```
**********************************************************
    1862 Thu Jun 12 17:28:20 2014
new/usr/src/cmd/mdb/Makefile.common
4546 mpt_sas needs enhancing to support LSI MPI2.5
**********************************************************
#
# 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 17:28:21 2014
new/usr/src/cmd/mdb/common/modules/mpt_sas/mpt_sas.c
4546 mpt_sas needs enhancing to support LSI MPI2.5
**********************************************************
______unchanged_portion_omitted_

void
mdi_info(struct mptsas *mp, int target)
mdi_info(struct mptsas m, int target)
{
        struct  dev_info        d;
        struct  mdi_phci        p;

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

        if (MDI_PHCI(&d)) {
                if (mdb_vread(&p, sizeof (p), (uintptr_t)d.devi_mdi_xhci)
                    == -1) {
                        mdb_warn("couldn't read m_dip.devi_mdi_xhci");
                        return;
                }
                if (p.ph_path_head)
                        mdb_pwalk("mdipi_phci_list", (mdb_walk_cb_t)mdi_info_cb,
                            &target, (uintptr_t)p.ph_path_head);
                return;
        }
}
______unchanged_portion_omitted_

void
display_targets(struct mptsas *mp, uint_t verbose)
display_targets(struct mptsas *mp)
{
        mptsas_target_t *ptgt;
        mptsas_smp_t *psmp;
        int loop, comma;
#endif /* ! codereview */

        mdb_printf("\n");
        mdb_printf("slot devhdl      wwn     ncmds throttle   "
            "dr_flag  dups\n");
        mdb_printf("------------------------------"
            "--------------------------------\n");
        for (ptgt = (mptsas_target_t *)krefhash_first(
            (uintptr_t)mp->m_targets);
        mdb_printf("The SCSI target information\n");
        for (ptgt = (mptsas_target_t *)krefhash_first((uintptr_t)mp->m_targets);
            ptgt != NULL;
            ptgt = krefhash_next((uintptr_t)mp->m_targets, ptgt)) {
                if (ptgt->m_addr.mta_wwn ||
                    ptgt->m_deviceinfo) {
                        mdb_printf("%4d ", ptgt->m_slot_num);
                        mdb_printf("%4d ", ptgt->m_devhdl);
                        if (ptgt->m_addr.mta_wwn)
                mdb_printf("\n");
                mdb_printf("devhdl %x, sasaddress %"PRIx64", phymask %x,"
                    "devinfo %x\n", ptgt->m_devhdl, ptgt->m_addr.mta_wwn,
                    ptgt->m_addr.mta_phymask, ptgt->m_deviceinfo);
                mdb_printf("throttle %x, dr_flag %x, m_t_ncmds %x, "
                    "enclosure %x, slot_num %x\n", ptgt->m_t_throttle,
                    ptgt->m_dr_flag, ptgt->m_t_ncmds, ptgt->m_enclosure,
```



```
                        ptgt->m_slot_num);
        }

        mdb_printf("\n");
        mdb_printf("The smp child information\n");
        for (psmp = (mptsas_smp_t *)krefhash_first(
            (uintptr_t)mp->m_smp_targets);
            psmp != NULL;
            psmp = krefhash_next((uintptr_t)mp->m_smp_targets, psmp)) {
                mdb_printf("\n");
                mdb_printf("devhdl %x, sasaddress %"PRIx64", phymask %x \n",
                    psmp->m_devhdl, psmp->m_addr.mta_wwn,
                    psmp->m_addr.mta_phymask);
        }
        mdb_printf("\n");
#if 0
        mdb_printf("targ         wwn      ncmds throttle "
            "dr_flag  timeout  dups\n");
        mdb_printf("------------------------------"
            "--------------------------------\n");
        for (i = 0; i < MPTSAS_MAX_TARGETS; i++) {
                if (s->m_target[i].m_addr.mta_wwn ||
                    s->m_target[i].m_deviceinfo) {
                        mdb_printf("%4d ", i);
                        if (s->m_target[i].m_addr.mta_wwn)
                                mdb_printf("%"PRIx64" ",
                                        ptgt->m_addr.mta_wwn);
                        mdb_printf("%3d", ptgt->m_t_ncmds);
                        switch (ptgt->m_t_throttle) {
                                        s->m_target[i].m_addr.mta_wwn);
                        mdb_printf("%3d", s->m_target[i].m_t_ncmds);
                        switch (s->m_target[i].m_t_throttle) {
                                case QFULL_THROTTLE:
                                        mdb_printf("   QFULL ");
                                        break;
                                case DRAIN_THROTTLE:
                                        mdb_printf("   DRAIN ");
                                        break;
                                case HOLD_THROTTLE:
                                        mdb_printf("    HOLD ");
                                        break;
                                case MAX_THROTTLE:
                                        mdb_printf("     MAX ");
                                        break;
                                case CHOKE_THROTTLE:
                                        mdb_printf("   CHOKE ");
                                        break;
                                default:
                                        mdb_printf("%8d ",
                                            ptgt->m_t_throttle);
                                            s->m_target[i].m_t_throttle);
                        }
                        switch (ptgt->m_dr_flag) {
                        switch (s->m_target[i].m_dr_flag) {
                                case MPTSAS_DR_INACTIVE:
                                        mdb_printf("  INACTIVE ");
                                        break;
                                case MPTSAS_DR_INTRANSITION:
                                        mdb_printf("TRANSITION ");
                                case MPTSAS_DR_PRE_OFFLINE_TIMEOUT:
                                        mdb_printf("   TIMEOUT ");
                                        break;
                                case MPTSAS_DR_PRE_OFFLINE_TIMEOUT_NO_CANCEL:
                                        mdb_printf("TIMEOUT_NC ");
                                        break;
                                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++)
```

```
                if (mdb_vread(&c, sizeof (mptsas_cmd_t), (uintptr_t)q) == -1) {
                        mdb_warn("couldn't follow m_doneq");
                        rv = DCMD_ERR;
                        goto exit;
                }

        for (ptgt = (mptsas_target_t *)krefhash_first(
            (uintptr_t)mp->m_targets);
            ptgt != NULL;
            ptgt = krefhash_next((uintptr_t)mp->m_targets, ptgt)) {
                if (ptgt->m_addr.mta_wwn ||
                    ptgt->m_deviceinfo) {
                        ncmds += ptgt->m_t_ncmds;
                }
        }
        for (i = 0; i < MPTSAS_MAX_TARGETS; i++)
                ncmds += s->m_target[i].m_t_ncmds;

        mdb_printf("\n");
        mdb_printf("   mpt.  slot               mptsas_slots     slot");
        mdb_printf("\n");
        mdb_printf("m_ncmds total"
            " targ throttle m_t_ncmds targ_tot wq dq");
        mdb_printf("\n");
        mdb_printf("----------------------------------------------------");
        mdb_printf("\n");

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

        saved_indent = mdb_dec_indent(0);
        mdb_dec_indent(saved_indent);

        for (i = 0; i < s->m_n_normal; i++)
                if (s->m_slot[i]) {
                        if (!header_output) {
                                mdb_printf("\n");
                                mdb_printf("mptsas_cmd          slot cmd_slot "
                                    "cmd_flags cmd_pkt_flags scsi_pkt      "
                                    "  targ,lun [ pkt_cdbp ...\n");
                                mdb_printf("-------------------------------"
                                    "-------------------------------------"
                                    "-------------------------------------"
                                    "------\n");
                                header_output = 1;
                        }
                        mdb_printf("%16p %4d %s %4d  %8x      %8x %16p ",
                            s->m_slot[i], i,
                            (i == slots[i].cmd_slot?"   ":"BAD"),
                            slots[i].cmd_slot,
                            slots[i].cmd_flags,
                            slots[i].cmd_pkt_flags,
                            slots[i].cmd_pkt);
                        (void) print_cdb(&slots[i]);
                }

        /* print the wait queue */

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



```
                if (q == m.m_waitq)
                        mdb_printf("\n");
                if (mdb_vread(&c, sizeof (mptsas_cmd_t), (uintptr_t)q)
                    == -1) {
                        mdb_warn("couldn't follow m_waitq");
                        rv = DCMD_ERR;
                        goto exit;
                }
                mdb_printf("%16p wait n/a %4d  %8x      %8x %16p ",
                    q, c.cmd_slot, c.cmd_flags, c.cmd_pkt_flags,
                    c.cmd_pkt);
                print_cdb(&c);
        }

        /* print the done queue */

        for (q = mp->m_doneq; q; q = c.cmd_linkp) {
                if (q == mp->m_doneq)
        for (q = m.m_doneq; q; q = c.cmd_linkp) {
                if (q == m.m_doneq)
                        mdb_printf("\n");
                if (mdb_vread(&c, sizeof (mptsas_cmd_t), (uintptr_t)q)
                    == -1) {
                        mdb_warn("couldn't follow m_doneq");
                        rv = DCMD_ERR;
                        goto exit;
                }
                mdb_printf("%16p done  n/a %4d  %8x      %8x %16p ",
                    q, c.cmd_slot, c.cmd_flags, c.cmd_pkt_flags,
                    c.cmd_pkt);
                print_cdb(&c);
        }

        mdb_inc_indent(saved_indent);

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

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

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

        /* now check for corruptions */

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

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

        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     ",
```

```
                                mp->m_protocol_flags[i]);
                                m.m_protocol_flags[i]);
                        break;
                case MPI2_PORTFACTS_PORTTYPE_SCSI:
                        mdb_printf("SCSI (0x%1x)   ",
                                mp->m_protocol_flags[i]);
                                m.m_protocol_flags[i]);
                        break;
                case MPI2_PORTFACTS_PORTTYPE_FC:
                        mdb_printf("FC (0x%1x)     ",
                                mp->m_protocol_flags[i]);
                                m.m_protocol_flags[i]);
                        break;
                case MPI2_PORTFACTS_PORTTYPE_ISCSI:
                        mdb_printf("iSCSI (0x%1x)  ",
                                mp->m_protocol_flags[i]);
                                m.m_protocol_flags[i]);
                        break;
                case MPI2_PORTFACTS_PORTTYPE_SAS:
                        mdb_printf("SAS (0x%1x)    ",
                                mp->m_protocol_flags[i]);
                                m.m_protocol_flags[i]);
                        break;
                default:
                        mdb_printf("unknown      ");
                }
        }
        mdb_printf("\n");
#endif /* ! codereview */
#endif
}

void
dump_debug_log(struct mptsas *mp)
{
        uint_t  idx;
        char    *logbuf;
        int     i;

        if (mdb_readsym(&idx, sizeof (uint_t),
                "mptsas_dbglog_idx") == -1) {
                mdb_warn("No debug log buffer present");
                return;
        }
        logbuf = mdb_alloc(16*256, UM_SLEEP);
        if (idx == 0) {
                mdb_warn("Logging turned off");
                return;
        }

        if (mdb_readsym(logbuf, 16*256,
                "mptsas_dbglog_bufs") == -1) {
                mdb_warn("No debug log buffer present");
                return;
        }
#endif /* ! codereview */
        mdb_printf("\n");
        idx &= 0xf;
        for (i = 0; i < 16; i++) {
                mdb_printf("%s\n", &logbuf[idx*256]);
                idx = (idx+1) & 0xf;
        }
        mdb_free(logbuf, 16*256);
#endif /* ! codereview */
}
```



```
static int
mptsas_dcmd(uintptr_t addr, uint_t flags, int argc, const mdb_arg_t *argv)
{
        struct mptsas           m;
        struct mptsas_slots     *s;

        int                     nslots;
        int                     slot_size = 0;
        uint_t                  verbose = FALSE;
        uint_t                  target_info = FALSE;
        uint_t                  slot_info = FALSE;
        uint_t                  device_info = FALSE;
        uint_t                  port_info = FALSE;
        uint_t                  debug_log = FALSE;
#endif /* ! codereview */
        int                     rv = DCMD_OK;
        void                    *mptsas_state;

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

                mptsas_state = NULL;
                if (mdb_readvar(&mptsas_state, "mptsas_state") == -1) {
                        mdb_warn("can't read mptsas_state");
                        return (DCMD_ERR);
                }
                if (mdb_pwalk_dcmd("genunix`softstate",
                        "mpt_sas`mptsas", argc,
                if (mdb_pwalk_dcmd("genunix`softstate", "mpt_sas`mptsas", argc,
                        argv, (uintptr_t)mptsas_state) == -1) {
                        mdb_warn("mdb_pwalk_dcmd failed");
                        return (DCMD_ERR);
                }
                return (DCMD_OK);
        }

        if (mdb_getopts(argc, argv,
            's', MDB_OPT_SETBITS, TRUE, &slot_info,
            'd', MDB_OPT_SETBITS, TRUE, &device_info,
            't', MDB_OPT_SETBITS, TRUE, &target_info,
            'p', MDB_OPT_SETBITS, TRUE, &port_info,
            'v', MDB_OPT_SETBITS, TRUE, &verbose,
            'D', MDB_OPT_SETBITS, TRUE, &debug_log,
#endif /* ! codereview */
            NULL) != argc)
                return (DCMD_USAGE);


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

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

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

        nslots = s->m_n_normal;

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

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

        s = mdb_alloc(slot_size, UM_SLEEP);

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

        /* processing completed */

        if (((flags & DCMD_ADDRSPEC) && !(flags & DCMD_LOOP)) ||
            (flags & DCMD_LOOPFIRST) || slot_info || device_info ||
            target_info) {
                if ((flags & DCMD_LOOP) && !(flags & DCMD_LOOPFIRST))
                        mdb_printf("\n");
                mdb_printf("        mptsas_t inst ncmds suspend  power");
                mdb_printf("\n");
                mdb_printf("========================================"
                    "=====================================");
                mdb_printf("\n");
        }

        mdb_printf("%16p %4d %5d ", addr, m.m_instance, m.m_ncmds);
        mdb_printf("%7d", m.m_suspended);
        switch (m.m_power_level) {
                case PM_LEVEL_D0:
                        mdb_printf(" ON=D0 ");
                        break;
                case PM_LEVEL_D1:
                        mdb_printf("    D1 ");
                        break;
                case PM_LEVEL_D2:
                        mdb_printf("    D2 ");
                        break;
                case PM_LEVEL_D3:
                        mdb_printf("OFF=D3 ");
                        break;
                default:
                        mdb_printf("INVALD ");
        }
        mdb_printf("\n");

        mdb_inc_indent(17);

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

        if (port_info)
                display_ports(&m);

        if (device_info)
                display_deviceinfo(&m);

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

        if (debug_log)
                dump_debug_log(&m);
                display_slotinfo();
```



```
        mdb_dec_indent(17);

        mdb_free(s, slot_size);

        return (rv);
}

void
mptsas_help()
{
        mdb_printf("Prints summary information about each mpt_sas instance, "
            "including warning\nmessages when slot usage doesn't match "
            "summary information.\n"
            "Without the address of a \"struct mptsas\", prints every "
            "instance.\n\n"
            "Switches:\n"
            "  -t[v]  includes information about targets, v = be more verbose\n"
            "  -t   includes information about targets\n"
            "  -p     includes information about port\n"
            "  -s     includes information about mpt slots\n"
            "  -d     includes information about the hardware\n"
            "  -D     print the mptsas specific debug log\n");
            "  -d   includes information about the hardware\n");
}

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

_____*unchanged_portion_omitted_*

```
**********************************************************
   23880 Thu Jun 12 17:28:21 2014
new/usr/src/cmd/mdb/common/modules/mpt_sas3/mpt_sas3.c
4546 mpt_sas needs enhancing to support LSI MPI2.5
**********************************************************
/*
 * 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);

        if (mdb_vread(&c, sizeof (c), (uintptr_t)pi.pi_client) == -1) {
                mdb_warn("couldn't read mdi_client");
                return (-1);
        }

        *dev_path = NULL;
        if (construct_path((uintptr_t)c.ct_dip, dev_path) != DCMD_OK)
                strcpy(dev_path, "unknown");

        mdb_printf("LUN %d: %s\n", mdi_lun, dev_path);
        mdb_printf("       dip: %p %s path", c.ct_dip,
```

```
            (pi.pi_preferred ? "preferred" : ""));
        switch (pi.pi_state & MDI_PATHINFO_STATE_MASK) {
                case MDI_PATHINFO_STATE_INIT:
                        mdb_printf(" initializing");
                        break;
                case MDI_PATHINFO_STATE_ONLINE:
                        mdb_printf(" online");
                        break;
                case MDI_PATHINFO_STATE_STANDBY:
                        mdb_printf(" standby");
                        break;
                case MDI_PATHINFO_STATE_FAULT:
                        mdb_printf(" fault");
                        break;
                case MDI_PATHINFO_STATE_OFFLINE:
                        mdb_printf(" offline");
                        break;
                default:
                        mdb_printf(" invalid state");
                        break;
        }
        mdb_printf("\n");
        return (0);
}

void
mdi_info(struct mptsas *mp, int target)
{
        struct  dev_info        d;
        struct  mdi_phci        p;

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

        if (MDI_PHCI(&d)) {
                if (mdb_vread(&p, sizeof (p), (uintptr_t)d.devi_mdi_xhci)
                    == -1) {
                        mdb_warn("couldn't read m_dip.devi_mdi_xhci");
                        return;
                }
                if (p.ph_path_head)
                        mdb_pwalk("mdipi_phci_list", (mdb_walk_cb_t)mdi_info_cb,
                            &target, (uintptr_t)p.ph_path_head);
                return;
        }
}

void
print_cdb(mptsas_cmd_t *m)
{
        struct  scsi_pkt        pkt;
        uchar_t cdb[512];       /* an arbitrarily large number */
        int     j;

        if (mdb_vread(&pkt, sizeof (pkt), (uintptr_t)m->cmd_pkt) == -1) {
                mdb_warn("couldn't read cmd_pkt");
                return;
        }

        /*
         * We use cmd_cdblen here because 5.10 doesn't
         * have the cdb length in the pkt
         */
        if (mdb_vread(&cdb, m->cmd_cdblen, (uintptr_t)pkt.pkt_cdbp) == -1) {
```



```
                mdb_warn("couldn't read pkt_cdbp");
                return;
        }

        mdb_printf("%3d,%-3d [ ",
            pkt.pkt_address.a_target, pkt.pkt_address.a_lun);

        for (j = 0; j < m->cmd_cdblen; j++)
                mdb_printf("%02x ", cdb[j]);

        mdb_printf("]\n");
}


void
display_ports(struct mptsas *mp)
{
        int i;
        mdb_printf("\n");
        mdb_printf("phy number and port mapping table\n");
        for (i = 0; i < MPTSAS_MAX_PHYS; i++) {
                if (mp->m_phy_info[i].attached_devhdl) {
                        mdb_printf("phy %x --> port %x, phymask %x,"
                        "attached_devhdl %x\n", i, mp->m_phy_info[i].port_num,
                            mp->m_phy_info[i].phy_mask,
                            mp->m_phy_info[i].attached_devhdl);
                }
        }
        mdb_printf("\n");
}

static uintptr_t
klist_head(list_t *lp, uintptr_t klp)
{
        if ((uintptr_t)lp->list_head.list_next ==
            klp + offsetof(struct list, list_head))
                return (NULL);

        return ((uintptr_t)(((char *)lp->list_head.list_next) -
            lp->list_offset));
}

static uintptr_t
klist_next(list_t *lp, uintptr_t klp, void *op)
{
        /* LINTED E_BAD_PTR_CAST_ALIG */
        struct list_node *np = (struct list_node *)(((char *)op) +
            lp->list_offset);

        if ((uintptr_t)np->list_next == klp + offsetof(struct list, list_head))
                return (NULL);

        return (((uintptr_t)(np->list_next)) - lp->list_offset);
}

static void *
krefhash_first(uintptr_t khp, uintptr_t *addr)
{
        refhash_t mh;
        uintptr_t klp;
        uintptr_t kop;
        void *rp;

        mdb_vread(&mh, sizeof (mh), khp);
        klp = klist_head(&mh.rh_objs, khp + offsetof(refhash_t, rh_objs));
        if (klp == 0)
```

```
                return (NULL);

        kop = klp - mh.rh_link_off;
        if (addr)
                *addr = kop;
        rp = mdb_alloc(mh.rh_obj_size, UM_SLEEP);
        mdb_vread(rp, mh.rh_obj_size, kop);

        return (rp);
}

static void *
krefhash_next(uintptr_t khp, void *op, uintptr_t *addr)
{
        refhash_t mh;
        void *prev = op;
        refhash_link_t *lp;
        uintptr_t klp;
        uintptr_t kop;
        refhash_link_t ml;
        void *rp;

        mdb_vread(&mh, sizeof (mh), khp);
        /* LINTED E_BAD_PTR_CAST_ALIG */
        lp = (refhash_link_t *)(((char *)(op)) + mh.rh_link_off);
        ml = *lp;
        while ((klp = klist_next(&mh.rh_objs,
            khp + offsetof(refhash_t, rh_objs), &ml)) != NULL) {
                mdb_vread(&ml, sizeof (ml), klp);
                if (!(ml.rhl_flags & RHL_F_DEAD))
                        break;
        }

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

        kop = klp - mh.rh_link_off;
        if (addr)
                *addr = kop;
        rp = mdb_alloc(mh.rh_obj_size, UM_SLEEP);
        mdb_vread(rp, mh.rh_obj_size, kop);

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

void
display_targets(struct mptsas *mp, uint_t verbose)
{
        mptsas_target_t *ptgt;
        mptsas_smp_t *psmp;
        int loop, comma;
        uintptr_t p_addr;

        mdb_printf("\n");
        mdb_printf(" mptsas_target_t slot devhdl      wwn     ncmds throttle   "
            "dr_flag dups\n");
        mdb_printf("--------------------------------------"
            "------------------------------\n");
        for (ptgt = (mptsas_target_t *)krefhash_first(
                (uintptr_t)mp->m_targets, &p_addr);
            ptgt != NULL;
            ptgt = krefhash_next((uintptr_t)mp->m_targets, ptgt, &p_addr)) {
                if (ptgt->m_addr.mta_wwn ||
```



```
                        ptgt->m_deviceinfo) {
                            mdb_printf("%16p ", p_addr);
                            mdb_printf("%4d ", ptgt->m_slot_num);
                            mdb_printf("%4d ", ptgt->m_devhdl);
                            if (ptgt->m_addr.mta_wwn)
                                    mdb_printf("%"PRIx64" ",
                                        ptgt->m_addr.mta_wwn);
                            mdb_printf("%3d", ptgt->m_t_ncmds);
                            switch (ptgt->m_t_throttle) {
                                    case QFULL_THROTTLE:
                                            mdb_printf("   QFULL ");
                                            break;
                                    case DRAIN_THROTTLE:
                                            mdb_printf("   DRAIN ");
                                            break;
                                    case HOLD_THROTTLE:
                                            mdb_printf("    HOLD ");
                                            break;
                                    case MAX_THROTTLE:
                                            mdb_printf("     MAX ");
                                            break;
                                    default:
                                            mdb_printf("%8d ",
                                                ptgt->m_t_throttle);
                            }
                            switch (ptgt->m_dr_flag) {
                                    case MPTSAS_DR_INACTIVE:
                                            mdb_printf("  INACTIVE ");
                                            break;
                                    case MPTSAS_DR_INTRANSITION:
                                            mdb_printf("TRANSITION ");
                                            break;
                                    default:
                                            mdb_printf("   UNKNOWN ");
                                            break;
                                    }
                            mdb_printf("%d\n",
                                ptgt->m_dups);

                            if (verbose) {
                                    mdb_inc_indent(5);
                                    if ((ptgt->m_deviceinfo &
                                        MPI2_SAS_DEVICE_INFO_MASK_DEVICE_TYPE) ==
                                        MPI2_SAS_DEVICE_INFO_FANOUT_EXPANDER)
                                            mdb_printf("Fanout expander: ");
                                    if ((ptgt->m_deviceinfo &
                                        MPI2_SAS_DEVICE_INFO_MASK_DEVICE_TYPE) ==
                                        MPI2_SAS_DEVICE_INFO_EDGE_EXPANDER)
                                            mdb_printf("Edge expander: ");
                                    if ((ptgt->m_deviceinfo &
                                        MPI2_SAS_DEVICE_INFO_MASK_DEVICE_TYPE) ==
                                        MPI2_SAS_DEVICE_INFO_END_DEVICE)
                                            mdb_printf("End device: ");
                                    if ((ptgt->m_deviceinfo &
                                        MPI2_SAS_DEVICE_INFO_MASK_DEVICE_TYPE) ==
                                        MPI2_SAS_DEVICE_INFO_NO_DEVICE)
                                            mdb_printf("No device ");

                                    for (loop = 0, comma = 0;
                                        loop < (sizeof (devinfo_array) /
                                        sizeof (devinfo_array[0])); loop++) {
                                            if (ptgt->m_deviceinfo &
                                                devinfo_array[loop].value) {
                                                    mdb_printf("%s%s",
                                                        (comma ? ", " : ""),
                                                        devinfo_array[loop].text);
```

```
                                                comma++;
                                        }
                                }
                                mdb_printf("\n");
                                mdi_info(mp, ptgt->m_slot_num);
                                mdb_dec_indent(5);
                        }
                }
        }

        mdb_printf("\n");
        mdb_printf("    mptsas_smp_t devhdl      wwn          phymask\n");
        mdb_printf("-------------------------------------"
            "-----------------\n");
        for (psmp = (mptsas_smp_t *)krefhash_first(
            (uintptr_t)mp->m_smp_targets, &p_addr);
            psmp != NULL;
            psmp = krefhash_next((uintptr_t)mp->m_smp_targets, psmp,
            &p_addr)) {
                mdb_printf("%16p   ", p_addr);
                mdb_printf("%4d  %"PRIx64"    %04x\n",
                    psmp->m_devhdl, psmp->m_addr.mta_wwn,
                    psmp->m_addr.mta_phymask);
                if (verbose) {
                        mdb_inc_indent(5);
                        if ((psmp->m_deviceinfo &
                                MPI2_SAS_DEVICE_INFO_MASK_DEVICE_TYPE) ==
                            MPI2_SAS_DEVICE_INFO_FANOUT_EXPANDER)
                                mdb_printf("Fanout expander: ");
                        if ((psmp->m_deviceinfo &
                                MPI2_SAS_DEVICE_INFO_MASK_DEVICE_TYPE) ==
                            MPI2_SAS_DEVICE_INFO_EDGE_EXPANDER)
                                mdb_printf("Edge expander: ");
                        if ((psmp->m_deviceinfo &
                                MPI2_SAS_DEVICE_INFO_MASK_DEVICE_TYPE) ==
                            MPI2_SAS_DEVICE_INFO_END_DEVICE)
                                mdb_printf("End device: ");
                        if ((psmp->m_deviceinfo &
                                MPI2_SAS_DEVICE_INFO_MASK_DEVICE_TYPE) ==
                            MPI2_SAS_DEVICE_INFO_NO_DEVICE)
                                mdb_printf("No device ");

                        for (loop = 0, comma = 0;
                             loop < (sizeof (devinfo_array) /
                                 sizeof (devinfo_array[0])); loop++) {
                                if (psmp->m_deviceinfo &
                                    devinfo_array[loop].value) {
                                        mdb_printf("%s%s",
                                            (comma ? ", " : ""),
                                            devinfo_array[loop].text);
                                        comma++;
                                }
                        }
                        mdb_printf("\n");
                        mdb_dec_indent(5);
                }
        }
}

int
display_slotinfo(struct mptsas *mp, struct mptsas_slots *s)
{
        int                     i, nslots;
        struct mptsas_cmd       c, *q, *slots;
        mptsas_target_t         *ptgt;
        int                     header_output = 0;
```



```
        int                     rv = DCMD_OK;
        int                     slots_in_use = 0;
        int                     tcmds = 0;
        int                     mismatch = 0;
        int                     wq, dq;
        int                     ncmds = 0;
        ulong_t                 saved_indent;

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

        for (i = 0; i < nslots; i++)
                if (s->m_slot[i]) {
                        slots_in_use++;
                        if (mdb_vread(&slots[i], sizeof (mptsas_cmd_t),
                            (uintptr_t)s->m_slot[i]) == -1) {
                                mdb_warn("couldn't read slot");
                                s->m_slot[i] = NULL;
                        }
                        if ((slots[i].cmd_flags & CFLAG_CMDIOC) == 0)
                                tcmds++;
                        if (i != slots[i].cmd_slot)
                                mismatch++;
                }

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

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

        for (ptgt = (mptsas_target_t *)krefhash_first(
                (uintptr_t)mp->m_targets, NULL);
            ptgt != NULL;
             ptgt = krefhash_next((uintptr_t)mp->m_targets, ptgt, NULL)) {
                if (ptgt->m_addr.mta_wwn ||
                    ptgt->m_deviceinfo) {
                        ncmds += ptgt->m_t_ncmds;
                }
        }

        mdb_printf("\n");
        mdb_printf("   mpt.  slot               mptsas_slots     slot");
        mdb_printf("\n");
        mdb_printf("m_ncmds total"
            " targ throttle m_t_ncmds targ_tot wq dq");
        mdb_printf("\n");
        mdb_printf("-----------------------------------------------------");
        mdb_printf("\n");

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

        saved_indent = mdb_dec_indent(0);
        mdb_dec_indent(saved_indent);
```

```
        for (i = 0; i < s->m_n_normal; i++)
                if (s->m_slot[i]) {
                        if (!header_output) {
                                mdb_printf("\n");
                                mdb_printf("mptsas_cmd          slot cmd_slot "
                                    "cmd_flags cmd_pkt_flags scsi_pkt      "
                                    "  targ,lun [ pkt_cdbp ...\n");
                                mdb_printf("-------------------------------"
                                    "-------------------------------------"
                                    "-------------------------------------"
                                    "------\n");
                                header_output = 1;
                        }
                        mdb_printf("%16p %4d %s %4d  %8x      %8x %16p ",
                            s->m_slot[i], i,
                            (i == slots[i].cmd_slot?"   ":"BAD"),
                            slots[i].cmd_slot,
                            slots[i].cmd_flags,
                            slots[i].cmd_pkt_flags,
                            slots[i].cmd_pkt);
                        (void) print_cdb(&slots[i]);
                }

        /* print the wait queue */

        for (q = mp->m_waitq; q; q = c.cmd_linkp) {
                if (q == mp->m_waitq)
                        mdb_printf("\n");
                if (mdb_vread(&c, sizeof (mptsas_cmd_t), (uintptr_t)q)
                    == -1) {
                        mdb_warn("couldn't follow m_waitq");
                        rv = DCMD_ERR;
                        goto exit;
                }
                mdb_printf("%16p wait n/a %4d  %8x      %8x %16p ",
                    q, c.cmd_slot, c.cmd_flags, c.cmd_pkt_flags,
                    c.cmd_pkt);
                print_cdb(&c);
        }

        /* print the done queue */

        for (q = mp->m_dlist.dl_q; q; q = c.cmd_linkp) {
                if (q == mp->m_dlist.dl_q)
                        mdb_printf("\n");
                if (mdb_vread(&c, sizeof (mptsas_cmd_t), (uintptr_t)q)
                    == -1) {
                        mdb_warn("couldn't follow m_doneq");
                        rv = DCMD_ERR;
                        goto exit;
                }
                mdb_printf("%16p done  n/a %4d  %8x      %8x %16p ",
                    q, c.cmd_slot, c.cmd_flags, c.cmd_pkt_flags,
                    c.cmd_pkt);
                print_cdb(&c);
        }

        mdb_inc_indent(saved_indent);

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

        if (tcmds != ncmds)
                mdb_printf("WARNING: the total of m_target[].m_t_ncmds does "
```



```
                    "not match the slots in use\n");

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

        /* now check for corruptions */

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

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

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

                if (mdb_vread(&c, sizeof (mptsas_cmd_t), (uintptr_t)q) == -1) {
                        mdb_warn("couldn't follow m_doneq");
                        rv = DCMD_ERR;
                        goto exit;
                }
                if ((c.cmd_flags & CFLAG_FINISHED) == 0)
                        mdb_printf("WARNING: m_doneq entry (mptsas_cmd_t) %p "
                            "should have CFLAG_FINISHED set\n", q);
                if (c.cmd_flags & CFLAG_IN_TRANSPORT)
                        mdb_printf("WARNING: m_doneq entry (mptsas_cmd_t) %p "
                            "should not have CFLAG_IN_TRANSPORT set\n", q);
                if (c.cmd_flags & CFLAG_CMDARQ)
                        mdb_printf("WARNING: m_doneq entry (mptsas_cmd_t) %p "
                            "should not have CFLAG_CMDARQ set\n", q);
                if (c.cmd_flags & CFLAG_COMPLETED)
                        mdb_printf("WARNING: m_doneq entry (mptsas_cmd_t) %p "
                            "should not have CFLAG_COMPLETED set\n", q);
        }

exit:
        mdb_free(slots, sizeof (mptsas_cmd_t) * nslots);
        return (rv);
}

void
display_deviceinfo(struct mptsas *mp)
{
        char    device_path[PATH_MAX];

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

        mdb_printf("\n");
        mdb_printf("base_wwid          phys "
            " prodid  devid          revid   ssid\n");
        mdb_printf("----------------------------"
```

```
            "---------------------------------\n");
        mdb_printf("%"PRIx64"     %2d  "
            "0x%04x 0x%04x ", mp->un.m_base_wwid, mp->m_num_phys,
            mp->m_productid, mp->m_devid);
        switch (mp->m_devid) {
                case MPI2_MFGPAGE_DEVID_SAS2004:
                        mdb_printf("(SAS2004) ");
                        break;
                case MPI2_MFGPAGE_DEVID_SAS2008:
                        mdb_printf("(SAS2008) ");
                        break;
                case MPI2_MFGPAGE_DEVID_SAS2108_1:
                case MPI2_MFGPAGE_DEVID_SAS2108_2:
                case MPI2_MFGPAGE_DEVID_SAS2108_3:
                        mdb_printf("(SAS2108) ");
                        break;
                case MPI2_MFGPAGE_DEVID_SAS2116_1:
                case MPI2_MFGPAGE_DEVID_SAS2116_2:
                        mdb_printf("(SAS2116) ");
                        break;
                case MPI2_MFGPAGE_DEVID_SSS6200:
                        mdb_printf("(SSS6200) ");
                        break;
                case MPI2_MFGPAGE_DEVID_SAS2208_1:
                case MPI2_MFGPAGE_DEVID_SAS2208_2:
                case MPI2_MFGPAGE_DEVID_SAS2208_3:
                case MPI2_MFGPAGE_DEVID_SAS2208_4:
                case MPI2_MFGPAGE_DEVID_SAS2208_5:
                case MPI2_MFGPAGE_DEVID_SAS2208_6:
#if 0
                /* Same as 2308_1/2 ?? */
                case MPI2_MFGPAGE_DEVID_SAS2208_7:
                case MPI2_MFGPAGE_DEVID_SAS2208_8:
#endif
                        mdb_printf("(SAS2208) ");
                        break;
                case MPI2_MFGPAGE_DEVID_SAS2308_1:
                case MPI2_MFGPAGE_DEVID_SAS2308_2:
                case MPI2_MFGPAGE_DEVID_SAS2308_3:
                        mdb_printf("(SAS2308) ");
                        break;
                case MPI25_MFGPAGE_DEVID_SAS3004:
                        mdb_printf("(SAS3004) ");
                        break;
                case MPI25_MFGPAGE_DEVID_SAS3008:
                        mdb_printf("(SAS3008) ");
                        break;
                case MPI25_MFGPAGE_DEVID_SAS3108_1:
                case MPI25_MFGPAGE_DEVID_SAS3108_2:
                case MPI25_MFGPAGE_DEVID_SAS3108_3:
                case MPI25_MFGPAGE_DEVID_SAS3108_4:
                case MPI25_MFGPAGE_DEVID_SAS3108_5:
                case MPI25_MFGPAGE_DEVID_SAS3108_6:
                        mdb_printf("(SAS3108) ");
                        break;
                default:
                        mdb_printf("(SAS????) ");
                        break;
        }
        mdb_printf("0x%02x 0x%04x\n", mp->m_revid, mp->m_ssid);
        mdb_printf("%s\n", device_path);
}

void
dump_debug_log(struct mptsas *mp)
{
```



```
        uint_t  idx;
        char    *logbuf;
        int     i;

        if (mdb_readsym(&idx, sizeof (uint_t),
                "mptsas_dbglog_idx") == -1) {
                mdb_warn("No debug log buffer present");
                return;
        }
        logbuf = mdb_alloc(32*256, UM_SLEEP);
        if (idx == 0) {
                mdb_warn("Logging turned off");
                return;
        }

        if (mdb_readsym(logbuf, 32*256,
                "mptsas_dbglog_bufs") == -1) {
                mdb_warn("No debug log buffer present");
                return;
        }
        mdb_printf("\n");
        idx &= 0x1f;
        for (i = 0; i < 32; i++) {
                mdb_printf("%s\n", &logbuf[idx*256]);
                idx = (idx+1) & 0x1f;
        }
        mdb_free(logbuf, 32*256);
}

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

        int                     nslots;
        int                     slot_size = 0;
        uint_t                  verbose = FALSE;
        uint_t                  target_info = FALSE;
        uint_t                  slot_info = FALSE;
        uint_t                  device_info = FALSE;
        uint_t                  port_info = FALSE;
        uint_t                  debug_log = FALSE;
        int                     rv = DCMD_OK;

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

                if (mdb_readvar(&mptsas3_state, "mptsas3_state") == -1) {
                        mdb_warn("can't read mptsas3_state");
                        return (DCMD_ERR);
                }
                if (mdb_pwalk_dcmd("genunix`softstate",
                        "mpt_sas3`mptsas3", argc,
                        argv, (uintptr_t)mptsas3_state) == -1) {
                        mdb_warn("mdb_pwalk_dcmd failed");
                        return (DCMD_ERR);
                }
                return (DCMD_OK);
        }

        if (mdb_getopts(argc, argv,
            's', MDB_OPT_SETBITS, TRUE, &slot_info,
            'd', MDB_OPT_SETBITS, TRUE, &device_info,
            't', MDB_OPT_SETBITS, TRUE, &target_info,
            'p', MDB_OPT_SETBITS, TRUE, &port_info,
```

```
            'v', MDB_OPT_SETBITS, TRUE, &verbose,
            'D', MDB_OPT_SETBITS, TRUE, &debug_log,
            NULL) != argc)
                return (DCMD_USAGE);


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

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

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

        nslots = s->m_n_normal;

        mdb_free(s, sizeof (mptsas_slots_t));

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

        s = mdb_alloc(slot_size, UM_SLEEP);

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

        /* processing completed */

        if (((flags & DCMD_ADDRSPEC) && !(flags & DCMD_LOOP)) ||
            (flags & DCMD_LOOPFIRST) || slot_info || device_info ||
            target_info) {
                if ((flags & DCMD_LOOP) && !(flags & DCMD_LOOPFIRST))
                        mdb_printf("\n");
                mdb_printf("         mptsas_t inst ncmds suspend  power");
                mdb_printf("\n");
                mdb_printf("========================================"
                    "=====================================");
                mdb_printf("\n");
        }

        mdb_printf("%16p %4d %5d ", addr, m.m_instance, m.m_ncmds);
        mdb_printf("%7d", m.m_suspended);
        switch (m.m_power_level) {
                case PM_LEVEL_D0:
                        mdb_printf(" ON=D0 ");
                        break;
                case PM_LEVEL_D1:
                        mdb_printf("    D1 ");
                        break;
                case PM_LEVEL_D2:
                        mdb_printf("    D2 ");
                        break;
                case PM_LEVEL_D3:
                        mdb_printf("OFF=D3 ");
                        break;
```



```
                default:
                        mdb_printf("INVALD ");
        }
        mdb_printf("\n");

        mdb_inc_indent(17);

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

        if (port_info)
                display_ports(&m);

        if (device_info)
                display_deviceinfo(&m);

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

        if (debug_log)
                dump_debug_log(&m);

        mdb_dec_indent(17);

        mdb_free(s, slot_size);

        return (rv);
}

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

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

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

const mdb_modinfo_t *
_mdb_init(void)
{
        return (&modinfo);
}
#endif /* ! codereview */
```

```
**********************************************************
    1154 Thu Jun 12 17:28:21 2014
new/usr/src/cmd/mdb/intel/amd64/mpt_sas3/Makefile
4546 mpt_sas needs enhancing to support LSI MPI2.5
**********************************************************
#
# 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 17:28:21 2014
new/usr/src/cmd/mdb/intel/ia32/mpt_sas3/Makefile
4546 mpt_sas needs enhancing to support LSI MPI2.5
**********************************************************
#
# 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 17:28:21 2014
new/usr/src/cmd/mdb/sparc/v9/mpt_sas3/Makefile
4546 mpt_sas needs enhancing to support LSI MPI2.5
**********************************************************
#
# 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 17:28:21 2014
new/usr/src/man/man7d/Makefile
4546 mpt_sas needs enhancing to support LSI MPI2.5
**********************************************************
#
# 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          \
```

```
                ttymux.7d       \
                tzmon.7d        \
                ugen.7d         \
                uhci.7d         \
                usb_ac.7d       \
                usb_as.7d       \
                usb_ia.7d       \
                usb_mid.7d      \
                usba.7d         \
                usbftdi.7d      \
                usbprn.7d       \
                usbsacm.7d      \
                usbsksp.7d      \
                usbsprl.7d      \
                usbvc.7d        \
                uwba.7d         \
                virtualkm.7d    \
                vni.7d          \
                vr.7d           \
                wscons.7d       \
                wusb_ca.7d      \
                wusb_df.7d      \
                xge.7d          \
                yge.7d          \
                zcons.7d        \
                zero.7d

sparc_MANFILES= audiocs.7d      \
                bbc_beep.7d     \
                ctsmc.7d        \
                cvc.7d          \
                cvcredir.7d     \
                dad.7d          \
                dm2s.7d         \
                dr.7d           \
                eri.7d          \
                fas.7d          \
                gpio_87317.7d   \
                grbeep.7d       \
                idn.7d          \
                mc-opl.7d       \
                n2rng.7d        \
                ncp.7d          \
                ntwdt.7d        \
                oplkmdrv.7d     \
                oplmsu.7d       \
                oplpanel.7d     \
                pcicmu.7d       \
                pcipsy.7d       \
                pcisch.7d       \
                schpc.7d        \
                sf.7d           \
                smbus.7d        \
                socal.7d        \
                ssd.7d          \
                su.7d           \
                todopl.7d       \
                tsalarm.7d      \
                zs.7d           \
                zsh.7d

i386_MANFILES=  ahci.7d         \
                amd8111s.7d     \
                amr.7d          \
                arcmsr.7d       \
                arn.7d          \
```



```
                asy.7d          \
                ata.7d          \
                atge.7d         \
                ath.7d          \
                atu.7d          \
                audio810.7d     \
                audiocmi.7d     \
                audiocmihd.7d   \
                audioemu10k.7d  \
                audiohd.7d      \
                audioixp.7d     \
                audiosolo.7d    \
                audiovia823x.7d \
                audiovia97.7d   \
                bcm_sata.7d     \
                bfe.7d          \
                cmdk.7d         \
                cpqary3.7d      \
                dnet.7d         \
                ecpp.7d         \
                heci.7d         \
                i915.7d         \
                ipmi.7d         \
                ipw.7d          \
                iwh.7d          \
                iwi.7d          \
                mega_sas.7d     \
                npe.7d          \
                ntxn.7d         \
                nv_sata.7d      \
                pcn.7d          \
                radeon.7d       \
                ral.7d          \
                rtw.7d          \
                rum.7d          \
                rwd.7d          \
                rwn.7d          \
                sda.7d          \
                sdcard.7d       \
                sdhost.7d       \
                si3124.7d       \
                smbios.7d       \
                uath.7d         \
                ural.7d         \
                urtw.7d         \
                wpi.7d          \
                zyd.7d

_MANLINKS=      1394.7d         \
                allkmem.7d      \
                bscbus.7d       \
                fdc.7d          \
                firewire.7d     \
                hwa1480_fw.7d   \
                i2bsc.7d        \
                kmem.7d         \
                lo0.7d          \
                ticots.7d       \
                ticotsord.7d    \
                urandom.7d      \
                usb.7d          \
                uwb.7d

sparc_MANLINKS= drmach.7d       \
                ngdr.7d         \
                ngdrmach.7d
```

