        cmd->cmd_tgt_addr = ptgt;
3974                        failure = 0;
3975                }

3977                if (failure || (cmdlen > sizeof (cmd->cmd_cdb)) ||
3978                    (tgtlen > PKT_PRIV_LEN) ||
3979                    (statuslen > EXTCMDS_STATUS_SIZE)) {
3980                        if (failure == 0) {
3981                                /*
3982                                 * if extern alloc fails, all will be
3983                                 * deallocated, including cmd
3984                                 */
3985                                failure = mptsas_pkt_alloc_extern(mpt, cmd,
3986                                    cmdlen, tgtlen, statuslen, kf);
3987                        }
3988                        if (failure) {
3989                                /*
3990                                 * if extern allocation fails, it will
3991                                 * deallocate the new pkt as well
3992                                 */
3993                                return (NULL);
3994                        }
3995                }
3996                new_cmd = cmd;

3998        } else {
3999                cmd = PKT2CMD(pkt);
4000                new_cmd = NULL;
4001        }


4004        /* grab cmd->cmd_cookiec here as oldcookiec */

4006        oldcookiec = cmd->cmd_cookiec;

4008        /*
4009         * If the dma was broken up into PARTIAL transfers cmd_nwin will be
4010         * greater than 0 and we'll need to grab the next dma window
4011         */
```



```
4012        /*
4013         * SLM-not doing extra command frame right now; may add later
4014         */

4016        if (cmd->cmd_nwin > 0) {

4018                /*
4019                 * Make sure we havn't gone past the the total number
4020                 * of windows
4021                 */
4022                if (++cmd->cmd_winindex >= cmd->cmd_nwin) {
4023                        return (NULL);
4024                }
4025                if (ddi_dma_getwin(cmd->cmd_dmahandle, cmd->cmd_winindex,
4026                    &cmd->cmd_dma_offset, &cmd->cmd_dma_len,
4027                    &cmd->cmd_cookie, &cmd->cmd_cookiec) == DDI_FAILURE) {
4028                        return (NULL);
4029                }
4030                goto get_dma_cookies;
4031        }


4034        if (flags & PKT_XARQ) {
4035                cmd->cmd_flags |= CFLAG_XARQ;
4036        }

4038        /*
4039         * DMA resource allocation.  This version assumes your
4040         * HBA has some sort of bus-mastering or onboard DMA capability, with a
4041         * scatter-gather list of length MPTSAS_MAX_DMA_SEGS, as given in the
4042         * ddi_dma_attr_t structure and passed to scsi_impl_dmaget.
4043         */
4044        if (bp && (bp->b_bcount != 0) &&
4045            (cmd->cmd_flags & CFLAG_DMAVALID) == 0) {

4047                int     cnt, dma_flags;
4048                mptti_t *dmap;          /* ptr to the S/G list */

4050                /*
4051                 * Set up DMA memory and position to the next DMA segment.
4052                 */
4053                ASSERT(cmd->cmd_dmahandle != NULL);

4055                if (bp->b_flags & B_READ) {
4056                        dma_flags = DDI_DMA_READ;
4057                        cmd->cmd_flags &= ~CFLAG_DMASEND;
4058                } else {
4059                        dma_flags = DDI_DMA_WRITE;
4060                        cmd->cmd_flags |= CFLAG_DMASEND;
4061                }
4062                if (flags & PKT_CONSISTENT) {
4063                        cmd->cmd_flags |= CFLAG_CMDIOPB;
4064                        dma_flags |= DDI_DMA_CONSISTENT;
4065                }

4067                if (flags & PKT_DMA_PARTIAL) {
4068                        dma_flags |= DDI_DMA_PARTIAL;
4069                }

4071                /*
4072                 * workaround for byte hole issue on psycho and
4073                 * schizo pre 2.1
4074                 */
4075                if ((bp->b_flags & B_READ) && ((bp->b_flags &
4076                    (B_PAGEIO|B_REMAPPED)) != B_PAGEIO) &&
4077                    ((uintptr_t)bp->b_un.b_addr & 0x7)) {
```

```
4078                        dma_flags |= DDI_DMA_CONSISTENT;
4079                }

4081                rval = ddi_dma_buf_bind_handle(cmd->cmd_dmahandle, bp,
4082                    dma_flags, callback, arg,
4083                    &cmd->cmd_cookie, &cmd->cmd_cookiec);
4084                if (rval == DDI_DMA_PARTIAL_MAP) {
4085                        (void) ddi_dma_numwin(cmd->cmd_dmahandle,
4086                            &cmd->cmd_nwin);
4087                        cmd->cmd_winindex = 0;
4088                        (void) ddi_dma_getwin(cmd->cmd_dmahandle,
4089                            cmd->cmd_winindex, &cmd->cmd_dma_offset,
4090                            &cmd->cmd_dma_len, &cmd->cmd_cookie,
4091                            &cmd->cmd_cookiec);
4092                } else if (rval && (rval != DDI_DMA_MAPPED)) {
4093                        switch (rval) {
4094                        case DDI_DMA_NORESOURCES:
4095                                bioerror(bp, 0);
4096                                break;
4097                        case DDI_DMA_BADATTR:
4098                        case DDI_DMA_NOMAPPING:
4099                                bioerror(bp, EFAULT);
4100                                break;
4101                        case DDI_DMA_TOOBIG:
4102                        default:
4103                                bioerror(bp, EINVAL);
4104                                break;
4105                        }
4106                        cmd->cmd_flags &= ~CFLAG_DMAVALID;
4107                        if (new_cmd) {
4108                                mptsas_scsi_destroy_pkt(ap, pkt);
4109                        }
4110                        return ((struct scsi_pkt *)NULL);
4111                }

4113 get_dma_cookies:
4114                cmd->cmd_flags |= CFLAG_DMAVALID;
4115                ASSERT(cmd->cmd_cookiec > 0);

4117                if (cmd->cmd_cookiec > MPTSAS_MAX_CMD_SEGS) {
4118                        mptsas_log(mpt, CE_NOTE, "large cookiec received %d\n",
4119                            cmd->cmd_cookiec);
4120                        bioerror(bp, EINVAL);
4121                        if (new_cmd) {
4122                                mptsas_scsi_destroy_pkt(ap, pkt);
4123                        }
4124                        return ((struct scsi_pkt *)NULL);
4125                }

4127                /*
4128                 * Allocate extra SGL buffer if needed.
4129                 */
4130                if ((cmd->cmd_cookiec > MPTSAS_MAX_FRAME_SGES64(mpt)) &&
4131                    (cmd->cmd_extra_frames == NULL)) {
4132                        if (mptsas_alloc_extra_sgl_frame(mpt, cmd) ==
4133                            DDI_FAILURE) {
4134                                mptsas_log(mpt, CE_WARN, "MPT SGL mem alloc "
4135                                    "failed");
4136                                bioerror(bp, ENOMEM);
4137                                if (new_cmd) {
4138                                        mptsas_scsi_destroy_pkt(ap, pkt);
4139                                }
4140                                return ((struct scsi_pkt *)NULL);
4141                        }
4142                }
```



```
4144                /*
4145                 * Always use scatter-gather transfer
4146                 * Use the loop below to store physical addresses of
4147                 * DMA segments, from the DMA cookies, into your HBA's
4148                 * scatter-gather list.
4149                 * We need to ensure we have enough kmem alloc'd
4150                 * for the sg entries since we are no longer using an
4151                 * array inside mptsas_cmd_t.
4152                 *
4153                 * We check cmd->cmd_cookiec against oldcookiec so
4154                 * the scatter-gather list is correctly allocated
4155                 */

4157                if (oldcookiec != cmd->cmd_cookiec) {
4158                        if (cmd->cmd_sg != (mptti_t *)NULL) {
4159                                kmem_free(cmd->cmd_sg, sizeof (mptti_t) *
4160                                    oldcookiec);
4161                                cmd->cmd_sg = NULL;
4162                        }
4163                }

4165                if (cmd->cmd_sg == (mptti_t *)NULL) {
4166                        cmd->cmd_sg = kmem_alloc((size_t)(sizeof (mptti_t)*
4167                            cmd->cmd_cookiec), kf);

4169                        if (cmd->cmd_sg == (mptti_t *)NULL) {
4170                                mptsas_log(mpt, CE_WARN,
4171                                    "unable to kmem_alloc enough memory "
4172                                    "for scatter/gather list");
4173                /*
4174                 * if we have an ENOMEM condition we need to behave
4175                 * the same way as the rest of this routine
4176                 */

4178                                bioerror(bp, ENOMEM);
4179                                if (new_cmd) {
4180                                        mptsas_scsi_destroy_pkt(ap, pkt);
4181                                }
4182                                return ((struct scsi_pkt *)NULL);
4183                        }
4184                }

4186                dmap = cmd->cmd_sg;

4188                ASSERT(cmd->cmd_cookie.dmac_size != 0);

4190                /*
4191                 * store the first segment into the S/G list
4192                 */
4193                dmap->count = cmd->cmd_cookie.dmac_size;
4194                dmap->addr.address64.Low = (uint32_t)
4195                    (cmd->cmd_cookie.dmac_laddress & 0xffffffffull);
4196                dmap->addr.address64.High = (uint32_t)
4197                    (cmd->cmd_cookie.dmac_laddress >> 32);

4199                /*
4200                 * dmacount counts the size of the dma for this window
4201                 * (if partial dma is being used).  totaldmacount
4202                 * keeps track of the total amount of dma we have
4203                 * transferred for all the windows (needed to calculate
4204                 * the resid value below).
4205                 */
4206                cmd->cmd_dmacount = cmd->cmd_cookie.dmac_size;
4207                cmd->cmd_totaldmacount += cmd->cmd_cookie.dmac_size;

4209                /*
```

```
4210                  * We already stored the first DMA scatter gather segment,
4211                  * start at 1 if we need to store more.
4212                  */
4213                 for (cnt = 1; cnt < cmd->cmd_cookiec; cnt++) {
4214                         /*
4215                          * Get next DMA cookie
4216                          */
4217                         ddi_dma_nextcookie(cmd->cmd_dmahandle,
4218                             &cmd->cmd_cookie);
4219                         dmap++;

4221                         cmd->cmd_dmacount += cmd->cmd_cookie.dmac_size;
4222                         cmd->cmd_totaldmacount += cmd->cmd_cookie.dmac_size;

4224                         /*
4225                          * store the segment parms into the S/G list
4226                          */
4227                         dmap->count = cmd->cmd_cookie.dmac_size;
4228                         dmap->addr.address64.Low = (uint32_t)
4229                             (cmd->cmd_cookie.dmac_laddress & 0xffffffffull);
4230                         dmap->addr.address64.High = (uint32_t)
4231                             (cmd->cmd_cookie.dmac_laddress >> 32);
4232                 }

4234                 /*
4235                  * If this was partially allocated we set the resid
4236                  * the amount of data NOT transferred in this window
4237                  * If there is only one window, the resid will be 0
4238                  */
4239                 pkt->pkt_resid = (bp->b_bcount - cmd->cmd_totaldmacount);
4240                 NDBG3(("mptsas_scsi_init_pkt: cmd_dmacount=%d.",
4241                     cmd->cmd_dmacount));
2703                 NDBG16(("mptsas_dmaget: cmd_dmacount=%d.", cmd->cmd_dmacount));
4242         }
4243         return (pkt);
4244 }
```

*_____unchanged_portion_omitted_*

```
4284 /*
4285  * kmem cache constructor and destructor:
4286  * When constructing, we bzero the cmd and allocate the dma handle
4287  * When destructing, just free the dma handle
4288  */
4289 static int
4290 mptsas_kmem_cache_constructor(void *buf, void *cdrarg, int kmflags)
4291 {
4292         mptsas_cmd_t            *cmd = buf;
4293         mptsas_t                *mpt  = cdrarg;
2756         struct scsi_address     ap;
2757         uint_t                  cookiec;
2758         ddi_dma_attr_t          arq_dma_attr;
4294         int                     (*callback)(caddr_t);

4296         callback = (kmflags == KM_SLEEP)? DDI_DMA_SLEEP: DDI_DMA_DONTWAIT;

4298         NDBG4(("mptsas_kmem_cache_constructor"));

2765         ap.a_hba_tran = mpt->m_tran;
2766         ap.a_target = 0;
2767         ap.a_lun = 0;

4300         /*
4301          * allocate a dma handle
4302          */
4303         if ((ddi_dma_alloc_handle(mpt->m_dip, &mpt->m_io_dma_attr, callback,
4304             NULL, &cmd->cmd_dmahandle)) != DDI_SUCCESS) {
```



```
4305                 cmd->cmd_dmahandle = NULL;
4306                 return (-1);
4307         }

2778         cmd->cmd_arq_buf = scsi_alloc_consistent_buf(&ap, (struct buf *)NULL,
2779             SENSE_LENGTH, B_READ, callback, NULL);
2780         if (cmd->cmd_arq_buf == NULL) {
2781                 ddi_dma_free_handle(&cmd->cmd_dmahandle);
2782                 cmd->cmd_dmahandle = NULL;
2783                 return (-1);
2784         }

2786         /*
2787          * allocate a arq handle
2788          */
2789         arq_dma_attr = mpt->m_msg_dma_attr;
2790         arq_dma_attr.dma_attr_sgllen = 1;
2791         if ((ddi_dma_alloc_handle(mpt->m_dip, &arq_dma_attr, callback,
2792             NULL, &cmd->cmd_arqhandle)) != DDI_SUCCESS) {
2793                 ddi_dma_free_handle(&cmd->cmd_dmahandle);
2794                 scsi_free_consistent_buf(cmd->cmd_arq_buf);
2795                 cmd->cmd_dmahandle = NULL;
2796                 cmd->cmd_arqhandle = NULL;
2797                 return (-1);
2798         }

2800         if (ddi_dma_buf_bind_handle(cmd->cmd_arqhandle,
2801             cmd->cmd_arq_buf, (DDI_DMA_READ | DDI_DMA_CONSISTENT),
2802             callback, NULL, &cmd->cmd_arqcookie, &cookiec) != DDI_SUCCESS) {
2803                 ddi_dma_free_handle(&cmd->cmd_dmahandle);
2804                 ddi_dma_free_handle(&cmd->cmd_arqhandle);
2805                 scsi_free_consistent_buf(cmd->cmd_arq_buf);
2806                 cmd->cmd_dmahandle = NULL;
2807                 cmd->cmd_arqhandle = NULL;
2808                 cmd->cmd_arq_buf = NULL;
2809                 return (-1);
2810         }

4308         return (0);
4309 }

4311 static void
4312 mptsas_kmem_cache_destructor(void *buf, void *cdrarg)
4313 {
4314 #ifndef __lock_lint
4315         _NOTE(ARGUNUSED(cdrarg))
4316 #endif
4317         mptsas_cmd_t    *cmd = buf;

4319         NDBG4(("mptsas_kmem_cache_destructor"));

2825         if (cmd->cmd_arqhandle) {
2826                 (void) ddi_dma_unbind_handle(cmd->cmd_arqhandle);
2827                 ddi_dma_free_handle(&cmd->cmd_arqhandle);
2828                 cmd->cmd_arqhandle = NULL;
2829         }
2830         if (cmd->cmd_arq_buf) {
2831                 scsi_free_consistent_buf(cmd->cmd_arq_buf);
2832                 cmd->cmd_arq_buf = NULL;
2833         }
4321         if (cmd->cmd_dmahandle) {
4322                 ddi_dma_free_handle(&cmd->cmd_dmahandle);
4323                 cmd->cmd_dmahandle = NULL;
4324         }
4325 }
```

```
4327 static int
4328 mptsas_cache_frames_constructor(void *buf, void *cdrarg, int kmflags)
4329 {
4330         mptsas_cache_frames_t   *p = buf;
4331         mptsas_t                *mpt = cdrarg;
4332         ddi_dma_attr_t          frame_dma_attr;
4333         size_t                  mem_size, alloc_len;
4334         ddi_dma_cookie_t        cookie;
4335         uint_t                  ncookie;
4336         int (*callback)(caddr_t) = (kmflags == KM_SLEEP)
4337             ? DDI_DMA_SLEEP: DDI_DMA_DONTWAIT;

4339         frame_dma_attr = mpt->m_msg_dma_attr;
4340         frame_dma_attr.dma_attr_align = 0x10;
4341         frame_dma_attr.dma_attr_sgllen = 1;

4343         if (ddi_dma_alloc_handle(mpt->m_dip, &frame_dma_attr, callback, NULL,
4344             &p->m_dma_hdl) != DDI_SUCCESS) {
4345                 mptsas_log(mpt, CE_WARN, "Unable to allocate dma handle for"
4346                     " extra SGL.");
4347                 return (DDI_FAILURE);
4348         }

4350         mem_size = (mpt->m_max_request_frames - 1) * mpt->m_req_frame_size;

4352         if (ddi_dma_mem_alloc(p->m_dma_hdl, mem_size, &mpt->m_dev_acc_attr,
4353             DDI_DMA_CONSISTENT, callback, NULL, (caddr_t *)&p->m_frames_addr,
4354             &alloc_len, &p->m_acc_hdl) != DDI_SUCCESS) {
4355                 ddi_dma_free_handle(&p->m_dma_hdl);
4356                 p->m_dma_hdl = NULL;
4357                 mptsas_log(mpt, CE_WARN, "Unable to allocate dma memory for"
4358                     " extra SGL.");
4359                 return (DDI_FAILURE);
4360         }

4362         if (ddi_dma_addr_bind_handle(p->m_dma_hdl, NULL, p->m_frames_addr,
4363             alloc_len, DDI_DMA_RDWR | DDI_DMA_CONSISTENT, callback, NULL,
4364             &cookie, &ncookie) != DDI_DMA_MAPPED) {
4365                 (void) ddi_dma_mem_free(&p->m_acc_hdl);
4366                 ddi_dma_free_handle(&p->m_dma_hdl);
4367                 p->m_dma_hdl = NULL;
4368                 mptsas_log(mpt, CE_WARN, "Unable to bind DMA resources for"
4369                     " extra SGL");
4370                 return (DDI_FAILURE);
4371         }

4373         /*
4374          * Store the SGL memory address.  This chip uses this
4375          * address to dma to and from the driver.  The second
4376          * address is the address mpt uses to fill in the SGL.
4377          */
4378         p->m_phys_addr = cookie.dmac_laddress;
2891         p->m_phys_addr = cookie.dmac_address;

4380         return (DDI_SUCCESS);
4381 }
```

_____*unchanged_portion_omitted_*

```
4402 /*
4403  * allocate and deallocate external pkt space (ie. not part of mptsas_cmd)
4404  * for non-standard length cdb, pkt_private, status areas
4405  * if allocation fails, then deallocate all external space and the pkt
4406  */
4407 /* ARGSUSED */
4408 static int
4409 mptsas_pkt_alloc_extern(mptsas_t *mpt, mptsas_cmd_t *cmd,
```



```
4410     int cmdlen, int tgtlen, int statuslen, int kf)
4411 {
4412         caddr_t                 cdbp, scbp, tgt;
2926         int                     (*callback)(caddr_t) = (kf == KM_SLEEP) ?
2927             DDI_DMA_SLEEP : DDI_DMA_DONTWAIT;
2928         struct scsi_address     ap;
4413         size_t                  senselength;
2930         ddi_dma_attr_t          ext_arq_dma_attr;
2931         uint_t                  cookiec;

4415         NDBG3(("mptsas_pkt_alloc_extern: "
4416             "cmd=0x%p cmdlen=%d tgtlen=%d statuslen=%d kf=%x",
4417             (void *)cmd, cmdlen, tgtlen, statuslen, kf));

4419         tgt = cdbp = scbp = NULL;
4420         cmd->cmd_scblen         = statuslen;
4421         cmd->cmd_privlen        = (uchar_t)tgtlen;

4423         if (cmdlen > sizeof (cmd->cmd_cdb)) {
4424                 if ((cdbp = kmem_zalloc((size_t)cmdlen, kf)) == NULL) {
4425                         goto fail;
4426                 }
4427                 cmd->cmd_pkt->pkt_cdbp = (opaque_t)cdbp;
4428                 cmd->cmd_flags |= CFLAG_CDBEXTERN;
4429         }
4430         if (tgtlen > PKT_PRIV_LEN) {
4431                 if ((tgt = kmem_zalloc((size_t)tgtlen, kf)) == NULL) {
4432                         goto fail;
4433                 }
4434                 cmd->cmd_flags |= CFLAG_PRIVEXTERN;
4435                 cmd->cmd_pkt->pkt_private = tgt;
4436         }
4437         if (statuslen > EXTCMDS_STATUS_SIZE) {
4438                 if ((scbp = kmem_zalloc((size_t)statuslen, kf)) == NULL) {
4439                         goto fail;
4440                 }
4441                 cmd->cmd_flags |= CFLAG_SCBEXTERN;
4442                 cmd->cmd_pkt->pkt_scbp = (opaque_t)scbp;

4444                 /* allocate sense data buf for DMA */

4446                 senselength = statuslen - MPTSAS_GET_ITEM_OFF(
4447                     struct scsi_arq_status, sts_sensedata);
4448                 if (senselength > mpt->m_req_sense_size) {
4449                         unsigned long i;
4450                         cmd->cmd_extrqslen = (uint16_t)senselength;
4451                         cmd->cmd_extrqschunks = (senselength +
4452                             (mpt->m_req_sense_size - 1))/mpt->m_req_sense_size;
4453                         i = rmalloc_wait(mpt->m_erqsense_map,
4454                             cmd->cmd_extrqschunks);
4455                         ASSERT(i != 0);
4456                         cmd->cmd_extrqsidx = i - 1;
4457                         cmd->cmd_arq_buf = mpt->m_extreq_sense +
4458                             (cmd->cmd_extrqsidx * mpt->m_req_sense_size);
4459                 } else {
4460 #endif /* ! codereview */
4461                         cmd->cmd_rqslen = (uchar_t)senselength;

2967                 ap.a_hba_tran = mpt->m_tran;
2968                 ap.a_target = 0;
2969                 ap.a_lun = 0;

2971                 cmd->cmd_ext_arq_buf = scsi_alloc_consistent_buf(&ap,
2972                     (struct buf *)NULL, senselength, B_READ,
2973                     callback, NULL);
```

```
2975                 if (cmd->cmd_ext_arq_buf == NULL) {
2976                         goto fail;
2977                 }
2978                 /*
2979                  * allocate a extern arq handle and bind the buf
2980                  */
2981                 ext_arq_dma_attr = mpt->m_msg_dma_attr;
2982                 ext_arq_dma_attr.dma_attr_sgllen = 1;
2983                 if ((ddi_dma_alloc_handle(mpt->m_dip,
2984                     &ext_arq_dma_attr, callback,
2985                     NULL, &cmd->cmd_ext_arqhandle)) != DDI_SUCCESS) {
2986                         goto fail;
2987                 }

2989                 if (ddi_dma_buf_bind_handle(cmd->cmd_ext_arqhandle,
2990                     cmd->cmd_ext_arq_buf, (DDI_DMA_READ | DDI_DMA_CONSISTENT),
2991                     callback, NULL, &cmd->cmd_ext_arqcookie,
2992                     &cookiec)
2993                     != DDI_SUCCESS) {
2994                         goto fail;
4462                 }
2996                 cmd->cmd_flags |= CFLAG_EXTARQBUFVALID;
4463         }
4464         return (0);
4465 fail:
4466         mptsas_pkt_destroy_extern(mpt, cmd);
4467         return (1);
4468 }

4470 /*
4471  * deallocate external pkt space and deallocate the pkt
4472  */
4473 static void
4474 mptsas_pkt_destroy_extern(mptsas_t *mpt, mptsas_cmd_t *cmd)
4475 {
4476         NDBG3(("mptsas_pkt_destroy_extern: cmd=0x%p", (void *)cmd));

4478         if (cmd->cmd_flags & CFLAG_FREE) {
4479                 mptsas_log(mpt, CE_PANIC,
4480                     "mptsas_pkt_destroy_extern: freeing free packet");
4481                 _NOTE(NOT_REACHED)
4482                 /* NOTREACHED */
4483         }
4484         if (cmd->cmd_extrqslen != 0) {
4485                 rmfree(mpt->m_erqsense_map, cmd->cmd_extrqschunks,
4486                     cmd->cmd_extrqsidx + 1);
4487         }
4488 #endif /* ! codereview */
4489         if (cmd->cmd_flags & CFLAG_CDBEXTERN) {
4490                 kmem_free(cmd->cmd_pkt->pkt_cdbp, (size_t)cmd->cmd_cdblen);
4491         }
4492         if (cmd->cmd_flags & CFLAG_SCBEXTERN) {
4493                 kmem_free(cmd->cmd_pkt->pkt_scbp, (size_t)cmd->cmd_scblen);
3018                 if (cmd->cmd_flags & CFLAG_EXTARQBUFVALID) {
3019                         (void) ddi_dma_unbind_handle(cmd->cmd_ext_arqhandle);
3020                 }
3021                 if (cmd->cmd_ext_arqhandle) {
3022                         ddi_dma_free_handle(&cmd->cmd_ext_arqhandle);
3023                         cmd->cmd_ext_arqhandle = NULL;
3024                 }
3025                 if (cmd->cmd_ext_arq_buf)
3026                         scsi_free_consistent_buf(cmd->cmd_ext_arq_buf);
4494         }
4495         if (cmd->cmd_flags & CFLAG_PRIVEXTERN) {
4496                 kmem_free(cmd->cmd_pkt->pkt_private, (size_t)cmd->cmd_privlen);
4497         }
```



```
4498         cmd->cmd_flags = CFLAG_FREE;
4499         kmem_cache_free(mpt->m_kmem_cache, (void *)cmd);
4500 }
_____unchanged_portion_omitted_

4521 /*
4522  * tran_dmafree(9E) - deallocate DMA resources allocated for command
4523  */
4524 /*ARGSUSED*/
4525 static void
4526 mptsas_scsi_dmafree(struct scsi_address *ap, struct scsi_pkt *pkt)
4527 {
4528         mptsas_cmd_t    *cmd = PKT2CMD(pkt);
4529         mptsas_t        *mpt = ADDR2MPT(ap);

4531         NDBG3(("mptsas_scsi_dmafree: target=%d pkt=0x%p",
4532             ap->a_target, (void *)pkt));

4534         if (cmd->cmd_flags & CFLAG_DMAVALID) {
4535                 (void) ddi_dma_unbind_handle(cmd->cmd_dmahandle);
4536                 cmd->cmd_flags &= ~CFLAG_DMAVALID;
4537         }

3072         if (cmd->cmd_flags & CFLAG_EXTARQBUFVALID) {
3073                 (void) ddi_dma_unbind_handle(cmd->cmd_ext_arqhandle);
3074                 cmd->cmd_flags &= ~CFLAG_EXTARQBUFVALID;
3075         }

4539         mptsas_free_extra_sgl_frame(mpt, cmd);
4540 }
_____unchanged_portion_omitted_

4553 static void
4554 mptsas_sge_mainframe(mptsas_cmd_t *cmd, pMpi2SCSIIORequest_t frame,
4555                 ddi_acc_handle_t acc_hdl, uint_t cookiec,
4556                 uint32_t end_flags)
3092 mptsas_sge_setup(mptsas_t *mpt, mptsas_cmd_t *cmd, uint32_t *control,
3093         pMpi2SCSIIORequest_t frame, ddi_acc_handle_t acc_hdl)
4557 {
4558         pMpi2SGESimple64_t      sge;
3095         uint_t                  cookiec;
4559         mptti_t                 *dmap;
4560         uint32_t                flags;
3098         pMpi2SGESimple64_t      sge;
3099         pMpi2SGEChain64_t       sgechain;
3100         ASSERT(cmd->cmd_flags & CFLAG_DMAVALID);

4562         dmap = cmd->cmd_sg;
3102         /*
3103          * Save the number of entries in the DMA
3104          * Scatter/Gather list
3105          */
3106         cookiec = cmd->cmd_cookiec;

3108         NDBG1(("mptsas_sge_setup: cookiec=%d", cookiec));

3110         /*
3111          * Set read/write bit in control.
3112          */
3113         if (cmd->cmd_flags & CFLAG_DMASEND) {
3114                 *control |= MPI2_SCSIIO_CONTROL_WRITE;
3115         } else {
3116                 *control |= MPI2_SCSIIO_CONTROL_READ;
3117         }

3119         ddi_put32(acc_hdl, &frame->DataLength, cmd->cmd_dmacount);
```

```
3121         /*
3122          * We have 2 cases here.  First where we can fit all the
3123          * SG elements into the main frame, and the case
3124          * where we can't.
3125          * If we have more cookies than we can attach to a frame
3126          * we will need to use a chain element to point
3127          * a location of memory where the rest of the S/G
3128          * elements reside.
3129          */
3130         if (cookiec <= MPTSAS_MAX_FRAME_SGES64(mpt)) {
3131                 dmap = cmd->cmd_sg;
4564         sge = (pMpi2SGESimple64_t)(&frame->SGL);
4565         while (cookiec--) {
4566                 ddi_put32(acc_hdl, &sge->Address.Low,
4567                     dmap->addr.address64.Low);
4568                 ddi_put32(acc_hdl, &sge->Address.High,
4569                     dmap->addr.address64.High);
4570                 ddi_put32(acc_hdl, &sge->FlagsLength, dmap->count);
3134                         ddi_put32(acc_hdl,
3135                             &sge->Address.Low, dmap->addr.address64.Low);
3136                         ddi_put32(acc_hdl,
3137                             &sge->Address.High, dmap->addr.address64.High);
3138                         ddi_put32(acc_hdl, &sge->FlagsLength,
3139                             dmap->count);
4571                 flags = ddi_get32(acc_hdl, &sge->FlagsLength);
4572                 flags |= ((uint32_t)
4573                     (MPI2_SGE_FLAGS_SIMPLE_ELEMENT |
4574                     MPI2_SGE_FLAGS_SYSTEM_ADDRESS |
4575                     MPI2_SGE_FLAGS_64_BIT_ADDRESSING) <<
4576                     MPI2_SGE_FLAGS_SHIFT);

4578                 /*
4579                  * If this is the last cookie, we set the flags
4580                  * to indicate so
4581                  */
4582                 if (cookiec == 0) {
4583                         flags |= end_flags;
3152                                 flags |=
3153                                     ((uint32_t)(MPI2_SGE_FLAGS_LAST_ELEMENT
3154                                     | MPI2_SGE_FLAGS_END_OF_BUFFER
3155                                     | MPI2_SGE_FLAGS_END_OF_LIST) <<
3156                                     MPI2_SGE_FLAGS_SHIFT);
4584                 }
4585                 if (cmd->cmd_flags & CFLAG_DMASEND) {
4586                         flags |= (MPI2_SGE_FLAGS_HOST_TO_IOC <<
4587                             MPI2_SGE_FLAGS_SHIFT);
4588                 } else {
4589                         flags |= (MPI2_SGE_FLAGS_IOC_TO_HOST <<
4590                             MPI2_SGE_FLAGS_SHIFT);
4591                 }
4592                 ddi_put32(acc_hdl, &sge->FlagsLength, flags);
4593                 dmap++;
4594                 sge++;
4595         }
4596 }

4598 static void
4599 mptsas_sge_chain(mptsas_t *mpt, mptsas_cmd_t *cmd,
4600     pMpi2SCSIIORequest_t frame, ddi_acc_handle_t acc_hdl)
4601 {
4602         pMpi2SGESimple64_t      sge;
4603         pMpi2SGEChain64_t       sgechain;
4604         uint64_t                nframe_phys_addr;
4605         uint_t                  cookiec;
4606         mptti_t                 *dmap;
```



```
4607         uint32_t                flags;
4608         int                     i, j, k, l, frames, sgemax;
4609         int                     temp, maxframe_sges;
4610         uint8_t                 chainflags;
4611         uint16_t                chainlength;
4612         mptsas_cache_frames_t   *p;

4614         cookiec = cmd->cmd_cookiec;

3169         } else {
4616         /*
4617          * Hereby we start to deal with multiple frames.
4618          * The process is as follows:
4619          * 1. Determine how many frames are needed for SGL element
4620          *    storage; Note that all frames are stored in contiguous
4621          *    memory space and in 64-bit DMA mode each element is
4622          *    3 double-words (12 bytes) long.
4623          * 2. Fill up the main frame. We need to do this separately
4624          *    since it contains the SCSI IO request header and needs
4625          *    dedicated processing. Note that the last 4 double-words
4626          *    of the SCSI IO header is for SGL element storage
4627          *    (MPI2_SGE_IO_UNION).
4628          * 3. Fill the chain element in the main frame, so the DMA
4629          *    engine can use the following frames.
4630          * 4. Enter a loop to fill the remaining frames. Note that the
4631          *    last frame contains no chain element.  The remaining
4632          *    frames go into the mpt SGL buffer allocated on the fly,
4633          *    not immediately following the main message frame, as in
4634          *    Gen1.
4635          * Some restrictions:
4636          * 1. For 64-bit DMA, the simple element and chain element
4637          *    are both of 3 double-words (12 bytes) in size, even
4638          *    though all frames are stored in the first 4G of mem
4639          *    range and the higher 32-bits of the address are always 0.
4640          * 2. On some controllers (like the 1064/1068), a frame can
4641          *    hold SGL elements with the last 1 or 2 double-words
4642          *    (4 or 8 bytes) un-used. On these controllers, we should
4643          *    recognize that there's not enough room for another SGL
4644          *    element and move the sge pointer to the next frame.
4645          */
3200                 int             i, j, k, l, frames, sgemax;
3201                 int             temp;
3202                 uint8_t         chainflags;
3203                 uint16_t        chainlength;
3204                 mptsas_cache_frames_t *p;

4647         /*
4648          * Sgemax is the number of SGE's that will fit
4649          * each extra frame and frames is total
4650          * number of frames we'll need.  1 sge entry per
4651          * frame is reseverd for the chain element thus the -1 below.
4652          */
4653         sgemax = ((mpt->m_req_frame_size / sizeof (MPI2_SGE_SIMPLE64)) - 1);
4654         maxframe_sges = MPTSAS_MAX_FRAME_SGES64(mpt);
4655         temp = (cookiec - (maxframe_sges - 1)) / sgemax;
3212                 sgemax = ((mpt->m_req_frame_size / sizeof (MPI2_SGE_SIMPLE64))
3213                     - 1);
3214                 temp = (cookiec - (MPTSAS_MAX_FRAME_SGES64(mpt) - 1)) / sgemax;

4657         /*
4658          * A little check to see if we need to round up the number
4659          * of frames we need
4660          */
4661         if ((cookiec - (maxframe_sges - 1)) - (temp * sgemax) > 1) {
3220                 if ((cookiec - (MPTSAS_MAX_FRAME_SGES64(mpt) - 1)) - (temp *
3221                     sgemax) > 1) {
```

```
4662                frames = (temp + 1);
4663        } else {
4664                frames = temp;
4665        }
4666        dmap = cmd->cmd_sg;
4667        sge = (pMpi2SGESimple64_t)(&frame->SGL);

4669        /*
4670         * First fill in the main frame
4671         */
4672        j = maxframe_sges - 1;
4673        mptsas_sge_mainframe(cmd, frame, acc_hdl, j,
4674            ((uint32_t)(MPI2_SGE_FLAGS_LAST_ELEMENT) <<
4675            MPI2_SGE_FLAGS_SHIFT));
4676        dmap += j;
4677        sge += j;
4678        j++;
3232                for (j = 1; j < MPTSAS_MAX_FRAME_SGES64(mpt); j++) {
3233                        ddi_put32(acc_hdl, &sge->Address.Low,
3234                            dmap->addr.address64.Low);
3235                        ddi_put32(acc_hdl, &sge->Address.High,
3236                            dmap->addr.address64.High);
3237                        ddi_put32(acc_hdl, &sge->FlagsLength, dmap->count);
3238                        flags = ddi_get32(acc_hdl, &sge->FlagsLength);
3239                        flags |= ((uint32_t)(MPI2_SGE_FLAGS_SIMPLE_ELEMENT |
3240                            MPI2_SGE_FLAGS_SYSTEM_ADDRESS |
3241                            MPI2_SGE_FLAGS_64_BIT_ADDRESSING) <<
3242                            MPI2_SGE_FLAGS_SHIFT);

3244                        /*
3245                         * If this is the last SGE of this frame
3246                         * we set the end of list flag
3247                         */
3248                        if (j == (MPTSAS_MAX_FRAME_SGES64(mpt) - 1)) {
3249                                flags |= ((uint32_t)
3250                                    (MPI2_SGE_FLAGS_LAST_ELEMENT) <<
3251                                    MPI2_SGE_FLAGS_SHIFT);
3252                        }
3253                        if (cmd->cmd_flags & CFLAG_DMASEND) {
3254                                flags |=
3255                                    (MPI2_SGE_FLAGS_HOST_TO_IOC <<
3256                                    MPI2_SGE_FLAGS_SHIFT);
3257                        } else {
3258                                flags |=
3259                                    (MPI2_SGE_FLAGS_IOC_TO_HOST <<
3260                                    MPI2_SGE_FLAGS_SHIFT);
3261                        }
3262                        ddi_put32(acc_hdl, &sge->FlagsLength, flags);
3263                        dmap++;
3264                        sge++;
3265                }

4680        /*
4681         * Fill in the chain element in the main frame.
4682         * About calculation on ChainOffset:
4683         * 1. Struct msg_scsi_io_request has 4 double-words (16 bytes)
4684         *    in the end reserved for SGL element storage
4685         *    (MPI2_SGE_IO_UNION); we should count it in our
4686         *    calculation.  See its definition in the header file.
4687         * 2. Constant j is the counter of the current SGL element
4688         *    that will be processed, and (j - 1) is the number of
4689         *    SGL elements that have been processed (stored in the
4690         *    main frame).
4691         * 3. ChainOffset value should be in units of double-words (4
4692         *    bytes) so the last value should be divided by 4.
4693         */
```



```
4694        ddi_put8(acc_hdl, &frame->ChainOffset,
4695            (sizeof (MPI2_SCSI_IO_REQUEST) -
4696            sizeof (MPI2_SGE_IO_UNION) +
4697            (j - 1) * sizeof (MPI2_SGE_SIMPLE64)) >> 2);
4698        sgechain = (pMpi2SGEChain64_t)sge;
4699        chainflags = (MPI2_SGE_FLAGS_CHAIN_ELEMENT |
4700            MPI2_SGE_FLAGS_SYSTEM_ADDRESS |
4701            MPI2_SGE_FLAGS_64_BIT_ADDRESSING);
4702        ddi_put8(acc_hdl, &sgechain->Flags, chainflags);

4704        /*
4705         * The size of the next frame is the accurate size of space
4706         * (in bytes) used to store the SGL elements. j is the counter
4707         * of SGL elements. (j - 1) is the number of SGL elements that
4708         * have been processed (stored in frames).
4709         */
4710        if (frames >= 2) {
4711                chainlength = mpt->m_req_frame_size /
4712                    sizeof (MPI2_SGE_SIMPLE64) *
4713                    sizeof (MPI2_SGE_SIMPLE64);
4714        } else {
4715                chainlength = ((cookiec - (j - 1)) *
4716                    sizeof (MPI2_SGE_SIMPLE64));
4717        }

4719        p = cmd->cmd_extra_frames;

4721        ddi_put16(acc_hdl, &sgechain->Length, chainlength);
4722        ddi_put32(acc_hdl, &sgechain->Address.Low,
4723            (p->m_phys_addr&0xffffffffull));
4724        ddi_put32(acc_hdl, &sgechain->Address.High, p->m_phys_addr>>32);
3310                    p->m_phys_addr);
3311                /* SGL is allocated in the first 4G mem range */
3312                ddi_put32(acc_hdl, &sgechain->Address.High, 0);

4726        /*
4727         * If there are more than 2 frames left we have to
4728         * fill in the next chain offset to the location of
4729         * the chain element in the next frame.
4730         * sgemax is the number of simple elements in an extra
4731         * frame. Note that the value NextChainOffset should be
4732         * in double-words (4 bytes).
4733         */
4734        if (frames >= 2) {
4735                ddi_put8(acc_hdl, &sgechain->NextChainOffset,
4736                    (sgemax * sizeof (MPI2_SGE_SIMPLE64)) >> 2);
4737        } else {
4738                ddi_put8(acc_hdl, &sgechain->NextChainOffset, 0);
4739        }

4741        /*
4742         * Jump to next frame;
4743         * Starting here, chain buffers go into the per command SGL.
4744         * This buffer is allocated when chain buffers are needed.
4745         */
4746        sge = (pMpi2SGESimple64_t)p->m_frames_addr;
4747        i = cookiec;

4749        /*
4750         * Start filling in frames with SGE's.  If we
4751         * reach the end of frame and still have SGE's
4752         * to fill we need to add a chain element and
4753         * use another frame.  j will be our counter
4754         * for what cookie we are at and i will be
4755         * the total cookiec. k is the current frame
4756         */
```

```
4757         for (k = 1; k <= frames; k++) {
4758                 for (l = 1; (l <= (sgemax + 1)) && (j <= i); j++, l++) {

4760                         /*
4761                          * If we have reached the end of frame
4762                          * and we have more SGE's to fill in
4763                          * we have to fill the final entry
4764                          * with a chain element and then
4765                          * continue to the next frame
4766                          */
4767                         if ((l == (sgemax + 1)) && (k != frames)) {
4768                                 sgechain = (pMpi2SGEChain64_t)sge;
4769                                 j--;
4770                                 chainflags = (
4771                                     MPI2_SGE_FLAGS_CHAIN_ELEMENT |
4772                                     MPI2_SGE_FLAGS_SYSTEM_ADDRESS |
4773                                     MPI2_SGE_FLAGS_64_BIT_ADDRESSING);
4774                                 ddi_put8(p->m_acc_hdl,
4775                                     &sgechain->Flags, chainflags);
4776                                 /*
4777                                  * k is the frame counter and (k + 1)
4778                                  * is the number of the next frame.
4779                                  * Note that frames are in contiguous
4780                                  * memory space.
4781                                  */
4782                                 nframe_phys_addr = p->m_phys_addr +
4783                                     (mpt->m_req_frame_size * k);
4784 #endif /* ! codereview */
4785                                 ddi_put32(p->m_acc_hdl,
4786                                     &sgechain->Address.Low,
4787                                     nframe_phys_addr&0xffffffffull);
3370                                         (p->m_phys_addr +
3371                                         (mpt->m_req_frame_size * k)));
4788                                 ddi_put32(p->m_acc_hdl,
4789                                     &sgechain->Address.High,
4790                                     nframe_phys_addr>>32);
3373                                         &sgechain->Address.High, 0);

4792                                 /*
4793                                  * If there are more than 2 frames left
4794                                  * we have to next chain offset to
4795                                  * the location of the chain element
4796                                  * in the next frame and fill in the
4797                                  * length of the next chain
4798                                  */
4799                                 if ((frames - k) >= 2) {
4800                                         ddi_put8(p->m_acc_hdl,
4801                                             &sgechain->NextChainOffset,
4802                                             (sgemax *
4803                                             sizeof (MPI2_SGE_SIMPLE64))
4804                                             >> 2);
4805                                         ddi_put16(p->m_acc_hdl,
4806                                             &sgechain->Length,
4807                                             mpt->m_req_frame_size /
4808                                             sizeof (MPI2_SGE_SIMPLE64) *
4809                                             sizeof (MPI2_SGE_SIMPLE64));
4810                                 } else {
4811                                         /*
4812                                          * This is the last frame. Set
4813                                          * the NextChainOffset to 0 and
4814                                          * Length is the total size of
4815                                          * all remaining simple elements
4816                                          */
4817                                         ddi_put8(p->m_acc_hdl,
4818                                             &sgechain->NextChainOffset,
4819                                             0);
```



```
4820                                         ddi_put16(p->m_acc_hdl,
4821                                             &sgechain->Length,
4822                                             (cookiec - j) *
4823                                             sizeof (MPI2_SGE_SIMPLE64));
4824                                 }

4826                                 /* Jump to the next frame */
4827                                 sge = (pMpi2SGESimple64_t)
4828                                     ((char *)p->m_frames_addr +
4829                                     (int)mpt->m_req_frame_size * k);

4831                                 continue;
4832                         }

4834                         ddi_put32(p->m_acc_hdl,
4835                             &sge->Address.Low,
4836                             dmap->addr.address64.Low);
4837                         ddi_put32(p->m_acc_hdl,
4838                             &sge->Address.High,
4839                             dmap->addr.address64.High);
4840                         ddi_put32(p->m_acc_hdl,
4841                             &sge->FlagsLength, dmap->count);
4842                         flags = ddi_get32(p->m_acc_hdl,
4843                             &sge->FlagsLength);
4844                         flags |= ((uint32_t)(
4845                             MPI2_SGE_FLAGS_SIMPLE_ELEMENT |
4846                             MPI2_SGE_FLAGS_SYSTEM_ADDRESS |
4847                             MPI2_SGE_FLAGS_64_BIT_ADDRESSING) <<
4848                             MPI2_SGE_FLAGS_SHIFT);

4850                         /*
4851                          * If we are at the end of the frame and
4852                          * there is another frame to fill in
4853                          * we set the last simple element as last
4854                          * element
4855                          */
4856                         if ((l == sgemax) && (k != frames)) {
4857                                 flags |= ((uint32_t)
4858                                     (MPI2_SGE_FLAGS_LAST_ELEMENT) <<
4859                                     MPI2_SGE_FLAGS_SHIFT);
4860                         }

4862                         /*
4863                          * If this is the final cookie we
4864                          * indicate it by setting the flags
4865                          */
4866                         if (j == i) {
4867                                 flags |= ((uint32_t)
4868                                     (MPI2_SGE_FLAGS_LAST_ELEMENT |
4869                                     MPI2_SGE_FLAGS_END_OF_BUFFER |
4870                                     MPI2_SGE_FLAGS_END_OF_LIST) <<
4871                                     MPI2_SGE_FLAGS_SHIFT);
4872                         }
4873                         if (cmd->cmd_flags & CFLAG_DMASEND) {
4874                                 flags |=
4875                                     (MPI2_SGE_FLAGS_HOST_TO_IOC <<
4876                                     MPI2_SGE_FLAGS_SHIFT);
4877                         } else {
4878                                 flags |=
4879                                     (MPI2_SGE_FLAGS_IOC_TO_HOST <<
4880                                     MPI2_SGE_FLAGS_SHIFT);
4881                         }
4882                         ddi_put32(p->m_acc_hdl,
4883                             &sge->FlagsLength, flags);
4884                         dmap++;
4885                         sge++;
```

```
4886                 }
4887         }

4889         /*
4890          * Sync DMA with the chain buffers that were just created
4891          */
4892         (void) ddi_dma_sync(p->m_dma_hdl, 0, 0, DDI_DMA_SYNC_FORDEV);
4476         }
4893 }

4895 static void
4896 mptsas_ieee_sge_mainframe(mptsas_cmd_t *cmd, pMpi2SCSIIORequest_t frame,
4897     ddi_acc_handle_t acc_hdl, uint_t cookiec,
4898     uint8_t end_flag)
3479 /*
3480  * Interrupt handling
3481  * Utility routine.  Poll for status of a command sent to HBA
3482  * without interrupts (a FLAG_NOINTR command).
3483  */
3484 int
3485 mptsas_poll(mptsas_t *mpt, mptsas_cmd_t *poll_cmd, int polltime)
4899 {
4900         pMpi2IeeeSgeSimple64_t  ieeesge;
4901         mptti_t                 *dmap;
4902         uint8_t                 flags;

4904         dmap = cmd->cmd_sg;
4487         int     rval = TRUE;

4906         NDBG1(("mptsas_ieee_sge_mainframe: cookiec=%d, %s", cookiec,
4907             cmd->cmd_flags & CFLAG_DMASEND?"Out":"In"));
4489         NDBG5(("mptsas_poll: cmd=0x%p", (void *)poll_cmd));

4909         ieeesge = (pMpi2IeeeSgeSimple64_t)(&frame->SGL);
4910         while (cookiec--) {
4911                 ddi_put32(acc_hdl, &ieeesge->Address.Low,
4912                     dmap->addr.address64.Low);
4913                 ddi_put32(acc_hdl, &ieeesge->Address.High,
4914                     dmap->addr.address64.High);
4915                 ddi_put32(acc_hdl, &ieeesge->Length, dmap->count);
4916                 NDBG1(("mptsas_ieee_sge_mainframe: len=%d, high=0x%x",
4917                     dmap->count, dmap->addr.address64.High));
4918                 flags = (MPI2_IEEE_SGE_FLAGS_SIMPLE_ELEMENT |
4919                     MPI2_IEEE_SGE_FLAGS_SYSTEM_ADDR);
3491         if ((poll_cmd->cmd_flags & CFLAG_TM_CMD) == 0) {
3492                 mptsas_restart_hba(mpt);
3493         }

4921                 /*
4922                  * If this is the last cookie, we set the flags
4923                  * to indicate so
3496          * Wait, using drv_usecwait(), long enough for the command to
3497          * reasonably return from the target if the target isn't
3498          * "dead".  A polled command may well be sent from scsi_poll, and
3499          * there are retries built in to scsi_poll if the transport
3500          * accepted the packet (TRAN_ACCEPT).  scsi_poll waits 1 second
3501          * and retries the transport up to scsi_poll_busycnt times
3502          * (currently 60) if
3503          * 1. pkt_reason is CMD_INCOMPLETE and pkt_state is 0, or
3504          * 2. pkt_reason is CMD_CMPLT and *pkt_scbp has STATUS_BUSY
3505          *
3506          * limit the waiting to avoid a hang in the event that the
3507          * cmd never gets started but we are still receiving interrupts
4924                  */
4925                 if (cookiec == 0) {
4926                         flags |= end_flag;
```



```
3509         while (!(poll_cmd->cmd_flags & CFLAG_FINISHED)) {
3510                 if (mptsas_wait_intr(mpt, polltime) == FALSE) {
3511                         NDBG5(("mptsas_poll: command incomplete"));
3512                         rval = FALSE;
3513                         break;
3514                 }
4927                 }

3517         if (rval == FALSE) {

4929                 /*
4930                  * XXX: Hmmm, what about the direction based on
4931                  * cmd->cmd_flags & CFLAG_DMASEND?
3520                  * this isn't supposed to happen, the hba must be wedged
3521                  * Mark this cmd as a timeout.
4932                  */
4933                 ddi_put8(acc_hdl, &ieeesge->Flags, flags);
4934                 dmap++;
4935                 ieeesge++;
4936         }
4937 }

4939 static void
4940 mptsas_ieee_sge_chain(mptsas_t *mpt, mptsas_cmd_t *cmd,
4941     pMpi2SCSIIORequest_t frame, ddi_acc_handle_t acc_hdl)
4942 {
4943         pMpi2IeeeSgeSimple64_t  ieeesge;
4944         pMpi25IeeeSgeChain64_t  ieeesgechain;
4945         uint64_t                nframe_phys_addr;
4946         uint_t                  cookiec;
4947         mptti_t                 *dmap;
4948         uint8_t                 flags;
4949         int                     i, j, k, l, frames, sgemax;
4950         int                     temp, maxframe_sges;
4951         uint8_t                 chainflags;
4952         uint32_t                chainlength;
4953         mptsas_cache_frames_t   *p;

4955         cookiec = cmd->cmd_cookiec;

4957         NDBG1(("mptsas_ieee_sge_chain: cookiec=%d", cookiec));

4959         /*
4960          * Hereby we start to deal with multiple frames.
4961          * The process is as follows:
4962          * 1. Determine how many frames are needed for SGL element
4963          *    storage; Note that all frames are stored in contiguous
4964          *    memory space and in 64-bit DMA mode each element is
4965          *    4 double-words (16 bytes) long.
4966          * 2. Fill up the main frame. We need to do this separately
4967          *    since it contains the SCSI IO request header and needs
4968          *    dedicated processing. Note that the last 4 double-words
4969          *    of the SCSI IO header is for SGL element storage
4970          *    (MPI2_SGE_IO_UNION).
4971          * 3. Fill the chain element in the main frame, so the DMA
4972          *    engine can use the following frames.
4973          * 4. Enter a loop to fill the remaining frames. Note that the
4974          *    last frame contains no chain element.  The remaining
4975          *    frames go into the mpt SGL buffer allocated on the fly,
4976          *    not immediately following the main message frame, as in
4977          *    Gen1.
4978          * Some restrictions:
4979          * 1. For 64-bit DMA, the simple element and chain element
4980          *    are both of 4 double-words (16 bytes) in size, even
4981          *    though all frames are stored in the first 4G of mem
4982          *    range and the higher 32-bits of the address are always 0.
```

```
4983          * 2. On some controllers (like the 1064/1068), a frame can
4984          *    hold SGL elements with the last 1 or 2 double-words
4985          *    (4 or 8 bytes) un-used. On these controllers, we should
4986          *    recognize that there's not enough room for another SGL
4987          *    element and move the sge pointer to the next frame.
4988          */

4990         /*
4991          * Sgemax is the number of SGE's that will fit
4992          * each extra frame and frames is total
4993          * number of frames we'll need.  1 sge entry per
4994          * frame is reseverd for the chain element thus the -1 below.
4995          */
4996         sgemax = ((mpt->m_req_frame_size / sizeof (MPI2_IEEE_SGE_SIMPLE64))
4997             - 1);
4998         maxframe_sges = MPTSAS_MAX_FRAME_SGES64(mpt);
4999         temp = (cookiec - (maxframe_sges - 1)) / sgemax;

5001         /*
5002          * A little check to see if we need to round up the number
5003          * of frames we need
5004          */
5005         if ((cookiec - (maxframe_sges - 1)) - (temp * sgemax) > 1) {
5006                 frames = (temp + 1);
5007         } else {
5008                 frames = temp;
5009         }
5010         NDBG1(("mptsas_ieee_sge_chain: temp=%d, frames=%d", temp, frames));
5011         dmap = cmd->cmd_sg;
5012         ieeesge = (pMpi2IeeeSgeSimple64_t)(&frame->SGL);

5014         /*
5015          * First fill in the main frame
5016          */
5017         j = maxframe_sges - 1;
5018         mptsas_ieee_sge_mainframe(cmd, frame, acc_hdl, j, 0);
5019         dmap += j;
5020         ieeesge += j;
5021         j++;

5023         /*
5024          * Fill in the chain element in the main frame.
5025          * About calculation on ChainOffset:
5026          * 1. Struct msg_scsi_io_request has 4 double-words (16 bytes)
5027          *    in the end reserved for SGL element storage
5028          *    (MPI2_SGE_IO_UNION); we should count it in our
5029          *    calculation.  See its definition in the header file.
5030          * 2. Constant j is the counter of the current SGL element
5031          *    that will be processed, and (j - 1) is the number of
5032          *    SGL elements that have been processed (stored in the
5033          *    main frame).
5034          * 3. ChainOffset value should be in units of quad-words (16
5035          *    bytes) so the last value should be divided by 16.
5036          */
5037         ddi_put8(acc_hdl, &frame->ChainOffset,
5038             (sizeof (MPI2_SCSI_IO_REQUEST) -
5039             sizeof (MPI2_SGE_IO_UNION) +
5040             (j - 1) * sizeof (MPI2_IEEE_SGE_SIMPLE64)) >> 4);
5041         ieeesgechain = (pMpi25IeeeSgeChain64_t)ieeesge;
5042         chainflags = (MPI2_IEEE_SGE_FLAGS_CHAIN_ELEMENT |
5043             MPI2_IEEE_SGE_FLAGS_SYSTEM_ADDR);
5044         ddi_put8(acc_hdl, &ieeesgechain->Flags, chainflags);

5046         /*
5047          * The size of the next frame is the accurate size of space
5048          * (in bytes) used to store the SGL elements. j is the counter
```



```
5049          * of SGL elements. (j - 1) is the number of SGL elements that
5050          * have been processed (stored in frames).
5051          */
5052         if (frames >= 2) {
5053                 chainlength = mpt->m_req_frame_size /
5054                     sizeof (MPI2_IEEE_SGE_SIMPLE64) *
5055                     sizeof (MPI2_IEEE_SGE_SIMPLE64);
5056         } else {
5057                 chainlength = ((cookiec - (j - 1)) *
5058                     sizeof (MPI2_IEEE_SGE_SIMPLE64));
5059         }

5061         p = cmd->cmd_extra_frames;

5063         ddi_put32(acc_hdl, &ieeesgechain->Length, chainlength);
5064         ddi_put32(acc_hdl, &ieeesgechain->Address.Low,
5065             p->m_phys_addr&0xffffffffull);
5066         ddi_put32(acc_hdl, &ieeesgechain->Address.High, p->m_phys_addr>>32);

5068         /*
5069          * If there are more than 2 frames left we have to
5070          * fill in the next chain offset to the location of
5071          * the chain element in the next frame.
5072          * sgemax is the number of simple elements in an extra
5073          * frame. Note that the value NextChainOffset should be
5074          * in double-words (4 bytes).
5075          */
5076         if (frames >= 2) {
5077                 ddi_put8(acc_hdl, &ieeesgechain->NextChainOffset,
5078                     (sgemax * sizeof (MPI2_IEEE_SGE_SIMPLE64)) >> 4);
5079         } else {
5080                 ddi_put8(acc_hdl, &ieeesgechain->NextChainOffset, 0);
5081         }

5083         /*
5084          * Jump to next frame;
5085          * Starting here, chain buffers go into the per command SGL.
5086          * This buffer is allocated when chain buffers are needed.
5087          */
5088         ieeesge = (pMpi2IeeeSgeSimple64_t)p->m_frames_addr;
5089         i = cookiec;

5091         /*
5092          * Start filling in frames with SGE's.  If we
5093          * reach the end of frame and still have SGE's
5094          * to fill we need to add a chain element and
5095          * use another frame.  j will be our counter
5096          * for what cookie we are at and i will be
5097          * the total cookiec. k is the current frame
5098          */
5099         for (k = 1; k <= frames; k++) {
5100                 for (l = 1; (l <= (sgemax + 1)) && (j <= i); j++, l++) {

5102                         /*
5103                          * If we have reached the end of frame
5104                          * and we have more SGE's to fill in
5105                          * we have to fill the final entry
5106                          * with a chain element and then
5107                          * continue to the next frame
5108                          */
5109                         if ((l == (sgemax + 1)) && (k != frames)) {
5110                                 ieeesgechain = (pMpi25IeeeSgeChain64_t)ieeesge;
5111                                 j--;
5112                                 chainflags =
5113                                     MPI2_IEEE_SGE_FLAGS_CHAIN_ELEMENT |
5114                                     MPI2_IEEE_SGE_FLAGS_SYSTEM_ADDR;
```

```
5115                                    ddi_put8(p->m_acc_hdl,
5116                                        &ieeesgechain->Flags, chainflags);
5117                                    /*
5118                                     * k is the frame counter and (k + 1)
5119                                     * is the number of the next frame.
5120                                     * Note that frames are in contiguous
5121                                     * memory space.
5122                                     */
5123                                    nframe_phys_addr = p->m_phys_addr +
5124                                        (mpt->m_req_frame_size * k);
5125                                    ddi_put32(p->m_acc_hdl,
5126                                        &ieeesgechain->Address.Low,
5127                                        nframe_phys_addr&0xffffffffull);
5128                                    ddi_put32(p->m_acc_hdl,
5129                                        &ieeesgechain->Address.High,
5130                                        nframe_phys_addr>>32);

5132                                    /*
5133                                     * If there are more than 2 frames left
5134                                     * we have to next chain offset to
5135                                     * the location of the chain element
5136                                     * in the next frame and fill in the
5137                                     * length of the next chain
5138                                     */
5139                                    if ((frames - k) >= 2) {
5140                                            ddi_put8(p->m_acc_hdl,
5141                                                &ieeesgechain->NextChainOffset,
5142                                                (sgemax *
5143                                                sizeof (MPI2_IEEE_SGE_SIMPLE64))
5144                                                >> 4);
5145                                            ddi_put32(p->m_acc_hdl,
5146                                                &ieeesgechain->Length,
5147                                                mpt->m_req_frame_size /
5148                                                sizeof (MPI2_IEEE_SGE_SIMPLE64) *
5149                                                sizeof (MPI2_IEEE_SGE_SIMPLE64));
5150                                    } else {
5151                                            /*
5152                                             * This is the last frame. Set
5153                                             * the NextChainOffset to 0 and
5154                                             * Length is the total size of
5155                                             * all remaining simple elements
5156                                             */
5157                                            ddi_put8(p->m_acc_hdl,
5158                                                &ieeesgechain->NextChainOffset,
5159                                                0);
5160                                            ddi_put32(p->m_acc_hdl,
5161                                                &ieeesgechain->Length,
5162                                                (cookiec - j) *
5163                                                sizeof (MPI2_IEEE_SGE_SIMPLE64));
5164                                    }

5166                                    /* Jump to the next frame */
5167                                    ieeesge = (pMpi2IeeeSgeSimple64_t)
5168                                        ((char *)p->m_frames_addr +
5169                                        (int)mpt->m_req_frame_size * k);

5171                                    continue;
5172                            }

5174                            ddi_put32(p->m_acc_hdl,
5175                                &ieeesge->Address.Low,
5176                                dmap->addr.address64.Low);
5177                            ddi_put32(p->m_acc_hdl,
5178                                &ieeesge->Address.High,
5179                                dmap->addr.address64.High);
5180                            ddi_put32(p->m_acc_hdl,
```



```
5181                                &ieeesge->Length, dmap->count);
5182                            flags = (MPI2_IEEE_SGE_FLAGS_SIMPLE_ELEMENT |
5183                                MPI2_IEEE_SGE_FLAGS_SYSTEM_ADDR);

5185                            /*
5186                             * If we are at the end of the frame and
5187                             * there is another frame to fill in
5188                             * do we need to do anything?
5189                             * if ((l == sgemax) && (k != frames)) {
5190                             * }
5191                             */

5193                            /*
5194                             * If this is the final cookie set end of list.
5195                             */
5196                            if (j == i) {
5197                                    flags |= MPI25_IEEE_SGE_FLAGS_END_OF_LIST;
5198                            }

5200                            ddi_put8(p->m_acc_hdl, &ieeesge->Flags, flags);
5201                            dmap++;
5202                            ieeesge++;
5203                    }
5204            }

5206            /*
5207             * Sync DMA with the chain buffers that were just created
5208             */
5209            (void) ddi_dma_sync(p->m_dma_hdl, 0, 0, DDI_DMA_SYNC_FORDEV);
5210    }

5212    static void
5213    mptsas_sge_setup(mptsas_t *mpt, mptsas_cmd_t *cmd, uint32_t *control,
5214        pMpi2SCSIIORequest_t frame, ddi_acc_handle_t acc_hdl)
5215    {
5216            ASSERT(cmd->cmd_flags & CFLAG_DMAVALID);

5218            NDBG1(("mptsas_sge_setup: cookiec=%d", cmd->cmd_cookiec));

5220            /*
5221             * Set read/write bit in control.
5222             */
5223            if (cmd->cmd_flags & CFLAG_DMASEND) {
5224                    *control |= MPI2_SCSIIO_CONTROL_WRITE;
5225            } else {
5226                    *control |= MPI2_SCSIIO_CONTROL_READ;
5227            }

5229            ddi_put32(acc_hdl, &frame->DataLength, cmd->cmd_dmacount);

5231            /*
5232             * We have 4 cases here.  First where we can fit all the
5233             * SG elements into the main frame, and the case
5234             * where we can't. The SG element is also different when using
5235             * MPI2.5 interface.
5236             * If we have more cookies than we can attach to a frame
5237             * we will need to use a chain element to point
5238             * a location of memory where the rest of the S/G
5239             * elements reside.
5240             */
5241            if (cmd->cmd_cookiec <= MPTSAS_MAX_FRAME_SGES64(mpt)) {
5242                    if (mpt->m_MPI25) {
5243                            mptsas_ieee_sge_mainframe(cmd, frame, acc_hdl,
5244                                cmd->cmd_cookiec,
5245                                MPI25_IEEE_SGE_FLAGS_END_OF_LIST);
5246                    } else {
```

```
5247                        mptsas_sge_mainframe(cmd, frame, acc_hdl,
5248                            cmd->cmd_cookiec,
5249                            ((uint32_t)(MPI2_SGE_FLAGS_LAST_ELEMENT
5250                            | MPI2_SGE_FLAGS_END_OF_BUFFER
5251                            | MPI2_SGE_FLAGS_END_OF_LIST) <<
5252                            MPI2_SGE_FLAGS_SHIFT));
5253                }
5254        } else {
5255                if (mpt->m_MPI25) {
5256                        mptsas_ieee_sge_chain(mpt, cmd, frame, acc_hdl);
5257                } else {
5258                        mptsas_sge_chain(mpt, cmd, frame, acc_hdl);
5259                }
5260        }
5261 }

5263 /*
5264  * Interrupt handling
5265  * Utility routine.  Poll for status of a command sent to HBA
5266  * without interrupts (a FLAG_NOINTR command).
5267  */
5268 int
5269 mptsas_poll(mptsas_t *mpt, mptsas_cmd_t *poll_cmd, int polltime)
5270 {
5271        int             rval = TRUE;
5272        uint32_t        int_mask;

5274        NDBG5(("mptsas_poll: cmd=0x%p, flags 0x%x", (void *)poll_cmd,
5275            poll_cmd->cmd_flags));

5277        /*
5278         * Get the current interrupt mask and disable interrupts.  When
5279         * re-enabling ints, set mask to saved value.
5280         */
5281        int_mask = ddi_get32(mpt->m_datap, &mpt->m_reg->HostInterruptMask);
5282        MPTSAS_DISABLE_INTR(mpt);

5284        mpt->m_polled_intr = 1;

5286        if ((poll_cmd->cmd_flags & CFLAG_TM_CMD) == 0) {
5287                mptsas_restart_hba(mpt);
5288        }

5290        /*
5291         * Wait, using drv_usecwait(), long enough for the command to
5292         * reasonably return from the target if the target isn't
5293         * "dead".  A polled command may well be sent from scsi_poll, and
5294         * there are retries built in to scsi_poll if the transport
5295         * accepted the packet (TRAN_ACCEPT).  scsi_poll waits 1 second
5296         * and retries the transport up to scsi_poll_busycnt times
5297         * (currently 60) if
5298         * 1. pkt_reason is CMD_INCOMPLETE and pkt_state is 0, or
5299         * 2. pkt_reason is CMD_CMPLT and *pkt_scbp has STATUS_BUSY
5300         *
5301         * limit the waiting to avoid a hang in the event that the
5302         * cmd never gets started but we are still receiving interrupts
5303         */
5304        while (!(poll_cmd->cmd_flags & CFLAG_FINISHED)) {
5305                if (mptsas_wait_intr(mpt, polltime) == FALSE) {
5306                        NDBG5(("mptsas_poll: command incomplete"));
5307                        rval = FALSE;
5308                        break;
5309                }
5310        }

5312        if (rval == FALSE) {
```



```
5314                /*
5315                 * this isn't supposed to happen, the hba must be wedged
5316                 * Mark this cmd as a timeout.
5317                 */
5318                mptsas_set_pkt_reason(mpt, poll_cmd, CMD_TIMEOUT,
5319                    (STAT_TIMEOUT|STAT_ABORTED));
3523                mptsas_set_pkt_reason(mpt, poll_cmd, CMD_TIMEOUT,
3524                    (STAT_TIMEOUT|STAT_ABORTED));

5321                if (poll_cmd->cmd_queued == FALSE) {

5323                        NDBG5(("mptsas_poll: not on waitq"));

5325                        poll_cmd->cmd_pkt->pkt_state |=
5326                            (STATE_GOT_BUS|STATE_GOT_TARGET|STATE_SENT_CMD);
5327                } else {

5329                        /* find and remove it from the waitq */
5330                        NDBG5(("mptsas_poll: delete from waitq"));
5331                        mptsas_waitq_delete(mpt, poll_cmd);
5332                }

5334        }
5335        mptsas_fma_check(mpt, poll_cmd);

5337        /*
5338         * Clear polling flag, re-enable interrupts.
5339         */
5340        mpt->m_polled_intr = 0;
5341        ddi_put32(mpt->m_datap, &mpt->m_reg->HostInterruptMask, int_mask);

5343        /*
5344         * If there are queued cmd, start them now.
5345         */
5346        if (mpt->m_waitq != NULL) {
5347                mptsas_restart_waitq(mpt);
5348        }

5350 #endif /* ! codereview */
5351        NDBG5(("mptsas_poll: done"));
5352        return (rval);
5353 }

5355 /*
5356  * Used for polling cmds and TM function
5357  */
5358 static int
5359 mptsas_wait_intr(mptsas_t *mpt, int polltime)
5360 {
5361        int                             cnt, rval = FALSE;
3541        int                             cnt;
5362        pMpi2ReplyDescriptorsUnion_t    reply_desc_union;
5363        mptsas_reply_pqueue_t           *rpqp;
3543        uint32_t                        int_mask;

5365        NDBG5(("mptsas_wait_intr"));
5366        ASSERT(mutex_owned(&mpt->m_mutex));

3547        mpt->m_polled_intr = 1;

5368        /*
5369         * Keep polling for at least (polltime * 1000) seconds
3550         * Get the current interrupt mask and disable interrupts.  When
3551         * re-enabling ints, set mask to saved value.
5370         */
```

```
5371         rpqp = mpt->m_rep_post_queues;
3553         int_mask = ddi_get32(mpt->m_datap, &mpt->m_reg->HostInterruptMask);
3554         MPTSAS_DISABLE_INTR(mpt);

5373         /*
5374          * Drop the main mutex and grab the mutex for reply queue 0
3557          * Keep polling for at least (polltime * 1000) seconds
5375          */
5376         mutex_exit(&mpt->m_mutex);
5377         mutex_enter(&rpqp->rpq_mutex);
5378 #endif /* ! codereview */
5379         for (cnt = 0; cnt < polltime; cnt++) {
5380                 (void) ddi_dma_sync(mpt->m_dma_post_queue_hdl, 0, 0,
5381                     DDI_DMA_SYNC_FORCPU);

5383                 /*
5384                  * Polled requests should only come back through
5385                  * the first interrupt.
5386                  */
5387 #endif /* ! codereview */
5388                 reply_desc_union = (pMpi2ReplyDescriptorsUnion_t)
5389                     MPTSAS_GET_NEXT_REPLY(rpqp, rpqp->rpq_index);
3559                 MPTSAS_GET_NEXT_REPLY(mpt, mpt->m_post_index);

5391                 if (ddi_get32(mpt->m_acc_post_queue_hdl,
5392                     &reply_desc_union->Words.Low) == 0xFFFFFFFF ||
5393                     ddi_get32(mpt->m_acc_post_queue_hdl,
5394                     &reply_desc_union->Words.High) == 0xFFFFFFFF) {
5395                         drv_usecwait(1000);
5396                         continue;
5397                 }

5399                 /*
5400                  * The reply is valid, process it according to its
5401                  * type.
5402                  */
5403                 mptsas_process_intr(mpt, rpqp, reply_desc_union);
3573                 mptsas_process_intr(mpt, reply_desc_union);

5405                 /*
5406                  * Clear the reply descriptor for re-use.
5407                  */
5408                 ddi_put64(mpt->m_acc_post_queue_hdl,
5409                     &((uint64_t *)(void *)rpqp->rpq_queue)[rpqp->rpq_index],
5410                     0xFFFFFFFFFFFFFFFF);
5411                 (void) ddi_dma_sync(mpt->m_dma_post_queue_hdl, 0, 0,
5412                     DDI_DMA_SYNC_FORDEV);

5414                 if (++rpqp->rpq_index == mpt->m_post_queue_depth) {
5415                         rpqp->rpq_index = 0;
3575                 if (++mpt->m_post_index == mpt->m_post_queue_depth) {
3576                         mpt->m_post_index = 0;
5416                 }

5418                 /*
5419                  * Update the reply index
3580                  * Update the global reply index
5420                  */
5421                 ddi_put32(mpt->m_datap,
5422                     &mpt->m_reg->ReplyPostHostIndex, rpqp->rpq_index);
5423                 rval = TRUE;
5424                 break;
5425         }
3583                     &mpt->m_reg->ReplyPostHostIndex, mpt->m_post_index);
3584                 mpt->m_polled_intr = 0;
```



```
5427         mutex_exit(&rpqp->rpq_mutex);
5428         mutex_enter(&mpt->m_mutex);
3586                 /*
3587                  * Re-enable interrupts and quit.
3588                  */
3589                 ddi_put32(mpt->m_datap, &mpt->m_reg->HostInterruptMask,
3590                     int_mask);
3591                 return (TRUE);

5430         return (rval);
3593         }

3595         /*
3596          * Clear polling flag, re-enable interrupts and quit.
3597          */
3598         mpt->m_polled_intr = 0;
3599         ddi_put32(mpt->m_datap, &mpt->m_reg->HostInterruptMask, int_mask);
3600         return (FALSE);
5431 }

5433 static void
5434 mptsas_handle_scsi_io_success(mptsas_t *mpt,
5435     mptsas_reply_pqueue_t *rpqp,
5436 #endif /* ! codereview */
5437     pMpi2ReplyDescriptorsUnion_t reply_desc)
5438 {
5439         pMpi2SCSIIOSuccessReplyDescriptor_t     scsi_io_success;
5440         uint16_t                                SMID;
5441         mptsas_slots_t                          *slots = mpt->m_active;
5442         mptsas_cmd_t                            *cmd = NULL;
5443         struct scsi_pkt                         *pkt;

3605         ASSERT(mutex_owned(&mpt->m_mutex));

5445         scsi_io_success = (pMpi2SCSIIOSuccessReplyDescriptor_t)reply_desc;
5446         SMID = ddi_get16(mpt->m_acc_post_queue_hdl, &scsi_io_success->SMID);

5448         /*
5449          * This is a success reply so just complete the IO.  First, do a sanity
5450          * check on the SMID.  The final slot is used for TM requests, which
5451          * would not come into this reply handler.
5452          */
5453         if ((SMID == 0) || (SMID > slots->m_n_normal)) {
5454                 mptsas_log(mpt, CE_WARN, "?Received invalid SMID of %d\n",
5455                     SMID);
5456                 ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_UNAFFECTED);
5457                 return;
5458         }

5460         cmd = slots->m_slot[SMID];

5462         /*
5463          * print warning and return if the slot is empty
5464          */
5465         if (cmd == NULL) {
5466                 mptsas_log(mpt, CE_WARN, "?NULL command for successful SCSI IO "
5467                     "in slot %d", SMID);
5468                 return;
5469         }
5470         ASSERT(cmd->cmd_rpqidx == rpqp->rpq_num);
5471 #endif /* ! codereview */

5473         pkt = CMD2PKT(cmd);
5474         pkt->pkt_state |= (STATE_GOT_BUS | STATE_GOT_TARGET | STATE_SENT_CMD |
5475             STATE_GOT_STATUS);
5476         if (cmd->cmd_flags & CFLAG_DMAVALID) {
```

```
5477                    pkt->pkt_state |= STATE_XFERRED_DATA;
5478            }
5479            pkt->pkt_resid = 0;

5481            if (cmd->cmd_flags & CFLAG_PASSTHRU) {
5482                    cmd->cmd_flags |= CFLAG_FINISHED;
5483                    cv_broadcast(&mpt->m_passthru_cv);
5484                    return;
5485            }
5486            if (!(cmd->cmd_flags & CFLAG_TM_CMD)) {
5487                    if (cmd->cmd_flags & CFLAG_CMDIOC) {
5488                            mutex_enter(&mpt->m_mutex);
5489                            mptsas_remove_cmd(mpt, cmd);
5490                            mutex_exit(&mpt->m_mutex);
5491 #endif /* ! codereview */
5492                    } else {
5493 #ifdef MPTSAS_DEBUG
5494                            /*
5495                             * In order to test timeout for a command set
5496                             * mptsas_test_timeout via mdb to avoid completion
5497                             * processing here.
5498                             */
5499                            if (mptsas_test_timeout) {
5500                                  mptsas_test_timeout = 0;
5501                                  return;
5502                            }
5503 #endif
5504                            /*
5505                             * This is the normal path, avoid grabbing
5506                             * the m_mutex.
5507                             */
5508                            mptsas_remove_cmd_nomtx(mpt, cmd);
5509                    }
3632                    mptsas_remove_cmd(mpt, cmd);
5510            }

5512            if (cmd->cmd_flags & CFLAG_RETRY) {
5513                    /*
5514                     * The target returned QFULL or busy, do not add tihs
5515                     * pkt to the doneq since the hba will retry
5516                     * this cmd.
5517                     *
5518                     * The pkt has already been resubmitted in
5519                     * mptsas_handle_qfull() or in mptsas_check_scsi_io_error().
5520                     * Remove this cmd_flag here.
5521                     */
5522                    cmd->cmd_flags &= ~CFLAG_RETRY;
5523            } else {
5524                    mptsas_rpdoneq_add(mpt, rpqp, cmd);
3647                    mptsas_doneq_add(mpt, cmd);
5525            }
5526 }

5528 static void
5529 mptsas_handle_address_reply(mptsas_t *mpt,
5530     pMpi2ReplyDescriptorsUnion_t reply_desc)
5531 {
5532            pMpi2AddressReplyDescriptor_t   address_reply;
5533            pMPI2DefaultReply_t             reply;
5534            mptsas_fw_diagnostic_buffer_t   *pBuffer;
5535            uint32_t                        reply_addr, reply_frame_dma_baseaddr;
3658            uint32_t                        reply_addr;
5536            uint16_t                        SMID, iocstatus;
5537            mptsas_slots_t                  *slots = mpt->m_active;
5538            mptsas_cmd_t                    *cmd = NULL;
5539            uint8_t                         function, buffer_type;
```



```
5540            m_replyh_arg_t                  *args;
5541            int                             reply_frame_no;

5543            ASSERT(mutex_owned(&mpt->m_mutex));

5545            address_reply = (pMpi2AddressReplyDescriptor_t)reply_desc;
5546            reply_addr = ddi_get32(mpt->m_acc_post_queue_hdl,
5547                &address_reply->ReplyFrameAddress);
5548            SMID = ddi_get16(mpt->m_acc_post_queue_hdl, &address_reply->SMID);

5550            /*
5551             * If reply frame is not in the proper range we should ignore this
5552             * message and exit the interrupt handler.
5553             */
5554            reply_frame_dma_baseaddr = mpt->m_reply_frame_dma_addr & 0xffffffffu;
5555            if ((reply_addr < reply_frame_dma_baseaddr) ||
5556                (reply_addr >= (reply_frame_dma_baseaddr +
3677            if ((reply_addr < mpt->m_reply_frame_dma_addr) ||
3678                (reply_addr >= (mpt->m_reply_frame_dma_addr +
5557                (mpt->m_reply_frame_size * mpt->m_max_replies))) ||
5558                ((reply_addr - reply_frame_dma_baseaddr) %
3680                ((reply_addr - mpt->m_reply_frame_dma_addr) %
5559                mpt->m_reply_frame_size != 0)) {
5560                    mptsas_log(mpt, CE_WARN, "?Received invalid reply frame "
5561                        "address 0x%x\n", reply_addr);
5562                    ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_UNAFFECTED);
5563                    return;
5564            }

5566            (void) ddi_dma_sync(mpt->m_dma_reply_frame_hdl, 0, 0,
5567                DDI_DMA_SYNC_FORCPU);
5568            reply = (pMPI2DefaultReply_t)(mpt->m_reply_frame + (reply_addr -
5569                reply_frame_dma_baseaddr));
3691                mpt->m_reply_frame_dma_addr));
5570            function = ddi_get8(mpt->m_acc_reply_frame_hdl, &reply->Function);

5572            NDBG31(("mptsas_handle_address_reply: function 0x%x, reply_addr=0x%x",
5573                function, reply_addr));

5575 #endif /* ! codereview */
5576            /*
5577             * don't get slot information and command for events since these values
5578             * don't exist
5579             */
5580            if ((function != MPI2_FUNCTION_EVENT_NOTIFICATION) &&
5581                (function != MPI2_FUNCTION_DIAG_BUFFER_POST)) {
5582                    /*
5583                     * This could be a TM reply, which use the last allocated SMID,
5584                     * so allow for that.
5585                     */
5586                    if ((SMID == 0) || (SMID > (slots->m_n_normal + 1))) {
5587                            mptsas_log(mpt, CE_WARN, "?Received invalid SMID of "
5588                                "%d\n", SMID);
5589                            ddi_fm_service_impact(mpt->m_dip,
5590                                DDI_SERVICE_UNAFFECTED);
5591                            return;
5592                    }

5594                    cmd = slots->m_slot[SMID];

5596                    /*
5597                     * print warning and return if the slot is empty
5598                     */
5599                    if (cmd == NULL) {
5600                            mptsas_log(mpt, CE_WARN, "?NULL command for address "
5601                                "reply in slot %d", SMID);
```

```
5602                        return;
5603                }
5604                if ((cmd->cmd_flags &
5605                    (CFLAG_PASSTHRU | CFLAG_CONFIG | CFLAG_FW_DIAG))) {
3694                if ((cmd->cmd_flags & CFLAG_PASSTHRU) ||
3695                    (cmd->cmd_flags & CFLAG_CONFIG) ||
3696                    (cmd->cmd_flags & CFLAG_FW_DIAG)) {
5606                        cmd->cmd_rfm = reply_addr;
5607                        cmd->cmd_flags |= CFLAG_FINISHED;
5608                        cv_broadcast(&mpt->m_passthru_cv);
5609                        cv_broadcast(&mpt->m_config_cv);
5610                        cv_broadcast(&mpt->m_fw_diag_cv);
5611                        return;
5612                } else if (!(cmd->cmd_flags & CFLAG_FW_CMD)) {
5613                        mptsas_remove_cmd(mpt, cmd);
5614                }
5615                NDBG31(("\t\tmptsas_process_intr: slot=%d", SMID));
5616        }
5617        /*
5618         * Depending on the function, we need to handle
5619         * the reply frame (and cmd) differently.
5620         */
5621        switch (function) {
5622        case MPI2_FUNCTION_SCSI_IO_REQUEST:
5623                mptsas_check_scsi_io_error(mpt, (pMpi2SCSIIOReply_t)reply, cmd);
5624                break;
5625        case MPI2_FUNCTION_SCSI_TASK_MGMT:
5626                cmd->cmd_rfm = reply_addr;
5627                mptsas_check_task_mgt(mpt, (pMpi2SCSIManagementReply_t)reply,
5628                    cmd);
5629                break;
5630        case MPI2_FUNCTION_FW_DOWNLOAD:
5631                cmd->cmd_flags |= CFLAG_FINISHED;
5632                cv_signal(&mpt->m_fw_cv);
5633                break;
5634        case MPI2_FUNCTION_EVENT_NOTIFICATION:
5635                reply_frame_no = (reply_addr - reply_frame_dma_baseaddr) /
3726                reply_frame_no = (reply_addr - mpt->m_reply_frame_dma_addr) /
5636                    mpt->m_reply_frame_size;
5637                args = &mpt->m_replyh_args[reply_frame_no];
5638                args->mpt = (void *)mpt;
5639                args->rfm = reply_addr;

5641                /*
5642                 * Record the event if its type is enabled in
5643                 * this mpt instance by ioctl.
5644                 */
5645                mptsas_record_event(args);

5647                /*
5648                 * Handle time critical events
5649                 * NOT_RESPONDING/ADDED only now
5650                 */
5651                if (mptsas_handle_event_sync(args) == DDI_SUCCESS) {
5652                        /*
5653                         * Would not return main process,
5654                         * just let taskq resolve ack action
5655                         * and ack would be sent in taskq thread
5656                         */
5657                        NDBG20(("send mptsas_handle_event_sync success"));
5658                }

5660                if (mpt->m_in_reset) {
5661                        NDBG20(("dropping event received during reset"));
5662                        return;
5663                }
```



```
5665                if ((ddi_taskq_dispatch(mpt->m_event_taskq, mptsas_handle_event,
5666                    (void *)args, DDI_NOSLEEP)) != DDI_SUCCESS) {
5667                        mptsas_log(mpt, CE_WARN, "No memory available"
5668                        "for dispatch taskq");
5669                        /*
5670                         * Return the reply frame to the free queue.
5671                         */
5672                        ddi_put32(mpt->m_acc_free_queue_hdl,
5673                            &((uint32_t *)(void *)
5674                            mpt->m_free_queue)[mpt->m_free_index], reply_addr);
5675                        (void) ddi_dma_sync(mpt->m_dma_free_queue_hdl, 0, 0,
5676                            DDI_DMA_SYNC_FORDEV);
5677                        if (++mpt->m_free_index == mpt->m_free_queue_depth) {
5678                                mpt->m_free_index = 0;
5679                        }

5681                        ddi_put32(mpt->m_datap,
5682                            &mpt->m_reg->ReplyFreeHostIndex, mpt->m_free_index);
5683                }
5684                return;
5685        case MPI2_FUNCTION_DIAG_BUFFER_POST:
5686                /*
5687                 * If SMID is 0, this implies that the reply is due to a
5688                 * release function with a status that the buffer has been
5689                 * released.  Set the buffer flags accordingly.
5690                 */
5691                if (SMID == 0) {
5692                        iocstatus = ddi_get16(mpt->m_acc_reply_frame_hdl,
5693                            &reply->IOCStatus);
5694                        buffer_type = ddi_get8(mpt->m_acc_reply_frame_hdl,
5695                            &(((pMpi2DiagBufferPostReply_t)reply)->BufferType));
5696                        if (iocstatus == MPI2_IOCSTATUS_DIAGNOSTIC_RELEASED) {
5697                                pBuffer =
5698                                    &mpt->m_fw_diag_buffer_list[buffer_type];
5699                                pBuffer->valid_data = TRUE;
5700                                pBuffer->owned_by_firmware = FALSE;
5701                                pBuffer->immediate = FALSE;
5702                        }
5703                } else {
5704                        /*
5705                         * Normal handling of diag post reply with SMID.
5706                         */
5707                        cmd = slots->m_slot[SMID];

5709                        /*
5710                         * print warning and return if the slot is empty
5711                         */
5712                        if (cmd == NULL) {
5713                                mptsas_log(mpt, CE_WARN, "?NULL command for "
5714                                    "address reply in slot %d", SMID);
5715                                return;
5716                        }
5717                        cmd->cmd_rfm = reply_addr;
5718                        cmd->cmd_flags |= CFLAG_FINISHED;
5719                        cv_broadcast(&mpt->m_fw_diag_cv);
5720                }
5721                return;
5722        default:
5723                mptsas_log(mpt, CE_WARN, "Unknown function 0x%x ", function);
5724                break;
5725        }

5727        /*
5728         * Return the reply frame to the free queue.
5729         */
```

```
5730         ddi_put32(mpt->m_acc_free_queue_hdl,
5731             &((uint32_t *)(void *)mpt->m_free_queue)[mpt->m_free_index],
5732             reply_addr);
5733         (void) ddi_dma_sync(mpt->m_dma_free_queue_hdl, 0, 0,
5734             DDI_DMA_SYNC_FORDEV);
5735         if (++mpt->m_free_index == mpt->m_free_queue_depth) {
5736                 mpt->m_free_index = 0;
5737         }
5738         ddi_put32(mpt->m_datap, &mpt->m_reg->ReplyFreeHostIndex,
5739             mpt->m_free_index);

5741         if (cmd->cmd_flags & CFLAG_FW_CMD)
5742                 return;

5744         if (cmd->cmd_flags & CFLAG_RETRY) {
5745                 /*
5746                  * The target returned QFULL or busy, do not add this
3837                  * The target returned QFULL or busy, do not add tihs
5747                  * pkt to the doneq since the hba will retry
5748                  * this cmd.
5749                  *
5750                  * The pkt has already been resubmitted in
5751                  * mptsas_handle_qfull() or in mptsas_check_scsi_io_error().
5752                  * Remove this cmd_flag here.
5753                  */
5754                 cmd->cmd_flags &= ~CFLAG_RETRY;
5755         } else {
5756                 mptsas_doneq_add(mpt, cmd);
5757         }
5758 }

5760 #ifdef MPTSAS_DEBUG
5761 static uint8_t mptsas_last_sense[256];
5762 #endif

5764 #endif /* ! codereview */
5765 static void
5766 mptsas_check_scsi_io_error(mptsas_t *mpt, pMpi2SCSIIOReply_t reply,
5767     mptsas_cmd_t *cmd)
5768 {
5769         uint8_t                 scsi_status, scsi_state;
5770         uint16_t                ioc_status, cmd_rqs_len;
3851         uint16_t                ioc_status;
5771         uint32_t                xferred, sensecount, responsedata, loginfo = 0;
5772         struct scsi_pkt         *pkt;
5773         struct scsi_arq_status  *arqstat;
3855         struct buf              *bp;
5774         mptsas_target_t         *ptgt = cmd->cmd_tgt_addr;
5775         uint8_t                 *sensedata = NULL;
5776         uint64_t                sas_wwn;
5777         uint8_t                 phy;
5778         char                    wwn_str[MPTSAS_WWN_STRLEN];

3859         if ((cmd->cmd_flags & (CFLAG_SCBEXTERN | CFLAG_EXTARQBUFVALID)) ==
3860             (CFLAG_SCBEXTERN | CFLAG_EXTARQBUFVALID)) {
3861                 bp = cmd->cmd_ext_arq_buf;
3862         } else {
3863                 bp = cmd->cmd_arq_buf;
3864         }

5780         scsi_status = ddi_get8(mpt->m_acc_reply_frame_hdl, &reply->SCSIStatus);
5781         ioc_status = ddi_get16(mpt->m_acc_reply_frame_hdl, &reply->IOCStatus);
5782         scsi_state = ddi_get8(mpt->m_acc_reply_frame_hdl, &reply->SCSIState);
5783         xferred = ddi_get32(mpt->m_acc_reply_frame_hdl, &reply->TransferCount);
5784         sensecount = ddi_get32(mpt->m_acc_reply_frame_hdl, &reply->SenseCount);
5785         responsedata = ddi_get32(mpt->m_acc_reply_frame_hdl,
```



```
5786             &reply->ResponseInfo);

5788         if (ioc_status & MPI2_IOCSTATUS_FLAG_LOG_INFO_AVAILABLE) {
5789                 sas_wwn = ptgt->m_addr.mta_wwn;
5790                 phy = ptgt->m_phynum;
5791                 if (sas_wwn == 0) {
5792                         (void) sprintf(wwn_str, "p%x", phy);
5793                 } else {
5794                         (void) sprintf(wwn_str, "w%016"PRIx64, sas_wwn);
5795                 }
5796 #endif /* ! codereview */
5797                 loginfo = ddi_get32(mpt->m_acc_reply_frame_hdl,
5798                     &reply->IOCLogInfo);
5799                 mptsas_log(mpt, CE_NOTE,
5800                     "?Log info 0x%x received for target %d %s.\n"
3875                     "?Log info 0x%x received for target %d.\n"
5801                     "\tscsi_status=0x%x, ioc_status=0x%x, scsi_state=0x%x",
5802                     loginfo, Tgt(cmd), wwn_str, scsi_status, ioc_status,
3877                     loginfo, Tgt(cmd), scsi_status, ioc_status,
5803                     scsi_state);
5804         }

5806         NDBG31(("\t\tscsi_status=0x%x, ioc_status=0x%x, scsi_state=0x%x",
5807             scsi_status, ioc_status, scsi_state));

5809         pkt = CMD2PKT(cmd);
5810         *(pkt->pkt_scbp) = scsi_status;

5812         if (loginfo == 0x31170000) {
5813                 /*
5814                  * if loginfo PL_LOGINFO_CODE_IO_DEVICE_MISSING_DELAY_RETRY
5815                  * 0x31170000 comes, that means the device missing delay
5816                  * is in progressing, the command need retry later.
5817                  */
5818                 *(pkt->pkt_scbp) = STATUS_BUSY;
5819                 return;
5820         }

5822         if ((scsi_state & MPI2_SCSI_STATE_NO_SCSI_STATUS) &&
5823             ((ioc_status & MPI2_IOCSTATUS_MASK) ==
5824             MPI2_IOCSTATUS_SCSI_DEVICE_NOT_THERE)) {
5825                 pkt->pkt_reason = CMD_INCOMPLETE;
5826                 pkt->pkt_state |= STATE_GOT_BUS;
5827                 mutex_enter(&ptgt->m_t_mutex);
5828 #endif /* ! codereview */
5829                 if (ptgt->m_reset_delay == 0) {
5830                         mptsas_set_throttle(mpt, ptgt,
5831                             DRAIN_THROTTLE);
5832                 }
5833                 mutex_exit(&ptgt->m_t_mutex);
5834 #endif /* ! codereview */
5835                 return;
5836         }

5838         if (scsi_state & MPI2_SCSI_STATE_RESPONSE_INFO_VALID) {
5839                 responsedata &= 0x000000FF;
5840                 if (responsedata & MPTSAS_SCSI_RESPONSE_CODE_TLR_OFF) {
5841                         mptsas_log(mpt, CE_NOTE, "Do not support the TLR\n");
5842                         pkt->pkt_reason = CMD_TLR_OFF;
5843                         return;
5844                 }
5845         }


5848         switch (scsi_status) {
5849         case MPI2_SCSI_STATUS_CHECK_CONDITION:
```

```
5850                 (void) ddi_dma_sync(mpt->m_dma_req_sense_hdl, 0, 0,
5851                     DDI_DMA_SYNC_FORCPU);
5852 #endif /* ! codereview */
5853                 pkt->pkt_resid = (cmd->cmd_dmacount - xferred);
5854                 arqstat = (void*)(pkt->pkt_scbp);
5855                 arqstat->sts_rqpkt_status = *((struct scsi_status *)
5856                     (pkt->pkt_scbp));
5857                 pkt->pkt_state |= (STATE_GOT_BUS | STATE_GOT_TARGET |
5858                     STATE_SENT_CMD | STATE_GOT_STATUS | STATE_ARQ_DONE);
5859                 if (cmd->cmd_flags & CFLAG_XARQ) {
5860                         pkt->pkt_state |= STATE_XARQ_DONE;
5861                 }
5862                 if (pkt->pkt_resid != cmd->cmd_dmacount) {
5863                         pkt->pkt_state |= STATE_XFERRED_DATA;
5864                 }
5865                 arqstat->sts_rqpkt_reason = pkt->pkt_reason;
5866                 arqstat->sts_rqpkt_state  = pkt->pkt_state;
5867                 arqstat->sts_rqpkt_state |= STATE_XFERRED_DATA;
5868                 arqstat->sts_rqpkt_statistics = pkt->pkt_statistics;
5869                 sensedata = (uint8_t *)&arqstat->sts_sensedata;
5870 #ifdef MPTSAS_DEBUG
5871                 bcopy((uchar_t *)cmd->cmd_arq_buf, mptsas_last_sense,
5872                     cmd->cmd_rqslen);
5873 #endif
5874                 if (cmd->cmd_extrqslen != 0) {
5875                         cmd_rqs_len = cmd->cmd_extrqslen;
5876                 } else {
5877                         cmd_rqs_len = cmd->cmd_rqslen;
5878                 }
5879                 bcopy((uchar_t *)cmd->cmd_arq_buf, sensedata,

3903                 bcopy((uchar_t *)bp->b_un.b_addr, sensedata,
5880                     ((cmd->cmd_rqslen >= sensecount) ? sensecount :
5881                     cmd_rqs_len));
5882                 arqstat->sts_rqpkt_resid = (cmd_rqs_len - sensecount);
3905                     cmd->cmd_rqslen));
3906                 arqstat->sts_rqpkt_resid = (cmd->cmd_rqslen - sensecount);
5883                 cmd->cmd_flags |= CFLAG_CMDARQ;
5884                 /*
5885                  * Set proper status for pkt if autosense was valid
5886                  */
5887                 if (scsi_state & MPI2_SCSI_STATE_AUTOSENSE_VALID) {
5888                         struct scsi_status zero_status = { 0 };
5889                         arqstat->sts_rqpkt_status = zero_status;
5890                 }

5892                 /*
5893                  * ASC=0x47 is parity error
5894                  * ASC=0x48 is initiator detected error received
5895                  */
5896                 if ((scsi_sense_key(sensedata) == KEY_ABORTED_COMMAND) &&
5897                     ((scsi_sense_asc(sensedata) == 0x47) ||
5898                     (scsi_sense_asc(sensedata) == 0x48))) {
5899                         mptsas_log(mpt, CE_NOTE, "Aborted_command!");
5900                 }

5902                 /*
5903                  * ASC/ASCQ=0x3F/0x0E means report_luns data changed
5904                  * ASC/ASCQ=0x25/0x00 means invalid lun
5905                  */
5906                 if (((scsi_sense_key(sensedata) == KEY_UNIT_ATTENTION) &&
5907                     (scsi_sense_asc(sensedata) == 0x3F) &&
5908                     (scsi_sense_ascq(sensedata) == 0x0E)) ||
5909                     ((scsi_sense_key(sensedata) == KEY_ILLEGAL_REQUEST) &&
5910                     (scsi_sense_asc(sensedata) == 0x25) &&
5911                     (scsi_sense_ascq(sensedata) == 0x00))) {
```



```
5912                         mptsas_topo_change_list_t *topo_node = NULL;

5914                         topo_node = kmem_zalloc(
5915                             sizeof (mptsas_topo_change_list_t),
5916                             KM_NOSLEEP);
5917                         if (topo_node == NULL) {
5918                                 mptsas_log(mpt, CE_NOTE, "No memory"
5919                                     "resource for handle SAS dynamic"
5920                                     "reconfigure.\n");
5921                                 break;
5922                         }
5923                         topo_node->mpt = mpt;
5924                         topo_node->event = MPTSAS_DR_EVENT_RECONFIG_TARGET;
5925                         topo_node->un.phymask = ptgt->m_addr.mta_phymask;
5926                         topo_node->devhdl = ptgt->m_devhdl;
5927                         topo_node->object = (void *)ptgt;
5928                         topo_node->flags = MPTSAS_TOPO_FLAG_LUN_ASSOCIATED;

5930                         if ((ddi_taskq_dispatch(mpt->m_dr_taskq,
5931                             mptsas_handle_dr,
5932                             (void *)topo_node,
5933                             DDI_NOSLEEP)) != DDI_SUCCESS) {
5934                                 kmem_free(topo_node,
5935                                     sizeof (mptsas_topo_change_list_t));
5936 #endif /* ! codereview */
5937                                 mptsas_log(mpt, CE_NOTE, "mptsas start taskq"
5938                                     "for handle SAS dynamic reconfigure"
5939                                     "failed. \n");
5940                         }
5941                 }
5942                 break;
5943         case MPI2_SCSI_STATUS_GOOD:
5944                 switch (ioc_status & MPI2_IOCSTATUS_MASK) {
5945                 case MPI2_IOCSTATUS_SCSI_DEVICE_NOT_THERE:
5946                         pkt->pkt_reason = CMD_DEV_GONE;
5947                         pkt->pkt_state |= STATE_GOT_BUS;
5948                         mutex_enter(&ptgt->m_t_mutex);
5949 #endif /* ! codereview */
5950                         if (ptgt->m_reset_delay == 0) {
5951                                 mptsas_set_throttle(mpt, ptgt, DRAIN_THROTTLE);
5952                         }
5953                         mutex_exit(&ptgt->m_t_mutex);
5954 #endif /* ! codereview */
5955                         NDBG31(("lost disk for target%d, command:%x",
5956                             Tgt(cmd), pkt->pkt_cdbp[0]));
5957                         break;
5958                 case MPI2_IOCSTATUS_SCSI_DATA_OVERRUN:
5959                         NDBG31(("data overrun: xferred=%d", xferred));
5960                         NDBG31(("dmacount=%d", cmd->cmd_dmacount));
5961                         pkt->pkt_reason = CMD_DATA_OVR;
5962                         pkt->pkt_state |= (STATE_GOT_BUS | STATE_GOT_TARGET
5963                             | STATE_SENT_CMD | STATE_GOT_STATUS
5964                             | STATE_XFERRED_DATA);
5965                         pkt->pkt_resid = 0;
5966                         break;
5967                 case MPI2_IOCSTATUS_SCSI_RESIDUAL_MISMATCH:
5968                 case MPI2_IOCSTATUS_SCSI_DATA_UNDERRUN:
5969                         NDBG31(("data underrun: xferred=%d", xferred));
5970                         NDBG31(("dmacount=%d", cmd->cmd_dmacount));
5971                         pkt->pkt_state |= (STATE_GOT_BUS | STATE_GOT_TARGET
5972                             | STATE_SENT_CMD | STATE_GOT_STATUS);
5973                         pkt->pkt_resid = (cmd->cmd_dmacount - xferred);
5974                         if (pkt->pkt_resid != cmd->cmd_dmacount) {
5975                                 pkt->pkt_state |= STATE_XFERRED_DATA;
5976                         }
5977                         break;
```

```
5978                case MPI2_IOCSTATUS_SCSI_TASK_TERMINATED:
5979                        if (cmd->cmd_active_expiration <= gethrtime()) {
5980                                /*
5981                                 * When timeout requested, propagate
5982                                 * proper reason and statistics to
5983                                 * target drivers.
5984                                 */
5985                                mptsas_set_pkt_reason(mpt, cmd, CMD_TIMEOUT,
5986                                    STAT_BUS_RESET | STAT_TIMEOUT);
5987                        } else {
5988                                mptsas_set_pkt_reason(mpt, cmd, CMD_RESET,
5989                                    STAT_BUS_RESET);
5990                        }
3958                    mptsas_set_pkt_reason(mpt,
3959                        cmd, CMD_RESET, STAT_BUS_RESET);
5991                        break;
5992                case MPI2_IOCSTATUS_SCSI_IOC_TERMINATED:
5993                case MPI2_IOCSTATUS_SCSI_EXT_TERMINATED:
5994                        mptsas_set_pkt_reason(mpt,
5995                            cmd, CMD_RESET, STAT_DEV_RESET);
5996                        break;
5997                case MPI2_IOCSTATUS_SCSI_IO_DATA_ERROR:
5998                case MPI2_IOCSTATUS_SCSI_PROTOCOL_ERROR:
5999                        pkt->pkt_state |= (STATE_GOT_BUS | STATE_GOT_TARGET);
6000                        mptsas_set_pkt_reason(mpt,
6001                            cmd, CMD_TERMINATED, STAT_TERMINATED);
6002                        break;
6003                case MPI2_IOCSTATUS_INSUFFICIENT_RESOURCES:
6004                case MPI2_IOCSTATUS_BUSY:
6005                        /*
6006                         * set throttles to drain
6007                         */
6008                        for (ptgt = refhash_first(mpt->m_targets); ptgt != NULL;
6009                            ptgt = refhash_next(mpt->m_targets, ptgt)) {
6010                                mptsas_set_throttle_mtx(mpt, ptgt,
6011                                    DRAIN_THROTTLE);
3979                                mptsas_set_throttle(mpt, ptgt, DRAIN_THROTTLE);
6012                        }

6014                        /*
6015                         * retry command
6016                         */
6017                        mptsas_retry_pkt(mpt, cmd);
3985                        cmd->cmd_flags |= CFLAG_RETRY;
3986                        cmd->cmd_pkt_flags |= FLAG_HEAD;

3988                        (void) mptsas_accept_pkt(mpt, cmd);
6018                        break;
6019                default:
6020                        mptsas_log(mpt, CE_WARN,
6021                            "unknown ioc_status = %x\n", ioc_status);
6022                        mptsas_log(mpt, CE_CONT, "scsi_state = %x, transfer "
6023                            "count = %x, scsi_status = %x", scsi_state,
6024                            xferred, scsi_status);
6025                        break;
6026                }
6027                break;
6028        case MPI2_SCSI_STATUS_TASK_SET_FULL:
6029                mptsas_handle_qfull(mpt, cmd);
6030                break;
6031        case MPI2_SCSI_STATUS_BUSY:
6032                NDBG31(("scsi_status busy received"));
6033                break;
6034        case MPI2_SCSI_STATUS_RESERVATION_CONFLICT:
6035                NDBG31(("scsi_status reservation conflict received"));
6036                break;
```



```
6037        default:
6038                mptsas_log(mpt, CE_WARN, "scsi_status=%x, ioc_status=%x\n",
6039                    scsi_status, ioc_status);
6040                mptsas_log(mpt, CE_WARN,
6041                    "mptsas_process_intr: invalid scsi status\n");
6042                break;
6043        }
6044 }
```

*_____unchanged_portion_omitted_*

```
6100 static void
6101 mptsas_doneq_thread(mptsas_thread_arg_t *arg)
4072 mptsas_doneq_thread(mptsas_doneq_thread_arg_t *arg)
6102 {
6103        mptsas_t                        *mpt = arg->mpt;
6104        uint32_t                        t = arg->t;
4075        uint64_t                        t = arg->t;
6105        mptsas_cmd_t                    *cmd;
6106        struct scsi_pkt                 *pkt;
6107        mptsas_doneq_thread_list_t      *item = &mpt->m_doneq_thread_id[t];

6109        mutex_enter(&item->mutex);
6110        while (item->flag & MPTSAS_DONEQ_THREAD_ACTIVE) {
6111                if (!item->dlist.dl_q) {
4082                if (!item->doneq) {
6112                        cv_wait(&item->cv, &item->mutex);
6113                }
6114                pkt = NULL;
6115                if ((cmd = mptsas_doneq_thread_rm(mpt, t)) != NULL) {
6116                        cmd->cmd_flags |= CFLAG_COMPLETED;
6117                        pkt = CMD2PKT(cmd);
6118                }
6119                mutex_exit(&item->mutex);
6120                if (pkt) {
6121                        mptsas_pkt_comp(pkt, cmd);
6122                }
6123                mutex_enter(&item->mutex);
6124        }
6125        mutex_exit(&item->mutex);
6126        mutex_enter(&mpt->m_qthread_mutex);
4097        mutex_enter(&mpt->m_doneq_mutex);
6127        mpt->m_doneq_thread_n--;
6128        cv_broadcast(&mpt->m_qthread_cv);
6129        mutex_exit(&mpt->m_qthread_mutex);
4099        cv_broadcast(&mpt->m_doneq_thread_cv);
4100        mutex_exit(&mpt->m_doneq_mutex);
6130 }


6133 /*
6134  * mpt interrupt handler.
6135  */
6136 static uint_t
6137 mptsas_intr(caddr_t arg1, caddr_t arg2)
6138 {
6139        mptsas_t                        *mpt = (void *)arg1;
6140        mptsas_reply_pqueue_t           *rpqp;
6141        int                             reply_q = (int)(uintptr_t)arg2;
6142 #endif /* ! codereview */
6143        pMpi2ReplyDescriptorsUnion_t    reply_desc_union;
6144        int                             found = 0, i, rpqidx;
6145        size_t                          dma_sync_len;
6146        off_t                           dma_sync_offset;
6147        uint32_t                        istat;
4111        uchar_t                         did_reply = FALSE;
```

```
6149          NDBG18(("mptsas_intr: arg1 0x%p reply_q 0x%d", (void *)arg1, reply_q));
4113          NDBG1(("mptsas_intr: arg1 0x%p arg2 0x%p", (void *)arg1, (void *)arg2));

6151          rpqp = &mpt->m_rep_post_queues[reply_q];
4115          mutex_enter(&mpt->m_mutex);

6153          /*
6154           * If interrupts are shared by two channels then check whether this
6155           * interrupt is genuinely for this channel by making sure first the
6156           * chip is in high power state.
6157           */
6158          if ((mpt->m_options & MPTSAS_OPT_PM) &&
6159              (mpt->m_power_level != PM_LEVEL_D0)) {
6160                  mpt->m_unclaimed_pm_interrupt_count++;
6161                  return (DDI_INTR_UNCLAIMED);
6162          }

6164          istat = MPTSAS_GET_ISTAT(mpt);
6165          if (!(istat & MPI2_HIS_REPLY_DESCRIPTOR_INTERRUPT)) {
6166                  NDBG18(("Interrupt bit not set, istat 0x%x", istat));
6167                  mpt->m_unclaimed_no_interrupt_count++;
6168                  /*
6169                   * Really need a good definition of when this is valid.
6170                   * It appears not to be if you have multiple reply post
6171                   * queues, there may be a better way - need LSI info.
6172                   * For now just count them.
6173                   */
6174  #if 0
4124                  mutex_exit(&mpt->m_mutex);
6175                  return (DDI_INTR_UNCLAIMED);
6176  #endif
6177  #endif /* ! codereview */
6178          }

6180          /*
6181           * If polling, interrupt was triggered by some shared interrupt because
6182           * IOC interrupts are disabled during polling, so polling routine will
6183           * handle any replies.  Considering this, if polling is happening,
6184           * return with interrupt unclaimed.
6185           */
6186          if (mpt->m_polled_intr) {
6187                  mptsas_log(mpt, CE_WARN,
6188                      "Unclaimed interrupt, rpq %d (Polling), istat 0x%x",
6189                      reply_q, istat);
6190                  mpt->m_unclaimed_polled_interrupt_count++;
4126                  mutex_exit(&mpt->m_mutex);
4127                  mptsas_log(mpt, CE_WARN, "mpt_sas: Unclaimed interrupt");
6191                  return (DDI_INTR_UNCLAIMED);
6192          }

6194          /*
6195           * At the moment this is the only place the mutex is grabbed.
6196           * So it should never fail!
4132           * Read the istat register.
6197           */
6198          if (mutex_tryenter(&rpqp->rpq_mutex) == 0) {
6199                  mutex_enter(&rpqp->rpq_mutex);
6200                  rpqp->rpq_intr_mutexbusy++;
6201          }

6203          dma_sync_len = mpt->m_post_queue_depth * 8;
6204          dma_sync_offset = dma_sync_len * reply_q;
6205          (void) ddi_dma_sync(mpt->m_dma_post_queue_hdl,
6206              dma_sync_offset, dma_sync_len, DDI_DMA_SYNC_FORCPU);

4134          if ((INTPENDING(mpt)) != 0) {
```



```
6208          /*
6209           * Go around the reply queue and process each descriptor until
6210           * we get to the next unused one.
6211           * It seems to be an occupational hazard that we get interrupts
6212           * with nothing to do. These are counted below.
4136                * read fifo until empty.
6213           */
6214          rpqidx = rpqp->rpq_index;
6215  #endif /* ! codereview */
6216  #ifndef __lock_lint
6217          _NOTE(CONSTCOND)
6218  #endif
6219          while (TRUE) {
4138                          (void) ddi_dma_sync(mpt->m_dma_post_queue_hdl, 0, 0,
4139                              DDI_DMA_SYNC_FORCPU);
6220                  reply_desc_union = (pMpi2ReplyDescriptorsUnion_t)
6221                      MPTSAS_GET_NEXT_REPLY(rpqp, rpqidx);
4141                          MPTSAS_GET_NEXT_REPLY(mpt, mpt->m_post_index);

6223                  if (ddi_get32(mpt->m_acc_post_queue_hdl,
6224                      &reply_desc_union->Words.Low) == 0xFFFFFFFF ||
6225                      ddi_get32(mpt->m_acc_post_queue_hdl,
6226                      &reply_desc_union->Words.High) == 0xFFFFFFFF) {
6227                          break;
6228                  }

6230                  found++;

6232                  ASSERT(ddi_get8(mpt->m_acc_post_queue_hdl,
6233                      &reply_desc_union->Default.MSIxIndex) == reply_q);

6235                  /*
6236                   * Process it according to its type.
6237                   */
6238                  mptsas_process_intr(mpt, rpqp, reply_desc_union);

6240  #endif /* ! codereview */
6241                  /*
6242                   * Clear the reply descriptor for re-use.
4150                           * The reply is valid, process it according to its
4151                           * type.  Also, set a flag for updating the reply index
4152                           * after they've all been processed.
6243                   */
6244                  ddi_put64(mpt->m_acc_post_queue_hdl,
6245                      &((uint64_t *)(void *)rpqp->rpq_queue)[rpqidx],
6246                      0xFFFFFFFFFFFFFFFF);
4154                          did_reply = TRUE;

4156                          mptsas_process_intr(mpt, reply_desc_union);

6248                  /*
6249                   * Increment post index and roll over if needed.
6250                   */
6251                  if (++rpqidx == mpt->m_post_queue_depth) {
6252                          rpqidx = 0;
6253                  }
4161                          if (++mpt->m_post_index == mpt->m_post_queue_depth) {
4162                                  mpt->m_post_index = 0;
6254          }

6256          if (found == 0) {
6257                  rpqp->rpq_intr_unclaimed++;
6258                  mutex_exit(&rpqp->rpq_mutex);
6259                  mpt->m_unclaimed_nocmd_interrupt_count++;
6260                  return (DDI_INTR_UNCLAIMED);
6261  #endif /* ! codereview */
```

```
6262         }
6263         rpqp->rpq_index = rpqidx;
6264 #endif /* ! codereview */

6266         rpqp->rpq_intr_count++;
6267         NDBG18(("mptsas_intr complete(%d), did %d loops", reply_q, found));

6269         (void) ddi_dma_sync(mpt->m_dma_post_queue_hdl,
6270             dma_sync_offset, dma_sync_len, DDI_DMA_SYNC_FORDEV);

6272         mpt->m_interrupt_count++;

6274         /*
6275          * Update the reply index if at least one reply was processed.
6276          * For more than 8 reply queues on SAS3 controllers we have to do
6277          * things a little different. See Chapter 20 in the MPI 2.5 spec.
6278          */
6279         if (mpt->m_post_reply_qcount > 8) {
6280 #endif /* ! codereview */
6281                 /*
6282                  * The offsets from the base are multiples of 0x10.
6283                  * We are indexing into 32 bit quantities so calculate
6284                  * the index for that.
4164                  * Update the global reply index if at least one reply was
4165                  * processed.
6285                  */
6286                 i = (reply_q&~0x7) >> 1;
4167                 if (did_reply) {
6287                 ddi_put32(mpt->m_datap,
6288                     &mpt->m_reg->SuppReplyPostHostIndex[i],
6289                     rpqp->rpq_index |
6290                     ((reply_q&0x7)<<MPI2_RPHI_MSIX_INDEX_SHIFT));
6291                 (void) ddi_get32(mpt->m_datap,
6292                     &mpt->m_reg->SuppReplyPostHostIndex[i]);
4169                         &mpt->m_reg->ReplyPostHostIndex, mpt->m_post_index);
4170                 }
6293         } else {
6294                 ddi_put32(mpt->m_datap,
6295                     &mpt->m_reg->ReplyPostHostIndex,
6296                     rpqp->rpq_index | (reply_q<<MPI2_RPHI_MSIX_INDEX_SHIFT));
6297                 (void) ddi_get32(mpt->m_datap,
6298                     &mpt->m_reg->ReplyPostHostIndex);
4172                 mutex_exit(&mpt->m_mutex);
4173                 return (DDI_INTR_UNCLAIMED);
6299         }
4175         NDBG1(("mptsas_intr complete"));

6301         /*
6302          * If no helper threads are created, process the doneq in ISR. If
6303          * helpers are created, use the doneq length as a metric to measure the
6304          * load on the interrupt CPU. If it is long enough, which indicates the
6305          * load is heavy, then we deliver the IO completions to the helpers.
6306          * This measurement has some limitations, although it is simple and
6307          * straightforward and works well for most of the cases at present.
6308          * To always use the threads set mptsas_doneq_length_threshold_prop
6309          * to zero in the mpt_sas3.conf file.
6310          *
6311          * Check the current reply queue done queue.
6312          */
6313         if (rpqp->rpq_dlist.dl_len) {
6314                 if (!mpt->m_doneq_thread_n ||
6315                     (rpqp->rpq_dlist.dl_len <= mpt->m_doneq_length_threshold)) {
6316                         mptsas_rpdoneq_empty(rpqp);
6317                 } else {
6318                         mptsas_deliver_doneq_thread(mpt, &rpqp->rpq_dlist);
6319                 }
```



```
6320         }

6322         mutex_exit(&rpqp->rpq_mutex);

6324         /*
6325          * Check the main done queue. If we find something
6326          * grab the mutex and check again before processing.
6327 #endif /* ! codereview */
6328          */
6329         if (mpt->m_dlist.dl_len) {
6330                 mutex_enter(&mpt->m_mutex);
6331                 if (mpt->m_dlist.dl_len) {
6332 #endif /* ! codereview */
6333                         if (!mpt->m_doneq_thread_n ||
6334                             (mpt->m_dlist.dl_len <=
6335                             mpt->m_doneq_length_threshold)) {
4184                     (mpt->m_doneq_len <= mpt->m_doneq_length_threshold)) {
6336                                 mptsas_doneq_empty(mpt);
6337                         } else {
6338                                 mptsas_deliver_doneq_thread(mpt, &mpt->m_dlist);
6339                         }
6340                 }
6341                 mutex_exit(&mpt->m_mutex);
4187                 mptsas_deliver_doneq_thread(mpt);
6342         }

6344         /*
6345          * If there are queued cmd, start them now.
6346          */
6347         if (mpt->m_waitq != NULL) {
6348                 mutex_enter(&mpt->m_mutex);
6349                 if (mpt->m_waitq != NULL && mpt->m_polled_intr == 0) {
6350 #endif /* ! codereview */
6351                         mptsas_restart_waitq(mpt);
6352                 }

6353                 mutex_exit(&mpt->m_mutex);
6354         }
6355 #endif /* ! codereview */
6356         return (DDI_INTR_CLAIMED);
6357 }

6359 static void
6360 mptsas_process_intr(mptsas_t *mpt, mptsas_reply_pqueue_t *rpqp,
4196 mptsas_process_intr(mptsas_t *mpt,
6361     pMpi2ReplyDescriptorsUnion_t reply_desc_union)
6362 {
6363         uint8_t reply_type;

6365         /*
6366          * Should get here with the reply queue mutex held, but not
6367          * the main mpt mutex. Want to avoid grabbing that during
6368          * normal operations if possible.
6369          */
6370         ASSERT(mutex_owned(&rpqp->rpq_mutex));
4201         ASSERT(mutex_owned(&mpt->m_mutex));

6372         /*
6373          * The reply is valid, process it according to its
6374          * type.  Also, set a flag for updated the reply index
6375          * after they've all been processed.
6376          */
6377         reply_type = ddi_get8(mpt->m_acc_post_queue_hdl,
6378             &reply_desc_union->Default.ReplyFlags);
6379         NDBG18(("mptsas_process_intr(rpq %d) reply_type 0x%x", rpqp->rpq_num,
6380             reply_type));
```

```
#endif /* ! codereview */
        reply_type &= MPI2_RPY_DESCRIPT_FLAGS_TYPE_MASK;
        if (reply_type == MPI2_RPY_DESCRIPT_FLAGS_SCSI_IO_SUCCESS ||
            reply_type == MPI25_RPY_DESCRIPT_FLAGS_FAST_PATH_SCSI_IO_SUCCESS) {
                mptsas_handle_scsi_io_success(mpt, rpqp, reply_desc_union);
        if (reply_type == MPI2_RPY_DESCRIPT_FLAGS_SCSI_IO_SUCCESS) {
                mptsas_handle_scsi_io_success(mpt, reply_desc_union);
        } else if (reply_type == MPI2_RPY_DESCRIPT_FLAGS_ADDRESS_REPLY) {
                mutex_enter(&mpt->m_mutex);
#endif /* ! codereview */
                mptsas_handle_address_reply(mpt, reply_desc_union);
                mutex_exit(&mpt->m_mutex);
#endif /* ! codereview */
        } else {
                mptsas_log(mpt, CE_WARN, "?Bad reply type %x", reply_type);
                ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_UNAFFECTED);
        }

        /*
         * Clear the reply descriptor for re-use and increment
         * index.
         */
        ddi_put64(mpt->m_acc_post_queue_hdl,
            &((uint64_t *)(void *)mpt->m_post_queue)[mpt->m_post_index],
            0xFFFFFFFFFFFFFFFF);
        (void) ddi_dma_sync(mpt->m_dma_post_queue_hdl, 0, 0,
            DDI_DMA_SYNC_FORDEV);
}

/*
 * handle qfull condition
 */
static void
mptsas_handle_qfull(mptsas_t *mpt, mptsas_cmd_t *cmd)
{
        mptsas_target_t *ptgt = cmd->cmd_tgt_addr;

        mutex_enter(&ptgt->m_t_mutex);
#endif /* ! codereview */
        if ((++cmd->cmd_qfull_retries > ptgt->m_qfull_retries) ||
            (ptgt->m_qfull_retries == 0)) {
                /*
                 * We have exhausted the retries on QFULL, or,
                 * the target driver has indicated that it
                 * wants to handle QFULL itself by setting
                 * qfull-retries capability to 0. In either case
                 * we want the target driver's QFULL handling
                 * to kick in. We do this by having pkt_reason
                 * as CMD_CMPLT and pkt_scbp as STATUS_QFULL.
                 */
                mptsas_set_throttle(mpt, ptgt, DRAIN_THROTTLE);
        } else {
                if (ptgt->m_reset_delay == 0) {
                        ptgt->m_t_throttle =
                            max((ptgt->m_t_ncmds - 2), 0);
                }
                mutex_exit(&ptgt->m_t_mutex);
#endif /* ! codereview */

                cmd->cmd_pkt_flags |= FLAG_HEAD;
                cmd->cmd_flags &= ~(CFLAG_TRANFLAG);
                cmd->cmd_flags |= CFLAG_RETRY;

                mptsas_retry_pkt(mpt, cmd);
                (void) mptsas_accept_pkt(mpt, cmd);
```



```
                mutex_enter(&ptgt->m_t_mutex);
#endif /* ! codereview */
                /*
                 * when target gives queue full status with no commands
                 * outstanding (m_t_ncmds == 0), throttle is set to 0
                 * (HOLD_THROTTLE), and the queue full handling start
                 * (see psarc/1994/313); if there are commands outstanding,
                 * throttle is set to (m_t_ncmds - 2)
                 */
                if (ptgt->m_t_throttle == HOLD_THROTTLE) {
                        /*
                         * By setting throttle to QFULL_THROTTLE, we
                         * avoid submitting new commands and in
                         * mptsas_restart_cmd find out slots which need
                         * their throttles to be cleared.
                         */
                        mptsas_set_throttle(mpt, ptgt, QFULL_THROTTLE);
                        if (mpt->m_restart_cmd_timeid == 0) {
                                mpt->m_restart_cmd_timeid =
                                    timeout(mptsas_restart_cmd, mpt,
                                    ptgt->m_qfull_retry_interval);
                        }
                }
        }
        mutex_exit(&ptgt->m_t_mutex);
#endif /* ! codereview */
}

mptsas_phymask_t
mptsas_physport_to_phymask(mptsas_t *mpt, uint8_t physport)
{
        mptsas_phymask_t        phy_mask = 0;
        uint8_t                 i = 0;

        NDBG20(("mptsas3%d physport_to_phymask enter", mpt->m_instance));
        NDBG20(("mptsas%d physport_to_phymask enter", mpt->m_instance));

        ASSERT(mutex_owned(&mpt->m_mutex));

        /*
         * If physport is 0xFF, this is a RAID volume.  Use phymask of 0.
         */
        if (physport == 0xFF) {
                return (0);
        }

        for (i = 0; i < MPTSAS_MAX_PHYS; i++) {
                if (mpt->m_phy_info[i].attached_devhdl &&
                    (mpt->m_phy_info[i].phy_mask != 0) &&
                    (mpt->m_phy_info[i].port_num == physport)) {
                        phy_mask = mpt->m_phy_info[i].phy_mask;
                        break;
                }
        }
        NDBG20(("mptsas3%d physport_to_phymask:physport :%x phymask :%x, ",
        NDBG20(("mptsas%d physport_to_phymask:physport :%x phymask :%x, ",
            mpt->m_instance, physport, phy_mask));
        return (phy_mask);
}
_____unchanged_portion_omitted_

static void
mptsas_update_phymask(mptsas_t *mpt)
{
        mptsas_phymask_t mask = 0, phy_mask;
        char            *phy_mask_name;
```

```
6536         uint8_t         current_port;
6537         int             i, j;

6539         NDBG20(("mptsas3%d update phymask ", mpt->m_instance));
4312         NDBG20(("mptsas%d update phymask ", mpt->m_instance));

6541         ASSERT(mutex_owned(&mpt->m_mutex));

6543         (void) mptsas_get_sas_io_unit_page(mpt);

6545         phy_mask_name = kmem_zalloc(MPTSAS_MAX_PHYS, KM_SLEEP);

6547         for (i = 0; i < mpt->m_num_phys; i++) {
6548                 phy_mask = 0x00;

6550                 if (mpt->m_phy_info[i].attached_devhdl == 0)
6551                         continue;

6553                 bzero(phy_mask_name, sizeof (phy_mask_name));

6555                 current_port = mpt->m_phy_info[i].port_num;

6557                 if ((mask & (1 << i)) != 0)
6558                         continue;

6560                 for (j = 0; j < mpt->m_num_phys; j++) {
6561                         if (mpt->m_phy_info[j].attached_devhdl &&
6562                             (mpt->m_phy_info[j].port_num == current_port)) {
6563                                 phy_mask |= (1 << j);
6564                         }
6565                 }
6566                 mask = mask | phy_mask;

6568                 for (j = 0; j < mpt->m_num_phys; j++) {
6569                         if ((phy_mask >> j) & 0x01) {
6570                                 mpt->m_phy_info[j].phy_mask = phy_mask;
6571                         }
6572                 }

6574                 (void) sprintf(phy_mask_name, "%x", phy_mask);

6576                 mutex_exit(&mpt->m_mutex);
6577                 /*
6578                  * register a iport, if the port has already been existed
6579                  * SCSA will do nothing and just return.
6580                  */
6581                 (void) scsi_hba_iport_register(mpt->m_dip, phy_mask_name);
6582                 mutex_enter(&mpt->m_mutex);
6583         }
6584         kmem_free(phy_mask_name, MPTSAS_MAX_PHYS);
6585         NDBG20(("mptsas3%d update phymask return", mpt->m_instance));
4358         NDBG20(("mptsas%d update phymask return", mpt->m_instance));
6586 }

6588 /*
6589  * mptsas_handle_dr is a task handler for DR, the DR action includes:
6590  * 1. Directly attched Device Added/Removed.
6591  * 2. Expander Device Added/Removed.
6592  * 3. Indirectly Attached Device Added/Expander.
6593  * 4. LUNs of a existing device status change.
6594  * 5. RAID volume created/deleted.
6595  * 6. Member of RAID volume is released because of RAID deletion.
6596  * 7. Physical disks are removed because of RAID creation.
6597  */
6598 static void
6599 mptsas_handle_dr(void *args) {
```



```
6600         mptsas_topo_change_list_t       *topo_node = NULL;
6601         mptsas_topo_change_list_t       *save_node = NULL;
6602         mptsas_t                        *mpt;
6603         dev_info_t                      *parent = NULL;
6604         mptsas_phymask_t                phymask = 0;
6605         char                            phy_mask_name[MPTSAS_MAX_PHYS];
4378         char                            *phy_mask_name;
6606         uint8_t                         flags = 0, physport = 0xff;
6607         uint8_t                         port_update = 0;
6608         uint_t                          event;

6610         topo_node = (mptsas_topo_change_list_t *)args;

6612         mpt = topo_node->mpt;
6613         event = topo_node->event;
6614         flags = topo_node->flags;

6616         NDBG20(("mptsas3%d handle_dr enter", mpt->m_instance));
4389         phy_mask_name = kmem_zalloc(MPTSAS_MAX_PHYS, KM_SLEEP);

4391         NDBG20(("mptsas%d handle_dr enter", mpt->m_instance));

6618         switch (event) {
6619         case MPTSAS_DR_EVENT_RECONFIG_TARGET:
6620                 if ((flags == MPTSAS_TOPO_FLAG_DIRECT_ATTACHED_DEVICE) ||
6621                     (flags == MPTSAS_TOPO_FLAG_EXPANDER_ATTACHED_DEVICE) ||
6622                     (flags == MPTSAS_TOPO_FLAG_RAID_PHYSDRV_ASSOCIATED)) {
6623                         /*
6624                          * Direct attached or expander attached device added
6625                          * into system or a Phys Disk that is being unhidden.
6626                          */
6627                         port_update = 1;
6628                 }
6629                 break;
6630         case MPTSAS_DR_EVENT_RECONFIG_SMP:
6631                 /*
6632                  * New expander added into system, it must be the head
6633                  * of topo_change_list_t
6634                  */
6635                 port_update = 1;
6636                 break;
6637         default:
6638                 port_update = 0;
6639                 break;
6640         }
6641         /*
6642          * All cases port_update == 1 may cause initiator port form change
6643          */
6644         mutex_enter(&mpt->m_mutex);
6645         if (mpt->m_port_chng && port_update) {
6646                 /*
6647                  * mpt->m_port_chng flag indicates some PHYs of initiator
6648                  * port have changed to online. So when expander added or
6649                  * directly attached device online event come, we force to
6650                  * update port information by issueing SAS IO Unit Page and
6651                  * update PHYMASKs.
6652                  */
6653                 (void) mptsas_update_phymask(mpt);
6654                 mpt->m_port_chng = 0;

6656         }
6657         mutex_exit(&mpt->m_mutex);

6659 #endif /* ! codereview */
6660         while (topo_node) {
6661                 phymask = 0;
```

```
6662                flags = topo_node->flags;
6663                event = topo_node->event;
6664                if (event == MPTSAS_DR_EVENT_REMOVE_HANDLE) {
6665                        goto handle_topo_change;
6666                }
6667                if ((event == MPTSAS_DR_EVENT_RECONFIG_TARGET) &&
6668                    (flags == MPTSAS_TOPO_FLAG_RAID_PHYSDRV_ASSOCIATED)) {
6669                        /*
6670                         * There is no any field in IR_CONFIG_CHANGE
6671                         * event indicate physport/phynum, let's get
6672                         * parent after SAS Device Page0 request.
6673                         */
6674                        goto handle_topo_change;
6675                }

6677 #endif /* ! codereview */
6678                if (parent == NULL) {
6679                        physport = topo_node->un.physport;
6433                        event = topo_node->event;
6434                        flags = topo_node->flags;
6680                        if (event & (MPTSAS_DR_EVENT_OFFLINE_TARGET |
6681                            MPTSAS_DR_EVENT_OFFLINE_SMP)) {
6682                                /*
6683                                 * For all offline events, phymask is known
6684                                 */
6685                                phymask = topo_node->un.phymask;
6686                                goto find_parent;
6687                        }
4443                        if (event & MPTSAS_TOPO_FLAG_REMOVE_HANDLE) {
4444                                goto handle_topo_change;
4445                        }
6688                        if (flags & MPTSAS_TOPO_FLAG_LUN_ASSOCIATED) {
6689                                phymask = topo_node->un.phymask;
6690                                goto find_parent;
6691                        }

4451                        if ((flags ==
4452                            MPTSAS_TOPO_FLAG_RAID_PHYSDRV_ASSOCIATED) &&
4453                            (event == MPTSAS_DR_EVENT_RECONFIG_TARGET)) {
4454                                /*
4455                                 * There is no any field in IR_CONFIG_CHANGE
4456                                 * event indicate physport/phynum, let's get
4457                                 * parent after SAS Device Page0 request.
4458                                 */
4459                                goto handle_topo_change;
4460                        }

6693                        mutex_enter(&mpt->m_mutex);
6694                        if (flags == MPTSAS_TOPO_FLAG_DIRECT_ATTACHED_DEVICE) {
6695                                /*
6696                                 * If the direct attached device added or a
6697                                 * phys disk is being unhidden, argument
6698                                 * physport actually is PHY#, so we have to get
6699                                 * phymask according PHY#.
6700                                 */
6701                                physport = mpt->m_phy_info[physport].port_num;
6702                        }

6704                        /*
6705                         * Translate physport to phymask so that we can search
6706                         * parent dip.
6707                         */
6708                        phymask = mptsas_physport_to_phymask(mpt, physport);
4477                        phymask = mptsas_physport_to_phymask(mpt,
4478                            physport);
6709                        mutex_exit(&mpt->m_mutex);
```



```
6711 find_parent:
6712                        bzero(phy_mask_name, MPTSAS_MAX_PHYS);
6713                        /*
6714                         * For RAID topology change node, write the iport name
6715                         * as v0.
6716                         */
6717                        if (flags & MPTSAS_TOPO_FLAG_RAID_ASSOCIATED) {
6718                                (void) sprintf(phy_mask_name, "v0");
6719                        } else {
6720                                /*
6721                                 * phymask can be 0 if the drive has been
4491                                 * phymask can bo 0 if the drive has been
6722                                 * pulled by the time an add event is
6723                                 * processed.  If phymask is 0, just skip this
6724                                 * event and continue.
6725                                 */
6726                                if (phymask == 0) {
4497                                        mutex_enter(&mpt->m_mutex);
6727                                        save_node = topo_node;
6728                                        topo_node = topo_node->next;
6729                                        ASSERT(save_node);
6730                                        kmem_free(save_node,
6731                                            sizeof (mptsas_topo_change_list_t));
4503                                        mutex_exit(&mpt->m_mutex);

6732                                        parent = NULL;
6733                                        continue;
6734                                }
6735                                (void) sprintf(phy_mask_name, "%x", phymask);
6736                        }
6737                        parent = scsi_hba_iport_find(mpt->m_dip,
6738                            phy_mask_name);
6739                        if (parent == NULL) {
6740                                mptsas_log(mpt, CE_WARN, "Failed to find an "
6741                                    "iport for \"%s\", should not happen!",
6742                                    phy_mask_name);
6743                                save_node = topo_node;
6744                                topo_node = topo_node->next;
6745                                ASSERT(save_node);
6746                                kmem_free(save_node,
6747                                    sizeof (mptsas_topo_change_list_t));
6748                                continue;
4514                                    "iport, should not happen!");
4515                                goto out;
6749                        }

6751                }
6752                ASSERT(parent);
6753 handle_topo_change:

6755                mutex_enter(&mpt->m_mutex);
6756                /*
6757                 * If HBA is being reset, don't perform operations depending
6758                 * on the IOC. We must free the topo list, however.
6759                 */
6760                if (!mpt->m_in_reset)
6761                        mptsas_handle_topo_change(topo_node, parent);
6762                else
6763                        NDBG20(("skipping topo change received during reset"));
6764                mutex_exit(&mpt->m_mutex);
6765 #endif /* ! codereview */
6766                save_node = topo_node;
6767                topo_node = topo_node->next;
6768                ASSERT(save_node);
6769                kmem_free(save_node, sizeof (mptsas_topo_change_list_t));
```

```
4531                mutex_exit(&mpt->m_mutex);

6771                if ((flags == MPTSAS_TOPO_FLAG_DIRECT_ATTACHED_DEVICE) ||
6772                    (flags == MPTSAS_TOPO_FLAG_RAID_PHYSDRV_ASSOCIATED) ||
6773                    (flags == MPTSAS_TOPO_FLAG_RAID_ASSOCIATED)) {
6774                        /*
6775                         * If direct attached device associated, make sure
6776                         * reset the parent before start the next one. But
6777                         * all devices associated with expander shares the
6778                         * parent.  Also, reset parent if this is for RAID.
6779                         */
6780                        parent = NULL;
6781                }
6782        }
6783 }

6785 static void
6786 mptsas_offline_target(mptsas_t *mpt, mptsas_target_t *ptgt,
6787     uint8_t topo_flags, dev_info_t *parent)
6788 {
6789         uint64_t        sas_wwn = 0;
6790         uint8_t         phy;
6791         char            wwn_str[MPTSAS_WWN_STRLEN];
6792         uint16_t        devhdl;
6793         int             circ = 0, circ1 = 0;
6794         int             rval = 0;

6796         sas_wwn = ptgt->m_addr.mta_wwn;
6797         phy = ptgt->m_phynum;
6798         devhdl = ptgt->m_devhdl;

6800         if (sas_wwn) {
6801                 (void) sprintf(wwn_str, "w%016"PRIx64, sas_wwn);
6802         } else {
6803                 (void) sprintf(wwn_str, "p%x", phy);
6804         }

6806         /*
6807          * Abort all outstanding command on the device
6808          */
6809         rval = mptsas_do_scsi_reset(mpt, devhdl);
6810         if (rval) {
6811                 NDBG20(("mptsas3%d: mptsas_offline_target: reset target "
6812                     "before offline devhdl:%x, phymask:%x, rval:%x",
6813                     mpt->m_instance, ptgt->m_devhdl,
6814                     ptgt->m_addr.mta_phymask, rval));
6815         }

6817         mutex_exit(&mpt->m_mutex);

6819         ndi_devi_enter(scsi_vhci_dip, &circ);
6820         ndi_devi_enter(parent, &circ1);
6821         rval = mptsas_offline_targetdev(parent, wwn_str);
6822         ndi_devi_exit(parent, circ1);
6823         ndi_devi_exit(scsi_vhci_dip, circ);
6824         NDBG20(("mptsas3%d: mptsas_offline_target %s devhdl:%x, "
6825             "phymask:%x, rval:%x", mpt->m_instance, wwn_str,
6826             ptgt->m_devhdl, ptgt->m_addr.mta_phymask, rval));

6828         /*
6829          * Clear parent's props for SMHBA support
6830          */
6831         if (topo_flags == MPTSAS_TOPO_FLAG_DIRECT_ATTACHED_DEVICE) {
6832                 if (ddi_prop_update_string(DDI_DEV_T_NONE, parent,
6833                     SCSI_ADDR_PROP_ATTACHED_PORT, "") !=
6834                     DDI_PROP_SUCCESS) {
```



```
6835                         (void) ddi_prop_remove(DDI_DEV_T_NONE, parent,
6836                             SCSI_ADDR_PROP_ATTACHED_PORT);
6837                         mptsas_log(mpt, CE_WARN, "mptsas attached port "
6838                             "prop update failed");
6839                 }
6840                 if (ddi_prop_update_int(DDI_DEV_T_NONE, parent,
6841                     MPTSAS_NUM_PHYS, 0) != DDI_PROP_SUCCESS) {
6842                         (void) ddi_prop_remove(DDI_DEV_T_NONE, parent,
6843                             MPTSAS_NUM_PHYS);
6844                         mptsas_log(mpt, CE_WARN, "mptsas num phys "
6845                             "prop update failed");
6846                 }
6847                 if (ddi_prop_update_int(DDI_DEV_T_NONE, parent,
6848                     MPTSAS_VIRTUAL_PORT, 1) != DDI_PROP_SUCCESS) {
6849                         (void) ddi_prop_remove(DDI_DEV_T_NONE, parent,
6850                             MPTSAS_VIRTUAL_PORT);
6851                         mptsas_log(mpt, CE_WARN, "mptsas virtual port "
6852                             "prop update failed");
6853                 }
6854         }

6856         mutex_enter(&mpt->m_mutex);
6857         ptgt->m_led_status = 0;
6858         (void) mptsas_flush_led_status(mpt, ptgt);
6859         if (rval == DDI_SUCCESS) {
6860                 mutex_destroy(&ptgt->m_t_mutex);
6861                 refhash_remove(mpt->m_targets, ptgt);
6862                 ptgt = NULL;
6863         } else {
6864                 /*
6865                  * clean DR_INTRANSITION flag to allow I/O down to
6866                  * PHCI driver since failover finished.
6867                  * Invalidate the devhdl
6868                  */
6869                 ptgt->m_devhdl = MPTSAS_INVALID_DEVHDL;
6870                 ptgt->m_tgt_unconfigured = 0;
6871                 ptgt->m_dr_flag = MPTSAS_DR_INACTIVE;
6872         }
4545 out:
4546         kmem_free(phy_mask_name, MPTSAS_MAX_PHYS);
6873 }

6875 static void
6876 mptsas_handle_topo_change(mptsas_topo_change_list_t *topo_node,
6877     dev_info_t *parent)
6878 {
6879         mptsas_target_t *ptgt = NULL;
6880         mptsas_smp_t    *psmp = NULL;
6881         mptsas_t        *mpt = (void *)topo_node->mpt;
6882         uint16_t        devhdl;
6883         uint16_t        attached_devhdl;
4558         uint64_t        sas_wwn = 0;
6884         int             rval = 0;
6885         uint32_t        page_address;
6886         uint8_t         flags;
4561         uint8_t         phy, flags;
4562         char            *addr = NULL;
6887         dev_info_t      *lundip;
6888         int             circ = 0, circ1 = 0;
6889         char            attached_wwnstr[MPTSAS_WWN_STRLEN];

6891         NDBG20(("mptsas3%d handle_topo_change enter, devhdl 0x%x,"
6892             "event 0x%x, flags 0x%x", mpt->m_instance, topo_node->devhdl,
6893             topo_node->event, topo_node->flags));
4567         NDBG20(("mptsas%d handle_topo_change enter", mpt->m_instance));
```

```
6895         ASSERT(mutex_owned(&mpt->m_mutex));
6896
6897         switch (topo_node->event) {
6898         case MPTSAS_DR_EVENT_RECONFIG_TARGET:
6899         {
6900                 char *phy_mask_name;
6901                 mptsas_phymask_t phymask = 0;
6902
6903                 if (topo_node->flags == MPTSAS_TOPO_FLAG_RAID_ASSOCIATED) {
6904                         /*
6905                          * Get latest RAID info.
6906                          */
6907                         (void) mptsas_get_raid_info(mpt);
6908                         ptgt = refhash_linear_search(mpt->m_targets,
6909                             mptsas_target_eval_devhdl, &topo_node->devhdl);
6910                         if (ptgt == NULL)
6911                                 break;
6912                 } else {
6913                         ptgt = (void *)topo_node->object;
6914                 }
6915
6916                 if (ptgt == NULL) {
6917                         /*
6918                          * If a Phys Disk was deleted, RAID info needs to be
6919                          * updated to reflect the new topology.
6920                          */
6921                         (void) mptsas_get_raid_info(mpt);
6922
6923                         /*
6924                          * Get sas device page 0 by DevHandle to make sure if
6925                          * SSP/SATA end device exist.
6926                          */
6927                         page_address = (MPI2_SAS_DEVICE_PGAD_FORM_HANDLE &
6928                             MPI2_SAS_DEVICE_PGAD_FORM_MASK) |
6929                             topo_node->devhdl;
6930
6931                         rval = mptsas_get_target_device_info(mpt, page_address,
6932                             &devhdl, &ptgt);
6933                         if (rval == DEV_INFO_WRONG_DEVICE_TYPE) {
6934                                 mptsas_log(mpt, CE_NOTE,
6935                                     "mptsas_handle_topo_change: target %d is "
6936                                     "not a SAS/SATA device. \n",
6937                                     topo_node->devhdl);
6938                         } else if (rval == DEV_INFO_FAIL_ALLOC) {
6939                                 mptsas_log(mpt, CE_NOTE,
6940                                     "mptsas_handle_topo_change: could not "
6941                                     "allocate memory. \n");
6942                         }
6943                         /*
6944                          * If rval is DEV_INFO_PHYS_DISK than there is nothing
6945                          * else to do, just leave.
6946                          */
6947                         if (rval != DEV_INFO_SUCCESS) {
6948                                 return;
6949                         }
6950                 }
6951
6952                 ASSERT(ptgt->m_devhdl == topo_node->devhdl);
6953
6954                 mutex_exit(&mpt->m_mutex);
6955                 flags = topo_node->flags;
6956
6957                 if (flags == MPTSAS_TOPO_FLAG_RAID_PHYSDRV_ASSOCIATED) {
6958                         phymask = ptgt->m_addr.mta_phymask;
6959                         phy_mask_name = kmem_zalloc(MPTSAS_MAX_PHYS, KM_SLEEP);
6960                         (void) sprintf(phy_mask_name, "%x", phymask);
```



```
6961                         parent = scsi_hba_iport_find(mpt->m_dip,
6962                             phy_mask_name);
6963                         kmem_free(phy_mask_name, MPTSAS_MAX_PHYS);
6964                         if (parent == NULL) {
6965                                 mptsas_log(mpt, CE_WARN, "Failed to find a "
6966                                     "iport for PD, should not happen!");
6967                                 mutex_enter(&mpt->m_mutex);
6968                                 break;
6969                         }
6970                 }
6971
6972                 if (flags == MPTSAS_TOPO_FLAG_RAID_ASSOCIATED) {
6973                         ndi_devi_enter(parent, &circ1);
6974                         (void) mptsas_config_raid(parent, topo_node->devhdl,
6975                             &lundip);
6976                         ndi_devi_exit(parent, circ1);
6977                 } else {
6978                         /*
6979                          * hold nexus for bus configure
6980                          */
6981                         ndi_devi_enter(scsi_vhci_dip, &circ);
6982                         ndi_devi_enter(parent, &circ1);
6983                         rval = mptsas_config_target(parent, ptgt);
6984                         /*
6985                          * release nexus for bus configure
6986                          */
6987                         ndi_devi_exit(parent, circ1);
6988                         ndi_devi_exit(scsi_vhci_dip, circ);
6989
6990                         /*
6991                          * Add parent's props for SMHBA support
6992                          */
6993                         if (flags == MPTSAS_TOPO_FLAG_DIRECT_ATTACHED_DEVICE) {
6994                                 bzero(attached_wwnstr,
6995                                     sizeof (attached_wwnstr));
6996                                 (void) sprintf(attached_wwnstr, "w%016"PRIx64,
6997                                     ptgt->m_addr.mta_wwn);
6998                                 if (ddi_prop_update_string(DDI_DEV_T_NONE,
6999                                     parent,
7000                                     SCSI_ADDR_PROP_ATTACHED_PORT,
7001                                     attached_wwnstr)
7002                                     != DDI_PROP_SUCCESS) {
7003                                         (void) ddi_prop_remove(DDI_DEV_T_NONE,
7004                                             parent,
7005                                             SCSI_ADDR_PROP_ATTACHED_PORT);
7006                                         mptsas_log(mpt, CE_WARN, "Failed to"
7007                                             "attached-port props");
7008                                         return;
7009                                 }
7010                                 if (ddi_prop_update_int(DDI_DEV_T_NONE, parent,
7011                                     MPTSAS_NUM_PHYS, 1) !=
7012                                     DDI_PROP_SUCCESS) {
7013                                         (void) ddi_prop_remove(DDI_DEV_T_NONE,
7014                                             parent, MPTSAS_NUM_PHYS);
7015                                         mptsas_log(mpt, CE_WARN, "Failed to"
7016                                             " create num-phys props");
7017                                         return;
7018                                 }
7019
7020                                 /*
7021                                  * Update PHY info for smhba
7022                                  */
7023                                 mutex_enter(&mpt->m_mutex);
7024                                 if (mptsas_smhba_phy_init(mpt)) {
7025                                         mutex_exit(&mpt->m_mutex);
7026                                         mptsas_log(mpt, CE_WARN, "mptsas phy"
```

```
7027                                        " update failed");
7028                                return;
7029                        }
7030                        mutex_exit(&mpt->m_mutex);

7032                        /*
7033                         * topo_node->un.physport is really the PHY#
7034                         * for direct attached devices
7035                         */
7036                        mptsas_smhba_set_one_phy_props(mpt, parent,
7037                            topo_node->un.physport, &attached_devhdl);

7039                        if (ddi_prop_update_int(DDI_DEV_T_NONE, parent,
7040                            MPTSAS_VIRTUAL_PORT, 0) !=
7041                            DDI_PROP_SUCCESS) {
7042                                (void) ddi_prop_remove(DDI_DEV_T_NONE,
7043                                    parent, MPTSAS_VIRTUAL_PORT);
7044                                mptsas_log(mpt, CE_WARN,
7045                                    "mptsas virtual-port"
7046                                    "port prop update failed");
7047                                return;
7048                        }
7049                }
7050        }
7051        mutex_enter(&mpt->m_mutex);

7053        NDBG20(("mptsas3%d handle_topo_change to online devhdl:%x, "
4727        NDBG20(("mptsas%d handle_topo_change to online devhdl:%x, "
7054            "phymask:%x.", mpt->m_instance, ptgt->m_devhdl,
7055            ptgt->m_addr.mta_phymask));
7056        break;
7057    }
7058    case MPTSAS_DR_EVENT_OFFLINE_TARGET:
7059    {
7060        devhdl = topo_node->devhdl;
7061        ptgt = refhash_linear_search(mpt->m_targets,
7062            mptsas_target_eval_devhdl, &devhdl);
7063        if (ptgt == NULL)
7064                break;

4740        sas_wwn = ptgt->m_addr.mta_wwn;
4741        phy = ptgt->m_phynum;

4743        addr = kmem_zalloc(SCSI_MAXNAMELEN, KM_SLEEP);

4745        if (sas_wwn) {
4746                (void) sprintf(addr, "w%016"PRIx64, sas_wwn);
4747        } else {
4748                (void) sprintf(addr, "p%x", phy);
4749        }
7066        ASSERT(ptgt->m_devhdl == devhdl);

7068        if ((topo_node->flags == MPTSAS_TOPO_FLAG_RAID_ASSOCIATED) ||
7069            (topo_node->flags ==
7070            MPTSAS_TOPO_FLAG_RAID_PHYSDRV_ASSOCIATED)) {
7071                /*
7072                 * Get latest RAID info if RAID volume status changes
7073                 * or Phys Disk status changes
7074                 */
7075                (void) mptsas_get_raid_info(mpt);
7076        }
4761        /*
4762         * Abort all outstanding command on the device
4763         */
4764        rval = mptsas_do_scsi_reset(mpt, devhdl);
4765        if (rval) {
```



```
4766                        NDBG20(("mptsas%d handle_topo_change to reset target "
4767                            "before offline devhdl:%x, phymask:%x, rval:%x",
4768                            mpt->m_instance, ptgt->m_devhdl,
4769                            ptgt->m_addr.mta_phymask, rval));
4770                }

4772                mutex_exit(&mpt->m_mutex);

4774                ndi_devi_enter(scsi_vhci_dip, &circ);
4775                ndi_devi_enter(parent, &circ1);
4776                rval = mptsas_offline_target(parent, addr);
4777                ndi_devi_exit(parent, circ1);
4778                ndi_devi_exit(scsi_vhci_dip, circ);
4779                NDBG20(("mptsas%d handle_topo_change to offline devhdl:%x, "
4780                    "phymask:%x, rval:%x", mpt->m_instance,
4781                    ptgt->m_devhdl, ptgt->m_addr.mta_phymask, rval));

4783                kmem_free(addr, SCSI_MAXNAMELEN);

4785                /*
4786                 * Clear parent's props for SMHBA support
4787                 */
4788                flags = topo_node->flags;
4789                if (flags == MPTSAS_TOPO_FLAG_DIRECT_ATTACHED_DEVICE) {
4790                        bzero(attached_wwnstr, sizeof (attached_wwnstr));
4791                        if (ddi_prop_update_string(DDI_DEV_T_NONE, parent,
4792                            SCSI_ADDR_PROP_ATTACHED_PORT, attached_wwnstr) !=
4793                            DDI_PROP_SUCCESS) {
4794                                (void) ddi_prop_remove(DDI_DEV_T_NONE, parent,
4795                                    SCSI_ADDR_PROP_ATTACHED_PORT);
4796                                mptsas_log(mpt, CE_WARN, "mptsas attached port "
4797                                    "prop update failed");
4798                                break;
4799                        }
4800                        if (ddi_prop_update_int(DDI_DEV_T_NONE, parent,
4801                            MPTSAS_NUM_PHYS, 0) !=
4802                            DDI_PROP_SUCCESS) {
4803                                (void) ddi_prop_remove(DDI_DEV_T_NONE, parent,
4804                                    MPTSAS_NUM_PHYS);
4805                                mptsas_log(mpt, CE_WARN, "mptsas num phys "
4806                                    "prop update failed");
4807                                break;
4808                        }
4809                        if (ddi_prop_update_int(DDI_DEV_T_NONE, parent,
4810                            MPTSAS_VIRTUAL_PORT, 1) !=
4811                            DDI_PROP_SUCCESS) {
4812                                (void) ddi_prop_remove(DDI_DEV_T_NONE, parent,
4813                                    MPTSAS_VIRTUAL_PORT);
4814                                mptsas_log(mpt, CE_WARN, "mptsas virtual port "
4815                                    "prop update failed");
4816                                break;
4817                        }
4818                }

7078                mptsas_offline_target(mpt, ptgt, topo_node->flags, parent);
4820                mutex_enter(&mpt->m_mutex);
4821                ptgt->m_led_status = 0;
4822                (void) mptsas_flush_led_status(mpt, ptgt);
4823                if (rval == DDI_SUCCESS) {
4824                        refhash_remove(mpt->m_targets, ptgt);
4825                        ptgt = NULL;
4826                } else {
4827                        /*
4828                         * clean DR_INTRANSITION flag to allow I/O down to
4829                         * PHCI driver since failover finished.
4830                         * Invalidate the devhdl
```

```
4831                         */
4832                        ptgt->m_devhdl = MPTSAS_INVALID_DEVHDL;
4833                        ptgt->m_tgt_unconfigured = 0;
4834                        mutex_enter(&mpt->m_tx_waitq_mutex);
4835                        ptgt->m_dr_flag = MPTSAS_DR_INACTIVE;
4836                        mutex_exit(&mpt->m_tx_waitq_mutex);
4837                }

7080                /*
7081                 * Send SAS IO Unit Control to free the dev handle
7082                 */
7083                if ((flags == MPTSAS_TOPO_FLAG_DIRECT_ATTACHED_DEVICE) ||
7084                    (flags == MPTSAS_TOPO_FLAG_EXPANDER_ATTACHED_DEVICE)) {
7085                        rval = mptsas_free_devhdl(mpt, devhdl);

7087                        NDBG20(("mptsas3%d handle_topo_change to remove "
4846                        NDBG20(("mptsas%d handle_topo_change to remove "
7088                            "devhdl:%x, rval:%x", mpt->m_instance, devhdl,
7089                            rval));
7090                }

7092                break;
7093        }
7094        case MPTSAS_DR_EVENT_REMOVE_HANDLE:
4853        case MPTSAS_TOPO_FLAG_REMOVE_HANDLE:
7095        {
7096                devhdl = topo_node->devhdl;

7098 #endif /* ! codereview */
7099                /*
7100                 * Do a reset first.
4856                 * If this is the remove handle event, do a reset first.
7101                 */
4858                if (topo_node->event == MPTSAS_TOPO_FLAG_REMOVE_HANDLE) {
7102                rval = mptsas_do_scsi_reset(mpt, devhdl);
4860                        if (rval) {
7103                NDBG20(("mpt%d reset target before remove "
7104                    "devhdl:%x, rval:%x", mpt->m_instance, devhdl, rval));
4862                                "devhdl:%x, rval:%x", mpt->m_instance,
4863                                devhdl, rval));
4864                        }
4865                }

7106                /*
7107                 * Send SAS IO Unit Control to free the dev handle
7108                 */
7109                rval = mptsas_free_devhdl(mpt, devhdl);
7110                NDBG20(("mptsas3%d handle_topo_change to remove "
4871                NDBG20(("mptsas%d handle_topo_change to remove "
7111                    "devhdl:%x, rval:%x", mpt->m_instance, devhdl,
7112                    rval));
7113                break;
7114        }
7115        case MPTSAS_DR_EVENT_RECONFIG_SMP:
7116        {
7117                mptsas_smp_t smp;
7118                dev_info_t *smpdip;

7120                devhdl = topo_node->devhdl;

7122                page_address = (MPI2_SAS_EXPAND_PGAD_FORM_HNDL &
7123                    MPI2_SAS_EXPAND_PGAD_FORM_MASK) | (uint32_t)devhdl;
7124                rval = mptsas_get_sas_expander_page0(mpt, page_address, &smp);
7125                if (rval != DDI_SUCCESS) {
7126                        mptsas_log(mpt, CE_WARN, "failed to online smp, "
7127                            "handle %x", devhdl);
```



```
7128                        return;
7129                }

7131                psmp = mptsas_smp_alloc(mpt, &smp);
7132                if (psmp == NULL) {
7133                        return;
7134                }

7136                mutex_exit(&mpt->m_mutex);
7137                ndi_devi_enter(parent, &circ1);
7138                (void) mptsas_online_smp(parent, psmp, &smpdip);
7139                ndi_devi_exit(parent, circ1);

7141                mutex_enter(&mpt->m_mutex);
7142                break;
7143        }
7144        case MPTSAS_DR_EVENT_OFFLINE_SMP:
7145        {
7146                devhdl = topo_node->devhdl;
7147                uint32_t dev_info;

7149                psmp = refhash_linear_search(mpt->m_smp_targets,
7150                    mptsas_smp_eval_devhdl, &devhdl);
7151                if (psmp == NULL)
7152                        break;
7153                /*
7154                 * The mptsas_smp_t data is released only if the dip is offlined
7155                 * successfully.
7156                 */
7157                mutex_exit(&mpt->m_mutex);

7159                ndi_devi_enter(parent, &circ1);
7160                rval = mptsas_offline_smp(parent, psmp, NDI_DEVI_REMOVE);
7161                ndi_devi_exit(parent, circ1);

7163                dev_info = psmp->m_deviceinfo;
7164                if ((dev_info & DEVINFO_DIRECT_ATTACHED) ==
7165                    DEVINFO_DIRECT_ATTACHED) {
7166                        if (ddi_prop_update_int(DDI_DEV_T_NONE, parent,
7167                            MPTSAS_VIRTUAL_PORT, 1) !=
7168                            DDI_PROP_SUCCESS) {
7169                                (void) ddi_prop_remove(DDI_DEV_T_NONE, parent,
7170                                    MPTSAS_VIRTUAL_PORT);
7171                                mptsas_log(mpt, CE_WARN, "mptsas virtual port "
7172                                    "prop update failed");
7173                                return;
7174                        }
7175                        /*
7176                         * Check whether the smp connected to the iport,
7177                         */
7178                        if (ddi_prop_update_int(DDI_DEV_T_NONE, parent,
7179                            MPTSAS_NUM_PHYS, 0) !=
7180                            DDI_PROP_SUCCESS) {
7181                                (void) ddi_prop_remove(DDI_DEV_T_NONE, parent,
7182                                    MPTSAS_NUM_PHYS);
7183                                mptsas_log(mpt, CE_WARN, "mptsas num phys"
7184                                    "prop update failed");
7185                                return;
7186                        }
7187                        /*
7188                         * Clear parent's attached-port props
7189                         */
7190                        bzero(attached_wwnstr, sizeof (attached_wwnstr));
7191                        if (ddi_prop_update_string(DDI_DEV_T_NONE, parent,
7192                            SCSI_ADDR_PROP_ATTACHED_PORT, attached_wwnstr) !=
7193                            DDI_PROP_SUCCESS) {
```

```
7194                                (void) ddi_prop_remove(DDI_DEV_T_NONE, parent,
7195                                    SCSI_ADDR_PROP_ATTACHED_PORT);
7196                                mptsas_log(mpt, CE_WARN, "mptsas attached port "
7197                                    "prop update failed");
7198                                return;
7199                        }
7200                }

7202                mutex_enter(&mpt->m_mutex);
7203                NDBG20(("mptsas3%d handle_topo_change to remove devhdl:%x, "
4964                NDBG20(("mptsas%d handle_topo_change to remove devhdl:%x, "
7204                    "rval:%x", mpt->m_instance, psmp->m_devhdl, rval));
7205                if (rval == DDI_SUCCESS) {
7206                        refhash_remove(mpt->m_smp_targets, psmp);
7207                } else {
7208                        psmp->m_devhdl = MPTSAS_INVALID_DEVHDL;
7209                }

7211                bzero(attached_wwnstr, sizeof (attached_wwnstr));

7213                break;
7214        }
7215        default:
7216                return;
7217        }
7218 }

7220 /*
7221  * Record the event if its type is enabled in mpt instance by ioctl.
7222  */
7223 static void
7224 mptsas_record_event(void *args)
7225 {
7226        m_replyh_arg_t                *replyh_arg;
7227        pMpi2EventNotificationReply_t   eventreply;
7228        uint32_t                      event, rfm;
7229        mptsas_t                      *mpt;
7230        int                           i, j;
7231        uint16_t                      event_data_len;
7232        boolean_t                     sendAEN = FALSE;

7234        replyh_arg = (m_replyh_arg_t *)args;
7235        rfm = replyh_arg->rfm;
7236        mpt = replyh_arg->mpt;

7238        eventreply = (pMpi2EventNotificationReply_t)
7239            (mpt->m_reply_frame + (rfm -
7240            (mpt->m_reply_frame_dma_addr&0xffffffffu)));
5000            (mpt->m_reply_frame + (rfm - mpt->m_reply_frame_dma_addr));
7241        event = ddi_get16(mpt->m_acc_reply_frame_hdl, &eventreply->Event);


7244        /*
7245         * Generate a system event to let anyone who cares know that a
7246         * LOG_ENTRY_ADDED event has occurred.  This is sent no matter what the
7247         * event mask is set to.
7248         */
7249        if (event == MPI2_EVENT_LOG_ENTRY_ADDED) {
7250                sendAEN = TRUE;
7251        }

7253        /*
7254         * Record the event only if it is not masked.  Determine which dword
7255         * and bit of event mask to test.
7256         */
7257        i = (uint8_t)(event / 32);
```



```
7258        j = (uint8_t)(event % 32);
7259        if ((i < 4) && ((1 << j) & mpt->m_event_mask[i])) {
7260                i = mpt->m_event_index;
7261                mpt->m_events[i].Type = event;
7262                mpt->m_events[i].Number = ++mpt->m_event_number;
7263                bzero(mpt->m_events[i].Data, MPTSAS_MAX_EVENT_DATA_LENGTH * 4);
7264                event_data_len = ddi_get16(mpt->m_acc_reply_frame_hdl,
7265                    &eventreply->EventDataLength);

7267                if (event_data_len > 0) {
7268                        /*
7269                         * Limit data to size in m_event entry
7270                         */
7271                        if (event_data_len > MPTSAS_MAX_EVENT_DATA_LENGTH) {
7272                                event_data_len = MPTSAS_MAX_EVENT_DATA_LENGTH;
7273                        }
7274                        for (j = 0; j < event_data_len; j++) {
7275                                mpt->m_events[i].Data[j] =
7276                                    ddi_get32(mpt->m_acc_reply_frame_hdl,
7277                                    &(eventreply->EventData[j]));
7278                        }

7280                        /*
7281                         * check for index wrap-around
7282                         */
7283                        if (++i == MPTSAS_EVENT_QUEUE_SIZE) {
7284                                i = 0;
7285                        }
7286                        mpt->m_event_index = (uint8_t)i;

7288                        /*
7289                         * Set flag to send the event.
7290                         */
7291                        sendAEN = TRUE;
7292                }
7293        }

7295        /*
7296         * Generate a system event if flag is set to let anyone who cares know
7297         * that an event has occurred.
7298         */
7299        if (sendAEN) {
7300                (void) ddi_log_sysevent(mpt->m_dip, DDI_VENDOR_LSI, "MPT_SAS",
7301                    "SAS", NULL, NULL, DDI_NOSLEEP);
7302        }
7303 }

7305 #define SMP_RESET_IN_PROGRESS MPI2_EVENT_SAS_TOPO_LR_SMP_RESET_IN_PROGRESS
7306 /*
7307  * handle sync events from ioc in interrupt
7308  * return value:
7309  * DDI_SUCCESS: The event is handled by this func
7310  * DDI_FAILURE: Event is not handled
7311  */
7312 static int
7313 mptsas_handle_event_sync(void *args)
7314 {
7315        m_replyh_arg_t                *replyh_arg;
7316        pMpi2EventNotificationReply_t   eventreply;
7317        uint32_t                      event, rfm;
7318        mptsas_t                      *mpt;
7319        uint_t                        iocstatus;

7321        replyh_arg = (m_replyh_arg_t *)args;
7322        rfm = replyh_arg->rfm;
7323        mpt = replyh_arg->mpt;
```

```
7325         ASSERT(mutex_owned(&mpt->m_mutex));

7327         eventreply = (pMpi2EventNotificationReply_t)
7328             (mpt->m_reply_frame + (rfm -
7329             (mpt->m_reply_frame_dma_addr&0xffffffffu)));
5088             (mpt->m_reply_frame + (rfm - mpt->m_reply_frame_dma_addr));
7330         event = ddi_get16(mpt->m_acc_reply_frame_hdl, &eventreply->Event);

7332         if (iocstatus = ddi_get16(mpt->m_acc_reply_frame_hdl,
7333             &eventreply->IOCStatus)) {
7334                 if (iocstatus == MPI2_IOCSTATUS_FLAG_LOG_INFO_AVAILABLE) {
7335                         mptsas_log(mpt, CE_WARN,
7336                             "!mptsas_handle_event_sync: event 0x%x, "
7337                             "IOCStatus=0x%x, "
7338                             "IOCLogInfo=0x%x", event, iocstatus,
5095                             "!mptsas_handle_event_sync: IOCStatus=0x%x, "
5096                             "IOCLogInfo=0x%x", iocstatus,
7339                             ddi_get32(mpt->m_acc_reply_frame_hdl,
7340                             &eventreply->IOCLogInfo));
7341                 } else {
7342                         mptsas_log(mpt, CE_WARN,
7343                             "mptsas_handle_event_sync: event 0x%x, "
7344                             "IOCStatus=0x%x, "
7345                             "(IOCLogInfo=0x%x)", event, iocstatus,
5101                             "mptsas_handle_event_sync: IOCStatus=0x%x, "
5102                             "IOCLogInfo=0x%x", iocstatus,
7346                             ddi_get32(mpt->m_acc_reply_frame_hdl,
7347                             &eventreply->IOCLogInfo));
7348                 }
7349         }

7351         /*
7352          * figure out what kind of event we got and handle accordingly
7353          */
7354         switch (event) {
7355         case MPI2_EVENT_SAS_TOPOLOGY_CHANGE_LIST:
7356         {
7357                 pMpi2EventDataSasTopologyChangeList_t   sas_topo_change_list;
7358                 uint8_t                         num_entries, expstatus, phy;
7359                 uint8_t                         phystatus, physport, state, i;
7360                 uint8_t                         start_phy_num, link_rate;
7361                 uint16_t                        dev_handle, reason_code;
7362                 uint16_t                        enc_handle, expd_handle;
7363                 char                            string[80], curr[80], prev[80];
7364                 mptsas_topo_change_list_t       *topo_head = NULL;
7365                 mptsas_topo_change_list_t       *topo_tail = NULL;
7366                 mptsas_topo_change_list_t       *topo_node = NULL;
7367                 mptsas_target_t                 *ptgt;
7368                 mptsas_smp_t                    *psmp;
7369                 uint8_t                         flags = 0, exp_flag;
7370                 smhba_info_t                    *pSmhba = NULL;

7372                 NDBG20(("mptsas_handle_event_sync: SAS topology change"));

7374                 sas_topo_change_list = (pMpi2EventDataSasTopologyChangeList_t)
7375                     eventreply->EventData;

7377                 enc_handle = ddi_get16(mpt->m_acc_reply_frame_hdl,
7378                     &sas_topo_change_list->EnclosureHandle);
7379                 expd_handle = ddi_get16(mpt->m_acc_reply_frame_hdl,
7380                     &sas_topo_change_list->ExpanderDevHandle);
7381                 num_entries = ddi_get8(mpt->m_acc_reply_frame_hdl,
7382                     &sas_topo_change_list->NumEntries);
7383                 start_phy_num = ddi_get8(mpt->m_acc_reply_frame_hdl,
7384                     &sas_topo_change_list->StartPhyNum);
```



```
7385                 expstatus = ddi_get8(mpt->m_acc_reply_frame_hdl,
7386                     &sas_topo_change_list->ExpStatus);
7387                 physport = ddi_get8(mpt->m_acc_reply_frame_hdl,
7388                     &sas_topo_change_list->PhysicalPort);

7390                 string[0] = 0;
7391                 if (expd_handle) {
7392                         flags = MPTSAS_TOPO_FLAG_EXPANDER_ASSOCIATED;
7393                         switch (expstatus) {
7394                         case MPI2_EVENT_SAS_TOPO_ES_ADDED:
7395                                 (void) sprintf(string, " added");
7396                                 /*
7397                                  * New expander device added
7398                                  */
7399                                 mpt->m_port_chng = 1;
7400                                 topo_node = kmem_zalloc(
7401                                     sizeof (mptsas_topo_change_list_t),
7402                                     KM_SLEEP);
7403                                 topo_node->mpt = mpt;
7404                                 topo_node->event = MPTSAS_DR_EVENT_RECONFIG_SMP;
7405                                 topo_node->un.physport = physport;
7406                                 topo_node->devhdl = expd_handle;
7407                                 topo_node->flags = flags;
7408                                 topo_node->object = NULL;
7409                                 if (topo_head == NULL) {
7410                                         topo_head = topo_tail = topo_node;
7411                                 } else {
7412                                         topo_tail->next = topo_node;
7413                                         topo_tail = topo_node;
7414                                 }
7415                                 break;
7416                         case MPI2_EVENT_SAS_TOPO_ES_NOT_RESPONDING:
7417                                 (void) sprintf(string, " not responding, "
7418                                     "removed");
7419                                 psmp = refhash_linear_search(mpt->m_smp_targets,
7420                                     mptsas_smp_eval_devhdl, &expd_handle);
7421                                 if (psmp == NULL)
7422                                         break;

7424                                 topo_node = kmem_zalloc(
7425                                     sizeof (mptsas_topo_change_list_t),
7426                                     KM_SLEEP);
7427                                 topo_node->mpt = mpt;
7428                                 topo_node->un.phymask =
7429                                     psmp->m_addr.mta_phymask;
7430                                 topo_node->event = MPTSAS_DR_EVENT_OFFLINE_SMP;
7431                                 topo_node->devhdl = expd_handle;
7432                                 topo_node->flags = flags;
7433                                 topo_node->object = NULL;
7434                                 if (topo_head == NULL) {
7435                                         topo_head = topo_tail = topo_node;
7436                                 } else {
7437                                         topo_tail->next = topo_node;
7438                                         topo_tail = topo_node;
7439                                 }
7440                                 break;
7441                         case MPI2_EVENT_SAS_TOPO_ES_RESPONDING:
7442                                 break;
7443                         case MPI2_EVENT_SAS_TOPO_ES_DELAY_NOT_RESPONDING:
7444                                 (void) sprintf(string, " not responding, "
7445                                     "delaying removal");
7446                                 break;
7447                         default:
7448                                 break;
7449                         }
7450                 } else {
```

```
7451                    flags = MPTSAS_TOPO_FLAG_DIRECT_ATTACHED_DEVICE;
7452            }

7454            NDBG20(("SAS TOPOLOGY CHANGE for enclosure %x expander %x%s\n",
7455                enc_handle, expd_handle, string));
7456            for (i = 0; i < num_entries; i++) {
7457                    phy = i + start_phy_num;
7458                    phystatus = ddi_get8(mpt->m_acc_reply_frame_hdl,
7459                        &sas_topo_change_list->PHY[i].PhyStatus);
7460                    dev_handle = ddi_get16(mpt->m_acc_reply_frame_hdl,
7461                        &sas_topo_change_list->PHY[i].AttachedDevHandle);
7462                    reason_code = phystatus & MPI2_EVENT_SAS_TOPO_RC_MASK;
7463                    /*
7464                     * Filter out processing of Phy Vacant Status unless
7465                     * the reason code is "Not Responding".  Process all
7466                     * other combinations of Phy Status and Reason Codes.
7467                     */
7468                    if ((phystatus &
7469                        MPI2_EVENT_SAS_TOPO_PHYSTATUS_VACANT) &&
7470                        (reason_code !=
7471                        MPI2_EVENT_SAS_TOPO_RC_TARG_NOT_RESPONDING)) {
7472                            continue;
7473                    }
7474                    curr[0] = 0;
7475                    prev[0] = 0;
7476                    string[0] = 0;
7477                    switch (reason_code) {
7478                    case MPI2_EVENT_SAS_TOPO_RC_TARG_ADDED:
7479                    {
7480                            NDBG20(("mptsas3%d phy %d physical_port %d "
5237                            NDBG20(("mptsas%d phy %d physical_port %d "
7481                                "dev_handle %d added", mpt->m_instance, phy,
7482                                physport, dev_handle));
7483                            link_rate = ddi_get8(mpt->m_acc_reply_frame_hdl,
7484                                &sas_topo_change_list->PHY[i].LinkRate);
7485                            state = (link_rate &
7486                                MPI2_EVENT_SAS_TOPO_LR_CURRENT_MASK) >>
7487                                MPI2_EVENT_SAS_TOPO_LR_CURRENT_SHIFT;
7488                            switch (state) {
7489                            case MPI2_EVENT_SAS_TOPO_LR_PHY_DISABLED:
7490                                    (void) sprintf(curr, "is disabled");
7491                                    break;
7492                            case MPI2_EVENT_SAS_TOPO_LR_NEGOTIATION_FAILED:
7493                                    (void) sprintf(curr, "is offline, "
7494                                        "failed speed negotiation");
7495                                    break;
7496                            case MPI2_EVENT_SAS_TOPO_LR_SATA_OOB_COMPLETE:
7497                                    (void) sprintf(curr, "SATA OOB "
7498                                        "complete");
7499                                    break;
7500                            case SMP_RESET_IN_PROGRESS:
7501                                    (void) sprintf(curr, "SMP reset in "
7502                                        "progress");
7503                                    break;
7504                            case MPI2_EVENT_SAS_TOPO_LR_RATE_1_5:
7505                                    (void) sprintf(curr, "is online at "
7506                                        "1.5 Gbps");
7507                                    break;
7508                            case MPI2_EVENT_SAS_TOPO_LR_RATE_3_0:
7509                                    (void) sprintf(curr, "is online at 3.0 "
7510                                        "Gbps");
7511                                    break;
7512                            case MPI2_EVENT_SAS_TOPO_LR_RATE_6_0:
7513                                    (void) sprintf(curr, "is online at 6.0 "
7514                                        "Gbps");
7515                                    break;
```



```
7516                            case MPI25_EVENT_SAS_TOPO_LR_RATE_12_0:
7517                                    (void) sprintf(curr,
7518                                        "is online at 12.0 Gbps");
7519                                    break;
7520 #endif /* ! codereview */
7521                            default:
7522                                    (void) sprintf(curr, "state is "
7523                                        "unknown");
7524                                    break;
7525                            }
7526                            /*
7527                             * New target device added into the system.
7528                             * Set association flag according to if an
7529                             * expander is used or not.
7530                             */
7531                            exp_flag =
7532                                MPTSAS_TOPO_FLAG_EXPANDER_ATTACHED_DEVICE;
7533                            if (flags ==
7534                                MPTSAS_TOPO_FLAG_EXPANDER_ASSOCIATED) {
7535                                    flags = exp_flag;
7536                            }
7537                            topo_node = kmem_zalloc(
7538                                sizeof (mptsas_topo_change_list_t),
7539                                KM_SLEEP);
7540                            topo_node->mpt = mpt;
7541                            topo_node->event =
7542                                MPTSAS_DR_EVENT_RECONFIG_TARGET;
7543                            if (expd_handle == 0) {
7544                                    /*
7545                                     * Per MPI 2, if expander dev handle
7546                                     * is 0, it's a directly attached
7547                                     * device. So driver use PHY to decide
7548                                     * which iport is associated
7549                                     */
7550                                    physport = phy;
7551                                    mpt->m_port_chng = 1;
7552                            }
7553                            topo_node->un.physport = physport;
7554                            topo_node->devhdl = dev_handle;
7555                            topo_node->flags = flags;
7556                            topo_node->object = NULL;
7557                            if (topo_head == NULL) {
7558                                    topo_head = topo_tail = topo_node;
7559                            } else {
7560                                    topo_tail->next = topo_node;
7561                                    topo_tail = topo_node;
7562                            }
7563                            break;
7564                    }
7565                    case MPI2_EVENT_SAS_TOPO_RC_TARG_NOT_RESPONDING:
7566                    {
7567                            NDBG20(("mptsas3%d phy %d physical_port %d "
5273                            NDBG20(("mptsas%d phy %d physical_port %d "
7568                                "dev_handle %d removed", mpt->m_instance,
7569                                phy, physport, dev_handle));
7570                            /*
7571                             * Set association flag according to if an
7572                             * expander is used or not.
7573                             */
7574                            exp_flag =
7575                                MPTSAS_TOPO_FLAG_EXPANDER_ATTACHED_DEVICE;
7576                            if (flags ==
7577                                MPTSAS_TOPO_FLAG_EXPANDER_ASSOCIATED) {
7578                                    flags = exp_flag;
7579                            }
7580                            /*
```

```
7581                                 * Target device is removed from the system
7582                                 * Before the device is really offline from
7583                                 * from system.
7584                                 */
7585                                ptgt = refhash_linear_search(mpt->m_targets,
7586                                    mptsas_target_eval_devhdl, &dev_handle);
7587                                /*
7588                                 * If ptgt is NULL here, it means that the
7589                                 * DevHandle is not in the hash table.  This is
7590                                 * reasonable sometimes.  For example, if a
7591                                 * disk was pulled, then added, then pulled
7592                                 * again, the disk will not have been put into
7593                                 * the hash table because the add event will
7594                                 * have an invalid phymask.  BUT, this does not
7595                                 * mean that the DevHandle is invalid.  The
7596                                 * controller will still have a valid DevHandle
7597                                 * that must be removed.  To do this, use the
7598                                 * MPTSAS_DR_EVENT_REMOVE_HANDLE event.
5304                                 * MPTSAS_TOPO_FLAG_REMOVE_HANDLE event.
7599                                 */
7600                                if (ptgt == NULL) {
7601                                        topo_node = kmem_zalloc(
7602                                            sizeof (mptsas_topo_change_list_t),
7603                                            KM_SLEEP);
7604                                        topo_node->mpt = mpt;
7605                                        topo_node->un.phymask = 0;
7606                                        topo_node->event =
7607                                            MPTSAS_DR_EVENT_REMOVE_HANDLE;
5313                                        MPTSAS_TOPO_FLAG_REMOVE_HANDLE;
7608                                        topo_node->devhdl = dev_handle;
7609                                        topo_node->flags = flags;
7610                                        topo_node->object = NULL;
7611                                        if (topo_head == NULL) {
7612                                                topo_head = topo_tail =
7613                                                    topo_node;
7614                                        } else {
7615                                                topo_tail->next = topo_node;
7616                                                topo_tail = topo_node;
7617                                        }
7618                                        break;
7619                                }

7621                                /*
7622                                 * Update DR flag immediately avoid I/O failure
7623                                 * before failover finish. We won't add
7624                                 * any following commands into waitq, instead,
5329                                 * before failover finish. Pay attention to the
5330                                 * mutex protect, we need grab m_tx_waitq_mutex
5331                                 * during set m_dr_flag because we won't add
5332                                 * the following command into waitq, instead,
7625                                 * we need return TRAN_BUSY in the tran_start
7626                                 * context.
7627                                 */
5336                                mutex_enter(&mpt->m_tx_waitq_mutex);
7628                                ptgt->m_dr_flag = MPTSAS_DR_INTRANSITION;
5338                                mutex_exit(&mpt->m_tx_waitq_mutex);

7630                                topo_node = kmem_zalloc(
7631                                    sizeof (mptsas_topo_change_list_t),
7632                                    KM_SLEEP);
7633                                topo_node->mpt = mpt;
7634                                topo_node->un.phymask =
7635                                    ptgt->m_addr.mta_phymask;
7636                                topo_node->event =
7637                                    MPTSAS_DR_EVENT_OFFLINE_TARGET;
7638                                topo_node->devhdl = dev_handle;
```



```
7639                                        topo_node->flags = flags;
7640                                        topo_node->object = NULL;
7641                                        if (topo_head == NULL) {
7642                                                topo_head = topo_tail = topo_node;
7643                                        } else {
7644                                                topo_tail->next = topo_node;
7645                                                topo_tail = topo_node;
7646                                        }
7647                                        break;
7648                                }
7649                                case MPI2_EVENT_SAS_TOPO_RC_PHY_CHANGED:
7650                                        link_rate = ddi_get8(mpt->m_acc_reply_frame_hdl,
7651                                            &sas_topo_change_list->PHY[i].LinkRate);
7652                                        state = (link_rate &
7653                                            MPI2_EVENT_SAS_TOPO_LR_CURRENT_MASK) >>
7654                                            MPI2_EVENT_SAS_TOPO_LR_CURRENT_SHIFT;
7655                                        pSmhba = &mpt->m_phy_info[i].smhba_info;
7656                                        pSmhba->negotiated_link_rate = state;
7657                                        switch (state) {
7658                                        case MPI2_EVENT_SAS_TOPO_LR_PHY_DISABLED:
7659                                                (void) sprintf(curr, "is disabled");
7660                                                mptsas_smhba_log_sysevent(mpt,
7661                                                    ESC_SAS_PHY_EVENT,
7662                                                    SAS_PHY_REMOVE,
7663                                                    &mpt->m_phy_info[i].smhba_info);
7664                                                mpt->m_phy_info[i].smhba_info.
7665                                                    negotiated_link_rate
7666                                                    = 0x1;
7667                                                break;
7668                                        case MPI2_EVENT_SAS_TOPO_LR_NEGOTIATION_FAILED:
7669                                                (void) sprintf(curr, "is offline, "
7670                                                    "failed speed negotiation");
7671                                                mptsas_smhba_log_sysevent(mpt,
7672                                                    ESC_SAS_PHY_EVENT,
7673                                                    SAS_PHY_OFFLINE,
7674                                                    &mpt->m_phy_info[i].smhba_info);
7675                                                break;
7676                                        case MPI2_EVENT_SAS_TOPO_LR_SATA_OOB_COMPLETE:
7677                                                (void) sprintf(curr, "SATA OOB "
7678                                                    "complete");
7679                                                break;
7680                                        case SMP_RESET_IN_PROGRESS:
7681                                                (void) sprintf(curr, "SMP reset in "
7682                                                    "progress");
7683                                                break;
7684                                        case MPI2_EVENT_SAS_TOPO_LR_RATE_1_5:
7685                                                (void) sprintf(curr, "is online at "
7686                                                    "1.5 Gbps");
7687                                                if ((expd_handle == 0) &&
7688                                                    (enc_handle == 1)) {
7689                                                        mpt->m_port_chng = 1;
7690                                                }
7691                                                mptsas_smhba_log_sysevent(mpt,
7692                                                    ESC_SAS_PHY_EVENT,
7693                                                    SAS_PHY_ONLINE,
7694                                                    &mpt->m_phy_info[i].smhba_info);
7695                                                break;
7696                                        case MPI2_EVENT_SAS_TOPO_LR_RATE_3_0:
7697                                                (void) sprintf(curr, "is online at 3.0 "
7698                                                    "Gbps");
7699                                                if ((expd_handle == 0) &&
7700                                                    (enc_handle == 1)) {
7701                                                        mpt->m_port_chng = 1;
7702                                                }
7703                                                mptsas_smhba_log_sysevent(mpt,
7704                                                    ESC_SAS_PHY_EVENT,
```

```
7705                                        SAS_PHY_ONLINE,
7706                                        &mpt->m_phy_info[i].smhba_info);
7707                                    break;
7708                                case MPI2_EVENT_SAS_TOPO_LR_RATE_6_0:
7709                                    (void) sprintf(curr, "is online at "
7710                                        "6.0 Gbps");
7711                                    if ((expd_handle == 0) &&
7712                                        (enc_handle == 1)) {
7713                                            mpt->m_port_chng = 1;
7714                                    }
7715                                    mptsas_smhba_log_sysevent(mpt,
7716                                        ESC_SAS_PHY_EVENT,
7717                                        SAS_PHY_ONLINE,
7718                                        &mpt->m_phy_info[i].smhba_info);
7719                                    break;
7720                                case MPI25_EVENT_SAS_TOPO_LR_RATE_12_0:
7721                                    (void) sprintf(curr, "is online at "
7722                                        "12.0 Gbps");
7723                                    if ((expd_handle == 0) &&
7724                                        (enc_handle == 1)) {
7725                                            mpt->m_port_chng = 1;
7726                                    }
7727                                    mptsas_smhba_log_sysevent(mpt,
7728                                        ESC_SAS_PHY_EVENT,
7729                                        SAS_PHY_ONLINE,
7730                                        &mpt->m_phy_info[i].smhba_info);
7731                                    break;
7732 #endif /* ! codereview */
7733                                default:
7734                                    (void) sprintf(curr, "state is "
7735                                        "unknown");
7736                                    break;
7737                                }

7739                                state = (link_rate &
7740                                    MPI2_EVENT_SAS_TOPO_LR_PREV_MASK) >>
7741                                    MPI2_EVENT_SAS_TOPO_LR_PREV_SHIFT;
7742                                switch (state) {
7743                                case MPI2_EVENT_SAS_TOPO_LR_PHY_DISABLED:
7744                                    (void) sprintf(prev, ", was disabled");
7745                                    break;
7746                                case MPI2_EVENT_SAS_TOPO_LR_NEGOTIATION_FAILED:
7747                                    (void) sprintf(prev, ", was offline, "
7748                                        "failed speed negotiation");
7749                                    break;
7750                                case MPI2_EVENT_SAS_TOPO_LR_SATA_OOB_COMPLETE:
7751                                    (void) sprintf(prev, ", was SATA OOB "
7752                                        "complete");
7753                                    break;
7754                                case SMP_RESET_IN_PROGRESS:
7755                                    (void) sprintf(prev, ", was SMP reset "
7756                                        "in progress");
7757                                    break;
7758                                case MPI2_EVENT_SAS_TOPO_LR_RATE_1_5:
7759                                    (void) sprintf(prev, ", was online at "
7760                                        "1.5 Gbps");
7761                                    break;
7762                                case MPI2_EVENT_SAS_TOPO_LR_RATE_3_0:
7763                                    (void) sprintf(prev, ", was online at "
7764                                        "3.0 Gbps");
7765                                    break;
7766                                case MPI2_EVENT_SAS_TOPO_LR_RATE_6_0:
7767                                    (void) sprintf(prev, ", was online at "
7768                                        "6.0 Gbps");
7769                                    break;
7770                                case MPI25_EVENT_SAS_TOPO_LR_RATE_12_0:
```



```
7771                                        (void) sprintf(prev, ", was online at "
7772                                            "12.0 Gbps");
7773                                        break;
7774 #endif /* ! codereview */
7775                                    default:
7776                                    break;
7777                                    }
7778                                    (void) sprintf(&string[strlen(string)], "link "
7779                                        "changed, ");
7780                                    break;
7781                                case MPI2_EVENT_SAS_TOPO_RC_NO_CHANGE:
7782                                    continue;
7783                                case MPI2_EVENT_SAS_TOPO_RC_DELAY_NOT_RESPONDING:
7784                                    (void) sprintf(&string[strlen(string)],
7785                                        "target not responding, delaying "
7786                                        "removal");
7787                                    break;
7788                                }
7789                                NDBG20(("mptsas3%d phy %d DevHandle %x, %s%s%s\n",
5430                                NDBG20(("mptsas%d phy %d DevHandle %x, %s%s%s\n",
7790                                    mpt->m_instance, phy, dev_handle, string, curr,
7791                                    prev));
7792                            }
7793                            if (topo_head != NULL) {
7794                                    /*
7795                                     * Launch DR taskq to handle topology change
7796                                     */
7797                                    if ((ddi_taskq_dispatch(mpt->m_dr_taskq,
7798                                        mptsas_handle_dr, (void *)topo_head,
7799                                        DDI_NOSLEEP)) != DDI_SUCCESS) {
7800                                        while (topo_head != NULL) {
7801                                            topo_node = topo_head;
7802                                            topo_head = topo_head->next;
7803                                            kmem_free(topo_node,
7804                                                sizeof (mptsas_topo_change_list_t));
7805                                        }
7806 #endif /* ! codereview */
7807                                        mptsas_log(mpt, CE_NOTE, "mptsas start taskq "
7808                                            "for handle SAS DR event failed. \n");
7809                                    }
7810                            }
7811                            break;
7812                    }
7813                    case MPI2_EVENT_IR_CONFIGURATION_CHANGE_LIST:
7814                    {
7815                            Mpi2EventDataIrConfigChangeList_t       *irChangeList;
7816                            mptsas_topo_change_list_t               *topo_head = NULL;
7817                            mptsas_topo_change_list_t               *topo_tail = NULL;
7818                            mptsas_topo_change_list_t               *topo_node = NULL;
7819                            mptsas_target_t                         *ptgt;
7820                            uint8_t                                 num_entries, i, reason;
7821                            uint16_t                                volhandle, diskhandle;

7823                            irChangeList = (pMpi2EventDataIrConfigChangeList_t)
7824                                eventreply->EventData;
7825                            num_entries = ddi_get8(mpt->m_acc_reply_frame_hdl,
7826                                &irChangeList->NumElements);

7828                            NDBG20(("mptsas3%d IR_CONFIGURATION_CHANGE_LIST event received",
5441                            NDBG20(("mptsas%d IR_CONFIGURATION_CHANGE_LIST event received",
7829                                mpt->m_instance));

7831                            for (i = 0; i < num_entries; i++) {
7832                                    reason = ddi_get8(mpt->m_acc_reply_frame_hdl,
7833                                        &irChangeList->ConfigElement[i].ReasonCode);
7834                                    volhandle = ddi_get16(mpt->m_acc_reply_frame_hdl,
```

```
7835                            &irChangeList->ConfigElement[i].VolDevHandle);
7836                    diskhandle = ddi_get16(mpt->m_acc_reply_frame_hdl,
7837                        &irChangeList->ConfigElement[i].PhysDiskDevHandle);

7839                    switch (reason) {
7840                    case MPI2_EVENT_IR_CHANGE_RC_ADDED:
7841                    case MPI2_EVENT_IR_CHANGE_RC_VOLUME_CREATED:
7842                    {
7843                            NDBG20(("mptsas %d volume added\n",
7844                                mpt->m_instance));

7846                            topo_node = kmem_zalloc(
7847                                sizeof (mptsas_topo_change_list_t),
7848                                KM_SLEEP);

7850                            topo_node->mpt = mpt;
7851                            topo_node->event =
7852                                MPTSAS_DR_EVENT_RECONFIG_TARGET;
7853                            topo_node->un.physport = 0xff;
7854                            topo_node->devhdl = volhandle;
7855                            topo_node->flags =
7856                                MPTSAS_TOPO_FLAG_RAID_ASSOCIATED;
7857                            topo_node->object = NULL;
7858                            if (topo_head == NULL) {
7859                                    topo_head = topo_tail = topo_node;
7860                            } else {
7861                                    topo_tail->next = topo_node;
7862                                    topo_tail = topo_node;
7863                            }
7864                            break;
7865                    }
7866                    case MPI2_EVENT_IR_CHANGE_RC_REMOVED:
7867                    case MPI2_EVENT_IR_CHANGE_RC_VOLUME_DELETED:
7868                    {
7869                            NDBG20(("mptsas %d volume deleted\n",
7870                                mpt->m_instance));
7871                            ptgt = refhash_linear_search(mpt->m_targets,
7872                                mptsas_target_eval_devhdl, &volhandle);
7873                            if (ptgt == NULL)
7874                                    break;

7876                            /*
7877                             * Clear any flags related to volume
7878                             */
7879                            (void) mptsas_delete_volume(mpt, volhandle);

7881                            /*
7882                             * Update DR flag immediately avoid I/O failure
7883                             */
5497                            mutex_enter(&mpt->m_tx_waitq_mutex);
7884                            ptgt->m_dr_flag = MPTSAS_DR_INTRANSITION;
5499                            mutex_exit(&mpt->m_tx_waitq_mutex);

7886                            topo_node = kmem_zalloc(
7887                                sizeof (mptsas_topo_change_list_t),
7888                                KM_SLEEP);
7889                            topo_node->mpt = mpt;
7890                            topo_node->un.phymask =
7891                                ptgt->m_addr.mta_phymask;
7892                            topo_node->event =
7893                                MPTSAS_DR_EVENT_OFFLINE_TARGET;
7894                            topo_node->devhdl = volhandle;
7895                            topo_node->flags =
7896                                MPTSAS_TOPO_FLAG_RAID_ASSOCIATED;
7897                            topo_node->object = (void *)ptgt;
7898                            if (topo_head == NULL) {
```



```
7899                                    topo_head = topo_tail = topo_node;
7900                            } else {
7901                                    topo_tail->next = topo_node;
7902                                    topo_tail = topo_node;
7903                            }
7904                            break;
7905                    }
7906                    case MPI2_EVENT_IR_CHANGE_RC_PD_CREATED:
7907                    case MPI2_EVENT_IR_CHANGE_RC_HIDE:
7908                    {
7909                            ptgt = refhash_linear_search(mpt->m_targets,
7910                                mptsas_target_eval_devhdl, &diskhandle);
7911                            if (ptgt == NULL)
7912                                    break;

7914                            /*
7915                             * Update DR flag immediately avoid I/O failure
7916                             */
5532                            mutex_enter(&mpt->m_tx_waitq_mutex);
7917                            ptgt->m_dr_flag = MPTSAS_DR_INTRANSITION;
5534                            mutex_exit(&mpt->m_tx_waitq_mutex);

7919                            topo_node = kmem_zalloc(
7920                                sizeof (mptsas_topo_change_list_t),
7921                                KM_SLEEP);
7922                            topo_node->mpt = mpt;
7923                            topo_node->un.phymask =
7924                                ptgt->m_addr.mta_phymask;
7925                            topo_node->event =
7926                                MPTSAS_DR_EVENT_OFFLINE_TARGET;
7927                            topo_node->devhdl = diskhandle;
7928                            topo_node->flags =
7929                                MPTSAS_TOPO_FLAG_RAID_PHYSDRV_ASSOCIATED;
7930                            topo_node->object = (void *)ptgt;
7931                            if (topo_head == NULL) {
7932                                    topo_head = topo_tail = topo_node;
7933                            } else {
7934                                    topo_tail->next = topo_node;
7935                                    topo_tail = topo_node;
7936                            }
7937                            break;
7938                    }
7939                    case MPI2_EVENT_IR_CHANGE_RC_UNHIDE:
7940                    case MPI2_EVENT_IR_CHANGE_RC_PD_DELETED:
7941                    {
7942                            /*
7943                             * The physical drive is released by a IR
7944                             * volume. But we cannot get the the physport
7945                             * or phynum from the event data, so we only
7946                             * can get the physport/phynum after SAS
7947                             * Device Page0 request for the devhdl.
7948                             */
7949                            topo_node = kmem_zalloc(
7950                                sizeof (mptsas_topo_change_list_t),
7951                                KM_SLEEP);
7952                            topo_node->mpt = mpt;
7953                            topo_node->un.phymask = 0;
7954                            topo_node->event =
7955                                MPTSAS_DR_EVENT_RECONFIG_TARGET;
7956                            topo_node->devhdl = diskhandle;
7957                            topo_node->flags =
7958                                MPTSAS_TOPO_FLAG_RAID_PHYSDRV_ASSOCIATED;
7959                            topo_node->object = NULL;
7960                            mpt->m_port_chng = 1;
7961                            if (topo_head == NULL) {
7962                                    topo_head = topo_tail = topo_node;
```

```
7963                                } else {
7964                                        topo_tail->next = topo_node;
7965                                        topo_tail = topo_node;
7966                                }
7967                                break;
7968                        }
7969                        default:
7970                                break;
7971                        }
7972                }

7974                if (topo_head != NULL) {
7975                        /*
7976                         * Launch DR taskq to handle topology change
7977                         */
7978                        if ((ddi_taskq_dispatch(mpt->m_dr_taskq,
7979                            mptsas_handle_dr, (void *)topo_head,
7980                            DDI_NOSLEEP)) != DDI_SUCCESS) {
7981                                while (topo_head != NULL) {
7982                                        topo_node = topo_head;
7983                                        topo_head = topo_head->next;
7984                                        kmem_free(topo_node,
7985                                            sizeof (mptsas_topo_change_list_t));
7986                                }
7987 #endif /* ! codereview */
7988                                mptsas_log(mpt, CE_NOTE, "mptsas start taskq "
7989                                    "for handle SAS DR event failed. \n");
7990                        }
7991                }
7992                break;
7993        }
7994        default:
7995                return (DDI_FAILURE);
7996        }

7998        return (DDI_SUCCESS);
7999 }

8001 /*
8002  * handle events from ioc
8003  */
8004 static void
8005 mptsas_handle_event(void *args)
8006 {
8007        m_replyh_arg_t                  *replyh_arg;
8008        pMpi2EventNotificationReply_t   eventreply;
8009        uint32_t                        event, iocloginfo, rfm;
8010        uint32_t                        status;
8011        uint8_t                         port;
8012        mptsas_t                        *mpt;
8013        uint_t                          iocstatus;

8015        replyh_arg = (m_replyh_arg_t *)args;
8016        rfm = replyh_arg->rfm;
8017        mpt = replyh_arg->mpt;

8019        mutex_enter(&mpt->m_mutex);
8020        /*
8021         * If HBA is being reset, drop incoming event.
8022         */
8023        if (mpt->m_in_reset) {
8024                NDBG20(("dropping event received prior to reset"));
8025                mutex_exit(&mpt->m_mutex);
8026                return;
8027        }
```



```
8029        eventreply = (pMpi2EventNotificationReply_t)
8030            (mpt->m_reply_frame + (rfm -
8031            (mpt->m_reply_frame_dma_addr&0xffffffffu)));
5598            (mpt->m_reply_frame + (rfm - mpt->m_reply_frame_dma_addr));
8032        event = ddi_get16(mpt->m_acc_reply_frame_hdl, &eventreply->Event);

8034        if (iocstatus = ddi_get16(mpt->m_acc_reply_frame_hdl,
8035            &eventreply->IOCStatus)) {
8036                if (iocstatus == MPI2_IOCSTATUS_FLAG_LOG_INFO_AVAILABLE) {
8037                        mptsas_log(mpt, CE_WARN,
8038                            "!mptsas_handle_event: IOCStatus=0x%x, "
8039                            "IOCLogInfo=0x%x", iocstatus,
8040                            ddi_get32(mpt->m_acc_reply_frame_hdl,
8041                            &eventreply->IOCLogInfo));
8042                } else {
8043                        mptsas_log(mpt, CE_WARN,
8044                            "mptsas_handle_event: IOCStatus=0x%x, "
8045                            "IOCLogInfo=0x%x", iocstatus,
8046                            ddi_get32(mpt->m_acc_reply_frame_hdl,
8047                            &eventreply->IOCLogInfo));
8048                }
8049        }

8051        /*
8052         * figure out what kind of event we got and handle accordingly
8053         */
8054        switch (event) {
8055        case MPI2_EVENT_LOG_ENTRY_ADDED:
8056                break;
8057        case MPI2_EVENT_LOG_DATA:
8058                iocloginfo = ddi_get32(mpt->m_acc_reply_frame_hdl,
8059                    &eventreply->IOCLogInfo);
8060                NDBG20(("mptsas %d log info %x received.\n", mpt->m_instance,
8061                    iocloginfo));
8062                break;
8063        case MPI2_EVENT_STATE_CHANGE:
8064                NDBG20(("mptsas3%d state change.", mpt->m_instance));
5631                NDBG20(("mptsas%d state change.", mpt->m_instance));
8065                break;
8066        case MPI2_EVENT_HARD_RESET_RECEIVED:
8067                NDBG20(("mptsas3%d event change.", mpt->m_instance));
5634                NDBG20(("mptsas%d event change.", mpt->m_instance));
8068                break;
8069        case MPI2_EVENT_SAS_DISCOVERY:
8070        {
8071                MPI2_EVENT_DATA_SAS_DISCOVERY   *sasdiscovery;
8072                char                            string[80];
8073                uint8_t                         rc;

8075                sasdiscovery =
8076                    (pMpi2EventDataSasDiscovery_t)eventreply->EventData;

8078                rc = ddi_get8(mpt->m_acc_reply_frame_hdl,
8079                    &sasdiscovery->ReasonCode);
8080                port = ddi_get8(mpt->m_acc_reply_frame_hdl,
8081                    &sasdiscovery->PhysicalPort);
8082                status = ddi_get32(mpt->m_acc_reply_frame_hdl,
8083                    &sasdiscovery->DiscoveryStatus);

8085                string[0] = 0;
8086                switch (rc) {
8087                case MPI2_EVENT_SAS_DISC_RC_STARTED:
8088                        (void) sprintf(string, "STARTING");
8089                        break;
8090                case MPI2_EVENT_SAS_DISC_RC_COMPLETED:
8091                        (void) sprintf(string, "COMPLETED");
```

```
8092                        break;
8093                default:
8094                        (void) sprintf(string, "UNKNOWN");
8095                        break;
8096                }

8098                NDBG20(("SAS DISCOVERY is %s for port %d, status %x", string,
8099                    port, status));

8101                break;
8102        }
8103        case MPI2_EVENT_EVENT_CHANGE:
8104                NDBG20(("mptsas3%d event change.", mpt->m_instance));
5671                NDBG20(("mptsas%d event change.", mpt->m_instance));
8105                break;
8106        case MPI2_EVENT_TASK_SET_FULL:
8107        {
8108                pMpi2EventDataTaskSetFull_t     taskfull;

8110                taskfull = (pMpi2EventDataTaskSetFull_t)eventreply->EventData;

8112                NDBG20(("TASK_SET_FULL received for mptsas3%d, depth %d\n",
5679                NDBG20(("TASK_SET_FULL received for mptsas%d, depth %d\n",
8113                    mpt->m_instance,  ddi_get16(mpt->m_acc_reply_frame_hdl,
8114                    &taskfull->CurrentDepth)));
8115                break;
8116        }
8117        case MPI2_EVENT_SAS_TOPOLOGY_CHANGE_LIST:
8118        {
8119                /*
8120                 * SAS TOPOLOGY CHANGE LIST Event has already been handled
8121                 * in mptsas_handle_event_sync() of interrupt context
8122                 */
8123                break;
8124        }
8125        case MPI2_EVENT_SAS_ENCL_DEVICE_STATUS_CHANGE:
8126        {
8127                pMpi2EventDataSasEnclDevStatusChange_t  encstatus;
8128                uint8_t                                 rc;
8129                char                                    string[80];

8131                encstatus = (pMpi2EventDataSasEnclDevStatusChange_t)
8132                    eventreply->EventData;

8134                rc = ddi_get8(mpt->m_acc_reply_frame_hdl,
8135                    &encstatus->ReasonCode);
8136                switch (rc) {
8137                case MPI2_EVENT_SAS_ENCL_RC_ADDED:
8138                        (void) sprintf(string, "added");
8139                        break;
8140                case MPI2_EVENT_SAS_ENCL_RC_NOT_RESPONDING:
8141                        (void) sprintf(string, ", not responding");
8142                        break;
8143                default:
8144                break;
8145                }
8146                NDBG20(("mptsas3%d ENCLOSURE STATUS CHANGE for enclosure "
8147                    "%x%s\n", mpt->m_instance,
8148                    ddi_get16(mpt->m_acc_reply_frame_hdl,
5713                NDBG20(("mptsas%d ENCLOSURE STATUS CHANGE for enclosure %x%s\n",
5714                    mpt->m_instance, ddi_get16(mpt->m_acc_reply_frame_hdl,
8149                    &encstatus->EnclosureHandle), string));
8150                break;
8151        }

8153        /*
```



```
8154         * MPI2_EVENT_SAS_DEVICE_STATUS_CHANGE is handled by
8155         * mptsas_handle_event_sync,in here just send ack message.
8156         */
8157        case MPI2_EVENT_SAS_DEVICE_STATUS_CHANGE:
8158        {
8159                pMpi2EventDataSasDeviceStatusChange_t   statuschange;
8160                uint8_t                                 rc;
8161                uint16_t                                devhdl;
8162                uint64_t                                wwn = 0;
8163                uint32_t                                wwn_lo, wwn_hi;

8165                statuschange = (pMpi2EventDataSasDeviceStatusChange_t)
8166                    eventreply->EventData;
8167                rc = ddi_get8(mpt->m_acc_reply_frame_hdl,
8168                    &statuschange->ReasonCode);
8169                wwn_lo = ddi_get32(mpt->m_acc_reply_frame_hdl,
8170                    (uint32_t *)(void *)&statuschange->SASAddress);
8171                wwn_hi = ddi_get32(mpt->m_acc_reply_frame_hdl,
8172                    (uint32_t *)(void *)&statuschange->SASAddress + 1);
8173                wwn = ((uint64_t)wwn_hi << 32) | wwn_lo;
8174                devhdl =  ddi_get16(mpt->m_acc_reply_frame_hdl,
8175                    &statuschange->DevHandle);

8177                NDBG13(("MPI2_EVENT_SAS_DEVICE_STATUS_CHANGE wwn is %"PRIx64,
8178                    wwn));

8180                switch (rc) {
8181                case MPI2_EVENT_SAS_DEV_STAT_RC_SMART_DATA:
8182                        NDBG20(("SMART data received, ASC/ASCQ = %02x/%02x",
8183                            ddi_get8(mpt->m_acc_reply_frame_hdl,
8184                            &statuschange->ASC),
8185                            ddi_get8(mpt->m_acc_reply_frame_hdl,
8186                            &statuschange->ASCQ)));
8187                        break;

8189                case MPI2_EVENT_SAS_DEV_STAT_RC_UNSUPPORTED:
8190                        NDBG20(("Device not supported"));
8191                        break;

8193                case MPI2_EVENT_SAS_DEV_STAT_RC_INTERNAL_DEVICE_RESET:
8194                        NDBG20(("IOC internally generated the Target Reset "
8195                            "for devhdl:%x", devhdl));
8196                        break;

8198                case MPI2_EVENT_SAS_DEV_STAT_RC_CMP_INTERNAL_DEV_RESET:
8199                        NDBG20(("IOC's internally generated Target Reset "
8200                            "completed for devhdl:%x", devhdl));
8201                        break;

8203                case MPI2_EVENT_SAS_DEV_STAT_RC_TASK_ABORT_INTERNAL:
8204                        NDBG20(("IOC internally generated Abort Task"));
8205                        break;

8207                case MPI2_EVENT_SAS_DEV_STAT_RC_CMP_TASK_ABORT_INTERNAL:
8208                        NDBG20(("IOC's internally generated Abort Task "
8209                            "completed"));
8210                        break;

8212                case MPI2_EVENT_SAS_DEV_STAT_RC_ABORT_TASK_SET_INTERNAL:
8213                        NDBG20(("IOC internally generated Abort Task Set"));
8214                        break;

8216                case MPI2_EVENT_SAS_DEV_STAT_RC_CLEAR_TASK_SET_INTERNAL:
8217                        NDBG20(("IOC internally generated Clear Task Set"));
8218                        break;
```

```
8220                case MPI2_EVENT_SAS_DEV_STAT_RC_QUERY_TASK_INTERNAL:
8221                        NDBG20(("IOC internally generated Query Task"));
8222                        break;

8224                case MPI2_EVENT_SAS_DEV_STAT_RC_ASYNC_NOTIFICATION:
8225                        NDBG20(("Device sent an Asynchronous Notification"));
8226                        break;

8228                default:
8229                        break;
8230                }
8231                break;
8232        }
8233        case MPI2_EVENT_IR_CONFIGURATION_CHANGE_LIST:
8234        {
8235                /*
8236                 * IR TOPOLOGY CHANGE LIST Event has already been handled
8237                 * in mpt_handle_event_sync() of interrupt context
8238                 */
8239                break;
8240        }
8241        case MPI2_EVENT_IR_OPERATION_STATUS:
8242        {
8243                Mpi2EventDataIrOperationStatus_t        *irOpStatus;
8244                char                                    reason_str[80];
8245                uint8_t                                 rc, percent;
8246                uint16_t                                handle;

8248                irOpStatus = (pMpi2EventDataIrOperationStatus_t)
8249                    eventreply->EventData;
8250                rc = ddi_get8(mpt->m_acc_reply_frame_hdl,
8251                    &irOpStatus->RAIDOperation);
8252                percent = ddi_get8(mpt->m_acc_reply_frame_hdl,
8253                    &irOpStatus->PercentComplete);
8254                handle = ddi_get16(mpt->m_acc_reply_frame_hdl,
8255                    &irOpStatus->VolDevHandle);

8257                switch (rc) {
8258                        case MPI2_EVENT_IR_RAIDOP_RESYNC:
8259                                (void) sprintf(reason_str, "resync");
8260                                break;
8261                        case MPI2_EVENT_IR_RAIDOP_ONLINE_CAP_EXPANSION:
8262                                (void) sprintf(reason_str, "online capacity "
8263                                    "expansion");
8264                                break;
8265                        case MPI2_EVENT_IR_RAIDOP_CONSISTENCY_CHECK:
8266                                (void) sprintf(reason_str, "consistency check");
8267                                break;
8268                        default:
8269                                (void) sprintf(reason_str, "unknown reason %x",
8270                                    rc);
8271                }

8273                NDBG20(("mptsas3%d raid operational status: (%s)"
5839                NDBG20(("mptsas%d raid operational status: (%s)"
8274                    "\thandle(0x%04x), percent complete(%d)\n",
8275                    mpt->m_instance, reason_str, handle, percent));
8276                break;
8277        }
8278        case MPI2_EVENT_SAS_BROADCAST_PRIMITIVE:
8279        {
8280                pMpi2EventDataSasBroadcastPrimitive_t   sas_broadcast;
8281                uint8_t                                 phy_num;
8282                uint8_t                                 primitive;

8284                sas_broadcast = (pMpi2EventDataSasBroadcastPrimitive_t)
```



```
8285                    eventreply->EventData;

8287                phy_num = ddi_get8(mpt->m_acc_reply_frame_hdl,
8288                    &sas_broadcast->PhyNum);
8289                primitive = ddi_get8(mpt->m_acc_reply_frame_hdl,
8290                    &sas_broadcast->Primitive);

8292                switch (primitive) {
8293                case MPI2_EVENT_PRIMITIVE_CHANGE:
8294                        mptsas_smhba_log_sysevent(mpt,
8295                            ESC_SAS_HBA_PORT_BROADCAST,
8296                            SAS_PORT_BROADCAST_CHANGE,
8297                            &mpt->m_phy_info[phy_num].smhba_info);
8298                        break;
8299                case MPI2_EVENT_PRIMITIVE_SES:
8300                        mptsas_smhba_log_sysevent(mpt,
8301                            ESC_SAS_HBA_PORT_BROADCAST,
8302                            SAS_PORT_BROADCAST_SES,
8303                            &mpt->m_phy_info[phy_num].smhba_info);
8304                        break;
8305                case MPI2_EVENT_PRIMITIVE_EXPANDER:
8306                        mptsas_smhba_log_sysevent(mpt,
8307                            ESC_SAS_HBA_PORT_BROADCAST,
8308                            SAS_PORT_BROADCAST_D01_4,
8309                            &mpt->m_phy_info[phy_num].smhba_info);
8310                        break;
8311                case MPI2_EVENT_PRIMITIVE_ASYNCHRONOUS_EVENT:
8312                        mptsas_smhba_log_sysevent(mpt,
8313                            ESC_SAS_HBA_PORT_BROADCAST,
8314                            SAS_PORT_BROADCAST_D04_7,
8315                            &mpt->m_phy_info[phy_num].smhba_info);
8316                        break;
8317                case MPI2_EVENT_PRIMITIVE_RESERVED3:
8318                        mptsas_smhba_log_sysevent(mpt,
8319                            ESC_SAS_HBA_PORT_BROADCAST,
8320                            SAS_PORT_BROADCAST_D16_7,
8321                            &mpt->m_phy_info[phy_num].smhba_info);
8322                        break;
8323                case MPI2_EVENT_PRIMITIVE_RESERVED4:
8324                        mptsas_smhba_log_sysevent(mpt,
8325                            ESC_SAS_HBA_PORT_BROADCAST,
8326                            SAS_PORT_BROADCAST_D29_7,
8327                            &mpt->m_phy_info[phy_num].smhba_info);
8328                        break;
8329                case MPI2_EVENT_PRIMITIVE_CHANGE0_RESERVED:
8330                        mptsas_smhba_log_sysevent(mpt,
8331                            ESC_SAS_HBA_PORT_BROADCAST,
8332                            SAS_PORT_BROADCAST_D24_0,
8333                            &mpt->m_phy_info[phy_num].smhba_info);
8334                        break;
8335                case MPI2_EVENT_PRIMITIVE_CHANGE1_RESERVED:
8336                        mptsas_smhba_log_sysevent(mpt,
8337                            ESC_SAS_HBA_PORT_BROADCAST,
8338                            SAS_PORT_BROADCAST_D27_4,
8339                            &mpt->m_phy_info[phy_num].smhba_info);
8340                        break;
8341                default:
8342                        NDBG16(("mptsas3%d: unknown BROADCAST PRIMITIVE"
5908                        NDBG20(("mptsas%d: unknown BROADCAST PRIMITIVE"
8343                            " %x received",
8344                            mpt->m_instance, primitive));
8345                        break;
8346                }
8347                NDBG16(("mptsas3%d sas broadcast primitive: "
5913                NDBG20(("mptsas%d sas broadcast primitive: "
8348                    "\tprimitive(0x%04x), phy(%d) complete\n",
```

```
8349                    mpt->m_instance, primitive, phy_num));
8350                break;
8351        }
8352        case MPI2_EVENT_IR_VOLUME:
8353        {
8354                Mpi2EventDataIrVolume_t         *irVolume;
8355                uint16_t                        devhandle;
8356                uint32_t                        state;
8357                int                             config, vol;
8358                uint8_t                         found = FALSE;

8360                irVolume = (pMpi2EventDataIrVolume_t)eventreply->EventData;
8361                state = ddi_get32(mpt->m_acc_reply_frame_hdl,
8362                    &irVolume->NewValue);
8363                devhandle = ddi_get16(mpt->m_acc_reply_frame_hdl,
8364                    &irVolume->VolDevHandle);

8366                NDBG20(("EVENT_IR_VOLUME event is received"));

8368                /*
8369                 * Get latest RAID info and then find the DevHandle for this
8370                 * event in the configuration.  If the DevHandle is not found
8371                 * just exit the event.
8372                 */
8373                (void) mptsas_get_raid_info(mpt);
8374                for (config = 0; (config < mpt->m_num_raid_configs) &&
8375                    (!found); config++) {
8376                        for (vol = 0; vol < MPTSAS_MAX_RAIDVOLS; vol++) {
8377                                if (mpt->m_raidconfig[config].m_raidvol[vol].
8378                                    m_raidhandle == devhandle) {
8379                                        found = TRUE;
8380                                        break;
8381                                }
8382                        }
8383                }
8384                if (!found) {
8385                        break;
8386                }

8388                switch (irVolume->ReasonCode) {
8389                case MPI2_EVENT_IR_VOLUME_RC_SETTINGS_CHANGED:
8390                {
8391                        uint32_t i;
8392                        mpt->m_raidconfig[config].m_raidvol[vol].m_settings =
8393                            state;

8395                        i = state & MPI2_RAIDVOL0_SETTING_MASK_WRITE_CACHING;
8396                        mptsas_log(mpt, CE_NOTE, " Volume %d settings changed"
8397                            ", auto-config of hot-swap drives is %s"
8398                            ", write caching is %s"
8399                            ", hot-spare pool mask is %02x\n",
8400                            vol, state &
8401                            MPI2_RAIDVOL0_SETTING_AUTO_CONFIG_HSWAP_DISABLE
8402                            ? "disabled" : "enabled",
8403                            i == MPI2_RAIDVOL0_SETTING_UNCHANGED
8404                            ? "controlled by member disks" :
8405                            i == MPI2_RAIDVOL0_SETTING_DISABLE_WRITE_CACHING
8406                            ? "disabled" :
8407                            i == MPI2_RAIDVOL0_SETTING_ENABLE_WRITE_CACHING
8408                            ? "enabled" :
8409                            "incorrectly set",
8410                            (state >> 16) & 0xff);
8411                                break;
8412                }
8413                case MPI2_EVENT_IR_VOLUME_RC_STATE_CHANGED:
8414                {
```



```
8415                        mpt->m_raidconfig[config].m_raidvol[vol].m_state =
8416                            (uint8_t)state;

8418                        mptsas_log(mpt, CE_NOTE,
8419                            "Volume %d is now %s\n", vol,
8420                            state == MPI2_RAID_VOL_STATE_OPTIMAL
8421                            ? "optimal" :
8422                            state == MPI2_RAID_VOL_STATE_DEGRADED
8423                            ? "degraded" :
8424                            state == MPI2_RAID_VOL_STATE_ONLINE
8425                            ? "online" :
8426                            state == MPI2_RAID_VOL_STATE_INITIALIZING
8427                            ? "initializing" :
8428                            state == MPI2_RAID_VOL_STATE_FAILED
8429                            ? "failed" :
8430                            state == MPI2_RAID_VOL_STATE_MISSING
8431                            ? "missing" :
8432                            "state unknown");
8433                        break;
8434                }
8435                case MPI2_EVENT_IR_VOLUME_RC_STATUS_FLAGS_CHANGED:
8436                {
8437                        mpt->m_raidconfig[config].m_raidvol[vol].
8438                            m_statusflags = state;

8440                        mptsas_log(mpt, CE_NOTE,
8441                            " Volume %d is now %s%s%s%s%s%s%s%s\n",
8442                            vol,
8443                            state & MPI2_RAIDVOL0_STATUS_FLAG_ENABLED
8444                            ? ", enabled" : ", disabled",
8445                            state & MPI2_RAIDVOL0_STATUS_FLAG_QUIESCED
8446                            ? ", quiesced" : "",
8447                            state & MPI2_RAIDVOL0_STATUS_FLAG_VOLUME_INACTIVE
8448                            ? ", inactive" : ", active",
8449                            state &
8450                            MPI2_RAIDVOL0_STATUS_FLAG_BAD_BLOCK_TABLE_FULL
8451                            ? ", bad block table is full" : "",
8452                            state &
8453                            MPI2_RAIDVOL0_STATUS_FLAG_RESYNC_IN_PROGRESS
8454                            ? ", resync in progress" : "",
8455                            state & MPI2_RAIDVOL0_STATUS_FLAG_BACKGROUND_INIT
8456                            ? ", background initialization in progress" : "",
8457                            state &
8458                            MPI2_RAIDVOL0_STATUS_FLAG_CAPACITY_EXPANSION
8459                            ? ", capacity expansion in progress" : "",
8460                            state &
8461                            MPI2_RAIDVOL0_STATUS_FLAG_CONSISTENCY_CHECK
8462                            ? ", consistency check in progress" : "",
8463                            state & MPI2_RAIDVOL0_STATUS_FLAG_DATA_SCRUB
8464                            ? ", data scrub in progress" : "");
8465                        break;
8466                }
8467                default:
8468                        break;
8469                }
8470                break;
8471        }
8472        case MPI2_EVENT_IR_PHYSICAL_DISK:
8473        {
8474                Mpi2EventDataIrPhysicalDisk_t   *irPhysDisk;
8475                uint16_t                        devhandle, enchandle, slot;
8476                uint32_t                        status, state;
8477                uint8_t                         physdisknum, reason;

8479                irPhysDisk = (Mpi2EventDataIrPhysicalDisk_t *)
8480                    eventreply->EventData;
```

```
8481                physdisknum = ddi_get8(mpt->m_acc_reply_frame_hdl,
8482                    &irPhysDisk->PhysDiskNum);
8483                devhandle = ddi_get16(mpt->m_acc_reply_frame_hdl,
8484                    &irPhysDisk->PhysDiskDevHandle);
8485                enchandle = ddi_get16(mpt->m_acc_reply_frame_hdl,
8486                    &irPhysDisk->EnclosureHandle);
8487                slot = ddi_get16(mpt->m_acc_reply_frame_hdl,
8488                    &irPhysDisk->Slot);
8489                state = ddi_get32(mpt->m_acc_reply_frame_hdl,
8490                    &irPhysDisk->NewValue);
8491                reason = ddi_get8(mpt->m_acc_reply_frame_hdl,
8492                    &irPhysDisk->ReasonCode);

8494                NDBG20(("EVENT_IR_PHYSICAL_DISK event is received"));

8496                switch (reason) {
8497                case MPI2_EVENT_IR_PHYSDISK_RC_SETTINGS_CHANGED:
8498                        mptsas_log(mpt, CE_NOTE,
8499                            " PhysDiskNum %d with DevHandle 0x%x in slot %d "
8500                            "for enclosure with handle 0x%x is now in hot "
8501                            "spare pool %d",
8502                            physdisknum, devhandle, slot, enchandle,
8503                            (state >> 16) & 0xff);
8504                        break;

8506                case MPI2_EVENT_IR_PHYSDISK_RC_STATUS_FLAGS_CHANGED:
8507                        status = state;
8508                        mptsas_log(mpt, CE_NOTE,
8509                            " PhysDiskNum %d with DevHandle 0x%x in slot %d "
8510                            "for enclosure with handle 0x%x is now "
8511                            "%s%s%s%s%s\n", physdisknum, devhandle, slot,
8512                            enchandle,
8513                            status & MPI2_PHYSDISK0_STATUS_FLAG_INACTIVE_VOLUME
8514                            ? ", inactive" : ", active",
8515                            status & MPI2_PHYSDISK0_STATUS_FLAG_OUT_OF_SYNC
8516                            ? ", out of sync" : "",
8517                            status & MPI2_PHYSDISK0_STATUS_FLAG_QUIESCED
8518                            ? ", quiesced" : "",
8519                            status &
8520                            MPI2_PHYSDISK0_STATUS_FLAG_WRITE_CACHE_ENABLED
8521                            ? ", write cache enabled" : "",
8522                            status & MPI2_PHYSDISK0_STATUS_FLAG_OCE_TARGET
8523                            ? ", capacity expansion target" : "");
8524                        break;

8526                case MPI2_EVENT_IR_PHYSDISK_RC_STATE_CHANGED:
8527                        mptsas_log(mpt, CE_NOTE,
8528                            " PhysDiskNum %d with DevHandle 0x%x in slot %d "
8529                            "for enclosure with handle 0x%x is now %s\n",
8530                            physdisknum, devhandle, slot, enchandle,
8531                            state == MPI2_RAID_PD_STATE_OPTIMAL
8532                            ? "optimal" :
8533                            state == MPI2_RAID_PD_STATE_REBUILDING
8534                            ? "rebuilding" :
8535                            state == MPI2_RAID_PD_STATE_DEGRADED
8536                            ? "degraded" :
8537                            state == MPI2_RAID_PD_STATE_HOT_SPARE
8538                            ? "a hot spare" :
8539                            state == MPI2_RAID_PD_STATE_ONLINE
8540                            ? "online" :
8541                            state == MPI2_RAID_PD_STATE_OFFLINE
8542                            ? "offline" :
8543                            state == MPI2_RAID_PD_STATE_NOT_COMPATIBLE
8544                            ? "not compatible" :
8545                            state == MPI2_RAID_PD_STATE_NOT_CONFIGURED
8546                            ? "not configured" :
```



```
8547                                "state unknown");
8548                        break;
8549                }
8550                break;
8551        }
8552        default:
8553                NDBG20(("mptsas3%d: unknown event %x received",
6119                NDBG20(("mptsas%d: unknown event %x received",
8554                    mpt->m_instance, event));
8555                break;
8556        }

8558        /*
8559         * Return the reply frame to the free queue.
8560         */
8561        ddi_put32(mpt->m_acc_free_queue_hdl,
8562            &((uint32_t *)(void *)mpt->m_free_queue)[mpt->m_free_index], rfm);
8563        (void) ddi_dma_sync(mpt->m_dma_free_queue_hdl, 0, 0,
8564            DDI_DMA_SYNC_FORDEV);
8565        if (++mpt->m_free_index == mpt->m_free_queue_depth) {
8566                mpt->m_free_index = 0;
8567        }
8568        ddi_put32(mpt->m_datap, &mpt->m_reg->ReplyFreeHostIndex,
8569            mpt->m_free_index);
8570        mutex_exit(&mpt->m_mutex);
8571 }

8573 /*
8574  * invoked from timeout() to restart qfull cmds with throttle == 0
8575  */
8576 static void
8577 mptsas_restart_cmd(void *arg)
8578 {
8579        mptsas_t        *mpt = arg;
8580        mptsas_target_t *ptgt = NULL;

8582        mutex_enter(&mpt->m_mutex);

8584        mpt->m_restart_cmd_timeid = 0;

8586        for (ptgt = refhash_first(mpt->m_targets); ptgt != NULL;
8587            ptgt = refhash_next(mpt->m_targets, ptgt)) {
8588                mutex_enter(&ptgt->m_t_mutex);
8589 #endif /* ! codereview */
8590                if (ptgt->m_reset_delay == 0) {
8591                        if (ptgt->m_t_throttle == QFULL_THROTTLE) {
8592                                mptsas_set_throttle(mpt, ptgt,
8593                                    MAX_THROTTLE);
8594                        }
8595                }
8596                mutex_exit(&ptgt->m_t_mutex);
8597 #endif /* ! codereview */
8598        }
8599        mptsas_restart_hba(mpt);
8600        mutex_exit(&mpt->m_mutex);
8601 }

8603 /*
8604  * Assume some checks have been done prior to calling this
8605  * function so we don't need to consider taking the m_mutex.
8606  */
8607 static void
8608 mptsas_remove_cmd_nomtx(mptsas_t *mpt, mptsas_cmd_t *cmd)
8609 {
8610        int             slot;
8611        mptsas_slots_t  *slots = mpt->m_active;
```

```
8612         mptsas_target_t *ptgt = cmd->cmd_tgt_addr;

8614         ASSERT(cmd != NULL);
8615         ASSERT(cmd->cmd_queued == FALSE);
8616         ASSERT((cmd->cmd_flags & CFLAG_CMDIOC) == 0);

8618         slot = cmd->cmd_slot;

8620         /*
8621          * remove the cmd.
8622          */
8623         if (cmd == slots->m_slot[slot]) {
8624                 NDBG31(("mptsas_remove_cmd_nomtx: removing cmd=0x%p, flags "
8625                     "0x%x", (void *)cmd, cmd->cmd_flags));
8626                 slots->m_slot[slot] = NULL;
8627                 ASSERT(mpt->m_ncmds != 0);
8628                 atomic_dec_32(&mpt->m_ncmds);
8629                 ASSERT(mpt->m_rep_post_queues[cmd->cmd_rpqidx].rpq_ncmds != 0);
8630                 atomic_dec_32(
8631                     &mpt->m_rep_post_queues[cmd->cmd_rpqidx].rpq_ncmds);

8633                 /*
8634                  * Decrement per target ncmds, we know this is not an
8635                  * IOC cmd and it therefore has a target associated with it.
8636                  */
8637                 mutex_enter(&ptgt->m_t_mutex);
8638                 ASSERT(ptgt->m_t_ncmds != 0);
8639                 ptgt->m_t_ncmds--;

8641                 /*
8642                  * reset throttle if we just ran an untagged command
8643                  * to a tagged target
8644                  */
8645                 if ((ptgt->m_t_ncmds == 0) &&
8646                     ((cmd->cmd_pkt_flags & FLAG_TAGMASK) == 0)) {
8647                         mptsas_set_throttle(mpt, ptgt, MAX_THROTTLE);
8648                 }

8650                 /*
8651                  * Remove this command from the active queue.
8652                  */
8653                 if (cmd->cmd_active_expiration != 0) {
8654                         TAILQ_REMOVE(&ptgt->m_active_cmdq, cmd,
8655                             cmd_active_link);
8656                         cmd->cmd_active_expiration = 0;
8657                 }
8658                 mutex_exit(&ptgt->m_t_mutex);
8659         }

8661         ASSERT(cmd != slots->m_slot[cmd->cmd_slot]);
8662 }

8664 #endif /* ! codereview */
8665 void
8666 mptsas_remove_cmd(mptsas_t *mpt, mptsas_cmd_t *cmd)
8667 {
8668         int             slot;
8669         mptsas_slots_t  *slots = mpt->m_active;
6154         int             t;
8670         mptsas_target_t *ptgt = cmd->cmd_tgt_addr;

8672         ASSERT(cmd != NULL);
8673         ASSERT(cmd->cmd_queued == FALSE);

8675         /*
8676          * Task Management cmds are removed in their own routines.  Also,
```



```
8677          * we don't want to modify timeout based on TM cmds.
8678          */
8679         if (cmd->cmd_flags & CFLAG_TM_CMD) {
8680                 return;
8681         }

6168         t = Tgt(cmd);
8683         slot = cmd->cmd_slot;

8685         /*
8686          * remove the cmd.
8687          */
8688         if (cmd == slots->m_slot[slot]) {
8689                 NDBG31(("mptsas_remove_cmd: removing cmd=0x%p, flags 0x%x",
8690                     (void *)cmd, cmd->cmd_flags));
6175                 NDBG31(("mptsas_remove_cmd: removing cmd=0x%p", (void *)cmd));
8691                 slots->m_slot[slot] = NULL;
8692                 ASSERT(mpt->m_ncmds != 0);
8693                 atomic_dec_32(&mpt->m_ncmds);
8694                 ASSERT(mpt->m_rep_post_queues[cmd->cmd_rpqidx].rpq_ncmds != 0);
8695                 atomic_dec_32(
8696                     &mpt->m_rep_post_queues[cmd->cmd_rpqidx].rpq_ncmds);
6177                 mpt->m_ncmds--;

8698                 /*
8699                  * only decrement per target ncmds if command
8700                  * has a target associated with it.
8701                  */
8702                 if ((cmd->cmd_flags & CFLAG_CMDIOC) == 0) {
8703                         mutex_enter(&ptgt->m_t_mutex);
8704                         ASSERT(ptgt->m_t_ncmds != 0);
8705 #endif /* ! codereview */
8706                         ptgt->m_t_ncmds--;

8708 #endif /* ! codereview */
8709                         /*
8710                          * reset throttle if we just ran an untagged command
8711                          * to a tagged target
8712                          */
8713                         if ((ptgt->m_t_ncmds == 0) &&
8714                             ((cmd->cmd_pkt_flags & FLAG_TAGMASK) == 0)) {
8715                                 mptsas_set_throttle(mpt, ptgt, MAX_THROTTLE);
8716                         }

8718                         /*
8719                          * Remove this command from the active queue.
8720                          */
8721                         if (cmd->cmd_active_expiration != 0) {
8722                                 TAILQ_REMOVE(&ptgt->m_active_cmdq, cmd,
8723                                     cmd_active_link);
8724                                 cmd->cmd_active_expiration = 0;
8725                         }
8726                         mutex_exit(&ptgt->m_t_mutex);
8727 #endif /* ! codereview */
8728                 }

8730         }

8732         /*
8733          * This is all we need to do for ioc commands.
8734          */
8735         if (cmd->cmd_flags & CFLAG_CMDIOC) {
8736                 mptsas_return_to_pool(mpt, cmd);
8737                 return;
8738         }
```

```
6184         /*
6185          * Figure out what to set tag Q timeout for...
6186          *
6187          * Optimize: If we have duplicate's of same timeout
6188          * we're using, then we'll use it again until we run
6189          * out of duplicates.  This should be the normal case
6190          * for block and raw I/O.
6191          * If no duplicates, we have to scan through tag que and
6192          * find the longest timeout value and use it.  This is
6193          * going to take a while...
6194          * Add 1 to m_n_normal to account for TM request.
6195          */
6196         if (cmd->cmd_pkt->pkt_time == ptgt->m_timebase) {
6197                 if (--(ptgt->m_dups) == 0) {
6198                         if (ptgt->m_t_ncmds) {
6199                                 mptsas_cmd_t *ssp;
6200                                 uint_t n = 0;
6201                                 ushort_t nslots = (slots->m_n_normal + 1);
6202                                 ushort_t i;
6203                                 /*
6204                                  * This crude check assumes we don't do
6205                                  * this too often which seems reasonable
6206                                  * for block and raw I/O.
6207                                  */
6208                                 for (i = 0; i < nslots; i++) {
6209                                         ssp = slots->m_slot[i];
6210                                         if (ssp && (Tgt(ssp) == t) &&
6211                                             (ssp->cmd_pkt->pkt_time > n)) {
6212                                                 n = ssp->cmd_pkt->pkt_time;
6213                                                 ptgt->m_dups = 1;
6214                                         } else if (ssp && (Tgt(ssp) == t) &&
6215                                             (ssp->cmd_pkt->pkt_time == n)) {
6216                                                 ptgt->m_dups++;
6217                                         }
6218                                 }
6219                                 ptgt->m_timebase = n;
6220                         } else {
6221                                 ptgt->m_dups = 0;
6222                                 ptgt->m_timebase = 0;
6223                         }
6224                 }
6225         }
6226         ptgt->m_timeout = ptgt->m_timebase;

8740         ASSERT(cmd != slots->m_slot[cmd->cmd_slot]);
8741 }

8743 /*
8744  * accept all cmds on the tx_waitq if any and then
8745  * start a fresh request from the top of the device queue.
8746  *
8747  * since there are always cmds queued on the tx_waitq, and rare cmds on
8748  * the instance waitq, so this function should not be invoked in the ISR,
8749  * the mptsas_restart_waitq() is invoked in the ISR instead. otherwise, the
8750  * burden belongs to the IO dispatch CPUs is moved the interrupt CPU.
8751  */
8752 static void
8753 mptsas_restart_hba(mptsas_t *mpt)
8754 {
8755         ASSERT(mutex_owned(&mpt->m_mutex));

8757         mptsas_accept_tx_waitqs(mpt);
6245         mutex_enter(&mpt->m_tx_waitq_mutex);
6246         if (mpt->m_tx_waitq) {
6247                 mptsas_accept_tx_waitq(mpt);
6248         }
```



```
6249         mutex_exit(&mpt->m_tx_waitq_mutex);
8758         mptsas_restart_waitq(mpt);
8759 }

8761 /*
8762  * start a fresh request from the top of the device queue
8763  */
8764 static void
8765 mptsas_restart_waitq(mptsas_t *mpt)
8766 {
8767         mptsas_cmd_t    *cmd, *next_cmd;
8768         mptsas_target_t *ptgt = NULL;

8770         NDBG1(("mptsas_restart_waitq: mpt=0x%p", (void *)mpt));

8772         ASSERT(mutex_owned(&mpt->m_mutex));

8774         /*
8775          * If there is a reset delay, don't start any cmds.  Otherwise, start
8776          * as many cmds as possible.
8777          * Since SMID 0 is reserved and the TM slot is reserved, the actual max
8778          * commands is m_max_requests - 2.
8779          */
8780         cmd = mpt->m_waitq;

8782         while (cmd != NULL) {
8783                 next_cmd = cmd->cmd_linkp;
8784                 if (cmd->cmd_flags & CFLAG_PASSTHRU) {
8785                         if (mptsas_save_cmd(mpt, cmd) == TRUE) {
8786                                 /*
8787                                  * passthru command get slot need
8788                                  * set CFLAG_PREPARED.
8789                                  */
8790                                 cmd->cmd_flags |= CFLAG_PREPARED;
8791                                 mptsas_waitq_delete(mpt, cmd);
8792                                 mptsas_start_passthru(mpt, cmd);
8793                         }
8794                         cmd = next_cmd;
8795                         continue;
8796                 }
8797                 if (cmd->cmd_flags & CFLAG_CONFIG) {
8798                         if (mptsas_save_cmd(mpt, cmd) == TRUE) {
8799                                 /*
8800                                  * Send the config page request and delete it
8801                                  * from the waitq.
8802                                  */
8803                                 cmd->cmd_flags |= CFLAG_PREPARED;
8804                                 mptsas_waitq_delete(mpt, cmd);
8805                                 mptsas_start_config_page_access(mpt, cmd);
8806                         }
8807                         cmd = next_cmd;
8808                         continue;
8809                 }
8810                 if (cmd->cmd_flags & CFLAG_FW_DIAG) {
8811                         if (mptsas_save_cmd(mpt, cmd) == TRUE) {
8812                                 /*
8813                                  * Send the FW Diag request and delete if from
8814                                  * the waitq.
8815                                  */
8816                                 cmd->cmd_flags |= CFLAG_PREPARED;
8817                                 mptsas_waitq_delete(mpt, cmd);
8818                                 mptsas_start_diag(mpt, cmd);
8819                         }
8820                         cmd = next_cmd;
8821                         continue;
8822                 }
```

```
8824                ptgt = cmd->cmd_tgt_addr;
8825                if (ptgt) {
8826                        mutex_enter(&ptgt->m_t_mutex);
8827                        if ((ptgt->m_t_throttle == DRAIN_THROTTLE) &&
6317                if (ptgt && (ptgt->m_t_throttle == DRAIN_THROTTLE) &&
8828                            (ptgt->m_t_ncmds == 0)) {
8829                                mptsas_set_throttle(mpt, ptgt, MAX_THROTTLE);
8830                        }
8831                        if ((mpt->m_ncmds <= (mpt->m_max_requests - 2)) &&
8832                            (ptgt->m_reset_delay == 0) &&
8833                            (ptgt->m_t_ncmds < ptgt->m_t_throttle)) {
8834                                mutex_exit(&ptgt->m_t_mutex);

6322                    (ptgt && (ptgt->m_reset_delay == 0)) &&
6323                    (ptgt && (ptgt->m_t_ncmds <
6324                    ptgt->m_t_throttle))) {
8836                                if (mptsas_save_cmd(mpt, cmd) == TRUE) {
8837                                        mptsas_waitq_delete(mpt, cmd);
8838                                        mutex_exit(&mpt->m_mutex);
8839 #endif /* ! codereview */
8840                                        (void) mptsas_start_cmd(mpt, cmd);
8841                                        mutex_enter(&mpt->m_mutex);
8842                                        cmd = mpt->m_waitq;
8843                                        continue;
8844                                }
8845                        } else {
8846                                mutex_exit(&ptgt->m_t_mutex);
8847 #endif /* ! codereview */
8848                        }
8849                }
8850                cmd = next_cmd;
8851        }
8852 }

8854 /*
8855  * Cmds are queued if scsi_start() doesn't get the m_mutex lock(no wait)
8856  * or if the decision has been made to always do that. Setting
8857  * mptsas_allow_txq_jumping to zero will allow higher performance on
8858  * a heavily loaded system as there is less disruption to the flow here.
8859  * There are 2 threads that handle one queue each. The idea is that
8860  * they take it in turn to grab the m_mutex to run the mptsas_accept_pkt()
8861  * function and then drop it while the cmd is started in mptsas_start_cmd().
8862  */
8863 static void
8864 mptsas_tx_waitq_thread(mptsas_thread_arg_t *arg)
8865 {
8866         mptsas_t *mpt = arg->mpt;
8867         mptsas_tx_waitqueue_t *txwq = &mpt->m_tx_waitq[arg->t];

8869         mutex_enter(&txwq->txwq_mutex);
8870         while (txwq->txwq_active) {
8871                 mptsas_drain_tx_waitq(mpt, txwq);
8872                 if (txwq->txwq_wdrain) {
8873                         cv_signal(&txwq->txwq_drain_cv);
8874                 }
8875                 cv_wait(&txwq->txwq_cv, &txwq->txwq_mutex);
8876         }
8877         mutex_exit(&txwq->txwq_mutex);
8878         mutex_enter(&mpt->m_qthread_mutex);
8879         mpt->m_txwq_thread_n--;
8880         cv_broadcast(&mpt->m_qthread_cv);
8881         mutex_exit(&mpt->m_qthread_mutex);
8882 }

8884 /*
```



```
8885  * Set the draining flag, disconnect the list and process one at a time
8886  * so that the cmds are sent in order.
8887  */
8888 static void
8889 mptsas_drain_tx_waitq(mptsas_t *mpt, mptsas_tx_waitqueue_t *txwq)
8890 {
8891         mptsas_cmd_t    *cmd, *ncmd;
8892         int             rval, start;
8893 #ifdef MPTSAS_DEBUG
8894         uint32_t        qlen;
8895 #endif

8897         txwq->txwq_draining = TRUE;
8898 #ifndef __lock_lint
8899         _NOTE(CONSTCOND)
8900 #endif
8901         while (TRUE) {

8903                 /*
8904                  * A Bus Reset could occur at any time but it will have to
8905                  * wait for the main mutex before flushing the tx_waitq.
8906                  * Pull all commands at once, then follow the list in order to
8907                  * reduce txwq_mutex hold time. If there is a Bus Reset at
8908                  * some point the commands will get to the waitq and then be
8909                  * flushed.
8910                  */
8911                 cmd = txwq->txwq_cmdq;

8913                 if (cmd == NULL) {
8914                         txwq->txwq_draining = FALSE;
8915                         return;
8916                 }
8917                 txwq->txwq_cmdq = NULL;
8918                 txwq->txwq_qtail = &txwq->txwq_cmdq;
8919 #ifdef MPTSAS_DEBUG
8920                 qlen = txwq->txwq_len;
8921 #endif
8922                 txwq->txwq_len = 0;
8923                 mutex_exit(&txwq->txwq_mutex);

8925                 while (cmd) {
8926                         ncmd = cmd->cmd_linkp;
8927                         cmd->cmd_linkp = NULL;
8928                         mutex_enter(&mpt->m_mutex);
8929                         start = mptsas_accept_pkt(mpt, cmd, &rval);
8930                         mutex_exit(&mpt->m_mutex);
8931                         if (start) {
8932                                 (void) mptsas_start_cmd(mpt, cmd);
8933                         }
8934                         if (rval != TRAN_ACCEPT)
8935                                 cmn_err(CE_WARN,
8936                                     "mpt: mptsas_drain_tx_waitq: failed "
8937                                     "(rval=0x%x) to accept cmd 0x%p on queue\n",
8938                                     rval, (void *)cmd);
8939                         cmd = ncmd;
8940 #ifdef MPTSAS_DEBUG
8941                         qlen--;
8942 #endif
8943                 }
8944                 ASSERT(qlen == 0);
8945                 mutex_enter(&txwq->txwq_mutex);
8946         }
8947 }

8949 #endif /* ! codereview */
8950 /*
```

```
 * Stop the drain threads from picking up a new list.
 * Optionally wait for the current list being processed to drain through.
 * Add to and processing the tx waitq is now on hold until unblock is called.
 * Cmds are queued if tran_start() doesn't get the m_mutexlock(no wait).
 * Accept all those queued cmds before new cmd is accept so that the
 * cmds are sent in order.
 */
static void
mptsas_block_tx_waitqs(mptsas_t *mpt, int wait)
mptsas_accept_tx_waitq(mptsas_t *mpt)
{
        int             i;
        uint8_t         wdrain = 0;
        mptsas_tx_waitqueue_t *txwq;
        mptsas_cmd_t *cmd;

        ASSERT(mutex_owned(&mpt->m_mutex));
        ASSERT(mutex_owned(&mpt->m_tx_waitq_mutex));

        if (mpt->m_txwq_thread_n == 0) {
                return;
        }

        /*
         * Turn off the use of the tx wait queues by scsi_start().
         * This is just a dynamic flag no need for a mutex.
         */
        mpt->m_txwq_enabled = BLOCKED;

        for (i = 0; i < NUM_TX_WAITQ; i++) {
                txwq = &mpt->m_tx_waitq[i];
                mutex_enter(&txwq->txwq_mutex);
                txwq->txwq_wdrain = TRUE;
                if (txwq->txwq_draining && wait)
                        wdrain |= (1<<i);
                mutex_exit(&txwq->txwq_mutex);
        }

        if (wdrain) {
#endif /* ! codereview */
                /*
                 * Because the threads disconnect the entire queue each time
                 * round in order to drain to completely drain we have to
                 * drop the main mutex otherwise the drain threads get stuck.
         * A Bus Reset could occur at any time and flush the tx_waitq,
         * so we cannot count on the tx_waitq to contain even one cmd.
         * And when the m_tx_waitq_mutex is released and run
         * mptsas_accept_pkt(), the tx_waitq may be flushed.
                 */
                mutex_exit(&mpt->m_mutex);
                for (i = 0; i < NUM_TX_WAITQ; i++) {
                        if (wdrain & (1<<i)) {
                                txwq = &mpt->m_tx_waitq[i];
                                mutex_enter(&txwq->txwq_mutex);
                                while (txwq->txwq_draining) {
                                        cv_wait(&txwq->txwq_drain_cv,
                                            &txwq->txwq_mutex);
                                }
                                mutex_exit(&txwq->txwq_mutex);
                        }
                }
                mutex_enter(&mpt->m_mutex);
        cmd = mpt->m_tx_waitq;
        for (;;) {
                if ((cmd = mpt->m_tx_waitq) == NULL) {
                        mpt->m_tx_draining = 0;
```



```
                        break;
        }
}

static void
mptsas_unblock_tx_waitqs(mptsas_t *mpt)
{
        int                     i;
        mptsas_tx_waitqueue_t   *txwq;

        if (mpt->m_txwq_thread_n == 0) {
                return;
                if ((mpt->m_tx_waitq = cmd->cmd_linkp) == NULL) {
                        mpt->m_tx_waitqtail = &mpt->m_tx_waitq;
        }

        for (i = 0; i < NUM_TX_WAITQ; i++) {
                txwq = &mpt->m_tx_waitq[i];
                mutex_enter(&txwq->txwq_mutex);
                txwq->txwq_wdrain = FALSE;
                cv_signal(&txwq->txwq_cv);
                mutex_exit(&txwq->txwq_mutex);
                cmd->cmd_linkp = NULL;
                mutex_exit(&mpt->m_tx_waitq_mutex);
                if (mptsas_accept_pkt(mpt, cmd) != TRAN_ACCEPT)
                        cmn_err(CE_WARN, "mpt: mptsas_accept_tx_waitq: failed "
                            "to accept cmd on queue\n");
                mutex_enter(&mpt->m_tx_waitq_mutex);
        }

        mpt->m_txwq_enabled = FALSE;
#endif /* ! codereview */
}

static void
mptsas_accept_tx_waitqs(mptsas_t *mpt)
{
        /*
         * Block with drain and unblock will leave us in a state where
         * we have the main mutex, there is nothing on the tx wait queues
         * and they are not in use until watch notices high activity again.
         */
        mptsas_block_tx_waitqs(mpt, 1);
        mptsas_unblock_tx_waitqs(mpt);
}
#endif /* ! codereview */

/*
 * mpt tag type lookup
 */
static char mptsas_tag_lookup[] =
        {0, MSG_HEAD_QTAG, MSG_ORDERED_QTAG, 0, MSG_SIMPLE_QTAG};

static int
mptsas_start_cmd(mptsas_t *mpt, mptsas_cmd_t *cmd)
{
        struct scsi_pkt         *pkt = CMD2PKT(cmd);
        uint32_t                control = 0;
        caddr_t                 mem, arsbuf;
        int                     n;
        caddr_t                 mem;
        pMpi2SCSIIORequest_t    io_request;
        ddi_dma_handle_t        dma_hdl = mpt->m_dma_req_frame_hdl;
        ddi_acc_handle_t        acc_hdl = mpt->m_acc_req_frame_hdl;
        mptsas_target_t         *ptgt = cmd->cmd_tgt_addr;
        uint16_t                SMID, io_flags = 0, ars_size;
```

```
9058         uint8_t                 MSIidx;
9059         uint64_t                request_desc;
9060         uint32_t                ars_dmaaddrlow;
9061         mptsas_cmd_t            *c;
6366         uint16_t                SMID, io_flags = 0;
6367         uint32_t                request_desc_low, request_desc_high;

9063         NDBG1(("mptsas_start_cmd: cmd=0x%p, flags 0x%x", (void *)cmd,
9064             cmd->cmd_flags));
6369         NDBG1(("mptsas_start_cmd: cmd=0x%p", (void *)cmd));

9066         /*
9067          * Set SMID and increment index.  Rollover to 1 instead of 0 if index
9068          * is at the max.  0 is an invalid SMID, so we call the first index 1.
9069          */
9070         SMID = cmd->cmd_slot;
9071         MSIidx = cmd->cmd_rpqidx;
9072 #endif /* ! codereview */

9074         /*
9075          * It is possible for back to back device reset to
9076          * happen before the reset delay has expired.  That's
9077          * ok, just let the device reset go out on the bus.
9078          */
9079         if ((cmd->cmd_pkt_flags & FLAG_NOINTR) == 0) {
9080                 ASSERT(ptgt->m_reset_delay == 0);
9081         }

9083         /*
9084          * if a non-tagged cmd is submitted to an active tagged target
9085          * then drain before submitting this cmd; SCSI-2 allows RQSENSE
9086          * to be untagged
9087          */
9088         mutex_enter(&ptgt->m_t_mutex);
9089 #endif /* ! codereview */
9090         if (((cmd->cmd_pkt_flags & FLAG_TAGMASK) == 0) &&
9091             (ptgt->m_t_ncmds > 1) &&
9092             ((cmd->cmd_flags & CFLAG_TM_CMD) == 0) &&
9093             (*(cmd->cmd_pkt->pkt_cdbp) != SCMD_REQUEST_SENSE)) {
9094                 if ((cmd->cmd_pkt_flags & FLAG_NOINTR) == 0) {
9095                         NDBG23(("target=%d, untagged cmd, start draining\n",
9096                             ptgt->m_devhdl));

9098                         if (ptgt->m_reset_delay == 0) {
9099                                 mptsas_set_throttle(mpt, ptgt, DRAIN_THROTTLE);
9100                         }
9101                         mutex_exit(&ptgt->m_t_mutex);
9102 #endif /* ! codereview */

9104                         mutex_enter(&mpt->m_mutex);
9105 #endif /* ! codereview */
9106                         mptsas_remove_cmd(mpt, cmd);
9107                         cmd->cmd_pkt_flags |= FLAG_HEAD;
9108                         mptsas_waitq_add(mpt, cmd);
9109                         mutex_exit(&mpt->m_mutex);
9110                 } else {
9111                         mutex_exit(&ptgt->m_t_mutex);
9112 #endif /* ! codereview */
9113                 }
9114                 return (DDI_FAILURE);
9115         }

9117         /*
9118          * Set correct tag bits.
9119          */
9120         if (cmd->cmd_pkt_flags & FLAG_TAGMASK) {
```



```
9121                 switch (mptsas_tag_lookup[((cmd->cmd_pkt_flags &
9122                     FLAG_TAGMASK) >> 12)]) {
9123                 case MSG_SIMPLE_QTAG:
9124                         control |= MPI2_SCSIIO_CONTROL_SIMPLEQ;
9125                         break;
9126                 case MSG_HEAD_QTAG:
9127                         control |= MPI2_SCSIIO_CONTROL_HEADOFQ;
9128                         break;
9129                 case MSG_ORDERED_QTAG:
9130                         control |= MPI2_SCSIIO_CONTROL_ORDEREDQ;
9131                         break;
9132                 default:
9133                         mptsas_log(mpt, CE_WARN, "mpt: Invalid tag type\n");
9134                         break;
9135                 }
9136         } else {
9137                 if (*(cmd->cmd_pkt->pkt_cdbp) != SCMD_REQUEST_SENSE) {
9138                         ptgt->m_t_throttle = 1;
9139                 }
9140                 control |= MPI2_SCSIIO_CONTROL_SIMPLEQ;
9141         }

9143         /*
9144          * Set timeout.
9145          */
9146         cmd->cmd_active_expiration =
9147             gethrtime() + (hrtime_t)pkt->pkt_time * NANOSEC;

9149         c = TAILQ_FIRST(&ptgt->m_active_cmdq);
9150         if (c == NULL ||
9151             c->cmd_active_expiration < cmd->cmd_active_expiration) {
9152                 /*
9153                  * Common case is that this is the last pending expiration
9154                  * (or queue is empty). Insert at head of the queue.
9155                  */
9156                 TAILQ_INSERT_HEAD(&ptgt->m_active_cmdq, cmd, cmd_active_link);
9157         } else {
9158                 /*
9159                  * Queue is not empty and first element expires later than
9160                  * this command. Search for element expiring sooner.
9161                  */
9162                 while ((c = TAILQ_NEXT(c, cmd_active_link)) != NULL) {
9163                         if (c->cmd_active_expiration <
9164                             cmd->cmd_active_expiration) {
9165                                 TAILQ_INSERT_BEFORE(c, cmd, cmd_active_link);
9166                                 break;
9167                         }
9168                 }
9169                 if (c == NULL) {
9170                         /*
9171                          * No element found expiring sooner, append to
9172                          * non-empty queue.
9173                          */
9174                         TAILQ_INSERT_TAIL(&ptgt->m_active_cmdq, cmd,
9175                             cmd_active_link);
9176                 }
9177         }

9179         mutex_exit(&ptgt->m_t_mutex);

9181 #endif /* ! codereview */
9182         if (cmd->cmd_pkt_flags & FLAG_TLR) {
9183                 control |= MPI2_SCSIIO_CONTROL_TLR_ON;
9184         }

9186         mem = mpt->m_req_frame + (mpt->m_req_frame_size * SMID);
```

```
9187         io_request = (pMpi2SCSIIORequest_t)mem;
9188         if (cmd->cmd_extrqslen != 0) {
9189                 /*
9190                  * Mapping of the buffer was done in mptsas_pkt_alloc_extern().
9191                  * Calculate the DMA address with the same offset.
9192                  */
9193                 arsbuf = cmd->cmd_arq_buf;
9194                 ars_size = cmd->cmd_extrqslen;
9195                 ars_dmaaddrlow = (mpt->m_req_sense_dma_addr +
9196                     ((uintptr_t)arsbuf - (uintptr_t)mpt->m_req_sense)) &
9197                     0xffffffffull;
9198         } else {
9199                 arsbuf = mpt->m_req_sense + (mpt->m_req_sense_size * (SMID-1));
9200                 cmd->cmd_arq_buf = arsbuf;
9201                 ars_size = mpt->m_req_sense_size;
9202                 ars_dmaaddrlow = (mpt->m_req_sense_dma_addr +
9203                     (mpt->m_req_sense_size * (SMID-1))) &
9204                     0xffffffffull;
9205         }
9206         bzero(io_request, sizeof (Mpi2SCSIIORequest_t));
9207         bzero(arsbuf, ars_size);
9208 #endif /* ! codereview */

6376         bzero(io_request, sizeof (Mpi2SCSIIORequest_t));
9210         ddi_put8(acc_hdl, &io_request->SGLOffset0, offsetof
9211             (MPI2_SCSI_IO_REQUEST, SGL) / 4);
9212         mptsas_init_std_hdr(acc_hdl, io_request, ptgt->m_devhdl, Lun(cmd), 0,
9213             MPI2_FUNCTION_SCSI_IO_REQUEST);

9215         (void) ddi_rep_put8(acc_hdl, (uint8_t *)pkt->pkt_cdbp,
9216             io_request->CDB.CDB32, cmd->cmd_cdblen, DDI_DEV_AUTOINCR);

9218         io_flags = cmd->cmd_cdblen;
9219         if (mptsas3_use_fastpath &&
9220             ptgt->m_io_flags & MPI25_SAS_DEVICE0_FLAGS_ENABLED_FAST_PATH) {
9221                 io_flags |= MPI25_SCSIIO_IOFLAGS_FAST_PATH;
9222                 request_desc = MPI25_REQ_DESCRIPT_FLAGS_FAST_PATH_SCSI_IO;
9223         } else {
9224                 request_desc = MPI2_REQ_DESCRIPT_FLAGS_SCSI_IO;
9225         }
9226 #endif /* ! codereview */
9227         ddi_put16(acc_hdl, &io_request->IoFlags, io_flags);
9228         /*
9229          * setup the Scatter/Gather DMA list for this request
9230          */
9231         if (cmd->cmd_cookiec > 0) {
9232                 mptsas_sge_setup(mpt, cmd, &control, io_request, acc_hdl);
9233         } else {
9234                 ddi_put32(acc_hdl, &io_request->SGL.MpiSimple.FlagsLength,
9235                     ((uint32_t)MPI2_SGE_FLAGS_LAST_ELEMENT |
9236                     MPI2_SGE_FLAGS_END_OF_BUFFER |
9237                     MPI2_SGE_FLAGS_SIMPLE_ELEMENT |
9238                     MPI2_SGE_FLAGS_END_OF_LIST) << MPI2_SGE_FLAGS_SHIFT);
9239         }

9241         /*
9242          * save ARQ information
9243          */
9244         ddi_put8(acc_hdl, &io_request->SenseBufferLength, cmd->cmd_rqslen);
9245         ddi_put32(acc_hdl, &io_request->SenseBufferLowAddress, ars_dmaaddrlow);
6386         if ((cmd->cmd_flags & (CFLAG_SCBEXTERN | CFLAG_EXTARQBUFVALID)) ==
6387             (CFLAG_SCBEXTERN | CFLAG_EXTARQBUFVALID)) {
6388                 ddi_put32(acc_hdl, &io_request->SenseBufferLowAddress,
6389                     cmd->cmd_ext_arqcookie.dmac_address);
6390         } else {
6391                 ddi_put32(acc_hdl, &io_request->SenseBufferLowAddress,
```



```
6392                     cmd->cmd_arqcookie.dmac_address);
6393         }

9247         ddi_put32(acc_hdl, &io_request->Control, control);

9249         NDBG31(("starting message=%d(0x%p), with cmd=0x%p",
9250             SMID, (void *)io_request, (void *)cmd));
6397         NDBG31(("starting message=0x%p, with cmd=0x%p",
6398             (void *)(uintptr_t)mpt->m_req_frame_dma_addr, (void *)cmd));

9252         (void) ddi_dma_sync(dma_hdl, 0, 0, DDI_DMA_SYNC_FORDEV);

9254         /*
9255          * Build request descriptor and write it to the request desc post reg.
9256          */
9257         request_desc |= (SMID << 16) + (MSIidx << 8);
9258         request_desc |= ((uint64_t)ptgt->m_devhdl << 48);
9259         MPTSAS_START_CMD(mpt, request_desc);
6405         request_desc_low = (SMID << 16) + MPI2_REQ_DESCRIPT_FLAGS_SCSI_IO;
6406         request_desc_high = ptgt->m_devhdl << 16;
6407         MPTSAS_START_CMD(mpt, request_desc_low, request_desc_high);

6409         /*
6410          * Start timeout.
6411          */
6412 #ifdef MPTSAS_TEST
6413         /*
6414          * Temporarily set timebase = 0;  needed for
6415          * timeout torture test.
6416          */
6417         if (mptsas_test_timeouts) {
6418                 ptgt->m_timebase = 0;
6419         }
6420 #endif
6421         n = pkt->pkt_time - ptgt->m_timebase;

6423         if (n == 0) {
6424                 (ptgt->m_dups)++;
6425                 ptgt->m_timeout = ptgt->m_timebase;
6426         } else if (n > 0) {
6427                 ptgt->m_timeout =
6428                     ptgt->m_timebase = pkt->pkt_time;
6429                 ptgt->m_dups = 1;
6430         } else if (n < 0) {
6431                 ptgt->m_timeout = ptgt->m_timebase;
6432         }
6433 #ifdef MPTSAS_TEST
6434         /*
6435          * Set back to a number higher than
6436          * mptsas_scsi_watchdog_tick
6437          * so timeouts will happen in mptsas_watchsubr
6438          */
6439         if (mptsas_test_timeouts) {
6440                 ptgt->m_timebase = 60;
6441         }
6442 #endif

9261 #if 0
9262         /* Is this of any benefit here, what is it going to catch? */
9263 #endif /* ! codereview */
9264         if ((mptsas_check_dma_handle(dma_hdl) != DDI_SUCCESS) ||
9265             (mptsas_check_acc_handle(acc_hdl) != DDI_SUCCESS)) {
9266                 ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_UNAFFECTED);
9267                 return (DDI_FAILURE);
9268         }
9269 #endif
```

```
#endif /* ! codereview */
        return (DDI_SUCCESS);
}

/*
 * Select a helper thread to handle given doneq.
 * Note that we don't require to have the main m_mutex here, but worst case
 * is that we wont follow the thread rotation to the letter.
 * However must ensure we have the mutex that covers the source dlist when
 * we actually hand off.
 * Select a helper thread to handle current doneq
 */
static void
mptsas_deliver_doneq_thread(mptsas_t *mpt, mptsas_done_list_t *dlist)
mptsas_deliver_doneq_thread(mptsas_t *mpt)
{
        uint32_t                        t, i, j = mpt->m_doneq_next_thread;
        uint64_t                        t, i;
        uint32_t                        min = 0xffffffff;
        mptsas_doneq_thread_list_t      *item;

        /*
         * No need to take indivudual list mutex's during the loop.
         * We are only reading values and the worst that will happen is that
         * we pick the wrong thread.
         */
#endif /* ! codereview */
        for (i = 0; i < mpt->m_doneq_thread_n; i++) {
                item = &mpt->m_doneq_thread_id[j];

                item = &mpt->m_doneq_thread_id[i];
                /*
                 * If the completed command on help thread[i] less than
                 * doneq_thread_threshold, then pick the thread[j]. Otherwise
                 * doneq_thread_threshold, then pick the thread[i]. Otherwise
                 * pick a thread which has least completed command.
                 */
                if (item->dlist.dl_len < mpt->m_doneq_thread_threshold) {
                        t = j;

                mutex_enter(&item->mutex);
                if (item->len < mpt->m_doneq_thread_threshold) {
                        t = i;
                        mutex_exit(&item->mutex);
                        break;
                }
                if (item->dlist.dl_len < min) {
                        min = item->dlist.dl_len;
                        t = j;
                }
                if (++j == mpt->m_doneq_thread_n) {
                        j = 0;
                }
                if (item->len < min) {
                        min = item->len;
                        t = i;
        }
        item = &mpt->m_doneq_thread_id[t];
        mutex_enter(&item->mutex);
        mptsas_doneq_mv(dlist, item);
        cv_signal(&item->cv);
#endif /* ! codereview */
        mutex_exit(&item->mutex);

        /*
         * Next time start at the next thread.
```



```
         * This will minimize the potential of grabing a lock
         * for a thread that is busy, either on a very busy systems
         * or on one that is configured to do all command completion
         * processing through threads.
         */
        if (++t == mpt->m_doneq_thread_n) {
                t = 0;
#endif /* ! codereview */
        }
        mpt->m_doneq_next_thread = (uint16_t)t;
        mutex_enter(&mpt->m_doneq_thread_id[t].mutex);
        mptsas_doneq_mv(mpt, t);
        cv_signal(&mpt->m_doneq_thread_id[t].cv);
        mutex_exit(&mpt->m_doneq_thread_id[t].mutex);
}

/*
 * move one doneq to another.
 * move the current global doneq to the doneq of thead[t]
 */
static void
mptsas_doneq_mv(mptsas_done_list_t *from, mptsas_doneq_thread_list_t *item)
mptsas_doneq_mv(mptsas_t *mpt, uint64_t t)
{
        mptsas_done_list_t              *to = &item->dlist;
#endif /* ! codereview */
        mptsas_cmd_t                    *cmd;
        mptsas_doneq_thread_list_t      *item = &mpt->m_doneq_thread_id[t];

        if ((cmd = from->dl_q) != NULL) {
                *to->dl_tail = cmd;
                to->dl_tail = from->dl_tail;
                to->dl_len += from->dl_len;
                from->dl_q = NULL;
                from->dl_tail = &from->dl_q;
                from->dl_len = 0;
        ASSERT(mutex_owned(&item->mutex));
        while ((cmd = mpt->m_doneq) != NULL) {
                if ((mpt->m_doneq = cmd->cmd_linkp) == NULL) {
                        mpt->m_donetail = &mpt->m_doneq;
                }
                cmd->cmd_linkp = NULL;
                *item->donetail = cmd;
                item->donetail = &cmd->cmd_linkp;
                mpt->m_doneq_len--;
                item->len++;
        }
}

void
mptsas_fma_check(mptsas_t *mpt, mptsas_cmd_t *cmd)
{
        struct scsi_pkt *pkt = CMD2PKT(cmd);

        /* Check all acc and dma handles */
        if ((mptsas_check_acc_handle(mpt->m_datap) !=
            DDI_SUCCESS) ||
            (mptsas_check_acc_handle(mpt->m_acc_req_frame_hdl) !=
            DDI_SUCCESS) ||
            (mptsas_check_acc_handle(mpt->m_acc_req_sense_hdl) !=
            DDI_SUCCESS) ||
#endif /* ! codereview */
            (mptsas_check_acc_handle(mpt->m_acc_reply_frame_hdl) !=
            DDI_SUCCESS) ||
            (mptsas_check_acc_handle(mpt->m_acc_free_queue_hdl) !=
            DDI_SUCCESS) ||
```

```
9372             (mptsas_check_acc_handle(mpt->m_acc_post_queue_hdl) !=
9373             DDI_SUCCESS) ||
9374             (mptsas_check_acc_handle(mpt->m_hshk_acc_hdl) !=
9375             DDI_SUCCESS) ||
9376             (mptsas_check_acc_handle(mpt->m_config_handle) !=
9377             DDI_SUCCESS)) {
9378                 ddi_fm_service_impact(mpt->m_dip,
9379                     DDI_SERVICE_UNAFFECTED);
9380                 ddi_fm_acc_err_clear(mpt->m_config_handle,
9381                     DDI_FME_VER0);
9382                 pkt->pkt_reason = CMD_TRAN_ERR;
9383                 pkt->pkt_statistics = 0;
9384         }
9385         if ((mptsas_check_dma_handle(mpt->m_dma_req_frame_hdl) !=
9386             DDI_SUCCESS) ||
9387             (mptsas_check_dma_handle(mpt->m_dma_req_sense_hdl) !=
9388             DDI_SUCCESS) ||
9389 #endif /* ! codereview */
9390             (mptsas_check_dma_handle(mpt->m_dma_reply_frame_hdl) !=
9391             DDI_SUCCESS) ||
9392             (mptsas_check_dma_handle(mpt->m_dma_free_queue_hdl) !=
9393             DDI_SUCCESS) ||
9394             (mptsas_check_dma_handle(mpt->m_dma_post_queue_hdl) !=
9395             DDI_SUCCESS) ||
9396             (mptsas_check_dma_handle(mpt->m_hshk_dma_hdl) !=
9397             DDI_SUCCESS)) {
9398                 ddi_fm_service_impact(mpt->m_dip,
9399                     DDI_SERVICE_UNAFFECTED);
9400                 pkt->pkt_reason = CMD_TRAN_ERR;
9401                 pkt->pkt_statistics = 0;
9402         }
9403         if (cmd->cmd_dmahandle &&
9404             (mptsas_check_dma_handle(cmd->cmd_dmahandle) != DDI_SUCCESS)) {
9405                 ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_UNAFFECTED);
9406                 pkt->pkt_reason = CMD_TRAN_ERR;
9407                 pkt->pkt_statistics = 0;
9408         }
9409         if ((cmd->cmd_extra_frames &&
9410             ((mptsas_check_dma_handle(cmd->cmd_extra_frames->m_dma_hdl) !=
9411             DDI_SUCCESS) ||
9412             (mptsas_check_acc_handle(cmd->cmd_extra_frames->m_acc_hdl) !=
9413             DDI_SUCCESS)))) {
9414                 ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_UNAFFECTED);
9415                 pkt->pkt_reason = CMD_TRAN_ERR;
9416                 pkt->pkt_statistics = 0;
9417         }
9418 }

9420 /*
9421  * These routines manipulate the queue of commands that
9422  * are waiting for their completion routines to be called.
9423  * The queue is usually in FIFO order but on an MP system
9424  * it's possible for the completion routines to get out
9425  * of order. If that's a problem you need to add a global
9426  * mutex around the code that calls the completion routine
9427  * in the interrupt handler.
9428  */
9429 static void
9430 mptsas_doneq_add(mptsas_t *mpt, mptsas_cmd_t *cmd)
9431 {
9432         struct scsi_pkt *pkt = CMD2PKT(cmd);

9434         NDBG31(("mptsas_doneq_add: cmd=0x%p", (void *)cmd));

9436         ASSERT((cmd->cmd_flags & CFLAG_COMPLETED) == 0);
9437         cmd->cmd_linkp = NULL;
```



```
9438         cmd->cmd_flags |= CFLAG_FINISHED;
9439         cmd->cmd_flags &= ~CFLAG_IN_TRANSPORT;

9441         mptsas_fma_check(mpt, cmd);

9443         /*
9444          * only add scsi pkts that have completion routines to
9445          * the doneq.  no intr cmds do not have callbacks.
9446          */
9447         if (pkt && (pkt->pkt_comp)) {
9448                 *mpt->m_dlist.dl_tail = cmd;
9449                 mpt->m_dlist.dl_tail = &cmd->cmd_linkp;
9450                 mpt->m_dlist.dl_len++;
6507         if (cmd->cmd_arqhandle &&
6508             (mptsas_check_dma_handle(cmd->cmd_arqhandle) != DDI_SUCCESS)) {
6509                 ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_UNAFFECTED);
6510                 pkt->pkt_reason = CMD_TRAN_ERR;
6511                 pkt->pkt_statistics = 0;
6512         }
6513         if (cmd->cmd_ext_arqhandle &&
6514             (mptsas_check_dma_handle(cmd->cmd_ext_arqhandle) != DDI_SUCCESS)) {
6515                 ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_UNAFFECTED);
6516                 pkt->pkt_reason = CMD_TRAN_ERR;
6517                 pkt->pkt_statistics = 0;
9451         }
9452 }

6521 /*
6522  * These routines manipulate the queue of commands that
6523  * are waiting for their completion routines to be called.
6524  * The queue is usually in FIFO order but on an MP system
6525  * it's possible for the completion routines to get out
6526  * of order. If that's a problem you need to add a global
6527  * mutex around the code that calls the completion routine
6528  * in the interrupt handler.
6529  */
9454 static void
9455 mptsas_rpdoneq_add(mptsas_t *mpt, mptsas_reply_pqueue_t *rpqp,
9456     mptsas_cmd_t *cmd)
6531 mptsas_doneq_add(mptsas_t *mpt, mptsas_cmd_t *cmd)
9457 {
9458         struct scsi_pkt *pkt = CMD2PKT(cmd);

9460         NDBG31(("mptsas_rpdoneq_add: cmd=0x%p", (void *)cmd));
6535         NDBG31(("mptsas_doneq_add: cmd=0x%p", (void *)cmd));

9462         ASSERT((cmd->cmd_flags & CFLAG_COMPLETED) == 0);
9463         cmd->cmd_linkp = NULL;
9464         cmd->cmd_flags |= CFLAG_FINISHED;
9465         cmd->cmd_flags &= ~CFLAG_IN_TRANSPORT;

9467         mptsas_fma_check(mpt, cmd);

9469         /*
9470          * only add scsi pkts that have completion routines to
9471          * the doneq.  no intr cmds do not have callbacks.
9472          */
9473         if (pkt && (pkt->pkt_comp)) {
9474                 *rpqp->rpq_dlist.dl_tail = cmd;
9475                 rpqp->rpq_dlist.dl_tail = &cmd->cmd_linkp;
9476                 rpqp->rpq_dlist.dl_len++;
6549                 *mpt->m_donetail = cmd;
6550                 mpt->m_donetail = &cmd->cmd_linkp;
6551                 mpt->m_doneq_len++;
9477         }
9478 }
```

```
9480 static mptsas_cmd_t *
9481 mptsas_doneq_thread_rm(mptsas_t *mpt, uint64_t t)
9482 {
9483         mptsas_cmd_t                    *cmd;
9484         mptsas_doneq_thread_list_t      *item = &mpt->m_doneq_thread_id[t];

9486         /* pop one off the done queue */
9487         if ((cmd = item->dlist.dl_q) != NULL) {
6562         if ((cmd = item->doneq) != NULL) {
9488                 /* if the queue is now empty fix the tail pointer */
9489                 NDBG31(("mptsas_doneq_thread_rm: cmd=0x%p", (void *)cmd));
9490                 if ((item->dlist.dl_q = cmd->cmd_linkp) == NULL) {
9491                         item->dlist.dl_tail = &item->dlist.dl_q;
6565                 if ((item->doneq = cmd->cmd_linkp) == NULL) {
6566                         item->donetail = &item->doneq;
9492                 }
9493                 cmd->cmd_linkp = NULL;
9494                 item->dlist.dl_len--;
6569                 item->len--;
9495         }
9496         return (cmd);
9497 }

9499 static void
9500 mptsas_doneq_empty(mptsas_t *mpt)
9501 {
9502         if (mpt->m_dlist.dl_q) {
6577         if (mpt->m_doneq && !mpt->m_in_callback) {
9503                 mptsas_cmd_t    *cmd, *next;
9504                 struct scsi_pkt *pkt;

9506                 cmd = mpt->m_dlist.dl_q;
9507                 mpt->m_dlist.dl_q = NULL;
9508                 mpt->m_dlist.dl_tail = &mpt->m_dlist.dl_q;
9509                 mpt->m_dlist.dl_len = 0;
6581                 mpt->m_in_callback = 1;
6582                 cmd = mpt->m_doneq;
6583                 mpt->m_doneq = NULL;
6584                 mpt->m_donetail = &mpt->m_doneq;
6585                 mpt->m_doneq_len = 0;

9511                 mutex_exit(&mpt->m_mutex);
9512                 /*
9513                  * run the completion routines of all the
9514                  * completed commands
9515                  */
9516                 while (cmd != NULL) {
9517                         next = cmd->cmd_linkp;
9518                         cmd->cmd_linkp = NULL;
9519                         /* run this command's completion routine */
9520                         cmd->cmd_flags |= CFLAG_COMPLETED;
9521                         pkt = CMD2PKT(cmd);
9522                         mptsas_pkt_comp(pkt, cmd);
9523                         cmd = next;
9524                 }
9525                 mutex_enter(&mpt->m_mutex);
9526         }
9527 }

9529 static void
9530 mptsas_rpdoneq_empty(mptsas_reply_pqueue_t *rpqp)
9531 {
9532         if (rpqp->rpq_dlist.dl_q) {
9533                 mptsas_cmd_t    *cmd, *next;
9534                 struct scsi_pkt *pkt;
```



```
9536                 cmd = rpqp->rpq_dlist.dl_q;
9537                 rpqp->rpq_dlist.dl_q = NULL;
9538                 rpqp->rpq_dlist.dl_tail = &rpqp->rpq_dlist.dl_q;
9539                 rpqp->rpq_dlist.dl_len = 0;

9541                 mutex_exit(&rpqp->rpq_mutex);
9542                 /*
9543                  * run the completion routines of all the
9544                  * completed commands
9545                  */
9546                 while (cmd != NULL) {
9547                         next = cmd->cmd_linkp;
9548                         cmd->cmd_linkp = NULL;
9549                         /* run this command's completion routine */
9550                         cmd->cmd_flags |= CFLAG_COMPLETED;
9551                         pkt = CMD2PKT(cmd);
9552                         mptsas_pkt_comp(pkt, cmd);
9553                         cmd = next;
9554                 }
9555                 mutex_enter(&rpqp->rpq_mutex);
9602                 mpt->m_in_callback = 0;
9556         }
9557 }
```

_____***unchanged_portion_omitted_***

```
6692 static mptsas_cmd_t *
6693 mptsas_tx_waitq_rm(mptsas_t *mpt)
6694 {
6695         mptsas_cmd_t *cmd;
6696         NDBG7(("mptsas_tx_waitq_rm"));

6698         MPTSAS_TX_WAITQ_RM(mpt, cmd);

6700         NDBG7(("mptsas_tx_waitq_rm: cmd=0x%p", (void *)cmd));

6702         return (cmd);
6703 }

6705 /*
6706  * remove specified cmd from the middle of the tx_waitq.
6707  */
6708 static void
6709 mptsas_tx_waitq_delete(mptsas_t *mpt, mptsas_cmd_t *cmd)
6710 {
6711         mptsas_cmd_t *prevp = mpt->m_tx_waitq;

6713         NDBG7(("mptsas_tx_waitq_delete: mpt=0x%p cmd=0x%p",
6714             (void *)mpt, (void *)cmd));

6716         if (prevp == cmd) {
6717                 if ((mpt->m_tx_waitq = cmd->cmd_linkp) == NULL)
6718                         mpt->m_tx_waitqtail = &mpt->m_tx_waitq;

6720                 cmd->cmd_linkp = NULL;
6721                 cmd->cmd_queued = FALSE;
6722                 NDBG7(("mptsas_tx_waitq_delete: mpt=0x%p cmd=0x%p",
6723                     (void *)mpt, (void *)cmd));
6724                 return;
6725         }

6727         while (prevp != NULL) {
6728                 if (prevp->cmd_linkp == cmd) {
6729                         if ((prevp->cmd_linkp = cmd->cmd_linkp) == NULL)
6730                                 mpt->m_tx_waitqtail = &prevp->cmd_linkp;
```

```
6732                        cmd->cmd_linkp = NULL;
6733                        cmd->cmd_queued = FALSE;
6734                        NDBG7(("mptsas_tx_waitq_delete: mpt=0x%p cmd=0x%p",
6735                            (void *)mpt, (void *)cmd));
6736                        return;
6737                }
6738                prevp = prevp->cmd_linkp;
6739        }
6740        cmn_err(CE_PANIC, "mpt: mptsas_tx_waitq_delete: queue botch");
6741 }

9645 /*
9646  * device and bus reset handling
9647  *
9648  * Notes:
9649  *      - RESET_ALL:    reset the controller
9650  *      - RESET_TARGET: reset the target specified in scsi_address
9651  */
9652 static int
9653 mptsas_scsi_reset(struct scsi_address *ap, int level)
9654 {
9655         mptsas_t                *mpt = ADDR2MPT(ap);
9656         int                     rval;
9657         mptsas_tgt_private_t    *tgt_private;
9658         mptsas_target_t         *ptgt = NULL;

9660         tgt_private = (mptsas_tgt_private_t *)ap->a_hba_tran->tran_tgt_private;
9661         ptgt = tgt_private->t_private;
9662         if (ptgt == NULL) {
9663                 return (FALSE);
9664         }
9665         NDBG22(("mptsas_scsi_reset: target=%d level=%d", ptgt->m_devhdl,
9666             level));

9668         mutex_enter(&mpt->m_mutex);
9669         /*
9670          * if we are not in panic set up a reset delay for this target
9671          */
9672         if (!ddi_in_panic()) {
9673                 mptsas_setup_bus_reset_delay(mpt);
9674         } else {
9675                 drv_usecwait(mpt->m_scsi_reset_delay * 1000);
9676         }
9677         rval = mptsas_do_scsi_reset(mpt, ptgt->m_devhdl);
9678         mutex_exit(&mpt->m_mutex);

9680         /*
9681          * The transport layer expect to only see TRUE and
9682          * FALSE. Therefore, we will adjust the return value
9683          * if mptsas_do_scsi_reset returns FAILED.
9684          */
9685         if (rval == FAILED)
9686                 rval = FALSE;
9687         return (rval);
9688 }
______unchanged_portion_omitted_

9760 static void
6858 void
9761 mptsas_set_throttle(mptsas_t *mpt, mptsas_target_t *ptgt, int what)
9762 {

9764         NDBG25(("mptsas_set_throttle: throttle=%x", what));

9766         /*
9767          * if the bus is draining/quiesced, no changes to the throttles
```



```
9768          * are allowed. Not allowing change of throttles during draining
9769          * limits error recovery but will reduce draining time
9770          *
9771          * all throttles should have been set to HOLD_THROTTLE
9772          */
9773         if (mpt->m_softstate & (MPTSAS_SS_QUIESCED | MPTSAS_SS_DRAINING)) {
9774                 return;
9775         }

9777         if (what == HOLD_THROTTLE) {
9778                 ptgt->m_t_throttle = HOLD_THROTTLE;
9779         } else if (ptgt->m_reset_delay == 0) {
9780                 ptgt->m_t_throttle = what;
9781         }
9782 }

9784 static void
9785 mptsas_set_throttle_mtx(mptsas_t *mpt, mptsas_target_t *ptgt, int what)
9786 {
9787         if (mpt->m_softstate & (MPTSAS_SS_QUIESCED | MPTSAS_SS_DRAINING)) {
9788                 return;
9789         }

9791         mutex_enter(&ptgt->m_t_mutex);
9792         mptsas_set_throttle(mpt, ptgt, what);
9793         mutex_exit(&ptgt->m_t_mutex);
9794 }

9796 /*
9797  * Find all commands in the tx_waitq's for target and lun (if lun not -1),
9798  * remove them from the queues and return the linked list.
9799  */
9800 static mptsas_cmd_t *
9801 mptsas_strip_targetlun_from_txwqs(mptsas_t *mpt, ushort_t target, int lun)
9802 {
9803         mptsas_cmd_t            *cmd, *clist, **tailp, **prev_tailp;
9804         mptsas_tx_waitqueue_t   *txwq;
9805         int                     i;

9807         clist = NULL;
9808         tailp = &clist;

9810         for (i = 0; i < NUM_TX_WAITQ; i++) {
9811                 txwq = &mpt->m_tx_waitq[i];
9812                 mutex_enter(&txwq->txwq_mutex);
9813                 prev_tailp = &txwq->txwq_cmdq;
9814                 cmd = txwq->txwq_cmdq;
9815                 while (cmd != NULL) {
9816                         if (Tgt(cmd) == target &&
9817                             (lun == -1 || (Lun(cmd) == lun))) {
9818                                 *prev_tailp = cmd->cmd_linkp;
9819                                 *tailp = cmd;
9820                                 tailp = &cmd->cmd_linkp;
9821                                 cmd = cmd->cmd_linkp;
9822                                 *tailp = NULL;
9823                         } else {
9824                                 prev_tailp = &cmd->cmd_linkp;
9825                                 cmd = cmd->cmd_linkp;
9826                         }
9827                 }
9828                 txwq->txwq_qtail = prev_tailp;
9829                 mutex_exit(&txwq->txwq_mutex);
9830         }
9831         return (clist);
9832 }
```

```
#endif /* ! codereview */
/*
 * Clean up from a device reset.
 * For the case of target reset, this function clears the waitq of all
 * commands for a particular target.   For the case of abort task set, this
 * function clears the waitq of all commonds for a particular target/lun.
 */
static void
mptsas_flush_target(mptsas_t *mpt, ushort_t target, int lun, uint8_t tasktype)
{
        mptsas_slots_t  *slots = mpt->m_active;
        mptsas_cmd_t    *cmd, *next_cmd;
        int             slot;
        uchar_t         reason;
        uint_t          stat;
        hrtime_t        timestamp;
#endif /* ! codereview */

        NDBG25(("mptsas_flush_target: target=%d lun=%d", target, lun));

        timestamp = gethrtime();

#endif /* ! codereview */
        /*
         * Make sure the I/O Controller has flushed all cmds
         * that are associated with this target for a target reset
         * and target/lun for abort task set.
         * Account for TM requests, which use the last SMID.
         */
        for (slot = 0; slot <= mpt->m_active->m_n_normal; slot++) {
                if ((cmd = slots->m_slot[slot]) == NULL)
                        continue;
                reason = CMD_RESET;
                stat = STAT_DEV_RESET;
                switch (tasktype) {
                case MPI2_SCSITASKMGMT_TASKTYPE_TARGET_RESET:
                        if (Tgt(cmd) == target) {
                                if (cmd->cmd_active_expiration <= timestamp) {
                                        /*
                                         * When timeout requested, propagate
                                         * proper reason and statistics to
                                         * target drivers.
                                         */
                                        reason = CMD_TIMEOUT;
                                        stat |= STAT_TIMEOUT;
                                }
#endif /* ! codereview */
                                NDBG25(("mptsas_flush_target discovered non-"
                                    "NULL cmd in slot %d, tasktype 0x%x", slot,
                                    tasktype));
                                mptsas_dump_cmd(mpt, cmd);
                                mptsas_remove_cmd(mpt, cmd);
                                mptsas_set_pkt_reason(mpt, cmd, reason, stat);
                                mptsas_doneq_add(mpt, cmd);
                        }
                        break;
                case MPI2_SCSITASKMGMT_TASKTYPE_ABRT_TASK_SET:
                        reason = CMD_ABORTED;
                        stat = STAT_ABORTED;
                        /*FALLTHROUGH*/
                case MPI2_SCSITASKMGMT_TASKTYPE_LOGICAL_UNIT_RESET:
                        if ((Tgt(cmd) == target) && (Lun(cmd) == lun)) {
                                if (cmd->cmd_active_expiration <= timestamp) {
                                        stat |= STAT_TIMEOUT;
                                }
#endif /* ! codereview */
```



```
                                NDBG25(("mptsas_flush_target discovered non-"
                                    "NULL cmd in slot %d, tasktype 0x%x", slot,
                                    tasktype));
                                mptsas_dump_cmd(mpt, cmd);
                                mptsas_remove_cmd(mpt, cmd);
                                mptsas_set_pkt_reason(mpt, cmd, reason, stat);
                                mptsas_set_pkt_reason(mpt, cmd,
                                    stat);
                                mptsas_doneq_add(mpt, cmd);
                        }
                        break;
                default:
                        break;
                }
        }

        /*
         * Flush the waitq and tx_waitq of this target's cmds
         */
        cmd = mpt->m_waitq;

        reason = CMD_RESET;
        stat = STAT_DEV_RESET;

        switch (tasktype) {
        case MPI2_SCSITASKMGMT_TASKTYPE_TARGET_RESET:
                while (cmd != NULL) {
                        next_cmd = cmd->cmd_linkp;
                        if (Tgt(cmd) == target) {
                                mptsas_waitq_delete(mpt, cmd);
                                mptsas_set_pkt_reason(mpt, cmd,
                                    reason, stat);
                                mptsas_doneq_add(mpt, cmd);
                        }
                        cmd = next_cmd;
                }
                cmd = mptsas_strip_targetlun_from_txwqs(mpt, target, -1);
                mutex_enter(&mpt->m_tx_waitq_mutex);
                cmd = mpt->m_tx_waitq;
                while (cmd != NULL) {
                        next_cmd = cmd->cmd_linkp;
                        mptsas_set_pkt_reason(mpt, cmd, reason, stat);
                        if (Tgt(cmd) == target) {
                                mptsas_tx_waitq_delete(mpt, cmd);
                                mutex_exit(&mpt->m_tx_waitq_mutex);
                                mptsas_set_pkt_reason(mpt, cmd,
                                    reason, stat);
                                mptsas_doneq_add(mpt, cmd);
                                mutex_enter(&mpt->m_tx_waitq_mutex);
                        }
                        cmd = next_cmd;
                }
                mutex_exit(&mpt->m_tx_waitq_mutex);
                break;
        case MPI2_SCSITASKMGMT_TASKTYPE_ABRT_TASK_SET:
                reason = CMD_ABORTED;
                stat =  STAT_ABORTED;
                /*FALLTHROUGH*/
        case MPI2_SCSITASKMGMT_TASKTYPE_LOGICAL_UNIT_RESET:
                while (cmd != NULL) {
                        next_cmd = cmd->cmd_linkp;
                        if ((Tgt(cmd) == target) && (Lun(cmd) == lun)) {
                                mptsas_waitq_delete(mpt, cmd);
                                mptsas_set_pkt_reason(mpt, cmd,
                                    reason, stat);
```

```
9954                                    mptsas_doneq_add(mpt, cmd);
9955                            }
9956                            cmd = next_cmd;
9957                    }
9958                    cmd = mptsas_strip_targetlun_from_txwqs(mpt, target, lun);
6943                    mutex_enter(&mpt->m_tx_waitq_mutex);
6944                    cmd = mpt->m_tx_waitq;
9959                    while (cmd != NULL) {
9960                            next_cmd = cmd->cmd_linkp;
9961                            mptsas_set_pkt_reason(mpt, cmd, reason, stat);
6947                            if ((Tgt(cmd) == target) && (Lun(cmd) == lun)) {
6948                                    mptsas_tx_waitq_delete(mpt, cmd);
6949                                    mutex_exit(&mpt->m_tx_waitq_mutex);
6950                                    mptsas_set_pkt_reason(mpt, cmd,
6951                                        reason, stat);
9962                            mptsas_doneq_add(mpt, cmd);
6953                                    mutex_enter(&mpt->m_tx_waitq_mutex);
6954                            }
9963                            cmd = next_cmd;
9964                    }
6957                    mutex_exit(&mpt->m_tx_waitq_mutex);
9965                    break;
9966            default:
9967                    mptsas_log(mpt, CE_WARN, "Unknown task management type %d.",
9968                        tasktype);
9969                    break;
9970            }
9971 }

9973 /*
9974  * Clean up hba state, abort all outstanding command and commands in waitq
9975  * reset timeout of all targets.
9976  */
9977 static void
9978 mptsas_flush_hba(mptsas_t *mpt)
9979 {
9980            mptsas_slots_t  *slots = mpt->m_active;
9981            mptsas_cmd_t    *cmd, *ncmd;
9982            int             slot, i;
6974            mptsas_cmd_t    *cmd;
6975            int             slot;

9984            NDBG25(("mptsas_flush_hba"));

9986            /*
9987             * The I/O Controller should have already sent back
9988             * all commands via the scsi I/O reply frame.  Make
9989             * sure all commands have been flushed.
9990             * Account for TM request, which use the last SMID.
9991             */
9992            for (slot = 0; slot <= mpt->m_active->m_n_normal; slot++) {
9993                    if ((cmd = slots->m_slot[slot]) == NULL)
9994                            continue;

9996                    if (cmd->cmd_flags & CFLAG_CMDIOC) {
9997                            /*
9998                             * Need to make sure to tell everyone that might be
9999                             * waiting on this command that it's going to fail.  If
10000                            * we get here, this command will never timeout because
10001                            * the active command table is going to be re-allocated,
10002                            * so there will be nothing to check against a time out.
10003                            * Instead, mark the command as failed due to reset.
10004                            */
10005                           mptsas_set_pkt_reason(mpt, cmd, CMD_RESET,
10006                               STAT_BUS_RESET);
10007                           if ((cmd->cmd_flags &
```



```
10008                                   (CFLAG_PASSTHRU | CFLAG_CONFIG | CFLAG_FW_DIAG))) {
7000                            if ((cmd->cmd_flags & CFLAG_PASSTHRU) ||
7001                                (cmd->cmd_flags & CFLAG_CONFIG) ||
7002                                (cmd->cmd_flags & CFLAG_FW_DIAG)) {
10009                                   cmd->cmd_flags |= CFLAG_FINISHED;
10010                                   cv_broadcast(&mpt->m_passthru_cv);
10011                                   cv_broadcast(&mpt->m_config_cv);
10012                                   cv_broadcast(&mpt->m_fw_diag_cv);
10013                           }
10014                           continue;
10015                   }

10017                   NDBG25(("mptsas_flush_hba discovered non-NULL cmd in slot %d",
10018                       slot));
10019                   mptsas_dump_cmd(mpt, cmd);

10021                   mptsas_remove_cmd(mpt, cmd);
10022                   mptsas_set_pkt_reason(mpt, cmd, CMD_RESET, STAT_BUS_RESET);
10023                   mptsas_doneq_add(mpt, cmd);
10024           }

10026           /*
10027            * Flush the waitq.
10028            */
10029           while ((cmd = mptsas_waitq_rm(mpt)) != NULL) {
10030                   mptsas_set_pkt_reason(mpt, cmd, CMD_RESET, STAT_BUS_RESET);
10031                   if ((cmd->cmd_flags & CFLAG_PASSTHRU) ||
10032                       (cmd->cmd_flags & CFLAG_CONFIG) ||
10033                       (cmd->cmd_flags & CFLAG_FW_DIAG)) {
10034                           cmd->cmd_flags |= CFLAG_FINISHED;
10035                           cv_broadcast(&mpt->m_passthru_cv);
10036                           cv_broadcast(&mpt->m_config_cv);
10037                           cv_broadcast(&mpt->m_fw_diag_cv);
10038                   } else {
10039                           mptsas_doneq_add(mpt, cmd);
10040                   }
10041           }

10043           /*
10044            * Flush the tx_waitqs
7038            * Flush the tx_waitq
10045            */
10046           for (i = 0; i < NUM_TX_WAITQ; i++) {
10047                   mutex_enter(&mpt->m_tx_waitq[i].txwq_mutex);
10048                   cmd = mpt->m_tx_waitq[i].txwq_cmdq;
10049                   mpt->m_tx_waitq[i].txwq_cmdq = NULL;
10050                   mpt->m_tx_waitq[i].txwq_qtail = &mpt->m_tx_waitq[i].txwq_cmdq;
10051                   mutex_exit(&mpt->m_tx_waitq[i].txwq_mutex);
10052                   while (cmd != NULL) {
10053                           ncmd = cmd->cmd_linkp;
10054                           mptsas_set_pkt_reason(mpt, cmd, CMD_RESET,
10055                               STAT_BUS_RESET);
7040           mutex_enter(&mpt->m_tx_waitq_mutex);
7041           while ((cmd = mptsas_tx_waitq_rm(mpt)) != NULL) {
7042                   mutex_exit(&mpt->m_tx_waitq_mutex);
7043                   mptsas_set_pkt_reason(mpt, cmd, CMD_RESET, STAT_BUS_RESET);
10056                           mptsas_doneq_add(mpt, cmd);
10057                           cmd = ncmd;
10058                   }
7045                   mutex_enter(&mpt->m_tx_waitq_mutex);
10059           }
7047           mutex_exit(&mpt->m_tx_waitq_mutex);