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

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

drmach.7d       := LINKSRC = dr.7d
ngdr.7d         := LINKSRC = dr.7d
ngdrmach.7d     := LINKSRC = dr.7d

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

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

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

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

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

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

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

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

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

.KEEP_STATE:

include         $(SRC)/man/Makefile.man

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

```
**********************************************************
    3679 Thu Jun 12 17:28:21 2014
new/usr/src/man/man7d/mpt_sas3.7d
4546 mpt_sas needs enhancing to support LSI MPI2.5
**********************************************************
'\" 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 .
```

```
ATTRIBUTE TYPE  ATTRIBUTE VALUE
_
Architecture    SPARC, x86
.TE

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

```
**********************************************************
   21467 Thu Jun 12 17:28:21 2014
new/usr/src/pkg/manifests/developer-debug-mdb.mf
4546 mpt_sas needs enhancing to support LSI MPI2.5
**********************************************************
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#

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

set name=pkg.fmri value=pkg:/developer/debug/mdb@$(PKGVERS)
set name=pkg.description value="Modular Debugger (MDB)"
set name=pkg.summary value="Modular Debugger"
set name=info.classification \
    value=org.opensolaris.category.2008:Development/System
set name=variant.arch value=$(ARCH)
dir path=kernel group=sys
dir path=kernel/kmdb group=sys
dir path=kernel/kmdb/$(ARCH64) group=sys
dir path=platform group=sys variant.opensolaris.zone=global
$(i386_ONLY)dir path=platform/i86pc group=sys variant.opensolaris.zone=global
$(i386_ONLY)dir path=platform/i86pc/kernel group=sys \
    variant.opensolaris.zone=global
$(i386_ONLY)dir path=platform/i86pc/kernel/kmdb group=sys
$(i386_ONLY)dir path=platform/i86pc/kernel/kmdb/$(ARCH64) group=sys
$(i386_ONLY)dir path=platform/i86xpv group=sys variant.opensolaris.zone=global
$(i386_ONLY)dir path=platform/i86xpv/kernel group=sys \
    variant.opensolaris.zone=global
$(i386_ONLY)dir path=platform/i86xpv/kernel/kmdb group=sys
$(i386_ONLY)dir path=platform/i86xpv/kernel/kmdb/$(ARCH64) group=sys
$(sparc_ONLY)dir path=platform/sun4u group=sys variant.opensolaris.zone=global
$(sparc_ONLY)dir path=platform/sun4u/kernel group=sys \
    variant.opensolaris.zone=global
$(sparc_ONLY)dir path=platform/sun4u/kernel/kmdb group=sys
$(sparc_ONLY)dir path=platform/sun4u/kernel/kmdb/$(ARCH64) group=sys
$(sparc_ONLY)dir path=platform/sun4v group=sys variant.opensolaris.zone=global
$(sparc_ONLY)dir path=platform/sun4v/kernel group=sys \
    variant.opensolaris.zone=global
$(sparc_ONLY)dir path=platform/sun4v/kernel/kmdb group=sys
$(sparc_ONLY)dir path=platform/sun4v/kernel/kmdb/$(ARCH64) group=sys
dir path=usr group=sys
dir path=usr/bin
dir path=usr/bin/$(ARCH32)
dir path=usr/bin/$(ARCH64)
dir path=usr/include
dir path=usr/include/sys
```



```
dir path=usr/lib
dir path=usr/lib/mdb group=sys
dir path=usr/lib/mdb/kvm group=sys
dir path=usr/lib/mdb/kvm/$(ARCH64) group=sys
dir path=usr/lib/mdb/proc group=sys
$(sparc_ONLY)dir path=usr/lib/mdb/proc/$(ARCH64) group=sys
$(i386_ONLY)dir path=usr/lib/mdb/proc/$(ARCH64)
dir path=usr/lib/mdb/raw group=sys
dir path=usr/platform group=sys
$(i386_ONLY)dir path=usr/platform/i86pc group=sys
$(i386_ONLY)dir path=usr/platform/i86pc/lib
$(i386_ONLY)dir path=usr/platform/i86pc/lib/mdb group=sys
$(i386_ONLY)dir path=usr/platform/i86pc/lib/mdb/kvm group=sys
$(i386_ONLY)dir path=usr/platform/i86pc/lib/mdb/kvm/$(ARCH64) group=sys
$(i386_ONLY)dir path=usr/platform/i86xpv group=sys
$(i386_ONLY)dir path=usr/platform/i86xpv/lib
$(i386_ONLY)dir path=usr/platform/i86xpv/lib/mdb group=sys
$(i386_ONLY)dir path=usr/platform/i86xpv/lib/mdb/kvm group=sys
$(i386_ONLY)dir path=usr/platform/i86xpv/lib/mdb/kvm/$(ARCH64) group=sys
$(sparc_ONLY)dir path=usr/platform/sun4u group=sys
$(sparc_ONLY)dir path=usr/platform/sun4u/lib
$(sparc_ONLY)dir path=usr/platform/sun4u/lib/mdb group=sys
$(sparc_ONLY)dir path=usr/platform/sun4u/lib/mdb/kvm group=sys
$(sparc_ONLY)dir path=usr/platform/sun4u/lib/mdb/kvm/$(ARCH64) group=sys
$(sparc_ONLY)dir path=usr/platform/sun4v group=sys
$(sparc_ONLY)dir path=usr/platform/sun4v/lib
$(sparc_ONLY)dir path=usr/platform/sun4v/lib/mdb group=sys
$(sparc_ONLY)dir path=usr/platform/sun4v/lib/mdb/kvm group=sys
$(sparc_ONLY)dir path=usr/platform/sun4v/lib/mdb/kvm/$(ARCH64) group=sys
dir path=usr/share/man
dir path=usr/share/man/man1
file path=kernel/kmdb/$(ARCH64)/arp group=sys mode=0555
file path=kernel/kmdb/$(ARCH64)/cpc group=sys mode=0555
$(i386_ONLY)file path=kernel/kmdb/$(ARCH64)/cpu.generic group=sys mode=0555
$(i386_ONLY)file path=kernel/kmdb/$(ARCH64)/cpu_ms.AuthenticAMD.15 group=sys \
    mode=0555
file path=kernel/kmdb/$(ARCH64)/crypto group=sys mode=0555
file path=kernel/kmdb/$(ARCH64)/genunix group=sys mode=0555
file path=kernel/kmdb/$(ARCH64)/hook group=sys mode=0555
$(sparc_ONLY)file path=kernel/kmdb/$(ARCH64)/intr group=sys mode=0555
file path=kernel/kmdb/$(ARCH64)/ip group=sys mode=0555
file path=kernel/kmdb/$(ARCH64)/ipc group=sys mode=0555
file path=kernel/kmdb/$(ARCH64)/ipp group=sys mode=0555
file path=kernel/kmdb/$(ARCH64)/krtld group=sys mode=0555
file path=kernel/kmdb/$(ARCH64)/lofs group=sys mode=0555
file path=kernel/kmdb/$(ARCH64)/logindmux group=sys mode=0555
file path=kernel/kmdb/$(ARCH64)/mac group=sys mode=0555
file path=kernel/kmdb/$(ARCH64)/md group=sys mode=0555
file path=kernel/kmdb/$(ARCH64)/mdb_ds group=sys mode=0555
file path=kernel/kmdb/$(ARCH64)/mpt group=sys mode=0555
file path=kernel/kmdb/$(ARCH64)/mpt_sas group=sys mode=0555
file path=kernel/kmdb/$(ARCH64)/mpt_sas3 group=sys mode=0555
#endif /* ! codereview */
file path=kernel/kmdb/$(ARCH64)/mr_sas group=sys mode=0555
file path=kernel/kmdb/$(ARCH64)/nca group=sys mode=0555
file path=kernel/kmdb/$(ARCH64)/neti group=sys mode=0555
file path=kernel/kmdb/$(ARCH64)/nfs group=sys mode=0555
file path=kernel/kmdb/$(ARCH64)/ptm group=sys mode=0555
file path=kernel/kmdb/$(ARCH64)/random group=sys mode=0555
file path=kernel/kmdb/$(ARCH64)/s1394 group=sys mode=0555
$(i386_ONLY)file path=kernel/kmdb/$(ARCH64)/sata group=sys mode=0555
file path=kernel/kmdb/$(ARCH64)/scsi_vhci group=sys mode=0555
file path=kernel/kmdb/$(ARCH64)/sctp group=sys mode=0555
file path=kernel/kmdb/$(ARCH64)/sd group=sys mode=0555
file path=kernel/kmdb/$(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
$(sparc_ONLY)link path=kernel/kmdb/$(ARCH64)/px target=intr
```

```
$(sparc_ONLY)link path=usr/lib/mdb/kvm/$(ARCH64)/niumx.so target=intr.so
$(sparc_ONLY)link path=usr/lib/mdb/kvm/$(ARCH64)/pcipsy.so target=intr.so
$(sparc_ONLY)link path=usr/lib/mdb/kvm/$(ARCH64)/pcisch.so target=intr.so
$(sparc_ONLY)link path=usr/lib/mdb/kvm/$(ARCH64)/px.so target=intr.so
```

```
**********************************************************
    1924 Thu Jun 12 17:28:21 2014
new/usr/src/pkg/manifests/driver-storage-mpt_sas3.mf
4546 mpt_sas needs enhancing to support LSI MPI2.5
**********************************************************
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#

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

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

```
**********************************************************
   44270 Thu Jun 12 17:28:22 2014
new/usr/src/uts/common/Makefile.files
4546 mpt_sas needs enhancing to support LSI MPI2.5
**********************************************************
#
# 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         \
                compress.o      \
```

```
                condvar.o       \
                conf.o          \
                console.o       \
                contract.o      \
                copyops.o       \
                core.o          \
                corectl.o       \
                cred.o          \
                cs_stubs.o      \
                dacf.o          \
                dacf_clnt.o     \
                damap.o \
                cyclic.o        \
                ddi.o           \
                ddifm.o         \
                ddi_hp_impl.o   \
                ddi_hp_ndi.o    \
                ddi_intr.o      \
                ddi_intr_impl.o \
                ddi_intr_irm.o  \
                ddi_nodeid.o    \
                ddi_periodic.o  \
                devcfg.o        \
                devcache.o      \
                device.o        \
                devid.o         \
                devid_cache.o   \
                devid_scsi.o    \
                devid_smp.o     \
                devpolicy.o     \
                disp_lock.o     \
                dnlc.o          \
                driver.o        \
                dumpsubr.o      \
                driver_lyr.o    \
                dtrace_subr.o   \
                errorq.o        \
                etheraddr.o     \
                evchannels.o    \
                exacct.o        \
                exacct_core.o   \
                exec.o          \
                exit.o          \
                fbio.o          \
                fcntl.o         \
                fdbuffer.o      \
                fdsync.o        \
                fem.o           \
                ffs.o           \
                fio.o           \
                flock.o         \
                fm.o            \
                fork.o          \
                vpm.o           \
                fs_reparse.o    \
                fs_subr.o       \
                fsflush.o       \
                ftrace.o        \
                getcwd.o        \
                getdents.o      \
                getloadavg.o    \
                getpagesizes.o  \
                getpid.o        \
                gfs.o           \
                rusagesys.o     \
                gid.o           \
```



```
                groups.o        \
                grow.o          \
                hat_refmod.o    \
                id32.o          \
                id_space.o      \
                inet_ntop.o     \
                instance.o      \
                ioctl.o         \
                ip_cksum.o      \
                issetugid.o     \
                ippconf.o       \
                kcpc.o          \
                kdi.o           \
                kiconv.o        \
                klpd.o          \
                kmem.o          \
                ksyms_snapshot.o        \
                l_strplumb.o    \
                labelsys.o      \
                link.o          \
                list.o          \
                lockstat_subr.o \
                log_sysevent.o  \
                logsubr.o       \
                lookup.o        \
                lseek.o         \
                ltos.o          \
                lwp.o           \
                lwp_create.o    \
                lwp_info.o      \
                lwp_self.o      \
                lwp_sobj.o      \
                lwp_timer.o     \
                lwpsys.o        \
                main.o          \
                mmapobjsys.o    \
                memcntl.o       \
                memstr.o        \
                lgrpsys.o       \
                mkdir.o         \
                mknod.o         \
                mount.o         \
                move.o          \
                msacct.o        \
                multidata.o     \
                nbmlock.o       \
                ndifm.o         \
                nice.o          \
                netstack.o      \
                ntptime.o       \
                nvpair.o        \
                nvpair_alloc_system.o   \
                nvpair_alloc_fixed.o    \
                fnvpair.o       \
                octet.o         \
                open.o          \
                p_online.o      \
                pathconf.o      \
                pathname.o      \
                pause.o         \
                serializer.o    \
                pci_intr_lib.o  \
                pci_cap.o       \
                pcifm.o         \
                pgrp.o          \
                pgrpsys.o       \
```

```
                pid.o           \
                pkp_hash.o      \
                policy.o        \
                poll.o          \
                pool.o          \
                pool_pset.o     \
                port_subr.o     \
                ppriv.o         \
                printf.o        \
                priocntl.o      \
                priv.o          \
                priv_const.o    \
                proc.o          \
                procset.o       \
                processor_bind.o        \
                processor_info.o        \
                profil.o        \
                project.o       \
                qsort.o         \
                rctl.o          \
                rctlsys.o       \
                readlink.o      \
                refstr.o        \
                rename.o        \
                resolvepath.o   \
                retire_store.o  \
                process.o       \
                rlimit.o        \
                rmap.o          \
                rw.o            \
                rwstlock.o      \
                sad_conf.o      \
                sid.o           \
                sidsys.o        \
                sched.o         \
                schedctl.o      \
                sctp_crc32.o    \
                seg_dev.o       \
                seg_kp.o        \
                seg_kpm.o       \
                seg_map.o       \
                seg_vn.o        \
                seg_spt.o       \
                semaphore.o     \
                sendfile.o      \
                session.o       \
                share.o         \
                shuttle.o       \
                sig.o           \
                sigaction.o     \
                sigaltstack.o   \
                signotify.o     \
                sigpending.o    \
                sigprocmask.o   \
                sigqueue.o      \
                sigsendset.o    \
                sigsuspend.o    \
                sigtimedwait.o  \
                sleepq.o        \
                sock_conf.o     \
                space.o         \
                sscanf.o        \
                stat.o          \
                statfs.o        \
                statvfs.o       \
                stol.o          \
```



```
                str_conf.o      \
                strcalls.o      \
                stream.o        \
                streamio.o      \
                strext.o        \
                strsubr.o       \
                strsun.o        \
                subr.o          \
                sunddi.o        \
                sunmdi.o        \
                sunndi.o        \
                sunpci.o        \
                sunpm.o         \
                sundlpi.o       \
                suntpi.o        \
                swap_subr.o     \
                swap_vnops.o    \
                symlink.o       \
                sync.o          \
                sysclass.o      \
                sysconfig.o     \
                sysent.o        \
                sysfs.o         \
                systeminfo.o    \
                task.o          \
                taskq.o         \
                tasksys.o       \
                time.o          \
                timer.o         \
                times.o         \
                timers.o        \
                thread.o        \
                tlabel.o        \
                tnf_res.o       \
                turnstile.o     \
                tty_common.o    \
                u8_textprep.o   \
                uadmin.o        \
                uconv.o         \
                ucredsys.o      \
                uid.o           \
                umask.o         \
                umount.o        \
                uname.o         \
                unix_bb.o       \
                unlink.o        \
                urw.o           \
                utime.o         \
                utssys.o        \
                uucopy.o        \
                vfs.o           \
                vfs_conf.o      \
                vmem.o          \
                vm_anon.o       \
                vm_as.o         \
                vm_meter.o      \
                vm_pageout.o    \
                vm_pvn.o        \
                vm_rm.o         \
                vm_seg.o        \
                vm_subr.o       \
                vm_swap.o       \
                vm_usage.o      \
                vnode.o         \
                vuid_queue.o    \
                vuid_store.o    \
```

```
                waitq.o         \
                watchpoint.o    \
                yield.o         \
                scsi_confdata.o \
                xattr.o         \
                xattr_common.o  \
                xdr_mblk.o      \
                xdr_mem.o       \
                xdr.o           \
                xdr_array.o     \
                xdr_refer.o     \
                xhat.o          \
                zone.o

#
#       Stubs for the stand-alone linker/loader
#
sparc_GENSTUBS_OBJS =   \
        kobj_stubs.o

i386_GENSTUBS_OBJS =

COMMON_GENSTUBS_OBJS =

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

#
#       DTrace and DTrace Providers
#
DTRACE_OBJS += dtrace.o dtrace_isa.o dtrace_asm.o

SDT_OBJS += sdt_subr.o

PROFILE_OBJS += profile.o

SYSTRACE_OBJS += systrace.o

LOCKSTAT_OBJS += lockstat.o

FASTTRAP_OBJS += fasttrap.o fasttrap_isa.o

DCPC_OBJS += dcpc.o

#
#       Driver (pseudo-driver) Modules
#
IPP_OBJS +=     ippctl.o

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

AUDIOEMU10K_OBJS += audioemu10k.o

AUDIOENS_OBJS += audioens.o

AUDIOVIA823X_OBJS += audiovia823x.o

AUDIOVIA97_OBJS += audiovia97.o

AUDIO1575_OBJS += audio1575.o

AUDIO810_OBJS += audio810.o

AUDIOCMI_OBJS += audiocmi.o
```



```
AUDIOCMIHD_OBJS += audiocmihd.o

AUDIOHD_OBJS += audiohd.o

AUDIOIXP_OBJS += audioixp.o

AUDIOLS_OBJS += audiols.o

AUDIOP16X_OBJS += audiop16x.o

AUDIOPCI_OBJS += audiopci.o

AUDIOSOLO_OBJS += audiosolo.o

AUDIOTS_OBJS += audiots.o

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

BLKDEV_OBJS += blkdev.o

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

CONSKBD_OBJS += conskbd.o

CONSMS_OBJS +=  consms.o

OLDPTY_OBJS +=  tty_ptyconf.o

PTC_OBJS +=     tty_pty.o

PTSL_OBJS +=    tty_pts.o

PTM_OBJS +=     ptm.o

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

PTS_OBJS +=     pts.o

PTY_OBJS +=     ptms_conf.o

SAD_OBJS +=     sad.o

MD4_OBJS +=     md4.o md4_mod.o

MD5_OBJS +=     md5.o md5_mod.o

SHA1_OBJS +=    sha1.o sha1_mod.o

SHA2_OBJS +=    sha2.o sha2_mod.o

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

DSCPMK_OBJS +=  dscpmk.o dscpmkddi.o

DLCOSMK_OBJS += dlcosmk.o dlcosmkddi.o

FLOWACCT_OBJS +=        flowacctddi.o flowacct.o

TOKENMT_OBJS += tokenmt.o tokenmtddi.o

TSWTCL_OBJS +=  tswtcl.o tswtclddi.o

ARP_OBJS +=     arpddi.o
```

```
ICMP_OBJS +=    icmpddi.o

ICMP6_OBJS +=   icmp6ddi.o

RTS_OBJS +=     rtsddi.o

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

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

IP6_OBJS +=     ip6ddi.o

HOOK_OBJS +=    hook.o

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

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

IPNET_OBJS +=   ipnet.o ipnet_bpf.o

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

IPSECESP_OBJS += ipsecespddi.o ipsecesp.o

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

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

SPPPTUN_OBJS += sppptun.o sppptun_mod.o

SPPPASYN_OBJS += spppasyn.o spppasyn_mod.o

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

TCP_OBJS +=     tcpddi.o
```



```
TCP6_OBJS +=    tcp6ddi.o

NCA_OBJS +=     ncaddi.o

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

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

PFP_SOCK_MOD_OBJS += sockmod_pfp.o

RDS_SOCK_MOD_OBJS += sockmod_rds.o

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

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

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

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

UDP_OBJS +=     udpddi.o

UDP6_OBJS +=    udp6ddi.o

SY_OBJS +=      gentty.o

TCO_OBJS +=     ticots.o

TCOO_OBJS +=    ticotsord.o

TCL_OBJS +=     ticlts.o

TL_OBJS +=      tl.o

DUMP_OBJS +=    dump.o

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

CLONE_OBJS +=   clone.o

CN_OBJS +=      cons.o

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

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

GLD_OBJS +=     gld.o gldutil.o

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

MAC_6TO4_OBJS +=        mac_6to4.o

MAC_ETHER_OBJS +=       mac_ether.o

MAC_IPV4_OBJS +=        mac_ipv4.o

MAC_IPV6_OBJS +=        mac_ipv6.o
```

```
MAC_WIFI_OBJS +=        mac_wifi.o

MAC_IB_OBJS +=          mac_ib.o

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

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

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

NET80211_OBJS += net80211.o net80211_proto.o net80211_input.o \
                 net80211_output.o net80211_node.o net80211_crypto.o \
                 net80211_crypto_none.o net80211_crypto_wep.o net80211_ioctl.o \
                 net80211_crypto_tkip.o net80211_crypto_ccmp.o  \
                 net80211_ht.o

VNIC_OBJS +=    vnic_ctl.o vnic_dev.o

SIMNET_OBJS +=  simnet.o

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

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

IBDM_OBJS +=    ibdm.o

IBDMA_OBJS +=   ibdma.o

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

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

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

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

DAPLT_OBJS +=   daplt.o

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

SOL_UCMA_OBJS +=        sol_ucma.o

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

SOL_UMAD_OBJS += sol_umad.o

KSTAT_OBJS +=   kstat.o
```



```
KSYMS_OBJS +=   ksyms.o

INSTANCE_OBJS += inst_sync.o

IWSCN_OBJS +=   iwscons.o

LOFI_OBJS +=    lofi.o LzmaDec.o

FSSNAP_OBJS +=  fssnap.o

FSSNAPIF_OBJS += fssnap_if.o

MM_OBJS +=      mem.o

PHYSMEM_OBJS += physmem.o

OPTIONS_OBJS += options.o

WINLOCK_OBJS += winlockio.o

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

PSEUDO_OBJS +=  pseudonex.o

RAMDISK_OBJS += ramdisk.o

LLC1_OBJS += llc1.o

USBKBM_OBJS += usbkbm.o

USBWCM_OBJS += usbwcm.o

BOFI_OBJS += bofi.o

HID_OBJS += hid.o

HWA_RC_OBJS += hwarc.o

USBSKEL_OBJS += usbskel.o

USBVC_OBJS += usbvc.o usbvc_v4l2.o

HIDPARSER_OBJS += hidparser.o

USB_AC_OBJS += usb_ac.o

USB_AS_OBJS += usb_as.o

USB_AH_OBJS += usb_ah.o

USBMS_OBJS += usbms.o

USBPRN_OBJS += usbprn.o

UGEN_OBJS += ugen.o

USBSER_OBJS += usbser.o usbser_rseq.o

USBSACM_OBJS += usbsacm.o

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

USBS49_FW_OBJS += keyspan_49fw.o

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

```
WUSB_CA_OBJS += wusb_ca.o

USBFTDI_OBJS += usbser_uftdi.o uftdi_dsd.o

USBECM_OBJS += usbecm.o

WC_OBJS += wscons.o vcons.o

VCONS_CONF_OBJS += vcons_conf.o

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

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

SCSI_VHCI_F_SYM_OBJS +=         sym.o

SCSI_VHCI_F_TPGS_OBJS +=        tpgs.o

SCSI_VHCI_F_ASYM_SUN_OBJS +=    asym_sun.o

SCSI_VHCI_F_SYM_HDS_OBJS +=     sym_hds.o

SCSI_VHCI_F_TAPE_OBJS +=        tape.o

SCSI_VHCI_F_TPGS_TAPE_OBJS +=   tpgs_tape.o

SGEN_OBJS +=    sgen.o

SMP_OBJS +=     smp.o

SATA_OBJS +=    sata.o

USBA_OBJS +=    hcdi.o  usba.o  usbai.o hubdi.o parser.o genconsole.o \
                usbai_pipe_mgmt.o usbai_req.o usbai_util.o usbai_register.o \
                usba_devdb.o usba10_calls.o usba_ugen.o whcdi.o wa.o
USBA_WITHOUT_WUSB_OBJS +=       hcdi.o  usba.o  usbai.o hubdi.o parser.o gencons
                usbai_pipe_mgmt.o usbai_req.o usbai_util.o usbai_register.o \
                usba_devdb.o usba10_calls.o usba_ugen.o

USBA10_OBJS +=  usba10.o

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

RSMOPS_OBJS +=  rsmops.o

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

HCI1394_OBJS += hci1394.o hci1394_async.o hci1394_attach.o hci1394_buf.o \
                hci1394_csr.o hci1394_detach.o hci1394_extern.o \
                hci1394_ioctl.o hci1394_isoch.o hci1394_isr.o \
                hci1394_ixl_comp.o hci1394_ixl_isr.o hci1394_ixl_misc.o \
                hci1394_ixl_update.o hci1394_misc.o hci1394_ohci.o \
                hci1394_q.o hci1394_s1394if.o hci1394_tlabel.o \
                hci1394_tlist.o hci1394_vendor.o

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



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

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

SBP2_OBJS += cfgrom.o sbp2.o

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

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

NCALL_OBJS +=   ncall.o \
                ncall_stub.o

RDC_OBJS +=     rdc.o \
                rdc_dev.o \
                rdc_io.o \
                rdc_clnt.o \
                rdc_prot_xdr.o \
                rdc_svc.o \
                rdc_bitmap.o \
                rdc_health.o \
                rdc_subr.o \
                rdc_diskq.o

RDCSRV_OBJS +=  rdcsrv.o

RDCSTUB_OBJS += rdc_stub.o

SDBC_OBJS +=    sd_bcache.o \
                sd_bio.o \
                sd_conf.o \
                sd_ft.o \
                sd_hash.o \
                sd_io.o \
                sd_misc.o \
                sd_pcu.o \
                sd_tdaemon.o \
                sd_trace.o \
                sd_iob_impl0.o \
                sd_iob_impl1.o \
                sd_iob_impl2.o \
                sd_iob_impl3.o \
                sd_iob_impl4.o \
                sd_iob_impl5.o \
                sd_iob_impl6.o \
                sd_iob_impl7.o \
                safestore.o \
                safestore_ram.o

NSCTL_OBJS +=   nsctl.o \
                nsc_cache.o \
                nsc_disk.o \
                nsc_dev.o \
                nsc_freeze.o \
                nsc_gen.o \
                nsc_mem.o \
                nsc_ncallio.o \
                nsc_power.o \
                nsc_resv.o \
                nsc_rmspin.o \
                nsc_solaris.o \
                nsc_trap.o \
                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 \
        ql_isr.o ql_mbx.o ql_nx.o ql_xioctl.o ql_fw_table.o
```

```
QLC_FW_2200_OBJS += ql_fw_2200.o

QLC_FW_2300_OBJS += ql_fw_2300.o

QLC_FW_2400_OBJS += ql_fw_2400.o

QLC_FW_2500_OBJS += ql_fw_2500.o

QLC_FW_6322_OBJS += ql_fw_6322.o

QLC_FW_8100_OBJS += ql_fw_8100.o

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

ZCONS_OBJS += zcons.o

NV_SATA_OBJS += nv_sata.o

SI3124_OBJS += si3124.o

AHCI_OBJS += ahci.o

PCIIDE_OBJS += pci-ide.o

PCEPP_OBJS += pcepp.o

CPC_OBJS += cpc.o

CPUID_OBJS += cpuid_drv.o

SYSEVENT_OBJS += sysevent.o

BL_OBJS += bl.o

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

FM_OBJS += devfm.o devfm_machdep.o

RTLS_OBJS +=    rtls.o

#
#                       exec modules
#
AOUTEXEC_OBJS +=aout.o

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

INTPEXEC_OBJS +=intp.o

SHBINEXEC_OBJS +=shbin.o

JAVAEXEC_OBJS +=java.o

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

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



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

DCFS_OBJS +=    dc_vnops.o

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

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

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

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

FDFS_OBJS +=    fdops.o

FIFO_OBJS +=    fifosubr.o      fifovnops.o

PIPE_OBJS +=    pipe.o

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

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

NAMEFS_OBJS +=  namevfs.o       namevno.o

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

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

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

```
                smb_sid.o \
                smb_native.o \
                smb_netbios_util.o

SMBSRV_OBJS +=  $(SMBSRV_SHARED_OBJS)                   \
                smb_acl.o                               \
                smb_alloc.o                             \
                smb_close.o                             \
                smb_common_open.o                       \
                smb_common_transact.o                   \
                smb_create.o                            \
                smb_delete.o                            \
                smb_directory.o                         \
                smb_dispatch.o                          \
                smb_echo.o                              \
                smb_fem.o                               \
                smb_find.o                              \
                smb_flush.o                             \
                smb_fsinfo.o                            \
                smb_fsops.o                             \
                smb_init.o                              \
                smb_kdoor.o                             \
                smb_kshare.o                            \
                smb_kutil.o                             \
                smb_lock.o                              \
                smb_lock_byte_range.o                   \
                smb_locking_andx.o                      \
                smb_logoff_andx.o                       \
                smb_mangle_name.o                       \
                smb_mbuf_marshaling.o                   \
                smb_mbuf_util.o                         \
                smb_negotiate.o                         \
                smb_net.o                               \
                smb_node.o                              \
                smb_nt_cancel.o                         \
                smb_nt_create_andx.o                    \
                smb_nt_transact_create.o                \
                smb_nt_transact_ioctl.o                 \
                smb_nt_transact_notify_change.o         \
                smb_nt_transact_quota.o                 \
                smb_nt_transact_security.o              \
                smb_odir.o                              \
                smb_ofile.o                             \
                smb_open_andx.o                         \
                smb_opipe.o                             \
                smb_oplock.o                            \
                smb_pathname.o                          \
                smb_print.o                             \
                smb_process_exit.o                      \
                smb_query_fileinfo.o                    \
                smb_read.o                              \
                smb_rename.o                            \
                smb_sd.o                                \
                smb_seek.o                              \
                smb_server.o                            \
                smb_session.o                           \
                smb_session_setup_andx.o                \
                smb_set_fileinfo.o                      \
                smb_signing.o                           \
                smb_tree.o                              \
                smb_trans2_create_directory.o           \
                smb_trans2_dfs.o                        \
                smb_trans2_find.o                       \
                smb_tree_connect.o                      \
                smb_unlock_byte_range.o                 \
                smb_user.o                              \
```



```
                smb_vfs.o                               \
                smb_vops.o                              \
                smb_vss.o                               \
                smb_write.o                             \
                smb_write_raw.o

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

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

MNTFS_OBJS +=   mntvfsops.o     mntvnops.o

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

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

SOCK_OBJS +=    socksubr.o      sockvfsops.o    sockparams.o    \
                socksyscalls.o  socktpi.o       sockstr.o \
                sockcommon_vnops.o      sockcommon_subr.o \
                sockcommon_sops.o       sockcommon.o    \
                sock_notsupp.o  socknotify.o \
                nl7c.o          nl7curi.o       nl7chttp.o      nl7clogd.o \
                nl7cnca.o       sodirect.o      sockfilter.o

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

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

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

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

SMBFS_COMMON_OBJS += smbfs_ntacl.o
SMBFS_OBJS +=   smbfs_vfsops.o  smbfs_vnops.o   smbfs_node.o    \
                smbfs_acl.o     smbfs_client.o  smbfs_smb.o     \
                smbfs_subr.o    smbfs_subr2.o   \
                smbfs_rwlock.o  smbfs_xattr.o   \
                $(SMBFS_COMMON_OBJS)


#
#                       LVM modules
#
MD_OBJS += md.o md_error.o md_ioctl.o md_mddb.o md_names.o \
        md_med.o md_rename.o md_subr.o

MD_COMMON_OBJS = md_convert.o md_crc.o md_revchk.o

MD_DERIVED_OBJS = metamed_xdr.o meta_basic_xdr.o

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

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

HOTSPARES_OBJS += hotspares.o

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

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

NOTIFY_OBJS += md_notify.o

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

ZFS_COMMON_OBJS +=              \
        arc.o                   \
        blkptr.o                \
        bplist.o                \
        bpobj.o                 \
        bptree.o                \
        dbuf.o                  \
        ddt.o                   \
        ddt_zap.o               \
        dmu.o                   \
        dmu_diff.o              \
        dmu_send.o              \
        dmu_object.o            \
        dmu_objset.o            \
        dmu_traverse.o          \
        dmu_tx.o                \
        dnode.o                 \
        dnode_sync.o            \
        dsl_bookmark.o          \
        dsl_dir.o               \
        dsl_dataset.o           \
        dsl_deadlist.o          \
        dsl_destroy.o           \
        dsl_pool.o              \
        dsl_synctask.o          \
        dsl_userhold.o          \
        dmu_zfetch.o            \
        dsl_deleg.o             \
        dsl_prop.o              \
        dsl_scan.o              \
        zfeature.o              \
        gzip.o                  \
        lz4.o                   \
        lzjb.o                  \
        metaslab.o              \
        range_tree.o            \
        refcount.o              \
        rrwlock.o               \
        sa.o                    \
        sha256.o                \
        spa.o                   \
        spa_config.o            \
        spa_errlog.o            \
        spa_history.o           \
        spa_misc.o              \
        space_map.o             \
        space_reftree.o         \
        txg.o                   \
        uberblock.o             \
        unique.o                \
        vdev.o                  \
        vdev_cache.o            \
        vdev_file.o             \
```



```
        vdev_label.o            \
        vdev_mirror.o           \
        vdev_missing.o          \
        vdev_queue.o            \
        vdev_raidz.o            \
        vdev_root.o             \
        zap.o                   \
        zap_leaf.o              \
        zap_micro.o             \
        zfs_byteswap.o          \
        zfs_debug.o             \
        zfs_fm.o                \
        zfs_fuid.o              \
        zfs_sa.o                \
        zfs_znode.o             \
        zil.o                   \
        zio.o                   \
        zio_checksum.o          \
        zio_compress.o          \
        zio_inject.o            \
        zle.o                   \
        zrlock.o

ZFS_SHARED_OBJS +=              \
        zfeature_common.o       \
        zfs_comutil.o           \
        zfs_deleg.o             \
        zfs_fletcher.o          \
        zfs_namecheck.o         \
        zfs_prop.o              \
        zpool_prop.o            \
        zprop_common.o

ZFS_OBJS +=                     \
        $(ZFS_COMMON_OBJS)      \
        $(ZFS_SHARED_OBJS)      \
        vdev_disk.o             \
        zfs_acl.o               \
        zfs_ctldir.o            \
        zfs_dir.o               \
        zfs_ioctl.o             \
        zfs_log.o               \
        zfs_onexit.o            \
        zfs_replay.o            \
        zfs_rlock.o             \
        zfs_vfsops.o            \
        zfs_vnops.o             \
        zvol.o

ZUT_OBJS +=                     \
        zut.o

#
#                       streams modules
#
BUFMOD_OBJS     +=      bufmod.o

CONNLD_OBJS +=  connld.o

DEDUMP_OBJS +=  dedump.o

DRCOMPAT_OBJS +=        drcompat.o

LDLINUX_OBJS += ldlinux.o

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

```
PCKT_OBJS +=    pckt.o

PFMOD_OBJS +=   pfmod.o

PTEM_OBJS +=    ptem.o

REDIRMOD_OBJS += strredirm.o

TIMOD_OBJS +=   timod.o

TIRDWR_OBJS +=  tirdwr.o

TTCOMPAT_OBJS +=ttcompat.o

LOG_OBJS +=     log.o

PIPEMOD_OBJS += pipemod.o

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

KLMMOD_OBJS +=  klmmod.o \
                nlm_impl.o \
                nlm_rpc_handle.o \
                nlm_dispatch.o \
                nlm_rpc_svc.o \
                nlm_client.o \
                nlm_service.o \
                nlm_prot_clnt.o \
                nlm_prot_xdr.o \
                nlm_rpc_clnt.o \
                nsm_addr_clnt.o \
                nsm_addr_xdr.o \
                sm_inter_clnt.o \
                sm_inter_xdr.o