10061           /*
10062            * Drain the taskqs prior to reallocating resources.
10063            */
```

```
10064         mutex_exit(&mpt->m_mutex);
10065         ddi_taskq_wait(mpt->m_event_taskq);
10066         ddi_taskq_wait(mpt->m_dr_taskq);
10067         mutex_enter(&mpt->m_mutex);
10068 }
_____unchanged_portion_omitted_

10107 static void
10108 mptsas_setup_bus_reset_delay(mptsas_t *mpt)
10109 {
10110         mptsas_target_t *ptgt = NULL;

10112         ASSERT(MUTEX_HELD(&mpt->m_mutex));

10114         NDBG22(("mptsas_setup_bus_reset_delay"));
10115         for (ptgt = refhash_first(mpt->m_targets); ptgt != NULL;
10116             ptgt = refhash_next(mpt->m_targets, ptgt)) {
10117                 mutex_enter(&ptgt->m_t_mutex);
10118 #endif /* ! codereview */
10119                 mptsas_set_throttle(mpt, ptgt, HOLD_THROTTLE);
10120                 ptgt->m_reset_delay = mpt->m_scsi_reset_delay;
10121                 mutex_exit(&ptgt->m_t_mutex);
10122 #endif /* ! codereview */
10123         }

10125         mptsas_start_watch_reset_delay();
10126 }

10128 /*
10129  * mptsas_watch_reset_delay(_subr) is invoked by timeout() and checks every
10130  * mpt instance for active reset delays
10131  */
10132 static void
10133 mptsas_watch_reset_delay(void *arg)
10134 {
10135 #ifndef __lock_lint
10136         _NOTE(ARGUNUSED(arg))
10137 #endif

10139         mptsas_t        *mpt;
10140         int             not_done = 0;

10142         NDBG22(("mptsas_watch_reset_delay"));

10144         mutex_enter(&mptsas_global_mutex);
10145         mptsas_reset_watch = 0;
10146         mutex_exit(&mptsas_global_mutex);
10147         rw_enter(&mptsas_global_rwlock, RW_READER);
10148         for (mpt = mptsas_head; mpt != NULL; mpt = mpt->m_next) {
10149                 if (mpt->m_tran == 0) {
10150                         continue;
10151                 }
10152                 mutex_enter(&mpt->m_mutex);
10153                 not_done += mptsas_watch_reset_delay_subr(mpt);
10154                 mutex_exit(&mpt->m_mutex);
10155         }
10156         rw_exit(&mptsas_global_rwlock);

10158         if (not_done) {
10159                 mptsas_start_watch_reset_delay();
10160         }
10161 }

10163 static int
10164 mptsas_watch_reset_delay_subr(mptsas_t *mpt)
10165 {
```



```
10166         int             done = 0;
10167         int             restart = 0;
10168         mptsas_target_t *ptgt = NULL;

10170         NDBG22(("mptsas_watch_reset_delay_subr: mpt=0x%p", (void *)mpt));

10172         ASSERT(mutex_owned(&mpt->m_mutex));

10174         for (ptgt = refhash_first(mpt->m_targets); ptgt != NULL;
10175             ptgt = refhash_next(mpt->m_targets, ptgt)) {
10176                 mutex_enter(&ptgt->m_t_mutex);
10177 #endif /* ! codereview */
10178                 if (ptgt->m_reset_delay != 0) {
10179                         ptgt->m_reset_delay -=
10180                             MPTSAS_WATCH_RESET_DELAY_TICK;
10181                         if (ptgt->m_reset_delay <= 0) {
10182                                 ptgt->m_reset_delay = 0;
10183                                 mptsas_set_throttle(mpt, ptgt,
10184                                     MAX_THROTTLE);
10185                                 restart++;
10186                         } else {
10187                                 done = -1;
10188                         }
10189                 }
10190                 mutex_exit(&ptgt->m_t_mutex);
10191 #endif /* ! codereview */
10192         }

10194         if (restart > 0) {
10195                 mptsas_restart_hba(mpt);
10196         }
10197         return (done);
10198 }

10200 #ifdef MPTSAS_TEST
10201 static void
10202 mptsas_test_reset(mptsas_t *mpt, int target)
10203 {
10204         mptsas_target_t    *ptgt = NULL;

10206         if (mptsas_rtest == target) {
10207                 if (mptsas_do_scsi_reset(mpt, target) == TRUE) {
10208                         mptsas_rtest = -1;
10209                 }
10210                 if (mptsas_rtest == -1) {
10211                         NDBG22(("mptsas_test_reset success"));
10212                 }
10213         }
10214 }
10215 #endif

10217 /*
10218  * abort handling:
10219  *
10220  * Notes:
10221  *      - if pkt is not NULL, abort just that command
10222  *      - if pkt is NULL, abort all outstanding commands for target
10223  */
10224 static int
10225 mptsas_scsi_abort(struct scsi_address *ap, struct scsi_pkt *pkt)
10226 {
10227         mptsas_t                *mpt = ADDR2MPT(ap);
10228         int                     rval;
10229         mptsas_tgt_private_t    *tgt_private;
10230         int                     target, lun;
```

```
10232         tgt_private = (mptsas_tgt_private_t *)ap->a_hba_tran->
10233             tran_tgt_private;
10234         ASSERT(tgt_private != NULL);
10235         target = tgt_private->t_private->m_devhdl;
10236         lun = tgt_private->t_lun;

10238         NDBG23(("mptsas_scsi_abort: target=%d.%d", target, lun));

10240         mutex_enter(&mpt->m_mutex);
10241         rval = mptsas_do_scsi_abort(mpt, target, lun, pkt);
10242         mutex_exit(&mpt->m_mutex);
10243         return (rval);
10244 }

10246 static int
10247 mptsas_do_scsi_abort(mptsas_t *mpt, int target, int lun, struct scsi_pkt *pkt)
10248 {
10249         mptsas_cmd_t    *sp = NULL;
10250         mptsas_slots_t  *slots = mpt->m_active;
10251         int             rval = FALSE;

10253         ASSERT(mutex_owned(&mpt->m_mutex));

10255         /*
10256          * Abort the command pkt on the target/lun in ap.  If pkt is
10257          * NULL, abort all outstanding commands on that target/lun.
10258          * If you can abort them, return 1, else return 0.
10259          * Each packet that's aborted should be sent back to the target
10260          * driver through the callback routine, with pkt_reason set to
10261          * CMD_ABORTED.
10262          *
10263          * abort cmd pkt on HBA hardware; clean out of outstanding
10264          * command lists, etc.
10265          */
10266         if (pkt != NULL) {
10267                 /* abort the specified packet */
10268                 sp = PKT2CMD(pkt);

10270                 if (sp->cmd_queued) {
10271                         NDBG23(("mptsas_do_scsi_abort: queued sp=0x%p aborted",
10272                             (void *)sp));
10273                         mptsas_waitq_delete(mpt, sp);
10274                         mptsas_set_pkt_reason(mpt, sp, CMD_ABORTED,
10275                             STAT_ABORTED);
10276                         mptsas_doneq_add(mpt, sp);
10277                         rval = TRUE;
10278                         goto done;
10279                 }

10281                 /*
10282                  * Have mpt firmware abort this command
10283                  */

10285                 if (slots->m_slot[sp->cmd_slot] != NULL) {
10286                         rval = mptsas_ioc_task_management(mpt,
10287                             MPI2_SCSITASKMGMT_TASKTYPE_ABORT_TASK, target,
10288                             lun, NULL, 0, 0);

10290                         /*
10291                          * The transport layer expects only TRUE and FALSE.
10292                          * Therefore, if mptsas_ioc_task_management returns
10293                          * FAILED we will return FALSE.
10294                          */
10295                         if (rval == FAILED)
10296                                 rval = FALSE;
10297                         goto done;
```



```
10298                 }
10299         }

10301         /*
10302          * If pkt is NULL then abort task set
10303          */
10304         rval = mptsas_ioc_task_management(mpt,
10305             MPI2_SCSITASKMGMT_TASKTYPE_ABRT_TASK_SET, target, lun, NULL, 0, 0);

10307         /*
10308          * The transport layer expects only TRUE and FALSE.
10309          * Therefore, if mptsas_ioc_task_management returns
10310          * FAILED we will return FALSE.
10311          */
10312         if (rval == FAILED)
10313                 rval = FALSE;

10315 #ifdef MPTSAS_TEST
10316         if (rval && mptsas_test_stop) {
10317                 debug_enter("mptsas_do_scsi_abort");
10318         }
10319 #endif

10321 done:
10322         mptsas_doneq_empty(mpt);
10323         return (rval);
10324 }

10326 /*
10327  * capability handling:
10328  * (*tran_getcap).  Get the capability named, and return its value.
10329  */
10330 static int
10331 mptsas_scsi_getcap(struct scsi_address *ap, char *cap, int tgtonly)
10332 {
10333         mptsas_t        *mpt = ADDR2MPT(ap);
10334         int             ckey;
10335         int             rval = FALSE;

10337         NDBG24(("mptsas_scsi_getcap: target=%d, cap=%s tgtonly=%x",
10338             ap->a_target, cap, tgtonly));

10340         mutex_enter(&mpt->m_mutex);

10342         if ((mptsas_scsi_capchk(cap, tgtonly, &ckey)) != TRUE) {
10343                 mutex_exit(&mpt->m_mutex);
10344                 return (UNDEFINED);
10345         }

10347         switch (ckey) {
10348         case SCSI_CAP_DMA_MAX:
10349                 rval = (int)mpt->m_msg_dma_attr.dma_attr_maxxfer;
10350                 break;
10351         case SCSI_CAP_ARQ:
10352                 rval = TRUE;
10353                 break;
10354         case SCSI_CAP_MSG_OUT:
10355         case SCSI_CAP_PARITY:
10356         case SCSI_CAP_UNTAGGED_QING:
10357                 rval = TRUE;
10358                 break;
10359         case SCSI_CAP_TAGGED_QING:
10360                 rval = TRUE;
10361                 break;
10362         case SCSI_CAP_RESET_NOTIFICATION:
10363                 rval = TRUE;
```

```
10364                   break;
10365           case SCSI_CAP_LINKED_CMDS:
10366                   rval = FALSE;
10367                   break;
10368           case SCSI_CAP_QFULL_RETRIES:
10369                   rval = ((mptsas_tgt_private_t *)(ap->a_hba_tran->
10370                       tran_tgt_private))->t_private->m_qfull_retries;
10371                   break;
10372           case SCSI_CAP_QFULL_RETRY_INTERVAL:
10373                   rval = drv_hztousec(((mptsas_tgt_private_t *)
10374                       (ap->a_hba_tran->tran_tgt_private))->
10375                       t_private->m_qfull_retry_interval) / 1000;
10376                   break;
10377           case SCSI_CAP_CDB_LEN:
10378                   rval = CDB_GROUP4;
10379                   break;
10380           case SCSI_CAP_INTERCONNECT_TYPE:
10381                   rval = INTERCONNECT_SAS;
10382                   break;
10383           case SCSI_CAP_TRAN_LAYER_RETRIES:
10384                   if (mpt->m_ioc_capabilities &
10385                       MPI2_IOCFACTS_CAPABILITY_TLR)
10386                           rval = TRUE;
10387                   else
10388                           rval = FALSE;
10389                   break;
10390           default:
10391                   rval = UNDEFINED;
10392                   break;
10393           }

10395           NDBG24(("mptsas_scsi_getcap: %s, rval=%x", cap, rval));

10397           mutex_exit(&mpt->m_mutex);
10398           return (rval);
10399   }

10401   /*
10402    * (*tran_setcap).  Set the capability named to the value given.
10403    */
10404   static int
10405   mptsas_scsi_setcap(struct scsi_address *ap, char *cap, int value, int tgtonly)
10406   {
10407           mptsas_t        *mpt = ADDR2MPT(ap);
10408           mptsas_target_t *ptgt;
10409   #endif /* ! codereview */
10410           int             ckey;
10411           int             rval = FALSE;

10413           NDBG24(("mptsas_scsi_setcap: target=%d, cap=%s value=%x tgtonly=%x",
10414               ap->a_target, cap, value, tgtonly));

10416           if (!tgtonly) {
10417                   return (rval);
10418           }

10420           mutex_enter(&mpt->m_mutex);

10422           if ((mptsas_scsi_capchk(cap, tgtonly, &ckey)) != TRUE) {
10423                   mutex_exit(&mpt->m_mutex);
10424                   return (UNDEFINED);
10425           }

10427           switch (ckey) {
10428           case SCSI_CAP_DMA_MAX:
10429           case SCSI_CAP_MSG_OUT:
```



```
10430           case SCSI_CAP_PARITY:
10431           case SCSI_CAP_INITIATOR_ID:
10432           case SCSI_CAP_LINKED_CMDS:
10433           case SCSI_CAP_UNTAGGED_QING:
10434           case SCSI_CAP_RESET_NOTIFICATION:
10435                   /*
10436                    * None of these are settable via
10437                    * the capability interface.
10438                    */
10439                   break;
10440           case SCSI_CAP_ARQ:
10441                   /*
10442                    * We cannot turn off arq so return false if asked to
10443                    */
10444                   if (value) {
10445                           rval = TRUE;
10446                   } else {
10447                           rval = FALSE;
10448                   }
10449                   break;
10450           case SCSI_CAP_TAGGED_QING:
10451                   ptgt = ((mptsas_tgt_private_t *)
10452                       (ap->a_hba_tran->tran_tgt_private))->t_private;
10453                   mptsas_set_throttle_mtx(mpt, ptgt, MAX_THROTTLE);
7105                  mptsas_set_throttle(mpt, ((mptsas_tgt_private_t *)
7106                      (ap->a_hba_tran->tran_tgt_private))->t_private,
7107                      MAX_THROTTLE);
10454                  rval = TRUE;
10455                  break;
10456          case SCSI_CAP_QFULL_RETRIES:
10457                  ((mptsas_tgt_private_t *)(ap->a_hba_tran->tran_tgt_private))->
10458                      t_private->m_qfull_retries = (uchar_t)value;
10459                  rval = TRUE;
10460                  break;
10461          case SCSI_CAP_QFULL_RETRY_INTERVAL:
10462                  ((mptsas_tgt_private_t *)(ap->a_hba_tran->tran_tgt_private))->
10463                      t_private->m_qfull_retry_interval =
10464                      drv_usectohz(value * 1000);
10465                  rval = TRUE;
10466                  break;
10467          default:
10468                  rval = UNDEFINED;
10469                  break;
10470          }
10471          mutex_exit(&mpt->m_mutex);
10472          return (rval);
10473  }
_____unchanged_portion_omitted_

10538  /*
10539   * Error logging, printing, and debug print routines.
10540   */
10541  static char *mptsas_label = "mpt_sas3";
7195  static char *mptsas_label = "mpt_sas";

10543  /*PRINTFLIKE3*/
10544  void
10545  mptsas_log(mptsas_t *mpt, int level, char *fmt, ...)
10546  {
10547          dev_info_t      *dev;
10548          va_list         ap;

10550          if (mpt) {
10551                  dev = mpt->m_dip;
10552          } else {
10553                  dev = 0;
```

```
10554         }

10556         mutex_enter(&mptsas_log_mutex);

10558         va_start(ap, fmt);
10559         (void) vsprintf(mptsas_log_buf, fmt, ap);
10560         va_end(ap);

10562         if (level == CE_CONT) {
10563                 scsi_log(dev, mptsas_label, level, "%s\n", mptsas_log_buf);
10564         } else {
10565                 scsi_log(dev, mptsas_label, level, "%s", mptsas_log_buf);
10566         }

10568         mutex_exit(&mptsas_log_mutex);
10569 }

10571 #ifdef MPTSAS_DEBUG
10572 /*
10573  * Use a circular buffer to log messages to private memory.
10574  * No mutexes, so there is the opportunity for this to miss lines.
10575  * But it's fast and does not hold up the proceedings too much.
10576  */
10577 static char mptsas_dbglog_bufs[32][256];
10578 static uint32_t mptsas_dbglog_idx = 1;

10580 /*PRINTFLIKE1*/
10581 void
10582 mptsas_debug_log(char *fmt, ...)
10583 {
10584         va_list         ap;
10585         uint32_t        idx;

10587         if (!mptsas_dbglog_idx) {
10588                 return;
10589         }
10590         idx = (mptsas_dbglog_idx++) & 0x1f;

10592         va_start(ap, fmt);
10593         (void) vsnprintf(mptsas_dbglog_bufs[idx],
10594             sizeof (mptsas_dbglog_bufs[0]), fmt, ap);
10595         va_end(ap);
10596 }

10598 #endif /* ! codereview */
10599 /*PRINTFLIKE1*/
10600 void
10601 mptsas_printf(char *fmt, ...)
10602 {
10603         dev_info_t      *dev = 0;
10604         va_list         ap;

10606         mutex_enter(&mptsas_log_mutex);

10608         va_start(ap, fmt);
10609         (void) vsprintf(mptsas_log_buf, fmt, ap);
10610         va_end(ap);

10612 #ifdef PROM_PRINTF
10613         prom_printf("%s:\t%s\n", mptsas_label, mptsas_log_buf);
10614 #else
10615         scsi_log(dev, mptsas_label, CE_CONT, "!%s\n", mptsas_log_buf);
7226         scsi_log(dev, mptsas_label, SCSI_DEBUG, "%s\n", mptsas_log_buf);
10616 #endif
10617         mutex_exit(&mptsas_log_mutex);
10618 }
```

*unchanged_portion_omitted_*


```
10687 int mptsas_monitor_for_txwqs = 1;
10688 #endif /* ! codereview */
10689 static void
10690 mptsas_watchsubr(mptsas_t *mpt)
10691 {
10692         int             i;
10693         mptsas_cmd_t    *cmd;
10694         mptsas_target_t *ptgt = NULL;
10695         hrtime_t        timestamp = gethrtime();
10696         boolean_t       restart_hba = B_FALSE;
10697 #endif /* ! codereview */

10699         ASSERT(MUTEX_HELD(&mpt->m_mutex));

10701         NDBG30(("mptsas_watchsubr: mpt=0x%p, ncmds %d, nstarted %d",
10702             (void *)mpt, mpt->m_ncmds, mpt->m_ncstarted));
7298         NDBG30(("mptsas_watchsubr: mpt=0x%p", (void *)mpt));

10704         mpt->m_lncstarted = mpt->m_ncstarted;
10705         if (mpt->m_txwq_thread_n != 0 && mpt->m_txwq_enabled != BLOCKED &&
10706             mptsas_monitor_for_txwqs) {
10707                 i = mpt->m_ncstarted/mptsas_scsi_watchdog_tick;
10708                 if (i > mpt->m_txwq_thread_threshold) {
10709                         mpt->m_txwq_enabled = TRUE;
10710                 } else if (i < (mpt->m_txwq_thread_threshold>>1)) {
10711                         mpt->m_txwq_enabled = FALSE;
10712                 }
7300 #ifdef MPTSAS_TEST
7301         if (mptsas_enable_untagged) {
7302                 mptsas_test_untagged++;
10713         }
10714         mpt->m_ncstarted = 0;
7304 #endif

10716         /*
10717          * Check for commands stuck in active slot
10718          * Account for TM requests, which use the last SMID.
10719          */
10720         for (i = 0; i <= mpt->m_active->m_n_normal; i++) {
10721                 if ((cmd = mpt->m_active->m_slot[i]) != NULL) {
10722                         if (cmd->cmd_active_expiration <= timestamp) {
10723 #endif /* ! codereview */
10724                                 if ((cmd->cmd_flags & CFLAG_CMDIOC) == 0) {
7312                                 cmd->cmd_active_timeout -=
7313                                     mptsas_scsi_watchdog_tick;
7314                                 if (cmd->cmd_active_timeout <= 0) {
10725                                         /*
10726                                          * There seems to be a command stuck
10727                                          * in the active slot.  Drain throttle.
10728                                          */
10729                                         ptgt = cmd->cmd_tgt_addr;
10730                                         mptsas_set_throttle_mtx(mpt, ptgt,
7319                                         mptsas_set_throttle(mpt,
7320                                             cmd->cmd_tgt_addr,
10731                                              DRAIN_THROTTLE);
10732                                 } else if (cmd->cmd_flags &
10733                                     (CFLAG_PASSTHRU | CFLAG_CONFIG |
10734                                     CFLAG_FW_DIAG)) {
7322                                 }
7323                         }
7324                         if ((cmd->cmd_flags & CFLAG_PASSTHRU) ||
7325                             (cmd->cmd_flags & CFLAG_CONFIG) ||
7326                             (cmd->cmd_flags & CFLAG_FW_DIAG)) {
7327                                 cmd->cmd_active_timeout -=
7328                                     mptsas_scsi_watchdog_tick;
```

```
7329                                if (cmd->cmd_active_timeout <= 0) {
10735                                       /*
10736                                        * passthrough command timeout
10737                                        */
10738                                       cmd->cmd_flags |= (CFLAG_FINISHED |
10739                                           CFLAG_TIMEOUT);
10740                                       cv_broadcast(&mpt->m_passthru_cv);
10741                                       cv_broadcast(&mpt->m_config_cv);
10742                                       cv_broadcast(&mpt->m_fw_diag_cv);
10743                                }
10744                        }
10745                }
10746        }

10748        for (ptgt = refhash_first(mpt->m_targets); ptgt != NULL;
10749            ptgt = refhash_next(mpt->m_targets, ptgt)) {
10750                mutex_enter(&ptgt->m_t_mutex);
10751 #endif /* ! codereview */
10752                /*
10753                 * If we were draining due to a qfull condition,
10754                 * go back to full throttle.
10755                 */
10756                if ((ptgt->m_t_throttle < MAX_THROTTLE) &&
10757                    (ptgt->m_t_throttle > HOLD_THROTTLE) &&
10758                    (ptgt->m_t_ncmds < ptgt->m_t_throttle)) {
10759                        mptsas_set_throttle(mpt, ptgt, MAX_THROTTLE);
10760                        restart_hba = B_TRUE;
7345                        mptsas_restart_hba(mpt);
10761                }

10763                cmd = TAILQ_LAST(&ptgt->m_active_cmdq, mptsas_active_cmdq);
10764                if (cmd != NULL) {
10765                        if (cmd->cmd_active_expiration <= timestamp) {
10766                                /*
10767                                 * Earliest command timeout expired.
10768                                 * Drain throttle.
10769                                 */
10770                                mptsas_set_throttle(mpt, ptgt, DRAIN_THROTTLE);
7348                if ((ptgt->m_t_ncmds > 0) &&
7349                    (ptgt->m_timebase)) {

10772                                /*
10773                                 * Check for remaining commands.
10774                                 */
10775                                cmd = TAILQ_FIRST(&ptgt->m_active_cmdq);
10776                                if (cmd->cmd_active_expiration > timestamp) {
10777                                        /*
10778                                         * Wait for remaining commands to
10779                                         * complete or time out.
10780                                         */
10781                                        NDBG23(("command timed out, "
10782                                            "pending drain"));
10783                                } else {
10784                                        mutex_exit(&ptgt->m_t_mutex);
7351                        if (ptgt->m_timebase <=
7352                            mptsas_scsi_watchdog_tick) {
7353                                ptgt->m_timebase +=
7354                                    mptsas_scsi_watchdog_tick;
7355                                continue;
7356                        }

10786                                        /*
10787                                         * All command timeouts expired.
10788                                         */
10789                                        mptsas_log(mpt, CE_NOTE,
10790                                            "Timeout of %d seconds "
```



```
10791                                           "expired with %d commands on "
10792                                           "target %d lun %d.",
10793                                           cmd->cmd_pkt->pkt_time,
10794                                           ptgt->m_t_ncmds,
10795                                           ptgt->m_devhdl, Lun(cmd));
7358                        ptgt->m_timeout -= mptsas_scsi_watchdog_tick;

10797                                   mptsas_cmd_timeout(mpt, ptgt);
7360                        if (ptgt->m_timeout < 0) {
7361                                mptsas_cmd_timeout(mpt, ptgt->m_devhdl);
10798                                   continue;
10799                           }
10800                        } else if (cmd->cmd_active_expiration <= timestamp +
10801                            (hrtime_t)mptsas_scsi_watchdog_tick * NANOSEC) {

7365                        if ((ptgt->m_timeout) <=
7366                            mptsas_scsi_watchdog_tick) {
10802                               NDBG23(("pending timeout"));
10803                               mptsas_set_throttle(mpt, ptgt, DRAIN_THROTTLE);
10804                        }
7368                                mptsas_set_throttle(mpt, ptgt,
7369                                    DRAIN_THROTTLE);
10805                }
10806                mutex_exit(&ptgt->m_t_mutex);
10807 #endif /* ! codereview */
10808        }
10809        if (restart_hba == B_TRUE) {
10810                mptsas_restart_hba(mpt);
10811 #endif /* ! codereview */
10812        }
10813 }

10815 /*
10816  * timeout recovery
10817  */
10818 static void
10819 mptsas_cmd_timeout(mptsas_t *mpt, mptsas_target_t *ptgt)
7371 mptsas_cmd_timeout(mptsas_t *mpt, uint16_t devhdl)
10820 {
10821        uint16_t        devhdl;
10822        uint64_t        sas_wwn;
10823        uint8_t         phy;
10824        char            wwn_str[MPTSAS_WWN_STRLEN];

10826        devhdl = ptgt->m_devhdl;
10827        sas_wwn = ptgt->m_addr.mta_wwn;
10828        phy = ptgt->m_phynum;
10829        if (sas_wwn == 0) {
10830                (void) sprintf(wwn_str, "p%x", phy);
10831        } else {
10832                (void) sprintf(wwn_str, "w%016"PRIx64, sas_wwn);
10833        }
10834 #endif /* ! codereview */

10836        NDBG29(("mptsas_cmd_timeout: target=%d", devhdl));
10837        mptsas_log(mpt, CE_WARN, "Disconnected command timeout for "
10838            "target %d %s,  enclosure %u .", devhdl, wwn_str,
10839            ptgt->m_enclosure);
7373           "Target %d", devhdl);

10841        /*
10842         * Abort all outstanding commands on the device.
7376         * If the current target is not the target passed in,
7377         * try to reset that target.
10843         */
10844        NDBG29(("mptsas_cmd_timeout: device reset"));
```

```
10845         if (mptsas_do_scsi_reset(mpt, devhdl) != TRUE) {
10846                 mptsas_log(mpt, CE_WARN, "Target %d reset for command timeout "
10847                     "recovery failed!", devhdl);
10848         }
10849 }
______unchanged_portion_omitted_

10880 static int
10881 mptsas_quiesce_bus(mptsas_t *mpt)
10882 {
10883         mptsas_target_t *ptgt = NULL;

10885         NDBG28(("mptsas_quiesce_bus"));
10886         mutex_enter(&mpt->m_mutex);

10888         /* Set all the throttles to zero */
10889         for (ptgt = refhash_first(mpt->m_targets); ptgt != NULL;
10890             ptgt = refhash_next(mpt->m_targets, ptgt)) {
10891                 mptsas_set_throttle_mtx(mpt, ptgt, HOLD_THROTTLE);
7426                mptsas_set_throttle(mpt, ptgt, HOLD_THROTTLE);
10892         }

10894         /* If there are any outstanding commands in the queue */
10895         if (mpt->m_ncmds) {
10896                 mpt->m_softstate |= MPTSAS_SS_DRAINING;
10897                 mpt->m_quiesce_timeid = timeout(mptsas_ncmds_checkdrain,
10898                     mpt, (MPTSAS_QUIESCE_TIMEOUT * drv_usectohz(1000000)));
10899                 if (cv_wait_sig(&mpt->m_cv, &mpt->m_mutex) == 0) {
10900                         /*
10901                          * Quiesce has been interrupted
10902                          */
10903                         mpt->m_softstate &= ~MPTSAS_SS_DRAINING;
10904                         for (ptgt = refhash_first(mpt->m_targets); ptgt != NULL;
10905                             ptgt = refhash_next(mpt->m_targets, ptgt)) {
10906                                 mptsas_set_throttle_mtx(mpt, ptgt,
10907                                     MAX_THROTTLE);
7441                               mptsas_set_throttle(mpt, ptgt, MAX_THROTTLE);
10908                         }
10909                         mptsas_restart_hba(mpt);
10910                         if (mpt->m_quiesce_timeid != 0) {
10911                                 timeout_id_t tid = mpt->m_quiesce_timeid;
10912                                 mpt->m_quiesce_timeid = 0;
10913                                 mutex_exit(&mpt->m_mutex);
10914                                 (void) untimeout(tid);
10915                                 return (-1);
10916                         }
10917                         mutex_exit(&mpt->m_mutex);
10918                         return (-1);
10919                 } else {
10920                         /* Bus has been quiesced */
10921                         ASSERT(mpt->m_quiesce_timeid == 0);
10922                         mpt->m_softstate &= ~MPTSAS_SS_DRAINING;
10923                         mpt->m_softstate |= MPTSAS_SS_QUIESCED;
10924                         mutex_exit(&mpt->m_mutex);
10925                         return (0);
10926                 }
10927         }
10928         /* Bus was not busy - QUIESCED */
10929         mutex_exit(&mpt->m_mutex);

10931         return (0);
10932 }

10934 static int
10935 mptsas_unquiesce_bus(mptsas_t *mpt)
10936 {
```



```
10937         mptsas_target_t *ptgt = NULL;

10939         NDBG28(("mptsas_unquiesce_bus"));
10940         mutex_enter(&mpt->m_mutex);
10941         mpt->m_softstate &= ~MPTSAS_SS_QUIESCED;
10942         for (ptgt = refhash_first(mpt->m_targets); ptgt != NULL;
10943             ptgt = refhash_next(mpt->m_targets, ptgt)) {
10944                 mptsas_set_throttle_mtx(mpt, ptgt, MAX_THROTTLE);
7478               mptsas_set_throttle(mpt, ptgt, MAX_THROTTLE);
10945         }
10946         mptsas_restart_hba(mpt);
10947         mutex_exit(&mpt->m_mutex);
10948         return (0);
10949 }

10951 static void
10952 mptsas_ncmds_checkdrain(void *arg)
10953 {
10954         mptsas_t        *mpt = arg;
10955         mptsas_target_t *ptgt = NULL;

10957         mutex_enter(&mpt->m_mutex);
10958         if (mpt->m_softstate & MPTSAS_SS_DRAINING) {
10959                 mpt->m_quiesce_timeid = 0;
10960                 if (mpt->m_ncmds == 0) {
10961                         /* Command queue has been drained */
10962                         cv_signal(&mpt->m_cv);
10963                 } else {
10964                         /*
10965                          * The throttle may have been reset because
10966                          * of a SCSI bus reset
10967                          */
10968                         for (ptgt = refhash_first(mpt->m_targets); ptgt != NULL;
10969                             ptgt = refhash_next(mpt->m_targets, ptgt)) {
10970                                 mptsas_set_throttle_mtx(mpt, ptgt,
10971                                     HOLD_THROTTLE);
7504                               mptsas_set_throttle(mpt, ptgt, HOLD_THROTTLE);
10972                         }

10974                         mpt->m_quiesce_timeid = timeout(mptsas_ncmds_checkdrain,
10975                             mpt, (MPTSAS_QUIESCE_TIMEOUT *
10976                             drv_usectohz(1000000)));
10977                 }
10978         }
10979         mutex_exit(&mpt->m_mutex);
10980 }

10982 /*ARGSUSED*/
10983 static void
10984 mptsas_dump_cmd(mptsas_t *mpt, mptsas_cmd_t *cmd)
10985 {
10986         int     i;
10987         uint8_t *cp = (uchar_t *)cmd->cmd_pkt->pkt_cdbp;
10988         char    buf[128];

10990         buf[0] = '\0';
10991         NDBG25(("?Cmd (0x%p) dump for Target %d Lun %d:\n", (void *)cmd,
10992             Tgt(cmd), Lun(cmd)));
10993         (void) sprintf(&buf[0], "\tcdb=[");
10994         for (i = 0; i < (int)cmd->cmd_cdblen; i++) {
10995                 (void) sprintf(&buf[strlen(buf)], " 0x%x", *cp++);
10996         }
10997         (void) sprintf(&buf[strlen(buf)], " ]");
10998         NDBG25(("?%s\n", buf));
10999         NDBG25(("?pkt_flags=0x%x pkt_statistics=0x%x pkt_state=0x%x\n",
11000             cmd->cmd_pkt->pkt_flags, cmd->cmd_pkt->pkt_statistics,
```

```
11001             cmd->cmd_pkt->pkt_state));
11002         NDBG25(("?pkt_scbp=0x%x cmd_flags=0x%x\n", cmd->cmd_pkt->pkt_scbp ?
11003             *(cmd->cmd_pkt->pkt_scbp) : 0, cmd->cmd_flags));
11004 }

11006 static void
11007 mptsas_passthru_sge(ddi_acc_handle_t acc_hdl, mptsas_pt_request_t *pt,
11008     pMpi2SGESimple64_t sgep)
11009 {
11010         uint32_t                sge_flags;
11011         uint32_t                data_size, dataout_size;
11012         ddi_dma_cookie_t        data_cookie;
11013         ddi_dma_cookie_t        dataout_cookie;

11015         data_size = pt->data_size;
11016         dataout_size = pt->dataout_size;
11017         data_cookie = pt->data_cookie;
11018         dataout_cookie = pt->dataout_cookie;

11020         if (dataout_size) {
11021                 sge_flags = dataout_size |
11022                     ((uint32_t)(MPI2_SGE_FLAGS_SIMPLE_ELEMENT |
11023                     MPI2_SGE_FLAGS_END_OF_BUFFER |
11024                     MPI2_SGE_FLAGS_HOST_TO_IOC |
11025                     MPI2_SGE_FLAGS_64_BIT_ADDRESSING) <<
11026                     MPI2_SGE_FLAGS_SHIFT);
11027                 ddi_put32(acc_hdl, &sgep->FlagsLength, sge_flags);
11028                 ddi_put32(acc_hdl, &sgep->Address.Low,
11029                     (uint32_t)(dataout_cookie.dmac_laddress & 0xffffffffull));
11030                 ddi_put32(acc_hdl, &sgep->Address.High,
11031                     (uint32_t)(dataout_cookie.dmac_laddress >> 32));
11032                 sgep++;
11033         }
11034         sge_flags = data_size;
11035         sge_flags |= ((uint32_t)(MPI2_SGE_FLAGS_SIMPLE_ELEMENT |
11036             MPI2_SGE_FLAGS_LAST_ELEMENT |
11037             MPI2_SGE_FLAGS_END_OF_BUFFER |
11038             MPI2_SGE_FLAGS_END_OF_LIST |
11039             MPI2_SGE_FLAGS_64_BIT_ADDRESSING) <<
11040             MPI2_SGE_FLAGS_SHIFT);
11041         if (pt->direction == MPTSAS_PASS_THRU_DIRECTION_WRITE) {
11042                 sge_flags |= ((uint32_t)(MPI2_SGE_FLAGS_HOST_TO_IOC) <<
11043                     MPI2_SGE_FLAGS_SHIFT);
11044         } else {
11045                 sge_flags |= ((uint32_t)(MPI2_SGE_FLAGS_IOC_TO_HOST) <<
11046                     MPI2_SGE_FLAGS_SHIFT);
11047         }
11048         ddi_put32(acc_hdl, &sgep->FlagsLength, sge_flags);
11049         ddi_put32(acc_hdl, &sgep->Address.Low,
11050             (uint32_t)(data_cookie.dmac_laddress & 0xffffffffull));
11051         ddi_put32(acc_hdl, &sgep->Address.High,
11052             (uint32_t)(data_cookie.dmac_laddress >> 32));
11053 }

11055 static void
11056 mptsas_passthru_ieee_sge(ddi_acc_handle_t acc_hdl, mptsas_pt_request_t *pt,
11057     pMpi2IeeeSgeSimple64_t ieeesgep)
11058 {
11059         uint8_t                 sge_flags;
11060         uint32_t                data_size, dataout_size;
11061         ddi_dma_cookie_t        data_cookie;
11062         ddi_dma_cookie_t        dataout_cookie;

11064         data_size = pt->data_size;
11065         dataout_size = pt->dataout_size;
11066         data_cookie = pt->data_cookie;
```



```
11067         dataout_cookie = pt->dataout_cookie;

11069         sge_flags = (MPI2_IEEE_SGE_FLAGS_SIMPLE_ELEMENT |
11070             MPI2_IEEE_SGE_FLAGS_SYSTEM_ADDR);
11071         if (dataout_size) {
11072                 ddi_put32(acc_hdl, &ieeesgep->Length, dataout_size);
11073                 ddi_put32(acc_hdl, &ieeesgep->Address.Low,
11074                     (uint32_t)(dataout_cookie.dmac_laddress &
11075                     0xffffffffull));
11076                 ddi_put32(acc_hdl, &ieeesgep->Address.High,
11077                     (uint32_t)(dataout_cookie.dmac_laddress >> 32));
11078                 ddi_put8(acc_hdl, &ieeesgep->Flags, sge_flags);
11079                 ieeesgep++;
11080         }
11081         sge_flags |= MPI25_IEEE_SGE_FLAGS_END_OF_LIST;
11082         ddi_put32(acc_hdl, &ieeesgep->Length, data_size);
11083         ddi_put32(acc_hdl, &ieeesgep->Address.Low,
11084             (uint32_t)(data_cookie.dmac_laddress & 0xffffffffull));
11085         ddi_put32(acc_hdl, &ieeesgep->Address.High,
11086             (uint32_t)(data_cookie.dmac_laddress >> 32));
11087         ddi_put8(acc_hdl, &ieeesgep->Flags, sge_flags);
7527         for (i = 0; i < (int)cmd->cmd_cdblen; i++) {
7528                 (void) sprintf(&buf[strlen(buf)], " 0x%x", *cp++);
7529         }
7530         (void) sprintf(&buf[strlen(buf)], " ]");
7531         NDBG25(("?%s\n", buf));
7532         NDBG25(("?pkt_flags=0x%x pkt_statistics=0x%x pkt_state=0x%x\n",
7533             cmd->cmd_pkt->pkt_flags, cmd->cmd_pkt->pkt_statistics,
7534             cmd->cmd_pkt->pkt_state));
7535         NDBG25(("?pkt_scbp=0x%x cmd_flags=0x%x\n", cmd->cmd_pkt->pkt_scbp ?
7536             *(cmd->cmd_pkt->pkt_scbp) : 0, cmd->cmd_flags));
11088 }

11090 static void
11091 mptsas_start_passthru(mptsas_t *mpt, mptsas_cmd_t *cmd)
11092 {
11093         caddr_t                 memp;
11094         pMPI2RequestHeader_t    request_hdrp;
11095         struct scsi_pkt         *pkt = cmd->cmd_pkt;
11096         mptsas_pt_request_t     *pt = pkt->pkt_ha_private;
11097         uint32_t                request_size;
11098         uint64_t                request_desc = 0;
11099         uint64_t                sense_bufp;
7546         uint32_t                request_size, data_size, dataout_size;
7547         uint32_t                direction;
7548         ddi_dma_cookie_t        data_cookie;
7549         ddi_dma_cookie_t        dataout_cookie;
7550         uint32_t                request_desc_low, request_desc_high = 0;
7551         uint32_t                i, sense_bufp;
11100         uint8_t                 desc_type;
11101         uint8_t                 *request, function;
11102         ddi_dma_handle_t        dma_hdl = mpt->m_dma_req_frame_hdl;
11103         ddi_acc_handle_t        acc_hdl = mpt->m_acc_req_frame_hdl;

11105         desc_type = MPI2_REQ_DESCRIPT_FLAGS_DEFAULT_TYPE;

11107         request = pt->request;
7560         direction = pt->direction;
11108         request_size = pt->request_size;
7562         data_size = pt->data_size;
7563         dataout_size = pt->dataout_size;
7564         data_cookie = pt->data_cookie;
7565         dataout_cookie = pt->dataout_cookie;

11110         /*
11111          * Store the passthrough message in memory location
```

```
11112          * corresponding to our slot number
11113          */
11114         memp = mpt->m_req_frame + (mpt->m_req_frame_size * cmd->cmd_slot);
11115         request_hdrp = (pMPI2RequestHeader_t)memp;
11116         bzero(memp, mpt->m_req_frame_size);

11118         bcopy(request, memp, request_size);
7575         for (i = 0; i < request_size; i++) {
7576                 bcopy(request + i, memp + i, 1);
7577         }

11120         NDBG15(("mptsas_start_passthru: Func 0x%x, MsgFlags 0x%x, "
11121             "size=%d, in %d, out %d", request_hdrp->Function,
11122             request_hdrp->MsgFlags, request_size,
11123             pt->data_size, pt->dataout_size));
7579         if (data_size || dataout_size) {
7580                 pMpi2SGESimple64_t      sgep;
7581                 uint32_t                sge_flags;

11125         /*
11126          * Add an SGE, even if the length is zero.
11127          */
11128         if (mpt->m_MPI25 && pt->simple == 0) {
11129                 mptsas_passthru_ieee_sge(acc_hdl, pt,
11130                     (pMpi2IeeeSgeSimple64_t)
11131                     ((uint8_t *)request_hdrp + pt->sgl_offset));
7583                 sgep = (pMpi2SGESimple64_t)((uint8_t *)request_hdrp +
7584                     request_size);
7585                 if (dataout_size) {

7587                         sge_flags = dataout_size |
7588                             ((uint32_t)(MPI2_SGE_FLAGS_SIMPLE_ELEMENT |
7589                             MPI2_SGE_FLAGS_END_OF_BUFFER |
7590                             MPI2_SGE_FLAGS_HOST_TO_IOC |
7591                             MPI2_SGE_FLAGS_64_BIT_ADDRESSING) <<
7592                             MPI2_SGE_FLAGS_SHIFT);
7593                         ddi_put32(acc_hdl, &sgep->FlagsLength, sge_flags);
7594                         ddi_put32(acc_hdl, &sgep->Address.Low,
7595                             (uint32_t)(dataout_cookie.dmac_laddress &
7596                             0xffffffffull));
7597                         ddi_put32(acc_hdl, &sgep->Address.High,
7598                             (uint32_t)(dataout_cookie.dmac_laddress
7599                             >> 32));
7600                         sgep++;
7601                 }
7602                 sge_flags = data_size;
7603                 sge_flags |= ((uint32_t)(MPI2_SGE_FLAGS_SIMPLE_ELEMENT |
7604                     MPI2_SGE_FLAGS_LAST_ELEMENT |
7605                     MPI2_SGE_FLAGS_END_OF_BUFFER |
7606                     MPI2_SGE_FLAGS_END_OF_LIST |
7607                     MPI2_SGE_FLAGS_64_BIT_ADDRESSING) <<
7608                     MPI2_SGE_FLAGS_SHIFT);
7609                 if (direction == MPTSAS_PASS_THRU_DIRECTION_WRITE) {
7610                         sge_flags |= ((uint32_t)(MPI2_SGE_FLAGS_HOST_TO_IOC) <<
7611                             MPI2_SGE_FLAGS_SHIFT);
11132         } else {
11133                 mptsas_passthru_sge(acc_hdl, pt,
11134                     (pMpi2SGESimple64_t)
11135                     ((uint8_t *)request_hdrp + pt->sgl_offset));
7613                         sge_flags |= ((uint32_t)(MPI2_SGE_FLAGS_IOC_TO_HOST) <<
7614                             MPI2_SGE_FLAGS_SHIFT);
7615                 }
7616                 ddi_put32(acc_hdl, &sgep->FlagsLength,
7617                     sge_flags);
7618                 ddi_put32(acc_hdl, &sgep->Address.Low,
7619                     (uint32_t)(data_cookie.dmac_laddress &
```



```
7620                         0xffffffffull));
7621                 ddi_put32(acc_hdl, &sgep->Address.High,
7622                         (uint32_t)(data_cookie.dmac_laddress >> 32));
11136         }

11138         function = request_hdrp->Function;
11139         if ((function == MPI2_FUNCTION_SCSI_IO_REQUEST) ||
11140             (function == MPI2_FUNCTION_RAID_SCSI_IO_PASSTHROUGH)) {
11141                 pMpi2SCSIIORequest_t    scsi_io_req;

11143                 NDBG15(("mptsas_start_passthru: Is SCSI IO Req"));
11144 #endif /* ! codereview */
11145                 scsi_io_req = (pMpi2SCSIIORequest_t)request_hdrp;
11146                 /*
11147                  * Put SGE for data and data_out buffer at the end of
11148                  * scsi_io_request message header.(64 bytes in total)
11149                  * Following above SGEs, the residual space will be
11150                  * used by sense data.
11151                  */
11152                 ddi_put8(acc_hdl,
11153                     &scsi_io_req->SenseBufferLength,
11154                     (uint8_t)(request_size - 64));

11156                 sense_bufp = (uint32_t)(mpt->m_req_frame_dma_addr +
11157                     (mpt->m_req_frame_size * cmd->cmd_slot) & 0xffffffffull);
7630                 sense_bufp = mpt->m_req_frame_dma_addr +
7631                     (mpt->m_req_frame_size * cmd->cmd_slot);
11158                 sense_bufp += 64;
11159                 ddi_put32(acc_hdl,
11160                     &scsi_io_req->SenseBufferLowAddress, sense_bufp);

11162                 /*
11163                  * Set SGLOffset0 value
11164                  */
11165                 ddi_put8(acc_hdl, &scsi_io_req->SGLOffset0,
11166                     offsetof(MPI2_SCSI_IO_REQUEST, SGL) / 4);

11168                 /*
11169                  * Setup descriptor info.  RAID passthrough must use the
11170                  * default request descriptor which is already set, so if this
11171                  * is a SCSI IO request, change the descriptor to SCSI IO.
11172                  */
11173                 if (function == MPI2_FUNCTION_SCSI_IO_REQUEST) {
11174                         desc_type = MPI2_REQ_DESCRIPT_FLAGS_SCSI_IO;
11175                         request_desc = (((uint64_t)ddi_get16(acc_hdl,
11176                             &scsi_io_req->DevHandle)) << 48);
7649                         request_desc_high = (ddi_get16(acc_hdl,
7650                             &scsi_io_req->DevHandle) << 16);
11177                 }
11178         }

11180         /*
11181          * We must wait till the message has been completed before
11182          * beginning the next message so we wait for this one to
11183          * finish.
11184          */
11185         (void) ddi_dma_sync(dma_hdl, 0, 0, DDI_DMA_SYNC_FORDEV);
11186         request_desc |= ((cmd->cmd_slot << 16) | desc_type);
7660         request_desc_low = (cmd->cmd_slot << 16) + desc_type;
11187         cmd->cmd_rfm = NULL;
11188         MPTSAS_START_CMD(mpt, request_desc);
7662         MPTSAS_START_CMD(mpt, request_desc_low, request_desc_high);
11189         if ((mptsas_check_dma_handle(dma_hdl) != DDI_SUCCESS) ||
11190             (mptsas_check_acc_handle(acc_hdl) != DDI_SUCCESS)) {
11191                 ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_UNAFFECTED);
11192         }
```

```
11193 }

11195 typedef void (mps_pre_f)(mptsas_t *, mptsas_pt_request_t *);
11196 static mps_pre_f        mpi_pre_ioc_facts;
11197 static mps_pre_f        mpi_pre_port_facts;
11198 static mps_pre_f        mpi_pre_fw_download;
11199 static mps_pre_f        mpi_pre_fw_25_download;
11200 static mps_pre_f        mpi_pre_fw_upload;
11201 static mps_pre_f        mpi_pre_fw_25_upload;
11202 static mps_pre_f        mpi_pre_sata_passthrough;
11203 static mps_pre_f        mpi_pre_smp_passthrough;
11204 static mps_pre_f        mpi_pre_config;
11205 static mps_pre_f        mpi_pre_sas_io_unit_control;
11206 static mps_pre_f        mpi_pre_scsi_io_req;

11208 /*
11209  * Prepare the pt for a SAS2 FW_DOWNLOAD request.
11210  */
11211 static void
11212 mpi_pre_fw_download(mptsas_t *mpt, mptsas_pt_request_t *pt)
11213 {
11214         pMpi2FWDownloadTCSGE_t tcsge;
11215         pMpi2FWDownloadRequest req;

11217         /*
11218          * If SAS3, call separate function.
11219          */
11220         if (mpt->m_MPI25) {
11221                 mpi_pre_fw_25_download(mpt, pt);
11222                 return;
11223         }

11225         /*
11226          * User requests should come in with the Transaction
11227          * context element where the SGL will go. Putting the
11228          * SGL after that seems to work, but don't really know
11229          * why. Other drivers tend to create an extra SGL and
11230          * refer to the TCE through that.
11231          */
11232         req = (pMpi2FWDownloadRequest)pt->request;
11233         tcsge = (pMpi2FWDownloadTCSGE_t)&req->SGL;
11234         if (tcsge->ContextSize != 0 || tcsge->DetailsLength != 12 ||
11235             tcsge->Flags != MPI2_SGE_FLAGS_TRANSACTION_ELEMENT) {
11236                 mptsas_log(mpt, CE_WARN, "FW Download tce invalid!");
11237         }

11239         pt->sgl_offset = offsetof(MPI2_FW_DOWNLOAD_REQUEST, SGL) +
11240             sizeof (*tcsge);
11241         if (pt->request_size != pt->sgl_offset)
11242                 NDBG15(("mpi_pre_fw_download(): Incorrect req size, "
11243                     "0x%x, should be 0x%x, dataoutsz 0x%x",
11244                     (int)pt->request_size, (int)pt->sgl_offset,
11245                     (int)pt->dataout_size));
11246         if (pt->data_size < sizeof (MPI2_FW_DOWNLOAD_REPLY))
11247                 NDBG15(("mpi_pre_fw_download(): Incorrect rep size, "
11248                     "0x%x, should be 0x%x", pt->data_size,
11249                     (int)sizeof (MPI2_FW_DOWNLOAD_REPLY)));
11250 }

11252 /*
11253  * Prepare the pt for a SAS3 FW_DOWNLOAD request.
11254  */
11255 static void
11256 mpi_pre_fw_25_download(mptsas_t *mpt, mptsas_pt_request_t *pt)
11257 {
11258         pMpi2FWDownloadTCSGE_t tcsge;
```



```
11259         pMpi2FWDownloadRequest req2;
11260         pMpi25FWDownloadRequest req25;

11262         /*
11263          * User requests should come in with the Transaction
11264          * context element where the SGL will go. The new firmware
11265          * Doesn't use TCE and has space in the main request for
11266          * this information. So move to the right place.
11267          */
11268         req2 = (pMpi2FWDownloadRequest)pt->request;
11269         req25 = (pMpi25FWDownloadRequest)pt->request;
11270         tcsge = (pMpi2FWDownloadTCSGE_t)&req2->SGL;
11271         if (tcsge->ContextSize != 0 || tcsge->DetailsLength != 12 ||
11272             tcsge->Flags != MPI2_SGE_FLAGS_TRANSACTION_ELEMENT) {
11273                 mptsas_log(mpt, CE_WARN, "FW Download tce invalid!");
11274         }
11275         req25->ImageOffset = tcsge->ImageOffset;
11276         req25->ImageSize = tcsge->ImageSize;

11278         pt->sgl_offset = offsetof(MPI25_FW_DOWNLOAD_REQUEST, SGL);
11279         if (pt->request_size != pt->sgl_offset)
11280                 NDBG15(("mpi_pre_fw_25_download(): Incorrect req size, "
11281                     "0x%x, should be 0x%x, dataoutsz 0x%x",
11282                     pt->request_size, pt->sgl_offset,
11283                     pt->dataout_size));
11284         if (pt->data_size < sizeof (MPI2_FW_DOWNLOAD_REPLY))
11285                 NDBG15(("mpi_pre_fw_25_download(): Incorrect rep size, "
11286                     "0x%x, should be 0x%x", pt->data_size,
11287                     (int)sizeof (MPI2_FW_UPLOAD_REPLY)));
11288 }

11290 /*
11291  * Prepare the pt for a SAS2 FW_UPLOAD request.
11292  */
11293 static void
11294 mpi_pre_fw_upload(mptsas_t *mpt, mptsas_pt_request_t *pt)
11295 {
11296         pMpi2FWUploadTCSGE_t tcsge;
11297         pMpi2FWUploadRequest_t req;

11299         /*
11300          * If SAS3, call separate function.
11301          */
11302         if (mpt->m_MPI25) {
11303                 mpi_pre_fw_25_upload(mpt, pt);
11304                 return;
11305         }

11307         /*
11308          * User requests should come in with the Transaction
11309          * context element where the SGL will go. Putting the
11310          * SGL after that seems to work, but don't really know
11311          * why. Other drivers tend to create an extra SGL and
11312          * refer to the TCE through that.
11313          */
11314         req = (pMpi2FWUploadRequest_t)pt->request;
11315         tcsge = (pMpi2FWUploadTCSGE_t)&req->SGL;
11316         if (tcsge->ContextSize != 0 || tcsge->DetailsLength != 12 ||
11317             tcsge->Flags != MPI2_SGE_FLAGS_TRANSACTION_ELEMENT) {
11318                 mptsas_log(mpt, CE_WARN, "FW Upload tce invalid!");
11319         }

11321         pt->sgl_offset = offsetof(MPI2_FW_UPLOAD_REQUEST, SGL) +
11322             sizeof (*tcsge);
11323         if (pt->request_size != pt->sgl_offset)
11324                 NDBG15(("mpi_pre_fw_upload(): Incorrect req size, "
```

```
                    "0x%x, should be 0x%x, dataoutsz 0x%x",
                    pt->request_size, pt->sgl_offset,
                    pt->dataout_size));
        if (pt->data_size < sizeof (MPI2_FW_UPLOAD_REPLY))
                NDBG15(("mpi_pre_fw_upload(): Incorrect rep size, "
                    "0x%x, should be 0x%x", pt->data_size,
                    (int)sizeof (MPI2_FW_UPLOAD_REPLY)));
}

/*
 * Prepare the pt a SAS3 FW_UPLOAD request.
 */
static void
mpi_pre_fw_25_upload(mptsas_t *mpt, mptsas_pt_request_t *pt)
{
        pMpi2FWUploadTCSGE_t tcsge;
        pMpi2FWUploadRequest_t req2;
        pMpi25FWUploadRequest_t req25;

        /*
         * User requests should come in with the Transaction
         * context element where the SGL will go. The new firmware
         * Doesn't use TCE and has space in the main request for
         * this information. So move to the right place.
         */
        req2 = (pMpi2FWUploadRequest_t)pt->request;
        req25 = (pMpi25FWUploadRequest_t)pt->request;
        tcsge = (pMpi2FWUploadTCSGE_t)&req2->SGL;
        if (tcsge->ContextSize != 0 || tcsge->DetailsLength != 12 ||
            tcsge->Flags != MPI2_SGE_FLAGS_TRANSACTION_ELEMENT) {
                mptsas_log(mpt, CE_WARN, "FW Upload tce invalid!");
        }
        req25->ImageOffset = tcsge->ImageOffset;
        req25->ImageSize = tcsge->ImageSize;

        pt->sgl_offset = offsetof(MPI25_FW_UPLOAD_REQUEST, SGL);
        if (pt->request_size != pt->sgl_offset)
                NDBG15(("mpi_pre_fw_25_upload(): Incorrect req size, "
                    "0x%x, should be 0x%x, dataoutsz 0x%x",
                    pt->request_size, pt->sgl_offset,
                    pt->dataout_size));
        if (pt->data_size < sizeof (MPI2_FW_UPLOAD_REPLY))
                NDBG15(("mpi_pre_fw_25_upload(): Incorrect rep size, "
                    "0x%x, should be 0x%x", pt->data_size,
                    (int)sizeof (MPI2_FW_UPLOAD_REPLY)));
}

/*
 * Prepare the pt for an IOC_FACTS request.
 */
static void
mpi_pre_ioc_facts(mptsas_t *mpt, mptsas_pt_request_t *pt)
{
#ifndef __lock_lint
        _NOTE(ARGUNUSED(mpt))
#endif
        if (pt->request_size != sizeof (MPI2_IOC_FACTS_REQUEST))
                NDBG15(("mpi_pre_ioc_facts(): Incorrect req size, "
                    "0x%x, should be 0x%x, dataoutsz 0x%x",
                    pt->request_size,
                    (int)sizeof (MPI2_IOC_FACTS_REQUEST),
                    pt->dataout_size));
        if (pt->data_size != sizeof (MPI2_IOC_FACTS_REPLY))
                NDBG15(("mpi_pre_ioc_facts(): Incorrect rep size, "
                    "0x%x, should be 0x%x", pt->data_size,
                    (int)sizeof (MPI2_IOC_FACTS_REPLY)));
```



```
        pt->sgl_offset = (uint16_t)pt->request_size;
}

/*
 * Prepare the pt for a PORT_FACTS request.
 */
static void
mpi_pre_port_facts(mptsas_t *mpt, mptsas_pt_request_t *pt)
{
#ifndef __lock_lint
        _NOTE(ARGUNUSED(mpt))
#endif
        if (pt->request_size != sizeof (MPI2_PORT_FACTS_REQUEST))
                NDBG15(("mpi_pre_port_facts(): Incorrect req size, "
                    "0x%x, should be 0x%x, dataoutsz 0x%x",
                    pt->request_size,
                    (int)sizeof (MPI2_PORT_FACTS_REQUEST),
                    pt->dataout_size));
        if (pt->data_size != sizeof (MPI2_PORT_FACTS_REPLY))
                NDBG15(("mpi_pre_port_facts(): Incorrect rep size, "
                    "0x%x, should be 0x%x", pt->data_size,
                    (int)sizeof (MPI2_PORT_FACTS_REPLY)));
        pt->sgl_offset = (uint16_t)pt->request_size;
}

/*
 * Prepare pt for a SATA_PASSTHROUGH request.
 */
static void
mpi_pre_sata_passthrough(mptsas_t *mpt, mptsas_pt_request_t *pt)
{
#ifndef __lock_lint
        _NOTE(ARGUNUSED(mpt))
#endif
        pt->sgl_offset = offsetof(MPI2_SATA_PASSTHROUGH_REQUEST, SGL);
        if (pt->request_size != pt->sgl_offset)
                NDBG15(("mpi_pre_sata_passthrough(): Incorrect req size, "
                    "0x%x, should be 0x%x, dataoutsz 0x%x",
                    pt->request_size, pt->sgl_offset,
                    pt->dataout_size));
        if (pt->data_size != sizeof (MPI2_SATA_PASSTHROUGH_REPLY))
                NDBG15(("mpi_pre_sata_passthrough(): Incorrect rep size, "
                    "0x%x, should be 0x%x", pt->data_size,
                    (int)sizeof (MPI2_SATA_PASSTHROUGH_REPLY)));
}

static void
mpi_pre_smp_passthrough(mptsas_t *mpt, mptsas_pt_request_t *pt)
{
#ifndef __lock_lint
        _NOTE(ARGUNUSED(mpt))
#endif
        pt->sgl_offset = offsetof(MPI2_SMP_PASSTHROUGH_REQUEST, SGL);
        if (pt->request_size != pt->sgl_offset)
                NDBG15(("mpi_pre_smp_passthrough(): Incorrect req size, "
                    "0x%x, should be 0x%x, dataoutsz 0x%x",
                    pt->request_size, pt->sgl_offset,
                    pt->dataout_size));
        if (pt->data_size != sizeof (MPI2_SMP_PASSTHROUGH_REPLY))
                NDBG15(("mpi_pre_smp_passthrough(): Incorrect rep size, "
                    "0x%x, should be 0x%x", pt->data_size,
                    (int)sizeof (MPI2_SMP_PASSTHROUGH_REPLY)));
}

/*
 * Prepare pt for a CONFIG request.
```

```
 */
static void
mpi_pre_config(mptsas_t *mpt, mptsas_pt_request_t *pt)
{
#ifndef __lock_lint
        _NOTE(ARGUNUSED(mpt))
#endif
        pt->sgl_offset = offsetof(MPI2_CONFIG_REQUEST, PageBufferSGE);
        if (pt->request_size != pt->sgl_offset)
                NDBG15(("mpi_pre_config(): Incorrect req size, 0x%x, "
                    "should be 0x%x, dataoutsz 0x%x", pt->request_size,
                    pt->sgl_offset, pt->dataout_size));
        if (pt->data_size != sizeof (MPI2_CONFIG_REPLY))
                NDBG15(("mpi_pre_config(): Incorrect rep size, 0x%x, "
                    "should be 0x%x", pt->data_size,
                    (int)sizeof (MPI2_CONFIG_REPLY)));
        pt->simple = 1;
}

/*
 * Prepare pt for a SCSI_IO_REQ request.
 */
static void
mpi_pre_scsi_io_req(mptsas_t *mpt, mptsas_pt_request_t *pt)
{
#ifndef __lock_lint
        _NOTE(ARGUNUSED(mpt))
#endif
        pt->sgl_offset = offsetof(MPI2_SCSI_IO_REQUEST, SGL);
        if (pt->request_size != pt->sgl_offset)
                NDBG15(("mpi_pre_config(): Incorrect req size, 0x%x, "
                    "should be 0x%x, dataoutsz 0x%x", pt->request_size,
                    pt->sgl_offset,
                    pt->dataout_size));
        if (pt->data_size != sizeof (MPI2_SCSI_IO_REPLY))
                NDBG15(("mpi_pre_config(): Incorrect rep size, 0x%x, "
                    "should be 0x%x", pt->data_size,
                    (int)sizeof (MPI2_SCSI_IO_REPLY)));
}

/*
 * Prepare the mps_command for a SAS_IO_UNIT_CONTROL request.
 */
static void
mpi_pre_sas_io_unit_control(mptsas_t *mpt, mptsas_pt_request_t *pt)
{
#ifndef __lock_lint
        _NOTE(ARGUNUSED(mpt))
#endif
        pt->sgl_offset = (uint16_t)pt->request_size;
}

/*
 * A set of functions to prepare an mps_command for the various
 * supported requests.
 */
struct mps_func {
        U8              Function;
        char            *Name;
        mps_pre_f       *f_pre;
} mps_func_list[] = {
        { MPI2_FUNCTION_IOC_FACTS, "IOC_FACTS",         mpi_pre_ioc_facts },
        { MPI2_FUNCTION_PORT_FACTS, "PORT_FACTS",       mpi_pre_port_facts },
        { MPI2_FUNCTION_FW_DOWNLOAD, "FW_DOWNLOAD",     mpi_pre_fw_download },
        { MPI2_FUNCTION_FW_UPLOAD, "FW_UPLOAD",         mpi_pre_fw_upload },
        { MPI2_FUNCTION_SATA_PASSTHROUGH, "SATA_PASSTHROUGH",
```



```
            mpi_pre_sata_passthrough },
        { MPI2_FUNCTION_SMP_PASSTHROUGH, "SMP_PASSTHROUGH",
            mpi_pre_smp_passthrough},
        { MPI2_FUNCTION_SCSI_IO_REQUEST, "SCSI_IO_REQUEST",
            mpi_pre_scsi_io_req},
        { MPI2_FUNCTION_CONFIG, "CONFIG",               mpi_pre_config},
        { MPI2_FUNCTION_SAS_IO_UNIT_CONTROL, "SAS_IO_UNIT_CONTROL",
            mpi_pre_sas_io_unit_control },
        { 0xFF, NULL,                           NULL } /* list end */
};

static void
mptsas_prep_sgl_offset(mptsas_t *mpt, mptsas_pt_request_t *pt)
{
        pMPI2RequestHeader_t    hdr;
        struct mps_func         *f;

        hdr = (pMPI2RequestHeader_t)pt->request;

        for (f = mps_func_list; f->f_pre != NULL; f++) {
                if (hdr->Function == f->Function) {
                        f->f_pre(mpt, pt);
                        NDBG15(("mptsas_prep_sgl_offset: Function %s,"
                            " sgl_offset 0x%x", f->Name,
                            pt->sgl_offset));
                        return;
                }
        }
        NDBG15(("mptsas_prep_sgl_offset: Unknown Function 0x%02x,"
            " returning req_size 0x%x for sgl_offset",
            hdr->Function, pt->request_size));
        pt->sgl_offset = (uint16_t)pt->request_size;
}
#endif /* ! codereview */


static int
mptsas_do_passthru(mptsas_t *mpt, uint8_t *request, uint8_t *reply,
    uint8_t *data, uint32_t request_size, uint32_t reply_size,
    uint32_t data_size, uint8_t direction, uint8_t *dataout,
    uint32_t data_size, uint32_t direction, uint8_t *dataout,
    uint32_t dataout_size, short timeout, int mode)
{
        mptsas_pt_request_t             pt;
        mptsas_dma_alloc_state_t        data_dma_state;
        mptsas_dma_alloc_state_t        dataout_dma_state;
        caddr_t                         memp;
        mptsas_cmd_t                    *cmd = NULL;
        struct scsi_pkt                 *pkt;
        uint32_t                        reply_len = 0, sense_len = 0;
        pMPI2RequestHeader_t            request_hdrp;
        pMPI2RequestHeader_t            request_msg;
        pMPI2DefaultReply_t             reply_msg;
        Mpi2SCSIIOReply_t               rep_msg;
        int                             i, status = 0, pt_flags = 0, rv = 0;
        int                             rvalue;
        uint8_t                         function;

        ASSERT(mutex_owned(&mpt->m_mutex));

        reply_msg = (pMPI2DefaultReply_t)(&rep_msg);
        bzero(reply_msg, sizeof (MPI2_DEFAULT_REPLY));
        request_msg = kmem_zalloc(request_size, KM_SLEEP);

        mutex_exit(&mpt->m_mutex);
        /*
```

```
11588          * copy in the request buffer since it could be used by
11589          * another thread when the pt request into waitq
11590          */
11591         if (ddi_copyin(request, request_msg, request_size, mode)) {
11592                 mutex_enter(&mpt->m_mutex);
11593                 status = EFAULT;
11594                 mptsas_log(mpt, CE_WARN, "failed to copy request data");
11595                 goto out;
11596         }
11597         mutex_enter(&mpt->m_mutex);

11599         function = request_msg->Function;
11600         if (function == MPI2_FUNCTION_SCSI_TASK_MGMT) {
11601                 pMpi2SCSITaskManagementRequest_t        task;
11602                 task = (pMpi2SCSITaskManagementRequest_t)request_msg;
11603                 mptsas_setup_bus_reset_delay(mpt);
11604                 rv = mptsas_ioc_task_management(mpt, task->TaskType,
11605                     task->DevHandle, (int)task->LUN[1], reply, reply_size,
11606                     mode);

11608                 if (rv != TRUE) {
11609                         status = EIO;
11610                         mptsas_log(mpt, CE_WARN, "task management failed");
11611                 }
11612                 goto out;
11613         }

11615         if (data_size != 0) {
11616                 data_dma_state.size = data_size;
11617                 if (mptsas_dma_alloc(mpt, &data_dma_state) != DDI_SUCCESS) {
11618                         status = ENOMEM;
11619                         mptsas_log(mpt, CE_WARN, "failed to alloc DMA "
11620                             "resource");
11621                         goto out;
11622                 }
11623                 pt_flags |= MPTSAS_DATA_ALLOCATED;
11624                 if (direction == MPTSAS_PASS_THRU_DIRECTION_WRITE) {
11625                         mutex_exit(&mpt->m_mutex);
11626                         for (i = 0; i < data_size; i++) {
11627                                 if (ddi_copyin(data + i, (uint8_t *)
11628                                     data_dma_state.memp + i, 1, mode)) {
11629                                         mutex_enter(&mpt->m_mutex);
11630                                         status = EFAULT;
11631                                         mptsas_log(mpt, CE_WARN, "failed to "
11632                                             "copy read data");
11633                                         goto out;
11634                                 }
11635                         }
11636                         mutex_enter(&mpt->m_mutex);
11637                 }
11638         }
11639         else
11640                 bzero(&data_dma_state, sizeof (data_dma_state));
11641 #endif /* ! codereview */

11643         if (dataout_size != 0) {
11644                 dataout_dma_state.size = dataout_size;
11645                 if (mptsas_dma_alloc(mpt, &dataout_dma_state) != DDI_SUCCESS) {
11646                         status = ENOMEM;
11647                         mptsas_log(mpt, CE_WARN, "failed to alloc DMA "
11648                             "resource");
11649                         goto out;
11650                 }
11651                 pt_flags |= MPTSAS_DATAOUT_ALLOCATED;
11652                 mutex_exit(&mpt->m_mutex);
11653                 for (i = 0; i < dataout_size; i++) {
```



```
11654                         if (ddi_copyin(dataout + i, (uint8_t *)
11655                             dataout_dma_state.memp + i, 1, mode)) {
11656                                 mutex_enter(&mpt->m_mutex);
11657                                 mptsas_log(mpt, CE_WARN, "failed to copy out"
11658                                     " data");
11659                                 status = EFAULT;
11660                                 goto out;
11661                         }
11662                 }
11663                 mutex_enter(&mpt->m_mutex);
11664         }
11665         else
11666                 bzero(&dataout_dma_state, sizeof (dataout_dma_state));
11667 #endif /* ! codereview */

11669         if ((rvalue = (mptsas_request_from_pool(mpt, &cmd, &pkt))) == -1) {
11670                 status = EAGAIN;
11671                 mptsas_log(mpt, CE_NOTE, "event ack command pool is full");
11672                 goto out;
11673         }
11674         pt_flags |= MPTSAS_REQUEST_POOL_CMD;

11676         bzero((caddr_t)cmd, sizeof (*cmd));
11677         bzero((caddr_t)pkt, scsi_pkt_size());
11678         bzero((caddr_t)&pt, sizeof (pt));

11680         cmd->ioc_cmd_slot = (uint32_t)(rvalue);

11682         pt.request = (uint8_t *)request_msg;
11683         pt.direction = direction;
11684         pt.simple = 0;
11685 #endif /* ! codereview */
11686         pt.request_size = request_size;
11687         pt.data_size = data_size;
11688         pt.dataout_size = dataout_size;
11689         pt.data_cookie = data_dma_state.cookie;
11690         pt.dataout_cookie = dataout_dma_state.cookie;
11691         mptsas_prep_sgl_offset(mpt, &pt);
11692 #endif /* ! codereview */

11694         /*
11695          * Form a blank cmd/pkt to store the acknowledgement message
11696          */
11697         pkt->pkt_cdbp           = (opaque_t)&cmd->cmd_cdb[0];
11698         pkt->pkt_scbp           = (opaque_t)&cmd->cmd_scb;
11699         pkt->pkt_ha_private     = (opaque_t)&pt;
11700         pkt->pkt_flags          = FLAG_HEAD;
11701         pkt->pkt_time           = timeout;
11702         cmd->cmd_pkt            = pkt;
11703         cmd->cmd_flags          = CFLAG_CMDIOC | CFLAG_PASSTHRU;

11705         /*
11706          * Save the command in a slot
11707          */
11708         if (mptsas_save_cmd(mpt, cmd) == TRUE) {
11709                 /*
11710                  * Once passthru command get slot, set cmd_flags
11711                  * CFLAG_PREPARED.
11712                  */
11713                 cmd->cmd_flags |= CFLAG_PREPARED;
11714                 mptsas_start_passthru(mpt, cmd);
11715         } else {
11716                 mptsas_waitq_add(mpt, cmd);
11717         }

11719         while ((cmd->cmd_flags & CFLAG_FINISHED) == 0) {
```

```
11720                   cv_wait(&mpt->m_passthru_cv, &mpt->m_mutex);
11721           }

11723           if (cmd->cmd_flags & CFLAG_PREPARED) {
11724                   memp = mpt->m_req_frame + (mpt->m_req_frame_size *
11725                       cmd->cmd_slot);
11726                   request_hdrp = (pMPI2RequestHeader_t)memp;
11727           }

11729           if (cmd->cmd_flags & CFLAG_TIMEOUT) {
11730                   status = ETIMEDOUT;
11731                   mptsas_log(mpt, CE_WARN, "passthrough command timeout");
11732                   pt_flags |= MPTSAS_CMD_TIMEOUT;
11733                   goto out;
11734           }

11736           if (cmd->cmd_rfm) {
11737                   /*
11738                    * cmd_rfm is zero means the command reply is a CONTEXT
11739                    * reply and no PCI Write to post the free reply SMFA
11740                    * because no reply message frame is used.
11741                    * cmd_rfm is non-zero means the reply is a ADDRESS
11742                    * reply and reply message frame is used.
11743                    */
11744                   pt_flags |= MPTSAS_ADDRESS_REPLY;
11745                   (void) ddi_dma_sync(mpt->m_dma_reply_frame_hdl, 0, 0,
11746                       DDI_DMA_SYNC_FORCPU);
11747                   reply_msg = (pMPI2DefaultReply_t)
11748                       (mpt->m_reply_frame + (cmd->cmd_rfm -
11749                       (mpt->m_reply_frame_dma_addr&0xffffffffu)));
7746                      mpt->m_reply_frame_dma_addr));
11750           }

11752           mptsas_fma_check(mpt, cmd);
11753           if (pkt->pkt_reason == CMD_TRAN_ERR) {
11754                   status = EAGAIN;
11755                   mptsas_log(mpt, CE_WARN, "passthru fma error");
11756                   goto out;
11757           }
11758           if (pkt->pkt_reason == CMD_RESET) {
11759                   status = EAGAIN;
11760                   mptsas_log(mpt, CE_WARN, "ioc reset abort passthru");
11761                   goto out;
11762           }

11764           if (pkt->pkt_reason == CMD_INCOMPLETE) {
11765                   status = EIO;
11766                   mptsas_log(mpt, CE_WARN, "passthrough command incomplete");
11767                   goto out;
11768           }

11770           mutex_exit(&mpt->m_mutex);
11771           if (cmd->cmd_flags & CFLAG_PREPARED) {
11772                   function = request_hdrp->Function;
11773                   if ((function == MPI2_FUNCTION_SCSI_IO_REQUEST) ||
11774                       (function == MPI2_FUNCTION_RAID_SCSI_IO_PASSTHROUGH)) {
11775                           reply_len = sizeof (MPI2_SCSI_IO_REPLY);
11776                           sense_len = reply_size - reply_len;
11777                   } else {
11778                           reply_len = reply_size;
11779                           sense_len = 0;
11780                   }

11782                   for (i = 0; i < reply_len; i++) {
11783                           if (ddi_copyout((uint8_t *)reply_msg + i, reply + i, 1,
11784                               mode)) {
```



```
11785                                   mutex_enter(&mpt->m_mutex);
11786                                   status = EFAULT;
11787                                   mptsas_log(mpt, CE_WARN, "failed to copy out "
11788                                       "reply data");
11789                                   goto out;
11790                           }
11791                   }
11792                   for (i = 0; i < sense_len; i++) {
11793                           if (ddi_copyout((uint8_t *)request_hdrp + 64 + i,
11794                               reply + reply_len + i, 1, mode)) {
11795                                   mutex_enter(&mpt->m_mutex);
11796                                   status = EFAULT;
11797                                   mptsas_log(mpt, CE_WARN, "failed to copy out "
11798                                       "sense data");
11799                                   goto out;
11800                           }
11801                   }
11802           }

11804           if (data_size) {
11805                   if (direction != MPTSAS_PASS_THRU_DIRECTION_WRITE) {
11806                           (void) ddi_dma_sync(data_dma_state.handle, 0, 0,
11807                               DDI_DMA_SYNC_FORCPU);
11808                           for (i = 0; i < data_size; i++) {
11809                                   if (ddi_copyout((uint8_t *)(
11810                                       data_dma_state.memp + i), data + i,  1,
11811                                       mode)) {
11812                                           mutex_enter(&mpt->m_mutex);
11813                                           status = EFAULT;
11814                                           mptsas_log(mpt, CE_WARN, "failed to "
11815                                               "copy out the reply data");
11816                                           goto out;
11817                                   }
11818                           }
11819                   }
11820           }
11821           mutex_enter(&mpt->m_mutex);
11822   out:
11823           /*
11824            * Put the reply frame back on the free queue, increment the free
11825            * index, and write the new index to the free index register.  But only
11826            * if this reply is an ADDRESS reply.
11827            */
11828           if (pt_flags & MPTSAS_ADDRESS_REPLY) {
11829                   ddi_put32(mpt->m_acc_free_queue_hdl,
11830                       &((uint32_t *)(void *)mpt->m_free_queue)[mpt->m_free_index],
11831                       cmd->cmd_rfm);
11832                   (void) ddi_dma_sync(mpt->m_dma_free_queue_hdl, 0, 0,
11833                       DDI_DMA_SYNC_FORDEV);
11834                   if (++mpt->m_free_index == mpt->m_free_queue_depth) {
11835                           mpt->m_free_index = 0;
11836                   }
11837                   ddi_put32(mpt->m_datap, &mpt->m_reg->ReplyFreeHostIndex,
11838                       mpt->m_free_index);
11839           }
11840           if (cmd && (cmd->cmd_flags & CFLAG_PREPARED)) {
11841                   mptsas_remove_cmd(mpt, cmd);
11842                   pt_flags &= (~MPTSAS_REQUEST_POOL_CMD);
11843           }
11844           if (pt_flags & MPTSAS_REQUEST_POOL_CMD)
11845                   mptsas_return_to_pool(mpt, cmd);
11846           if (pt_flags & MPTSAS_DATA_ALLOCATED) {
11847                   if (mptsas_check_dma_handle(data_dma_state.handle) !=
11848                       DDI_SUCCESS) {
11849                           ddi_fm_service_impact(mpt->m_dip,
11850                               DDI_SERVICE_UNAFFECTED);
```

```
11851                           status = EFAULT;
11852                   }
11853                   mptsas_dma_free(&data_dma_state);
11854           }
11855           if (pt_flags & MPTSAS_DATAOUT_ALLOCATED) {
11856                   if (mptsas_check_dma_handle(dataout_dma_state.handle) !=
11857                       DDI_SUCCESS) {
11858                           ddi_fm_service_impact(mpt->m_dip,
11859                               DDI_SERVICE_UNAFFECTED);
11860                           status = EFAULT;
11861                   }
11862                   mptsas_dma_free(&dataout_dma_state);
11863           }
11864           if (pt_flags & MPTSAS_CMD_TIMEOUT) {
11865                   if ((mptsas_restart_ioc(mpt)) == DDI_FAILURE) {
11866                           mptsas_log(mpt, CE_WARN, "mptsas_restart_ioc failed");
11867                   }
11868           }
11869           if (request_msg)
11870                   kmem_free(request_msg, request_size);

11872           return (status);
11873   }

11875   static int
11876   mptsas_pass_thru(mptsas_t *mpt, mptsas_pass_thru_t *data, int mode)
11877   {
11878           /*
11879            * If timeout is 0, set timeout to default of 60 seconds.
11880            */
11881           if (data->Timeout == 0) {
11882                   data->Timeout = MPTSAS_PASS_THRU_TIME_DEFAULT;
11883           }

11885           if (((data->DataSize == 0) &&
11886               (data->DataDirection == MPTSAS_PASS_THRU_DIRECTION_NONE)) ||
11887               ((data->DataSize != 0) &&
11888               ((data->DataDirection == MPTSAS_PASS_THRU_DIRECTION_READ) ||
11889               (data->DataDirection == MPTSAS_PASS_THRU_DIRECTION_WRITE) ||
11890               ((data->DataDirection == MPTSAS_PASS_THRU_DIRECTION_BOTH) &&
11891               (data->DataOutSize != 0))))) {
11892                   if (data->DataDirection == MPTSAS_PASS_THRU_DIRECTION_BOTH) {
11893                           data->DataDirection = MPTSAS_PASS_THRU_DIRECTION_READ;
11894                   } else {
11895                           data->DataOutSize = 0;
11896                   }
11897                   /*
11898                    * Send passthru request messages
11899                    */
11900                   return (mptsas_do_passthru(mpt,
11901                       (uint8_t *)((uintptr_t)data->PtrRequest),
11902                       (uint8_t *)((uintptr_t)data->PtrReply),
11903                       (uint8_t *)((uintptr_t)data->PtrData),
11904                       data->RequestSize, data->ReplySize,
11905                       data->DataSize, (uint8_t)data->DataDirection,
7902                       data->DataSize, data->DataDirection,
11906                       (uint8_t *)((uintptr_t)data->PtrDataOut),
11907                       data->DataOutSize, data->Timeout, mode));
11908           } else {
11909                   return (EINVAL);
11910           }
11911   }
______unchanged_portion_omitted_

11927   static void
11928   mptsas_start_diag(mptsas_t *mpt, mptsas_cmd_t *cmd)
```



```
11929   {
11930           pMpi2DiagBufferPostRequest_t    pDiag_post_msg;
11931           pMpi2DiagReleaseRequest_t       pDiag_release_msg;
11932           struct scsi_pkt                 *pkt = cmd->cmd_pkt;
11933           mptsas_diag_request_t           *diag = pkt->pkt_ha_private;
11934           uint32_t                        i;
11935           uint64_t                        request_desc;
7931           uint32_t                        request_desc_low, i;

11937           ASSERT(mutex_owned(&mpt->m_mutex));

11939           /*
11940            * Form the diag message depending on the post or release function.
11941            */
11942           if (diag->function == MPI2_FUNCTION_DIAG_BUFFER_POST) {
11943                   pDiag_post_msg = (pMpi2DiagBufferPostRequest_t)
11944                       (mpt->m_req_frame + (mpt->m_req_frame_size *
11945                       cmd->cmd_slot));
11946                   bzero(pDiag_post_msg, mpt->m_req_frame_size);
11947                   ddi_put8(mpt->m_acc_req_frame_hdl, &pDiag_post_msg->Function,
11948                       diag->function);
11949                   ddi_put8(mpt->m_acc_req_frame_hdl, &pDiag_post_msg->BufferType,
11950                       diag->pBuffer->buffer_type);
11951                   ddi_put8(mpt->m_acc_req_frame_hdl,
11952                       &pDiag_post_msg->ExtendedType,
11953                       diag->pBuffer->extended_type);
11954                   ddi_put32(mpt->m_acc_req_frame_hdl,
11955                       &pDiag_post_msg->BufferLength,
11956                       diag->pBuffer->buffer_data.size);
11957                   for (i = 0; i < (sizeof (pDiag_post_msg->ProductSpecific) / 4);
11958                       i++) {
11959                           ddi_put32(mpt->m_acc_req_frame_hdl,
11960                               &pDiag_post_msg->ProductSpecific[i],
11961                               diag->pBuffer->product_specific[i]);
11962                   }
11963                   ddi_put32(mpt->m_acc_req_frame_hdl,
11964                       &pDiag_post_msg->BufferAddress.Low,
11965                       (uint32_t)(diag->pBuffer->buffer_data.cookie.dmac_laddress
11966                       & 0xffffffffull));
11967                   ddi_put32(mpt->m_acc_req_frame_hdl,
11968                       &pDiag_post_msg->BufferAddress.High,
11969                       (uint32_t)(diag->pBuffer->buffer_data.cookie.dmac_laddress
11970                       >> 32));
11971           } else {
11972                   pDiag_release_msg = (pMpi2DiagReleaseRequest_t)
11973                       (mpt->m_req_frame + (mpt->m_req_frame_size *
11974                       cmd->cmd_slot));
11975                   bzero(pDiag_release_msg, mpt->m_req_frame_size);
11976                   ddi_put8(mpt->m_acc_req_frame_hdl,
11977                       &pDiag_release_msg->Function, diag->function);
11978                   ddi_put8(mpt->m_acc_req_frame_hdl,
11979                       &pDiag_release_msg->BufferType,
11980                       diag->pBuffer->buffer_type);
11981           }

11983           /*
11984            * Send the message
11985            */
11986           (void) ddi_dma_sync(mpt->m_dma_req_frame_hdl, 0, 0,
11987               DDI_DMA_SYNC_FORDEV);
11988           request_desc = (cmd->cmd_slot << 16) |
7984           request_desc_low = (cmd->cmd_slot << 16) +
11989               MPI2_REQ_DESCRIPT_FLAGS_DEFAULT_TYPE;
11990           cmd->cmd_rfm = NULL;
11991           MPTSAS_START_CMD(mpt, request_desc);
7987           MPTSAS_START_CMD(mpt, request_desc_low, 0);
```

```
11992         if ((mptsas_check_dma_handle(mpt->m_dma_req_frame_hdl) !=
11993             DDI_SUCCESS) ||
11994             (mptsas_check_acc_handle(mpt->m_acc_req_frame_hdl) !=
11995             DDI_SUCCESS)) {
11996                 ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_UNAFFECTED);
11997         }
11998 }

12000 static int
12001 mptsas_post_fw_diag_buffer(mptsas_t *mpt,
12002     mptsas_fw_diagnostic_buffer_t *pBuffer, uint32_t *return_code)
12003 {
12004         mptsas_diag_request_t           diag;
12005         int                             status, slot_num, post_flags = 0;
12006         mptsas_cmd_t                    *cmd = NULL;
12007         struct scsi_pkt                 *pkt;
12008         pMpi2DiagBufferPostReply_t      reply;
12009         uint16_t                        iocstatus;
12010         uint32_t                        iocloginfo, transfer_length;

12012         /*
12013          * If buffer is not enabled, just leave.
12014          */
12015         *return_code = MPTSAS_FW_DIAG_ERROR_POST_FAILED;
12016         if (!pBuffer->enabled) {
12017                 status = DDI_FAILURE;
12018                 goto out;
12019         }

12021         /*
12022          * Clear some flags initially.
12023          */
12024         pBuffer->force_release = FALSE;
12025         pBuffer->valid_data = FALSE;
12026         pBuffer->owned_by_firmware = FALSE;

12028         /*
12029          * Get a cmd buffer from the cmd buffer pool
12030          */
12031         if ((slot_num = (mptsas_request_from_pool(mpt, &cmd, &pkt))) == -1) {
12032                 status = DDI_FAILURE;
12033                 mptsas_log(mpt, CE_NOTE, "command pool is full: Post FW Diag");
12034                 goto out;
12035         }
12036         post_flags |= MPTSAS_REQUEST_POOL_CMD;

12038         bzero((caddr_t)cmd, sizeof (*cmd));
12039         bzero((caddr_t)pkt, scsi_pkt_size());

12041         cmd->ioc_cmd_slot = (uint32_t)(slot_num);

12043         diag.pBuffer = pBuffer;
12044         diag.function = MPI2_FUNCTION_DIAG_BUFFER_POST;

12046         /*
12047          * Form a blank cmd/pkt to store the acknowledgement message
12048          */
12049         pkt->pkt_ha_private     = (opaque_t)&diag;
12050         pkt->pkt_flags          = FLAG_HEAD;
12051         pkt->pkt_time           = 60;
12052         cmd->cmd_pkt            = pkt;
12053         cmd->cmd_flags          = CFLAG_CMDIOC | CFLAG_FW_DIAG;

12055         /*
12056          * Save the command in a slot
12057          */
```



```
12058         if (mptsas_save_cmd(mpt, cmd) == TRUE) {
12059                 /*
12060                  * Once passthru command get slot, set cmd_flags
12061                  * CFLAG_PREPARED.
12062                  */
12063                 cmd->cmd_flags |= CFLAG_PREPARED;
12064                 mptsas_start_diag(mpt, cmd);
12065         } else {
12066                 mptsas_waitq_add(mpt, cmd);
12067         }

12069         while ((cmd->cmd_flags & CFLAG_FINISHED) == 0) {
12070                 cv_wait(&mpt->m_fw_diag_cv, &mpt->m_mutex);
12071         }

12073         if (cmd->cmd_flags & CFLAG_TIMEOUT) {
12074                 status = DDI_FAILURE;
12075                 mptsas_log(mpt, CE_WARN, "Post FW Diag command timeout");
12076                 goto out;
12077         }

12079         /*
12080          * cmd_rfm points to the reply message if a reply was given.  Check the
12081          * IOCStatus to make sure everything went OK with the FW diag request
12082          * and set buffer flags.
12083          */
12084         if (cmd->cmd_rfm) {
12085                 post_flags |= MPTSAS_ADDRESS_REPLY;
12086                 (void) ddi_dma_sync(mpt->m_dma_reply_frame_hdl, 0, 0,
12087                     DDI_DMA_SYNC_FORCPU);
12088                 reply = (pMpi2DiagBufferPostReply_t)(mpt->m_reply_frame +
12089                     (cmd->cmd_rfm -
12090                     (mpt->m_reply_frame_dma_addr&0xffffffffu)));
8085                    (cmd->cmd_rfm - mpt->m_reply_frame_dma_addr));

12092                 /*
12093                  * Get the reply message data
12094                  */
12095                 iocstatus = ddi_get16(mpt->m_acc_reply_frame_hdl,
12096                     &reply->IOCStatus);
12097                 iocloginfo = ddi_get32(mpt->m_acc_reply_frame_hdl,
12098                     &reply->IOCLogInfo);
12099                 transfer_length = ddi_get32(mpt->m_acc_reply_frame_hdl,
12100                     &reply->TransferLength);

12102                 /*
12103                  * If post failed quit.
12104                  */
12105                 if (iocstatus != MPI2_IOCSTATUS_SUCCESS) {
12106                         status = DDI_FAILURE;
12107                         NDBG13(("post FW Diag Buffer failed: IOCStatus=0x%x, "
12108                             "IOCLogInfo=0x%x, TransferLength=0x%x", iocstatus,
12109                             iocloginfo, transfer_length));
12110                         goto out;
12111                 }

12113                 /*
12114                  * Post was successful.
12115                  */
12116                 pBuffer->valid_data = TRUE;
12117                 pBuffer->owned_by_firmware = TRUE;
12118                 *return_code = MPTSAS_FW_DIAG_ERROR_SUCCESS;
12119                 status = DDI_SUCCESS;
12120         }

12122 out:
```

```
12123         /*
12124          * Put the reply frame back on the free queue, increment the free
12125          * index, and write the new index to the free index register.  But only
12126          * if this reply is an ADDRESS reply.
12127          */
12128         if (post_flags & MPTSAS_ADDRESS_REPLY) {
12129                 ddi_put32(mpt->m_acc_free_queue_hdl,
12130                     &((uint32_t *)(void *)mpt->m_free_queue)[mpt->m_free_index],
12131                     cmd->cmd_rfm);
12132                 (void) ddi_dma_sync(mpt->m_dma_free_queue_hdl, 0, 0,
12133                     DDI_DMA_SYNC_FORDEV);
12134                 if (++mpt->m_free_index == mpt->m_free_queue_depth) {
12135                         mpt->m_free_index = 0;
12136                 }
12137                 ddi_put32(mpt->m_datap, &mpt->m_reg->ReplyFreeHostIndex,
12138                     mpt->m_free_index);
12139         }
12140         if (cmd && (cmd->cmd_flags & CFLAG_PREPARED)) {
12141                 mptsas_remove_cmd(mpt, cmd);
12142                 post_flags &= (~MPTSAS_REQUEST_POOL_CMD);
12143         }
12144         if (post_flags & MPTSAS_REQUEST_POOL_CMD) {
12145                 mptsas_return_to_pool(mpt, cmd);
12146         }

12148         return (status);
12149 }

12151 static int
12152 mptsas_release_fw_diag_buffer(mptsas_t *mpt,
12153     mptsas_fw_diagnostic_buffer_t *pBuffer, uint32_t *return_code,
12154     uint32_t diag_type)
12155 {
12156         mptsas_diag_request_t   diag;
12157         int                     status, slot_num, rel_flags = 0;
12158         mptsas_cmd_t            *cmd = NULL;
12159         struct scsi_pkt         *pkt;
12160         pMpi2DiagReleaseReply_t reply;
12161         uint16_t                iocstatus;
12162         uint32_t                iocloginfo;

12164         /*
12165          * If buffer is not enabled, just leave.
12166          */
12167         *return_code = MPTSAS_FW_DIAG_ERROR_RELEASE_FAILED;
12168         if (!pBuffer->enabled) {
12169                 mptsas_log(mpt, CE_NOTE, "This buffer type is not supported "
12170                     "by the IOC");
12171                 status = DDI_FAILURE;
12172                 goto out;
12173         }

12175         /*
12176          * Clear some flags initially.
12177          */
12178         pBuffer->force_release = FALSE;
12179         pBuffer->valid_data = FALSE;
12180         pBuffer->owned_by_firmware = FALSE;

12182         /*
12183          * Get a cmd buffer from the cmd buffer pool
12184          */
12185         if ((slot_num = (mptsas_request_from_pool(mpt, &cmd, &pkt))) == -1) {
12186                 status = DDI_FAILURE;
12187                 mptsas_log(mpt, CE_NOTE, "command pool is full: Release FW "
12188                     "Diag");
```



```
12189                 goto out;
12190         }
12191         rel_flags |= MPTSAS_REQUEST_POOL_CMD;

12193         bzero((caddr_t)cmd, sizeof (*cmd));
12194         bzero((caddr_t)pkt, scsi_pkt_size());

12196         cmd->ioc_cmd_slot = (uint32_t)(slot_num);

12198         diag.pBuffer = pBuffer;
12199         diag.function = MPI2_FUNCTION_DIAG_RELEASE;

12201         /*
12202          * Form a blank cmd/pkt to store the acknowledgement message
12203          */
12204         pkt->pkt_ha_private     = (opaque_t)&diag;
12205         pkt->pkt_flags          = FLAG_HEAD;
12206         pkt->pkt_time           = 60;
12207         cmd->cmd_pkt            = pkt;
12208         cmd->cmd_flags          = CFLAG_CMDIOC | CFLAG_FW_DIAG;

12210         /*
12211          * Save the command in a slot
12212          */
12213         if (mptsas_save_cmd(mpt, cmd) == TRUE) {
12214                 /*
12215                  * Once passthru command get slot, set cmd_flags
12216                  * CFLAG_PREPARED.
12217                  */
12218                 cmd->cmd_flags |= CFLAG_PREPARED;
12219                 mptsas_start_diag(mpt, cmd);
12220         } else {
12221                 mptsas_waitq_add(mpt, cmd);
12222         }

12224         while ((cmd->cmd_flags & CFLAG_FINISHED) == 0) {
12225                 cv_wait(&mpt->m_fw_diag_cv, &mpt->m_mutex);
12226         }

12228         if (cmd->cmd_flags & CFLAG_TIMEOUT) {
12229                 status = DDI_FAILURE;
12230                 mptsas_log(mpt, CE_WARN, "Release FW Diag command timeout");
12231                 goto out;
12232         }

12234         /*
12235          * cmd_rfm points to the reply message if a reply was given.  Check the
12236          * IOCStatus to make sure everything went OK with the FW diag request
12237          * and set buffer flags.
12238          */
12239         if (cmd->cmd_rfm) {
12240                 rel_flags |= MPTSAS_ADDRESS_REPLY;
12241                 (void) ddi_dma_sync(mpt->m_dma_reply_frame_hdl, 0, 0,
12242                     DDI_DMA_SYNC_FORCPU);
12243                 reply = (pMpi2DiagReleaseReply_t)(mpt->m_reply_frame +
12244                     (cmd->cmd_rfm -
12245                     (mpt->m_reply_frame_dma_addr&0xffffffffu)));
8239                    (cmd->cmd_rfm - mpt->m_reply_frame_dma_addr));

12247                 /*
12248                  * Get the reply message data
12249                  */
12250                 iocstatus = ddi_get16(mpt->m_acc_reply_frame_hdl,
12251                     &reply->IOCStatus);
12252                 iocloginfo = ddi_get32(mpt->m_acc_reply_frame_hdl,
12253                     &reply->IOCLogInfo);
```

```
12255                 /*
12256                  * If release failed quit.
12257                  */
12258                 if ((iocstatus != MPI2_IOCSTATUS_SUCCESS) ||
12259                     pBuffer->owned_by_firmware) {
12260                         status = DDI_FAILURE;
12261                         NDBG13(("release FW Diag Buffer failed: "
12262                             "IOCStatus=0x%x, IOCLogInfo=0x%x", iocstatus,
12263                             iocloginfo));
12264                         goto out;
12265                 }

12267                 /*
12268                  * Release was successful.
12269                  */
12270                 *return_code = MPTSAS_FW_DIAG_ERROR_SUCCESS;
12271                 status = DDI_SUCCESS;

12273                 /*
12274                  * If this was for an UNREGISTER diag type command, clear the
12275                  * unique ID.
12276                  */
12277                 if (diag_type == MPTSAS_FW_DIAG_TYPE_UNREGISTER) {
12278                         pBuffer->unique_id = MPTSAS_FW_DIAG_INVALID_UID;
12279                 }
12280         }

12282 out:
12283         /*
12284          * Put the reply frame back on the free queue, increment the free
12285          * index, and write the new index to the free index register.  But only
12286          * if this reply is an ADDRESS reply.
12287          */
12288         if (rel_flags & MPTSAS_ADDRESS_REPLY) {
12289                 ddi_put32(mpt->m_acc_free_queue_hdl,
12290                     &((uint32_t *)(void *)mpt->m_free_queue)[mpt->m_free_index],
12291                     cmd->cmd_rfm);
12292                 (void) ddi_dma_sync(mpt->m_dma_free_queue_hdl, 0, 0,
12293                     DDI_DMA_SYNC_FORDEV);
12294                 if (++mpt->m_free_index == mpt->m_free_queue_depth) {
12295                         mpt->m_free_index = 0;
12296                 }
12297                 ddi_put32(mpt->m_datap, &mpt->m_reg->ReplyFreeHostIndex,
12298                     mpt->m_free_index);
12299         }
12300         if (cmd && (cmd->cmd_flags & CFLAG_PREPARED)) {
12301                 mptsas_remove_cmd(mpt, cmd);
12302                 rel_flags &= (~MPTSAS_REQUEST_POOL_CMD);
12303         }
12304         if (rel_flags & MPTSAS_REQUEST_POOL_CMD) {
12305                 mptsas_return_to_pool(mpt, cmd);
12306         }

12308         return (status);
12309 }
```

unchanged_portion_omitted_

```
12987 static void
12988 mptsas_read_adapter_data(mptsas_t *mpt, mptsas_adapter_data_t *adapter_data)
12989 {
12990         char    *driver_verstr = MPTSAS_MOD_STRING;

12992         mptsas_lookup_pci_data(mpt, adapter_data);
12993         adapter_data->AdapterType = MPTIOCTL_ADAPTER_TYPE_SAS3;
8987         adapter_data->AdapterType = MPTIOCTL_ADAPTER_TYPE_SAS2;
```



```
12994         adapter_data->PCIDeviceHwId = (uint32_t)mpt->m_devid;
12995         adapter_data->PCIDeviceHwRev = (uint32_t)mpt->m_revid;
12996         adapter_data->SubSystemId = (uint32_t)mpt->m_ssid;
12997         adapter_data->SubsystemVendorId = (uint32_t)mpt->m_svid;
12998         (void) strcpy((char *)&adapter_data->DriverVersion[0], driver_verstr);
12999         adapter_data->BiosVersion = 0;
13000         (void) mptsas_get_bios_page3(mpt, &adapter_data->BiosVersion);
13001 }
```

unchanged_portion_omitted_

```
13243 static int
13244 mptsas_ioctl(dev_t dev, int cmd, intptr_t data, int mode, cred_t *credp,
13245     int *rval)
13246 {
13247         int                     status = 0;
13248         mptsas_t                *mpt;
13249         mptsas_update_flash_t   flashdata;
13250         mptsas_pass_thru_t      passthru_data;
13251         mptsas_adapter_data_t   adapter_data;
13252         mptsas_pci_info_t       pci_info;
13253         int                     copylen;

13255         int                     iport_flag = 0;
13256         dev_info_t              *dip = NULL;
13257         mptsas_phymask_t        phymask = 0;
13258         struct devctl_iocdata   *dcp = NULL;
13259         char                    *addr = NULL;
13260         mptsas_target_t         *ptgt = NULL;

13262         *rval = MPTIOCTL_STATUS_GOOD;
13263         if (secpolicy_sys_config(credp, B_FALSE) != 0) {
13264                 return (EPERM);
13265         }

13267         mpt = ddi_get_soft_state(mptsas3_state, MINOR2INST(getminor(dev)));
9261         mpt = ddi_get_soft_state(mptsas_state, MINOR2INST(getminor(dev)));
13268         if (mpt == NULL) {
13269                 /*
13270                  * Called from iport node, get the states
13271                  */
13272                 iport_flag = 1;
13273                 dip = mptsas_get_dip_from_dev(dev, &phymask);
13274                 if (dip == NULL) {
13275                         return (ENXIO);
13276                 }
13277                 mpt = DIP2MPT(dip);
13278         }
13279         /* Make sure power level is D0 before accessing registers */
13280         mutex_enter(&mpt->m_mutex);
13281         if (mpt->m_options & MPTSAS_OPT_PM) {
13282                 (void) pm_busy_component(mpt->m_dip, 0);
13283                 if (mpt->m_power_level != PM_LEVEL_D0) {
13284                         mutex_exit(&mpt->m_mutex);
13285                         if (pm_raise_power(mpt->m_dip, 0, PM_LEVEL_D0) !=
13286                             DDI_SUCCESS) {
13287                                 mptsas_log(mpt, CE_WARN,
13288                                     "mptsas3%d: mptsas_ioctl: Raise power "
9282                                     "mptsas%d: mptsas_ioctl: Raise power "
13289                                     "request failed.", mpt->m_instance);
13290                                 (void) pm_idle_component(mpt->m_dip, 0);
13291                                 return (ENXIO);
13292                         }
13293                 } else {
13294                         mutex_exit(&mpt->m_mutex);
13295                 }
13296         } else {
```

```
13297                   mutex_exit(&mpt->m_mutex);
13298           }

13300           if (iport_flag) {
13301                   status = scsi_hba_ioctl(dev, cmd, data, mode, credp, rval);
13302                   if (status != 0) {
13303                           goto out;
13304                   }
13305                   /*
13306                    * The following code control the OK2RM LED, it doesn't affect
13307                    * the ioctl return status.
13308                    */
13309                   if ((cmd == DEVCTL_DEVICE_ONLINE) ||
13310                       (cmd == DEVCTL_DEVICE_OFFLINE)) {
13311                           if (ndi_dc_allochdl((void *)data, &dcp) !=
13312                               NDI_SUCCESS) {
13313                                   goto out;
13314                           }
13315                           addr = ndi_dc_getaddr(dcp);
13316                           ptgt = mptsas_addr_to_ptgt(mpt, addr, phymask);
13317                           if (ptgt == NULL) {
13318                                   NDBG14(("mptsas_ioctl led control: tgt %s not "
13319                                       "found", addr));
13320                                   ndi_dc_freehdl(dcp);
13321                                   goto out;
13322                           }
13323                           mutex_enter(&mpt->m_mutex);
13324                           if (cmd == DEVCTL_DEVICE_ONLINE) {
13325                                   ptgt->m_tgt_unconfigured = 0;
13326                           } else if (cmd == DEVCTL_DEVICE_OFFLINE) {
13327                                   ptgt->m_tgt_unconfigured = 1;
13328                           }
13329                           if (cmd == DEVCTL_DEVICE_OFFLINE) {
13330                                   ptgt->m_led_status |=
13331                                       (1 << (MPTSAS_LEDCTL_LED_OK2RM - 1));
13332                           } else {
13333                                   ptgt->m_led_status &=
13334                                       ~(1 << (MPTSAS_LEDCTL_LED_OK2RM - 1));
13335                           }
13336                           (void) mptsas_flush_led_status(mpt, ptgt);
13337                           mutex_exit(&mpt->m_mutex);
13338                           ndi_dc_freehdl(dcp);
13339                   }
13340                   goto out;
13341           }
13342           switch (cmd) {
13343                   case MPTIOCTL_GET_DISK_INFO:
13344                           status = get_disk_info(mpt, data, mode);
13345                           break;
13346                   case MPTIOCTL_LED_CONTROL:
13347                           status = led_control(mpt, data, mode);
13348                           break;
13349                   case MPTIOCTL_UPDATE_FLASH:
13350                           if (ddi_copyin((void *)data, &flashdata,
13351                                   sizeof (struct mptsas_update_flash), mode)) {
13352                                   status = EFAULT;
13353                                   break;
13354                           }

13356                           mutex_enter(&mpt->m_mutex);
13357                           if (mptsas_update_flash(mpt,
13358                               (caddr_t)(long)flashdata.PtrBuffer,
13359                               flashdata.ImageSize, flashdata.ImageType, mode)) {
13360                                   status = EFAULT;
13361                           }
```



```
13363                           /*
13364                            * Reset the chip to start using the new
13365                            * firmware.  Reset if failed also.
13366                            */
13367                           mpt->m_softstate &= ~MPTSAS_SS_MSG_UNIT_RESET;
13368                           if (mptsas_restart_ioc(mpt) == DDI_FAILURE) {
13369                                   status = EFAULT;
13370                           }
13371                           mutex_exit(&mpt->m_mutex);
13372                           break;
13373                   case MPTIOCTL_PASS_THRU:
13374                           /*
13375                            * The user has requested to pass through a command to
13376                            * be executed by the MPT firmware.  Call our routine
13377                            * which does this.  Only allow one passthru IOCTL at
13378                            * one time. Other threads will block on
13379                            * m_passthru_mutex, which is of adaptive variant.
13380                            */
13381                           if (ddi_copyin((void *)data, &passthru_data,
13382                               sizeof (mptsas_pass_thru_t), mode)) {
13383                                   status = EFAULT;
13384                                   break;
13385                           }
13386                           mutex_enter(&mpt->m_passthru_mutex);
13387                           mutex_enter(&mpt->m_mutex);
13388                           status = mptsas_pass_thru(mpt, &passthru_data, mode);
13389                           mutex_exit(&mpt->m_mutex);
13390                           mutex_exit(&mpt->m_passthru_mutex);

13392                           break;
13393                   case MPTIOCTL_GET_ADAPTER_DATA:
13394                           /*
13395                            * The user has requested to read adapter data.  Call
13396                            * our routine which does this.
13397                            */
13398                           bzero(&adapter_data, sizeof (mptsas_adapter_data_t));
13399                           if (ddi_copyin((void *)data, (void *)&adapter_data,
13400                               sizeof (mptsas_adapter_data_t), mode)) {
13401                                   status = EFAULT;
13402                                   break;
13403                           }
13404                           if (adapter_data.StructureLength >=
13405                               sizeof (mptsas_adapter_data_t)) {
13406                                   adapter_data.StructureLength = (uint32_t)
13407                                       sizeof (mptsas_adapter_data_t);
13408                                   copylen = sizeof (mptsas_adapter_data_t);
13409                                   mutex_enter(&mpt->m_mutex);
13410                                   mptsas_read_adapter_data(mpt, &adapter_data);
13411                                   mutex_exit(&mpt->m_mutex);
13412                           } else {
13413                                   adapter_data.StructureLength = (uint32_t)
13414                                       sizeof (mptsas_adapter_data_t);
13415                                   copylen = sizeof (adapter_data.StructureLength);
13416                                   *rval = MPTIOCTL_STATUS_LEN_TOO_SHORT;
13417                           }
13418                           if (ddi_copyout((void *)(&adapter_data), (void *)data,
13419                               copylen, mode) != 0) {
13420                                   status = EFAULT;
13421                           }
13422                           break;
13423                   case MPTIOCTL_GET_PCI_INFO:
13424                           /*
13425                            * The user has requested to read pci info.  Call
13426                            * our routine which does this.
13427                            */
13428                           bzero(&pci_info, sizeof (mptsas_pci_info_t));
```

```
13429                           mutex_enter(&mpt->m_mutex);
13430                           mptsas_read_pci_info(mpt, &pci_info);
13431                           mutex_exit(&mpt->m_mutex);
13432                           if (ddi_copyout((void *)(&pci_info), (void *)data,
13433                               sizeof (mptsas_pci_info_t), mode) != 0) {
13434                                   status = EFAULT;
13435                           }
13436                           break;
13437                   case MPTIOCTL_RESET_ADAPTER:
13438                           mutex_enter(&mpt->m_mutex);
13439                           mpt->m_softstate &= ~MPTSAS_SS_MSG_UNIT_RESET;
13440                           if ((mptsas_restart_ioc(mpt)) == DDI_FAILURE) {
13441                                   mptsas_log(mpt, CE_WARN, "reset adapter IOCTL "
13442                                       "failed");
13443                                   status = EFAULT;
13444                           }
13445                           mutex_exit(&mpt->m_mutex);
13446                           break;
13447                   case MPTIOCTL_DIAG_ACTION:
13448                           /*
13449                            * The user has done a diag buffer action.  Call our
13450                            * routine which does this.  Only allow one diag action
13451                            * at one time.
13452                            */
13453                           mutex_enter(&mpt->m_mutex);
13454                           if (mpt->m_diag_action_in_progress) {
13455                                   mutex_exit(&mpt->m_mutex);
13456                                   return (EBUSY);
13457                           }
13458                           mpt->m_diag_action_in_progress = 1;
13459                           status = mptsas_diag_action(mpt,
13460                               (mptsas_diag_action_t *)data, mode);
13461                           mpt->m_diag_action_in_progress = 0;
13462                           mutex_exit(&mpt->m_mutex);
13463                           break;
13464                   case MPTIOCTL_EVENT_QUERY:
13465                           /*
13466                            * The user has done an event query. Call our routine
13467                            * which does this.
13468                            */
13469                           status = mptsas_event_query(mpt,
13470                               (mptsas_event_query_t *)data, mode, rval);
13471                           break;
13472                   case MPTIOCTL_EVENT_ENABLE:
13473                           /*
13474                            * The user has done an event enable. Call our routine
13475                            * which does this.
13476                            */
13477                           status = mptsas_event_enable(mpt,
13478                               (mptsas_event_enable_t *)data, mode, rval);
13479                           break;
13480                   case MPTIOCTL_EVENT_REPORT:
13481                           /*
13482                            * The user has done an event report. Call our routine
13483                            * which does this.
13484                            */
13485                           status = mptsas_event_report(mpt,
13486                               (mptsas_event_report_t *)data, mode, rval);
13487                           break;
13488                   case MPTIOCTL_REG_ACCESS:
13489                           /*
13490                            * The user has requested register access.  Call our
13491                            * routine which does this.
13492                            */
13493                           status = mptsas_reg_access(mpt,
13494                               (mptsas_reg_access_t *)data, mode);
```



```
13495                           break;
13496                   default:
13497                           status = scsi_hba_ioctl(dev, cmd, data, mode, credp,
13498                               rval);
13499                           break;
13500           }

13502   out:
13503           return (status);
13504   }

13506   int
13507   mptsas_restart_ioc(mptsas_t *mpt)
13508   {
13509           int             rval = DDI_SUCCESS;
13510           mptsas_target_t *ptgt = NULL;

13512           ASSERT(mutex_owned(&mpt->m_mutex));

13514           /*
13515            * Set a flag telling I/O path that we're processing a reset.  This is
13516            * needed because after the reset is complete, the hash table still
13517            * needs to be rebuilt.  If I/Os are started before the hash table is
13518            * rebuilt, I/O errors will occur.  This flag allows I/Os to be marked
13519            * so that they can be retried.
13520            */
13521           mpt->m_in_reset = TRUE;

13523           /*
13524            * Set all throttles to HOLD
13525            */
13526           for (ptgt = refhash_first(mpt->m_targets); ptgt != NULL;
13527               ptgt = refhash_next(mpt->m_targets, ptgt)) {
13528                   mptsas_set_throttle_mtx(mpt, ptgt, HOLD_THROTTLE);
9522                   mptsas_set_throttle(mpt, ptgt, HOLD_THROTTLE);
13529           }

13531           /*
13532            * Disable interrupts
13533            */
13534           MPTSAS_DISABLE_INTR(mpt);

13536           /*
13537            * Abort all commands: outstanding commands, commands in waitq and
13538            * tx_waitq.
13539            */
13540           mptsas_flush_hba(mpt);

13542           /*
13543            * Reinitialize the chip.
13544            */
13545           if (mptsas_init_chip(mpt, FALSE) == DDI_FAILURE) {
13546                   rval = DDI_FAILURE;
13547           }

13549           /*
13550            * Enable interrupts again
13551            */
13552           MPTSAS_ENABLE_INTR(mpt);

13554           /*
13555            * If mptsas_init_chip was successful, update the driver data.
13556            */
13557           if (rval == DDI_SUCCESS) {
13558                   mptsas_update_driver_data(mpt);
13559           }
```

```
13561         /*
13562          * Reset the throttles
13563          */
13564         for (ptgt = refhash_first(mpt->m_targets); ptgt != NULL;
13565             ptgt = refhash_next(mpt->m_targets, ptgt)) {
13566                 mptsas_set_throttle_mtx(mpt, ptgt, MAX_THROTTLE);
9560                mptsas_set_throttle(mpt, ptgt, MAX_THROTTLE);
13567         }

13569         mptsas_doneq_empty(mpt);
13570         mptsas_restart_hba(mpt);

13572         if (rval != DDI_SUCCESS) {
13573                 mptsas_fm_ereport(mpt, DDI_FM_DEVICE_NO_RESPONSE);
13574                 ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_LOST);
13575         }

13577         /*
13578          * Clear the reset flag so that I/Os can continue.
13579          */
13580         mpt->m_in_reset = FALSE;

13582         return (rval);
13583 }

13585 static int
13586 mptsas_init_chip(mptsas_t *mpt, int first_time)
13587 {
13588         ddi_dma_cookie_t        cookie;
13589         mptsas_reply_pqueue_t   *rpqp;
13590         uint32_t                i, j;
9583         uint32_t                i;
13591         int                     rval;

13593         /*
13594          * Check to see if the firmware image is valid
13595          */
13596         if (ddi_get32(mpt->m_datap, &mpt->m_reg->HostDiagnostic) &
13597             MPI2_DIAG_FLASH_BAD_SIG) {
13598                 mptsas_log(mpt, CE_WARN, "mptsas bad flash signature!");
13599                 goto fail;
13600         }

13602         /*
13603          * Reset the chip
13604          */
13605         rval = mptsas_ioc_reset(mpt, first_time);
13606         if (rval == MPTSAS_RESET_FAIL) {
13607                 mptsas_log(mpt, CE_WARN, "hard reset failed!");
13608                 goto fail;
13609         }

13611         if ((rval == MPTSAS_SUCCESS_MUR) && (!first_time)) {
13612                 goto mur;
13613         }
13614         /*
13615          * Setup configuration space
13616          */
13617         if (mptsas_config_space_init(mpt) == FALSE) {
13618                 mptsas_log(mpt, CE_WARN, "mptsas_config_space_init "
13619                     "failed!");
13620                 goto fail;
13621         }

13623         /*
```



```
13624          * IOC facts can change after a diag reset so all buffers that are
13625          * based on these numbers must be de-allocated and re-allocated.  Get
13626          * new IOC facts each time chip is initialized.
13627          */
13628         if (mptsas_ioc_get_facts(mpt) == DDI_FAILURE) {
13629                 mptsas_log(mpt, CE_WARN, "mptsas_ioc_get_facts failed");
13630                 goto fail;
13631         }

13633         /*
13634          * Now we know chip MSIX capabilitites and it's not been done
13635          * previously register interrupts accordingly. Need to know this
13636          * information before allocating the reply frames below.
13637          */
13638         if (mpt->m_intr_cnt == 0) {
13639                 if (mptsas_register_intrs(mpt) == FALSE)
13640                         goto fail;
13641         }

13643 #endif /* ! codereview */
13644         mpt->m_targets = refhash_create(MPTSAS_TARGET_BUCKET_COUNT,
13645             mptsas_target_addr_hash, mptsas_target_addr_cmp,
13646             mptsas_target_free, sizeof (mptsas_target_t),
13647             offsetof(mptsas_target_t, m_link),
13648             offsetof(mptsas_target_t, m_addr), KM_SLEEP);

13650         if (mptsas_alloc_active_slots(mpt, KM_SLEEP)) {
13651                 goto fail;
13652         }
13653         /*
13654          * Allocate request message frames, reply free queue, reply descriptor
13655          * post queue, and reply message frames using latest IOC facts.
13656          */
13657         if (mptsas_alloc_request_frames(mpt) == DDI_FAILURE) {
13658                 mptsas_log(mpt, CE_WARN, "mptsas_alloc_request_frames failed");
13659                 goto fail;
13660         }
13661         if (mptsas_alloc_sense_bufs(mpt) == DDI_FAILURE) {
13662                 mptsas_log(mpt, CE_WARN, "mptsas_alloc_sense_bufs failed");
13663                 goto fail;
13664         }
13665 #endif /* ! codereview */
13666         if (mptsas_alloc_free_queue(mpt) == DDI_FAILURE) {
13667                 mptsas_log(mpt, CE_WARN, "mptsas_alloc_free_queue failed!");
13668                 goto fail;
13669         }
13670         if (mptsas_alloc_post_queue(mpt) == DDI_FAILURE) {
13671                 mptsas_log(mpt, CE_WARN, "mptsas_alloc_post_queue failed!");
13672                 goto fail;
13673         }
13674         if (mptsas_alloc_reply_frames(mpt) == DDI_FAILURE) {
13675                 mptsas_log(mpt, CE_WARN, "mptsas_alloc_reply_frames failed!");
13676                 goto fail;
13677         }

13679 mur:
13680         /*
13681          * Re-Initialize ioc to operational state
13682          */
13683         if (mptsas_ioc_init(mpt) == DDI_FAILURE) {
13684                 mptsas_log(mpt, CE_WARN, "mptsas_ioc_init failed");
13685                 goto fail;
13686         }

13688         mptsas_alloc_reply_args(mpt);
```

```
13690          /*
9626           * Initialize reply post index.  Reply free index is initialized after
9627           * the next loop.
9628           */
9629          mpt->m_post_index = 0;

9631          /*
13691           * Initialize the Reply Free Queue with the physical addresses of our
13692           * reply frames.
13693           */
13694          cookie.dmac_address = mpt->m_reply_frame_dma_addr&0xffffffffu;
9635          cookie.dmac_address = mpt->m_reply_frame_dma_addr;
13695          for (i = 0; i < mpt->m_max_replies; i++) {
13696                  ddi_put32(mpt->m_acc_free_queue_hdl,
13697                      &((uint32_t *)(void *)mpt->m_free_queue)[i],
13698                      cookie.dmac_address);
13699                  cookie.dmac_address += mpt->m_reply_frame_size;
13700          }
13701          (void) ddi_dma_sync(mpt->m_dma_free_queue_hdl, 0, 0,
13702              DDI_DMA_SYNC_FORDEV);

13704          /*
13705           * Initialize the reply free index to one past the last frame on the
13706           * queue.  This will signify that the queue is empty to start with.
13707           */
13708          mpt->m_free_index = i;
13709          ddi_put32(mpt->m_datap, &mpt->m_reg->ReplyFreeHostIndex, i);

13711          /*
13712           * Initialize the reply post queue to 0xFFFFFFFF,0xFFFFFFFF's
13713           * and the indexes to 0.
9653           * Initialize the reply post queue to 0xFFFFFFFF,0xFFFFFFFF's.
13714           */
13715          rpqp = mpt->m_rep_post_queues;
13716          for (j = 0; j < mpt->m_post_reply_qcount; j++) {
13717  #endif /* ! codereview */
13718                  for (i = 0; i < mpt->m_post_queue_depth; i++) {
13719                          ddi_put64(mpt->m_acc_post_queue_hdl,
13720                              &((uint64_t *)(void *)rpqp->rpq_queue)[i],
9655                         &((uint64_t *)(void *)mpt->m_post_queue)[i],
13721                                  0xFFFFFFFFFFFFFFFF);
13722                  }
13723                  rpqp->rpq_index = 0;
13724                  rpqp++;
13725          }
13726  #endif /* ! codereview */
13727          (void) ddi_dma_sync(mpt->m_dma_post_queue_hdl, 0, 0,
13728              DDI_DMA_SYNC_FORDEV);

13730          /*
13731           * Initialise all the reply post queue indexes.
13732           */
13733          for (j = 0; j < mpt->m_post_reply_qcount; j++) {
13734                  ddi_put32(mpt->m_datap, &mpt->m_reg->ReplyPostHostIndex,
13735                      j << MPI2_RPHI_MSIX_INDEX_SHIFT);
13736          }

13738          /*
13739  #endif /* ! codereview */
13740           * Enable ports
13741           */
13742          if (mptsas_ioc_enable_port(mpt) == DDI_FAILURE) {
13743                  mptsas_log(mpt, CE_WARN, "mptsas_ioc_enable_port failed");
13744                  goto fail;
13745          }
```



```
13747          /*
13748           * enable events
13749           */
13750          if (mptsas_ioc_enable_event_notification(mpt)) {
13751                  mptsas_log(mpt, CE_WARN,
13752                      "mptsas_ioc_enable_event_notification failed");
13753  #endif /* ! codereview */
13754                  goto fail;
13755          }

13757          /*
13758           * We need checks in attach and these.
13759           * chip_init is called in mult. places
13760           */

13762          if ((mptsas_check_dma_handle(mpt->m_dma_req_frame_hdl) !=
13763              DDI_SUCCESS) ||
13764              (mptsas_check_dma_handle(mpt->m_dma_req_sense_hdl) !=
13765              DDI_SUCCESS) ||
13766  #endif /* ! codereview */
13767              (mptsas_check_dma_handle(mpt->m_dma_reply_frame_hdl) !=
13768              DDI_SUCCESS) ||
13769              (mptsas_check_dma_handle(mpt->m_dma_free_queue_hdl) !=
13770              DDI_SUCCESS) ||
13771              (mptsas_check_dma_handle(mpt->m_dma_post_queue_hdl) !=
13772              DDI_SUCCESS) ||
13773              (mptsas_check_dma_handle(mpt->m_hshk_dma_hdl) !=
13774              DDI_SUCCESS)) {
13775                  ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_UNAFFECTED);
13776                  goto fail;
13777          }

13779          /* Check all acc handles */
13780          if ((mptsas_check_acc_handle(mpt->m_datap) != DDI_SUCCESS) ||
13781              (mptsas_check_acc_handle(mpt->m_acc_req_frame_hdl) !=
13782              DDI_SUCCESS) ||
13783              (mptsas_check_acc_handle(mpt->m_acc_req_sense_hdl) !=
13784              DDI_SUCCESS) ||
13785  #endif /* ! codereview */
13786              (mptsas_check_acc_handle(mpt->m_acc_reply_frame_hdl) !=
13787              DDI_SUCCESS) ||
13788              (mptsas_check_acc_handle(mpt->m_acc_free_queue_hdl) !=
13789              DDI_SUCCESS) ||
13790              (mptsas_check_acc_handle(mpt->m_acc_post_queue_hdl) !=
13791              DDI_SUCCESS) ||
13792              (mptsas_check_acc_handle(mpt->m_hshk_acc_hdl) !=
13793              DDI_SUCCESS) ||
13794              (mptsas_check_acc_handle(mpt->m_config_handle) !=
13795              DDI_SUCCESS)) {
13796                  ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_UNAFFECTED);
13797                  goto fail;
13798          }

13800          return (DDI_SUCCESS);

13802  fail:
13803          return (DDI_FAILURE);
13804  }

13806  static int
13807  mptsas_get_pci_cap(mptsas_t *mpt)
13808  {
13809          ushort_t caps_ptr, cap, cap_count;

13811          if (mpt->m_config_handle == NULL)
13812                  return (FALSE);
```

```
13813         /*
13814          * Check if capabilities list is supported and if so,
13815          * get initial capabilities pointer and clear bits 0,1.
13816          */
13817         if (pci_config_get16(mpt->m_config_handle, PCI_CONF_STAT)
13818             & PCI_STAT_CAP) {
13819                 caps_ptr = P2ALIGN(pci_config_get8(mpt->m_config_handle,
13820                     PCI_CONF_CAP_PTR), 4);
13821         } else {
13822                 caps_ptr = PCI_CAP_NEXT_PTR_NULL;
13823         }

13825         /*
13826          * Walk capabilities if supported.
13827          */
13828         for (cap_count = 0; caps_ptr != PCI_CAP_NEXT_PTR_NULL; ) {

13830                 /*
13831                  * Check that we haven't exceeded the maximum number of
13832                  * capabilities and that the pointer is in a valid range.
13833                  */
13834                 if (++cap_count > 48) {
13835                         mptsas_log(mpt, CE_WARN,
13836                             "too many device capabilities.\n");
13837                         break;
13838                 }
13839                 if (caps_ptr < 64) {
13840                         mptsas_log(mpt, CE_WARN,
13841                             "capabilities pointer 0x%x out of range.\n",
13842                             caps_ptr);
13843                         break;
13844                 }

13846                 /*
13847                  * Get next capability and check that it is valid.
13848                  * For now, we only support power management.
13849                  */
13850                 cap = pci_config_get8(mpt->m_config_handle, caps_ptr);
13851                 switch (cap) {
13852                         case PCI_CAP_ID_PM:
13853                                 mptsas_log(mpt, CE_NOTE,
13854                                     "?mptsas3%d supports power management.\n",
9658                                  "?mptsas%d supports power management.\n",
13855                                     mpt->m_instance);
13856                                 mpt->m_options |= MPTSAS_OPT_PM;

13858                                 /* Save PMCSR offset */
13859                                 mpt->m_pmcsr_offset = caps_ptr + PCI_PMCSR;
13860                                 break;
13861                         case PCI_CAP_ID_MSI:
13862                                 mptsas_log(mpt, CE_NOTE,
13863                                     "?mptsas3%d supports MSI.\n",
13864                                     mpt->m_instance);
13865                                 mpt->m_options |= MPTSAS_OPT_MSI;
13866                                 break;
13867                         case PCI_CAP_ID_MSI_X:
13868                                 mptsas_log(mpt, CE_NOTE,
13869                                     "?mptsas3%d supports MSI-X.\n",
13870                                     mpt->m_instance);
13871                                 mpt->m_options |= MPTSAS_OPT_MSI_X;
13872                                 break;
13873 #endif /* ! codereview */
13874                         /*
13875                          * The following capabilities are valid.  Any others
13876                          * will cause a message to be logged.
13877                          */
```



```
13878                         case PCI_CAP_ID_VPD:
9665                          case PCI_CAP_ID_MSI:
13879                         case PCI_CAP_ID_PCIX:
13880                         case PCI_CAP_ID_PCI_E:
9668                          case PCI_CAP_ID_MSI_X:
13881                                 break;
13882                         default:
13883                                 mptsas_log(mpt, CE_NOTE,
13884                                     "?mptsas3%d unrecognized capability "
9672                                  "?mptsas%d unrecognized capability "
13885                                     "0x%x.\n", mpt->m_instance, cap);
13886                                 break;
13887                 }

13889                 /*
13890                  * Get next capabilities pointer and clear bits 0,1.
13891                  */
13892                 caps_ptr = P2ALIGN(pci_config_get8(mpt->m_config_handle,
13893                     (caps_ptr + PCI_CAP_NEXT_PTR)), 4);
13894         }
13895         return (TRUE);
13896 }

13898 static int
13899 mptsas_init_pm(mptsas_t *mpt)
13900 {
13901         char            pmc_name[16];
13902         char            *pmc[] = {
13903                                 NULL,
13904                                 "0=Off (PCI D3 State)",
13905                                 "3=On (PCI D0 State)",
13906                                 NULL
13907                         };
13908         uint16_t        pmcsr_stat;

9698          if (mptsas_get_pci_cap(mpt) == FALSE) {
9699                  return (DDI_FAILURE);
9700          }
13910         /*
13911          * If PCI's capability does not support PM, then don't need
13912          * to registe the pm-components
13913          */
13914         if (!(mpt->m_options & MPTSAS_OPT_PM))
13915                 return (DDI_SUCCESS);
13916         /*
13917          * If power management is supported by this chip, create
13918          * pm-components property for the power management framework
13919          */
13920         (void) sprintf(pmc_name, "NAME=mptsas3%d", mpt->m_instance);
9711          (void) sprintf(pmc_name, "NAME=mptsas%d", mpt->m_instance);
13921         pmc[0] = pmc_name;
13922         if (ddi_prop_update_string_array(DDI_DEV_T_NONE, mpt->m_dip,
13923             "pm-components", pmc, 3) != DDI_PROP_SUCCESS) {
13924                 mpt->m_options &= ~MPTSAS_OPT_PM;
13925                 mptsas_log(mpt, CE_WARN,
13926                     "mptsas3%d: pm-component property creation failed.",
9717                      "mptsas%d: pm-component property creation failed.",
13927                      mpt->m_instance);
13928                 return (DDI_FAILURE);
13929         }

13931         /*
13932          * Power on device.
13933          */
13934         (void) pm_busy_component(mpt->m_dip, 0);
13935         pmcsr_stat = pci_config_get16(mpt->m_config_handle,
```

```
13936             mpt->m_pmcsr_offset);
13937         if ((pmcsr_stat & PCI_PMCSR_STATE_MASK) != PCI_PMCSR_D0) {
13938                 mptsas_log(mpt, CE_WARN, "mptsas3%d: Power up the device",
9729                 mptsas_log(mpt, CE_WARN, "mptsas%d: Power up the device",
13939                     mpt->m_instance);
13940                 pci_config_put16(mpt->m_config_handle, mpt->m_pmcsr_offset,
13941                     PCI_PMCSR_D0);
13942         }
13943         if (pm_power_has_changed(mpt->m_dip, 0, PM_LEVEL_D0) != DDI_SUCCESS) {
13944                 mptsas_log(mpt, CE_WARN, "pm_power_has_changed failed");
13945                 return (DDI_FAILURE);
13946         }
13947         mpt->m_power_level = PM_LEVEL_D0;
13948         /*
13949          * Set pm idle delay.
13950          */
13951         mpt->m_pm_idle_delay = ddi_prop_get_int(DDI_DEV_T_ANY,
13952             mpt->m_dip, 0, "mptsas-pm-idle-delay", MPTSAS_PM_IDLE_TIMEOUT);

13954         return (DDI_SUCCESS);
13955 }

13957 static int
13958 mptsas_register_intrs(mptsas_t *mpt)
13959 {
13960         dev_info_t *dip;
13961         int intr_types;

13963         dip = mpt->m_dip;

13965         /* Get supported interrupt types */
13966         if (ddi_intr_get_supported_types(dip, &intr_types) != DDI_SUCCESS) {
13967                 mptsas_log(mpt, CE_WARN, "ddi_intr_get_supported_types "
13968                     "failed\n");
13969                 return (FALSE);
13970         }

13972         NDBG6(("ddi_intr_get_supported_types() returned: 0x%x", intr_types));

13974         /*
13975          * Try MSIX first.
13976          */
13977         if (mptsas_enable_msix && (intr_types & DDI_INTR_TYPE_MSIX)) {
13978                 if (mptsas_add_intrs(mpt, DDI_INTR_TYPE_MSIX) == DDI_SUCCESS) {
13979                         NDBG6(("Using MSI-X interrupt type"));
13980                         mpt->m_intr_type = DDI_INTR_TYPE_MSIX;
13981                         return (TRUE);
13982                 }
13983         }

13985         /*
13986 #endif /* ! codereview */
13987          * Try MSI, but fall back to FIXED
13988          */
13989         if (mptsas_enable_msi && (intr_types & DDI_INTR_TYPE_MSI)) {
13990                 if (mptsas_add_intrs(mpt, DDI_INTR_TYPE_MSI) == DDI_SUCCESS) {
13991                         NDBG6(("Using MSI interrupt type"));
9766                         NDBG0(("Using MSI interrupt type"));
13992                         mpt->m_intr_type = DDI_INTR_TYPE_MSI;
13993                         return (TRUE);
13994                 }
13995         }
13996         if (intr_types & DDI_INTR_TYPE_FIXED) {
13997                 if (mptsas_add_intrs(mpt, DDI_INTR_TYPE_FIXED) == DDI_SUCCESS) {
13998                         NDBG6(("Using FIXED interrupt type"));
9773                         NDBG0(("Using FIXED interrupt type"));
```



```
13999                         mpt->m_intr_type = DDI_INTR_TYPE_FIXED;
14000                         return (TRUE);
14001                 } else {
14002                         NDBG6(("FIXED interrupt registration failed"));
9777                         NDBG0(("FIXED interrupt registration failed"));
14003                         return (FALSE);
14004                 }
14005         }

14007         return (FALSE);
14008 }
______unchanged_portion_omitted_

14016 /*
14017  * mptsas_add_intrs:
14018  *
14019  * Register FIXED or MSI interrupts.
14020  */
14021 static int
14022 mptsas_add_intrs(mptsas_t *mpt, int intr_type)
14023 {
14024         dev_info_t      *dip = mpt->m_dip;
14025         int             avail, actual, count = 0;
14026         int             i, flag, ret;

14028         NDBG6(("mptsas_add_intrs:interrupt type 0x%x", intr_type));

14030         /* Get number of interrupts */
14031         ret = ddi_intr_get_nintrs(dip, intr_type, &count);
14032         if ((ret != DDI_SUCCESS) || (count <= 0)) {
14033                 mptsas_log(mpt, CE_WARN, "ddi_intr_get_nintrs() failed, "
14034                     "ret %d count %d\n", ret, count);

14036                 return (DDI_FAILURE);
14037         }

14039         /* Get number of interrupts available to this device */
9814         /* Get number of available interrupts */
14040         ret = ddi_intr_get_navail(dip, intr_type, &avail);
14041         if ((ret != DDI_SUCCESS) || (avail == 0)) {
14042                 mptsas_log(mpt, CE_WARN, "ddi_intr_get_navail() failed, "
14043                     "ret %d avail %d\n", ret, avail);

14045                 return (DDI_FAILURE);
14046         }

14048         if (count < avail) {
9823         if (avail < count) {
14049                 mptsas_log(mpt, CE_NOTE, "ddi_intr_get_nvail returned %d, "
14050                     "navail() returned %d", count, avail);
14051         }

14053         NDBG6(("mptsas_add_intrs:count %d, avail %d", count, avail));

14055         if (intr_type == DDI_INTR_TYPE_MSIX) {
14056                 if (!mptsas3_max_msix_intrs) {
14057                         return (DDI_FAILURE);
14058                 }

14060                 /*
14061                  * Restrict the number of interrupts, firstly by
14062                  * the number returned from the IOCInfo, then by
14063                  * overall restriction.
14064                  */
14065                 if (avail > mpt->m_max_msix_vectors) {
14066                         avail = mpt->m_max_msix_vectors?
```

```
14067                           mpt->m_max_msix_vectors:1;
14068                   NDBG6(("mptsas_add_intrs: mmmv avail %d", avail));
14069           }
14070           if (avail > mptsas3_max_msix_intrs) {
14071                   avail = mptsas3_max_msix_intrs;
14072                   NDBG6(("mptsas_add_intrs: m3mmi avail %d", avail));
14073           }
14074   }
14075   if (intr_type == DDI_INTR_TYPE_MSI) {
14076           NDBG6(("mptsas_add_intrs: MSI avail %d", avail));
14077           avail = 1;
9828  /* Mpt only have one interrupt routine */
9829  if ((intr_type == DDI_INTR_TYPE_MSI) && (count > 1)) {
9830          count = 1;
14078   }

14080   /* Allocate an array of interrupt handles */
14081   mpt->m_intr_size = avail * sizeof (ddi_intr_handle_t);
9834  mpt->m_intr_size = count * sizeof (ddi_intr_handle_t);
14082   mpt->m_htable = kmem_alloc(mpt->m_intr_size, KM_SLEEP);

14084   flag = DDI_INTR_ALLOC_NORMAL;

14086   /* call ddi_intr_alloc() */
14087   ret = ddi_intr_alloc(dip, mpt->m_htable, intr_type, 0,
14088       avail, &actual, flag);
9841       count, &actual, flag);

14090   if ((ret != DDI_SUCCESS) || (actual == 0)) {
14091           mptsas_log(mpt, CE_WARN, "ddi_intr_alloc() failed, ret %d\n",
14092               ret);
14093           kmem_free(mpt->m_htable, mpt->m_intr_size);
14094           return (DDI_FAILURE);
14095   }

14097   NDBG6(("mptsas_add_intrs: actual %d, avail %d", actual, avail));
14098 #endif /* ! codereview */
14099   /* use interrupt count returned or abort? */
14100   if (actual < avail) {
14101           mptsas_log(mpt, CE_NOTE,
14102               "Interrupts requested: %d, received: %d\n",
14103               avail, actual);
9850  if (actual < count) {
9851          mptsas_log(mpt, CE_NOTE, "Requested: %d, Received: %d\n",
9852              count, actual);
14104   }

9855  mpt->m_intr_cnt = actual;

14106   /*
14107    * Get priority for first msi, assume remaining are all the same
14108    */
14109   if ((ret = ddi_intr_get_pri(mpt->m_htable[0],
14110       &mpt->m_intr_pri)) != DDI_SUCCESS) {
14111           mptsas_log(mpt, CE_WARN, "ddi_intr_get_pri() failed %d\n", ret);

14113           /* Free already allocated intr */
14114           for (i = 0; i < actual; i++) {
14115                   (void) ddi_intr_free(mpt->m_htable[i]);
14116           }

14118           kmem_free(mpt->m_htable, mpt->m_intr_size);
14119           return (DDI_FAILURE);
14120   }

14122   /* Test for high level mutex */
```



```
14123   if (mpt->m_intr_pri >= ddi_intr_get_hilevel_pri()) {
14124           mptsas_log(mpt, CE_WARN, "mptsas_add_intrs: "
14125               "Hi level interrupt not supported\n");

14127           /* Free already allocated intr */
14128           for (i = 0; i < actual; i++) {
14129                   (void) ddi_intr_free(mpt->m_htable[i]);
14130           }

14132           kmem_free(mpt->m_htable, mpt->m_intr_size);
14133           return (DDI_FAILURE);
14134   }

14136   /* Call ddi_intr_add_handler() */
14137   for (i = 0; i < actual; i++) {
14138           if ((ret = ddi_intr_add_handler(mpt->m_htable[i], mptsas_intr,
14139               (caddr_t)mpt, (caddr_t)(uintptr_t)i)) != DDI_SUCCESS) {
14140                   mptsas_log(mpt, CE_WARN, "ddi_intr_add_handler() "
14141                       "failed %d\n", ret);

14143                   /* Free already allocated intr */
14144                   for (i = 0; i < actual; i++) {
14145                           (void) ddi_intr_free(mpt->m_htable[i]);
14146                   }

14148                   kmem_free(mpt->m_htable, mpt->m_intr_size);
14149                   return (DDI_FAILURE);
14150           }
14151   }

14153   if ((ret = ddi_intr_get_cap(mpt->m_htable[0], &mpt->m_intr_cap))
14154       != DDI_SUCCESS) {
14155           mptsas_log(mpt, CE_WARN, "ddi_intr_get_cap() failed %d\n", ret);

14157           /* Free already allocated intr */
14158           for (i = 0; i < actual; i++) {
14159                   (void) ddi_intr_free(mpt->m_htable[i]);
14160           }

14162           kmem_free(mpt->m_htable, mpt->m_intr_size);
14163           return (DDI_FAILURE);
14164   }

14166   mpt->m_intr_cnt = actual;

14168 #endif /* ! codereview */
14169   /*
14170    * Enable interrupts
14171    */
14172   if (mpt->m_intr_cap & DDI_INTR_FLAG_BLOCK) {
14173           /* Call ddi_intr_block_enable() for MSI interrupts */
14174           (void) ddi_intr_block_enable(mpt->m_htable, mpt->m_intr_cnt);
14175   } else {
14176           /* Call ddi_intr_enable for MSI or FIXED interrupts */
14177           for (i = 0; i < mpt->m_intr_cnt; i++) {
14178                   (void) ddi_intr_enable(mpt->m_htable[i]);
14179           }
14180   }

14182   switch (intr_type) {
14183   case DDI_INTR_TYPE_MSIX:
14184           mptsas_log(mpt, CE_NOTE, "?Using %d MSI-X interrupt(s) "
14185               "(Available sys %d, mpt %d, Requested %d)\n",
14186               actual, count, mpt->m_max_msix_vectors, avail);
14187           break;
14188   case DDI_INTR_TYPE_MSI:
```

```
14189                 mptsas_log(mpt, CE_NOTE, "Using single MSI interrupt\n");
14190                 break;
14191         case DDI_INTR_TYPE_FIXED:
14192         default:
14193                 mptsas_log(mpt, CE_NOTE, "Using single fixed interrupt\n");
14194                 break;
14195         }

14197 #endif /* ! codereview */
14198         return (DDI_SUCCESS);
14199 }

14201 /*
14202  * mptsas_rem_intrs:
14203  *
14204  * Unregister FIXED or MSI interrupts
14205  */
14206 static void
14207 mptsas_rem_intrs(mptsas_t *mpt)
14208 {
14209         int     i;

14211         NDBG6(("mptsas_rem_intrs"));

14213         /* Disable all interrupts */
14214         if (mpt->m_intr_cap & DDI_INTR_FLAG_BLOCK) {
14215                 /* Call ddi_intr_block_disable() */
14216                 (void) ddi_intr_block_disable(mpt->m_htable, mpt->m_intr_cnt);
14217         } else {
14218                 for (i = 0; i < mpt->m_intr_cnt; i++) {
14219                         (void) ddi_intr_disable(mpt->m_htable[i]);
14220                 }
14221         }

14223         /* Call ddi_intr_remove_handler() */
14224         for (i = 0; i < mpt->m_intr_cnt; i++) {
14225                 (void) ddi_intr_remove_handler(mpt->m_htable[i]);
14226                 (void) ddi_intr_free(mpt->m_htable[i]);
14227         }

14228         kmem_free(mpt->m_htable, mpt->m_intr_size);
14229         mpt->m_intr_cnt = 0;
14230 #endif /* ! codereview */
14231 }

14233 /*
14234  * The IO fault service error handling callback function
14235  */
14236 /*ARGSUSED*/
14237 static int
14238 mptsas_fm_error_cb(dev_info_t *dip, ddi_fm_error_t *err, const void *impl_data)
14239 {
14240         /*
14241          * as the driver can always deal with an error in any dma or
14242          * access handle, we can just return the fme_status value.
14243          */
14244         pci_ereport_post(dip, err, NULL);
14245         return (err->fme_status);
14246 }

14248 /*
14249  * mptsas_fm_init - initialize fma capabilities and register with IO
14250  *               fault services.
14251  */
14252 static void
14253 mptsas_fm_init(mptsas_t *mpt)
```



```
14254 {
14255         /*
14256          * Need to change iblock to priority for new MSI intr
14257          */
14258         ddi_iblock_cookie_t     fm_ibc;

14260         /* Only register with IO Fault Services if we have some capability */
14261         if (mpt->m_fm_capabilities) {
14262                 /* Adjust access and dma attributes for FMA */
14263                 mpt->m_reg_acc_attr.devacc_attr_access = DDI_FLAGERR_ACC;
14264                 mpt->m_msg_dma_attr.dma_attr_flags |= DDI_DMA_FLAGERR;
14265                 mpt->m_io_dma_attr.dma_attr_flags |= DDI_DMA_FLAGERR;

14267                 /*
14268                  * Register capabilities with IO Fault Services.
14269                  * mpt->m_fm_capabilities will be updated to indicate
14270                  * capabilities actually supported (not requested.)
14271                  */
14272                 ddi_fm_init(mpt->m_dip, &mpt->m_fm_capabilities, &fm_ibc);

14274                 /*
14275                  * Initialize pci ereport capabilities if ereport
14276                  * capable (should always be.)
14277                  */
14278                 if (DDI_FM_EREPORT_CAP(mpt->m_fm_capabilities) ||
14279                     DDI_FM_ERRCB_CAP(mpt->m_fm_capabilities)) {
14280                         pci_ereport_setup(mpt->m_dip);
14281                 }

14283                 /*
14284                  * Register error callback if error callback capable.
14285                  */
14286                 if (DDI_FM_ERRCB_CAP(mpt->m_fm_capabilities)) {
14287                         ddi_fm_handler_register(mpt->m_dip,
14288                             mptsas_fm_error_cb, (void *) mpt);
14289                 }
14290         }
14291 }

14293 /*
14294  * mptsas_fm_fini - Releases fma capabilities and un-registers with IO
14295  *               fault services.
14296  *
14297  */
14298 static void
14299 mptsas_fm_fini(mptsas_t *mpt)
14300 {
14301         /* Only unregister FMA capabilities if registered */
14302         if (mpt->m_fm_capabilities) {

14304                 /*
14305                  * Un-register error callback if error callback capable.
14306                  */

14308                 if (DDI_FM_ERRCB_CAP(mpt->m_fm_capabilities)) {
14309                         ddi_fm_handler_unregister(mpt->m_dip);
14310                 }

14312                 /*
14313                  * Release any resources allocated by pci_ereport_setup()
14314                  */

14316                 if (DDI_FM_EREPORT_CAP(mpt->m_fm_capabilities) ||
14317                     DDI_FM_ERRCB_CAP(mpt->m_fm_capabilities)) {
14318                         pci_ereport_teardown(mpt->m_dip);
14319                 }
```

```
14321                 /* Unregister from IO Fault Services */
14322                 ddi_fm_fini(mpt->m_dip);

14324                 /* Adjust access and dma attributes for FMA */
14325                 mpt->m_reg_acc_attr.devacc_attr_access = DDI_DEFAULT_ACC;
14326                 mpt->m_msg_dma_attr.dma_attr_flags &= ~DDI_DMA_FLAGERR;
14327                 mpt->m_io_dma_attr.dma_attr_flags &= ~DDI_DMA_FLAGERR;

14329         }
14330 }

14332 int
14333 mptsas_check_acc_handle(ddi_acc_handle_t handle)
14334 {
14335         ddi_fm_error_t  de;

14337         if (handle == NULL)
14338                 return (DDI_FAILURE);
14339         ddi_fm_acc_err_get(handle, &de, DDI_FME_VER0);
14340         return (de.fme_status);
14341 }