KLMOPS_OBJS +=  klmops.o

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

RLMOD_OBJS += rlmod.o

TELMOD_OBJS += telmod.o

CRYPTMOD_OBJS += cryptmod.o

KB_OBJS +=      kbd.o           keytables.o

#
#                       ID mapping module
#
IDMAP_OBJS +=   idmap_mod.o     idmap_kapi.o    idmap_xdr.o     idmap_cache.o

#
#                       scheduling class modules
#
SDC_OBJS +=             sysdc.o
```



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

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

IA_OBJS +=              ia.o

FSS_OBJS +=             fss.o

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

#
#                       Inter-Process Communication (IPC) modules
#
IPC_OBJS +=     ipc.o

IPCMSG_OBJS +=  msg.o

IPCSEM_OBJS +=  sem.o

IPCSHM_OBJS +=  shm.o

#
#                       bignum module
#
COMMON_BIGNUM_OBJS += bignum_mod.o bignumimpl.o

BIGNUM_OBJS += $(COMMON_BIGNUM_OBJS) $(BIGNUM_PSR_OBJS)

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

CRYPTOADM_OBJS += cryptoadm.o

CRYPTO_OBJS +=  crypto.o

DPROV_OBJS +=   dprov.o

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

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

ARCFOURPROV_OBJS += arcfour.o arcfour_crypt.o

BLOWFISHPROV_OBJS += blowfish.o blowfish_impl.o

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

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

```
SWRANDPROV_OBJS += swrand.o

#
#                       kernel SSL
#
KSSL_OBJS +=    kssl.o kssl_ioctl.o

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

#
#                       misc. modules
#

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

PCIC_OBJS +=    pcic.o

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

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

CONSCONFIG_OBJS += consconfig.o

CONSCONFIG_DACF_OBJS  += consconfig_dacf.o consplat.o

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

KBTRANS_OBJS +=                         \
                kbtrans.o               \
                kbtrans_keytables.o     \
                kbtrans_polled.o        \
                kbtrans_streams.o       \
                usb_keytables.o

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

KGSSD_DERIVED_OBJS = gssd_xdr.o

KGSS_DUMMY_OBJS += dmech.o

KSOCKET_OBJS += ksocket.o ksocket_mod.o

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

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

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

CRYPTO_ARCFOUR= k5_arcfour.o

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

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



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

# crypto/crc32
CRYPTO_CRC32= crc32.o

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

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

K5_KRB= kfree.o copy_key.o \
        parse.o init_ctx.o \
        ser_adata.o ser_addr.o \
        ser_auth.o ser_cksum.o \
        ser_key.o ser_princ.o \
        serialize.o unparse.o \
        ser_actx.o

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

SEAL=   seal.o unseal.o

MECH=   delete_sec_context.o \
        import_sec_context.o \
        gssapi_krb5.o \
        k5seal.o k5unseal.o k5sealv3.o \
        ser_sctx.o \
        sign.o \
        util_crypt.o  \
        util_validate.o  util_ordering.o  \
        util_seqnum.o util_set.o util_seed.o \
        wrap_size_limit.o verify.o


MECH_GEN= util_token.o


KGSS_KRB5_OBJS += krb5mech.o \
        $(MECH) $(SEAL) $(MECH_GEN) \
        $(CRYPTO) $(CRYPTO_DES) $(CRYPTO_DK) $(CRYPTO_ARCFOUR) \
        $(CRYPTO_ENC) $(CRYPTO_HASH) \
        $(CRYPTO_KEYHASH) $(CRYPTO_CRC32) \
        $(CRYPTO_OLD) \
        $(CRYPTO_RAW) $(K5_KRB) $(K5_OS)

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

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

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

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

STRPLUMB_OBJS += strplumb.o

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

PROF_OBJS +=    prf.o
```

```
SE_OBJS += se_driver.o

SYSACCT_OBJS += acct.o

ACCTCTL_OBJS += acctctl.o

EXACCTSYS_OBJS += exacctsys.o

KAIO_OBJS += aio.o

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

BUSRA_OBJS += busra.o

PCS_OBJS += pcs.o

PSET_OBJS +=    pset.o

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

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

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

HUBD_OBJS += hubd.o

USB_MID_OBJS += usb_mid.o

USB_IA_OBJS += usb_ia.o

UWBA_OBJS += uwba.o uwbai.o

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

HWAHC_OBJS += hwahc.o hwahc_util.o

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

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

IPD_OBJS += ipd.o

IBD_OBJS +=     ibd.o ibd_cm.o

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

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

DLPISTUB_OBJS += dlpistub.o

SDP_OBJS +=     sdpddi.o

TRILL_OBJS +=   trill.o

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

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



```
RPCIB_OBJS += rpcib.o

KMDB_OBJS += kdrv.o

AFE_OBJS += afe.o

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

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

EFE_OBJS += efe.o

ELXL_OBJS += elxl.o

HME_OBJS += hme.o

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

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

PCN_OBJS += pcn.o

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

URTW_OBJS += urtw.o

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

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

ATU_OBJS += atu.o

IPW_OBJS += ipw2100_hw.o ipw2100.o

IWI_OBJS += ipw2200_hw.o ipw2200.o

IWH_OBJS += iwh.o

IWK_OBJS += iwk2.o

IWP_OBJS += iwp.o

MWL_OBJS += mwl.o

MWLFW_OBJS += mwlfw_mode.o

WPI_OBJS += wpi.o

RAL_OBJS += rt2560.o ral_rate.o

RUM_OBJS += rum.o

RWD_OBJS += rt2661.o

RWN_OBJS += rt2860.o

UATH_OBJS += uath.o

UATHFW_OBJS += uathfw_mod.o

URAL_OBJS += ural.o
```

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

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

MXFE_OBJS += mxfe.o

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

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

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

BFE_OBJS += bfe.o

BRIDGE_OBJS += bridge.o

IDM_SHARED_OBJS += base64.o

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

VR_OBJS += vr.o

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

YGE_OBJS = yge.o

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

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

PCIEB_OBJS += pcieb.o

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

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

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

#
#       Chelsio Terminator 4 10G NIC driver module
#
```



```
CXGBE_OBJS =    cxgbe.o

#
#       PCI strings file
#
PCI_STRING_OBJS = pci_strings.o

NET_DACF_OBJS += net_dacf.o

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

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

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

#
#       e1000g module
#
E1000G_OBJS +=  e1000g_debug.o e1000g_main.o e1000g_alloc.o \
                e1000g_tx.o e1000g_rx.o e1000g_stat.o \
                e1000g_osdep.o e1000g_workarounds.o


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

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

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

#
#       NIU 10G/1G driver module
#
NXGE_OBJS =     nxge_mac.o nxge_ipp.o nxge_rxdma.o              \
                nxge_txdma.o nxge_txc.o nxge_main.o             \
                nxge_hw.o nxge_fzc.o nxge_virtual.o             \
```

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

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

NXGE_HCALL_OBJS =       \
                nxge_hcall.o

#
# Virtio modules
#

# Virtio core
VIRTIO_OBJS = virtio.o

# Virtio block driver
VIOBLK_OBJS = vioblk.o

#
#       kiconv modules
#
KICONV_EMEA_OBJS += kiconv_emea.o

KICONV_JA_OBJS += kiconv_ja.o

KICONV_KO_OBJS += kiconv_cck_common.o kiconv_ko.o

KICONV_SC_OBJS += kiconv_cck_common.o kiconv_sc.o

KICONV_TC_OBJS += kiconv_cck_common.o kiconv_tc.o

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

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

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

#
#       MEGARAID_SAS module
#
MEGA_SAS_OBJS = megaraid_sas.o

#
#       MR_SAS module
#
```



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

#
#       CPQARY3 module
#
CPQARY3_OBJS =  cpqary3.o cpqary3_noe.o cpqary3_talk2ctlr.o     \
                cpqary3_isr.o cpqary3_transport.o cpqary3_mem.o \
                cpqary3_scsi.o cpqary3_util.o cpqary3_ioctl.o   \
                cpqary3_bd.o

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

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

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

#       nulldriver module
#
NULLDRIVER_OBJS =       nulldriver.o

TPM_OBJS =      tpm.o tpm_hcall.o
```

```
**********************************************************
   73286 Thu Jun 12 17:28:22 2014
new/usr/src/uts/common/Makefile.rules
4546 mpt_sas needs enhancing to support LSI MPI2.5
**********************************************************
#
# 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 $@ $<
        $(CTFCONVERT_O)

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

```
        $(CTFCONVERT_O)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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



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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
        $(CTFCONVERT_O)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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



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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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



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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/audio/ac97/%.c
```



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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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



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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$(LINTS_DIR)/bz2%.ln:           $(COMMONBASE)/bzip2/%.c
```

```
         @($(LHEAD) $(LINT.c) -C $(LINTS_DIR)/`basename $@ .ln` $(BZ2LINT) $< $(

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/ctf/%.c
```



```
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/dtrace/%.c
        @($(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))

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

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/audio/impl/%.c
```

```
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/comstar/lu/stmf_sbd/%.c
```



```
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/ib/clients/rds/%.c
```

```
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/iwh/%.c
```



```
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

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

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

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

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/kb8042/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/mii/%.c
        @($(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:              $(UTSBASE)/common/io/ppp/spppptun/%.c
        @($(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))
```

```
**********************************************************
    1804 Thu Jun 12 17:28:22 2014
new/usr/src/uts/common/io/scsi/adapters/mpt_sas3/mpt_sas3.conf
4546 mpt_sas needs enhancing to support LSI MPI2.5
**********************************************************
#
# 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.
#
#

#
# The mpt_sas driver, as a pHCI driver, must specify the vHCI class it
# belongs to(scsi_vhci).
#
ddi-vhci-class="scsi_vhci";
#
# I/O multipathing feature (MPxIO) can be enabled or disabled using
# mpxio-disable property. Setting mpxio-disable="no" will activate
# I/O multipathing; setting mpxio-disable="yes" disables the feature.
#
# Global mpxio-disable property:
#
# To globally enable MPxIO on all LSI MPT SAS 2.0 controllers set:
# mpxio-disable="no";
#
# To globally disable MPxIO on all LSI MPT SAS 2.0 controllers set:
# mpxio-disable="yes";
#
# You can also enable or disable MPxIO on a per HBA basis.
# Per HBA settings override the global setting for the specified HBAs.
# To disable MPxIO on a controller whose parent is /pci@7c0/pci@0/pci@9
# and the unit-address is "0" set:
# name="mpt_sas" parent="/pci@7c0/pci@0/pci@9" unit-address="0" mpxio-disable="y
#
mpxio-disable="no";
#endif /* ! codereview */
```

```
**********************************************************
  468560 Thu Jun 12 17:28:22 2014
new/usr/src/uts/common/io/scsi/adapters/mpt_sas3/mptsas3.c
4546 mpt_sas needs enhancing to support LSI MPI2.5
**********************************************************
/*
 * 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 (c) 2014, Joyent, Inc. All rights reserved.
 * Copyright (c) 2014, Tegile Systems Inc. All rights reserved.
 * Copyright 2014 OmniTI Computer Consulting, Inc. All rights reserved.
 */

/*
 * 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.
 */

/*
 * mptsas3 - This is a driver based on LSI Logic’s MPT2.0/2.5 interface.
 *
 */
```



```
#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/file.h>
#include <sys/policy.h>
#include <sys/model.h>
#include <sys/sysevent.h>
#include <sys/sysevent/eventdefs.h>
#include <sys/sysevent/dr.h>
#include <sys/sata/sata_defs.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_sas3/mpi/mpi2_tool.h>
#include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_raid.h>
#pragma pack()

/*
 * private header files.
 *
 */
#include <sys/scsi/impl/scsi_reset_notify.h>
#include <sys/scsi/adapters/mpt_sas3/mptsas3_var.h>
#include <sys/scsi/adapters/mpt_sas3/mptsas3_ioctl.h>
#include <sys/scsi/adapters/mpt_sas3/mptsas3_smhba.h>
#include <sys/scsi/adapters/mpt_sas3/mptsas3_hash.h>
#include <sys/raidioctl.h>

#include <sys/fs/dv_node.h>     /* devfs_clean */

/*
 * FMA header files
 */
#include <sys/ddifm.h>
#include <sys/fm/protocol.h>
#include <sys/fm/util.h>
#include <sys/fm/io/ddi.h>

/*
 * autoconfiguration data and routines.
 */
static int mptsas_attach(dev_info_t *dip, ddi_attach_cmd_t cmd);
static int mptsas_detach(dev_info_t *devi, ddi_detach_cmd_t cmd);
static int mptsas_power(dev_info_t *dip, int component, int level);

/*
 * cb_ops function
 */
static int mptsas_ioctl(dev_t dev, int cmd, intptr_t data, int mode,
        cred_t *credp, int *rval);
#ifdef __sparc
static int mptsas_reset(dev_info_t *devi, ddi_reset_cmd_t cmd);
```

```
#else  /* __sparc */
static int mptsas_quiesce(dev_info_t *devi);
#endif  /* __sparc */

/*
 * Resource initilaization for hardware
 */
static void mptsas_setup_cmd_reg(mptsas_t *mpt);
static void mptsas_disable_bus_master(mptsas_t *mpt);
static void mptsas_hba_fini(mptsas_t *mpt);
static void mptsas_cfg_fini(mptsas_t *mptsas_blkp);
static int mptsas_hba_setup(mptsas_t *mpt);
static void mptsas_hba_teardown(mptsas_t *mpt);
static int mptsas_config_space_init(mptsas_t *mpt);
static void mptsas_config_space_fini(mptsas_t *mpt);
static void mptsas_iport_register(mptsas_t *mpt);
static int mptsas_smp_setup(mptsas_t *mpt);
static void mptsas_smp_teardown(mptsas_t *mpt);
static int mptsas_cache_create(mptsas_t *mpt);
static void mptsas_cache_destroy(mptsas_t *mpt);
static int mptsas_alloc_request_frames(mptsas_t *mpt);
static int mptsas_alloc_sense_bufs(mptsas_t *mpt);
static int mptsas_alloc_reply_frames(mptsas_t *mpt);
static int mptsas_alloc_free_queue(mptsas_t *mpt);
static int mptsas_alloc_post_queue(mptsas_t *mpt);
static void mptsas_free_post_queue(mptsas_t *mpt);
static void mptsas_alloc_reply_args(mptsas_t *mpt);
static int mptsas_alloc_extra_sgl_frame(mptsas_t *mpt, mptsas_cmd_t *cmd);
static void mptsas_free_extra_sgl_frame(mptsas_t *mpt, mptsas_cmd_t *cmd);
static int mptsas_init_chip(mptsas_t *mpt, int first_time);

/*
 * SCSA function prototypes
 */
static int mptsas_scsi_start(struct scsi_address *ap, struct scsi_pkt *pkt);
static int mptsas_scsi_reset(struct scsi_address *ap, int level);
static int mptsas_scsi_abort(struct scsi_address *ap, struct scsi_pkt *pkt);
static int mptsas_scsi_getcap(struct scsi_address *ap, char *cap, int tgtonly);
static int mptsas_scsi_setcap(struct scsi_address *ap, char *cap, int value,
    int tgtonly);
static void mptsas_scsi_dmafree(struct scsi_address *ap, struct scsi_pkt *pkt);
static struct scsi_pkt *mptsas_scsi_init_pkt(struct scsi_address *ap,
    struct scsi_pkt *pkt, struct buf *bp, int cmdlen, int statuslen,
        int tgtlen, int flags, int (*callback)(), caddr_t arg);
static void mptsas_scsi_sync_pkt(struct scsi_address *ap, struct scsi_pkt *pkt);
static void mptsas_scsi_destroy_pkt(struct scsi_address *ap,
    struct scsi_pkt *pkt);
static int mptsas_scsi_tgt_init(dev_info_t *hba_dip, dev_info_t *tgt_dip,
    scsi_hba_tran_t *hba_tran, struct scsi_device *sd);
static void mptsas_scsi_tgt_free(dev_info_t *hba_dip, dev_info_t *tgt_dip,
    scsi_hba_tran_t *hba_tran, struct scsi_device *sd);
static int mptsas_scsi_reset_notify(struct scsi_address *ap, int flag,
    void (*callback)(caddr_t), caddr_t arg);
static int mptsas_get_name(struct scsi_device *sd, char *name, int len);
static int mptsas_get_bus_addr(struct scsi_device *sd, char *name, int len);
static int mptsas_scsi_quiesce(dev_info_t *dip);
static int mptsas_scsi_unquiesce(dev_info_t *dip);
static int mptsas_bus_config(dev_info_t *pdip, uint_t flags,
    ddi_bus_config_op_t op, void *arg, dev_info_t **childp);

/*
 * SMP functions
 */
static int mptsas_smp_start(struct smp_pkt *smp_pkt);

/*
```



```
 * internal function prototypes.
 */
static void mptsas_list_add(mptsas_t *mpt);
static void mptsas_list_del(mptsas_t *mpt);

static int mptsas_quiesce_bus(mptsas_t *mpt);
static int mptsas_unquiesce_bus(mptsas_t *mpt);

static int mptsas_alloc_handshake_msg(mptsas_t *mpt, size_t alloc_size);
static void mptsas_free_handshake_msg(mptsas_t *mpt);

static void mptsas_ncmds_checkdrain(void *arg);

static int mptsas_prepare_pkt(mptsas_cmd_t *cmd);
static void mptsas_retry_pkt(mptsas_t *mpt, mptsas_cmd_t *sp);
static int mptsas_save_cmd_to_slot(mptsas_t *mpt, mptsas_cmd_t *cmd);
static int mptsas_accept_pkt(mptsas_t *mpt, mptsas_cmd_t *sp,
    int *tran_rval);
static void mptsas_accept_tx_waitqs(mptsas_t *mpt);
static void mptsas_unblock_tx_waitqs(mptsas_t *mpt);
static void mptsas_drain_tx_waitq(mptsas_t *mpt, mptsas_tx_waitqueue_t *txwq);
static int mptsas_check_targ_intxtion(mptsas_target_t *ptgt, int cmd_pkt_flags);

static int mptsas_do_detach(dev_info_t *dev);
static int mptsas_do_scsi_reset(mptsas_t *mpt, uint16_t devhdl);
static int mptsas_do_scsi_abort(mptsas_t *mpt, int target, int lun,
    struct scsi_pkt *pkt);
static int mptsas_scsi_capchk(char *cap, int tgtonly, int *cidxp);

static void mptsas_handle_qfull(mptsas_t *mpt, mptsas_cmd_t *cmd);
static void mptsas_handle_event(void *args);
static int mptsas_handle_event_sync(void *args);
static void mptsas_handle_dr(void *args);
static void mptsas_handle_topo_change(mptsas_topo_change_list_t *topo_node,
    dev_info_t *pdip);

static void mptsas_restart_cmd(void *);

static void mptsas_flush_hba(mptsas_t *mpt);
static void mptsas_flush_target(mptsas_t *mpt, ushort_t target, int lun,
        uint8_t tasktype);
static void mptsas_set_pkt_reason(mptsas_t *mpt, mptsas_cmd_t *cmd,
    uchar_t reason, uint_t stat);

static uint_t mptsas_intr(caddr_t arg1, caddr_t arg2);
static void mptsas_process_intr(mptsas_t *mpt, mptsas_reply_pqueue_t *rpqp,
    pMpi2ReplyDescriptorsUnion_t reply_desc_union);
static void mptsas_handle_scsi_io_success(mptsas_t *mpt,
    mptsas_reply_pqueue_t *rpqp, pMpi2ReplyDescriptorsUnion_t reply_desc);
static void mptsas_handle_address_reply(mptsas_t *mpt,
    pMpi2ReplyDescriptorsUnion_t reply_desc);
static int mptsas_wait_intr(mptsas_t *mpt, int polltime);
static void mptsas_sge_setup(mptsas_t *mpt, mptsas_cmd_t *cmd,
    uint32_t *control, pMpi2SCSIIORequest_t frame, ddi_acc_handle_t acc_hdl);

static void mptsas_watch(void *arg);
static void mptsas_watchsubr(mptsas_t *mpt);
static void mptsas_cmd_timeout(mptsas_t *mpt, mptsas_target_t *ptgt);

static void mptsas_start_passthru(mptsas_t *mpt, mptsas_cmd_t *cmd);
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 dataout_size, short timeout, int mode);
static int mptsas_free_devhdl(mptsas_t *mpt, uint16_t devhdl);
```

```
static uint8_t mptsas_get_fw_diag_buffer_number(mptsas_t *mpt,
    uint32_t unique_id);
static void mptsas_start_diag(mptsas_t *mpt, mptsas_cmd_t *cmd);
static int mptsas_post_fw_diag_buffer(mptsas_t *mpt,
    mptsas_fw_diagnostic_buffer_t *pBuffer, uint32_t *return_code);
static int mptsas_release_fw_diag_buffer(mptsas_t *mpt,
    mptsas_fw_diagnostic_buffer_t *pBuffer, uint32_t *return_code,
    uint32_t diag_type);
static int mptsas_diag_register(mptsas_t *mpt,
    mptsas_fw_diag_register_t *diag_register, uint32_t *return_code);
static int mptsas_diag_unregister(mptsas_t *mpt,
    mptsas_fw_diag_unregister_t *diag_unregister, uint32_t *return_code);
static int mptsas_diag_query(mptsas_t *mpt, mptsas_fw_diag_query_t *diag_query,
    uint32_t *return_code);
static int mptsas_diag_read_buffer(mptsas_t *mpt,
    mptsas_diag_read_buffer_t *diag_read_buffer, uint8_t *ioctl_buf,
    uint32_t *return_code, int ioctl_mode);
static int mptsas_diag_release(mptsas_t *mpt,
    mptsas_fw_diag_release_t *diag_release, uint32_t *return_code);
static int mptsas_do_diag_action(mptsas_t *mpt, uint32_t action,
    uint8_t *diag_action, uint32_t length, uint32_t *return_code,
    int ioctl_mode);
static int mptsas_diag_action(mptsas_t *mpt, mptsas_diag_action_t *data,
    int mode);

static int mptsas_pkt_alloc_extern(mptsas_t *mpt, mptsas_cmd_t *cmd,
    int cmdlen, int tgtlen, int statuslen, int kf);
static void mptsas_pkt_destroy_extern(mptsas_t *mpt, mptsas_cmd_t *cmd);

static int mptsas_kmem_cache_constructor(void *buf, void *cdrarg, int kmflags);
static void mptsas_kmem_cache_destructor(void *buf, void *cdrarg);

static int mptsas_cache_frames_constructor(void *buf, void *cdrarg,
    int kmflags);
static void mptsas_cache_frames_destructor(void *buf, void *cdrarg);

static void mptsas_check_scsi_io_error(mptsas_t *mpt, pMpi2SCSIIOReply_t reply,
    mptsas_cmd_t *cmd);
static void mptsas_check_task_mgt(mptsas_t *mpt,
    pMpi2SCSIManagementReply_t reply, mptsas_cmd_t *cmd);
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);

static int mptsas_alloc_active_slots(mptsas_t *mpt, int flag);
static void mptsas_free_active_slots(mptsas_t *mpt);
static int mptsas_start_cmd(mptsas_t *mpt, mptsas_cmd_t *cmd);

static void mptsas_restart_hba(mptsas_t *mpt);
static void mptsas_restart_waitq(mptsas_t *mpt);

static void mptsas_deliver_doneq_thread(mptsas_t *mpt,
    mptsas_done_list_t *dlist);
static void mptsas_doneq_add(mptsas_t *mpt, mptsas_cmd_t *cmd);
static void mptsas_rpdoneq_add(mptsas_t *mpt, mptsas_reply_pqueue_t *rpqp,
    mptsas_cmd_t *cmd);
static void mptsas_doneq_mv(mptsas_done_list_t *from,
    mptsas_doneq_thread_list_t *item);

static mptsas_cmd_t *mptsas_doneq_thread_rm(mptsas_t *mpt, uint64_t t);
static void mptsas_doneq_empty(mptsas_t *mpt);
static void mptsas_rpdoneq_empty(mptsas_reply_pqueue_t *rpqp);
static void mptsas_doneq_thread(mptsas_thread_arg_t *arg);
static void mptsas_tx_waitq_thread(mptsas_thread_arg_t *arg);

static mptsas_cmd_t *mptsas_waitq_rm(mptsas_t *mpt);
```



```
static void mptsas_waitq_delete(mptsas_t *mpt, mptsas_cmd_t *cmd);

static void mptsas_start_watch_reset_delay();
static void mptsas_setup_bus_reset_delay(mptsas_t *mpt);
static void mptsas_watch_reset_delay(void *arg);
static int mptsas_watch_reset_delay_subr(mptsas_t *mpt);
static void mptsas_set_throttle(struct mptsas *mpt, mptsas_target_t *ptgt,
    int what);
static void mptsas_set_throttle_mtx(struct mptsas *mpt, mptsas_target_t *ptgt,
    int what);
static void mptsas_remove_cmd_nomtx(mptsas_t *mpt, mptsas_cmd_t *cmd);

/*
 * helper functions
 */
static void mptsas_dump_cmd(mptsas_t *mpt, mptsas_cmd_t *cmd);

static dev_info_t *mptsas_find_child(dev_info_t *pdip, char *name);
static dev_info_t *mptsas_find_child_phy(dev_info_t *pdip, uint8_t phy);
static dev_info_t *mptsas_find_child_addr(dev_info_t *pdip, uint64_t sasaddr,
    int lun);
static mdi_pathinfo_t *mptsas_find_path_addr(dev_info_t *pdip, uint64_t sasaddr,
    int lun);
static mdi_pathinfo_t *mptsas_find_path_phy(dev_info_t *pdip, uint8_t phy);
static dev_info_t *mptsas_find_smp_child(dev_info_t *pdip, char *str_wwn);

static int mptsas_parse_address(char *name, uint64_t *wwid, uint8_t *phy,
    int *lun);
static int mptsas_parse_smp_name(char *name, uint64_t *wwn);

static mptsas_target_t *mptsas_phy_to_tgt(mptsas_t *mpt,
    mptsas_phymask_t phymask, uint8_t phy);
static mptsas_target_t *mptsas_wwid_to_ptgt(mptsas_t *mpt,
    mptsas_phymask_t phymask, uint64_t wwid);
static mptsas_smp_t *mptsas_wwid_to_psmp(mptsas_t *mpt,
    mptsas_phymask_t phymask, uint64_t wwid);

static int mptsas_inquiry(mptsas_t *mpt, mptsas_target_t *ptgt, int lun,
    uchar_t page, unsigned char *buf, int len, int *rlen, uchar_t evpd);

static int mptsas_get_target_device_info(mptsas_t *mpt, uint32_t page_address,
    uint16_t *handle, mptsas_target_t **pptgt);
static void mptsas_update_phymask(mptsas_t *mpt);

static int mptsas_send_sep(mptsas_t *mpt, mptsas_target_t *ptgt,
    uint32_t *status, uint8_t cmd);
static dev_info_t *mptsas_get_dip_from_dev(dev_t dev,
    mptsas_phymask_t *phymask);
static mptsas_target_t *mptsas_addr_to_ptgt(mptsas_t *mpt, char *addr,
    mptsas_phymask_t phymask);
static int mptsas_flush_led_status(mptsas_t *mpt, mptsas_target_t *ptgt);


/*
 * Enumeration / DR functions
 */
static void mptsas_config_all(dev_info_t *pdip);
static int mptsas_config_one_addr(dev_info_t *pdip, uint64_t sasaddr, int lun,
    dev_info_t **lundip);
static int mptsas_config_one_phy(dev_info_t *pdip, uint8_t phy, int lun,
    dev_info_t **lundip);

static int mptsas_config_target(dev_info_t *pdip, mptsas_target_t *ptgt);
static int mptsas_offline_targetdev(dev_info_t *pdip, char *name);
static void mptsas_offline_target(mptsas_t *mpt, mptsas_target_t *ptgt,
    uint8_t topo_flags, dev_info_t *parent);
```

```
static int mptsas_config_raid(dev_info_t *pdip, uint16_t target,
    dev_info_t **dip);

static int mptsas_config_luns(dev_info_t *pdip, mptsas_target_t *ptgt);
static int mptsas_probe_lun(dev_info_t *pdip, int lun,
    dev_info_t **dip, mptsas_target_t *ptgt);

static int mptsas_create_lun(dev_info_t *pdip, struct scsi_inquiry *sd_inq,
    dev_info_t **dip, mptsas_target_t *ptgt, int lun);

static int mptsas_create_phys_lun(dev_info_t *pdip, struct scsi_inquiry *sd,
    char *guid, dev_info_t **dip, mptsas_target_t *ptgt, int lun);
static int mptsas_create_virt_lun(dev_info_t *pdip, struct scsi_inquiry *sd,
    char *guid, dev_info_t **dip, mdi_pathinfo_t **pip, mptsas_target_t *ptgt,
    int lun);

static void mptsas_offline_missed_luns(dev_info_t *pdip,
    uint16_t *repluns, int lun_cnt, mptsas_target_t *ptgt);
static int mptsas_offline_lun(dev_info_t *pdip, dev_info_t *rdip,
    mdi_pathinfo_t *rpip, uint_t flags);

static int mptsas_config_smp(dev_info_t *pdip, uint64_t sas_wwn,
    dev_info_t **smp_dip);
static int mptsas_offline_smp(dev_info_t *pdip, mptsas_smp_t *smp_node,
    uint_t flags);

static int mptsas_event_query(mptsas_t *mpt, mptsas_event_query_t *data,
    int mode, int *rval);
static int mptsas_event_enable(mptsas_t *mpt, mptsas_event_enable_t *data,
    int mode, int *rval);
static int mptsas_event_report(mptsas_t *mpt, mptsas_event_report_t *data,
    int mode, int *rval);
static void mptsas_record_event(void *args);
static int mptsas_reg_access(mptsas_t *mpt, mptsas_reg_access_t *data,
    int mode);

mptsas_target_t *mptsas_tgt_alloc(mptsas_t *, uint16_t, uint64_t,
    uint32_t, mptsas_phymask_t, uint8_t);
static mptsas_smp_t *mptsas_smp_alloc(mptsas_t *, mptsas_smp_t *);
static int mptsas_online_smp(dev_info_t *pdip, mptsas_smp_t *smp_node,
    dev_info_t **smp_dip);

/*
 * Power management functions
 */
static int mptsas_get_pci_cap(mptsas_t *mpt);
static int mptsas_init_pm(mptsas_t *mpt);

/*
 * MPT MSI tunable:
 *
 * By default MSI is enabled on all supported platforms.
 */
boolean_t mptsas_enable_msi = B_TRUE;
boolean_t mptsas_enable_msix = B_TRUE;
boolean_t mptsas_physical_bind_failed_page_83 = B_FALSE;

/*
 * Global switch for use of MPI2.5 FAST PATH.
 */
boolean_t mptsas3_use_fastpath = B_TRUE;

static int mptsas_register_intrs(mptsas_t *);
static void mptsas_unregister_intrs(mptsas_t *);
static int mptsas_add_intrs(mptsas_t *, int);
```



```
static void mptsas_rem_intrs(mptsas_t *);

/*
 * FMA Prototypes
 */
static void mptsas_fm_init(mptsas_t *mpt);
static void mptsas_fm_fini(mptsas_t *mpt);
static int mptsas_fm_error_cb(dev_info_t *, ddi_fm_error_t *, const void *);

extern pri_t minclsyspri, maxclsyspri;
/*
 * NCPUS is used to determine some optimal configurations for number
 * of threads created to perform specific jobs. If we are invoked because
 * a disk is part of the root file system ncpus may still be 1 so check
 * boot_ncpus as well.
 */
extern int ncpus, boot_ncpus;
#define NCPUS   max(ncpus, boot_ncpus)

/*
 * This device is created by the SCSI pseudo nexus driver (SCSI vHCI).  It is
 * under this device that the paths to a physical device are created when
 * MPxIO is used.
 */
extern dev_info_t       *scsi_vhci_dip;

/*
 * Tunable timeout value for Inquiry VPD page 0x83
 * By default the value is 30 seconds.
 */
int mptsas_inq83_retry_timeout = 30;

/*
 * Tunable for default SCSI pkt timeout. Defaults to 5 seconds, which should
 * be plenty for INQUIRY and REPORT_LUNS, which are the only commands currently
 * issued by mptsas directly.
 */
int mptsas_scsi_pkt_time = 5;

/*
 * This is used to allocate memory for message frame storage, not for
 * data I/O DMA. All message frames must be stored in the first 4G of
 * physical memory.
 */
ddi_dma_attr_t mptsas_dma_attrs = {
        DMA_ATTR_V0,    /* attribute layout version             */
        0x0ull,         /* address low - should be 0 (longlong) */
        0xffffffffull,  /* address high - 32-bit max range      */
        0x00ffffffull,  /* count max - max DMA object size      */
        4,              /* allocation alignment requirements    */
        0x78,           /* burstsizes - binary encoded values   */
        1,              /* minxfer - gran. of DMA engine        */
        0x00ffffffull,  /* maxxfer - gran. of DMA engine        */
        0xffffffffull,  /* max segment size (DMA boundary)      */
        MPTSAS_MAX_DMA_SEGS, /* scatter/gather list length      */
        512,            /* granularity - device transfer size   */
        0               /* flags, set to 0                      */
};

/*
 * This is used for data I/O DMA memory allocation. (full 64-bit DMA
 * physical addresses are supported.)
 */
ddi_dma_attr_t mptsas_dma_attrs64 = {
        DMA_ATTR_V0,    /* attribute layout version             */
        0x0ull,         /* address low - should be 0 (longlong) */
```

```
        0xffffffffffffffffull,  /* address high - 64-bit max    */
        0x00ffffffull,  /* count max - max DMA object size      */
        4,              /* allocation alignment requirements    */
        0x78,           /* burstsizes - binary encoded values   */
        1,              /* minxfer - gran. of DMA engine        */
        0x00ffffffull,  /* maxxfer - gran. of DMA engine        */
        0xffffffffull,  /* max segment size (DMA boundary)      */
        MPTSAS_MAX_DMA_SEGS, /* scatter/gather list length      */
        512,            /* granularity - device transfer size   */
        0               /* flags, set to 0 */
};

ddi_device_acc_attr_t mptsas_dev_attr = {
        DDI_DEVICE_ATTR_V1,
        DDI_STRUCTURE_LE_ACC,
        DDI_STRICTORDER_ACC,
        DDI_DEFAULT_ACC
};

static struct cb_ops mptsas_cb_ops = {
        scsi_hba_open,          /* open */
        scsi_hba_close,         /* close */
        nodev,                  /* strategy */
        nodev,                  /* print */
        nodev,                  /* dump */
        nodev,                  /* read */
        nodev,                  /* write */
        mptsas_ioctl,           /* ioctl */
        nodev,                  /* devmap */
        nodev,                  /* mmap */
        nodev,                  /* segmap */
        nochpoll,               /* chpoll */
        ddi_prop_op,            /* cb_prop_op */
        NULL,                   /* streamtab */
        D_MP,                   /* cb_flag */
        CB_REV,                 /* rev */
        nodev,                  /* aread */
        nodev                   /* awrite */
};

static struct dev_ops mptsas_ops = {
        DEVO_REV,               /* devo_rev, */
        0,                      /* refcnt  */
        ddi_no_info,            /* info */
        nulldev,                /* identify */
        nulldev,                /* probe */
        mptsas_attach,          /* attach */
        mptsas_detach,          /* detach */
#ifdef  __sparc
        mptsas_reset,
#else
        nodev,                  /* reset */
#endif  /* __sparc */
        &mptsas_cb_ops,         /* driver operations */
        NULL,                   /* bus operations */
        mptsas_power,           /* power management */
#ifdef  __sparc
        ddi_quiesce_not_needed
#else
        mptsas_quiesce          /* quiesce */
#endif  /* __sparc */
};


#define MPTSAS_MOD_STRING "MPTSAS3 HBA Driver 00.00.01"
```



```
static struct modldrv modldrv = {
        &mod_driverops, /* Type of module. This one is a driver */
        MPTSAS_MOD_STRING, /* Name of the module. */
        &mptsas_ops,    /* driver ops */
};

static struct modlinkage modlinkage = {
        MODREV_1, &modldrv, NULL
};
#define TARGET_PROP     "target"
#define LUN_PROP        "lun"
#define LUN64_PROP      "lun64"
#define SAS_PROP        "sas-mpt"
#define MDI_GUID        "wwn"
#define NDI_GUID        "guid"
#define MPTSAS_DEV_GONE "mptsas_dev_gone"

/*
 * Local static data
 */
#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;
#endif  /* defined(MPTSAS_DEBUG) */
uint32_t mptsas_debug_resets = 0;

static kmutex_t         mptsas_global_mutex;
static void             *mptsas3_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;

/*
 * 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,
            MPTSAS_INITIAL_SOFT_SPACE);
        if (status != 0) {
                return (status);
        }

        if ((status = scsi_hba_init(&modlinkage)) != 0) {
                ddi_soft_state_fini(&mptsas3_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);
                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"));

        if ((status = mod_remove(&modlinkage)) == 0) {
                ddi_soft_state_fini(&mptsas3_state);
                scsi_hba_fini(&modlinkage);
                mutex_destroy(&mptsas_global_mutex);
                rw_destroy(&mptsas_global_rwlock);
                mutex_destroy(&mptsas_log_mutex);
        }
        return (status);
}

/*
 * The loadable-module _info(9E) entry point
 */
int
_info(struct modinfo *modinfop)
{
        /* CONSTCOND */
        ASSERT(NO_COMPETING_THREADS);
        NDBG0(("mptsas _info"));

        return (mod_info(&modlinkage, modinfop));
}

static int
mptsas_target_eval_devhdl(const void *op, void *arg)
{
        uint16_t dh = *(uint16_t *)arg;
        const mptsas_target_t *tp = op;

        return ((int)tp->m_devhdl - (int)dh);
}

static int
mptsas_target_eval_slot(const void *op, void *arg)
{
        mptsas_led_control_t *lcp = arg;
        const mptsas_target_t *tp = op;

        if (tp->m_enclosure != lcp->Enclosure)
                return ((int)tp->m_enclosure - (int)lcp->Enclosure);

        return ((int)tp->m_slot_num - (int)lcp->Slot);
}

static int
mptsas_target_eval_nowwn(const void *op, void *arg)
{
        uint8_t phy = *(uint8_t *)arg;
        const mptsas_target_t *tp = op;

        if (tp->m_addr.mta_wwn != 0)
```

```
                return (-1);

        return ((int)tp->m_phynum - (int)phy);
}

static int
mptsas_smp_eval_devhdl(const void *op, void *arg)
{
        uint16_t dh = *(uint16_t *)arg;
        const mptsas_smp_t *sp = op;

        return ((int)sp->m_devhdl - (int)dh);
}

static uint64_t
mptsas_target_addr_hash(const void *tp)
{
        const mptsas_target_addr_t *tap = tp;

        return ((tap->mta_wwn & 0xfffffffffffffULL) |
            ((uint64_t)tap->mta_phymask << 48));
}

static int
mptsas_target_addr_cmp(const void *a, const void *b)
{
        const mptsas_target_addr_t *aap = a;
        const mptsas_target_addr_t *bap = b;

        if (aap->mta_wwn < bap->mta_wwn)
                return (-1);
        if (aap->mta_wwn > bap->mta_wwn)
                return (1);
        return ((int)bap->mta_phymask - (int)aap->mta_phymask);
}

static void
mptsas_target_free(void *op)
{
        kmem_free(op, sizeof (mptsas_target_t));
}

static void
mptsas_smp_free(void *op)
{
        kmem_free(op, sizeof (mptsas_smp_t));
}

static void
mptsas_destroy_hashes(mptsas_t *mpt)
{
        mptsas_target_t *tp;
        mptsas_smp_t *sp;

        for (tp = refhash_first(mpt->m_targets); tp != NULL;
            tp = refhash_next(mpt->m_targets, tp)) {
                mutex_destroy(&tp->m_t_mutex);
                refhash_remove(mpt->m_targets, tp);
        }
        for (sp = refhash_first(mpt->m_smp_targets); sp != NULL;
            sp = refhash_next(mpt->m_smp_targets, sp)) {
                refhash_remove(mpt->m_smp_targets, sp);
        }
        refhash_destroy(mpt->m_targets);
        refhash_destroy(mpt->m_smp_targets);
        mpt->m_targets = NULL;
```



```
        mpt->m_smp_targets = NULL;
}

static int
mptsas_iport_attach(dev_info_t *dip, ddi_attach_cmd_t cmd)
{
        dev_info_t              *pdip;
        mptsas_t                *mpt;
        scsi_hba_tran_t         *hba_tran;
        char                    *iport = NULL;
        char                    phymask[MPTSAS_MAX_PHYS];
        mptsas_phymask_t        phy_mask = 0;
        int                     dynamic_port = 0;
        uint32_t                page_address;
        char                    initiator_wwnstr[MPTSAS_WWN_STRLEN];
        int                     rval = DDI_FAILURE;
        int                     i = 0;
        uint8_t                 numphys = 0;
        uint8_t                 phy_id;
        uint8_t                 phy_port = 0;
        uint16_t                attached_devhdl = 0;
        uint32_t                dev_info;
        uint64_t                attached_sas_wwn;
        uint16_t                dev_hdl;
        uint16_t                pdev_hdl;
        uint16_t                bay_num, enclosure, io_flags;
        char                    attached_wwnstr[MPTSAS_WWN_STRLEN];

        /* CONSTCOND */
        ASSERT(NO_COMPETING_THREADS);

        switch (cmd) {
        case DDI_ATTACH:
                break;

        case DDI_RESUME:
                /*
                 * If this a scsi-iport node, nothing to do here.
                 */
                return (DDI_SUCCESS);

        default:
                return (DDI_FAILURE);
        }

        pdip = ddi_get_parent(dip);

        if ((hba_tran = ndi_flavorv_get(pdip, SCSA_FLAVOR_SCSI_DEVICE)) ==
            NULL) {
                cmn_err(CE_WARN, "Failed attach iport because fail to "
                    "get tran vector for the HBA node");
                return (DDI_FAILURE);
        }

        mpt = TRAN2MPT(hba_tran);
        ASSERT(mpt != NULL);
        if (mpt == NULL)
                return (DDI_FAILURE);

        if ((hba_tran = ndi_flavorv_get(dip, SCSA_FLAVOR_SCSI_DEVICE)) ==
            NULL) {
                mptsas_log(mpt, CE_WARN, "Failed attach iport because fail to "
                    "get tran vector for the iport node");
                return (DDI_FAILURE);
        }
```

```
        /*
         * Overwrite parent's tran_hba_private to iport's tran vector
         */
        hba_tran->tran_hba_private = mpt;

        ddi_report_dev(dip);

        /*
         * Get SAS address for initiator port according dev_handle
         */
        iport = ddi_get_name_addr(dip);
        if (iport && strncmp(iport, "v0", 2) == 0) {
                if (ddi_prop_update_int(DDI_DEV_T_NONE, dip,
                    MPTSAS_VIRTUAL_PORT, 1) !=
                    DDI_PROP_SUCCESS) {
                        (void) ddi_prop_remove(DDI_DEV_T_NONE, dip,
                            MPTSAS_VIRTUAL_PORT);
                        mptsas_log(mpt, CE_WARN, "mptsas virtual port "
                            "prop update failed");
                        return (DDI_FAILURE);
                }
                return (DDI_SUCCESS);
        }

        mutex_enter(&mpt->m_mutex);
        for (i = 0; i < MPTSAS_MAX_PHYS; i++) {
                bzero(phymask, sizeof (phymask));
                (void) sprintf(phymask,
                    "%x", mpt->m_phy_info[i].phy_mask);
                if (strcmp(phymask, iport) == 0) {
                        break;
                }
        }

        if (i == MPTSAS_MAX_PHYS) {
                mptsas_log(mpt, CE_WARN, "Failed attach port %s because port"
                    "seems not exist", iport);
                mutex_exit(&mpt->m_mutex);
                return (DDI_FAILURE);
        }

        phy_mask = mpt->m_phy_info[i].phy_mask;

        if (mpt->m_phy_info[i].port_flags & AUTO_PORT_CONFIGURATION)
                dynamic_port = 1;
        else
                dynamic_port = 0;

        /*
         * Update PHY info for smhba
         */
        if (mptsas_smhba_phy_init(mpt)) {
                mutex_exit(&mpt->m_mutex);
                mptsas_log(mpt, CE_WARN, "mptsas phy update "
                    "failed");
                return (DDI_FAILURE);
        }

        mutex_exit(&mpt->m_mutex);

        numphys = 0;
        for (i = 0; i < MPTSAS_MAX_PHYS; i++) {
                if ((phy_mask >> i) & 0x01) {
                        numphys++;
                }
        }
```



```
        bzero(initiator_wwnstr, sizeof (initiator_wwnstr));
        (void) sprintf(initiator_wwnstr, "w%016"PRIx64,
            mpt->un.m_base_wwid);

        if (ddi_prop_update_string(DDI_DEV_T_NONE, dip,
            SCSI_ADDR_PROP_INITIATOR_PORT, initiator_wwnstr) !=
            DDI_PROP_SUCCESS) {
                (void) ddi_prop_remove(DDI_DEV_T_NONE,
                    dip, SCSI_ADDR_PROP_INITIATOR_PORT);
                mptsas_log(mpt, CE_WARN, "mptsas Initiator port "
                    "prop update failed");
                return (DDI_FAILURE);
        }
        if (ddi_prop_update_int(DDI_DEV_T_NONE, dip,
            MPTSAS_NUM_PHYS, numphys) !=
            DDI_PROP_SUCCESS) {
                (void) ddi_prop_remove(DDI_DEV_T_NONE, dip, MPTSAS_NUM_PHYS);
                return (DDI_FAILURE);
        }

        if (ddi_prop_update_int(DDI_DEV_T_NONE, dip,
            "phymask", phy_mask) !=
            DDI_PROP_SUCCESS) {
                (void) ddi_prop_remove(DDI_DEV_T_NONE, dip, "phymask");
                mptsas_log(mpt, CE_WARN, "mptsas phy mask "
                    "prop update failed");
                return (DDI_FAILURE);
        }

        if (ddi_prop_update_int(DDI_DEV_T_NONE, dip,
            "dynamic-port", dynamic_port) !=
            DDI_PROP_SUCCESS) {
                (void) ddi_prop_remove(DDI_DEV_T_NONE, dip, "dynamic-port");
                mptsas_log(mpt, CE_WARN, "mptsas dynamic port "
                    "prop update failed");
                return (DDI_FAILURE);
        }
        if (ddi_prop_update_int(DDI_DEV_T_NONE, dip,
            MPTSAS_VIRTUAL_PORT, 0) !=
            DDI_PROP_SUCCESS) {
                (void) ddi_prop_remove(DDI_DEV_T_NONE, dip,
                    MPTSAS_VIRTUAL_PORT);
                mptsas_log(mpt, CE_WARN, "mptsas virtual port "
                    "prop update failed");
                return (DDI_FAILURE);
        }
        mptsas_smhba_set_all_phy_props(mpt, dip, numphys, phy_mask,
            &attached_devhdl);

        mutex_enter(&mpt->m_mutex);
        page_address = (MPI2_SAS_DEVICE_PGAD_FORM_HANDLE &
            MPI2_SAS_DEVICE_PGAD_FORM_MASK) | (uint32_t)attached_devhdl;
        rval = mptsas_get_sas_device_page0(mpt, page_address, &dev_hdl,
            &attached_sas_wwn, &dev_info, &phy_port, &phy_id,
            &pdev_hdl, &bay_num, &enclosure, &io_flags);
        if (rval != DDI_SUCCESS) {
                mptsas_log(mpt, CE_WARN,
                    "Failed to get device page0 for handle:%d",
                    attached_devhdl);
                mutex_exit(&mpt->m_mutex);
                return (DDI_FAILURE);
        }

        for (i = 0; i < MPTSAS_MAX_PHYS; i++) {
                bzero(phymask, sizeof (phymask));
```

```
                (void) sprintf(phymask, "%x", mpt->m_phy_info[i].phy_mask);
                if (strcmp(phymask, iport) == 0) {
                        (void) sprintf(&mpt->m_phy_info[i].smhba_info.path[0],
                            "%x",
                            mpt->m_phy_info[i].phy_mask);
                }
        }
        mutex_exit(&mpt->m_mutex);

        bzero(attached_wwnstr, sizeof (attached_wwnstr));
        (void) sprintf(attached_wwnstr, "w%016"PRIx64,
            attached_sas_wwn);
        if (ddi_prop_update_string(DDI_DEV_T_NONE, dip,
            SCSI_ADDR_PROP_ATTACHED_PORT, attached_wwnstr) !=
            DDI_PROP_SUCCESS) {
                (void) ddi_prop_remove(DDI_DEV_T_NONE,
                    dip, SCSI_ADDR_PROP_ATTACHED_PORT);
                return (DDI_FAILURE);
        }

        /* Create kstats for each phy on this iport */

        mptsas_create_phy_stats(mpt, iport, dip);

        /*
         * register sas hba iport with mdi (MPxIO/vhci)
         */
        if (mdi_phci_register(MDI_HCI_CLASS_SCSI,
            dip, 0) == MDI_SUCCESS) {
                mpt->m_mpxio_enable = TRUE;
        }
        return (DDI_SUCCESS);
}

/*
 * Notes:
 *      Set up all device state and allocate data structures,
 *      mutexes, condition variables, etc. for device operation.
 *      Add interrupts needed.
 *      Return DDI_SUCCESS if device is ready, else return DDI_FAILURE.
 */
static int
mptsas_attach(dev_info_t *dip, ddi_attach_cmd_t cmd)
{
        mptsas_t                *mpt = NULL;
        int                     instance, i, j;
        int                     q_thread_num;
        char                    map_setup = 0;
        char                    config_setup = 0;
        char                    hba_attach_setup = 0;
        char                    smp_attach_setup = 0;
        char                    mutex_init_done = 0;
        char                    event_taskq_create = 0;
        char                    dr_taskq_create = 0;
        char                    doneq_thread_create = 0;
        char                    txwq_thread_create = 0;
        char                    added_watchdog = 0;
        scsi_hba_tran_t         *hba_tran;
        uint_t                  mem_bar = MEM_SPACE;
        int                     rval = DDI_FAILURE;

        /* CONSTCOND */
        ASSERT(NO_COMPETING_THREADS);

        if (scsi_hba_iport_unit_address(dip)) {
                return (mptsas_iport_attach(dip, cmd));
```



```
        }

        switch (cmd) {
        case DDI_ATTACH:
                break;

        case DDI_RESUME:
                if ((hba_tran = ddi_get_driver_private(dip)) == NULL)
                        return (DDI_FAILURE);

                mpt = TRAN2MPT(hba_tran);

                if (!mpt) {
                        return (DDI_FAILURE);
                }

                /*
                 * Reset hardware and softc to "no outstanding commands"
                 * Note that a check condition can result on first command
                 * to a target.
                 */
                mutex_enter(&mpt->m_mutex);

                /*
                 * raise power.
                 */
                if (mpt->m_options & MPTSAS_OPT_PM) {
                        mutex_exit(&mpt->m_mutex);
                        (void) pm_busy_component(dip, 0);
                        rval = pm_power_has_changed(dip, 0, PM_LEVEL_D0);
                        if (rval == DDI_SUCCESS) {
                                mutex_enter(&mpt->m_mutex);
                        } else {
                                /*
                                 * The pm_raise_power() call above failed,
                                 * and that can only occur if we were unable
                                 * to reset the hardware.  This is probably
                                 * due to unhealty hardware, and because
                                 * important filesystems(such as the root
                                 * filesystem) could be on the attached disks,
                                 * it would not be a good idea to continue,
                                 * as we won't be entirely certain we are
                                 * writing correct data.  So we panic() here
                                 * to not only prevent possible data corruption,
                                 * but to give developers or end users a hope
                                 * of identifying and correcting any problems.
                                 */
                                fm_panic("mptsas could not reset hardware "
                                    "during resume");
                        }
                }

                mpt->m_suspended = 0;

                /*
                 * Reinitialize ioc
                 */
                mpt->m_softstate |= MPTSAS_SS_MSG_UNIT_RESET;
                if (mptsas_init_chip(mpt, FALSE) == DDI_FAILURE) {
                        mutex_exit(&mpt->m_mutex);
                        if (mpt->m_options & MPTSAS_OPT_PM) {
                                (void) pm_idle_component(dip, 0);
                        }
                        fm_panic("mptsas init chip fail during resume");
                }
                /*
```

```
                 * mptsas_update_driver_data needs interrupts so enable them
                 * first.
                 */
                MPTSAS_ENABLE_INTR(mpt);
                mptsas_update_driver_data(mpt);

                /* start requests, if possible */
                mptsas_restart_hba(mpt);

                mutex_exit(&mpt->m_mutex);

                /*
                 * Restart watch thread
                 */
                mutex_enter(&mptsas_global_mutex);
                if (mptsas_timeout_id == 0) {
                        mptsas_timeout_id = timeout(mptsas_watch, NULL,
                            mptsas_tick);
                        mptsas_timeouts_enabled = 1;
                }
                mutex_exit(&mptsas_global_mutex);

                /* report idle status to pm framework */
                if (mpt->m_options & MPTSAS_OPT_PM) {
                        (void) pm_idle_component(dip, 0);
                }

                return (DDI_SUCCESS);

        default:
                return (DDI_FAILURE);

        }

        instance = ddi_get_instance(dip);

        /*
         * Allocate softc information.
         */
        if (ddi_soft_state_zalloc(mptsas3_state, instance) != DDI_SUCCESS) {
                mptsas_log(NULL, CE_WARN,
                    "mptsas%d: cannot allocate soft state", instance);
                goto fail;
        }

        mpt = ddi_get_soft_state(mptsas3_state, instance);

        if (mpt == NULL) {
                mptsas_log(NULL, CE_WARN,
                    "mptsas%d: cannot get soft state", instance);
                goto fail;
        }

        /* Indicate that we are 'sizeof (scsi_*(9S))' clean. */
        scsi_size_clean(dip);

        mpt->m_dip = dip;
        mpt->m_instance = instance;

        /* Make a per-instance copy of the structures */
        mpt->m_io_dma_attr = mptsas_dma_attrs64;
        if (mptsas_use_64bit_msgaddr) {
                mpt->m_msg_dma_attr = mptsas_dma_attrs64;
        } else {
                mpt->m_msg_dma_attr = mptsas_dma_attrs;
        }
```



```
        mpt->m_reg_acc_attr = mptsas_dev_attr;
        mpt->m_dev_acc_attr = mptsas_dev_attr;

        /*
         * Round down the arq sense buffer size to nearest 16 bytes.
         */
        mpt->m_req_sense_size = EXTCMDS_STATUS_SIZE;

        /*
         * Initialize FMA
         */
        mpt->m_fm_capabilities = ddi_getprop(DDI_DEV_T_ANY, mpt->m_dip,
            DDI_PROP_CANSLEEP | DDI_PROP_DONTPASS, "fm-capable",
            DDI_FM_EREPORT_CAPABLE | DDI_FM_ACCCHK_CAPABLE |
            DDI_FM_DMACHK_CAPABLE | DDI_FM_ERRCB_CAPABLE);

        mptsas_fm_init(mpt);

        if (mptsas_alloc_handshake_msg(mpt,
            sizeof (Mpi2SCSITaskManagementRequest_t)) == DDI_FAILURE) {
                mptsas_log(mpt, CE_WARN, "cannot initialize handshake msg.");
                goto fail;
        }

        /*
         * Setup configuration space
         */
        if (mptsas_config_space_init(mpt) == FALSE) {
                mptsas_log(mpt, CE_WARN, "mptsas_config_space_init failed");
                goto fail;
        }
        config_setup++;

        if (ddi_regs_map_setup(dip, mem_bar, (caddr_t *)&mpt->m_reg,
            0, 0, &mpt->m_reg_acc_attr, &mpt->m_datap) != DDI_SUCCESS) {
                mptsas_log(mpt, CE_WARN, "map setup failed");
                goto fail;
        }
        map_setup++;

        /*
         * A taskq is created for dealing with the event handler
         */
        if ((mpt->m_event_taskq = ddi_taskq_create(dip, "mptsas_event_taskq",
            1, TASKQ_DEFAULTPRI, 0)) == NULL) {
                mptsas_log(mpt, CE_NOTE, "ddi_taskq_create failed");
                goto fail;
        }
        event_taskq_create++;

        /*
         * A taskq is created for dealing with dr events
         */
        if ((mpt->m_dr_taskq = ddi_taskq_create(dip,
            "mptsas_dr_taskq",
            1, TASKQ_DEFAULTPRI, 0)) == NULL) {
                mptsas_log(mpt, CE_NOTE, "ddi_taskq_create for discovery "
                    "failed");
                goto fail;
        }
        dr_taskq_create++;

        cv_init(&mpt->m_qthread_cv, NULL, CV_DRIVER, NULL);
        mutex_init(&mpt->m_qthread_mutex, NULL, MUTEX_DRIVER, NULL);

        i = ddi_prop_get_int(DDI_DEV_T_ANY, dip,
```

```
            0, "mptsas_enable_txwq_prop", NCPUS > 1);
        if (i) {
                mpt->m_txwq_thread_n = NUM_TX_WAITQ;
                mpt->m_txwq_enabled = FALSE;
                if (ddi_prop_get_int(DDI_DEV_T_ANY, dip,
                    0, "mptsas_allow_txwq_jumping", 0)) {
                        mpt->m_txwq_allow_q_jumping = TRUE;
                }
                i = ddi_prop_get_int(DDI_DEV_T_ANY, dip,
                    0, "mptsas_txwq_threashold_prop", 80000);
                mpt->m_txwq_thread_threshold = (uint16_t)i;
        } else {
                mpt->m_txwq_thread_n = 0;
                mpt->m_txwq_enabled = FALSE;
        }

        if (mpt->m_txwq_thread_n) {
                mutex_enter(&mpt->m_qthread_mutex);
                for (j = 0; j < NUM_TX_WAITQ; j++) {
                        mutex_init(&mpt->m_tx_waitq[j].txwq_mutex, NULL,
                            MUTEX_DRIVER,
                            NULL);
                        cv_init(&mpt->m_tx_waitq[j].txwq_cv, NULL, CV_DRIVER,
                            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++;
        }

        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));

        if (mpt->m_doneq_thread_n) {
                mutex_enter(&mpt->m_qthread_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;
                        mutex_exit(&mpt->m_doneq_thread_id[j].mutex);
                }
                mutex_exit(&mpt->m_qthread_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);
        mutex_init(&mpt->m_passthru_mutex, NULL, MUTEX_DRIVER, NULL);
        for (i = 0; i < MPTSAS_MAX_PHYS; i++) {
                mutex_init(&mpt->m_phy_info[i].smhba_info.phy_mutex,
                    NULL, MUTEX_DRIVER, NULL);
        }

        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++;

        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;
        }

        /*
         * Fill in the phy_info structure and get the base WWID
```

```
 */
        if (mptsas_get_manufacture_page5(mpt) == DDI_FAILURE) {
                mptsas_log(mpt, CE_WARN,
                    "mptsas_get_manufacture_page5 failed!");
                goto fail;
        }

        if (mptsas_get_sas_io_unit_page_hndshk(mpt)) {
                mptsas_log(mpt, CE_WARN,
                    "mptsas_get_sas_io_unit_page_hndshk failed!");
                goto fail;
        }

        if (mptsas_get_manufacture_page0(mpt) == DDI_FAILURE) {
                mptsas_log(mpt, CE_WARN,
                    "mptsas_get_manufacture_page0 failed!");
                goto fail;
        }

        /*
         * If we only have one interrupt the default for doneq_thread_threshold
         * should be 0 so that all completion processing goes to the threads.
         * Only change it if it wasn't set from .conf file.
         */
        if (mpt->m_doneq_thread_n != 0 &&
            ddi_prop_exists(DDI_DEV_T_ANY, dip,
            0, "mptsas_doneq_length_threshold_prop") == 0 &&
            mpt->m_intr_cnt == 1) {
                mpt->m_doneq_length_threshold = 0;
        }


        mutex_exit(&mpt->m_mutex);

        /*
         * Register the iport for multiple port HBA
         */
        mptsas_iport_register(mpt);

        /*
         * initialize SCSI HBA transport structure
         */
        if (mptsas_hba_setup(mpt) == FALSE)
                goto fail;
        hba_attach_setup++;

        if (mptsas_smp_setup(mpt) == FALSE)
                goto fail;
        smp_attach_setup++;

        if (mptsas_cache_create(mpt) == FALSE)
                goto fail;

        mpt->m_scsi_reset_delay = ddi_prop_get_int(DDI_DEV_T_ANY,
            dip, 0, "scsi-reset-delay", SCSI_DEFAULT_RESET_DELAY);
        if (mpt->m_scsi_reset_delay == 0) {
                mptsas_log(mpt, CE_NOTE,
                    "scsi_reset_delay of 0 is not recommended,"
                    " resetting to SCSI_DEFAULT_RESET_DELAY\n");
                mpt->m_scsi_reset_delay = SCSI_DEFAULT_RESET_DELAY;
        }

        /*
         * Initialize the wait and done FIFO queue
         */
        mpt->m_dlist.dl_tail = &mpt->m_dlist.dl_q;
```



```
        mpt->m_waitqtail = &mpt->m_waitq;

        /*
         * ioc cmd queue initialize
         */
        mpt->m_ioc_event_cmdtail = &mpt->m_ioc_event_cmdq;
        mpt->m_dev_handle = 0xFFFF;

        MPTSAS_ENABLE_INTR(mpt);

        /*
         * enable event notification
         */
        mutex_enter(&mpt->m_mutex);
        if (mptsas_ioc_enable_event_notification(mpt)) {
                mutex_exit(&mpt->m_mutex);
                goto fail;
        }
        mutex_exit(&mpt->m_mutex);

        /*
         * 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);
        added_watchdog++;

        /*
         * Initialize PHY info for smhba.
         * This requires watchdog to be enabled otherwise if interrupts
         * don't work the system will hang.
         */
        if (mptsas_smhba_setup(mpt)) {
                mptsas_log(mpt, CE_WARN, "mptsas phy initialization "
                    "failed");
                goto fail;
        }

        /* Check all dma handles allocated in attach */
        if ((mptsas_check_dma_handle(mpt->m_dma_req_frame_hdl)
            != DDI_SUCCESS) ||
            (mptsas_check_dma_handle(mpt->m_dma_req_sense_hdl)
            != DDI_SUCCESS) ||
            (mptsas_check_dma_handle(mpt->m_dma_reply_frame_hdl)
            != DDI_SUCCESS) ||
            (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) ||
            (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.
         */

        /* 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);
                        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);
                }

                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) {
                        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_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);
                        }
                        for (j = 0; j < q_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);
                }
                if (txwq_thread_create) {
                        mutex_enter(&mpt->m_qthread_mutex);
                        q_thread_num = mpt->m_txwq_thread_n;
                        for (j = 0; j < q_thread_num; j++) {
                                mutex_enter(&mpt->m_tx_waitq[j].txwq_mutex);
                                mpt->m_tx_waitq[j].txwq_active = FALSE;
                                cv_signal(&mpt->m_tx_waitq[j].txwq_cv);
                                mutex_exit(&mpt->m_tx_waitq[j].txwq_mutex);
                        }
                        while (mpt->m_txwq_thread_n) {
                                cv_wait(&mpt->m_qthread_cv,
                                    &mpt->m_qthread_mutex);
                        }
                        for (j = 0; j < q_thread_num; j++) {
                                cv_destroy(&mpt->m_tx_waitq[j].txwq_cv);
                                cv_destroy(&mpt->m_tx_waitq[j].txwq_drain_cv);
                                mutex_destroy(&mpt->m_tx_waitq[j].txwq_mutex);
                        }
                }
                if (event_taskq_create) {
                        ddi_taskq_destroy(mpt->m_event_taskq);
                }
                if (dr_taskq_create) {
                        ddi_taskq_destroy(mpt->m_dr_taskq);
                }
                if (mutex_init_done) {
                        mutex_destroy(&mpt->m_qthread_mutex);
                        mutex_destroy(&mpt->m_passthru_mutex);
                        mutex_destroy(&mpt->m_mutex);
                        for (i = 0; i < MPTSAS_MAX_PHYS; i++) {
                                mutex_destroy(
                                    &mpt->m_phy_info[i].smhba_info.phy_mutex);
                        }
                        cv_destroy(&mpt->m_qthread_cv);
```

```
                        cv_destroy(&mpt->m_cv);
                        cv_destroy(&mpt->m_passthru_cv);
                        cv_destroy(&mpt->m_fw_cv);
                        cv_destroy(&mpt->m_config_cv);
                        cv_destroy(&mpt->m_fw_diag_cv);
                }

                if (map_setup) {
                        mptsas_cfg_fini(mpt);
                }
                if (config_setup) {
                        mptsas_config_space_fini(mpt);
                }
                mptsas_free_handshake_msg(mpt);
                mptsas_hba_fini(mpt);

                mptsas_fm_fini(mpt);
                ddi_soft_state_free(mptsas3_state, instance);
                ddi_prop_remove_all(dip);
        }
        return (DDI_FAILURE);
}

static int
mptsas_suspend(dev_info_t *devi)
{
        mptsas_t        *mpt, *g;
        scsi_hba_tran_t *tran;

        if (scsi_hba_iport_unit_address(devi)) {
                return (DDI_SUCCESS);
        }

        if ((tran = ddi_get_driver_private(devi)) == NULL)
                return (DDI_SUCCESS);

        mpt = TRAN2MPT(tran);
        if (!mpt) {
                return (DDI_SUCCESS);
        }

        mutex_enter(&mpt->m_mutex);

        if (mpt->m_suspended++) {
                mutex_exit(&mpt->m_mutex);
                return (DDI_SUCCESS);
        }

        /*
         * Cancel timeout threads for this mpt
         */
        if (mpt->m_quiesce_timeid) {
                timeout_id_t tid = mpt->m_quiesce_timeid;
                mpt->m_quiesce_timeid = 0;
                mutex_exit(&mpt->m_mutex);
                (void) untimeout(tid);
                mutex_enter(&mpt->m_mutex);
        }

        if (mpt->m_restart_cmd_timeid) {
                timeout_id_t tid = mpt->m_restart_cmd_timeid;
                mpt->m_restart_cmd_timeid = 0;
                mutex_exit(&mpt->m_mutex);
                (void) untimeout(tid);
                mutex_enter(&mpt->m_mutex);
        }
```



```
        mutex_exit(&mpt->m_mutex);

        (void) pm_idle_component(mpt->m_dip, 0);

        /*
         * Cancel watch threads if all mpts suspended
         */
        rw_enter(&mptsas_global_rwlock, RW_WRITER);
        for (g = mptsas_head; g != NULL; g = g->m_next) {
                if (!g->m_suspended)
                        break;
        }
        rw_exit(&mptsas_global_rwlock);

        mutex_enter(&mptsas_global_mutex);
        if (g == NULL) {
                timeout_id_t tid;

                mptsas_timeouts_enabled = 0;
                if (mptsas_timeout_id) {
                        tid = mptsas_timeout_id;
                        mptsas_timeout_id = 0;
                        mutex_exit(&mptsas_global_mutex);
                        (void) untimeout(tid);
                        mutex_enter(&mptsas_global_mutex);
                }
                if (mptsas_reset_watch) {
                        tid = mptsas_reset_watch;
                        mptsas_reset_watch = 0;
                        mutex_exit(&mptsas_global_mutex);
                        (void) untimeout(tid);
                        mutex_enter(&mptsas_global_mutex);
                }
        }
        mutex_exit(&mptsas_global_mutex);

        mutex_enter(&mpt->m_mutex);

        /*
         * If this mpt is not in full power(PM_LEVEL_D0), just return.
         */
        if ((mpt->m_options & MPTSAS_OPT_PM) &&
            (mpt->m_power_level != PM_LEVEL_D0)) {
                mutex_exit(&mpt->m_mutex);
                return (DDI_SUCCESS);
        }

        /* Disable HBA interrupts in hardware */
        MPTSAS_DISABLE_INTR(mpt);
        /*
         * Send RAID action system shutdown to sync IR
         */
        mptsas_raid_action_system_shutdown(mpt);

        mutex_exit(&mpt->m_mutex);

        /* drain the taskq */
        ddi_taskq_wait(mpt->m_event_taskq);
        ddi_taskq_wait(mpt->m_dr_taskq);

        return (DDI_SUCCESS);
}

#ifdef  __sparc
/*ARGSUSED*/
```

```
static int
mptsas_reset(dev_info_t *devi, ddi_reset_cmd_t cmd)
{
        mptsas_t        *mpt;
        scsi_hba_tran_t *tran;

        /*
         * If this call is for iport, just return.
         */
        if (scsi_hba_iport_unit_address(devi))
                return (DDI_SUCCESS);

        if ((tran = ddi_get_driver_private(devi)) == NULL)
                return (DDI_SUCCESS);

        if ((mpt = TRAN2MPT(tran)) == NULL)
                return (DDI_SUCCESS);

        /*
         * Send RAID action system shutdown to sync IR.  Disable HBA
         * interrupts in hardware first.
         */
        MPTSAS_DISABLE_INTR(mpt);
        mptsas_raid_action_system_shutdown(mpt);

        return (DDI_SUCCESS);
}
#else /* __sparc */
/*
 * quiesce(9E) entry point.
 *
 * This function is called when the system is single-threaded at high
 * PIL with preemption disabled. Therefore, this function must not be
 * blocked.
 *
 * This function returns DDI_SUCCESS on success, or DDI_FAILURE on failure.
 * DDI_FAILURE indicates an error condition and should almost never happen.
 */
static int
mptsas_quiesce(dev_info_t *devi)
{
        mptsas_t        *mpt;
        scsi_hba_tran_t *tran;

        /*
         * If this call is for iport, just return.
         */
        if (scsi_hba_iport_unit_address(devi))
                return (DDI_SUCCESS);

        if ((tran = ddi_get_driver_private(devi)) == NULL)
                return (DDI_SUCCESS);

        if ((mpt = TRAN2MPT(tran)) == NULL)
                return (DDI_SUCCESS);

        /* Disable HBA interrupts in hardware */
        MPTSAS_DISABLE_INTR(mpt);
        /* Send RAID action system shutdonw to sync IR */
        mptsas_raid_action_system_shutdown(mpt);

        return (DDI_SUCCESS);
}
#endif  /* __sparc */

/*
```



```
 * detach(9E).  Remove all device allocations and system resources;
 * disable device interrupts.
 * Return DDI_SUCCESS if done; DDI_FAILURE if there’s a problem.
 */
static int
mptsas_detach(dev_info_t *devi, ddi_detach_cmd_t cmd)
{
        /* CONSTCOND */
        ASSERT(NO_COMPETING_THREADS);
        NDBG0(("mptsas_detach: dip=0x%p cmd=0x%p", (void *)devi, (void *)cmd));

        switch (cmd) {
        case DDI_DETACH:
                return (mptsas_do_detach(devi));

        case DDI_SUSPEND:
                return (mptsas_suspend(devi));

        default:
                return (DDI_FAILURE);
        }
        /* NOTREACHED */
}

static int
mptsas_do_detach(dev_info_t *dip)
{
        mptsas_t        *mpt;
        scsi_hba_tran_t *tran;
        int             circ = 0;
        int             circ1 = 0;
        mdi_pathinfo_t  *pip = NULL;
        int             i;
        int             q_thread_num = 0;

        NDBG0(("mptsas_do_detach: dip=0x%p", (void *)dip));

        if ((tran = ndi_flavorv_get(dip, SCSA_FLAVOR_SCSI_DEVICE)) == NULL)
                return (DDI_FAILURE);

        mpt = TRAN2MPT(tran);
        if (!mpt) {
                return (DDI_FAILURE);
        }
        /*
         * Still have pathinfo child, should not detach mpt driver
         */
        if (scsi_hba_iport_unit_address(dip)) {
                if (mpt->m_mpxio_enable) {
                        /*
                         * MPxIO enabled for the iport
                         */
                        ndi_devi_enter(scsi_vhci_dip, &circ1);
                        ndi_devi_enter(dip, &circ);
                        while (pip = mdi_get_next_client_path(dip, NULL)) {
                                if (mdi_pi_free(pip, 0) == MDI_SUCCESS) {
                                        continue;
                                }
                                ndi_devi_exit(dip, circ);
                                ndi_devi_exit(scsi_vhci_dip, circ1);
                                NDBG12(("detach failed because of "
                                    "outstanding path info"));
                                return (DDI_FAILURE);
                        }
                        ndi_devi_exit(dip, circ);
                        ndi_devi_exit(scsi_vhci_dip, circ1);
```

```
                        (void) mdi_phci_unregister(dip, 0);
                }

                ddi_prop_remove_all(dip);

                return (DDI_SUCCESS);
        }

        /* Make sure power level is D0 before accessing registers */
        if (mpt->m_options & MPTSAS_OPT_PM) {
                (void) pm_busy_component(dip, 0);
                if (mpt->m_power_level != PM_LEVEL_D0) {
                        if (pm_raise_power(dip, 0, PM_LEVEL_D0) !=
                            DDI_SUCCESS) {
                                mptsas_log(mpt, CE_WARN,
                                    "mptsas3%d: Raise power request failed.",
                                    mpt->m_instance);
                                (void) pm_idle_component(dip, 0);
                                return (DDI_FAILURE);
                        }
                }
        }

        /*
         * Send RAID action system shutdown to sync IR.  After action, send a
         * Message Unit Reset. Since after that DMA resource will be freed,
         * set ioc to READY state will avoid HBA initiated DMA operation.
         */
        mutex_enter(&mpt->m_mutex);
        MPTSAS_DISABLE_INTR(mpt);
        mptsas_raid_action_system_shutdown(mpt);
        mpt->m_softstate |= MPTSAS_SS_MSG_UNIT_RESET;
        (void) mptsas_ioc_reset(mpt, FALSE);
        mutex_exit(&mpt->m_mutex);
        mptsas_rem_intrs(mpt);
        ddi_taskq_destroy(mpt->m_event_taskq);
        ddi_taskq_destroy(mpt->m_dr_taskq);

        if (mpt->m_doneq_thread_n) {
                mutex_enter(&mpt->m_qthread_mutex);
                q_thread_num = mpt->m_doneq_thread_n;
                for (i = 0; i < mpt->m_doneq_thread_n; i++) {
                        mutex_enter(&mpt->m_doneq_thread_id[i].mutex);
                        mpt->m_doneq_thread_id[i].flag &=
                            (~MPTSAS_DONEQ_THREAD_ACTIVE);
                        cv_signal(&mpt->m_doneq_thread_id[i].cv);
                        mutex_exit(&mpt->m_doneq_thread_id[i].mutex);
                }
                while (mpt->m_doneq_thread_n) {
                        cv_wait(&mpt->m_qthread_cv,
                            &mpt->m_qthread_mutex);
                }
                for (i = 0;  i < q_thread_num; i++) {
                        cv_destroy(&mpt->m_doneq_thread_id[i].cv);
                        mutex_destroy(&mpt->m_doneq_thread_id[i].mutex);
                }
                kmem_free(mpt->m_doneq_thread_id,
                    sizeof (mptsas_doneq_thread_list_t)
                    * q_thread_num);
                mutex_exit(&mpt->m_qthread_mutex);
        }
        if (mpt->m_txwq_thread_n) {
                mutex_enter(&mpt->m_qthread_mutex);
                q_thread_num = mpt->m_txwq_thread_n;
                for (i = 0; i < q_thread_num; i++) {
                        mutex_enter(&mpt->m_tx_waitq[i].txwq_mutex);
```



```
                        mpt->m_tx_waitq[i].txwq_active = FALSE;
                        cv_signal(&mpt->m_tx_waitq[i].txwq_cv);
                        mutex_exit(&mpt->m_tx_waitq[i].txwq_mutex);
                }
                while (mpt->m_txwq_thread_n) {
                        cv_wait(&mpt->m_qthread_cv,
                            &mpt->m_qthread_mutex);
                }
                for (i = 0; i < q_thread_num; i++) {
                        cv_destroy(&mpt->m_tx_waitq[i].txwq_cv);
                        cv_destroy(&mpt->m_tx_waitq[i].txwq_drain_cv);
                        mutex_destroy(&mpt->m_tx_waitq[i].txwq_mutex);
                }
        }

        scsi_hba_reset_notify_tear_down(mpt->m_reset_notify_listf);

        mptsas_list_del(mpt);

        /*
         * Cancel timeout threads for this mpt
         */
        mutex_enter(&mpt->m_mutex);
        if (mpt->m_quiesce_timeid) {
                timeout_id_t tid = mpt->m_quiesce_timeid;
                mpt->m_quiesce_timeid = 0;
                mutex_exit(&mpt->m_mutex);
                (void) untimeout(tid);
                mutex_enter(&mpt->m_mutex);
        }

        if (mpt->m_restart_cmd_timeid) {
                timeout_id_t tid = mpt->m_restart_cmd_timeid;
                mpt->m_restart_cmd_timeid = 0;
                mutex_exit(&mpt->m_mutex);
                (void) untimeout(tid);
                mutex_enter(&mpt->m_mutex);
        }

        mutex_exit(&mpt->m_mutex);

        /*
         * last mpt? ... if active, CANCEL watch threads.
         */
        mutex_enter(&mptsas_global_mutex);
        if (mptsas_head == NULL) {
                timeout_id_t tid;
                /*
                 * Clear mptsas_timeouts_enable so that the watch thread
                 * gets restarted on DDI_ATTACH
                 */
                mptsas_timeouts_enabled = 0;
                if (mptsas_timeout_id) {
                        tid = mptsas_timeout_id;
                        mptsas_timeout_id = 0;
                        mutex_exit(&mptsas_global_mutex);
                        (void) untimeout(tid);
                        mutex_enter(&mptsas_global_mutex);
                }
                if (mptsas_reset_watch) {
                        tid = mptsas_reset_watch;
                        mptsas_reset_watch = 0;
                        mutex_exit(&mptsas_global_mutex);
                        (void) untimeout(tid);
                        mutex_enter(&mptsas_global_mutex);
                }
```

```
        }
        mutex_exit(&mptsas_global_mutex);

        /*
         * Delete Phy stats
         */
        mptsas_destroy_phy_stats(mpt);

        mptsas_destroy_hashes(mpt);

        /*
         * Delete nt_active.
         */
        mutex_enter(&mpt->m_mutex);
        mptsas_free_active_slots(mpt);
        mutex_exit(&mpt->m_mutex);

        /* deallocate everything that was allocated in mptsas_attach */
        mptsas_cache_destroy(mpt);

        mptsas_hba_fini(mpt);
        mptsas_cfg_fini(mpt);

        /* Lower the power informing PM Framework */
        if (mpt->m_options & MPTSAS_OPT_PM) {
                if (pm_lower_power(dip, 0, PM_LEVEL_D3) != DDI_SUCCESS)
                        mptsas_log(mpt, CE_WARN,
                            "!mptsas3%d: Lower power request failed "
                            "during detach, ignoring.",
                            mpt->m_instance);
        }

        mutex_destroy(&mpt->m_qthread_mutex);
        mutex_destroy(&mpt->m_passthru_mutex);
        mutex_destroy(&mpt->m_mutex);
        for (i = 0; i < MPTSAS_MAX_PHYS; i++) {
                mutex_destroy(&mpt->m_phy_info[i].smhba_info.phy_mutex);
        }
        cv_destroy(&mpt->m_qthread_cv);
        cv_destroy(&mpt->m_cv);
        cv_destroy(&mpt->m_passthru_cv);
        cv_destroy(&mpt->m_fw_cv);
        cv_destroy(&mpt->m_config_cv);
        cv_destroy(&mpt->m_fw_diag_cv);


        mptsas_smp_teardown(mpt);
        mptsas_hba_teardown(mpt);

        mptsas_config_space_fini(mpt);

        mptsas_free_handshake_msg(mpt);

        mptsas_fm_fini(mpt);
        ddi_soft_state_free(mptsas3_state, ddi_get_instance(dip));
        ddi_prop_remove_all(dip);

        return (DDI_SUCCESS);
}

static void
mptsas_list_add(mptsas_t *mpt)
{
        rw_enter(&mptsas_global_rwlock, RW_WRITER);

        if (mptsas_head == NULL) {
```



```
                mptsas_head = mpt;
        } else {
                mptsas_tail->m_next = mpt;
        }
        mptsas_tail = mpt;
        rw_exit(&mptsas_global_rwlock);
}

static void
mptsas_list_del(mptsas_t *mpt)
{
        mptsas_t *m;
        /*
         * Remove device instance from the global linked list
         */
        rw_enter(&mptsas_global_rwlock, RW_WRITER);
        if (mptsas_head == mpt) {
                m = mptsas_head = mpt->m_next;
        } else {
                for (m = mptsas_head; m != NULL; m = m->m_next) {
                        if (m->m_next == mpt) {
                                m->m_next = mpt->m_next;
                                break;
                        }
                }
                if (m == NULL) {
                        mptsas_log(mpt, CE_PANIC, "Not in softc list!");
                }
        }

        if (mptsas_tail == mpt) {
                mptsas_tail = m;
        }
        rw_exit(&mptsas_global_rwlock);
}

static int
mptsas_alloc_handshake_msg(mptsas_t *mpt, size_t alloc_size)
{
        ddi_dma_attr_t  task_dma_attrs;

        mpt->m_hshk_dma_size = 0;
        task_dma_attrs = mpt->m_msg_dma_attr;
        task_dma_attrs.dma_attr_sgllen = 1;
        task_dma_attrs.dma_attr_granular = (uint32_t)(alloc_size);