14343 int
14344 mptsas_check_dma_handle(ddi_dma_handle_t handle)
14345 {
14346         ddi_fm_error_t  de;

14348         if (handle == NULL)
14349                 return (DDI_FAILURE);
14350         ddi_fm_dma_err_get(handle, &de, DDI_FME_VER0);
14351         return (de.fme_status);
14352 }

14354 void
14355 mptsas_fm_ereport(mptsas_t *mpt, char *detail)
14356 {
14357         uint64_t        ena;
14358         char            buf[FM_MAX_CLASS];

14360         (void) snprintf(buf, FM_MAX_CLASS, "%s.%s", DDI_FM_DEVICE, detail);
14361         ena = fm_ena_generate(0, FM_ENA_FMT1);
14362         if (DDI_FM_EREPORT_CAP(mpt->m_fm_capabilities)) {
14363                 ddi_fm_ereport_post(mpt->m_dip, buf, ena, DDI_NOSLEEP,
14364                     FM_VERSION, DATA_TYPE_UINT8, FM_EREPORT_VERS0, NULL);
14365         }
14366 }

14368 static int
14369 mptsas_get_target_device_info(mptsas_t *mpt, uint32_t page_address,
14370     uint16_t *dev_handle, mptsas_target_t **pptgt)
14371 {
14372         int             rval;
14373         uint32_t        dev_info;
14374         uint64_t        sas_wwn;
14375         mptsas_phymask_t phymask;
14376         uint8_t         physport, phynum, config, disk;
14377         uint64_t        devicename;
14378         uint16_t        pdev_hdl;
14379         mptsas_target_t *tmp_tgt = NULL;
14380         uint16_t        bay_num, enclosure, io_flags;
9919         uint16_t        bay_num, enclosure;

14382         ASSERT(*pptgt == NULL);

14384         rval = mptsas_get_sas_device_page0(mpt, page_address, dev_handle,
```



```
14385             &sas_wwn, &dev_info, &physport, &phynum, &pdev_hdl,
14386             &bay_num, &enclosure, &io_flags);
9925             &bay_num, &enclosure);
14387         if (rval != DDI_SUCCESS) {
14388                 rval = DEV_INFO_FAIL_PAGE0;
14389                 return (rval);
14390         }

14392         if ((dev_info & (MPI2_SAS_DEVICE_INFO_SSP_TARGET |
14393             MPI2_SAS_DEVICE_INFO_SATA_DEVICE |
14394             MPI2_SAS_DEVICE_INFO_ATAPI_DEVICE)) == NULL) {
14395                 rval = DEV_INFO_WRONG_DEVICE_TYPE;
14396                 return (rval);
14397         }

14399         /*
14400          * Check if the dev handle is for a Phys Disk. If so, set return value
14401          * and exit.  Don't add Phys Disks to hash.
14402          */
14403         for (config = 0; config < mpt->m_num_raid_configs; config++) {
14404                 for (disk = 0; disk < MPTSAS_MAX_DISKS_IN_CONFIG; disk++) {
14405                         if (*dev_handle == mpt->m_raidconfig[config].
14406                             m_physdisk_devhdl[disk]) {
14407                                 rval = DEV_INFO_PHYS_DISK;
14408                                 return (rval);
14409                         }
14410                 }
14411         }

14413         /*
14414          * Get SATA Device Name from SAS device page0 for
14415          * sata device, if device name doesn't exist, set mta_wwn to
14416          * 0 for direct attached SATA. For the device behind the expander
14417          * we still can use STP address assigned by expander.
14418          */
14419         if (dev_info & (MPI2_SAS_DEVICE_INFO_SATA_DEVICE |
14420             MPI2_SAS_DEVICE_INFO_ATAPI_DEVICE)) {
14421                 mutex_exit(&mpt->m_mutex);
14422                 /* alloc a tmp_tgt to send the cmd */
14423                 tmp_tgt = kmem_zalloc(sizeof (struct mptsas_target),
14424                     KM_SLEEP);
14425                 tmp_tgt->m_devhdl = *dev_handle;
14426                 tmp_tgt->m_deviceinfo = dev_info;
14427                 tmp_tgt->m_qfull_retries = QFULL_RETRIES;
14428                 tmp_tgt->m_qfull_retry_interval =
14429                     drv_usectohz(QFULL_RETRY_INTERVAL * 1000);
14430                 tmp_tgt->m_t_throttle = MAX_THROTTLE;
14431                 mutex_init(&tmp_tgt->m_t_mutex, NULL, MUTEX_DRIVER, NULL);
14432 #endif /* ! codereview */
14433                 devicename = mptsas_get_sata_guid(mpt, tmp_tgt, 0);
14434                 mutex_destroy(&tmp_tgt->m_t_mutex);
14435 #endif /* ! codereview */
14436                 kmem_free(tmp_tgt, sizeof (struct mptsas_target));
14437                 mutex_enter(&mpt->m_mutex);
14438                 if (devicename != 0 && (((devicename >> 56) & 0xf0) == 0x50)) {
14439                         sas_wwn = devicename;
14440                 } else if (dev_info & MPI2_SAS_DEVICE_INFO_DIRECT_ATTACH) {
14441                         sas_wwn = 0;
14442                 }
14443         }

14445         phymask = mptsas_physport_to_phymask(mpt, physport);
14446         *pptgt = mptsas_tgt_alloc(mpt, *dev_handle, sas_wwn,
14447             dev_info, phymask, phynum);
14448         if (*pptgt == NULL) {
14449                 mptsas_log(mpt, CE_WARN, "Failed to allocated target"
```

```
                    "structure!");
                rval = DEV_INFO_FAIL_ALLOC;
                return (rval);
        }
        (*pptgt)->m_io_flags = io_flags;
#endif /* ! codereview */
        (*pptgt)->m_enclosure = enclosure;
        (*pptgt)->m_slot_num = bay_num;
        return (DEV_INFO_SUCCESS);
}

uint64_t
mptsas_get_sata_guid(mptsas_t *mpt, mptsas_target_t *ptgt, int lun)
{
        uint64_t        sata_guid = 0, *pwwn = NULL;
        int             target = ptgt->m_devhdl;
        uchar_t         *inq83 = NULL;
        int             inq83_len = 0xFF;
        uchar_t         *dblk = NULL;
        int             inq83_retry = 3;
        int             rval = DDI_FAILURE;

        inq83   = kmem_zalloc(inq83_len, KM_SLEEP);

inq83_retry:
        rval = mptsas_inquiry(mpt, ptgt, lun, 0x83, inq83,
            inq83_len, NULL, 1);
        if (rval != DDI_SUCCESS) {
                mptsas_log(mpt, CE_WARN, "!mptsas request inquiry page "
                    "0x83 for target:%x, lun:%x failed!", target, lun);
                goto out;
        }
        /* According to SAT2, the first descriptor is logic unit name */
        dblk = &inq83[4];
        if ((dblk[1] & 0x30) != 0) {
                mptsas_log(mpt, CE_WARN, "!Descriptor is not lun associated.");
                goto out;
        }
        pwwn = (uint64_t *)(void *)(&dblk[4]);
        if ((dblk[4] & 0xf0) == 0x50) {
                sata_guid = BE_64(*pwwn);
                goto out;
        } else if (dblk[4] == 'A') {
                NDBG20(("SATA drive has no NAA format GUID."));
                goto out;
        } else {
                /* The data is not ready, wait and retry */
                inq83_retry--;
                if (inq83_retry <= 0) {
                        goto out;
                }
                NDBG20(("The GUID is not ready, retry..."));
                delay(1 * drv_usectohz(1000000));
                goto inq83_retry;
        }
out:
        kmem_free(inq83, inq83_len);
        return (sata_guid);
}

static int
mptsas_inquiry(mptsas_t *mpt, mptsas_target_t *ptgt, int lun, uchar_t page,
    unsigned char *buf, int len, int *reallen, uchar_t evpd)
{
        uchar_t                 cdb[CDB_GROUP0];
        struct scsi_address     ap;
```



```
        struct buf              *data_bp = NULL;
        int                     resid = 0;
        int                     ret = DDI_FAILURE;

        ASSERT(len <= 0xffff);

        ap.a_target = MPTSAS_INVALID_DEVHDL;
        ap.a_lun = (uchar_t)(lun);
        ap.a_hba_tran = mpt->m_tran;

        data_bp = scsi_alloc_consistent_buf(&ap,
            (struct buf *)NULL, len, B_READ, NULL_FUNC, NULL);
        if (data_bp == NULL) {
                return (ret);
        }
        bzero(cdb, CDB_GROUP0);
        cdb[0] = SCMD_INQUIRY;
        cdb[1] = evpd;
        cdb[2] = page;
        cdb[3] = (len & 0xff00) >> 8;
        cdb[4] = (len & 0x00ff);
        cdb[5] = 0;

        ret = mptsas_send_scsi_cmd(mpt, &ap, ptgt, &cdb[0], CDB_GROUP0, data_bp,
            &resid);
        if (ret == DDI_SUCCESS) {
                if (reallen) {
                        *reallen = len - resid;
                }
                bcopy((caddr_t)data_bp->b_un.b_addr, buf, len);
        }
        if (data_bp) {
                scsi_free_consistent_buf(data_bp);
        }
        return (ret);
}

static int
mptsas_send_scsi_cmd(mptsas_t *mpt, struct scsi_address *ap,
    mptsas_target_t *ptgt, uchar_t *cdb, int cdblen, struct buf *data_bp,
    int *resid)
{
        struct scsi_pkt         *pktp = NULL;
        scsi_hba_tran_t         *tran_clone = NULL;
        mptsas_tgt_private_t    *tgt_private = NULL;
        int                     ret = DDI_FAILURE;

        /*
         * scsi_hba_tran_t->tran_tgt_private is used to pass the address
         * information to scsi_init_pkt, allocate a scsi_hba_tran structure
         * to simulate the cmds from sd
         */
        tran_clone = kmem_alloc(
            sizeof (scsi_hba_tran_t), KM_SLEEP);
        if (tran_clone == NULL) {
                goto out;
        }
        bcopy((caddr_t)mpt->m_tran,
            (caddr_t)tran_clone, sizeof (scsi_hba_tran_t));
        tgt_private = kmem_alloc(
            sizeof (mptsas_tgt_private_t), KM_SLEEP);
        if (tgt_private == NULL) {
                goto out;
        }
        tgt_private->t_lun = ap->a_lun;
        tgt_private->t_private = ptgt;
```

```
14582         tran_clone->tran_tgt_private = tgt_private;
14583         ap->a_hba_tran = tran_clone;

14585         pktp = scsi_init_pkt(ap, (struct scsi_pkt *)NULL,
14586             data_bp, cdblen, sizeof (struct scsi_arq_status),
14587             0, PKT_CONSISTENT, NULL, NULL);
14588         if (pktp == NULL) {
14589                 goto out;
14590         }
14591         bcopy(cdb, pktp->pkt_cdbp, cdblen);
14592         pktp->pkt_flags = FLAG_NOPARITY;
14593         pktp->pkt_time = mptsas_scsi_pkt_time;
14594 #endif /* ! codereview */
14595         if (scsi_poll(pktp) < 0) {
14596                 goto out;
14597         }
14598         if (((struct scsi_status *)pktp->pkt_scbp)->sts_chk) {
14599                 goto out;
14600         }
14601         if (resid != NULL) {
14602                 *resid = pktp->pkt_resid;
14603         }

14605         ret = DDI_SUCCESS;
14606 out:
14607         if (pktp) {
14608                 scsi_destroy_pkt(pktp);
14609         }
14610         if (tran_clone) {
14611                 kmem_free(tran_clone, sizeof (scsi_hba_tran_t));
14612         }
14613         if (tgt_private) {
14614                 kmem_free(tgt_private, sizeof (mptsas_tgt_private_t));
14615         }
14616         return (ret);
14617 }
14618 static int
14619 mptsas_parse_address(char *name, uint64_t *wwid, uint8_t *phy, int *lun)
14620 {
14621         char    *cp = NULL;
14622         char    *ptr = NULL;
14623         size_t  s = 0;
14624         char    *wwid_str = NULL;
14625         char    *lun_str = NULL;
14626         long    lunnum;
14627         long    phyid = -1;
14628         int     rc = DDI_FAILURE;

14630         ptr = name;
14631         ASSERT(ptr[0] == 'w' || ptr[0] == 'p');
14632         ptr++;
14633         if ((cp = strchr(ptr, ',')) == NULL) {
14634                 return (DDI_FAILURE);
14635         }

14637         wwid_str = kmem_zalloc(SCSI_MAXNAMELEN, KM_SLEEP);
14638         s = (uintptr_t)cp - (uintptr_t)ptr;

14640         bcopy(ptr, wwid_str, s);
14641         wwid_str[s] = '\0';

14643         ptr = ++cp;

14645         if ((cp = strchr(ptr, '\0')) == NULL) {
14646                 goto out;
14647         }
```



```
14648         lun_str =  kmem_zalloc(SCSI_MAXNAMELEN, KM_SLEEP);
14649         s = (uintptr_t)cp - (uintptr_t)ptr;

14651         bcopy(ptr, lun_str, s);
14652         lun_str[s] = '\0';

14654         if (name[0] == 'p') {
14655                 rc = ddi_strtol(wwid_str, NULL, 0x10, &phyid);
14656         } else {
14657                 rc = scsi_wwnstr_to_wwn(wwid_str, wwid);
14658         }
14659         if (rc != DDI_SUCCESS)
14660                 goto out;

14662         if (phyid != -1) {
14663                 ASSERT(phyid < MPTSAS_MAX_PHYS);
14664                 *phy = (uint8_t)phyid;
14665         }
14666         rc = ddi_strtol(lun_str, NULL, 0x10, &lunnum);
14667         if (rc != 0)
14668                 goto out;

14670         *lun = (int)lunnum;
14671         rc = DDI_SUCCESS;
14672 out:
14673         if (wwid_str)
14674                 kmem_free(wwid_str, SCSI_MAXNAMELEN);
14675         if (lun_str)
14676                 kmem_free(lun_str, SCSI_MAXNAMELEN);

14678         return (rc);
14679 }

14681 /*
14682  * mptsas_parse_smp_name() is to parse sas wwn string
14683  * which format is "wWWN"
14684  */
14685 static int
14686 mptsas_parse_smp_name(char *name, uint64_t *wwn)
14687 {
14688         char    *ptr = name;

14690         if (*ptr != 'w') {
14691                 return (DDI_FAILURE);
14692         }

14694         ptr++;
14695         if (scsi_wwnstr_to_wwn(ptr, wwn)) {
14696                 return (DDI_FAILURE);
14697         }
14698         return (DDI_SUCCESS);
14699 }

14701 static int
14702 mptsas_bus_config(dev_info_t *pdip, uint_t flag,
14703     ddi_bus_config_op_t op, void *arg, dev_info_t **childp)
14704 {
14705         int             ret = NDI_FAILURE;
14706         int             circ = 0;
14707         int             circ1 = 0;
14708         mptsas_t        *mpt;
14709         char            *ptr = NULL;
14710         char            *devnm = NULL;
14711         uint64_t        wwid = 0;
14712         uint8_t         phy = 0xFF;
14713         int             lun = 0;
```

```
14714         uint_t          mflags = flag;
14715         int             bconfig = TRUE;

14717         if (scsi_hba_iport_unit_address(pdip) == 0) {
14718                 return (DDI_FAILURE);
14719         }

14721         mpt = DIP2MPT(pdip);
14722         if (!mpt) {
14723                 return (DDI_FAILURE);
14724         }
14725         /*
14726          * Hold the nexus across the bus_config
14727          */
14728         ndi_devi_enter(scsi_vhci_dip, &circ);
14729         ndi_devi_enter(pdip, &circ1);
14730         switch (op) {
14731         case BUS_CONFIG_ONE:
14732                 /* parse wwid/target name out of name given */
14733                 if ((ptr = strchr((char *)arg, '@')) == NULL) {
14734                         ret = NDI_FAILURE;
14735                         break;
14736                 }
14737                 ptr++;
14738                 if (strncmp((char *)arg, "smp", 3) == 0) {
14739                         /*
14740                          * This is a SMP target device
14741                          */
14742                         ret = mptsas_parse_smp_name(ptr, &wwid);
14743                         if (ret != DDI_SUCCESS) {
14744                                 ret = NDI_FAILURE;
14745                                 break;
14746                         }
14747                         ret = mptsas_config_smp(pdip, wwid, childp);
14748                 } else if ((ptr[0] == 'w') || (ptr[0] == 'p')) {
14749                         /*
14750                          * OBP could pass down a non-canonical form
14751                          * bootpath without LUN part when LUN is 0.
14752                          * So driver need adjust the string.
14753                          */
14754                         if (strchr(ptr, ',') == NULL) {
14755                                 devnm = kmem_zalloc(SCSI_MAXNAMELEN, KM_SLEEP);
14756                                 (void) sprintf(devnm, "%s,0", (char *)arg);
14757                                 ptr = strchr(devnm, '@');
14758                                 ptr++;
14759                         }

14761                         /*
14762                          * The device path is wWWID format and the device
14763                          * is not SMP target device.
14764                          */
14765                         ret = mptsas_parse_address(ptr, &wwid, &phy, &lun);
14766                         if (ret != DDI_SUCCESS) {
14767                                 ret = NDI_FAILURE;
14768                                 break;
14769                         }
14770                         *childp = NULL;
14771                         if (ptr[0] == 'w') {
14772                                 ret = mptsas_config_one_addr(pdip, wwid,
14773                                     lun, childp);
14774                         } else if (ptr[0] == 'p') {
14775                                 ret = mptsas_config_one_phy(pdip, phy, lun,
14776                                     childp);
14777                         }

14779                         /*
```



```
14780                          * If this is CD/DVD device in OBP path, the
14781                          * ndi_busop_bus_config can be skipped as config one
14782                          * operation is done above.
14783                          */
14784                         if ((ret == NDI_SUCCESS) && (*childp != NULL) &&
14785                             (strcmp(ddi_node_name(*childp), "cdrom") == 0) &&
14786                             (strncmp((char *)arg, "disk", 4) == 0)) {
14787                                 bconfig = FALSE;
14788                                 ndi_hold_devi(*childp);
14789                         }
14790                 } else {
14791                         ret = NDI_FAILURE;
14792                         break;
14793                 }

14795                 /*
14796                  * DDI group instructed us to use this flag.
14797                  */
14798                 mflags |= NDI_MDI_FALLBACK;
14799                 break;
14800         case BUS_CONFIG_DRIVER:
14801         case BUS_CONFIG_ALL:
14802                 mptsas_config_all(pdip);
14803                 ret = NDI_SUCCESS;
14804                 break;
14805         }

14807         if ((ret == NDI_SUCCESS) && bconfig) {
14808                 ret = ndi_busop_bus_config(pdip, mflags, op,
14809                     (devnm == NULL) ? arg : devnm, childp, 0);
14810         }

14812         ndi_devi_exit(pdip, circ1);
14813         ndi_devi_exit(scsi_vhci_dip, circ);
14814         if (devnm != NULL)
14815                 kmem_free(devnm, SCSI_MAXNAMELEN);
14816         return (ret);
14817 }

14819 static int
14820 mptsas_probe_lun(dev_info_t *pdip, int lun, dev_info_t **dip,
14821     mptsas_target_t *ptgt)
14822 {
14823         int                     rval = DDI_FAILURE;
14824         struct scsi_inquiry     *sd_inq = NULL;
14825         mptsas_t                *mpt = DIP2MPT(pdip);

14827         sd_inq = (struct scsi_inquiry *)kmem_alloc(SUN_INQSIZE, KM_SLEEP);

14829         rval = mptsas_inquiry(mpt, ptgt, lun, 0, (uchar_t *)sd_inq,
14830             SUN_INQSIZE, 0, (uchar_t)0);

14832         if ((rval == DDI_SUCCESS) && MPTSAS_VALID_LUN(sd_inq)) {
14833                 rval = mptsas_create_lun(pdip, sd_inq, dip, ptgt, lun);
14834         } else {
14835                 rval = DDI_FAILURE;
14836         }

14838         kmem_free(sd_inq, SUN_INQSIZE);
14839         return (rval);
14840 }

14842 static int
14843 mptsas_config_one_addr(dev_info_t *pdip, uint64_t sasaddr, int lun,
14844     dev_info_t **lundip)
14845 {
```

```
14846         int             rval;
14847         mptsas_t                *mpt = DIP2MPT(pdip);
14848         int             phymask;
14849         mptsas_target_t *ptgt = NULL;

14851         /*
14852          * Get the physical port associated to the iport
14853          */
14854         phymask = ddi_prop_get_int(DDI_DEV_T_ANY, pdip, 0,
14855             "phymask", 0);

14857         ptgt = mptsas_wwid_to_ptgt(mpt, phymask, sasaddr);
14858         if (ptgt == NULL) {
14859                 /*
14860                  * didn't match any device by searching
14861                  */
14862                 return (DDI_FAILURE);
14863         }
14864         /*
14865          * If the LUN already exists and the status is online,
14866          * we just return the pointer to dev_info_t directly.
14867          * For the mdi_pathinfo node, we'll handle it in
14868          * mptsas_create_virt_lun()
14869          * TODO should be also in mptsas_handle_dr
14870          */

14872         *lundip = mptsas_find_child_addr(pdip, sasaddr, lun);
14873         if (*lundip != NULL) {
14874                 /*
14875                  * TODO Another senario is, we hotplug the same disk
14876                  * on the same slot, the devhdl changed, is this
14877                  * possible?
14878                  * tgt_private->t_private != ptgt
14879                  */
14880                 if (sasaddr != ptgt->m_addr.mta_wwn) {
14881                         /*
14882                          * The device has changed although the devhdl is the
14883                          * same (Enclosure mapping mode, change drive on the
14884                          * same slot)
14885                          */
14886                         return (DDI_FAILURE);
14887                 }
14888                 return (DDI_SUCCESS);
14889         }

14891         if (phymask == 0) {
14892                 /*
14893                  * Configure IR volume
14894                  */
14895                 rval =  mptsas_config_raid(pdip, ptgt->m_devhdl, lundip);
14896                 return (rval);
14897         }
14898         rval = mptsas_probe_lun(pdip, lun, lundip, ptgt);

14900         return (rval);
14901 }

14903 static int
14904 mptsas_config_one_phy(dev_info_t *pdip, uint8_t phy, int lun,
14905     dev_info_t **lundip)
14906 {
14907         int             rval;
14908         mptsas_t        *mpt = DIP2MPT(pdip);
14909         mptsas_phymask_t phymask;
14910         mptsas_target_t *ptgt = NULL;
```



```
14912         /*
14913          * Get the physical port associated to the iport
14914          */
14915         phymask = (mptsas_phymask_t)ddi_prop_get_int(DDI_DEV_T_ANY, pdip, 0,
14916             "phymask", 0);

14918         ptgt = mptsas_phy_to_tgt(mpt, phymask, phy);
14919         if (ptgt == NULL) {
14920                 /*
14921                  * didn't match any device by searching
14922                  */
14923                 return (DDI_FAILURE);
14924         }

14926         /*
14927          * If the LUN already exists and the status is online,
14928          * we just return the pointer to dev_info_t directly.
14929          * For the mdi_pathinfo node, we'll handle it in
14930          * mptsas_create_virt_lun().
14931          */

14933         *lundip = mptsas_find_child_phy(pdip, phy);
14934         if (*lundip != NULL) {
14935                 return (DDI_SUCCESS);
14936         }

14938         rval = mptsas_probe_lun(pdip, lun, lundip, ptgt);

14940         return (rval);
14941 }

14943 static int
14944 mptsas_retrieve_lundata(int lun_cnt, uint8_t *buf, uint16_t *lun_num,
14945     uint8_t *lun_addr_type)
14946 {
14947         uint32_t        lun_idx = 0;

14949         ASSERT(lun_num != NULL);
14950         ASSERT(lun_addr_type != NULL);

14952         lun_idx = (lun_cnt + 1) * MPTSAS_SCSI_REPORTLUNS_ADDRESS_SIZE;
14953         /* determine report luns addressing type */
14954         switch (buf[lun_idx] & MPTSAS_SCSI_REPORTLUNS_ADDRESS_MASK) {
14955                 /*
14956                  * Vendors in the field have been found to be concatenating
14957                  * bus/target/lun to equal the complete lun value instead
14958                  * of switching to flat space addressing
14959                  */
14960                 /* 00b - peripheral device addressing method */
14961         case MPTSAS_SCSI_REPORTLUNS_ADDRESS_PERIPHERAL:
14962                 /* FALLTHRU */
14963                 /* 10b - logical unit addressing method */
14964         case MPTSAS_SCSI_REPORTLUNS_ADDRESS_LOGICAL_UNIT:
14965                 /* FALLTHRU */
14966                 /* 01b - flat space addressing method */
14967         case MPTSAS_SCSI_REPORTLUNS_ADDRESS_FLAT_SPACE:
14968                 /* byte0 bit0-5=msb lun byte1 bit0-7=lsb lun */
14969                 *lun_addr_type = (buf[lun_idx] &
14970                     MPTSAS_SCSI_REPORTLUNS_ADDRESS_MASK) >> 6;
14971                 *lun_num = (buf[lun_idx] & 0x3F) << 8;
14972                 *lun_num |= buf[lun_idx + 1];
14973                 return (DDI_SUCCESS);
14974         default:
14975                 return (DDI_FAILURE);
14976         }
14977 }
```

```
14979 static int
14980 mptsas_config_luns(dev_info_t *pdip, mptsas_target_t *ptgt)
14981 {
14982         struct buf              *repluns_bp = NULL;
14983         struct scsi_address     ap;
14984         uchar_t                 cdb[CDB_GROUP5];
14985         int                     ret = DDI_FAILURE;
14986         int                     retry = 0;
14987         int                     lun_list_len = 0;
14988         uint16_t                lun_num = 0;
14989         uint8_t                 lun_addr_type = 0;
14990         uint32_t                lun_cnt = 0;
14991         uint32_t                lun_total = 0;
14992         dev_info_t              *cdip = NULL;
14993         uint16_t                *saved_repluns = NULL;
14994         char                    *buffer = NULL;
14995         int                     buf_len = 128;
14996         mptsas_t                *mpt = DIP2MPT(pdip);
14997         uint64_t                sas_wwn = 0;
14998         uint8_t                 phy = 0xFF;
14999         uint32_t                dev_info = 0;

15001         mutex_enter(&mpt->m_mutex);
15002         sas_wwn = ptgt->m_addr.mta_wwn;
15003         phy = ptgt->m_phynum;
15004         dev_info = ptgt->m_deviceinfo;
15005         mutex_exit(&mpt->m_mutex);

15007         if (sas_wwn == 0) {
15008                 /*
15009                  * It's a SATA without Device Name
15010                  * So don't try multi-LUNs
15011                  */
15012                 if (mptsas_find_child_phy(pdip, phy)) {
15013                         return (DDI_SUCCESS);
15014                 } else {
15015                         /*
15016                          * need configure and create node
15017                          */
15018                         return (DDI_FAILURE);
15019                 }
15020         }

15022         /*
15023          * WWN (SAS address or Device Name exist)
15024          */
15025         if (dev_info & (MPI2_SAS_DEVICE_INFO_SATA_DEVICE |
15026             MPI2_SAS_DEVICE_INFO_ATAPI_DEVICE)) {
15027                 /*
15028                  * SATA device with Device Name
15029                  * So don't try multi-LUNs
15030                  */
15031                 if (mptsas_find_child_addr(pdip, sas_wwn, 0)) {
15032                         return (DDI_SUCCESS);
15033                 } else {
15034                         return (DDI_FAILURE);
15035                 }
15036         }

15038         do {
15039                 ap.a_target = MPTSAS_INVALID_DEVHDL;
15040                 ap.a_lun = 0;
15041                 ap.a_hba_tran = mpt->m_tran;
15042                 repluns_bp = scsi_alloc_consistent_buf(&ap,
15043                     (struct buf *)NULL, buf_len, B_READ, NULL_FUNC, NULL);
```



```
15044                 if (repluns_bp == NULL) {
15045                         retry++;
15046                         continue;
15047                 }
15048                 bzero(cdb, CDB_GROUP5);
15049                 cdb[0] = SCMD_REPORT_LUNS;
15050                 cdb[6] = (buf_len & 0xff000000) >> 24;
15051                 cdb[7] = (buf_len & 0x00ff0000) >> 16;
15052                 cdb[8] = (buf_len & 0x0000ff00) >> 8;
15053                 cdb[9] = (buf_len & 0x000000ff);

15055                 ret = mptsas_send_scsi_cmd(mpt, &ap, ptgt, &cdb[0], CDB_GROUP5,
15056                     repluns_bp, NULL);
15057                 if (ret != DDI_SUCCESS) {
15058                         scsi_free_consistent_buf(repluns_bp);
15059                         retry++;
15060                         continue;
15061                 }
15062                 lun_list_len = BE_32(*(int *)((void *)(
15063                     repluns_bp->b_un.b_addr)));
15064                 if (buf_len >= lun_list_len + 8) {
15065                         ret = DDI_SUCCESS;
15066                         break;
15067                 }
15068                 scsi_free_consistent_buf(repluns_bp);
15069                 buf_len = lun_list_len + 8;

15071         } while (retry < 3);

15073         if (ret != DDI_SUCCESS)
15074                 return (ret);
15075         buffer = (char *)repluns_bp->b_un.b_addr;
15076         /*
15077          * find out the number of luns returned by the SCSI ReportLun call
15078          * and allocate buffer space
15079          */
15080         lun_total = lun_list_len / MPTSAS_SCSI_REPORTLUNS_ADDRESS_SIZE;
15081         saved_repluns = kmem_zalloc(sizeof (uint16_t) * lun_total, KM_SLEEP);
15082         if (saved_repluns == NULL) {
15083                 scsi_free_consistent_buf(repluns_bp);
15084                 return (DDI_FAILURE);
15085         }
15086         for (lun_cnt = 0; lun_cnt < lun_total; lun_cnt++) {
15087                 if (mptsas_retrieve_lundata(lun_cnt, (uint8_t *)(buffer),
15088                     &lun_num, &lun_addr_type) != DDI_SUCCESS) {
15089                         continue;
15090                 }
15091                 saved_repluns[lun_cnt] = lun_num;
15092                 if (cdip = mptsas_find_child_addr(pdip, sas_wwn, lun_num))
15093                         ret = DDI_SUCCESS;
15094                 else
15095                         ret = mptsas_probe_lun(pdip, lun_num, &cdip,
15096                             ptgt);
15097                 if ((ret == DDI_SUCCESS) && (cdip != NULL)) {
15098                         (void) ndi_prop_remove(DDI_DEV_T_NONE, cdip,
15099                             MPTSAS_DEV_GONE);
15100                 }
15101         }
15102         mptsas_offline_missed_luns(pdip, saved_repluns, lun_total, ptgt);
15103         kmem_free(saved_repluns, sizeof (uint16_t) * lun_total);
15104         scsi_free_consistent_buf(repluns_bp);
15105         return (DDI_SUCCESS);
15106 }

15108 static int
15109 mptsas_config_raid(dev_info_t *pdip, uint16_t target, dev_info_t **dip)
```

```
15110 {
15111         int                     rval = DDI_FAILURE;
15112         struct scsi_inquiry     *sd_inq = NULL;
15113         mptsas_t                *mpt = DIP2MPT(pdip);
15114         mptsas_target_t         *ptgt = NULL;

15116         mutex_enter(&mpt->m_mutex);
15117         ptgt = refhash_linear_search(mpt->m_targets,
15118             mptsas_target_eval_devhdl, &target);
15119         mutex_exit(&mpt->m_mutex);
15120         if (ptgt == NULL) {
15121                 mptsas_log(mpt, CE_WARN, "Volume with VolDevHandle of 0x%x "
15122                     "not found.", target);
15123                 return (rval);
15124         }

15126         sd_inq = (struct scsi_inquiry *)kmem_alloc(SUN_INQSIZE, KM_SLEEP);
15127         rval = mptsas_inquiry(mpt, ptgt, 0, 0, (uchar_t *)sd_inq,
15128             SUN_INQSIZE, 0, (uchar_t)0);

15130         if ((rval == DDI_SUCCESS) && MPTSAS_VALID_LUN(sd_inq)) {
15131                 rval = mptsas_create_phys_lun(pdip, sd_inq, NULL, dip, ptgt,
15132                     0);
15133         } else {
15134                 rval = DDI_FAILURE;
15135         }

15137         kmem_free(sd_inq, SUN_INQSIZE);
15138         return (rval);
15139 }

15141 /*
15142  * configure all RAID volumes for virtual iport
15143  */
15144 static void
15145 mptsas_config_all_viport(dev_info_t *pdip)
15146 {
15147         mptsas_t        *mpt = DIP2MPT(pdip);
15148         int             config, vol;
15149         int             target;
15150         dev_info_t      *lundip = NULL;

15152         /*
15153          * Get latest RAID info and search for any Volume DevHandles.  If any
15154          * are found, configure the volume.
15155          */
15156         mutex_enter(&mpt->m_mutex);
15157         for (config = 0; config < mpt->m_num_raid_configs; config++) {
15158                 for (vol = 0; vol < MPTSAS_MAX_RAIDVOLS; vol++) {
15159                         if (mpt->m_raidconfig[config].m_raidvol[vol].m_israid
15160                             == 1) {
15161                                 target = mpt->m_raidconfig[config].
15162                                     m_raidvol[vol].m_raidhandle;
15163                                 mutex_exit(&mpt->m_mutex);
15164                                 (void) mptsas_config_raid(pdip, target,
15165                                     &lundip);
15166                                 mutex_enter(&mpt->m_mutex);
15167                         }
15168                 }
15169         }
15170         mutex_exit(&mpt->m_mutex);
15171 }

15173 static void
15174 mptsas_offline_missed_luns(dev_info_t *pdip, uint16_t *repluns,
15175     int lun_cnt, mptsas_target_t *ptgt)
```



```
15176 {
15177         dev_info_t      *child = NULL, *savechild = NULL;
15178         mdi_pathinfo_t  *pip = NULL, *savepip = NULL;
15179         uint64_t        sas_wwn, wwid;
15180         uint8_t         phy;
15181         int             lun;
15182         int             i;
15183         int             find;
15184         char            *addr;
15185         char            *nodename;
15186         mptsas_t        *mpt = DIP2MPT(pdip);

15188         mutex_enter(&mpt->m_mutex);
15189         wwid = ptgt->m_addr.mta_wwn;
15190         mutex_exit(&mpt->m_mutex);

15192         child = ddi_get_child(pdip);
15193         while (child) {
15194                 find = 0;
15195                 savechild = child;
15196                 child = ddi_get_next_sibling(child);

15198                 nodename = ddi_node_name(savechild);
15199                 if (strcmp(nodename, "smp") == 0) {
15200                         continue;
15201                 }

15203                 addr = ddi_get_name_addr(savechild);
15204                 if (addr == NULL) {
15205                         continue;
15206                 }

15208                 if (mptsas_parse_address(addr, &sas_wwn, &phy, &lun) !=
15209                     DDI_SUCCESS) {
15210                         continue;
15211                 }

15213                 if (wwid == sas_wwn) {
15214                         for (i = 0; i < lun_cnt; i++) {
15215                                 if (repluns[i] == lun) {
15216                                         find = 1;
15217                                         break;
15218                                 }
15219                         }
15220                 } else {
15221                         continue;
15222                 }
15223                 if (find == 0) {
15224                         /*
15225                          * The lun has not been there already
15226                          */
15227                         (void) mptsas_offline_lun(pdip, savechild, NULL,
15228                             NDI_DEVI_REMOVE);
15229                 }
15230         }

15232         pip = mdi_get_next_client_path(pdip, NULL);
15233         while (pip) {
15234                 find = 0;
15235                 savepip = pip;
15236                 addr = MDI_PI(pip)->pi_addr;

15238                 pip = mdi_get_next_client_path(pdip, pip);

15240                 if (addr == NULL) {
15241                         continue;
```

```
15242                 }
15243
15244                 if (mptsas_parse_address(addr, &sas_wwn, &phy,
15245                     &lun) != DDI_SUCCESS) {
15246                         continue;
15247                 }
15248
15249                 if (sas_wwn == wwid) {
15250                         for (i = 0; i < lun_cnt; i++) {
15251                                 if (repluns[i] == lun) {
15252                                         find = 1;
15253                                         break;
15254                                 }
15255                         }
15256                 } else {
15257                         continue;
15258                 }
15259
15260                 if (find == 0) {
15261                         /*
15262                          * The lun has not been there already
15263                          */
15264                         (void) mptsas_offline_lun(pdip, NULL, savepip,
15265                             NDI_DEVI_REMOVE);
15266                 }
15267         }
15268 }
15269
15270 void
15271 mptsas_update_hashtab(struct mptsas *mpt)
15272 {
15273         uint32_t        page_address;
15274         int             rval = 0;
15275         uint16_t        dev_handle;
15276         mptsas_target_t *ptgt = NULL;
15277         mptsas_smp_t    smp_node;
15278
15279         /*
15280          * Get latest RAID info.
15281          */
15282         (void) mptsas_get_raid_info(mpt);
15283
15284         dev_handle = mpt->m_smp_devhdl;
15285         for (; mpt->m_done_traverse_smp == 0; ) {
15286                 page_address = (MPI2_SAS_EXPAND_PGAD_FORM_GET_NEXT_HNDL &
15287                     MPI2_SAS_EXPAND_PGAD_FORM_MASK) | (uint32_t)dev_handle;
15288                 if (mptsas_get_sas_expander_page0(mpt, page_address, &smp_node)
15289                     != DDI_SUCCESS) {
15290                         break;
15291                 }
15292                 mpt->m_smp_devhdl = dev_handle = smp_node.m_devhdl;
15293                 (void) mptsas_smp_alloc(mpt, &smp_node);
15294         }
15295
15296         /*
15297          * Config target devices
15298          */
15299         dev_handle = mpt->m_dev_handle;
15300
15301         /*
15302          * Do loop to get sas device page 0 by GetNextHandle till the
15303          * the last handle. If the sas device is a SATA/SSP target,
15304          * we try to config it.
15305          */
15306         for (; mpt->m_done_traverse_dev == 0; ) {
15307                 ptgt = NULL;
```



```
15308                 page_address =
15309                     (MPI2_SAS_DEVICE_PGAD_FORM_GET_NEXT_HANDLE &
15310                     MPI2_SAS_DEVICE_PGAD_FORM_MASK) |
15311                     (uint32_t)dev_handle;
15312                 rval = mptsas_get_target_device_info(mpt, page_address,
15313                     &dev_handle, &ptgt);
15314                 if ((rval == DEV_INFO_FAIL_PAGE0) ||
15315                     (rval == DEV_INFO_FAIL_ALLOC)) {
15316                         break;
15317                 }
15318
15319                 mpt->m_dev_handle = dev_handle;
15320         }
15321
15322 }
15323
15324 void
15325 mptsas_update_driver_data(struct mptsas *mpt)
15326 {
15327         mptsas_target_t *tp;
15328         mptsas_smp_t *sp;
15329
15330         ASSERT(MUTEX_HELD(&mpt->m_mutex));
15331
15332         /*
15333          * TODO after hard reset, update the driver data structures
15334          * 1. update port/phymask mapping table mpt->m_phy_info
15335          * 2. invalid all the entries in hash table
15336          *    m_devhdl = 0xffff and m_deviceinfo = 0
15337          * 3. call sas_device_page/expander_page to update hash table
15338          */
15339         mptsas_update_phymask(mpt);
15340         /*
15341          * Invalid the existing entries
15342          *
15343          * XXX - It seems like we should just delete everything here.  We are
15344          * holding the lock and are about to refresh all the targets in both
15345          * hashes anyway.  Given the path we're in, what outstanding async
15346          * event could possibly be trying to reference one of these things
15347          * without taking the lock, and how would that be useful anyway?
15348          */
15349         for (tp = refhash_first(mpt->m_targets); tp != NULL;
15350             tp = refhash_next(mpt->m_targets, tp)) {
15351                 tp->m_devhdl = MPTSAS_INVALID_DEVHDL;
15352                 tp->m_deviceinfo = 0;
15353                 tp->m_dr_flag = MPTSAS_DR_INACTIVE;
15354         }
15355         for (sp = refhash_first(mpt->m_smp_targets); sp != NULL;
15356             sp = refhash_next(mpt->m_smp_targets, sp)) {
15357                 sp->m_devhdl = MPTSAS_INVALID_DEVHDL;
15358                 sp->m_deviceinfo = 0;
15359         }
15360         mpt->m_done_traverse_dev = 0;
15361         mpt->m_done_traverse_smp = 0;
15362         mpt->m_dev_handle = mpt->m_smp_devhdl = MPTSAS_INVALID_DEVHDL;
15363         mptsas_update_hashtab(mpt);
15364 }
15365
15366 static void
15367 mptsas_config_all(dev_info_t *pdip)
15368 {
15369         dev_info_t      *smpdip = NULL;
15370         mptsas_t        *mpt = DIP2MPT(pdip);
15371         int             phymask = 0;
15372         mptsas_phymask_t phy_mask;
15373         mptsas_target_t *ptgt = NULL;
```

```
15374         mptsas_smp_t    *psmp;

15376         /*
15377          * Get the phymask associated to the iport
15378          */
15379         phymask = ddi_prop_get_int(DDI_DEV_T_ANY, pdip, 0,
15380             "phymask", 0);

15382         /*
15383          * Enumerate RAID volumes here (phymask == 0).
15384          */
15385         if (phymask == 0) {
15386                 mptsas_config_all_viport(pdip);
15387                 return;
15388         }

15390         mutex_enter(&mpt->m_mutex);

15392         if (!mpt->m_done_traverse_dev || !mpt->m_done_traverse_smp) {
15393                 mptsas_update_hashtab(mpt);
15394         }

15396         for (psmp = refhash_first(mpt->m_smp_targets); psmp != NULL;
15397             psmp = refhash_next(mpt->m_smp_targets, psmp)) {
15398                 phy_mask = psmp->m_addr.mta_phymask;
15399                 if (phy_mask == phymask) {
15400                         smpdip = NULL;
15401                         mutex_exit(&mpt->m_mutex);
15402                         (void) mptsas_online_smp(pdip, psmp, &smpdip);
15403                         mutex_enter(&mpt->m_mutex);
15404                 }
15405         }

15407         for (ptgt = refhash_first(mpt->m_targets); ptgt != NULL;
15408             ptgt = refhash_next(mpt->m_targets, ptgt)) {
15409                 phy_mask = ptgt->m_addr.mta_phymask;
15410                 if (phy_mask == phymask) {
15411                         mutex_exit(&mpt->m_mutex);
15412                         (void) mptsas_config_target(pdip, ptgt);
15413                         mutex_enter(&mpt->m_mutex);
15414                 }
15415         }
15416         mutex_exit(&mpt->m_mutex);
15417 }

15419 static int
15420 mptsas_config_target(dev_info_t *pdip, mptsas_target_t *ptgt)
15421 {
15422         int             rval = DDI_FAILURE;
15423         dev_info_t      *tdip;

15425         rval = mptsas_config_luns(pdip, ptgt);
15426         if (rval != DDI_SUCCESS) {
15427                 /*
15428                  * The return value means the SCMD_REPORT_LUNS
15429                  * did not execute successfully. The target maybe
15430                  * doesn't support such command.
15431                  */
15432                 rval = mptsas_probe_lun(pdip, 0, &tdip, ptgt);
15433         }
15434         return (rval);
15435 }

15437 /*
15438  * Return fail if not all the childs/paths are freed.
15439  * if there is any path under the HBA, the return value will be always fail
```



```
15440  * because we didn't call mdi_pi_free for path
15441  */
15442 static int
15443 mptsas_offline_targetdev(dev_info_t *pdip, char *name)
9970 mptsas_offline_target(dev_info_t *pdip, char *name)
15444 {
15445         dev_info_t              *child = NULL, *prechild = NULL;
15446         mdi_pathinfo_t          *pip = NULL, *savepip = NULL;
15447         int                     tmp_rval, rval = DDI_SUCCESS;
15448         char                    *addr, *cp;
15449         size_t                  s;
15450         mptsas_t                *mpt = DIP2MPT(pdip);

15452         child = ddi_get_child(pdip);
15453         while (child) {
15454                 addr = ddi_get_name_addr(child);
15455                 prechild = child;
15456                 child = ddi_get_next_sibling(child);

15458                 if (addr == NULL) {
15459                         continue;
15460                 }
15461                 if ((cp = strchr(addr, ',')) == NULL) {
15462                         continue;
15463                 }

15465                 s = (uintptr_t)cp - (uintptr_t)addr;

15467                 if (strncmp(addr, name, s) != 0) {
15468                         continue;
15469                 }

15471                 tmp_rval = mptsas_offline_lun(pdip, prechild, NULL,
15472                     NDI_DEVI_REMOVE);
15473                 if (tmp_rval != DDI_SUCCESS) {
15474                         rval = DDI_FAILURE;
15475                         if (ndi_prop_create_boolean(DDI_DEV_T_NONE,
15476                             prechild, MPTSAS_DEV_GONE) !=
15477                             DDI_PROP_SUCCESS) {
15478                                 mptsas_log(mpt, CE_WARN,
10005                                 mptsas_log(mpt, CE_WARN, "mptsas driver "
15479                                     "unable to create property for "
15480                                     "SAS %s (MPTSAS_DEV_GONE)", addr);
15481                         }
15482                 }
15483         }

15485         pip = mdi_get_next_client_path(pdip, NULL);
15486         while (pip) {
15487                 addr = MDI_PI(pip)->pi_addr;
15488                 savepip = pip;
15489                 pip = mdi_get_next_client_path(pdip, pip);
15490                 if (addr == NULL) {
15491                         continue;
15492                 }

15494                 if ((cp = strchr(addr, ',')) == NULL) {
15495                         continue;
15496                 }

15498                 s = (uintptr_t)cp - (uintptr_t)addr;

15500                 if (strncmp(addr, name, s) != 0) {
15501                         continue;
15502                 }
```

```
15504                 (void) mptsas_offline_lun(pdip, NULL, savepip,
15505                     NDI_DEVI_REMOVE);
15506                 /*
15507                  * driver will not invoke mdi_pi_free, so path will not
15508                  * be freed forever, return DDI_FAILURE.
15509                  */
15510                 rval = DDI_FAILURE;
15511         }
15512         return (rval);
15513 }
_____unchanged_portion_omitted_

15842 static int
15843 mptsas_create_virt_lun(dev_info_t *pdip, struct scsi_inquiry *inq, char *guid,
15844     dev_info_t **lun_dip, mdi_pathinfo_t **pip, mptsas_target_t *ptgt, int lun)
15845 {
15846         int                     target;
15847         char                    *nodename = NULL;
15848         char                    **compatible = NULL;
15849         int                     ncompatible     = 0;
15850         int                     mdi_rtn = MDI_FAILURE;
15851         int                     rval = DDI_FAILURE;
15852         char                    *old_guid = NULL;
15853         mptsas_t                *mpt = DIP2MPT(pdip);
15854         char                    *lun_addr = NULL;
15855         char                    wwn_str[MPTSAS_WWN_STRLEN];
10382         char                    *wwn_str = NULL;
10383         char                    *attached_wwn_str = NULL;
15856         char                    *component = NULL;
15857         uint8_t                 phy = 0xFF;
15858         uint64_t                sas_wwn;
15859         int64_t                 lun64 = 0;
15860         uint32_t                devinfo;
15861         uint16_t                dev_hdl;
15862         uint16_t                pdev_hdl;
15863         uint64_t                dev_sas_wwn;
15864         uint64_t                pdev_sas_wwn;
15865         uint32_t                pdev_info;
15866         uint8_t                 physport;
15867         uint8_t                 phy_id;
15868         uint32_t                page_address;
15869         uint16_t                bay_num, enclosure, io_flags;
10397         uint16_t                bay_num, enclosure;
15870         char                    pdev_wwn_str[MPTSAS_WWN_STRLEN];
15871         uint32_t                dev_info;

15873         mutex_enter(&mpt->m_mutex);
15874         target = ptgt->m_devhdl;
15875         sas_wwn = ptgt->m_addr.mta_wwn;
15876         devinfo = ptgt->m_deviceinfo;
15877         phy = ptgt->m_phynum;
15878         mutex_exit(&mpt->m_mutex);

15880         if (sas_wwn) {
15881                 *pip = mptsas_find_path_addr(pdip, sas_wwn, lun);
15882         } else {
15883                 *pip = mptsas_find_path_phy(pdip, phy);
15884         }

15886         if (*pip != NULL) {
15887                 *lun_dip = MDI_PI(*pip)->pi_client->ct_dip;
15888                 ASSERT(*lun_dip != NULL);
15889                 if (ddi_prop_lookup_string(DDI_DEV_T_ANY, *lun_dip,
15890                     (DDI_PROP_DONTPASS | DDI_PROP_NOTPROM),
15891                     MDI_CLIENT_GUID_PROP, &old_guid) == DDI_SUCCESS) {
15892                         if (strncmp(guid, old_guid, strlen(guid)) == 0) {
```



```
15893                                 /*
15894                                  * Same path back online again.
15895                                  */
15896                                 (void) ddi_prop_free(old_guid);
15897                                 if ((!MDI_PI_IS_ONLINE(*pip)) &&
15898                                     (!MDI_PI_IS_STANDBY(*pip)) &&
15899                                     (ptgt->m_tgt_unconfigured == 0)) {
15900                                         rval = mdi_pi_online(*pip, 0);
15901                                         mutex_enter(&mpt->m_mutex);
15902                                         ptgt->m_led_status = 0;
15903                                         (void) mptsas_flush_led_status(mpt,
15904                                             ptgt);
15905                                         mutex_exit(&mpt->m_mutex);
15906                                 } else {
15907                                         rval = DDI_SUCCESS;
15908                                 }
15909                                 if (rval != DDI_SUCCESS) {
15910                                         mptsas_log(mpt, CE_WARN, "path:target: "
15911                                             "%x, lun:%x online failed!", target,
15912                                             lun);
15913                                         *pip = NULL;
15914                                         *lun_dip = NULL;
15915                                 }
15916                                 return (rval);
15917                         } else {
15918                                 /*
15919                                  * The GUID of the LUN has changed which maybe
15920                                  * because customer mapped another volume to the
15921                                  * same LUN.
15922                                  */
15923                                 mptsas_log(mpt, CE_WARN, "The GUID of the "
15924                                     "target:%x, lun:%x was changed, maybe "
15925                                     "because someone mapped another volume "
15926                                     "to the same LUN", target, lun);
15927                                 (void) ddi_prop_free(old_guid);
15928                                 if (!MDI_PI_IS_OFFLINE(*pip)) {
15929                                         rval = mdi_pi_offline(*pip, 0);
15930                                         if (rval != MDI_SUCCESS) {
15931                                                 mptsas_log(mpt, CE_WARN, "path:"
15932                                                     "target:%x, lun:%x offline "
15933                                                     "failed!", target, lun);
15934                                                 *pip = NULL;
15935                                                 *lun_dip = NULL;
15936                                                 return (DDI_FAILURE);
15937                                         }
15938                                 }
15939                                 if (mdi_pi_free(*pip, 0) != MDI_SUCCESS) {
15940                                         mptsas_log(mpt, CE_WARN, "path:target:"
15941                                             "%x, lun:%x free failed!", target,
15942                                             lun);
15943                                         *pip = NULL;
15944                                         *lun_dip = NULL;
15945                                         return (DDI_FAILURE);
15946                                 }
15947                         }
15948                 } else {
15949                         mptsas_log(mpt, CE_WARN, "Can't get client-guid "
15950                             "property for path:target:%x, lun:%x", target, lun);
15951                         *pip = NULL;
15952                         *lun_dip = NULL;
15953                         return (DDI_FAILURE);
15954                 }
15955         }
15956         scsi_hba_nodename_compatible_get(inq, NULL,
15957             inq->inq_dtype, NULL, &nodename, &compatible, &ncompatible);
```

```
15959         /*
15960          * if nodename can't be determined then print a message and skip it
15961          */
15962         if (nodename == NULL) {
15963                 mptsas_log(mpt, CE_WARN, "found no compatible "
10491                 mptsas_log(mpt, CE_WARN, "mptsas driver found no compatible "
15964                     "driver for target%d lun %d dtype:0x%02x", target, lun,
15965                     inq->inq_dtype);
15966                 return (DDI_FAILURE);
15967         }

10497         wwn_str = kmem_zalloc(MPTSAS_WWN_STRLEN, KM_SLEEP);
15969         /* The property is needed by MPAPI */
15970         (void) sprintf(wwn_str, "%016"PRIx64, sas_wwn);

15972         lun_addr = kmem_zalloc(SCSI_MAXNAMELEN, KM_SLEEP);
15973         if (guid) {
15974                 (void) sprintf(lun_addr, "w%s,%x", wwn_str, lun);
15975                 (void) sprintf(wwn_str, "w%016"PRIx64, sas_wwn);
15976         } else {
15977                 (void) sprintf(lun_addr, "p%x,%x", phy, lun);
15978                 (void) sprintf(wwn_str, "p%x", phy);
15979         }

15981         mdi_rtn = mdi_pi_alloc_compatible(pdip, nodename,
15982             guid, lun_addr, compatible, ncompatible,
15983             0, pip);
15984         if (mdi_rtn == MDI_SUCCESS) {

15986                 if (mdi_prop_update_string(*pip, MDI_GUID,
15987                     guid) != DDI_SUCCESS) {
15988                         mptsas_log(mpt, CE_WARN, "unable to "
10517                         mptsas_log(mpt, CE_WARN, "mptsas driver unable to "
15989                             "create prop for target %d lun %d (MDI_GUID)",
15990                             target, lun);
15991                         mdi_rtn = MDI_FAILURE;
15992                         goto virt_create_done;
15993                 }

15995                 if (mdi_prop_update_int(*pip, LUN_PROP,
15996                     lun) != DDI_SUCCESS) {
15997                         mptsas_log(mpt, CE_WARN, "unable to "
10526                         mptsas_log(mpt, CE_WARN, "mptsas driver unable to "
15998                             "create prop for target %d lun %d (LUN_PROP)",
15999                             target, lun);
16000                         mdi_rtn = MDI_FAILURE;
16001                         goto virt_create_done;
16002                 }
16003                 lun64 = (int64_t)lun;
16004                 if (mdi_prop_update_int64(*pip, LUN64_PROP,
16005                     lun64) != DDI_SUCCESS) {
16006                         mptsas_log(mpt, CE_WARN, "unable to "
10535                         mptsas_log(mpt, CE_WARN, "mptsas driver unable to "
16007                             "create prop for target %d (LUN64_PROP)",
16008                             target);
16009                         mdi_rtn = MDI_FAILURE;
16010                         goto virt_create_done;
16011                 }
16012                 if (mdi_prop_update_string_array(*pip, "compatible",
16013                     compatible, ncompatible) !=
16014                     DDI_PROP_SUCCESS) {
16015                         mptsas_log(mpt, CE_WARN, "unable to "
10544                         mptsas_log(mpt, CE_WARN, "mptsas driver unable to "
16016                             "create prop for target %d lun %d (COMPATIBLE)",
16017                             target, lun);
16018                         mdi_rtn = MDI_FAILURE;
```



```
16019                         goto virt_create_done;
16020                 }
16021                 if (sas_wwn && (mdi_prop_update_string(*pip,
16022                     SCSI_ADDR_PROP_TARGET_PORT, wwn_str) != DDI_PROP_SUCCESS)) {
16023                         mptsas_log(mpt, CE_WARN, "unable to "
10552                         mptsas_log(mpt, CE_WARN, "mptsas driver unable to "
16024                             "create prop for target %d lun %d "
16025                             "(target-port)", target, lun);
16026                         mdi_rtn = MDI_FAILURE;
16027                         goto virt_create_done;
16028                 } else if ((sas_wwn == 0) && (mdi_prop_update_int(*pip,
16029                     "sata-phy", phy) != DDI_PROP_SUCCESS)) {
16030                         /*
16031                          * Direct attached SATA device without DeviceName
16032                          */
16033                         mptsas_log(mpt, CE_WARN, "unable to "
10562                         mptsas_log(mpt, CE_WARN, "mptsas driver unable to "
16034                             "create prop for SAS target %d lun %d "
16035                             "(sata-phy)", target, lun);
16036                         mdi_rtn = MDI_FAILURE;
16037                         goto virt_create_done;
16038                 }
16039                 mutex_enter(&mpt->m_mutex);

16041                 page_address = (MPI2_SAS_DEVICE_PGAD_FORM_HANDLE &
16042                     MPI2_SAS_DEVICE_PGAD_FORM_MASK) |
16043                     (uint32_t)ptgt->m_devhdl;
16044                 rval = mptsas_get_sas_device_page0(mpt, page_address,
16045                     &dev_hdl, &dev_sas_wwn, &dev_info, &physport,
16046                     &phy_id, &pdev_hdl, &bay_num, &enclosure, &io_flags);
10575                     &phy_id, &pdev_hdl, &bay_num, &enclosure);
16047                 if (rval != DDI_SUCCESS) {
16048                         mutex_exit(&mpt->m_mutex);
16049                         mptsas_log(mpt, CE_WARN, "mptsas unable to get "
16050                             "parent device for handle %d", page_address);
16051                         mdi_rtn = MDI_FAILURE;
16052                         goto virt_create_done;
16053                 }

16055                 page_address = (MPI2_SAS_DEVICE_PGAD_FORM_HANDLE &
16056                     MPI2_SAS_DEVICE_PGAD_FORM_MASK) | (uint32_t)pdev_hdl;
16057                 rval = mptsas_get_sas_device_page0(mpt, page_address,
16058                     &dev_hdl, &pdev_sas_wwn, &pdev_info, &physport,
16059                     &phy_id, &pdev_hdl, &bay_num, &enclosure, &io_flags);
10588                     &phy_id, &pdev_hdl, &bay_num, &enclosure);
16060                 if (rval != DDI_SUCCESS) {
16061                         mutex_exit(&mpt->m_mutex);
16062                         mptsas_log(mpt, CE_WARN, "mptsas unable to get"
16063                             "device info for handle %d", page_address);
16064                         mdi_rtn = MDI_FAILURE;
16065                         goto virt_create_done;
16066                 }

16068                 mutex_exit(&mpt->m_mutex);

16070                 /*
16071                  * If this device direct attached to the controller
16072                  * set the attached-port to the base wwid
16073                  */
16074                 if ((ptgt->m_deviceinfo & DEVINFO_DIRECT_ATTACHED)
16075                     != DEVINFO_DIRECT_ATTACHED) {
16076                         (void) sprintf(pdev_wwn_str, "w%016"PRIx64,
16077                             pdev_sas_wwn);
16078                 } else {
16079                         /*
16080                          * Update the iport's attached-port to guid
```

```
16081                          */
16082                         if (sas_wwn == 0) {
16083                                 (void) sprintf(wwn_str, "p%x", phy);
16084                         } else {
16085                                 (void) sprintf(wwn_str, "w%016"PRIx64, sas_wwn);
16086                         }
16087                         if (ddi_prop_update_string(DDI_DEV_T_NONE,
16088                             pdip, SCSI_ADDR_PROP_ATTACHED_PORT, wwn_str) !=
16089                             DDI_PROP_SUCCESS) {
16090                                 mptsas_log(mpt, CE_WARN,
16091                                     "mptsas unable to create "
16092                                     "property for iport target-port"
16093                                     " %s (sas_wwn)",
16094                                     wwn_str);
16095                                 mdi_rtn = MDI_FAILURE;
16096                                 goto virt_create_done;
16097                         }

16099                         (void) sprintf(pdev_wwn_str, "w%016"PRIx64,
16100                             mpt->un.m_base_wwid);
16101                 }

16103                 if (mdi_prop_update_string(*pip,
16104                     SCSI_ADDR_PROP_ATTACHED_PORT, pdev_wwn_str) !=
16105                     DDI_PROP_SUCCESS) {
16106                         mptsas_log(mpt, CE_WARN, "unable to create "
10635                            mptsas_log(mpt, CE_WARN, "mptsas unable to create "
16107                             "property for iport attached-port %s (sas_wwn)",
16108                             pdev_wwn_str);
10637                            attached_wwn_str);
16109                         mdi_rtn = MDI_FAILURE;
16110                         goto virt_create_done;
16111                 }


16114                 if (inq->inq_dtype == 0) {
16115                         component = kmem_zalloc(MAXPATHLEN, KM_SLEEP);
16116                         /*
16117                          * set obp path for pathinfo
16118                          */
16119                         (void) snprintf(component, MAXPATHLEN,
16120                             "disk@%s", lun_addr);

16122                         if (mdi_pi_pathname_obp_set(*pip, component) !=
16123                             DDI_SUCCESS) {
16124                                 mptsas_log(mpt, CE_WARN,
10653                                    mptsas_log(mpt, CE_WARN, "mpt_sas driver "
16125                                     "unable to set obp-path for object %s",
16126                                     component);
16127                                 mdi_rtn = MDI_FAILURE;
16128                                 goto virt_create_done;
16129                         }
16130                 }

16132                 *lun_dip = MDI_PI(*pip)->pi_client->ct_dip;
16133                 if (devinfo & (MPI2_SAS_DEVICE_INFO_SATA_DEVICE |
16134                     MPI2_SAS_DEVICE_INFO_ATAPI_DEVICE)) {
16135                         if ((ndi_prop_update_int(DDI_DEV_T_NONE, *lun_dip,
16136                             "pm-capable", 1)) !=
16137                             DDI_PROP_SUCCESS) {
16138                                 mptsas_log(mpt, CE_WARN,
10667                                    mptsas_log(mpt, CE_WARN, "mptsas driver"
16139                                     "failed to create pm-capable "
16140                                     "property, target %d", target);
16141                                 mdi_rtn = MDI_FAILURE;
16142                                 goto virt_create_done;
```



```
16143                         }
16144                 }
16145                 /*
16146                  * Create the phy-num property
16147                  */
16148                 if (mdi_prop_update_int(*pip, "phy-num",
16149                     ptgt->m_phynum) != DDI_SUCCESS) {
16150                         mptsas_log(mpt, CE_WARN, "unable to "
10679                            mptsas_log(mpt, CE_WARN, "mptsas driver unable to "
16151                             "create phy-num property for target %d lun %d",
16152                             target, lun);
16153                         mdi_rtn = MDI_FAILURE;
16154                         goto virt_create_done;
16155                 }
16156                 NDBG20(("new path:%s onlining,", MDI_PI(*pip)->pi_addr));
16157                 mdi_rtn = mdi_pi_online(*pip, 0);
16158                 if (mdi_rtn == MDI_SUCCESS) {
16159                         mutex_enter(&mpt->m_mutex);
16160                         ptgt->m_led_status = 0;
16161                         (void) mptsas_flush_led_status(mpt, ptgt);
16162                         mutex_exit(&mpt->m_mutex);
16163                 }
16164                 if (mdi_rtn == MDI_NOT_SUPPORTED) {
16165                         mdi_rtn = MDI_FAILURE;
16166                 }
16167 virt_create_done:
16168                 if (*pip && mdi_rtn != MDI_SUCCESS) {
16169                         (void) mdi_pi_free(*pip, 0);
16170                         *pip = NULL;
16171                         *lun_dip = NULL;
16172                 }
16173         }

16175         scsi_hba_nodename_compatible_free(nodename, compatible);
16176         if (lun_addr != NULL) {
16177                 kmem_free(lun_addr, SCSI_MAXNAMELEN);
16178         }
10708         if (wwn_str != NULL) {
10709                 kmem_free(wwn_str, MPTSAS_WWN_STRLEN);
10710         }
16179         if (component != NULL) {
16180                 kmem_free(component, MAXPATHLEN);
16181         }

16183         return ((mdi_rtn == MDI_SUCCESS) ? DDI_SUCCESS : DDI_FAILURE);
16184 }

16186 static int
16187 mptsas_create_phys_lun(dev_info_t *pdip, struct scsi_inquiry *inq,
16188     char *guid, dev_info_t **lun_dip, mptsas_target_t *ptgt, int lun)
16189 {
16190         int                     target;
16191         int                     rval;
16192         int                     ndi_rtn = NDI_FAILURE;
16193         uint64_t                be_sas_wwn;
16194         char                    *nodename = NULL;
16195         char                    **compatible = NULL;
16196         int                     ncompatible = 0;
16197         int                     instance = 0;
16198         mptsas_t                *mpt = DIP2MPT(pdip);
16199         char                    wwn_str[MPTSAS_WWN_STRLEN];
16200         char                    component[MAXPATHLEN];
10731         char                    *wwn_str = NULL;
10732         char                    *component = NULL;
10733         char                    *attached_wwn_str = NULL;
16201         uint8_t                 phy = 0xFF;
```

```
16202         uint64_t                sas_wwn;
16203         uint32_t                devinfo;
16204         uint16_t                dev_hdl;
16205         uint16_t                pdev_hdl;
16206         uint64_t                pdev_sas_wwn;
16207         uint64_t                dev_sas_wwn;
16208         uint32_t                pdev_info;
16209         uint8_t                 physport;
16210         uint8_t                 phy_id;
16211         uint32_t                page_address;
16212         uint16_t                bay_num, enclosure, io_flags;
10745         uint16_t                bay_num, enclosure;
16213         char                    pdev_wwn_str[MPTSAS_WWN_STRLEN];
16214         uint32_t                dev_info;
16215         int64_t                 lun64 = 0;

16217         mutex_enter(&mpt->m_mutex);
16218         target = ptgt->m_devhdl;
16219         sas_wwn = ptgt->m_addr.mta_wwn;
16220         devinfo = ptgt->m_deviceinfo;
16221         phy = ptgt->m_phynum;
16222         mutex_exit(&mpt->m_mutex);

16224         /*
16225          * generate compatible property with binding-set "mpt"
16226          */
16227         scsi_hba_nodename_compatible_get(inq, NULL, inq->inq_dtype, NULL,
16228             &nodename, &compatible, &ncompatible);

16230         /*
16231          * if nodename can't be determined then print a message and skip it
16232          */
16233         if (nodename == NULL) {
16234                 mptsas_log(mpt, CE_WARN, "mptsas found no compatible driver "
16235                     "for target %d lun %d", target, lun);
16236                 return (DDI_FAILURE);
16237         }

16239         ndi_rtn = ndi_devi_alloc(pdip, nodename,
16240             DEVI_SID_NODEID, lun_dip);

16242         /*
16243          * if lun alloc success, set props
16244          */
16245         if (ndi_rtn == NDI_SUCCESS) {

16247                 if (ndi_prop_update_int(DDI_DEV_T_NONE,
16248                     *lun_dip, LUN_PROP, lun) !=
16249                     DDI_PROP_SUCCESS) {
16250                         mptsas_log(mpt, CE_WARN, "mptsas unable to create "
16251                             "property for target %d lun %d (LUN_PROP)",
16252                             target, lun);
16253                         ndi_rtn = NDI_FAILURE;
16254                         goto phys_create_done;
16255                 }

16257                 lun64 = (int64_t)lun;
16258                 if (ndi_prop_update_int64(DDI_DEV_T_NONE,
16259                     *lun_dip, LUN64_PROP, lun64) !=
16260                     DDI_PROP_SUCCESS) {
16261                         mptsas_log(mpt, CE_WARN, "mptsas unable to create "
16262                             "property for target %d lun64 %d (LUN64_PROP)",
16263                             target, lun);
16264                         ndi_rtn = NDI_FAILURE;
16265                         goto phys_create_done;
16266                 }
```



```
16267                 if (ndi_prop_update_string_array(DDI_DEV_T_NONE,
16268                     *lun_dip, "compatible", compatible, ncompatible)
16269                     != DDI_PROP_SUCCESS) {
16270                         mptsas_log(mpt, CE_WARN, "mptsas unable to create "
16271                             "property for target %d lun %d (COMPATIBLE)",
16272                             target, lun);
16273                         ndi_rtn = NDI_FAILURE;
16274                         goto phys_create_done;
16275                 }

16277                 /*
16278                  * We need the SAS WWN for non-multipath devices, so
16279                  * we'll use the same property as that multipathing
16280                  * devices need to present for MPAPI. If we don't have
16281                  * a WWN (e.g. parallel SCSI), don't create the prop.
16282                  */
10816                 wwn_str = kmem_zalloc(MPTSAS_WWN_STRLEN, KM_SLEEP);
16283                 (void) sprintf(wwn_str, "w%016"PRIx64, sas_wwn);
16284                 if (sas_wwn && ndi_prop_update_string(DDI_DEV_T_NONE,
16285                     *lun_dip, SCSI_ADDR_PROP_TARGET_PORT, wwn_str)
16286                     != DDI_PROP_SUCCESS) {
16287                         mptsas_log(mpt, CE_WARN, "mptsas unable to "
16288                             "create property for SAS target %d lun %d "
16289                             "(target-port)", target, lun);
16290                         ndi_rtn = NDI_FAILURE;
16291                         goto phys_create_done;
16292                 }

16294                 be_sas_wwn = BE_64(sas_wwn);
16295                 if (sas_wwn && ndi_prop_update_byte_array(
16296                     DDI_DEV_T_NONE, *lun_dip, "port-wwn",
16297                     (uchar_t *)&be_sas_wwn, 8) != DDI_PROP_SUCCESS) {
16298                         mptsas_log(mpt, CE_WARN, "mptsas unable to "
16299                             "create property for SAS target %d lun %d "
16300                             "(port-wwn)", target, lun);
16301                         ndi_rtn = NDI_FAILURE;
16302                         goto phys_create_done;
16303                 } else if ((sas_wwn == 0) && (ndi_prop_update_int(
16304                     DDI_DEV_T_NONE, *lun_dip, "sata-phy", phy) !=
16305                     DDI_PROP_SUCCESS)) {
16306                         /*
16307                          * Direct attached SATA device without DeviceName
16308                          */
16309                         mptsas_log(mpt, CE_WARN, "mptsas unable to "
16310                             "create property for SAS target %d lun %d "
16311                             "(sata-phy)", target, lun);
16312                         ndi_rtn = NDI_FAILURE;
16313                         goto phys_create_done;
16314                 }

16316                 if (ndi_prop_create_boolean(DDI_DEV_T_NONE,
16317                     *lun_dip, SAS_PROP) != DDI_PROP_SUCCESS) {
16318                         mptsas_log(mpt, CE_WARN, "mptsas unable to"
16319                             "create property for SAS target %d lun %d"
16320                             " (SAS_PROP)", target, lun);
16321                         ndi_rtn = NDI_FAILURE;
16322                         goto phys_create_done;
16323                 }
16324                 if (guid && (ndi_prop_update_string(DDI_DEV_T_NONE,
16325                     *lun_dip, NDI_GUID, guid) != DDI_SUCCESS)) {
16326                         mptsas_log(mpt, CE_WARN, "mptsas unable "
16327                             "to create guid property for target %d "
16328                             "lun %d", target, lun);
16329                         ndi_rtn = NDI_FAILURE;
16330                         goto phys_create_done;
16331                 }
```

```
16333                 /*
16334                  * The following code is to set properties for SM-HBA support,
16335                  * it doesn't apply to RAID volumes
16336                  */
16337                 if (ptgt->m_addr.mta_phymask == 0)
16338                         goto phys_raid_lun;

16340                 mutex_enter(&mpt->m_mutex);

16342                 page_address = (MPI2_SAS_DEVICE_PGAD_FORM_HANDLE &
16343                     MPI2_SAS_DEVICE_PGAD_FORM_MASK) |
16344                     (uint32_t)ptgt->m_devhdl;
16345                 rval = mptsas_get_sas_device_page0(mpt, page_address,
16346                     &dev_hdl, &dev_sas_wwn, &dev_info,
16347                     &physport, &phy_id, &pdev_hdl,
16348                     &bay_num, &enclosure, &io_flags);
10882                     &bay_num, &enclosure);
16349                 if (rval != DDI_SUCCESS) {
16350                         mutex_exit(&mpt->m_mutex);
16351                         mptsas_log(mpt, CE_WARN, "mptsas unable to get"
16352                             "parent device for handle %d.", page_address);
16353                         ndi_rtn = NDI_FAILURE;
16354                         goto phys_create_done;
16355                 }

16357                 page_address = (MPI2_SAS_DEVICE_PGAD_FORM_HANDLE &
16358                     MPI2_SAS_DEVICE_PGAD_FORM_MASK) | (uint32_t)pdev_hdl;
16359                 rval = mptsas_get_sas_device_page0(mpt, page_address,
16360                     &dev_hdl, &pdev_sas_wwn, &pdev_info, &physport,
16361                     &phy_id, &pdev_hdl, &bay_num, &enclosure, &io_flags);
10894                     &dev_hdl, &pdev_sas_wwn, &pdev_info,
10895                     &physport, &phy_id, &pdev_hdl, &bay_num, &enclosure);
16362                 if (rval != DDI_SUCCESS) {
16363                         mutex_exit(&mpt->m_mutex);
16364                         mptsas_log(mpt, CE_WARN, "mptsas unable to create "
16365                             "device for handle %d.", page_address);
16366                         ndi_rtn = NDI_FAILURE;
16367                         goto phys_create_done;
16368                 }

16370                 mutex_exit(&mpt->m_mutex);

16372                 /*
16373                  * If this device direct attached to the controller
16374                  * set the attached-port to the base wwid
16375                  */
16376                 if ((ptgt->m_deviceinfo & DEVINFO_DIRECT_ATTACHED)
16377                     != DEVINFO_DIRECT_ATTACHED) {
16378                         (void) sprintf(pdev_wwn_str, "w%016"PRIx64,
16379                             pdev_sas_wwn);
16380                 } else {
16381                         /*
16382                          * Update the iport's attached-port to guid
16383                          */
16384                         if (sas_wwn == 0) {
16385                                 (void) sprintf(wwn_str, "p%x", phy);
16386                         } else {
16387                                 (void) sprintf(wwn_str, "w%016"PRIx64, sas_wwn);
16388                         }
16389                         if (ddi_prop_update_string(DDI_DEV_T_NONE,
16390                             pdip, SCSI_ADDR_PROP_ATTACHED_PORT, wwn_str) !=
16391                             DDI_PROP_SUCCESS) {
16392                                 mptsas_log(mpt, CE_WARN,
16393                                     "mptsas unable to create "
16394                                     "property for iport target-port"
```



```
16395                                     " %s (sas_wwn)",
16396                                     wwn_str);
16397                                 ndi_rtn = NDI_FAILURE;
16398                                 goto phys_create_done;
16399                         }

16401                         (void) sprintf(pdev_wwn_str, "w%016"PRIx64,
16402                             mpt->un.m_base_wwid);
16403                 }

16405                 if (ndi_prop_update_string(DDI_DEV_T_NONE,
16406                     *lun_dip, SCSI_ADDR_PROP_ATTACHED_PORT, pdev_wwn_str) !=
16407                     DDI_PROP_SUCCESS) {
16408                         mptsas_log(mpt, CE_WARN,
16409                             "mptsas unable to create "
16410                             "property for iport attached-port %s (sas_wwn)",
16411                             pdev_wwn_str);
10945                             attached_wwn_str);
16412                         ndi_rtn = NDI_FAILURE;
16413                         goto phys_create_done;
16414                 }

16416                 if (IS_SATA_DEVICE(dev_info)) {
16417                         if (ndi_prop_update_string(DDI_DEV_T_NONE,
16418                             *lun_dip, MPTSAS_VARIANT, "sata") !=
16419                             DDI_PROP_SUCCESS) {
16420                                 mptsas_log(mpt, CE_WARN,
16421                                     "mptsas unable to create "
16422                                     "property for device variant ");
16423                                 ndi_rtn = NDI_FAILURE;
16424                                 goto phys_create_done;
16425                         }
16426                 }

16428                 if (IS_ATAPI_DEVICE(dev_info)) {
16429                         if (ndi_prop_update_string(DDI_DEV_T_NONE,
16430                             *lun_dip, MPTSAS_VARIANT, "atapi") !=
16431                             DDI_PROP_SUCCESS) {
16432                                 mptsas_log(mpt, CE_WARN,
16433                                     "mptsas unable to create "
16434                                     "property for device variant ");
16435                                 ndi_rtn = NDI_FAILURE;
16436                                 goto phys_create_done;
16437                         }
16438                 }

16440 phys_raid_lun:
16441                 /*
16442                  * if this is a SAS controller, and the target is a SATA
16443                  * drive, set the 'pm-capable' property for sd and if on
16444                  * an OPL platform, also check if this is an ATAPI
16445                  * device.
16446                  */
16447                 instance = ddi_get_instance(mpt->m_dip);
16448                 if (devinfo & (MPI2_SAS_DEVICE_INFO_SATA_DEVICE |
16449                     MPI2_SAS_DEVICE_INFO_ATAPI_DEVICE)) {
16450                         NDBG2(("mptsas3%d: creating pm-capable property, "
16984                         NDBG2(("mptsas%d: creating pm-capable property, "
16451                             "target %d", instance, target));

16453                         if ((ndi_prop_update_int(DDI_DEV_T_NONE,
16454                             *lun_dip, "pm-capable", 1)) !=
16455                             DDI_PROP_SUCCESS) {
16456                                 mptsas_log(mpt, CE_WARN, "mptsas "
16457                                     "failed to create pm-capable "
16458                                     "property, target %d", target);
```

```
16459                                   ndi_rtn = NDI_FAILURE;
16460                                   goto phys_create_done;
16461                           }

16463                   }

16465                   if ((inq->inq_dtype == 0) || (inq->inq_dtype == 5)) {
16466                           /*
16467                            * add 'obp-path' properties for devinfo
16468                            */
16469                           bzero(wwn_str, sizeof (wwn_str));
16470                           (void) sprintf(wwn_str, "%016"PRIx64, sas_wwn);
11005                          component = kmem_zalloc(MAXPATHLEN, KM_SLEEP);
16471                           if (guid) {
16472                                   (void) snprintf(component, MAXPATHLEN,
16473                                       "disk@w%s,%x", wwn_str, lun);
16474                           } else {
16475                                   (void) snprintf(component, MAXPATHLEN,
16476                                       "disk@p%x,%x", phy, lun);
16477                           }
16478                           if (ddi_pathname_obp_set(*lun_dip, component)
16479                               != DDI_SUCCESS) {
16480                                   mptsas_log(mpt, CE_WARN, "mpt_sas driver "
16481                                       "unable to set obp-path for SAS "
16482                                       "object %s", component);
16483                                   ndi_rtn = NDI_FAILURE;
16484                                   goto phys_create_done;
16485                           }
16486                   }
16487                   /*
16488                    * Create the phy-num property for non-raid disk
16489                    */
16490                   if (ptgt->m_addr.mta_phymask != 0) {
16491                           if (ndi_prop_update_int(DDI_DEV_T_NONE,
16492                               *lun_dip, "phy-num", ptgt->m_phynum) !=
16493                               DDI_PROP_SUCCESS) {
16494                                   mptsas_log(mpt, CE_WARN,
11029                                  mptsas_log(mpt, CE_WARN, "mptsas driver "
16495                                       "failed to create phy-num property for "
16496                                       "target %d", target);
16497                                   ndi_rtn = NDI_FAILURE;
16498                                   goto phys_create_done;
16499                           }
16500                   }
16501 phys_create_done:
16502                   /*
16503                    * If props were setup ok, online the lun
16504                    */
16505                   if (ndi_rtn == NDI_SUCCESS) {
16506                           /*
16507                            * Try to online the new node
16508                            */
16509                           ndi_rtn = ndi_devi_online(*lun_dip, NDI_ONLINE_ATTACH);
16510                   }
16511                   if (ndi_rtn == NDI_SUCCESS) {
16512                           mutex_enter(&mpt->m_mutex);
16513                           ptgt->m_led_status = 0;
16514                           (void) mptsas_flush_led_status(mpt, ptgt);
16515                           mutex_exit(&mpt->m_mutex);
16516                   }

16518                   /*
16519                    * If success set rtn flag, else unwire alloc'd lun
16520                    */
16521                   if (ndi_rtn != NDI_SUCCESS) {
16522                           NDBG12(("unable to online "
```



```
11057                          NDBG12(("mptsas driver unable to online "
16523                                  "target %d lun %d", target, lun));
16524                          ndi_prop_remove_all(*lun_dip);
16525                          (void) ndi_devi_free(*lun_dip);
16526                          *lun_dip = NULL;
16527                  }
16528          }

16530          scsi_hba_nodename_compatible_free(nodename, compatible);

11067          if (wwn_str != NULL) {
11068                  kmem_free(wwn_str, MPTSAS_WWN_STRLEN);
11069          }
11070          if (component != NULL) {
11071                  kmem_free(component, MAXPATHLEN);
11072          }


16532          return ((ndi_rtn == NDI_SUCCESS) ? DDI_SUCCESS : DDI_FAILURE);
16533 }
______unchanged_portion_omitted_

16579 static int
16580 mptsas_online_smp(dev_info_t *pdip, mptsas_smp_t *smp_node,
16581     dev_info_t **smp_dip)
16582 {
16583          char            wwn_str[MPTSAS_WWN_STRLEN];
16584          char            attached_wwn_str[MPTSAS_WWN_STRLEN];
16585          int             ndi_rtn = NDI_FAILURE;
16586          int             rval = 0;
16587          mptsas_smp_t    dev_info;
16588          uint32_t        page_address;
16589          mptsas_t        *mpt = DIP2MPT(pdip);
16590          uint16_t        dev_hdl;
16591          uint64_t        sas_wwn;
16592          uint64_t        smp_sas_wwn;
16593          uint8_t         physport;
16594          uint8_t         phy_id;
16595          uint16_t        pdev_hdl;
16596          uint8_t         numphys = 0;
16597          uint16_t        i = 0;
16598          char            phymask[MPTSAS_MAX_PHYS];
16599          char            *iport = NULL;
16600          mptsas_phymask_t        phy_mask = 0;
16601          uint16_t        attached_devhdl;
16602          uint16_t        bay_num, enclosure, io_flags;
11145         uint16_t        bay_num, enclosure;

16604          (void) sprintf(wwn_str, "%"PRIx64, smp_node->m_addr.mta_wwn);

16606          /*
16607           * Probe smp device, prevent the node of removed device from being
16608           * configured succesfully
16609           */
16610          if (mptsas_probe_smp(pdip, smp_node->m_addr.mta_wwn) != NDI_SUCCESS) {
16611                  return (DDI_FAILURE);
16612          }

16614          if ((*smp_dip = mptsas_find_smp_child(pdip, wwn_str)) != NULL) {
16615                  return (DDI_SUCCESS);
16616          }

16618          ndi_rtn = ndi_devi_alloc(pdip, "smp", DEVI_SID_NODEID, smp_dip);

16620          /*
16621           * if lun alloc success, set props
```

```
16622          */
16623         if (ndi_rtn == NDI_SUCCESS) {
16624                 /*
16625                  * Set the flavor of the child to be SMP flavored
16626                  */
16627                 ndi_flavor_set(*smp_dip, SCSA_FLAVOR_SMP);

16629                 if (ndi_prop_update_string(DDI_DEV_T_NONE,
16630                     *smp_dip, SMP_WWN, wwn_str) !=
16631                     DDI_PROP_SUCCESS) {
16632                         mptsas_log(mpt, CE_WARN, "mptsas unable to create "
16633                             "property for smp device %s (sas_wwn)",
16634                             wwn_str);
16635                         ndi_rtn = NDI_FAILURE;
16636                         goto smp_create_done;
16637                 }
16638                 (void) sprintf(wwn_str, "w%"PRIx64, smp_node->m_addr.mta_wwn);
16639                 if (ndi_prop_update_string(DDI_DEV_T_NONE,
16640                     *smp_dip, SCSI_ADDR_PROP_TARGET_PORT, wwn_str) !=
16641                     DDI_PROP_SUCCESS) {
16642                         mptsas_log(mpt, CE_WARN, "mptsas unable to create "
16643                             "property for iport target-port %s (sas_wwn)",
16644                             wwn_str);
16645                         ndi_rtn = NDI_FAILURE;
16646                         goto smp_create_done;
16647                 }

16649                 mutex_enter(&mpt->m_mutex);

16651                 page_address = (MPI2_SAS_EXPAND_PGAD_FORM_HNDL &
16652                     MPI2_SAS_EXPAND_PGAD_FORM_MASK) | smp_node->m_devhdl;
16653                 rval = mptsas_get_sas_expander_page0(mpt, page_address,
16654                     &dev_info);
16655                 if (rval != DDI_SUCCESS) {
16656                         mutex_exit(&mpt->m_mutex);
16657                         mptsas_log(mpt, CE_WARN,
16658                             "mptsas unable to get expander "
16659                             "parent device info for %x", page_address);
16660                         ndi_rtn = NDI_FAILURE;
16661                         goto smp_create_done;
16662                 }

16664                 smp_node->m_pdevhdl = dev_info.m_pdevhdl;
16665                 page_address = (MPI2_SAS_DEVICE_PGAD_FORM_HANDLE &
16666                     MPI2_SAS_DEVICE_PGAD_FORM_MASK) |
16667                     (uint32_t)dev_info.m_pdevhdl;
16668                 rval = mptsas_get_sas_device_page0(mpt, page_address,
16669                     &dev_hdl, &sas_wwn, &smp_node->m_pdevinfo, &physport,
16670                     &phy_id, &pdev_hdl, &bay_num, &enclosure, &io_flags);
11212                    &dev_hdl, &sas_wwn, &smp_node->m_pdevinfo,
11213                    &physport, &phy_id, &pdev_hdl, &bay_num, &enclosure);
16671                 if (rval != DDI_SUCCESS) {
16672                         mutex_exit(&mpt->m_mutex);
16673                         mptsas_log(mpt, CE_WARN, "mptsas unable to get "
16674                             "device info for %x", page_address);
16675                         ndi_rtn = NDI_FAILURE;
16676                         goto smp_create_done;
16677                 }

16679                 page_address = (MPI2_SAS_DEVICE_PGAD_FORM_HANDLE &
16680                     MPI2_SAS_DEVICE_PGAD_FORM_MASK) |
16681                     (uint32_t)dev_info.m_devhdl;
16682                 rval = mptsas_get_sas_device_page0(mpt, page_address,
16683                     &dev_hdl, &smp_sas_wwn, &smp_node->m_deviceinfo,
16684                     &physport, &phy_id, &pdev_hdl, &bay_num, &enclosure,
16685                     &io_flags);
```



```
11227                    &physport, &phy_id, &pdev_hdl, &bay_num, &enclosure);
16686                 if (rval != DDI_SUCCESS) {
16687                         mutex_exit(&mpt->m_mutex);
16688                         mptsas_log(mpt, CE_WARN, "mptsas unable to get "
16689                             "device info for %x", page_address);
16690                         ndi_rtn = NDI_FAILURE;
16691                         goto smp_create_done;
16692                 }
16693                 mutex_exit(&mpt->m_mutex);

16695                 /*
16696                  * If this smp direct attached to the controller
16697                  * set the attached-port to the base wwid
16698                  */
16699                 if ((smp_node->m_deviceinfo & DEVINFO_DIRECT_ATTACHED)
16700                     != DEVINFO_DIRECT_ATTACHED) {
16701                         (void) sprintf(attached_wwn_str, "w%016"PRIx64,
16702                             sas_wwn);
16703                 } else {
16704                         (void) sprintf(attached_wwn_str, "w%016"PRIx64,
16705                             mpt->un.m_base_wwid);
16706                 }

16708                 if (ndi_prop_update_string(DDI_DEV_T_NONE,
16709                     *smp_dip, SCSI_ADDR_PROP_ATTACHED_PORT, attached_wwn_str) !=
16710                     DDI_PROP_SUCCESS) {
16711                         mptsas_log(mpt, CE_WARN, "mptsas unable to create "
16712                             "property for smp attached-port %s (sas_wwn)",
16713                             attached_wwn_str);
16714                         ndi_rtn = NDI_FAILURE;
16715                         goto smp_create_done;
16716                 }

16718                 if (ndi_prop_create_boolean(DDI_DEV_T_NONE,
16719                     *smp_dip, SMP_PROP) != DDI_PROP_SUCCESS) {
16720                         mptsas_log(mpt, CE_WARN, "mptsas unable to "
16721                             "create property for SMP %s (SMP_PROP) ",
16722                             wwn_str);
16723                         ndi_rtn = NDI_FAILURE;
16724                         goto smp_create_done;
16725                 }

16727                 /*
16728                  * check the smp to see whether it direct
16729                  * attached to the controller
16730                  */
16731                 if ((smp_node->m_deviceinfo & DEVINFO_DIRECT_ATTACHED)
16732                     != DEVINFO_DIRECT_ATTACHED) {
16733                         goto smp_create_done;
16734                 }
16735                 numphys = ddi_prop_get_int(DDI_DEV_T_ANY, pdip,
16736                     DDI_PROP_DONTPASS, MPTSAS_NUM_PHYS, -1);
16737                 if (numphys > 0) {
16738                         goto smp_create_done;
16739                 }
16740                 /*
16741                  * this iport is an old iport, we need to
16742                  * reconfig the props for it.
16743                  */
16744                 if (ddi_prop_update_int(DDI_DEV_T_NONE, pdip,
16745                     MPTSAS_VIRTUAL_PORT, 0) !=
16746                     DDI_PROP_SUCCESS) {
16747                         (void) ddi_prop_remove(DDI_DEV_T_NONE, pdip,
16748                             MPTSAS_VIRTUAL_PORT);
16749                         mptsas_log(mpt, CE_WARN, "mptsas virtual port "
16750                             "prop update failed");
```

```
16751                           goto smp_create_done;
16752                   }

16754                   mutex_enter(&mpt->m_mutex);
16755                   numphys = 0;
16756                   iport = ddi_get_name_addr(pdip);
16757                   for (i = 0; i < MPTSAS_MAX_PHYS; i++) {
16758                           bzero(phymask, sizeof (phymask));
16759                           (void) sprintf(phymask,
16760                               "%x", mpt->m_phy_info[i].phy_mask);
16761                           if (strcmp(phymask, iport) == 0) {
16762                                   phy_mask = mpt->m_phy_info[i].phy_mask;
16763                                   break;
16764                           }
16765                   }

16767                   for (i = 0; i < MPTSAS_MAX_PHYS; i++) {
16768                           if ((phy_mask >> i) & 0x01) {
16769                                   numphys++;
16770                           }
16771                   }
16772                   /*
16773                    * Update PHY info for smhba
16774                    */
16775                   if (mptsas_smhba_phy_init(mpt)) {
16776                           mutex_exit(&mpt->m_mutex);
16777                           mptsas_log(mpt, CE_WARN, "mptsas phy update "
16778                               "failed");
16779                           goto smp_create_done;
16780                   }
16781                   mutex_exit(&mpt->m_mutex);

16783                   mptsas_smhba_set_all_phy_props(mpt, pdip, numphys, phy_mask,
16784                       &attached_devhdl);

16786                   if (ddi_prop_update_int(DDI_DEV_T_NONE, pdip,
16787                       MPTSAS_NUM_PHYS, numphys) !=
16788                       DDI_PROP_SUCCESS) {
16789                           (void) ddi_prop_remove(DDI_DEV_T_NONE, pdip,
16790                               MPTSAS_NUM_PHYS);
16791                           mptsas_log(mpt, CE_WARN, "mptsas update "
16792                               "num phys props failed");
16793                           goto smp_create_done;
16794                   }
16795                   /*
16796                    * Add parent's props for SMHBA support
16797                    */
16798                   if (ddi_prop_update_string(DDI_DEV_T_NONE, pdip,
16799                       SCSI_ADDR_PROP_ATTACHED_PORT, wwn_str) !=
16800                       DDI_PROP_SUCCESS) {
16801                           (void) ddi_prop_remove(DDI_DEV_T_NONE, pdip,
16802                               SCSI_ADDR_PROP_ATTACHED_PORT);
16803                           mptsas_log(mpt, CE_WARN, "mptsas update iport"
16804                               "attached-port failed");
16805                           goto smp_create_done;
16806                   }

16808 smp_create_done:
16809                   /*
16810                    * If props were setup ok, online the lun
16811                    */
16812                   if (ndi_rtn == NDI_SUCCESS) {
16813                           /*
16814                            * Try to online the new node
16815                            */
16816                           ndi_rtn = ndi_devi_online(*smp_dip, NDI_ONLINE_ATTACH);
```



```
16817                   }

16819                   /*
16820                    * If success set rtn flag, else unwire alloc'd lun
16821                    */
16822                   if (ndi_rtn != NDI_SUCCESS) {
16823                           NDBG12(("mptsas unable to online "
16824                               "SMP target %s", wwn_str));
16825                           ndi_prop_remove_all(*smp_dip);
16826                           (void) ndi_devi_free(*smp_dip);
16827                   }
16828           }

16830           return ((ndi_rtn == NDI_SUCCESS) ? DDI_SUCCESS : DDI_FAILURE);
16831   }

16833   /* smp transport routine */
16834   static int mptsas_smp_start(struct smp_pkt *smp_pkt)
16835   {
16836           uint64_t                        wwn;
16837           Mpi2SmpPassthroughRequest_t     req;
16838           Mpi2SmpPassthroughReply_t       rep;
16839           uint8_t                         direction = 0;
11381           uint32_t                        direction = 0;
16840           mptsas_t                        *mpt;
16841           int                             ret;
16842           uint64_t                        tmp64;

16844           mpt = (mptsas_t *)smp_pkt->smp_pkt_address->
16845               smp_a_hba_tran->smp_tran_hba_private;

16847           bcopy(smp_pkt->smp_pkt_address->smp_a_wwn, &wwn, SAS_WWN_BYTE_SIZE);
16848           /*
16849            * Need to compose a SMP request message
16850            * and call mptsas_do_passthru() function
16851            */
16852           bzero(&req, sizeof (req));
16853           bzero(&rep, sizeof (rep));
16854           req.PassthroughFlags = 0;
16855           req.PhysicalPort = 0xff;
16856           req.ChainOffset = 0;
16857           req.Function = MPI2_FUNCTION_SMP_PASSTHROUGH;

16859           if ((smp_pkt->smp_pkt_reqsize & 0xffff0000ul) != 0) {
16860                   smp_pkt->smp_pkt_reason = ERANGE;
16861                   return (DDI_FAILURE);
16862           }
16863           req.RequestDataLength = LE_16((uint16_t)(smp_pkt->smp_pkt_reqsize - 4));

16865           req.MsgFlags = 0;
16866           tmp64 = LE_64(wwn);
16867           bcopy(&tmp64, &req.SASAddress, SAS_WWN_BYTE_SIZE);
16868           if (smp_pkt->smp_pkt_rspsize > 0) {
16869                   direction |= MPTSAS_PASS_THRU_DIRECTION_READ;
16870           }
16871           if (smp_pkt->smp_pkt_reqsize > 0) {
16872                   direction |= MPTSAS_PASS_THRU_DIRECTION_WRITE;
16873           }

16875           mutex_enter(&mpt->m_mutex);
16876           ret = mptsas_do_passthru(mpt, (uint8_t *)&req, (uint8_t *)&rep,
16877               (uint8_t *)smp_pkt->smp_pkt_rsp,
16878               offsetof(Mpi2SmpPassthroughRequest_t, SGL), sizeof (rep),
16879               smp_pkt->smp_pkt_rspsize - 4, direction,
16880               (uint8_t *)smp_pkt->smp_pkt_req, smp_pkt->smp_pkt_reqsize - 4,
16881               smp_pkt->smp_pkt_timeout, FKIOCTL);
```

```
16882         mutex_exit(&mpt->m_mutex);
16883         if (ret != 0) {
16884                 cmn_err(CE_WARN, "smp_start do passthru error %d", ret);
16885                 smp_pkt->smp_pkt_reason = (uchar_t)(ret);
16886                 return (DDI_FAILURE);
16887         }
16888         /* do passthrough success, check the smp status */
16889         if (LE_16(rep.IOCStatus) != MPI2_IOCSTATUS_SUCCESS) {
16890                 switch (LE_16(rep.IOCStatus)) {
16891                 case MPI2_IOCSTATUS_SCSI_DEVICE_NOT_THERE:
16892                         smp_pkt->smp_pkt_reason = ENODEV;
16893                         break;
16894                 case MPI2_IOCSTATUS_SAS_SMP_DATA_OVERRUN:
16895                         smp_pkt->smp_pkt_reason = EOVERFLOW;
16896                         break;
16897                 case MPI2_IOCSTATUS_SAS_SMP_REQUEST_FAILED:
16898                         smp_pkt->smp_pkt_reason = EIO;
16899                         break;
16900                 default:
16901                         mptsas_log(mpt, CE_NOTE, "smp_start: get unknown ioc"
16902                             "status:%x", LE_16(rep.IOCStatus));
16903                         smp_pkt->smp_pkt_reason = EIO;
16904                         break;
16905                 }
16906                 return (DDI_FAILURE);
16907         }
16908         if (rep.SASStatus != MPI2_SASSTATUS_SUCCESS) {
16909                 mptsas_log(mpt, CE_NOTE, "smp_start: get error SAS status:%x",
16910                     rep.SASStatus);
16911                 smp_pkt->smp_pkt_reason = EIO;
16912                 return (DDI_FAILURE);
16913         }

16915         return (DDI_SUCCESS);
16916 }
```

*unchanged_portion_omitted_*

```
17112 mptsas_target_t *
17113 mptsas_tgt_alloc(mptsas_t *mpt, uint16_t devhdl, uint64_t wwid,
17114     uint32_t devinfo, mptsas_phymask_t phymask, uint8_t phynum)
17115 {
17116         mptsas_target_t *tmp_tgt = NULL;
17117         mptsas_target_addr_t addr;

17119         addr.mta_wwn = wwid;
17120         addr.mta_phymask = phymask;
17121         tmp_tgt = refhash_lookup(mpt->m_targets, &addr);
17122         if (tmp_tgt != NULL) {
17123                 NDBG20(("Hash item already exist"));
17124                 tmp_tgt->m_deviceinfo = devinfo;
17125                 tmp_tgt->m_devhdl = devhdl;     /* XXX - duplicate? */
17126                 return (tmp_tgt);
17127         }
17128         tmp_tgt = kmem_zalloc(sizeof (struct mptsas_target), KM_SLEEP);
17129         if (tmp_tgt == NULL) {
17130                 cmn_err(CE_WARN, "Fatal, allocated tgt failed");
17131                 return (NULL);
17132         }
17133         tmp_tgt->m_devhdl = devhdl;
17134         tmp_tgt->m_addr.mta_wwn = wwid;
17135         tmp_tgt->m_deviceinfo = devinfo;
17136         tmp_tgt->m_addr.mta_phymask = phymask;
17137         tmp_tgt->m_phynum = phynum;
17138         /* Initialized the tgt structure */
17139         tmp_tgt->m_qfull_retries = QFULL_RETRIES;
17140         tmp_tgt->m_qfull_retry_interval =
```



```
17141             drv_usectohz(QFULL_RETRY_INTERVAL * 1000);
17142         tmp_tgt->m_t_throttle = MAX_THROTTLE;
17143         mutex_init(&tmp_tgt->m_t_mutex, NULL, MUTEX_DRIVER, NULL);
17144         TAILQ_INIT(&tmp_tgt->m_active_cmdq);
17145 #endif /* ! codereview */

17147         refhash_insert(mpt->m_targets, tmp_tgt);

17149         return (tmp_tgt);
17150 }

17152 static void
17153 mptsas_smp_target_copy(mptsas_smp_t *src, mptsas_smp_t *dst)
17154 {
17155         dst->m_devhdl = src->m_devhdl;
17156         dst->m_deviceinfo = src->m_deviceinfo;
17157         dst->m_pdevhdl = src->m_pdevhdl;
17158         dst->m_pdevinfo = src->m_pdevinfo;
17159 }

17161 #endif /* ! codereview */
17162 static mptsas_smp_t *
17163 mptsas_smp_alloc(mptsas_t *mpt, mptsas_smp_t *data)
17164 {
17165         mptsas_target_addr_t addr;
17166         mptsas_smp_t *ret_data;

17168         addr.mta_wwn = data->m_addr.mta_wwn;
17169         addr.mta_phymask = data->m_addr.mta_phymask;
17170         ret_data = refhash_lookup(mpt->m_smp_targets, &addr);
17171         /*
17172          * If there's already a matching SMP target, update its fields
17173          * in place.  Since the address is not changing, it's safe to do
17174          * this.  We cannot just bcopy() here because the structure we've
17175          * been given has invalid hash links.
17176          */
17177 #endif /* ! codereview */
17178         if (ret_data != NULL) {
17179                 mptsas_smp_target_copy(data, ret_data);
11685                 bcopy(data, ret_data, sizeof (mptsas_smp_t)); /* XXX - dupl */
17180                 return (ret_data);
17181         }

17183         ret_data = kmem_alloc(sizeof (mptsas_smp_t), KM_SLEEP);
17184         bcopy(data, ret_data, sizeof (mptsas_smp_t));
17185         refhash_insert(mpt->m_smp_targets, ret_data);
17186         return (ret_data);
17187 }
```

*unchanged_portion_omitted_*

```
17329 int
17330 mptsas_dma_addr_create(mptsas_t *mpt, ddi_dma_attr_t dma_attr,
17331     ddi_dma_handle_t *dma_hdp, ddi_acc_handle_t *acc_hdp, caddr_t *dma_memp,
17332     uint32_t alloc_size, ddi_dma_cookie_t *cookiep)
17333 {
17334         ddi_dma_cookie_t        new_cookie;
17335         size_t                  alloc_len;
17336         uint_t                  ncookie;

17338         if (cookiep == NULL)
17339                 cookiep = &new_cookie;

17341         if (ddi_dma_alloc_handle(mpt->m_dip, &dma_attr, DDI_DMA_SLEEP,
17342             NULL, dma_hdp) != DDI_SUCCESS) {
11849                 dma_hdp = NULL;
17343                 return (FALSE);
```

```
17344         }

17346         if (ddi_dma_mem_alloc(*dma_hdp, alloc_size, &mpt->m_dev_acc_attr,
17347             DDI_DMA_CONSISTENT, DDI_DMA_SLEEP, NULL, dma_memp, &alloc_len,
17348             acc_hdp) != DDI_SUCCESS) {
17349                 ddi_dma_free_handle(dma_hdp);
11857                 dma_hdp = NULL;
17350                 return (FALSE);
17351         }

17353         if (ddi_dma_addr_bind_handle(*dma_hdp, NULL, *dma_memp, alloc_len,
17354             (DDI_DMA_RDWR | DDI_DMA_CONSISTENT), DDI_DMA_SLEEP, NULL,
17355             cookiep, &ncookie) != DDI_DMA_MAPPED) {
17356                 (void) ddi_dma_mem_free(acc_hdp);
17357                 ddi_dma_free_handle(dma_hdp);
11866                 dma_hdp = NULL;
17358                 return (FALSE);
17359         }

17361         return (TRUE);
17362 }

17364 void
17365 mptsas_dma_addr_destroy(ddi_dma_handle_t *dma_hdp, ddi_acc_handle_t *acc_hdp)
17366 {
17367         if (*dma_hdp == NULL)
17368                 return;

17370         (void) ddi_dma_unbind_handle(*dma_hdp);
17371         (void) ddi_dma_mem_free(acc_hdp);
17372         ddi_dma_free_handle(dma_hdp);
11882         dma_hdp = NULL;
17373 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    4802 Thu Jun 12 17:42:16 2014
new/usr/src/uts/common/io/scsi/adapters/mpt_sas3/mptsas3_hash.c
hg changesets 607a5b46a793..b706c96317c3
Fix ncpus for early boot config
Purge the ack to the interrupt before exiting mptsas_intr()
Changes from code review
Changes to enable driver to compile.
Header paths, object lists, etc.
**********************************************************
```

```
/*
 * This file and its contents are supplied under the terms of the
 * Common Development and Distribution License ("CDDL"), version 1.0.
 * You may only use this file in accordance with the terms of version
 * 1.0 of the CDDL.
 *
 * A full copy of the text of the CDDL should have accompanied this
 * source.  A copy of the CDDL is also available via the Internet at
 * http://www.illumos.org/license/CDDL.
 */

/*
 * Copyright 2014 Joyent, Inc.  All rights reserved.
 * Copyright (c) 2014, Tegile Systems Inc. All rights reserved.
#endif /* ! codereview */
 */

#include <sys/scsi/adapters/mpt_sas3/mptsas3_hash.h>
#include <sys/scsi/adapters/mpt_sas/mptsas_hash.h>
#include <sys/sysmacros.h>
#include <sys/types.h>
#include <sys/kmem.h>
#include <sys/list.h>
#include <sys/ddi.h>

#ifdef lint
extern refhash_link_t *obj_to_link(refhash_t *, void *);
extern void *link_to_obj(refhash_t *, refhash_link_t *);
extern void *obj_to_tag(refhash_t *, void *);
#else
#define obj_to_link(_h, _o)     \
        ((refhash_link_t *)(((char *)(_o)) + (_h)->rh_link_off))
#define link_to_obj(_h, _l)     \
        ((void *)(((char *)(_l)) - (_h)->rh_link_off))
#define obj_to_tag(_h, _o)      \
        ((void *)(((char *)(_o)) + (_h)->rh_tag_off))
#endif

refhash_t *
refhash_create(uint_t bucket_count, refhash_hash_f hash,
    refhash_cmp_f cmp, refhash_dtor_f dtor, size_t obj_size, size_t link_off,
    size_t tag_off, int km_flags)
{
        refhash_t *hp;
        uint_t i;

        hp = kmem_alloc(sizeof (refhash_t), km_flags);
        if (hp == NULL)
                return (NULL);
        hp->rh_buckets = kmem_zalloc(bucket_count * sizeof (list_t), km_flags);
        if (hp->rh_buckets == NULL) {
                kmem_free(hp, sizeof (refhash_t));
                return (NULL);
        }
        hp->rh_bucket_count = bucket_count;
```



```
        for (i = 0; i < bucket_count; i++) {
                list_create(&hp->rh_buckets[i], sizeof (refhash_link_t),
                    offsetof(refhash_link_t, rhl_chain_link));
        }
        list_create(&hp->rh_objs, sizeof (refhash_link_t),
            offsetof(refhash_link_t, rhl_global_link));

        hp->rh_obj_size = obj_size;
        hp->rh_link_off = link_off;
        hp->rh_tag_off = tag_off;
        hp->rh_hash = hash;
        hp->rh_cmp = cmp;
        hp->rh_dtor = dtor;

        return (hp);
}
______unchanged_portion_omitted_
```

```
**********************************************************
   83656 Thu Jun 12 17:42:16 2014
new/usr/src/uts/common/io/scsi/adapters/mpt_sas3/mptsas3_impl.c
Update tx waitq's code.
Create 2 threads, divide the workflow and deliver
to the hardware from the threads.
Enable Fast Path for capable devices.
Lint and cstyle fixes.
Fix problem with running against 64bit msgaddr attributes for DMA.
Default is now to run like this.
Major rework of mutexes.
During normal operation do not grab m_mutex during interrupt.
Use reply post queues instead.
Fixes to some address arithmetic using 32bit values.
Merge fixes for "4403 mpt_sas panic when pulling a drive", commit f7d0d869a9ae78
Initial modifications using the code changes present between
the LSI source code for FreeBSD drivers. Specifically the changes
between from mpslsi-source-17.00.00.00 -> mpslsi-source-03.00.00.00.
This mainly involves using a different scatter/gather element in
frame setup.
Changes to enable driver to compile.
Header paths, object lists, etc.
**********************************************************
/*
 * 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) 2009, 2010, Oracle and/or its affiliates. All rights reserved.
 * Copyright 2012 Nexenta Systems, Inc. All rights reserved.
 * Copyright (c) 2014, Tegile Systems Inc. All rights reserved.
 * Copyright 2014 OmniTI Computer Consulting, Inc. All rights reserved.
#endif /* ! codereview */
 */

/*
 * Copyright (c) 2000 to 2010, LSI Corporation.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms of all code within
 * this file that is exclusively owned by LSI, with or without
 * modification, is permitted provided that, in addition to the CDDL 1.0
 * License requirements, the following conditions are met:
 *
 *    Neither the name of the author nor the names of its contributors may be
 *    used to endorse or promote products derived from this software without
 *    specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS 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
 * COPYRIGHT OWNER 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.
 */

/*
 * mptsas_impl - This file contains all the basic functions for communicating
 * to MPT based hardware.
 */

#if defined(lint) || defined(DEBUG)
#define MPTSAS_DEBUG
#endif

/*
 * standard header files
 */
#include <sys/note.h>
#include <sys/scsi/scsi.h>
#include <sys/pci.h>

#pragma pack(1)
#include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_type.h>
#include <sys/scsi/adapters/mpt_sas3/mpi/mpi2.h>
#include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_cnfg.h>
#include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_init.h>
#include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_ioc.h>
#include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_sas.h>
#include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_tool.h>
#include <sys/scsi/adapters/mpt_sas/mpi/mpi2_type.h>
#include <sys/scsi/adapters/mpt_sas/mpi/mpi2.h>
#include <sys/scsi/adapters/mpt_sas/mpi/mpi2_cnfg.h>
#include <sys/scsi/adapters/mpt_sas/mpi/mpi2_init.h>
#include <sys/scsi/adapters/mpt_sas/mpi/mpi2_ioc.h>
#include <sys/scsi/adapters/mpt_sas/mpi/mpi2_sas.h>
#include <sys/scsi/adapters/mpt_sas/mpi/mpi2_tool.h>
#pragma pack()

/*
 * private header files.
 */
#include <sys/scsi/adapters/mpt_sas3/mptsas3_var.h>
#include <sys/scsi/adapters/mpt_sas3/mptsas3_smhba.h>
#include <sys/scsi/adapters/mpt_sas/mptsas_var.h>
#include <sys/scsi/adapters/mpt_sas/mptsas_smhba.h>

/*
 * FMA header files.
 */
#include <sys/fm/io/ddi.h>

#if defined(MPTSAS_DEBUG)
extern uint32_t mptsas_debug_flags;
extern uint32_t mptsas_dbglog_imask;
#endif /* ! codereview */
#endif

/*
```

```
 *  prototypes
 */
static void mptsas_ioc_event_cmdq_add(mptsas_t *mpt, m_event_struct_t *cmd);
static void mptsas_ioc_event_cmdq_delete(mptsas_t *mpt, m_event_struct_t *cmd);
static m_event_struct_t *mptsas_ioc_event_find_by_cmd(mptsas_t *mpt,
    struct mptsas_cmd *cmd);

/*
 * add ioc evnet cmd into the queue
 */
static void
mptsas_ioc_event_cmdq_add(mptsas_t *mpt, m_event_struct_t *cmd)
{
        if ((cmd->m_event_linkp = mpt->m_ioc_event_cmdq) == NULL) {
                mpt->m_ioc_event_cmdtail = &cmd->m_event_linkp;
                mpt->m_ioc_event_cmdq = cmd;
        } else {
                cmd->m_event_linkp = NULL;
                *(mpt->m_ioc_event_cmdtail) = cmd;
                mpt->m_ioc_event_cmdtail = &cmd->m_event_linkp;
        }
}

/*
 * remove specified cmd from the ioc event queue
 */
static void
mptsas_ioc_event_cmdq_delete(mptsas_t *mpt, m_event_struct_t *cmd)
{
        m_event_struct_t        *prev = mpt->m_ioc_event_cmdq;
        if (prev == cmd) {
                if ((mpt->m_ioc_event_cmdq = cmd->m_event_linkp) == NULL) {
                        mpt->m_ioc_event_cmdtail = &mpt->m_ioc_event_cmdq;
                }
                cmd->m_event_linkp = NULL;
                return;
        }
        while (prev != NULL) {
                if (prev->m_event_linkp == cmd) {
                        prev->m_event_linkp = cmd->m_event_linkp;
                        if (cmd->m_event_linkp == NULL) {
                                mpt->m_ioc_event_cmdtail = &prev->m_event_linkp;
                        }

                        cmd->m_event_linkp = NULL;
                        return;
                }
                prev = prev->m_event_linkp;
        }
}

static m_event_struct_t *
mptsas_ioc_event_find_by_cmd(mptsas_t *mpt, struct mptsas_cmd *cmd)
{
        m_event_struct_t        *ioc_cmd = NULL;

        ioc_cmd = mpt->m_ioc_event_cmdq;
        while (ioc_cmd != NULL) {
                if (&(ioc_cmd->m_event_cmd) == cmd) {
                        return (ioc_cmd);
                }
                ioc_cmd = ioc_cmd->m_event_linkp;
        }
        ioc_cmd = NULL;
        return (ioc_cmd);
}
```



```
void
mptsas_destroy_ioc_event_cmd(mptsas_t *mpt)
{
        m_event_struct_t        *ioc_cmd = NULL;
        m_event_struct_t        *ioc_cmd_tmp = NULL;
        ioc_cmd = mpt->m_ioc_event_cmdq;

        /*
         * because the IOC event queue is resource of per instance for driver,
         * it's not only ACK event commands used it, but also some others used
         * it. We need destroy all ACK event commands when IOC reset, but can't
         * disturb others.So we use filter to clear the ACK event cmd in ioc
         * event queue, and other requests should be reserved, and they would
         * be free by its owner.
         */
        while (ioc_cmd != NULL) {
                if (ioc_cmd->m_event_cmd.cmd_flags & CFLAG_CMDACK) {
                        NDBG20(("destroy!! remove Ack Flag ioc_cmd\n"));
                        if ((mpt->m_ioc_event_cmdq =
                            ioc_cmd->m_event_linkp) == NULL)
                                mpt->m_ioc_event_cmdtail =
                                    &mpt->m_ioc_event_cmdq;
                        ioc_cmd_tmp = ioc_cmd;
                        ioc_cmd = ioc_cmd->m_event_linkp;
                        kmem_free(ioc_cmd_tmp, M_EVENT_STRUCT_SIZE);
                } else {
                        /*
                         * it's not ack cmd, so continue to check next one
                         */

                        NDBG20(("destroy!! it's not Ack Flag, continue\n"));
                        ioc_cmd = ioc_cmd->m_event_linkp;
                }

        }
}

void
mptsas_start_config_page_access(mptsas_t *mpt, mptsas_cmd_t *cmd)
{
        pMpi2ConfigRequest_t    request;
        pMpi2SGESimple64_t      sge;
        struct scsi_pkt         *pkt = cmd->cmd_pkt;
        mptsas_config_request_t *config = pkt->pkt_ha_private;
        uint8_t                 direction;
        uint32_t                length, flagslength;
        uint64_t                request_desc;
        uint32_t                length, flagslength, request_desc_low;

        ASSERT(mutex_owned(&mpt->m_mutex));

        /*
         * Point to the correct message and clear it as well as the global
         * config page memory.
         */
        request = (pMpi2ConfigRequest_t)(mpt->m_req_frame +
            (mpt->m_req_frame_size * cmd->cmd_slot));
        bzero(request, mpt->m_req_frame_size);

        /*
         * Form the request message.
         */
        ddi_put8(mpt->m_acc_req_frame_hdl, &request->Function,
            MPI2_FUNCTION_CONFIG);
        ddi_put8(mpt->m_acc_req_frame_hdl, &request->Action, config->action);
```

```
232         direction = MPI2_SGE_FLAGS_IOC_TO_HOST;
233         length = 0;
234         sge = (pMpi2SGESimple64_t)&request->PageBufferSGE;
235         if (config->action == MPI2_CONFIG_ACTION_PAGE_HEADER) {
236                 if (config->page_type > MPI2_CONFIG_PAGETYPE_MASK) {
237                         ddi_put8(mpt->m_acc_req_frame_hdl,
238                             &request->Header.PageType,
239                             MPI2_CONFIG_PAGETYPE_EXTENDED);
240                         ddi_put8(mpt->m_acc_req_frame_hdl,
241                             &request->ExtPageType, config->page_type);
242                 } else {
243                         ddi_put8(mpt->m_acc_req_frame_hdl,
244                             &request->Header.PageType, config->page_type);
245                 }
246         } else {
247                 ddi_put8(mpt->m_acc_req_frame_hdl, &request->ExtPageType,
248                     config->ext_page_type);
249                 ddi_put16(mpt->m_acc_req_frame_hdl, &request->ExtPageLength,
250                     config->ext_page_length);
251                 ddi_put8(mpt->m_acc_req_frame_hdl, &request->Header.PageType,
252                     config->page_type);
253                 ddi_put8(mpt->m_acc_req_frame_hdl, &request->Header.PageLength,
254                     config->page_length);
255                 ddi_put8(mpt->m_acc_req_frame_hdl,
256                     &request->Header.PageVersion, config->page_version);
257                 if ((config->page_type & MPI2_CONFIG_PAGETYPE_MASK) ==
258                     MPI2_CONFIG_PAGETYPE_EXTENDED) {
259                         length = config->ext_page_length * 4;
260                 } else {
261                         length = config->page_length * 4;
262                 }

264                 if (config->action == MPI2_CONFIG_ACTION_PAGE_WRITE_NVRAM) {
265                         direction = MPI2_SGE_FLAGS_HOST_TO_IOC;
266                 }
267                 ddi_put32(mpt->m_acc_req_frame_hdl, &sge->Address.Low,
268                     (uint32_t)(cmd->cmd_dma_addr&0xffffffffu));
101                     (uint32_t)cmd->cmd_dma_addr);
269                 ddi_put32(mpt->m_acc_req_frame_hdl, &sge->Address.High,
270                     (uint32_t)(cmd->cmd_dma_addr >> 32));
271         }
272         ddi_put8(mpt->m_acc_req_frame_hdl, &request->Header.PageNumber,
273             config->page_number);
274         ddi_put32(mpt->m_acc_req_frame_hdl, &request->PageAddress,
275             config->page_address);
276         flagslength = ((uint32_t)(MPI2_SGE_FLAGS_LAST_ELEMENT |
277             MPI2_SGE_FLAGS_END_OF_BUFFER |
278             MPI2_SGE_FLAGS_SIMPLE_ELEMENT |
279             MPI2_SGE_FLAGS_SYSTEM_ADDRESS |
280             MPI2_SGE_FLAGS_64_BIT_ADDRESSING |
281             direction |
282             MPI2_SGE_FLAGS_END_OF_LIST) << MPI2_SGE_FLAGS_SHIFT);
283         flagslength |= length;
284         ddi_put32(mpt->m_acc_req_frame_hdl, &sge->FlagsLength, flagslength);

286         (void) ddi_dma_sync(mpt->m_dma_req_frame_hdl, 0, 0,
287             DDI_DMA_SYNC_FORDEV);
288         request_desc = (cmd->cmd_slot << 16) +
121         request_desc_low = (cmd->cmd_slot << 16) +
289             MPI2_REQ_DESCRIPT_FLAGS_DEFAULT_TYPE;
290         cmd->cmd_rfm = NULL;
291         MPTSAS_START_CMD(mpt, request_desc);
124         MPTSAS_START_CMD(mpt, request_desc_low, 0);
292         if ((mptsas_check_dma_handle(mpt->m_dma_req_frame_hdl) !=
293             DDI_SUCCESS) ||
294             (mptsas_check_acc_handle(mpt->m_acc_req_frame_hdl) !=
```



```
295             DDI_SUCCESS)) {
296                 ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_UNAFFECTED);
297         }
298 }

300 int
301 mptsas_access_config_page(mptsas_t *mpt, uint8_t action, uint8_t page_type,
302     uint8_t page_number, uint32_t page_address, int (*callback) (mptsas_t *,
303     caddr_t, ddi_acc_handle_t, uint16_t, uint32_t, va_list), ...)
304 {
305         va_list                 ap;
306         ddi_dma_attr_t          attrs;
307         ddi_dma_cookie_t        cookie;
308         ddi_acc_handle_t        accessp;
309         size_t                  len = 0;
310         mptsas_config_request_t config;
311         int                     rval = DDI_SUCCESS, config_flags = 0;
312         mptsas_cmd_t            *cmd;
313         struct scsi_pkt         *pkt;
314         pMpi2ConfigReply_t      reply;
315         uint16_t                iocstatus = 0;
316         uint32_t                iocloginfo;
317         caddr_t                 page_memp;
318         boolean_t               free_dma = B_FALSE;
319 #endif /* ! codereview */

321         va_start(ap, callback);
322         ASSERT(mutex_owned(&mpt->m_mutex));

324         /*
325          * Get a command from the pool.
326          */
327         if ((rval = (mptsas_request_from_pool(mpt, &cmd, &pkt))) == -1) {
328                 mptsas_log(mpt, CE_NOTE, "command pool is full for config "
329                     "page request");
330                 rval = DDI_FAILURE;
331                 goto page_done;
332         }
333         config_flags |= MPTSAS_REQUEST_POOL_CMD;

335         bzero((caddr_t)cmd, sizeof (*cmd));
336         bzero((caddr_t)pkt, scsi_pkt_size());
337         bzero((caddr_t)&config, sizeof (config));

339         /*
340          * Save the data for this request to be used in the call to start the
341          * config header request.
342          */
343         config.action = MPI2_CONFIG_ACTION_PAGE_HEADER;
344         config.page_type = page_type;
345         config.page_number = page_number;
346         config.page_address = page_address;

348         /*
349          * Form a blank cmd/pkt to store the acknowledgement message
350          */
351         pkt->pkt_ha_private     = (opaque_t)&config;
352         pkt->pkt_flags          = FLAG_HEAD;
353         pkt->pkt_time           = 60;
354         cmd->cmd_pkt            = pkt;
355         cmd->cmd_flags          = CFLAG_CMDIOC | CFLAG_CONFIG;

357         /*
358          * Save the config header request message in a slot.
359          */
360         if (mptsas_save_cmd(mpt, cmd) == TRUE) {
```

```
361                 cmd->cmd_flags |= CFLAG_PREPARED;
362                 mptsas_start_config_page_access(mpt, cmd);
363         } else {
364                 mptsas_waitq_add(mpt, cmd);
365         }

367         /*
368          * If this is a request for a RAID info page, or any page called during
369          * the RAID info page request, poll because these config page requests
370          * are nested.  Poll to avoid data corruption due to one page's data
371          * overwriting the outer page request's data.  This can happen when
372          * the mutex is released in cv_wait.
373          */
374         if ((page_type == MPI2_CONFIG_EXTPAGETYPE_RAID_CONFIG) ||
375             (page_type == MPI2_CONFIG_PAGETYPE_RAID_VOLUME) ||
376             (page_type == MPI2_CONFIG_PAGETYPE_RAID_PHYSDISK)) {
377                 (void) mptsas_poll(mpt, cmd, pkt->pkt_time * 1000);
378         } else {
379                 while ((cmd->cmd_flags & CFLAG_FINISHED) == 0) {
380                         cv_wait(&mpt->m_config_cv, &mpt->m_mutex);
381                 }
382         }

384         /*
385          * Check if the header request completed without timing out
386          */
387         if (cmd->cmd_flags & CFLAG_TIMEOUT) {
388                 mptsas_log(mpt, CE_WARN, "config header request timeout");
389                 rval = DDI_FAILURE;
390                 goto page_done;
391         }

393         /*
394          * cmd_rfm points to the reply message if a reply was given.  Check the
395          * IOCStatus to make sure everything went OK with the header request.
396          */
397         if (cmd->cmd_rfm) {
398                 config_flags |= MPTSAS_ADDRESS_REPLY;
399                 (void) ddi_dma_sync(mpt->m_dma_reply_frame_hdl, 0, 0,
400                     DDI_DMA_SYNC_FORCPU);
401                 reply = (pMpi2ConfigReply_t)(mpt->m_reply_frame + (cmd->cmd_rfm
402                     - (mpt->m_reply_frame_dma_addr&0xffffffffu)));
151                 - mpt->m_reply_frame_dma_addr));
403                 config.page_type = ddi_get8(mpt->m_acc_reply_frame_hdl,
404                     &reply->Header.PageType);
405                 config.page_number = ddi_get8(mpt->m_acc_reply_frame_hdl,
406                     &reply->Header.PageNumber);
407                 config.page_length = ddi_get8(mpt->m_acc_reply_frame_hdl,
408                     &reply->Header.PageLength);
409                 config.page_version = ddi_get8(mpt->m_acc_reply_frame_hdl,
410                     &reply->Header.PageVersion);
411                 config.ext_page_type = ddi_get8(mpt->m_acc_reply_frame_hdl,
412                     &reply->ExtPageType);
413                 config.ext_page_length = ddi_get16(mpt->m_acc_reply_frame_hdl,
414                     &reply->ExtPageLength);

416                 iocstatus = ddi_get16(mpt->m_acc_reply_frame_hdl,
417                     &reply->IOCStatus);
418                 iocloginfo = ddi_get32(mpt->m_acc_reply_frame_hdl,
419                     &reply->IOCLogInfo);

421                 if (iocstatus) {
422                         NDBG13(("mptsas_access_config_page header: "
423                             "IOCStatus=0x%x, IOCLogInfo=0x%x", iocstatus,
424                             iocloginfo));
425                         rval = DDI_FAILURE;
```



```
426                         goto page_done;
427                 }

429                 if ((config.page_type & MPI2_CONFIG_PAGETYPE_MASK) ==
430                     MPI2_CONFIG_PAGETYPE_EXTENDED)
431                         len = (config.ext_page_length * 4);
432                 else
433                         len = (config.page_length * 4);

435         }

437         if (pkt->pkt_reason == CMD_RESET) {
438                 mptsas_log(mpt, CE_WARN, "ioc reset abort config header "
439                     "request");
440                 rval = DDI_FAILURE;
441                 goto page_done;
442         }

444         /*
445          * Put the reply frame back on the free queue, increment the free
446          * index, and write the new index to the free index register.  But only
447          * if this reply is an ADDRESS reply.
448          */
449         if (config_flags & MPTSAS_ADDRESS_REPLY) {
450                 ddi_put32(mpt->m_acc_free_queue_hdl,
451                     &((uint32_t *)(void *)mpt->m_free_queue)[mpt->m_free_index],
452                     cmd->cmd_rfm);
453                 (void) ddi_dma_sync(mpt->m_dma_free_queue_hdl, 0, 0,
454                     DDI_DMA_SYNC_FORDEV);
455                 if (++mpt->m_free_index == mpt->m_free_queue_depth) {
456                         mpt->m_free_index = 0;
457                 }
458                 ddi_put32(mpt->m_datap, &mpt->m_reg->ReplyFreeHostIndex,
459                     mpt->m_free_index);
460                 config_flags &= (~MPTSAS_ADDRESS_REPLY);
461         }

463         /*
464          * Allocate DMA buffer here.  Store the info regarding this buffer in
465          * the cmd struct so that it can be used for this specific command and
466          * de-allocated after the command completes.  The size of the reply
467          * will not be larger than the reply frame size.
468          */
469         attrs = mpt->m_msg_dma_attr;
470         attrs.dma_attr_sgllen = 1;
471         attrs.dma_attr_granular = (uint32_t)len;

473         if (mptsas_dma_addr_create(mpt, attrs,
474             &cmd->cmd_dmahandle, &accessp, &page_memp,
475             len, &cookie) == FALSE) {
476                 mptsas_log(mpt, CE_WARN,
477                     "mptsas_dma_addr_create(len=0x%x) failed", (int)len);
478                 rval = DDI_FAILURE;
479 #endif /* ! codereview */
480                 goto page_done;
481         }
482         /* NOW we can safely call mptsas_dma_addr_destroy(). */
483         free_dma = B_TRUE;

485 #endif /* ! codereview */
486         cmd->cmd_dma_addr = cookie.dmac_laddress;
487         bzero(page_memp, len);

489         /*
490          * Save the data for this request to be used in the call to start the
491          * config page read
```

```
492          */
493         config.action = action;
494         config.page_address = page_address;
496         /*
497          * Re-use the cmd that was used to get the header.  Reset some of the
498          * values.
499          */
500         bzero((caddr_t)pkt, scsi_pkt_size());
501         pkt->pkt_ha_private     = (opaque_t)&config;
502         pkt->pkt_flags          = FLAG_HEAD;
503         pkt->pkt_time           = 60;
504         cmd->cmd_flags          = CFLAG_PREPARED | CFLAG_CMDIOC | CFLAG_CONFIG;
506         /*
507          * Send the config page request.  cmd is re-used from header request.
508          */
509         mptsas_start_config_page_access(mpt, cmd);
511         /*
512          * If this is a request for a RAID info page, or any page called during
513          * the RAID info page request, poll because these config page requests
514          * are nested.  Poll to avoid data corruption due to one page's data
515          * overwriting the outer page request's data.  This can happen when
516          * the mutex is released in cv_wait.
517          */
518         if ((page_type == MPI2_CONFIG_EXTPAGETYPE_RAID_CONFIG) ||
519             (page_type == MPI2_CONFIG_PAGETYPE_RAID_VOLUME) ||
520             (page_type == MPI2_CONFIG_PAGETYPE_RAID_PHYSDISK)) {
521                 (void) mptsas_poll(mpt, cmd, pkt->pkt_time * 1000);
522         } else {
523                 while ((cmd->cmd_flags & CFLAG_FINISHED) == 0) {
524                         cv_wait(&mpt->m_config_cv, &mpt->m_mutex);
525                 }
526         }
528         /*
529          * Check if the request completed without timing out
530          */
531         if (cmd->cmd_flags & CFLAG_TIMEOUT) {
532                 mptsas_log(mpt, CE_WARN, "config page request timeout");
533                 rval = DDI_FAILURE;
534                 goto page_done;
535         }
537         /*
538          * cmd_rfm points to the reply message if a reply was given.  The reply
539          * frame and the config page are returned from this function in the
540          * param list.
541          */
542         if (cmd->cmd_rfm) {
543                 config_flags |= MPTSAS_ADDRESS_REPLY;
544                 (void) ddi_dma_sync(mpt->m_dma_reply_frame_hdl, 0, 0,
545                     DDI_DMA_SYNC_FORCPU);
546                 (void) ddi_dma_sync(cmd->cmd_dmahandle, 0, 0,
547                     DDI_DMA_SYNC_FORCPU);
548                 reply = (pMpi2ConfigReply_t)(mpt->m_reply_frame + (cmd->cmd_rfm
549                     - (mpt->m_reply_frame_dma_addr&0xffffffff)));
225                     - mpt->m_reply_frame_dma_addr));
550                 iocstatus = ddi_get16(mpt->m_acc_reply_frame_hdl,
551                     &reply->IOCStatus);
552                 iocstatus = MPTSAS_IOCSTATUS(iocstatus);
553                 iocloginfo = ddi_get32(mpt->m_acc_reply_frame_hdl,
554                     &reply->IOCLogInfo);
555         }
```



```
557         if (callback(mpt, page_memp, accessp, iocstatus, iocloginfo, ap)) {
558                 rval = DDI_FAILURE;
559                 goto page_done;
560         }
562         mptsas_fma_check(mpt, cmd);
563         /*
564          * Check the DMA/ACC handles and then free the DMA buffer.
565          */
566         if ((mptsas_check_dma_handle(cmd->cmd_dmahandle) != DDI_SUCCESS) ||
567             (mptsas_check_acc_handle(accessp) != DDI_SUCCESS)) {
568                 ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_UNAFFECTED);
569                 rval = DDI_FAILURE;
570         }
572         if (pkt->pkt_reason == CMD_TRAN_ERR) {
573                 mptsas_log(mpt, CE_WARN, "config fma error");
574                 rval = DDI_FAILURE;
575                 goto page_done;
576         }
577         if (pkt->pkt_reason == CMD_RESET) {
578                 mptsas_log(mpt, CE_WARN, "ioc reset abort config request");
579                 rval = DDI_FAILURE;
580                 goto page_done;
581         }
583 page_done:
584         va_end(ap);
585         /*
586          * Put the reply frame back on the free queue, increment the free
587          * index, and write the new index to the free index register.  But only
588          * if this reply is an ADDRESS reply.
589          */
590         if (config_flags & MPTSAS_ADDRESS_REPLY) {
591                 ddi_put32(mpt->m_acc_free_queue_hdl,
592                     &((uint32_t *)(void *)mpt->m_free_queue)[mpt->m_free_index],
593                     cmd->cmd_rfm);
594                 (void) ddi_dma_sync(mpt->m_dma_free_queue_hdl, 0, 0,
595                     DDI_DMA_SYNC_FORDEV);
596                 if (++mpt->m_free_index == mpt->m_free_queue_depth) {
597                         mpt->m_free_index = 0;
598                 }
599                 ddi_put32(mpt->m_datap, &mpt->m_reg->ReplyFreeHostIndex,
600                     mpt->m_free_index);
601         }
603         if (free_dma)
604 #endif /* ! codereview */
605                 mptsas_dma_addr_destroy(&cmd->cmd_dmahandle, &accessp);
607         if (cmd && (cmd->cmd_flags & CFLAG_PREPARED)) {
608                 mptsas_remove_cmd(mpt, cmd);
609                 config_flags &= (~MPTSAS_REQUEST_POOL_CMD);
610         }
611         if (config_flags & MPTSAS_REQUEST_POOL_CMD)
612                 mptsas_return_to_pool(mpt, cmd);
614         if (config_flags & MPTSAS_CMD_TIMEOUT) {
615                 mpt->m_softstate &= ~MPTSAS_SS_MSG_UNIT_RESET;
616                 if ((mptsas_restart_ioc(mpt)) == DDI_FAILURE) {
617                         mptsas_log(mpt, CE_WARN, "mptsas_restart_ioc failed");
618                 }
619         }
621         return (rval);
622 }
```

```
624 int
625 mptsas_send_config_request_msg(mptsas_t *mpt, uint8_t action, uint8_t pagetype,
626         uint32_t pageaddress, uint8_t pagenumber, uint8_t pageversion,
627         uint8_t pagelength, uint32_t SGEflagslength, uint64_t SGEaddress)
279         uint8_t pagelength, uint32_t SGEflagslength, uint32_t SGEaddress32)
628 {
629         pMpi2ConfigRequest_t    config;
630         int                     send_numbytes;

632         bzero(mpt->m_hshk_memp, sizeof (MPI2_CONFIG_REQUEST));
633         config = (pMpi2ConfigRequest_t)mpt->m_hshk_memp;
634         ddi_put8(mpt->m_hshk_acc_hdl, &config->Function, MPI2_FUNCTION_CONFIG);
635         ddi_put8(mpt->m_hshk_acc_hdl, &config->Action, action);
636         ddi_put8(mpt->m_hshk_acc_hdl, &config->Header.PageNumber, pagenumber);
637         ddi_put8(mpt->m_hshk_acc_hdl, &config->Header.PageType, pagetype);
638         ddi_put32(mpt->m_hshk_acc_hdl, &config->PageAddress, pageaddress);
639         ddi_put8(mpt->m_hshk_acc_hdl, &config->Header.PageVersion, pageversion);
640         ddi_put8(mpt->m_hshk_acc_hdl, &config->Header.PageLength, pagelength);
641         ddi_put32(mpt->m_hshk_acc_hdl,
642             &config->PageBufferSGE.MpiSimple.FlagsLength, SGEflagslength);
643         ddi_put32(mpt->m_hshk_acc_hdl,
644             &config->PageBufferSGE.MpiSimple.u.Address64.Low,
645             SGEaddress&0xffffffffu);
646         ddi_put32(mpt->m_hshk_acc_hdl,
647             &config->PageBufferSGE.MpiSimple.u.Address64.High,
648             SGEaddress >> 32);
296             &config->PageBufferSGE.MpiSimple.u.Address32, SGEaddress32);
649         send_numbytes = sizeof (MPI2_CONFIG_REQUEST);

651         /*
652          * Post message via handshake
653          */
654         if (mptsas_send_handshake_msg(mpt, (caddr_t)config, send_numbytes,
655             mpt->m_hshk_acc_hdl)) {
656                 return (-1);
657         }
658         return (0);
659 }

661 int
662 mptsas_send_extended_config_request_msg(mptsas_t *mpt, uint8_t action,
663         uint8_t extpagetype, uint32_t pageaddress, uint8_t pagenumber,
664         uint8_t pageversion, uint16_t extpagelength,
665         uint32_t SGEflagslength, uint64_t SGEaddress)
313         uint32_t SGEflagslength, uint32_t SGEaddress32)
666 {
667         pMpi2ConfigRequest_t    config;
668         int                     send_numbytes;

670         bzero(mpt->m_hshk_memp, sizeof (MPI2_CONFIG_REQUEST));
671         config = (pMpi2ConfigRequest_t)mpt->m_hshk_memp;
672         ddi_put8(mpt->m_hshk_acc_hdl, &config->Function, MPI2_FUNCTION_CONFIG);
673         ddi_put8(mpt->m_hshk_acc_hdl, &config->Action, action);
674         ddi_put8(mpt->m_hshk_acc_hdl, &config->Header.PageNumber, pagenumber);
675         ddi_put8(mpt->m_hshk_acc_hdl, &config->Header.PageType,
676             MPI2_CONFIG_PAGETYPE_EXTENDED);
677         ddi_put8(mpt->m_hshk_acc_hdl, &config->ExtPageType, extpagetype);
678         ddi_put32(mpt->m_hshk_acc_hdl, &config->PageAddress, pageaddress);
679         ddi_put8(mpt->m_hshk_acc_hdl, &config->Header.PageVersion, pageversion);
680         ddi_put16(mpt->m_hshk_acc_hdl, &config->ExtPageLength, extpagelength);
681         ddi_put32(mpt->m_hshk_acc_hdl,
682             &config->PageBufferSGE.MpiSimple.FlagsLength, SGEflagslength);
683         ddi_put32(mpt->m_hshk_acc_hdl,
684             &config->PageBufferSGE.MpiSimple.u.Address64.Low,
685             SGEaddress&0xffffffffu);
```



```
686         ddi_put32(mpt->m_hshk_acc_hdl,
687             &config->PageBufferSGE.MpiSimple.u.Address64.High,
688             SGEaddress >> 32);
332             &config->PageBufferSGE.MpiSimple.u.Address32, SGEaddress32);
689         send_numbytes = sizeof (MPI2_CONFIG_REQUEST);

691         /*
692          * Post message via handshake
693          */
694         if (mptsas_send_handshake_msg(mpt, (caddr_t)config, send_numbytes,
695             mpt->m_hshk_acc_hdl)) {
696                 return (-1);
697         }
698         return (0);
699 }
```

*_____unchanged_portion_omitted_*

```
1093 /*
1094  * NOTE: We should be able to queue TM requests in the controller to make this
1095  * a lot faster.  If resetting all targets, for example, we can load the hi
1096  * priority queue with its limit and the controller will reply as they are
1097  * completed.  This way, we don't have to poll for one reply at a time.
1098  * Think about enhancing this later.
1099  */
1100 int
1101 mptsas_ioc_task_management(mptsas_t *mpt, int task_type, uint16_t dev_handle,
1102         int lun, uint8_t *reply, uint32_t reply_size, int mode)
1103 {
1104         /*
1105          * In order to avoid allocating variables on the stack,
1106          * we make use of the pre-existing mptsas_cmd_t and
1107          * scsi_pkt which are included in the mptsas_t which
1108          * is passed to this routine.
1109          */

1111         pMpi2SCSITaskManagementRequest_t        task;
1112         int                                     rval = FALSE;
1113         mptsas_cmd_t                            *cmd;
1114         struct scsi_pkt                         *pkt;
1115         mptsas_slots_t                          *slots = mpt->m_active;
1116         uint32_t                                i;
1117         uint64_t                                request_desc;
 760         uint32_t                                request_desc_low, i;
1118         pMPI2DefaultReply_t                     reply_msg;

1120         /*
1121          * Can't start another task management routine.
1122          */
1123         if (slots->m_slot[MPTSAS_TM_SLOT(mpt)] != NULL) {
1124                 mptsas_log(mpt, CE_WARN, "Can only start 1 task management"
1125                     " command at a time\n");
1126                 return (FALSE);
1127         }

1129         cmd = &(mpt->m_event_task_mgmt.m_event_cmd);
1130         pkt = &(mpt->m_event_task_mgmt.m_event_pkt);

1132         bzero((caddr_t)cmd, sizeof (*cmd));
1133         bzero((caddr_t)pkt, scsi_pkt_size());

1135         pkt->pkt_cdbp           = (opaque_t)&cmd->cmd_cdb[0];
1136         pkt->pkt_scbp           = (opaque_t)&cmd->cmd_scb;
1137         pkt->pkt_ha_private     = (opaque_t)cmd;
1138         pkt->pkt_flags          = (FLAG_NOINTR | FLAG_HEAD);
1139         pkt->pkt_time           = 60;
1140         pkt->pkt_address.a_target = dev_handle;
```

```
1141         pkt->pkt_address.a_lun = (uchar_t)lun;
1142         cmd->cmd_pkt            = pkt;
1143         cmd->cmd_scblen         = 1;
1144         cmd->cmd_flags          = CFLAG_TM_CMD;
1145         cmd->cmd_slot           = MPTSAS_TM_SLOT(mpt);

1147         slots->m_slot[MPTSAS_TM_SLOT(mpt)] = cmd;

1149         /*
1150          * Store the TM message in memory location corresponding to the TM slot
1151          * number.
1152          */
1153         task = (pMpi2SCSITaskManagementRequest_t)(mpt->m_req_frame +
1154             (mpt->m_req_frame_size * cmd->cmd_slot));
1155         bzero(task, mpt->m_req_frame_size);

1157         /*
1158          * form message for requested task
1159          */
1160         mptsas_init_std_hdr(mpt->m_acc_req_frame_hdl, task, dev_handle, lun, 0,
1161             MPI2_FUNCTION_SCSI_TASK_MGMT);

1163         /*
1164          * Set the task type
1165          */
1166         ddi_put8(mpt->m_acc_req_frame_hdl, &task->TaskType, task_type);

1168         /*
1169          * Send TM request using High Priority Queue.
1170          */
1171         (void) ddi_dma_sync(mpt->m_dma_req_frame_hdl, 0, 0,
1172             DDI_DMA_SYNC_FORDEV);
1173         request_desc = (cmd->cmd_slot << 16) +
 816         request_desc_low = (cmd->cmd_slot << 16) +
1174             MPI2_REQ_DESCRIPT_FLAGS_HIGH_PRIORITY;
1175         MPTSAS_START_CMD(mpt, request_desc);
 818         MPTSAS_START_CMD(mpt, request_desc_low, 0);
1176         rval = mptsas_poll(mpt, cmd, MPTSAS_POLL_TIME);

1178         if (pkt->pkt_reason == CMD_INCOMPLETE)
1179                 rval = FALSE;

1181         /*
1182          * If a reply frame was used and there is a reply buffer to copy the
1183          * reply data into, copy it.  If this fails, log a message, but don't
1184          * fail the TM request.
1185          */
1186         if (cmd->cmd_rfm && reply) {
1187                 (void) ddi_dma_sync(mpt->m_dma_reply_frame_hdl, 0, 0,
1188                     DDI_DMA_SYNC_FORCPU);
1189                 reply_msg = (pMPI2DefaultReply_t)
1190                     (mpt->m_reply_frame + (cmd->cmd_rfm -
1191                     (mpt->m_reply_frame_dma_addr&0xffffffffu)));
 834                     mpt->m_reply_frame_dma_addr));
1192                 if (reply_size > sizeof (MPI2_SCSI_TASK_MANAGE_REPLY)) {
1193                         reply_size = sizeof (MPI2_SCSI_TASK_MANAGE_REPLY);
1194                 }
1195                 mutex_exit(&mpt->m_mutex);
1196                 for (i = 0; i < reply_size; i++) {
1197                         if (ddi_copyout((uint8_t *)reply_msg + i, reply + i, 1,
1198                             mode)) {
1199                                 mptsas_log(mpt, CE_WARN, "failed to copy out "
1200                                     "reply data for TM request");
1201                                 break;
1202                         }
1203                 }
```



```
1204                 mutex_enter(&mpt->m_mutex);
1205         }

1207         /*
1208          * clear the TM slot before returning
1209          */
1210         slots->m_slot[MPTSAS_TM_SLOT(mpt)] = NULL;

1212         /*
1213          * If we lost our task management command
1214          * we need to reset the ioc
1215          */
1216         if (rval == FALSE) {
1217                 mptsas_log(mpt, CE_WARN, "mptsas_ioc_task_management failed "
1218                     "try to reset ioc to recovery!");
1219                 mpt->m_softstate &= ~MPTSAS_SS_MSG_UNIT_RESET;
1220                 if (mptsas_restart_ioc(mpt)) {
1221                         mptsas_log(mpt, CE_WARN, "mptsas_restart_ioc failed");
1222                         rval = FAILED;
1223                 }
1224         }

1226         return (rval);
1227 }

1229 /*
1230  * Complete firmware download frame for v2.0 cards.
1231  */
1232 static void
1233 mptsas_uflash2(pMpi2FWDownloadRequest fwdownload,
1234     ddi_acc_handle_t acc_hdl, uint32_t size, uint8_t type,
1235     ddi_dma_cookie_t flsh_cookie)
1236 {
1237         pMpi2FWDownloadTCSGE_t  tcsge;
1238         pMpi2SGESimple64_t      sge;
1239         uint32_t                flagslength;

1241         ddi_put8(acc_hdl, &fwdownload->Function,
1242             MPI2_FUNCTION_FW_DOWNLOAD);
1243         ddi_put8(acc_hdl, &fwdownload->ImageType, type);
1244         ddi_put8(acc_hdl, &fwdownload->MsgFlags,
1245             MPI2_FW_DOWNLOAD_MSGFLGS_LAST_SEGMENT);
1246         ddi_put32(acc_hdl, &fwdownload->TotalImageSize, size);

1248         tcsge = (pMpi2FWDownloadTCSGE_t)&fwdownload->SGL;
1249         ddi_put8(acc_hdl, &tcsge->ContextSize, 0);
1250         ddi_put8(acc_hdl, &tcsge->DetailsLength, 12);
1251         ddi_put8(acc_hdl, &tcsge->Flags, 0);
1252         ddi_put32(acc_hdl, &tcsge->ImageOffset, 0);
1253         ddi_put32(acc_hdl, &tcsge->ImageSize, size);

1255         sge = (pMpi2SGESimple64_t)(tcsge + 1);
1256         flagslength = size;
1257         flagslength |= ((uint32_t)(MPI2_SGE_FLAGS_LAST_ELEMENT |
1258             MPI2_SGE_FLAGS_END_OF_BUFFER |
1259             MPI2_SGE_FLAGS_SIMPLE_ELEMENT |
1260             MPI2_SGE_FLAGS_SYSTEM_ADDRESS |
1261             MPI2_SGE_FLAGS_64_BIT_ADDRESSING |
1262             MPI2_SGE_FLAGS_HOST_TO_IOC |
1263             MPI2_SGE_FLAGS_END_OF_LIST) << MPI2_SGE_FLAGS_SHIFT);
1264         ddi_put32(acc_hdl, &sge->FlagsLength, flagslength);
1265         ddi_put32(acc_hdl, &sge->Address.Low,
1266             flsh_cookie.dmac_address);
1267         ddi_put32(acc_hdl, &sge->Address.High,
1268             (uint32_t)(flsh_cookie.dmac_laddress >> 32));
1269 }
```

```
1271 /*
1272  * Complete firmware download frame for v2.5 cards.
1273  */
1274 static void
1275 mptsas_uflash25(pMpi25FWDownloadRequest fwdownload,
1276     ddi_acc_handle_t acc_hdl, uint32_t size, uint8_t type,
1277     ddi_dma_cookie_t flsh_cookie)
1278 {
1279         pMpi2IeeeSgeSimple64_t  sge;
1280         uint8_t                 flags;

1282         ddi_put8(acc_hdl, &fwdownload->Function,
1283             MPI2_FUNCTION_FW_DOWNLOAD);
1284         ddi_put8(acc_hdl, &fwdownload->ImageType, type);
1285         ddi_put8(acc_hdl, &fwdownload->MsgFlags,
1286             MPI2_FW_DOWNLOAD_MSGFLGS_LAST_SEGMENT);
1287         ddi_put32(acc_hdl, &fwdownload->TotalImageSize, size);

1289         ddi_put32(acc_hdl, &fwdownload->ImageOffset, 0);
1290         ddi_put32(acc_hdl, &fwdownload->ImageSize, size);

1292         sge = (pMpi2IeeeSgeSimple64_t)&fwdownload->SGL;
1293         flags = MPI2_IEEE_SGE_FLAGS_SIMPLE_ELEMENT |
1294             MPI2_IEEE_SGE_FLAGS_SYSTEM_ADDR |
1295             MPI25_IEEE_SGE_FLAGS_END_OF_LIST;
1296         ddi_put8(acc_hdl, &sge->Flags, flags);
1297         ddi_put32(acc_hdl, &sge->Length, size);
1298         ddi_put32(acc_hdl, &sge->Address.Low,
1299             flsh_cookie.dmac_address);
1300         ddi_put32(acc_hdl, &sge->Address.High,
1301             (uint32_t)(flsh_cookie.dmac_laddress >> 32));
1302 }

1304 static int mptsas_enable_mpi25_flashupdate = 0;

1306 #endif /* ! codereview */
1307 int
1308 mptsas_update_flash(mptsas_t *mpt, caddr_t ptrbuffer, uint32_t size,
1309     uint8_t type, int mode)
1310 {

1312         /*
1313          * In order to avoid allocating variables on the stack,
1314          * we make use of the pre-existing mptsas_cmd_t and
1315          * scsi_pkt which are included in the mptsas_t which
1316          * is passed to this routine.
1317          */

1319         ddi_dma_attr_t          flsh_dma_attrs;
1320         ddi_dma_cookie_t        flsh_cookie;
1321         ddi_dma_handle_t        flsh_dma_handle;
1322         ddi_acc_handle_t        flsh_accessp;
1323         caddr_t                 memp, flsh_memp;
 872         uint32_t                flagslength;
 873         pMpi2FWDownloadRequest  fwdownload;
 874         pMpi2FWDownloadTCSGE_t  tcsge;
 875         pMpi2SGESimple64_t      sge;
1324         mptsas_cmd_t            *cmd;
1325         struct scsi_pkt         *pkt;
1326         int                     i;
1327         int                     rvalue = 0;
1328         uint64_t                request_desc;

1330         if (mpt->m_MPI25) {
1331                 /*
```



```
1332                  * The code is there but not tested yet.
1333                  * User has to know there are risks here.
1334                  */
1335                 mptsas_log(mpt, CE_WARN, "mptsas_update_flash(): "
1336                     "Updating firmware through MPI 2.5 has not been "
1337                     "tested yet!\n"
1338                     "To enable set mptsas_enable_mpi25_flashupdate to 1\n");
1339                 if (!mptsas_enable_mpi25_flashupdate)
1340                         return (-1);
1341         }
 880         uint32_t                request_desc_low;

1343         if ((rvalue = (mptsas_request_from_pool(mpt, &cmd, &pkt))) == -1) {
1344                 mptsas_log(mpt, CE_WARN, "mptsas_update_flash(): allocation "
1345                     "failed. event ack command pool is full\n");
1346                 return (rvalue);
1347         }

1349         bzero((caddr_t)cmd, sizeof (*cmd));
1350         bzero((caddr_t)pkt, scsi_pkt_size());
1351         cmd->ioc_cmd_slot = (uint32_t)rvalue;

1353         /*
1354          * dynamically create a customized dma attribute structure
1355          * that describes the flash file.
1356          */
1357         flsh_dma_attrs = mpt->m_msg_dma_attr;
1358         flsh_dma_attrs.dma_attr_sgllen = 1;

1360         if (mptsas_dma_addr_create(mpt, flsh_dma_attrs, &flsh_dma_handle,
1361             &flsh_accessp, &flsh_memp, size, &flsh_cookie) == FALSE) {
1362                 mptsas_log(mpt, CE_WARN,
1363                     "(unable to allocate dma resource.");
1364                 mptsas_return_to_pool(mpt, cmd);
1365                 return (-1);
1366         }

1368         bzero(flsh_memp, size);

1370         for (i = 0; i < size; i++) {
1371                 (void) ddi_copyin(ptrbuffer + i, flsh_memp + i, 1, mode);
1372         }
1373         (void) ddi_dma_sync(flsh_dma_handle, 0, 0, DDI_DMA_SYNC_FORDEV);

1375         /*
1376          * form a cmd/pkt to store the fw download message
1377          */
1378         pkt->pkt_cdbp           = (opaque_t)&cmd->cmd_cdb[0];
1379         pkt->pkt_scbp           = (opaque_t)&cmd->cmd_scb;
1380         pkt->pkt_ha_private     = (opaque_t)cmd;
1381         pkt->pkt_flags          = FLAG_HEAD;
1382         pkt->pkt_time           = 60;
1383         cmd->cmd_pkt            = pkt;
1384         cmd->cmd_scblen         = 1;
1385         cmd->cmd_flags          = CFLAG_CMDIOC | CFLAG_FW_CMD;

1387         /*
1388          * Save the command in a slot
1389          */
1390         if (mptsas_save_cmd(mpt, cmd) == FALSE) {
1391                 mptsas_dma_addr_destroy(&flsh_dma_handle, &flsh_accessp);
1392                 mptsas_return_to_pool(mpt, cmd);
1393                 return (-1);
1394         }

1396         /*
```

```
1397          * Fill in fw download message
1398          */
1399         ASSERT(cmd->cmd_slot != 0);
1400         memp = mpt->m_req_frame + (mpt->m_req_frame_size * cmd->cmd_slot);
1401         bzero(memp, mpt->m_req_frame_size);
 941         fwdownload = (void *)memp;
 942         ddi_put8(mpt->m_acc_req_frame_hdl, &fwdownload->Function,
 943             MPI2_FUNCTION_FW_DOWNLOAD);
 944         ddi_put8(mpt->m_acc_req_frame_hdl, &fwdownload->ImageType, type);
 945         ddi_put8(mpt->m_acc_req_frame_hdl, &fwdownload->MsgFlags,
 946             MPI2_FW_DOWNLOAD_MSGFLGS_LAST_SEGMENT);
 947         ddi_put32(mpt->m_acc_req_frame_hdl, &fwdownload->TotalImageSize, size);

1403         if (mpt->m_MPI25)
1404                 mptsas_uflash25((pMpi25FWDownloadRequest)memp,
1405                     mpt->m_acc_req_frame_hdl, size, type, flsh_cookie);
1406         else
1407                 mptsas_uflash2((pMpi2FWDownloadRequest)memp,
1408                     mpt->m_acc_req_frame_hdl, size, type, flsh_cookie);
 949         tcsge = (pMpi2FWDownloadTCSGE_t)&fwdownload->SGL;
 950         ddi_put8(mpt->m_acc_req_frame_hdl, &tcsge->ContextSize, 0);
 951         ddi_put8(mpt->m_acc_req_frame_hdl, &tcsge->DetailsLength, 12);
 952         ddi_put8(mpt->m_acc_req_frame_hdl, &tcsge->Flags, 0);
 953         ddi_put32(mpt->m_acc_req_frame_hdl, &tcsge->ImageOffset, 0);
 954         ddi_put32(mpt->m_acc_req_frame_hdl, &tcsge->ImageSize, size);

 956         sge = (pMpi2SGESimple64_t)(tcsge + 1);
 957         flagslength = size;
 958         flagslength |= ((uint32_t)(MPI2_SGE_FLAGS_LAST_ELEMENT |
 959             MPI2_SGE_FLAGS_END_OF_BUFFER |
 960             MPI2_SGE_FLAGS_SIMPLE_ELEMENT |
 961             MPI2_SGE_FLAGS_SYSTEM_ADDRESS |
 962             MPI2_SGE_FLAGS_64_BIT_ADDRESSING |
 963             MPI2_SGE_FLAGS_HOST_TO_IOC |
 964             MPI2_SGE_FLAGS_END_OF_LIST) << MPI2_SGE_FLAGS_SHIFT);
 965         ddi_put32(mpt->m_acc_req_frame_hdl, &sge->FlagsLength, flagslength);
 966         ddi_put32(mpt->m_acc_req_frame_hdl, &sge->Address.Low,
 967             flsh_cookie.dmac_address);
 968         ddi_put32(mpt->m_acc_req_frame_hdl, &sge->Address.High,
 969             (uint32_t)(flsh_cookie.dmac_laddress >> 32));

1410         /*
1411          * Start command
1412          */
1413         (void) ddi_dma_sync(mpt->m_dma_req_frame_hdl, 0, 0,
1414             DDI_DMA_SYNC_FORDEV);
1415         request_desc = (cmd->cmd_slot << 16) +
 976         request_desc_low = (cmd->cmd_slot << 16) +
1416             MPI2_REQ_DESCRIPT_FLAGS_DEFAULT_TYPE;
1417         cmd->cmd_rfm = NULL;
1418         MPTSAS_START_CMD(mpt, request_desc);
 979         MPTSAS_START_CMD(mpt, request_desc_low, 0);

1420         rvalue = 0;
1421         (void) cv_reltimedwait(&mpt->m_fw_cv, &mpt->m_mutex,
1422             drv_usectohz(60 * MICROSEC), TR_CLOCK_TICK);
1423         if (!(cmd->cmd_flags & CFLAG_FINISHED)) {
1424                 mpt->m_softstate &= ~MPTSAS_SS_MSG_UNIT_RESET;
1425                 if ((mptsas_restart_ioc(mpt)) == DDI_FAILURE) {
1426                         mptsas_log(mpt, CE_WARN, "mptsas_restart_ioc failed");
1427                 }
1428                 rvalue = -1;
1429         }
1430         mptsas_remove_cmd(mpt, cmd);
1431         mptsas_dma_addr_destroy(&flsh_dma_handle, &flsh_accessp);
```



```
1433         return (rvalue);
1434 }

1436 static int
1437 mptsas_sasdevpage_0_cb(mptsas_t *mpt, caddr_t page_memp,
1438     ddi_acc_handle_t accessp, uint16_t iocstatus, uint32_t iocloginfo,
1439     va_list ap)
1440 {
1441 #ifndef __lock_lint
1442         _NOTE(ARGUNUSED(ap))
1443 #endif
1444         pMpi2SasDevicePage0_t   sasdevpage;
1445         int                     rval = DDI_SUCCESS, i;
1446         uint8_t                 *sas_addr = NULL;
1447         uint8_t                 tmp_sas_wwn[SAS_WWN_BYTE_SIZE];
1448         uint16_t                *devhdl, *bay_num, *enclosure;
1449         uint64_t                *sas_wwn;
1450         uint32_t                *dev_info;
1451         uint8_t                 *physport, *phynum;
1452         uint16_t                *pdevhdl, *io_flags;
1013         uint16_t                *pdevhdl;
1453         uint32_t                page_address;

1455         if ((iocstatus != MPI2_IOCSTATUS_SUCCESS) &&
1456             (iocstatus != MPI2_IOCSTATUS_CONFIG_INVALID_PAGE)) {
1457                 mptsas_log(mpt, CE_WARN, "mptsas_get_sas_device_page0 "
1458                     "header: IOCStatus=0x%x, IOCLogInfo=0x%x",
1459                     iocstatus, iocloginfo);
1460                 rval = DDI_FAILURE;
1461                 return (rval);
1462         }
1463         page_address = va_arg(ap, uint32_t);
1464         /*
1465          * The INVALID_PAGE status is normal if using GET_NEXT_HANDLE and there
1466          * are no more pages.  If everything is OK up to this point but the
1467          * status is INVALID_PAGE, change rval to FAILURE and quit.  Also,
1468          * signal that device traversal is complete.
1469          */
1470         if (iocstatus == MPI2_IOCSTATUS_CONFIG_INVALID_PAGE) {
1471                 if ((page_address & MPI2_SAS_DEVICE_PGAD_FORM_MASK) ==
1472                     MPI2_SAS_DEVICE_PGAD_FORM_GET_NEXT_HANDLE) {
1473                         mpt->m_done_traverse_dev = 1;
1474                 }
1475                 rval = DDI_FAILURE;
1476                 return (rval);
1477         }
1478         devhdl = va_arg(ap, uint16_t *);
1479         sas_wwn = va_arg(ap, uint64_t *);
1480         dev_info = va_arg(ap, uint32_t *);
1481         physport = va_arg(ap, uint8_t *);
1482         phynum = va_arg(ap, uint8_t *);
1483         pdevhdl = va_arg(ap, uint16_t *);
1484         bay_num = va_arg(ap, uint16_t *);
1485         enclosure = va_arg(ap, uint16_t *);
1486         io_flags = va_arg(ap, uint16_t *);

1488         sasdevpage = (pMpi2SasDevicePage0_t)page_memp;

1490         *dev_info = ddi_get32(accessp, &sasdevpage->DeviceInfo);
1491         *devhdl = ddi_get16(accessp, &sasdevpage->DevHandle);
1492         sas_addr = (uint8_t *)(&sasdevpage->SASAddress);
1493         for (i = 0; i < SAS_WWN_BYTE_SIZE; i++) {
1494                 tmp_sas_wwn[i] = ddi_get8(accessp, sas_addr + i);
1495         }
1496         bcopy(tmp_sas_wwn, sas_wwn, SAS_WWN_BYTE_SIZE);
```

```
1497         *sas_wwn = LE_64(*sas_wwn);
1498         *physport = ddi_get8(accessp, &sasdevpage->PhysicalPort);
1499         *phynum = ddi_get8(accessp, &sasdevpage->PhyNum);
1500         *pdevhdl = ddi_get16(accessp, &sasdevpage->ParentDevHandle);
1501         *bay_num = ddi_get16(accessp, &sasdevpage->Slot);
1502         *enclosure = ddi_get16(accessp, &sasdevpage->EnclosureHandle);
1503         *io_flags = ddi_get16(accessp, &sasdevpage->Flags);

1505         if (*io_flags & MPI25_SAS_DEVICE0_FLAGS_FAST_PATH_CAPABLE) {
1506                 /*
1507                  * Leave a messages about FP cabability in the log.
1508                  */
1509                 mptsas_log(mpt, CE_CONT,
1510                     "!w%016"PRIx64" FastPath Capable%s", *sas_wwn,
1511                     (*io_flags &
1512                     MPI25_SAS_DEVICE0_FLAGS_ENABLED_FAST_PATH)?
1513                     " and Enabled":" but Disabled");
1514         }

1516 #endif /* ! codereview */
1517         return (rval);
1518 }

1520 /*
1521  * Request MPI configuration page SAS device page 0 to get DevHandle, device
1522  * info and SAS address.
1523  */
1524 int
1525 mptsas_get_sas_device_page0(mptsas_t *mpt, uint32_t page_address,
1526     uint16_t *dev_handle, uint64_t *sas_wwn, uint32_t *dev_info,
1527     uint8_t *physport, uint8_t *phynum, uint16_t *pdev_handle,
1528     uint16_t *bay_num, uint16_t *enclosure, uint16_t *io_flags)
1064     uint16_t *bay_num, uint16_t *enclosure)
1529 {
1530         int rval = DDI_SUCCESS;

1532         ASSERT(mutex_owned(&mpt->m_mutex));

1534         /*
1535          * Get the header and config page.  reply contains the reply frame,
1536          * which holds status info for the request.
1537          */
1538         rval = mptsas_access_config_page(mpt,
1539             MPI2_CONFIG_ACTION_PAGE_READ_CURRENT,
1540             MPI2_CONFIG_EXTPAGETYPE_SAS_DEVICE, 0, page_address,
1541             mptsas_sasdevpage_0_cb, page_address, dev_handle, sas_wwn,
1542             dev_info, physport, phynum, pdev_handle,
1543             bay_num, enclosure, io_flags);
1079             bay_num, enclosure);

1545         return (rval);
1546 }
```

*unchanged_portion_omitted_*

```
1934 /*
1935  * Read IO unit page 0 to get information for each PHY. If needed, Read IO Unit
1936  * page1 to update the PHY information.  This is the handshaking version of
1937  * this function, which should be called during initialization only.
1938  */
1939 int
1940 mptsas_get_sas_io_unit_page_hndshk(mptsas_t *mpt)
1941 {
1942         ddi_dma_attr_t          recv_dma_attrs, page_dma_attrs;
1943         ddi_dma_cookie_t        page_cookie;
1944         ddi_dma_handle_t        recv_dma_handle, page_dma_handle;
1945         ddi_acc_handle_t        recv_accessp, page_accessp;
```



```
1946         pMpi2ConfigReply_t      configreply;
1947         pMpi2SasIOUnitPage0_t   sasioupage0;
1948         pMpi2SasIOUnitPage1_t   sasioupage1;
1949         int                     recv_numbytes;
1950         caddr_t                 recv_memp, page_memp;
1951         int                     i, num_phys, start_phy = 0;
1952         int                     page0_size =
1953             sizeof (MPI2_CONFIG_PAGE_SASIOUNIT_0) +
1954             (sizeof (MPI2_SAS_IO_UNIT0_PHY_DATA) * (MPTSAS_MAX_PHYS - 1));
1955         int                     page1_size =
1956             sizeof (MPI2_CONFIG_PAGE_SASIOUNIT_1) +
1957             (sizeof (MPI2_SAS_IO_UNIT1_PHY_DATA) * (MPTSAS_MAX_PHYS - 1));
1958         uint32_t                flags_length;
1959         uint32_t                cpdi[MPTSAS_MAX_PHYS];
1960         uint32_t                readpage1 = 0, retrypage0 = 0;
1961         uint16_t                iocstatus;
1962         uint8_t                 port_flags, page_number, action;
1963         uint32_t                reply_size = 256; /* Big enough for any page */
1964         uint_t                  state;
1965         int                     rval = DDI_FAILURE;
1966         boolean_t               free_recv = B_FALSE, free_page = B_FALSE;
1967 #endif /* ! codereview */

1969         /*
1970          * Initialize our "state machine".  This is a bit convoluted,
1971          * but it keeps us from having to do the ddi allocations numerous
1972          * times.
1973          */

1975         NDBG20(("mptsas_get_sas_io_unit_page_hndshk enter"));
1976         ASSERT(mutex_owned(&mpt->m_mutex));
1977         state = IOUC_READ_PAGE0;

1979         /*
1980          * dynamically create a customized dma attribute structure
1981          * that describes mpt's config reply page request structure.
1982          */
1983         recv_dma_attrs = mpt->m_msg_dma_attr;
1984         recv_dma_attrs.dma_attr_sgllen = 1;
1985         recv_dma_attrs.dma_attr_granular = (sizeof (MPI2_CONFIG_REPLY));

1987         if (mptsas_dma_addr_create(mpt, recv_dma_attrs,
1988             &recv_dma_handle, &recv_accessp, &recv_memp,
1989             (sizeof (MPI2_CONFIG_REPLY)), NULL) == FALSE) {
1990                 mptsas_log(mpt, CE_WARN,
1991                     "mptsas_get_sas_io_unit_page_hndshk: recv dma failed");
1992                 goto cleanup;
1993         }
1994         /* Now safe to call mptsas_dma_addr_destroy(recv_dma_handle). */
1995         free_recv = B_TRUE;
1996 #endif /* ! codereview */

1998         page_dma_attrs = mpt->m_msg_dma_attr;
1999         page_dma_attrs.dma_attr_sgllen = 1;
2000         page_dma_attrs.dma_attr_granular = reply_size;

2002         if (mptsas_dma_addr_create(mpt, page_dma_attrs,
2003             &page_dma_handle, &page_accessp, &page_memp,
2004             reply_size, &page_cookie) == FALSE) {
2005                 mptsas_log(mpt, CE_WARN,
2006                     "mptsas_get_sas_io_unit_page_hndshk: page dma failed");
2007                 goto cleanup;
2008         }
2009         /* Now safe to call mptsas_dma_addr_destroy(page_dma_handle). */
2010         free_page = B_TRUE;
2011 #endif /* ! codereview */
```

```
2013         /*
2014          * Now we cycle through the state machine.  Here's what happens:
2015          * 1. Read IO unit page 0 and set phy information
2016          * 2. See if Read IO unit page1 is needed because of port configuration
2017          * 3. Read IO unit page 1 and update phy information.
2018          */

2020         sasioupage0 = (pMpi2SasIOUnitPage0_t)page_memp;
2021         sasioupage1 = (pMpi2SasIOUnitPage1_t)page_memp;

2023         while (state != IOUC_DONE) {
2024                 switch (state) {
2025                 case IOUC_READ_PAGE0:
2026                         page_number = 0;
2027                         action = MPI2_CONFIG_ACTION_PAGE_READ_CURRENT;
2028                         flags_length = (uint32_t)page0_size;
2029                         flags_length |= ((uint32_t)(
2030                             MPI2_SGE_FLAGS_LAST_ELEMENT |
2031                             MPI2_SGE_FLAGS_END_OF_BUFFER |
2032                             MPI2_SGE_FLAGS_SIMPLE_ELEMENT |
2033                             MPI2_SGE_FLAGS_SYSTEM_ADDRESS |
2034                             MPI2_SGE_FLAGS_64_BIT_ADDRESSING |
1502                             MPI2_SGE_FLAGS_32_BIT_ADDRESSING |
2035                             MPI2_SGE_FLAGS_IOC_TO_HOST |
2036                             MPI2_SGE_FLAGS_END_OF_LIST) <<
2037                             MPI2_SGE_FLAGS_SHIFT);

2039                         break;

2041                 case IOUC_READ_PAGE1:
2042                         page_number = 1;
2043                         action = MPI2_CONFIG_ACTION_PAGE_READ_CURRENT;
2044                         flags_length = (uint32_t)page1_size;
2045                         flags_length |= ((uint32_t)(
2046                             MPI2_SGE_FLAGS_LAST_ELEMENT |
2047                             MPI2_SGE_FLAGS_END_OF_BUFFER |
2048                             MPI2_SGE_FLAGS_SIMPLE_ELEMENT |
2049                             MPI2_SGE_FLAGS_SYSTEM_ADDRESS |
2050                             MPI2_SGE_FLAGS_64_BIT_ADDRESSING |
1518                             MPI2_SGE_FLAGS_32_BIT_ADDRESSING |
2051                             MPI2_SGE_FLAGS_IOC_TO_HOST |
2052                             MPI2_SGE_FLAGS_END_OF_LIST) <<
2053                             MPI2_SGE_FLAGS_SHIFT);

2055                         break;
2056                 default:
2057                         break;
2058                 }

2060                 bzero(recv_memp, sizeof (MPI2_CONFIG_REPLY));
2061                 configreply = (pMpi2ConfigReply_t)recv_memp;
2062                 recv_numbytes = sizeof (MPI2_CONFIG_REPLY);

2064                 if (mptsas_send_extended_config_request_msg(mpt,
2065                     MPI2_CONFIG_ACTION_PAGE_HEADER,
2066                     MPI2_CONFIG_EXTPAGETYPE_SAS_IO_UNIT,
2067                     0, page_number, 0, 0, 0, 0)) {
2068                         goto cleanup;
2069                 }

2071                 if (mptsas_get_handshake_msg(mpt, recv_memp, recv_numbytes,
2072                     recv_accessp)) {
2073                         goto cleanup;
2074                 }
```



```
2076                 iocstatus = ddi_get16(recv_accessp, &configreply->IOCStatus);
2077                 iocstatus = MPTSAS_IOCSTATUS(iocstatus);

2079                 if (iocstatus != MPI2_IOCSTATUS_SUCCESS) {
2080                         mptsas_log(mpt, CE_WARN,
2081                             "mptsas_get_sas_io_unit_page_hndshk: read page "
2082                             "header iocstatus = 0x%x", iocstatus);
2083                         goto cleanup;
2084                 }

2086                 if (action != MPI2_CONFIG_ACTION_PAGE_WRITE_NVRAM) {
2087                         bzero(page_memp, reply_size);
2088                 }

2090                 if (mptsas_send_extended_config_request_msg(mpt, action,
2091                     MPI2_CONFIG_EXTPAGETYPE_SAS_IO_UNIT, 0, page_number,
2092                     ddi_get8(recv_accessp, &configreply->Header.PageVersion),
2093                     ddi_get16(recv_accessp, &configreply->ExtPageLength),
2094                     flags_length, page_cookie.dmac_laddress)) {
1562                     flags_length, page_cookie.dmac_address)) {
2095                         goto cleanup;
2096                 }

2098                 if (mptsas_get_handshake_msg(mpt, recv_memp, recv_numbytes,
2099                     recv_accessp)) {
2100                         goto cleanup;
2101                 }

2103                 iocstatus = ddi_get16(recv_accessp, &configreply->IOCStatus);
2104                 iocstatus = MPTSAS_IOCSTATUS(iocstatus);

2106                 if (iocstatus != MPI2_IOCSTATUS_SUCCESS) {
2107                         mptsas_log(mpt, CE_WARN,
2108                             "mptsas_get_sas_io_unit_page_hndshk: IO unit "
2109                             "config failed for action %d, iocstatus = 0x%x",
2110                             action, iocstatus);
2111                         goto cleanup;
2112                 }

2114                 switch (state) {
2115                 case IOUC_READ_PAGE0:
2116                         if ((ddi_dma_sync(page_dma_handle, 0, 0,
2117                             DDI_DMA_SYNC_FORCPU)) != DDI_SUCCESS) {
2118                                 goto cleanup;
2119                         }

2121                         num_phys = ddi_get8(page_accessp,
2122                             &sasioupage0->NumPhys);
2123                         ASSERT(num_phys == mpt->m_num_phys);
2124                         if (num_phys > MPTSAS_MAX_PHYS) {
2125                                 mptsas_log(mpt, CE_WARN, "Number of phys "
2126                                     "supported by HBA (%d) is more than max "
2127                                     "supported by driver (%d).  Driver will "
2128                                     "not attach.", num_phys,
2129                                     MPTSAS_MAX_PHYS);
2130                                 rval = DDI_FAILURE;
2131                                 goto cleanup;
2132                         }
2133                         for (i = start_phy; i < num_phys; i++, start_phy = i) {
2134                                 cpdi[i] = ddi_get32(page_accessp,
2135                                     &sasioupage0->PhyData[i].
2136                                     ControllerPhyDeviceInfo);
2137                                 port_flags = ddi_get8(page_accessp,
2138                                     &sasioupage0->PhyData[i].PortFlags);

2140                                 mpt->m_phy_info[i].port_num =
```

```
2141                                    ddi_get8(page_accessp,
2142                                    &sasioupage0->PhyData[i].Port);
2143                            mpt->m_phy_info[i].ctrl_devhdl =
2144                                ddi_get16(page_accessp, &sasioupage0->
2145                                PhyData[i].ControllerDevHandle);
2146                            mpt->m_phy_info[i].attached_devhdl =
2147                                ddi_get16(page_accessp, &sasioupage0->
2148                                PhyData[i].AttachedDevHandle);
2149                            mpt->m_phy_info[i].phy_device_type = cpdi[i];
2150                            mpt->m_phy_info[i].port_flags = port_flags;

2152                            if (port_flags & DISCOVERY_IN_PROGRESS) {
2153                                    retrypage0++;
2154                                    NDBG20(("Discovery in progress, can't "
2155                                        "verify IO unit config, then NO.%d"
2156                                        " times retry", retrypage0));
2157                                    break;
2158                            } else {
2159                                    retrypage0 = 0;
2160                            }
2161                            if (!(port_flags & AUTO_PORT_CONFIGURATION)) {
2162                                    /*
2163                                     * some PHY configuration described in
2164                                     * SAS IO Unit Page1
2165                                     */
2166                                    readpage1 = 1;
2167                            }
2168                    }

2170                    /*
2171                     * retry 30 times if discovery is in process
2172                     */
2173                    if (retrypage0 && (retrypage0 < 30)) {
2174                            drv_usecwait(1000 * 100);
2175                            state = IOUC_READ_PAGE0;
2176                            break;
2177                    } else if (retrypage0 == 30) {
2178                            mptsas_log(mpt, CE_WARN,
2179                                "!Discovery in progress, can't "
2180                                "verify IO unit config, then after"
2181                                " 30 times retry, give up!");
2182                            state = IOUC_DONE;
2183                            rval = DDI_FAILURE;
2184                            break;
2185                    }

2187                    if (readpage1 == 0) {
2188                            state = IOUC_DONE;
2189                            rval = DDI_SUCCESS;
2190                            break;
2191                    }

2193                    state = IOUC_READ_PAGE1;
2194                    break;

2196            case IOUC_READ_PAGE1:
2197                    if ((ddi_dma_sync(page_dma_handle, 0, 0,
2198                        DDI_DMA_SYNC_FORCPU)) != DDI_SUCCESS) {
2199                            goto cleanup;
2200                    }

2202                    num_phys = ddi_get8(page_accessp,
2203                        &sasioupage1->NumPhys);
2204                    ASSERT(num_phys == mpt->m_num_phys);
2205                    if (num_phys > MPTSAS_MAX_PHYS) {
2206                            mptsas_log(mpt, CE_WARN, "Number of phys "
```



```
2207                                    "supported by HBA (%d) is more than max "
2208                                    "supported by driver (%d).  Driver will "
2209                                    "not attach.", num_phys,
2210                                    MPTSAS_MAX_PHYS);
2211                            rval = DDI_FAILURE;
2212                            goto cleanup;
2213                    }
2214                    for (i = 0; i < num_phys; i++) {
2215                            cpdi[i] = ddi_get32(page_accessp,
2216                                &sasioupage1->PhyData[i].
2217                                ControllerPhyDeviceInfo);
2218                            port_flags = ddi_get8(page_accessp,
2219                                &sasioupage1->PhyData[i].PortFlags);
2220                            mpt->m_phy_info[i].port_num =
2221                                ddi_get8(page_accessp,
2222                                &sasioupage1->PhyData[i].Port);
2223                            mpt->m_phy_info[i].port_flags = port_flags;
2224                            mpt->m_phy_info[i].phy_device_type = cpdi[i];

2226                    }

2228                    state = IOUC_DONE;
2229                    rval = DDI_SUCCESS;
2230                    break;
2231            }
2232    }
2233    if ((mptsas_check_dma_handle(recv_dma_handle) != DDI_SUCCESS) ||
2234        (mptsas_check_dma_handle(page_dma_handle) != DDI_SUCCESS)) {
2235            ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_UNAFFECTED);
2236            rval = DDI_FAILURE;
2237            goto cleanup;
2238    }
2239    if ((mptsas_check_acc_handle(recv_accessp) != DDI_SUCCESS) ||
2240        (mptsas_check_acc_handle(page_accessp) != DDI_SUCCESS)) {
2241            ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_UNAFFECTED);
2242            rval = DDI_FAILURE;
2243            goto cleanup;
2244    }

2246 cleanup:
2247        if (free_recv)
2248 #endif /* ! codereview */
2249                mptsas_dma_addr_destroy(&recv_dma_handle, &recv_accessp);
2250        if (free_page)
2251 #endif /* ! codereview */
2252                mptsas_dma_addr_destroy(&page_dma_handle, &page_accessp);
2253        if (rval != DDI_SUCCESS) {
2254                mptsas_fm_ereport(mpt, DDI_FM_DEVICE_NO_RESPONSE);
2255                ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_LOST);
2256        }
2257        return (rval);
2258 }

2260 /*
2261  * mptsas_get_manufacture_page5
2262  *
2263  * This function will retrieve the base WWID from the adapter.  Since this
2264  * function is only called during the initialization process, use handshaking.
2265  */
2266 int
2267 mptsas_get_manufacture_page5(mptsas_t *mpt)
2268 {
2269        ddi_dma_attr_t                  recv_dma_attrs, page_dma_attrs;
2270        ddi_dma_cookie_t                page_cookie;
2271        ddi_dma_handle_t                recv_dma_handle, page_dma_handle;
2272        ddi_acc_handle_t                recv_accessp, page_accessp;
```

```
2273         pMpi2ConfigReply_t              configreply;
2274         caddr_t                         recv_memp, page_memp;
2275         int                             recv_numbytes;
2276         pMpi2ManufacturingPage5_t       m5;
2277         uint32_t                        flagslength;
2278         int                             rval = DDI_SUCCESS;
2279         uint_t                          iocstatus;
2280         boolean_t               free_recv = B_FALSE, free_page = B_FALSE;
2281 #endif /* ! codereview */

2283         MPTSAS_DISABLE_INTR(mpt);

2285         if (mptsas_send_config_request_msg(mpt, MPI2_CONFIG_ACTION_PAGE_HEADER,
2286             MPI2_CONFIG_PAGETYPE_MANUFACTURING, 0, 5, 0, 0, 0, 0)) {
2287                 rval = DDI_FAILURE;
2288                 goto done;
2289         }

2291         /*
2292          * dynamically create a customized dma attribute structure
2293          * that describes the MPT's config reply page request structure.
2294          */
2295         recv_dma_attrs = mpt->m_msg_dma_attr;
2296         recv_dma_attrs.dma_attr_sgllen = 1;
2297         recv_dma_attrs.dma_attr_granular = (sizeof (MPI2_CONFIG_REPLY));

2299         if (mptsas_dma_addr_create(mpt, recv_dma_attrs,
2300             &recv_dma_handle, &recv_accessp, &recv_memp,
2301             (sizeof (MPI2_CONFIG_REPLY)), NULL) == FALSE) {
2302                 rval = DDI_FAILURE;
2303 #endif /* ! codereview */
2304                 goto done;
2305         }
2306         /* Now safe to call mptsas_dma_addr_destroy(recv_dma_handle). */
2307         free_recv = B_TRUE;
2308 #endif /* ! codereview */

2310         bzero(recv_memp, sizeof (MPI2_CONFIG_REPLY));
2311         configreply = (pMpi2ConfigReply_t)recv_memp;
2312         recv_numbytes = sizeof (MPI2_CONFIG_REPLY);

2314         /*
2315          * get config reply message
2316          */
2317         if (mptsas_get_handshake_msg(mpt, recv_memp, recv_numbytes,
2318             recv_accessp)) {
2319                 rval = DDI_FAILURE;
2320                 goto done;
2321         }

2323         if (iocstatus = ddi_get16(recv_accessp, &configreply->IOCStatus)) {
2324                 mptsas_log(mpt, CE_WARN, "mptsas_get_manufacture_page5 update: "
2325                     "IOCStatus=0x%x, IOCLogInfo=0x%x", iocstatus,
2326                     ddi_get32(recv_accessp, &configreply->IOCLogInfo));
2327                 goto done;
2328         }

2330         /*
2331          * dynamically create a customized dma attribute structure
2332          * that describes the MPT's config page structure.
2333          */
2334         page_dma_attrs = mpt->m_msg_dma_attr;
2335         page_dma_attrs.dma_attr_sgllen = 1;
2336         page_dma_attrs.dma_attr_granular = (sizeof (MPI2_CONFIG_PAGE_MAN_5));

2338         if (mptsas_dma_addr_create(mpt, page_dma_attrs, &page_dma_handle,
```



```
2339             &page_accessp, &page_memp, (sizeof (MPI2_CONFIG_PAGE_MAN_5)),
2340             &page_cookie) == FALSE) {
2341                 rval = DDI_FAILURE;
2342 #endif /* ! codereview */
2343                 goto done;
2344         }
2345         /* Now safe to call mptsas_dma_addr_destroy(page_dma_handle). */
2346         free_page = B_TRUE;

2348 #endif /* ! codereview */
2349         bzero(page_memp, sizeof (MPI2_CONFIG_PAGE_MAN_5));
2350         m5 = (pMpi2ManufacturingPage5_t)page_memp;
2351         NDBG20(("mptsas_get_manufacture_page5: paddr 0x%x%08x",
2352             (uint32_t)(page_cookie.dmac_laddress>>32),
2353             (uint32_t)(page_cookie.dmac_laddress&0xffffffffUL)));
2354 #endif /* ! codereview */

2356         /*
2357          * Give reply address to IOC to store config page in and send
2358          * config request out.
2359          */

2361         flagslength = sizeof (MPI2_CONFIG_PAGE_MAN_5);
2362         flagslength |= ((uint32_t)(MPI2_SGE_FLAGS_LAST_ELEMENT |
2363             MPI2_SGE_FLAGS_END_OF_BUFFER | MPI2_SGE_FLAGS_SIMPLE_ELEMENT |
2364             MPI2_SGE_FLAGS_SYSTEM_ADDRESS | MPI2_SGE_FLAGS_64_BIT_ADDRESSING |
1715             MPI2_SGE_FLAGS_SYSTEM_ADDRESS | MPI2_SGE_FLAGS_32_BIT_ADDRESSING |
2365             MPI2_SGE_FLAGS_IOC_TO_HOST |
2366             MPI2_SGE_FLAGS_END_OF_LIST) << MPI2_SGE_FLAGS_SHIFT);

2368         if (mptsas_send_config_request_msg(mpt,
2369             MPI2_CONFIG_ACTION_PAGE_READ_CURRENT,
2370             MPI2_CONFIG_PAGETYPE_MANUFACTURING, 0, 5,
2371             ddi_get8(recv_accessp, &configreply->Header.PageVersion),
2372             ddi_get8(recv_accessp, &configreply->Header.PageLength),
2373             flagslength, page_cookie.dmac_laddress)) {
1724             flagslength, page_cookie.dmac_address)) {
2374                 rval = DDI_FAILURE;
2375                 goto done;
2376         }

2378         /*
2379          * get reply view handshake
2380          */
2381         if (mptsas_get_handshake_msg(mpt, recv_memp, recv_numbytes,
2382             recv_accessp)) {
2383                 rval = DDI_FAILURE;
2384                 goto done;
2385         }

2387         if (iocstatus = ddi_get16(recv_accessp, &configreply->IOCStatus)) {
2388                 mptsas_log(mpt, CE_WARN, "mptsas_get_manufacture_page5 config: "
2389                     "IOCStatus=0x%x, IOCLogInfo=0x%x", iocstatus,
2390                     ddi_get32(recv_accessp, &configreply->IOCLogInfo));
2391                 goto done;
2392         }

2394         (void) ddi_dma_sync(page_dma_handle, 0, 0, DDI_DMA_SYNC_FORCPU);

2396         /*
2397          * Fusion-MPT stores fields in little-endian format.  This is
2398          * why the low-order 32 bits are stored first.
2399          */
2400         mpt->un.sasaddr.m_base_wwid_lo =
2401             ddi_get32(page_accessp, (uint32_t *)(void *)&m5->Phy[0].WWID);
2402         mpt->un.sasaddr.m_base_wwid_hi =
```

```
2403             ddi_get32(page_accessp, (uint32_t *)(void *)&m5->Phy[0].WWID + 1);

2405         if (ddi_prop_update_int64(DDI_DEV_T_NONE, mpt->m_dip,
2406             "base-wwid", mpt->un.m_base_wwid) != DDI_PROP_SUCCESS) {
2407                 NDBG2(("%s%d: failed to create base-wwid property",
2408                     ddi_driver_name(mpt->m_dip), ddi_get_instance(mpt->m_dip)));
2409         }

2411         /*
2412          * Set the number of PHYs present.
2413          */
2414         mpt->m_num_phys = ddi_get8(page_accessp, (uint8_t *)&m5->NumPhys);

2416         if (ddi_prop_update_int(DDI_DEV_T_NONE, mpt->m_dip,
2417             "num-phys", mpt->m_num_phys) != DDI_PROP_SUCCESS) {
2418                 NDBG2(("%s%d: failed to create num-phys property",
2419                     ddi_driver_name(mpt->m_dip), ddi_get_instance(mpt->m_dip)));
2420         }

2422         mptsas_log(mpt, CE_NOTE, "!mpt%d: Initiator WWNs: 0x%016llx-0x%016llx",
2423             mpt->m_instance, (unsigned long long)mpt->un.m_base_wwid,
2424             (unsigned long long)mpt->un.m_base_wwid + mpt->m_num_phys - 1);

2426         if ((mptsas_check_dma_handle(recv_dma_handle) != DDI_SUCCESS) ||
2427             (mptsas_check_dma_handle(page_dma_handle) != DDI_SUCCESS)) {
2428                 ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_UNAFFECTED);
2429                 rval = DDI_FAILURE;
2430                 goto done;
2431         }
2432         if ((mptsas_check_acc_handle(recv_accessp) != DDI_SUCCESS) ||
2433             (mptsas_check_acc_handle(page_accessp) != DDI_SUCCESS)) {
2434                 ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_UNAFFECTED);
2435                 rval = DDI_FAILURE;
2436         }
2437 done:
2438         /*
2439          * free up memory
2440          */
2441         if (free_recv)
2442 #endif /* ! codereview */
2443                 mptsas_dma_addr_destroy(&recv_dma_handle, &recv_accessp);
2444         if (free_page)
2445 #endif /* ! codereview */
2446                 mptsas_dma_addr_destroy(&page_dma_handle, &page_accessp);
2447         MPTSAS_ENABLE_INTR(mpt);

2449         return (rval);
2450 }

2452 static int
2453 mptsas_sasphypage_0_cb(mptsas_t *mpt, caddr_t page_memp,
2454     ddi_acc_handle_t accessp, uint16_t iocstatus, uint32_t ioclogginfo,
2455     va_list ap)
2456 {
2457 #ifndef __lock_lint
2458         _NOTE(ARGUNUSED(ap))
2459 #endif
2460         pMpi2SasPhyPage0_t      sasphypage;
2461         int                     rval = DDI_SUCCESS;
2462         uint16_t                *owner_devhdl, *attached_devhdl;
2463         uint8_t                 *attached_phy_identify;
2464         uint32_t                *attached_phy_info;
2465         uint8_t                 *programmed_link_rate;
2466         uint8_t                 *hw_link_rate;
2467         uint8_t                 *change_count;
2468         uint32_t                *phy_info;
```



```
2469         uint8_t                 *negotiated_link_rate;
2470         uint32_t                page_address;

2472         if ((iocstatus != MPI2_IOCSTATUS_SUCCESS) &&
2473             (iocstatus != MPI2_IOCSTATUS_CONFIG_INVALID_PAGE)) {
2474                 mptsas_log(mpt, CE_WARN, "mptsas_get_sas_expander_page0 "
2475                     "config: IOCStatus=0x%x, IOCLogInfo=0x%x",
2476                     iocstatus, ioclogginfo);
2477                 rval = DDI_FAILURE;
2478                 return (rval);
2479         }
2480         page_address = va_arg(ap, uint32_t);
2481         /*
2482          * The INVALID_PAGE status is normal if using GET_NEXT_HANDLE and there
2483          * are no more pages.  If everything is OK up to this point but the
2484          * status is INVALID_PAGE, change rval to FAILURE and quit.  Also,
2485          * signal that device traversal is complete.
2486          */
2487         if (iocstatus == MPI2_IOCSTATUS_CONFIG_INVALID_PAGE) {
2488                 if ((page_address & MPI2_SAS_EXPAND_PGAD_FORM_MASK) ==
2489                     MPI2_SAS_EXPAND_PGAD_FORM_GET_NEXT_HNDL) {
2490                         mpt->m_done_traverse_smp = 1;
2491                 }
2492                 rval = DDI_FAILURE;
2493                 return (rval);
2494         }
2495         owner_devhdl = va_arg(ap, uint16_t *);
2496         attached_devhdl = va_arg(ap, uint16_t *);
2497         attached_phy_identify = va_arg(ap, uint8_t *);
2498         attached_phy_info = va_arg(ap, uint32_t *);
2499         programmed_link_rate = va_arg(ap, uint8_t *);
2500         hw_link_rate = va_arg(ap, uint8_t *);
2501         change_count = va_arg(ap, uint8_t *);
2502         phy_info = va_arg(ap, uint32_t *);
2503         negotiated_link_rate = va_arg(ap, uint8_t *);

2505         sasphypage = (pMpi2SasPhyPage0_t)page_memp;

2507         *owner_devhdl =
2508             ddi_get16(accessp, &sasphypage->OwnerDevHandle);
2509         *attached_devhdl =
2510             ddi_get16(accessp, &sasphypage->AttachedDevHandle);
2511         *attached_phy_identify =
2512             ddi_get8(accessp, &sasphypage->AttachedPhyIdentifier);
2513         *attached_phy_info =
2514             ddi_get32(accessp, &sasphypage->AttachedPhyInfo);
2515         *programmed_link_rate =
2516             ddi_get8(accessp, &sasphypage->ProgrammedLinkRate);
2517         *hw_link_rate =
2518             ddi_get8(accessp, &sasphypage->HwLinkRate);
2519         *change_count =
2520             ddi_get8(accessp, &sasphypage->ChangeCount);
2521         *phy_info =
2522             ddi_get32(accessp, &sasphypage->PhyInfo);
2523         *negotiated_link_rate =
2524             ddi_get8(accessp, &sasphypage->NegotiatedLinkRate);

2526         return (rval);
2527 }

2529 /*
2530  * Request MPI configuration page SAS phy page 0 to get DevHandle, phymask
2531  * and SAS address.
2532  */
2533 int
2534 mptsas_get_sas_phy_page0(mptsas_t *mpt, uint32_t page_address,
```

```
2535     smhba_info_t *info)
2536 {
2537         int                     rval = DDI_SUCCESS;

2539         ASSERT(mutex_owned(&mpt->m_mutex));

2541         /*
2542          * Get the header and config page.  reply contains the reply frame,
2543          * which holds status info for the request.
2544          */
2545         rval = mptsas_access_config_page(mpt,
2546             MPI2_CONFIG_ACTION_PAGE_READ_CURRENT,
2547             MPI2_CONFIG_EXTPAGETYPE_SAS_PHY, 0, page_address,
2548             mptsas_sasphypage_0_cb, page_address, &info->owner_devhdl,
2549             &info->attached_devhdl, &info->attached_phy_identify,
2550             &info->attached_phy_info, &info->programmed_link_rate,
2551             &info->hw_link_rate, &info->change_count,
2552             &info->phy_info, &info->negotiated_link_rate);

2554         return (rval);
2555 }

2557 static int
2558 mptsas_sasphypage_1_cb(mptsas_t *mpt, caddr_t page_memp,
2559     ddi_acc_handle_t accessp, uint16_t iocstatus, uint32_t ioclogginfo,
2560     va_list ap)
2561 {
2562 #ifndef __lock_lint
2563         _NOTE(ARGUNUSED(ap))
2564 #endif
2565         pMpi2SasPhyPage1_t      sasphypage;
2566         int                     rval = DDI_SUCCESS;

2568         uint32_t                *invalid_dword_count;
2569         uint32_t                *running_disparity_error_count;
2570         uint32_t                *loss_of_dword_sync_count;
2571         uint32_t                *phy_reset_problem_count;
2572         uint32_t                page_address;

2574         if ((iocstatus != MPI2_IOCSTATUS_SUCCESS) &&
2575             (iocstatus != MPI2_IOCSTATUS_CONFIG_INVALID_PAGE)) {
2576                 mptsas_log(mpt, CE_WARN, "mptsas_get_sas_expander_page1 "
2577                     "config: IOCStatus=0x%x, IOCLogInfo=0x%x",
2578                     iocstatus, ioclogginfo);
2579                 rval = DDI_FAILURE;
2580                 return (rval);
2581         }
2582         page_address = va_arg(ap, uint32_t);
2583         /*
2584          * The INVALID_PAGE status is normal if using GET_NEXT_HANDLE and there
2585          * are no more pages.  If everything is OK up to this point but the
2586          * status is INVALID_PAGE, change rval to FAILURE and quit.  Also,
2587          * signal that device traversal is complete.
2588          */
2589         if (iocstatus == MPI2_IOCSTATUS_CONFIG_INVALID_PAGE) {
2590                 if ((page_address & MPI2_SAS_EXPAND_PGAD_FORM_MASK) ==
2591                     MPI2_SAS_EXPAND_PGAD_FORM_GET_NEXT_HNDL) {
2592                         mpt->m_done_traverse_smp = 1;
2593                 }
2594                 rval = DDI_FAILURE;
2595                 return (rval);
2596         }

2598         invalid_dword_count = va_arg(ap, uint32_t *);
2599         running_disparity_error_count = va_arg(ap, uint32_t *);
2600         loss_of_dword_sync_count = va_arg(ap, uint32_t *);
```



```
2601         phy_reset_problem_count = va_arg(ap, uint32_t *);

2603         sasphypage = (pMpi2SasPhyPage1_t)page_memp;

2605         *invalid_dword_count =
2606             ddi_get32(accessp, &sasphypage->InvalidDwordCount);
2607         *running_disparity_error_count =
2608             ddi_get32(accessp, &sasphypage->RunningDisparityErrorCount);
2609         *loss_of_dword_sync_count =
2610             ddi_get32(accessp, &sasphypage->LossDwordSynchCount);
2611         *phy_reset_problem_count =
2612             ddi_get32(accessp, &sasphypage->PhyResetProblemCount);

2614         return (rval);
2615 }

2617 /*
2618  * Request MPI configuration page SAS phy page 0 to get DevHandle, phymask
2619  * and SAS address.
2620  */
2621 int
2622 mptsas_get_sas_phy_page1(mptsas_t *mpt, uint32_t page_address,
2623     smhba_info_t *info)
2624 {
2625         int                     rval = DDI_SUCCESS;

2627         ASSERT(mutex_owned(&mpt->m_mutex));

2629         /*
2630          * Get the header and config page.  reply contains the reply frame,
2631          * which holds status info for the request.
2632          */
2633         rval = mptsas_access_config_page(mpt,
2634             MPI2_CONFIG_ACTION_PAGE_READ_CURRENT,
2635             MPI2_CONFIG_EXTPAGETYPE_SAS_PHY, 1, page_address,
2636             mptsas_sasphypage_1_cb, page_address,
2637             &info->invalid_dword_count,
2638             &info->running_disparity_error_count,
2639             &info->loss_of_dword_sync_count,
2640             &info->phy_reset_problem_count);

2642         return (rval);
2643 }
2644 /*
2645  * mptsas_get_manufacture_page0
2646  *
2647  * This function will retrieve the base
2648  * Chip name, Board Name,Board Trace number from the adapter.
2649  * Since this function is only called during the
2650  * initialization process, use handshaking.
2651  */
2652 int
2653 mptsas_get_manufacture_page0(mptsas_t *mpt)
2654 {
2655         ddi_dma_attr_t                  recv_dma_attrs, page_dma_attrs;
2656         ddi_dma_cookie_t                page_cookie;
2657         ddi_dma_handle_t                recv_dma_handle, page_dma_handle;
2658         ddi_acc_handle_t                recv_accessp, page_accessp;
2659         pMpi2ConfigReply_t              configreply;
2660         caddr_t                         recv_memp, page_memp;
2661         int                             recv_numbytes;
2662         pMpi2ManufacturingPage0_t       m0;
2663         uint32_t                        flagslength;
2664         int                             rval = DDI_SUCCESS;
2665         uint_t                          iocstatus;
2666         uint8_t                         i = 0;
```

```
2667         boolean_t               free_recv = B_FALSE, free_page = B_FALSE;
2668 #endif /* ! codereview */

2670         MPTSAS_DISABLE_INTR(mpt);

2672         if (mptsas_send_config_request_msg(mpt, MPI2_CONFIG_ACTION_PAGE_HEADER,
2673             MPI2_CONFIG_PAGETYPE_MANUFACTURING, 0, 0, 0, 0, 0, 0)) {
2674                 rval = DDI_FAILURE;
2675                 goto done;
2676         }

2678         /*
2679          * dynamically create a customized dma attribute structure
2680          * that describes the MPT's config reply page request structure.
2681          */
2682         recv_dma_attrs = mpt->m_msg_dma_attr;
2683         recv_dma_attrs.dma_attr_sgllen = 1;
2684         recv_dma_attrs.dma_attr_granular = (sizeof (MPI2_CONFIG_REPLY));

2686         if (mptsas_dma_addr_create(mpt, recv_dma_attrs, &recv_dma_handle,
2687             &recv_accessp, &recv_memp, (sizeof (MPI2_CONFIG_REPLY)),
2688             NULL) == FALSE) {
2689                 rval = DDI_FAILURE;
2690 #endif /* ! codereview */
2691                 goto done;
2692         }
2693         /* Now safe to call mptsas_dma_addr_destroy(recv_dma_handle). */
2694         free_recv = B_TRUE;

2696 #endif /* ! codereview */
2697         bzero(recv_memp, sizeof (MPI2_CONFIG_REPLY));
2698         configreply = (pMpi2ConfigReply_t)recv_memp;
2699         recv_numbytes = sizeof (MPI2_CONFIG_REPLY);

2701         /*
2702          * get config reply message
2703          */
2704         if (mptsas_get_handshake_msg(mpt, recv_memp, recv_numbytes,
2705             recv_accessp)) {
2706                 rval = DDI_FAILURE;
2707                 goto done;
2708         }

2710         if (iocstatus = ddi_get16(recv_accessp, &configreply->IOCStatus)) {
2711                 mptsas_log(mpt, CE_WARN, "mptsas_get_manufacture_page5 update: "
2712                     "IOCStatus=0x%x, IOCLogInfo=0x%x", iocstatus,
2713                     ddi_get32(recv_accessp, &configreply->IOCLogInfo));
2714                 goto done;
2715         }

2717         /*
2718          * dynamically create a customized dma attribute structure
2719          * that describes the MPT's config page structure.
2720          */
2721         page_dma_attrs = mpt->m_msg_dma_attr;
2722         page_dma_attrs.dma_attr_sgllen = 1;
2723         page_dma_attrs.dma_attr_granular = (sizeof (MPI2_CONFIG_PAGE_MAN_0));

2725         if (mptsas_dma_addr_create(mpt, page_dma_attrs, &page_dma_handle,
2726             &page_accessp, &page_memp, (sizeof (MPI2_CONFIG_PAGE_MAN_0)),
2727             &page_cookie) == FALSE) {
2728                 rval = DDI_FAILURE;
2729 #endif /* ! codereview */
2730                 goto done;
2731         }
2732         /* Now safe to call mptsas_dma_addr_destroy(page_dma_handle). */
```



```
2733         free_page = B_TRUE;

2735 #endif /* ! codereview */
2736         bzero(page_memp, sizeof (MPI2_CONFIG_PAGE_MAN_0));
2737         m0 = (pMpi2ManufacturingPage0_t)page_memp;

2739         /*
2740          * Give reply address to IOC to store config page in and send
2741          * config request out.
2742          */

2744         flagslength = sizeof (MPI2_CONFIG_PAGE_MAN_0);
2745         flagslength |= ((uint32_t)(MPI2_SGE_FLAGS_LAST_ELEMENT |
2746             MPI2_SGE_FLAGS_END_OF_BUFFER | MPI2_SGE_FLAGS_SIMPLE_ELEMENT |
2747             MPI2_SGE_FLAGS_SYSTEM_ADDRESS | MPI2_SGE_FLAGS_64_BIT_ADDRESSING |
1792             MPI2_SGE_FLAGS_SYSTEM_ADDRESS | MPI2_SGE_FLAGS_32_BIT_ADDRESSING |
2748             MPI2_SGE_FLAGS_IOC_TO_HOST |
2749             MPI2_SGE_FLAGS_END_OF_LIST) << MPI2_SGE_FLAGS_SHIFT);

2751         if (mptsas_send_config_request_msg(mpt,
2752             MPI2_CONFIG_ACTION_PAGE_READ_CURRENT,
2753             MPI2_CONFIG_PAGETYPE_MANUFACTURING, 0, 0,
2754             ddi_get8(recv_accessp, &configreply->Header.PageVersion),
2755             ddi_get8(recv_accessp, &configreply->Header.PageLength),
2756             flagslength, page_cookie.dmac_laddress)) {
1801             flagslength, page_cookie.dmac_address)) {
2757                 rval = DDI_FAILURE;
2758                 goto done;
2759         }

2761         /*
2762          * get reply view handshake
2763          */
2764         if (mptsas_get_handshake_msg(mpt, recv_memp, recv_numbytes,
2765             recv_accessp)) {
2766                 rval = DDI_FAILURE;
2767                 goto done;
2768         }

2770         if (iocstatus = ddi_get16(recv_accessp, &configreply->IOCStatus)) {
2771                 mptsas_log(mpt, CE_WARN, "mptsas_get_manufacture_page0 config: "
2772                     "IOCStatus=0x%x, IOCLogInfo=0x%x", iocstatus,
2773                     ddi_get32(recv_accessp, &configreply->IOCLogInfo));
2774                 goto done;
2775         }

2777         (void) ddi_dma_sync(page_dma_handle, 0, 0, DDI_DMA_SYNC_FORCPU);

2779         /*
2780          * Fusion-MPT stores fields in little-endian format.  This is
2781          * why the low-order 32 bits are stored first.
2782          */

2784         for (i = 0; i < 16; i++) {
2785                 mpt->m_MANU_page0.ChipName[i] =
2786                     ddi_get8(page_accessp,
2787                     (uint8_t *)(void *)&m0->ChipName[i]);
2788         }

2790         for (i = 0; i < 8; i++) {
2791                 mpt->m_MANU_page0.ChipRevision[i] =
2792                     ddi_get8(page_accessp,
2793                     (uint8_t *)(void *)&m0->ChipRevision[i]);
2794         }

2796         for (i = 0; i < 16; i++) {
```

```
2797                 mpt->m_MANU_page0.BoardName[i] =
2798                     ddi_get8(page_accessp,
2799                     (uint8_t *)(void *)&m0->BoardName[i]);
2800         }

2802         for (i = 0; i < 16; i++) {
2803                 mpt->m_MANU_page0.BoardAssembly[i] =
2804                     ddi_get8(page_accessp,
2805                     (uint8_t *)(void *)&m0->BoardAssembly[i]);
2806         }

2808         for (i = 0; i < 16; i++) {
2809                 mpt->m_MANU_page0.BoardTracerNumber[i] =
2810                     ddi_get8(page_accessp,
2811                     (uint8_t *)(void *)&m0->BoardTracerNumber[i]);
2812         }

2814         if ((mptsas_check_dma_handle(recv_dma_handle) != DDI_SUCCESS) ||
2815             (mptsas_check_dma_handle(page_dma_handle) != DDI_SUCCESS)) {
2816                 ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_UNAFFECTED);
2817                 rval = DDI_FAILURE;
2818                 goto done;
2819         }
2820         if ((mptsas_check_acc_handle(recv_accessp) != DDI_SUCCESS) ||
2821             (mptsas_check_acc_handle(page_accessp) != DDI_SUCCESS)) {
2822                 ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_UNAFFECTED);
2823                 rval = DDI_FAILURE;
2824         }
2825 done:
2826         /*
2827          * free up memory
2828          */
2829         if (free_recv)
2830 #endif /* ! codereview */
2831                 mptsas_dma_addr_destroy(&recv_dma_handle, &recv_accessp);
2832         if (free_page)
2833 #endif /* ! codereview */
2834                 mptsas_dma_addr_destroy(&page_dma_handle, &page_accessp);
2835         MPTSAS_ENABLE_INTR(mpt);

2837         return (rval);
2838 }
```

```
**********************************************************
   21016 Thu Jun 12 17:42:17 2014
new/usr/src/uts/common/io/scsi/adapters/mpt_sas3/mptsas3_init.c
Fix identifying 2308 cards
Add support for more than 8 MSI-X interrupts.
Tidy up interrupt assignement and card ID messages.
Added code to support using MSI-X interrupts across multiple
reply queues. Not tested with anything other than 3008 yet.
Use MSI-X interrupts, just one for now.
Pre-allocate array for request sense buffers, similar to command frames.
No more messing about with scsi_alloc_consistent_buf().
Initial modifications using the code changes present between
the LSI source code for FreeBSD drivers. Specifically the changes
between from mpslsi-source-17.00.00.00 -> mpslsi-source-03.00.00.00.
This mainly involves using a different scatter/gather element in
frame setup.
Changes to enable driver to compile.
Header paths, object lists, etc.
**********************************************************
```

```
/*
 * 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.
 * Copyright (c) 2014, Tegile Systems Inc. All rights reserved.
#endif /* ! codereview */
 */

/*
 * Copyright (c) 2000 to 2009, LSI Corporation.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms of all code within
 * this file that is exclusively owned by LSI, with or without
 * modification, is permitted provided that, in addition to the CDDL 1.0
 * License requirements, the following conditions are met:
 *
 *    Neither the name of the author nor the names of its contributors may be
 *    used to endorse or promote products derived from this software without
 *    specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS 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
 * COPYRIGHT OWNER 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.
 */

/*
 * mptsas_init - This file contains all the functions used to initialize
 * MPT2.0 based hardware.
 */

#if defined(lint) || defined(DEBUG)
#define MPTSAS_DEBUG
#endif

/*
 * standard header files
 */
#include <sys/note.h>
#include <sys/scsi/scsi.h>

#pragma pack(1)
#include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_type.h>
#include <sys/scsi/adapters/mpt_sas3/mpi/mpi2.h>
#include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_cnfg.h>
#include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_init.h>
#include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_ioc.h>
#include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_tool.h>
#include <sys/scsi/adapters/mpt_sas/mpi/mpi2_type.h>
#include <sys/scsi/adapters/mpt_sas/mpi/mpi2.h>
#include <sys/scsi/adapters/mpt_sas/mpi/mpi2_cnfg.h>
#include <sys/scsi/adapters/mpt_sas/mpi/mpi2_init.h>
#include <sys/scsi/adapters/mpt_sas/mpi/mpi2_ioc.h>
#include <sys/scsi/adapters/mpt_sas/mpi/mpi2_tool.h>
#pragma pack()
/*
 * private header files.
 */
#include <sys/scsi/adapters/mpt_sas3/mptsas3_var.h>
#include <sys/scsi/adapters/mpt_sas/mptsas_var.h>

static int mptsas_ioc_do_get_facts(mptsas_t *mpt, caddr_t memp, int var,
        ddi_acc_handle_t accessp);
static int mptsas_ioc_do_get_facts_reply(mptsas_t *mpt, caddr_t memp, int var,
        ddi_acc_handle_t accessp);
static int mptsas_ioc_do_get_port_facts(mptsas_t *mpt, caddr_t memp, int var,
        ddi_acc_handle_t accessp);
static int mptsas_ioc_do_get_port_facts_reply(mptsas_t *mpt, caddr_t memp,
    int var, ddi_acc_handle_t accessp);
static int mptsas_ioc_do_enable_port(mptsas_t *mpt, caddr_t memp, int var,
        ddi_acc_handle_t accessp);
static int mptsas_ioc_do_enable_port_reply(mptsas_t *mpt, caddr_t memp, int var,
        ddi_acc_handle_t accessp);
static int mptsas_ioc_do_enable_event_notification(mptsas_t *mpt, caddr_t memp,
        int var, ddi_acc_handle_t accessp);
static int mptsas_ioc_do_enable_event_notification_reply(mptsas_t *mpt,
    caddr_t memp, int var, ddi_acc_handle_t accessp);
static int mptsas_do_ioc_init(mptsas_t *mpt, caddr_t memp, int var,
        ddi_acc_handle_t accessp);
static int mptsas_do_ioc_init_reply(mptsas_t *mpt, caddr_t memp, int var,
        ddi_acc_handle_t accessp);

static const char *
mptsas_devid_type_string(mptsas_t *mpt)
```

```
 59 mptsas_product_type_string(mptsas_t *mpt)
107 {
108         switch (mpt->m_devid) {
109         case MPI2_MFGPAGE_DEVID_SAS2008:
110                 return ("SAS2008");
111         case MPI2_MFGPAGE_DEVID_SAS2004:
112                 return ("SAS2004");
113         case MPI2_MFGPAGE_DEVID_SAS2108_1:
114         case MPI2_MFGPAGE_DEVID_SAS2108_2:
115         case MPI2_MFGPAGE_DEVID_SAS2108_3:
116                 return ("SAS2108");
117         case MPI2_MFGPAGE_DEVID_SAS2116_1:
118         case MPI2_MFGPAGE_DEVID_SAS2116_2:
119                 return ("SAS2116");
120         case MPI2_MFGPAGE_DEVID_SAS2208_1:
121         case MPI2_MFGPAGE_DEVID_SAS2208_2:
122         case MPI2_MFGPAGE_DEVID_SAS2208_3:
123         case MPI2_MFGPAGE_DEVID_SAS2208_4:
124         case MPI2_MFGPAGE_DEVID_SAS2208_5:
125         case MPI2_MFGPAGE_DEVID_SAS2208_6:
126 #if 0
127         /* These are the same as the next 2?? */
128         case MPI2_MFGPAGE_DEVID_SAS2208_7:
129         case MPI2_MFGPAGE_DEVID_SAS2208_8:
130 #endif
131                 return ("SAS2208");
132         case MPI2_MFGPAGE_DEVID_SAS2308_1:
133         case MPI2_MFGPAGE_DEVID_SAS2308_2:
134         case MPI2_MFGPAGE_DEVID_SAS2308_3:
135                 return ("SAS2308");
136         case MPI25_MFGPAGE_DEVID_SAS3004:
137                 return ("SAS3004");
138         case MPI25_MFGPAGE_DEVID_SAS3008:
139                 return ("SAS3008");
140         case MPI25_MFGPAGE_DEVID_SAS3108_1:
141         case MPI25_MFGPAGE_DEVID_SAS3108_2:
142         case MPI25_MFGPAGE_DEVID_SAS3108_3:
143         case MPI25_MFGPAGE_DEVID_SAS3108_4:
144         case MPI25_MFGPAGE_DEVID_SAS3108_5:
145         case MPI25_MFGPAGE_DEVID_SAS3108_6:
146                 return ("SAS3108");
 61         switch (mpt->m_productid & MPI2_FW_HEADER_PID_PROD_MASK) {

 63         case MPI2_FW_HEADER_PID_PROD_A:
 64                 return ("A");
147         default:
148                 return ("?");
149         }
150 }
```

*unchanged_portion_omitted_*

```
199 static int
200 mptsas_ioc_do_get_facts_reply(mptsas_t *mpt, caddr_t memp, int var,
201                 ddi_acc_handle_t accessp)
202 {
203 #ifndef __lock_lint
204         _NOTE(ARGUNUSED(var))
205 #endif

207         pMpi2IOCFactsReply_t    factsreply;
208         int                     numbytes;
209         uint_t                  iocstatus;
210         char                    buf[32];
211         uint16_t                numReplyFrames;
212         uint16_t                queueSize, queueDiff;
213         int                     simple_sge_main;
```



```
214         int                     simple_sge_next;
215         uint32_t                capabilities;
216         uint16_t                msgversion;
217 #endif /* ! codereview */

219         bzero(memp, sizeof (*factsreply));
220         factsreply = (void *)memp;
221         numbytes = sizeof (*factsreply);

223         /*
224          * get ioc facts reply message
225          */
226         if (mptsas_get_handshake_msg(mpt, memp, numbytes, accessp)) {
227                 return (DDI_FAILURE);
228         }

230         if (iocstatus = ddi_get16(accessp, &factsreply->IOCStatus)) {
231                 mptsas_log(mpt, CE_WARN, "mptsas_ioc_do_get_facts_reply: "
232                     "IOCStatus=0x%x, IOCLogInfo=0x%x", iocstatus,
233                     ddi_get32(accessp, &factsreply->IOCLogInfo));
234                 return (DDI_FAILURE);
235         }

237         /*
238          * store key values from reply to mpt structure
239          */
240         mpt->m_fwversion = ddi_get32(accessp, &factsreply->FWVersion.Word);
241         mpt->m_productid = ddi_get16(accessp, &factsreply->ProductID);


244         (void) sprintf(buf, "%u.%u.%u.%u",
245             ddi_get8(accessp, &factsreply->FWVersion.Struct.Major),
246             ddi_get8(accessp, &factsreply->FWVersion.Struct.Minor),
247             ddi_get8(accessp, &factsreply->FWVersion.Struct.Unit),
248             ddi_get8(accessp, &factsreply->FWVersion.Struct.Dev));
249         mptsas_log(mpt, CE_NOTE, "?MPT Firmware version v%s (%s)\n",
250             buf, mptsas_devid_type_string(mpt));
134         mptsas_log(mpt, CE_NOTE, "?mpt%d Firmware version v%s (%s)\n",
135             mpt->m_instance, buf, mptsas_product_type_string(mpt));
251         (void) ddi_prop_update_string(DDI_DEV_T_NONE, mpt->m_dip,
252             "firmware-version", buf);

254         /*
255          * Set up request info.
256          */
257         mpt->m_max_requests = ddi_get16(accessp,
258             &factsreply->RequestCredit) - 1;
259         mpt->m_req_frame_size = ddi_get16(accessp,
260             &factsreply->IOCRequestFrameSize) * 4;

262         /*
263          * Size of reply free queue should be the number of requests
264          * plus some additional for events (32).  Make sure number of
265          * reply frames is not a multiple of 16 so that the queue sizes
266          * are calculated correctly later to be a multiple of 16.
267          */
268         mpt->m_reply_frame_size = ddi_get8(accessp,
269             &factsreply->ReplyFrameSize) * 4;
270         numReplyFrames = mpt->m_max_requests + 32;
271         if (!(numReplyFrames % 16)) {
272                 numReplyFrames--;
273         }
274         mpt->m_max_replies = numReplyFrames;
275         queueSize = numReplyFrames;
276         queueSize += 16 - (queueSize % 16);
277         mpt->m_free_queue_depth = queueSize;
```

```
279         /*
280          * Size of reply descriptor post queue should be the number of
281          * request frames + the number of reply frames + 1 and needs to
282          * be a multiple of 16.  This size can be no larger than
283          * MaxReplyDescriptorPostQueueDepth from IOCFacts.  If the
284          * calculated queue size is larger than allowed, subtract a
285          * multiple of 16 from m_max_requests, m_max_replies, and
286          * m_reply_free_depth.
287          *
288          * There is no indication in the spec that you can reduce the
289          * queue size if you have many.
290 #endif /* ! codereview */
291          */
292         queueSize = mpt->m_max_requests + numReplyFrames + 1;
293         if (queueSize % 16) {
294                 queueSize += 16 - (queueSize % 16);
295         }
296         mpt->m_post_queue_depth = ddi_get16(accessp,
297             &factsreply->MaxReplyDescriptorPostQueueDepth);
298         if (queueSize > mpt->m_post_queue_depth) {
299                 queueDiff = queueSize - mpt->m_post_queue_depth;
300                 if (queueDiff % 16) {
301                         queueDiff += 16 - (queueDiff % 16);
302                 }
303                 mpt->m_max_requests -= queueDiff;
304                 mpt->m_max_replies -= queueDiff;
305                 mpt->m_free_queue_depth -= queueDiff;
306                 queueSize -= queueDiff;
307         }
308         mpt->m_post_queue_depth = queueSize;

310         /*
311          * Set up max chain depth.
312          */
313         mpt->m_max_chain_depth = ddi_get8(accessp,
314             &factsreply->MaxChainDepth);
315         mpt->m_ioc_capabilities = ddi_get32(accessp,
316             &factsreply->IOCCapabilities);
317         if (mpt->m_ioc_capabilities & MPI2_IOCFACTS_CAPABILITY_MSI_X_INDEX) {
318                 mpt->m_max_msix_vectors = ddi_get8(accessp,
319                     &factsreply->MaxMSIxVectors);
320         }

322         /*
323          * Set flag to check for SAS3 support.
324          */
325         msgversion = ddi_get16(accessp, &factsreply->MsgVersion);
326         if (msgversion == MPI2_VERSION_02_05) {
327                 mptsas_log(mpt, CE_NOTE, "?mpt_sas3%d SAS 3 Supported\n",
328                     mpt->m_instance);
329                 mpt->m_MPI25 = TRUE;
330         } else {
331                 mptsas_log(mpt, CE_NOTE, "?mpt_sas3%d MPI Version 0x%x\n",
332                     mpt->m_instance, msgversion);
333         }
334 #endif /* ! codereview */

336         /*
337          * Calculate max frames per request based on DMA S/G length.
338          */
339         simple_sge_main = MPTSAS_MAX_FRAME_SGES64(mpt) - 1;
340         simple_sge_next = mpt->m_req_frame_size /
341             (mpt->m_MPI25 ? sizeof (MPI2_IEEE_SGE_SIMPLE64) :
342             sizeof (MPI2_SGE_SIMPLE64)) - 1;
172             sizeof (MPI2_SGE_SIMPLE64) - 1;
```



```
344         mpt->m_max_request_frames = (MPTSAS_MAX_DMA_SEGS -
345             simple_sge_main) / simple_sge_next + 1;
346         if (((MPTSAS_MAX_DMA_SEGS - simple_sge_main) %
347             simple_sge_next) > 1) {
348                 mpt->m_max_request_frames++;
349         }

351         /*
352          * Check if controller supports FW diag buffers and set flag to enable
353          * each type.
354          */
355         capabilities = mpt->m_ioc_capabilities;
185         capabilities = ddi_get32(accessp, &factsreply->IOCCapabilities);
356         if (capabilities & MPI2_IOCFACTS_CAPABILITY_DIAG_TRACE_BUFFER) {
357                 mpt->m_fw_diag_buffer_list[MPI2_DIAG_BUF_TYPE_TRACE].enabled =
358                     TRUE;
359         }
360         if (capabilities & MPI2_IOCFACTS_CAPABILITY_SNAPSHOT_BUFFER) {
361                 mpt->m_fw_diag_buffer_list[MPI2_DIAG_BUF_TYPE_SNAPSHOT].
362                     enabled = TRUE;
363         }
364         if (capabilities & MPI2_IOCFACTS_CAPABILITY_EXTENDED_BUFFER) {
365                 mpt->m_fw_diag_buffer_list[MPI2_DIAG_BUF_TYPE_EXTENDED].
366                     enabled = TRUE;
367         }

369         /*
370          * Check if controller supports replaying events when issuing Message
371          * Unit Reset and set flag to enable MUR.
372          */
373         if (capabilities & MPI2_IOCFACTS_CAPABILITY_EVENT_REPLAY) {
374                 mpt->m_event_replay = TRUE;
375         }

377         /*
378          * Check if controller supports IR.
379          */
380         if (capabilities & MPI2_IOCFACTS_CAPABILITY_INTEGRATED_RAID) {
381                 mpt->m_ir_capable = TRUE;
382         }

384         return (DDI_SUCCESS);
385 }
```

_____unchanged_portion_omitted_

```
649 static int
650 mptsas_do_ioc_init(mptsas_t *mpt, caddr_t memp, int var,
651     ddi_acc_handle_t accessp)
652 {
653 #ifndef __lock_lint
654         _NOTE(ARGUNUSED(var))
655 #endif

657         pMpi2IOCInitRequest_t   init;
658         int                     numbytes;
659         timespec_t              time;
660         uint64_t                mSec;

662         bzero(memp, sizeof (*init));
663         init = (void *)memp;
664         ddi_put8(accessp, &init->Function, MPI2_FUNCTION_IOC_INIT);
665         ddi_put8(accessp, &init->WhoInit, MPI2_WHOINIT_HOST_DRIVER);
666         ddi_put16(accessp, &init->MsgVersion, MPI2_VERSION);
667         ddi_put16(accessp, &init->HeaderVersion, MPI2_HEADER_VERSION);
668         if (mpt->m_intr_type == DDI_INTR_TYPE_MSIX) {
```

```
669                 ddi_put8(accessp, &init->HostMSIxVectors, mpt->m_intr_cnt);
670         }
671 #endif /* ! codereview */
672         ddi_put16(accessp, &init->SystemRequestFrameSize,
673             mpt->m_req_frame_size / 4);
674         ddi_put16(accessp, &init->ReplyDescriptorPostQueueDepth,
675             mpt->m_post_queue_depth);
676         ddi_put16(accessp, &init->ReplyFreeQueueDepth,
677             mpt->m_free_queue_depth);

679         /*
680          * These addresses are set using the DMA cookie addresses from when the
681          * memory was allocated.  Sense buffer hi address should be 0.
682          */
683         ddi_put32(accessp, &init->SenseBufferAddressHigh,
684             (uint32_t)(mpt->m_req_sense_dma_addr >> 32));
498         ddi_put32(accessp, &init->SenseBufferAddressHigh, 0);
685         ddi_put32(accessp, &init->SystemReplyAddressHigh,
686             (uint32_t)(mpt->m_reply_frame_dma_addr >> 32));
687         ddi_put32(accessp, &init->SystemRequestFrameBaseAddress.High,
688             (uint32_t)(mpt->m_req_frame_dma_addr >> 32));
689         ddi_put32(accessp, &init->SystemRequestFrameBaseAddress.Low,
690             (uint32_t)mpt->m_req_frame_dma_addr);
691         ddi_put32(accessp, &init->ReplyDescriptorPostQueueAddress.High,
692             (uint32_t)(mpt->m_post_queue_dma_addr >> 32));
693         ddi_put32(accessp, &init->ReplyDescriptorPostQueueAddress.Low,
694             (uint32_t)mpt->m_post_queue_dma_addr);
695         ddi_put32(accessp, &init->ReplyFreeQueueAddress.High,
696             (uint32_t)(mpt->m_free_queue_dma_addr >> 32));
697         ddi_put32(accessp, &init->ReplyFreeQueueAddress.Low,
698             (uint32_t)mpt->m_free_queue_dma_addr);

700         /*
701          * Fill in the timestamp with the number of milliseconds since midnight
702          * of January 1, 1970 UT (Greenwich Mean Time).  Time is returned in
703          * seconds and nanoseconds.  Translate both to milliseconds and add
704          * them together to get total milliseconds.
705          */
706         gethrestime(&time);
707         mSec = time.tv_sec * MILLISEC;
708         mSec += (time.tv_nsec / MICROSEC);
709         ddi_put32(accessp, &init->TimeStamp.High, (uint32_t)(mSec >> 32));
710         ddi_put32(accessp, &init->TimeStamp.Low, (uint32_t)mSec);

712         numbytes = sizeof (*init);

714         /*
715          * Post message via handshake
716          */
717         if (mptsas_send_handshake_msg(mpt, memp, numbytes, accessp)) {
718                 return (DDI_FAILURE);
719         }

721         return (DDI_SUCCESS);
722 }
```

*unchanged_portion_omitted_*

```
**********************************************************
   22084 Thu Jun 12 17:42:17 2014
new/usr/src/uts/common/io/scsi/adapters/mpt_sas3/mptsas3_raid.c
hg changesets 607a5b46a793..b706c96317c3
Fix ncpus for early boot config
Purge the ack to the interrupt before exiting mptsas_intr()
Changes from code review
Update tx waitq's code.
Create 2 threads, divide the workflow and deliver
to the hardware from the threads.
Added code to support using MSI-X interrupts across multiple
reply queues. Not tested with anything other than 3008 yet.
Changes to enable driver to compile.
Header paths, object lists, etc.
**********************************************************
```

```
/*
 * 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 2010 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 * Copyright (c) 2014, Tegile Systems Inc. All rights reserved.
#endif /* ! codereview */
 */

/*
 * Copyright (c) 2000 to 2010, LSI Corporation.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms of all code within
 * this file that is exclusively owned by LSI, with or without
 * modification, is permitted provided that, in addition to the CDDL 1.0
 * License requirements, the following conditions are met:
 *
 *    Neither the name of the author nor the names of its contributors may be
 *    used to endorse or promote products derived from this software without
 *    specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS 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
 * COPYRIGHT OWNER 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.
 */

/*
 * mptsas_raid - This file contains all the RAID related functions for the
 * MPT interface.
 */

#if defined(lint) || defined(DEBUG)
#define MPTSAS_DEBUG
#endif

#define MPI_RAID_VOL_PAGE_0_PHYSDISK_MAX        2

/*
 * standard header files
 */
#include <sys/note.h>
#include <sys/scsi/scsi.h>
#include <sys/byteorder.h>
#include <sys/raidioctl.h>

#pragma pack(1)

#include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_type.h>
#include <sys/scsi/adapters/mpt_sas3/mpi/mpi2.h>
#include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_cnfg.h>
#include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_init.h>
#include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_ioc.h>
#include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_raid.h>
#include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_tool.h>
#include <sys/scsi/adapters/mpt_sas/mpi/mpi2_type.h>
#include <sys/scsi/adapters/mpt_sas/mpi/mpi2.h>
#include <sys/scsi/adapters/mpt_sas/mpi/mpi2_cnfg.h>
#include <sys/scsi/adapters/mpt_sas/mpi/mpi2_init.h>
#include <sys/scsi/adapters/mpt_sas/mpi/mpi2_ioc.h>
#include <sys/scsi/adapters/mpt_sas/mpi/mpi2_raid.h>
#include <sys/scsi/adapters/mpt_sas/mpi/mpi2_tool.h>

#pragma pack()

/*
 * private header files.
 */
#include <sys/scsi/adapters/mpt_sas3/mptsas3_var.h>
#include <sys/scsi/adapters/mpt_sas/mptsas_var.h>

static int mptsas_get_raid_wwid(mptsas_t *mpt, mptsas_raidvol_t *raidvol);

extern int mptsas_check_dma_handle(ddi_dma_handle_t handle);
extern int mptsas_check_acc_handle(ddi_acc_handle_t handle);
extern mptsas_target_t *mptsas_tgt_alloc(mptsas_t *, uint16_t,
    uint64_t, uint32_t, mptsas_phymask_t, uint8_t);

static int
mptsas_raidconf_page_0_cb(mptsas_t *mpt, caddr_t page_memp,
    ddi_acc_handle_t accessp, uint16_t iocstatus, uint32_t iocloginfo,
    va_list ap)
{
#ifndef __lock_lint
        _NOTE(ARGUNUSED(ap))
#endif
        pMpi2RaidConfigurationPage0_t   raidconfig_page0;
        pMpi2RaidConfig0ConfigElement_t element;
        uint32_t *confignum;
```

```
110         int rval = DDI_SUCCESS, i;
111         uint8_t numelements, vol, disk;
112         uint16_t elementtype, voldevhandle;
113         uint16_t etype_vol, etype_pd, etype_hs;
114         uint16_t etype_oce;
115         m_raidconfig_t *raidconfig;
116         uint64_t raidwwn;
117         uint32_t native;
118         mptsas_target_t *ptgt;
119         uint32_t configindex;

121         if (iocstatus == MPI2_IOCSTATUS_CONFIG_INVALID_PAGE) {
122                 return (DDI_FAILURE);
123         }

125         if (iocstatus != MPI2_IOCSTATUS_SUCCESS) {
126                 mptsas_log(mpt, CE_WARN, "mptsas_get_raid_conf_page0 "
127                     "config: IOCStatus=0x%x, IOCLogInfo=0x%x",
128                     iocstatus, iocloginfo);
129                 rval = DDI_FAILURE;
130                 return (rval);
131         }
132         confignum = va_arg(ap,  uint32_t *);
133         configindex = va_arg(ap, uint32_t);
134         raidconfig_page0 = (pMpi2RaidConfigurationPage0_t)page_memp;
135         /*
136          * Get all RAID configurations.
137          */
138         etype_vol = MPI2_RAIDCONFIG0_EFLAGS_VOLUME_ELEMENT;
139         etype_pd = MPI2_RAIDCONFIG0_EFLAGS_VOL_PHYS_DISK_ELEMENT;
140         etype_hs = MPI2_RAIDCONFIG0_EFLAGS_HOT_SPARE_ELEMENT;
141         etype_oce = MPI2_RAIDCONFIG0_EFLAGS_OCE_ELEMENT;
142         /*
143          * Set up page address for next time through.
144          */
145         *confignum =  ddi_get8(accessp,
146             &raidconfig_page0->ConfigNum);

148         /*
149          * Point to the right config in the structure.
150          * Increment the number of valid RAID configs.
151          */
152         raidconfig = &mpt->m_raidconfig[configindex];
153         mpt->m_num_raid_configs++;

155         /*
156          * Set the native flag if this is not a foreign
157          * configuration.
158          */
159         native = ddi_get32(accessp, &raidconfig_page0->Flags);
160         if (native & MPI2_RAIDCONFIG0_FLAG_FOREIGN_CONFIG) {
161                 native = FALSE;
162         } else {
163                 native = TRUE;
164         }
165         raidconfig->m_native = (uint8_t)native;

167         /*
168          * Get volume information for the volumes in the
169          * config.
170          */
171         numelements = ddi_get8(accessp, &raidconfig_page0->NumElements);
172         vol = 0;
173         disk = 0;
174         element = (pMpi2RaidConfig0ConfigElement_t)
175             &raidconfig_page0->ConfigElement;
```



```
177         for (i = 0; ((i < numelements) && native); i++, element++) {
178                 /*
179                  * Get the element type.  Could be Volume,
180                  * PhysDisk, Hot Spare, or Online Capacity
181                  * Expansion PhysDisk.
182                  */
183                 elementtype = ddi_get16(accessp, &element->ElementFlags);
184                 elementtype &= MPI2_RAIDCONFIG0_EFLAGS_MASK_ELEMENT_TYPE;

186                 /*
187                  * For volumes, get the RAID settings and the
188                  * WWID.
189                  */
190                 if (elementtype == etype_vol) {
191                         voldevhandle = ddi_get16(accessp,
192                             &element->VolDevHandle);
193                         raidconfig->m_raidvol[vol].m_israid = 1;
194                         raidconfig->m_raidvol[vol].
195                             m_raidhandle = voldevhandle;
196                         /*
197                          * Get the settings for the raid
198                          * volume.  This includes the
199                          * DevHandles for the disks making up
200                          * the raid volume.
201                          */
202                         if (mptsas_get_raid_settings(mpt,
203                             &raidconfig->m_raidvol[vol]))
204                                 continue;

206                         /*
207                          * Get the WWID of the RAID volume for
208                          * SAS HBA
209                          */
210                         if (mptsas_get_raid_wwid(mpt,
211                             &raidconfig->m_raidvol[vol]))
212                                 continue;

214                         raidwwn = raidconfig->m_raidvol[vol].
215                             m_raidwwid;

217                         /*
218                          * RAID uses phymask of 0.
219                          */
220                         ptgt = mptsas_tgt_alloc(mpt,
221                             voldevhandle, raidwwn, 0, 0, 0);

223                         raidconfig->m_raidvol[vol].m_raidtgt =
224                             ptgt;

226                         /*
227                          * Increment volume index within this
228                          * raid config.
229                          */
230                         vol++;
231                 } else if ((elementtype == etype_pd) ||
232                     (elementtype == etype_hs) ||
233                     (elementtype == etype_oce)) {
234                         /*
235                          * For all other element types, put
236                          * their DevHandles in the phys disk
237                          * list of the config.  These are all
238                          * some variation of a Phys Disk and
239                          * this list is used to keep these
240                          * disks from going online.
241                          */
```

```
242                         raidconfig->m_physdisk_devhdl[disk] = ddi_get16(accessp,
243                             &element->PhysDiskDevHandle);
245                         /*
246                          * Increment disk index within this
247                          * raid config.
248                          */
249                         disk++;
250                 }
251         }
253         return (rval);
254 }
_____unchanged_portion_omitted_
562 /*
563  * RAID Action for System Shutdown. This request uses the dedicated TM slot to
564  * avoid a call to mptsas_save_cmd.  Since Solaris requires that the mutex is
565  * not held during the mptsas_quiesce function, this RAID action must not use
566  * the normal code path of requests and replies.
567  */
568 void
569 mptsas_raid_action_system_shutdown(mptsas_t *mpt)
570 {
571         mptsas_reply_pqueue_t           *rpqp;
572 #endif /* ! codereview */
573         pMpi2RaidActionRequest_t        action;
574         uint8_t                         ir_active = FALSE, reply_type;
575         uint8_t                         function, found_reply = FALSE;
576         uint16_t                        SMID, action_type;
577         mptsas_slots_t                  *slots = mpt->m_active;
578         int                             config, vol;
579         mptsas_cmd_t                    *cmd;
580         uint32_t                        reply_addr;
581         uint64_t                        request_desc;
519         uint32_t                        request_desc_low, reply_addr;
582         int                             cnt;
583         pMpi2ReplyDescriptorsUnion_t    reply_desc_union;
584         pMPI2DefaultReply_t             reply;
585         pMpi2AddressReplyDescriptor_t   address_reply;
587         /*
588          * Before doing the system shutdown RAID Action, make sure that the IOC
589          * supports IR and make sure there is a valid volume for the request.
590          */
591         if (mpt->m_ir_capable) {
592                 for (config = 0; (config < mpt->m_num_raid_configs) &&
593                     (!ir_active); config++) {
594                         for (vol = 0; vol < MPTSAS_MAX_RAIDVOLS; vol++) {
595                                 if (mpt->m_raidconfig[config].m_raidvol[vol].
596                                     m_israid) {
597                                         ir_active = TRUE;
598                                         break;
599                                 }
600                         }
601                 }
602         }
603         if (!ir_active) {
604                 return;
605         }
607         /*
608          * If TM slot is already being used (highly unlikely), show message and
609          * don't issue the RAID action.
610          */
611         if (slots->m_slot[MPTSAS_TM_SLOT(mpt)] != NULL) {
```



```
612                 mptsas_log(mpt, CE_WARN, "RAID Action slot in use.  Cancelling"
613                     " System Shutdown RAID Action.\n");
614                 return;
615         }
617         /*
618          * Create the cmd and put it in the dedicated TM slot.
619          */
620         cmd = &(mpt->m_event_task_mgmt.m_event_cmd);
621         bzero((caddr_t)cmd, sizeof (*cmd));
622         cmd->cmd_pkt = NULL;
623         cmd->cmd_slot = MPTSAS_TM_SLOT(mpt);
624         slots->m_slot[MPTSAS_TM_SLOT(mpt)] = cmd;
626         /*
627          * Form message for raid action.
628          */
629         action = (pMpi2RaidActionRequest_t)(mpt->m_req_frame +
630             (mpt->m_req_frame_size * cmd->cmd_slot));
631         bzero(action, mpt->m_req_frame_size);
632         action->Function = MPI2_FUNCTION_RAID_ACTION;
633         action->Action = MPI2_RAID_ACTION_SYSTEM_SHUTDOWN_INITIATED;
635         /*
636          * Send RAID Action.
637          * Defaults to MSIxIndex of 0, so check reply q 0 below.
638 #endif /* ! codereview */
639          */
640         (void) ddi_dma_sync(mpt->m_dma_req_frame_hdl, 0, 0,
641             DDI_DMA_SYNC_FORDEV);
642         request_desc = (cmd->cmd_slot << 16) +
575         request_desc_low = (cmd->cmd_slot << 16) +
643             MPI2_REQ_DESCRIPT_FLAGS_DEFAULT_TYPE;
644         MPTSAS_START_CMD(mpt, request_desc);
577         MPTSAS_START_CMD(mpt, request_desc_low, 0);
646         /*
647          * Even though reply does not matter because the system is shutting
648          * down, wait no more than 5 seconds here to get the reply just because
649          * we don't want to leave it hanging if it's coming.  Poll because
650          * interrupts are disabled when this function is called.
651          */
652         rpqp = mpt->m_rep_post_queues;
653 #endif /* ! codereview */
654         for (cnt = 0; cnt < 5000; cnt++) {
655                 /*
656                  * Check for a reply.
657                  */
658                 (void) ddi_dma_sync(mpt->m_dma_post_queue_hdl, 0, 0,
659                     DDI_DMA_SYNC_FORCPU);
661                 reply_desc_union = (pMpi2ReplyDescriptorsUnion_t)
662                     MPTSAS_GET_NEXT_REPLY(rpqp, rpqp->rpq_index);
585                     MPTSAS_GET_NEXT_REPLY(mpt, mpt->m_post_index);
664                 if (ddi_get32(mpt->m_acc_post_queue_hdl,
665                     &reply_desc_union->Words.Low) == 0xFFFFFFFF ||
666                     ddi_get32(mpt->m_acc_post_queue_hdl,
667                     &reply_desc_union->Words.High) == 0xFFFFFFFF) {
668                         drv_usecwait(1000);
669                         continue;
670                 }
672                 /*
673                  * There is a reply.  If it's not an address reply, ignore it.
674                  */
```

```
675                 reply_type = ddi_get8(mpt->m_acc_post_queue_hdl,
676                     &reply_desc_union->Default.ReplyFlags);
677                 reply_type &= MPI2_RPY_DESCRIPT_FLAGS_TYPE_MASK;
678                 if (reply_type != MPI2_RPY_DESCRIPT_FLAGS_ADDRESS_REPLY) {
679                         goto clear_and_continue;
680                 }

682                 /*
683                  * SMID must be the TM slot since that's what we're using for
684                  * this RAID action.  If not, ignore this reply.
685                  */
686                 address_reply =
687                     (pMpi2AddressReplyDescriptor_t)reply_desc_union;
688                 SMID = ddi_get16(mpt->m_acc_post_queue_hdl,
689                     &address_reply->SMID);
690                 if (SMID != MPTSAS_TM_SLOT(mpt)) {
691                         goto clear_and_continue;
692                 }

694                 /*
695                  * If reply frame is not in the proper range ignore it.
696                  */
697                 reply_addr = ddi_get32(mpt->m_acc_post_queue_hdl,
698                     &address_reply->ReplyFrameAddress);
699                 if ((reply_addr < mpt->m_reply_frame_dma_addr) ||
700                     (reply_addr >= (mpt->m_reply_frame_dma_addr +
701                     (mpt->m_reply_frame_size * mpt->m_free_queue_depth))) ||
702                     ((reply_addr - mpt->m_reply_frame_dma_addr) %
703                     mpt->m_reply_frame_size != 0)) {
704                         goto clear_and_continue;
705                 }

707                 /*
708                  * If not a RAID action reply ignore it.
709                  */
710                 (void) ddi_dma_sync(mpt->m_dma_reply_frame_hdl, 0, 0,
711                     DDI_DMA_SYNC_FORCPU);
712                 reply = (pMPI2DefaultReply_t)(mpt->m_reply_frame +
713                     (reply_addr - mpt->m_reply_frame_dma_addr));
714                 function = ddi_get8(mpt->m_acc_reply_frame_hdl,
715                     &reply->Function);
716                 if (function != MPI2_FUNCTION_RAID_ACTION) {
717                         goto clear_and_continue;
718                 }

720                 /*
721                  * Finally, make sure this is the System Shutdown RAID action.
722                  * If not, ignore reply.
723                  */
724                 action_type = ddi_get16(mpt->m_acc_reply_frame_hdl,
725                     &reply->FunctionDependent1);
726                 if (action_type !=
727                     MPI2_RAID_ACTION_SYSTEM_SHUTDOWN_INITIATED) {
728                         goto clear_and_continue;
729                 }
730                 found_reply = TRUE;

732 clear_and_continue:
733                 /*
734                  * Clear the reply descriptor for re-use and increment index.
735                  */
736                 ddi_put64(mpt->m_acc_post_queue_hdl,
737                     &((uint64_t *)(void *)rpqp->rpq_queue)[rpqp->rpq_index],
660                     &((uint64_t *)(void *)mpt->m_post_queue)[mpt->m_post_index],
738                     0xFFFFFFFFFFFFFFFF);
739                 (void) ddi_dma_sync(mpt->m_dma_post_queue_hdl, 0, 0,
```



```
740                     DDI_DMA_SYNC_FORDEV);

742                 /*
743                  * Update the reply index and keep looking for the
666                  * Update the global reply index and keep looking for the
744                  * reply if not found yet.
745                  */
746                 if (++rpqp->rpq_index == mpt->m_post_queue_depth) {
747                         rpqp->rpq_index = 0;
669                 if (++mpt->m_post_index == mpt->m_post_queue_depth) {
670                         mpt->m_post_index = 0;
748                 }

750 #endif /* ! codereview */
751                 ddi_put32(mpt->m_datap, &mpt->m_reg->ReplyPostHostIndex,
752                     rpqp->rpq_index);
672                     mpt->m_post_index);
753                 if (!found_reply) {
754                         continue;
755                 }

757                 break;
758         }

760         /*
761          * clear the used slot as the last step.
762          */
763         slots->m_slot[MPTSAS_TM_SLOT(mpt)] = NULL;
764 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   14649 Thu Jun 12 17:42:17 2014
new/usr/src/uts/common/io/scsi/adapters/mpt_sas3/mptsas3_smhba.c
hg changesets 607a5b46a793..b706c96317c3
Fix ncpus for early boot config
Purge the ack to the interrupt before exiting mptsas_intr()
Changes from code review
Change some obvious references sas -> sas3.
Changes to enable driver to compile.
Header paths, object lists, etc.
**********************************************************
```

```
/*
 * 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) 2009, 2010, Oracle and/or its affiliates. All rights reserved.
 * Copyright 2013 Nexenta Systems, Inc. All rights reserved.
 * Copyright (c) 2014, Tegile Systems Inc. All rights reserved.
#endif /* ! codereview */
 */
/*
 * This file contains SM-HBA support for MPT SAS3 driver
 * This file contains SM-HBA support for MPT SAS driver
 */

#if defined(lint) || defined(DEBUG)
#define MPTSAS_DEBUG
#endif

/*
 * standard header files
 */
#include <sys/note.h>
#include <sys/scsi/scsi.h>
#include <sys/pci.h>
#include <sys/scsi/generic/sas.h>
#include <sys/scsi/impl/scsi_sas.h>

#pragma pack(1)
#include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_type.h>
#include <sys/scsi/adapters/mpt_sas3/mpi/mpi2.h>
#include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_cnfg.h>
#include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_init.h>
#include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_ioc.h>
#include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_sas.h>
#include <sys/scsi/adapters/mpt_sas/mpi/mpi2_type.h>
#include <sys/scsi/adapters/mpt_sas/mpi/mpi2.h>
#include <sys/scsi/adapters/mpt_sas/mpi/mpi2_cnfg.h>
```



```
#include <sys/scsi/adapters/mpt_sas/mpi/mpi2_init.h>
#include <sys/scsi/adapters/mpt_sas/mpi/mpi2_ioc.h>
#include <sys/scsi/adapters/mpt_sas/mpi/mpi2_sas.h>
#pragma pack()

/*
 * private header files.
 */
#include <sys/scsi/adapters/mpt_sas3/mptsas3_var.h>
#include <sys/scsi/adapters/mpt_sas3/mptsas3_smhba.h>
#include <sys/scsi/adapters/mpt_sas/mptsas_var.h>
#include <sys/scsi/adapters/mpt_sas/mptsas_smhba.h>

/*
 * SM - HBA statics
 */
extern char *mptsas_driver_rev;

static void mptsas_smhba_create_phy_props(nvlist_t **, smhba_info_t *, uint8_t,
    uint16_t *);
static void mptsas_smhba_update_phy_props(mptsas_t *, dev_info_t *, nvlist_t **,
    uint8_t);

static void
mptsas_smhba_add_hba_prop(mptsas_t *mpt, data_type_t dt,
    char *prop_name, void *prop_val);

void
mptsas_smhba_show_phy_info(mptsas_t *mpt);

static void
mptsas_smhba_add_hba_prop(mptsas_t *mpt, data_type_t dt,
    char *prop_name, void *prop_val)
{
        ASSERT(mpt != NULL);

        switch (dt) {
        case DATA_TYPE_INT32:
                if (ddi_prop_update_int(DDI_DEV_T_NONE, mpt->m_dip,
                    prop_name, *(int *)prop_val)) {
                        mptsas_log(mpt, CE_WARN,
                            "%s: %s prop update failed", __func__, prop_name);
                }
                break;
        case DATA_TYPE_STRING:
                if (ddi_prop_update_string(DDI_DEV_T_NONE, mpt->m_dip,
                    prop_name, (char *)prop_val)) {
                        mptsas_log(mpt, CE_WARN,
                            "%s: %s prop update failed", __func__, prop_name);
                }
                break;
        default:
                mptsas_log(mpt, CE_WARN, "%s: "
                    "Unhandled datatype(%d) for (%s). Skipping prop update.",
                    __func__, dt, prop_name);
        }
}
______unchanged_portion_omitted_

void
mptsas_create_phy_stats(mptsas_t *mpt, char *iport, dev_info_t *dip)
{
        sas_phy_stats_t         *ps;
        smhba_info_t            *phyp;
        int                     ndata;
        char                    ks_name[KSTAT_STRLEN];
```

```
308         char                    phymask[MPTSAS_MAX_PHYS];
309         int                     i;

311         ASSERT(iport != NULL);
312         ASSERT(mpt != NULL);

314         for (i = 0; i < mpt->m_num_phys; i++) {

316                 bzero(phymask, sizeof (phymask));
317                 (void) sprintf(phymask, "%x", mpt->m_phy_info[i].phy_mask);
318                 if (strcmp(phymask, iport) == 0) {

320                         phyp = &mpt->m_phy_info[i].smhba_info;
321                         mutex_enter(&phyp->phy_mutex);

323                         if (phyp->phy_stats != NULL) {
324                                 mutex_exit(&phyp->phy_mutex);
325                                 /* We've already created this kstat instance */
326                                 continue;
327                         }

329                         ndata = (sizeof (sas_phy_stats_t)/
330                             sizeof (kstat_named_t));
331                         (void) snprintf(ks_name, sizeof (ks_name),
332                             "%s.%llx.%d.%d", ddi_driver_name(dip),
333                             (longlong_t)mpt->un.m_base_wwid,
334                             ddi_get_instance(dip), i);

336                         phyp->phy_stats = kstat_create("mptsas3",
332                         phyp->phy_stats = kstat_create("mptsas",
337                             ddi_get_instance(dip), ks_name, KSTAT_SAS_PHY_CLASS,
338                             KSTAT_TYPE_NAMED, ndata, 0);

340                         if (phyp->phy_stats == NULL) {
341                                 mutex_exit(&phyp->phy_mutex);
342                                 mptsas_log(mpt, CE_WARN,
343                                     "%s: Failed to create %s kstats", __func__,
344                                     ks_name);
345                                 continue;
346                         }

348                         ps = (sas_phy_stats_t *)phyp->phy_stats->ks_data;

350                         kstat_named_init(&ps->seconds_since_last_reset,
351                             "SecondsSinceLastReset", KSTAT_DATA_ULONGLONG);
352                         kstat_named_init(&ps->tx_frames,
353                             "TxFrames", KSTAT_DATA_ULONGLONG);
354                         kstat_named_init(&ps->rx_frames,
355                             "RxFrames", KSTAT_DATA_ULONGLONG);
356                         kstat_named_init(&ps->tx_words,
357                             "TxWords", KSTAT_DATA_ULONGLONG);
358                         kstat_named_init(&ps->rx_words,
359                             "RxWords", KSTAT_DATA_ULONGLONG);
360                         kstat_named_init(&ps->invalid_dword_count,
361                             "InvalidDwordCount", KSTAT_DATA_ULONGLONG);
362                         kstat_named_init(&ps->running_disparity_error_count,
363                             "RunningDisparityErrorCount", KSTAT_DATA_ULONGLONG);
364                         kstat_named_init(&ps->loss_of_dword_sync_count,
365                             "LossofDwordSyncCount", KSTAT_DATA_ULONGLONG);
366                         kstat_named_init(&ps->phy_reset_problem_count,
367                             "PhyResetProblemCount", KSTAT_DATA_ULONGLONG);

369                         phyp->phy_stats->ks_private = phyp;
370                         phyp->phy_stats->ks_update = mptsas_update_phy_stats;
371                         kstat_install(phyp->phy_stats);
372                         mutex_exit(&phyp->phy_mutex);
```



```
373                 }
374         }
375 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   52532 Thu Jun 12 17:42:17 2014
new/usr/src/uts/common/sys/scsi/adapters/mpt_sas3/mpi/mpi2.h
Add support for more than 8 MSI-X interrupts.
Tidy up interrupt assignement and card ID messages.
Initial modifications using the code changes present between
the LSI source code for FreeBSD drivers. Specifically the changes
between from mpslsi-source-17.00.00.00 -> mpslsi-source-03.00.00.00.
This mainly involves using a different scatter/gather element in
frame setup.
**********************************************************
```

```
/*
 * 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) 2000-2012 LSI Corporation.
 * Copyright (c) 2000 to 2009, LSI Corporation.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms of all code within
 * this file that is exclusively owned by LSI, with or without
 * modification, is permitted provided that, in addition to the CDDL 1.0
 * License requirements, the following conditions are met:
 *
 *    Neither the name of the author nor the names of its contributors may be
 *    used to endorse or promote products derived from this software without
 *    specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS 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
 * COPYRIGHT OWNER 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.
 */

/*
 *           Name:  mpi2.h
 *          Title:  MPI Message independent structures and definitions
 *                  including System Interface Register Set and
 *                  scatter/gather formats.
 *  Creation Date:  June 21, 2006
 *
 *  mpi2.h Version:  02.00.xx
 *
 *  NOTE: Names (typedefs, defines, etc.) beginning with an MPI25 or Mpi25
 *        prefix are for use only on MPI v2.5 products, and must not be used
 *        with MPI v2.0 products. Unless otherwise noted, names beginning with
 *        MPI2 or Mpi2 are for use with both MPI v2.0 and MPI v2.5 products.
 *  mpi2.h Version:  02.00.13
 *
 *  Version History
 *  ---------------
 *
 *  Date      Version   Description
 *  --------  --------  ------------------------------------------------------
 *  04-30-07  02.00.00  Corresponds to Fusion-MPT MPI Specification Rev A.
 *  06-04-07  02.00.01  Bumped MPI2_HEADER_VERSION_UNIT.
 *  06-26-07  02.00.02  Bumped MPI2_HEADER_VERSION_UNIT.
 *  08-31-07  02.00.03  Bumped MPI2_HEADER_VERSION_UNIT.
 *                      Moved ReplyPostHostIndex register to offset 0x6C of the
 *                      MPI2_SYSTEM_INTERFACE_REGS and modified the define for
 *                      MPI2_REPLY_POST_HOST_INDEX_OFFSET.
 *                      Added union of request descriptors.
 *                      Added union of reply descriptors.
 *  10-31-07  02.00.04  Bumped MPI2_HEADER_VERSION_UNIT.
 *                      Added define for MPI2_VERSION_02_00.
 *                      Fixed the size of the FunctionDependent5 field in the
 *                      MPI2_DEFAULT_REPLY structure.
 *  12-18-07  02.00.05  Bumped MPI2_HEADER_VERSION_UNIT.
 *                      Removed the MPI-defined Fault Codes and extended the
 *                      product specific codes up to 0xEFFF.
 *                      Added a sixth key value for the WriteSequence register
 *                      and changed the flush value to 0x0.
 *                      Added message function codes for Diagnostic Buffer Post
 *                      and Diagnsotic Release.
 *                      New IOCStatus define: MPI2_IOCSTATUS_DIAGNOSTIC_RELEASED
 *                      Moved MPI2_VERSION_UNION from mpi2_ioc.h.
 *  02-29-08  02.00.06  Bumped MPI2_HEADER_VERSION_UNIT.
 *  03-03-08  02.00.07  Bumped MPI2_HEADER_VERSION_UNIT.
 *  05-21-08  02.00.08  Bumped MPI2_HEADER_VERSION_UNIT.
 *                      Added #defines for marking a reply descriptor as unused.
 *  06-27-08  02.00.09  Bumped MPI2_HEADER_VERSION_UNIT.
 *  10-02-08  02.00.10  Bumped MPI2_HEADER_VERSION_UNIT.
 *                      Moved LUN field defines from mpi2_init.h.
 *  01-19-09  02.00.11  Bumped MPI2_HEADER_VERSION_UNIT.
 *  05-06-09  02.00.12  Bumped MPI2_HEADER_VERSION_UNIT.
 *                      In all request and reply descriptors, replaced VF_ID
 *                      field with MSIxIndex field.
 *                      Removed DevHandle field from
 *                      MPI2_SCSI_IO_SUCCESS_REPLY_DESCRIPTOR and made those
 *                      bytes reserved.
 *                      Added RAID Accelerator functionality.
 *  07-30-09  02.00.13  Bumped MPI2_HEADER_VERSION_UNIT.
 *  --------------------------------------------------------------------------
 */

#ifndef MPI2_H
#define MPI2_H


/*****************************************************************************
*
*        MPI Version Definitions
*
*****************************************************************************/

#define MPI2_VERSION_MAJOR                  (0x02)
```

```
115 #define MPI2_VERSION_MINOR                  (0x00)
118 #define MPI2_VERSION_MAJOR_MASK             (0xFF00)
119 #define MPI2_VERSION_MAJOR_SHIFT            (8)
120 #define MPI2_VERSION_MINOR_MASK             (0x00FF)
121 #define MPI2_VERSION_MINOR_SHIFT            (0)

123 /* major version for all MPI v2.x */
124 #define MPI2_VERSION_MAJOR                  (0x02)

126 /* minor version for MPI v2.0 compatible products */
127 #define MPI2_VERSION_MINOR                  (0x00)
128 #endif /* ! codereview */
129 #define MPI2_VERSION ((MPI2_VERSION_MAJOR << MPI2_VERSION_MAJOR_SHIFT) |   \
130                                     MPI2_VERSION_MINOR)
131 #define MPI2_VERSION_02_00                  (0x0200)


134 /* minor version for MPI v2.5 compatible products */
135 #define MPI25_VERSION_MINOR                 (0x05)
136 #define MPI25_VERSION ((MPI2_VERSION_MAJOR << MPI2_VERSION_MAJOR_SHIFT) |   \
137                                     MPI25_VERSION_MINOR)
138 #define MPI2_VERSION_02_05                  (0x0205)
139 #endif /* ! codereview */

120 #define MPI2_VERSION_02_00                  (0x0200)

142 /* Unit and Dev versioning for this MPI header set */
143 #define MPI2_HEADER_VERSION_UNIT            (0x12)
122 /* versioning for this MPI header set */
123 #define MPI2_HEADER_VERSION_UNIT            (0x0D)
144 #define MPI2_HEADER_VERSION_DEV             (0x00)
145 #define MPI2_HEADER_VERSION_UNIT_MASK       (0xFF00)
146 #define MPI2_HEADER_VERSION_UNIT_SHIFT      (8)
147 #define MPI2_HEADER_VERSION_DEV_MASK        (0x00FF)
148 #define MPI2_HEADER_VERSION_DEV_SHIFT       (0)
149 #define MPI2_HEADER_VERSION ((MPI2_HEADER_VERSION_UNIT << 8) | MPI2_HEADER_VERSI


152 /*****************************************************************************
153 *
154 *        IOC State Definitions
155 *
156 *****************************************************************************/

158 #define MPI2_IOC_STATE_RESET                (0x00000000)
159 #define MPI2_IOC_STATE_READY                (0x10000000)
160 #define MPI2_IOC_STATE_OPERATIONAL          (0x20000000)
161 #define MPI2_IOC_STATE_FAULT                (0x40000000)

163 #define MPI2_IOC_STATE_MASK                 (0xF0000000)
164 #define MPI2_IOC_STATE_SHIFT                (28)

166 /* Fault state range for prodcut specific codes */
167 #define MPI2_FAULT_PRODUCT_SPECIFIC_MIN                 (0x0000)
168 #define MPI2_FAULT_PRODUCT_SPECIFIC_MAX                 (0xEFFF)


171 /*****************************************************************************
172 *
173 *        System Interface Register Definitions
174 *
175 *****************************************************************************/

177 typedef volatile struct _MPI2_SYSTEM_INTERFACE_REGS
178 {
179     U32         Doorbell;                   /* 0x00 */
```



```
180     U32         WriteSequence;              /* 0x04 */
181     U32         HostDiagnostic;             /* 0x08 */
182     U32         Reserved1;                  /* 0x0C */
183     U32         DiagRWData;                 /* 0x10 */
184     U32         DiagRWAddressLow;           /* 0x14 */
185     U32         DiagRWAddressHigh;          /* 0x18 */
186     U32         Reserved2[5];               /* 0x1C */
187     U32         HostInterruptStatus;        /* 0x30 */
188     U32         HostInterruptMask;          /* 0x34 */
189     U32         DCRData;                    /* 0x38 */
190     U32         DCRAddress;                 /* 0x3C */
191     U32         Reserved3[2];               /* 0x40 */
192     U32         ReplyFreeHostIndex;         /* 0x48 */
193     U32         Reserved4[8];               /* 0x4C */
194     U32         ReplyPostHostIndex;         /* 0x6C */
195     U32         Reserved5;                  /* 0x70 */
196     U32         HCBSize;                    /* 0x74 */
197     U32         HCBAddressLow;              /* 0x78 */
198     U32         HCBAddressHigh;             /* 0x7C */
199     U32         Reserved6[16];              /* 0x80 */
200     U32         RequestDescriptorPostLow;   /* 0xC0 */
201     U32         RequestDescriptorPostHigh;  /* 0xC4 */
202     U32         Reserved7[14];              /* 0xC8 */
203     U32         Reserved8[128];             /* 0x100 */
204     U32         Reserved10[3];              /* 0x300 */
205     U32         SuppReplyPostHostIndex[32]; /* 0x30C */
206 #endif /* ! codereview */
207 } MPI2_SYSTEM_INTERFACE_REGS, MPI2_POINTER PTR_MPI2_SYSTEM_INTERFACE_REGS,
208   Mpi2SystemInterfaceRegs_t, MPI2_POINTER pMpi2SystemInterfaceRegs_t;

210 /*
211  * Defines for working with the Doorbell register.
212  */
213 #define MPI2_DOORBELL_OFFSET                    (0x00000000)

215 /* IOC --> System values */
216 #define MPI2_DOORBELL_USED                      (0x08000000)
217 #define MPI2_DOORBELL_WHO_INIT_MASK             (0x07000000)
218 #define MPI2_DOORBELL_WHO_INIT_SHIFT            (24)
219 #define MPI2_DOORBELL_FAULT_CODE_MASK           (0x0000FFFF)
220 #define MPI2_DOORBELL_DATA_MASK                 (0x0000FFFF)

222 /* System --> IOC values */
223 #define MPI2_DOORBELL_FUNCTION_MASK             (0xFF000000)
224 #define MPI2_DOORBELL_FUNCTION_SHIFT            (24)
225 #define MPI2_DOORBELL_ADD_DWORDS_MASK           (0x00FF0000)
226 #define MPI2_DOORBELL_ADD_DWORDS_SHIFT          (16)


229 /*
230  * Defines for the WriteSequence register
231  */
232 #define MPI2_WRITE_SEQUENCE_OFFSET              (0x00000004)
233 #define MPI2_WRSEQ_KEY_VALUE_MASK               (0x0000000F)
234 #define MPI2_WRSEQ_FLUSH_KEY_VALUE              (0x0)
235 #define MPI2_WRSEQ_1ST_KEY_VALUE                (0xF)
236 #define MPI2_WRSEQ_2ND_KEY_VALUE                (0x4)
237 #define MPI2_WRSEQ_3RD_KEY_VALUE                (0xB)
238 #define MPI2_WRSEQ_4TH_KEY_VALUE                (0x2)
239 #define MPI2_WRSEQ_5TH_KEY_VALUE                (0x7)
240 #define MPI2_WRSEQ_6TH_KEY_VALUE                (0xD)

242 /*
243  * Defines for the HostDiagnostic register
244  */
245 #define MPI2_HOST_DIAGNOSTIC_OFFSET             (0x00000008)
```

```
#define MPI2_DIAG_BOOT_DEVICE_SELECT_MASK       (0x00001800)
#define MPI2_DIAG_BOOT_DEVICE_SELECT_DEFAULT    (0x00000000)
#define MPI2_DIAG_BOOT_DEVICE_SELECT_HCDW       (0x00000800)

#define MPI2_DIAG_CLEAR_FLASH_BAD_SIG           (0x00000400)
#define MPI2_DIAG_FORCE_HCB_ON_RESET            (0x00000200)
#define MPI2_DIAG_HCB_MODE                      (0x00000100)
#define MPI2_DIAG_DIAG_WRITE_ENABLE             (0x00000080)
#define MPI2_DIAG_FLASH_BAD_SIG                 (0x00000040)
#define MPI2_DIAG_RESET_HISTORY                 (0x00000020)
#define MPI2_DIAG_DIAG_RW_ENABLE                (0x00000010)
#define MPI2_DIAG_RESET_ADAPTER                 (0x00000004)
#define MPI2_DIAG_HOLD_IOC_RESET                (0x00000002)

/*
 * Offsets for DiagRWData and address
 */
#define MPI2_DIAG_RW_DATA_OFFSET                (0x00000010)
#define MPI2_DIAG_RW_ADDRESS_LOW_OFFSET         (0x00000014)
#define MPI2_DIAG_RW_ADDRESS_HIGH_OFFSET        (0x00000018)

/*
 * Defines for the HostInterruptStatus register
 */
#define MPI2_HOST_INTERRUPT_STATUS_OFFSET       (0x00000030)
#define MPI2_HIS_SYS2IOC_DB_STATUS              (0x80000000)
#define MPI2_HIS_IOP_DOORBELL_STATUS            MPI2_HIS_SYS2IOC_DB_STATUS
#define MPI2_HIS_RESET_IRQ_STATUS               (0x40000000)
#define MPI2_HIS_REPLY_DESCRIPTOR_INTERRUPT     (0x00000008)
#define MPI2_HIS_IOC2SYS_DB_STATUS              (0x00000001)
#define MPI2_HIS_DOORBELL_INTERRUPT             MPI2_HIS_IOC2SYS_DB_STATUS

/*
 * Defines for the HostInterruptMask register
 */
#define MPI2_HOST_INTERRUPT_MASK_OFFSET         (0x00000034)
#define MPI2_HIM_RESET_IRQ_MASK                 (0x40000000)
#define MPI2_HIM_REPLY_INT_MASK                 (0x00000008)
#define MPI2_HIM_RIM                            MPI2_HIM_REPLY_INT_MASK
#define MPI2_HIM_IOC2SYS_DB_MASK                (0x00000001)
#define MPI2_HIM_DIM                            MPI2_HIM_IOC2SYS_DB_MASK

/*
 * Offsets for DCRData and address
 */
#define MPI2_DCR_DATA_OFFSET                    (0x00000038)
#define MPI2_DCR_ADDRESS_OFFSET                 (0x0000003C)

/*
 * Offset for the Reply Free Queue
 */
#define MPI2_REPLY_FREE_HOST_INDEX_OFFSET       (0x00000048)

/*
 * Offset for the Reply Descriptor Post Queue
 */
#define MPI2_REPLY_POST_HOST_INDEX_OFFSET       (0x0000006C)
#define MPI2_REPLY_POST_HOST_INDEX_MASK         (0x00FFFFFF)
#define MPI2_RPHI_MSIX_INDEX_MASK               (0xFF000000)
#define MPI2_RPHI_MSIX_INDEX_SHIFT              (24)
#endif /* ! codereview */

/*
 * Defines for the HCBSize and address
 */
```



```
#define MPI2_HCB_SIZE_OFFSET                    (0x00000074)
#define MPI2_HCB_SIZE_SIZE_MASK                 (0xFFFFF000)
#define MPI2_HCB_SIZE_HCB_ENABLE                (0x00000001)

#define MPI2_HCB_ADDRESS_LOW_OFFSET             (0x00000078)
#define MPI2_HCB_ADDRESS_HIGH_OFFSET            (0x0000007C)

/*
 * Offsets for the Request Queue
 */
#define MPI2_REQUEST_DESCRIPTOR_POST_LOW_OFFSET     (0x000000C0)
#define MPI2_REQUEST_DESCRIPTOR_POST_HIGH_OFFSET    (0x000000C4)

/*
 * Offset for the Supplimentary Host Index Base
 * For use with more than 8 MSI-X interrupts.
 */
#define MPI2_SUP_REPLY_POST_HOST_INDEX_OFFSET   (0x0000030C)

#endif /* ! codereview */

/*****************************************************************************
*
*        Message Descriptors
*
*****************************************************************************/

/* Request Descriptors */

/* Default Request Descriptor */
typedef struct _MPI2_DEFAULT_REQUEST_DESCRIPTOR
{
    U8              RequestFlags;               /* 0x00 */
    U8              MSIxIndex;                  /* 0x01 */
    U16             SMID;                       /* 0x02 */
    U16             LMID;                       /* 0x04 */
    U16             DescriptorTypeDependent;    /* 0x06 */
} MPI2_DEFAULT_REQUEST_DESCRIPTOR,
  MPI2_POINTER PTR_MPI2_DEFAULT_REQUEST_DESCRIPTOR,
  Mpi2DefaultRequestDescriptor_t, MPI2_POINTER pMpi2DefaultRequestDescriptor_t;

/* defines for the RequestFlags field */
#define MPI2_REQ_DESCRIPT_FLAGS_TYPE_MASK               (0x0E)
#define MPI2_REQ_DESCRIPT_FLAGS_SCSI_IO                 (0x00)
#define MPI2_REQ_DESCRIPT_FLAGS_SCSI_TARGET             (0x02)
#define MPI2_REQ_DESCRIPT_FLAGS_HIGH_PRIORITY           (0x06)
#define MPI2_REQ_DESCRIPT_FLAGS_DEFAULT_TYPE            (0x08)
#define MPI2_REQ_DESCRIPT_FLAGS_RAID_ACCELERATOR        (0x0A)
#define MPI25_REQ_DESCRIPT_FLAGS_FAST_PATH_SCSI_IO      (0x0C)
#endif /* ! codereview */

#define MPI2_REQ_DESCRIPT_FLAGS_IOC_FIFO_MARKER (0x01)


/* High Priority Request Descriptor */
typedef struct _MPI2_HIGH_PRIORITY_REQUEST_DESCRIPTOR
{
    U8              RequestFlags;               /* 0x00 */
    U8              MSIxIndex;                  /* 0x01 */
    U16             SMID;                       /* 0x02 */
    U16             LMID;                       /* 0x04 */
    U16             Reserved1;                  /* 0x06 */
} MPI2_HIGH_PRIORITY_REQUEST_DESCRIPTOR,
  MPI2_POINTER PTR_MPI2_HIGH_PRIORITY_REQUEST_DESCRIPTOR,
  Mpi2HighPriorityRequestDescriptor_t,
  MPI2_POINTER pMpi2HighPriorityRequestDescriptor_t;
```

```
380 /* SCSI IO Request Descriptor */
381 typedef struct _MPI2_SCSI_IO_REQUEST_DESCRIPTOR
382 {
383     U8              RequestFlags;               /* 0x00 */
384     U8              MSIxIndex;                  /* 0x01 */
385     U16             SMID;                       /* 0x02 */
386     U16             LMID;                       /* 0x04 */
387     U16             DevHandle;                  /* 0x06 */
388 } MPI2_SCSI_IO_REQUEST_DESCRIPTOR,
389   MPI2_POINTER PTR_MPI2_SCSI_IO_REQUEST_DESCRIPTOR,
390   Mpi2SCSIIORequestDescriptor_t, MPI2_POINTER pMpi2SCSIIORequestDescriptor_t;


393 /* SCSI Target Request Descriptor */
394 typedef struct _MPI2_SCSI_TARGET_REQUEST_DESCRIPTOR
395 {
396     U8              RequestFlags;               /* 0x00 */
397     U8              MSIxIndex;                  /* 0x01 */
398     U16             SMID;                       /* 0x02 */
399     U16             LMID;                       /* 0x04 */
400     U16             IoIndex;                    /* 0x06 */
401 } MPI2_SCSI_TARGET_REQUEST_DESCRIPTOR,
402   MPI2_POINTER PTR_MPI2_SCSI_TARGET_REQUEST_DESCRIPTOR,
403   Mpi2SCSITargetRequestDescriptor_t,
404   MPI2_POINTER pMpi2SCSITargetRequestDescriptor_t;


407 /* RAID Accelerator Request Descriptor */
408 typedef struct _MPI2_RAID_ACCEL_REQUEST_DESCRIPTOR
409 {
410     U8              RequestFlags;               /* 0x00 */
411     U8              MSIxIndex;                  /* 0x01 */
412     U16             SMID;                       /* 0x02 */
413     U16             LMID;                       /* 0x04 */
414     U16             Reserved;                   /* 0x06 */
415 } MPI2_RAID_ACCEL_REQUEST_DESCRIPTOR,
416   MPI2_POINTER PTR_MPI2_RAID_ACCEL_REQUEST_DESCRIPTOR,
417   Mpi2RAIDAcceleratorRequestDescriptor_t,
418   MPI2_POINTER pMpi2RAIDAcceleratorRequestDescriptor_t;


421 /* Fast Path SCSI IO Request Descriptor */
422 typedef MPI2_SCSI_IO_REQUEST_DESCRIPTOR
423     MPI25_FP_SCSI_IO_REQUEST_DESCRIPTOR,
424     MPI2_POINTER PTR_MPI25_FP_SCSI_IO_REQUEST_DESCRIPTOR,
425     Mpi25FastPathSCSIIORequestDescriptor_t,
426     MPI2_POINTER pMpi25FastPathSCSIIORequestDescriptor_t;


429 #endif /* ! codereview */
430 /* union of Request Descriptors */
431 typedef union _MPI2_REQUEST_DESCRIPTOR_UNION
432 {
433     MPI2_DEFAULT_REQUEST_DESCRIPTOR             Default;
434     MPI2_HIGH_PRIORITY_REQUEST_DESCRIPTOR       HighPriority;
435     MPI2_SCSI_IO_REQUEST_DESCRIPTOR             SCSIIO;
436     MPI2_SCSI_TARGET_REQUEST_DESCRIPTOR         SCSITarget;
437     MPI2_RAID_ACCEL_REQUEST_DESCRIPTOR          RAIDAccelerator;
438     MPI25_FP_SCSI_IO_REQUEST_DESCRIPTOR         FastPathSCSIIO;
439 #endif /* ! codereview */
440     U64                                         Words;
441 } MPI2_REQUEST_DESCRIPTOR_UNION, MPI2_POINTER PTR_MPI2_REQUEST_DESCRIPTOR_UNION,
442   Mpi2RequestDescriptorUnion_t, MPI2_POINTER pMpi2RequestDescriptorUnion_t;
```



```
445 /* Reply Descriptors */

447 /* Default Reply Descriptor */
448 typedef struct _MPI2_DEFAULT_REPLY_DESCRIPTOR
449 {
450     U8              ReplyFlags;                 /* 0x00 */
451     U8              MSIxIndex;                  /* 0x01 */
452     U16             DescriptorTypeDependent1;   /* 0x02 */
453     U32             DescriptorTypeDependent2;   /* 0x04 */
454 } MPI2_DEFAULT_REPLY_DESCRIPTOR, MPI2_POINTER PTR_MPI2_DEFAULT_REPLY_DESCRIPTOR,
455   Mpi2DefaultReplyDescriptor_t, MPI2_POINTER pMpi2DefaultReplyDescriptor_t;

457 /* defines for the ReplyFlags field */
458 #define MPI2_RPY_DESCRIPT_FLAGS_TYPE_MASK                   (0x0F)
459 #define MPI2_RPY_DESCRIPT_FLAGS_SCSI_IO_SUCCESS             (0x00)
460 #define MPI2_RPY_DESCRIPT_FLAGS_ADDRESS_REPLY               (0x01)
461 #define MPI2_RPY_DESCRIPT_FLAGS_TARGETASSIST_SUCCESS        (0x02)
462 #define MPI2_RPY_DESCRIPT_FLAGS_TARGET_COMMAND_BUFFER       (0x03)
463 #define MPI2_RPY_DESCRIPT_FLAGS_RAID_ACCELERATOR_SUCCESS    (0x05)
464 #define MPI25_RPY_DESCRIPT_FLAGS_FAST_PATH_SCSI_IO_SUCCESS  (0x06)
465 #endif /* ! codereview */
466 #define MPI2_RPY_DESCRIPT_FLAGS_UNUSED                      (0x0F)

468 /* values for marking a reply descriptor as unused */
469 #define MPI2_RPY_DESCRIPT_UNUSED_WORD0_MARK             (0xFFFFFFFF)
470 #define MPI2_RPY_DESCRIPT_UNUSED_WORD1_MARK             (0xFFFFFFFF)

472 /* Address Reply Descriptor */
473 typedef struct _MPI2_ADDRESS_REPLY_DESCRIPTOR
474 {
475     U8              ReplyFlags;                 /* 0x00 */
476     U8              MSIxIndex;                  /* 0x01 */
477     U16             SMID;                       /* 0x02 */
478     U32             ReplyFrameAddress;          /* 0x04 */
479 } MPI2_ADDRESS_REPLY_DESCRIPTOR, MPI2_POINTER PTR_MPI2_ADDRESS_REPLY_DESCRIPTOR,
480   Mpi2AddressReplyDescriptor_t, MPI2_POINTER pMpi2AddressReplyDescriptor_t;

482 #define MPI2_ADDRESS_REPLY_SMID_INVALID                 (0x00)


485 /* SCSI IO Success Reply Descriptor */
486 typedef struct _MPI2_SCSI_IO_SUCCESS_REPLY_DESCRIPTOR
487 {
488     U8              ReplyFlags;                 /* 0x00 */
489     U8              MSIxIndex;                  /* 0x01 */
490     U16             SMID;                       /* 0x02 */
491     U16             TaskTag;                    /* 0x04 */
492     U16             Reserved1;                  /* 0x06 */
493 } MPI2_SCSI_IO_SUCCESS_REPLY_DESCRIPTOR,
494   MPI2_POINTER PTR_MPI2_SCSI_IO_SUCCESS_REPLY_DESCRIPTOR,
495   Mpi2SCSIIOSuccessReplyDescriptor_t,
496   MPI2_POINTER pMpi2SCSIIOSuccessReplyDescriptor_t;


499 /* TargetAssist Success Reply Descriptor */
500 typedef struct _MPI2_TARGETASSIST_SUCCESS_REPLY_DESCRIPTOR
501 {
502     U8              ReplyFlags;                 /* 0x00 */
503     U8              MSIxIndex;                  /* 0x01 */
504     U16             SMID;                       /* 0x02 */
505     U8              SequenceNumber;             /* 0x04 */
506     U8              Reserved1;                  /* 0x05 */
507     U16             IoIndex;                    /* 0x06 */
508 } MPI2_TARGETASSIST_SUCCESS_REPLY_DESCRIPTOR,
509   MPI2_POINTER PTR_MPI2_TARGETASSIST_SUCCESS_REPLY_DESCRIPTOR,
```

```
  Mpi2TargetAssistSuccessReplyDescriptor_t,
  MPI2_POINTER pMpi2TargetAssistSuccessReplyDescriptor_t;


/* Target Command Buffer Reply Descriptor */
typedef struct _MPI2_TARGET_COMMAND_BUFFER_REPLY_DESCRIPTOR
{
    U8              ReplyFlags;                 /* 0x00 */
    U8              MSIxIndex;                  /* 0x01 */
    U8              VP_ID;                      /* 0x02 */
    U8              Flags;                      /* 0x03 */
    U16             InitiatorDevHandle;         /* 0x04 */
    U16             IoIndex;                    /* 0x06 */
} MPI2_TARGET_COMMAND_BUFFER_REPLY_DESCRIPTOR,
  MPI2_POINTER PTR_MPI2_TARGET_COMMAND_BUFFER_REPLY_DESCRIPTOR,
  Mpi2TargetCommandBufferReplyDescriptor_t,
  MPI2_POINTER pMpi2TargetCommandBufferReplyDescriptor_t;

/* defines for Flags field */
#define MPI2_RPY_DESCRIPT_TCB_FLAGS_PHYNUM_MASK     (0x3F)


/* RAID Accelerator Success Reply Descriptor */
typedef struct _MPI2_RAID_ACCELERATOR_SUCCESS_REPLY_DESCRIPTOR
{
    U8              ReplyFlags;                 /* 0x00 */
    U8              MSIxIndex;                  /* 0x01 */
    U16             SMID;                       /* 0x02 */
    U32             Reserved;                   /* 0x04 */
} MPI2_RAID_ACCELERATOR_SUCCESS_REPLY_DESCRIPTOR,
  MPI2_POINTER PTR_MPI2_RAID_ACCELERATOR_SUCCESS_REPLY_DESCRIPTOR,
  Mpi2RAIDAcceleratorSuccessReplyDescriptor_t,
  MPI2_POINTER pMpi2RAIDAcceleratorSuccessReplyDescriptor_t;


/* Fast Path SCSI IO Success Reply Descriptor */
typedef MPI2_SCSI_IO_SUCCESS_REPLY_DESCRIPTOR
    MPI25_FP_SCSI_IO_SUCCESS_REPLY_DESCRIPTOR,
    MPI2_POINTER PTR_MPI25_FP_SCSI_IO_SUCCESS_REPLY_DESCRIPTOR,
    Mpi25FastPathSCSIIOSuccessReplyDescriptor_t,
    MPI2_POINTER pMpi25FastPathSCSIIOSuccessReplyDescriptor_t;


#endif /* ! codereview */
/* union of Reply Descriptors */
typedef union _MPI2_REPLY_DESCRIPTORS_UNION
{
    MPI2_DEFAULT_REPLY_DESCRIPTOR                   Default;
    MPI2_ADDRESS_REPLY_DESCRIPTOR                   AddressReply;
    MPI2_SCSI_IO_SUCCESS_REPLY_DESCRIPTOR           SCSIIOSuccess;
    MPI2_TARGETASSIST_SUCCESS_REPLY_DESCRIPTOR      TargetAssistSuccess;
    MPI2_TARGET_COMMAND_BUFFER_REPLY_DESCRIPTOR     TargetCommandBuffer;
    MPI2_RAID_ACCELERATOR_SUCCESS_REPLY_DESCRIPTOR  RAIDAcceleratorSuccess;
    MPI25_FP_SCSI_IO_SUCCESS_REPLY_DESCRIPTOR       FastPathSCSIIOSuccess;
#endif /* ! codereview */
    U64                                             Words;
} MPI2_REPLY_DESCRIPTORS_UNION, MPI2_POINTER PTR_MPI2_REPLY_DESCRIPTORS_UNION,
  Mpi2ReplyDescriptorsUnion_t, MPI2_POINTER pMpi2ReplyDescriptorsUnion_t;



/*****************************************************************************
*
*        Message Functions
*             0x80 -> 0x8F reserved for private message use per product
*
```



```
*
*****************************************************************************/

#define MPI2_FUNCTION_SCSI_IO_REQUEST               (0x00) /* SCSI IO */
#define MPI2_FUNCTION_SCSI_TASK_MGMT                (0x01) /* SCSI Task Manageme
#define MPI2_FUNCTION_IOC_INIT                      (0x02) /* IOC Init */
#define MPI2_FUNCTION_IOC_FACTS                     (0x03) /* IOC Facts */
#define MPI2_FUNCTION_CONFIG                        (0x04) /* Configuration */
#define MPI2_FUNCTION_PORT_FACTS                    (0x05) /* Port Facts */
#define MPI2_FUNCTION_PORT_ENABLE                   (0x06) /* Port Enable */
#define MPI2_FUNCTION_EVENT_NOTIFICATION            (0x07) /* Event Notification
#define MPI2_FUNCTION_EVENT_ACK                     (0x08) /* Event Acknowledge
#define MPI2_FUNCTION_FW_DOWNLOAD                   (0x09) /* FW Download */
#define MPI2_FUNCTION_TARGET_ASSIST                 (0x0B) /* Target Assist */
#define MPI2_FUNCTION_TARGET_STATUS_SEND            (0x0C) /* Target Status Send
#define MPI2_FUNCTION_TARGET_MODE_ABORT             (0x0D) /* Target Mode Abort
#define MPI2_FUNCTION_FW_UPLOAD                     (0x12) /* FW Upload */
#define MPI2_FUNCTION_RAID_ACTION                   (0x15) /* RAID Action */
#define MPI2_FUNCTION_RAID_SCSI_IO_PASSTHROUGH      (0x16) /* SCSI IO RAID Passt
#define MPI2_FUNCTION_TOOLBOX                       (0x17) /* Toolbox */
#define MPI2_FUNCTION_SCSI_ENCLOSURE_PROCESSOR      (0x18) /* SCSI Enclosure Pro
#define MPI2_FUNCTION_SMP_PASSTHROUGH               (0x1A) /* SMP Passthrough */
#define MPI2_FUNCTION_SAS_IO_UNIT_CONTROL           (0x1B) /* SAS IO Unit Contro
#define MPI2_FUNCTION_SATA_PASSTHROUGH              (0x1C) /* SATA Passthrough *
#define MPI2_FUNCTION_DIAG_BUFFER_POST              (0x1D) /* Diagnostic Buffer
#define MPI2_FUNCTION_DIAG_RELEASE                  (0x1E) /* Diagnostic Release
#define MPI2_FUNCTION_TARGET_CMD_BUF_BASE_POST      (0x24) /* Target Command Buf
#define MPI2_FUNCTION_TARGET_CMD_BUF_LIST_POST      (0x25) /* Target Command Buf
#define MPI2_FUNCTION_RAID_ACCELERATOR              (0x2C) /* RAID Accelerator *
#define MPI2_FUNCTION_HOST_BASED_DISCOVERY_ACTION   (0x2F) /* Host Based Discove
#define MPI2_FUNCTION_PWR_MGMT_CONTROL              (0x30) /* Power Management C
#define MPI2_FUNCTION_MIN_PRODUCT_SPECIFIC          (0xF0) /* beginning of produ
#define MPI2_FUNCTION_MAX_PRODUCT_SPECIFIC          (0xFF) /* end of product-spe
#endif /* ! codereview */



/* Doorbell functions */
#define MPI2_FUNCTION_IOC_MESSAGE_UNIT_RESET        (0x40)
/* #define MPI2_FUNCTION_IO_UNIT_RESET                 (0x41) */
#define MPI2_FUNCTION_HANDSHAKE                     (0x42)


/*****************************************************************************
*
*        IOC Status Values
*
*****************************************************************************/

/* mask for IOCStatus status value */
#define MPI2_IOCSTATUS_MASK                     (0x7FFF)

/*****************************************************************************
*  Common IOCStatus values for all replies
*****************************************************************************/

#define MPI2_IOCSTATUS_SUCCESS                      (0x0000)
#define MPI2_IOCSTATUS_INVALID_FUNCTION             (0x0001)
#define MPI2_IOCSTATUS_BUSY                         (0x0002)
#define MPI2_IOCSTATUS_INVALID_SGL                  (0x0003)
#define MPI2_IOCSTATUS_INTERNAL_ERROR               (0x0004)
#define MPI2_IOCSTATUS_INVALID_VPID                 (0x0005)
#define MPI2_IOCSTATUS_INSUFFICIENT_RESOURCES       (0x0006)
#define MPI2_IOCSTATUS_INVALID_FIELD                (0x0007)
#define MPI2_IOCSTATUS_INVALID_STATE                (0x0008)
#define MPI2_IOCSTATUS_OP_STATE_NOT_SUPPORTED       (0x0009)
```

```
642 /*****************************************************************************
643 *  Config IOCStatus values
644 *****************************************************************************/

646 #define MPI2_IOCSTATUS_CONFIG_INVALID_ACTION        (0x0020)
647 #define MPI2_IOCSTATUS_CONFIG_INVALID_TYPE          (0x0021)
648 #define MPI2_IOCSTATUS_CONFIG_INVALID_PAGE          (0x0022)
649 #define MPI2_IOCSTATUS_CONFIG_INVALID_DATA          (0x0023)
650 #define MPI2_IOCSTATUS_CONFIG_NO_DEFAULTS           (0x0024)
651 #define MPI2_IOCSTATUS_CONFIG_CANT_COMMIT           (0x0025)

653 /*****************************************************************************
654 *  SCSI IO Reply
655 *****************************************************************************/

657 #define MPI2_IOCSTATUS_SCSI_RECOVERED_ERROR         (0x0040)
658 #define MPI2_IOCSTATUS_SCSI_INVALID_DEVHANDLE       (0x0042)
659 #define MPI2_IOCSTATUS_SCSI_DEVICE_NOT_THERE        (0x0043)
660 #define MPI2_IOCSTATUS_SCSI_DATA_OVERRUN            (0x0044)
661 #define MPI2_IOCSTATUS_SCSI_DATA_UNDERRUN           (0x0045)
662 #define MPI2_IOCSTATUS_SCSI_IO_DATA_ERROR           (0x0046)
663 #define MPI2_IOCSTATUS_SCSI_PROTOCOL_ERROR          (0x0047)
664 #define MPI2_IOCSTATUS_SCSI_TASK_TERMINATED         (0x0048)
665 #define MPI2_IOCSTATUS_SCSI_RESIDUAL_MISMATCH       (0x0049)
666 #define MPI2_IOCSTATUS_SCSI_TASK_MGMT_FAILED        (0x004A)
667 #define MPI2_IOCSTATUS_SCSI_IOC_TERMINATED          (0x004B)
668 #define MPI2_IOCSTATUS_SCSI_EXT_TERMINATED          (0x004C)

670 /*****************************************************************************
671 *  For use by SCSI Initiator and SCSI Target end-to-end data protection
672 *****************************************************************************/

674 #define MPI2_IOCSTATUS_EEDP_GUARD_ERROR             (0x004D)
675 #define MPI2_IOCSTATUS_EEDP_REF_TAG_ERROR           (0x004E)
676 #define MPI2_IOCSTATUS_EEDP_APP_TAG_ERROR           (0x004F)

678 /*****************************************************************************
679 *  SCSI Target values
680 *****************************************************************************/

682 #define MPI2_IOCSTATUS_TARGET_INVALID_IO_INDEX      (0x0062)
683 #define MPI2_IOCSTATUS_TARGET_ABORTED               (0x0063)
684 #define MPI2_IOCSTATUS_TARGET_NO_CONN_RETRYABLE     (0x0064)
685 #define MPI2_IOCSTATUS_TARGET_NO_CONNECTION         (0x0065)
686 #define MPI2_IOCSTATUS_TARGET_XFER_COUNT_MISMATCH   (0x006A)
687 #define MPI2_IOCSTATUS_TARGET_DATA_OFFSET_ERROR     (0x006D)
688 #define MPI2_IOCSTATUS_TARGET_TOO_MUCH_WRITE_DATA   (0x006E)
689 #define MPI2_IOCSTATUS_TARGET_IU_TOO_SHORT          (0x006F)
690 #define MPI2_IOCSTATUS_TARGET_ACK_NAK_TIMEOUT       (0x0070)
691 #define MPI2_IOCSTATUS_TARGET_NAK_RECEIVED          (0x0071)

693 /*****************************************************************************
694 *  Serial Attached SCSI values
695 *****************************************************************************/

697 #define MPI2_IOCSTATUS_SAS_SMP_REQUEST_FAILED       (0x0090)
698 #define MPI2_IOCSTATUS_SAS_SMP_DATA_OVERRUN         (0x0091)

700 /*****************************************************************************
701 *  Diagnostic Buffer Post / Diagnostic Release values
702 *****************************************************************************/

704 #define MPI2_IOCSTATUS_DIAGNOSTIC_RELEASED          (0x00A0)

706 /*****************************************************************************
```



```
707 *  RAID Accelerator values
708 *****************************************************************************/

710 #define MPI2_IOCSTATUS_RAID_ACCEL_ERROR             (0x00B0)

712 /*****************************************************************************
713 *  IOCStatus flag to indicate that log info is available
714 *****************************************************************************/

716 #define MPI2_IOCSTATUS_FLAG_LOG_INFO_AVAILABLE      (0x8000)

718 /*****************************************************************************
719 *  IOCLogInfo Types
720 *****************************************************************************/

722 #define MPI2_IOCLOGINFO_TYPE_MASK               (0xF0000000)
723 #define MPI2_IOCLOGINFO_TYPE_SHIFT              (28)
724 #define MPI2_IOCLOGINFO_TYPE_NONE               (0x0)
725 #define MPI2_IOCLOGINFO_TYPE_SCSI               (0x1)
726 #define MPI2_IOCLOGINFO_TYPE_FC                 (0x2)
727 #define MPI2_IOCLOGINFO_TYPE_SAS                (0x3)
728 #define MPI2_IOCLOGINFO_TYPE_ISCSI              (0x4)
729 #define MPI2_IOCLOGINFO_LOG_DATA_MASK           (0x0FFFFFFF)


732 /*****************************************************************************
733 *
734 *        Standard Message Structures
735 *
736 *****************************************************************************/

738 /*****************************************************************************
739 * Request Message Header for all request messages
740 *****************************************************************************/

742 typedef struct _MPI2_REQUEST_HEADER
743 {
744     U16             FunctionDependent1;         /* 0x00 */
745     U8              ChainOffset;                /* 0x02 */
746     U8              Function;                   /* 0x03 */
747     U16             FunctionDependent2;         /* 0x04 */
748     U8              FunctionDependent3;         /* 0x06 */
749     U8              MsgFlags;                   /* 0x07 */
750     U8              VP_ID;                      /* 0x08 */
751     U8              VF_ID;                      /* 0x09 */
752     U16             Reserved1;                  /* 0x0A */
753 } MPI2_REQUEST_HEADER, MPI2_POINTER PTR_MPI2_REQUEST_HEADER,
_____unchanged_portion_omitted_
838   Mpi2SGESimpleUnion_t, MPI2_POINTER pMpi2SGESimpleUnion_t;