        /* allocate Task Management ddi_dma resources */
        if (mptsas_dma_addr_create(mpt, task_dma_attrs,
            &mpt->m_hshk_dma_hdl, &mpt->m_hshk_acc_hdl, &mpt->m_hshk_memp,
            alloc_size, NULL) == FALSE) {
                return (DDI_FAILURE);
        }
        mpt->m_hshk_dma_size = alloc_size;

        return (DDI_SUCCESS);
}

static void
mptsas_free_handshake_msg(mptsas_t *mpt)
{
        if (mpt->m_hshk_dma_size == 0)
                return;
        mptsas_dma_addr_destroy(&mpt->m_hshk_dma_hdl, &mpt->m_hshk_acc_hdl);
        mpt->m_hshk_dma_size = 0;
}
```

```
static int
mptsas_hba_setup(mptsas_t *mpt)
{
        scsi_hba_tran_t         *hba_tran;
        int                     tran_flags;

        /* Allocate a transport structure */
        hba_tran = mpt->m_tran = scsi_hba_tran_alloc(mpt->m_dip,
            SCSI_HBA_CANSLEEP);
        ASSERT(mpt->m_tran != NULL);

        hba_tran->tran_hba_private      = mpt;
        hba_tran->tran_tgt_private      = NULL;

        hba_tran->tran_tgt_init         = mptsas_scsi_tgt_init;
        hba_tran->tran_tgt_free         = mptsas_scsi_tgt_free;

        hba_tran->tran_start            = mptsas_scsi_start;
        hba_tran->tran_reset            = mptsas_scsi_reset;
        hba_tran->tran_abort            = mptsas_scsi_abort;
        hba_tran->tran_getcap           = mptsas_scsi_getcap;
        hba_tran->tran_setcap           = mptsas_scsi_setcap;
        hba_tran->tran_init_pkt         = mptsas_scsi_init_pkt;
        hba_tran->tran_destroy_pkt      = mptsas_scsi_destroy_pkt;

        hba_tran->tran_dmafree          = mptsas_scsi_dmafree;
        hba_tran->tran_sync_pkt         = mptsas_scsi_sync_pkt;
        hba_tran->tran_reset_notify     = mptsas_scsi_reset_notify;

        hba_tran->tran_get_bus_addr     = mptsas_get_bus_addr;
        hba_tran->tran_get_name         = mptsas_get_name;

        hba_tran->tran_quiesce          = mptsas_scsi_quiesce;
        hba_tran->tran_unquiesce        = mptsas_scsi_unquiesce;
        hba_tran->tran_bus_reset        = NULL;

        hba_tran->tran_add_eventcall    = NULL;
        hba_tran->tran_get_eventcookie  = NULL;
        hba_tran->tran_post_event       = NULL;
        hba_tran->tran_remove_eventcall = NULL;

        hba_tran->tran_bus_config       = mptsas_bus_config;

        hba_tran->tran_interconnect_type = INTERCONNECT_SAS;

        /*
         * All children of the HBA are iports. We need tran was cloned.
         * So we pass the flags to SCSA. SCSI_HBA_TRAN_CLONE will be
         * inherited to iport's tran vector.
         */
        tran_flags = (SCSI_HBA_HBA | SCSI_HBA_TRAN_CLONE);

        if (scsi_hba_attach_setup(mpt->m_dip, &mpt->m_msg_dma_attr,
            hba_tran, tran_flags) != DDI_SUCCESS) {
                mptsas_log(mpt, CE_WARN, "hba attach setup failed");
                scsi_hba_tran_free(hba_tran);
                mpt->m_tran = NULL;
                return (FALSE);
        }
        return (TRUE);
}

static void
mptsas_hba_teardown(mptsas_t *mpt)
{
        (void) scsi_hba_detach(mpt->m_dip);
```



```
        if (mpt->m_tran != NULL) {
                scsi_hba_tran_free(mpt->m_tran);
                mpt->m_tran = NULL;
        }
}

static void
mptsas_iport_register(mptsas_t *mpt)
{
        int i, j;
        mptsas_phymask_t        mask = 0x0;
        /*
         * initial value of mask is 0
         */
        mutex_enter(&mpt->m_mutex);
        for (i = 0; i < mpt->m_num_phys; i++) {
                mptsas_phymask_t phy_mask = 0x0;
                char phy_mask_name[MPTSAS_MAX_PHYS];
                uint8_t current_port;

                if (mpt->m_phy_info[i].attached_devhdl == 0)
                        continue;

                bzero(phy_mask_name, sizeof (phy_mask_name));

                current_port = mpt->m_phy_info[i].port_num;

                if ((mask & (1 << i)) != 0)
                        continue;

                for (j = 0; j < mpt->m_num_phys; j++) {
                        if (mpt->m_phy_info[j].attached_devhdl &&
                            (mpt->m_phy_info[j].port_num == current_port)) {
                                phy_mask |= (1 << j);
                        }
                }
                mask = mask | phy_mask;

                for (j = 0; j < mpt->m_num_phys; j++) {
                        if ((phy_mask >> j) & 0x01) {
                                mpt->m_phy_info[j].phy_mask = phy_mask;
                        }
                }

                (void) sprintf(phy_mask_name, "%x", phy_mask);

                mutex_exit(&mpt->m_mutex);
                /*
                 * register a iport
                 */
                (void) scsi_hba_iport_register(mpt->m_dip, phy_mask_name);
                mutex_enter(&mpt->m_mutex);
        }
        mutex_exit(&mpt->m_mutex);
        /*
         * register a virtual port for RAID volume always
         */
        (void) scsi_hba_iport_register(mpt->m_dip, "v0");

}

static int
mptsas_smp_setup(mptsas_t *mpt)
{
        mpt->m_smptran = smp_hba_tran_alloc(mpt->m_dip);
        ASSERT(mpt->m_smptran != NULL);
```

```
        mpt->m_smptran->smp_tran_hba_private = mpt;
        mpt->m_smptran->smp_tran_start = mptsas_smp_start;
        if (smp_hba_attach_setup(mpt->m_dip, mpt->m_smptran) != DDI_SUCCESS) {
                mptsas_log(mpt, CE_WARN, "smp attach setup failed");
                smp_hba_tran_free(mpt->m_smptran);
                mpt->m_smptran = NULL;
                return (FALSE);
        }
        /*
         * Initialize smp hash table
         */
        mpt->m_smp_targets = refhash_create(MPTSAS_SMP_BUCKET_COUNT,
            mptsas_target_addr_hash, mptsas_target_addr_cmp,
            mptsas_smp_free, sizeof (mptsas_smp_t),
            offsetof(mptsas_smp_t, m_link), offsetof(mptsas_smp_t, m_addr),
            KM_SLEEP);
        mpt->m_smp_devhdl = 0xFFFF;

        return (TRUE);
}

static void
mptsas_smp_teardown(mptsas_t *mpt)
{
        (void) smp_hba_detach(mpt->m_dip);
        if (mpt->m_smptran != NULL) {
                smp_hba_tran_free(mpt->m_smptran);
                mpt->m_smptran = NULL;
        }
        mpt->m_smp_devhdl = 0;
}

static int
mptsas_cache_create(mptsas_t *mpt)
{
        int instance = mpt->m_instance;
        char buf[64];

        /*
         * create kmem cache for packets
         */
        (void) sprintf(buf, "mptsas3%d_cache", instance);
        mpt->m_kmem_cache = kmem_cache_create(buf,
            sizeof (struct mptsas_cmd) + scsi_pkt_size(), 16,
            mptsas_kmem_cache_constructor, mptsas_kmem_cache_destructor,
            NULL, (void *)mpt, NULL, 0);

        if (mpt->m_kmem_cache == NULL) {
                mptsas_log(mpt, CE_WARN, "creating kmem cache failed");
                return (FALSE);
        }

        /*
         * create kmem cache for extra SGL frames if SGL cannot
         * be accomodated into main request frame.
         */
        (void) sprintf(buf, "mptsas3%d_cache_frames", instance);
        mpt->m_cache_frames = kmem_cache_create(buf,
            sizeof (mptsas_cache_frames_t), 16,
            mptsas_cache_frames_constructor, mptsas_cache_frames_destructor,
            NULL, (void *)mpt, NULL, 0);

        if (mpt->m_cache_frames == NULL) {
                mptsas_log(mpt, CE_WARN, "creating cache for frames failed");
                return (FALSE);
        }
```



```
        return (TRUE);
}

static void
mptsas_cache_destroy(mptsas_t *mpt)
{
        /* deallocate in reverse order */
        if (mpt->m_cache_frames) {
                kmem_cache_destroy(mpt->m_cache_frames);
                mpt->m_cache_frames = NULL;
        }
        if (mpt->m_kmem_cache) {
                kmem_cache_destroy(mpt->m_kmem_cache);
                mpt->m_kmem_cache = NULL;
        }
}

static int
mptsas_power(dev_info_t *dip, int component, int level)
{
#ifndef __lock_lint
        _NOTE(ARGUNUSED(component))
#endif
        mptsas_t        *mpt;
        int             rval = DDI_SUCCESS;
        int             polls = 0;
        uint32_t        ioc_status;

        if (scsi_hba_iport_unit_address(dip) != 0)
                return (DDI_SUCCESS);

        mpt = ddi_get_soft_state(mptsas3_state, ddi_get_instance(dip));
        if (mpt == NULL) {
                return (DDI_FAILURE);
        }

        mutex_enter(&mpt->m_mutex);

        /*
         * If the device is busy, don’t lower its power level
         */
        if (mpt->m_busy && (mpt->m_power_level > level)) {
                mutex_exit(&mpt->m_mutex);
                return (DDI_FAILURE);
        }
        switch (level) {
        case PM_LEVEL_D0:
                NDBG11(("mptsas3%d: turning power ON.", mpt->m_instance));
                MPTSAS_POWER_ON(mpt);
                /*
                 * Wait up to 30 seconds for IOC to come out of reset.
                 */
                while (((ioc_status = ddi_get32(mpt->m_datap,
                    &mpt->m_reg->Doorbell)) &
                    MPI2_IOC_STATE_MASK) == MPI2_IOC_STATE_RESET) {
                        if (polls++ > 3000) {
                                break;
                        }
                        delay(drv_usectohz(10000));
                }
                /*
                 * If IOC is not in operational state, try to hard reset it.
                 */
                if ((ioc_status & MPI2_IOC_STATE_MASK) !=
                    MPI2_IOC_STATE_OPERATIONAL) {
```

```
                        mpt->m_softstate &= ~MPTSAS_SS_MSG_UNIT_RESET;
                        if (mptsas_restart_ioc(mpt) == DDI_FAILURE) {
                                mptsas_log(mpt, CE_WARN,
                                    "mptsas_power: hard reset failed");
                                mutex_exit(&mpt->m_mutex);
                                return (DDI_FAILURE);
                        }
                }
                mpt->m_power_level = PM_LEVEL_D0;
                break;
        case PM_LEVEL_D3:
                NDBG11(("mptsas3%d: turning power OFF.", mpt->m_instance));
                MPTSAS_POWER_OFF(mpt);
                break;
        default:
                mptsas_log(mpt, CE_WARN, "mptsas3%d: unknown power level <%x>.",
                    mpt->m_instance, level);
                rval = DDI_FAILURE;
                break;
        }
        mutex_exit(&mpt->m_mutex);
        return (rval);
}

/*
 * Initialize configuration space and figure out which
 * chip and revison of the chip the mpt driver is using.
 */
static int
mptsas_config_space_init(mptsas_t *mpt)
{
        NDBG0(("mptsas_config_space_init"));

        if (mpt->m_config_handle != NULL)
                return (TRUE);

        if (pci_config_setup(mpt->m_dip,
            &mpt->m_config_handle) != DDI_SUCCESS) {
                mptsas_log(mpt, CE_WARN, "cannot map configuration space.");
                return (FALSE);
        }

        /*
         * This is a workaround for a XMITS ASIC bug which does not
         * drive the CBE upper bits.
         */
        if (pci_config_get16(mpt->m_config_handle, PCI_CONF_STAT) &
            PCI_STAT_PERROR) {
                pci_config_put16(mpt->m_config_handle, PCI_CONF_STAT,
                    PCI_STAT_PERROR);
        }

        mptsas_setup_cmd_reg(mpt);

        /*
         * Get the chip device id:
         */
        mpt->m_devid = pci_config_get16(mpt->m_config_handle, PCI_CONF_DEVID);

        /*
         * Save the revision.
         */
        mpt->m_revid = pci_config_get8(mpt->m_config_handle, PCI_CONF_REVID);

        /*
         * Save the SubSystem Vendor and Device IDs
```



```
         */
        mpt->m_svid = pci_config_get16(mpt->m_config_handle, PCI_CONF_SUBVENID);
        mpt->m_ssid = pci_config_get16(mpt->m_config_handle, PCI_CONF_SUBSYSID);

        /*
         * Set the latency timer to 0x40 as specified by the upa -> pci
         * bridge chip design team.  This may be done by the sparc pci
         * bus nexus driver, but the driver should make sure the latency
         * timer is correct for performance reasons.
         */
        pci_config_put8(mpt->m_config_handle, PCI_CONF_LATENCY_TIMER,
            MPTSAS_LATENCY_TIMER);

        (void) mptsas_get_pci_cap(mpt);
        return (TRUE);
}

static void
mptsas_config_space_fini(mptsas_t *mpt)
{
        if (mpt->m_config_handle != NULL) {
                mptsas_disable_bus_master(mpt);
                pci_config_teardown(&mpt->m_config_handle);
                mpt->m_config_handle = NULL;
        }
}

static void
mptsas_setup_cmd_reg(mptsas_t *mpt)
{
        ushort_t        cmdreg;

        /*
         * Set the command register to the needed values.
         */
        cmdreg = pci_config_get16(mpt->m_config_handle, PCI_CONF_COMM);
        cmdreg |= (PCI_COMM_ME | PCI_COMM_SERR_ENABLE |
            PCI_COMM_PARITY_DETECT | PCI_COMM_MAE);
        cmdreg &= ~PCI_COMM_IO;
        pci_config_put16(mpt->m_config_handle, PCI_CONF_COMM, cmdreg);
}

static void
mptsas_disable_bus_master(mptsas_t *mpt)
{
        ushort_t        cmdreg;

        /*
         * Clear the master enable bit in the PCI command register.
         * This prevents any bus mastering activity like DMA.
         */
        cmdreg = pci_config_get16(mpt->m_config_handle, PCI_CONF_COMM);
        cmdreg &= ~PCI_COMM_ME;
        pci_config_put16(mpt->m_config_handle, PCI_CONF_COMM, cmdreg);
}

int
mptsas_dma_alloc(mptsas_t *mpt, mptsas_dma_alloc_state_t *dma_statep)
{
        ddi_dma_attr_t  attrs;

        attrs = mpt->m_io_dma_attr;
        attrs.dma_attr_sgllen = 1;

        ASSERT(dma_statep != NULL);
```

```
        if (mptsas_dma_addr_create(mpt, attrs, &dma_statep->handle,
            &dma_statep->accessp, &dma_statep->memp, dma_statep->size,
            &dma_statep->cookie) == FALSE) {
                return (DDI_FAILURE);
        }

        return (DDI_SUCCESS);
}

void
mptsas_dma_free(mptsas_dma_alloc_state_t *dma_statep)
{
        ASSERT(dma_statep != NULL);
        mptsas_dma_addr_destroy(&dma_statep->handle, &dma_statep->accessp);
        dma_statep->size = 0;
}

int
mptsas_do_dma(mptsas_t *mpt, uint32_t size, int var, int (*callback)())
{
        ddi_dma_attr_t          attrs;
        ddi_dma_handle_t        dma_handle;
        caddr_t                 memp;
        ddi_acc_handle_t        accessp;
        int                     rval;

        ASSERT(mutex_owned(&mpt->m_mutex));

        attrs = mpt->m_msg_dma_attr;
        attrs.dma_attr_sgllen = 1;
        attrs.dma_attr_granular = size;

        if (mptsas_dma_addr_create(mpt, attrs, &dma_handle,
            &accessp, &memp, size, NULL) == FALSE) {
                return (DDI_FAILURE);
        }

        rval = (*callback) (mpt, memp, var, accessp);

        if ((mptsas_check_dma_handle(dma_handle) != DDI_SUCCESS) ||
            (mptsas_check_acc_handle(accessp) != DDI_SUCCESS)) {
                ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_UNAFFECTED);
                rval = DDI_FAILURE;
        }

        mptsas_dma_addr_destroy(&dma_handle, &accessp);
        return (rval);

}

static int
mptsas_alloc_request_frames(mptsas_t *mpt)
{
        ddi_dma_attr_t          frame_dma_attrs;
        caddr_t                 memp;
        ddi_dma_cookie_t        cookie;
        size_t                  mem_size;

        /*
         * re-alloc when it has already alloced
         */
        if (mpt->m_dma_flags & MPTSAS_REQ_FRAME) {
                mptsas_dma_addr_destroy(&mpt->m_dma_req_frame_hdl,
                    &mpt->m_acc_req_frame_hdl);
                mpt->m_dma_flags &= ~MPTSAS_REQ_FRAME;
        }
```



```
        /*
         * The size of the request frame pool is:
         *   Number of Request Frames * Request Frame Size
         */
        mem_size = mpt->m_max_requests * mpt->m_req_frame_size;

        /*
         * set the DMA attributes.  System Request Message Frames must be
         * aligned on a 16-byte boundry.
         */
        frame_dma_attrs = mpt->m_msg_dma_attr;
        frame_dma_attrs.dma_attr_align = 16;
        frame_dma_attrs.dma_attr_sgllen = 1;

        /*
         * allocate the request frame pool.
         */
        if (mptsas_dma_addr_create(mpt, frame_dma_attrs,
            &mpt->m_dma_req_frame_hdl, &mpt->m_acc_req_frame_hdl, &memp,
            mem_size, &cookie) == FALSE) {
                return (DDI_FAILURE);
        }

        /*
         * Store the request frame memory address.  This chip uses this
         * address to dma to and from the driver's frame.  The second
         * address is the address mpt uses to fill in the frame.
         */
        mpt->m_req_frame_dma_addr = cookie.dmac_laddress;
        mpt->m_req_frame = memp;

        /*
         * Clear the request frame pool.
         */
        bzero(mpt->m_req_frame, mem_size);

        mpt->m_dma_flags |= MPTSAS_REQ_FRAME;
        return (DDI_SUCCESS);
}

static int
mptsas_alloc_sense_bufs(mptsas_t *mpt)
{
        ddi_dma_attr_t          sense_dma_attrs;
        caddr_t                 memp;
        ddi_dma_cookie_t        cookie;
        size_t                  mem_size;
        int                     num_extrqsense_bufs;

        /*
         * re-alloc when it has already alloced
         */
        if (mpt->m_dma_flags & MPTSAS_REQ_SENSE) {
                rmfreemap(mpt->m_erqsense_map);
                mptsas_dma_addr_destroy(&mpt->m_dma_req_sense_hdl,
                    &mpt->m_acc_req_sense_hdl);
                mpt->m_dma_flags &= ~MPTSAS_REQ_SENSE;
        }

        /*
         * The size of the request sense pool is:
         *   (Number of Request Frames - 2 ) * Request Sense Size +
         *   extra memory for extended sense requests.
         */
        mem_size = ((mpt->m_max_requests - 2) * mpt->m_req_sense_size) +
```

```
            mptsas_extreq_sense_bufsize;

        /*
         * set the DMA attributes.  ARQ buffers
         * aligned on a 16-byte boundry.
         */
        sense_dma_attrs = mpt->m_msg_dma_attr;
        sense_dma_attrs.dma_attr_align = 16;
        sense_dma_attrs.dma_attr_sgllen = 1;

        /*
         * allocate the request sense buffer pool.
         */
        if (mptsas_dma_addr_create(mpt, sense_dma_attrs,
            &mpt->m_dma_req_sense_hdl, &mpt->m_acc_req_sense_hdl, &memp,
            mem_size, &cookie) == FALSE) {
                return (DDI_FAILURE);
        }

        /*
         * Store the request sense base memory address.  This chip uses this
         * address to dma the request sense data.  The second
         * address is the address mpt uses to access the data.
         * The third is the base for the extended rqsense buffers.
         */
        mpt->m_req_sense_dma_addr = cookie.dmac_laddress;
        mpt->m_req_sense = memp;
        memp += (mpt->m_max_requests - 2) * mpt->m_req_sense_size;
        mpt->m_extreq_sense = memp;

        /*
         * The extra memory is divided up into multiples of the base
         * buffer size in order to allocate via rmalloc().
         * Note that the rmallocmap cannot start at zero!
         */
        num_extrqsense_bufs = mptsas_extreq_sense_bufsize /
            mpt->m_req_sense_size;
        mpt->m_erqsense_map = rmallocmap_wait(num_extrqsense_bufs);
        rmfree(mpt->m_erqsense_map, num_extrqsense_bufs, 1);

        /*
         * Clear the pool.
         */
        bzero(mpt->m_req_sense, mem_size);

        mpt->m_dma_flags |= MPTSAS_REQ_SENSE;
        return (DDI_SUCCESS);
}

static int
mptsas_alloc_reply_frames(mptsas_t *mpt)
{
        ddi_dma_attr_t          frame_dma_attrs;
        caddr_t                 memp;
        ddi_dma_cookie_t        cookie;
        size_t                  mem_size;

        /*
         * re-alloc when it has already alloced
         */
        if (mpt->m_dma_flags & MPTSAS_REPLY_FRAME) {
                mptsas_dma_addr_destroy(&mpt->m_dma_reply_frame_hdl,
                    &mpt->m_acc_reply_frame_hdl);
                mpt->m_dma_flags &= ~MPTSAS_REPLY_FRAME;
        }
```



```
        /*
         * The size of the reply frame pool is:
         *   Number of Reply Frames * Reply Frame Size
         */
        mem_size = mpt->m_max_replies * mpt->m_reply_frame_size;

        /*
         * set the DMA attributes.   System Reply Message Frames must be
         * aligned on a 4-byte boundry.  This is the default.
         */
        frame_dma_attrs = mpt->m_msg_dma_attr;
        frame_dma_attrs.dma_attr_sgllen = 1;

        /*
         * allocate the reply frame pool
         */
        if (mptsas_dma_addr_create(mpt, frame_dma_attrs,
            &mpt->m_dma_reply_frame_hdl, &mpt->m_acc_reply_frame_hdl, &memp,
            mem_size, &cookie) == FALSE) {
                return (DDI_FAILURE);
        }

        /*
         * Store the reply frame memory address.  This chip uses this
         * address to dma to and from the driver's frame.  The second
         * address is the address mpt uses to process the frame.
         */
        mpt->m_reply_frame_dma_addr = cookie.dmac_laddress;
        mpt->m_reply_frame = memp;

        /*
         * Clear the reply frame pool.
         */
        bzero(mpt->m_reply_frame, mem_size);

        mpt->m_dma_flags |= MPTSAS_REPLY_FRAME;
        return (DDI_SUCCESS);
}

static int
mptsas_alloc_free_queue(mptsas_t *mpt)
{
        ddi_dma_attr_t          frame_dma_attrs;
        caddr_t                 memp;
        ddi_dma_cookie_t        cookie;
        size_t                  mem_size;

        /*
         * re-alloc when it has already alloced
         */
        if (mpt->m_dma_flags & MPTSAS_FREE_QUEUE) {
                mptsas_dma_addr_destroy(&mpt->m_dma_free_queue_hdl,
                    &mpt->m_acc_free_queue_hdl);
                mpt->m_dma_flags &= ~MPTSAS_FREE_QUEUE;
        }

        /*
         * The reply free queue size is:
         *   Reply Free Queue Depth * 4
         * The "4" is the size of one 32 bit address (low part of 64-bit
         *   address)
         */
        mem_size = mpt->m_free_queue_depth * 4;

        /*
         * set the DMA attributes  The Reply Free Queue must be aligned on a
```

```
         * 16-byte boundry.
         */
        frame_dma_attrs = mpt->m_msg_dma_attr;
        frame_dma_attrs.dma_attr_align = 16;
        frame_dma_attrs.dma_attr_sgllen = 1;

        /*
         * allocate the reply free queue
         */
        if (mptsas_dma_addr_create(mpt, frame_dma_attrs,
            &mpt->m_dma_free_queue_hdl, &mpt->m_acc_free_queue_hdl, &memp,
            mem_size, &cookie) == FALSE) {
                return (DDI_FAILURE);
        }

        /*
         * Store the reply free queue memory address.  This chip uses this
         * address to read from the reply free queue.  The second address
         * is the address mpt uses to manage the queue.
         */
        mpt->m_free_queue_dma_addr = cookie.dmac_laddress;
        mpt->m_free_queue = memp;

        /*
         * Clear the reply free queue memory.
         */
        bzero(mpt->m_free_queue, mem_size);

        mpt->m_dma_flags |= MPTSAS_FREE_QUEUE;
        return (DDI_SUCCESS);
}

static void
mptsas_free_post_queue(mptsas_t *mpt)
{
        mptsas_reply_pqueue_t   *rpqp;
        int                     i;

        if (mpt->m_dma_flags & MPTSAS_POST_QUEUE) {
                mptsas_dma_addr_destroy(&mpt->m_dma_post_queue_hdl,
                    &mpt->m_acc_post_queue_hdl);
                rpqp = mpt->m_rep_post_queues;
                for (i = 0; i < mpt->m_post_reply_qcount; i++) {
                        mutex_destroy(&rpqp->rpq_mutex);
                        rpqp++;
                }
                kmem_free(mpt->m_rep_post_queues,
                    sizeof (mptsas_reply_pqueue_t) *
                    mpt->m_post_reply_qcount);
                mpt->m_dma_flags &= ~MPTSAS_POST_QUEUE;
        }
}

static int
mptsas_alloc_post_queue(mptsas_t *mpt)
{
        ddi_dma_attr_t          frame_dma_attrs;
        caddr_t                 memp;
        ddi_dma_cookie_t        cookie;
        size_t                  mem_size;
        mptsas_reply_pqueue_t   *rpqp;
        int                     i;

        /*
         * re-alloc when it has already alloced
         */
```



```
        mptsas_free_post_queue(mpt);

        /*
         * The reply descriptor post queue size is:
         *   Reply Descriptor Post Queue Depth * 8
         * The "8" is the size of each descriptor (8 bytes or 64 bits).
         */
        mpt->m_post_reply_qcount = mpt->m_intr_cnt;
        mem_size = mpt->m_post_queue_depth * 8 * mpt->m_post_reply_qcount;

        /*
         * set the DMA attributes.  The Reply Descriptor Post Queue must be
         * aligned on a 16-byte boundry.
         */
        frame_dma_attrs = mpt->m_msg_dma_attr;
        frame_dma_attrs.dma_attr_align = 16;
        frame_dma_attrs.dma_attr_sgllen = 1;

        /*
         * Allocate the reply post queue(s).
         * MPI2.5 introduces a method to allocate multiple queues
         * using a redirect table. For now stick to one contiguous
         * chunck. This can get as big as 1Mbyte for 16 queues.
         * The spec gives no indication that the queue size can be
         * reduced if you have many of them.
         */
        if (mptsas_dma_addr_create(mpt, frame_dma_attrs,
            &mpt->m_dma_post_queue_hdl, &mpt->m_acc_post_queue_hdl, &memp,
            mem_size, &cookie) == FALSE) {
                return (DDI_FAILURE);
        }

        /*
         * Store the reply descriptor post queue memory address.  This chip
         * uses this address to write to the reply descriptor post queue.  The
         * second address is the address mpt uses to manage the queue.
         */
        mpt->m_post_queue_dma_addr = cookie.dmac_laddress;
        mpt->m_post_queue = memp;

        mpt->m_rep_post_queues = kmem_zalloc(sizeof (mptsas_reply_pqueue_t) *
            mpt->m_post_reply_qcount, KM_SLEEP);
        rpqp = mpt->m_rep_post_queues;
        for (i = 0; i < mpt->m_post_reply_qcount; i++) {
                rpqp->rpq_queue = memp;
                mutex_init(&rpqp->rpq_mutex, NULL, MUTEX_DRIVER, NULL);
                rpqp->rpq_dlist.dl_tail = &rpqp->rpq_dlist.dl_q;
                rpqp->rpq_num = (uint8_t)i;
                memp += (mpt->m_post_queue_depth * 8);
                rpqp++;
        }

        /*
         * Clear the reply post queue memory.
         */
        bzero(mpt->m_post_queue, mem_size);

        mpt->m_dma_flags |= MPTSAS_POST_QUEUE;
        return (DDI_SUCCESS);
}

static void
mptsas_alloc_reply_args(mptsas_t *mpt)
{
        if (mpt->m_replyh_args == NULL) {
                mpt->m_replyh_args = kmem_zalloc(sizeof (m_replyh_arg_t) *
```

```
                    mpt->m_max_replies, KM_SLEEP);
        } else {
                bzero(mpt->m_replyh_args, sizeof (m_replyh_arg_t) *
                    mpt->m_max_replies);
        }
}

static int
mptsas_alloc_extra_sgl_frame(mptsas_t *mpt, mptsas_cmd_t *cmd)
{
        mptsas_cache_frames_t   *frames = NULL;
        if (cmd->cmd_extra_frames == NULL) {
                frames = kmem_cache_alloc(mpt->m_cache_frames, KM_NOSLEEP);
                if (frames == NULL) {
                        return (DDI_FAILURE);
                }
                cmd->cmd_extra_frames = frames;
        }
        return (DDI_SUCCESS);
}

static void
mptsas_free_extra_sgl_frame(mptsas_t *mpt, mptsas_cmd_t *cmd)
{
        if (cmd->cmd_extra_frames) {
                kmem_cache_free(mpt->m_cache_frames,
                    (void *)cmd->cmd_extra_frames);
                cmd->cmd_extra_frames = NULL;
        }
}

static void
mptsas_cfg_fini(mptsas_t *mpt)
{
        NDBG0(("mptsas_cfg_fini"));
        ddi_regs_map_free(&mpt->m_datap);
}

static void
mptsas_hba_fini(mptsas_t *mpt)
{
        NDBG0(("mptsas_hba_fini"));

        /*
         * Free up any allocated memory
         */
        if (mpt->m_dma_flags & MPTSAS_REQ_FRAME) {
                mptsas_dma_addr_destroy(&mpt->m_dma_req_frame_hdl,
                    &mpt->m_acc_req_frame_hdl);
        }

        if (mpt->m_dma_flags & MPTSAS_REQ_SENSE) {
                rmfreemap(mpt->m_erqsense_map);
                mptsas_dma_addr_destroy(&mpt->m_dma_req_sense_hdl,
                    &mpt->m_acc_req_sense_hdl);
        }

        if (mpt->m_dma_flags & MPTSAS_REPLY_FRAME) {
                mptsas_dma_addr_destroy(&mpt->m_dma_reply_frame_hdl,
                    &mpt->m_acc_reply_frame_hdl);
        }

        if (mpt->m_dma_flags & MPTSAS_FREE_QUEUE) {
                mptsas_dma_addr_destroy(&mpt->m_dma_free_queue_hdl,
                    &mpt->m_acc_free_queue_hdl);
        }
```



```
        mptsas_free_post_queue(mpt);

        if (mpt->m_replyh_args != NULL) {
                kmem_free(mpt->m_replyh_args, sizeof (m_replyh_arg_t)
                    * mpt->m_max_replies);
        }
}

static int
mptsas_name_child(dev_info_t *lun_dip, char *name, int len)
{
        int             lun = 0;
        char            *sas_wwn = NULL;
        int             phynum = -1;
        int             reallen = 0;

        /* Get the target num */
        lun = ddi_prop_get_int(DDI_DEV_T_ANY, lun_dip, DDI_PROP_DONTPASS,
            LUN_PROP, 0);

        if ((phynum = ddi_prop_get_int(DDI_DEV_T_ANY, lun_dip,
            DDI_PROP_DONTPASS, "sata-phy", -1)) != -1) {
                /*
                 * Stick in the address of form "pPHY,LUN"
                 */
                reallen = snprintf(name, len, "p%x,%x", phynum, lun);
        } else if (ddi_prop_lookup_string(DDI_DEV_T_ANY, lun_dip,
            DDI_PROP_DONTPASS, SCSI_ADDR_PROP_TARGET_PORT, &sas_wwn)
            == DDI_PROP_SUCCESS) {
                /*
                 * Stick in the address of the form "wWWN,LUN"
                 */
                reallen = snprintf(name, len, "%s,%x", sas_wwn, lun);
                ddi_prop_free(sas_wwn);
        } else {
                return (DDI_FAILURE);
        }

        ASSERT(reallen < len);
        if (reallen >= len) {
                mptsas_log(0, CE_WARN, "!mptsas_get_name: name parameter "
                    "length too small, it needs to be %d bytes", reallen + 1);
        }
        return (DDI_SUCCESS);
}

/*
 * tran_tgt_init(9E) - target device instance initialization
 */
static int
mptsas_scsi_tgt_init(dev_info_t *hba_dip, dev_info_t *tgt_dip,
    scsi_hba_tran_t *hba_tran, struct scsi_device *sd)
{
#ifndef __lock_lint
        _NOTE(ARGUNUSED(hba_tran))
#endif

        /*
         * At this point, the scsi_device structure already exists
         * and has been initialized.
         *
         * Use this function to allocate target-private data structures,
         * if needed by this HBA.  Add revised flow-control and queue
         * properties for child here, if desired and if you can tell they
         * support tagged queueing by now.
```

```
 */
mptsas_t                *mpt;
int                     lun = sd->sd_address.a_lun;
mdi_pathinfo_t          *pip = NULL;
mptsas_tgt_private_t    *tgt_private = NULL;
mptsas_target_t         *ptgt = NULL;
char                    *psas_wwn = NULL;
mptsas_phymask_t        phymask = 0;
uint64_t                sas_wwn = 0;
mptsas_target_addr_t    addr;
mpt = SDEV2MPT(sd);

ASSERT(scsi_hba_iport_unit_address(hba_dip) != 0);

NDBG0(("mptsas_scsi_tgt_init: hbadip=0x%p tgtdip=0x%p lun=%d",
    (void *)hba_dip, (void *)tgt_dip, lun));

if (ndi_dev_is_persistent_node(tgt_dip) == 0) {
        (void) ndi_merge_node(tgt_dip, mptsas_name_child);
        ddi_set_name_addr(tgt_dip, NULL);
        return (DDI_FAILURE);
}
/*
 * phymask is 0 means the virtual port for RAID
 */
phymask = (mptsas_phymask_t)ddi_prop_get_int(DDI_DEV_T_ANY, hba_dip, 0,
    "phymask", 0);
if (mdi_component_is_client(tgt_dip, NULL) == MDI_SUCCESS) {
        if ((pip = (void *)(sd->sd_private)) == NULL) {
                /*
                 * Very bad news if this occurs. Somehow scsi_vhci has
                 * lost the pathinfo node for this target.
                 */
                return (DDI_NOT_WELL_FORMED);
        }

        if (mdi_prop_lookup_int(pip, LUN_PROP, &lun) !=
            DDI_PROP_SUCCESS) {
                mptsas_log(mpt, CE_WARN, "Get lun property failed\n");
                return (DDI_FAILURE);
        }

        if (mdi_prop_lookup_string(pip, SCSI_ADDR_PROP_TARGET_PORT,
            &psas_wwn) == MDI_SUCCESS) {
                if (scsi_wwnstr_to_wwn(psas_wwn, &sas_wwn)) {
                        sas_wwn = 0;
                }
                (void) mdi_prop_free(psas_wwn);
        }
} else {
        lun = ddi_prop_get_int(DDI_DEV_T_ANY, tgt_dip,
            DDI_PROP_DONTPASS, LUN_PROP, 0);
        if (ddi_prop_lookup_string(DDI_DEV_T_ANY, tgt_dip,
            DDI_PROP_DONTPASS, SCSI_ADDR_PROP_TARGET_PORT, &psas_wwn) ==
            DDI_PROP_SUCCESS) {
                if (scsi_wwnstr_to_wwn(psas_wwn, &sas_wwn)) {
                        sas_wwn = 0;
                }
                ddi_prop_free(psas_wwn);
        } else {
                sas_wwn = 0;
        }
}

ASSERT((sas_wwn != 0) || (phymask != 0));
addr.mta_wwn = sas_wwn;
```



```
addr.mta_phymask = phymask;
mutex_enter(&mpt->m_mutex);
ptgt = refhash_lookup(mpt->m_targets, &addr);
mutex_exit(&mpt->m_mutex);
if (ptgt == NULL) {
        mptsas_log(mpt, CE_WARN, "!tgt_init: target doesn't exist or "
            "gone already! phymask:%x, saswwn %"PRIx64, phymask,
            sas_wwn);
        return (DDI_FAILURE);
}
if (hba_tran->tran_tgt_private == NULL) {
        tgt_private = kmem_zalloc(sizeof (mptsas_tgt_private_t),
            KM_SLEEP);
        tgt_private->t_lun = lun;
        tgt_private->t_private = ptgt;
        hba_tran->tran_tgt_private = tgt_private;
}

if (mdi_component_is_client(tgt_dip, NULL) == MDI_SUCCESS) {
        return (DDI_SUCCESS);
}
mutex_enter(&mpt->m_mutex);

if (ptgt->m_deviceinfo &
    (MPI2_SAS_DEVICE_INFO_SATA_DEVICE |
    MPI2_SAS_DEVICE_INFO_ATAPI_DEVICE)) {
        uchar_t *inq89 = NULL;
        int inq89_len = 0x238;
        int reallen = 0;
        int rval = 0;
        struct sata_id *sid = NULL;
        char model[SATA_ID_MODEL_LEN + 1];
        char fw[SATA_ID_FW_LEN + 1];
        char *vid, *pid;
        int i;

        mutex_exit(&mpt->m_mutex);
        /*
         * According SCSI/ATA Translation -2 (SAT-2) revision 01a
         * chapter 12.4.2 VPD page 89h includes 512 bytes ATA IDENTIFY
         * DEVICE data or ATA IDENTIFY PACKET DEVICE data.
         */
        inq89 = kmem_zalloc(inq89_len, KM_SLEEP);
        rval = mptsas_inquiry(mpt, ptgt, 0, 0x89,
            inq89, inq89_len, &reallen, 1);

        if (rval != 0) {
                if (inq89 != NULL) {
                        kmem_free(inq89, inq89_len);
                }

                mptsas_log(mpt, CE_WARN, "!mptsas request inquiry page "
                    "0x89 for SATA target:%x failed!", ptgt->m_devhdl);
                return (DDI_SUCCESS);
        }
        sid = (void *)(&inq89[60]);

        swab(sid->ai_model, model, SATA_ID_MODEL_LEN);
        swab(sid->ai_fw, fw, SATA_ID_FW_LEN);

        model[SATA_ID_MODEL_LEN] = 0;
        fw[SATA_ID_FW_LEN] = 0;