841 /*****************************************************************************
842 *  MPI Chain Element structures - for MPI v2.0 products only
411 *  MPI Chain Element structures
843 *****************************************************************************/

845 typedef struct _MPI2_SGE_CHAIN32
846 {
847     U16                     Length;
848     U8                      NextChainOffset;
849     U8                      Flags;
850     U32                     Address;
851 } MPI2_SGE_CHAIN32, MPI2_POINTER PTR_MPI2_SGE_CHAIN32,
_____unchanged_portion_omitted_
874   Mpi2SGEChainUnion_t, MPI2_POINTER pMpi2SGEChainUnion_t;
```

```
 877 /*****************************************************************************
 878 *  MPI Transaction Context Element structures - for MPI v2.0 products only
 447 *  MPI Transaction Context Element structures
 879 *****************************************************************************/

 881 typedef struct _MPI2_SGE_TRANSACTION32
 882 {
 883     U8                      Reserved;
 884     U8                      ContextSize;
 885     U8                      DetailsLength;
 886     U8                      Flags;
 887     U32                     TransactionContext[1];
 888     U32                     TransactionDetails[1];
 889 } MPI2_SGE_TRANSACTION32, MPI2_POINTER PTR_MPI2_SGE_TRANSACTION32,
_____unchanged_portion_omitted_
 940   Mpi2SGETransactionUnion_t, MPI2_POINTER pMpi2SGETransactionUnion_t;


 943 /*****************************************************************************
 944 *  MPI SGE union for IO SGL's - for MPI v2.0 products only
 513 *  MPI SGE union for IO SGL's
 945 *****************************************************************************/

 947 typedef struct _MPI2_MPI_SGE_IO_UNION
 948 {
 949     union
 950     {
 951         MPI2_SGE_SIMPLE_UNION   Simple;
 952         MPI2_SGE_CHAIN_UNION    Chain;
 953     } u;
 954 } MPI2_MPI_SGE_IO_UNION, MPI2_POINTER PTR_MPI2_MPI_SGE_IO_UNION,
 955   Mpi2MpiSGEIOUnion_t, MPI2_POINTER pMpi2MpiSGEIOUnion_t;


 958 /*****************************************************************************
 959 *  MPI SGE union for SGL's with Simple and Transaction elements - for MPI v2.0 p
 528 *  MPI SGE union for SGL's with Simple and Transaction elements
 960 *****************************************************************************/

 962 typedef struct _MPI2_SGE_TRANS_SIMPLE_UNION
 963 {
 964     union
 965     {
 966         MPI2_SGE_SIMPLE_UNION       Simple;
 967         MPI2_SGE_TRANSACTION_UNION  Transaction;
 968     } u;
 969 } MPI2_SGE_TRANS_SIMPLE_UNION, MPI2_POINTER PTR_MPI2_SGE_TRANS_SIMPLE_UNION,
_____unchanged_portion_omitted_
 986   Mpi2MpiSgeUnion_t, MPI2_POINTER pMpi2MpiSgeUnion_t;


 989 /*****************************************************************************
 990 *  MPI SGE field definition and masks
 991 *****************************************************************************/

 993 /* Flags field bit definitions */

 995 #define MPI2_SGE_FLAGS_LAST_ELEMENT             (0x80)
 996 #define MPI2_SGE_FLAGS_END_OF_BUFFER            (0x40)
 997 #define MPI2_SGE_FLAGS_ELEMENT_TYPE_MASK        (0x30)
 998 #define MPI2_SGE_FLAGS_LOCAL_ADDRESS            (0x08)
 999 #define MPI2_SGE_FLAGS_DIRECTION                (0x04)
1000 #define MPI2_SGE_FLAGS_ADDRESS_SIZE             (0x02)
1001 #define MPI2_SGE_FLAGS_END_OF_LIST              (0x01)
```



```
1003 #define MPI2_SGE_FLAGS_SHIFT                    (24)

1005 #define MPI2_SGE_LENGTH_MASK                    (0x00FFFFFF)
1006 #define MPI2_SGE_CHAIN_LENGTH_MASK              (0x0000FFFF)

1008 /* Element Type */

1010 #define MPI2_SGE_FLAGS_TRANSACTION_ELEMENT      (0x00) /* for MPI v2.0 products
 579 #define MPI2_SGE_FLAGS_TRANSACTION_ELEMENT      (0x00)
1011 #define MPI2_SGE_FLAGS_SIMPLE_ELEMENT           (0x10)
1012 #define MPI2_SGE_FLAGS_CHAIN_ELEMENT            (0x30) /* for MPI v2.0 products
 581 #define MPI2_SGE_FLAGS_CHAIN_ELEMENT            (0x30)
1013 #define MPI2_SGE_FLAGS_ELEMENT_MASK             (0x30)

1015 /* Address location */

1017 #define MPI2_SGE_FLAGS_SYSTEM_ADDRESS           (0x00)

1019 /* Direction */

1021 #define MPI2_SGE_FLAGS_IOC_TO_HOST              (0x00)
1022 #define MPI2_SGE_FLAGS_HOST_TO_IOC              (0x04)

1024 #define MPI2_SGE_FLAGS_DEST                     (MPI2_SGE_FLAGS_IOC_TO_HOST)
1025 #define MPI2_SGE_FLAGS_SOURCE                   (MPI2_SGE_FLAGS_HOST_TO_IOC)

1027 #endif /* ! codereview */
1028 /* Address Size */

1030 #define MPI2_SGE_FLAGS_32_BIT_ADDRESSING        (0x00)
1031 #define MPI2_SGE_FLAGS_64_BIT_ADDRESSING        (0x02)

1033 /* Context Size */

1035 #define MPI2_SGE_FLAGS_32_BIT_CONTEXT           (0x00)
1036 #define MPI2_SGE_FLAGS_64_BIT_CONTEXT           (0x02)
1037 #define MPI2_SGE_FLAGS_96_BIT_CONTEXT           (0x04)
1038 #define MPI2_SGE_FLAGS_128_BIT_CONTEXT          (0x06)

1040 #define MPI2_SGE_CHAIN_OFFSET_MASK              (0x00FF0000)
1041 #define MPI2_SGE_CHAIN_OFFSET_SHIFT             (16)

1043 /*****************************************************************************
1044 *  MPI SGE operation Macros
1045 *****************************************************************************/

1047 /* SIMPLE FlagsLength manipulations... */
1048 #define MPI2_SGE_SET_FLAGS(f)           ((U32)(f) << MPI2_SGE_FLAGS_SHIFT)
1049 #define MPI2_SGE_GET_FLAGS(f)           (((f) & ~MPI2_SGE_LENGTH_MASK) >> MPI2_SG
1050 #define MPI2_SGE_LENGTH(f)              ((f) & MPI2_SGE_LENGTH_MASK)
1051 #define MPI2_SGE_CHAIN_LENGTH(f)        ((f) & MPI2_SGE_CHAIN_LENGTH_MASK)

1053 #define MPI2_SGE_SET_FLAGS_LENGTH(f,l) (MPI2_SGE_SET_FLAGS(f) | MPI2_SGE_LENGTH(

1055 #define MPI2_pSGE_GET_FLAGS(psg)            MPI2_SGE_GET_FLAGS((psg)->FlagsLengt
1056 #define MPI2_pSGE_GET_LENGTH(psg)           MPI2_SGE_LENGTH((psg)->FlagsLength)
1057 #define MPI2_pSGE_SET_FLAGS_LENGTH(psg,f,l) (psg)->FlagsLength = MPI2_SGE_SET_FL

1059 /* CAUTION - The following are READ-MODIFY-WRITE! */
1060 #define MPI2_pSGE_SET_FLAGS(psg,f)      (psg)->FlagsLength |= MPI2_SGE_SET_FLAGS
1061 #define MPI2_pSGE_SET_LENGTH(psg,l)     (psg)->FlagsLength |= MPI2_SGE_LENGTH(l)

1063 #define MPI2_GET_CHAIN_OFFSET(x)    ((x & MPI2_SGE_CHAIN_OFFSET_MASK) >> MPI2_SG


1066 /*****************************************************************************
```

```
1067 *
1068 *        Fusion-MPT IEEE Scatter Gather Elements
1069 *
1070 ****************************************************************************/

1072 /****************************************************************************
1073 *  IEEE Simple Element structures
1074 ****************************************************************************/

1076 /* MPI2_IEEE_SGE_SIMPLE32 is for MPI v2.0 products only */
1077 #endif /* ! codereview */
1078 typedef struct _MPI2_IEEE_SGE_SIMPLE32
1079 {
1080     U32                     Address;
1081     U32                     FlagsLength;
1082 } MPI2_IEEE_SGE_SIMPLE32, MPI2_POINTER PTR_MPI2_IEEE_SGE_SIMPLE32,
1083   Mpi2IeeeSgeSimple32_t, MPI2_POINTER pMpi2IeeeSgeSimple32_t;

1085 typedef struct _MPI2_IEEE_SGE_SIMPLE64
1086 {
1087     U64                     Address;
1088     U32                     Length;
1089     U16                     Reserved1;
1090     U8                      Reserved2;
1091     U8                      Flags;
1092 } MPI2_IEEE_SGE_SIMPLE64, MPI2_POINTER PTR_MPI2_IEEE_SGE_SIMPLE64,
1093   Mpi2IeeeSgeSimple64_t, MPI2_POINTER pMpi2IeeeSgeSimple64_t;

1095 typedef union _MPI2_IEEE_SGE_SIMPLE_UNION
1096 {
1097     MPI2_IEEE_SGE_SIMPLE32  Simple32;
1098     MPI2_IEEE_SGE_SIMPLE64  Simple64;
1099 } MPI2_IEEE_SGE_SIMPLE_UNION, MPI2_POINTER PTR_MPI2_IEEE_SGE_SIMPLE_UNION,
1100   Mpi2IeeeSgeSimpleUnion_t, MPI2_POINTER pMpi2IeeeSgeSimpleUnion_t;


1103 /****************************************************************************
1104 *  IEEE Chain Element structures
1105 ****************************************************************************/

1107 /* MPI2_IEEE_SGE_CHAIN32 is for MPI v2.0 products only */
1108 #endif /* ! codereview */
1109 typedef MPI2_IEEE_SGE_SIMPLE32  MPI2_IEEE_SGE_CHAIN32;

1111 /* MPI2_IEEE_SGE_CHAIN64 is for MPI v2.0 products only */
1112 #endif /* ! codereview */
1113 typedef MPI2_IEEE_SGE_SIMPLE64  MPI2_IEEE_SGE_CHAIN64;

1115 typedef union _MPI2_IEEE_SGE_CHAIN_UNION
1116 {
1117     MPI2_IEEE_SGE_CHAIN32   Chain32;
1118     MPI2_IEEE_SGE_CHAIN64   Chain64;
1119 } MPI2_IEEE_SGE_CHAIN_UNION, MPI2_POINTER PTR_MPI2_IEEE_SGE_CHAIN_UNION,
1120   Mpi2IeeeSgeChainUnion_t, MPI2_POINTER pMpi2IeeeSgeChainUnion_t;

1122 /* MPI25_IEEE_SGE_CHAIN64 is for MPI v2.5 products only */
1123 typedef struct _MPI25_IEEE_SGE_CHAIN64
1124 {
1125     U64                     Address;
1126     U32                     Length;
1127     U16                     Reserved1;
1128     U8                      NextChainOffset;
1129     U8                      Flags;
1130 } MPI25_IEEE_SGE_CHAIN64, MPI2_POINTER PTR_MPI25_IEEE_SGE_CHAIN64,
1131   Mpi25IeeeSgeChain64_t, MPI2_POINTER pMpi25IeeeSgeChain64_t;
```



```
1133 #endif /* ! codereview */

1135 /****************************************************************************
1136 *  All IEEE SGE types union
1137 ****************************************************************************/

1139 /* MPI2_IEEE_SGE_UNION is for MPI v2.0 products only */
1140 #endif /* ! codereview */
1141 typedef struct _MPI2_IEEE_SGE_UNION
1142 {
1143     union
1144     {
1145         MPI2_IEEE_SGE_SIMPLE_UNION  Simple;
1146         MPI2_IEEE_SGE_CHAIN_UNION   Chain;
1147     } u;
1148 } MPI2_IEEE_SGE_UNION, MPI2_POINTER PTR_MPI2_IEEE_SGE_UNION,
1149   Mpi2IeeeSgeUnion_t, MPI2_POINTER pMpi2IeeeSgeUnion_t;


1152 /****************************************************************************
1153 *  IEEE SGE union for IO SGL's
1154 ****************************************************************************/

1156 typedef union _MPI25_SGE_IO_UNION
1157 {
1158     MPI2_IEEE_SGE_SIMPLE64      IeeeSimple;
1159     MPI25_IEEE_SGE_CHAIN64      IeeeChain;
1160 } MPI25_SGE_IO_UNION, MPI2_POINTER PTR_MPI25_SGE_IO_UNION,
1161   Mpi25SGEIOUnion_t, MPI2_POINTER pMpi25SGEIOUnion_t;


1164 /****************************************************************************
1165 #endif /* ! codereview */
1166 *  IEEE SGE field definitions and masks
1167 ****************************************************************************/

1169 /* Flags field bit definitions */

1171 #define MPI2_IEEE_SGE_FLAGS_ELEMENT_TYPE_MASK   (0x80)
1172 #define MPI25_IEEE_SGE_FLAGS_END_OF_LIST        (0x40)
1173 #endif /* ! codereview */

1175 #define MPI2_IEEE32_SGE_FLAGS_SHIFT             (24)

1177 #define MPI2_IEEE32_SGE_LENGTH_MASK             (0x00FFFFFF)

1179 /* Element Type */

1181 #define MPI2_IEEE_SGE_FLAGS_SIMPLE_ELEMENT      (0x00)
1182 #define MPI2_IEEE_SGE_FLAGS_CHAIN_ELEMENT       (0x80)

1184 /* Data Location Address Space */

1186 #define MPI2_IEEE_SGE_FLAGS_ADDR_MASK           (0x03)
1187 #define MPI2_IEEE_SGE_FLAGS_SYSTEM_ADDR         (0x00) /* IEEE Simple Element on
1188 #define MPI2_IEEE_SGE_FLAGS_IOCDDR_ADDR         (0x01) /* IEEE Simple Element on
 593 #define MPI2_IEEE_SGE_FLAGS_SYSTEM_ADDR         (0x00)
 594 #define MPI2_IEEE_SGE_FLAGS_IOCDDR_ADDR         (0x01)
1189 #define MPI2_IEEE_SGE_FLAGS_IOCPLB_ADDR         (0x02)
1190 #define MPI2_IEEE_SGE_FLAGS_IOCPLBNTA_ADDR      (0x03) /* IEEE Simple Element on
1191 #define MPI2_IEEE_SGE_FLAGS_SYSTEMPLBCPI_ADDR   (0x03) /* IEEE Chain Element onl
 596 #define MPI2_IEEE_SGE_FLAGS_IOCPLBNTA_ADDR      (0x03)


1194 /****************************************************************************
1195 *  IEEE SGE operation Macros
```

```
1196 *****************************************************************************/

1198 /* SIMPLE FlagsLength manipulations... */
1199 #define MPI2_IEEE32_SGE_SET_FLAGS(f)     ((U32)(f) << MPI2_IEEE32_SGE_FLAGS_SHIF
1200 #define MPI2_IEEE32_SGE_GET_FLAGS(f)     (((f) & ~MPI2_IEEE32_SGE_LENGTH_MASK) >
1201 #define MPI2_IEEE32_SGE_LENGTH(f)        ((f) & MPI2_IEEE32_SGE_LENGTH_MASK)

1203 #define MPI2_IEEE32_SGE_SET_FLAGS_LENGTH(f, l)      (MPI2_IEEE32_SGE_SET_FLAGS(f

1205 #define MPI2_IEEE32_pSGE_GET_FLAGS(psg)             MPI2_IEEE32_SGE_GET_FLAGS((p
1206 #define MPI2_IEEE32_pSGE_GET_LENGTH(psg)            MPI2_IEEE32_SGE_LENGTH((psg)
1207 #define MPI2_IEEE32_pSGE_SET_FLAGS_LENGTH(psg,f,l)  (psg)->FlagsLength = MPI2_IE

1209 /* CAUTION - The following are READ-MODIFY-WRITE! */
1210 #define MPI2_IEEE32_pSGE_SET_FLAGS(psg,f)    (psg)->FlagsLength |= MPI2_IEEE32_S
1211 #define MPI2_IEEE32_pSGE_SET_LENGTH(psg,l)   (psg)->FlagsLength |= MPI2_IEEE32_S




1216 /*****************************************************************************
1217 *
1218 *        Fusion-MPT MPI/IEEE Scatter Gather Unions
1219 *
1220 *****************************************************************************/

1222 typedef union _MPI2_SIMPLE_SGE_UNION
1223 {
1224     MPI2_SGE_SIMPLE_UNION       MpiSimple;
1225     MPI2_IEEE_SGE_SIMPLE_UNION  IeeeSimple;
1226 } MPI2_SIMPLE_SGE_UNION, MPI2_POINTER PTR_MPI2_SIMPLE_SGE_UNION,
______unchanged_portion_omitted_
```

```
**********************************************************
  144624 Thu Jun 12 17:42:17 2014
new/usr/src/uts/common/sys/scsi/adapters/mpt_sas3/mpi/mpi2_cnfg.h
Initial modifications using the code changes present between
the LSI source code for FreeBSD drivers. Specifically the changes
between from mpslsi-source-17.00.00.00 -> mpslsi-source-03.00.00.00.
This mainly involves using a different scatter/gather element in
frame setup.
**********************************************************
/*
 * 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) 2000-2012 LSI Corporation.
 * Copyright (c) 2000 to 2009, LSI Corporation.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms of all code within
 * this file that is exclusively owned by LSI, with or without
 * modification, is permitted provided that, in addition to the CDDL 1.0
 * License requirements, the following conditions are met:
 *
 *    Neither the name of the author nor the names of its contributors may be
 *    used to endorse or promote products derived from this software without
 *    specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS 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
 * COPYRIGHT OWNER 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.
 */

/*
 *          Name:  mpi2_cnfg.h
 *         Title:  MPI Configuration messages and pages
 *  Creation Date:  November 10, 2006
 *
 *  mpi2_cnfg.h Version:  02.00.xx
 *
 *  NOTE: Names (typedefs, defines, etc.) beginning with an MPI25 or Mpi25
```



```
 *        prefix are for use only on MPI v2.5 products, and must not be used
 *        with MPI v2.0 products. Unless otherwise noted, names beginning with
 *        MPI2 or Mpi2 are for use with both MPI v2.0 and MPI v2.5 products.
 *    mpi2_cnfg.h Version:  02.00.12
 *
 *  Version History
 *  ---------------
 *
 *  Date      Version   Description
 *  --------  --------  ------------------------------------------------------
 *  04-30-07  02.00.00  Corresponds to Fusion-MPT MPI Specification Rev A.
 *  06-04-07  02.00.01  Added defines for SAS IO Unit Page 2 PhyFlags.
 *                      Added Manufacturing Page 11.
 *                      Added MPI2_SAS_EXPANDER0_FLAGS_CONNECTOR_END_DEVICE
 *                      define.
 *  06-26-07  02.00.02  Adding generic structure for product-specific
 *                      Manufacturing pages: MPI2_CONFIG_PAGE_MANUFACTURING_PS.
 *                      Rework of BIOS Page 2 configuration page.
 *                      Fixed MPI2_BIOSPAGE2_BOOT_DEVICE to be a union of the
 *                      forms.
 *                      Added configuration pages IOC Page 8 and Driver
 *                      Persistent Mapping Page 0.
 *  08-31-07  02.00.03  Modified configuration pages dealing with Integrated
 *                      RAID (Manufacturing Page 4, RAID Volume Pages 0 and 1,
 *                      RAID Physical Disk Pages 0 and 1, RAID Configuration
 *                      Page 0).
 *                      Added new value for AccessStatus field of SAS Device
 *                      Page 0 (_SATA_NEEDS_INITIALIZATION).
 *  10-31-07  02.00.04  Added missing SEPDevHandle field to
 *                      MPI2_CONFIG_PAGE_SAS_ENCLOSURE_0.
 *  12-18-07  02.00.05  Modified IO Unit Page 0 to use 32-bit version fields for
 *                      NVDATA.
 *                      Modified IOC Page 7 to use masks and added field for
 *                      SASBroadcastPrimitiveMasks.
 *                      Added MPI2_CONFIG_PAGE_BIOS_4.
 *                      Added MPI2_CONFIG_PAGE_LOG_0.
 *  02-29-08  02.00.06  Modified various names to make them 32-character unique.
 *                      Added SAS Device IDs.
 *                      Updated Integrated RAID configuration pages including
 *                      Manufacturing Page 4, IOC Page 6, and RAID Configuration
 *                      Page 0.
 *  05-21-08  02.00.07  Added define MPI2_MANPAGE4_MIX_SSD_SAS_SATA.
 *                      Added define MPI2_MANPAGE4_PHYSDISK_128MB_COERCION.
 *                      Fixed define MPI2_IOCPAGE8_FLAGS_ENCLOSURE_SLOT_MAPPING.
 *                      Added missing MaxNumRoutedSasAddresses field to
 *                      MPI2_CONFIG_PAGE_EXPANDER_0.
 *                      Added SAS Port Page 0.
 *                      Modified structure layout for
 *                      MPI2_CONFIG_PAGE_DRIVER_MAPPING_0.
 *  06-27-08  02.00.08  Changed MPI2_CONFIG_PAGE_RD_PDISK_1 to use
 *                      MPI2_RAID_PHYS_DISK1_PATH_MAX to size the array.
 *  10-02-08  02.00.09  Changed MPI2_RAID_PGAD_CONFIGNUM_MASK from 0x0000FFFF
 *                      to 0x000000FF.
 *                      Added two new values for the Physical Disk Coercion Size
 *                      bits in the Flags field of Manufacturing Page 4.
 *                      Added product-specific Manufacturing pages 16 to 31.
 *                      Modified Flags bits for controlling write cache on SATA
 *                      drives in IO Unit Page 1.
 *                      Added new bit to AdditionalControlFlags of SAS IO Unit
 *                      Page 1 to control Invalid Topology Correction.
 *                      Added additional defines for RAID Volume Page 0
 *                      VolumeStatusFlags field.
 *                      Modified meaning of RAID Volume Page 0 VolumeSettings
 *                      define for auto-configure of hot-swap drives.
 *                      Added SupportedPhysDisks field to RAID Volume Page 1 and
 *                      added related defines.
```

```
 *                      Added PhysDiskAttributes field (and related defines) to
 *                      RAID Physical Disk Page 0.
 *                      Added MPI2_SAS_PHYINFO_PHY_VACANT define.
 *                      Added three new DiscoveryStatus bits for SAS IO Unit
 *                      Page 0 and SAS Expander Page 0.
 *                      Removed multiplexing information from SAS IO Unit pages.
 *                      Added BootDeviceWaitTime field to SAS IO Unit Page 4.
 *                      Removed Zone Address Resolved bit from PhyInfo and from
 *                      Expander Page 0 Flags field.
 *                      Added two new AccessStatus values to SAS Device Page 0
 *                      for indicating routing problems. Added 3 reserved words
 *                      to this page.
 *  01-19-09  02.00.10  Fixed defines for GPIOVal field of IO Unit Page 3.
 *                      Inserted missing reserved field into structure for IOC
 *                      Page 6.
 *                      Added more pending task bits to RAID Volume Page 0
 *                      VolumeStatusFlags defines.
 *                      Added MPI2_PHYSDISK0_STATUS_FLAG_NOT_CERTIFIED define.
 *                      Added a new DiscoveryStatus bit for SAS IO Unit Page 0
 *                      and SAS Expander Page 0 to flag a downstream initiator
 *                      when in simplified routing mode.
 *                      Removed SATA Init Failure defines for DiscoveryStatus
 *                      fields of SAS IO Unit Page 0 and SAS Expander Page 0.
 *                      Added MPI2_SAS_DEVICE0_ASTATUS_DEVICE_BLOCKED define.
 *                      Added PortGroups, DmaGroup, and ControlGroup fields to
 *                      SAS Device Page 0.
 *  05-06-09  02.00.11  Added structures and defines for IO Unit Page 5 and IO
 *                      Unit Page 6.
 *                      Added expander reduced functionality data to SAS
 *                      Expander Page 0.
 *                      Added SAS PHY Page 2 and SAS PHY Page 3.
 *  07-30-09  02.00.12  Added IO Unit Page 7.
 *                      Added new device ids.
 *                      Added SAS IO Unit Page 5.
 *                      Added partial and slumber power management capable flags
 *                      to SAS Device Page 0 Flags field.
 *                      Added PhyInfo defines for power condition.
 *                      Added Ethernet configuration pages.
 *  10-28-09  02.00.13  Added MPI2_IOUNITPAGE1_ENABLE_HOST_BASED_DISCOVERY.
 *                      Added SAS PHY Page 4 structure and defines.
 *  02-10-10  02.00.14  Modified the comments for the configuration page
 *                      structures that contain an array of data. The host
 *                      should use the "count" field in the page data (e.g. the
 *                      NumPhys field) to determine the number of valid elements
 *                      in the array.
 *                      Added/modified some MPI2_MFGPAGE_DEVID_SAS defines.
 *                      Added PowerManagementCapabilities to IO Unit Page 7.
 *                      Added PortWidthModGroup field to
 *                      MPI2_SAS_IO_UNIT5_PHY_PM_SETTINGS.
 *                      Added MPI2_CONFIG_PAGE_SASIOUNIT_6 and related defines.
 *                      Added MPI2_CONFIG_PAGE_SASIOUNIT_7 and related defines.
 *                      Added MPI2_CONFIG_PAGE_SASIOUNIT_8 and related defines.
 *  05-12-10  02.00.15  Added MPI2_RAIDVOL0_STATUS_FLAG_VOL_NOT_CONSISTENT
 *                      define.
 *                      Added MPI2_PHYSDISK0_INCOMPATIBLE_MEDIA_TYPE define.
 *                      Added MPI2_SAS_NEG_LINK_RATE_UNSUPPORTED_PHY define.
 *  08-11-10  02.00.16  Removed IO Unit Page 1 device path (multi-pathing)
 *                      defines.
 *  11-10-10  02.00.17  Added ReceptacleID field (replacing Reserved1) to
 *                      MPI2_MANPAGE7_CONNECTOR_INFO and reworked defines for
 *                      the Pinout field.
 *                      Added BoardTemperature and BoardTemperatureUnits fields
 *                      to MPI2_CONFIG_PAGE_IO_UNIT_7.
 *                      Added MPI2_CONFIG_EXTPAGETYPE_EXT_MANUFACTURING define
 *                      and MPI2_CONFIG_PAGE_EXT_MAN_PS structure.
 */
#endif /* ! codereview */
```



```
 *  --------------------------------------------------------------------------
 */

#ifndef MPI2_CNFG_H
#define MPI2_CNFG_H

/*****************************************************************************
*   Configuration Page Header and defines
*****************************************************************************/

/* Config Page Header */
typedef struct _MPI2_CONFIG_PAGE_HEADER
{
    U8                 PageVersion;               /* 0x00 */
    U8                 PageLength;                /* 0x01 */
    U8                 PageNumber;                /* 0x02 */
    U8                 PageType;                  /* 0x03 */
} MPI2_CONFIG_PAGE_HEADER, MPI2_POINTER PTR_MPI2_CONFIG_PAGE_HEADER,
  Mpi2ConfigPageHeader_t, MPI2_POINTER pMpi2ConfigPageHeader_t;

typedef union _MPI2_CONFIG_PAGE_HEADER_UNION
{
   MPI2_CONFIG_PAGE_HEADER  Struct;
   U8                       Bytes[4];
   U16                      Word16[2];
   U32                      Word32;
} MPI2_CONFIG_PAGE_HEADER_UNION, MPI2_POINTER PTR_MPI2_CONFIG_PAGE_HEADER_UNION,
  Mpi2ConfigPageHeaderUnion, MPI2_POINTER pMpi2ConfigPageHeaderUnion;

/* Extended Config Page Header */
typedef struct _MPI2_CONFIG_EXTENDED_PAGE_HEADER
{
    U8                 PageVersion;               /* 0x00 */
    U8                 Reserved1;                 /* 0x01 */
    U8                 PageNumber;                /* 0x02 */
    U8                 PageType;                  /* 0x03 */
    U16                ExtPageLength;             /* 0x04 */
    U8                 ExtPageType;               /* 0x06 */
    U8                 Reserved2;                 /* 0x07 */
} MPI2_CONFIG_EXTENDED_PAGE_HEADER,
  MPI2_POINTER PTR_MPI2_CONFIG_EXTENDED_PAGE_HEADER,
  Mpi2ConfigExtendedPageHeader_t, MPI2_POINTER pMpi2ConfigExtendedPageHeader_t;

typedef union _MPI2_CONFIG_EXT_PAGE_HEADER_UNION
{
   MPI2_CONFIG_PAGE_HEADER          Struct;
   MPI2_CONFIG_EXTENDED_PAGE_HEADER Ext;
   U8                               Bytes[8];
   U16                              Word16[4];
   U32                              Word32[2];
} MPI2_CONFIG_EXT_PAGE_HEADER_UNION, MPI2_POINTER PTR_MPI2_CONFIG_EXT_PAGE_HEADE
  Mpi2ConfigPageExtendedHeaderUnion, MPI2_POINTER pMpi2ConfigPageExtendedHeaderU


/* PageType field values */
#define MPI2_CONFIG_PAGEATTR_READ_ONLY              (0x00)
#define MPI2_CONFIG_PAGEATTR_CHANGEABLE             (0x10)
#define MPI2_CONFIG_PAGEATTR_PERSISTENT             (0x20)
#define MPI2_CONFIG_PAGEATTR_MASK                   (0xF0)

#define MPI2_CONFIG_PAGETYPE_IO_UNIT                (0x00)
#define MPI2_CONFIG_PAGETYPE_IOC                    (0x01)
#define MPI2_CONFIG_PAGETYPE_BIOS                   (0x02)
#define MPI2_CONFIG_PAGETYPE_RAID_VOLUME            (0x08)
#define MPI2_CONFIG_PAGETYPE_MANUFACTURING          (0x09)
#define MPI2_CONFIG_PAGETYPE_RAID_PHYSDISK          (0x0A)
```

```
#define MPI2_CONFIG_PAGETYPE_EXTENDED               (0x0F)
#define MPI2_CONFIG_PAGETYPE_MASK                   (0x0F)

#define MPI2_CONFIG_TYPENUM_MASK                    (0x0FFF)


/* ExtPageType field values */
#define MPI2_CONFIG_EXTPAGETYPE_SAS_IO_UNIT         (0x10)
#define MPI2_CONFIG_EXTPAGETYPE_SAS_EXPANDER        (0x11)
#define MPI2_CONFIG_EXTPAGETYPE_SAS_DEVICE          (0x12)
#define MPI2_CONFIG_EXTPAGETYPE_SAS_PHY             (0x13)
#define MPI2_CONFIG_EXTPAGETYPE_LOG                 (0x14)
#define MPI2_CONFIG_EXTPAGETYPE_ENCLOSURE           (0x15)
#define MPI2_CONFIG_EXTPAGETYPE_RAID_CONFIG         (0x16)
#define MPI2_CONFIG_EXTPAGETYPE_DRIVER_MAPPING      (0x17)
#define MPI2_CONFIG_EXTPAGETYPE_SAS_PORT            (0x18)
#define MPI2_CONFIG_EXTPAGETYPE_ETHERNET            (0x19)
#define MPI2_CONFIG_EXTPAGETYPE_EXT_MANUFACTURING   (0x1A)
#endif /* ! codereview */


/****************************************************************************
*   PageAddress defines
****************************************************************************/

/* RAID Volume PageAddress format */
#define MPI2_RAID_VOLUME_PGAD_FORM_MASK             (0xF0000000)
#define MPI2_RAID_VOLUME_PGAD_FORM_GET_NEXT_HANDLE  (0x00000000)
#define MPI2_RAID_VOLUME_PGAD_FORM_HANDLE           (0x10000000)

#define MPI2_RAID_VOLUME_PGAD_HANDLE_MASK           (0x0000FFFF)


/* RAID Physical Disk PageAddress format */
#define MPI2_PHYSDISK_PGAD_FORM_MASK                    (0xF0000000)
#define MPI2_PHYSDISK_PGAD_FORM_GET_NEXT_PHYSDISKNUM    (0x00000000)
#define MPI2_PHYSDISK_PGAD_FORM_PHYSDISKNUM             (0x10000000)
#define MPI2_PHYSDISK_PGAD_FORM_DEVHANDLE               (0x20000000)

#define MPI2_PHYSDISK_PGAD_PHYSDISKNUM_MASK             (0x000000FF)
#define MPI2_PHYSDISK_PGAD_DEVHANDLE_MASK               (0x0000FFFF)


/* SAS Expander PageAddress format */
#define MPI2_SAS_EXPAND_PGAD_FORM_MASK              (0xF0000000)
#define MPI2_SAS_EXPAND_PGAD_FORM_GET_NEXT_HNDL     (0x00000000)
#define MPI2_SAS_EXPAND_PGAD_FORM_HNDL_PHY_NUM      (0x10000000)
#define MPI2_SAS_EXPAND_PGAD_FORM_HNDL              (0x20000000)

#define MPI2_SAS_EXPAND_PGAD_HANDLE_MASK            (0x0000FFFF)
#define MPI2_SAS_EXPAND_PGAD_PHYNUM_MASK            (0x00FF0000)
#define MPI2_SAS_EXPAND_PGAD_PHYNUM_SHIFT           (16)


/* SAS Device PageAddress format */
#define MPI2_SAS_DEVICE_PGAD_FORM_MASK              (0xF0000000)
#define MPI2_SAS_DEVICE_PGAD_FORM_GET_NEXT_HANDLE   (0x00000000)
#define MPI2_SAS_DEVICE_PGAD_FORM_HANDLE            (0x20000000)

#define MPI2_SAS_DEVICE_PGAD_HANDLE_MASK            (0x0000FFFF)


/* SAS PHY PageAddress format */
#define MPI2_SAS_PHY_PGAD_FORM_MASK                 (0xF0000000)
#define MPI2_SAS_PHY_PGAD_FORM_PHY_NUMBER           (0x00000000)
#define MPI2_SAS_PHY_PGAD_FORM_PHY_TBL_INDEX        (0x10000000)
```



```
#define MPI2_SAS_PHY_PGAD_PHY_NUMBER_MASK           (0x000000FF)
#define MPI2_SAS_PHY_PGAD_PHY_TBL_INDEX_MASK        (0x0000FFFF)


/* SAS Port PageAddress format */
#define MPI2_SASPORT_PGAD_FORM_MASK                 (0xF0000000)
#define MPI2_SASPORT_PGAD_FORM_GET_NEXT_PORT        (0x00000000)
#define MPI2_SASPORT_PGAD_FORM_PORT_NUM             (0x10000000)

#define MPI2_SASPORT_PGAD_PORTNUMBER_MASK           (0x00000FFF)


/* SAS Enclosure PageAddress format */
#define MPI2_SAS_ENCLOS_PGAD_FORM_MASK              (0xF0000000)
#define MPI2_SAS_ENCLOS_PGAD_FORM_GET_NEXT_HANDLE   (0x00000000)
#define MPI2_SAS_ENCLOS_PGAD_FORM_HANDLE            (0x10000000)

#define MPI2_SAS_ENCLOS_PGAD_HANDLE_MASK            (0x0000FFFF)


/* RAID Configuration PageAddress format */
#define MPI2_RAID_PGAD_FORM_MASK                    (0xF0000000)
#define MPI2_RAID_PGAD_FORM_GET_NEXT_CONFIGNUM      (0x00000000)
#define MPI2_RAID_PGAD_FORM_CONFIGNUM               (0x10000000)
#define MPI2_RAID_PGAD_FORM_ACTIVE_CONFIG           (0x20000000)

#define MPI2_RAID_PGAD_CONFIGNUM_MASK               (0x000000FF)


/* Driver Persistent Mapping PageAddress format */
#define MPI2_DPM_PGAD_FORM_MASK                     (0xF0000000)
#define MPI2_DPM_PGAD_FORM_ENTRY_RANGE              (0x00000000)

#define MPI2_DPM_PGAD_ENTRY_COUNT_MASK              (0x0FFF0000)
#define MPI2_DPM_PGAD_ENTRY_COUNT_SHIFT             (16)
#define MPI2_DPM_PGAD_START_ENTRY_MASK              (0x0000FFFF)


/* Ethernet PageAddress format */
#define MPI2_ETHERNET_PGAD_FORM_MASK                (0xF0000000)
#define MPI2_ETHERNET_PGAD_FORM_IF_NUM              (0x00000000)

#define MPI2_ETHERNET_PGAD_IF_NUMBER_MASK           (0x000000FF)



/****************************************************************************
*   Configuration messages
****************************************************************************/

/* Configuration Request Message */
typedef struct _MPI2_CONFIG_REQUEST
{
    U8                      Action;                 /* 0x00 */
    U8                      SGLFlags;               /* 0x01 */
    U8                      ChainOffset;            /* 0x02 */
    U8                      Function;               /* 0x03 */
    U16                     ExtPageLength;          /* 0x04 */
    U8                      ExtPageType;            /* 0x06 */
    U8                      MsgFlags;               /* 0x07 */
    U8                      VP_ID;                  /* 0x08 */
    U8                      VF_ID;                  /* 0x09 */
    U16                     Reserved1;              /* 0x0A */
    U32                     Reserved2;              /* 0x0C */
    U32                     Reserved3;              /* 0x10 */
```

```
    MPI2_CONFIG_PAGE_HEADER Header;                 /* 0x14 */
    U32                     PageAddress;            /* 0x18 */
    MPI2_SGE_IO_UNION       PageBufferSGE;          /* 0x1C */
} MPI2_CONFIG_REQUEST, MPI2_POINTER PTR_MPI2_CONFIG_REQUEST,
  Mpi2ConfigRequest_t, MPI2_POINTER pMpi2ConfigRequest_t;

/* values for the Action field */
#define MPI2_CONFIG_ACTION_PAGE_HEADER              (0x00)
#define MPI2_CONFIG_ACTION_PAGE_READ_CURRENT        (0x01)
#define MPI2_CONFIG_ACTION_PAGE_WRITE_CURRENT       (0x02)
#define MPI2_CONFIG_ACTION_PAGE_DEFAULT             (0x03)
#define MPI2_CONFIG_ACTION_PAGE_WRITE_NVRAM         (0x04)
#define MPI2_CONFIG_ACTION_PAGE_READ_DEFAULT        (0x05)
#define MPI2_CONFIG_ACTION_PAGE_READ_NVRAM          (0x06)
#define MPI2_CONFIG_ACTION_PAGE_GET_CHANGEABLE      (0x07)

/* values for SGLFlags field are in the SGL section of mpi2.h */


/* Config Reply Message */
typedef struct _MPI2_CONFIG_REPLY
{
    U8                      Action;                 /* 0x00 */
    U8                      SGLFlags;               /* 0x01 */
    U8                      MsgLength;              /* 0x02 */
    U8                      Function;               /* 0x03 */
    U16                     ExtPageLength;          /* 0x04 */
    U8                      ExtPageType;            /* 0x06 */
    U8                      MsgFlags;               /* 0x07 */
    U8                      VP_ID;                  /* 0x08 */
    U8                      VF_ID;                  /* 0x09 */
    U16                     Reserved1;              /* 0x0A */
    U16                     Reserved2;              /* 0x0C */
    U16                     IOCStatus;              /* 0x0E */
    U32                     IOCLogInfo;             /* 0x10 */
    MPI2_CONFIG_PAGE_HEADER Header;                 /* 0x14 */
} MPI2_CONFIG_REPLY, MPI2_POINTER PTR_MPI2_CONFIG_REPLY,
  Mpi2ConfigReply_t, MPI2_POINTER pMpi2ConfigReply_t;



/*****************************************************************************
*
*               C o n f i g u r a t i o n    P a g e s
*
*****************************************************************************/

/****************************************************************************
*   Manufacturing Config pages
****************************************************************************/

#define MPI2_MFGPAGE_VENDORID_LSI                   (0x1000)

/* MPI v2.0 SAS products */
/* SAS */
#define MPI2_MFGPAGE_DEVID_SAS2004                  (0x0070)
#define MPI2_MFGPAGE_DEVID_SAS2008                  (0x0072)
#define MPI2_MFGPAGE_DEVID_SAS2108_1                (0x0074)
#define MPI2_MFGPAGE_DEVID_SAS2108_2                (0x0076)
#define MPI2_MFGPAGE_DEVID_SAS2108_3                (0x0077)
#define MPI2_MFGPAGE_DEVID_SAS2116_1                (0x0064)
#define MPI2_MFGPAGE_DEVID_SAS2116_2                (0x0065)

#define MPI2_MFGPAGE_DEVID_SSS6200                  (0x007E)

#endif /* ! codereview */
```



```
#define MPI2_MFGPAGE_DEVID_SAS2208_1                (0x0080)
#define MPI2_MFGPAGE_DEVID_SAS2208_2                (0x0081)
#define MPI2_MFGPAGE_DEVID_SAS2208_3                (0x0082)
#define MPI2_MFGPAGE_DEVID_SAS2208_4                (0x0083)
#define MPI2_MFGPAGE_DEVID_SAS2208_5                (0x0084)
#define MPI2_MFGPAGE_DEVID_SAS2208_6                (0x0085)
#define MPI2_MFGPAGE_DEVID_SAS2208_7                (0x0086)
#define MPI2_MFGPAGE_DEVID_SAS2208_8                (0x0087)
#define MPI2_MFGPAGE_DEVID_SAS2308_1                (0x0086)
#define MPI2_MFGPAGE_DEVID_SAS2308_2                (0x0087)
#define MPI2_MFGPAGE_DEVID_SAS2308_3                (0x006E)

/* MPI v2.5 SAS products */
#define MPI25_MFGPAGE_DEVID_SAS3004                 (0x0096)
#define MPI25_MFGPAGE_DEVID_SAS3008                 (0x0097)
#define MPI25_MFGPAGE_DEVID_SAS3108_1               (0x0090)
#define MPI25_MFGPAGE_DEVID_SAS3108_2               (0x0091)
#define MPI25_MFGPAGE_DEVID_SAS3108_3               (0x0092)
#define MPI25_MFGPAGE_DEVID_SAS3108_4               (0x0093)
#define MPI25_MFGPAGE_DEVID_SAS3108_5               (0x0094)
#define MPI25_MFGPAGE_DEVID_SAS3108_6               (0x0095)

/* Manufacturing Page 0 */

typedef struct _MPI2_CONFIG_PAGE_MAN_0
{
    MPI2_CONFIG_PAGE_HEADER Header;                 /* 0x00 */
    U8                      ChipName[16];           /* 0x04 */
    U8                      ChipRevision[8];        /* 0x14 */
    U8                      BoardName[16];          /* 0x1C */
    U8                      BoardAssembly[16];      /* 0x2C */
    U8                      BoardTracerNumber[16];  /* 0x3C */
} MPI2_CONFIG_PAGE_MAN_0,
_____unchanged_portion_omitted_
  MPI2_POINTER PTR_MPI2_CONFIG_PAGE_MAN_6,
  Mpi2ManufacturingPage6_t, MPI2_POINTER pMpi2ManufacturingPage6_t;

#define MPI2_MANUFACTURING6_PAGEVERSION             (0x00)


/* Manufacturing Page 7 */

typedef struct _MPI2_MANPAGE7_CONNECTOR_INFO
{
    U32                     Pinout;                 /* 0x00 */
    U8                      Connector[16];          /* 0x04 */
    U8                      Location;               /* 0x14 */
    U8                      ReceptacleID;           /* 0x15 */
    U8                      Reserved1;              /* 0x15 */
    U16                     Slot;                   /* 0x16 */
    U32                     Reserved2;              /* 0x18 */
} MPI2_MANPAGE7_CONNECTOR_INFO, MPI2_POINTER PTR_MPI2_MANPAGE7_CONNECTOR_INFO,
_____unchanged_portion_omitted_
  MPI2_POINTER PTR_MPI2_CONFIG_PAGE_MAN_7,
  Mpi2ManufacturingPage7_t, MPI2_POINTER pMpi2ManufacturingPage7_t;

#define MPI2_MANUFACTURING7_PAGEVERSION             (0x01)
#define MPI2_MANUFACTURING7_PAGEVERSION             (0x00)

/* defines for the Flags field */
#define MPI2_MANPAGE7_FLAG_USE_SLOT_INFO            (0x00000001)


/*
 * Generic structure to use for product-specific manufacturing pages
```

```
 742  * (currently Manufacturing Page 8 through Manufacturing Page 31).
 743  */

 745 typedef struct _MPI2_CONFIG_PAGE_MAN_PS
 746 {
 747     MPI2_CONFIG_PAGE_HEADER         Header;             /* 0x00 */
 748     U32                             ProductSpecificInfo;/* 0x04 */
 749 } MPI2_CONFIG_PAGE_MAN_PS,
_____unchanged_portion_omitted_
 804   Mpi2IOUnitPage1_t, MPI2_POINTER pMpi2IOUnitPage1_t;

 806 #define MPI2_IOUNITPAGE1_PAGEVERSION                    (0x04)

 808 /* IO Unit Page 1 Flags defines */
 809 #define MPI25_IOUNITPAGE1_NEW_DEVICE_FAST_PATH_DISABLE  (0x00002000)
 810 #define MPI25_IOUNITPAGE1_DISABLE_FAST_PATH             (0x00001000)
 811 #define MPI2_IOUNITPAGE1_ENABLE_HOST_BASED_DISCOVERY    (0x00000800)
 812 #endif /* ! codereview */
 813 #define MPI2_IOUNITPAGE1_MASK_SATA_WRITE_CACHE          (0x00000600)
 814 #define MPI2_IOUNITPAGE1_SATA_WRITE_CACHE_SHIFT         (9)
 815 #endif /* ! codereview */
 816 #define MPI2_IOUNITPAGE1_ENABLE_SATA_WRITE_CACHE        (0x00000000)
 817 #define MPI2_IOUNITPAGE1_DISABLE_SATA_WRITE_CACHE       (0x00000200)
 818 #define MPI2_IOUNITPAGE1_UNCHANGED_SATA_WRITE_CACHE     (0x00000400)
 819 #define MPI2_IOUNITPAGE1_NATIVE_COMMAND_Q_DISABLE       (0x00000100)
 820 #define MPI2_IOUNITPAGE1_DISABLE_IR                     (0x00000040)
 821 #define MPI2_IOUNITPAGE1_DISABLE_TASK_SET_FULL_HANDLING (0x00000020)
 822 #define MPI2_IOUNITPAGE1_IR_USE_STATIC_VOLUME_ID        (0x00000004)
 823 #define MPI2_IOUNITPAGE1_MULTI_PATHING                  (0x00000002)
 824 #define MPI2_IOUNITPAGE1_SINGLE_PATHING                 (0x00000000)


 827 /* IO Unit Page 3 */

 829 /*
 830  * Host code (drivers, BIOS, utilities, etc.) should leave this define set to
 831  * one and check Header.PageLength at runtime.
 832  */
 833 #ifndef MPI2_IO_UNIT_PAGE_3_GPIO_VAL_MAX
 834 #define MPI2_IO_UNIT_PAGE_3_GPIO_VAL_MAX    (1)
 835 #endif

 837 typedef struct _MPI2_CONFIG_PAGE_IO_UNIT_3
 838 {
 839     MPI2_CONFIG_PAGE_HEADER Header;                             /* 0x00 */
 840     U8                      GPIOCount;                          /* 0x04 */
 841     U8                      Reserved1;                          /* 0x05 */
 842     U16                     Reserved2;                          /* 0x06 */
 843     U16                     GPIOVal[MPI2_IO_UNIT_PAGE_3_GPIO_VAL_MAX];/* 0x08 */
 844 } MPI2_CONFIG_PAGE_IO_UNIT_3, MPI2_POINTER PTR_MPI2_CONFIG_PAGE_IO_UNIT_3,
 845   Mpi2IOUnitPage3_t, MPI2_POINTER pMpi2IOUnitPage3_t;

 847 #define MPI2_IOUNITPAGE3_PAGEVERSION                    (0x01)

 849 /* defines for IO Unit Page 3 GPIOVal field */
 850 #define MPI2_IOUNITPAGE3_GPIO_FUNCTION_MASK             (0xFFFC)
 851 #define MPI2_IOUNITPAGE3_GPIO_FUNCTION_SHIFT            (2)
 852 #define MPI2_IOUNITPAGE3_GPIO_SETTING_OFF               (0x0000)
 853 #define MPI2_IOUNITPAGE3_GPIO_SETTING_ON                (0x0001)


 856 /* IO Unit Page 5 */

 858 /*
 859  * Upper layer code (drivers, utilities, etc.) should leave this define set to
 860  * one and check Header.PageLength or NumDmaEngines at runtime.
```



```
 861  */
 862 #ifndef MPI2_IOUNITPAGE5_DMAENGINE_ENTRIES
 863 #define MPI2_IOUNITPAGE5_DMAENGINE_ENTRIES      (1)
 864 #endif

 866 typedef struct _MPI2_CONFIG_PAGE_IO_UNIT_5
 867 {
 868     MPI2_CONFIG_PAGE_HEADER Header;                                 /* 0x00
 869     U64                     RaidAcceleratorBufferBaseAddress;       /* 0x04
 870     U64                     RaidAcceleratorBufferSize;              /* 0x0C
 871     U64                     RaidAcceleratorControlBaseAddress;      /* 0x14
 872     U8                      RAControlSize;                          /* 0x1C
 873     U8                      NumDmaEngines;                          /* 0x1D
 874     U8                      RAMinControlSize;                       /* 0x1E
 875     U8                      RAMaxControlSize;                       /* 0x1F
 876     U32                     Reserved1;                              /* 0x20
 877     U32                     Reserved2;                              /* 0x24
 878     U32                     Reserved3;                              /* 0x28
 879     U32                     DmaEngineCapabilities[MPI2_IOUNITPAGE5_DMAENGINE_ENT
 880 } MPI2_CONFIG_PAGE_IO_UNIT_5, MPI2_POINTER PTR_MPI2_CONFIG_PAGE_IO_UNIT_5,
 881   Mpi2IOUnitPage5_t, MPI2_POINTER pMpi2IOUnitPage5_t;

 883 #define MPI2_IOUNITPAGE5_PAGEVERSION                    (0x00)

 885 /* defines for IO Unit Page 5 DmaEngineCapabilities field */
 886 #define MPI2_IOUNITPAGE5_DMA_CAP_MASK_MAX_REQUESTS      (0xFF00)
 887 #define MPI2_IOUNITPAGE5_DMA_CAP_SHIFT_MAX_REQUESTS     (16)

 889 #define MPI2_IOUNITPAGE5_DMA_CAP_EEDP                   (0x0008)
 890 #define MPI2_IOUNITPAGE5_DMA_CAP_PARITY_GENERATION      (0x0004)
 891 #define MPI2_IOUNITPAGE5_DMA_CAP_HASHING                (0x0002)
 892 #define MPI2_IOUNITPAGE5_DMA_CAP_ENCRYPTION             (0x0001)


 895 /* IO Unit Page 6 */

 897 typedef struct _MPI2_CONFIG_PAGE_IO_UNIT_6
 898 {
 899     MPI2_CONFIG_PAGE_HEADER Header;                                 /* 0x00 */
 900     U16                     Flags;                                  /* 0x04 */
 901     U8                      RAHostControlSize;                      /* 0x06 */
 902     U8                      Reserved0;                              /* 0x07 */
 903     U64                     RaidAcceleratorHostControlBaseAddress;  /* 0x08 */
 904     U32                     Reserved1;                              /* 0x10 */
 905     U32                     Reserved2;                              /* 0x14 */
 906     U32                     Reserved3;                              /* 0x18 */
 907 } MPI2_CONFIG_PAGE_IO_UNIT_6, MPI2_POINTER PTR_MPI2_CONFIG_PAGE_IO_UNIT_6,
 908   Mpi2IOUnitPage6_t, MPI2_POINTER pMpi2IOUnitPage6_t;

 910 #define MPI2_IOUNITPAGE6_PAGEVERSION                    (0x00)

 912 /* defines for IO Unit Page 6 Flags field */
 913 #define MPI2_IOUNITPAGE6_FLAGS_ENABLE_RAID_ACCELERATOR  (0x0001)


 916 /* IO Unit Page 7 */

 918 typedef struct _MPI2_CONFIG_PAGE_IO_UNIT_7
 919 {
 920     MPI2_CONFIG_PAGE_HEADER Header;                                 /* 0x00 */
 921     U16                     Reserved1;                              /* 0x04 */
 922     U8                      PCIeWidth;                              /* 0x06 */
 923     U8                      PCIeSpeed;                              /* 0x07 */
 924     U32                     ProcessorState;                         /* 0x08 */
 925     U32                     PowerManagementCapabilities;            /* 0x0C */
 503     U32                     Reserved2;                              /* 0x0C */
```

```
    U16                     IOCTemperature;                     /* 0x10 */
    U8                      IOCTemperatureUnits;                /* 0x12 */
    U8                      IOCSpeed;                           /* 0x13 */
    U16                     BoardTemperature;                   /* 0x14 */
    U8                      BoardTemperatureUnits;              /* 0x16 */
    U8                      Reserved3;                          /* 0x17 */
    U32                     Reserved3;                          /* 0x14 */
} MPI2_CONFIG_PAGE_IO_UNIT_7, MPI2_POINTER PTR_MPI2_CONFIG_PAGE_IO_UNIT_7,
  Mpi2IOUnitPage7_t, MPI2_POINTER pMpi2IOUnitPage7_t;

#define MPI2_IOUNITPAGE7_PAGEVERSION                (0x02)
#define MPI2_IOUNITPAGE7_PAGEVERSION                (0x00)

/* defines for IO Unit Page 7 PCIeWidth field */
#define MPI2_IOUNITPAGE7_PCIE_WIDTH_X1              (0x01)
#define MPI2_IOUNITPAGE7_PCIE_WIDTH_X2              (0x02)
#define MPI2_IOUNITPAGE7_PCIE_WIDTH_X4              (0x04)
#define MPI2_IOUNITPAGE7_PCIE_WIDTH_X8              (0x08)

/* defines for IO Unit Page 7 PCIeSpeed field */
#define MPI2_IOUNITPAGE7_PCIE_SPEED_2_5_GBPS        (0x00)
#define MPI2_IOUNITPAGE7_PCIE_SPEED_5_0_GBPS        (0x01)
#define MPI2_IOUNITPAGE7_PCIE_SPEED_8_0_GBPS        (0x02)

/* defines for IO Unit Page 7 ProcessorState field */
#define MPI2_IOUNITPAGE7_PSTATE_MASK_SECOND         (0x0000000F)
#define MPI2_IOUNITPAGE7_PSTATE_SHIFT_SECOND        (0)

#define MPI2_IOUNITPAGE7_PSTATE_NOT_PRESENT         (0x00)
#define MPI2_IOUNITPAGE7_PSTATE_DISABLED            (0x01)
#define MPI2_IOUNITPAGE7_PSTATE_ENABLED             (0x02)

/* defines for IO Unit Page 7 PowerManagementCapabilities field */
#define MPI2_IOUNITPAGE7_PMCAP_12_5_PCT_IOCSPEED    (0x00000400)
#define MPI2_IOUNITPAGE7_PMCAP_25_0_PCT_IOCSPEED    (0x00000200)
#define MPI2_IOUNITPAGE7_PMCAP_50_0_PCT_IOCSPEED    (0x00000100)
#define MPI2_IOUNITPAGE7_PMCAP_PCIE_WIDTH_CHANGE    (0x00000008)
#define MPI2_IOUNITPAGE7_PMCAP_PCIE_SPEED_CHANGE    (0x00000004)

#endif /* ! codereview */
/* defines for IO Unit Page 7 IOCTemperatureUnits field */
#define MPI2_IOUNITPAGE7_IOC_TEMP_NOT_PRESENT       (0x00)
#define MPI2_IOUNITPAGE7_IOC_TEMP_FAHRENHEIT        (0x01)
#define MPI2_IOUNITPAGE7_IOC_TEMP_CELSIUS           (0x02)

/* defines for IO Unit Page 7 IOCSpeed field */
#define MPI2_IOUNITPAGE7_IOC_SPEED_FULL             (0x01)
#define MPI2_IOUNITPAGE7_IOC_SPEED_HALF             (0x02)
#define MPI2_IOUNITPAGE7_IOC_SPEED_QUARTER          (0x04)
#define MPI2_IOUNITPAGE7_IOC_SPEED_EIGHTH           (0x08)

/* defines for IO Unit Page 7 BoardTemperatureUnits field */
#define MPI2_IOUNITPAGE7_BOARD_TEMP_NOT_PRESENT     (0x00)
#define MPI2_IOUNITPAGE7_BOARD_TEMP_FAHRENHEIT      (0x01)
#define MPI2_IOUNITPAGE7_BOARD_TEMP_CELSIUS         (0x02)
#endif /* ! codereview */


/*****************************************************************************
*   IOC Config Pages
*****************************************************************************/

/* IOC Page 0 */

typedef struct _MPI2_CONFIG_PAGE_IOC_0
{
```



```
    MPI2_CONFIG_PAGE_HEADER Header;                 /* 0x00 */
    U32                     Reserved1;              /* 0x04 */
    U32                     Reserved2;              /* 0x08 */
    U16                     VendorID;               /* 0x0C */
    U16                     DeviceID;               /* 0x0E */
    U8                      RevisionID;             /* 0x10 */
    U8                      Reserved3;              /* 0x11 */
    U16                     Reserved4;              /* 0x12 */
    U32                     ClassCode;              /* 0x14 */
    U16                     SubsystemVendorID;      /* 0x18 */
    U16                     SubsystemID;            /* 0x1A */
} MPI2_CONFIG_PAGE_IOC_0, MPI2_POINTER PTR_MPI2_CONFIG_PAGE_IOC_0,
  Mpi2IOCPage0_t, MPI2_POINTER pMpi2IOCPage0_t;

#define MPI2_IOCPAGE0_PAGEVERSION                   (0x02)


/* IOC Page 1 */

typedef struct _MPI2_CONFIG_PAGE_IOC_1
{
    MPI2_CONFIG_PAGE_HEADER Header;                 /* 0x00 */
    U32                     Flags;                  /* 0x04 */
    U32                     CoalescingTimeout;      /* 0x08 */
    U8                      CoalescingDepth;        /* 0x0C */
    U8                      PCISlotNum;             /* 0x0D */
    U8                      PCIBusNum;              /* 0x0E */
    U8                      PCIDomainSegment;       /* 0x0F */
    U32                     Reserved1;              /* 0x10 */
    U32                     Reserved2;              /* 0x14 */
} MPI2_CONFIG_PAGE_IOC_1, MPI2_POINTER PTR_MPI2_CONFIG_PAGE_IOC_1,
  Mpi2IOCPage1_t, MPI2_POINTER pMpi2IOCPage1_t;

#define MPI2_IOCPAGE1_PAGEVERSION                   (0x05)

/* defines for IOC Page 1 Flags field */
#define MPI2_IOCPAGE1_REPLY_COALESCING              (0x00000001)

#define MPI2_IOCPAGE1_PCISLOTNUM_UNKNOWN            (0xFF)
#define MPI2_IOCPAGE1_PCIBUSNUM_UNKNOWN             (0xFF)
#define MPI2_IOCPAGE1_PCIDOMAIN_UNKNOWN             (0xFF)

/* IOC Page 6 */

typedef struct _MPI2_CONFIG_PAGE_IOC_6
{
    MPI2_CONFIG_PAGE_HEADER Header;                     /* 0x00 */
    U32                     CapabilitiesFlags;          /* 0x04 */
    U8                      MaxDrivesRAID0;             /* 0x08 */
    U8                      MaxDrivesRAID1;             /* 0x09 */
    U8                      MaxDrivesRAID1E;            /* 0x0A */
    U8                      MaxDrivesRAID10;            /* 0x0B */
    U8                      MinDrivesRAID0;             /* 0x0C */
    U8                      MinDrivesRAID1;             /* 0x0D */
    U8                      MinDrivesRAID1E;            /* 0x0E */
    U8                      MinDrivesRAID10;            /* 0x0F */
    U32                     Reserved1;                  /* 0x10 */
    U8                      MaxGlobalHotSpares;         /* 0x14 */
    U8                      MaxPhysDisks;               /* 0x15 */
    U8                      MaxVolumes;                 /* 0x16 */
    U8                      MaxConfigs;                 /* 0x17 */
    U8                      MaxOCEDisks;                /* 0x18 */
    U8                      Reserved2;                  /* 0x19 */
    U16                     Reserved3;                  /* 0x1A */
    U32                     SupportedStripeSizeMapRAID0;    /* 0x1C */
    U32                     SupportedStripeSizeMapRAID1E;   /* 0x20 */
```

```
    U32                     SupportedStripeSizeMapRAID10;   /* 0x24 */
    U32                     Reserved4;                      /* 0x28 */
    U32                     Reserved5;                      /* 0x2C */
    U16                     DefaultMetadataSize;            /* 0x30 */
    U16                     Reserved6;                      /* 0x32 */
    U16                     MaxBadBlockTableEntries;        /* 0x34 */
    U16                     Reserved7;                      /* 0x36 */
    U32                     IRNvsramVersion;                /* 0x38 */
} MPI2_CONFIG_PAGE_IOC_6, MPI2_POINTER PTR_MPI2_CONFIG_PAGE_IOC_6,
  Mpi2IOCPage6_t, MPI2_POINTER pMpi2IOCPage6_t;

#define MPI2_IOCPAGE6_PAGEVERSION                       (0x04)

/* defines for IOC Page 6 CapabilitiesFlags */
#define MPI2_IOCPAGE6_CAP_FLAGS_RAID10_SUPPORT          (0x00000010)
#define MPI2_IOCPAGE6_CAP_FLAGS_RAID1_SUPPORT           (0x00000008)
#define MPI2_IOCPAGE6_CAP_FLAGS_RAID1E_SUPPORT          (0x00000004)
#define MPI2_IOCPAGE6_CAP_FLAGS_RAID0_SUPPORT           (0x00000002)
#define MPI2_IOCPAGE6_CAP_FLAGS_GLOBAL_HOT_SPARE        (0x00000001)


/* IOC Page 7 */

#define MPI2_IOCPAGE7_EVENTMASK_WORDS       (4)

typedef struct _MPI2_CONFIG_PAGE_IOC_7
{
    MPI2_CONFIG_PAGE_HEADER Header;                     /* 0x00 */
    U32                     Reserved1;                  /* 0x04 */
    U32                     EventMasks[MPI2_IOCPAGE7_EVENTMASK_WORDS];/* 0x08 */
    U16                     SASBroadcastPrimitiveMasks; /* 0x18 */
    U16                     Reserved2;                  /* 0x1A */
    U32                     Reserved3;                  /* 0x1C */
} MPI2_CONFIG_PAGE_IOC_7, MPI2_POINTER PTR_MPI2_CONFIG_PAGE_IOC_7,
  Mpi2IOCPage7_t, MPI2_POINTER pMpi2IOCPage7_t;

#define MPI2_IOCPAGE7_PAGEVERSION                       (0x01)


/* IOC Page 8 */

typedef struct _MPI2_CONFIG_PAGE_IOC_8
{
    MPI2_CONFIG_PAGE_HEADER Header;                     /* 0x00 */
    U8                      NumDevsPerEnclosure;        /* 0x04 */
    U8                      Reserved1;                  /* 0x05 */
    U16                     Reserved2;                  /* 0x06 */
    U16                     MaxPersistentEntries;       /* 0x08 */
    U16                     MaxNumPhysicalMappedIDs;    /* 0x0A */
    U16                     Flags;                      /* 0x0C */
    U16                     Reserved3;                  /* 0x0E */
    U16                     IRVolumeMappingFlags;       /* 0x10 */
    U16                     Reserved4;                  /* 0x12 */
    U32                     Reserved5;                  /* 0x14 */
} MPI2_CONFIG_PAGE_IOC_8, MPI2_POINTER PTR_MPI2_CONFIG_PAGE_IOC_8,
  Mpi2IOCPage8_t, MPI2_POINTER pMpi2IOCPage8_t;

#define MPI2_IOCPAGE8_PAGEVERSION                       (0x00)

/* defines for IOC Page 8 Flags field */
#define MPI2_IOCPAGE8_FLAGS_DA_START_SLOT_1             (0x00000020)
#define MPI2_IOCPAGE8_FLAGS_RESERVED_TARGETID_0         (0x00000010)

#define MPI2_IOCPAGE8_FLAGS_MASK_MAPPING_MODE           (0x0000000E)
#define MPI2_IOCPAGE8_FLAGS_DEVICE_PERSISTENCE_MAPPING  (0x00000000)
#define MPI2_IOCPAGE8_FLAGS_ENCLOSURE_SLOT_MAPPING      (0x00000002)
```



```
#define MPI2_IOCPAGE8_FLAGS_DISABLE_PERSISTENT_MAPPING  (0x00000001)
#define MPI2_IOCPAGE8_FLAGS_ENABLE_PERSISTENT_MAPPING   (0x00000000)

/* defines for IOC Page 8 IRVolumeMappingFlags */
#define MPI2_IOCPAGE8_IRFLAGS_MASK_VOLUME_MAPPING_MODE  (0x00000003)
#define MPI2_IOCPAGE8_IRFLAGS_LOW_VOLUME_MAPPING        (0x00000000)
#define MPI2_IOCPAGE8_IRFLAGS_HIGH_VOLUME_MAPPING       (0x00000001)


/*****************************************************************************
*   BIOS Config Pages
*****************************************************************************/

/* BIOS Page 1 */

typedef struct _MPI2_CONFIG_PAGE_BIOS_1
{
    MPI2_CONFIG_PAGE_HEADER Header;                     /* 0x00 */
    U32                     BiosOptions;                /* 0x04 */
    U32                     IOCSettings;                /* 0x08 */
    U32                     Reserved1;                  /* 0x0C */
    U32                     DeviceSettings;             /* 0x10 */
    U16                     NumberOfDevices;            /* 0x14 */
    U16                     Reserved2;                  /* 0x16 */
    U16                     IOTimeoutBlockDevicesNonRM; /* 0x18 */
    U16                     IOTimeoutSequential;        /* 0x1A */
    U16                     IOTimeoutOther;             /* 0x1C */
    U16                     IOTimeoutBlockDevicesRM;    /* 0x1E */
} MPI2_CONFIG_PAGE_BIOS_1, MPI2_POINTER PTR_MPI2_CONFIG_PAGE_BIOS_1,
  Mpi2BiosPage1_t, MPI2_POINTER pMpi2BiosPage1_t;

#define MPI2_BIOSPAGE1_PAGEVERSION                      (0x04)

/* values for BIOS Page 1 BiosOptions field */
#define MPI2_BIOSPAGE1_OPTIONS_DISABLE_BIOS             (0x00000001)

/* values for BIOS Page 1 IOCSettings field */
#define MPI2_BIOSPAGE1_IOCSET_MASK_BOOT_PREFERENCE      (0x00030000)
#define MPI2_BIOSPAGE1_IOCSET_ENCLOSURE_SLOT_BOOT       (0x00000000)
#define MPI2_BIOSPAGE1_IOCSET_SAS_ADDRESS_BOOT          (0x00010000)

#define MPI2_BIOSPAGE1_IOCSET_MASK_RM_SETTING           (0x000000C0)
#define MPI2_BIOSPAGE1_IOCSET_NONE_RM_SETTING           (0x00000000)
#define MPI2_BIOSPAGE1_IOCSET_BOOT_RM_SETTING           (0x00000040)
#define MPI2_BIOSPAGE1_IOCSET_MEDIA_RM_SETTING          (0x00000080)

#define MPI2_BIOSPAGE1_IOCSET_MASK_ADAPTER_SUPPORT      (0x00000030)
#define MPI2_BIOSPAGE1_IOCSET_NO_SUPPORT                (0x00000000)
#define MPI2_BIOSPAGE1_IOCSET_BIOS_SUPPORT              (0x00000010)
#define MPI2_BIOSPAGE1_IOCSET_OS_SUPPORT                (0x00000020)
#define MPI2_BIOSPAGE1_IOCSET_ALL_SUPPORT               (0x00000030)

#define MPI2_BIOSPAGE1_IOCSET_ALTERNATE_CHS             (0x00000008)

/* values for BIOS Page 1 DeviceSettings field */
#define MPI2_BIOSPAGE1_DEVSET_DISABLE_SMART_POLLING     (0x00000010)
#define MPI2_BIOSPAGE1_DEVSET_DISABLE_SEQ_LUN           (0x00000008)
#define MPI2_BIOSPAGE1_DEVSET_DISABLE_RM_LUN            (0x00000004)
#define MPI2_BIOSPAGE1_DEVSET_DISABLE_NON_RM_LUN        (0x00000002)
#define MPI2_BIOSPAGE1_DEVSET_DISABLE_OTHER_LUN         (0x00000001)


/* BIOS Page 2 */

typedef struct _MPI2_BOOT_DEVICE_ADAPTER_ORDER
```

```
{
    U32         Reserved1;                          /* 0x00 */
    U32         Reserved2;                          /* 0x04 */
    U32         Reserved3;                          /* 0x08 */
    U32         Reserved4;                          /* 0x0C */
    U32         Reserved5;                          /* 0x10 */
    U32         Reserved6;                          /* 0x14 */
} MPI2_BOOT_DEVICE_ADAPTER_ORDER,
  MPI2_POINTER PTR_MPI2_BOOT_DEVICE_ADAPTER_ORDER,
  Mpi2BootDeviceAdapterOrder_t, MPI2_POINTER pMpi2BootDeviceAdapterOrder_t;

typedef struct _MPI2_BOOT_DEVICE_SAS_WWID
{
    U64         SASAddress;                         /* 0x00 */
    U8          LUN[8];                             /* 0x08 */
    U32         Reserved1;                          /* 0x10 */
    U32         Reserved2;                          /* 0x14 */
} MPI2_BOOT_DEVICE_SAS_WWID, MPI2_POINTER PTR_MPI2_BOOT_DEVICE_SAS_WWID,
  Mpi2BootDeviceSasWwid_t, MPI2_POINTER pMpi2BootDeviceSasWwid_t;

typedef struct _MPI2_BOOT_DEVICE_ENCLOSURE_SLOT
{
    U64         EnclosureLogicalID;                 /* 0x00 */
    U32         Reserved1;                          /* 0x08 */
    U32         Reserved2;                          /* 0x0C */
    U16         SlotNumber;                         /* 0x10 */
    U16         Reserved3;                          /* 0x12 */
    U32         Reserved4;                          /* 0x14 */
} MPI2_BOOT_DEVICE_ENCLOSURE_SLOT,
  MPI2_POINTER PTR_MPI2_BOOT_DEVICE_ENCLOSURE_SLOT,
  Mpi2BootDeviceEnclosureSlot_t, MPI2_POINTER pMpi2BootDeviceEnclosureSlot_t;

typedef struct _MPI2_BOOT_DEVICE_DEVICE_NAME
{
    U64         DeviceName;                         /* 0x00 */
    U8          LUN[8];                             /* 0x08 */
    U32         Reserved1;                          /* 0x10 */
    U32         Reserved2;                          /* 0x14 */
} MPI2_BOOT_DEVICE_DEVICE_NAME, MPI2_POINTER PTR_MPI2_BOOT_DEVICE_DEVICE_NAME,
  Mpi2BootDeviceDeviceName_t, MPI2_POINTER pMpi2BootDeviceDeviceName_t;

typedef union _MPI2_MPI2_BIOSPAGE2_BOOT_DEVICE
{
    MPI2_BOOT_DEVICE_ADAPTER_ORDER  AdapterOrder;
    MPI2_BOOT_DEVICE_SAS_WWID       SasWwid;
    MPI2_BOOT_DEVICE_ENCLOSURE_SLOT EnclosureSlot;
    MPI2_BOOT_DEVICE_DEVICE_NAME    DeviceName;
} MPI2_BIOSPAGE2_BOOT_DEVICE, MPI2_POINTER PTR_MPI2_BIOSPAGE2_BOOT_DEVICE,
  Mpi2BiosPage2BootDevice_t, MPI2_POINTER pMpi2BiosPage2BootDevice_t;

typedef struct _MPI2_CONFIG_PAGE_BIOS_2
{
    MPI2_CONFIG_PAGE_HEADER     Header;             /* 0x00 */
    U32                         Reserved1;          /* 0x04 */
    U32                         Reserved2;          /* 0x08 */
    U32                         Reserved3;          /* 0x0C */
    U32                         Reserved4;          /* 0x10 */
    U32                         Reserved5;          /* 0x14 */
    U32                         Reserved6;          /* 0x18 */
    U8                          ReqBootDeviceForm;  /* 0x1C */
    U8                          Reserved7;          /* 0x1D */
    U16                         Reserved8;          /* 0x1E */
    MPI2_BIOSPAGE2_BOOT_DEVICE  RequestedBootDevice;/* 0x20 */
    U8                          ReqAltBootDeviceForm;/* 0x38 */
    U8                          Reserved9;          /* 0x39 */
    U16                         Reserved10;         /* 0x3A */
```



```
    MPI2_BIOSPAGE2_BOOT_DEVICE  RequestedAltBootDevice; /* 0x3C */
    U8                          CurrentBootDeviceForm;  /* 0x58 */
    U8                          Reserved11;             /* 0x59 */
    U16                         Reserved12;             /* 0x5A */
    MPI2_BIOSPAGE2_BOOT_DEVICE  CurrentBootDevice;      /* 0x58 */
} MPI2_CONFIG_PAGE_BIOS_2, MPI2_POINTER PTR_MPI2_CONFIG_PAGE_BIOS_2,
  Mpi2BiosPage2_t, MPI2_POINTER pMpi2BiosPage2_t;

#define MPI2_BIOSPAGE2_PAGEVERSION                      (0x04)

/* values for BIOS Page 2 BootDeviceForm fields */
#define MPI2_BIOSPAGE2_FORM_MASK                        (0x0F)
#define MPI2_BIOSPAGE2_FORM_NO_DEVICE_SPECIFIED         (0x00)
#define MPI2_BIOSPAGE2_FORM_SAS_WWID                    (0x05)
#define MPI2_BIOSPAGE2_FORM_ENCLOSURE_SLOT              (0x06)
#define MPI2_BIOSPAGE2_FORM_DEVICE_NAME                 (0x07)


/* BIOS Page 3 */

typedef struct _MPI2_ADAPTER_INFO
{
    U8      PciBusNumber;                           /* 0x00 */
    U8      PciDeviceAndFunctionNumber;             /* 0x01 */
    U16     AdapterFlags;                           /* 0x02 */
} MPI2_ADAPTER_INFO, MPI2_POINTER PTR_MPI2_ADAPTER_INFO,
  Mpi2AdapterInfo_t, MPI2_POINTER pMpi2AdapterInfo_t;

#define MPI2_ADAPTER_INFO_FLAGS_EMBEDDED                (0x0001)
#define MPI2_ADAPTER_INFO_FLAGS_INIT_STATUS             (0x0002)

typedef struct _MPI2_CONFIG_PAGE_BIOS_3
{
    MPI2_CONFIG_PAGE_HEADER Header;                     /* 0x00 */
    U32                     GlobalFlags;                /* 0x04 */
    U32                     BiosVersion;                /* 0x08 */
    MPI2_ADAPTER_INFO       AdapterOrder[4];            /* 0x0C */
    U32                     Reserved1;                  /* 0x1C */
} MPI2_CONFIG_PAGE_BIOS_3, MPI2_POINTER PTR_MPI2_CONFIG_PAGE_BIOS_3,
  Mpi2BiosPage3_t, MPI2_POINTER pMpi2BiosPage3_t;

#define MPI2_BIOSPAGE3_PAGEVERSION                      (0x00)

/* values for BIOS Page 3 GlobalFlags */
#define MPI2_BIOSPAGE3_FLAGS_PAUSE_ON_ERROR             (0x00000002)
#define MPI2_BIOSPAGE3_FLAGS_VERBOSE_ENABLE             (0x00000004)
#define MPI2_BIOSPAGE3_FLAGS_HOOK_INT_40_DISABLE        (0x00000010)

#define MPI2_BIOSPAGE3_FLAGS_DEV_LIST_DISPLAY_MASK      (0x000000E0)
#define MPI2_BIOSPAGE3_FLAGS_INSTALLED_DEV_DISPLAY      (0x00000000)
#define MPI2_BIOSPAGE3_FLAGS_ADAPTER_DISPLAY            (0x00000020)
#define MPI2_BIOSPAGE3_FLAGS_ADAPTER_DEV_DISPLAY        (0x00000040)


/* BIOS Page 4 */

/*
 * Host code (drivers, BIOS, utilities, etc.) should leave this define set to
 * one and check Header.PageLength or NumPhys at runtime.
 */
#ifndef MPI2_BIOS_PAGE_4_PHY_ENTRIES
#define MPI2_BIOS_PAGE_4_PHY_ENTRIES        (1)
#endif

typedef struct _MPI2_BIOS4_ENTRY
{
```

```
    U64                     ReassignmentWWID;       /* 0x00 */
    U64                     ReassignmentDeviceName; /* 0x08 */
} MPI2_BIOS4_ENTRY, MPI2_POINTER PTR_MPI2_BIOS4_ENTRY,
  Mpi2MBios4Entry_t, MPI2_POINTER pMpi2Bios4Entry_t;

typedef struct _MPI2_CONFIG_PAGE_BIOS_4
{
    MPI2_CONFIG_PAGE_HEADER Header;                         /* 0x00 */
    U8                      NumPhys;                        /* 0x04 */
    U8                      Reserved1;                      /* 0x05 */
    U16                     Reserved2;                      /* 0x06 */
    MPI2_BIOS4_ENTRY        Phy[MPI2_BIOS_PAGE_4_PHY_ENTRIES];  /* 0x08 */
} MPI2_CONFIG_PAGE_BIOS_4, MPI2_POINTER PTR_MPI2_CONFIG_PAGE_BIOS_4,
  Mpi2BiosPage4_t, MPI2_POINTER pMpi2BiosPage4_t;

#define MPI2_BIOSPAGE4_PAGEVERSION                  (0x01)


/*****************************************************************************
*   RAID Volume Config Pages
*****************************************************************************/

/* RAID Volume Page 0 */

typedef struct _MPI2_RAIDVOL0_PHYS_DISK
{
    U8                      RAIDSetNum;             /* 0x00 */
    U8                      PhysDiskMap;            /* 0x01 */
    U8                      PhysDiskNum;            /* 0x02 */
    U8                      Reserved;               /* 0x03 */
} MPI2_RAIDVOL0_PHYS_DISK, MPI2_POINTER PTR_MPI2_RAIDVOL0_PHYS_DISK,
  Mpi2RaidVol0PhysDisk_t, MPI2_POINTER pMpi2RaidVol0PhysDisk_t;

/* defines for the PhysDiskMap field */
#define MPI2_RAIDVOL0_PHYSDISK_PRIMARY              (0x01)
#define MPI2_RAIDVOL0_PHYSDISK_SECONDARY            (0x02)

typedef struct _MPI2_RAIDVOL0_SETTINGS
{
    U16                     Settings;               /* 0x00 */
    U8                      HotSparePool;           /* 0x01 */
    U8                      Reserved;               /* 0x02 */
} MPI2_RAIDVOL0_SETTINGS, MPI2_POINTER PTR_MPI2_RAIDVOL0_SETTINGS,
  Mpi2RaidVol0Settings_t, MPI2_POINTER pMpi2RaidVol0Settings_t;

/* RAID Volume Page 0 HotSparePool defines, also used in RAID Physical Disk */
#define MPI2_RAID_HOT_SPARE_POOL_0                  (0x01)
#define MPI2_RAID_HOT_SPARE_POOL_1                  (0x02)
#define MPI2_RAID_HOT_SPARE_POOL_2                  (0x04)
#define MPI2_RAID_HOT_SPARE_POOL_3                  (0x08)
#define MPI2_RAID_HOT_SPARE_POOL_4                  (0x10)
#define MPI2_RAID_HOT_SPARE_POOL_5                  (0x20)
#define MPI2_RAID_HOT_SPARE_POOL_6                  (0x40)
#define MPI2_RAID_HOT_SPARE_POOL_7                  (0x80)

/* RAID Volume Page 0 VolumeSettings defines */
#define MPI2_RAIDVOL0_SETTING_USE_PRODUCT_ID_SUFFIX     (0x0008)
#define MPI2_RAIDVOL0_SETTING_AUTO_CONFIG_HSWAP_DISABLE (0x0004)

#define MPI2_RAIDVOL0_SETTING_MASK_WRITE_CACHING        (0x0003)
#define MPI2_RAIDVOL0_SETTING_UNCHANGED                 (0x0000)
#define MPI2_RAIDVOL0_SETTING_DISABLE_WRITE_CACHING     (0x0001)
#define MPI2_RAIDVOL0_SETTING_ENABLE_WRITE_CACHING      (0x0002)

/*
 * Host code (drivers, BIOS, utilities, etc.) should leave this define set to
```



```
 * one and check Header.PageLength at runtime.
 */
#ifndef MPI2_RAID_VOL_PAGE_0_PHYSDISK_MAX
#define MPI2_RAID_VOL_PAGE_0_PHYSDISK_MAX       (1)
#endif

typedef struct _MPI2_CONFIG_PAGE_RAID_VOL_0
{
    MPI2_CONFIG_PAGE_HEADER Header;                     /* 0x00 */
    U16                     DevHandle;                  /* 0x04 */
    U8                      VolumeState;                /* 0x06 */
    U8                      VolumeType;                 /* 0x07 */
    U32                     VolumeStatusFlags;          /* 0x08 */
    MPI2_RAIDVOL0_SETTINGS  VolumeSettings;             /* 0x0C */
    U64                     MaxLBA;                     /* 0x10 */
    U32                     StripeSize;                 /* 0x18 */
    U16                     BlockSize;                  /* 0x1C */
    U16                     Reserved1;                  /* 0x1E */
    U8                      SupportedPhysDisks;         /* 0x20 */
    U8                      ResyncRate;                 /* 0x21 */
    U16                     DataScrubDuration;          /* 0x22 */
    U8                      NumPhysDisks;               /* 0x24 */
    U8                      Reserved2;                  /* 0x25 */
    U8                      Reserved3;                  /* 0x26 */
    U8                      InactiveStatus;             /* 0x27 */
    MPI2_RAIDVOL0_PHYS_DISK PhysDisk[MPI2_RAID_VOL_PAGE_0_PHYSDISK_MAX]; /* 0x28
} MPI2_CONFIG_PAGE_RAID_VOL_0, MPI2_POINTER PTR_MPI2_CONFIG_PAGE_RAID_VOL_0,
  Mpi2RaidVolPage0_t, MPI2_POINTER pMpi2RaidVolPage0_t;

#define MPI2_RAIDVOLPAGE0_PAGEVERSION           (0x0A)

/* values for RAID VolumeState */
#define MPI2_RAID_VOL_STATE_MISSING                         (0x00)
#define MPI2_RAID_VOL_STATE_FAILED                          (0x01)
#define MPI2_RAID_VOL_STATE_INITIALIZING                    (0x02)
#define MPI2_RAID_VOL_STATE_ONLINE                          (0x03)
#define MPI2_RAID_VOL_STATE_DEGRADED                        (0x04)
#define MPI2_RAID_VOL_STATE_OPTIMAL                         (0x05)

/* values for RAID VolumeType */
#define MPI2_RAID_VOL_TYPE_RAID0                            (0x00)
#define MPI2_RAID_VOL_TYPE_RAID1E                           (0x01)
#define MPI2_RAID_VOL_TYPE_RAID1                            (0x02)
#define MPI2_RAID_VOL_TYPE_RAID10                           (0x05)
#define MPI2_RAID_VOL_TYPE_UNKNOWN                          (0xFF)

/* values for RAID Volume Page 0 VolumeStatusFlags field */
#define MPI2_RAIDVOL0_STATUS_FLAG_PENDING_RESYNC            (0x02000000)
#define MPI2_RAIDVOL0_STATUS_FLAG_BACKG_INIT_PENDING        (0x01000000)
#define MPI2_RAIDVOL0_STATUS_FLAG_MDC_PENDING               (0x00800000)
#define MPI2_RAIDVOL0_STATUS_FLAG_USER_CONSIST_PENDING      (0x00400000)
#define MPI2_RAIDVOL0_STATUS_FLAG_MAKE_DATA_CONSISTENT      (0x00200000)
#define MPI2_RAIDVOL0_STATUS_FLAG_DATA_SCRUB                (0x00100000)
#define MPI2_RAIDVOL0_STATUS_FLAG_CONSISTENCY_CHECK         (0x00080000)
#define MPI2_RAIDVOL0_STATUS_FLAG_CAPACITY_EXPANSION        (0x00040000)
#define MPI2_RAIDVOL0_STATUS_FLAG_BACKGROUND_INIT           (0x00020000)
#define MPI2_RAIDVOL0_STATUS_FLAG_RESYNC_IN_PROGRESS        (0x00010000)
#define MPI2_RAIDVOL0_STATUS_FLAG_VOL_NOT_CONSISTENT        (0x00000080)
#endif /* ! codereview */
#define MPI2_RAIDVOL0_STATUS_FLAG_OCE_ALLOWED               (0x00000040)
#define MPI2_RAIDVOL0_STATUS_FLAG_BGI_COMPLETE              (0x00000020)
#define MPI2_RAIDVOL0_STATUS_FLAG_1E_OFFSET_MIRROR          (0x00000000)
#define MPI2_RAIDVOL0_STATUS_FLAG_1E_ADJACENT_MIRROR        (0x00000010)
#define MPI2_RAIDVOL0_STATUS_FLAG_BAD_BLOCK_TABLE_FULL      (0x00000008)
#define MPI2_RAIDVOL0_STATUS_FLAG_VOLUME_INACTIVE           (0x00000004)
#define MPI2_RAIDVOL0_STATUS_FLAG_QUIESCED                  (0x00000002)
```

```
#define MPI2_RAIDVOL0_STATUS_FLAG_ENABLED                   (0x00000001)

/* values for RAID Volume Page 0 SupportedPhysDisks field */
#define MPI2_RAIDVOL0_SUPPORT_SOLID_STATE_DISKS             (0x08)
#define MPI2_RAIDVOL0_SUPPORT_HARD_DISKS                    (0x04)
#define MPI2_RAIDVOL0_SUPPORT_SAS_PROTOCOL                  (0x02)
#define MPI2_RAIDVOL0_SUPPORT_SATA_PROTOCOL                 (0x01)

/* values for RAID Volume Page 0 InactiveStatus field */
#define MPI2_RAIDVOLPAGE0_UNKNOWN_INACTIVE                  (0x00)
#define MPI2_RAIDVOLPAGE0_STALE_METADATA_INACTIVE           (0x01)
#define MPI2_RAIDVOLPAGE0_FOREIGN_VOLUME_INACTIVE           (0x02)
#define MPI2_RAIDVOLPAGE0_INSUFFICIENT_RESOURCE_INACTIVE    (0x03)
#define MPI2_RAIDVOLPAGE0_CLONE_VOLUME_INACTIVE             (0x04)
#define MPI2_RAIDVOLPAGE0_INSUFFICIENT_METADATA_INACTIVE    (0x05)
#define MPI2_RAIDVOLPAGE0_PREVIOUSLY_DELETED                (0x06)


/* RAID Volume Page 1 */

typedef struct _MPI2_CONFIG_PAGE_RAID_VOL_1
{
    MPI2_CONFIG_PAGE_HEADER Header;                 /* 0x00 */
    U16                     DevHandle;              /* 0x04 */
    U16                     Reserved0;              /* 0x06 */
    U8                      GUID[24];               /* 0x08 */
    U8                      Name[16];               /* 0x20 */
    U64                     WWID;                   /* 0x30 */
    U32                     Reserved1;              /* 0x38 */
    U32                     Reserved2;              /* 0x3C */
} MPI2_CONFIG_PAGE_RAID_VOL_1, MPI2_POINTER PTR_MPI2_CONFIG_PAGE_RAID_VOL_1,
  Mpi2RaidVolPage1_t, MPI2_POINTER pMpi2RaidVolPage1_t;

#define MPI2_RAIDVOLPAGE1_PAGEVERSION           (0x03)


/*****************************************************************************
*   RAID Physical Disk Config Pages
*****************************************************************************/

/* RAID Physical Disk Page 0 */

typedef struct _MPI2_RAIDPHYSDISK0_SETTINGS
{
    U16                     Reserved1;              /* 0x00 */
    U8                      HotSparePool;           /* 0x02 */
    U8                      Reserved2;              /* 0x03 */
} MPI2_RAIDPHYSDISK0_SETTINGS, MPI2_POINTER PTR_MPI2_RAIDPHYSDISK0_SETTINGS,
  Mpi2RaidPhysDisk0Settings_t, MPI2_POINTER pMpi2RaidPhysDisk0Settings_t;

/* use MPI2_RAID_HOT_SPARE_POOL_ defines for the HotSparePool field */

typedef struct _MPI2_RAIDPHYSDISK0_INQUIRY_DATA
{
    U8                      VendorID[8];            /* 0x00 */
    U8                      ProductID[16];          /* 0x08 */
    U8                      ProductRevLevel[4];     /* 0x18 */
    U8                      SerialNum[32];          /* 0x1C */
} MPI2_RAIDPHYSDISK0_INQUIRY_DATA,
  MPI2_POINTER PTR_MPI2_RAIDPHYSDISK0_INQUIRY_DATA,
  Mpi2RaidPhysDisk0InquiryData_t, MPI2_POINTER pMpi2RaidPhysDisk0InquiryData_t;

typedef struct _MPI2_CONFIG_PAGE_RD_PDISK_0
{
    MPI2_CONFIG_PAGE_HEADER         Header;             /* 0x00 */
    U16                             DevHandle;          /* 0x04 */
```



```
    U8                              Reserved1;          /* 0x06 */
    U8                              PhysDiskNum;        /* 0x07 */
    MPI2_RAIDPHYSDISK0_SETTINGS     PhysDiskSettings;   /* 0x08 */
    U32                             Reserved2;          /* 0x0C */
    MPI2_RAIDPHYSDISK0_INQUIRY_DATA InquiryData;        /* 0x10 */
    U32                             Reserved3;          /* 0x4C */
    U8                              PhysDiskState;      /* 0x50 */
    U8                              OfflineReason;      /* 0x51 */
    U8                              IncompatibleReason; /* 0x52 */
    U8                              PhysDiskAttributes; /* 0x53 */
    U32                             PhysDiskStatusFlags;/* 0x54 */
    U64                             DeviceMaxLBA;       /* 0x58 */
    U64                             HostMaxLBA;         /* 0x60 */
    U64                             CoercedMaxLBA;      /* 0x68 */
    U16                             BlockSize;          /* 0x70 */
    U16                             Reserved5;          /* 0x72 */
    U32                             Reserved6;          /* 0x74 */
} MPI2_CONFIG_PAGE_RD_PDISK_0,
  MPI2_POINTER PTR_MPI2_CONFIG_PAGE_RD_PDISK_0,
  Mpi2RaidPhysDiskPage0_t, MPI2_POINTER pMpi2RaidPhysDiskPage0_t;

#define MPI2_RAIDPHYSDISKPAGE0_PAGEVERSION          (0x05)

/* PhysDiskState defines */
#define MPI2_RAID_PD_STATE_NOT_CONFIGURED               (0x00)
#define MPI2_RAID_PD_STATE_NOT_COMPATIBLE               (0x01)
#define MPI2_RAID_PD_STATE_OFFLINE                      (0x02)
#define MPI2_RAID_PD_STATE_ONLINE                       (0x03)
#define MPI2_RAID_PD_STATE_HOT_SPARE                    (0x04)
#define MPI2_RAID_PD_STATE_DEGRADED                     (0x05)
#define MPI2_RAID_PD_STATE_REBUILDING                   (0x06)
#define MPI2_RAID_PD_STATE_OPTIMAL                      (0x07)

/* OfflineReason defines */
#define MPI2_PHYSDISK0_ONLINE                           (0x00)
#define MPI2_PHYSDISK0_OFFLINE_MISSING                  (0x01)
#define MPI2_PHYSDISK0_OFFLINE_FAILED                   (0x03)
#define MPI2_PHYSDISK0_OFFLINE_INITIALIZING             (0x04)
#define MPI2_PHYSDISK0_OFFLINE_REQUESTED                (0x05)
#define MPI2_PHYSDISK0_OFFLINE_FAILED_REQUESTED         (0x06)
#define MPI2_PHYSDISK0_OFFLINE_OTHER                    (0xFF)

/* IncompatibleReason defines */
#define MPI2_PHYSDISK0_COMPATIBLE                       (0x00)
#define MPI2_PHYSDISK0_INCOMPATIBLE_PROTOCOL            (0x01)
#define MPI2_PHYSDISK0_INCOMPATIBLE_BLOCKSIZE           (0x02)
#define MPI2_PHYSDISK0_INCOMPATIBLE_MAX_LBA             (0x03)
#define MPI2_PHYSDISK0_INCOMPATIBLE_SATA_EXTENDED_CMD   (0x04)
#define MPI2_PHYSDISK0_INCOMPATIBLE_REMOVEABLE_MEDIA    (0x05)
#define MPI2_PHYSDISK0_INCOMPATIBLE_MEDIA_TYPE          (0x06)
#endif /* ! codereview */
#define MPI2_PHYSDISK0_INCOMPATIBLE_UNKNOWN             (0xFF)

/* PhysDiskAttributes defines */
#define MPI2_PHYSDISK0_ATTRIB_SOLID_STATE_DRIVE         (0x08)
#define MPI2_PHYSDISK0_ATTRIB_HARD_DISK_DRIVE           (0x04)

#define MPI2_PHYSDISK0_ATTRIB_PROTOCOL_MASK             (0x03)
#endif /* ! codereview */
#define MPI2_PHYSDISK0_ATTRIB_SAS_PROTOCOL              (0x02)
#define MPI2_PHYSDISK0_ATTRIB_SATA_PROTOCOL             (0x01)

/* PhysDiskStatusFlags defines */
#define MPI2_PHYSDISK0_STATUS_FLAG_NOT_CERTIFIED        (0x00000040)
#define MPI2_PHYSDISK0_STATUS_FLAG_OCE_TARGET           (0x00000020)
#define MPI2_PHYSDISK0_STATUS_FLAG_WRITE_CACHE_ENABLED  (0x00000010)
```

```
#define MPI2_PHYSDISK0_STATUS_FLAG_OPTIMAL_PREVIOUS     (0x00000000)
#define MPI2_PHYSDISK0_STATUS_FLAG_NOT_OPTIMAL_PREVIOUS (0x00000008)
#define MPI2_PHYSDISK0_STATUS_FLAG_INACTIVE_VOLUME      (0x00000004)
#define MPI2_PHYSDISK0_STATUS_FLAG_QUIESCED             (0x00000002)
#define MPI2_PHYSDISK0_STATUS_FLAG_OUT_OF_SYNC          (0x00000001)


/* RAID Physical Disk Page 1 */

/*
 * Host code (drivers, BIOS, utilities, etc.) should leave this define set to
 * one and check Header.PageLength or NumPhysDiskPaths at runtime.
 */
#ifndef MPI2_RAID_PHYS_DISK1_PATH_MAX
#define MPI2_RAID_PHYS_DISK1_PATH_MAX   (1)
#endif

typedef struct _MPI2_RAIDPHYSDISK1_PATH
{
    U16             DevHandle;          /* 0x00 */
    U16             Reserved1;          /* 0x02 */
    U64             WWID;               /* 0x04 */
    U64             OwnerWWID;          /* 0x0C */
    U8              OwnerIdentifier;    /* 0x14 */
    U8              Reserved2;          /* 0x15 */
    U16             Flags;              /* 0x16 */
} MPI2_RAIDPHYSDISK1_PATH, MPI2_POINTER PTR_MPI2_RAIDPHYSDISK1_PATH,
  Mpi2RaidPhysDisk1Path_t, MPI2_POINTER pMpi2RaidPhysDisk1Path_t;

/* RAID Physical Disk Page 1 Physical Disk Path Flags field defines */
#define MPI2_RAID_PHYSDISK1_FLAG_PRIMARY        (0x0004)
#define MPI2_RAID_PHYSDISK1_FLAG_BROKEN         (0x0002)
#define MPI2_RAID_PHYSDISK1_FLAG_INVALID        (0x0001)

typedef struct _MPI2_CONFIG_PAGE_RD_PDISK_1
{
    MPI2_CONFIG_PAGE_HEADER         Header;                     /* 0x00 */
    U8                              NumPhysDiskPaths;           /* 0x04 */
    U8                              PhysDiskNum;                /* 0x05 */
    U16                             Reserved1;                  /* 0x06 */
    U32                             Reserved2;                  /* 0x08 */
    MPI2_RAIDPHYSDISK1_PATH         PhysicalDiskPath[MPI2_RAID_PHYS_DISK1_PATH_M
} MPI2_CONFIG_PAGE_RD_PDISK_1,
  MPI2_POINTER PTR_MPI2_CONFIG_PAGE_RD_PDISK_1,
  Mpi2RaidPhysDiskPage1_t, MPI2_POINTER pMpi2RaidPhysDiskPage1_t;

#define MPI2_RAIDPHYSDISKPAGE1_PAGEVERSION          (0x02)


/*****************************************************************************
*   values for fields used by several types of SAS Config Pages
*****************************************************************************/

/* values for NegotiatedLinkRates fields */
#define MPI2_SAS_NEG_LINK_RATE_MASK_LOGICAL             (0xF0)
#define MPI2_SAS_NEG_LINK_RATE_SHIFT_LOGICAL            (4)
#define MPI2_SAS_NEG_LINK_RATE_MASK_PHYSICAL            (0x0F)
/* link rates used for Negotiated Physical and Logical Link Rate */
#define MPI2_SAS_NEG_LINK_RATE_UNKNOWN_LINK_RATE        (0x00)
#define MPI2_SAS_NEG_LINK_RATE_PHY_DISABLED             (0x01)
#define MPI2_SAS_NEG_LINK_RATE_NEGOTIATION_FAILED       (0x02)
#define MPI2_SAS_NEG_LINK_RATE_SATA_OOB_COMPLETE        (0x03)
#define MPI2_SAS_NEG_LINK_RATE_PORT_SELECTOR            (0x04)
#define MPI2_SAS_NEG_LINK_RATE_SMP_RESET_IN_PROGRESS    (0x05)
#define MPI2_SAS_NEG_LINK_RATE_UNSUPPORTED_PHY          (0x06)
#endif /* ! codereview */
```



```
#define MPI2_SAS_NEG_LINK_RATE_1_5                      (0x08)
#define MPI2_SAS_NEG_LINK_RATE_3_0                      (0x09)
#define MPI2_SAS_NEG_LINK_RATE_6_0                      (0x0A)
#define MPI25_SAS_NEG_LINK_RATE_12_0                    (0x0B)
#endif /* ! codereview */


/* values for AttachedPhyInfo fields */
#define MPI2_SAS_APHYINFO_INSIDE_ZPSDS_PERSISTENT       (0x00000040)
#define MPI2_SAS_APHYINFO_REQUESTED_INSIDE_ZPSDS        (0x00000020)
#define MPI2_SAS_APHYINFO_BREAK_REPLY_CAPABLE           (0x00000010)

#define MPI2_SAS_APHYINFO_REASON_MASK                   (0x0000000F)
#define MPI2_SAS_APHYINFO_REASON_UNKNOWN                (0x00000000)
#define MPI2_SAS_APHYINFO_REASON_POWER_ON               (0x00000001)
#define MPI2_SAS_APHYINFO_REASON_HARD_RESET             (0x00000002)
#define MPI2_SAS_APHYINFO_REASON_SMP_PHY_CONTROL        (0x00000003)
#define MPI2_SAS_APHYINFO_REASON_LOSS_OF_SYNC           (0x00000004)
#define MPI2_SAS_APHYINFO_REASON_MULTIPLEXING_SEQ       (0x00000005)
#define MPI2_SAS_APHYINFO_REASON_IT_NEXUS_LOSS_TIMER    (0x00000006)
#define MPI2_SAS_APHYINFO_REASON_BREAK_TIMEOUT          (0x00000007)
#define MPI2_SAS_APHYINFO_REASON_PHY_TEST_STOPPED       (0x00000008)


/* values for PhyInfo fields */
#define MPI2_SAS_PHYINFO_PHY_VACANT                     (0x80000000)

#define MPI2_SAS_PHYINFO_PHY_POWER_CONDITION_MASK       (0x18000000)
#define MPI2_SAS_PHYINFO_SHIFT_PHY_POWER_CONDITION      (27)
#endif /* ! codereview */
#define MPI2_SAS_PHYINFO_PHY_POWER_ACTIVE               (0x00000000)
#define MPI2_SAS_PHYINFO_PHY_POWER_PARTIAL              (0x08000000)
#define MPI2_SAS_PHYINFO_PHY_POWER_SLUMBER              (0x10000000)

#define MPI2_SAS_PHYINFO_CHANGED_REQ_INSIDE_ZPSDS       (0x04000000)
#define MPI2_SAS_PHYINFO_INSIDE_ZPSDS_PERSISTENT        (0x02000000)
#define MPI2_SAS_PHYINFO_REQ_INSIDE_ZPSDS               (0x01000000)
#define MPI2_SAS_PHYINFO_ZONE_GROUP_PERSISTENT          (0x00400000)
#define MPI2_SAS_PHYINFO_INSIDE_ZPSDS                   (0x00200000)
#define MPI2_SAS_PHYINFO_ZONING_ENABLED                 (0x00100000)

#define MPI2_SAS_PHYINFO_REASON_MASK                    (0x000F0000)
#define MPI2_SAS_PHYINFO_REASON_UNKNOWN                 (0x00000000)
#define MPI2_SAS_PHYINFO_REASON_POWER_ON                (0x00010000)
#define MPI2_SAS_PHYINFO_REASON_HARD_RESET              (0x00020000)
#define MPI2_SAS_PHYINFO_REASON_SMP_PHY_CONTROL         (0x00030000)
#define MPI2_SAS_PHYINFO_REASON_LOSS_OF_SYNC            (0x00040000)
#define MPI2_SAS_PHYINFO_REASON_MULTIPLEXING_SEQ        (0x00050000)
#define MPI2_SAS_PHYINFO_REASON_IT_NEXUS_LOSS_TIMER     (0x00060000)
#define MPI2_SAS_PHYINFO_REASON_BREAK_TIMEOUT           (0x00070000)
#define MPI2_SAS_PHYINFO_REASON_PHY_TEST_STOPPED        (0x00080000)

#define MPI2_SAS_PHYINFO_MULTIPLEXING_SUPPORTED         (0x00008000)
#define MPI2_SAS_PHYINFO_SATA_PORT_ACTIVE               (0x00004000)
#define MPI2_SAS_PHYINFO_SATA_PORT_SELECTOR_PRESENT     (0x00002000)
#define MPI2_SAS_PHYINFO_VIRTUAL_PHY                    (0x00001000)

#define MPI2_SAS_PHYINFO_MASK_PARTIAL_PATHWAY_TIME      (0x00000F00)
#define MPI2_SAS_PHYINFO_SHIFT_PARTIAL_PATHWAY_TIME     (8)

#define MPI2_SAS_PHYINFO_MASK_ROUTING_ATTRIBUTE         (0x000000F0)
#define MPI2_SAS_PHYINFO_DIRECT_ROUTING                 (0x00000000)
#define MPI2_SAS_PHYINFO_SUBTRACTIVE_ROUTING            (0x00000010)
#define MPI2_SAS_PHYINFO_TABLE_ROUTING                  (0x00000020)
```

```
/* values for SAS ProgrammedLinkRate fields */
#define MPI2_SAS_PRATE_MAX_RATE_MASK                    (0xF0)
#define MPI2_SAS_PRATE_MAX_RATE_NOT_PROGRAMMABLE        (0x00)
#define MPI2_SAS_PRATE_MAX_RATE_1_5                     (0x80)
#define MPI2_SAS_PRATE_MAX_RATE_3_0                     (0x90)
#define MPI2_SAS_PRATE_MAX_RATE_6_0                     (0xA0)
#define MPI2_SAS_PRATE_MIN_RATE_MASK                    (0x0F)
#define MPI2_SAS_PRATE_MIN_RATE_NOT_PROGRAMMABLE        (0x00)
#define MPI2_SAS_PRATE_MIN_RATE_1_5                     (0x08)
#define MPI2_SAS_PRATE_MIN_RATE_3_0                     (0x09)
#define MPI2_SAS_PRATE_MIN_RATE_6_0                     (0x0A)
#define MPI25_SAS_PRATE_MIN_RATE_12_0                   (0x0B)
#endif /* ! codereview */


/* values for SAS HwLinkRate fields */
#define MPI2_SAS_HWRATE_MAX_RATE_MASK                   (0xF0)
#define MPI2_SAS_HWRATE_MAX_RATE_1_5                    (0x80)
#define MPI2_SAS_HWRATE_MAX_RATE_3_0                    (0x90)
#define MPI2_SAS_HWRATE_MAX_RATE_6_0                    (0xA0)
#define MPI2_SAS_HWRATE_MIN_RATE_MASK                   (0x0F)
#define MPI2_SAS_HWRATE_MIN_RATE_1_5                    (0x08)
#define MPI2_SAS_HWRATE_MIN_RATE_3_0                    (0x09)
#define MPI2_SAS_HWRATE_MIN_RATE_6_0                    (0x0A)
#define MPI25_SAS_HWRATE_MIN_RATE_12_0                  (0x0B)
#endif /* ! codereview */



/*****************************************************************************
*   SAS IO Unit Config Pages
*****************************************************************************/

/* SAS IO Unit Page 0 */

typedef struct _MPI2_SAS_IO_UNIT0_PHY_DATA
{
    U8          Port;                   /* 0x00 */
    U8          PortFlags;              /* 0x01 */
    U8          PhyFlags;               /* 0x02 */
    U8          NegotiatedLinkRate;     /* 0x03 */
    U32         ControllerPhyDeviceInfo;/* 0x04 */
    U16         AttachedDevHandle;      /* 0x08 */
    U16         ControllerDevHandle;    /* 0x0A */
    U32         DiscoveryStatus;        /* 0x0C */
    U32         Reserved;               /* 0x10 */
} MPI2_SAS_IO_UNIT0_PHY_DATA, MPI2_POINTER PTR_MPI2_SAS_IO_UNIT0_PHY_DATA,
  Mpi2SasIOUnit0PhyData_t, MPI2_POINTER pMpi2SasIOUnit0PhyData_t;

/*
 * Host code (drivers, BIOS, utilities, etc.) should leave this define set to
 * one and check Header.ExtPageLength or NumPhys at runtime.
 */
#ifndef MPI2_SAS_IOUNIT0_PHY_MAX
#define MPI2_SAS_IOUNIT0_PHY_MAX        (1)
#endif

typedef struct _MPI2_CONFIG_PAGE_SASIOUNIT_0
{
    MPI2_CONFIG_EXTENDED_PAGE_HEADER    Header;                         /* 0
    U32                                 Reserved1;                      /* 0
    U8                                  NumPhys;                        /* 0
    U8                                  Reserved2;                      /* 0
    U16                                 Reserved3;                      /* 0
    MPI2_SAS_IO_UNIT0_PHY_DATA          PhyData[MPI2_SAS_IOUNIT0_PHY_MAX];  /* 0
} MPI2_CONFIG_PAGE_SASIOUNIT_0,
```



```
  MPI2_POINTER PTR_MPI2_CONFIG_PAGE_SASIOUNIT_0,
  Mpi2SasIOUnitPage0_t, MPI2_POINTER pMpi2SasIOUnitPage0_t;

#define MPI2_SASIOUNITPAGE0_PAGEVERSION                     (0x05)

/* values for SAS IO Unit Page 0 PortFlags */
#define MPI2_SASIOUNIT0_PORTFLAGS_DISCOVERY_IN_PROGRESS     (0x08)
#define MPI2_SASIOUNIT0_PORTFLAGS_AUTO_PORT_CONFIG          (0x01)

/* values for SAS IO Unit Page 0 PhyFlags */
#define MPI2_SASIOUNIT0_PHYFLAGS_ZONING_ENABLED             (0x10)
#define MPI2_SASIOUNIT0_PHYFLAGS_PHY_DISABLED               (0x08)

/* use MPI2_SAS_NEG_LINK_RATE_ defines for the NegotiatedLinkRate field */

/* see mpi2_sas.h for values for SAS IO Unit Page 0 ControllerPhyDeviceInfo valu

/* values for SAS IO Unit Page 0 DiscoveryStatus */
#define MPI2_SASIOUNIT0_DS_MAX_ENCLOSURES_EXCEED            (0x80000000)
#define MPI2_SASIOUNIT0_DS_MAX_EXPANDERS_EXCEED             (0x40000000)
#define MPI2_SASIOUNIT0_DS_MAX_DEVICES_EXCEED               (0x20000000)
#define MPI2_SASIOUNIT0_DS_MAX_TOPO_PHYS_EXCEED             (0x10000000)
#define MPI2_SASIOUNIT0_DS_DOWNSTREAM_INITIATOR             (0x08000000)
#define MPI2_SASIOUNIT0_DS_MULTI_SUBTRACTIVE_SUBTRACTIVE    (0x00008000)
#define MPI2_SASIOUNIT0_DS_EXP_MULTI_SUBTRACTIVE            (0x00004000)
#define MPI2_SASIOUNIT0_DS_MULTI_PORT_DOMAIN                (0x00002000)
#define MPI2_SASIOUNIT0_DS_TABLE_TO_SUBTRACTIVE_LINK        (0x00001000)
#define MPI2_SASIOUNIT0_DS_UNSUPPORTED_DEVICE               (0x00000800)
#define MPI2_SASIOUNIT0_DS_TABLE_LINK                       (0x00000400)
#define MPI2_SASIOUNIT0_DS_SUBTRACTIVE_LINK                 (0x00000200)
#define MPI2_SASIOUNIT0_DS_SMP_CRC_ERROR                    (0x00000100)
#define MPI2_SASIOUNIT0_DS_SMP_FUNCTION_FAILED              (0x00000080)
#define MPI2_SASIOUNIT0_DS_INDEX_NOT_EXIST                  (0x00000040)
#define MPI2_SASIOUNIT0_DS_OUT_ROUTE_ENTRIES                (0x00000020)
#define MPI2_SASIOUNIT0_DS_SMP_TIMEOUT                      (0x00000010)
#define MPI2_SASIOUNIT0_DS_MULTIPLE_PORTS                   (0x00000004)
#define MPI2_SASIOUNIT0_DS_UNADDRESSABLE_DEVICE             (0x00000002)
#define MPI2_SASIOUNIT0_DS_LOOP_DETECTED                    (0x00000001)


/* SAS IO Unit Page 1 */

typedef struct _MPI2_SAS_IO_UNIT1_PHY_DATA
{
    U8          Port;                       /* 0x00 */
    U8          PortFlags;                  /* 0x01 */
    U8          PhyFlags;                   /* 0x02 */
    U8          MaxMinLinkRate;             /* 0x03 */
    U32         ControllerPhyDeviceInfo;    /* 0x04 */
    U16         MaxTargetPortConnectTime;   /* 0x08 */
    U16         Reserved1;                  /* 0x0A */
} MPI2_SAS_IO_UNIT1_PHY_DATA, MPI2_POINTER PTR_MPI2_SAS_IO_UNIT1_PHY_DATA,
  Mpi2SasIOUnit1PhyData_t, MPI2_POINTER pMpi2SasIOUnit1PhyData_t;

/*
 * Host code (drivers, BIOS, utilities, etc.) should leave this define set to
 * one and check Header.ExtPageLength or NumPhys at runtime.
 */
#ifndef MPI2_SAS_IOUNIT1_PHY_MAX
#define MPI2_SAS_IOUNIT1_PHY_MAX        (1)
#endif

typedef struct _MPI2_CONFIG_PAGE_SASIOUNIT_1
{
    MPI2_CONFIG_EXTENDED_PAGE_HEADER    Header;                         /* 0
    U16                                 ControlFlags;                   /* 0
```

```
    U16                                 SASNarrowMaxQueueDepth;             /* 0
    U16                                 AdditionalControlFlags;             /* 0
    U16                                 SASWideMaxQueueDepth;               /* 0
    U8                                  NumPhys;                            /* 0
    U8                                  SATAMaxQDepth;                      /* 0
    U8                                  ReportDeviceMissingDelay;           /* 0
    U8                                  IODeviceMissingDelay;               /* 0
    MPI2_SAS_IO_UNIT1_PHY_DATA          PhyData[MPI2_SAS_IOUNIT1_PHY_MAX];  /* 0
} MPI2_CONFIG_PAGE_SASIOUNIT_1,
  MPI2_POINTER PTR_MPI2_CONFIG_PAGE_SASIOUNIT_1,
  Mpi2SasIOUnitPage1_t, MPI2_POINTER pMpi2SasIOUnitPage1_t;

#define MPI2_SASIOUNITPAGE1_PAGEVERSION     (0x09)

/* values for SAS IO Unit Page 1 ControlFlags */
#define MPI2_SASIOUNIT1_CONTROL_DEVICE_SELF_TEST                (0x8000)
#define MPI2_SASIOUNIT1_CONTROL_SATA_3_0_MAX                    (0x4000)
#define MPI2_SASIOUNIT1_CONTROL_SATA_1_5_MAX                    (0x2000)
#define MPI2_SASIOUNIT1_CONTROL_SATA_SW_PRESERVE                (0x1000)

#define MPI2_SASIOUNIT1_CONTROL_MASK_DEV_SUPPORT                (0x0600)
#define MPI2_SASIOUNIT1_CONTROL_SHIFT_DEV_SUPPORT               (9)
#define MPI2_SASIOUNIT1_CONTROL_DEV_SUPPORT_BOTH                (0x0)
#define MPI2_SASIOUNIT1_CONTROL_DEV_SAS_SUPPORT                 (0x1)
#define MPI2_SASIOUNIT1_CONTROL_DEV_SATA_SUPPORT                (0x2)

#define MPI2_SASIOUNIT1_CONTROL_SATA_48BIT_LBA_REQUIRED         (0x0080)
#define MPI2_SASIOUNIT1_CONTROL_SATA_SMART_REQUIRED             (0x0040)
#define MPI2_SASIOUNIT1_CONTROL_SATA_NCQ_REQUIRED               (0x0020)
#define MPI2_SASIOUNIT1_CONTROL_SATA_FUA_REQUIRED               (0x0010)
#define MPI2_SASIOUNIT1_CONTROL_TABLE_SUBTRACTIVE_ILLEGAL       (0x0008)
#define MPI2_SASIOUNIT1_CONTROL_SUBTRACTIVE_ILLEGAL             (0x0004)
#define MPI2_SASIOUNIT1_CONTROL_FIRST_LVL_DISC_ONLY             (0x0002)
#define MPI2_SASIOUNIT1_CONTROL_CLEAR_AFFILIATION               (0x0001)

/* values for SAS IO Unit Page 1 AdditionalControlFlags */
#define MPI2_SASIOUNIT1_ACONTROL_MULTI_PORT_DOMAIN_ILLEGAL      (0x0080)
#define MPI2_SASIOUNIT1_ACONTROL_SATA_ASYNCHROUNOUS_NOTIFICATION (0x0040)
#define MPI2_SASIOUNIT1_ACONTROL_INVALID_TOPOLOGY_CORRECTION    (0x0020)
#define MPI2_SASIOUNIT1_ACONTROL_PORT_ENABLE_ONLY_SATA_LINK_RESET (0x0010)
#define MPI2_SASIOUNIT1_ACONTROL_OTHER_AFFILIATION_SATA_LINK_RESET (0x0008)
#define MPI2_SASIOUNIT1_ACONTROL_SELF_AFFILIATION_SATA_LINK_RESET (0x0004)
#define MPI2_SASIOUNIT1_ACONTROL_NO_AFFILIATION_SATA_LINK_RESET (0x0002)
#define MPI2_SASIOUNIT1_ACONTROL_ALLOW_TABLE_TO_TABLE           (0x0001)

/* defines for SAS IO Unit Page 1 ReportDeviceMissingDelay */
#define MPI2_SASIOUNIT1_REPORT_MISSING_TIMEOUT_MASK             (0x7F)
#define MPI2_SASIOUNIT1_REPORT_MISSING_UNIT_16                  (0x80)

/* values for SAS IO Unit Page 1 PortFlags */
#define MPI2_SASIOUNIT1_PORT_FLAGS_AUTO_PORT_CONFIG             (0x01)

/* values for SAS IO Unit Page 1 PhyFlags */
#define MPI2_SASIOUNIT1_PHYFLAGS_ZONING_ENABLE                  (0x10)
#define MPI2_SASIOUNIT1_PHYFLAGS_PHY_DISABLE                    (0x08)

/* values for SAS IO Unit Page 1 MaxMinLinkRate */
#define MPI2_SASIOUNIT1_MAX_RATE_MASK                           (0xF0)
#define MPI2_SASIOUNIT1_MAX_RATE_1_5                            (0x80)
#define MPI2_SASIOUNIT1_MAX_RATE_3_0                            (0x90)
#define MPI2_SASIOUNIT1_MAX_RATE_6_0                            (0xA0)
#define MPI25_SASIOUNIT1_MAX_RATE_12_0                          (0xB0)
#endif /* ! codereview */
#define MPI2_SASIOUNIT1_MIN_RATE_MASK                           (0x0F)
#define MPI2_SASIOUNIT1_MIN_RATE_1_5                            (0x08)
#define MPI2_SASIOUNIT1_MIN_RATE_3_0                            (0x09)
```



```
#define MPI2_SASIOUNIT1_MIN_RATE_6_0                            (0x0A)
#define MPI25_SASIOUNIT1_MIN_RATE_12_0                          (0x0B)
#endif /* ! codereview */

/* see mpi2_sas.h for values for SAS IO Unit Page 1 ControllerPhyDeviceInfo valu


/* SAS IO Unit Page 4 */

typedef struct _MPI2_SAS_IOUNIT4_SPINUP_GROUP
{
    U8          MaxTargetSpinup;            /* 0x00 */
    U8          SpinupDelay;                /* 0x01 */
    U16         Reserved1;                  /* 0x02 */
} MPI2_SAS_IOUNIT4_SPINUP_GROUP, MPI2_POINTER PTR_MPI2_SAS_IOUNIT4_SPINUP_GROUP,
  Mpi2SasIOUnit4SpinupGroup_t, MPI2_POINTER pMpi2SasIOUnit4SpinupGroup_t;

/*
 * Host code (drivers, BIOS, utilities, etc.) should leave this define set to
 * four and check Header.ExtPageLength or NumPhys at runtime.
 */
#ifndef MPI2_SAS_IOUNIT4_PHY_MAX
#define MPI2_SAS_IOUNIT4_PHY_MAX        (4)
#endif

typedef struct _MPI2_CONFIG_PAGE_SASIOUNIT_4
{
    MPI2_CONFIG_EXTENDED_PAGE_HEADER    Header;                         /* 0x00
    MPI2_SAS_IOUNIT4_SPINUP_GROUP       SpinupGroupParameters[4];       /* 0x08
    U32                                 Reserved1;                      /* 0x18
    U32                                 Reserved2;                      /* 0x1C
    U32                                 Reserved3;                      /* 0x20
    U8                                  BootDeviceWaitTime;             /* 0x24
    U8                                  Reserved4;                      /* 0x25
    U16                                 Reserved5;                      /* 0x26
    U8                                  NumPhys;                        /* 0x28
    U8                                  PEInitialSpinupDelay;           /* 0x29
    U8                                  PEReplyDelay;                   /* 0x2A
    U8                                  Flags;                          /* 0x2B
    U8                                  PHY[MPI2_SAS_IOUNIT4_PHY_MAX];  /* 0x2C
} MPI2_CONFIG_PAGE_SASIOUNIT_4,
  MPI2_POINTER PTR_MPI2_CONFIG_PAGE_SASIOUNIT_4,
  Mpi2SasIOUnitPage4_t, MPI2_POINTER pMpi2SasIOUnitPage4_t;

#define MPI2_SASIOUNITPAGE4_PAGEVERSION     (0x02)

/* defines for Flags field */
#define MPI2_SASIOUNIT4_FLAGS_AUTO_PORTENABLE               (0x01)

/* defines for PHY field */
#define MPI2_SASIOUNIT4_PHY_SPINUP_GROUP_MASK               (0x03)


/* SAS IO Unit Page 5 */

typedef struct _MPI2_SAS_IO_UNIT5_PHY_PM_SETTINGS
{
    U8          ControlFlags;               /* 0x00 */
    U8          PortWidthModGroup;          /* 0x01 */
 532    U8          Reserved1;                  /* 0x01 */
1973    U16         InactivityTimerExponent;    /* 0x02 */
1974    U8          SATAPartialTimeout;         /* 0x04 */
1975    U8          Reserved2;                  /* 0x05 */
1976    U8          SATASlumberTimeout;         /* 0x06 */
1977    U8          Reserved3;                  /* 0x07 */
1978    U8          SASPartialTimeout;          /* 0x08 */
```

```
1979     U8          Reserved4;                  /* 0x09 */
1980     U8          SASSlumberTimeout;          /* 0x0A */
1981     U8          Reserved5;                  /* 0x0B */
1982 } MPI2_SAS_IO_UNIT5_PHY_PM_SETTINGS,
1983   MPI2_POINTER PTR_MPI2_SAS_IO_UNIT5_PHY_PM_SETTINGS,
1984   Mpi2SasIOUnit5PhyPmSettings_t, MPI2_POINTER pMpi2SasIOUnit5PhyPmSettings_t;

1986 /* defines for ControlFlags field */
1987 #define MPI2_SASIOUNIT5_CONTROL_SAS_SLUMBER_ENABLE      (0x08)
1988 #define MPI2_SASIOUNIT5_CONTROL_SAS_PARTIAL_ENABLE      (0x04)
1989 #define MPI2_SASIOUNIT5_CONTROL_SATA_SLUMBER_ENABLE     (0x02)
1990 #define MPI2_SASIOUNIT5_CONTROL_SATA_PARTIAL_ENABLE     (0x01)

1992 /* defines for PortWidthModeGroup field */
1993 #define MPI2_SASIOUNIT5_PWMG_DISABLE                    (0xFF)

1995 #endif /* ! codereview */
1996 /* defines for InactivityTimerExponent field */
1997 #define MPI2_SASIOUNIT5_ITE_MASK_SAS_SLUMBER            (0x7000)
1998 #define MPI2_SASIOUNIT5_ITE_SHIFT_SAS_SLUMBER           (12)
1999 #define MPI2_SASIOUNIT5_ITE_MASK_SAS_PARTIAL            (0x0700)
2000 #define MPI2_SASIOUNIT5_ITE_SHIFT_SAS_PARTIAL           (8)
2001 #define MPI2_SASIOUNIT5_ITE_MASK_SATA_SLUMBER           (0x0070)
2002 #define MPI2_SASIOUNIT5_ITE_SHIFT_SATA_SLUMBER          (4)
2003 #define MPI2_SASIOUNIT5_ITE_MASK_SATA_PARTIAL           (0x0007)
2004 #define MPI2_SASIOUNIT5_ITE_SHIFT_SATA_PARTIAL          (0)

2006 #define MPI2_SASIOUNIT5_ITE_TEN_SECONDS                 (7)
2007 #define MPI2_SASIOUNIT5_ITE_ONE_SECOND                  (6)
2008 #define MPI2_SASIOUNIT5_ITE_HUNDRED_MILLISECONDS        (5)
2009 #define MPI2_SASIOUNIT5_ITE_TEN_MILLISECONDS            (4)
2010 #define MPI2_SASIOUNIT5_ITE_ONE_MILLISECOND             (3)
2011 #define MPI2_SASIOUNIT5_ITE_HUNDRED_MICROSECONDS        (2)
2012 #define MPI2_SASIOUNIT5_ITE_TEN_MICROSECONDS            (1)
2013 #define MPI2_SASIOUNIT5_ITE_ONE_MICROSECOND             (0)

2015 /*
2016  * Host code (drivers, BIOS, utilities, etc.) should leave this define set to
2017  * one and check Header.ExtPageLength or NumPhys at runtime.
2018  */
2019 #ifndef MPI2_SAS_IOUNIT5_PHY_MAX
2020 #define MPI2_SAS_IOUNIT5_PHY_MAX        (1)
2021 #endif

2023 typedef struct _MPI2_CONFIG_PAGE_SASIOUNIT_5
2024 {
2025     MPI2_CONFIG_EXTENDED_PAGE_HEADER    Header;                             /* 0
2026     U8                                  NumPhys;                            /* 0
2027     U8                                  Reserved1;                          /* 0
2028     U16                                 Reserved2;                          /* 0
2029     U32                                 Reserved3;                          /* 0
2030     MPI2_SAS_IO_UNIT5_PHY_PM_SETTINGS   SASPhyPowerManagementSettings[MPI2_SAS_I
2031 } MPI2_CONFIG_PAGE_SASIOUNIT_5,
2032   MPI2_POINTER PTR_MPI2_CONFIG_PAGE_SASIOUNIT_5,
2033   Mpi2SasIOUnitPage5_t, MPI2_POINTER pMpi2SasIOUnitPage5_t;

2035 #define MPI2_SASIOUNITPAGE5_PAGEVERSION     (0x01)


2038 /* SAS IO Unit Page 6 */

2040 typedef struct _MPI2_SAS_IO_UNIT6_PORT_WIDTH_MOD_GROUP_STATUS
2041 {
2042     U8          CurrentStatus;              /* 0x00 */
2043     U8          CurrentModulation;          /* 0x01 */
2044     U8          CurrentUtilization;         /* 0x02 */
```



```
2045     U8          Reserved1;                  /* 0x03 */
2046     U32         Reserved2;                  /* 0x04 */
2047 } MPI2_SAS_IO_UNIT6_PORT_WIDTH_MOD_GROUP_STATUS,
2048   MPI2_POINTER PTR_MPI2_SAS_IO_UNIT6_PORT_WIDTH_MOD_GROUP_STATUS,
2049   Mpi2SasIOUnit6PortWidthModGroupStatus_t,
2050   MPI2_POINTER pMpi2SasIOUnit6PortWidthModGroupStatus_t;

2052 /* defines for CurrentStatus field */
2053 #define MPI2_SASIOUNIT6_STATUS_UNAVAILABLE                      (0x00)
2054 #define MPI2_SASIOUNIT6_STATUS_UNCONFIGURED                     (0x01)
2055 #define MPI2_SASIOUNIT6_STATUS_INVALID_CONFIG                   (0x02)
2056 #define MPI2_SASIOUNIT6_STATUS_LINK_DOWN                        (0x03)
2057 #define MPI2_SASIOUNIT6_STATUS_OBSERVATION_ONLY                 (0x04)
2058 #define MPI2_SASIOUNIT6_STATUS_INACTIVE                         (0x05)
2059 #define MPI2_SASIOUNIT6_STATUS_ACTIVE_IOUNIT                    (0x06)
2060 #define MPI2_SASIOUNIT6_STATUS_ACTIVE_HOST                      (0x07)

2062 /* defines for CurrentModulation field */
2063 #define MPI2_SASIOUNIT6_MODULATION_25_PERCENT                   (0x00)
2064 #define MPI2_SASIOUNIT6_MODULATION_50_PERCENT                   (0x01)
2065 #define MPI2_SASIOUNIT6_MODULATION_75_PERCENT                   (0x02)
2066 #define MPI2_SASIOUNIT6_MODULATION_100_PERCENT                  (0x03)

2068 /*
2069  * Host code (drivers, BIOS, utilities, etc.) should leave this define set to
2070  * one and check the value returned for NumGroups at runtime.
2071  */
2072 #ifndef MPI2_SAS_IOUNIT6_GROUP_MAX
2073 #define MPI2_SAS_IOUNIT6_GROUP_MAX      (1)
2074 #endif

2076 typedef struct _MPI2_CONFIG_PAGE_SASIOUNIT_6
2077 {
2078     MPI2_CONFIG_EXTENDED_PAGE_HEADER    Header;                         /* 0x00 */
2079     U32                                 Reserved1;                      /* 0x08 */
2080     U32                                 Reserved2;                      /* 0x0C */
2081     U8                                  NumGroups;                      /* 0x10 */
2082     U8                                  Reserved3;                      /* 0x11 */
2083     U16                                 Reserved4;                      /* 0x12 */
2084     MPI2_SAS_IO_UNIT6_PORT_WIDTH_MOD_GROUP_STATUS
2085         PortWidthModulationGroupStatus[MPI2_SAS_IOUNIT6_GROUP_MAX]; /* 0x14 */
2086 } MPI2_CONFIG_PAGE_SASIOUNIT_6,
2087   MPI2_POINTER PTR_MPI2_CONFIG_PAGE_SASIOUNIT_6,
2088   Mpi2SasIOUnitPage6_t, MPI2_POINTER pMpi2SasIOUnitPage6_t;

2090 #define MPI2_SASIOUNITPAGE6_PAGEVERSION     (0x00)


2093 /* SAS IO Unit Page 7 */

2095 typedef struct _MPI2_SAS_IO_UNIT7_PORT_WIDTH_MOD_GROUP_SETTINGS
2096 {
2097     U8          Flags;                      /* 0x00 */
2098     U8          Reserved1;                  /* 0x01 */
2099     U16         Reserved2;                  /* 0x02 */
2100     U8          Threshold75Pct;             /* 0x04 */
2101     U8          Threshold50Pct;             /* 0x05 */
2102     U8          Threshold25Pct;             /* 0x06 */
2103     U8          Reserved3;                  /* 0x07 */
2104 } MPI2_SAS_IO_UNIT7_PORT_WIDTH_MOD_GROUP_SETTINGS,
2105   MPI2_POINTER PTR_MPI2_SAS_IO_UNIT7_PORT_WIDTH_MOD_GROUP_SETTINGS,
2106   Mpi2SasIOUnit7PortWidthModGroupSettings_t,
2107   MPI2_POINTER pMpi2SasIOUnit7PortWidthModGroupSettings_t;

2109 /* defines for Flags field */
2110 #define MPI2_SASIOUNIT7_FLAGS_ENABLE_PORT_WIDTH_MODULATION  (0x01)
```

```
2113 /*
2114  * Host code (drivers, BIOS, utilities, etc.) should leave this define set to
2115  * one and check the value returned for NumGroups at runtime.
2116  */
2117 #ifndef MPI2_SAS_IOUNIT7_GROUP_MAX
2118 #define MPI2_SAS_IOUNIT7_GROUP_MAX      (1)
2119 #endif

2121 typedef struct _MPI2_CONFIG_PAGE_SASIOUNIT_7
2122 {
2123     MPI2_CONFIG_EXTENDED_PAGE_HEADER           Header;             /* 0x00 */
2124     U8                                         SamplingInterval;   /* 0x08 */
2125     U8                                         WindowLength;       /* 0x09 */
2126     U16                                        Reserved1;          /* 0x0A */
2127     U32                                        Reserved2;          /* 0x0C */
2128     U32                                        Reserved3;          /* 0x10 */
2129     U8                                         NumGroups;          /* 0x14 */
2130     U8                                         Reserved4;          /* 0x15 */
2131     U16                                        Reserved5;          /* 0x16 */
2132     MPI2_SAS_IO_UNIT7_PORT_WIDTH_MOD_GROUP_SETTINGS
2133         PortWidthModulationGroupSettings[MPI2_SAS_IOUNIT7_GROUP_MAX]; /* 0x18 */
2134 } MPI2_CONFIG_PAGE_SASIOUNIT_7,
2135   MPI2_POINTER PTR_MPI2_CONFIG_PAGE_SASIOUNIT_7,
2136   Mpi2SasIOUnitPage7_t, MPI2_POINTER pMpi2SasIOUnitPage7_t;

2138 #define MPI2_SASIOUNITPAGE7_PAGEVERSION     (0x00)


2141 /* SAS IO Unit Page 8 */

2143 typedef struct _MPI2_CONFIG_PAGE_SASIOUNIT_8
2144 {
2145     MPI2_CONFIG_EXTENDED_PAGE_HEADER    Header;                         /* 0
2146     U32                                 Reserved1;                      /* 0
2147     U32                                 PowerManagementCapabilities;    /* 0
2148     U32                                 Reserved2;                      /* 0
2149 } MPI2_CONFIG_PAGE_SASIOUNIT_8,
2150   MPI2_POINTER PTR_MPI2_CONFIG_PAGE_SASIOUNIT_8,
2151   Mpi2SasIOUnitPage8_t, MPI2_POINTER pMpi2SasIOUnitPage8_t;

2153 #define MPI2_SASIOUNITPAGE8_PAGEVERSION     (0x00)

2155 /* defines for PowerManagementCapabilities field */
2156 #define MPI2_SASIOUNIT8_PM_HOST_PORT_WIDTH_MOD          (0x000001000)
2157 #define MPI2_SASIOUNIT8_PM_HOST_SAS_SLUMBER_MODE        (0x000000800)
2158 #define MPI2_SASIOUNIT8_PM_HOST_SAS_PARTIAL_MODE        (0x000000400)
2159 #define MPI2_SASIOUNIT8_PM_HOST_SATA_SLUMBER_MODE       (0x000000200)
2160 #define MPI2_SASIOUNIT8_PM_HOST_SATA_PARTIAL_MODE       (0x000000100)
2161 #define MPI2_SASIOUNIT8_PM_IOUNIT_PORT_WIDTH_MOD        (0x000000010)
2162 #define MPI2_SASIOUNIT8_PM_IOUNIT_SAS_SLUMBER_MODE      (0x000000008)
2163 #define MPI2_SASIOUNIT8_PM_IOUNIT_SAS_PARTIAL_MODE      (0x000000004)
2164 #define MPI2_SASIOUNIT8_PM_IOUNIT_SATA_SLUMBER_MODE     (0x000000002)
2165 #define MPI2_SASIOUNIT8_PM_IOUNIT_SATA_PARTIAL_MODE     (0x000000001)
 552 #define MPI2_SASIOUNITPAGE5_PAGEVERSION     (0x00)



2170 /*****************************************************************************
2171 *   SAS Expander Config Pages
2172 *****************************************************************************/

2174 /* SAS Expander Page 0 */
```



```
2176 typedef struct _MPI2_CONFIG_PAGE_EXPANDER_0
2177 {
2178     MPI2_CONFIG_EXTENDED_PAGE_HEADER    Header;                     /* 0x00 */
2179     U8                                  PhysicalPort;               /* 0x08 */
2180     U8                                  ReportGenLength;            /* 0x09 */
2181     U16                                 EnclosureHandle;            /* 0x0A */
2182     U64                                 SASAddress;                 /* 0x0C */
2183     U32                                 DiscoveryStatus;            /* 0x14 */
2184     U16                                 DevHandle;                  /* 0x18 */
2185     U16                                 ParentDevHandle;            /* 0x1A */
2186     U16                                 ExpanderChangeCount;        /* 0x1C */
2187     U16                                 ExpanderRouteIndexes;       /* 0x1E */
2188     U8                                  NumPhys;                    /* 0x20 */
2189     U8                                  SASLevel;                   /* 0x21 */
2190     U16                                 Flags;                      /* 0x22 */
2191     U16                                 STPBusInactivityTimeLimit;  /* 0x24 */
2192     U16                                 STPMaxConnectTimeLimit;     /* 0x26 */
2193     U16                                 STP_SMP_NexusLossTime;      /* 0x28 */
2194     U16                                 MaxNumRoutedSasAddresses;   /* 0x2A */
2195     U64                                 ActiveZoneManagerSASAddress;/* 0x2C */
2196     U16                                 ZoneLockInactivityLimit;    /* 0x34 */
2197     U16                                 Reserved1;                  /* 0x36 */
2198     U8                                  TimeToReducedFunc;          /* 0x38 */
2199     U8                                  InitialTimeToReducedFunc;   /* 0x39 */
2200     U8                                  MaxReducedFuncTime;         /* 0x3A */
2201     U8                                  Reserved2;                  /* 0x3B */
2202 } MPI2_CONFIG_PAGE_EXPANDER_0, MPI2_POINTER PTR_MPI2_CONFIG_PAGE_EXPANDER_0,
______unchanged_portion_omitted_
2323   Mpi2SasDevicePage0_t, MPI2_POINTER pMpi2SasDevicePage0_t;

2325 #define MPI2_SASDEVICE0_PAGEVERSION         (0x08)

2327 /* values for SAS Device Page 0 AccessStatus field */
2328 #define MPI2_SAS_DEVICE0_ASTATUS_NO_ERRORS                  (0x00)
2329 #define MPI2_SAS_DEVICE0_ASTATUS_SATA_INIT_FAILED           (0x01)
2330 #define MPI2_SAS_DEVICE0_ASTATUS_SATA_CAPABILITY_FAILED     (0x02)
2331 #define MPI2_SAS_DEVICE0_ASTATUS_SATA_AFFILIATION_CONFLICT  (0x03)
2332 #define MPI2_SAS_DEVICE0_ASTATUS_SATA_NEEDS_INITIALIZATION  (0x04)
2333 #define MPI2_SAS_DEVICE0_ASTATUS_ROUTE_NOT_ADDRESSABLE      (0x05)
2334 #define MPI2_SAS_DEVICE0_ASTATUS_SMP_ERROR_NOT_ADDRESSABLE  (0x06)
2335 #define MPI2_SAS_DEVICE0_ASTATUS_DEVICE_BLOCKED             (0x07)
2336 /* specific values for SATA Init failures */
2337 #define MPI2_SAS_DEVICE0_ASTATUS_SIF_UNKNOWN                (0x10)
2338 #define MPI2_SAS_DEVICE0_ASTATUS_SIF_AFFILIATION_CONFLICT   (0x11)
2339 #define MPI2_SAS_DEVICE0_ASTATUS_SIF_DIAG                   (0x12)
2340 #define MPI2_SAS_DEVICE0_ASTATUS_SIF_IDENTIFICATION         (0x13)
2341 #define MPI2_SAS_DEVICE0_ASTATUS_SIF_CHECK_POWER            (0x14)
2342 #define MPI2_SAS_DEVICE0_ASTATUS_SIF_PIO_SN                 (0x15)
2343 #define MPI2_SAS_DEVICE0_ASTATUS_SIF_MDMA_SN                (0x16)
2344 #define MPI2_SAS_DEVICE0_ASTATUS_SIF_UDMA_SN                (0x17)
2345 #define MPI2_SAS_DEVICE0_ASTATUS_SIF_ZONING_VIOLATION       (0x18)
2346 #define MPI2_SAS_DEVICE0_ASTATUS_SIF_NOT_ADDRESSABLE        (0x19)
2347 #define MPI2_SAS_DEVICE0_ASTATUS_SIF_MAX                    (0x1F)

2349 /* see mpi2_sas.h for values for SAS Device Page 0 DeviceInfo values */

2351 /* values for SAS Device Page 0 Flags field */
2352 #define MPI25_SAS_DEVICE0_FLAGS_ENABLED_FAST_PATH           (0x4000)
2353 #define MPI25_SAS_DEVICE0_FLAGS_FAST_PATH_CAPABLE           (0x2000)
2354 #endif /* ! codereview */
2355 #define MPI2_SAS_DEVICE0_FLAGS_SLUMBER_PM_CAPABLE           (0x1000)
2356 #define MPI2_SAS_DEVICE0_FLAGS_PARTIAL_PM_CAPABLE           (0x0800)
2357 #define MPI2_SAS_DEVICE0_FLAGS_SATA_ASYNCHRONOUS_NOTIFY     (0x0400)
2358 #define MPI2_SAS_DEVICE0_FLAGS_SATA_SW_PRESERVE             (0x0200)
2359 #define MPI2_SAS_DEVICE0_FLAGS_UNSUPPORTED_DEVICE           (0x0100)
2360 #define MPI2_SAS_DEVICE0_FLAGS_SATA_48BIT_LBA_SUPPORTED     (0x0080)
```

```
#define MPI2_SAS_DEVICE0_FLAGS_SATA_SMART_SUPPORTED         (0x0040)
#define MPI2_SAS_DEVICE0_FLAGS_SATA_NCQ_SUPPORTED           (0x0020)
#define MPI2_SAS_DEVICE0_FLAGS_SATA_FUA_SUPPORTED           (0x0010)
#define MPI2_SAS_DEVICE0_FLAGS_PORT_SELECTOR_ATTACH         (0x0008)
#define MPI2_SAS_DEVICE0_FLAGS_DEVICE_PRESENT               (0x0001)


/* SAS Device Page 1 */

typedef struct _MPI2_CONFIG_PAGE_SAS_DEV_1
{
    MPI2_CONFIG_EXTENDED_PAGE_HEADER    Header;                 /* 0x00 */
    U32                                 Reserved1;              /* 0x08 */
    U64                                 SASAddress;             /* 0x0C */
    U32                                 Reserved2;              /* 0x14 */
    U16                                 DevHandle;              /* 0x18 */
    U16                                 Reserved3;              /* 0x1A */
    U8                                  InitialRegDeviceFIS[20];/* 0x1C */
} MPI2_CONFIG_PAGE_SAS_DEV_1, MPI2_POINTER PTR_MPI2_CONFIG_PAGE_SAS_DEV_1,
  Mpi2SasDevicePage1_t, MPI2_POINTER pMpi2SasDevicePage1_t;

#define MPI2_SASDEVICE1_PAGEVERSION         (0x01)


/*****************************************************************************
*   SAS PHY Config Pages
*****************************************************************************/

/* SAS PHY Page 0 */

typedef struct _MPI2_CONFIG_PAGE_SAS_PHY_0
{
    MPI2_CONFIG_EXTENDED_PAGE_HEADER    Header;                 /* 0x00 */
    U16                                 OwnerDevHandle;         /* 0x08 */
    U16                                 Reserved1;              /* 0x0A */
    U16                                 AttachedDevHandle;      /* 0x0C */
    U8                                  AttachedPhyIdentifier;  /* 0x0E */
    U8                                  Reserved2;              /* 0x0F */
    U32                                 AttachedPhyInfo;        /* 0x10 */
    U8                                  ProgrammedLinkRate;     /* 0x14 */
    U8                                  HwLinkRate;             /* 0x15 */
    U8                                  ChangeCount;            /* 0x16 */
    U8                                  Flags;                  /* 0x17 */
    U32                                 PhyInfo;                /* 0x18 */
    U8                                  NegotiatedLinkRate;     /* 0x1C */
    U8                                  Reserved3;              /* 0x1D */
    U16                                 Reserved4;              /* 0x1E */
} MPI2_CONFIG_PAGE_SAS_PHY_0, MPI2_POINTER PTR_MPI2_CONFIG_PAGE_SAS_PHY_0,
  Mpi2SasPhyPage0_t, MPI2_POINTER pMpi2SasPhyPage0_t;

#define MPI2_SASPHY0_PAGEVERSION            (0x03)

/* use MPI2_SAS_PRATE_ defines for the ProgrammedLinkRate field */

/* use MPI2_SAS_HWRATE_ defines for the HwLinkRate field */

/* values for SAS PHY Page 0 Flags field */
#define MPI2_SAS_PHY0_FLAGS_SGPIO_DIRECT_ATTACH_ENC             (0x01)

/* use MPI2_SAS_APHYINFO_ defines for AttachedPhyInfo field */

/* use MPI2_SAS_NEG_LINK_RATE_ defines for the NegotiatedLinkRate field */

/* use MPI2_SAS_PHYINFO_ for the PhyInfo field */
```



```
/* SAS PHY Page 1 */

typedef struct _MPI2_CONFIG_PAGE_SAS_PHY_1
{
    MPI2_CONFIG_EXTENDED_PAGE_HEADER    Header;                     /* 0x00 */
    U32                                 Reserved1;                  /* 0x08 */
    U32                                 InvalidDwordCount;          /* 0x0C */
    U32                                 RunningDisparityErrorCount; /* 0x10 */
    U32                                 LossDwordSynchCount;        /* 0x14 */
    U32                                 PhyResetProblemCount;       /* 0x18 */
} MPI2_CONFIG_PAGE_SAS_PHY_1, MPI2_POINTER PTR_MPI2_CONFIG_PAGE_SAS_PHY_1,
  Mpi2SasPhyPage1_t, MPI2_POINTER pMpi2SasPhyPage1_t;

#define MPI2_SASPHY1_PAGEVERSION            (0x01)


/* SAS PHY Page 2 */

typedef struct _MPI2_SASPHY2_PHY_EVENT
{
    U8          PhyEventCode;       /* 0x00 */
    U8          Reserved1;          /* 0x01 */
    U16         Reserved2;          /* 0x02 */
    U32         PhyEventInfo;       /* 0x04 */
} MPI2_SASPHY2_PHY_EVENT, MPI2_POINTER PTR_MPI2_SASPHY2_PHY_EVENT,
  Mpi2SasPhy2PhyEvent_t, MPI2_POINTER pMpi2SasPhy2PhyEvent_t;

/* use MPI2_SASPHY3_EVENT_CODE_ for the PhyEventCode field */


/*
 * Host code (drivers, BIOS, utilities, etc.) should leave this define set to
 * one and check Header.ExtPageLength or NumPhyEvents at runtime.
 */
#ifndef MPI2_SASPHY2_PHY_EVENT_MAX
#define MPI2_SASPHY2_PHY_EVENT_MAX      (1)
#endif

typedef struct _MPI2_CONFIG_PAGE_SAS_PHY_2
{
    MPI2_CONFIG_EXTENDED_PAGE_HEADER    Header;                     /* 0x00 */
    U32                                 Reserved1;                  /* 0x08 */
    U8                                  NumPhyEvents;               /* 0x0C */
    U8                                  Reserved2;                  /* 0x0D */
    U16                                 Reserved3;                  /* 0x0E */
    MPI2_SASPHY2_PHY_EVENT              PhyEvent[MPI2_SASPHY2_PHY_EVENT_MAX]; /*
} MPI2_CONFIG_PAGE_SAS_PHY_2, MPI2_POINTER PTR_MPI2_CONFIG_PAGE_SAS_PHY_2,
  Mpi2SasPhyPage2_t, MPI2_POINTER pMpi2SasPhyPage2_t;