        /*
         * split model into into vid/pid
         */
```

```
                for (i = 0, pid = model; i < SATA_ID_MODEL_LEN; i++, pid++)
                        if ((*pid == ' ') || (*pid == '\t'))
                                break;
                if (i < SATA_ID_MODEL_LEN) {
                        vid = model;
                        /*
                         * terminate vid, establish pid
                         */
                        *pid++ = 0;
                } else {
                        /*
                         * vid will stay "ATA     ", the rule is same
                         * as sata framework implementation.
                         */
                        vid = NULL;
                        /*
                         * model is all pid
                         */
                        pid = model;
                }

                /*
                 * override SCSA "inquiry-*" properties
                 */
                if (vid)
                        (void) scsi_device_prop_update_inqstring(sd,
                            INQUIRY_VENDOR_ID, vid, strlen(vid));
                if (pid)
                        (void) scsi_device_prop_update_inqstring(sd,
                            INQUIRY_PRODUCT_ID, pid, strlen(pid));
                (void) scsi_device_prop_update_inqstring(sd,
                    INQUIRY_REVISION_ID, fw, strlen(fw));

                if (inq89 != NULL) {
                        kmem_free(inq89, inq89_len);
                }
        } else {
                mutex_exit(&mpt->m_mutex);
        }

        return (DDI_SUCCESS);
}
/*
 * tran_tgt_free(9E) - target device instance deallocation
 */
static void
mptsas_scsi_tgt_free(dev_info_t *hba_dip, dev_info_t *tgt_dip,
    scsi_hba_tran_t *hba_tran, struct scsi_device *sd)
{
#ifndef __lock_lint
        _NOTE(ARGUNUSED(hba_dip, tgt_dip, hba_tran, sd))
#endif

        mptsas_tgt_private_t    *tgt_private = hba_tran->tran_tgt_private;

        if (tgt_private != NULL) {
                kmem_free(tgt_private, sizeof (mptsas_tgt_private_t));
                hba_tran->tran_tgt_private = NULL;
        }
}

/*
 * scsi_pkt handling
 *
 * Visible to the external world via the transport structure.
 */
```



```
/*
 * Notes:
 *      - transport the command to the addressed SCSI target/lun device
 *      - normal operation is to schedule the command to be transported,
 *        and return TRAN_ACCEPT if this is successful.
 *      - if NO_INTR, tran_start must poll device for command completion
 */
static int
mptsas_scsi_start(struct scsi_address *ap, struct scsi_pkt *pkt)
{
#ifndef __lock_lint
        _NOTE(ARGUNUSED(ap))
#endif
        mptsas_t        *mpt = PKT2MPT(pkt);
        mptsas_cmd_t    *cmd = PKT2CMD(pkt);
        int             rval, start;
        uint8_t         pref;
        mptsas_target_t *ptgt = cmd->cmd_tgt_addr;
        mptsas_tx_waitqueue_t *txwq;

        NDBG1(("mptsas_scsi_start: pkt=0x%p", (void *)pkt));
        ASSERT(ptgt);
        if (ptgt == NULL)
                return (TRAN_FATAL_ERROR);

        /*
         * prepare the pkt before taking mutex.
         */
        rval = mptsas_prepare_pkt(cmd);
        if (rval != TRAN_ACCEPT) {
                return (rval);
        }

        /*
         * Send the command to target/lun, however your HBA requires it.
         * If busy, return TRAN_BUSY; if there's some other formatting error
         * in the packet, return TRAN_BADPKT; otherwise, fall through to the
         * return of TRAN_ACCEPT.
         *
         * Remember that access to shared resources, including the mptsas_t
         * data structure and the HBA hardware registers, must be protected
         * with mutexes, here and everywhere.
         *
         * Also remember that at interrupt time, you'll get an argument
         * to the interrupt handler which is a pointer to your mptsas_t
         * structure; you'll have to remember which commands are outstanding
         * and which scsi_pkt is the currently-running command so the
         * interrupt handler can refer to the pkt to set completion
         * status, call the target driver back through pkt_comp, etc.
         *
         * If the instance lock is held by other thread, don't spin to wait
         * for it. Instead, queue the cmd and next time when the instance lock
         * is not held, accept all the queued cmd. A extra tx_waitq is
         * introduced to protect the queue.
         *
         * The polled cmd will not be queud and accepted as usual.
         *
         * Under the tx_waitq mutex, record whether a thread is draining
         * the tx_waitq.  An IO requesting thread that finds the instance
         * mutex contended appends to the tx_waitq and while holding the
         * tx_wait mutex, if the draining flag is not set, sets it and then
         * proceeds to spin for the instance mutex. This scheme ensures that
         * the last cmd in a burst be processed.
         *
         * we enable this feature only when the helper threads are enabled,
```

```
 * at which we think the loads are heavy.
 *
 * per instance, per queue mutex m_tx_waitq[i].txwq_mutex is
 * introduced to protect the txwq_qtail, txwq_cmdq, txwq_len
 */

        if (mpt->m_txwq_enabled == TRUE) {
                int gotmtx = 0;

                if (mpt->m_txwq_allow_q_jumping) {
                        gotmtx = mutex_tryenter(&mpt->m_mutex);
                }
                if (gotmtx == 0) {
                        /* We didn't get the mutex or didn't try */
                        if (cmd->cmd_pkt_flags & FLAG_NOINTR) {
                                mutex_enter(&mpt->m_mutex);
                                /* Polled commands queue jump */
                                mptsas_accept_tx_waitqs(mpt);
                        } else {
                                rval = mptsas_check_targ_intxtion(
                                    cmd->cmd_tgt_addr,
                                    cmd->cmd_pkt_flags);
                                if (rval != TRAN_ACCEPT) {
                                        return (rval);
                                }

                                cmd->cmd_flags |= CFLAG_TXQ;
                                pref = mpt->m_pref_tx_waitq;
                                txwq = &mpt->m_tx_waitq[pref];

                                if (mutex_tryenter(&txwq->txwq_mutex) == 0) {
                                        txwq = &mpt->m_tx_waitq[pref^1];
                                        mutex_enter(&txwq->txwq_mutex);
                                } else {
                                        pref ^= 1;
                                        mpt->m_pref_tx_waitq = pref;
                                }

                                *txwq->txwq_qtail = cmd;
                                txwq->txwq_qtail = &cmd->cmd_linkp;
                                txwq->txwq_len++;
                                if (!txwq->txwq_draining) {
                                        cv_signal(&txwq->txwq_cv);
                                }
                                mutex_exit(&txwq->txwq_mutex);
                                return (rval);
                        }
                }
        } else {
                mutex_enter(&mpt->m_mutex);
        }
        rval = mptsas_check_targ_intxtion(cmd->cmd_tgt_addr,
            cmd->cmd_pkt_flags);
        if (rval != TRAN_ACCEPT) {
                mutex_exit(&mpt->m_mutex);
                return (rval);
        }

        start = mptsas_accept_pkt(mpt, cmd, &rval);
        mutex_exit(&mpt->m_mutex);
        if (start) {
                (void) mptsas_start_cmd(mpt, cmd);
        }

        return (rval);
}
```



```
static int
mptsas_check_targ_intxtion(mptsas_target_t *ptgt, int cmd_pkt_flags)
{
        /*
         * ptgt->m_dr_flag is a variable that is only ever changed by
         * direct write under the main m_mutex.
         * It doesn't need a mutex hold to protect this read.
         */

        if (ptgt->m_dr_flag == MPTSAS_DR_INTRANSITION) {
                if (cmd_pkt_flags & FLAG_NOQUEUE) {
                        /*
                         * The command should be allowed to retry by returning
                         * TRAN_BUSY to stall the I/O's which come from
                         * scsi_vhci since the device/path is in unstable state
                         * now.
                         */
                        return (TRAN_BUSY);
                } else {
                        /*
                         * The device is offline, just fail the command by
                         * return TRAN_FATAL_ERROR.
                         */
                        return (TRAN_FATAL_ERROR);
                }
        }
        return (TRAN_ACCEPT);
}

#if 0
/*
 * Note that this function has a side effect of releasing the
 * per target mutex.
 */
static void
mptsas_offline_target_direct(mptsas_t *mpt, mptsas_target_t *ptgt)
{
        char                            phy_mask_name[MPTSAS_MAX_PHYS];
        mptsas_phymask_t                phymask = ptgt->m_addr.mta_phymask;
        dev_info_t                      *parent;

        ASSERT(mutex_owned(&mpt->m_mutex));

        ptgt->m_dr_flag = MPTSAS_DR_INTRANSITION;
        bzero(phy_mask_name, MPTSAS_MAX_PHYS);
        (void) sprintf(phy_mask_name, "%x", phymask);
        parent = scsi_hba_iport_find(mpt->m_dip, phy_mask_name);

        if (parent != NULL) {
                mptsas_offline_target(mpt, ptgt,
                    ptgt->m_deviceinfo & DEVINFO_DIRECT_ATTACHED ?
                    MPTSAS_TOPO_FLAG_DIRECT_ATTACHED_DEVICE :
                    MPTSAS_TOPO_FLAG_EXPANDER_ATTACHED_DEVICE,
                    parent);
        } else {
                mptsas_log(mpt, CE_WARN, "Failed to find an "
                    "iport for \"%s\", should not happen!",
                    phy_mask_name);
        }
}
#endif

/*
 * In order to be efficient with the m_mutex (which can be dropped before
 * calling mptsas_start_cmd()) indicate if start_cmd should be called via the
```

```
 * returned value (FALSE or TRUE). Caller is then responsible for doing the
 * right thing with the m_mutex.
 */
static int
mptsas_accept_pkt(mptsas_t *mpt, mptsas_cmd_t *cmd, int *tran_rval)
{
        int             rval = TRAN_ACCEPT;
        mptsas_target_t *ptgt = cmd->cmd_tgt_addr;

        NDBG1(("mptsas_accept_pkt: cmd=0x%p", (void *)cmd));

        ASSERT(mutex_owned(&mpt->m_mutex));

        if ((cmd->cmd_flags & CFLAG_PREPARED) == 0) {
                rval = mptsas_prepare_pkt(cmd);
                if (rval != TRAN_ACCEPT) {
                        cmd->cmd_flags &= ~CFLAG_TRANFLAG;
                        goto set_tranrval;
                }
        }

        /*
         * If the command came from the tx wait q it may have slipped
         * by the check for dr_flag before being added to the queue.
         * Fail here with abort status.
         */
        if (cmd->cmd_flags & CFLAG_TXQ) {
                rval = mptsas_check_targ_intxtion(cmd->cmd_tgt_addr,
                    cmd->cmd_pkt_flags);
                if (rval != TRAN_ACCEPT) {
                        mptsas_set_pkt_reason(mpt, cmd, CMD_ABORTED,
                            STAT_ABORTED);
                        mptsas_doneq_add(mpt, cmd);
                        mptsas_doneq_empty(mpt);
                        goto set_tranrval;
                }
        }
        /*
         * If HBA is being reset, the DevHandles are being re-initialized,
         * which means that they could be invalid even if the target is still
         * attached.  Check if being reset and if DevHandle is being
         * re-initialized.  If this is the case, return BUSY so the I/O can be
         * retried later.
         */
        if ((ptgt->m_devhdl == MPTSAS_INVALID_DEVHDL) && mpt->m_in_reset) {
                mptsas_set_pkt_reason(mpt, cmd, CMD_RESET, STAT_BUS_RESET);
                if (cmd->cmd_flags & CFLAG_TXQ) {
                        mptsas_doneq_add(mpt, cmd);
                        mptsas_doneq_empty(mpt);
                } else {
                        rval = TRAN_BUSY;
                }
                goto set_tranrval;
        }

        mutex_enter(&ptgt->m_t_mutex);
        /*
         * reset the throttle if we were draining
         */
        if ((ptgt->m_t_ncmds == 0) &&
            (ptgt->m_t_throttle == DRAIN_THROTTLE)) {
                NDBG23(("reset throttle"));
                ASSERT(ptgt->m_reset_delay == 0);
                mptsas_set_throttle(mpt, ptgt, MAX_THROTTLE);
        }
```



```
        /*
         * If device handle has already been invalidated, just
         * fail the command. In theory, for a command from scsi_vhci
         * client it’s impossible to receive a command with an invalid
         * devhdl since devhdl is set after path offline, target
         * driver is not supposed to select an offlined path.
         */
        if (ptgt->m_devhdl == MPTSAS_INVALID_DEVHDL) {
                NDBG3(("rejecting command, it might because invalid devhdl "
                    "request."));
                mutex_exit(&ptgt->m_t_mutex);
                mptsas_set_pkt_reason(mpt, cmd, CMD_DEV_GONE, STAT_TERMINATED);
                if (cmd->cmd_flags & CFLAG_TXQ) {
                        mptsas_doneq_add(mpt, cmd);
                        mptsas_doneq_empty(mpt);
                } else {
                        rval = TRAN_FATAL_ERROR;
                }
                goto set_tranrval;
        }
        /*
         * The first case is the normal case.  mpt gets a command from the
         * target driver and starts it.
         * Since SMID 0 is reserved and the TM slot is reserved, the actual max
         * commands is m_max_requests - 2.
         */
        if ((mpt->m_ncmds <= (mpt->m_max_requests - 2)) &&
            (ptgt->m_t_throttle > HOLD_THROTTLE) &&
            (ptgt->m_t_ncmds < ptgt->m_t_throttle) &&
            (ptgt->m_reset_delay == 0) && (mpt->m_polled_intr == 0) &&
            (ptgt->m_t_nwait == 0) &&
            ((cmd->cmd_pkt_flags & FLAG_NOINTR) == 0)) {
                ASSERT((cmd->cmd_flags & CFLAG_CMDIOC) == 0);
                if (mptsas_save_cmd_to_slot(mpt, cmd) == TRUE) {
                        ptgt->m_t_ncmds++;
                        mutex_exit(&ptgt->m_t_mutex);
                        cmd->cmd_active_expiration = 0;
                        *tran_rval = rval;
                        return (TRUE);
                } else {
                        mutex_exit(&ptgt->m_t_mutex);
                        mptsas_waitq_add(mpt, cmd);
                }
        } else {
                mutex_exit(&ptgt->m_t_mutex);
                /*
                 * Add this pkt to the work queue
                 */
                mptsas_waitq_add(mpt, cmd);

                if (cmd->cmd_pkt_flags & FLAG_NOINTR) {
                        (void) mptsas_poll(mpt, cmd, MPTSAS_POLL_TIME);

                        /*
                         * Only flush the doneq if this is not a TM
                         * cmd.  For TM cmds the flushing of the
                         * doneq will be done in those routines.
                         */
                        if ((cmd->cmd_flags & CFLAG_TM_CMD) == 0) {
                                mptsas_doneq_empty(mpt);
                        }
                }
        }
set_tranrval:
        *tran_rval = rval;
        return (FALSE);
```

```
}

static void
mptsas_retry_pkt(mptsas_t *mpt, mptsas_cmd_t *cmd)
{
        int             rval;

        cmd->cmd_pkt_flags |= FLAG_HEAD;
        cmd->cmd_flags |= CFLAG_RETRY;
        cmd->cmd_flags &= ~CFLAG_TXQ;
        if (mptsas_accept_pkt(mpt, cmd, &rval)) {
                (void) mptsas_start_cmd(mpt, cmd);
        }

        /*
         * If there was a problem clear the retry flag so that the
         * command will be completed with error rather than get lost!
         */
        if (rval != TRAN_ACCEPT)
                cmd->cmd_flags &= ~CFLAG_RETRY;
}

static int
mptsas_save_cmd_to_slot(mptsas_t *mpt, mptsas_cmd_t *cmd)
{
        mptsas_slots_t *slots = mpt->m_active;
        uint_t slot, start_rotor, rotor, n_normal;

        /*
         * Account for reserved TM request slot and reserved SMID of 0.
         */
        ASSERT(slots->m_n_normal == (mpt->m_max_requests - 2));

        /*
         * Find the next available slot, beginning at m_rotor.  If no slot is
         * available, we'll return FALSE to indicate that.  This mechanism
         * considers only the normal slots, not the reserved slot 0 nor the
         * task management slot m_n_normal + 1.  The rotor is left to point to
         * the normal slot after the one we select, unless we select the last
         * normal slot in which case it returns to slot 1.
         */
        start_rotor = rotor = slots->m_rotor;
        n_normal = slots->m_n_normal;
        do {
                slot = rotor++;
                if (rotor > n_normal)
                        rotor = 1;

                if (rotor == start_rotor)
                        break;
        } while (slots->m_slot[slot] != NULL);
        slots->m_rotor = rotor;

        if (slots->m_slot[slot] != NULL)
                return (FALSE);

        ASSERT(slot != 0 && slot <= slots->m_n_normal);

        cmd->cmd_slot = slot;
        slots->m_slot[slot] = cmd;
        atomic_inc_32(&mpt->m_ncmds);

        /*
         * Distribute the commands amongst the reply queues (Interrupt vectors).
         * Stick to 0 for polled.
         */
```



```
        if (!(cmd->cmd_pkt_flags & FLAG_NOINTR) &&
            !(cmd->cmd_flags & (CFLAG_PASSTHRU|CFLAG_CONFIG|CFLAG_FW_DIAG)) &&
            (mpt->m_post_reply_qcount > 1)) {
                cmd->cmd_rpqidx = slot % mpt->m_post_reply_qcount;
        }
        atomic_inc_32(&mpt->m_rep_post_queues[cmd->cmd_rpqidx].rpq_ncmds);
        return (TRUE);
}

int
mptsas_save_cmd(mptsas_t *mpt, mptsas_cmd_t *cmd)
{
        mptsas_target_t *ptgt = cmd->cmd_tgt_addr;

        ASSERT(MUTEX_HELD(&mpt->m_mutex));

        if (!mptsas_save_cmd_to_slot(mpt, cmd)) {
                return (FALSE);
        }

        /*
         * only increment per target ncmds if this is not a
         * command that has no target associated with it (i.e. a
         * event acknoledgement)
         */
        if ((cmd->cmd_flags & CFLAG_CMDIOC) == 0) {
                /*
                 * Expiration time is set in mptsas_start_cmd
                 */
                mutex_enter(&ptgt->m_t_mutex);
                ptgt->m_t_ncmds++;
                mutex_exit(&ptgt->m_t_mutex);
                cmd->cmd_active_expiration = 0;
        } else {
                /*
                 * Initialize expiration time for passthrough commands,
                 */
                cmd->cmd_active_expiration = gethrtime() +
                    (hrtime_t)cmd->cmd_pkt->pkt_time * NANOSEC;
        }
        return (TRUE);
}

/*
 * prepare the pkt:
 * the pkt may have been resubmitted or just reused so
 * initialize some fields and do some checks.
 */
static int
mptsas_prepare_pkt(mptsas_cmd_t *cmd)
{
        struct scsi_pkt *pkt = CMD2PKT(cmd);

        NDBG1(("mptsas_prepare_pkt: cmd=0x%p", (void *)cmd));

        /*
         * Reinitialize some fields that need it; the packet may
         * have been resubmitted
         */
        pkt->pkt_reason = CMD_CMPLT;
        pkt->pkt_state = 0;
        pkt->pkt_statistics = 0;
        pkt->pkt_resid = 0;
        cmd->cmd_age = 0;
        cmd->cmd_pkt_flags = pkt->pkt_flags;
```

```
        /*
         * zero status byte.
         */
        *(pkt->pkt_scbp) = 0;

        if (cmd->cmd_flags & CFLAG_DMAVALID) {
                pkt->pkt_resid = cmd->cmd_dmacount;

                /*
                 * consistent packets need to be sync'ed first
                 * (only for data going out)
                 */
                if ((cmd->cmd_flags & CFLAG_CMDIOPB) &&
                    (cmd->cmd_flags & CFLAG_DMASEND)) {
                        (void) ddi_dma_sync(cmd->cmd_dmahandle, 0, 0,
                            DDI_DMA_SYNC_FORDEV);
                }
        }

        cmd->cmd_flags =
            (cmd->cmd_flags & ~(CFLAG_TRANFLAG)) |
            CFLAG_PREPARED | CFLAG_IN_TRANSPORT;

        return (TRAN_ACCEPT);
}

/*
 * tran_init_pkt(9E) - allocate scsi_pkt(9S) for command
 *
 * One of three possibilities:
 *      - allocate scsi_pkt
 *      - allocate scsi_pkt and DMA resources
 *      - allocate DMA resources to an already-allocated pkt
 */
static struct scsi_pkt *
mptsas_scsi_init_pkt(struct scsi_address *ap, struct scsi_pkt *pkt,
    struct buf *bp, int cmdlen, int statuslen, int tgtlen, int flags,
    int (*callback)(), caddr_t arg)
{
        mptsas_cmd_t            *cmd, *new_cmd;
        mptsas_t                *mpt = ADDR2MPT(ap);
        int                     failure = 1;
        uint_t                  oldcookiec;
        mptsas_target_t         *ptgt = NULL;
        int                     rval;
        mptsas_tgt_private_t    *tgt_private;
        int                     kf;

        kf = (callback == SLEEP_FUNC)? KM_SLEEP: KM_NOSLEEP;

        tgt_private = (mptsas_tgt_private_t *)ap->a_hba_tran->
            tran_tgt_private;
        ASSERT(tgt_private != NULL);
        if (tgt_private == NULL) {
                return (NULL);
        }
        ptgt = tgt_private->t_private;
        ASSERT(ptgt != NULL);
        if (ptgt == NULL)
                return (NULL);
        ap->a_target = ptgt->m_devhdl;
        ap->a_lun = tgt_private->t_lun;

        ASSERT(callback == NULL_FUNC || callback == SLEEP_FUNC);
#ifdef MPTSAS_TEST_EXTRN_ALLOC
        statuslen *= 100; tgtlen *= 4;
```



```
#endif
        NDBG3(("mptsas_scsi_init_pkt:\n"
            "\ttgt=%d in=0x%p bp=0x%p clen=%d slen=%d tlen=%d flags=%x",
            ap->a_target, (void *)pkt, (void *)bp,
            cmdlen, statuslen, tgtlen, flags));

        /*
         * Allocate the new packet.
         */
        if (pkt == NULL) {
                ddi_dma_handle_t        save_dma_handle;

                cmd = kmem_cache_alloc(mpt->m_kmem_cache, kf);

                if (cmd) {
                        save_dma_handle = cmd->cmd_dmahandle;
                        bzero(cmd, sizeof (*cmd) + scsi_pkt_size());
                        cmd->cmd_dmahandle = save_dma_handle;

                        pkt = (void *)((uchar_t *)cmd +
                            sizeof (struct mptsas_cmd));
                        pkt->pkt_ha_private = (opaque_t)cmd;
                        pkt->pkt_address = *ap;
                        pkt->pkt_private = (opaque_t)cmd->cmd_pkt_private;
                        pkt->pkt_scbp = (opaque_t)&cmd->cmd_scb;
                        pkt->pkt_cdbp = (opaque_t)&cmd->cmd_cdb;
                        cmd->cmd_pkt = (struct scsi_pkt *)pkt;
                        cmd->cmd_cdblen = (uchar_t)cmdlen;
                        cmd->cmd_scblen = statuslen;
                        cmd->cmd_rqslen = SENSE_LENGTH;
                        cmd->cmd_tgt_addr = ptgt;
                        failure = 0;
                }

                if (failure || (cmdlen > sizeof (cmd->cmd_cdb)) ||
                    (tgtlen > PKT_PRIV_LEN) ||
                    (statuslen > EXTCMDS_STATUS_SIZE)) {
                        if (failure == 0) {
                                /*
                                 * if extern alloc fails, all will be
                                 * deallocated, including cmd
                                 */
                                failure = mptsas_pkt_alloc_extern(mpt, cmd,
                                    cmdlen, tgtlen, statuslen, kf);
                        }
                        if (failure) {
                                /*
                                 * if extern allocation fails, it will
                                 * deallocate the new pkt as well
                                 */
                                return (NULL);
                        }
                }
                new_cmd = cmd;

        } else {
                cmd = PKT2CMD(pkt);
                new_cmd = NULL;
        }


        /* grab cmd->cmd_cookiec here as oldcookiec */

        oldcookiec = cmd->cmd_cookiec;

        /*
```

```
 * If the dma was broken up into PARTIAL transfers cmd_nwin will be
 * greater than 0 and we'll need to grab the next dma window
 */
/*
 * SLM-not doing extra command frame right now; may add later
 */

if (cmd->cmd_nwin > 0) {

        /*
         * Make sure we havn't gone past the the total number
         * of windows
         */
        if (++cmd->cmd_winindex >= cmd->cmd_nwin) {
                return (NULL);
        }
        if (ddi_dma_getwin(cmd->cmd_dmahandle, cmd->cmd_winindex,
            &cmd->cmd_dma_offset, &cmd->cmd_dma_len,
            &cmd->cmd_cookie, &cmd->cmd_cookiec) == DDI_FAILURE) {
                return (NULL);
        }
        goto get_dma_cookies;
}


if (flags & PKT_XARQ) {
        cmd->cmd_flags |= CFLAG_XARQ;
}

/*
 * DMA resource allocation.  This version assumes your
 * HBA has some sort of bus-mastering or onboard DMA capability, with a
 * scatter-gather list of length MPTSAS_MAX_DMA_SEGS, as given in the
 * ddi_dma_attr_t structure and passed to scsi_impl_dmaget.
 */
if (bp && (bp->b_bcount != 0) &&
    (cmd->cmd_flags & CFLAG_DMAVALID) == 0) {

        int     cnt, dma_flags;
        mptti_t *dmap;          /* ptr to the S/G list */

        /*
         * Set up DMA memory and position to the next DMA segment.
         */
        ASSERT(cmd->cmd_dmahandle != NULL);

        if (bp->b_flags & B_READ) {
                dma_flags = DDI_DMA_READ;
                cmd->cmd_flags &= ~CFLAG_DMASEND;
        } else {
                dma_flags = DDI_DMA_WRITE;
                cmd->cmd_flags |= CFLAG_DMASEND;
        }
        if (flags & PKT_CONSISTENT) {
                cmd->cmd_flags |= CFLAG_CMDIOPB;
                dma_flags |= DDI_DMA_CONSISTENT;
        }

        if (flags & PKT_DMA_PARTIAL) {
                dma_flags |= DDI_DMA_PARTIAL;
        }

        /*
         * workaround for byte hole issue on psycho and
         * schizo pre 2.1
         */
```



```
        if ((bp->b_flags & B_READ) && ((bp->b_flags &
            (B_PAGEIO|B_REMAPPED)) != B_PAGEIO) &&
            ((uintptr_t)bp->b_un.b_addr & 0x7)) {
                dma_flags |= DDI_DMA_CONSISTENT;
        }

        rval = ddi_dma_buf_bind_handle(cmd->cmd_dmahandle, bp,
            dma_flags, callback, arg,
            &cmd->cmd_cookie, &cmd->cmd_cookiec);
        if (rval == DDI_DMA_PARTIAL_MAP) {
                (void) ddi_dma_numwin(cmd->cmd_dmahandle,
                    &cmd->cmd_nwin);
                cmd->cmd_winindex = 0;
                (void) ddi_dma_getwin(cmd->cmd_dmahandle,
                    cmd->cmd_winindex, &cmd->cmd_dma_offset,
                    &cmd->cmd_dma_len, &cmd->cmd_cookie,
                    &cmd->cmd_cookiec);
        } else if (rval && (rval != DDI_DMA_MAPPED)) {
                switch (rval) {
                case DDI_DMA_NORESOURCES:
                        bioerror(bp, 0);
                        break;
                case DDI_DMA_BADATTR:
                case DDI_DMA_NOMAPPING:
                        bioerror(bp, EFAULT);
                        break;
                case DDI_DMA_TOOBIG:
                default:
                        bioerror(bp, EINVAL);
                        break;
                }
                cmd->cmd_flags &= ~CFLAG_DMAVALID;
                if (new_cmd) {
                        mptsas_scsi_destroy_pkt(ap, pkt);
                }
                return ((struct scsi_pkt *)NULL);
        }

get_dma_cookies:
        cmd->cmd_flags |= CFLAG_DMAVALID;
        ASSERT(cmd->cmd_cookiec > 0);

        if (cmd->cmd_cookiec > MPTSAS_MAX_CMD_SEGS) {
                mptsas_log(mpt, CE_NOTE, "large cookiec received %d\n",
                    cmd->cmd_cookiec);
                bioerror(bp, EINVAL);
                if (new_cmd) {
                        mptsas_scsi_destroy_pkt(ap, pkt);
                }
                return ((struct scsi_pkt *)NULL);
        }

        /*
         * Allocate extra SGL buffer if needed.
         */
        if ((cmd->cmd_cookiec > MPTSAS_MAX_FRAME_SGES64(mpt)) &&
            (cmd->cmd_extra_frames == NULL)) {
                if (mptsas_alloc_extra_sgl_frame(mpt, cmd) ==
                    DDI_FAILURE) {
                        mptsas_log(mpt, CE_WARN, "MPT SGL mem alloc "
                            "failed");
                        bioerror(bp, ENOMEM);
                        if (new_cmd) {
                                mptsas_scsi_destroy_pkt(ap, pkt);
                        }
                        return ((struct scsi_pkt *)NULL);
```

```
                        }
                }

                /*
                 * Always use scatter-gather transfer
                 * Use the loop below to store physical addresses of
                 * DMA segments, from the DMA cookies, into your HBA's
                 * scatter-gather list.
                 * We need to ensure we have enough kmem alloc'd
                 * for the sg entries since we are no longer using an
                 * array inside mptsas_cmd_t.
                 *
                 * We check cmd->cmd_cookiec against oldcookiec so
                 * the scatter-gather list is correctly allocated
                 */

                if (oldcookiec != cmd->cmd_cookiec) {
                        if (cmd->cmd_sg != (mptti_t *)NULL) {
                                kmem_free(cmd->cmd_sg, sizeof (mptti_t) *
                                    oldcookiec);
                                cmd->cmd_sg = NULL;
                        }
                }

                if (cmd->cmd_sg == (mptti_t *)NULL) {
                        cmd->cmd_sg = kmem_alloc((size_t)(sizeof (mptti_t)*
                            cmd->cmd_cookiec), kf);

                        if (cmd->cmd_sg == (mptti_t *)NULL) {
                                mptsas_log(mpt, CE_WARN,
                                    "unable to kmem_alloc enough memory "
                                    "for scatter/gather list");
                /*
                 * if we have an ENOMEM condition we need to behave
                 * the same way as the rest of this routine
                 */

                                bioerror(bp, ENOMEM);
                                if (new_cmd) {
                                        mptsas_scsi_destroy_pkt(ap, pkt);
                                }
                                return ((struct scsi_pkt *)NULL);
                        }
                }

                dmap = cmd->cmd_sg;

                ASSERT(cmd->cmd_cookie.dmac_size != 0);

                /*
                 * store the first segment into the S/G list
                 */
                dmap->count = cmd->cmd_cookie.dmac_size;
                dmap->addr.address64.Low = (uint32_t)
                    (cmd->cmd_cookie.dmac_laddress & 0xffffffffull);
                dmap->addr.address64.High = (uint32_t)
                    (cmd->cmd_cookie.dmac_laddress >> 32);

                /*
                 * dmacount counts the size of the dma for this window
                 * (if partial dma is being used).  totaldmacount
                 * keeps track of the total amount of dma we have
                 * transferred for all the windows (needed to calculate
                 * the resid value below).
                 */
                cmd->cmd_dmacount = cmd->cmd_cookie.dmac_size;
```



```
                cmd->cmd_totaldmacount += cmd->cmd_cookie.dmac_size;

                /*
                 * We already stored the first DMA scatter gather segment,
                 * start at 1 if we need to store more.
                 */
                for (cnt = 1; cnt < cmd->cmd_cookiec; cnt++) {
                        /*
                         * Get next DMA cookie
                         */
                        ddi_dma_nextcookie(cmd->cmd_dmahandle,
                            &cmd->cmd_cookie);
                        dmap++;

                        cmd->cmd_dmacount += cmd->cmd_cookie.dmac_size;
                        cmd->cmd_totaldmacount += cmd->cmd_cookie.dmac_size;

                        /*
                         * store the segment parms into the S/G list
                         */
                        dmap->count = cmd->cmd_cookie.dmac_size;
                        dmap->addr.address64.Low = (uint32_t)
                            (cmd->cmd_cookie.dmac_laddress & 0xffffffffull);
                        dmap->addr.address64.High = (uint32_t)
                            (cmd->cmd_cookie.dmac_laddress >> 32);
                }

                /*
                 * If this was partially allocated we set the resid
                 * the amount of data NOT transferred in this window
                 * If there is only one window, the resid will be 0
                 */
                pkt->pkt_resid = (bp->b_bcount - cmd->cmd_totaldmacount);
                NDBG3(("mptsas_scsi_init_pkt: cmd_dmacount=%d.",
                    cmd->cmd_dmacount));
        }
        return (pkt);
}

/*
 * tran_destroy_pkt(9E) - scsi_pkt(9s) deallocation
 *
 * Notes:
 *      - also frees DMA resources if allocated
 *      - implicit DMA synchonization
 */
static void
mptsas_scsi_destroy_pkt(struct scsi_address *ap, struct scsi_pkt *pkt)
{
        mptsas_cmd_t    *cmd = PKT2CMD(pkt);
        mptsas_t        *mpt = ADDR2MPT(ap);

        NDBG3(("mptsas_scsi_destroy_pkt: target=%d pkt=0x%p",
            ap->a_target, (void *)pkt));

        if (cmd->cmd_flags & CFLAG_DMAVALID) {
                (void) ddi_dma_unbind_handle(cmd->cmd_dmahandle);
                cmd->cmd_flags &= ~CFLAG_DMAVALID;
        }

        if (cmd->cmd_sg) {
                kmem_free(cmd->cmd_sg, sizeof (mptti_t) * cmd->cmd_cookiec);
                cmd->cmd_sg = NULL;
        }

        mptsas_free_extra_sgl_frame(mpt, cmd);
```

```
        if ((cmd->cmd_flags &
            (CFLAG_FREE | CFLAG_CDBEXTERN | CFLAG_PRIVEXTERN |
            CFLAG_SCBEXTERN)) == 0) {
                cmd->cmd_flags = CFLAG_FREE;
                kmem_cache_free(mpt->m_kmem_cache, (void *)cmd);
        } else {
                mptsas_pkt_destroy_extern(mpt, cmd);
        }
}

/*
 * kmem cache constructor and destructor:
 * When constructing, we bzero the cmd and allocate the dma handle
 * When destructing, just free the dma handle
 */
static int
mptsas_kmem_cache_constructor(void *buf, void *cdrarg, int kmflags)
{
        mptsas_cmd_t            *cmd = buf;
        mptsas_t                *mpt  = cdrarg;
        int                     (*callback)(caddr_t);

        callback = (kmflags == KM_SLEEP)? DDI_DMA_SLEEP: DDI_DMA_DONTWAIT;

        NDBG4(("mptsas_kmem_cache_constructor"));

        /*
         * allocate a dma handle
         */
        if ((ddi_dma_alloc_handle(mpt->m_dip, &mpt->m_io_dma_attr, callback,
            NULL, &cmd->cmd_dmahandle)) != DDI_SUCCESS) {
                cmd->cmd_dmahandle = NULL;
                return (-1);
        }
        return (0);
}

static void
mptsas_kmem_cache_destructor(void *buf, void *cdrarg)
{
#ifndef __lock_lint
        _NOTE(ARGUNUSED(cdrarg))
#endif
        mptsas_cmd_t    *cmd = buf;

        NDBG4(("mptsas_kmem_cache_destructor"));

        if (cmd->cmd_dmahandle) {
                ddi_dma_free_handle(&cmd->cmd_dmahandle);
                cmd->cmd_dmahandle = NULL;
        }
}

static int
mptsas_cache_frames_constructor(void *buf, void *cdrarg, int kmflags)
{
        mptsas_cache_frames_t   *p = buf;
        mptsas_t                *mpt = cdrarg;
        ddi_dma_attr_t          frame_dma_attr;
        size_t                  mem_size, alloc_len;
        ddi_dma_cookie_t        cookie;
        uint_t                  ncookie;
        int (*callback)(caddr_t) = (kmflags == KM_SLEEP)
            ? DDI_DMA_SLEEP: DDI_DMA_DONTWAIT;
```



```
        frame_dma_attr = mpt->m_msg_dma_attr;
        frame_dma_attr.dma_attr_align = 0x10;
        frame_dma_attr.dma_attr_sgllen = 1;

        if (ddi_dma_alloc_handle(mpt->m_dip, &frame_dma_attr, callback, NULL,
            &p->m_dma_hdl) != DDI_SUCCESS) {
                mptsas_log(mpt, CE_WARN, "Unable to allocate dma handle for"
                    " extra SGL.");
                return (DDI_FAILURE);
        }

        mem_size = (mpt->m_max_request_frames - 1) * mpt->m_req_frame_size;

        if (ddi_dma_mem_alloc(p->m_dma_hdl, mem_size, &mpt->m_dev_acc_attr,
            DDI_DMA_CONSISTENT, callback, NULL, (caddr_t *)&p->m_frames_addr,
            &alloc_len, &p->m_acc_hdl) != DDI_SUCCESS) {
                ddi_dma_free_handle(&p->m_dma_hdl);
                p->m_dma_hdl = NULL;
                mptsas_log(mpt, CE_WARN, "Unable to allocate dma memory for"
                    " extra SGL.");
                return (DDI_FAILURE);
        }

        if (ddi_dma_addr_bind_handle(p->m_dma_hdl, NULL, p->m_frames_addr,
            alloc_len, DDI_DMA_RDWR | DDI_DMA_CONSISTENT, callback, NULL,
            &cookie, &ncookie) != DDI_DMA_MAPPED) {
                (void) ddi_dma_mem_free(&p->m_acc_hdl);
                ddi_dma_free_handle(&p->m_dma_hdl);
                p->m_dma_hdl = NULL;
                mptsas_log(mpt, CE_WARN, "Unable to bind DMA resources for"
                    " extra SGL");
                return (DDI_FAILURE);
        }

        /*
         * Store the SGL memory address.  This chip uses this
         * address to dma to and from the driver.  The second
         * address is the address mpt uses to fill in the SGL.
         */
        p->m_phys_addr = cookie.dmac_laddress;

        return (DDI_SUCCESS);
}

static void
mptsas_cache_frames_destructor(void *buf, void *cdrarg)
{
#ifndef __lock_lint
        _NOTE(ARGUNUSED(cdrarg))
#endif
        mptsas_cache_frames_t   *p = buf;
        if (p->m_dma_hdl != NULL) {
                (void) ddi_dma_unbind_handle(p->m_dma_hdl);
                (void) ddi_dma_mem_free(&p->m_acc_hdl);
                ddi_dma_free_handle(&p->m_dma_hdl);
                p->m_phys_addr = NULL;
                p->m_frames_addr = NULL;
                p->m_dma_hdl = NULL;
                p->m_acc_hdl = NULL;
        }

}

/*
 * allocate and deallocate external pkt space (ie. not part of mptsas_cmd)
 * for non-standard length cdb, pkt_private, status areas
```

```
 * if allocation fails, then deallocate all external space and the pkt
 */
/* ARGSUSED */
static int
mptsas_pkt_alloc_extern(mptsas_t *mpt, mptsas_cmd_t *cmd,
    int cmdlen, int tgtlen, int statuslen, int kf)
{
        caddr_t                 cdbp, scbp, tgt;
        size_t                  senselength;