#define MPI2_SASPHY2_PAGEVERSION            (0x00)


/* SAS PHY Page 3 */

typedef struct _MPI2_SASPHY3_PHY_EVENT_CONFIG
{
    U8          PhyEventCode;       /* 0x00 */
    U8          Reserved1;          /* 0x01 */
    U16         Reserved2;          /* 0x02 */
    U8          CounterType;        /* 0x04 */
    U8          ThresholdWindow;    /* 0x05 */
    U8          TimeUnits;          /* 0x06 */
    U8          Reserved3;          /* 0x07 */
    U32         EventThreshold;     /* 0x08 */
    U16         ThresholdFlags;     /* 0x0C */
    U16         Reserved4;          /* 0x0E */
```

```
} MPI2_SASPHY3_PHY_EVENT_CONFIG, MPI2_POINTER PTR_MPI2_SASPHY3_PHY_EVENT_CONFIG,
  Mpi2SasPhy3PhyEventConfig_t, MPI2_POINTER pMpi2SasPhy3PhyEventConfig_t;

/* values for PhyEventCode field */
#define MPI2_SASPHY3_EVENT_CODE_NO_EVENT                    (0x00)
#define MPI2_SASPHY3_EVENT_CODE_INVALID_DWORD               (0x01)
#define MPI2_SASPHY3_EVENT_CODE_RUNNING_DISPARITY_ERROR     (0x02)
#define MPI2_SASPHY3_EVENT_CODE_LOSS_DWORD_SYNC             (0x03)
#define MPI2_SASPHY3_EVENT_CODE_PHY_RESET_PROBLEM           (0x04)
#define MPI2_SASPHY3_EVENT_CODE_ELASTICITY_BUF_OVERFLOW     (0x05)
#define MPI2_SASPHY3_EVENT_CODE_RX_ERROR                    (0x06)
#define MPI2_SASPHY3_EVENT_CODE_RX_ADDR_FRAME_ERROR         (0x20)
#define MPI2_SASPHY3_EVENT_CODE_TX_AC_OPEN_REJECT           (0x21)
#define MPI2_SASPHY3_EVENT_CODE_RX_AC_OPEN_REJECT           (0x22)
#define MPI2_SASPHY3_EVENT_CODE_TX_RC_OPEN_REJECT           (0x23)
#define MPI2_SASPHY3_EVENT_CODE_RX_RC_OPEN_REJECT           (0x24)
#define MPI2_SASPHY3_EVENT_CODE_RX_AIP_PARTIAL_WAITING_ON   (0x25)
#define MPI2_SASPHY3_EVENT_CODE_RX_AIP_CONNECT_WAITING_ON   (0x26)
#define MPI2_SASPHY3_EVENT_CODE_TX_BREAK                    (0x27)
#define MPI2_SASPHY3_EVENT_CODE_RX_BREAK                    (0x28)
#define MPI2_SASPHY3_EVENT_CODE_BREAK_TIMEOUT               (0x29)
#define MPI2_SASPHY3_EVENT_CODE_CONNECTION                  (0x2A)
#define MPI2_SASPHY3_EVENT_CODE_PEAKTX_PATHWAY_BLOCKED      (0x2B)
#define MPI2_SASPHY3_EVENT_CODE_PEAKTX_ARB_WAIT_TIME        (0x2C)
#define MPI2_SASPHY3_EVENT_CODE_PEAK_ARB_WAIT_TIME          (0x2D)
#define MPI2_SASPHY3_EVENT_CODE_PEAK_CONNECT_TIME           (0x2E)
#define MPI2_SASPHY3_EVENT_CODE_TX_SSP_FRAMES               (0x40)
#define MPI2_SASPHY3_EVENT_CODE_RX_SSP_FRAMES               (0x41)
#define MPI2_SASPHY3_EVENT_CODE_TX_SSP_ERROR_FRAMES         (0x42)
#define MPI2_SASPHY3_EVENT_CODE_RX_SSP_ERROR_FRAMES         (0x43)
#define MPI2_SASPHY3_EVENT_CODE_TX_CREDIT_BLOCKED           (0x44)
#define MPI2_SASPHY3_EVENT_CODE_RX_CREDIT_BLOCKED           (0x45)
#define MPI2_SASPHY3_EVENT_CODE_TX_SATA_FRAMES              (0x50)
#define MPI2_SASPHY3_EVENT_CODE_RX_SATA_FRAMES              (0x51)
#define MPI2_SASPHY3_EVENT_CODE_SATA_OVERFLOW               (0x52)
#define MPI2_SASPHY3_EVENT_CODE_TX_SMP_FRAMES               (0x60)
#define MPI2_SASPHY3_EVENT_CODE_RX_SMP_FRAMES               (0x61)
#define MPI2_SASPHY3_EVENT_CODE_RX_SMP_ERROR_FRAMES         (0x63)
#define MPI2_SASPHY3_EVENT_CODE_HOTPLUG_TIMEOUT             (0xD0)
#define MPI2_SASPHY3_EVENT_CODE_MISALIGNED_MUX_PRIMITIVE    (0xD1)
#define MPI2_SASPHY3_EVENT_CODE_RX_AIP                      (0xD2)

/* values for the CounterType field */
#define MPI2_SASPHY3_COUNTER_TYPE_WRAPPING                  (0x00)
#define MPI2_SASPHY3_COUNTER_TYPE_SATURATING                (0x01)
#define MPI2_SASPHY3_COUNTER_TYPE_PEAK_VALUE                (0x02)

/* values for the TimeUnits field */
#define MPI2_SASPHY3_TIME_UNITS_10_MICROSECONDS             (0x00)
#define MPI2_SASPHY3_TIME_UNITS_100_MICROSECONDS            (0x01)
#define MPI2_SASPHY3_TIME_UNITS_1_MILLISECOND               (0x02)
#define MPI2_SASPHY3_TIME_UNITS_10_MILLISECONDS             (0x03)

/* values for the ThresholdFlags field */
#define MPI2_SASPHY3_TFLAGS_PHY_RESET                       (0x0002)
#define MPI2_SASPHY3_TFLAGS_EVENT_NOTIFY                    (0x0001)

/*
 * Host code (drivers, BIOS, utilities, etc.) should leave this define set to
 * one and check Header.ExtPageLength or NumPhyEvents at runtime.
 */
#ifndef MPI2_SASPHY3_PHY_EVENT_MAX
#define MPI2_SASPHY3_PHY_EVENT_MAX      (1)
#endif

typedef struct _MPI2_CONFIG_PAGE_SAS_PHY_3
```



```
{
    MPI2_CONFIG_EXTENDED_PAGE_HEADER    Header;                     /* 0x00 */
    U32                                 Reserved1;                  /* 0x08 */
    U8                                  NumPhyEvents;               /* 0x0C */
    U8                                  Reserved2;                  /* 0x0D */
    U16                                 Reserved3;                  /* 0x0E */
    MPI2_SASPHY3_PHY_EVENT_CONFIG       PhyEventConfig[MPI2_SASPHY3_PHY_EVENT_MA
} MPI2_CONFIG_PAGE_SAS_PHY_3, MPI2_POINTER PTR_MPI2_CONFIG_PAGE_SAS_PHY_3,
  Mpi2SasPhyPage3_t, MPI2_POINTER pMpi2SasPhyPage3_t;

#define MPI2_SASPHY3_PAGEVERSION            (0x00)


/* SAS PHY Page 4 */

typedef struct _MPI2_CONFIG_PAGE_SAS_PHY_4
{
    MPI2_CONFIG_EXTENDED_PAGE_HEADER    Header;                     /* 0x00 */
    U16                                 Reserved1;                  /* 0x08 */
    U8                                  Reserved2;                  /* 0x0A */
    U8                                  Flags;                      /* 0x0B */
    U8                                  InitialFrame[28];           /* 0x0C */
} MPI2_CONFIG_PAGE_SAS_PHY_4, MPI2_POINTER PTR_MPI2_CONFIG_PAGE_SAS_PHY_4,
  Mpi2SasPhyPage4_t, MPI2_POINTER pMpi2SasPhyPage4_t;

#define MPI2_SASPHY4_PAGEVERSION            (0x00)

/* values for the Flags field */
#define MPI2_SASPHY4_FLAGS_FRAME_VALID       (0x02)
#define MPI2_SASPHY4_FLAGS_SATA_FRAME        (0x01)




#endif /* ! codereview */
/*****************************************************************************
*   SAS Port Config Pages
*****************************************************************************/

/* SAS Port Page 0 */

typedef struct _MPI2_CONFIG_PAGE_SAS_PORT_0
{
    MPI2_CONFIG_EXTENDED_PAGE_HEADER    Header;                     /* 0x00 */
    U8                                  PortNumber;                 /* 0x08 */
    U8                                  PhysicalPort;               /* 0x09 */
    U8                                  PortWidth;                  /* 0x0A */
    U8                                  PhysicalPortWidth;          /* 0x0B */
    U8                                  ZoneGroup;                  /* 0x0C */
    U8                                  Reserved1;                  /* 0x0D */
    U16                                 Reserved2;                  /* 0x0E */
    U64                                 SASAddress;                 /* 0x10 */
    U32                                 DeviceInfo;                 /* 0x18 */
    U32                                 Reserved3;                  /* 0x1C */
    U32                                 Reserved4;                  /* 0x20 */
} MPI2_CONFIG_PAGE_SAS_PORT_0, MPI2_POINTER PTR_MPI2_CONFIG_PAGE_SAS_PORT_0,
  Mpi2SasPortPage0_t, MPI2_POINTER pMpi2SasPortPage0_t;

#define MPI2_SASPORT0_PAGEVERSION           (0x00)

/* see mpi2_sas.h for values for SAS Port Page 0 DeviceInfo values */


/*****************************************************************************
*   SAS Enclosure Config Pages
*****************************************************************************/
```

```
/* SAS Enclosure Page 0 */

typedef struct _MPI2_CONFIG_PAGE_SAS_ENCLOSURE_0
{
    MPI2_CONFIG_EXTENDED_PAGE_HEADER    Header;                 /* 0x00 */
    U32                                 Reserved1;              /* 0x08 */
    U64                                 EnclosureLogicalID;     /* 0x0C */
    U16                                 Flags;                  /* 0x14 */
    U16                                 EnclosureHandle;        /* 0x16 */
    U16                                 NumSlots;               /* 0x18 */
    U16                                 StartSlot;              /* 0x1A */
    U16                                 Reserved2;              /* 0x1C */
    U16                                 SEPDevHandle;           /* 0x1E */
    U32                                 Reserved3;              /* 0x20 */
    U32                                 Reserved4;              /* 0x24 */
} MPI2_CONFIG_PAGE_SAS_ENCLOSURE_0,
  MPI2_POINTER PTR_MPI2_CONFIG_PAGE_SAS_ENCLOSURE_0,
  Mpi2SasEnclosurePage0_t, MPI2_POINTER pMpi2SasEnclosurePage0_t;

#define MPI2_SASENCLOSURE0_PAGEVERSION      (0x03)

/* values for SAS Enclosure Page 0 Flags field */
#define MPI2_SAS_ENCLS0_FLAGS_MNG_MASK              (0x000F)
#define MPI2_SAS_ENCLS0_FLAGS_MNG_UNKNOWN           (0x0000)
#define MPI2_SAS_ENCLS0_FLAGS_MNG_IOC_SES           (0x0001)
#define MPI2_SAS_ENCLS0_FLAGS_MNG_IOC_SGPIO         (0x0002)
#define MPI2_SAS_ENCLS0_FLAGS_MNG_EXP_SGPIO         (0x0003)
#define MPI2_SAS_ENCLS0_FLAGS_MNG_SES_ENCLOSURE     (0x0004)
#define MPI2_SAS_ENCLS0_FLAGS_MNG_IOC_GPIO          (0x0005)


/*****************************************************************************
*   Log Config Page
*****************************************************************************/

/* Log Page 0 */

/*
 * Host code (drivers, BIOS, utilities, etc.) should leave this define set to
 * one and check Header.ExtPageLength or NumPhys at runtime.
 */
#ifndef MPI2_LOG_0_NUM_LOG_ENTRIES
#define MPI2_LOG_0_NUM_LOG_ENTRIES          (1)
#endif

#define MPI2_LOG_0_LOG_DATA_LENGTH          (0x1C)

typedef struct _MPI2_LOG_0_ENTRY
{
    U64         TimeStamp;                          /* 0x00 */
    U32         Reserved1;                          /* 0x08 */
    U16         LogSequence;                        /* 0x0C */
    U16         LogEntryQualifier;                  /* 0x0E */
    U8          VP_ID;                              /* 0x10 */
    U8          VF_ID;                              /* 0x11 */
    U16         Reserved2;                          /* 0x12 */
    U8          LogData[MPI2_LOG_0_LOG_DATA_LENGTH];/* 0x14 */
} MPI2_LOG_0_ENTRY, MPI2_POINTER PTR_MPI2_LOG_0_ENTRY,
  Mpi2Log0Entry_t, MPI2_POINTER pMpi2Log0Entry_t;

/* values for Log Page 0 LogEntry LogEntryQualifier field */
#define MPI2_LOG_0_ENTRY_QUAL_ENTRY_UNUSED          (0x0000)
#define MPI2_LOG_0_ENTRY_QUAL_POWER_ON_RESET        (0x0001)
#define MPI2_LOG_0_ENTRY_QUAL_TIMESTAMP_UPDATE      (0x0002)
#define MPI2_LOG_0_ENTRY_QUAL_MIN_IMPLEMENT_SPEC    (0x8000)
```



```
#define MPI2_LOG_0_ENTRY_QUAL_MAX_IMPLEMENT_SPEC    (0xFFFF)

typedef struct _MPI2_CONFIG_PAGE_LOG_0
{
    MPI2_CONFIG_EXTENDED_PAGE_HEADER    Header;                 /* 0x00 */
    U32                                 Reserved1;              /* 0x08 */
    U32                                 Reserved2;              /* 0x0C */
    U16                                 NumLogEntries;          /* 0x10 */
    U16                                 Reserved3;              /* 0x12 */
    MPI2_LOG_0_ENTRY                    LogEntry[MPI2_LOG_0_NUM_LOG_ENTRIES]; /*
} MPI2_CONFIG_PAGE_LOG_0, MPI2_POINTER PTR_MPI2_CONFIG_PAGE_LOG_0,
  Mpi2LogPage0_t, MPI2_POINTER pMpi2LogPage0_t;

#define MPI2_LOG_0_PAGEVERSION              (0x02)


/*****************************************************************************
*   RAID Config Page
*****************************************************************************/

/* RAID Page 0 */

/*
 * Host code (drivers, BIOS, utilities, etc.) should leave this define set to
 * one and check Header.ExtPageLength or NumPhys at runtime.
 */
#ifndef MPI2_RAIDCONFIG0_MAX_ELEMENTS
#define MPI2_RAIDCONFIG0_MAX_ELEMENTS       (1)
#endif

typedef struct _MPI2_RAIDCONFIG0_CONFIG_ELEMENT
{
    U16                     ElementFlags;               /* 0x00 */
    U16                     VolDevHandle;               /* 0x02 */
    U8                      HotSparePool;               /* 0x04 */
    U8                      PhysDiskNum;                /* 0x05 */
    U16                     PhysDiskDevHandle;          /* 0x06 */
} MPI2_RAIDCONFIG0_CONFIG_ELEMENT,
  MPI2_POINTER PTR_MPI2_RAIDCONFIG0_CONFIG_ELEMENT,
  Mpi2RaidConfig0ConfigElement_t, MPI2_POINTER pMpi2RaidConfig0ConfigElement_t;

/* values for the ElementFlags field */
#define MPI2_RAIDCONFIG0_EFLAGS_MASK_ELEMENT_TYPE       (0x000F)
#define MPI2_RAIDCONFIG0_EFLAGS_VOLUME_ELEMENT          (0x0000)
#define MPI2_RAIDCONFIG0_EFLAGS_VOL_PHYS_DISK_ELEMENT   (0x0001)
#define MPI2_RAIDCONFIG0_EFLAGS_HOT_SPARE_ELEMENT       (0x0002)
#define MPI2_RAIDCONFIG0_EFLAGS_OCE_ELEMENT             (0x0003)


typedef struct _MPI2_CONFIG_PAGE_RAID_CONFIGURATION_0
{
    MPI2_CONFIG_EXTENDED_PAGE_HEADER    Header;                 /* 0x00 */
    U8                                  NumHotSpares;           /* 0x08 */
    U8                                  NumPhysDisks;           /* 0x09 */
    U8                                  NumVolumes;             /* 0x0A */
    U8                                  ConfigNum;              /* 0x0B */
    U32                                 Flags;                  /* 0x0C */
    U8                                  ConfigGUID[24];         /* 0x10 */
    U32                                 Reserved1;              /* 0x28 */
    U8                                  NumElements;            /* 0x2C */
    U8                                  Reserved2;              /* 0x2D */
    U16                                 Reserved3;              /* 0x2E */
    MPI2_RAIDCONFIG0_CONFIG_ELEMENT     ConfigElement[MPI2_RAIDCONFIG0_MAX_ELEME
} MPI2_CONFIG_PAGE_RAID_CONFIGURATION_0,
  MPI2_POINTER PTR_MPI2_CONFIG_PAGE_RAID_CONFIGURATION_0,
  Mpi2RaidConfigurationPage0_t, MPI2_POINTER pMpi2RaidConfigurationPage0_t;
```

```
#define MPI2_RAIDCONFIG0_PAGEVERSION            (0x00)

/* values for RAID Configuration Page 0 Flags field */
#define MPI2_RAIDCONFIG0_FLAG_FOREIGN_CONFIG        (0x00000001)


/*****************************************************************************
*   Driver Persistent Mapping Config Pages
*****************************************************************************/

/* Driver Persistent Mapping Page 0 */

typedef struct _MPI2_CONFIG_PAGE_DRIVER_MAP0_ENTRY
{
    U64                                 PhysicalIdentifier;         /* 0x00 */
    U16                                 MappingInformation;         /* 0x08 */
    U16                                 DeviceIndex;                /* 0x0A */
    U32                                 PhysicalBitsMapping;        /* 0x0C */
    U32                                 Reserved1;                  /* 0x10 */
} MPI2_CONFIG_PAGE_DRIVER_MAP0_ENTRY,
  MPI2_POINTER PTR_MPI2_CONFIG_PAGE_DRIVER_MAP0_ENTRY,
  Mpi2DriverMap0Entry_t, MPI2_POINTER pMpi2DriverMap0Entry_t;

typedef struct _MPI2_CONFIG_PAGE_DRIVER_MAPPING_0
{
    MPI2_CONFIG_EXTENDED_PAGE_HEADER    Header;                     /* 0x00 */
    MPI2_CONFIG_PAGE_DRIVER_MAP0_ENTRY  Entry;                      /* 0x08 */
} MPI2_CONFIG_PAGE_DRIVER_MAPPING_0,
  MPI2_POINTER PTR_MPI2_CONFIG_PAGE_DRIVER_MAPPING_0,
  Mpi2DriverMappingPage0_t, MPI2_POINTER pMpi2DriverMappingPage0_t;

#define MPI2_DRIVERMAPPING0_PAGEVERSION         (0x00)

/* values for Driver Persistent Mapping Page 0 MappingInformation field */
#define MPI2_DRVMAP0_MAPINFO_SLOT_MASK              (0x07F0)
#define MPI2_DRVMAP0_MAPINFO_SLOT_SHIFT             (4)
#define MPI2_DRVMAP0_MAPINFO_MISSING_MASK           (0x000F)


/*****************************************************************************
*   Ethernet Config Pages
*****************************************************************************/

/* Ethernet Page 0 */

/* IP address (union of IPv4 and IPv6) */
typedef union _MPI2_ETHERNET_IP_ADDR
{
    U32     IPv4Addr;
    U32     IPv6Addr[4];
} MPI2_ETHERNET_IP_ADDR, MPI2_POINTER PTR_MPI2_ETHERNET_IP_ADDR,
  Mpi2EthernetIpAddr_t, MPI2_POINTER pMpi2EthernetIpAddr_t;

#define MPI2_ETHERNET_HOST_NAME_LENGTH          (32)

typedef struct _MPI2_CONFIG_PAGE_ETHERNET_0
{
    MPI2_CONFIG_EXTENDED_PAGE_HEADER    Header;                 /* 0x00 */
    U8                                  NumInterfaces;          /* 0x08 */
    U8                                  Reserved0;              /* 0x09 */
    U16                                 Reserved1;              /* 0x0A */
    U32                                 Status;                 /* 0x0C */
    U8                                  MediaState;             /* 0x10 */
    U8                                  Reserved2;              /* 0x11 */
    U16                                 Reserved3;              /* 0x12 */
```



```
    U8                                  MacAddress[6];          /* 0x14 */
    U8                                  Reserved4;              /* 0x1A */
    U8                                  Reserved5;              /* 0x1B */
    MPI2_ETHERNET_IP_ADDR               IpAddress;              /* 0x1C */
    MPI2_ETHERNET_IP_ADDR               SubnetMask;             /* 0x2C */
    MPI2_ETHERNET_IP_ADDR               GatewayIpAddress;       /* 0x3C */
    MPI2_ETHERNET_IP_ADDR               DNS1IpAddress;          /* 0x4C */
    MPI2_ETHERNET_IP_ADDR               DNS2IpAddress;          /* 0x5C */
    MPI2_ETHERNET_IP_ADDR               DhcpIpAddress;          /* 0x6C */
    U8                                  HostName[MPI2_ETHERNET_HOST_NAME_LENGTH]
} MPI2_CONFIG_PAGE_ETHERNET_0, MPI2_POINTER PTR_MPI2_CONFIG_PAGE_ETHERNET_0,
  Mpi2EthernetPage0_t, MPI2_POINTER pMpi2EthernetPage0_t;

#define MPI2_ETHERNETPAGE0_PAGEVERSION   (0x00)

/* values for Ethernet Page 0 Status field */
#define MPI2_ETHPG0_STATUS_IPV6_CAPABLE             (0x80000000)
#define MPI2_ETHPG0_STATUS_IPV4_CAPABLE             (0x40000000)
#define MPI2_ETHPG0_STATUS_CONSOLE_CONNECTED        (0x20000000)
#define MPI2_ETHPG0_STATUS_DEFAULT_IF               (0x00000100)
#define MPI2_ETHPG0_STATUS_FW_DWNLD_ENABLED         (0x00000080)
#define MPI2_ETHPG0_STATUS_TELNET_ENABLED           (0x00000040)
#define MPI2_ETHPG0_STATUS_SSH2_ENABLED             (0x00000020)
#define MPI2_ETHPG0_STATUS_DHCP_CLIENT_ENABLED      (0x00000010)
#define MPI2_ETHPG0_STATUS_IPV6_ENABLED             (0x00000008)
#define MPI2_ETHPG0_STATUS_IPV4_ENABLED             (0x00000004)
#define MPI2_ETHPG0_STATUS_IPV6_ADDRESSES           (0x00000002)
#define MPI2_ETHPG0_STATUS_ETH_IF_ENABLED           (0x00000001)

/* values for Ethernet Page 0 MediaState field */
#define MPI2_ETHPG0_MS_DUPLEX_MASK                  (0x80)
#define MPI2_ETHPG0_MS_HALF_DUPLEX                  (0x00)
#define MPI2_ETHPG0_MS_FULL_DUPLEX                  (0x80)

#define MPI2_ETHPG0_MS_CONNECT_SPEED_MASK           (0x07)
#define MPI2_ETHPG0_MS_NOT_CONNECTED                (0x00)
#define MPI2_ETHPG0_MS_10MBIT                       (0x01)
#define MPI2_ETHPG0_MS_100MBIT                      (0x02)
#define MPI2_ETHPG0_MS_1GBIT                        (0x03)


/* Ethernet Page 1 */

typedef struct _MPI2_CONFIG_PAGE_ETHERNET_1
{
    MPI2_CONFIG_EXTENDED_PAGE_HEADER    Header;                 /* 0x00 */
    U32                                 Reserved0;              /* 0x08 */
    U32                                 Flags;                  /* 0x0C */
    U8                                  MediaState;             /* 0x10 */
    U8                                  Reserved1;              /* 0x11 */
    U16                                 Reserved2;              /* 0x12 */
    U8                                  MacAddress[6];          /* 0x14 */
    U8                                  Reserved3;              /* 0x1A */
    U8                                  Reserved4;              /* 0x1B */
    MPI2_ETHERNET_IP_ADDR               StaticIpAddress;        /* 0x1C */
    MPI2_ETHERNET_IP_ADDR               StaticSubnetMask;       /* 0x2C */
    MPI2_ETHERNET_IP_ADDR               StaticGatewayIpAddress; /* 0x3C */
    MPI2_ETHERNET_IP_ADDR               StaticDNS1IpAddress;    /* 0x4C */
    MPI2_ETHERNET_IP_ADDR               StaticDNS2IpAddress;    /* 0x5C */
    U32                                 Reserved5;              /* 0x6C */
    U32                                 Reserved6;              /* 0x70 */
    U32                                 Reserved7;              /* 0x74 */
    U32                                 Reserved8;              /* 0x78 */
    U8                                  HostName[MPI2_ETHERNET_HOST_NAME_LENGTH]
} MPI2_CONFIG_PAGE_ETHERNET_1, MPI2_POINTER PTR_MPI2_CONFIG_PAGE_ETHERNET_1,
  Mpi2EthernetPage1_t, MPI2_POINTER pMpi2EthernetPage1_t;
```

```
#define MPI2_ETHERNETPAGE1_PAGEVERSION   (0x00)

/* values for Ethernet Page 1 Flags field */
#define MPI2_ETHPG1_FLAG_SET_DEFAULT_IF             (0x00000100)
#define MPI2_ETHPG1_FLAG_ENABLE_FW_DOWNLOAD         (0x00000080)
#define MPI2_ETHPG1_FLAG_ENABLE_TELNET              (0x00000040)
#define MPI2_ETHPG1_FLAG_ENABLE_SSH2                (0x00000020)
#define MPI2_ETHPG1_FLAG_ENABLE_DHCP_CLIENT         (0x00000010)
#define MPI2_ETHPG1_FLAG_ENABLE_IPV6                (0x00000008)
#define MPI2_ETHPG1_FLAG_ENABLE_IPV4                (0x00000004)
#define MPI2_ETHPG1_FLAG_USE_IPV6_ADDRESSES         (0x00000002)
#define MPI2_ETHPG1_FLAG_ENABLE_ETH_IF              (0x00000001)

/* values for Ethernet Page 1 MediaState field */
#define MPI2_ETHPG1_MS_DUPLEX_MASK                  (0x80)
#define MPI2_ETHPG1_MS_HALF_DUPLEX                  (0x00)
#define MPI2_ETHPG1_MS_FULL_DUPLEX                  (0x80)

#define MPI2_ETHPG1_MS_DATA_RATE_MASK               (0x07)
#define MPI2_ETHPG1_MS_DATA_RATE_AUTO               (0x00)
#define MPI2_ETHPG1_MS_DATA_RATE_10MBIT             (0x01)
#define MPI2_ETHPG1_MS_DATA_RATE_100MBIT            (0x02)
#define MPI2_ETHPG1_MS_DATA_RATE_1GBIT              (0x03)


/*****************************************************************************
*   Extended Manufacturing Config Pages
*****************************************************************************/

/*
 * Generic structure to use for product-specific extended manufacturing pages
 * (currently Extended Manufacturing Page 40 through Extended Manufacturing
 * Page 60).
 */

typedef struct _MPI2_CONFIG_PAGE_EXT_MAN_PS
{
    MPI2_CONFIG_EXTENDED_PAGE_HEADER    Header;                 /* 0x00 */
    U32                                 ProductSpecificInfo;    /* 0x08 */
} MPI2_CONFIG_PAGE_EXT_MAN_PS,
  MPI2_POINTER PTR_MPI2_CONFIG_PAGE_EXT_MAN_PS,
  Mpi2ExtManufacturingPagePS_t, MPI2_POINTER pMpi2ExtManufacturingPagePS_t;

/* PageVersion should be provided by product-specific code */

#endif /* ! codereview */
#endif
```

```
**********************************************************
   29921 Thu Jun 12 17:42:18 2014
new/usr/src/uts/common/sys/scsi/adapters/mpt_sas3/mpi/mpi2_init.h
Initial modifications using the code changes present between
the LSI source code for FreeBSD drivers. Specifically the changes
between from mpslsi-source-17.00.00.00 -> mpslsi-source-03.00.00.00.
This mainly involves using a different scatter/gather element in
frame setup.
**********************************************************
/*
 * 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) 2000-2012 LSI Corporation.
 * Copyright (c) 2000 to 2009, LSI Corporation.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms of all code within
 * this file that is exclusively owned by LSI, with or without
 * modification, is permitted provided that, in addition to the CDDL 1.0
 * License requirements, the following conditions are met:
 *
 *    Neither the name of the author nor the names of its contributors may be
 *    used to endorse or promote products derived from this software without
 *    specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS 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
 * COPYRIGHT OWNER 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.
 */

/*
 *          Name:  mpi2_init.h
 *         Title:  MPI SCSI initiator mode messages and structures
 *  Creation Date:  June 23, 2006
 *
 *  mpi2_init.h Version:  02.00.xx
 *
 *  NOTE: Names (typedefs, defines, etc.) beginning with an MPI25 or Mpi25
```



```
 *        prefix are for use only on MPI v2.5 products, and must not be used
 *        with MPI v2.0 products. Unless otherwise noted, names beginning with
 *        MPI2 or Mpi2 are for use with both MPI v2.0 and MPI v2.5 products.
 *    mpi2_init.h Version:  02.00.07
 *
 *  Version History
 *  ---------------
 *
 *  Date      Version   Description
 *  --------  --------  ------------------------------------------------------
 *  04-30-07  02.00.00  Corresponds to Fusion-MPT MPI Specification Rev A.
 *  10-31-07  02.00.01  Fixed name for pMpi2SCSITaskManagementRequest_t.
 *  12-18-07  02.00.02  Modified Task Management Target Reset Method defines.
 *  02-29-08  02.00.03  Added Query Task Set and Query Unit Attention.
 *  03-03-08  02.00.04  Fixed name of struct _MPI2_SCSI_TASK_MANAGE_REPLY.
 *  05-21-08  02.00.05  Fixed typo in name of Mpi2SepRequest_t.
 *  10-02-08  02.00.06  Removed Untagged and No Disconnect values from SCSI IO
 *                      Control field Task Attribute flags.
 *                      Moved LUN field defines to mpi2.h becasue they are
 *                      common to many structures.
 *  05-06-09  02.00.07  Changed task management type of Query Unit Attention to
 *                      Query Asynchronous Event.
 *                      Defined two new bits in the SlotStatus field of the SCSI
 *                      Enclosure Processor Request and Reply.
 *  10-28-09  02.00.08  Added defines for decoding the ResponseInfo bytes for
 *                      both SCSI IO Error Reply and SCSI Task Management Reply.
 *                      Added ResponseInfo field to MPI2_SCSI_TASK_MANAGE_REPLY.
 *                      Added MPI2_SCSITASKMGMT_RSP_TM_OVERLAPPED_TAG define.
 *  02-10-10  02.00.09  Removed unused structure that had "#if 0" around it.
 *  05-12-10  02.00.10  Added optional vendor-unique region to SCSI IO Request.
 *  11-10-10  02.00.11  Added MPI2_SCSIIO_NUM_SGLOFFSETS define.
#endif /* ! codereview */
 *  --------------------------------------------------------------------------
 */

#ifndef MPI2_INIT_H
#define MPI2_INIT_H

/*****************************************************************************
*
*               SCSI Initiator Messages
*
*****************************************************************************/

/****************************************************************************
*  SCSI IO messages and associated structures
****************************************************************************/

typedef struct
{
    U8                      CDB[20];                /* 0x00 */
    U32                     PrimaryReferenceTag;    /* 0x14 */
    U16                     PrimaryApplicationTag;  /* 0x18 */
    U16                     PrimaryApplicationTagMask;  /* 0x1A */
    U32                     TransferLength;         /* 0x1C */
} MPI2_SCSI_IO_CDB_EEDP32, MPI2_POINTER PTR_MPI2_SCSI_IO_CDB_EEDP32,
  Mpi2ScsiIoCdbEedp32_t, MPI2_POINTER pMpi2ScsiIoCdbEedp32_t;

/* TBD: I don't think this is needed for MPI2/Gen2 */
#if 0
typedef struct
{
    U8                      CDB[16];                /* 0x00 */
    U32                     DataLength;             /* 0x10 */
    U32                     PrimaryReferenceTag;    /* 0x14 */
    U16                     PrimaryApplicationTag;  /* 0x18 */
```

```
 83     U16                     PrimaryApplicationTagMask;  /* 0x1A */
 84     U32                     TransferLength;             /* 0x1C */
 85 } MPI2_SCSI_IO32_CDB_EEDP16, MPI2_POINTER PTR_MPI2_SCSI_IO32_CDB_EEDP16,
 86   Mpi2ScsiIo32CdbEedp16_t, MPI2_POINTER pMpi2ScsiIo32CdbEedp16_t;
 87 #endif

113 typedef union
114 {
115     U8                      CDB32[32];
116     MPI2_SCSI_IO_CDB_EEDP32 EEDP32;
117     MPI2_SGE_SIMPLE_UNION   SGE;
118 } MPI2_SCSI_IO_CDB_UNION, MPI2_POINTER PTR_MPI2_SCSI_IO_CDB_UNION,
119   Mpi2ScsiIoCdb_t, MPI2_POINTER pMpi2ScsiIoCdb_t;

121 /* MPI v2.0 SCSI IO Request Message */
 97 /* SCSI IO Request Message */
122 typedef struct _MPI2_SCSI_IO_REQUEST
123 {
124     U16                     DevHandle;                  /* 0x00 */
125     U8                      ChainOffset;                /* 0x02 */
126     U8                      Function;                   /* 0x03 */
127     U16                     Reserved1;                  /* 0x04 */
128     U8                      Reserved2;                  /* 0x06 */
129     U8                      MsgFlags;                   /* 0x07 */
130     U8                      VP_ID;                      /* 0x08 */
131     U8                      VF_ID;                      /* 0x09 */
132     U16                     Reserved3;                  /* 0x0A */
133     U32                     SenseBufferLowAddress;      /* 0x0C */
134     U16                     SGLFlags;                   /* 0x10 */
135     U8                      SenseBufferLength;          /* 0x12 */
136     U8                      Reserved4;                  /* 0x13 */
137     U8                      SGLOffset0;                 /* 0x14 */
138     U8                      SGLOffset1;                 /* 0x15 */
139     U8                      SGLOffset2;                 /* 0x16 */
140     U8                      SGLOffset3;                 /* 0x17 */
141     U32                     SkipCount;                  /* 0x18 */
142     U32                     DataLength;                 /* 0x1C */
143     U32                     BidirectionalDataLength;    /* 0x20 */
144     U16                     IoFlags;                    /* 0x24 */
145     U16                     EEDPFlags;                  /* 0x26 */
146     U32                     EEDPBlockSize;              /* 0x28 */
147     U32                     SecondaryReferenceTag;      /* 0x2C */
148     U16                     SecondaryApplicationTag;    /* 0x30 */
149     U16                     ApplicationTagTranslationMask;  /* 0x32 */
150     U8                      LUN[8];                     /* 0x34 */
151     U32                     Control;                    /* 0x3C */
152     MPI2_SCSI_IO_CDB_UNION  CDB;                        /* 0x40 */
153     MPI2_SGE_IO_UNION       SGL;                        /* 0x60 */
154 } MPI2_SCSI_IO_REQUEST, MPI2_POINTER PTR_MPI2_SCSI_IO_REQUEST,
155   Mpi2SCSIIORequest_t, MPI2_POINTER pMpi2SCSIIORequest_t;

157 /* SCSI IO MsgFlags bits */

159 /* MsgFlags for SenseBufferAddressSpace */
160 #define MPI2_SCSIIO_MSGFLAGS_MASK_SENSE_ADDR        (0x0C)
161 #define MPI2_SCSIIO_MSGFLAGS_SYSTEM_SENSE_ADDR      (0x00)
162 #define MPI2_SCSIIO_MSGFLAGS_IOCDDR_SENSE_ADDR      (0x04)
163 #define MPI2_SCSIIO_MSGFLAGS_IOCPLB_SENSE_ADDR      (0x08)
164 #define MPI2_SCSIIO_MSGFLAGS_IOCPLBNTA_SENSE_ADDR   (0x0C)

166 /* SCSI IO SGLFlags bits */

168 /* base values for Data Location Address Space */
169 #define MPI2_SCSIIO_SGLFLAGS_ADDR_MASK              (0x0C)
170 #define MPI2_SCSIIO_SGLFLAGS_SYSTEM_ADDR            (0x00)
171 #define MPI2_SCSIIO_SGLFLAGS_IOCDDR_ADDR            (0x04)
```



```
172 #define MPI2_SCSIIO_SGLFLAGS_IOCPLB_ADDR            (0x08)
173 #define MPI2_SCSIIO_SGLFLAGS_IOCPLBNTA_ADDR         (0x0C)

175 /* base values for Type */
176 #define MPI2_SCSIIO_SGLFLAGS_TYPE_MASK              (0x03)
177 #define MPI2_SCSIIO_SGLFLAGS_TYPE_MPI               (0x00)
178 #define MPI2_SCSIIO_SGLFLAGS_TYPE_IEEE32            (0x01)
179 #define MPI2_SCSIIO_SGLFLAGS_TYPE_IEEE64            (0x02)

181 /* shift values for each sub-field */
182 #define MPI2_SCSIIO_SGLFLAGS_SGL3_SHIFT             (12)
183 #define MPI2_SCSIIO_SGLFLAGS_SGL2_SHIFT             (8)
184 #define MPI2_SCSIIO_SGLFLAGS_SGL1_SHIFT             (4)
185 #define MPI2_SCSIIO_SGLFLAGS_SGL0_SHIFT             (0)

187 /* number of SGLOffset fields */
188 #define MPI2_SCSIIO_NUM_SGLOFFSETS                  (4)

190 #endif /* ! codereview */
191 /* SCSI IO IoFlags bits */

193 /* Large CDB Address Space */
194 #define MPI2_SCSIIO_CDB_ADDR_MASK                   (0x6000)
195 #define MPI2_SCSIIO_CDB_ADDR_SYSTEM                 (0x0000)
196 #define MPI2_SCSIIO_CDB_ADDR_IOCDDR                 (0x2000)
197 #define MPI2_SCSIIO_CDB_ADDR_IOCPLB                 (0x4000)
198 #define MPI2_SCSIIO_CDB_ADDR_IOCPLBNTA              (0x6000)

200 #define MPI2_SCSIIO_IOFLAGS_LARGE_CDB               (0x1000)
201 #define MPI2_SCSIIO_IOFLAGS_BIDIRECTIONAL           (0x0800)
202 #define MPI2_SCSIIO_IOFLAGS_MULTICAST               (0x0400)
203 #define MPI2_SCSIIO_IOFLAGS_CMD_DETERMINES_DATA_DIR (0x0200)
204 #define MPI2_SCSIIO_IOFLAGS_CDBLENGTH_MASK          (0x01FF)

206 /* SCSI IO EEDPFlags bits */

208 #define MPI2_SCSIIO_EEDPFLAGS_INC_PRI_REFTAG        (0x8000)
209 #define MPI2_SCSIIO_EEDPFLAGS_INC_SEC_REFTAG        (0x4000)
210 #define MPI2_SCSIIO_EEDPFLAGS_INC_PRI_APPTAG        (0x2000)
211 #define MPI2_SCSIIO_EEDPFLAGS_INC_SEC_APPTAG        (0x1000)

213 #define MPI2_SCSIIO_EEDPFLAGS_CHECK_REFTAG          (0x0400)
214 #define MPI2_SCSIIO_EEDPFLAGS_CHECK_APPTAG          (0x0200)
215 #define MPI2_SCSIIO_EEDPFLAGS_CHECK_GUARD           (0x0100)

217 #define MPI2_SCSIIO_EEDPFLAGS_PASSTHRU_REFTAG       (0x0008)

219 #define MPI2_SCSIIO_EEDPFLAGS_MASK_OP               (0x0007)
220 #define MPI2_SCSIIO_EEDPFLAGS_NOOP_OP               (0x0000)
221 #define MPI2_SCSIIO_EEDPFLAGS_CHECK_OP              (0x0001)
222 #define MPI2_SCSIIO_EEDPFLAGS_STRIP_OP              (0x0002)
223 #define MPI2_SCSIIO_EEDPFLAGS_CHECK_REMOVE_OP       (0x0003)
224 #define MPI2_SCSIIO_EEDPFLAGS_INSERT_OP             (0x0004)
225 #define MPI2_SCSIIO_EEDPFLAGS_REPLACE_OP            (0x0006)
226 #define MPI2_SCSIIO_EEDPFLAGS_CHECK_REGEN_OP        (0x0007)

228 /* SCSI IO LUN fields: use MPI2_LUN_ from mpi2.h */

230 /* SCSI IO Control bits */
231 #define MPI2_SCSIIO_CONTROL_ADDCDBLEN_MASK      (0xFC000000)
232 #define MPI2_SCSIIO_CONTROL_ADDCDBLEN_SHIFT     (26)

234 #define MPI2_SCSIIO_CONTROL_DATADIRECTION_MASK  (0x03000000)
235 #define MPI2_SCSIIO_CONTROL_NODATATRANSFER      (0x00000000)
236 #define MPI2_SCSIIO_CONTROL_WRITE               (0x01000000)
237 #define MPI2_SCSIIO_CONTROL_READ                (0x02000000)
```

```
#define MPI2_SCSIIO_CONTROL_BIDIRECTIONAL       (0x03000000)

#define MPI2_SCSIIO_CONTROL_TASKPRI_MASK        (0x00007800)
#define MPI2_SCSIIO_CONTROL_TASKPRI_SHIFT       (11)

#define MPI2_SCSIIO_CONTROL_TASKATTRIBUTE_MASK  (0x00000700)
#define MPI2_SCSIIO_CONTROL_SIMPLEQ             (0x00000000)
#define MPI2_SCSIIO_CONTROL_HEADOFQ             (0x00000100)
#define MPI2_SCSIIO_CONTROL_ORDEREDQ            (0x00000200)
#define MPI2_SCSIIO_CONTROL_ACAQ                (0x00000400)

#define MPI2_SCSIIO_CONTROL_TLR_MASK            (0x000000C0)
#define MPI2_SCSIIO_CONTROL_NO_TLR              (0x00000000)
#define MPI2_SCSIIO_CONTROL_TLR_ON              (0x00000040)
#define MPI2_SCSIIO_CONTROL_TLR_OFF             (0x00000080)


/* MPI v2.5 CDB field */
typedef union _MPI25_SCSI_IO_CDB_UNION
{
    U8                      CDB32[32];
    MPI2_SCSI_IO_CDB_EEDP32 EEDP32;
    MPI2_IEEE_SGE_SIMPLE64  SGE;
} MPI25_SCSI_IO_CDB_UNION, MPI2_POINTER PTR_MPI25_SCSI_IO_CDB_UNION,
  Mpi25ScsiIoCdb_t, MPI2_POINTER pMpi25ScsiIoCdb_t;

/* MPI v2.5 SCSI IO Request Message */
typedef struct _MPI25_SCSI_IO_REQUEST
{
    U16                     DevHandle;                  /* 0x00 */
    U8                      ChainOffset;                /* 0x02 */
    U8                      Function;                   /* 0x03 */
    U16                     Reserved1;                  /* 0x04 */
    U8                      Reserved2;                  /* 0x06 */
    U8                      MsgFlags;                   /* 0x07 */
    U8                      VP_ID;                      /* 0x08 */
    U8                      VF_ID;                      /* 0x09 */
    U16                     Reserved3;                  /* 0x0A */
    U32                     SenseBufferLowAddress;      /* 0x0C */
    U8                      DMAFlags;                   /* 0x10 */
    U8                      Reserved5;                  /* 0x11 */
    U8                      SenseBufferLength;          /* 0x12 */
    U8                      Reserved4;                  /* 0x13 */
    U8                      SGLOffset0;                 /* 0x14 */
    U8                      SGLOffset1;                 /* 0x15 */
    U8                      SGLOffset2;                 /* 0x16 */
    U8                      SGLOffset3;                 /* 0x17 */
    U32                     SkipCount;                  /* 0x18 */
    U32                     DataLength;                 /* 0x1C */
    U32                     BidirectionalDataLength;    /* 0x20 */
    U16                     IoFlags;                    /* 0x24 */
    U16                     EEDPFlags;                  /* 0x26 */
    U16                     EEDPBlockSize;              /* 0x28 */
    U16                     Reserved6;                  /* 0x2A */
    U32                     SecondaryReferenceTag;      /* 0x2C */
    U16                     SecondaryApplicationTag;    /* 0x30 */
    U16                     ApplicationTagTranslationMask; /* 0x32 */
    U8                      LUN[8];                     /* 0x34 */
    U32                     Control;                    /* 0x3C */
    MPI25_SCSI_IO_CDB_UNION CDB;                        /* 0x40 */

#ifdef MPI25_SCSI_IO_VENDOR_UNIQUE_REGION /* typically this is left undefined */
    MPI25_SCSI_IO_VENDOR_UNIQUE VendorRegion;
#endif

    MPI25_SGE_IO_UNION      SGL;                        /* 0x60 */
```



```
} MPI25_SCSI_IO_REQUEST, MPI2_POINTER PTR_MPI25_SCSI_IO_REQUEST,
  Mpi25SCSIIORequest_t, MPI2_POINTER pMpi25SCSIIORequest_t;

/* use MPI2_SCSIIO_MSGFLAGS_ defines for the MsgFlags field */

/* Defines for the DMAFlags field
 *  Each setting affects 4 SGLS, from SGL0 to SGL3.
 *      D = Data
 *      C = Cache DIF
 *      I = Interleaved
 *      H = Host DIF
 */
#define MPI25_SCSIIO_DMAFLAGS_OP_MASK               (0x0F)
#define MPI25_SCSIIO_DMAFLAGS_OP_D_D_D_D            (0x00)
#define MPI25_SCSIIO_DMAFLAGS_OP_D_D_D_C            (0x01)
#define MPI25_SCSIIO_DMAFLAGS_OP_D_D_D_I            (0x02)
#define MPI25_SCSIIO_DMAFLAGS_OP_D_D_C_C            (0x03)
#define MPI25_SCSIIO_DMAFLAGS_OP_D_D_C_I            (0x04)
#define MPI25_SCSIIO_DMAFLAGS_OP_D_D_I_I            (0x05)
#define MPI25_SCSIIO_DMAFLAGS_OP_D_C_C_C            (0x06)
#define MPI25_SCSIIO_DMAFLAGS_OP_D_C_C_I            (0x07)
#define MPI25_SCSIIO_DMAFLAGS_OP_D_C_I_I            (0x08)
#define MPI25_SCSIIO_DMAFLAGS_OP_D_I_I_I            (0x09)
#define MPI25_SCSIIO_DMAFLAGS_OP_D_H_D_D            (0x0A)
#define MPI25_SCSIIO_DMAFLAGS_OP_D_H_D_C            (0x0B)
#define MPI25_SCSIIO_DMAFLAGS_OP_D_H_D_I            (0x0C)
#define MPI25_SCSIIO_DMAFLAGS_OP_D_H_C_C            (0x0D)
#define MPI25_SCSIIO_DMAFLAGS_OP_D_H_C_I            (0x0E)
#define MPI25_SCSIIO_DMAFLAGS_OP_D_H_I_I            (0x0F)

/* number of SGLOffset fields */
#define MPI25_SCSIIO_NUM_SGLOFFSETS                 (4)

/* defines for the IoFlags field */
#define MPI25_SCSIIO_IOFLAGS_IO_PATH_MASK           (0xC000)
#define MPI25_SCSIIO_IOFLAGS_NORMAL_PATH            (0x0000)
#define MPI25_SCSIIO_IOFLAGS_FAST_PATH              (0x4000)

#define MPI25_SCSIIO_IOFLAGS_LARGE_CDB                  (0x1000)
#define MPI25_SCSIIO_IOFLAGS_BIDIRECTIONAL              (0x0800)
#define MPI25_SCSIIO_IOFLAGS_CMD_DETERMINES_DATA_DIR    (0x0200)
#define MPI25_SCSIIO_IOFLAGS_CDBLENGTH_MASK             (0x01FF)

/* MPI v2.5 defines for the EEDPFlags bits */
/* use MPI2_SCSIIO_EEDPFLAGS_ defines for the other EEDPFlags bits */
#define MPI25_SCSIIO_EEDPFLAGS_ESCAPE_MODE_MASK             (0x00C0)
#define MPI25_SCSIIO_EEDPFLAGS_COMPATIBLE_MODE              (0x0000)
#define MPI25_SCSIIO_EEDPFLAGS_DO_NOT_DISABLE_MODE          (0x0040)
#define MPI25_SCSIIO_EEDPFLAGS_APPTAG_DISABLE_MODE          (0x0080)
#define MPI25_SCSIIO_EEDPFLAGS_APPTAG_REFTAG_DISABLE_MODE   (0x00C0)

#define MPI25_SCSIIO_EEDPFLAGS_HOST_GUARD_METHOD_MASK       (0x0030)
#define MPI25_SCSIIO_EEDPFLAGS_T10_CRC_HOST_GUARD           (0x0000)
#define MPI25_SCSIIO_EEDPFLAGS_IP_CHKSUM_HOST_GUARD         (0x0010)

/* use MPI2_LUN_ defines from mpi2.h for the LUN field */

/* use MPI2_SCSIIO_CONTROL_ defines for the Control field */


/* NOTE: The SCSI IO Reply is the same for MPI 2.0 and MPI 2.5, so
 *       MPI2_SCSI_IO_REPLY is used for both.
 */

#endif /* ! codereview */
```

```
370 /* SCSI IO Error Reply Message */
371 typedef struct _MPI2_SCSI_IO_REPLY
372 {
373     U16                     DevHandle;                  /* 0x00 */
374     U8                      MsgLength;                  /* 0x02 */
375     U8                      Function;                   /* 0x03 */
376     U16                     Reserved1;                  /* 0x04 */
377     U8                      Reserved2;                  /* 0x06 */
378     U8                      MsgFlags;                   /* 0x07 */
379     U8                      VP_ID;                      /* 0x08 */
380     U8                      VF_ID;                      /* 0x09 */
381     U16                     Reserved3;                  /* 0x0A */
382     U8                      SCSIStatus;                 /* 0x0C */
383     U8                      SCSIState;                  /* 0x0D */
384     U16                     IOCStatus;                  /* 0x0E */
385     U32                     IOCLogInfo;                 /* 0x10 */
386     U32                     TransferCount;              /* 0x14 */
387     U32                     SenseCount;                 /* 0x18 */
388     U32                     ResponseInfo;               /* 0x1C */
389     U16                     TaskTag;                    /* 0x20 */
390     U16                     Reserved4;                  /* 0x22 */
391     U32                     BidirectionalTransferCount; /* 0x24 */
392     U32                     Reserved5;                  /* 0x28 */
393     U32                     Reserved6;                  /* 0x2C */
394 } MPI2_SCSI_IO_REPLY, MPI2_POINTER PTR_MPI2_SCSI_IO_REPLY,
395   Mpi2SCSIIOReply_t, MPI2_POINTER pMpi2SCSIIOReply_t;

397 /* SCSI IO Reply SCSIStatus values (SAM-4 status codes) */

399 #define MPI2_SCSI_STATUS_GOOD                   (0x00)
400 #define MPI2_SCSI_STATUS_CHECK_CONDITION        (0x02)
401 #define MPI2_SCSI_STATUS_CONDITION_MET          (0x04)
402 #define MPI2_SCSI_STATUS_BUSY                   (0x08)
403 #define MPI2_SCSI_STATUS_INTERMEDIATE           (0x10)
404 #define MPI2_SCSI_STATUS_INTERMEDIATE_CONDMET   (0x14)
405 #define MPI2_SCSI_STATUS_RESERVATION_CONFLICT   (0x18)
406 #define MPI2_SCSI_STATUS_COMMAND_TERMINATED     (0x22) /* obsolete */
407 #define MPI2_SCSI_STATUS_TASK_SET_FULL          (0x28)
408 #define MPI2_SCSI_STATUS_ACA_ACTIVE             (0x30)
409 #define MPI2_SCSI_STATUS_TASK_ABORTED           (0x40)

411 /* SCSI IO Reply SCSIState flags */

413 #define MPI2_SCSI_STATE_RESPONSE_INFO_VALID     (0x10)
414 #define MPI2_SCSI_STATE_TERMINATED              (0x08)
415 #define MPI2_SCSI_STATE_NO_SCSI_STATUS          (0x04)
416 #define MPI2_SCSI_STATE_AUTOSENSE_FAILED        (0x02)
417 #define MPI2_SCSI_STATE_AUTOSENSE_VALID         (0x01)

419 /* masks and shifts for the ResponseInfo field */

421 #define MPI2_SCSI_RI_MASK_REASONCODE            (0x000000FF)
422 #define MPI2_SCSI_RI_SHIFT_REASONCODE           (0)

424 #endif /* ! codereview */
425 #define MPI2_SCSI_TASKTAG_UNKNOWN               (0xFFFF)


428 /*****************************************************************************
429 *  SCSI Task Management messages
430 *****************************************************************************/

432 /* SCSI Task Management Request Message */
433 typedef struct _MPI2_SCSI_TASK_MANAGE_REQUEST
434 {
435     U16                     DevHandle;                  /* 0x00 */
```



```
436     U8                      ChainOffset;                /* 0x02 */
437     U8                      Function;                   /* 0x03 */
438     U8                      Reserved1;                  /* 0x04 */
439     U8                      TaskType;                   /* 0x05 */
440     U8                      Reserved2;                  /* 0x06 */
441     U8                      MsgFlags;                   /* 0x07 */
442     U8                      VP_ID;                      /* 0x08 */
443     U8                      VF_ID;                      /* 0x09 */
444     U16                     Reserved3;                  /* 0x0A */
445     U8                      LUN[8];                     /* 0x0C */
446     U32                     Reserved4[7];               /* 0x14 */
447     U16                     TaskMID;                    /* 0x30 */
448     U16                     Reserved5;                  /* 0x32 */
449 } MPI2_SCSI_TASK_MANAGE_REQUEST,
450   MPI2_POINTER PTR_MPI2_SCSI_TASK_MANAGE_REQUEST,
451   Mpi2SCSITaskManagementRequest_t,
452   MPI2_POINTER pMpi2SCSITaskManagementRequest_t;

454 /* TaskType values */

456 #define MPI2_SCSITASKMGMT_TASKTYPE_ABORT_TASK           (0x01)
457 #define MPI2_SCSITASKMGMT_TASKTYPE_ABRT_TASK_SET        (0x02)
458 #define MPI2_SCSITASKMGMT_TASKTYPE_TARGET_RESET         (0x03)
459 #define MPI2_SCSITASKMGMT_TASKTYPE_LOGICAL_UNIT_RESET   (0x05)
460 #define MPI2_SCSITASKMGMT_TASKTYPE_CLEAR_TASK_SET       (0x06)
461 #define MPI2_SCSITASKMGMT_TASKTYPE_QUERY_TASK           (0x07)
462 #define MPI2_SCSITASKMGMT_TASKTYPE_CLR_ACA              (0x08)
463 #define MPI2_SCSITASKMGMT_TASKTYPE_QRY_TASK_SET         (0x09)
464 #define MPI2_SCSITASKMGMT_TASKTYPE_QRY_ASYNC_EVENT      (0x0A)

466 /* obsolete TaskType name */
467 #define MPI2_SCSITASKMGMT_TASKTYPE_QRY_UNIT_ATTENTION   (MPI2_SCSITASKMGMT_TASKT

469 /* MsgFlags bits */

471 #define MPI2_SCSITASKMGMT_MSGFLAGS_MASK_TARGET_RESET    (0x18)
472 #define MPI2_SCSITASKMGMT_MSGFLAGS_LINK_RESET           (0x00)
473 #define MPI2_SCSITASKMGMT_MSGFLAGS_NEXUS_RESET_SRST     (0x08)
474 #define MPI2_SCSITASKMGMT_MSGFLAGS_SAS_HARD_LINK_RESET  (0x10)

476 #define MPI2_SCSITASKMGMT_MSGFLAGS_DO_NOT_SEND_TASK_IU  (0x01)



480 /* SCSI Task Management Reply Message */
481 typedef struct _MPI2_SCSI_TASK_MANAGE_REPLY
482 {
483     U16                     DevHandle;                  /* 0x00 */
484     U8                      MsgLength;                  /* 0x02 */
485     U8                      Function;                   /* 0x03 */
486     U8                      ResponseCode;               /* 0x04 */
487     U8                      TaskType;                   /* 0x05 */
488     U8                      Reserved1;                  /* 0x06 */
489     U8                      MsgFlags;                   /* 0x07 */
490     U8                      VP_ID;                      /* 0x08 */
491     U8                      VF_ID;                      /* 0x09 */
492     U16                     Reserved2;                  /* 0x0A */
493     U16                     Reserved3;                  /* 0x0C */
494     U16                     IOCStatus;                  /* 0x0E */
495     U32                     IOCLogInfo;                 /* 0x10 */
496     U32                     TerminationCount;           /* 0x14 */
497     U32                     ResponseInfo;               /* 0x18 */
498 #endif /* ! codereview */
499 } MPI2_SCSI_TASK_MANAGE_REPLY,
500   MPI2_POINTER PTR_MPI2_SCSI_TASK_MANAGE_REPLY,
501   Mpi2SCSITaskManagementReply_t, MPI2_POINTER pMpi2SCSIManagementReply_t;
```

```
503 /* ResponseCode values */

505 #define MPI2_SCSITASKMGMT_RSP_TM_COMPLETE               (0x00)
506 #define MPI2_SCSITASKMGMT_RSP_INVALID_FRAME             (0x02)
507 #define MPI2_SCSITASKMGMT_RSP_TM_NOT_SUPPORTED          (0x04)
508 #define MPI2_SCSITASKMGMT_RSP_TM_FAILED                 (0x05)
509 #define MPI2_SCSITASKMGMT_RSP_TM_SUCCEEDED              (0x08)
510 #define MPI2_SCSITASKMGMT_RSP_TM_INVALID_LUN            (0x09)
511 #define MPI2_SCSITASKMGMT_RSP_TM_OVERLAPPED_TAG         (0x0A)
512 #endif /* ! codereview */
513 #define MPI2_SCSITASKMGMT_RSP_IO_QUEUED_ON_IOC          (0x80)

515 /* masks and shifts for the ResponseInfo field */

517 #define MPI2_SCSITASKMGMT_RI_MASK_REASONCODE            (0x000000FF)
518 #define MPI2_SCSITASKMGMT_RI_SHIFT_REASONCODE           (0)
519 #define MPI2_SCSITASKMGMT_RI_MASK_ARI2                  (0x0000FF00)
520 #define MPI2_SCSITASKMGMT_RI_SHIFT_ARI2                 (8)
521 #define MPI2_SCSITASKMGMT_RI_MASK_ARI1                  (0x00FF0000)
522 #define MPI2_SCSITASKMGMT_RI_SHIFT_ARI1                 (16)
523 #define MPI2_SCSITASKMGMT_RI_MASK_ARI0                  (0xFF000000)
524 #define MPI2_SCSITASKMGMT_RI_SHIFT_ARI0                 (24)

526 #endif /* ! codereview */

528 /*****************************************************************************
529 *  SCSI Enclosure Processor messages
530 *****************************************************************************/

532 /* SCSI Enclosure Processor Request Message */
533 typedef struct _MPI2_SEP_REQUEST
534 {
535     U16                     DevHandle;          /* 0x00 */
536     U8                      ChainOffset;        /* 0x02 */
537     U8                      Function;           /* 0x03 */
538     U8                      Action;             /* 0x04 */
539     U8                      Flags;              /* 0x05 */
540     U8                      Reserved1;          /* 0x06 */
541     U8                      MsgFlags;           /* 0x07 */
542     U8                      VP_ID;              /* 0x08 */
543     U8                      VF_ID;              /* 0x09 */
544     U16                     Reserved2;          /* 0x0A */
545     U32                     SlotStatus;         /* 0x0C */
546     U32                     Reserved3;          /* 0x10 */
547     U32                     Reserved4;          /* 0x14 */
548     U32                     Reserved5;          /* 0x18 */
549     U16                     Slot;               /* 0x1C */
550     U16                     EnclosureHandle;    /* 0x1E */
551 } MPI2_SEP_REQUEST, MPI2_POINTER PTR_MPI2_SEP_REQUEST,
552   Mpi2SepRequest_t, MPI2_POINTER pMpi2SepRequest_t;

554 /* Action defines */
555 #define MPI2_SEP_REQ_ACTION_WRITE_STATUS                (0x00)
556 #define MPI2_SEP_REQ_ACTION_READ_STATUS                 (0x01)

558 /* Flags defines */
559 #define MPI2_SEP_REQ_FLAGS_DEVHANDLE_ADDRESS            (0x00)
560 #define MPI2_SEP_REQ_FLAGS_ENCLOSURE_SLOT_ADDRESS       (0x01)

562 /* SlotStatus defines */
563 #define MPI2_SEP_REQ_SLOTSTATUS_REQUEST_REMOVE          (0x00040000)
564 #define MPI2_SEP_REQ_SLOTSTATUS_IDENTIFY_REQUEST        (0x00020000)
565 #define MPI2_SEP_REQ_SLOTSTATUS_REBUILD_STOPPED         (0x00000200)
566 #define MPI2_SEP_REQ_SLOTSTATUS_HOT_SPARE               (0x00000100)
567 #define MPI2_SEP_REQ_SLOTSTATUS_UNCONFIGURED            (0x00000080)
```



```
568 #define MPI2_SEP_REQ_SLOTSTATUS_PREDICTED_FAULT         (0x00000040)
569 #define MPI2_SEP_REQ_SLOTSTATUS_IN_CRITICAL_ARRAY       (0x00000010)
570 #define MPI2_SEP_REQ_SLOTSTATUS_IN_FAILED_ARRAY         (0x00000008)
571 #define MPI2_SEP_REQ_SLOTSTATUS_DEV_REBUILDING          (0x00000004)
572 #define MPI2_SEP_REQ_SLOTSTATUS_DEV_FAULTY              (0x00000002)
573 #define MPI2_SEP_REQ_SLOTSTATUS_NO_ERROR                (0x00000001)


576 /* SCSI Enclosure Processor Reply Message */
577 typedef struct _MPI2_SEP_REPLY
578 {
579     U16                     DevHandle;          /* 0x00 */
580     U8                      MsgLength;          /* 0x02 */
581     U8                      Function;           /* 0x03 */
582     U8                      Action;             /* 0x04 */
583     U8                      Flags;              /* 0x05 */
584     U8                      Reserved1;          /* 0x06 */
585     U8                      MsgFlags;           /* 0x07 */
586     U8                      VP_ID;              /* 0x08 */
587     U8                      VF_ID;              /* 0x09 */
588     U16                     Reserved2;          /* 0x0A */
589     U16                     Reserved3;          /* 0x0C */
590     U16                     IOCStatus;          /* 0x0E */
591     U32                     IOCLogInfo;         /* 0x10 */
592     U32                     SlotStatus;         /* 0x14 */
593     U32                     Reserved4;          /* 0x18 */
594     U16                     Slot;               /* 0x1C */
595     U16                     EnclosureHandle;    /* 0x1E */
596 } MPI2_SEP_REPLY, MPI2_POINTER PTR_MPI2_SEP_REPLY,
597   Mpi2SepReply_t, MPI2_POINTER pMpi2SepReply_t;

599 /* SlotStatus defines */
600 #define MPI2_SEP_REPLY_SLOTSTATUS_REMOVE_READY          (0x00040000)
601 #define MPI2_SEP_REPLY_SLOTSTATUS_IDENTIFY_REQUEST      (0x00020000)
602 #define MPI2_SEP_REPLY_SLOTSTATUS_REBUILD_STOPPED       (0x00000200)
603 #define MPI2_SEP_REPLY_SLOTSTATUS_HOT_SPARE             (0x00000100)
604 #define MPI2_SEP_REPLY_SLOTSTATUS_UNCONFIGURED          (0x00000080)
605 #define MPI2_SEP_REPLY_SLOTSTATUS_PREDICTED_FAULT       (0x00000040)
606 #define MPI2_SEP_REPLY_SLOTSTATUS_IN_CRITICAL_ARRAY     (0x00000010)
607 #define MPI2_SEP_REPLY_SLOTSTATUS_IN_FAILED_ARRAY       (0x00000008)
608 #define MPI2_SEP_REPLY_SLOTSTATUS_DEV_REBUILDING        (0x00000004)
609 #define MPI2_SEP_REPLY_SLOTSTATUS_DEV_FAULTY            (0x00000002)
610 #define MPI2_SEP_REPLY_SLOTSTATUS_NO_ERROR              (0x00000001)


613 #endif
```

```
**********************************************************
   83013 Thu Jun 12 17:42:18 2014
new/usr/src/uts/common/sys/scsi/adapters/mpt_sas3/mpi/mpi2_ioc.h
Initial modifications using the code changes present between
the LSI source code for FreeBSD drivers. Specifically the changes
between from mpslsi-source-17.00.00.00 -> mpslsi-source-03.00.00.00.
This mainly involves using a different scatter/gather element in
frame setup.
**********************************************************
/*
 * 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) 2000-2012 LSI Corporation.
 * Copyright (c) 2000 to 2009, LSI Corporation.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms of all code within
 * this file that is exclusively owned by LSI, with or without
 * modification, is permitted provided that, in addition to the CDDL 1.0
 * License requirements, the following conditions are met:
 *
 *    Neither the name of the author nor the names of its contributors may be
 *    used to endorse or promote products derived from this software without
 *    specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS 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
 * COPYRIGHT OWNER 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.
 */

/*
 *          Name:  mpi2_ioc.h
 *         Title:  MPI IOC, Port, Event, FW Download, and FW Upload messages
 *  Creation Date:  October 11, 2006
 *
 *  mpi2_ioc.h Version:  02.00.xx
 *
 *  NOTE: Names (typedefs, defines, etc.) beginning with an MPI25 or Mpi25
```



```
 *       prefix are for use only on MPI v2.5 products, and must not be used
 *       with MPI v2.0 products. Unless otherwise noted, names beginning with
 *       MPI2 or Mpi2 are for use with both MPI v2.0 and MPI v2.5 products.
 * mpi2_ioc.h Version:  02.00.12
 *
 * Version History
 * ---------------
 *
 * Date      Version   Description
 * --------  --------  ------------------------------------------------------
 * 04-30-07  02.00.00  Corresponds to Fusion-MPT MPI Specification Rev A.
 * 06-04-07  02.00.01  In IOCFacts Reply structure, renamed MaxDevices to
 *                     MaxTargets.
 *                     Added TotalImageSize field to FWDownload Request.
 *                     Added reserved words to FWUpload Request.
 * 06-26-07  02.00.02  Added IR Configuration Change List Event.
 * 08-31-07  02.00.03  Removed SystemReplyQueueDepth field from the IOCInit
 *                     request and replaced it with
 *                     ReplyDescriptorPostQueueDepth and ReplyFreeQueueDepth.
 *                     Replaced the MinReplyQueueDepth field of the IOCFacts
 *                     reply with MaxReplyDescriptorPostQueueDepth.
 *                     Added MPI2_RDPQ_DEPTH_MIN define to specify the minimum
 *                     depth for the Reply Descriptor Post Queue.
 *                     Added SASAddress field to Initiator Device Table
 *                     Overflow Event data.
 * 10-31-07  02.00.04  Added ReasonCode MPI2_EVENT_SAS_INIT_RC_NOT_RESPONDING
 *                     for SAS Initiator Device Status Change Event data.
 *                     Modified Reason Code defines for SAS Topology Change
 *                     List Event data, including adding a bit for PHY Vacant
 *                     status, and adding a mask for the Reason Code.
 *                     Added define for
 *                     MPI2_EVENT_SAS_TOPO_ES_DELAY_NOT_RESPONDING.
 *                     Added define for MPI2_EXT_IMAGE_TYPE_MEGARAID.
 * 12-18-07  02.00.05  Added Boot Status defines for the IOCExceptions field of
 *                     the IOCFacts Reply.
 *                     Removed MPI2_IOCFACTS_CAPABILITY_EXTENDED_BUFFER define.
 *                     Moved MPI2_VERSION_UNION to mpi2.h.
 *                     Changed MPI2_EVENT_NOTIFICATION_REQUEST to use masks
 *                     instead of enables, and added SASBroadcastPrimitiveMasks
 *                     field.
 *                     Added Log Entry Added Event and related structure.
 * 02-29-08  02.00.06  Added define MPI2_IOCFACTS_CAPABILITY_INTEGRATED_RAID.
 *                     Removed define MPI2_IOCFACTS_PROTOCOL_SMP_TARGET.
 *                     Added MaxVolumes and MaxPersistentEntries fields to
 *                     IOCFacts reply.
 *                     Added ProtocalFlags and IOCCapabilities fields to
 *                     MPI2_FW_IMAGE_HEADER.
 *                     Removed MPI2_PORTENABLE_FLAGS_ENABLE_SINGLE_PORT.
 * 03-03-08  02.00.07  Fixed MPI2_FW_IMAGE_HEADER by changing Reserved26 to
 *                     a U16 (from a U32).
 *                     Removed extra 's' from EventMasks name.
 * 06-27-08  02.00.08  Fixed an offset in a comment.
 * 10-02-08  02.00.09  Removed SystemReplyFrameSize from MPI2_IOC_INIT_REQUEST.
 *                     Removed CurReplyFrameSize from MPI2_IOC_FACTS_REPLY and
 *                     renamed MinReplyFrameSize to ReplyFrameSize.
 *                     Added MPI2_IOCFACTS_EXCEPT_IR_FOREIGN_CONFIG_MAX.
 *                     Added two new RAIDOperation values for Integrated RAID
 *                     Operations Status Event data.
 *                     Added four new IR Configuration Change List Event data
 *                     ReasonCode values.
 *                     Added two new ReasonCode defines for SAS Device Status
 *                     Change Event data.
 *                     Added three new DiscoveryStatus bits for the SAS
 *                     Discovery event data.
 *                     Added Multiplexing Status Change bit to the PhyStatus
 *                     field of the SAS Topology Change List event data.
```

```
 *                      Removed define for MPI2_INIT_IMAGE_BOOTFLAGS_XMEMCOPY.
 *                      BootFlags are now product-specific.
 *                      Added defines for the indivdual signature bytes
 *                      for MPI2_INIT_IMAGE_FOOTER.
 *  01-19-09  02.00.10  Added MPI2_IOCFACTS_CAPABILITY_EVENT_REPLAY define.
 *                      Added MPI2_EVENT_SAS_DISC_DS_DOWNSTREAM_INITIATOR
 *                      define.
 *                      Added MPI2_EVENT_SAS_DEV_STAT_RC_SATA_INIT_FAILURE
 *                      define.
 *                      Removed MPI2_EVENT_SAS_DISC_DS_SATA_INIT_FAILURE define.
 *  05-06-09  02.00.11  Added MPI2_IOCFACTS_CAPABILITY_RAID_ACCELERATOR define.
 *                      Added MPI2_IOCFACTS_CAPABILITY_MSI_X_INDEX define.
 *                      Added two new reason codes for SAS Device Status Change
 *                      Event.
 *                      Added new event: SAS PHY Counter.
 *  07-30-09  02.00.12  Added GPIO Interrupt event define and structure.
 *                      Added MPI2_IOCFACTS_CAPABILITY_EXTENDED_BUFFER define.
 *                      Added new product id family for 2208.
 *  10-28-09  02.00.13  Added HostMSIxVectors field to MPI2_IOC_INIT_REQUEST.
 *                      Added MaxMSIxVectors field to MPI2_IOC_FACTS_REPLY.
 *                      Added MinDevHandle field to MPI2_IOC_FACTS_REPLY.
 *                      Added MPI2_IOCFACTS_CAPABILITY_HOST_BASED_DISCOVERY.
 *                      Added MPI2_EVENT_HOST_BASED_DISCOVERY_PHY define.
 *                      Added MPI2_EVENT_SAS_TOPO_ES_NO_EXPANDER define.
 *                      Added Host Based Discovery Phy Event data.
 *                      Added defines for ProductID Product field
 *                      (MPI2_FW_HEADER_PID_).
 *                      Modified values for SAS ProductID Family
 *                      (MPI2_FW_HEADER_PID_FAMILY_).
 *  02-10-10  02.00.14  Added SAS Quiesce Event structure and defines.
 *                      Added PowerManagementControl Request structures and
 *                      defines.
 *  05-12-10  02.00.15  Marked Task Set Full Event as obsolete.
 *                      Added MPI2_EVENT_SAS_TOPO_LR_UNSUPPORTED_PHY define.
 *  11-10-10  02.00.16  Added MPI2_FW_DOWNLOAD_ITYPE_MIN_PRODUCT_SPECIFIC.
#endif /* ! codereview */
 *  --------------------------------------------------------------------------
 */

#ifndef MPI2_IOC_H
#define MPI2_IOC_H

/*****************************************************************************
*
*               IOC Messages
*
*****************************************************************************/

/*****************************************************************************
*  IOCInit message
*****************************************************************************/

/* IOCInit Request message */
typedef struct _MPI2_IOC_INIT_REQUEST
{
    U8                      WhoInit;                        /* 0x00 */
    U8                      Reserved1;                      /* 0x01 */
    U8                      ChainOffset;                    /* 0x02 */
    U8                      Function;                       /* 0x03 */
    U16                     Reserved2;                      /* 0x04 */
    U8                      Reserved3;                      /* 0x06 */
    U8                      MsgFlags;                       /* 0x07 */
    U8                      VP_ID;                          /* 0x08 */
    U8                      VF_ID;                          /* 0x09 */
    U16                     Reserved4;                      /* 0x0A */
    U16                     MsgVersion;                     /* 0x0C */
```



```
    U16                     HeaderVersion;                  /* 0x0E */
    U32                     Reserved5;                      /* 0x10 */
    U16                     Reserved6;                      /* 0x14 */
    U8                      Reserved7;                      /* 0x16 */
    U8                      HostMSIxVectors;                /* 0x17 */
    U16                     Reserved8;                      /* 0x18 */
    U32                     Reserved6;                      /* 0x14 */
    U16                     Reserved7;                      /* 0x18 */
    U16                     SystemRequestFrameSize;         /* 0x1A */
    U16                     ReplyDescriptorPostQueueDepth;  /* 0x1C */
    U16                     ReplyFreeQueueDepth;            /* 0x1E */
    U32                     SenseBufferAddressHigh;         /* 0x20 */
    U32                     SystemReplyAddressHigh;         /* 0x24 */
    U64                     SystemRequestFrameBaseAddress;  /* 0x28 */
    U64                     ReplyDescriptorPostQueueAddress;/* 0x30 */
    U64                     ReplyFreeQueueAddress;          /* 0x38 */
    U64                     TimeStamp;                      /* 0x40 */
} MPI2_IOC_INIT_REQUEST, MPI2_POINTER PTR_MPI2_IOC_INIT_REQUEST,
_____unchanged_portion_omitted_
  Mpi2IOCFactsRequest_t, MPI2_POINTER pMpi2IOCFactsRequest_t;


/* IOCFacts Reply message */
typedef struct _MPI2_IOC_FACTS_REPLY
{
    U16                     MsgVersion;                     /* 0x00 */
    U8                      MsgLength;                      /* 0x02 */
    U8                      Function;                       /* 0x03 */
    U16                     HeaderVersion;                  /* 0x04 */
    U8                      IOCNumber;                      /* 0x06 */
    U8                      MsgFlags;                       /* 0x07 */
    U8                      VP_ID;                          /* 0x08 */
    U8                      VF_ID;                          /* 0x09 */
    U16                     Reserved1;                      /* 0x0A */
    U16                     IOCExceptions;                  /* 0x0C */
    U16                     IOCStatus;                      /* 0x0E */
    U32                     IOCLogInfo;                     /* 0x10 */
    U8                      MaxChainDepth;                  /* 0x14 */
    U8                      WhoInit;                        /* 0x15 */
    U8                      NumberOfPorts;                  /* 0x16 */
    U8                      MaxMSIxVectors;                 /* 0x17 */
    U8                      Reserved2;                      /* 0x17 */
    U16                     RequestCredit;                  /* 0x18 */
    U16                     ProductID;                      /* 0x1A */
    U32                     IOCCapabilities;                /* 0x1C */
    MPI2_VERSION_UNION      FWVersion;                      /* 0x20 */
    U16                     IOCRequestFrameSize;            /* 0x24 */
    U16                     Reserved3;                      /* 0x26 */
    U16                     MaxInitiators;                  /* 0x28 */
    U16                     MaxTargets;                     /* 0x2A */
    U16                     MaxSasExpanders;                /* 0x2C */
    U16                     MaxEnclosures;                  /* 0x2E */
    U16                     ProtocolFlags;                  /* 0x30 */
    U16                     HighPriorityCredit;             /* 0x32 */
    U16                     MaxReplyDescriptorPostQueueDepth; /* 0x34 */
    U8                      ReplyFrameSize;                 /* 0x36 */
    U8                      MaxVolumes;                     /* 0x37 */
    U16                     MaxDevHandle;                   /* 0x38 */
    U16                     MaxPersistentEntries;           /* 0x3A */
    U16                     MinDevHandle;                   /* 0x3C */
    U16                     Reserved4;                      /* 0x3E */
    U32                     Reserved4;                      /* 0x3C */
} MPI2_IOC_FACTS_REPLY, MPI2_POINTER PTR_MPI2_IOC_FACTS_REPLY,
  Mpi2IOCFactsReply_t, MPI2_POINTER pMpi2IOCFactsReply_t;

/* MsgVersion */
```

```
311 #define MPI2_IOCFACTS_MSGVERSION_MAJOR_MASK             (0xFF00)
312 #define MPI2_IOCFACTS_MSGVERSION_MAJOR_SHIFT            (8)
313 #define MPI2_IOCFACTS_MSGVERSION_MINOR_MASK             (0x00FF)
314 #define MPI2_IOCFACTS_MSGVERSION_MINOR_SHIFT            (0)

316 /* HeaderVersion */
317 #define MPI2_IOCFACTS_HDRVERSION_UNIT_MASK              (0xFF00)
318 #define MPI2_IOCFACTS_HDRVERSION_UNIT_SHIFT             (8)
319 #define MPI2_IOCFACTS_HDRVERSION_DEV_MASK               (0x00FF)
320 #define MPI2_IOCFACTS_HDRVERSION_DEV_SHIFT              (0)

322 /* IOCExceptions */
323 #define MPI2_IOCFACTS_EXCEPT_IR_FOREIGN_CONFIG_MAX      (0x0100)

325 #define MPI2_IOCFACTS_EXCEPT_BOOTSTAT_MASK              (0x00E0)
326 #define MPI2_IOCFACTS_EXCEPT_BOOTSTAT_GOOD              (0x0000)
327 #define MPI2_IOCFACTS_EXCEPT_BOOTSTAT_BACKUP            (0x0020)
328 #define MPI2_IOCFACTS_EXCEPT_BOOTSTAT_RESTORED          (0x0040)
329 #define MPI2_IOCFACTS_EXCEPT_BOOTSTAT_CORRUPT_BACKUP    (0x0060)

331 #define MPI2_IOCFACTS_EXCEPT_METADATA_UNSUPPORTED       (0x0010)
332 #define MPI2_IOCFACTS_EXCEPT_MANUFACT_CHECKSUM_FAIL     (0x0008)
333 #define MPI2_IOCFACTS_EXCEPT_FW_CHECKSUM_FAIL           (0x0004)
334 #define MPI2_IOCFACTS_EXCEPT_RAID_CONFIG_INVALID        (0x0002)
335 #define MPI2_IOCFACTS_EXCEPT_CONFIG_CHECKSUM_FAIL       (0x0001)

337 /* defines for WhoInit field are after the IOCInit Request */

339 /* ProductID field uses MPI2_FW_HEADER_PID_ */

341 /* IOCCapabilities */
342 #define MPI25_IOCFACTS_CAPABILITY_FAST_PATH_CAPABLE     (0x00020000)
343 #define MPI2_IOCFACTS_CAPABILITY_HOST_BASED_DISCOVERY   (0x00010000)
344 #endif /* ! codereview */
345 #define MPI2_IOCFACTS_CAPABILITY_MSI_X_INDEX            (0x00008000)
346 #define MPI2_IOCFACTS_CAPABILITY_RAID_ACCELERATOR       (0x00004000)
347 #define MPI2_IOCFACTS_CAPABILITY_EVENT_REPLAY           (0x00002000)
348 #define MPI2_IOCFACTS_CAPABILITY_INTEGRATED_RAID        (0x00001000)
349 #define MPI2_IOCFACTS_CAPABILITY_TLR                    (0x00000800)
350 #define MPI2_IOCFACTS_CAPABILITY_MULTICAST              (0x00000100)
351 #define MPI2_IOCFACTS_CAPABILITY_BIDIRECTIONAL_TARGET   (0x00000080)
352 #define MPI2_IOCFACTS_CAPABILITY_EEDP                   (0x00000040)
353 #define MPI2_IOCFACTS_CAPABILITY_EXTENDED_BUFFER        (0x00000020)
354 #define MPI2_IOCFACTS_CAPABILITY_SNAPSHOT_BUFFER        (0x00000010)
355 #define MPI2_IOCFACTS_CAPABILITY_DIAG_TRACE_BUFFER      (0x00000008)
356 #define MPI2_IOCFACTS_CAPABILITY_TASK_SET_FULL_HANDLING (0x00000004)

358 /* ProtocolFlags */
359 #define MPI2_IOCFACTS_PROTOCOL_SCSI_TARGET              (0x0001)
360 #define MPI2_IOCFACTS_PROTOCOL_SCSI_INITIATOR           (0x0002)


363 /****************************************************************************
364 *  PortFacts message
365 ****************************************************************************/

367 /* PortFacts Request message */
368 typedef struct _MPI2_PORT_FACTS_REQUEST
369 {
370     U16                     Reserved1;                  /* 0x00 */
371     U8                      ChainOffset;                /* 0x02 */
372     U8                      Function;                   /* 0x03 */
373     U16                     Reserved2;                  /* 0x04 */
374     U8                      PortNumber;                 /* 0x06 */
375     U8                      MsgFlags;                   /* 0x07 */
376     U8                      VP_ID;                      /* 0x08 */
```



```
377     U8                      VF_ID;                      /* 0x09 */
378     U16                     Reserved3;                  /* 0x0A */
379 } MPI2_PORT_FACTS_REQUEST, MPI2_POINTER PTR_MPI2_PORT_FACTS_REQUEST,
380   Mpi2PortFactsRequest_t, MPI2_POINTER pMpi2PortFactsRequest_t;

382 /* PortFacts Reply message */
383 typedef struct _MPI2_PORT_FACTS_REPLY
384 {
385     U16                     Reserved1;                  /* 0x00 */
386     U8                      MsgLength;                  /* 0x02 */
387     U8                      Function;                   /* 0x03 */
388     U16                     Reserved2;                  /* 0x04 */
389     U8                      PortNumber;                 /* 0x06 */
390     U8                      MsgFlags;                   /* 0x07 */
391     U8                      VP_ID;                      /* 0x08 */
392     U8                      VF_ID;                      /* 0x09 */
393     U16                     Reserved3;                  /* 0x0A */
394     U16                     Reserved4;                  /* 0x0C */
395     U16                     IOCStatus;                  /* 0x0E */
396     U32                     IOCLogInfo;                 /* 0x10 */
397     U8                      Reserved5;                  /* 0x14 */
398     U8                      PortType;                   /* 0x15 */
399     U16                     Reserved6;                  /* 0x16 */
400     U16                     MaxPostedCmdBuffers;        /* 0x18 */
401     U16                     Reserved7;                  /* 0x1A */
402 } MPI2_PORT_FACTS_REPLY, MPI2_POINTER PTR_MPI2_PORT_FACTS_REPLY,
403   Mpi2PortFactsReply_t, MPI2_POINTER pMpi2PortFactsReply_t;

405 /* PortType values */
406 #define MPI2_PORTFACTS_PORTTYPE_INACTIVE            (0x00)
407 #define MPI2_PORTFACTS_PORTTYPE_FC                  (0x10)
408 #define MPI2_PORTFACTS_PORTTYPE_ISCSI               (0x20)
409 #define MPI2_PORTFACTS_PORTTYPE_SAS_PHYSICAL        (0x30)
410 #define MPI2_PORTFACTS_PORTTYPE_SAS_VIRTUAL         (0x31)


413 /****************************************************************************
414 *  PortEnable message
415 ****************************************************************************/

417 /* PortEnable Request message */
418 typedef struct _MPI2_PORT_ENABLE_REQUEST
419 {
420     U16                     Reserved1;                  /* 0x00 */
421     U8                      ChainOffset;                /* 0x02 */
422     U8                      Function;                   /* 0x03 */
423     U8                      Reserved2;                  /* 0x04 */
424     U8                      PortFlags;                  /* 0x05 */
425     U8                      Reserved3;                  /* 0x06 */
426     U8                      MsgFlags;                   /* 0x07 */
427     U8                      VP_ID;                      /* 0x08 */
428     U8                      VF_ID;                      /* 0x09 */
429     U16                     Reserved4;                  /* 0x0A */
430 } MPI2_PORT_ENABLE_REQUEST, MPI2_POINTER PTR_MPI2_PORT_ENABLE_REQUEST,
431   Mpi2PortEnableRequest_t, MPI2_POINTER pMpi2PortEnableRequest_t;


434 /* PortEnable Reply message */
435 typedef struct _MPI2_PORT_ENABLE_REPLY
436 {
437     U16                     Reserved1;                  /* 0x00 */
438     U8                      MsgLength;                  /* 0x02 */
439     U8                      Function;                   /* 0x03 */
440     U8                      Reserved2;                  /* 0x04 */
441     U8                      PortFlags;                  /* 0x05 */
442     U8                      Reserved3;                  /* 0x06 */
```

```
    U8                      MsgFlags;                       /* 0x07 */
    U8                      VP_ID;                          /* 0x08 */
    U8                      VF_ID;                          /* 0x09 */
    U16                     Reserved4;                      /* 0x0A */
    U16                     Reserved5;                      /* 0x0C */
    U16                     IOCStatus;                      /* 0x0E */
    U32                     IOCLogInfo;                     /* 0x10 */
} MPI2_PORT_ENABLE_REPLY, MPI2_POINTER PTR_MPI2_PORT_ENABLE_REPLY,
  Mpi2PortEnableReply_t, MPI2_POINTER pMpi2PortEnableReply_t;


/****************************************************************************
*  EventNotification message
****************************************************************************/

/* EventNotification Request message */
#define MPI2_EVENT_NOTIFY_EVENTMASK_WORDS           (4)

typedef struct _MPI2_EVENT_NOTIFICATION_REQUEST
{
    U16                     Reserved1;                      /* 0x00 */
    U8                      ChainOffset;                    /* 0x02 */
    U8                      Function;                       /* 0x03 */
    U16                     Reserved2;                      /* 0x04 */
    U8                      Reserved3;                      /* 0x06 */
    U8                      MsgFlags;                       /* 0x07 */
    U8                      VP_ID;                          /* 0x08 */
    U8                      VF_ID;                          /* 0x09 */
    U16                     Reserved4;                      /* 0x0A */
    U32                     Reserved5;                      /* 0x0C */
    U32                     Reserved6;                      /* 0x10 */
    U32                     EventMasks[MPI2_EVENT_NOTIFY_EVENTMASK_WORDS];/* 0x1
    U16                     SASBroadcastPrimitiveMasks;     /* 0x24 */
    U16                     Reserved7;                      /* 0x26 */
    U32                     Reserved8;                      /* 0x28 */
} MPI2_EVENT_NOTIFICATION_REQUEST,
  MPI2_POINTER PTR_MPI2_EVENT_NOTIFICATION_REQUEST,
  Mpi2EventNotificationRequest_t, MPI2_POINTER pMpi2EventNotificationRequest_t;


/* EventNotification Reply message */
typedef struct _MPI2_EVENT_NOTIFICATION_REPLY
{
    U16                     EventDataLength;                /* 0x00 */
    U8                      MsgLength;                      /* 0x02 */
    U8                      Function;                       /* 0x03 */
    U16                     Reserved1;                      /* 0x04 */
    U8                      AckRequired;                    /* 0x06 */
    U8                      MsgFlags;                       /* 0x07 */
    U8                      VP_ID;                          /* 0x08 */
    U8                      VF_ID;                          /* 0x09 */
    U16                     Reserved2;                      /* 0x0A */
    U16                     Reserved3;                      /* 0x0C */
    U16                     IOCStatus;                      /* 0x0E */
    U32                     IOCLogInfo;                     /* 0x10 */
    U16                     Event;                          /* 0x14 */
    U16                     Reserved4;                      /* 0x16 */
    U32                     EventContext;                   /* 0x18 */
    U32                     EventData[1];                   /* 0x1C */
} MPI2_EVENT_NOTIFICATION_REPLY, MPI2_POINTER PTR_MPI2_EVENT_NOTIFICATION_REPLY,
  Mpi2EventNotificationReply_t, MPI2_POINTER pMpi2EventNotificationReply_t;

/* AckRequired */
#define MPI2_EVENT_NOTIFICATION_ACK_NOT_REQUIRED    (0x00)
#define MPI2_EVENT_NOTIFICATION_ACK_REQUIRED        (0x01)
```



```
/* Event */
#define MPI2_EVENT_LOG_DATA                         (0x0001)
#define MPI2_EVENT_STATE_CHANGE                     (0x0002)
#define MPI2_EVENT_HARD_RESET_RECEIVED              (0x0005)
#define MPI2_EVENT_EVENT_CHANGE                     (0x000A)
#define MPI2_EVENT_TASK_SET_FULL                    (0x000E) /* obsolete */
#define MPI2_EVENT_TASK_SET_FULL                    (0x000E)
#define MPI2_EVENT_SAS_DEVICE_STATUS_CHANGE         (0x000F)
#define MPI2_EVENT_IR_OPERATION_STATUS              (0x0014)
#define MPI2_EVENT_SAS_DISCOVERY                    (0x0016)
#define MPI2_EVENT_SAS_BROADCAST_PRIMITIVE          (0x0017)
#define MPI2_EVENT_SAS_INIT_DEVICE_STATUS_CHANGE    (0x0018)
#define MPI2_EVENT_SAS_INIT_TABLE_OVERFLOW          (0x0019)
#define MPI2_EVENT_SAS_TOPOLOGY_CHANGE_LIST         (0x001C)
#define MPI2_EVENT_SAS_ENCL_DEVICE_STATUS_CHANGE    (0x001D)
#define MPI2_EVENT_IR_VOLUME                        (0x001E)
#define MPI2_EVENT_IR_PHYSICAL_DISK                 (0x001F)
#define MPI2_EVENT_IR_CONFIGURATION_CHANGE_LIST     (0x0020)
#define MPI2_EVENT_LOG_ENTRY_ADDED                  (0x0021)
#define MPI2_EVENT_SAS_PHY_COUNTER                  (0x0022)
#define MPI2_EVENT_GPIO_INTERRUPT                   (0x0023)
#define MPI2_EVENT_HOST_BASED_DISCOVERY_PHY         (0x0024)
#define MPI2_EVENT_SAS_QUIESCE                      (0x0025)
#endif /* ! codereview */


/* Log Entry Added Event data */

/* the following structure matches MPI2_LOG_0_ENTRY in mpi2_cnfg.h */
#define MPI2_EVENT_DATA_LOG_DATA_LENGTH             (0x1C)

typedef struct _MPI2_EVENT_DATA_LOG_ENTRY_ADDED
{
    U64         TimeStamp;                              /* 0x00 */
    U32         Reserved1;                              /* 0x08 */
    U16         LogSequence;                            /* 0x0C */
    U16         LogEntryQualifier;                      /* 0x0E */
    U8          VP_ID;                                  /* 0x10 */
    U8          VF_ID;                                  /* 0x11 */
    U16         Reserved2;                              /* 0x12 */
    U8          LogData[MPI2_EVENT_DATA_LOG_DATA_LENGTH];/* 0x14 */
} MPI2_EVENT_DATA_LOG_ENTRY_ADDED,
  MPI2_POINTER PTR_MPI2_EVENT_DATA_LOG_ENTRY_ADDED,
  Mpi2EventDataLogEntryAdded_t, MPI2_POINTER pMpi2EventDataLogEntryAdded_t;

/* GPIO Interrupt Event data */

typedef struct _MPI2_EVENT_DATA_GPIO_INTERRUPT
{
    U8          GPIONum;                                /* 0x00 */
    U8          Reserved1;                              /* 0x01 */
    U16         Reserved2;                              /* 0x02 */
} MPI2_EVENT_DATA_GPIO_INTERRUPT,
  MPI2_POINTER PTR_MPI2_EVENT_DATA_GPIO_INTERRUPT,
  Mpi2EventDataGpioInterrupt_t, MPI2_POINTER pMpi2EventDataGpioInterrupt_t;

/* Hard Reset Received Event data */

typedef struct _MPI2_EVENT_DATA_HARD_RESET_RECEIVED
{
    U8                      Reserved1;                      /* 0x00 */
    U8                      Port;                           /* 0x01 */
    U16                     Reserved2;                      /* 0x02 */
} MPI2_EVENT_DATA_HARD_RESET_RECEIVED,
  MPI2_POINTER PTR_MPI2_EVENT_DATA_HARD_RESET_RECEIVED,
  Mpi2EventDataHardResetReceived_t,
```

```
574   MPI2_POINTER pMpi2EventDataHardResetReceived_t;

576 /* Task Set Full Event data */
577 /*   this event is obsolete */
578 #endif /* ! codereview */

580 typedef struct _MPI2_EVENT_DATA_TASK_SET_FULL
581 {
582     U16                     DevHandle;                  /* 0x00 */
583     U16                     CurrentDepth;               /* 0x02 */
584 } MPI2_EVENT_DATA_TASK_SET_FULL, MPI2_POINTER PTR_MPI2_EVENT_DATA_TASK_SET_FULL,
585   Mpi2EventDataTaskSetFull_t, MPI2_POINTER pMpi2EventDataTaskSetFull_t;


588 /* SAS Device Status Change Event data */

590 typedef struct _MPI2_EVENT_DATA_SAS_DEVICE_STATUS_CHANGE
591 {
592     U16                     TaskTag;                    /* 0x00 */
593     U8                      ReasonCode;                 /* 0x02 */
594     U8                      Reserved1;                  /* 0x03 */
595     U8                      ASC;                        /* 0x04 */
596     U8                      ASCQ;                       /* 0x05 */
597     U16                     DevHandle;                  /* 0x06 */
598     U32                     Reserved2;                  /* 0x08 */
599     U64                     SASAddress;                 /* 0x0C */
600     U8                      LUN[8];                     /* 0x14 */
601 } MPI2_EVENT_DATA_SAS_DEVICE_STATUS_CHANGE,
602   MPI2_POINTER PTR_MPI2_EVENT_DATA_SAS_DEVICE_STATUS_CHANGE,
603   Mpi2EventDataSasDeviceStatusChange_t,
604   MPI2_POINTER pMpi2EventDataSasDeviceStatusChange_t;

606 /* SAS Device Status Change Event data ReasonCode values */
607 #define MPI2_EVENT_SAS_DEV_STAT_RC_SMART_DATA                   (0x05)
608 #define MPI2_EVENT_SAS_DEV_STAT_RC_UNSUPPORTED                  (0x07)
609 #define MPI2_EVENT_SAS_DEV_STAT_RC_INTERNAL_DEVICE_RESET        (0x08)
610 #define MPI2_EVENT_SAS_DEV_STAT_RC_TASK_ABORT_INTERNAL          (0x09)
611 #define MPI2_EVENT_SAS_DEV_STAT_RC_ABORT_TASK_SET_INTERNAL      (0x0A)
612 #define MPI2_EVENT_SAS_DEV_STAT_RC_CLEAR_TASK_SET_INTERNAL      (0x0B)
613 #define MPI2_EVENT_SAS_DEV_STAT_RC_QUERY_TASK_INTERNAL          (0x0C)
614 #define MPI2_EVENT_SAS_DEV_STAT_RC_ASYNC_NOTIFICATION           (0x0D)
615 #define MPI2_EVENT_SAS_DEV_STAT_RC_CMP_INTERNAL_DEV_RESET       (0x0E)
616 #define MPI2_EVENT_SAS_DEV_STAT_RC_CMP_TASK_ABORT_INTERNAL      (0x0F)
617 #define MPI2_EVENT_SAS_DEV_STAT_RC_SATA_INIT_FAILURE            (0x10)
618 #define MPI2_EVENT_SAS_DEV_STAT_RC_EXPANDER_REDUCED_FUNCTIONALITY       (0x11)
619 #define MPI2_EVENT_SAS_DEV_STAT_RC_CMP_EXPANDER_REDUCED_FUNCTIONALITY   (0x12)


622 /* Integrated RAID Operation Status Event data */

624 typedef struct _MPI2_EVENT_DATA_IR_OPERATION_STATUS
625 {
626     U16                     VolDevHandle;               /* 0x00 */
627     U16                     Reserved1;                  /* 0x02 */
628     U8                      RAIDOperation;              /* 0x04 */
629     U8                      PercentComplete;            /* 0x05 */
630     U16                     Reserved2;                  /* 0x06 */
631     U32                     Resereved3;                 /* 0x08 */
632 } MPI2_EVENT_DATA_IR_OPERATION_STATUS,
633   MPI2_POINTER PTR_MPI2_EVENT_DATA_IR_OPERATION_STATUS,
634   Mpi2EventDataIrOperationStatus_t,
635   MPI2_POINTER pMpi2EventDataIrOperationStatus_t;

637 /* Integrated RAID Operation Status Event data RAIDOperation values */
638 #define MPI2_EVENT_IR_RAIDOP_RESYNC                     (0x00)
639 #define MPI2_EVENT_IR_RAIDOP_ONLINE_CAP_EXPANSION       (0x01)
```



```
640 #define MPI2_EVENT_IR_RAIDOP_CONSISTENCY_CHECK          (0x02)
641 #define MPI2_EVENT_IR_RAIDOP_BACKGROUND_INIT            (0x03)
642 #define MPI2_EVENT_IR_RAIDOP_MAKE_DATA_CONSISTENT       (0x04)


645 /* Integrated RAID Volume Event data */

647 typedef struct _MPI2_EVENT_DATA_IR_VOLUME
648 {
649     U16                     VolDevHandle;               /* 0x00 */
650     U8                      ReasonCode;                 /* 0x02 */
651     U8                      Reserved1;                  /* 0x03 */
652     U32                     NewValue;                   /* 0x04 */
653     U32                     PreviousValue;              /* 0x08 */
654 } MPI2_EVENT_DATA_IR_VOLUME, MPI2_POINTER PTR_MPI2_EVENT_DATA_IR_VOLUME,
655   Mpi2EventDataIrVolume_t, MPI2_POINTER pMpi2EventDataIrVolume_t;

657 /* Integrated RAID Volume Event data ReasonCode values */
658 #define MPI2_EVENT_IR_VOLUME_RC_SETTINGS_CHANGED        (0x01)
659 #define MPI2_EVENT_IR_VOLUME_RC_STATUS_FLAGS_CHANGED    (0x02)
660 #define MPI2_EVENT_IR_VOLUME_RC_STATE_CHANGED           (0x03)


663 /* Integrated RAID Physical Disk Event data */

665 typedef struct _MPI2_EVENT_DATA_IR_PHYSICAL_DISK
666 {
667     U16                     Reserved1;                  /* 0x00 */
668     U8                      ReasonCode;                 /* 0x02 */
669     U8                      PhysDiskNum;                /* 0x03 */
670     U16                     PhysDiskDevHandle;          /* 0x04 */
671     U16                     Reserved2;                  /* 0x06 */
672     U16                     Slot;                       /* 0x08 */
673     U16                     EnclosureHandle;            /* 0x0A */
674     U32                     NewValue;                   /* 0x0C */
675     U32                     PreviousValue;              /* 0x10 */
676 } MPI2_EVENT_DATA_IR_PHYSICAL_DISK,
677   MPI2_POINTER PTR_MPI2_EVENT_DATA_IR_PHYSICAL_DISK,
678   Mpi2EventDataIrPhysicalDisk_t, MPI2_POINTER pMpi2EventDataIrPhysicalDisk_t;

680 /* Integrated RAID Physical Disk Event data ReasonCode values */
681 #define MPI2_EVENT_IR_PHYSDISK_RC_SETTINGS_CHANGED      (0x01)
682 #define MPI2_EVENT_IR_PHYSDISK_RC_STATUS_FLAGS_CHANGED  (0x02)
683 #define MPI2_EVENT_IR_PHYSDISK_RC_STATE_CHANGED         (0x03)


686 /* Integrated RAID Configuration Change List Event data */

688 /*
689  * Host code (drivers, BIOS, utilities, etc.) should leave this define set to
690  * one and check NumElements at runtime.
691  */
692 #ifndef MPI2_EVENT_IR_CONFIG_ELEMENT_COUNT
693 #define MPI2_EVENT_IR_CONFIG_ELEMENT_COUNT          (1)
694 #endif

696 typedef struct _MPI2_EVENT_IR_CONFIG_ELEMENT
697 {
698     U16                     ElementFlags;               /* 0x00 */
699     U16                     VolDevHandle;               /* 0x02 */
700     U8                      ReasonCode;                 /* 0x04 */
701     U8                      PhysDiskNum;                /* 0x05 */
702     U16                     PhysDiskDevHandle;          /* 0x06 */
703 } MPI2_EVENT_IR_CONFIG_ELEMENT, MPI2_POINTER PTR_MPI2_EVENT_IR_CONFIG_ELEMENT,
704   Mpi2EventIrConfigElement_t, MPI2_POINTER pMpi2EventIrConfigElement_t;
```

```
706 /* IR Configuration Change List Event data ElementFlags values */
707 #define MPI2_EVENT_IR_CHANGE_EFLAGS_ELEMENT_TYPE_MASK   (0x000F)
708 #define MPI2_EVENT_IR_CHANGE_EFLAGS_VOLUME_ELEMENT      (0x0000)
709 #define MPI2_EVENT_IR_CHANGE_EFLAGS_VOLPHYSDISK_ELEMENT (0x0001)
710 #define MPI2_EVENT_IR_CHANGE_EFLAGS_HOTSPARE_ELEMENT    (0x0002)

712 /* IR Configuration Change List Event data ReasonCode values */
713 #define MPI2_EVENT_IR_CHANGE_RC_ADDED                   (0x01)
714 #define MPI2_EVENT_IR_CHANGE_RC_REMOVED                 (0x02)
715 #define MPI2_EVENT_IR_CHANGE_RC_NO_CHANGE               (0x03)
716 #define MPI2_EVENT_IR_CHANGE_RC_HIDE                    (0x04)
717 #define MPI2_EVENT_IR_CHANGE_RC_UNHIDE                  (0x05)
718 #define MPI2_EVENT_IR_CHANGE_RC_VOLUME_CREATED          (0x06)
719 #define MPI2_EVENT_IR_CHANGE_RC_VOLUME_DELETED          (0x07)
720 #define MPI2_EVENT_IR_CHANGE_RC_PD_CREATED              (0x08)
721 #define MPI2_EVENT_IR_CHANGE_RC_PD_DELETED              (0x09)

723 typedef struct _MPI2_EVENT_DATA_IR_CONFIG_CHANGE_LIST
724 {
725     U8                              NumElements;        /* 0x00 */
726     U8                              Reserved1;          /* 0x01 */
727     U8                              Reserved2;          /* 0x02 */
728     U8                              ConfigNum;          /* 0x03 */
729     U32                             Flags;              /* 0x04 */
730     MPI2_EVENT_IR_CONFIG_ELEMENT    ConfigElement[MPI2_EVENT_IR_CONFIG_ELEMENT_C
731 } MPI2_EVENT_DATA_IR_CONFIG_CHANGE_LIST,
732   MPI2_POINTER PTR_MPI2_EVENT_DATA_IR_CONFIG_CHANGE_LIST,
733   Mpi2EventDataIrConfigChangeList_t,
734   MPI2_POINTER pMpi2EventDataIrConfigChangeList_t;

736 /* IR Configuration Change List Event data Flags values */
737 #define MPI2_EVENT_IR_CHANGE_FLAGS_FOREIGN_CONFIG   (0x00000001)


740 /* SAS Discovery Event data */

742 typedef struct _MPI2_EVENT_DATA_SAS_DISCOVERY
743 {
744     U8                      Flags;                  /* 0x00 */
745     U8                      ReasonCode;             /* 0x01 */
746     U8                      PhysicalPort;           /* 0x02 */
747     U8                      Reserved1;              /* 0x03 */
748     U32                     DiscoveryStatus;        /* 0x04 */
749 } MPI2_EVENT_DATA_SAS_DISCOVERY,
750   MPI2_POINTER PTR_MPI2_EVENT_DATA_SAS_DISCOVERY,
751   Mpi2EventDataSasDiscovery_t, MPI2_POINTER pMpi2EventDataSasDiscovery_t;

753 /* SAS Discovery Event data Flags values */
754 #define MPI2_EVENT_SAS_DISC_DEVICE_CHANGE               (0x02)
755 #define MPI2_EVENT_SAS_DISC_IN_PROGRESS                 (0x01)

757 /* SAS Discovery Event data ReasonCode values */
758 #define MPI2_EVENT_SAS_DISC_RC_STARTED                  (0x01)
759 #define MPI2_EVENT_SAS_DISC_RC_COMPLETED                (0x02)

761 /* SAS Discovery Event data DiscoveryStatus values */
762 #define MPI2_EVENT_SAS_DISC_DS_MAX_ENCLOSURES_EXCEED        (0x80000000)
763 #define MPI2_EVENT_SAS_DISC_DS_MAX_EXPANDERS_EXCEED         (0x40000000)
764 #define MPI2_EVENT_SAS_DISC_DS_MAX_DEVICES_EXCEED           (0x20000000)
765 #define MPI2_EVENT_SAS_DISC_DS_MAX_TOPO_PHYS_EXCEED         (0x10000000)
766 #define MPI2_EVENT_SAS_DISC_DS_DOWNSTREAM_INITIATOR         (0x08000000)
767 #define MPI2_EVENT_SAS_DISC_DS_MULTI_SUBTRACTIVE_SUBTRACTIVE    (0x00008000)
768 #define MPI2_EVENT_SAS_DISC_DS_EXP_MULTI_SUBTRACTIVE        (0x00004000)
769 #define MPI2_EVENT_SAS_DISC_DS_MULTI_PORT_DOMAIN            (0x00002000)
770 #define MPI2_EVENT_SAS_DISC_DS_TABLE_TO_SUBTRACTIVE_LINK    (0x00001000)
771 #define MPI2_EVENT_SAS_DISC_DS_UNSUPPORTED_DEVICE           (0x00000800)
```



```
772 #define MPI2_EVENT_SAS_DISC_DS_TABLE_LINK                   (0x00000400)
773 #define MPI2_EVENT_SAS_DISC_DS_SUBTRACTIVE_LINK             (0x00000200)
774 #define MPI2_EVENT_SAS_DISC_DS_SMP_CRC_ERROR                (0x00000100)
775 #define MPI2_EVENT_SAS_DISC_DS_SMP_FUNCTION_FAILED          (0x00000080)
776 #define MPI2_EVENT_SAS_DISC_DS_INDEX_NOT_EXIST              (0x00000040)
777 #define MPI2_EVENT_SAS_DISC_DS_OUT_ROUTE_ENTRIES            (0x00000020)
778 #define MPI2_EVENT_SAS_DISC_DS_SMP_TIMEOUT                  (0x00000010)
779 #define MPI2_EVENT_SAS_DISC_DS_MULTIPLE_PORTS               (0x00000004)
780 #define MPI2_EVENT_SAS_DISC_DS_UNADDRESSABLE_DEVICE         (0x00000002)
781 #define MPI2_EVENT_SAS_DISC_DS_LOOP_DETECTED                (0x00000001)


784 /* SAS Broadcast Primitive Event data */

786 typedef struct _MPI2_EVENT_DATA_SAS_BROADCAST_PRIMITIVE
787 {
788     U8                      PhyNum;                 /* 0x00 */
789     U8                      Port;                   /* 0x01 */
790     U8                      PortWidth;              /* 0x02 */
791     U8                      Primitive;              /* 0x03 */
792 } MPI2_EVENT_DATA_SAS_BROADCAST_PRIMITIVE,
793   MPI2_POINTER PTR_MPI2_EVENT_DATA_SAS_BROADCAST_PRIMITIVE,
794   Mpi2EventDataSasBroadcastPrimitive_t,
795   MPI2_POINTER pMpi2EventDataSasBroadcastPrimitive_t;

797 /* defines for the Primitive field */
798 #define MPI2_EVENT_PRIMITIVE_CHANGE                         (0x01)
799 #define MPI2_EVENT_PRIMITIVE_SES                            (0x02)
800 #define MPI2_EVENT_PRIMITIVE_EXPANDER                       (0x03)
801 #define MPI2_EVENT_PRIMITIVE_ASYNCHRONOUS_EVENT             (0x04)
802 #define MPI2_EVENT_PRIMITIVE_RESERVED3                      (0x05)
803 #define MPI2_EVENT_PRIMITIVE_RESERVED4                      (0x06)
804 #define MPI2_EVENT_PRIMITIVE_CHANGE0_RESERVED               (0x07)
805 #define MPI2_EVENT_PRIMITIVE_CHANGE1_RESERVED               (0x08)


808 /* SAS Initiator Device Status Change Event data */

810 typedef struct _MPI2_EVENT_DATA_SAS_INIT_DEV_STATUS_CHANGE
811 {
812     U8                      ReasonCode;             /* 0x00 */
813     U8                      PhysicalPort;           /* 0x01 */
814     U16                     DevHandle;              /* 0x02 */
815     U64                     SASAddress;             /* 0x04 */
816 } MPI2_EVENT_DATA_SAS_INIT_DEV_STATUS_CHANGE,
817   MPI2_POINTER PTR_MPI2_EVENT_DATA_SAS_INIT_DEV_STATUS_CHANGE,
818   Mpi2EventDataSasInitDevStatusChange_t,
819   MPI2_POINTER pMpi2EventDataSasInitDevStatusChange_t;

821 /* SAS Initiator Device Status Change event ReasonCode values */
822 #define MPI2_EVENT_SAS_INIT_RC_ADDED                (0x01)
823 #define MPI2_EVENT_SAS_INIT_RC_NOT_RESPONDING       (0x02)


826 /* SAS Initiator Device Table Overflow Event data */

828 typedef struct _MPI2_EVENT_DATA_SAS_INIT_TABLE_OVERFLOW
829 {
830     U16                     MaxInit;                /* 0x00 */
831     U16                     CurrentInit;            /* 0x02 */
832     U64                     SASAddress;             /* 0x04 */
833 } MPI2_EVENT_DATA_SAS_INIT_TABLE_OVERFLOW,
834   MPI2_POINTER PTR_MPI2_EVENT_DATA_SAS_INIT_TABLE_OVERFLOW,
835   Mpi2EventDataSasInitTableOverflow_t,
836   MPI2_POINTER pMpi2EventDataSasInitTableOverflow_t;
```

```
/* SAS Topology Change List Event data */

/*
 * Host code (drivers, BIOS, utilities, etc.) should leave this define set to
 * one and check NumEntries at runtime.
 */
#ifndef MPI2_EVENT_SAS_TOPO_PHY_COUNT
#define MPI2_EVENT_SAS_TOPO_PHY_COUNT           (1)
#endif

typedef struct _MPI2_EVENT_SAS_TOPO_PHY_ENTRY
{
    U16                     AttachedDevHandle;          /* 0x00 */
    U8                      LinkRate;                   /* 0x02 */
    U8                      PhyStatus;                  /* 0x03 */
} MPI2_EVENT_SAS_TOPO_PHY_ENTRY, MPI2_POINTER PTR_MPI2_EVENT_SAS_TOPO_PHY_ENTRY,
  Mpi2EventSasTopoPhyEntry_t, MPI2_POINTER pMpi2EventSasTopoPhyEntry_t;

typedef struct _MPI2_EVENT_DATA_SAS_TOPOLOGY_CHANGE_LIST
{
    U16                             EnclosureHandle;        /* 0x00 */
    U16                             ExpanderDevHandle;      /* 0x02 */
    U8                              NumPhys;                /* 0x04 */
    U8                              Reserved1;              /* 0x05 */
    U16                             Reserved2;              /* 0x06 */
    U8                              NumEntries;             /* 0x08 */
    U8                              StartPhyNum;            /* 0x09 */
    U8                              ExpStatus;              /* 0x0A */
    U8                              PhysicalPort;           /* 0x0B */
    MPI2_EVENT_SAS_TOPO_PHY_ENTRY   PHY[MPI2_EVENT_SAS_TOPO_PHY_COUNT]; /* 0x0C*
} MPI2_EVENT_DATA_SAS_TOPOLOGY_CHANGE_LIST,
  MPI2_POINTER PTR_MPI2_EVENT_DATA_SAS_TOPOLOGY_CHANGE_LIST,
  Mpi2EventDataSasTopologyChangeList_t,
  MPI2_POINTER pMpi2EventDataSasTopologyChangeList_t;

/* values for the ExpStatus field */
#define MPI2_EVENT_SAS_TOPO_ES_NO_EXPANDER                  (0x00)
#endif /* ! codereview */
#define MPI2_EVENT_SAS_TOPO_ES_ADDED                        (0x01)
#define MPI2_EVENT_SAS_TOPO_ES_NOT_RESPONDING               (0x02)
#define MPI2_EVENT_SAS_TOPO_ES_RESPONDING                   (0x03)
#define MPI2_EVENT_SAS_TOPO_ES_DELAY_NOT_RESPONDING         (0x04)

/* defines for the LinkRate field */
#define MPI2_EVENT_SAS_TOPO_LR_CURRENT_MASK                 (0xF0)
#define MPI2_EVENT_SAS_TOPO_LR_CURRENT_SHIFT                (4)
#define MPI2_EVENT_SAS_TOPO_LR_PREV_MASK                    (0x0F)
#define MPI2_EVENT_SAS_TOPO_LR_PREV_SHIFT                   (0)

#define MPI2_EVENT_SAS_TOPO_LR_UNKNOWN_LINK_RATE            (0x00)
#define MPI2_EVENT_SAS_TOPO_LR_PHY_DISABLED                 (0x01)
#define MPI2_EVENT_SAS_TOPO_LR_NEGOTIATION_FAILED           (0x02)
#define MPI2_EVENT_SAS_TOPO_LR_SATA_OOB_COMPLETE            (0x03)
#define MPI2_EVENT_SAS_TOPO_LR_PORT_SELECTOR                (0x04)
#define MPI2_EVENT_SAS_TOPO_LR_SMP_RESET_IN_PROGRESS        (0x05)
#define MPI2_EVENT_SAS_TOPO_LR_UNSUPPORTED_PHY              (0x06)
#endif /* ! codereview */
#define MPI2_EVENT_SAS_TOPO_LR_RATE_1_5                     (0x08)
#define MPI2_EVENT_SAS_TOPO_LR_RATE_3_0                     (0x09)
#define MPI2_EVENT_SAS_TOPO_LR_RATE_6_0                     (0x0A)
#define MPI25_EVENT_SAS_TOPO_LR_RATE_12_0                   (0x0B)
#endif /* ! codereview */

/* values for the PhyStatus field */
#define MPI2_EVENT_SAS_TOPO_PHYSTATUS_VACANT                (0x80)
```



```
#define MPI2_EVENT_SAS_TOPO_PS_MULTIPLEX_CHANGE             (0x10)
/* values for the PhyStatus ReasonCode sub-field */
#define MPI2_EVENT_SAS_TOPO_RC_MASK                         (0x0F)
#define MPI2_EVENT_SAS_TOPO_RC_TARG_ADDED                   (0x01)
#define MPI2_EVENT_SAS_TOPO_RC_TARG_NOT_RESPONDING          (0x02)
#define MPI2_EVENT_SAS_TOPO_RC_PHY_CHANGED                  (0x03)
#define MPI2_EVENT_SAS_TOPO_RC_NO_CHANGE                    (0x04)
#define MPI2_EVENT_SAS_TOPO_RC_DELAY_NOT_RESPONDING         (0x05)


/* SAS Enclosure Device Status Change Event data */

typedef struct _MPI2_EVENT_DATA_SAS_ENCL_DEV_STATUS_CHANGE
{
    U16                     EnclosureHandle;            /* 0x00 */
    U8                      ReasonCode;                 /* 0x02 */
    U8                      PhysicalPort;               /* 0x03 */
    U64                     EnclosureLogicalID;         /* 0x04 */
    U16                     NumSlots;                   /* 0x0C */
    U16                     StartSlot;                  /* 0x0E */
    U32                     PhyBits;                    /* 0x10 */
} MPI2_EVENT_DATA_SAS_ENCL_DEV_STATUS_CHANGE,
  MPI2_POINTER PTR_MPI2_EVENT_DATA_SAS_ENCL_DEV_STATUS_CHANGE,
  Mpi2EventDataSasEnclDevStatusChange_t,
  MPI2_POINTER pMpi2EventDataSasEnclDevStatusChange_t;

/* SAS Enclosure Device Status Change event ReasonCode values */
#define MPI2_EVENT_SAS_ENCL_RC_ADDED                (0x01)
#define MPI2_EVENT_SAS_ENCL_RC_NOT_RESPONDING       (0x02)


/* SAS PHY Counter Event data */

typedef struct _MPI2_EVENT_DATA_SAS_PHY_COUNTER
{
    U64         TimeStamp;          /* 0x00 */
    U32         Reserved1;          /* 0x08 */
    U8          PhyEventCode;       /* 0x0C */
    U8          PhyNum;             /* 0x0D */
    U16         Reserved2;          /* 0x0E */
    U32         PhyEventInfo;       /* 0x10 */
    U8          CounterType;        /* 0x14 */
    U8          ThresholdWindow;    /* 0x15 */
    U8          TimeUnits;          /* 0x16 */
    U8          Reserved3;          /* 0x17 */
    U32         EventThreshold;     /* 0x18 */
    U16         ThresholdFlags;     /* 0x1C */
    U16         Reserved4;          /* 0x1E */
} MPI2_EVENT_DATA_SAS_PHY_COUNTER,
  MPI2_POINTER PTR_MPI2_EVENT_DATA_SAS_PHY_COUNTER,
  Mpi2EventDataSasPhyCounter_t, MPI2_POINTER pMpi2EventDataSasPhyCounter_t;

/* use MPI2_SASPHY3_EVENT_CODE_ values from mpi2_cnfg.h for the PhyEventCode fie

/* use MPI2_SASPHY3_COUNTER_TYPE_ values from mpi2_cnfg.h for the CounterType fi

/* use MPI2_SASPHY3_TIME_UNITS_ values from mpi2_cnfg.h for the TimeUnits field

/* use MPI2_SASPHY3_TFLAGS_ values from mpi2_cnfg.h for the ThresholdFlags field


/* SAS Quiesce Event data */

typedef struct _MPI2_EVENT_DATA_SAS_QUIESCE
{
    U8                      ReasonCode;                 /* 0x00 */
```

```
    U8                      Reserved1;                  /* 0x01 */
    U16                     Reserved2;                  /* 0x02 */
    U32                     Reserved3;                  /* 0x04 */
} MPI2_EVENT_DATA_SAS_QUIESCE,
  MPI2_POINTER PTR_MPI2_EVENT_DATA_SAS_QUIESCE,
  Mpi2EventDataSasQuiesce_t, MPI2_POINTER pMpi2EventDataSasQuiesce_t;

/* SAS Quiesce Event data ReasonCode values */
#define MPI2_EVENT_SAS_QUIESCE_RC_STARTED                   (0x01)
#define MPI2_EVENT_SAS_QUIESCE_RC_COMPLETED                 (0x02)


/* Host Based Discovery Phy Event data */

typedef struct _MPI2_EVENT_HBD_PHY_SAS
{
    U8          Flags;                      /* 0x00 */
    U8          NegotiatedLinkRate;         /* 0x01 */
    U8          PhyNum;                     /* 0x02 */
    U8          PhysicalPort;               /* 0x03 */
    U32         Reserved1;                  /* 0x04 */
    U8          InitialFrame[28];           /* 0x08 */
} MPI2_EVENT_HBD_PHY_SAS, MPI2_POINTER PTR_MPI2_EVENT_HBD_PHY_SAS,
  Mpi2EventHbdPhySas_t, MPI2_POINTER pMpi2EventHbdPhySas_t;

/* values for the Flags field */
#define MPI2_EVENT_HBD_SAS_FLAGS_FRAME_VALID        (0x02)
#define MPI2_EVENT_HBD_SAS_FLAGS_SATA_FRAME         (0x01)

/* use MPI2_SAS_NEG_LINK_RATE_ defines from mpi2_cnfg.h for the NegotiatedLinkRa

typedef union _MPI2_EVENT_HBD_DESCRIPTOR
{
    MPI2_EVENT_HBD_PHY_SAS      Sas;
} MPI2_EVENT_HBD_DESCRIPTOR, MPI2_POINTER PTR_MPI2_EVENT_HBD_DESCRIPTOR,
  Mpi2EventHbdDescriptor_t, MPI2_POINTER pMpi2EventHbdDescriptor_t;

typedef struct _MPI2_EVENT_DATA_HBD_PHY
{
    U8                          DescriptorType;     /* 0x00 */
    U8                          Reserved1;          /* 0x01 */
    U16                         Reserved2;          /* 0x02 */
    U32                         Reserved3;          /* 0x04 */
    MPI2_EVENT_HBD_DESCRIPTOR   Descriptor;         /* 0x08 */
} MPI2_EVENT_DATA_HBD_PHY, MPI2_POINTER PTR_MPI2_EVENT_DATA_HBD_PHY,
  Mpi2EventDataHbdPhy_t, MPI2_POINTER pMpi2EventDataMpi2EventDataHbdPhy_t;

/* values for the DescriptorType field */
#define MPI2_EVENT_HBD_DT_SAS               (0x01)



#endif /* ! codereview */
/*****************************************************************************
*  EventAck message
*****************************************************************************/

/* EventAck Request message */
typedef struct _MPI2_EVENT_ACK_REQUEST
{
    U16                     Reserved1;                  /* 0x00 */
    U8                      ChainOffset;                /* 0x02 */
    U8                      Function;                   /* 0x03 */
    U16                     Reserved2;                  /* 0x04 */
    U8                      Reserved3;                  /* 0x06 */
    U8                      MsgFlags;                   /* 0x07 */
```



```
    U8                      VP_ID;                      /* 0x08 */
    U8                      VF_ID;                      /* 0x09 */
    U16                     Reserved4;                  /* 0x0A */
    U16                     Event;                      /* 0x0C */
    U16                     Reserved5;                  /* 0x0E */
    U32                     EventContext;               /* 0x10 */
} MPI2_EVENT_ACK_REQUEST, MPI2_POINTER PTR_MPI2_EVENT_ACK_REQUEST,
  Mpi2EventAckRequest_t, MPI2_POINTER pMpi2EventAckRequest_t;


/* EventAck Reply message */
typedef struct _MPI2_EVENT_ACK_REPLY
{
    U16                     Reserved1;                  /* 0x00 */
    U8                      MsgLength;                  /* 0x02 */
    U8                      Function;                   /* 0x03 */
    U16                     Reserved2;                  /* 0x04 */
    U8                      Reserved3;                  /* 0x06 */
    U8                      MsgFlags;                   /* 0x07 */
    U8                      VP_ID;                      /* 0x08 */
    U8                      VF_ID;                      /* 0x09 */
    U16                     Reserved4;                  /* 0x0A */
    U16                     Reserved5;                  /* 0x0C */
    U16                     IOCStatus;                  /* 0x0E */
    U32                     IOCLogInfo;                 /* 0x10 */
} MPI2_EVENT_ACK_REPLY, MPI2_POINTER PTR_MPI2_EVENT_ACK_REPLY,
  Mpi2EventAckReply_t, MPI2_POINTER pMpi2EventAckReply_t;


/*****************************************************************************
*  FWDownload message
*****************************************************************************/

/* MPI v2.0 FWDownload Request message */
/* FWDownload Request message */
typedef struct _MPI2_FW_DOWNLOAD_REQUEST
{
    U8                      ImageType;                  /* 0x00 */
    U8                      Reserved1;                  /* 0x01 */
    U8                      ChainOffset;                /* 0x02 */
    U8                      Function;                   /* 0x03 */
    U16                     Reserved2;                  /* 0x04 */
    U8                      Reserved3;                  /* 0x06 */
    U8                      MsgFlags;                   /* 0x07 */
    U8                      VP_ID;                      /* 0x08 */
    U8                      VF_ID;                      /* 0x09 */
    U16                     Reserved4;                  /* 0x0A */
    U32                     TotalImageSize;             /* 0x0C */
    U32                     Reserved5;                  /* 0x10 */
    MPI2_MPI_SGE_UNION      SGL;                        /* 0x14 */
} MPI2_FW_DOWNLOAD_REQUEST, MPI2_POINTER PTR_MPI2_FW_DOWNLOAD_REQUEST,
  Mpi2FWDownloadRequest, MPI2_POINTER pMpi2FWDownloadRequest;

#define MPI2_FW_DOWNLOAD_MSGFLGS_LAST_SEGMENT   (0x01)

#define MPI2_FW_DOWNLOAD_ITYPE_FW                   (0x01)
#define MPI2_FW_DOWNLOAD_ITYPE_BIOS                 (0x02)
#define MPI2_FW_DOWNLOAD_ITYPE_MANUFACTURING        (0x06)
#define MPI2_FW_DOWNLOAD_ITYPE_CONFIG_1             (0x07)
#define MPI2_FW_DOWNLOAD_ITYPE_CONFIG_2             (0x08)
#define MPI2_FW_DOWNLOAD_ITYPE_MEGARAID             (0x09)
#define MPI2_FW_DOWNLOAD_ITYPE_COMPLETE             (0x0A)
#endif /* ! codereview */
#define MPI2_FW_DOWNLOAD_ITYPE_COMMON_BOOT_BLOCK    (0x0B)
#define MPI2_FW_DOWNLOAD_ITYPE_MIN_PRODUCT_SPECIFIC (0xF0)
#endif /* ! codereview */
```

```
1102 /* MPI v2.0 FWDownload TransactionContext Element */
 327 /* FWDownload TransactionContext Element */
1103 typedef struct _MPI2_FW_DOWNLOAD_TCSGE
1104 {
1105     U8                      Reserved1;              /* 0x00 */
1106     U8                      ContextSize;            /* 0x01 */
1107     U8                      DetailsLength;          /* 0x02 */
1108     U8                      Flags;                  /* 0x03 */
1109     U32                     Reserved2;              /* 0x04 */
1110     U32                     ImageOffset;            /* 0x08 */
1111     U32                     ImageSize;              /* 0x0C */
1112 } MPI2_FW_DOWNLOAD_TCSGE, MPI2_POINTER PTR_MPI2_FW_DOWNLOAD_TCSGE,
1113   Mpi2FWDownloadTCSGE_t, MPI2_POINTER pMpi2FWDownloadTCSGE_t;


1116 /* MPI v2.5 FWDownload Request message */
1117 typedef struct _MPI25_FW_DOWNLOAD_REQUEST
1118 {
1119     U8                      ImageType;              /* 0x00 */
1120     U8                      Reserved1;              /* 0x01 */
1121     U8                      ChainOffset;            /* 0x02 */
1122     U8                      Function;               /* 0x03 */
1123     U16                     Reserved2;              /* 0x04 */
1124     U8                      Reserved3;              /* 0x06 */
1125     U8                      MsgFlags;               /* 0x07 */
1126     U8                      VP_ID;                  /* 0x08 */
1127     U8                      VF_ID;                  /* 0x09 */
1128     U16                     Reserved4;              /* 0x0A */
1129     U32                     TotalImageSize;         /* 0x0C */
1130     U32                     Reserved5;              /* 0x10 */
1131     U32                     Reserved6;              /* 0x14 */
1132     U32                     ImageOffset;            /* 0x18 */
1133     U32                     ImageSize;              /* 0x1C */
1134     MPI25_SGE_IO_UNION      SGL;                    /* 0x20 */
1135 } MPI25_FW_DOWNLOAD_REQUEST, MPI2_POINTER PTR_MPI25_FW_DOWNLOAD_REQUEST,
1136   Mpi25FWDownloadRequest, MPI2_POINTER pMpi25FWDownloadRequest;


1139 #endif /* ! codereview */
1140 /* FWDownload Reply message */
1141 typedef struct _MPI2_FW_DOWNLOAD_REPLY
1142 {
1143     U8                      ImageType;              /* 0x00 */
1144     U8                      Reserved1;              /* 0x01 */
1145     U8                      MsgLength;              /* 0x02 */
1146     U8                      Function;               /* 0x03 */
1147     U16                     Reserved2;              /* 0x04 */
1148     U8                      Reserved3;              /* 0x06 */
1149     U8                      MsgFlags;               /* 0x07 */
1150     U8                      VP_ID;                  /* 0x08 */
1151     U8                      VF_ID;                  /* 0x09 */
1152     U16                     Reserved4;              /* 0x0A */
1153     U16                     Reserved5;              /* 0x0C */
1154     U16                     IOCStatus;              /* 0x0E */
1155     U32                     IOCLogInfo;             /* 0x10 */
1156 } MPI2_FW_DOWNLOAD_REPLY, MPI2_POINTER PTR_MPI2_FW_DOWNLOAD_REPLY,
1157   Mpi2FWDownloadReply_t, MPI2_POINTER pMpi2FWDownloadReply_t;


1160 /*****************************************************************************
1161 *  FWUpload message
1162 *****************************************************************************/

1164 /* MPI v2.0 FWUpload Request message */
 340 /* FWUpload Request message */
```



```
1165 typedef struct _MPI2_FW_UPLOAD_REQUEST
1166 {
1167     U8                      ImageType;              /* 0x00 */
1168     U8                      Reserved1;              /* 0x01 */
1169     U8                      ChainOffset;            /* 0x02 */
1170     U8                      Function;               /* 0x03 */
1171     U16                     Reserved2;              /* 0x04 */
1172     U8                      Reserved3;              /* 0x06 */
1173     U8                      MsgFlags;               /* 0x07 */
1174     U8                      VP_ID;                  /* 0x08 */
1175     U8                      VF_ID;                  /* 0x09 */
1176     U16                     Reserved4;              /* 0x0A */
1177     U32                     Reserved5;              /* 0x0C */
1178     U32                     Reserved6;              /* 0x10 */
1179     MPI2_MPI_SGE_UNION      SGL;                    /* 0x14 */
1180 } MPI2_FW_UPLOAD_REQUEST, MPI2_POINTER PTR_MPI2_FW_UPLOAD_REQUEST,
1181   Mpi2FWUploadRequest_t, MPI2_POINTER pMpi2FWUploadRequest_t;

1183 #define MPI2_FW_UPLOAD_ITYPE_FW_CURRENT         (0x00)
1184 #define MPI2_FW_UPLOAD_ITYPE_FW_FLASH           (0x01)
1185 #define MPI2_FW_UPLOAD_ITYPE_BIOS_FLASH         (0x02)
1186 #define MPI2_FW_UPLOAD_ITYPE_FW_BACKUP          (0x05)
1187 #define MPI2_FW_UPLOAD_ITYPE_MANUFACTURING      (0x06)
1188 #define MPI2_FW_UPLOAD_ITYPE_CONFIG_1           (0x07)
1189 #define MPI2_FW_UPLOAD_ITYPE_CONFIG_2           (0x08)
1190 #define MPI2_FW_UPLOAD_ITYPE_MEGARAID           (0x09)
1191 #define MPI2_FW_UPLOAD_ITYPE_COMPLETE           (0x0A)
1192 #define MPI2_FW_UPLOAD_ITYPE_COMMON_BOOT_BLOCK  (0x0B)

1194 /* MPI v2.0 FWUpload TransactionContext Element */
1195 #endif /* ! codereview */
1196 typedef struct _MPI2_FW_UPLOAD_TCSGE
1197 {
1198     U8                      Reserved1;              /* 0x00 */
1199     U8                      ContextSize;            /* 0x01 */
1200     U8                      DetailsLength;          /* 0x02 */
1201     U8                      Flags;                  /* 0x03 */
1202     U32                     Reserved2;              /* 0x04 */
1203     U32                     ImageOffset;            /* 0x08 */
1204     U32                     ImageSize;              /* 0x0C */
1205 } MPI2_FW_UPLOAD_TCSGE, MPI2_POINTER PTR_MPI2_FW_UPLOAD_TCSGE,
1206   Mpi2FWUploadTCSGE_t, MPI2_POINTER pMpi2FWUploadTCSGE_t;


1209 /* MPI v2.5 FWUpload Request message */
1210 typedef struct _MPI25_FW_UPLOAD_REQUEST
1211 {
1212     U8                      ImageType;              /* 0x00 */
1213     U8                      Reserved1;              /* 0x01 */
1214     U8                      ChainOffset;            /* 0x02 */
1215     U8                      Function;               /* 0x03 */
1216     U16                     Reserved2;              /* 0x04 */
1217     U8                      Reserved3;              /* 0x06 */
1218     U8                      MsgFlags;               /* 0x07 */
1219     U8                      VP_ID;                  /* 0x08 */
1220     U8                      VF_ID;                  /* 0x09 */
1221     U16                     Reserved4;              /* 0x0A */
1222     U32                     Reserved5;              /* 0x0C */
1223     U32                     Reserved6;              /* 0x10 */
1224     U32                     Reserved7;              /* 0x14 */
1225     U32                     ImageOffset;            /* 0x18 */
1226     U32                     ImageSize;              /* 0x1C */
1227     MPI25_SGE_IO_UNION      SGL;                    /* 0x20 */
1228 } MPI25_FW_UPLOAD_REQUEST, MPI2_POINTER PTR_MPI25_FW_UPLOAD_REQUEST,
1229   Mpi25FWUploadRequest_t, MPI2_POINTER pMpi25FWUploadRequest_t;
```

```
#endif /* ! codereview */
/* FWUpload Reply message */
typedef struct _MPI2_FW_UPLOAD_REPLY
{
    U8                      ImageType;              /* 0x00 */
    U8                      Reserved1;              /* 0x01 */
    U8                      MsgLength;              /* 0x02 */
    U8                      Function;               /* 0x03 */
    U16                     Reserved2;              /* 0x04 */
    U8                      Reserved3;              /* 0x06 */
    U8                      MsgFlags;               /* 0x07 */
    U8                      VP_ID;                  /* 0x08 */
    U8                      VF_ID;                  /* 0x09 */
    U16                     Reserved4;              /* 0x0A */
    U16                     Reserved5;              /* 0x0C */
    U16                     IOCStatus;              /* 0x0E */
    U32                     IOCLogInfo;             /* 0x10 */
    U32                     ActualImageSize;        /* 0x14 */
} MPI2_FW_UPLOAD_REPLY, MPI2_POINTER PTR_MPI2_FW_UPLOAD_REPLY,
  Mpi2FWUploadReply_t, MPI2_POINTER pMPi2FWUploadReply_t;


/* FW Image Header */
typedef struct _MPI2_FW_IMAGE_HEADER
{
    U32                     Signature;              /* 0x00 */
    U32                     Signature0;             /* 0x04 */
    U32                     Signature1;             /* 0x08 */
    U32                     Signature2;             /* 0x0C */
    MPI2_VERSION_UNION      MPIVersion;             /* 0x10 */
    MPI2_VERSION_UNION      FWVersion;              /* 0x14 */
    MPI2_VERSION_UNION      NVDATAVersion;          /* 0x18 */
    MPI2_VERSION_UNION      PackageVersion;         /* 0x1C */
    U16                     VendorID;               /* 0x20 */
    U16                     ProductID;              /* 0x22 */
    U16                     ProtocolFlags;          /* 0x24 */
    U16                     Reserved26;             /* 0x26 */
    U32                     IOCCapabilities;        /* 0x28 */
    U32                     ImageSize;              /* 0x2C */
    U32                     NextImageHeaderOffset;  /* 0x30 */
    U32                     Checksum;               /* 0x34 */
    U32                     Reserved38;             /* 0x38 */
    U32                     Reserved3C;             /* 0x3C */
    U32                     Reserved40;             /* 0x40 */
    U32                     Reserved44;             /* 0x44 */
    U32                     Reserved48;             /* 0x48 */
    U32                     Reserved4C;             /* 0x4C */
    U32                     Reserved50;             /* 0x50 */
    U32                     Reserved54;             /* 0x54 */
    U32                     Reserved58;             /* 0x58 */
    U32                     Reserved5C;             /* 0x5C */
    U32                     Reserved60;             /* 0x60 */
    U32                     FirmwareVersionNameWhat;/* 0x64 */
    U8                      FirmwareVersionName[32];/* 0x68 */
    U32                     VendorNameWhat;         /* 0x88 */
    U8                      VendorName[32];         /* 0x8C */
    U32                     PackageNameWhat;        /* 0x88 */
    U8                      PackageName[32];        /* 0x8C */
    U32                     ReservedD0;             /* 0xD0 */
    U32                     ReservedD4;             /* 0xD4 */
    U32                     ReservedD8;             /* 0xD8 */
    U32                     ReservedDC;             /* 0xDC */
    U32                     ReservedE0;             /* 0xE0 */
    U32                     ReservedE4;             /* 0xE4 */
    U32                     ReservedE8;             /* 0xE8 */
```



```
    U32                     ReservedEC;             /* 0xEC */
    U32                     ReservedF0;             /* 0xF0 */
    U32                     ReservedF4;             /* 0xF4 */
    U32                     ReservedF8;             /* 0xF8 */
    U32                     ReservedFC;             /* 0xFC */
} MPI2_FW_IMAGE_HEADER, MPI2_POINTER PTR_MPI2_FW_IMAGE_HEADER,
  Mpi2FWImageHeader_t, MPI2_POINTER pMpi2FWImageHeader_t;

/* Signature field */
#define MPI2_FW_HEADER_SIGNATURE_OFFSET         (0x00)
#define MPI2_FW_HEADER_SIGNATURE_MASK           (0xFF000000)
#define MPI2_FW_HEADER_SIGNATURE                (0xEA000000)

/* Signature0 field */
#define MPI2_FW_HEADER_SIGNATURE0_OFFSET        (0x04)
#define MPI2_FW_HEADER_SIGNATURE0               (0x5AFAA55A)

/* Signature1 field */
#define MPI2_FW_HEADER_SIGNATURE1_OFFSET        (0x08)
#define MPI2_FW_HEADER_SIGNATURE1               (0xA55AFAA5)

/* Signature2 field */
#define MPI2_FW_HEADER_SIGNATURE2_OFFSET        (0x0C)
#define MPI2_FW_HEADER_SIGNATURE2               (0x5AA55AFA)


/* defines for using the ProductID field */
#define MPI2_FW_HEADER_PID_TYPE_MASK            (0xF000)
#define MPI2_FW_HEADER_PID_TYPE_SAS             (0x2000)

#define MPI2_FW_HEADER_PID_PROD_MASK                    (0x0F00)
#define MPI2_FW_HEADER_PID_PROD_A                       (0x0000)
#define MPI2_FW_HEADER_PID_PROD_TARGET_INITIATOR_SCSI   (0x0200)
#define MPI2_FW_HEADER_PID_PROD_IR_SCSI                 (0x0700)
#endif /* ! codereview */

#define MPI2_FW_HEADER_PID_FAMILY_MASK          (0x00FF)
/* SAS ProductID Family bits */
#define MPI2_FW_HEADER_PID_FAMILY_2108_SAS      (0x0013)
#define MPI2_FW_HEADER_PID_FAMILY_2208_SAS      (0x0014)
#define MPI25_FW_HEADER_PID_FAMILY_3108_SAS     (0x0021)
 370 /* SAS */
 371 #define MPI2_FW_HEADER_PID_FAMILY_2108_SAS      (0x0010)
 372 #define MPI2_FW_HEADER_PID_FAMILY_2208_SAS      (0x0011)

/* use MPI2_IOCFACTS_PROTOCOL_ defines for ProtocolFlags field */

/* use MPI2_IOCFACTS_CAPABILITY_ defines for IOCCapabilities field */


#define MPI2_FW_HEADER_IMAGESIZE_OFFSET         (0x2C)
#define MPI2_FW_HEADER_NEXTIMAGE_OFFSET         (0x30)
#define MPI2_FW_HEADER_VERNMHWAT_OFFSET         (0x64)

#define MPI2_FW_HEADER_WHAT_SIGNATURE           (0x29232840)

#define MPI2_FW_HEADER_SIZE                     (0x100)


/* Extended Image Header */
typedef struct _MPI2_EXT_IMAGE_HEADER

{
    U8                      ImageType;              /* 0x00 */
    U8                      Reserved1;              /* 0x01 */
    U16                     Reserved2;              /* 0x02 */
```

```
1360     U32                     Checksum;                   /* 0x04 */
1361     U32                     ImageSize;                  /* 0x08 */
1362     U32                     NextImageHeaderOffset;      /* 0x0C */
1363     U32                     PackageVersion;             /* 0x10 */
1364     U32                     Reserved3;                  /* 0x14 */
1365     U32                     Reserved4;                  /* 0x18 */
1366     U32                     Reserved5;                  /* 0x1C */
1367     U8                      IdentifyString[32];         /* 0x20 */
1368 } MPI2_EXT_IMAGE_HEADER, MPI2_POINTER PTR_MPI2_EXT_IMAGE_HEADER,
______unchanged_portion_omitted_
1513   Mpi2InitImageFooter_t, MPI2_POINTER pMpi2InitImageFooter_t;

1515 /* defines for the BootFlags field */
1516 #define MPI2_INIT_IMAGE_BOOTFLAGS_OFFSET        (0x00)

1518 /* defines for the ImageSize field */
1519 #define MPI2_INIT_IMAGE_IMAGESIZE_OFFSET        (0x04)

1521 /* defines for the Signature0 field */
1522 #define MPI2_INIT_IMAGE_SIGNATURE0_OFFSET       (0x08)
1523 #define MPI2_INIT_IMAGE_SIGNATURE0              (0x5AA55AEA)

1525 /* defines for the Signature1 field */
1526 #define MPI2_INIT_IMAGE_SIGNATURE1_OFFSET       (0x0C)
1527 #define MPI2_INIT_IMAGE_SIGNATURE1              (0xA55AEAA5)

1529 /* defines for the Signature2 field */
1530 #define MPI2_INIT_IMAGE_SIGNATURE2_OFFSET       (0x10)
1531 #define MPI2_INIT_IMAGE_SIGNATURE2              (0x5AEAA55A)

1533 /* Signature fields as individual bytes */
1534 #define MPI2_INIT_IMAGE_SIGNATURE_BYTE_0        (0xEA)
1535 #define MPI2_INIT_IMAGE_SIGNATURE_BYTE_1        (0x5A)
1536 #define MPI2_INIT_IMAGE_SIGNATURE_BYTE_2        (0xA5)
1537 #define MPI2_INIT_IMAGE_SIGNATURE_BYTE_3        (0x5A)

1539 #define MPI2_INIT_IMAGE_SIGNATURE_BYTE_4        (0xA5)
1540 #define MPI2_INIT_IMAGE_SIGNATURE_BYTE_5        (0xEA)
1541 #define MPI2_INIT_IMAGE_SIGNATURE_BYTE_6        (0x5A)
1542 #define MPI2_INIT_IMAGE_SIGNATURE_BYTE_7        (0xA5)

1544 #define MPI2_INIT_IMAGE_SIGNATURE_BYTE_8        (0x5A)
1545 #define MPI2_INIT_IMAGE_SIGNATURE_BYTE_9        (0xA5)
1546 #define MPI2_INIT_IMAGE_SIGNATURE_BYTE_A        (0xEA)
1547 #define MPI2_INIT_IMAGE_SIGNATURE_BYTE_B        (0x5A)

1549 /* defines for the ResetVector field */
1550 #define MPI2_INIT_IMAGE_RESETVECTOR_OFFSET      (0x14)


1553 /*****************************************************************************
1554 *  PowerManagementControl message
1555 *****************************************************************************/

1557 /* PowerManagementControl Request message */
1558 typedef struct _MPI2_PWR_MGMT_CONTROL_REQUEST
1559 {
1560     U8                      Feature;                    /* 0x00 */
1561     U8                      Reserved1;                  /* 0x01 */
1562     U8                      ChainOffset;                /* 0x02 */
1563     U8                      Function;                   /* 0x03 */
1564     U16                     Reserved2;                  /* 0x04 */
1565     U8                      Reserved3;                  /* 0x06 */
1566     U8                      MsgFlags;                   /* 0x07 */
1567     U8                      VP_ID;                      /* 0x08 */
1568     U8                      VF_ID;                      /* 0x09 */
```



```
1569     U16                     Reserved4;                  /* 0x0A */
1570     U8                      Parameter1;                 /* 0x0C */
1571     U8                      Parameter2;                 /* 0x0D */
1572     U8                      Parameter3;                 /* 0x0E */
1573     U8                      Parameter4;                 /* 0x0F */
1574     U32                     Reserved5;                  /* 0x10 */
1575     U32                     Reserved6;                  /* 0x14 */
1576 } MPI2_PWR_MGMT_CONTROL_REQUEST, MPI2_POINTER PTR_MPI2_PWR_MGMT_CONTROL_REQUEST,
1577   Mpi2PwrMgmtControlRequest_t, MPI2_POINTER pMpi2PwrMgmtControlRequest_t;