        NDBG3(("mptsas_pkt_alloc_extern: "
            "cmd=0x%p cmdlen=%d tgtlen=%d statuslen=%d kf=%x",
            (void *)cmd, cmdlen, tgtlen, statuslen, kf));

        tgt = cdbp = scbp = NULL;
        cmd->cmd_scblen         = statuslen;
        cmd->cmd_privlen        = (uchar_t)tgtlen;

        if (cmdlen > sizeof (cmd->cmd_cdb)) {
                if ((cdbp = kmem_zalloc((size_t)cmdlen, kf)) == NULL) {
                        goto fail;
                }
                cmd->cmd_pkt->pkt_cdbp = (opaque_t)cdbp;
                cmd->cmd_flags |= CFLAG_CDBEXTERN;
        }
        if (tgtlen > PKT_PRIV_LEN) {
                if ((tgt = kmem_zalloc((size_t)tgtlen, kf)) == NULL) {
                        goto fail;
                }
                cmd->cmd_flags |= CFLAG_PRIVEXTERN;
                cmd->cmd_pkt->pkt_private = tgt;
        }
        if (statuslen > EXTCMDS_STATUS_SIZE) {
                if ((scbp = kmem_zalloc((size_t)statuslen, kf)) == NULL) {
                        goto fail;
                }
                cmd->cmd_flags |= CFLAG_SCBEXTERN;
                cmd->cmd_pkt->pkt_scbp = (opaque_t)scbp;

                /* allocate sense data buf for DMA */

                senselength = statuslen - MPTSAS_GET_ITEM_OFF(
                    struct scsi_arq_status, sts_sensedata);
                if (senselength > mpt->m_req_sense_size) {
                        unsigned long i;
                        cmd->cmd_extrqslen = (uint16_t)senselength;
                        cmd->cmd_extrqschunks = (senselength +
                            (mpt->m_req_sense_size - 1))/mpt->m_req_sense_size;
                        i = rmalloc_wait(mpt->m_erqsense_map,
                            cmd->cmd_extrqschunks);
                        ASSERT(i != 0);
                        cmd->cmd_extrqsidx = i - 1;
                        cmd->cmd_arq_buf = mpt->m_extreq_sense +
                            (cmd->cmd_extrqsidx * mpt->m_req_sense_size);
                } else {
                        cmd->cmd_rqslen = (uchar_t)senselength;
                }
        }
        return (0);
fail:
        mptsas_pkt_destroy_extern(mpt, cmd);
        return (1);
}

/*
 * deallocate external pkt space and deallocate the pkt
```



```
 */
static void
mptsas_pkt_destroy_extern(mptsas_t *mpt, mptsas_cmd_t *cmd)
{
        NDBG3(("mptsas_pkt_destroy_extern: cmd=0x%p", (void *)cmd));

        if (cmd->cmd_flags & CFLAG_FREE) {
                mptsas_log(mpt, CE_PANIC,
                    "mptsas_pkt_destroy_extern: freeing free packet");
                _NOTE(NOT_REACHED)
                /* NOTREACHED */
        }
        if (cmd->cmd_extrqslen != 0) {
                rmfree(mpt->m_erqsense_map, cmd->cmd_extrqschunks,
                    cmd->cmd_extrqsidx + 1);
        }
        if (cmd->cmd_flags & CFLAG_CDBEXTERN) {
                kmem_free(cmd->cmd_pkt->pkt_cdbp, (size_t)cmd->cmd_cdblen);
        }
        if (cmd->cmd_flags & CFLAG_SCBEXTERN) {
                kmem_free(cmd->cmd_pkt->pkt_scbp, (size_t)cmd->cmd_scblen);
        }
        if (cmd->cmd_flags & CFLAG_PRIVEXTERN) {
                kmem_free(cmd->cmd_pkt->pkt_private, (size_t)cmd->cmd_privlen);
        }
        cmd->cmd_flags = CFLAG_FREE;
        kmem_cache_free(mpt->m_kmem_cache, (void *)cmd);
}

/*
 * tran_sync_pkt(9E) - explicit DMA synchronization
 */
/*ARGSUSED*/
static void
mptsas_scsi_sync_pkt(struct scsi_address *ap, struct scsi_pkt *pkt)
{
        mptsas_cmd_t    *cmd = PKT2CMD(pkt);

        NDBG3(("mptsas_scsi_sync_pkt: target=%d, pkt=0x%p",
            ap->a_target, (void *)pkt));

        if (cmd->cmd_dmahandle) {
                (void) ddi_dma_sync(cmd->cmd_dmahandle, 0, 0,
                    (cmd->cmd_flags & CFLAG_DMASEND) ?
                    DDI_DMA_SYNC_FORDEV : DDI_DMA_SYNC_FORCPU);
        }
}

/*
 * tran_dmafree(9E) - deallocate DMA resources allocated for command
 */
/*ARGSUSED*/
static void
mptsas_scsi_dmafree(struct scsi_address *ap, struct scsi_pkt *pkt)
{
        mptsas_cmd_t    *cmd = PKT2CMD(pkt);
        mptsas_t        *mpt = ADDR2MPT(ap);

        NDBG3(("mptsas_scsi_dmafree: target=%d pkt=0x%p",
            ap->a_target, (void *)pkt));

        if (cmd->cmd_flags & CFLAG_DMAVALID) {
                (void) ddi_dma_unbind_handle(cmd->cmd_dmahandle);
                cmd->cmd_flags &= ~CFLAG_DMAVALID;
        }
```

```
        mptsas_free_extra_sgl_frame(mpt, cmd);
}

static void
mptsas_pkt_comp(struct scsi_pkt *pkt, mptsas_cmd_t *cmd)
{
        if ((cmd->cmd_flags & CFLAG_CMDIOPB) &&
            (!(cmd->cmd_flags & CFLAG_DMASEND))) {
                (void) ddi_dma_sync(cmd->cmd_dmahandle, 0, 0,
                    DDI_DMA_SYNC_FORCPU);
        }
        (*pkt->pkt_comp)(pkt);
}

static void
mptsas_sge_mainframe(mptsas_cmd_t *cmd, pMpi2SCSIIORequest_t frame,
                ddi_acc_handle_t acc_hdl, uint_t cookiec,
                uint32_t end_flags)
{
        pMpi2SGESimple64_t      sge;
        mptti_t                 *dmap;
        uint32_t                flags;

        dmap = cmd->cmd_sg;

        sge = (pMpi2SGESimple64_t)(&frame->SGL);
        while (cookiec--) {
                ddi_put32(acc_hdl, &sge->Address.Low,
                    dmap->addr.address64.Low);
                ddi_put32(acc_hdl, &sge->Address.High,
                    dmap->addr.address64.High);
                ddi_put32(acc_hdl, &sge->FlagsLength, dmap->count);
                flags = ddi_get32(acc_hdl, &sge->FlagsLength);
                flags |= ((uint32_t)
                    (MPI2_SGE_FLAGS_SIMPLE_ELEMENT |
                    MPI2_SGE_FLAGS_SYSTEM_ADDRESS |
                    MPI2_SGE_FLAGS_64_BIT_ADDRESSING) <<
                    MPI2_SGE_FLAGS_SHIFT);

                /*
                 * If this is the last cookie, we set the flags
                 * to indicate so
                 */
                if (cookiec == 0) {
                        flags |= end_flags;
                }
                if (cmd->cmd_flags & CFLAG_DMASEND) {
                        flags |= (MPI2_SGE_FLAGS_HOST_TO_IOC <<
                            MPI2_SGE_FLAGS_SHIFT);
                } else {
                        flags |= (MPI2_SGE_FLAGS_IOC_TO_HOST <<
                            MPI2_SGE_FLAGS_SHIFT);
                }
                ddi_put32(acc_hdl, &sge->FlagsLength, flags);
                dmap++;
                sge++;
        }
}

static void
mptsas_sge_chain(mptsas_t *mpt, mptsas_cmd_t *cmd,
    pMpi2SCSIIORequest_t frame, ddi_acc_handle_t acc_hdl)
{
        pMpi2SGESimple64_t      sge;
        pMpi2SGEChain64_t       sgechain;
        uint64_t                nframe_phys_addr;
```



```
        uint_t                  cookiec;
        mptti_t                 *dmap;
        uint32_t                flags;
        int                     i, j, k, l, frames, sgemax;
        int                     temp, maxframe_sges;
        uint8_t                 chainflags;
        uint16_t                chainlength;
        mptsas_cache_frames_t   *p;

        cookiec = cmd->cmd_cookiec;

        /*
         * Hereby we start to deal with multiple frames.
         * The process is as follows:
         * 1. Determine how many frames are needed for SGL element
         *    storage; Note that all frames are stored in contiguous
         *    memory space and in 64-bit DMA mode each element is
         *    3 double-words (12 bytes) long.
         * 2. Fill up the main frame. We need to do this separately
         *    since it contains the SCSI IO request header and needs
         *    dedicated processing. Note that the last 4 double-words
         *    of the SCSI IO header is for SGL element storage
         *    (MPI2_SGE_IO_UNION).
         * 3. Fill the chain element in the main frame, so the DMA
         *    engine can use the following frames.
         * 4. Enter a loop to fill the remaining frames. Note that the
         *    last frame contains no chain element.  The remaining
         *    frames go into the mpt SGL buffer allocated on the fly,
         *    not immediately following the main message frame, as in
         *    Gen1.
         * Some restrictions:
         * 1. For 64-bit DMA, the simple element and chain element
         *    are both of 3 double-words (12 bytes) in size, even
         *    though all frames are stored in the first 4G of mem
         *    range and the higher 32-bits of the address are always 0.
         * 2. On some controllers (like the 1064/1068), a frame can
         *    hold SGL elements with the last 1 or 2 double-words
         *    (4 or 8 bytes) un-used. On these controllers, we should
         *    recognize that there’s not enough room for another SGL
         *    element and move the sge pointer to the next frame.
         */

        /*
         * Sgemax is the number of SGE’s that will fit
         * each extra frame and frames is total
         * number of frames we’ll need.  1 sge entry per
         * frame is reseverd for the chain element thus the -1 below.
         */
        sgemax = ((mpt->m_req_frame_size / sizeof (MPI2_SGE_SIMPLE64)) - 1);
        maxframe_sges = MPTSAS_MAX_FRAME_SGES64(mpt);
        temp = (cookiec - (maxframe_sges - 1)) / sgemax;

        /*
         * A little check to see if we need to round up the number
         * of frames we need
         */
        if ((cookiec - (maxframe_sges - 1)) - (temp * sgemax) > 1) {
                frames = (temp + 1);
        } else {
                frames = temp;
        }
        dmap = cmd->cmd_sg;
        sge = (pMpi2SGESimple64_t)(&frame->SGL);

        /*
         * First fill in the main frame
```

```
         */
        j = maxframe_sges - 1;
        mptsas_sge_mainframe(cmd, frame, acc_hdl, j,
            ((uint32_t)(MPI2_SGE_FLAGS_LAST_ELEMENT) <<
            MPI2_SGE_FLAGS_SHIFT));
        dmap += j;
        sge += j;
        j++;

        /*
         * Fill in the chain element in the main frame.
         * About calculation on ChainOffset:
         * 1. Struct msg_scsi_io_request has 4 double-words (16 bytes)
         *    in the end reserved for SGL element storage
         *    (MPI2_SGE_IO_UNION); we should count it in our
         *    calculation.  See its definition in the header file.
         * 2. Constant j is the counter of the current SGL element
         *    that will be processed, and (j - 1) is the number of
         *    SGL elements that have been processed (stored in the
         *    main frame).
         * 3. ChainOffset value should be in units of double-words (4
         *    bytes) so the last value should be divided by 4.
         */
        ddi_put8(acc_hdl, &frame->ChainOffset,
            (sizeof (MPI2_SCSI_IO_REQUEST) -
            sizeof (MPI2_SGE_IO_UNION) +
            (j - 1) * sizeof (MPI2_SGE_SIMPLE64)) >> 2);
        sgechain = (pMpi2SGEChain64_t)sge;
        chainflags = (MPI2_SGE_FLAGS_CHAIN_ELEMENT |
            MPI2_SGE_FLAGS_SYSTEM_ADDRESS |
            MPI2_SGE_FLAGS_64_BIT_ADDRESSING);
        ddi_put8(acc_hdl, &sgechain->Flags, chainflags);

        /*
         * The size of the next frame is the accurate size of space
         * (in bytes) used to store the SGL elements. j is the counter
         * of SGL elements. (j - 1) is the number of SGL elements that
         * have been processed (stored in frames).
         */
        if (frames >= 2) {
                chainlength = mpt->m_req_frame_size /
                    sizeof (MPI2_SGE_SIMPLE64) *
                    sizeof (MPI2_SGE_SIMPLE64);
        } else {
                chainlength = ((cookiec - (j - 1)) *
                    sizeof (MPI2_SGE_SIMPLE64));
        }

        p = cmd->cmd_extra_frames;

        ddi_put16(acc_hdl, &sgechain->Length, chainlength);
        ddi_put32(acc_hdl, &sgechain->Address.Low,
            (p->m_phys_addr&0xffffffffull));
        ddi_put32(acc_hdl, &sgechain->Address.High, p->m_phys_addr>>32);

        /*
         * If there are more than 2 frames left we have to
         * fill in the next chain offset to the location of
         * the chain element in the next frame.
         * sgemax is the number of simple elements in an extra
         * frame. Note that the value NextChainOffset should be
         * in double-words (4 bytes).
         */
        if (frames >= 2) {
                ddi_put8(acc_hdl, &sgechain->NextChainOffset,
                    (sgemax * sizeof (MPI2_SGE_SIMPLE64)) >> 2);
```



```
        } else {
                ddi_put8(acc_hdl, &sgechain->NextChainOffset, 0);
        }

        /*
         * Jump to next frame;
         * Starting here, chain buffers go into the per command SGL.
         * This buffer is allocated when chain buffers are needed.
         */
        sge = (pMpi2SGESimple64_t)p->m_frames_addr;
        i = cookiec;

        /*
         * Start filling in frames with SGE's.  If we
         * reach the end of frame and still have SGE's
         * to fill we need to add a chain element and
         * use another frame.  j will be our counter
         * for what cookie we are at and i will be
         * the total cookiec. k is the current frame
         */
        for (k = 1; k <= frames; k++) {
                for (l = 1; (l <= (sgemax + 1)) && (j <= i); j++, l++) {

                        /*
                         * If we have reached the end of frame
                         * and we have more SGE's to fill in
                         * we have to fill the final entry
                         * with a chain element and then
                         * continue to the next frame
                         */
                        if ((l == (sgemax + 1)) && (k != frames)) {
                                sgechain = (pMpi2SGEChain64_t)sge;
                                j--;
                                chainflags = (
                                    MPI2_SGE_FLAGS_CHAIN_ELEMENT |
                                    MPI2_SGE_FLAGS_SYSTEM_ADDRESS |
                                    MPI2_SGE_FLAGS_64_BIT_ADDRESSING);
                                ddi_put8(p->m_acc_hdl,
                                    &sgechain->Flags, chainflags);
                                /*
                                 * k is the frame counter and (k + 1)
                                 * is the number of the next frame.
                                 * Note that frames are in contiguous
                                 * memory space.
                                 */
                                nframe_phys_addr = p->m_phys_addr +
                                    (mpt->m_req_frame_size * k);
                                ddi_put32(p->m_acc_hdl,
                                    &sgechain->Address.Low,
                                    nframe_phys_addr&0xffffffffull);
                                ddi_put32(p->m_acc_hdl,
                                    &sgechain->Address.High,
                                    nframe_phys_addr>>32);

                                /*
                                 * If there are more than 2 frames left
                                 * we have to next chain offset to
                                 * the location of the chain element
                                 * in the next frame and fill in the
                                 * length of the next chain
                                 */
                                if ((frames - k) >= 2) {
                                        ddi_put8(p->m_acc_hdl,
                                            &sgechain->NextChainOffset,
                                            (sgemax *
                                            sizeof (MPI2_SGE_SIMPLE64))
```

```
                                        >> 2);
                                ddi_put16(p->m_acc_hdl,
                                    &sgechain->Length,
                                    mpt->m_req_frame_size /
                                    sizeof (MPI2_SGE_SIMPLE64) *
                                    sizeof (MPI2_SGE_SIMPLE64));
                        } else {
                                /*
                                 * This is the last frame. Set
                                 * the NextChainOffset to 0 and
                                 * Length is the total size of
                                 * all remaining simple elements
                                 */
                                ddi_put8(p->m_acc_hdl,
                                    &sgechain->NextChainOffset,
                                    0);
                                ddi_put16(p->m_acc_hdl,
                                    &sgechain->Length,
                                    (cookiec - j) *
                                    sizeof (MPI2_SGE_SIMPLE64));
                        }

                        /* Jump to the next frame */
                        sge = (pMpi2SGESimple64_t)
                            ((char *)p->m_frames_addr +
                            (int)mpt->m_req_frame_size * k);

                        continue;
                }

                ddi_put32(p->m_acc_hdl,
                    &sge->Address.Low,
                    dmap->addr.address64.Low);
                ddi_put32(p->m_acc_hdl,
                    &sge->Address.High,
                    dmap->addr.address64.High);
                ddi_put32(p->m_acc_hdl,
                    &sge->FlagsLength, dmap->count);
                flags = ddi_get32(p->m_acc_hdl,
                    &sge->FlagsLength);
                flags |= ((uint32_t)(
                    MPI2_SGE_FLAGS_SIMPLE_ELEMENT |
                    MPI2_SGE_FLAGS_SYSTEM_ADDRESS |
                    MPI2_SGE_FLAGS_64_BIT_ADDRESSING) <<
                    MPI2_SGE_FLAGS_SHIFT);

                /*
                 * If we are at the end of the frame and
                 * there is another frame to fill in
                 * we set the last simple element as last
                 * element
                 */
                if ((l == sgemax) && (k != frames)) {
                        flags |= ((uint32_t)
                            (MPI2_SGE_FLAGS_LAST_ELEMENT) <<
                            MPI2_SGE_FLAGS_SHIFT);
                }

                /*
                 * If this is the final cookie we
                 * indicate it by setting the flags
                 */
                if (j == i) {
                        flags |= ((uint32_t)
                            (MPI2_SGE_FLAGS_LAST_ELEMENT |
                            MPI2_SGE_FLAGS_END_OF_BUFFER |
```



```
                                        MPI2_SGE_FLAGS_END_OF_LIST) <<
                                        MPI2_SGE_FLAGS_SHIFT);
                                }
                                if (cmd->cmd_flags & CFLAG_DMASEND) {
                                        flags |=
                                            (MPI2_SGE_FLAGS_HOST_TO_IOC <<
                                            MPI2_SGE_FLAGS_SHIFT);
                                } else {
                                        flags |=
                                            (MPI2_SGE_FLAGS_IOC_TO_HOST <<
                                            MPI2_SGE_FLAGS_SHIFT);
                                }
                                ddi_put32(p->m_acc_hdl,
                                    &sge->FlagsLength, flags);
                                dmap++;
                                sge++;
                        }
                }

                /*
                 * Sync DMA with the chain buffers that were just created
                 */
                (void) ddi_dma_sync(p->m_dma_hdl, 0, 0, DDI_DMA_SYNC_FORDEV);
}

static void
mptsas_ieee_sge_mainframe(mptsas_cmd_t *cmd, pMpi2SCSIIORequest_t frame,
    ddi_acc_handle_t acc_hdl, uint_t cookiec,
    uint8_t end_flag)
{
        pMpi2IeeeSgeSimple64_t  ieeesge;
        mptti_t                 *dmap;
        uint8_t                 flags;

        dmap = cmd->cmd_sg;

        NDBG1(("mptsas_ieee_sge_mainframe: cookiec=%d, %s", cookiec,
            cmd->cmd_flags & CFLAG_DMASEND?"Out":"In"));

        ieeesge = (pMpi2IeeeSgeSimple64_t)(&frame->SGL);
        while (cookiec--) {
                ddi_put32(acc_hdl, &ieeesge->Address.Low,
                    dmap->addr.address64.Low);
                ddi_put32(acc_hdl, &ieeesge->Address.High,
                    dmap->addr.address64.High);
                ddi_put32(acc_hdl, &ieeesge->Length, dmap->count);
                NDBG1(("mptsas_ieee_sge_mainframe: len=%d, high=0x%x",
                    dmap->count, dmap->addr.address64.High));
                flags = (MPI2_IEEE_SGE_FLAGS_SIMPLE_ELEMENT |
                    MPI2_IEEE_SGE_FLAGS_SYSTEM_ADDR);

                /*
                 * If this is the last cookie, we set the flags
                 * to indicate so
                 */
                if (cookiec == 0) {
                        flags |= end_flag;
                }

                /*
                 * XXX: Hmmm, what about the direction based on
                 * cmd->cmd_flags & CFLAG_DMASEND?
                 */
                ddi_put8(acc_hdl, &ieeesge->Flags, flags);
                dmap++;
                ieeesge++;
```

```
        }
}

static void
mptsas_ieee_sge_chain(mptsas_t *mpt, mptsas_cmd_t *cmd,
    pMpi2SCSIIORequest_t frame, ddi_acc_handle_t acc_hdl)
{
        pMpi2IeeeSgeSimple64_t  ieeesge;
        pMpi25IeeeSgeChain64_t  ieeesgechain;
        uint64_t                nframe_phys_addr;
        uint_t                  cookiec;
        mptti_t                 *dmap;
        uint8_t                 flags;
        int                     i, j, k, l, frames, sgemax;
        int                     temp, maxframe_sges;
        uint8_t                 chainflags;
        uint32_t                chainlength;
        mptsas_cache_frames_t   *p;

        cookiec = cmd->cmd_cookiec;

        NDBG1(("mptsas_ieee_sge_chain: cookiec=%d", cookiec));

        /*
         * Hereby we start to deal with multiple frames.
         * The process is as follows:
         * 1. Determine how many frames are needed for SGL element
         *    storage; Note that all frames are stored in contiguous
         *    memory space and in 64-bit DMA mode each element is
         *    4 double-words (16 bytes) long.
         * 2. Fill up the main frame. We need to do this separately
         *    since it contains the SCSI IO request header and needs
         *    dedicated processing. Note that the last 4 double-words
         *    of the SCSI IO header is for SGL element storage
         *    (MPI2_SGE_IO_UNION).
         * 3. Fill the chain element in the main frame, so the DMA
         *    engine can use the following frames.
         * 4. Enter a loop to fill the remaining frames. Note that the
         *    last frame contains no chain element.  The remaining
         *    frames go into the mpt SGL buffer allocated on the fly,
         *    not immediately following the main message frame, as in
         *    Gen1.
         * Some restrictions:
         * 1. For 64-bit DMA, the simple element and chain element
         *    are both of 4 double-words (16 bytes) in size, even
         *    though all frames are stored in the first 4G of mem
         *    range and the higher 32-bits of the address are always 0.
         * 2. On some controllers (like the 1064/1068), a frame can
         *    hold SGL elements with the last 1 or 2 double-words
         *    (4 or 8 bytes) un-used. On these controllers, we should
         *    recognize that there’s not enough room for another SGL
         *    element and move the sge pointer to the next frame.
         */

        /*
         * Sgemax is the number of SGE’s that will fit
         * each extra frame and frames is total
         * number of frames we’ll need.  1 sge entry per
         * frame is reseverd for the chain element thus the -1 below.
         */
        sgemax = ((mpt->m_req_frame_size / sizeof (MPI2_IEEE_SGE_SIMPLE64))
            - 1);
        maxframe_sges = MPTSAS_MAX_FRAME_SGES64(mpt);
        temp = (cookiec - (maxframe_sges - 1)) / sgemax;

        /*
```



```
         * A little check to see if we need to round up the number
         * of frames we need
         */
        if ((cookiec - (maxframe_sges - 1)) - (temp * sgemax) > 1) {
                frames = (temp + 1);
        } else {
                frames = temp;
        }
        NDBG1(("mptsas_ieee_sge_chain: temp=%d, frames=%d", temp, frames));
        dmap = cmd->cmd_sg;
        ieeesge = (pMpi2IeeeSgeSimple64_t)(&frame->SGL);

        /*
         * First fill in the main frame
         */
        j = maxframe_sges - 1;
        mptsas_ieee_sge_mainframe(cmd, frame, acc_hdl, j, 0);
        dmap += j;
        ieeesge += j;
        j++;

        /*
         * Fill in the chain element in the main frame.
         * About calculation on ChainOffset:
         * 1. Struct msg_scsi_io_request has 4 double-words (16 bytes)
         *    in the end reserved for SGL element storage
         *    (MPI2_SGE_IO_UNION); we should count it in our
         *    calculation.  See its definition in the header file.
         * 2. Constant j is the counter of the current SGL element
         *    that will be processed, and (j - 1) is the number of
         *    SGL elements that have been processed (stored in the
         *    main frame).
         * 3. ChainOffset value should be in units of quad-words (16
         *    bytes) so the last value should be divided by 16.
         */
        ddi_put8(acc_hdl, &frame->ChainOffset,
            (sizeof (MPI2_SCSI_IO_REQUEST) -
            sizeof (MPI2_SGE_IO_UNION) +
            (j - 1) * sizeof (MPI2_IEEE_SGE_SIMPLE64)) >> 4);
        ieeesgechain = (pMpi25IeeeSgeChain64_t)ieeesge;
        chainflags = (MPI2_IEEE_SGE_FLAGS_CHAIN_ELEMENT |
            MPI2_IEEE_SGE_FLAGS_SYSTEM_ADDR);
        ddi_put8(acc_hdl, &ieeesgechain->Flags, chainflags);

        /*
         * The size of the next frame is the accurate size of space
         * (in bytes) used to store the SGL elements. j is the counter
         * of SGL elements. (j - 1) is the number of SGL elements that
         * have been processed (stored in frames).
         */
        if (frames >= 2) {
                chainlength = mpt->m_req_frame_size /
                    sizeof (MPI2_IEEE_SGE_SIMPLE64) *
                    sizeof (MPI2_IEEE_SGE_SIMPLE64);
        } else {
                chainlength = ((cookiec - (j - 1)) *
                    sizeof (MPI2_IEEE_SGE_SIMPLE64));
        }

        p = cmd->cmd_extra_frames;

        ddi_put32(acc_hdl, &ieeesgechain->Length, chainlength);
        ddi_put32(acc_hdl, &ieeesgechain->Address.Low,
            p->m_phys_addr&0xffffffffull);
        ddi_put32(acc_hdl, &ieeesgechain->Address.High, p->m_phys_addr>>32);
```

```
        /*
         * If there are more than 2 frames left we have to
         * fill in the next chain offset to the location of
         * the chain element in the next frame.
         * sgemax is the number of simple elements in an extra
         * frame. Note that the value NextChainOffset should be
         * in double-words (4 bytes).
         */
        if (frames >= 2) {
                ddi_put8(acc_hdl, &ieeesgechain->NextChainOffset,
                    (sgemax * sizeof (MPI2_IEEE_SGE_SIMPLE64)) >> 4);
        } else {
                ddi_put8(acc_hdl, &ieeesgechain->NextChainOffset, 0);
        }

        /*
         * Jump to next frame;
         * Starting here, chain buffers go into the per command SGL.
         * This buffer is allocated when chain buffers are needed.
         */
        ieeesge = (pMpi2IeeeSgeSimple64_t)p->m_frames_addr;
        i = cookiec;

        /*
         * Start filling in frames with SGE's.  If we
         * reach the end of frame and still have SGE's
         * to fill we need to add a chain element and
         * use another frame.  j will be our counter
         * for what cookie we are at and i will be
         * the total cookiec. k is the current frame
         */
        for (k = 1; k <= frames; k++) {
                for (l = 1; (l <= (sgemax + 1)) && (j <= i); j++, l++) {

                        /*
                         * If we have reached the end of frame
                         * and we have more SGE's to fill in
                         * we have to fill the final entry
                         * with a chain element and then
                         * continue to the next frame
                         */
                        if ((l == (sgemax + 1)) && (k != frames)) {
                                ieeesgechain = (pMpi25IeeeSgeChain64_t)ieeesge;
                                j--;
                                chainflags =
                                    MPI2_IEEE_SGE_FLAGS_CHAIN_ELEMENT |
                                    MPI2_IEEE_SGE_FLAGS_SYSTEM_ADDR;
                                ddi_put8(p->m_acc_hdl,
                                    &ieeesgechain->Flags, chainflags);
                                /*
                                 * k is the frame counter and (k + 1)
                                 * is the number of the next frame.
                                 * Note that frames are in contiguous
                                 * memory space.
                                 */
                                nframe_phys_addr = p->m_phys_addr +
                                    (mpt->m_req_frame_size * k);
                                ddi_put32(p->m_acc_hdl,
                                    &ieeesgechain->Address.Low,
                                    nframe_phys_addr&0xffffffffull);
                                ddi_put32(p->m_acc_hdl,
                                    &ieeesgechain->Address.High,
                                    nframe_phys_addr>>32);

                                /*
                                 * If there are more than 2 frames left
```



```
                                 * we have to next chain offset to
                                 * the location of the chain element
                                 * in the next frame and fill in the
                                 * length of the next chain
                                 */
                                if ((frames - k) >= 2) {
                                        ddi_put8(p->m_acc_hdl,
                                            &ieeesgechain->NextChainOffset,
                                            (sgemax *
                                            sizeof (MPI2_IEEE_SGE_SIMPLE64))
                                            >> 4);
                                        ddi_put32(p->m_acc_hdl,
                                            &ieeesgechain->Length,
                                            mpt->m_req_frame_size /
                                            sizeof (MPI2_IEEE_SGE_SIMPLE64) *
                                            sizeof (MPI2_IEEE_SGE_SIMPLE64));
                                } else {
                                        /*
                                         * This is the last frame. Set
                                         * the NextChainOffset to 0 and
                                         * Length is the total size of
                                         * all remaining simple elements
                                         */
                                        ddi_put8(p->m_acc_hdl,
                                            &ieeesgechain->NextChainOffset,
                                            0);
                                        ddi_put32(p->m_acc_hdl,
                                            &ieeesgechain->Length,
                                            (cookiec - j) *
                                            sizeof (MPI2_IEEE_SGE_SIMPLE64));
                                }

                                /* Jump to the next frame */
                                ieeesge = (pMpi2IeeeSgeSimple64_t)
                                    ((char *)p->m_frames_addr +
                                    (int)mpt->m_req_frame_size * k);

                                continue;
                        }

                        ddi_put32(p->m_acc_hdl,
                            &ieeesge->Address.Low,
                            dmap->addr.address64.Low);
                        ddi_put32(p->m_acc_hdl,
                            &ieeesge->Address.High,
                            dmap->addr.address64.High);
                        ddi_put32(p->m_acc_hdl,
                            &ieeesge->Length, dmap->count);
                        flags = (MPI2_IEEE_SGE_FLAGS_SIMPLE_ELEMENT |
                            MPI2_IEEE_SGE_FLAGS_SYSTEM_ADDR);

                        /*
                         * If we are at the end of the frame and
                         * there is another frame to fill in
                         * do we need to do anything?
                         * if ((l == sgemax) && (k != frames)) {
                         * }
                         */

                        /*
                         * If this is the final cookie set end of list.
                         */
                        if (j == i) {
                                flags |= MPI25_IEEE_SGE_FLAGS_END_OF_LIST;
                        }
```

```
                        ddi_put8(p->m_acc_hdl, &ieeesge->Flags, flags);
                        dmap++;
                        ieeesge++;
                }
        }

        /*
         * Sync DMA with the chain buffers that were just created
         */
        (void) ddi_dma_sync(p->m_dma_hdl, 0, 0, DDI_DMA_SYNC_FORDEV);
}

static void
mptsas_sge_setup(mptsas_t *mpt, mptsas_cmd_t *cmd, uint32_t *control,
    pMpi2SCSIIORequest_t frame, ddi_acc_handle_t acc_hdl)
{
        ASSERT(cmd->cmd_flags & CFLAG_DMAVALID);

        NDBG1(("mptsas_sge_setup: cookiec=%d", cmd->cmd_cookiec));

        /*
         * Set read/write bit in control.
         */
        if (cmd->cmd_flags & CFLAG_DMASEND) {
                *control |= MPI2_SCSIIO_CONTROL_WRITE;
        } else {
                *control |= MPI2_SCSIIO_CONTROL_READ;
        }

        ddi_put32(acc_hdl, &frame->DataLength, cmd->cmd_dmacount);

        /*
         * We have 4 cases here.  First where we can fit all the
         * SG elements into the main frame, and the case
         * where we can't. The SG element is also different when using
         * MPI2.5 interface.
         * If we have more cookies than we can attach to a frame
         * we will need to use a chain element to point
         * a location of memory where the rest of the S/G
         * elements reside.
         */
        if (cmd->cmd_cookiec <= MPTSAS_MAX_FRAME_SGES64(mpt)) {
                if (mpt->m_MPI25) {
                        mptsas_ieee_sge_mainframe(cmd, frame, acc_hdl,
                            cmd->cmd_cookiec,
                            MPI25_IEEE_SGE_FLAGS_END_OF_LIST);
                } else {
                        mptsas_sge_mainframe(cmd, frame, acc_hdl,
                            cmd->cmd_cookiec,
                            ((uint32_t)(MPI2_SGE_FLAGS_LAST_ELEMENT
                            | MPI2_SGE_FLAGS_END_OF_BUFFER
                            | MPI2_SGE_FLAGS_END_OF_LIST) <<
                            MPI2_SGE_FLAGS_SHIFT));
                }
        } else {
                if (mpt->m_MPI25) {
                        mptsas_ieee_sge_chain(mpt, cmd, frame, acc_hdl);
                } else {
                        mptsas_sge_chain(mpt, cmd, frame, acc_hdl);
                }
        }
}

/*
 * Interrupt handling
 * Utility routine.  Poll for status of a command sent to HBA
```



```
 * without interrupts (a FLAG_NOINTR command).
 */
int
mptsas_poll(mptsas_t *mpt, mptsas_cmd_t *poll_cmd, int polltime)
{
        int             rval = TRUE;
        uint32_t        int_mask;

        NDBG5(("mptsas_poll: cmd=0x%p, flags 0x%x", (void *)poll_cmd,
            poll_cmd->cmd_flags));

        /*
         * Get the current interrupt mask and disable interrupts.  When
         * re-enabling ints, set mask to saved value.
         */
        int_mask = ddi_get32(mpt->m_datap, &mpt->m_reg->HostInterruptMask);
        MPTSAS_DISABLE_INTR(mpt);

        mpt->m_polled_intr = 1;

        if ((poll_cmd->cmd_flags & CFLAG_TM_CMD) == 0) {
                mptsas_restart_hba(mpt);
        }

        /*
         * Wait, using drv_usecwait(), long enough for the command to
         * reasonably return from the target if the target isn't
         * "dead".  A polled command may well be sent from scsi_poll, and
         * there are retries built in to scsi_poll if the transport
         * accepted the packet (TRAN_ACCEPT).  scsi_poll waits 1 second
         * and retries the transport up to scsi_poll_busycnt times
         * (currently 60) if
         * 1. pkt_reason is CMD_INCOMPLETE and pkt_state is 0, or
         * 2. pkt_reason is CMD_CMPLT and *pkt_scbp has STATUS_BUSY
         *
         * limit the waiting to avoid a hang in the event that the
         * cmd never gets started but we are still receiving interrupts
         */
        while (!(poll_cmd->cmd_flags & CFLAG_FINISHED)) {
                if (mptsas_wait_intr(mpt, polltime) == FALSE) {
                        NDBG5(("mptsas_poll: command incomplete"));
                        rval = FALSE;
                        break;
                }
        }

        if (rval == FALSE) {

                /*
                 * this isn't supposed to happen, the hba must be wedged
                 * Mark this cmd as a timeout.
                 */
                mptsas_set_pkt_reason(mpt, poll_cmd, CMD_TIMEOUT,
                    (STAT_TIMEOUT|STAT_ABORTED));

                if (poll_cmd->cmd_queued == FALSE) {

                        NDBG5(("mptsas_poll: not on waitq"));

                        poll_cmd->cmd_pkt->pkt_state |=
                            (STATE_GOT_BUS|STATE_GOT_TARGET|STATE_SENT_CMD);
                } else {

                        /* find and remove it from the waitq */
                        NDBG5(("mptsas_poll: delete from waitq"));
                        mptsas_waitq_delete(mpt, poll_cmd);
```

```
                }

        }
        mptsas_fma_check(mpt, poll_cmd);

        /*
         * Clear polling flag, re-enable interrupts.
         */
        mpt->m_polled_intr = 0;
        ddi_put32(mpt->m_datap, &mpt->m_reg->HostInterruptMask, int_mask);

        /*
         * If there are queued cmd, start them now.
         */
        if (mpt->m_waitq != NULL) {
                mptsas_restart_waitq(mpt);
        }

        NDBG5(("mptsas_poll: done"));
        return (rval);
}

/*
 * Used for polling cmds and TM function
 */
static int
mptsas_wait_intr(mptsas_t *mpt, int polltime)
{
        int                             cnt, rval = FALSE;
        pMpi2ReplyDescriptorsUnion_t    reply_desc_union;
        mptsas_reply_pqueue_t           *rpqp;

        NDBG5(("mptsas_wait_intr"));
        ASSERT(mutex_owned(&mpt->m_mutex));

        /*
         * Keep polling for at least (polltime * 1000) seconds
         */
        rpqp = mpt->m_rep_post_queues;

        /*
         * Drop the main mutex and grab the mutex for reply queue 0
         */
        mutex_exit(&mpt->m_mutex);
        mutex_enter(&rpqp->rpq_mutex);
        for (cnt = 0; cnt < polltime; cnt++) {
                (void) ddi_dma_sync(mpt->m_dma_post_queue_hdl, 0, 0,
                    DDI_DMA_SYNC_FORCPU);

                /*
                 * Polled requests should only come back through
                 * the first interrupt.
                 */
                reply_desc_union = (pMpi2ReplyDescriptorsUnion_t)
                    MPTSAS_GET_NEXT_REPLY(rpqp, rpqp->rpq_index);

                if (ddi_get32(mpt->m_acc_post_queue_hdl,
                    &reply_desc_union->Words.Low) == 0xFFFFFFFF ||
                    ddi_get32(mpt->m_acc_post_queue_hdl,
                    &reply_desc_union->Words.High) == 0xFFFFFFFF) {
                        drv_usecwait(1000);
                        continue;
                }

                /*
                 * The reply is valid, process it according to its
```



```
                 * type.
                 */
                mptsas_process_intr(mpt, rpqp, reply_desc_union);

                /*
                 * Clear the reply descriptor for re-use.
                 */
                ddi_put64(mpt->m_acc_post_queue_hdl,
                    &((uint64_t *)(void *)rpqp->rpq_queue)[rpqp->rpq_index],
                    0xFFFFFFFFFFFFFFFF);
                (void) ddi_dma_sync(mpt->m_dma_post_queue_hdl, 0, 0,
                    DDI_DMA_SYNC_FORDEV);

                if (++rpqp->rpq_index == mpt->m_post_queue_depth) {
                        rpqp->rpq_index = 0;
                }

                /*
                 * Update the reply index
                 */
                ddi_put32(mpt->m_datap,
                    &mpt->m_reg->ReplyPostHostIndex, rpqp->rpq_index);
                rval = TRUE;
                break;
        }

        mutex_exit(&rpqp->rpq_mutex);
        mutex_enter(&mpt->m_mutex);

        return (rval);
}

static void
mptsas_handle_scsi_io_success(mptsas_t *mpt,
    mptsas_reply_pqueue_t *rpqp,
    pMpi2ReplyDescriptorsUnion_t reply_desc)
{
        pMpi2SCSIIOSuccessReplyDescriptor_t     scsi_io_success;
        uint16_t                                SMID;
        mptsas_slots_t                          *slots = mpt->m_active;
        mptsas_cmd_t                            *cmd = NULL;
        struct scsi_pkt                         *pkt;

        scsi_io_success = (pMpi2SCSIIOSuccessReplyDescriptor_t)reply_desc;
        SMID = ddi_get16(mpt->m_acc_post_queue_hdl, &scsi_io_success->SMID);

        /*
         * This is a success reply so just complete the IO.  First, do a sanity
         * check on the SMID.  The final slot is used for TM requests, which
         * would not come into this reply handler.
         */
        if ((SMID == 0) || (SMID > slots->m_n_normal)) {
                mptsas_log(mpt, CE_WARN, "?Received invalid SMID of %d\n",
                    SMID);
                ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_UNAFFECTED);
                return;
        }

        cmd = slots->m_slot[SMID];

        /*
         * print warning and return if the slot is empty
         */
        if (cmd == NULL) {
                mptsas_log(mpt, CE_WARN, "?NULL command for successful SCSI IO "
                    "in slot %d", SMID);
```