1579 /* defines for the Feature field */
1580 #define MPI2_PM_CONTROL_FEATURE_DA_PHY_POWER_COND       (0x01)
1581 #define MPI2_PM_CONTROL_FEATURE_PORT_WIDTH_MODULATION   (0x02)
1582 #define MPI2_PM_CONTROL_FEATURE_PCIE_LINK               (0x03)
1583 #define MPI2_PM_CONTROL_FEATURE_IOC_SPEED               (0x04)
1584 #define MPI2_PM_CONTROL_FEATURE_MIN_PRODUCT_SPECIFIC    (0x80)
1585 #define MPI2_PM_CONTROL_FEATURE_MAX_PRODUCT_SPECIFIC    (0xFF)

1587 /* parameter usage for the MPI2_PM_CONTROL_FEATURE_DA_PHY_POWER_COND Feature */
1588 /* Parameter1 contains a PHY number */
1589 /* Parameter2 indicates power condition action using these defines */
1590 #define MPI2_PM_CONTROL_PARAM2_PARTIAL                  (0x01)
1591 #define MPI2_PM_CONTROL_PARAM2_SLUMBER                  (0x02)
1592 #define MPI2_PM_CONTROL_PARAM2_EXIT_PWR_MGMT            (0x03)
1593 /* Parameter3 and Parameter4 are reserved */

1595 /* parameter usage for the MPI2_PM_CONTROL_FEATURE_PORT_WIDTH_MODULATION Feature
1596 /* Parameter1 contains SAS port width modulation group number */
1597 /* Parameter2 indicates IOC action using these defines */
1598 #define MPI2_PM_CONTROL_PARAM2_REQUEST_OWNERSHIP        (0x01)
1599 #define MPI2_PM_CONTROL_PARAM2_CHANGE_MODULATION        (0x02)
1600 #define MPI2_PM_CONTROL_PARAM2_RELINQUISH_OWNERSHIP     (0x03)
1601 /* Parameter3 indicates desired modulation level using these defines */
1602 #define MPI2_PM_CONTROL_PARAM3_25_PERCENT               (0x00)
1603 #define MPI2_PM_CONTROL_PARAM3_50_PERCENT               (0x01)
1604 #define MPI2_PM_CONTROL_PARAM3_75_PERCENT               (0x02)
1605 #define MPI2_PM_CONTROL_PARAM3_100_PERCENT              (0x03)
1606 /* Parameter4 is reserved */

1608 /* parameter usage for the MPI2_PM_CONTROL_FEATURE_PCIE_LINK Feature */
1609 /* Parameter1 indicates desired PCIe link speed using these defines */
1610 #define MPI2_PM_CONTROL_PARAM1_PCIE_2_5_GBPS            (0x00)
1611 #define MPI2_PM_CONTROL_PARAM1_PCIE_5_0_GBPS            (0x01)
1612 #define MPI2_PM_CONTROL_PARAM1_PCIE_8_0_GBPS            (0x02)
1613 /* Parameter2 indicates desired PCIe link width using these defines */
1614 #define MPI2_PM_CONTROL_PARAM2_WIDTH_X1                 (0x01)
1615 #define MPI2_PM_CONTROL_PARAM2_WIDTH_X2                 (0x02)
1616 #define MPI2_PM_CONTROL_PARAM2_WIDTH_X4                 (0x04)
1617 #define MPI2_PM_CONTROL_PARAM2_WIDTH_X8                 (0x08)
1618 /* Parameter3 and Parameter4 are reserved */

1620 /* parameter usage for the MPI2_PM_CONTROL_FEATURE_IOC_SPEED Feature */
1621 /* Parameter1 indicates desired IOC hardware clock speed using these defines */
1622 #define MPI2_PM_CONTROL_PARAM1_FULL_IOC_SPEED           (0x01)
1623 #define MPI2_PM_CONTROL_PARAM1_HALF_IOC_SPEED           (0x02)
1624 #define MPI2_PM_CONTROL_PARAM1_QUARTER_IOC_SPEED        (0x04)
1625 #define MPI2_PM_CONTROL_PARAM1_EIGHTH_IOC_SPEED         (0x08)
1626 /* Parameter2, Parameter3, and Parameter4 are reserved */


1629 /* PowerManagementControl Reply message */
1630 typedef struct _MPI2_PWR_MGMT_CONTROL_REPLY
1631 {
1632     U8                      Feature;                    /* 0x00 */
1633     U8                      Reserved1;                  /* 0x01 */
1634     U8                      MsgLength;                  /* 0x02 */
```

```
1635     U8                      Function;                   /* 0x03 */
1636     U16                     Reserved2;                  /* 0x04 */
1637     U8                      Reserved3;                  /* 0x06 */
1638     U8                      MsgFlags;                   /* 0x07 */
1639     U8                      VP_ID;                      /* 0x08 */
1640     U8                      VF_ID;                      /* 0x09 */
1641     U16                     Reserved4;                  /* 0x0A */
1642     U16                     Reserved5;                  /* 0x0C */
1643     U16                     IOCStatus;                  /* 0x0E */
1644     U32                     IOCLogInfo;                 /* 0x10 */
1645 } MPI2_PWR_MGMT_CONTROL_REPLY, MPI2_POINTER PTR_MPI2_PWR_MGMT_CONTROL_REPLY,
1646   Mpi2PwrMgmtControlReply_t, MPI2_POINTER pMpi2PwrMgmtControlReply_t;


1649 #endif /* ! codereview */
1650 #endif
```

```
**********************************************************
   16251 Thu Jun 12 17:42:18 2014
new/usr/src/uts/common/sys/scsi/adapters/mpt_sas3/mpi/mpi2_raid.h
Initial modifications using the code changes present between
the LSI source code for FreeBSD drivers. Specifically the changes
between from mpslsi-source-17.00.00.00 -> mpslsi-source-03.00.00.00.
This mainly involves using a different scatter/gather element in
frame setup.
**********************************************************
/*
 * 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) 2000-2012 LSI Corporation.
 * Copyright (c) 2000 to 2009, LSI Corporation.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms of all code within
 * this file that is exclusively owned by LSI, with or without
 * modification, is permitted provided that, in addition to the CDDL 1.0
 * License requirements, the following conditions are met:
 *
 *    Neither the name of the author nor the names of its contributors may be
 *    used to endorse or promote products derived from this software without
 *    specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS 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
 * COPYRIGHT OWNER 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.
 */

/*
 *          Name:  mpi2_raid.h
 *         Title:  MPI Integrated RAID messages and structures
 *  Creation Date:  April 26, 2007
 *
 *    mpi2_raid.h Version:  02.00.05
 *    mpi2_raid.h Version:  02.00.04
 *
```



```
 *  Version History
 *  ---------------
 *
 *  Date      Version   Description
 *  --------  --------  ------------------------------------------------------
 *  04-30-07  02.00.00  Corresponds to Fusion-MPT MPI Specification Rev A.
 *  08-31-07  02.00.01  Modifications to RAID Action request and reply,
 *                      including the Actions and ActionData.
 *  02-29-08  02.00.02  Added MPI2_RAID_ACTION_ADATA_DISABL_FULL_REBUILD.
 *  05-21-08  02.00.03  Added MPI2_RAID_VOL_CREATION_NUM_PHYSDISKS so that
 *                      the PhysDisk array in MPI2_RAID_VOLUME_CREATION_STRUCT
 *                      can be sized by the build environment.
 *  07-30-09  02.00.04  Added proper define for the Use Default Settings bit of
 *                      VolumeCreationFlags and marked the old one as obsolete.
 *  05-12-10  02.00.05  Added MPI2_RAID_VOL_FLAGS_OP_MDC define.
#endif /* ! codereview */
 *  --------------------------------------------------------------------------
 */

#ifndef MPI2_RAID_H
#define MPI2_RAID_H

/*****************************************************************************
*
*               Integrated RAID Messages
*
*****************************************************************************/

/*****************************************************************************
*  RAID Action messages
*****************************************************************************/

/* ActionDataWord defines for use with MPI2_RAID_ACTION_DELETE_VOLUME action */
#define MPI2_RAID_ACTION_ADATA_KEEP_LBA0            (0x00000000)
#define MPI2_RAID_ACTION_ADATA_ZERO_LBA0            (0x00000001)

/* use MPI2_RAIDVOL0_SETTING_ defines from mpi2_cnfg.h for MPI2_RAID_ACTION_CHAN

/* ActionDataWord defines for use with MPI2_RAID_ACTION_DISABLE_ALL_VOLUMES acti
#define MPI2_RAID_ACTION_ADATA_DISABL_FULL_REBUILD  (0x00000001)

/* ActionDataWord for MPI2_RAID_ACTION_SET_RAID_FUNCTION_RATE Action */
typedef struct _MPI2_RAID_ACTION_RATE_DATA
{
    U8              RateToChange;               /* 0x00 */
    U8              RateOrMode;                 /* 0x01 */
    U16             DataScrubDuration;          /* 0x02 */
} MPI2_RAID_ACTION_RATE_DATA, MPI2_POINTER PTR_MPI2_RAID_ACTION_RATE_DATA,
  Mpi2RaidActionRateData_t, MPI2_POINTER pMpi2RaidActionRateData_t;

#define MPI2_RAID_ACTION_SET_RATE_RESYNC            (0x00)
#define MPI2_RAID_ACTION_SET_RATE_DATA_SCRUB        (0x01)
#define MPI2_RAID_ACTION_SET_RATE_POWERSAVE_MODE    (0x02)

/* ActionDataWord for MPI2_RAID_ACTION_START_RAID_FUNCTION Action */
typedef struct _MPI2_RAID_ACTION_START_RAID_FUNCTION
{
    U8              RAIDFunction;                       /* 0x00 */
    U8              Flags;                              /* 0x01 */
    U16             Reserved1;                          /* 0x02 */
} MPI2_RAID_ACTION_START_RAID_FUNCTION,
  MPI2_POINTER PTR_MPI2_RAID_ACTION_START_RAID_FUNCTION,
  Mpi2RaidActionStartRaidFunction_t,
  MPI2_POINTER pMpi2RaidActionStartRaidFunction_t;

/* defines for the RAIDFunction field */
```

```
#define MPI2_RAID_ACTION_START_BACKGROUND_INIT      (0x00)
#define MPI2_RAID_ACTION_START_ONLINE_CAP_EXPANSION (0x01)
#define MPI2_RAID_ACTION_START_CONSISTENCY_CHECK    (0x02)

/* defines for the Flags field */
#define MPI2_RAID_ACTION_START_NEW                  (0x00)
#define MPI2_RAID_ACTION_START_RESUME               (0x01)

/* ActionDataWord for MPI2_RAID_ACTION_STOP_RAID_FUNCTION Action */
typedef struct _MPI2_RAID_ACTION_STOP_RAID_FUNCTION
{
    U8              RAIDFunction;                       /* 0x00 */
    U8              Flags;                              /* 0x01 */
    U16             Reserved1;                          /* 0x02 */
} MPI2_RAID_ACTION_STOP_RAID_FUNCTION,
  MPI2_POINTER PTR_MPI2_RAID_ACTION_STOP_RAID_FUNCTION,
  Mpi2RaidActionStopRaidFunction_t,
  MPI2_POINTER pMpi2RaidActionStopRaidFunction_t;

/* defines for the RAIDFunction field */
#define MPI2_RAID_ACTION_STOP_BACKGROUND_INIT       (0x00)
#define MPI2_RAID_ACTION_STOP_ONLINE_CAP_EXPANSION  (0x01)
#define MPI2_RAID_ACTION_STOP_CONSISTENCY_CHECK     (0x02)

/* defines for the Flags field */
#define MPI2_RAID_ACTION_STOP_ABORT                 (0x00)
#define MPI2_RAID_ACTION_STOP_PAUSE                 (0x01)

/* ActionDataWord for MPI2_RAID_ACTION_CREATE_HOT_SPARE Action */
typedef struct _MPI2_RAID_ACTION_HOT_SPARE
{
    U8              HotSparePool;               /* 0x00 */
    U8              Reserved1;                  /* 0x01 */
    U16             DevHandle;                  /* 0x02 */
} MPI2_RAID_ACTION_HOT_SPARE, MPI2_POINTER PTR_MPI2_RAID_ACTION_HOT_SPARE,
  Mpi2RaidActionHotSpare_t, MPI2_POINTER pMpi2RaidActionHotSpare_t;

/* ActionDataWord for MPI2_RAID_ACTION_DEVICE_FW_UPDATE_MODE Action */
typedef struct _MPI2_RAID_ACTION_FW_UPDATE_MODE
{
    U8              Flags;                              /* 0x00 */
    U8              DeviceFirmwareUpdateModeTimeout;    /* 0x01 */
    U16             Reserved1;                          /* 0x02 */
} MPI2_RAID_ACTION_FW_UPDATE_MODE,
  MPI2_POINTER PTR_MPI2_RAID_ACTION_FW_UPDATE_MODE,
  Mpi2RaidActionFwUpdateMode_t, MPI2_POINTER pMpi2RaidActionFwUpdateMode_t;

/* ActionDataWord defines for use with MPI2_RAID_ACTION_DEVICE_FW_UPDATE_MODE ac
#define MPI2_RAID_ACTION_ADATA_DISABLE_FW_UPDATE        (0x00)
#define MPI2_RAID_ACTION_ADATA_ENABLE_FW_UPDATE         (0x01)

typedef union _MPI2_RAID_ACTION_DATA
{
    U32                                 Word;
    MPI2_RAID_ACTION_RATE_DATA          Rates;
    MPI2_RAID_ACTION_START_RAID_FUNCTION    StartRaidFunction;
    MPI2_RAID_ACTION_STOP_RAID_FUNCTION     StopRaidFunction;
    MPI2_RAID_ACTION_HOT_SPARE          HotSpare;
    MPI2_RAID_ACTION_FW_UPDATE_MODE     FwUpdateMode;
} MPI2_RAID_ACTION_DATA, MPI2_POINTER PTR_MPI2_RAID_ACTION_DATA,
  Mpi2RaidActionData_t, MPI2_POINTER pMpi2RaidActionData_t;


/* RAID Action Request Message */
typedef struct _MPI2_RAID_ACTION_REQUEST
{
```



```
    U8                      Action;                     /* 0x00 */
    U8                      Reserved1;                  /* 0x01 */
    U8                      ChainOffset;                /* 0x02 */
    U8                      Function;                   /* 0x03 */
    U16                     VolDevHandle;               /* 0x04 */
    U8                      PhysDiskNum;                /* 0x06 */
    U8                      MsgFlags;                   /* 0x07 */
    U8                      VP_ID;                      /* 0x08 */
    U8                      VF_ID;                      /* 0x09 */
    U16                     Reserved2;                  /* 0x0A */
    U32                     Reserved3;                  /* 0x0C */
    MPI2_RAID_ACTION_DATA   ActionDataWord;             /* 0x10 */
    MPI2_SGE_SIMPLE_UNION   ActionDataSGE;              /* 0x14 */
} MPI2_RAID_ACTION_REQUEST, MPI2_POINTER PTR_MPI2_RAID_ACTION_REQUEST,
  Mpi2RaidActionRequest_t, MPI2_POINTER pMpi2RaidActionRequest_t;

/* RAID Action request Action values */

#define MPI2_RAID_ACTION_INDICATOR_STRUCT           (0x01)
#define MPI2_RAID_ACTION_CREATE_VOLUME              (0x02)
#define MPI2_RAID_ACTION_DELETE_VOLUME              (0x03)
#define MPI2_RAID_ACTION_DISABLE_ALL_VOLUMES        (0x04)
#define MPI2_RAID_ACTION_ENABLE_ALL_VOLUMES         (0x05)
#define MPI2_RAID_ACTION_PHYSDISK_OFFLINE           (0x0A)
#define MPI2_RAID_ACTION_PHYSDISK_ONLINE            (0x0B)
#define MPI2_RAID_ACTION_FAIL_PHYSDISK              (0x0F)
#define MPI2_RAID_ACTION_ACTIVATE_VOLUME            (0x11)
#define MPI2_RAID_ACTION_DEVICE_FW_UPDATE_MODE      (0x15)
#define MPI2_RAID_ACTION_CHANGE_VOL_WRITE_CACHE     (0x17)
#define MPI2_RAID_ACTION_SET_VOLUME_NAME            (0x18)
#define MPI2_RAID_ACTION_SET_RAID_FUNCTION_RATE     (0x19)
#define MPI2_RAID_ACTION_ENABLE_FAILED_VOLUME       (0x1C)
#define MPI2_RAID_ACTION_CREATE_HOT_SPARE           (0x1D)
#define MPI2_RAID_ACTION_DELETE_HOT_SPARE           (0x1E)
#define MPI2_RAID_ACTION_SYSTEM_SHUTDOWN_INITIATED  (0x20)
#define MPI2_RAID_ACTION_START_RAID_FUNCTION        (0x21)
#define MPI2_RAID_ACTION_STOP_RAID_FUNCTION         (0x22)
#define MPI2_RAID_ACTION_FAST_PATH_PERMITTED        (0x24)

/* RAID Volume Creation Structure */

/*
 * The following define can be customized for the targeted product.
 */
#ifndef MPI2_RAID_VOL_CREATION_NUM_PHYSDISKS
#define MPI2_RAID_VOL_CREATION_NUM_PHYSDISKS        (1)
#endif

typedef struct _MPI2_RAID_VOLUME_PHYSDISK
{
    U8                      RAIDSetNum;                 /* 0x00 */
    U8                      PhysDiskMap;                /* 0x01 */
    U16                     PhysDiskDevHandle;          /* 0x02 */
} MPI2_RAID_VOLUME_PHYSDISK, MPI2_POINTER PTR_MPI2_RAID_VOLUME_PHYSDISK,
_____unchanged_portion_omitted_
  Mpi2RaidVolIndicator_t, MPI2_POINTER pMpi2RaidVolIndicator_t;

/* defines for RAID Volume Indicator Flags field */
#define MPI2_RAID_VOL_FLAGS_OP_MASK                 (0x0000000F)
#define MPI2_RAID_VOL_FLAGS_OP_BACKGROUND_INIT      (0x00000000)
#define MPI2_RAID_VOL_FLAGS_OP_ONLINE_CAP_EXPANSION (0x00000001)
#define MPI2_RAID_VOL_FLAGS_OP_CONSISTENCY_CHECK    (0x00000002)
#define MPI2_RAID_VOL_FLAGS_OP_RESYNC               (0x00000003)
#define MPI2_RAID_VOL_FLAGS_OP_MDC                  (0x00000004)
#endif /* ! codereview */
```

```
313 /* RAID Action Reply ActionData union */
314 typedef union _MPI2_RAID_ACTION_REPLY_DATA
315 {
316     U32                     Word[5];
317     MPI2_RAID_VOL_INDICATOR RaidVolumeIndicator;
318     U16                     VolDevHandle;
319     U8                      VolumeState;
320     U8                      PhysDiskNum;
321 } MPI2_RAID_ACTION_REPLY_DATA, MPI2_POINTER PTR_MPI2_RAID_ACTION_REPLY_DATA,
322   Mpi2RaidActionReplyData_t, MPI2_POINTER pMpi2RaidActionReplyData_t;

324 /* use MPI2_RAIDVOL0_SETTING_ defines from mpi2_cnfg.h for MPI2_RAID_ACTION_CHAN


327 /* RAID Action Reply Message */
328 typedef struct _MPI2_RAID_ACTION_REPLY
329 {
330     U8                          Action;                 /* 0x00 */
331     U8                          Reserved1;              /* 0x01 */
332     U8                          MsgLength;              /* 0x02 */
333     U8                          Function;               /* 0x03 */
334     U16                         VolDevHandle;           /* 0x04 */
335     U8                          PhysDiskNum;            /* 0x06 */
336     U8                          MsgFlags;               /* 0x07 */
337     U8                          VP_ID;                  /* 0x08 */
338     U8                          VF_ID;                  /* 0x09 */
339     U16                         Reserved2;              /* 0x0A */
340     U16                         Reserved3;              /* 0x0C */
341     U16                         IOCStatus;              /* 0x0E */
342     U32                         IOCLogInfo;             /* 0x10 */
343     MPI2_RAID_ACTION_REPLY_DATA ActionData;             /* 0x14 */
344 } MPI2_RAID_ACTION_REPLY, MPI2_POINTER PTR_MPI2_RAID_ACTION_REPLY,
345   Mpi2RaidActionReply_t, MPI2_POINTER pMpi2RaidActionReply_t;


348 #endif
```

```
**********************************************************
   16332 Thu Jun 12 17:42:18 2014
new/usr/src/uts/common/sys/scsi/adapters/mpt_sas3/mpi/mpi2_sas.h
Initial modifications using the code changes present between
the LSI source code for FreeBSD drivers. Specifically the changes
between from mpslsi-source-17.00.00.00 -> mpslsi-source-03.00.00.00.
This mainly involves using a different scatter/gather element in
frame setup.
**********************************************************
/*
 * 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) 2000-2012 LSI Corporation.
 * Copyright (c) 2000 to 2009, LSI Corporation.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms of all code within
 * this file that is exclusively owned by LSI, with or without
 * modification, is permitted provided that, in addition to the CDDL 1.0
 * License requirements, the following conditions are met:
 *
 *    Neither the name of the author nor the names of its contributors may be
 *    used to endorse or promote products derived from this software without
 *    specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS 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
 * COPYRIGHT OWNER 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.
 */

/*
 *          Name:  mpi2_sas.h
 *         Title:  MPI Serial Attached SCSI structures and definitions
 *  Creation Date:  February 9, 2007
 *
 *  mpi2_sas.h Version:  02.00.xx
 *
 *  NOTE: Names (typedefs, defines, etc.) beginning with an MPI25 or Mpi25
```



```
 *        prefix are for use only on MPI v2.5 products, and must not be used
 *        with MPI v2.0 products. Unless otherwise noted, names beginning with
 *        MPI2 or Mpi2 are for use with both MPI v2.0 and MPI v2.5 products.
 *  mpi2.h Version:  02.00.02
 *
 *  Version History
 *  ---------------
 *
 *  Date      Version   Description
 *  --------  --------  ------------------------------------------------------
 *  04-30-07  02.00.00  Corresponds to Fusion-MPT MPI Specification Rev A.
 *  06-26-07  02.00.01  Added Clear All Persistent Operation to SAS IO Unit
 *                      Control Request.
 *  10-02-08  02.00.02  Added Set IOC Parameter Operation to SAS IO Unit Control
 *                      Request.
 *  10-28-09  02.00.03  Changed the type of SGL in MPI2_SATA_PASSTHROUGH_REQUEST
 *                      to MPI2_SGE_IO_UNION since it supports chained SGLs.
 *  05-12-10  02.00.04  Modified some comments.
 *  08-11-10  02.00.05  Added NCQ operations to SAS IO Unit Control.
#endif /* ! codereview */
 *  --------------------------------------------------------------------------
 */

#ifndef MPI2_SAS_H
#define MPI2_SAS_H

/*
 * Values for SASStatus.
 */
#define MPI2_SASSTATUS_SUCCESS                          (0x00)
#define MPI2_SASSTATUS_UNKNOWN_ERROR                    (0x01)
#define MPI2_SASSTATUS_INVALID_FRAME                    (0x02)
#define MPI2_SASSTATUS_UTC_BAD_DEST                     (0x03)
#define MPI2_SASSTATUS_UTC_BREAK_RECEIVED               (0x04)
#define MPI2_SASSTATUS_UTC_CONNECT_RATE_NOT_SUPPORTED   (0x05)
#define MPI2_SASSTATUS_UTC_PORT_LAYER_REQUEST           (0x06)
#define MPI2_SASSTATUS_UTC_PROTOCOL_NOT_SUPPORTED       (0x07)
#define MPI2_SASSTATUS_UTC_STP_RESOURCES_BUSY           (0x08)
#define MPI2_SASSTATUS_UTC_WRONG_DESTINATION            (0x09)
#define MPI2_SASSTATUS_SHORT_INFORMATION_UNIT           (0x0A)
#define MPI2_SASSTATUS_LONG_INFORMATION_UNIT            (0x0B)
#define MPI2_SASSTATUS_XFER_RDY_INCORRECT_WRITE_DATA    (0x0C)
#define MPI2_SASSTATUS_XFER_RDY_REQUEST_OFFSET_ERROR    (0x0D)
#define MPI2_SASSTATUS_XFER_RDY_NOT_EXPECTED            (0x0E)
#define MPI2_SASSTATUS_DATA_INCORRECT_DATA_LENGTH       (0x0F)
#define MPI2_SASSTATUS_DATA_TOO_MUCH_READ_DATA          (0x10)
#define MPI2_SASSTATUS_DATA_OFFSET_ERROR                (0x11)
#define MPI2_SASSTATUS_SDSF_NAK_RECEIVED                (0x12)
#define MPI2_SASSTATUS_SDSF_CONNECTION_FAILED           (0x13)
#define MPI2_SASSTATUS_INITIATOR_RESPONSE_TIMEOUT       (0x14)


/*
 * Values for the SAS DeviceInfo field used in SAS Device Status Change Event
 * data and SAS Configuration pages.
 */
#define MPI2_SAS_DEVICE_INFO_SEP                (0x00004000)
#define MPI2_SAS_DEVICE_INFO_ATAPI_DEVICE       (0x00002000)
#define MPI2_SAS_DEVICE_INFO_LSI_DEVICE         (0x00001000)
#define MPI2_SAS_DEVICE_INFO_DIRECT_ATTACH      (0x00000800)
#define MPI2_SAS_DEVICE_INFO_SSP_TARGET         (0x00000400)
#define MPI2_SAS_DEVICE_INFO_STP_TARGET         (0x00000200)
#define MPI2_SAS_DEVICE_INFO_SMP_TARGET         (0x00000100)
#define MPI2_SAS_DEVICE_INFO_SATA_DEVICE        (0x00000080)
#define MPI2_SAS_DEVICE_INFO_SSP_INITIATOR      (0x00000040)
#define MPI2_SAS_DEVICE_INFO_STP_INITIATOR      (0x00000020)
```

```
#define MPI2_SAS_DEVICE_INFO_SMP_INITIATOR      (0x00000010)
#define MPI2_SAS_DEVICE_INFO_SATA_HOST          (0x00000008)

#define MPI2_SAS_DEVICE_INFO_MASK_DEVICE_TYPE   (0x00000007)
#define MPI2_SAS_DEVICE_INFO_NO_DEVICE          (0x00000000)
#define MPI2_SAS_DEVICE_INFO_END_DEVICE         (0x00000001)
#define MPI2_SAS_DEVICE_INFO_EDGE_EXPANDER      (0x00000002)
#define MPI2_SAS_DEVICE_INFO_FANOUT_EXPANDER    (0x00000003)


/*****************************************************************************
*
*        SAS Messages
*
*****************************************************************************/

/*****************************************************************************
*  SMP Passthrough messages
*****************************************************************************/

/* SMP Passthrough Request Message */
typedef struct _MPI2_SMP_PASSTHROUGH_REQUEST
{
    U8                      PassthroughFlags;   /* 0x00 */
    U8                      PhysicalPort;       /* 0x01 */
    U8                      ChainOffset;        /* 0x02 */
    U8                      Function;           /* 0x03 */
    U16                     RequestDataLength;  /* 0x04 */
    U8                      SGLFlags;           /* 0x06 */ /* MPI v2.0 only. Res
    U8                      SGLFlags;           /* 0x06 */
    U8                      MsgFlags;           /* 0x07 */
    U8                      VP_ID;              /* 0x08 */
    U8                      VF_ID;              /* 0x09 */
    U16                     Reserved1;          /* 0x0A */
    U32                     Reserved2;          /* 0x0C */
    U64                     SASAddress;         /* 0x10 */
    U32                     Reserved3;          /* 0x18 */
    U32                     Reserved4;          /* 0x1C */
    MPI2_SIMPLE_SGE_UNION   SGL;                /* 0x20 */ /* MPI v2.5: IEEE Sim
    MPI2_SIMPLE_SGE_UNION   SGL;                /* 0x20 */
} MPI2_SMP_PASSTHROUGH_REQUEST, MPI2_POINTER PTR_MPI2_SMP_PASSTHROUGH_REQUEST,
  Mpi2SmpPassthroughRequest_t, MPI2_POINTER pMpi2SmpPassthroughRequest_t;

/* values for PassthroughFlags field */
#define MPI2_SMP_PT_REQ_PT_FLAGS_IMMEDIATE      (0x80)

/* MPI v2.0: use MPI2_SGLFLAGS_ defines from mpi2.h for the SGLFlags field */
/* values for SGLFlags field are in the SGL section of mpi2.h */


/* SMP Passthrough Reply Message */
typedef struct _MPI2_SMP_PASSTHROUGH_REPLY
{
    U8                      PassthroughFlags;   /* 0x00 */
    U8                      PhysicalPort;       /* 0x01 */
    U8                      MsgLength;          /* 0x02 */
    U8                      Function;           /* 0x03 */
    U16                     ResponseDataLength; /* 0x04 */
    U8                      SGLFlags;           /* 0x06 */
    U8                      MsgFlags;           /* 0x07 */
    U8                      VP_ID;              /* 0x08 */
    U8                      VF_ID;              /* 0x09 */
    U16                     Reserved1;          /* 0x0A */
    U8                      Reserved2;          /* 0x0C */
    U8                      SASStatus;          /* 0x0D */
    U16                     IOCStatus;          /* 0x0E */
```



```
    U32                     IOCLogInfo;         /* 0x10 */
    U32                     Reserved3;          /* 0x14 */
    U8                      ResponseData[4];    /* 0x18 */
} MPI2_SMP_PASSTHROUGH_REPLY, MPI2_POINTER PTR_MPI2_SMP_PASSTHROUGH_REPLY,
  Mpi2SmpPassthroughReply_t, MPI2_POINTER pMpi2SmpPassthroughReply_t;

/* values for PassthroughFlags field */
#define MPI2_SMP_PT_REPLY_PT_FLAGS_IMMEDIATE    (0x80)

/* values for SASStatus field are at the top of this file */


/*****************************************************************************
*  SATA Passthrough messages
*****************************************************************************/

/* SATA Passthrough Request Message */
typedef struct _MPI2_SATA_PASSTHROUGH_REQUEST
{
    U16                     DevHandle;          /* 0x00 */
    U8                      ChainOffset;        /* 0x02 */
    U8                      Function;           /* 0x03 */
    U16                     PassthroughFlags;   /* 0x04 */
    U8                      SGLFlags;           /* 0x06 */ /* MPI v2.0 only. Res
    U8                      SGLFlags;           /* 0x06 */
    U8                      MsgFlags;           /* 0x07 */
    U8                      VP_ID;              /* 0x08 */
    U8                      VF_ID;              /* 0x09 */
    U16                     Reserved1;          /* 0x0A */
    U32                     Reserved2;          /* 0x0C */
    U32                     Reserved3;          /* 0x10 */
    U32                     Reserved4;          /* 0x14 */
    U32                     DataLength;         /* 0x18 */
    U8                      CommandFIS[20];     /* 0x1C */
    MPI2_SGE_IO_UNION       SGL;                /* 0x30 */ /* MPI v2.5: IEEE 64
    MPI2_SIMPLE_SGE_UNION   SGL;                /* 0x20 */
} MPI2_SATA_PASSTHROUGH_REQUEST, MPI2_POINTER PTR_MPI2_SATA_PASSTHROUGH_REQUEST,
  Mpi2SataPassthroughRequest_t, MPI2_POINTER pMpi2SataPassthroughRequest_t;

/* values for PassthroughFlags field */
#define MPI2_SATA_PT_REQ_PT_FLAGS_EXECUTE_DIAG      (0x0100)
#define MPI2_SATA_PT_REQ_PT_FLAGS_DMA               (0x0020)
#define MPI2_SATA_PT_REQ_PT_FLAGS_PIO               (0x0010)
#define MPI2_SATA_PT_REQ_PT_FLAGS_UNSPECIFIED_VU    (0x0004)
#define MPI2_SATA_PT_REQ_PT_FLAGS_WRITE             (0x0002)
#define MPI2_SATA_PT_REQ_PT_FLAGS_READ              (0x0001)

/* MPI v2.0: use MPI2_SGLFLAGS_ defines from mpi2.h for the SGLFlags field */
/* values for SGLFlags field are in the SGL section of mpi2.h */


/* SATA Passthrough Reply Message */
typedef struct _MPI2_SATA_PASSTHROUGH_REPLY
{
    U16                     DevHandle;          /* 0x00 */
    U8                      MsgLength;          /* 0x02 */
    U8                      Function;           /* 0x03 */
    U16                     PassthroughFlags;   /* 0x04 */
    U8                      SGLFlags;           /* 0x06 */
    U8                      MsgFlags;           /* 0x07 */
    U8                      VP_ID;              /* 0x08 */
    U8                      VF_ID;              /* 0x09 */
    U16                     Reserved1;          /* 0x0A */
    U8                      Reserved2;          /* 0x0C */
    U8                      SASStatus;          /* 0x0D */
    U16                     IOCStatus;          /* 0x0E */
```

```
 247     U32                     IOCLogInfo;         /* 0x10 */
 248     U8                      StatusFIS[20];      /* 0x14 */
 249     U32                     StatusControlRegisters; /* 0x28 */
 250     U32                     TransferCount;      /* 0x2C */
 251 } MPI2_SATA_PASSTHROUGH_REPLY, MPI2_POINTER PTR_MPI2_SATA_PASSTHROUGH_REPLY,
_____unchanged_portion_omitted_
 286   MPI2_POINTER PTR_MPI2_SAS_IOUNIT_CONTROL_REQUEST,
 287   Mpi2SasIoUnitControlRequest_t, MPI2_POINTER pMpi2SasIoUnitControlRequest_t;

 289 /* values for the Operation field */
 290 #define MPI2_SAS_OP_CLEAR_ALL_PERSISTENT        (0x02)
 291 #define MPI2_SAS_OP_PHY_LINK_RESET              (0x06)
 292 #define MPI2_SAS_OP_PHY_HARD_RESET              (0x07)
 293 #define MPI2_SAS_OP_PHY_CLEAR_ERROR_LOG         (0x08)
 294 #define MPI2_SAS_OP_SEND_PRIMITIVE              (0x0A)
 295 #define MPI2_SAS_OP_FORCE_FULL_DISCOVERY        (0x0B)
 296 #define MPI2_SAS_OP_TRANSMIT_PORT_SELECT_SIGNAL (0x0C)
 297 #define MPI2_SAS_OP_REMOVE_DEVICE               (0x0D)
 298 #define MPI2_SAS_OP_LOOKUP_MAPPING              (0x0E)
 299 #define MPI2_SAS_OP_SET_IOC_PARAMETER           (0x0F)
 300 #define MPI25_SAS_OP_ENABLE_FP_DEVICE           (0x10)
 301 #define MPI25_SAS_OP_DISABLE_FP_DEVICE          (0x11)
 302 #define MPI25_SAS_OP_ENABLE_FP_ALL              (0x12)
 303 #define MPI25_SAS_OP_DISABLE_FP_ALL             (0x13)
 304 #define MPI2_SAS_OP_DEV_ENABLE_NCQ              (0x14)
 305 #define MPI2_SAS_OP_DEV_DISABLE_NCQ             (0x15)
 306 #endif /* ! codereview */
 307 #define MPI2_SAS_OP_PRODUCT_SPECIFIC_MIN        (0x80)

 309 /* values for the PrimFlags field */
 310 #define MPI2_SAS_PRIMFLAGS_SINGLE               (0x08)
 311 #define MPI2_SAS_PRIMFLAGS_TRIPLE               (0x02)
 312 #define MPI2_SAS_PRIMFLAGS_REDUNDANT            (0x01)

 314 /* values for the LookupMethod field */
 315 #define MPI2_SAS_LOOKUP_METHOD_SAS_ADDRESS          (0x01)
 316 #define MPI2_SAS_LOOKUP_METHOD_SAS_ENCLOSURE_SLOT   (0x02)
 317 #define MPI2_SAS_LOOKUP_METHOD_SAS_DEVICE_NAME      (0x03)


 320 /* SAS IO Unit Control Reply Message */
 321 typedef struct _MPI2_SAS_IOUNIT_CONTROL_REPLY
 322 {
 323     U8                      Operation;          /* 0x00 */
 324     U8                      Reserved1;          /* 0x01 */
 325     U8                      MsgLength;          /* 0x02 */
 326     U8                      Function;           /* 0x03 */
 327     U16                     DevHandle;          /* 0x04 */
 328     U8                      IOCParameter;       /* 0x06 */
 329     U8                      MsgFlags;           /* 0x07 */
 330     U8                      VP_ID;              /* 0x08 */
 331     U8                      VF_ID;              /* 0x09 */
 332     U16                     Reserved3;          /* 0x0A */
 333     U16                     Reserved4;          /* 0x0C */
 334     U16                     IOCStatus;          /* 0x0E */
 335     U32                     IOCLogInfo;         /* 0x10 */
 336 } MPI2_SAS_IOUNIT_CONTROL_REPLY,
 337   MPI2_POINTER PTR_MPI2_SAS_IOUNIT_CONTROL_REPLY,
 338   Mpi2SasIoUnitControlReply_t, MPI2_POINTER pMpi2SasIoUnitControlReply_t;


 341 #endif
```

```
**********************************************************
   24463 Thu Jun 12 17:42:18 2014
new/usr/src/uts/common/sys/scsi/adapters/mpt_sas3/mpi/mpi2_tool.h
Initial modifications using the code changes present between
the LSI source code for FreeBSD drivers. Specifically the changes
between from mpslsi-source-17.00.00.00 -> mpslsi-source-03.00.00.00.
This mainly involves using a different scatter/gather element in
frame setup.
**********************************************************
```

```
/*
 * 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) 2000-2012 LSI Corporation.
 * Copyright (c) 2000 to 2009, LSI Corporation.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms of all code within
 * this file that is exclusively owned by LSI, with or without
 * modification, is permitted provided that, in addition to the CDDL 1.0
 * License requirements, the following conditions are met:
 *
 *    Neither the name of the author nor the names of its contributors may be
 *    used to endorse or promote products derived from this software without
 *    specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS 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
 * COPYRIGHT OWNER 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.
 */

/*
 *          Name:  mpi2_tool.h
 *         Title:  MPI diagnostic tool structures and definitions
 *  Creation Date:  March 26, 2007
 *
 *    mpi2_tool.h Version:  02.00.06
 *    mpi2_tool.h Version:  02.00.04
 *
```



```
 *  Version History
 *  ---------------
 *
 *  Date      Version   Description
 *  --------  --------  ------------------------------------------------------
 *  04-30-07  02.00.00  Corresponds to Fusion-MPT MPI Specification Rev A.
 *  12-18-07  02.00.01  Added Diagnostic Buffer Post and Diagnostic Release
 *                      structures and defines.
 *  02-29-08  02.00.02  Modified various names to make them 32-character unique.
 *  05-06-09  02.00.03  Added ISTWI Read Write Tool and Diagnostic CLI Tool.
 *  07-30-09  02.00.04  Added ExtendedType field to DiagnosticBufferPost request
 *                      and reply messages.
 *                      Added MPI2_DIAG_BUF_TYPE_EXTENDED.
 *                      Incremented MPI2_DIAG_BUF_TYPE_COUNT.
 *  05-12-10  02.00.05  Added Diagnostic Data Upload tool.
 *  08-11-10  02.00.06  Added defines that were missing for Diagnostic Buffer
 *                      Post Request.
#endif /* ! codereview */
 *  ------------------------------------------------------------------------
 */

#ifndef MPI2_TOOL_H
#define MPI2_TOOL_H

/*****************************************************************************
*
*               Toolbox Messages
*
*****************************************************************************/

/* defines for the Tools */
#define MPI2_TOOLBOX_CLEAN_TOOL                     (0x00)
#define MPI2_TOOLBOX_MEMORY_MOVE_TOOL               (0x01)
#define MPI2_TOOLBOX_DIAG_DATA_UPLOAD_TOOL          (0x02)
#endif /* ! codereview */
#define MPI2_TOOLBOX_ISTWI_READ_WRITE_TOOL          (0x03)
#define MPI2_TOOLBOX_BEACON_TOOL                    (0x05)
#define MPI2_TOOLBOX_DIAGNOSTIC_CLI_TOOL            (0x06)


/*****************************************************************************
*  Toolbox reply
*****************************************************************************/

typedef struct _MPI2_TOOLBOX_REPLY
{
    U8                      Tool;                   /* 0x00 */
    U8                      Reserved1;              /* 0x01 */
    U8                      MsgLength;              /* 0x02 */
    U8                      Function;               /* 0x03 */
    U16                     Reserved2;              /* 0x04 */
    U8                      Reserved3;              /* 0x06 */
    U8                      MsgFlags;               /* 0x07 */
    U8                      VP_ID;                  /* 0x08 */
    U8                      VF_ID;                  /* 0x09 */
    U16                     Reserved4;              /* 0x0A */
    U16                     Reserved5;              /* 0x0C */
    U16                     IOCStatus;              /* 0x0E */
    U32                     IOCLogInfo;             /* 0x10 */
} MPI2_TOOLBOX_REPLY, MPI2_POINTER PTR_MPI2_TOOLBOX_REPLY,
  Mpi2ToolboxReply_t, MPI2_POINTER pMpi2ToolboxReply_t;


/*****************************************************************************
*  Toolbox Clean Tool request
*****************************************************************************/
```

```
typedef struct _MPI2_TOOLBOX_CLEAN_REQUEST
{
    U8                      Tool;                   /* 0x00 */
    U8                      Reserved1;              /* 0x01 */
    U8                      ChainOffset;            /* 0x02 */
    U8                      Function;               /* 0x03 */
    U16                     Reserved2;              /* 0x04 */
    U8                      Reserved3;              /* 0x06 */
    U8                      MsgFlags;               /* 0x07 */
    U8                      VP_ID;                  /* 0x08 */
    U8                      VF_ID;                  /* 0x09 */
    U16                     Reserved4;              /* 0x0A */
    U32                     Flags;                  /* 0x0C */
   } MPI2_TOOLBOX_CLEAN_REQUEST, MPI2_POINTER PTR_MPI2_TOOLBOX_CLEAN_REQUEST,
  Mpi2ToolboxCleanRequest_t, MPI2_POINTER pMpi2ToolboxCleanRequest_t;

/* values for the Flags field */
#define MPI2_TOOLBOX_CLEAN_BOOT_SERVICES            (0x80000000)
#define MPI2_TOOLBOX_CLEAN_PERSIST_MANUFACT_PAGES   (0x40000000)
#define MPI2_TOOLBOX_CLEAN_OTHER_PERSIST_PAGES      (0x20000000)
#define MPI2_TOOLBOX_CLEAN_FW_CURRENT               (0x10000000)
#define MPI2_TOOLBOX_CLEAN_FW_BACKUP                (0x08000000)
#define MPI2_TOOLBOX_CLEAN_MEGARAID                 (0x02000000)
#define MPI2_TOOLBOX_CLEAN_INITIALIZATION           (0x01000000)
#define MPI2_TOOLBOX_CLEAN_FLASH                    (0x00000004)
#define MPI2_TOOLBOX_CLEAN_SEEPROM                  (0x00000002)
#define MPI2_TOOLBOX_CLEAN_NVSRAM                   (0x00000001)


/*****************************************************************************
*  Toolbox Memory Move request
*****************************************************************************/

typedef struct _MPI2_TOOLBOX_MEM_MOVE_REQUEST
{
    U8                      Tool;                   /* 0x00 */
    U8                      Reserved1;              /* 0x01 */
    U8                      ChainOffset;            /* 0x02 */
    U8                      Function;               /* 0x03 */
    U16                     Reserved2;              /* 0x04 */
    U8                      Reserved3;              /* 0x06 */
    U8                      MsgFlags;               /* 0x07 */
    U8                      VP_ID;                  /* 0x08 */
    U8                      VF_ID;                  /* 0x09 */
    U16                     Reserved4;              /* 0x0A */
    MPI2_SGE_SIMPLE_UNION   SGL;                    /* 0x0C */
} MPI2_TOOLBOX_MEM_MOVE_REQUEST, MPI2_POINTER PTR_MPI2_TOOLBOX_MEM_MOVE_REQUEST,
  Mpi2ToolboxMemMoveRequest_t, MPI2_POINTER pMpi2ToolboxMemMoveRequest_t;


/*****************************************************************************
*  Toolbox Diagnostic Data Upload request
*****************************************************************************/

typedef struct _MPI2_TOOLBOX_DIAG_DATA_UPLOAD_REQUEST
{
    U8                      Tool;                   /* 0x00 */
    U8                      Reserved1;              /* 0x01 */
    U8                      ChainOffset;            /* 0x02 */
    U8                      Function;               /* 0x03 */
    U16                     Reserved2;              /* 0x04 */
    U8                      Reserved3;              /* 0x06 */
    U8                      MsgFlags;               /* 0x07 */
    U8                      VP_ID;                  /* 0x08 */
    U8                      VF_ID;                  /* 0x09 */
```



```
    U16                     Reserved4;              /* 0x0A */
    U8                      SGLFlags;               /* 0x0C */
    U8                      Reserved5;              /* 0x0D */
    U16                     Reserved6;              /* 0x0E */
    U32                     Flags;                  /* 0x10 */
    U32                     DataLength;             /* 0x14 */
    MPI2_SGE_SIMPLE_UNION   SGL;                    /* 0x18 */
} MPI2_TOOLBOX_DIAG_DATA_UPLOAD_REQUEST,
  MPI2_POINTER PTR_MPI2_TOOLBOX_DIAG_DATA_UPLOAD_REQUEST,
  Mpi2ToolboxDiagDataUploadRequest_t,
  MPI2_POINTER pMpi2ToolboxDiagDataUploadRequest_t;

/* use MPI2_SGLFLAGS_ defines from mpi2.h for the SGLFlags field */


typedef struct _MPI2_DIAG_DATA_UPLOAD_HEADER
{
    U32                     DiagDataLength;         /* 00h */
    U8                      FormatCode;             /* 04h */
    U8                      Reserved1;              /* 05h */
    U16                     Reserved2;              /* 06h */
} MPI2_DIAG_DATA_UPLOAD_HEADER, MPI2_POINTER PTR_MPI2_DIAG_DATA_UPLOAD_HEADER,
  Mpi2DiagDataUploadHeader_t, MPI2_POINTER pMpi2DiagDataUploadHeader_t;


/*****************************************************************************
#endif /* ! codereview */
*  Toolbox ISTWI Read Write Tool
*****************************************************************************/

/* Toolbox ISTWI Read Write Tool request message */
typedef struct _MPI2_TOOLBOX_ISTWI_READ_WRITE_REQUEST
{
    U8                      Tool;                   /* 0x00 */
    U8                      Reserved1;              /* 0x01 */
    U8                      ChainOffset;            /* 0x02 */
    U8                      Function;               /* 0x03 */
    U16                     Reserved2;              /* 0x04 */
    U8                      Reserved3;              /* 0x06 */
    U8                      MsgFlags;               /* 0x07 */
    U8                      VP_ID;                  /* 0x08 */
    U8                      VF_ID;                  /* 0x09 */
    U16                     Reserved4;              /* 0x0A */
    U32                     Reserved5;              /* 0x0C */
    U32                     Reserved6;              /* 0x10 */
    U8                      DevIndex;               /* 0x14 */
    U8                      Action;                 /* 0x15 */
    U8                      SGLFlags;               /* 0x16 */
    U8                      Reserved7;              /* 0x17 */
    U16                     TxDataLength;           /* 0x18 */
    U16                     RxDataLength;           /* 0x1A */
    U32                     Reserved8;              /* 0x1C */
    U32                     Reserved9;              /* 0x20 */
    U32                     Reserved10;             /* 0x24 */
    U32                     Reserved11;             /* 0x28 */
    U32                     Reserved12;             /* 0x2C */
    MPI2_SGE_SIMPLE_UNION   SGL;                    /* 0x30 */
} MPI2_TOOLBOX_ISTWI_READ_WRITE_REQUEST,
  MPI2_POINTER PTR_MPI2_TOOLBOX_ISTWI_READ_WRITE_REQUEST,
  Mpi2ToolboxIstwiReadWriteRequest_t,
  MPI2_POINTER pMpi2ToolboxIstwiReadWriteRequest_t;

/* values for the Action field */
#define MPI2_TOOL_ISTWI_ACTION_READ_DATA            (0x01)
#define MPI2_TOOL_ISTWI_ACTION_WRITE_DATA           (0x02)
#define MPI2_TOOL_ISTWI_ACTION_SEQUENCE             (0x03)
```

```
#define MPI2_TOOL_ISTWI_ACTION_RESERVE_BUS          (0x10)
#define MPI2_TOOL_ISTWI_ACTION_RELEASE_BUS          (0x11)
#define MPI2_TOOL_ISTWI_ACTION_RESET                (0x12)

/* values for SGLFlags field are in the SGL section of mpi2.h */


/* Toolbox ISTWI Read Write Tool reply message */
typedef struct _MPI2_TOOLBOX_ISTWI_REPLY
{
    U8                      Tool;                   /* 0x00 */
    U8                      Reserved1;              /* 0x01 */
    U8                      MsgLength;              /* 0x02 */
    U8                      Function;               /* 0x03 */
    U16                     Reserved2;              /* 0x04 */
    U8                      Reserved3;              /* 0x06 */
    U8                      MsgFlags;               /* 0x07 */
    U8                      VP_ID;                  /* 0x08 */
    U8                      VF_ID;                  /* 0x09 */
    U16                     Reserved4;              /* 0x0A */
    U16                     Reserved5;              /* 0x0C */
    U16                     IOCStatus;              /* 0x0E */
    U32                     IOCLogInfo;             /* 0x10 */
    U8                      DevIndex;               /* 0x14 */
    U8                      Action;                 /* 0x15 */
    U8                      IstwiStatus;            /* 0x16 */
    U8                      Reserved6;              /* 0x17 */
    U16                     TxDataCount;            /* 0x18 */
    U16                     RxDataCount;            /* 0x1A */
} MPI2_TOOLBOX_ISTWI_REPLY, MPI2_POINTER PTR_MPI2_TOOLBOX_ISTWI_REPLY,
  Mpi2ToolboxIstwiReply_t, MPI2_POINTER pMpi2ToolboxIstwiReply_t;


/*****************************************************************************
*  Toolbox Beacon Tool request
*****************************************************************************/

typedef struct _MPI2_TOOLBOX_BEACON_REQUEST
{
    U8                      Tool;                   /* 0x00 */
    U8                      Reserved1;              /* 0x01 */
    U8                      ChainOffset;            /* 0x02 */
    U8                      Function;               /* 0x03 */
    U16                     Reserved2;              /* 0x04 */
    U8                      Reserved3;              /* 0x06 */
    U8                      MsgFlags;               /* 0x07 */
    U8                      VP_ID;                  /* 0x08 */
    U8                      VF_ID;                  /* 0x09 */
    U16                     Reserved4;              /* 0x0A */
    U8                      Reserved5;              /* 0x0C */
    U8                      PhysicalPort;           /* 0x0D */
    U8                      Reserved6;              /* 0x0E */
    U8                      Flags;                  /* 0x0F */
} MPI2_TOOLBOX_BEACON_REQUEST, MPI2_POINTER PTR_MPI2_TOOLBOX_BEACON_REQUEST,
  Mpi2ToolboxBeaconRequest_t, MPI2_POINTER pMpi2ToolboxBeaconRequest_t;

/* values for the Flags field */
#define MPI2_TOOLBOX_FLAGS_BEACONMODE_OFF       (0x00)
#define MPI2_TOOLBOX_FLAGS_BEACONMODE_ON        (0x01)


/*****************************************************************************
*  Toolbox Diagnostic CLI Tool
*****************************************************************************/

#define MPI2_TOOLBOX_DIAG_CLI_CMD_LENGTH    (0x5C)
```



```
/* Toolbox Diagnostic CLI Tool request message */
typedef struct _MPI2_TOOLBOX_DIAGNOSTIC_CLI_REQUEST
{
    U8                      Tool;                   /* 0x00 */
    U8                      Reserved1;              /* 0x01 */
    U8                      ChainOffset;            /* 0x02 */
    U8                      Function;               /* 0x03 */
    U16                     Reserved2;              /* 0x04 */
    U8                      Reserved3;              /* 0x06 */
    U8                      MsgFlags;               /* 0x07 */
    U8                      VP_ID;                  /* 0x08 */
    U8                      VF_ID;                  /* 0x09 */
    U16                     Reserved4;              /* 0x0A */
    U8                      SGLFlags;               /* 0x0C */
    U8                      Reserved5;              /* 0x0D */
    U16                     Reserved6;              /* 0x0E */
    U32                     DataLength;             /* 0x10 */
    U8                      DiagnosticCliCommand[MPI2_TOOLBOX_DIAG_CLI_CMD_LENGT
    MPI2_SGE_SIMPLE_UNION   SGL;                    /* 0x70 */
} MPI2_TOOLBOX_DIAGNOSTIC_CLI_REQUEST,
  MPI2_POINTER PTR_MPI2_TOOLBOX_DIAGNOSTIC_CLI_REQUEST,
  Mpi2ToolboxDiagnosticCliRequest_t,
  MPI2_POINTER pMpi2ToolboxDiagnosticCliRequest_t;

/* use MPI2_SGLFLAGS_ defines from mpi2.h for the SGLFlags field */
 70 /* values for SGLFlags field are in the SGL section of mpi2.h */


/* Toolbox Diagnostic CLI Tool reply message */
typedef struct _MPI2_TOOLBOX_DIAGNOSTIC_CLI_REPLY
{
    U8                      Tool;                   /* 0x00 */
    U8                      Reserved1;              /* 0x01 */
    U8                      MsgLength;              /* 0x02 */
    U8                      Function;               /* 0x03 */
    U16                     Reserved2;              /* 0x04 */
    U8                      Reserved3;              /* 0x06 */
    U8                      MsgFlags;               /* 0x07 */
    U8                      VP_ID;                  /* 0x08 */
    U8                      VF_ID;                  /* 0x09 */
    U16                     Reserved4;              /* 0x0A */
    U16                     Reserved5;              /* 0x0C */
    U16                     IOCStatus;              /* 0x0E */
    U32                     IOCLogInfo;             /* 0x10 */
    U32                     ReturnedDataLength;     /* 0x14 */
} MPI2_TOOLBOX_DIAGNOSTIC_CLI_REPLY,
_____unchanged_portion_omitted_
  Mpi2DiagBufferPostRequest_t, MPI2_POINTER pMpi2DiagBufferPostRequest_t;

/* values for the ExtendedType field */
#define MPI2_DIAG_EXTENDED_TYPE_UTILIZATION         (0x02)

/* values for the BufferType field */
#define MPI2_DIAG_BUF_TYPE_TRACE                    (0x00)
#define MPI2_DIAG_BUF_TYPE_SNAPSHOT                 (0x01)
#define MPI2_DIAG_BUF_TYPE_EXTENDED                 (0x02)
/* count of the number of buffer types */
#define MPI2_DIAG_BUF_TYPE_COUNT                    (0x03)

/* values for the Flags field */
#define MPI2_DIAG_BUF_FLAG_RELEASE_ON_FULL          (0x00000002) /* for MPI v2.0
#define MPI2_DIAG_BUF_FLAG_IMMEDIATE_RELEASE        (0x00000001)
#endif /* ! codereview */

/*****************************************************************************
```

```
418 *  Diagnostic Buffer Post reply
419 ****************************************************************************/

421 typedef struct _MPI2_DIAG_BUFFER_POST_REPLY
422 {
423     U8                      ExtendedType;               /* 0x00 */
424     U8                      BufferType;                 /* 0x01 */
425     U8                      MsgLength;                  /* 0x02 */
426     U8                      Function;                   /* 0x03 */
427     U16                     Reserved2;                  /* 0x04 */
428     U8                      Reserved3;                  /* 0x06 */
429     U8                      MsgFlags;                   /* 0x07 */
430     U8                      VP_ID;                      /* 0x08 */
431     U8                      VF_ID;                      /* 0x09 */
432     U16                     Reserved4;                  /* 0x0A */
433     U16                     Reserved5;                  /* 0x0C */
434     U16                     IOCStatus;                  /* 0x0E */
435     U32                     IOCLogInfo;                 /* 0x10 */
436     U32                     TransferLength;             /* 0x14 */
437 } MPI2_DIAG_BUFFER_POST_REPLY, MPI2_POINTER PTR_MPI2_DIAG_BUFFER_POST_REPLY,
438   Mpi2DiagBufferPostReply_t, MPI2_POINTER pMpi2DiagBufferPostReply_t;


441 /****************************************************************************
442 *  Diagnostic Release request
443 ****************************************************************************/

445 typedef struct _MPI2_DIAG_RELEASE_REQUEST
446 {
447     U8                      Reserved1;                  /* 0x00 */
448     U8                      BufferType;                 /* 0x01 */
449     U8                      ChainOffset;                /* 0x02 */
450     U8                      Function;                   /* 0x03 */
451     U16                     Reserved2;                  /* 0x04 */
452     U8                      Reserved3;                  /* 0x06 */
453     U8                      MsgFlags;                   /* 0x07 */
454     U8                      VP_ID;                      /* 0x08 */
455     U8                      VF_ID;                      /* 0x09 */
456     U16                     Reserved4;                  /* 0x0A */
457 } MPI2_DIAG_RELEASE_REQUEST, MPI2_POINTER PTR_MPI2_DIAG_RELEASE_REQUEST,
458   Mpi2DiagReleaseRequest_t, MPI2_POINTER pMpi2DiagReleaseRequest_t;


461 /****************************************************************************
462 *  Diagnostic Buffer Post reply
463 ****************************************************************************/

465 typedef struct _MPI2_DIAG_RELEASE_REPLY
466 {
467     U8                      Reserved1;                  /* 0x00 */
468     U8                      BufferType;                 /* 0x01 */
469     U8                      MsgLength;                  /* 0x02 */
470     U8                      Function;                   /* 0x03 */
471     U16                     Reserved2;                  /* 0x04 */
472     U8                      Reserved3;                  /* 0x06 */
473     U8                      MsgFlags;                   /* 0x07 */
474     U8                      VP_ID;                      /* 0x08 */
475     U8                      VF_ID;                      /* 0x09 */
476     U16                     Reserved4;                  /* 0x0A */
477     U16                     Reserved5;                  /* 0x0C */
478     U16                     IOCStatus;                  /* 0x0E */
479     U32                     IOCLogInfo;                 /* 0x10 */
480 } MPI2_DIAG_RELEASE_REPLY, MPI2_POINTER PTR_MPI2_DIAG_RELEASE_REPLY,
481   Mpi2DiagReleaseReply_t, MPI2_POINTER pMpi2DiagReleaseReply_t;
```



```
484 #endif
```

```
**********************************************************
    4215 Thu Jun 12 17:42:18 2014
new/usr/src/uts/common/sys/scsi/adapters/mpt_sas3/mpi/mpi2_type.h
Initial modifications using the code changes present between
the LSI source code for FreeBSD drivers. Specifically the changes
between from mpslsi-source-17.00.00.00 -> mpslsi-source-03.00.00.00.
This mainly involves using a different scatter/gather element in
frame setup.
**********************************************************
```

```
/*
 * 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) 2000-2012 LSI Corporation.
 * Copyright (c) 2000 to 2009, LSI Corporation.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms of all code within
 * this file that is exclusively owned by LSI, with or without
 * modification, is permitted provided that, in addition to the CDDL 1.0
 * License requirements, the following conditions are met:
 *
 *    Neither the name of the author nor the names of its contributors may be
 *    used to endorse or promote products derived from this software without
 *    specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS 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
 * COPYRIGHT OWNER 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.
 */

/*
 *          Name:  mpi2_type.h
 *         Title:  MPI basic type definitions
 *  Creation Date:  August 16, 2006
 *
 *    mpi2_type.h Version:  02.00.00
 *
 *  Version History
```



```
 *  ---------------
 *
 *  Date      Version   Description
 *  --------  --------  ------------------------------------------------------
 *  04-30-07  02.00.00  Corresponds to Fusion-MPT MPI Specification Rev A.
 *  ------------------------------------------------------------------------
 */

#ifndef MPI2_TYPE_H
#define MPI2_TYPE_H


/*****************************************************************************
 * Define MPI2_POINTER if it hasn't already been defined. By default
 * MPI2_POINTER is defined to be a near pointer. MPI2_POINTER can be defined as
 * a far pointer by defining MPI2_POINTER as "far *" before this header file is
 * included.
 */
#ifndef MPI2_POINTER
#define MPI2_POINTER     *
#endif

/* the basic types may have already been included by mpi_type.h */
#ifndef MPI_TYPE_H
/*****************************************************************************
 *
 *               Basic Types
 *
 *****************************************************************************/

typedef signed   char   S8;
typedef unsigned char   U8;
typedef signed   short  S16;
typedef unsigned short  U16;


#if defined(unix) || defined(__arm) || defined(ALPHA) || defined(__PPC__) || def

    typedef signed   int   S32;
    typedef unsigned int   U32;

#else

    typedef signed   long  S32;
    typedef unsigned long  U32;

#endif


typedef struct _S64
{
    U32          Low;
    S32          High;
} S64;
```

_____*unchanged_portion_omitted_*

```
**********************************************************
    1986 Thu Jun 12 17:42:19 2014
new/usr/src/uts/common/sys/scsi/adapters/mpt_sas3/mptsas3_hash.h
hg changesets 607a5b46a793..b706c96317c3
Fix ncpus for early boot config
Purge the ack to the interrupt before exiting mptsas_intr()
Changes from code review
Changes to enable driver to compile.
Header paths, object lists, etc.
**********************************************************
```

```
/*
 * This file and its contents are supplied under the terms of the
 * Common Development and Distribution License ("CDDL"), version 1.0.
 * You may only use this file in accordance with the terms of version
 * 1.0 of the CDDL.
 *
 * A full copy of the text of the CDDL should have accompanied this
 * source.  A copy of the CDDL is also available via the Internet at
 * http://www.illumos.org/license/CDDL.
 */

/*
 * Copyright 2014 Joyent, Inc.  All rights reserved.
 * Copyright (c) 2014, Tegile Systems Inc. All rights reserved.
#endif /* ! codereview */
 */

#ifndef _SYS_SCSI_ADAPTERS_MPTSAS3_HASH_H
#define _SYS_SCSI_ADAPTERS_MPTSAS3_HASH_H
#ifndef _SYS_SCSI_ADAPTERS_MPTHASH_H
#define _SYS_SCSI_ADAPTERS_MPTHASH_H

#include <sys/types.h>
#include <sys/list.h>

#define RHL_F_DEAD      0x01

typedef struct refhash_link {
        list_node_t rhl_chain_link;
        list_node_t rhl_global_link;
        uint_t rhl_flags;
        uint_t rhl_refcnt;
} refhash_link_t;
```

*unchanged_portion_omitted_*

```
extern refhash_t *refhash_create(uint_t, refhash_hash_f, refhash_cmp_f,
    refhash_dtor_f, size_t, size_t, size_t, int);
extern void refhash_destroy(refhash_t *);
extern void refhash_insert(refhash_t *, void *);
extern void refhash_remove(refhash_t *, void *);
extern void *refhash_lookup(refhash_t *, const void *);
extern void *refhash_linear_search(refhash_t *, refhash_eval_f, void *);
extern void refhash_hold(refhash_t *, void *);
extern void refhash_rele(refhash_t *, void *);
extern void *refhash_first(refhash_t *);
extern void *refhash_next(refhash_t *, void *);
extern boolean_t refhash_obj_valid(refhash_t *hp, const void *);

#endif  /* _SYS_SCSI_ADAPTERS_MPTSAS3_HASH_H */
#endif  /* _SYS_SCSI_ADAPTERS_MPTHASH_H */
```

```
**********************************************************
    9415 Thu Jun 12 17:42:19 2014
new/usr/src/uts/common/sys/scsi/adapters/mpt_sas3/mptsas3_ioctl.h
hg changesets 607a5b46a793..b706c96317c3
Fix ncpus for early boot config
Purge the ack to the interrupt before exiting mptsas_intr()
Changes from code review
Initial modifications using the code changes present between
the LSI source code for FreeBSD drivers. Specifically the changes
between from mpslsi-source-17.00.00.00 -> mpslsi-source-03.00.00.00.
This mainly involves using a different scatter/gather element in
frame setup.
Changes to enable driver to compile.
Header paths, object lists, etc.
**********************************************************
```

```
/*
 * 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 2010 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */
/*
 * Copyright (c) 2013, Joyent, Inc. All rights reserved.
 * Copyright (c) 2014, Tegile Systems Inc. All rights reserved.
#endif /* ! codereview */
 */

/*
 * Copyright (c) 2000 to 2010, LSI Corporation.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms of all code within
 * this file that is exclusively owned by LSI, with or without
 * modification, is permitted provided that, in addition to the CDDL 1.0
 * License requirements, the following conditions are met:
 *
 *    Neither the name of the author nor the names of its contributors may be
 *    used to endorse or promote products derived from this software without
 *    specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS 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
 * COPYRIGHT OWNER 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 _MPTSAS3_IOCTL_H
#define _MPTSAS3_IOCTL_H
#ifndef _MPTSAS_IOCTL_H
#define _MPTSAS_IOCTL_H

#ifdef __cplusplus
extern "C" {
#endif

#include <sys/types.h>

#define MPTIOCTL                        ('I' << 8)
#define MPTIOCTL_GET_ADAPTER_DATA       (MPTIOCTL | 1)
#define MPTIOCTL_UPDATE_FLASH           (MPTIOCTL | 2)
#define MPTIOCTL_RESET_ADAPTER          (MPTIOCTL | 3)
#define MPTIOCTL_PASS_THRU              (MPTIOCTL | 4)
#define MPTIOCTL_EVENT_QUERY            (MPTIOCTL | 5)
#define MPTIOCTL_EVENT_ENABLE           (MPTIOCTL | 6)
#define MPTIOCTL_EVENT_REPORT           (MPTIOCTL | 7)
#define MPTIOCTL_GET_PCI_INFO           (MPTIOCTL | 8)
#define MPTIOCTL_DIAG_ACTION            (MPTIOCTL | 9)
#define MPTIOCTL_REG_ACCESS             (MPTIOCTL | 10)
#define MPTIOCTL_GET_DISK_INFO          (MPTIOCTL | 11)
#define MPTIOCTL_LED_CONTROL            (MPTIOCTL | 12)

/*
 *  The following are our ioctl() return status values.  If everything went
 *  well, we return good status.  If the buffer length sent to us is too short
 *  we return a status to tell the user.
 */
#define MPTIOCTL_STATUS_GOOD            0
#define MPTIOCTL_STATUS_LEN_TOO_SHORT   1

typedef struct mptsas_pci_bits
{
        union {
                struct {
                        uint32_t        DeviceNumber    :5;
                        uint32_t        FunctionNumber  :3;
                        uint32_t        BusNumber       :24;
                } bits;
                uint32_t        AsDWORD;
        } u;
        uint32_t        PciSegmentId;
} mptsas_pci_bits_t;
/*
 *  The following is the MPTIOCTL_GET_ADAPTER_DATA data structure.  This data
 *  structure is setup so that we hopefully are properly aligned for both
 *  32-bit and 64-bit mode applications.
 *
 *  Adapter Type - Value = 6 = SCSI Protocol through SAS-3 adapter
 *  Adapter Type - Value = 4 = SCSI Protocol through SAS-2 adapter
 *
 *  MPI Port Number - The PCI Function number for this device
 *
 *  PCI Device HW Id - The PCI device number for this device
 *
 */
#define MPTIOCTL_ADAPTER_TYPE_SAS3      6
```

```
 83 #define MPTIOCTL_ADAPTER_TYPE_SAS2      4
115 typedef struct mptsas_adapter_data
116 {
117         uint32_t                StructureLength;
118         uint32_t                AdapterType;
119         uint32_t                MpiPortNumber;
120         uint32_t                PCIDeviceHwId;
121         uint32_t                PCIDeviceHwRev;
122         uint32_t                SubSystemId;
123         uint32_t                SubsystemVendorId;
124         uint32_t                Reserved1;
125         uint32_t                MpiFirmwareVersion;
126         uint32_t                BiosVersion;
127         uint8_t                 DriverVersion[32];
128         uint8_t                 Reserved2;
129         uint8_t                 ScsiId;
130         uint16_t                Reserved3;
131         mptsas_pci_bits_t       PciInformation;
132 } mptsas_adapter_data_t;
______unchanged_portion_omitted_
360 #endif

362 #endif  /* _MPTSAS3_IOCTL_H */
331 #endif  /* _MPTSAS_IOCTL_H */
```

```
**********************************************************
    2958 Thu Jun 12 17:42:19 2014
new/usr/src/uts/common/sys/scsi/adapters/mpt_sas3/mptsas3_smhba.h
hg changesets 607a5b46a793..b706c96317c3
Fix ncpus for early boot config
Purge the ack to the interrupt before exiting mptsas_intr()
Changes from code review
Changes to enable driver to compile.
Header paths, object lists, etc.
**********************************************************
```

```
/*
 * 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) 2009, 2010, Oracle and/or its affiliates. All rights reserved.
 * Copyright 2013 Nexenta Systems, Inc. All rights reserved.
 * Copyright (c) 2014, Tegile Systems Inc. All rights reserved.
#endif /* ! codereview */
 */

/*
 * SM-HBA interfaces/definitions for MPT SAS driver.
 */

#ifndef _MPTSAS3_SMHBA_H
#define _MPTSAS3_SMHBA_H
#ifndef _MPTSAS_SMHBA_H
#define _MPTSAS_SMHBA_H
#ifdef  __cplusplus
extern "C" {
#endif

/* Leverage definition of data_type_t in nvpair.h */
#include <sys/nvpair.h>
#include <sys/scsi/adapters/mpt_sas3/mptsas3_var.h>
#include <sys/scsi/adapters/mpt_sas/mptsas_var.h>

#define MPTSAS_NUM_PHYS         "num-phys"
#define MPTSAS_NUM_PHYS_HBA     "num-phys-hba"
#define MPTSAS_SMHBA_SUPPORTED  "sm-hba-supported"
#define MPTSAS_DRV_VERSION      "driver-version"
#define MPTSAS_HWARE_VERSION    "hardware-version"
#define MPTSAS_FWARE_VERSION    "firmware-version"
#define MPTSAS_SUPPORTED_PROTOCOL       "supported-protocol"
#define MPTSAS_VIRTUAL_PORT     "virtual-port"

#define MPTSAS_MANUFACTURER     "Manufacturer"
#define MPTSAS_SERIAL_NUMBER    "SerialNumber"
```



```
#define MPTSAS_MODEL_NAME       "ModelName"
#define MPTSAS_VARIANT          "variant"

#define IS_ATAPI_DEVICE(x)      ((x) & 0x2000)
#define IS_SATA_DEVICE(x)       ((x) & 0x80)
#define DEVINFO_DIRECT_ATTACHED 0x0800

/*
 * Interfaces to add properties required for SM-HBA
 *
 * _add_xxx_prop() interfaces add only 1 prop that is specified in the args.
 * _set_xxx_props() interfaces add more than 1 prop for a set of phys/devices.
 */
int mptsas_smhba_setup(mptsas_t *);
void mptsas_smhba_show_phy_info(mptsas_t *);
void mptsas_smhba_set_all_phy_props(mptsas_t *mpt, dev_info_t *dip,
    uint8_t phy_nums, mptsas_phymask_t phy_mask, uint16_t *attached_devhdl);
void mptsas_smhba_set_one_phy_props(mptsas_t *mpt, dev_info_t *dip,
    uint8_t phy_id, uint16_t *attached_devhdl);
void mptsas_smhba_log_sysevent(mptsas_t *mpt, char *subclass, char *etype,
    smhba_info_t *phyp);
void
mptsas_create_phy_stats(mptsas_t *mpt, char *iport, dev_info_t *dip);
int mptsas_update_phy_stats(kstat_t *ks, int rw);
void mptsas_destroy_phy_stats(mptsas_t *mpt);
int mptsas_smhba_phy_init(mptsas_t *mpt);
int mptsas_smhba_phy_state_update(mptsas_t *mpt, uint8_t phy);
#ifdef  __cplusplus
}
#endif
#endif /* _MPTSAS3_SMHBA_H */
#endif  /* _MPTSAS_SMHBA_H */
```

```
**********************************************************
   46778 Thu Jun 12 17:42:19 2014
new/usr/src/uts/common/sys/scsi/adapters/mpt_sas3/mptsas3_var.h
Code reconciliation with other base.
Update tx waitq's code.
Create 2 threads, divide the workflow and deliver
to the hardware from the threads.
Optimise mutex's and code paths.
Split out offline target code.
Enable Fast Path for capable devices.
Merge fixes for Illumos issue 4819, fix mpt_sas command timeout handling.
Tweeks debug flags.
Lint and cstyle fixes.
Fix problem with running against 64bit msgaddr attributes for DMA.
Default is now to run like this.
Fixes for Illumos issue 4682.
Fix hang bug to do with tx_wq.
Re-arrange mptsas_poll() to disable interrupts before issuing the
command.
Improve the tx_waitq code path.
Major rework of mutexes.
During normal operation do not grab m_mutex during interrupt.
Use reply post queues instead.
Distribute command done processing around the threads.
Improved auto-request sense memory usage.
Re-arrange mptsas_intr() to reduce number of spurious interrupts.
Should not need m_in_callback flag. It prevents concurrent
command completion processing.
Added code to support using MSI-X interrupts across multiple
reply queues. Not tested with anything other than 3008 yet.
Use MSI-X interrupts, just one for now.
Pre-allocate array for request sense buffers, similar to command frames.
No more messing about with scsi_alloc_consistent_buf().
Add rolling buffer for *all* debug messages.
Improve mdb module and seperate out into mpt_sas3.
Initial modifications using the code changes present between
the LSI source code for FreeBSD drivers. Specifically the changes
between from mpslsi-source-17.00.00.00 -> mpslsi-source-03.00.00.00.
This mainly involves using a different scatter/gather element in
frame setup.
Changes to enable driver to compile.
Header paths, object lists, etc.
**********************************************************
```

```
/*
 * 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) 2009, 2010, Oracle and/or its affiliates. All rights reserved.
```



```
 * Copyright 2014 Nexenta Systems, Inc. All rights reserved.
 * Copyright 2012 Nexenta Systems, Inc. All rights reserved.
 * Copyright (c) 2013, Joyent, Inc. All rights reserved.
 * Copyright (c) 2014, Tegile Systems Inc. All rights reserved.
#endif /* ! codereview */
 */

/*
 * Copyright (c) 2000 to 2010, LSI Corporation.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms of all code within
 * this file that is exclusively owned by LSI, with or without
 * modification, is permitted provided that, in addition to the CDDL 1.0
 * License requirements, the following conditions are met:
 *
 *    Neither the name of the author nor the names of its contributors may be
 *    used to endorse or promote products derived from this software without
 *    specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS 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
 * COPYRIGHT OWNER 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_SCSI_ADAPTERS_MPTSAS3_VAR_H
#define _SYS_SCSI_ADAPTERS_MPTSAS3_VAR_H
#ifndef _SYS_SCSI_ADAPTERS_MPTVAR_H
#define _SYS_SCSI_ADAPTERS_MPTVAR_H

#include <sys/byteorder.h>
#include <sys/queue.h>
#endif /* ! codereview */
#include <sys/isa_defs.h>
#include <sys/sunmdi.h>
#include <sys/mdi_impldefs.h>
#include <sys/scsi/adapters/mpt_sas3/mptsas3_hash.h>
#include <sys/scsi/adapters/mpt_sas3/mptsas3_ioctl.h>
#include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_tool.h>
#include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_cnfg.h>
#include <sys/scsi/adapters/mpt_sas/mptsas_hash.h>
#include <sys/scsi/adapters/mpt_sas/mptsas_ioctl.h>
#include <sys/scsi/adapters/mpt_sas/mpi/mpi2_tool.h>
#include <sys/scsi/adapters/mpt_sas/mpi/mpi2_cnfg.h>

#ifdef  __cplusplus
extern "C" {
#endif

/*
 * Compile options
 */
#ifdef DEBUG
#define MPTSAS_DEBUG            /* turn on debugging code */
#endif  /* DEBUG */

#define MPTSAS_INITIAL_SOFT_SPACE       4
```

```
 84 #define MAX_MPI_PORTS           16

 86 /*
 87  * Note below macro definition and data type definition
 88  * are used for phy mask handling, it should be changed
 89  * simultaneously.
 90  */
 91 #define MPTSAS_MAX_PHYS         16
 92 typedef uint16_t                mptsas_phymask_t;

 94 #define MPTSAS_INVALID_DEVHDL   0xffff
 95 #define MPTSAS_SATA_GUID        "sata-guid"

 97 /*
 98  * Hash table sizes for SMP targets (i.e., expanders) and ordinary SSP/STP
 99  * targets.  There's no need to go overboard here, as the ordinary paths for
100  * I/O do not normally require hashed target lookups.  These should be good
101  * enough and then some for any fabric within the hardware's capabilities.
102  */
103 #define MPTSAS_SMP_BUCKET_COUNT         23
104 #define MPTSAS_TARGET_BUCKET_COUNT      97

106 /*
107  * MPT HW defines
108  */
109 #define MPTSAS_MAX_DISKS_IN_CONFIG      14
110 #define MPTSAS_MAX_DISKS_IN_VOL         10
111 #define MPTSAS_MAX_HOTSPARES            2
112 #define MPTSAS_MAX_RAIDVOLS             2
113 #define MPTSAS_MAX_RAIDCONFIGS          5

115 /*
116  * 64-bit SAS WWN is displayed as 16 characters as HEX characters,
117  * plus two means the prefix 'w' and end of the string '\0'.
118  */
119 #define MPTSAS_WWN_STRLEN       (16 + 2)
120 #define MPTSAS_MAX_GUID_LEN     64

122 /*
123  * DMA routine flags
124  */
125 #define MPTSAS_DMA_HANDLE_ALLOCD        0x2
126 #define MPTSAS_DMA_MEMORY_ALLOCD        0x4
127 #define MPTSAS_DMA_HANDLE_BOUND 0x8

129 /*
130  * If the HBA supports DMA or bus-mastering, you may have your own
131  * scatter-gather list for physically non-contiguous memory in one
132  * I/O operation; if so, there's probably a size for that list.
133  * It must be placed in the ddi_dma_lim_t structure, so that the system
134  * DMA-support routines can use it to break up the I/O request, so we
135  * define it here.
136  */
137 #if defined(__sparc)
138 #define MPTSAS_MAX_DMA_SEGS     1
139 #define MPTSAS_MAX_CMD_SEGS     1
140 #else
141 #define MPTSAS_MAX_DMA_SEGS     256
142 #define MPTSAS_MAX_CMD_SEGS     257
143 #endif
144 #define MPTSAS_MAX_FRAME_SGES(mpt) \
145         (((mpt->m_req_frame_size - (sizeof (MPI2_SCSI_IO_REQUEST))) / 8) + 1)

147 /*
148  * Calculating how many 64-bit DMA simple elements can be stored in the first
```



```
112  * Caculating how many 64-bit DMA simple elements can be stored in the first
149  * frame. Note that msg_scsi_io_request contains 2 double-words (8 bytes) for
150  * element storage.  And 64-bit dma element is 3 double-words (12 bytes) in
151  * size. IEEE 64-bit dma element used for SAS3 controllers is 4 double-words
152  * (16 bytes).
115  * size.
153  */
154 #define MPTSAS_MAX_FRAME_SGES64(mpt) \
155         ((mpt->m_req_frame_size - \
156         sizeof (MPI2_SCSI_IO_REQUEST) + sizeof (MPI2_SGE_IO_UNION)) / \
157         (mpt->m_MPI25 ? sizeof (MPI2_IEEE_SGE_SIMPLE64) : \
158         sizeof (MPI2_SGE_SIMPLE64)))
119         (sizeof (MPI2_SCSI_IO_REQUEST)) + sizeof (MPI2_SGE_IO_UNION)) / 12)

160 /*
161  * Scatter-gather list structure defined by HBA hardware
162  */
163 typedef struct NcrTableIndirect {       /* Table Indirect entries */
164         uint32_t count;         /* 24 bit count */
165         union {
166                 uint32_t address32;     /* 32 bit address */
167                 struct {
168                         uint32_t Low;
169                         uint32_t High;
170                 } address64;            /* 64 bit address */
171         } addr;
172 } mptti_t;
      unchanged_portion_omitted_

210 TAILQ_HEAD(mptsas_active_cmdq, mptsas_cmd);
211 typedef struct mptsas_active_cmdq mptsas_active_cmdq_t;

213 #endif /* ! codereview */
214 typedef struct mptsas_target {
215                 kmutex_t                m_t_mutex;
216 #endif /* ! codereview */
217                 mptsas_target_addr_t    m_addr;
218                 refhash_link_t          m_link;
171                 uint8_t                 m_dr_flag;
219                 uint16_t                m_devhdl;
220                 uint32_t                m_deviceinfo;
221                 uint32_t                m_dups;
222 #endif /* ! codereview */
223                 uint8_t                 m_phynum;
224                 mptsas_active_cmdq_t    m_active_cmdq;
174                 uint32_t                m_dups;
175                 int32_t                 m_timeout;
176                 int32_t                 m_timebase;
225                 int32_t                 m_t_throttle;
226                 int32_t                 m_t_ncmds;
227                 int32_t                 m_reset_delay;
228                 int32_t                 m_t_nwait;
229                 uint16_t                m_io_flags;
230                 uint16_t                m_enclosure;
231                 uint16_t                m_slot_num;

232                 uint16_t                m_qfull_retry_interval;
233                 uint8_t                 m_qfull_retries;
234                 uint8_t                 m_tgt_unconfigured;
184                 uint16_t                m_enclosure;
185                 uint16_t                m_slot_num;
186                 uint32_t                m_tgt_unconfigured;
235                 uint8_t                 m_led_status;
236                 uint8_t                 m_dr_flag;

237 } mptsas_target_t;
```

```
/*
 * If you change this structure, be sure that mptsas_smp_target_copy()
 * does the right thing.
 */
#endif /* ! codereview */
typedef struct mptsas_smp {
        mptsas_target_addr_t    m_addr;
        refhash_link_t          m_link;
        uint16_t                m_devhdl;
        uint32_t                m_deviceinfo;
        uint16_t                m_pdevhdl;
        uint32_t                m_pdevinfo;
} mptsas_smp_t;

typedef struct mptsas_cache_frames {
        ddi_dma_handle_t m_dma_hdl;
        ddi_acc_handle_t m_acc_hdl;
        caddr_t m_frames_addr;
        uint64_t m_phys_addr;
        uint32_t m_phys_addr;
} mptsas_cache_frames_t;

typedef struct  mptsas_cmd {
        uint_t                  cmd_flags;      /* flags from scsi_init_pkt */
        ddi_dma_handle_t        cmd_dmahandle;  /* dma handle */
        ddi_dma_cookie_t        cmd_cookie;
        uint_t                  cmd_cookiec;
        uint_t                  cmd_winindex;
        uint_t                  cmd_nwin;
        uint_t                  cmd_cur_cookie;
        off_t                   cmd_dma_offset;
        size_t                  cmd_dma_len;
        uint32_t                cmd_totaldmacount;
        caddr_t                 cmd_arq_buf;

        ddi_dma_handle_t        cmd_arqhandle;  /* dma arq handle */
        ddi_dma_cookie_t        cmd_arqcookie;
        struct buf              *cmd_arq_buf;
        ddi_dma_handle_t        cmd_ext_arqhandle; /* dma extern arq handle */
        ddi_dma_cookie_t        cmd_ext_arqcookie;
        struct buf              *cmd_ext_arq_buf;

        int                     cmd_pkt_flags;

        /* pending expiration time for command in active slot */
        hrtime_t                cmd_active_expiration;
        TAILQ_ENTRY(mptsas_cmd) cmd_active_link;
        /* timer for command in active slot */
        int                     cmd_active_timeout;

        struct scsi_pkt         *cmd_pkt;
        struct scsi_arq_status  cmd_scb;
        uchar_t                 cmd_cdblen;     /* length of cdb */
        uchar_t                 cmd_rqslen;     /* len of requested rqsense */
        uchar_t                 cmd_privlen;
        uint16_t                cmd_extrqslen;  /* len of extended rqsense */
        uint16_t                cmd_extrqschunks; /* len in map chunks */
        uint16_t                cmd_extrqsidx;  /* Index into map */
#endif /* ! codereview */
        uint_t                  cmd_scblen;
        uint32_t                cmd_dmacount;
        uint64_t                cmd_dma_addr;
        uchar_t                 cmd_age;
        ushort_t                cmd_qfull_retries;
        uchar_t                 cmd_queued;     /* true if queued */
```



```
        struct mptsas_cmd       *cmd_linkp;
        mptti_t                 *cmd_sg; /* Scatter/Gather structure */
        uchar_t                 cmd_cdb[SCSI_CDB_SIZE];
        uint64_t                cmd_pkt_private[PKT_PRIV_LEN];
        uint32_t                cmd_slot;
        uint32_t                ioc_cmd_slot;
        uint8_t                 cmd_rpqidx;
#endif /* ! codereview */

        mptsas_cache_frames_t   *cmd_extra_frames;

        uint32_t                cmd_rfm;
        mptsas_target_t         *cmd_tgt_addr;
} mptsas_cmd_t;

/*
 * These are the defined cmd_flags for this structure.
 */
#define CFLAG_CMDDISC           0x000001 /* cmd currently disconnected */
#define CFLAG_WATCH             0x000002 /* watchdog time for this command */
#define CFLAG_FINISHED          0x000004 /* command completed */
#define CFLAG_CHKSEG            0x000008 /* check cmd_data within seg */
#define CFLAG_COMPLETED         0x000010 /* completion routine called */
#define CFLAG_PREPARED          0x000020 /* pkt has been init'ed */
#define CFLAG_IN_TRANSPORT      0x000040 /* in use by host adapter driver */
#define CFLAG_RESTORE_PTRS      0x000080 /* implicit restore ptr on reconnect */
#define CFLAG_ARQ_IN_PROGRESS   0x000100 /* auto request sense in progress */
#define CFLAG_TRANFLAG          0x0001ff /* covers transport part of flags */
#define CFLAG_TM_CMD            0x000200 /* cmd is a task management command */
#define CFLAG_CMDARQ            0x000400 /* cmd is a 'rqsense' command */
#define CFLAG_DMAVALID          0x000800 /* dma mapping valid */
#define CFLAG_DMASEND           0x001000 /* data is going 'out' */
#define CFLAG_CMDIOPB           0x002000 /* this is an 'iopb' packet */
#define CFLAG_CDBEXTERN         0x004000 /* cdb kmem_alloc'd */
#define CFLAG_SCBEXTERN         0x008000 /* scb kmem_alloc'd */
#define CFLAG_FREE              0x010000 /* packet is on free list */
#define CFLAG_PRIVEXTERN        0x020000 /* target private kmem_alloc'd */
#define CFLAG_DMA_PARTIAL       0x040000 /* partial xfer OK */
#define CFLAG_QFULL_STATUS      0x080000 /* pkt got qfull status */
#define CFLAG_TIMEOUT           0x100000 /* passthru/config command timeout */
#define CFLAG_PMM_RECEIVED      0x200000 /* use cmd_pmm* for saving pointers */
#define CFLAG_RETRY             0x400000 /* cmd has been retried */
#define CFLAG_CMDIOC            0x800000 /* cmd is just for for ioc, no io */
#define CFLAG_EXTARQBUFVALID    0x1000000 /* extern arq buf handle is valid */
#define CFLAG_PASSTHRU          0x2000000 /* cmd is a passthrough command */
#define CFLAG_XARQ              0x4000000 /* cmd requests for extra sense */
#define CFLAG_CMDACK            0x8000000 /* cmd for event ack */
#define CFLAG_TXQ               0x10000000 /* cmd queued in the tx_waitq */
#define CFLAG_FW_CMD            0x20000000 /* cmd is a fw up/down command */
#define CFLAG_CONFIG            0x40000000 /* cmd is for config header/page */
#define CFLAG_FW_DIAG           0x80000000 /* cmd is for FW diag buffers */

#ifdef MPTSAS_DEBUG
/* Could be used with cmn_err %b */
#define CFLAGS_DEBUG_BITS "\\020\\0CmdDisc\\1Watch\\2Finished\\3ChkSeg" \
        "\\4Cmpltd\\5Prepd\\6InTran\\7RestPtrs\\8ARQIP\\9TM\\10CArq" \
        "\\11DMAVal\\12DMASnd\\13CIopb\\14CDBExt\\15SCBExt\\16Free" \
        "\\17PrivExt\\18DMAPrtl\\19QFull\\20Tout\\21PMMRcv\\22Retry" \
        "\\23CIoc\\25PThru\\26XArq\\27CAck\\28TXq\\29FWCmd\\30Config\\31FWDiag"
#endif

#endif /* ! codereview */
#define MPTSAS_SCSI_REPORTLUNS_ADDRESS_SIZE                     8
#define MPTSAS_SCSI_REPORTLUNS_ADDRESS_MASK                     0xC0
#define MPTSAS_SCSI_REPORTLUNS_ADDRESS_PERIPHERAL                       0x00
#define MPTSAS_SCSI_REPORTLUNS_ADDRESS_FLAT_SPACE                       0x40
```

```
359 #define MPTSAS_SCSI_REPORTLUNS_ADDRESS_LOGICAL_UNIT             0x80
360 #define MPTSAS_SCSI_REPORTLUNS_ADDRESS_EXTENDED_UNIT            0xC0
361 #define MPTSAS_SCSI_REPORTLUNS_ADDRESS_LOGICAL_UNIT_2B          0x00
362 #define MPTSAS_SCSI_REPORTLUNS_ADDRESS_LOGICAL_UNIT_4B          0x01
363 #define MPTSAS_SCSI_REPORTLUNS_ADDRESS_LOGICAL_UNIT_6B          0x10
364 #define MPTSAS_SCSI_REPORTLUNS_ADDRESS_LOGICAL_UNIT_8B          0x20
365 #define MPTSAS_SCSI_REPORTLUNS_ADDRESS_LOGICAL_UNIT_SIZE                0x30

367 #define MPTSAS_HASH_ARRAY_SIZE  16
368 /*
369  * hash table definition
370  */

372 #define MPTSAS_HASH_FIRST       0xffff
373 #define MPTSAS_HASH_NEXT        0x0000

375 typedef struct mptsas_dma_alloc_state
376 {
377         ddi_dma_handle_t        handle;
378         caddr_t                 memp;
379         size_t                  size;
380         ddi_acc_handle_t        accessp;
381         ddi_dma_cookie_t        cookie;
382 } mptsas_dma_alloc_state_t;

384 /*
385  * passthrough request structure
386  */
387 typedef struct mptsas_pt_request {
388         uint8_t *request;
389         uint32_t request_size;
390         uint32_t data_size;
391         uint32_t dataout_size;
392         uint8_t direction;
393         uint8_t simple;
394         uint16_t sgl_offset;
232         uint32_t direction;
395         ddi_dma_cookie_t data_cookie;
396         ddi_dma_cookie_t dataout_cookie;
397 } mptsas_pt_request_t;
```

*unchanged_portion_omitted_*

```
559 _NOTE(DATA_READABLE_WITHOUT_LOCK(mptsas_topo_change_list_t::mpt))
560 _NOTE(DATA_READABLE_WITHOUT_LOCK(mptsas_topo_change_list_t::event))
561 _NOTE(DATA_READABLE_WITHOUT_LOCK(mptsas_topo_change_list_t::physport))
562 _NOTE(DATA_READABLE_WITHOUT_LOCK(mptsas_topo_change_list_t::devhdl))
563 _NOTE(DATA_READABLE_WITHOUT_LOCK(mptsas_topo_change_list_t::object))
564 _NOTE(DATA_READABLE_WITHOUT_LOCK(mptsas_topo_change_list_t::flags))

566 /*
567  * Status types when calling mptsas_get_target_device_info
568  */
569 #define DEV_INFO_SUCCESS                0x0
570 #define DEV_INFO_FAIL_PAGE0             0x1
571 #define DEV_INFO_WRONG_DEVICE_TYPE      0x2
572 #define DEV_INFO_PHYS_DISK              0x3
573 #define DEV_INFO_FAIL_ALLOC             0x4

575 /*
576  * mpt hotplug event defines
577  */
578 #define MPTSAS_DR_EVENT_RECONFIG_TARGET 0x01
579 #define MPTSAS_DR_EVENT_OFFLINE_TARGET  0x02
580 #define MPTSAS_DR_EVENT_REMOVE_HANDLE   0x04
418 #define MPTSAS_TOPO_FLAG_REMOVE_HANDLE  0x04
```



```
582 /*
583  * SMP target hotplug events
584  */
585 #define MPTSAS_DR_EVENT_RECONFIG_SMP    0x10
586 #define MPTSAS_DR_EVENT_OFFLINE_SMP     0x20
587 #define MPTSAS_DR_EVENT_MASK            0x3F

589 /*
590  * mpt hotplug status definition for m_dr_flag
591  */

593 /*
594  * MPTSAS_DR_INACTIVE
595  *
596  * The target is in a normal operating state.
597  * No dynamic reconfiguration operation is in progress.
598  */
599 #define MPTSAS_DR_INACTIVE                              0x0
600 /*
601  * MPTSAS_DR_INTRANSITION
602  *
603  * The target is in a transition mode since
604  * hotplug event happens and offline procedure has not
605  * been finished
606  */
607 #define MPTSAS_DR_INTRANSITION                  0x1

609 typedef struct mptsas_tgt_private {
610         int t_lun;
611         struct mptsas_target *t_private;
612 } mptsas_tgt_private_t;
```

*unchanged_portion_omitted_*

```
674 typedef struct mptsas_thread_arg {
512 typedef struct mptsas_doneq_thread_arg {
675         void            *mpt;
676         uint32_t        t;
677 } mptsas_thread_arg_t;

679 typedef struct mptsas_done_list {
680         mptsas_cmd_t            *dl_q;
681         mptsas_cmd_t            **dl_tail;
682         uint32_t                dl_len;
683 } mptsas_done_list_t;
514         uint64_t        t;
515 } mptsas_doneq_thread_arg_t;

685 #define MPTSAS_DONEQ_THREAD_ACTIVE      0x1
686 typedef struct mptsas_doneq_thread_list {
687         mptsas_done_list_t      dlist;
519         mptsas_cmd_t            *doneq;
520         mptsas_cmd_t            **donetail;
688         kthread_t               *threadp;
689         kcondvar_t              cv;
690         ushort_t                reserv1;
691         uint32_t                reserv2;
692         kmutex_t                mutex;
693         uint32_t                flag;
694         mptsas_thread_arg_t     arg;
527         uint32_t                len;
528         mptsas_doneq_thread_arg_t       arg;
695 } mptsas_doneq_thread_list_t;

697 typedef struct mptsas_reply_pqueue {
```

```
698         kmutex_t                rpq_mutex;
699         uint8_t                 rpq_num;
700         caddr_t                 rpq_queue; /* Pointer to this queue base */
701         uint32_t                rpq_index; /* Index of next replyq entry */
702         uint32_t                rpq_ncmds; /* Number of outstanding commands */
703         mptsas_done_list_t      rpq_dlist;
704         uint32_t                rpq_intr_count;
705         uint32_t                rpq_intr_unclaimed;
706         uint32_t                rpq_intr_mutexbusy;
707 } mptsas_reply_pqueue_t;


710 typedef struct mptsas_tx_waitqueue {
711         kmutex_t        txwq_mutex;
712         kcondvar_t      txwq_cv;
713         kcondvar_t      txwq_drain_cv;
714         kthread_t       *txwq_threadp;
715         mptsas_cmd_t    *txwq_cmdq;     /* TX cmd queue for active request */
716         mptsas_cmd_t    **txwq_qtail;   /* tx_wait queue tail ptr */
717         uint32_t        txwq_len;       /* TX queue length */
718         mptsas_thread_arg_t     arg;
719         uint8_t         txwq_active;    /* Thread active flag */
720         uint8_t         txwq_draining;  /* TX queue draining flag */
721         uint8_t         txwq_wdrain;
722 } mptsas_tx_waitqueue_t;

724 #define NUM_TX_WAITQ    2

726 #endif /* ! codereview */
727 typedef struct mptsas {
728         int             m_instance;

730         struct mptsas *m_next;

732         scsi_hba_tran_t         *m_tran;
733         smp_hba_tran_t          *m_smptran;
734         kmutex_t                m_mutex;
735         kmutex_t                m_passthru_mutex;
736         kcondvar_t              m_cv;
737         kcondvar_t              m_passthru_cv;
738         kcondvar_t              m_fw_cv;
739         kcondvar_t              m_config_cv;
740         kcondvar_t              m_fw_diag_cv;
741         dev_info_t              *m_dip;

743         /*
744          * soft state flags
745          */
746         uint_t          m_softstate;

748         refhash_t       *m_targets;
749         refhash_t       *m_smp_targets;

751         m_raidconfig_t  m_raidconfig[MPTSAS_MAX_RAIDCONFIGS];
752         uint8_t         m_num_raid_configs;
753         uint8_t         m_pref_tx_waitq;
754 #endif /* ! codereview */

756         struct mptsas_slots *m_active;  /* outstanding cmds */

758         mptsas_cmd_t    *m_waitq;       /* cmd queue for active request */
759         mptsas_cmd_t    **m_waitqtail;  /* wait queue tail ptr */
760         mptsas_tx_waitqueue_t   m_tx_waitq[NUM_TX_WAITQ];
761         uint16_t                m_txwq_thread_threshold;
762         uint16_t                m_txwq_thread_n;
763         uint8_t                 m_txwq_enabled;
```



```
764         uint8_t                 m_txwq_allow_q_jumping;
765         mptsas_done_list_t      m_dlist; /* List of completed commands */

532         kmutex_t        m_tx_waitq_mutex;
533         mptsas_cmd_t    *m_tx_waitq;    /* TX cmd queue for active request */
534         mptsas_cmd_t    **m_tx_waitqtail;       /* tx_wait queue tail ptr */
535         int             m_tx_draining;  /* TX queue draining flag */

537         mptsas_cmd_t    *m_doneq;       /* queue of completed commands */
538         mptsas_cmd_t    **m_donetail;   /* queue tail ptr */

767         /*
768          * variables for helper threads (fan-out interrupts)
769          */
770         mptsas_doneq_thread_list_t      *m_doneq_thread_id;
771         uint16_t                m_doneq_thread_n;
772         uint16_t                m_doneq_next_thread;
544         uint32_t                m_doneq_thread_n;
773         uint32_t                m_doneq_thread_threshold;
774         uint32_t                m_doneq_length_threshold;
775         kcondvar_t              m_qthread_cv;
776         kmutex_t                m_qthread_mutex;
547         uint32_t                m_doneq_len;
548         kcondvar_t              m_doneq_thread_cv;
549         kmutex_t                m_doneq_mutex;

778         uint32_t                m_ncmds; /* number of outstanding commands */
779         uint32_t                m_ncstarted; /* ncmds started per interval */
780         uint32_t                m_lncstarted; /* record of last value */
551         int             m_ncmds;        /* number of outstanding commands */
781         m_event_struct_t *m_ioc_event_cmdq;     /* cmd queue for ioc event */
782         m_event_struct_t **m_ioc_event_cmdtail; /* ioc cmd queue tail */

784         ddi_acc_handle_t m_datap;       /* operating regs data access handle */

786         struct _MPI2_SYSTEM_INTERFACE_REGS      *m_reg;

788         ushort_t        m_devid;        /* device id of chip. */
789         uchar_t         m_revid;        /* revision of chip. */
790         uint16_t        m_svid;         /* subsystem Vendor ID of chip */
791         uint16_t        m_ssid;         /* subsystem Device ID of chip */

793         uchar_t         m_sync_offset;  /* default offset for this chip. */

795         timeout_id_t    m_quiesce_timeid;

797         ddi_dma_handle_t m_dma_req_frame_hdl;
798         ddi_acc_handle_t m_acc_req_frame_hdl;
799         ddi_dma_handle_t m_dma_req_sense_hdl;
800         ddi_acc_handle_t m_acc_req_sense_hdl;
801 #endif /* ! codereview */
802         ddi_dma_handle_t m_dma_reply_frame_hdl;
803         ddi_acc_handle_t m_acc_reply_frame_hdl;
804         ddi_dma_handle_t m_dma_free_queue_hdl;
805         ddi_acc_handle_t m_acc_free_queue_hdl;
806         ddi_dma_handle_t m_dma_post_queue_hdl;
807         ddi_acc_handle_t m_acc_post_queue_hdl;
808         uint8_t         m_dma_flags;
809 #endif /* ! codereview */

811         /*
812          * list of reset notification requests
813          */
814         struct scsi_reset_notify_entry  *m_reset_notify_listf;

816         /*
```

```
817          * qfull handling
818          */
819         timeout_id_t    m_restart_cmd_timeid;

821         /*
822          * scsi reset delay per bus
823          */
824         uint_t          m_scsi_reset_delay;

826         int             m_pm_idle_delay;

828         uchar_t         m_polled_intr;  /* intr was polled. */
829         uchar_t         m_suspended;    /* true if driver is suspended */

831         struct kmem_cache *m_kmem_cache;
832         struct kmem_cache *m_cache_frames;

834         /*
835          * hba options.
836          */
837         uint_t          m_options;

570         int             m_in_callback;

839         int             m_power_level;  /* current power level */

841         int             m_busy;         /* power management busy state */

843         off_t           m_pmcsr_offset; /* PMCSR offset */

845         ddi_acc_handle_t m_config_handle;

847         ddi_dma_attr_t          m_io_dma_attr;  /* Used for data I/O */
848         ddi_dma_attr_t          m_msg_dma_attr; /* Used for message frames */
849         ddi_device_acc_attr_t   m_dev_acc_attr;
850         ddi_device_acc_attr_t   m_reg_acc_attr;

852         /*
853          * request/reply variables
854          */
855         caddr_t         m_req_frame;
856         uint64_t        m_req_frame_dma_addr;
857         caddr_t         m_req_sense;
858         caddr_t         m_extreq_sense;
859         uint64_t        m_req_sense_dma_addr;
860 #endif /* ! codereview */
861         caddr_t         m_reply_frame;
862         uint64_t        m_reply_frame_dma_addr;
863         caddr_t         m_free_queue;
864         uint64_t        m_free_queue_dma_addr;
865         caddr_t         m_post_queue;
866         uint64_t        m_post_queue_dma_addr;
867         struct map      *m_erqsense_map;
868         mptsas_reply_pqueue_t   *m_rep_post_queues;
869 #endif /* ! codereview */

871         m_replyh_arg_t *m_replyh_args;

873         uint16_t        m_max_requests;
874         uint16_t        m_req_frame_size;
875         uint16_t        m_req_sense_size;
876 #endif /* ! codereview */

878         /*
879          * Max frames per request reported in IOC Facts
590          * Max frames per request reprted in IOC Facts
```



```
880          */
881         uint8_t         m_max_chain_depth;
882         /*
883          * Max frames per request which is used in reality. It's adjusted
884          * according DMA SG length attribute, and shall not exceed the
885          * m_max_chain_depth.
886          */
887         uint8_t         m_max_request_frames;
888         uint8_t         m_max_msix_vectors;
889         uint8_t         m_reply_frame_size;
890         uint8_t         m_post_reply_qcount;
891 #endif /* ! codereview */

893         uint16_t        m_free_queue_depth;
894         uint16_t        m_post_queue_depth;
895         uint16_t        m_max_replies;
896         uint32_t        m_free_index;
599         uint32_t        m_post_index;
600         uint8_t         m_reply_frame_size;
897         uint32_t        m_ioc_capabilities;

899         /*
900          * Housekeeping.
901          */
902         uint64_t        m_interrupt_count;
903         uint32_t        m_unclaimed_pm_interrupt_count;
904         uint32_t        m_unclaimed_polled_interrupt_count;
905         uint32_t        m_unclaimed_no_interrupt_count;
906         uint32_t        m_unclaimed_nocmd_interrupt_count;

908         /*
909 #endif /* ! codereview */
910          * indicates if the firmware was upload by the driver
911          * at boot time
912          */
913         ushort_t        m_fwupload;

915         uint16_t        m_productid;

917         /*
918          * per instance data structures for dma memory resources for
919          * MPI handshake protocol. only one handshake cmd can run at a time.
920          */
921         ddi_dma_handle_t        m_hshk_dma_hdl;
922         ddi_acc_handle_t        m_hshk_acc_hdl;
923         caddr_t                 m_hshk_memp;
924         size_t                  m_hshk_dma_size;

926         /* Firmware version on the card at boot time */
927         uint32_t                m_fwversion;

929         /* MSI specific fields */
930         ddi_intr_handle_t       *m_htable;      /* For array of interrupts */
931         int                     m_intr_type;    /* What type of interrupt */
932         int                     m_intr_cnt;     /* # of intrs count returned */
933         size_t                  m_intr_size;    /* Size of intr array */
934         uint_t                  m_intr_pri;     /* Interrupt priority   */
935         int                     m_intr_cap;     /* Interrupt capabilities */
936         ddi_taskq_t             *m_event_taskq;

938         /* SAS specific information */

940         union {
941                 uint64_t        m_base_wwid;    /* Base WWID */
942                 struct {
943 #ifdef _BIG_ENDIAN
```

```
944                         uint32_t        m_base_wwid_hi;
945                         uint32_t        m_base_wwid_lo;
946 #else
947                         uint32_t        m_base_wwid_lo;
948                         uint32_t        m_base_wwid_hi;
949 #endif
950                 } sasaddr;
951         } un;

953         uint8_t                 m_num_phys;             /* # of PHYs */
954         mptsas_phy_info_t       m_phy_info[MPTSAS_MAX_PHYS];
955         uint8_t                 m_port_chng;    /* initiator port changes */
956         MPI2_CONFIG_PAGE_MAN_0   m_MANU_page0;   /* Manufactor page 0 info */
957         MPI2_CONFIG_PAGE_MAN_1   m_MANU_page1;   /* Manufactor page 1 info */

959         /* FMA Capabilities */
960         int                     m_fm_capabilities;
961         ddi_taskq_t             *m_dr_taskq;
962         int                     m_mpxio_enable;
963         uint8_t                 m_done_traverse_dev;
964         uint8_t                 m_done_traverse_smp;
965         int                     m_diag_action_in_progress;
966         uint16_t                m_dev_handle;
967         uint16_t                m_smp_devhdl;

969         /*
970          * Event recording
971          */
972         uint8_t                 m_event_index;
973         uint32_t                m_event_number;
974         uint32_t                m_event_mask[4];
975         mptsas_event_entry_t    m_events[MPTSAS_EVENT_QUEUE_SIZE];

977         /*
978          * FW diag Buffer List
979          */
980         mptsas_fw_diagnostic_buffer_t
981                 m_fw_diag_buffer_list[MPI2_DIAG_BUF_TYPE_COUNT];

983         /* GEN3 support */
984         uint8_t                 m_MPI25;

986 #endif /* ! codereview */
987         /*
988          * Event Replay flag (MUR support)
989          */
990         uint8_t                 m_event_replay;

992         /*
993          * IR Capable flag
994          */
995         uint8_t                 m_ir_capable;

997         /*
998          * Is HBA processing a diag reset?
999          */
1000        uint8_t                 m_in_reset;

1002        /*
1003         * per instance cmd data structures for task management cmds
1004         */
1005        m_event_struct_t        m_event_task_mgmt;      /* must be last */
1006                                                        /* ... scsi_pkt_size */
1007 } mptsas_t;
1008 #define MPTSAS_SIZE     (sizeof (struct mptsas) - \
1009                         sizeof (struct scsi_pkt) + scsi_pkt_size())
```



```
1010 /*
1011  * Only one of below two conditions is satisfied, we
1012  * think the target is associated to the iport and
1013  * allow call into mptsas_probe_lun().
1014  * 1. physicalsport == physport
1015  * 2. (phymask & (1 << physport)) == 0
1016  * The condition #2 is because LSI uses lowest PHY
1017  * number as the value of physical port when auto port
1018  * configuration.
1019  */
1020 #define IS_SAME_PORT(physicalport, physport, phymask, dynamicport) \
1021         ((physicalport == physport) || (dynamicport && (phymask & \
1022         (1 << physport))))

1024 _NOTE(MUTEX_PROTECTS_DATA(mptsas::m_mutex, mptsas))
1025 _NOTE(SCHEME_PROTECTS_DATA("safe sharing", mptsas::m_next))
1026 _NOTE(SCHEME_PROTECTS_DATA("stable data", mptsas::m_dip mptsas::m_tran))
1027 _NOTE(SCHEME_PROTECTS_DATA("stable data", mptsas::m_kmem_cache))
1028 _NOTE(DATA_READABLE_WITHOUT_LOCK(mptsas::m_io_dma_attr.dma_attr_sgllen))
1029 _NOTE(DATA_READABLE_WITHOUT_LOCK(mptsas::m_devid))
1030 _NOTE(DATA_READABLE_WITHOUT_LOCK(mptsas::m_productid))
 604 _NOTE(DATA_READABLE_WITHOUT_LOCK(mptsas::m_port_type))
1031 _NOTE(DATA_READABLE_WITHOUT_LOCK(mptsas::m_mpxio_enable))
 606 _NOTE(DATA_READABLE_WITHOUT_LOCK(mptsas::m_ntargets))
1032 _NOTE(DATA_READABLE_WITHOUT_LOCK(mptsas::m_instance))

1034 /*
1035  * These should eventually migrate into the mpt header files
1036  * that may become the /kernel/misc/mpt module...
1037  */
1038 #define mptsas_init_std_hdr(hdl, mp, DevHandle, Lun, ChainOffset, Function) \
1039         mptsas_put_msg_DevHandle(hdl, mp, DevHandle); \
1040         mptsas_put_msg_ChainOffset(hdl, mp, ChainOffset); \
1041         mptsas_put_msg_Function(hdl, mp, Function); \
1042         mptsas_put_msg_Lun(hdl, mp, Lun)

1044 #define mptsas_put_msg_DevHandle(hdl, mp, val) \
1045         ddi_put16(hdl, &(mp)->DevHandle, (val))
1046 #define mptsas_put_msg_ChainOffset(hdl, mp, val) \
1047         ddi_put8(hdl, &(mp)->ChainOffset, (val))
1048 #define mptsas_put_msg_Function(hdl, mp, val) \
1049         ddi_put8(hdl, &(mp)->Function, (val))
1050 #define mptsas_put_msg_Lun(hdl, mp, val) \
1051         ddi_put8(hdl, &(mp)->LUN[1], (val))

1053 #define mptsas_get_msg_Function(hdl, mp) \
1054         ddi_get8(hdl, &(mp)->Function)

1056 #define mptsas_get_msg_MsgFlags(hdl, mp) \
1057         ddi_get8(hdl, &(mp)->MsgFlags)

1059 #define MPTSAS_ENABLE_DRWE(hdl) \
1060         ddi_put32(hdl->m_datap, &hdl->m_reg->WriteSequence, \
1061                 MPI2_WRSEQ_FLUSH_KEY_VALUE); \
1062         ddi_put32(hdl->m_datap, &hdl->m_reg->WriteSequence, \
1063                 MPI2_WRSEQ_1ST_KEY_VALUE); \
1064         ddi_put32(hdl->m_datap, &hdl->m_reg->WriteSequence, \
1065                 MPI2_WRSEQ_2ND_KEY_VALUE); \
1066         ddi_put32(hdl->m_datap, &hdl->m_reg->WriteSequence, \
1067                 MPI2_WRSEQ_3RD_KEY_VALUE); \
1068         ddi_put32(hdl->m_datap, &hdl->m_reg->WriteSequence, \
1069                 MPI2_WRSEQ_4TH_KEY_VALUE); \
1070         ddi_put32(hdl->m_datap, &hdl->m_reg->WriteSequence, \
1071                 MPI2_WRSEQ_5TH_KEY_VALUE); \
1072         ddi_put32(hdl->m_datap, &hdl->m_reg->WriteSequence, \
1073                 MPI2_WRSEQ_6TH_KEY_VALUE);
```

```
1075 /*
1076  * m_options flags
1077  */
1078 #define MPTSAS_OPT_PM           0x01    /* Power Management */
1079 #define MPTSAS_OPT_MSI          0x02    /* PCI MSI Interrupts */
1080 #define MPTSAS_OPT_MSI_X        0x04    /* PCI MSI-X Interrupts */
1081 #endif /* ! codereview */

1083 /*
1084  * m_softstate flags
1085  */
1086 #define MPTSAS_SS_DRAINING              0x02
1087 #define MPTSAS_SS_QUIESCED              0x04
1088 #define MPTSAS_SS_MSG_UNIT_RESET        0x08
1089 #define MPTSAS_DID_MSG_UNIT_RESET       0x10

1091 /*
1092  * m_dma_flags (allocated).
1093  */
1094 #define MPTSAS_REQ_FRAME        0x01
1095 #define MPTSAS_REQ_SENSE        0x02
1096 #define MPTSAS_REPLY_FRAME      0x04
1097 #define MPTSAS_FREE_QUEUE       0x08
1098 #define MPTSAS_POST_QUEUE       0x10

1100 /*
1101 #endif /* ! codereview */
1102  * regspec defines.
1103  */
1104 #define CONFIG_SPACE    0       /* regset[0] - configuration space */
1105 #define IO_SPACE        1       /* regset[1] - used for i/o mapped device */
1106 #define MEM_SPACE       2       /* regset[2] - used for memory mapped device */
1107 #define BASE_REG2       3       /* regset[3] - used for 875 scripts ram */

1109 /*
1110  * Handy constants
1111  */
1112 #define FALSE           0
1113 #define TRUE            1
1114 #define BLOCKED         2
1115 #endif /* ! codereview */
1116 #define UNDEFINED       -1
1117 #define FAILED          -2

1119 /*
1120  * power management.
1121  */
1122 #define MPTSAS_POWER_ON(mpt) { \
1123         pci_config_put16(mpt->m_config_handle, mpt->m_pmcsr_offset, \
1124             PCI_PMCSR_D0); \
1125         delay(drv_usectohz(10000)); \
1126         (void) pci_restore_config_regs(mpt->m_dip); \
1127         mptsas_setup_cmd_reg(mpt); \
1128 }

1130 #define MPTSAS_POWER_OFF(mpt) { \
1131         (void) pci_save_config_regs(mpt->m_dip); \
1132         pci_config_put16(mpt->m_config_handle, mpt->m_pmcsr_offset, \
1133             PCI_PMCSR_D3HOT); \
1134         mpt->m_power_level = PM_LEVEL_D3; \
1135 }

1137 /*
1138  * inq_dtype:
1139  * Bits 5 through 7 are the Peripheral Device Qualifier
```



```
1140  * 001b: device not connected to the LUN
1141  * Bits 0 through 4 are the Peripheral Device Type
1142  * 1fh: Unknown or no device type
1143  *
1144  * Although the inquiry may return success, the following value
1145  * means no valid LUN connected.
1146  */
1147 #define MPTSAS_VALID_LUN(sd_inq) \
1148         (((sd_inq->inq_dtype & 0xe0) != 0x20) && \
1149         ((sd_inq->inq_dtype & 0x1f) != 0x1f))

1151 /*
1152  * Default is to have 10 retries on receiving QFULL status and
1153  * each retry to be after 100 ms.
1154  */
1155 #define QFULL_RETRIES           10
1156 #define QFULL_RETRY_INTERVAL    100

1158 /*
1159  * Handy macros
1160  */
1161 #define Tgt(sp) ((sp)->cmd_pkt->pkt_address.a_target)
1162 #define Lun(sp) ((sp)->cmd_pkt->pkt_address.a_lun)

1164 #define IS_HEX_DIGIT(n) (((n) >= '0' && (n) <= '9') || \
1165         ((n) >= 'a' && (n) <= 'f') || ((n) >= 'A' && (n) <= 'F'))

1167 /*
1168  * poll time for mptsas_pollret() and mptsas_wait_intr()
1169  */
1170 #define MPTSAS_POLL_TIME        30000   /* 30 seconds */

1172 /*
1173  * default time for mptsas_do_passthru
1174  */
1175 #define MPTSAS_PASS_THRU_TIME_DEFAULT   60      /* 60 seconds */

1177 /*
1178  * macro to return the effective address of a given per-target field
1179  */
1180 #define EFF_ADDR(start, offset)         ((start) + (offset))

1182 #define SDEV2ADDR(devp)         (&((devp)->sd_address))
1183 #define SDEV2TRAN(devp)         ((devp)->sd_address.a_hba_tran)
1184 #define PKT2TRAN(pkt)           ((pkt)->pkt_address.a_hba_tran)
1185 #define ADDR2TRAN(ap)           ((ap)->a_hba_tran)
1186 #define DIP2TRAN(dip)           (ddi_get_driver_private(dip))


1189 #define TRAN2MPT(hba)           ((mptsas_t *)(hba)->tran_hba_private)
1190 #define DIP2MPT(dip)            (TRAN2MPT((scsi_hba_tran_t *)DIP2TRAN(dip)))
1191 #define SDEV2MPT(sd)            (TRAN2MPT(SDEV2TRAN(sd)))
1192 #define PKT2MPT(pkt)            (TRAN2MPT(PKT2TRAN(pkt)))

1194 #define ADDR2MPT(ap)            (TRAN2MPT(ADDR2TRAN(ap)))

1196 #define POLL_TIMEOUT            (2 * SCSI_POLL_TIMEOUT * 1000000)
1197 #define SHORT_POLL_TIMEOUT      (1000000)       /* in usec, about 1 secs */
1198 #define MPTSAS_QUIESCE_TIMEOUT  1               /* 1 sec */
1199 #define MPTSAS_PM_IDLE_TIMEOUT  60              /* 60 seconds */

1201 #define MPTSAS_GET_ISTAT(mpt)  (ddi_get32((mpt)->m_datap, \
1202                         &(mpt)->m_reg->HostInterruptStatus))

1204 #define MPTSAS_SET_SIGP(P) \
1205                 ClrSetBits(mpt->m_devaddr + NREG_ISTAT, 0, NB_ISTAT_SIGP)
```

```
#define MPTSAS_RESET_SIGP(P) (void) ddi_get8(mpt->m_datap, \
                        (uint8_t *)(mpt->m_devaddr + NREG_CTEST2))

#define MPTSAS_GET_INTCODE(P) (ddi_get32(mpt->m_datap, \
                        (uint32_t *)(mpt->m_devaddr + NREG_DSPS)))


#define MPTSAS_START_CMD(mpt, req_desc) \
        ddi_put64(mpt->m_datap, \
            (uint64_t *)(void *)&mpt->m_reg->RequestDescriptorPostLow, \
            req_desc);                                                 \
        atomic_inc_32(&mpt->m_ncstarted)

#define MPTSAS_START_CMD(mpt, req_desc_lo, req_desc_hi) \
        ddi_put32(mpt->m_datap, &mpt->m_reg->RequestDescriptorPostLow,\
            req_desc_lo);\
        ddi_put32(mpt->m_datap, &mpt->m_reg->RequestDescriptorPostHigh,\
            req_desc_hi);

#define INTPENDING(mpt) \
        (MPTSAS_GET_ISTAT(mpt) & MPI2_HIS_REPLY_DESCRIPTOR_INTERRUPT)

/*
 * Mask all interrupts to disable
 */
#define MPTSAS_DISABLE_INTR(mpt)        \
        ddi_put32((mpt)->m_datap, &(mpt)->m_reg->HostInterruptMask, \
            (MPI2_HIM_RIM | MPI2_HIM_DIM | MPI2_HIM_RESET_IRQ_MASK))

/*
 * Mask Doorbell and Reset interrupts to enable reply desc int.
 */
#define MPTSAS_ENABLE_INTR(mpt) \
        ddi_put32(mpt->m_datap, &mpt->m_reg->HostInterruptMask, \
        (MPI2_HIM_DIM | MPI2_HIM_RESET_IRQ_MASK))

#define MPTSAS_GET_NEXT_REPLY(rpqp, index)  \
        &((uint64_t *)(void *)rpqp->rpq_queue)[index]
#define MPTSAS_GET_NEXT_REPLY(mpt, index)  \
        &((uint64_t *)(void *)mpt->m_post_queue)[index]

#define MPTSAS_GET_NEXT_FRAME(mpt, SMID) \
        (mpt->m_req_frame + (mpt->m_req_frame_size * SMID))

#define ClrSetBits32(hdl, reg, clr, set) \
        ddi_put32(hdl, (reg), \
            ((ddi_get32(mpt->m_datap, (reg)) & ~(clr)) | (set)))

#define ClrSetBits(reg, clr, set) \
        ddi_put8(mpt->m_datap, (uint8_t *)(reg), \
                ((ddi_get8(mpt->m_datap, (uint8_t *)(reg)) & ~(clr)) | (set)))

#define MPTSAS_WAITQ_RM(mpt, cmdp)      \
        if ((cmdp = mpt->m_waitq) != NULL) { \
                /* If the queue is now empty fix the tail pointer */    \
                if ((mpt->m_waitq = cmdp->cmd_linkp) == NULL) \
                        mpt->m_waitqtail = &mpt->m_waitq; \
                cmdp->cmd_linkp = NULL; \
                cmdp->cmd_queued = FALSE; \
        }

#define MPTSAS_TX_WAITQ_RM(mpt, cmdp)   \
        if ((cmdp = mpt->m_tx_waitq) != NULL) { \
                /* If the queue is now empty fix the tail pointer */    \
                if ((mpt->m_tx_waitq = cmdp->cmd_linkp) == NULL) \
```



```
                        mpt->m_tx_waitqtail = &mpt->m_tx_waitq; \
                cmdp->cmd_linkp = NULL; \
                cmdp->cmd_queued = FALSE; \
        }

/*
 * defaults for the global properties
 */
#define DEFAULT_SCSI_OPTIONS    SCSI_OPTIONS_DR
#define DEFAULT_TAG_AGE_LIMIT   2
#define DEFAULT_WD_TICK         1
#define DEFAULT_WD_TICK         10

/*
 * invalid hostid.
 */
#define MPTSAS_INVALID_HOSTID  -1

/*
 * Get/Set hostid from SCSI port configuration page
 */
#define MPTSAS_GET_HOST_ID(configuration) (configuration & 0xFF)
#define MPTSAS_SET_HOST_ID(hostid) (hostid | ((1 << hostid) << 16))

/*
 * Config space.
 */
#define MPTSAS_LATENCY_TIMER    0x40

/*
 * Offset to firmware version
 */
#define MPTSAS_FW_VERSION_OFFSET        9

/*
 * Offset and masks to get at the ProductId field
 */
#define MPTSAS_FW_PRODUCTID_OFFSET      8
#define MPTSAS_FW_PRODUCTID_MASK        0xFFFF0000
#define MPTSAS_FW_PRODUCTID_SHIFT       16

/*
 * Subsystem ID for HBAs.
 */
#define MPTSAS_HBA_SUBSYSTEM_ID    0x10C0
#define MPTSAS_RHEA_SUBSYSTEM_ID        0x10B0

/*
 * reset delay tick
 */
#define MPTSAS_WATCH_RESET_DELAY_TICK 50        /* specified in milli seconds */

/*
 * Ioc reset return values
 */
#define MPTSAS_RESET_FAIL       -1
#define MPTSAS_NO_RESET         0
#define MPTSAS_SUCCESS_HARDRESET        1
#define MPTSAS_SUCCESS_MUR      2

/*
 * throttle support.
 */
#define MAX_THROTTLE    32
#define HOLD_THROTTLE   0
#define DRAIN_THROTTLE  -1
```

```
#define QFULL_THROTTLE  -2

/*
 * Passthrough/config request flags
 */
#define MPTSAS_DATA_ALLOCATED           0x0001
#define MPTSAS_DATAOUT_ALLOCATED        0x0002
#define MPTSAS_REQUEST_POOL_CMD         0x0004
#define MPTSAS_ADDRESS_REPLY            0x0008
#define MPTSAS_CMD_TIMEOUT              0x0010

/*
 * response code tlr flag
 */
#define MPTSAS_SCSI_RESPONSE_CODE_TLR_OFF       0x02

/*
 * System Events
 */
#ifndef DDI_VENDOR_LSI
#define DDI_VENDOR_LSI  "LSI"
#endif  /* DDI_VENDOR_LSI */

/*
 * Shared functions
 */
int mptsas_save_cmd(struct mptsas *mpt, struct mptsas_cmd *cmd);
void mptsas_remove_cmd(mptsas_t *mpt, mptsas_cmd_t *cmd);
void mptsas_waitq_add(mptsas_t *mpt, mptsas_cmd_t *cmd);
void mptsas_log(struct mptsas *mpt, int level, char *fmt, ...);
int mptsas_poll(mptsas_t *mpt, mptsas_cmd_t *poll_cmd, int polltime);
int mptsas_do_dma(mptsas_t *mpt, uint32_t size, int var, int (*callback)());
int mptsas_send_config_request_msg(mptsas_t *mpt, uint8_t action,
        uint8_t pagetype, uint32_t pageaddress, uint8_t pagenumber,
        uint8_t pageversion, uint8_t pagelength, uint32_t
        SGEflagslength, uint32_t SGEaddress32);
int mptsas_send_extended_config_request_msg(mptsas_t *mpt, uint8_t action,
        uint8_t extpagetype, uint32_t pageaddress, uint8_t pagenumber,
        uint8_t pageversion, uint16_t extpagelength,
        uint32_t SGEflagslength, uint32_t SGEaddress32);
int mptsas_update_flash(mptsas_t *mpt, caddr_t ptrbuffer, uint32_t size,
        uint8_t type, int mode);
int mptsas_check_flash(mptsas_t *mpt, caddr_t origfile, uint32_t size,
        uint8_t type, int mode);
int mptsas_download_firmware();
int mptsas_can_download_firmware();
int mptsas_dma_alloc(mptsas_t *mpt, mptsas_dma_alloc_state_t *dma_statep);
void mptsas_dma_free(mptsas_dma_alloc_state_t *dma_statep);
mptsas_phymask_t mptsas_physport_to_phymask(mptsas_t *mpt, uint8_t physport);
void mptsas_fma_check(mptsas_t *mpt, mptsas_cmd_t *cmd);
int mptsas_check_acc_handle(ddi_acc_handle_t handle);
int mptsas_check_dma_handle(ddi_dma_handle_t handle);
void mptsas_fm_ereport(mptsas_t *mpt, char *detail);
int mptsas_dma_addr_create(mptsas_t *mpt, ddi_dma_attr_t dma_attr,
    ddi_dma_handle_t *dma_hdp, ddi_acc_handle_t *acc_hdp, caddr_t *dma_memp,
    uint32_t alloc_size, ddi_dma_cookie_t *cookiep);
void mptsas_dma_addr_destroy(ddi_dma_handle_t *, ddi_acc_handle_t *);

/*
 * impl functions
 */
int mptsas_ioc_wait_for_response(mptsas_t *mpt);
int mptsas_ioc_wait_for_doorbell(mptsas_t *mpt);
int mptsas_ioc_reset(mptsas_t *mpt, int);
int mptsas_send_handshake_msg(mptsas_t *mpt, caddr_t memp, int numbytes,
    ddi_acc_handle_t accessp);
```



```
int mptsas_get_handshake_msg(mptsas_t *mpt, caddr_t memp, int numbytes,
    ddi_acc_handle_t accessp);
int mptsas_send_config_request_msg(mptsas_t *mpt, uint8_t action,
    uint8_t pagetype, uint32_t pageaddress, uint8_t pagenumber,
    uint8_t pageversion, uint8_t pagelength, uint32_t SGEflagslength,
    uint64_t SGEaddress);
    uint32_t SGEaddress32);
int mptsas_send_extended_config_request_msg(mptsas_t *mpt, uint8_t action,
    uint8_t extpagetype, uint32_t pageaddress, uint8_t pagenumber,
    uint8_t pageversion, uint16_t extpagelength,
    uint32_t SGEflagslength, uint64_t SGEaddress);
    uint32_t SGEflagslength, uint32_t SGEaddress32);

int mptsas_request_from_pool(mptsas_t *mpt, mptsas_cmd_t **cmd,
    struct scsi_pkt **pkt);
void mptsas_return_to_pool(mptsas_t *mpt, mptsas_cmd_t *cmd);
void mptsas_destroy_ioc_event_cmd(mptsas_t *mpt);
void mptsas_start_config_page_access(mptsas_t *mpt, mptsas_cmd_t *cmd);
int mptsas_access_config_page(mptsas_t *mpt, uint8_t action, uint8_t page_type,
    uint8_t page_number, uint32_t page_address, int (*callback) (mptsas_t *,
    caddr_t, ddi_acc_handle_t, uint16_t, uint32_t, va_list), ...);

int mptsas_ioc_task_management(mptsas_t *mpt, int task_type,
    uint16_t dev_handle, int lun, uint8_t *reply, uint32_t reply_size,
    int mode);
int mptsas_send_event_ack(mptsas_t *mpt, uint32_t event, uint32_t eventcntx);
void mptsas_send_pending_event_ack(mptsas_t *mpt);
void mptsas_set_throttle(struct mptsas *mpt, mptsas_target_t *ptgt, int what);
int mptsas_restart_ioc(mptsas_t *mpt);
void mptsas_update_driver_data(struct mptsas *mpt);
uint64_t mptsas_get_sata_guid(mptsas_t *mpt, mptsas_target_t *ptgt, int lun);

/*
 * init functions
 */
int mptsas_ioc_get_facts(mptsas_t *mpt);
int mptsas_ioc_get_port_facts(mptsas_t *mpt, int port);
int mptsas_ioc_enable_port(mptsas_t *mpt);
int mptsas_ioc_enable_event_notification(mptsas_t *mpt);
int mptsas_ioc_init(mptsas_t *mpt);

/*
 * configuration pages operation
 */
int mptsas_get_sas_device_page0(mptsas_t *mpt, uint32_t page_address,
    uint16_t *dev_handle, uint64_t *sas_wwn, uint32_t *dev_info,
    uint8_t *physport, uint8_t *phynum, uint16_t *pdevhandle,
    uint16_t *slot_num, uint16_t *enclosure, uint16_t *io_flags);
    uint16_t *slot_num, uint16_t *enclosure);
int mptsas_get_sas_io_unit_page(mptsas_t *mpt);
int mptsas_get_sas_io_unit_page_hndshk(mptsas_t *mpt);
int mptsas_get_sas_expander_page0(mptsas_t *mpt, uint32_t page_address,
    mptsas_smp_t *info);
int mptsas_set_ioc_params(mptsas_t *mpt);
int mptsas_get_manufacture_page5(mptsas_t *mpt);
int mptsas_get_sas_port_page0(mptsas_t *mpt, uint32_t page_address,
    uint64_t *sas_wwn, uint8_t *portwidth);
int mptsas_get_bios_page3(mptsas_t *mpt,  uint32_t *bios_version);
int
mptsas_get_sas_phy_page0(mptsas_t *mpt, uint32_t page_address,
    smhba_info_t *info);
int
mptsas_get_sas_phy_page1(mptsas_t *mpt, uint32_t page_address,
    smhba_info_t *info);
int
mptsas_get_manufacture_page0(mptsas_t *mpt);
```

```
1450 void
1451 mptsas_create_phy_stats(mptsas_t *mpt, char *iport, dev_info_t *dip);
1452 void mptsas_destroy_phy_stats(mptsas_t *mpt);
1453 int mptsas_smhba_phy_init(mptsas_t *mpt);
1454 /*
1455  * RAID functions
1456  */
1457 int mptsas_get_raid_settings(mptsas_t *mpt, mptsas_raidvol_t *raidvol);
1458 int mptsas_get_raid_info(mptsas_t *mpt);
1459 int mptsas_get_physdisk_settings(mptsas_t *mpt, mptsas_raidvol_t *raidvol,
1460     uint8_t physdisknum);
1461 int mptsas_delete_volume(mptsas_t *mpt, uint16_t volid);
1462 void mptsas_raid_action_system_shutdown(mptsas_t *mpt);

1464 #define MPTSAS_IOCSTATUS(status) (status & MPI2_IOCSTATUS_MASK)
1465 /*
1466  * debugging.
1467  */
1468 #if defined(MPTSAS_DEBUG)

1470 void mptsas_printf(char *fmt, ...);
1471 void mptsas_debug_log(char *fmt, ...);
1472 #endif /* ! codereview */

1474 #define MPTSAS_DBGPR(m, args)   \
1475         if (mptsas_debug_flags & (m)) \
1476                 mptsas_printf args;   \
1477         if (~mptsas_dbglog_imask & (m)) \
1478                 mptsas_debug_log args
 919                 mptsas_printf args
1479 #else   /* ! defined(MPTSAS_DEBUG) */
1480 #define MPTSAS_DBGPR(m, args)
1481 #endif  /* defined(MPTSAS_DEBUG) */

1483 #define NDBG0(args)     MPTSAS_DBGPR(0x01, args)        /* init */
1484 #define NDBG1(args)     MPTSAS_DBGPR(0x02, args)        /* normal running */
1485 #define NDBG2(args)     MPTSAS_DBGPR(0x04, args)        /* property handling */
1486 #define NDBG3(args)     MPTSAS_DBGPR(0x08, args)        /* pkt handling */

1488 #define NDBG4(args)     MPTSAS_DBGPR(0x10, args)        /* kmem alloc/free */
1489 #define NDBG5(args)     MPTSAS_DBGPR(0x20, args)        /* polled cmds */
1490 #define NDBG6(args)     MPTSAS_DBGPR(0x40, args)        /* interrupt setup */
 931 #define NDBG6(args)     MPTSAS_DBGPR(0x40, args)        /* interrupts */
1491 #define NDBG7(args)     MPTSAS_DBGPR(0x80, args)        /* queue handling */

1493 #define NDBG8(args)     MPTSAS_DBGPR(0x0100, args)      /* arq */
1494 #define NDBG9(args)     MPTSAS_DBGPR(0x0200, args)      /* Tagged Q'ing */
1495 #define NDBG10(args)    MPTSAS_DBGPR(0x0400, args)      /* halting chip */
1496 #define NDBG11(args)    MPTSAS_DBGPR(0x0800, args)      /* power management */

1498 #define NDBG12(args)    MPTSAS_DBGPR(0x1000, args)      /* enumeration */
1499 #define NDBG13(args)    MPTSAS_DBGPR(0x2000, args)      /* configuration page */
1500 #define NDBG14(args)    MPTSAS_DBGPR(0x4000, args)      /* LED control */
1501 #define NDBG15(args)    MPTSAS_DBGPR(0x8000, args)      /* Passthrough */
 942 #define NDBG15(args)    MPTSAS_DBGPR(0x8000, args)

1503 #define NDBG16(args)    MPTSAS_DBGPR(0x010000, args)    /* SAS Broadcasts */
 944 #define NDBG16(args)    MPTSAS_DBGPR(0x010000, args)
1504 #define NDBG17(args)    MPTSAS_DBGPR(0x020000, args)    /* scatter/gather */
1505 #define NDBG18(args)    MPTSAS_DBGPR(0x040000, args)    /* Interrupts */
 946 #define NDBG18(args)    MPTSAS_DBGPR(0x040000, args)
1506 #define NDBG19(args)    MPTSAS_DBGPR(0x080000, args)    /* handshaking */

1508 #define NDBG20(args)    MPTSAS_DBGPR(0x100000, args)    /* events */
1509 #define NDBG21(args)    MPTSAS_DBGPR(0x200000, args)    /* dma */
1510 #define NDBG22(args)    MPTSAS_DBGPR(0x400000, args)    /* reset */
```



```
1511 #define NDBG23(args)    MPTSAS_DBGPR(0x800000, args)    /* abort */

1513 #define NDBG24(args)    MPTSAS_DBGPR(0x1000000, args)   /* capabilities */
1514 #define NDBG25(args)    MPTSAS_DBGPR(0x2000000, args)   /* flushing */
1515 #define NDBG26(args)    MPTSAS_DBGPR(0x4000000, args)
1516 #define NDBG27(args)    MPTSAS_DBGPR(0x8000000, args)

1518 #define NDBG28(args)    MPTSAS_DBGPR(0x10000000, args)  /* hotplug */
1519 #define NDBG29(args)    MPTSAS_DBGPR(0x20000000, args)  /* timeouts */
1520 #define NDBG30(args)    MPTSAS_DBGPR(0x40000000, args)  /* mptsas_watch */
1521 #define NDBG31(args)    MPTSAS_DBGPR(0x80000000, args)  /* negotations */

1523 /*
1524  * auto request sense
1525  */
1526 #define RQ_MAKECOM_COMMON(pkt, flag, cmd) \
1527         (pkt)->pkt_flags = (flag), \
1528         ((union scsi_cdb *)(pkt)->pkt_cdbp)->scc_cmd = (cmd), \
1529         ((union scsi_cdb *)(pkt)->pkt_cdbp)->scc_lun = \
1530             (pkt)->pkt_address.a_lun

1532 #define RQ_MAKECOM_G0(pkt, flag, cmd, addr, cnt) \
1533         RQ_MAKECOM_COMMON((pkt), (flag), (cmd)), \
1534         FORMG0ADDR(((union scsi_cdb *)(pkt)->pkt_cdbp), (addr)), \
1535         FORMG0COUNT(((union scsi_cdb *)(pkt)->pkt_cdbp), (cnt))


1538 #ifdef  __cplusplus
1539 }
1540 #endif

1542 #endif  /* _SYS_SCSI_ADAPTERS_MPTSAS3_VAR_H */
 983 #endif  /* _SYS_SCSI_ADAPTERS_MPTVAR_H */
```

```
**********************************************************
    2972 Thu Jun 12 17:42:19 2014
new/usr/src/uts/intel/mpt_sas3/Makefile
Changes to enable driver to compile.
Header paths, object lists, etc.
**********************************************************
```

```
#
# 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.
#
#       This makefile drives the production of the mpt_sas3 driver
#       kernel module.
#       This makefile drives the production of the mpt_sas driver kernel module.
#
#       intel architecture dependent
#


#
#       Paths to the base of the uts directory trees
#
UTSBASE   = ../../../../src/uts

#
#       Define the module and object file sets.
#
MODULE          = mpt_sas3
OBJECTS         = $(MPTSAS3_OBJS:%=$(OBJS_DIR)/%)
LINTS           = $(MPTSAS3_OBJS:%.o=$(LINTS_DIR)/%.ln)
MODULE          = mpt_sas
OBJECTS         = $(MPTSAS_OBJS:%=$(OBJS_DIR)/%)
LINTS           = $(MPTSAS_OBJS:%.o=$(LINTS_DIR)/%.ln)
ROOTMODULE      = $(ROOT_DRV_DIR)/$(MODULE)
CONF_SRCDIR     = $(UTSBASE)/common/io/scsi/adapters/mpt_sas3/
WARLOCK_OUT     = $(MPTSAS3_OBJS:%.o=%.ll)
CONF_SRCDIR     = $(UTSBASE)/common/io/scsi/adapters/mpt_sas/
WARLOCK_OUT     = $(MPTSAS_OBJS:%.o=%.ll)
WARLOCK_OK      = $(MODULE).ok
WLCMD_DIR       = $(UTSBASE)/common/io/warlock

#
#       Kernel Module Dependencies
#
LDFLAGS += -dy -Nmisc/scsi -Ndrv/scsi_vhci

#
#       Define targets
#
```



```
ALL_TARGET      = $(BINARY) $(CONFMOD)
LINT_TARGET     = $(MODULE).lint
INSTALL_TARGET  = $(BINARY) $(ROOTMODULE) $(ROOT_CONFFILE)

#
#       Include common rules.
#
include $(UTSBASE)/intel/Makefile.intel

CERRWARN        += -_gcc=-Wno-parentheses
CERRWARN        += -_gcc=-Wno-uninitialized
CERRWARN        += -_gcc=-Wno-unused-label
CERRWARN        += -_gcc=-Wno-switch

#
#       Default build targets.
#
.KEEP_STATE:

all:            $(ALL_DEPS)

def:            $(DEF_DEPS)

clean:          $(CLEAN_DEPS)
                $(RM) $(WARLOCK_OUT) $(WARLOCK_OK)

clobber:        $(CLOBBER_DEPS)
                $(RM) $(WARLOCK_OUT) $(WARLOCK_OK)

lint:           $(LINT_DEPS)

modlintlib:     $(MODLINTLIB_DEPS)

clean.lint:     $(CLEAN_LINT_DEPS)

install:        $(INSTALL_DEPS)

#
#       Include common targets.
#
include $(UTSBASE)/intel/Makefile.targ

#
#       Defines for local commands.
#
WARLOCK         = warlock
WLCC            = wlcc
TOUCH           = touch
TEST            = test

#
#       lock_lint rules
#
SCSI_FILES = $(SCSI_OBJS:%.o= -l $(UTSBASE)/intel/scsi/%.ll)

warlock: $(WARLOCK_OK)

$(WARLOCK_OK): $(WARLOCK_OUT) warlock_ddi.files scsi.files \
        $(WLCMD_DIR)/mptsas.wlcmd
        $(WARLOCK) -c $(WLCMD_DIR)/mptsas.wlcmd $(WARLOCK_OUT) \
        $(SCSI_FILES) \
        $(UTSBASE)/intel/warlock/scsi.ll \
        -l $(UTSBASE)/intel/warlock/ddi_dki_impl.ll
        $(TOUCH) $@

%.ll: $(UTSBASE)/common/io/scsi/adapters/mpt_sas3/%.c
```

```
119 %.ll: $(UTSBASE)/common/io/scsi/adapters/mpt_sas/%.c
121         $(WLCC) $(CPPFLAGS) -DDEBUG -o $@ $<

123 warlock_ddi.files:
124         @cd $(UTSBASE)/intel/warlock; pwd; $(MAKE) warlock

126 scsi.files:
127         @cd $(UTSBASE)/intel/scsi; pwd; $(MAKE) warlock
```

```
**********************************************************
    3055 Thu Jun 12 17:42:20 2014
new/usr/src/uts/sparc/mpt_sas3/Makefile
Changes to enable driver to compile.
Header paths, object lists, etc.
**********************************************************
```

```
#
# 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.
#
#       This makefile drives the production of the mpt_sas3 driver
#       kernel module.
#       This makefile drives the production of the mpt_sas driver kernel module.
#
#       Sparc architecture dependent
#


#
#       Path to the base of the uts directory tree (usually /usr/src/uts).
#
UTSBASE   = ../../../../src/uts

#
#       Define the module and object file sets.
#
MODULE          = mpt_sas3
OBJECTS         = $(MPTSAS3_OBJS:%=$(OBJS_DIR)/%)
LINTS           = $(MPTSAS3_OBJS:%.o=$(LINTS_DIR)/%.ln)
MODULE          = mpt_sas
OBJECTS         = $(MPTSAS_OBJS:%=$(OBJS_DIR)/%)
LINTS           = $(MPTSAS_OBJS:%.o=$(LINTS_DIR)/%.ln)
ROOTMODULE      = $(ROOT_DRV_DIR)/$(MODULE)
CONF_SRCDIR     = $(UTSBASE)/common/io/scsi/adapters/mpt_sas3
WARLOCK_OUT     = $(MPTSAS3_OBJS:%.o=%.ll)
CONF_SRCDIR     = $(UTSBASE)/common/io/scsi/adapters/mpt_sas
WARLOCK_OUT     = $(MPTSAS_OBJS:%.o=%.ll)
WARLOCK_OK      = $(MODULE).ok
WLCMD_DIR       = $(UTSBASE)/common/io/warlock

#
#       Kernel Module Dependencies
#
LDFLAGS += -dy -Nmisc/scsi -Ndrv/scsi_vhci

#
#       Define targets
#
```



```
ALL_TARGET      = $(BINARY) $(CONFMOD)
LINT_TARGET     = $(MODULE).lint
INSTALL_TARGET  = $(BINARY) $(ROOTMODULE) $(ROOT_CONFFILE)

#
#       Include common rules.
#
include $(UTSBASE)/sparc/Makefile.sparc

#
# lint pass one enforcement
#
CFLAGS += $(CCVERBOSE)

CERRWARN        += -_gcc=-Wno-parentheses
CERRWARN        += -_gcc=-Wno-uninitialized
CERRWARN        += -_gcc=-Wno-unused-label
CERRWARN        += -_gcc=-Wno-switch

#
#       Default build targets.
#
.KEEP_STATE:

all:            $(ALL_DEPS)

def:            $(DEF_DEPS)

clean:          $(CLEAN_DEPS)
                $(RM) $(WARLOCK_OUT) $(WARLOCK_OK)

clobber:        $(CLOBBER_DEPS)
                $(RM) $(WARLOCK_OUT) $(WARLOCK_OK)

lint:           $(LINT_DEPS)

modlintlib:     $(MODLINTLIB_DEPS)

clean.lint:     $(CLEAN_LINT_DEPS)

install:        $(INSTALL_DEPS)

#
#       Include common targets.
#
include $(UTSBASE)/sparc/Makefile.targ

#
#       Defines for local commands.
#
WARLOCK         = warlock
WLCC            = wlcc
TOUCH           = touch
TEST            = test

#
#       lock_lint rules
#
SCSI_FILES = $(SCSI_OBJS:%.o= -l $(UTSBASE)/sparc/scsi/%.ll)

warlock: $(WARLOCK_OK)

$(WARLOCK_OK): $(WARLOCK_OUT) warlock_ddi.files scsi.files \
        $(WLCMD_DIR)/mptsas3.wlcmd
        $(WARLOCK) -c $(WLCMD_DIR)/mptsas3.wlcmd $(WARLOCK_OUT) \
        $(WLCMD_DIR)/mptsas.wlcmd
```

```
118         $(WARLOCK) -c $(WLCMD_DIR)/mptsas.wlcmd $(WARLOCK_OUT) \
120         $(UTSBASE)/sparc/warlock/scsi.ll \
121         $(SCSI_FILES) \
122         -l $(UTSBASE)/sparc/warlock/ddi_dki_impl.ll
123         $(TOUCH) $@

125 %.ll: $(UTSBASE)/common/io/scsi/adapters/mpt_sas3/%.c
124 %.ll: $(UTSBASE)/common/io/scsi/adapters/mpt_sas/%.c
126         $(WLCC) $(CPPFLAGS) -DDEBUG -o $@ $<

128 warlock_ddi.files:
129         @cd $(UTSBASE)/sparc/warlock; pwd; $(MAKE) warlock

131 scsi.files:
132         @cd $(UTSBASE)/sparc/scsi; pwd; $(MAKE) warlock
```