```
                return;
        }
        ASSERT(cmd->cmd_rpqidx == rpqp->rpq_num);

        pkt = CMD2PKT(cmd);
        pkt->pkt_state |= (STATE_GOT_BUS | STATE_GOT_TARGET | STATE_SENT_CMD |
            STATE_GOT_STATUS);
        if (cmd->cmd_flags & CFLAG_DMAVALID) {
                pkt->pkt_state |= STATE_XFERRED_DATA;
        }
        pkt->pkt_resid = 0;

        if (cmd->cmd_flags & CFLAG_PASSTHRU) {
                cmd->cmd_flags |= CFLAG_FINISHED;
                cv_broadcast(&mpt->m_passthru_cv);
                return;
        }
        if (!(cmd->cmd_flags & CFLAG_TM_CMD)) {
                if (cmd->cmd_flags & CFLAG_CMDIOC) {
                        mutex_enter(&mpt->m_mutex);
                        mptsas_remove_cmd(mpt, cmd);
                        mutex_exit(&mpt->m_mutex);
                } else {
#ifdef MPTSAS_DEBUG
                        /*
                         * In order to test timeout for a command set
                         * mptsas_test_timeout via mdb to avoid completion
                         * processing here.
                         */
                        if (mptsas_test_timeout) {
                                mptsas_test_timeout = 0;
                                return;
                        }
#endif
                        /*
                         * This is the normal path, avoid grabbing
                         * the m_mutex.
                         */
                        mptsas_remove_cmd_nomtx(mpt, cmd);
                }
        }

        if (cmd->cmd_flags & CFLAG_RETRY) {
                /*
                 * The target returned QFULL or busy, do not add tihs
                 * pkt to the doneq since the hba will retry
                 * this cmd.
                 *
                 * The pkt has already been resubmitted in
                 * mptsas_handle_qfull() or in mptsas_check_scsi_io_error().
                 * Remove this cmd_flag here.
                 */
                cmd->cmd_flags &= ~CFLAG_RETRY;
        } else {
                mptsas_rpdoneq_add(mpt, rpqp, cmd);
        }
}

static void
mptsas_handle_address_reply(mptsas_t *mpt,
    pMpi2ReplyDescriptorsUnion_t reply_desc)
{
        pMpi2AddressReplyDescriptor_t   address_reply;
        pMPI2DefaultReply_t             reply;
        mptsas_fw_diagnostic_buffer_t   *pBuffer;
        uint32_t                        reply_addr, reply_frame_dma_baseaddr;
```



```
        uint16_t                        SMID, iocstatus;
        mptsas_slots_t                  *slots = mpt->m_active;
        mptsas_cmd_t                    *cmd = NULL;
        uint8_t                         function, buffer_type;
        m_replyh_arg_t                  *args;
        int                             reply_frame_no;

        ASSERT(mutex_owned(&mpt->m_mutex));

        address_reply = (pMpi2AddressReplyDescriptor_t)reply_desc;
        reply_addr = ddi_get32(mpt->m_acc_post_queue_hdl,
            &address_reply->ReplyFrameAddress);
        SMID = ddi_get16(mpt->m_acc_post_queue_hdl, &address_reply->SMID);

        /*
         * If reply frame is not in the proper range we should ignore this
         * message and exit the interrupt handler.
         */
        reply_frame_dma_baseaddr = mpt->m_reply_frame_dma_addr & 0xfffffffful;
        if ((reply_addr < reply_frame_dma_baseaddr) ||
            (reply_addr >= (reply_frame_dma_baseaddr +
            (mpt->m_reply_frame_size * mpt->m_max_replies))) ||
            ((reply_addr - reply_frame_dma_baseaddr) %
            mpt->m_reply_frame_size != 0)) {
                mptsas_log(mpt, CE_WARN, "?Received invalid reply frame "
                    "address 0x%x\n", reply_addr);
                ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_UNAFFECTED);
                return;
        }

        (void) ddi_dma_sync(mpt->m_dma_reply_frame_hdl, 0, 0,
            DDI_DMA_SYNC_FORCPU);
        reply = (pMPI2DefaultReply_t)(mpt->m_reply_frame + (reply_addr -
            reply_frame_dma_baseaddr));
        function = ddi_get8(mpt->m_acc_reply_frame_hdl, &reply->Function);

        NDBG31(("mptsas_handle_address_reply: function 0x%x, reply_addr=0x%x",
            function, reply_addr));

        /*
         * don't get slot information and command for events since these values
         * don't exist
         */
        if ((function != MPI2_FUNCTION_EVENT_NOTIFICATION) &&
            (function != MPI2_FUNCTION_DIAG_BUFFER_POST)) {
                /*
                 * This could be a TM reply, which use the last allocated SMID,
                 * so allow for that.
                 */
                if ((SMID == 0) || (SMID > (slots->m_n_normal + 1))) {
                        mptsas_log(mpt, CE_WARN, "?Received invalid SMID of "
                            "%d\n", SMID);
                        ddi_fm_service_impact(mpt->m_dip,
                            DDI_SERVICE_UNAFFECTED);
                        return;
                }

                cmd = slots->m_slot[SMID];

                /*
                 * print warning and return if the slot is empty
                 */
                if (cmd == NULL) {
                        mptsas_log(mpt, CE_WARN, "?NULL command for address "
                            "reply in slot %d", SMID);
                        return;
```

```
                }
                if ((cmd->cmd_flags &
                    (CFLAG_PASSTHRU | CFLAG_CONFIG | CFLAG_FW_DIAG))) {
                        cmd->cmd_rfm = reply_addr;
                        cmd->cmd_flags |= CFLAG_FINISHED;
                        cv_broadcast(&mpt->m_passthru_cv);
                        cv_broadcast(&mpt->m_config_cv);
                        cv_broadcast(&mpt->m_fw_diag_cv);
                        return;
                } else if (!(cmd->cmd_flags & CFLAG_FW_CMD)) {
                        mptsas_remove_cmd(mpt, cmd);
                }
                NDBG31(("\t\tmptsas_process_intr: slot=%d", SMID));
        }
        /*
         * Depending on the function, we need to handle
         * the reply frame (and cmd) differently.
         */
        switch (function) {
        case MPI2_FUNCTION_SCSI_IO_REQUEST:
                mptsas_check_scsi_io_error(mpt, (pMpi2SCSIIOReply_t)reply, cmd);
                break;
        case MPI2_FUNCTION_SCSI_TASK_MGMT:
                cmd->cmd_rfm = reply_addr;
                mptsas_check_task_mgt(mpt, (pMpi2SCSIManagementReply_t)reply,
                    cmd);
                break;
        case MPI2_FUNCTION_FW_DOWNLOAD:
                cmd->cmd_flags |= CFLAG_FINISHED;
                cv_signal(&mpt->m_fw_cv);
                break;
        case MPI2_FUNCTION_EVENT_NOTIFICATION:
                reply_frame_no = (reply_addr - reply_frame_dma_baseaddr) /
                    mpt->m_reply_frame_size;
                args = &mpt->m_replyh_args[reply_frame_no];
                args->mpt = (void *)mpt;
                args->rfm = reply_addr;

                /*
                 * Record the event if its type is enabled in
                 * this mpt instance by ioctl.
                 */
                mptsas_record_event(args);

                /*
                 * Handle time critical events
                 * NOT_RESPONDING/ADDED only now
                 */
                if (mptsas_handle_event_sync(args) == DDI_SUCCESS) {
                        /*
                         * Would not return main process,
                         * just let taskq resolve ack action
                         * and ack would be sent in taskq thread
                         */
                        NDBG20(("send mptsas_handle_event_sync success"));
                }

                if (mpt->m_in_reset) {
                        NDBG20(("dropping event received during reset"));
                        return;
                }

                if ((ddi_taskq_dispatch(mpt->m_event_taskq, mptsas_handle_event,
                    (void *)args, DDI_NOSLEEP)) != DDI_SUCCESS) {
                        mptsas_log(mpt, CE_WARN, "No memory available"
                        "for dispatch taskq");
```



```
                        /*
                         * Return the reply frame to the free queue.
                         */
                        ddi_put32(mpt->m_acc_free_queue_hdl,
                            &((uint32_t *)(void *)
                            mpt->m_free_queue)[mpt->m_free_index], reply_addr);
                        (void) ddi_dma_sync(mpt->m_dma_free_queue_hdl, 0, 0,
                            DDI_DMA_SYNC_FORDEV);
                        if (++mpt->m_free_index == mpt->m_free_queue_depth) {
                                mpt->m_free_index = 0;
                        }

                        ddi_put32(mpt->m_datap,
                            &mpt->m_reg->ReplyFreeHostIndex, mpt->m_free_index);
                }
                return;
        case MPI2_FUNCTION_DIAG_BUFFER_POST:
                /*
                 * If SMID is 0, this implies that the reply is due to a
                 * release function with a status that the buffer has been
                 * released.  Set the buffer flags accordingly.
                 */
                if (SMID == 0) {
                        iocstatus = ddi_get16(mpt->m_acc_reply_frame_hdl,
                            &reply->IOCStatus);
                        buffer_type = ddi_get8(mpt->m_acc_reply_frame_hdl,
                            &(((pMpi2DiagBufferPostReply_t)reply)->BufferType));
                        if (iocstatus == MPI2_IOCSTATUS_DIAGNOSTIC_RELEASED) {
                                pBuffer =
                                    &mpt->m_fw_diag_buffer_list[buffer_type];
                                pBuffer->valid_data = TRUE;
                                pBuffer->owned_by_firmware = FALSE;
                                pBuffer->immediate = FALSE;
                        }
                } else {
                        /*
                         * Normal handling of diag post reply with SMID.
                         */
                        cmd = slots->m_slot[SMID];

                        /*
                         * print warning and return if the slot is empty
                         */
                        if (cmd == NULL) {
                                mptsas_log(mpt, CE_WARN, "?NULL command for "
                                    "address reply in slot %d", SMID);
                                return;
                        }
                        cmd->cmd_rfm = reply_addr;
                        cmd->cmd_flags |= CFLAG_FINISHED;
                        cv_broadcast(&mpt->m_fw_diag_cv);
                }
                return;
        default:
                mptsas_log(mpt, CE_WARN, "Unknown function 0x%x ", function);
                break;
        }

        /*
         * Return the reply frame to the free queue.
         */
        ddi_put32(mpt->m_acc_free_queue_hdl,
            &((uint32_t *)(void *)mpt->m_free_queue)[mpt->m_free_index],
            reply_addr);
        (void) ddi_dma_sync(mpt->m_dma_free_queue_hdl, 0, 0,
            DDI_DMA_SYNC_FORDEV);
```

```
        if (++mpt->m_free_index == mpt->m_free_queue_depth) {
                mpt->m_free_index = 0;
        }
        ddi_put32(mpt->m_datap, &mpt->m_reg->ReplyFreeHostIndex,
            mpt->m_free_index);

        if (cmd->cmd_flags & CFLAG_FW_CMD)
                return;

        if (cmd->cmd_flags & CFLAG_RETRY) {
                /*
                 * The target returned QFULL or busy, do not add this
                 * pkt to the doneq since the hba will retry
                 * this cmd.
                 *
                 * The pkt has already been resubmitted in
                 * mptsas_handle_qfull() or in mptsas_check_scsi_io_error().
                 * Remove this cmd_flag here.
                 */
                cmd->cmd_flags &= ~CFLAG_RETRY;
        } else {
                mptsas_doneq_add(mpt, cmd);
        }
}

#ifdef MPTSAS_DEBUG
static uint8_t mptsas_last_sense[256];
#endif

static void
mptsas_check_scsi_io_error(mptsas_t *mpt, pMpi2SCSIIOReply_t reply,
    mptsas_cmd_t *cmd)
{
        uint8_t                 scsi_status, scsi_state;
        uint16_t                ioc_status, cmd_rqs_len;
        uint32_t                xferred, sensecount, responsedata, loginfo = 0;
        struct scsi_pkt         *pkt;
        struct scsi_arq_status  *arqstat;
        mptsas_target_t         *ptgt = cmd->cmd_tgt_addr;
        uint8_t                 *sensedata = NULL;
        uint64_t                sas_wwn;
        uint8_t                 phy;
        char                    wwn_str[MPTSAS_WWN_STRLEN];

        scsi_status = ddi_get8(mpt->m_acc_reply_frame_hdl, &reply->SCSIStatus);
        ioc_status = ddi_get16(mpt->m_acc_reply_frame_hdl, &reply->IOCStatus);
        scsi_state = ddi_get8(mpt->m_acc_reply_frame_hdl, &reply->SCSIState);
        xferred = ddi_get32(mpt->m_acc_reply_frame_hdl, &reply->TransferCount);
        sensecount = ddi_get32(mpt->m_acc_reply_frame_hdl, &reply->SenseCount);
        responsedata = ddi_get32(mpt->m_acc_reply_frame_hdl,
            &reply->ResponseInfo);

        if (ioc_status & MPI2_IOCSTATUS_FLAG_LOG_INFO_AVAILABLE) {
                sas_wwn = ptgt->m_addr.mta_wwn;
                phy = ptgt->m_phynum;
                if (sas_wwn == 0) {
                        (void) sprintf(wwn_str, "p%x", phy);
                } else {
                        (void) sprintf(wwn_str, "w%016"PRIx64, sas_wwn);
                }
                loginfo = ddi_get32(mpt->m_acc_reply_frame_hdl,
                    &reply->IOCLogInfo);
                mptsas_log(mpt, CE_NOTE,
                    "?Log info 0x%x received for target %d %s.\n"
                    "\tscsi_status=0x%x, ioc_status=0x%x, scsi_state=0x%x",
                    loginfo, Tgt(cmd), wwn_str, scsi_status, ioc_status,
```



```
                    scsi_state);
        }

        NDBG31(("\t\tscsi_status=0x%x, ioc_status=0x%x, scsi_state=0x%x",
            scsi_status, ioc_status, scsi_state));

        pkt = CMD2PKT(cmd);
        *(pkt->pkt_scbp) = scsi_status;

        if (loginfo == 0x31170000) {
                /*
                 * if loginfo PL_LOGINFO_CODE_IO_DEVICE_MISSING_DELAY_RETRY
                 * 0x31170000 comes, that means the device missing delay
                 * is in progressing, the command need retry later.
                 */
                *(pkt->pkt_scbp) = STATUS_BUSY;
                return;
        }

        if ((scsi_state & MPI2_SCSI_STATE_NO_SCSI_STATUS) &&
            ((ioc_status & MPI2_IOCSTATUS_MASK) ==
            MPI2_IOCSTATUS_SCSI_DEVICE_NOT_THERE)) {
                pkt->pkt_reason = CMD_INCOMPLETE;
                pkt->pkt_state |= STATE_GOT_BUS;
                mutex_enter(&ptgt->m_t_mutex);
                if (ptgt->m_reset_delay == 0) {
                        mptsas_set_throttle(mpt, ptgt,
                            DRAIN_THROTTLE);
                }
                mutex_exit(&ptgt->m_t_mutex);
                return;
        }

        if (scsi_state & MPI2_SCSI_STATE_RESPONSE_INFO_VALID) {
                responsedata &= 0x000000FF;
                if (responsedata & MPTSAS_SCSI_RESPONSE_CODE_TLR_OFF) {
                        mptsas_log(mpt, CE_NOTE, "Do not support the TLR\n");
                        pkt->pkt_reason = CMD_TLR_OFF;
                        return;
                }
        }


        switch (scsi_status) {
        case MPI2_SCSI_STATUS_CHECK_CONDITION:
                (void) ddi_dma_sync(mpt->m_dma_req_sense_hdl, 0, 0,
                    DDI_DMA_SYNC_FORCPU);
                pkt->pkt_resid = (cmd->cmd_dmacount - xferred);
                arqstat = (void*)(pkt->pkt_scbp);
                arqstat->sts_rqpkt_status = *((struct scsi_status *)
                    (pkt->pkt_scbp));
                pkt->pkt_state |= (STATE_GOT_BUS | STATE_GOT_TARGET |
                    STATE_SENT_CMD | STATE_GOT_STATUS | STATE_ARQ_DONE);
                if (cmd->cmd_flags & CFLAG_XARQ) {
                        pkt->pkt_state |= STATE_XARQ_DONE;
                }
                if (pkt->pkt_resid != cmd->cmd_dmacount) {
                        pkt->pkt_state |= STATE_XFERRED_DATA;
                }
                arqstat->sts_rqpkt_reason = pkt->pkt_reason;
                arqstat->sts_rqpkt_state  = pkt->pkt_state;
                arqstat->sts_rqpkt_state |= STATE_XFERRED_DATA;
                arqstat->sts_rqpkt_statistics = pkt->pkt_statistics;
                sensedata = (uint8_t *)&arqstat->sts_sensedata;
#ifdef MPTSAS_DEBUG
                bcopy((uchar_t *)cmd->cmd_arq_buf, mptsas_last_sense,
```

```
                    cmd->cmd_rqslen);
#endif
                if (cmd->cmd_extrqslen != 0) {
                        cmd_rqs_len = cmd->cmd_extrqslen;
                } else {
                        cmd_rqs_len = cmd->cmd_rqslen;
                }
                bcopy((uchar_t *)cmd->cmd_arq_buf, sensedata,
                    ((cmd->cmd_rqslen >= sensecount) ? sensecount :
                    cmd_rqs_len));
                arqstat->sts_rqpkt_resid = (cmd_rqs_len - sensecount);
                cmd->cmd_flags |= CFLAG_CMDARQ;
                /*
                 * Set proper status for pkt if autosense was valid
                 */
                if (scsi_state & MPI2_SCSI_STATE_AUTOSENSE_VALID) {
                        struct scsi_status zero_status = { 0 };
                        arqstat->sts_rqpkt_status = zero_status;
                }

                /*
                 * ASC=0x47 is parity error
                 * ASC=0x48 is initiator detected error received
                 */
                if ((scsi_sense_key(sensedata) == KEY_ABORTED_COMMAND) &&
                    ((scsi_sense_asc(sensedata) == 0x47) ||
                    (scsi_sense_asc(sensedata) == 0x48))) {
                        mptsas_log(mpt, CE_NOTE, "Aborted_command!");
                }

                /*
                 * ASC/ASCQ=0x3F/0x0E means report_luns data changed
                 * ASC/ASCQ=0x25/0x00 means invalid lun
                 */
                if (((scsi_sense_key(sensedata) == KEY_UNIT_ATTENTION) &&
                    (scsi_sense_asc(sensedata) == 0x3F) &&
                    (scsi_sense_ascq(sensedata) == 0x0E)) ||
                    ((scsi_sense_key(sensedata) == KEY_ILLEGAL_REQUEST) &&
                    (scsi_sense_asc(sensedata) == 0x25) &&
                    (scsi_sense_ascq(sensedata) == 0x00))) {
                        mptsas_topo_change_list_t *topo_node = NULL;

                        topo_node = kmem_zalloc(
                            sizeof (mptsas_topo_change_list_t),
                            KM_NOSLEEP);
                        if (topo_node == NULL) {
                                mptsas_log(mpt, CE_NOTE, "No memory"
                                    "resource for handle SAS dynamic"
                                    "reconfigure.\n");
                                break;
                        }
                        topo_node->mpt = mpt;
                        topo_node->event = MPTSAS_DR_EVENT_RECONFIG_TARGET;
                        topo_node->un.phymask = ptgt->m_addr.mta_phymask;
                        topo_node->devhdl = ptgt->m_devhdl;
                        topo_node->object = (void *)ptgt;
                        topo_node->flags = MPTSAS_TOPO_FLAG_LUN_ASSOCIATED;

                        if ((ddi_taskq_dispatch(mpt->m_dr_taskq,
                            mptsas_handle_dr,
                            (void *)topo_node,
                            DDI_NOSLEEP)) != DDI_SUCCESS) {
                                kmem_free(topo_node,
                                    sizeof (mptsas_topo_change_list_t));
                                mptsas_log(mpt, CE_NOTE, "mptsas start taskq"
                                    "for handle SAS dynamic reconfigure"
```



```
                                        "failed. \n");
                        }
                }
                break;
        case MPI2_SCSI_STATUS_GOOD:
                switch (ioc_status & MPI2_IOCSTATUS_MASK) {
                case MPI2_IOCSTATUS_SCSI_DEVICE_NOT_THERE:
                        pkt->pkt_reason = CMD_DEV_GONE;
                        pkt->pkt_state |= STATE_GOT_BUS;
                        mutex_enter(&ptgt->m_t_mutex);
                        if (ptgt->m_reset_delay == 0) {
                                mptsas_set_throttle(mpt, ptgt, DRAIN_THROTTLE);
                        }
                        mutex_exit(&ptgt->m_t_mutex);
                        NDBG31(("lost disk for target%d, command:%x",
                            Tgt(cmd), pkt->pkt_cdbp[0]));
                        break;
                case MPI2_IOCSTATUS_SCSI_DATA_OVERRUN:
                        NDBG31(("data overrun: xferred=%d", xferred));
                        NDBG31(("dmacount=%d", cmd->cmd_dmacount));
                        pkt->pkt_reason = CMD_DATA_OVR;
                        pkt->pkt_state |= (STATE_GOT_BUS | STATE_GOT_TARGET
                            | STATE_SENT_CMD | STATE_GOT_STATUS
                            | STATE_XFERRED_DATA);
                        pkt->pkt_resid = 0;
                        break;
                case MPI2_IOCSTATUS_SCSI_RESIDUAL_MISMATCH:
                case MPI2_IOCSTATUS_SCSI_DATA_UNDERRUN:
                        NDBG31(("data underrun: xferred=%d", xferred));
                        NDBG31(("dmacount=%d", cmd->cmd_dmacount));
                        pkt->pkt_state |= (STATE_GOT_BUS | STATE_GOT_TARGET
                            | STATE_SENT_CMD | STATE_GOT_STATUS);
                        pkt->pkt_resid = (cmd->cmd_dmacount - xferred);
                        if (pkt->pkt_resid != cmd->cmd_dmacount) {
                                pkt->pkt_state |= STATE_XFERRED_DATA;
                        }
                        break;
                case MPI2_IOCSTATUS_SCSI_TASK_TERMINATED:
                        if (cmd->cmd_active_expiration <= gethrtime()) {
                                /*
                                 * When timeout requested, propagate
                                 * proper reason and statistics to
                                 * target drivers.
                                 */
                                mptsas_set_pkt_reason(mpt, cmd, CMD_TIMEOUT,
                                    STAT_BUS_RESET | STAT_TIMEOUT);
                        } else {
                                mptsas_set_pkt_reason(mpt, cmd, CMD_RESET,
                                    STAT_BUS_RESET);
                        }
                        break;
                case MPI2_IOCSTATUS_SCSI_IOC_TERMINATED:
                case MPI2_IOCSTATUS_SCSI_EXT_TERMINATED:
                        mptsas_set_pkt_reason(mpt,
                            cmd, CMD_RESET, STAT_DEV_RESET);
                        break;
                case MPI2_IOCSTATUS_SCSI_IO_DATA_ERROR:
                case MPI2_IOCSTATUS_SCSI_PROTOCOL_ERROR:
                        pkt->pkt_state |= (STATE_GOT_BUS | STATE_GOT_TARGET);
                        mptsas_set_pkt_reason(mpt,
                            cmd, CMD_TERMINATED, STAT_TERMINATED);
                        break;
                case MPI2_IOCSTATUS_INSUFFICIENT_RESOURCES:
                case MPI2_IOCSTATUS_BUSY:
                        /*
                         * set throttles to drain
```

```
                         */
                        for (ptgt = refhash_first(mpt->m_targets); ptgt != NULL;
                            ptgt = refhash_next(mpt->m_targets, ptgt)) {
                                mptsas_set_throttle_mtx(mpt, ptgt,
                                    DRAIN_THROTTLE);
                        }

                        /*
                         * retry command
                         */
                        mptsas_retry_pkt(mpt, cmd);
                        break;
                default:
                        mptsas_log(mpt, CE_WARN,
                            "unknown ioc_status = %x\n", ioc_status);
                        mptsas_log(mpt, CE_CONT, "scsi_state = %x, transfer "
                            "count = %x, scsi_status = %x", scsi_state,
                            xferred, scsi_status);
                        break;
                }
                break;
        case MPI2_SCSI_STATUS_TASK_SET_FULL:
                mptsas_handle_qfull(mpt, cmd);
                break;
        case MPI2_SCSI_STATUS_BUSY:
                NDBG31(("scsi_status busy received"));
                break;
        case MPI2_SCSI_STATUS_RESERVATION_CONFLICT:
                NDBG31(("scsi_status reservation conflict received"));
                break;
        default:
                mptsas_log(mpt, CE_WARN, "scsi_status=%x, ioc_status=%x\n",
                    scsi_status, ioc_status);
                mptsas_log(mpt, CE_WARN,
                    "mptsas_process_intr: invalid scsi status\n");
                break;
        }
}

static void
mptsas_check_task_mgt(mptsas_t *mpt, pMpi2SCSIManagementReply_t reply,
        mptsas_cmd_t *cmd)
{
        uint8_t         task_type;
        uint16_t        ioc_status;
        uint32_t        log_info;
        uint16_t        dev_handle;
        struct scsi_pkt *pkt = CMD2PKT(cmd);

        task_type = ddi_get8(mpt->m_acc_reply_frame_hdl, &reply->TaskType);
        ioc_status = ddi_get16(mpt->m_acc_reply_frame_hdl, &reply->IOCStatus);
        log_info = ddi_get32(mpt->m_acc_reply_frame_hdl, &reply->IOCLogInfo);
        dev_handle = ddi_get16(mpt->m_acc_reply_frame_hdl, &reply->DevHandle);

        if (ioc_status != MPI2_IOCSTATUS_SUCCESS) {
                mptsas_log(mpt, CE_WARN, "mptsas_check_task_mgt: Task 0x%x "
                    "failed. IOCStatus=0x%x IOCLogInfo=0x%x target=%d\n",
                    task_type, ioc_status, log_info, dev_handle);
                pkt->pkt_reason = CMD_INCOMPLETE;
                return;
        }

        switch (task_type) {
        case MPI2_SCSITASKMGMT_TASKTYPE_ABORT_TASK:
        case MPI2_SCSITASKMGMT_TASKTYPE_CLEAR_TASK_SET:
        case MPI2_SCSITASKMGMT_TASKTYPE_QUERY_TASK:
```



```
        case MPI2_SCSITASKMGMT_TASKTYPE_CLR_ACA:
        case MPI2_SCSITASKMGMT_TASKTYPE_QRY_TASK_SET:
        case MPI2_SCSITASKMGMT_TASKTYPE_QRY_UNIT_ATTENTION:
                break;
        case MPI2_SCSITASKMGMT_TASKTYPE_ABRT_TASK_SET:
        case MPI2_SCSITASKMGMT_TASKTYPE_LOGICAL_UNIT_RESET:
        case MPI2_SCSITASKMGMT_TASKTYPE_TARGET_RESET:
                /*
                 * Check for invalid DevHandle of 0 in case application
                 * sends bad command.  DevHandle of 0 could cause problems.
                 */
                if (dev_handle == 0) {
                        mptsas_log(mpt, CE_WARN, "!Can't flush target with"
                            " DevHandle of 0.");
                } else {
                        mptsas_flush_target(mpt, dev_handle, Lun(cmd),
                            task_type);
                }
                break;
        default:
                mptsas_log(mpt, CE_WARN, "Unknown task management type %d.",
                    task_type);
                mptsas_log(mpt, CE_WARN, "ioc status = %x", ioc_status);
                break;
        }
}

static void
mptsas_doneq_thread(mptsas_thread_arg_t *arg)
{
        mptsas_t                        *mpt = arg->mpt;
        uint32_t                        t = arg->t;
        mptsas_cmd_t                    *cmd;
        struct scsi_pkt                 *pkt;
        mptsas_doneq_thread_list_t      *item = &mpt->m_doneq_thread_id[t];

        mutex_enter(&item->mutex);
        while (item->flag & MPTSAS_DONEQ_THREAD_ACTIVE) {
                if (!item->dlist.dl_q) {
                        cv_wait(&item->cv, &item->mutex);
                }
                pkt = NULL;
                if ((cmd = mptsas_doneq_thread_rm(mpt, t)) != NULL) {
                        cmd->cmd_flags |= CFLAG_COMPLETED;
                        pkt = CMD2PKT(cmd);
                }
                mutex_exit(&item->mutex);
                if (pkt) {
                        mptsas_pkt_comp(pkt, cmd);
                }
                mutex_enter(&item->mutex);
        }
        mutex_exit(&item->mutex);
        mutex_enter(&mpt->m_qthread_mutex);
        mpt->m_doneq_thread_n--;
        cv_broadcast(&mpt->m_qthread_cv);
        mutex_exit(&mpt->m_qthread_mutex);
}


/*
 * mpt interrupt handler.
 */
static uint_t
mptsas_intr(caddr_t arg1, caddr_t arg2)
{
```

```
        mptsas_t                        *mpt = (void *)arg1;
        mptsas_reply_pqueue_t           *rpqp;
        int                             reply_q = (int)(uintptr_t)arg2;
        pMpi2ReplyDescriptorsUnion_t    reply_desc_union;
        int                             found = 0, i, rpqidx;
        size_t                          dma_sync_len;
        off_t                           dma_sync_offset;
        uint32_t                        istat;

        NDBG18(("mptsas_intr: arg1 0x%p reply_q 0x%d", (void *)arg1, reply_q));

        rpqp = &mpt->m_rep_post_queues[reply_q];

        /*
         * If interrupts are shared by two channels then check whether this
         * interrupt is genuinely for this channel by making sure first the
         * chip is in high power state.
         */
        if ((mpt->m_options & MPTSAS_OPT_PM) &&
            (mpt->m_power_level != PM_LEVEL_D0)) {
                mpt->m_unclaimed_pm_interrupt_count++;
                return (DDI_INTR_UNCLAIMED);
        }

        istat = MPTSAS_GET_ISTAT(mpt);
        if (!(istat & MPI2_HIS_REPLY_DESCRIPTOR_INTERRUPT)) {
                NDBG18(("Interrupt bit not set, istat 0x%x", istat));
                mpt->m_unclaimed_no_interrupt_count++;
                /*
                 * Really need a good definition of when this is valid.
                 * It appears not to be if you have multiple reply post
                 * queues, there may be a better way - need LSI info.
                 * For now just count them.
                 */
#if 0
                return (DDI_INTR_UNCLAIMED);
#endif
        }

        /*
         * If polling, interrupt was triggered by some shared interrupt because
         * IOC interrupts are disabled during polling, so polling routine will
         * handle any replies.  Considering this, if polling is happening,
         * return with interrupt unclaimed.
         */
        if (mpt->m_polled_intr) {
                mptsas_log(mpt, CE_WARN,
                    "Unclaimed interrupt, rpq %d (Polling), istat 0x%x",
                    reply_q, istat);
                mpt->m_unclaimed_polled_interrupt_count++;
                return (DDI_INTR_UNCLAIMED);
        }

        /*
         * At the moment this is the only place the mutex is grabbed.
         * So it should never fail!
         */
        if (mutex_tryenter(&rpqp->rpq_mutex) == 0) {
                mutex_enter(&rpqp->rpq_mutex);
                rpqp->rpq_intr_mutexbusy++;
        }

        dma_sync_len = mpt->m_post_queue_depth * 8;
        dma_sync_offset = dma_sync_len * reply_q;
        (void) ddi_dma_sync(mpt->m_dma_post_queue_hdl,
            dma_sync_offset, dma_sync_len, DDI_DMA_SYNC_FORCPU);
```



```
        /*
         * Go around the reply queue and process each descriptor until
         * we get to the next unused one.
         * It seems to be an occupational hazard that we get interrupts
         * with nothing to do. These are counted below.
         */
        rpqidx = rpqp->rpq_index;
#ifndef __lock_lint
        _NOTE(CONSTCOND)
#endif
        while (TRUE) {
                reply_desc_union = (pMpi2ReplyDescriptorsUnion_t)
                    MPTSAS_GET_NEXT_REPLY(rpqp, rpqidx);

                if (ddi_get32(mpt->m_acc_post_queue_hdl,
                    &reply_desc_union->Words.Low) == 0xFFFFFFFF ||
                    ddi_get32(mpt->m_acc_post_queue_hdl,
                    &reply_desc_union->Words.High) == 0xFFFFFFFF) {
                        break;
                }

                found++;

                ASSERT(ddi_get8(mpt->m_acc_post_queue_hdl,
                    &reply_desc_union->Default.MSIxIndex) == reply_q);

                /*
                 * Process it according to its type.
                 */
                mptsas_process_intr(mpt, rpqp, reply_desc_union);

                /*
                 * Clear the reply descriptor for re-use.
                 */
                ddi_put64(mpt->m_acc_post_queue_hdl,
                    &((uint64_t *)(void *)rpqp->rpq_queue)[rpqidx],
                    0xFFFFFFFFFFFFFFFF);

                /*
                 * Increment post index and roll over if needed.
                 */
                if (++rpqidx == mpt->m_post_queue_depth) {
                        rpqidx = 0;
                }
        }

        if (found == 0) {
                rpqp->rpq_intr_unclaimed++;
                mutex_exit(&rpqp->rpq_mutex);
                mpt->m_unclaimed_nocmd_interrupt_count++;
                return (DDI_INTR_UNCLAIMED);
        }
        rpqp->rpq_index = rpqidx;

        rpqp->rpq_intr_count++;
        NDBG18(("mptsas_intr complete(%d), did %d loops", reply_q, found));

        (void) ddi_dma_sync(mpt->m_dma_post_queue_hdl,
            dma_sync_offset, dma_sync_len, DDI_DMA_SYNC_FORDEV);

        mpt->m_interrupt_count++;

        /*
         * Update the reply index if at least one reply was processed.
         * For more than 8 reply queues on SAS3 controllers we have to do
```

```
 * things a little different. See Chapter 20 in the MPI 2.5 spec.
 */
if (mpt->m_post_reply_qcount > 8) {
        /*
         * The offsets from the base are multiples of 0x10.
         * We are indexing into 32 bit quantities so calculate
         * the index for that.
         */
        i = (reply_q&~0x7) >> 1;
        ddi_put32(mpt->m_datap,
            &mpt->m_reg->SuppReplyPostHostIndex[i],
            rpqp->rpq_index |
            ((reply_q&0x7)<<MPI2_RPHI_MSIX_INDEX_SHIFT));
        (void) ddi_get32(mpt->m_datap,
            &mpt->m_reg->SuppReplyPostHostIndex[i]);
} else {
        ddi_put32(mpt->m_datap,
            &mpt->m_reg->ReplyPostHostIndex,
            rpqp->rpq_index | (reply_q<<MPI2_RPHI_MSIX_INDEX_SHIFT));
        (void) ddi_get32(mpt->m_datap,
            &mpt->m_reg->ReplyPostHostIndex);
}

/*
 * If no helper threads are created, process the doneq in ISR. If
 * helpers are created, use the doneq length as a metric to measure the
 * load on the interrupt CPU. If it is long enough, which indicates the
 * load is heavy, then we deliver the IO completions to the helpers.
 * This measurement has some limitations, although it is simple and
 * straightforward and works well for most of the cases at present.
 * To always use the threads set mptsas_doneq_length_threshold_prop
 * to zero in the mpt_sas3.conf file.
 *
 * Check the current reply queue done queue.
 */
if (rpqp->rpq_dlist.dl_len) {
        if (!mpt->m_doneq_thread_n ||
            (rpqp->rpq_dlist.dl_len <= mpt->m_doneq_length_threshold)) {
                mptsas_rpdoneq_empty(rpqp);
        } else {
                mptsas_deliver_doneq_thread(mpt, &rpqp->rpq_dlist);
        }
}

mutex_exit(&rpqp->rpq_mutex);

/*
 * Check the main done queue. If we find something
 * grab the mutex and check again before processing.
 */
if (mpt->m_dlist.dl_len) {
        mutex_enter(&mpt->m_mutex);
        if (mpt->m_dlist.dl_len) {
                if (!mpt->m_doneq_thread_n ||
                    (mpt->m_dlist.dl_len <=
                    mpt->m_doneq_length_threshold)) {
                        mptsas_doneq_empty(mpt);
                } else {
                        mptsas_deliver_doneq_thread(mpt, &mpt->m_dlist);
                }
        }
        mutex_exit(&mpt->m_mutex);
}

/*
 * If there are queued cmd, start them now.
```



```
 */
if (mpt->m_waitq != NULL) {
        mutex_enter(&mpt->m_mutex);
        if (mpt->m_waitq != NULL && mpt->m_polled_intr == 0) {
                mptsas_restart_waitq(mpt);
        }
        mutex_exit(&mpt->m_mutex);
}
return (DDI_INTR_CLAIMED);
}

static void
mptsas_process_intr(mptsas_t *mpt, mptsas_reply_pqueue_t *rpqp,
    pMpi2ReplyDescriptorsUnion_t reply_desc_union)
{
        uint8_t reply_type;

        /*
         * Should get here with the reply queue mutex held, but not
         * the main mpt mutex. Want to avoid grabbing that during
         * normal operations if possible.
         */
        ASSERT(mutex_owned(&rpqp->rpq_mutex));

        /*
         * The reply is valid, process it according to its
         * type.  Also, set a flag for updated the reply index
         * after they've all been processed.
         */
        reply_type = ddi_get8(mpt->m_acc_post_queue_hdl,
            &reply_desc_union->Default.ReplyFlags);
        NDBG18(("mptsas_process_intr(rpq %d) reply_type 0x%x", rpqp->rpq_num,
            reply_type));
        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);
        } else if (reply_type == MPI2_RPY_DESCRIPT_FLAGS_ADDRESS_REPLY) {
                mutex_enter(&mpt->m_mutex);
                mptsas_handle_address_reply(mpt, reply_desc_union);
                mutex_exit(&mpt->m_mutex);
        } else {
                mptsas_log(mpt, CE_WARN, "?Bad reply type %x", reply_type);
                ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_UNAFFECTED);
        }
}

/*
 * 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);
        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);

                cmd->cmd_flags &= ~(CFLAG_TRANFLAG);

                mptsas_retry_pkt(mpt, cmd);

                mutex_enter(&ptgt->m_t_mutex);
                /*
                 * 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);
}

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));

        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, ",
            mpt->m_instance, physport, phy_mask));
```



```
        return (phy_mask);
}

/*
 * mpt free device handle after device gone, by use of passthrough
 */
static int
mptsas_free_devhdl(mptsas_t *mpt, uint16_t devhdl)
{
        Mpi2SasIoUnitControlRequest_t   req;
        Mpi2SasIoUnitControlReply_t     rep;
        int                             ret;

        ASSERT(mutex_owned(&mpt->m_mutex));

        /*
         * Need to compose a SAS IO Unit Control request message
         * and call mptsas_do_passthru() function
         */
        bzero(&req, sizeof (req));
        bzero(&rep, sizeof (rep));

        req.Function = MPI2_FUNCTION_SAS_IO_UNIT_CONTROL;
        req.Operation = MPI2_SAS_OP_REMOVE_DEVICE;
        req.DevHandle = LE_16(devhdl);

        ret = mptsas_do_passthru(mpt, (uint8_t *)&req, (uint8_t *)&rep, NULL,
            sizeof (req), sizeof (rep), NULL, 0, NULL, 0, 60, FKIOCTL);
        if (ret != 0) {
                cmn_err(CE_WARN, "mptsas_free_devhdl: passthru SAS IO Unit "
                    "Control error %d", ret);
                return (DDI_FAILURE);
        }

        /* do passthrough success, check the ioc status */
        if (LE_16(rep.IOCStatus) != MPI2_IOCSTATUS_SUCCESS) {
                cmn_err(CE_WARN, "mptsas_free_devhdl: passthru SAS IO Unit "
                    "Control IOCStatus %d", LE_16(rep.IOCStatus));
                return (DDI_FAILURE);
        }

        return (DDI_SUCCESS);
}

static void
mptsas_update_phymask(mptsas_t *mpt)
{
        mptsas_phymask_t mask = 0, phy_mask;
        char             *phy_mask_name;
        uint8_t          current_port;
        int              i, j;

        NDBG20(("mptsas3%d update phymask ", mpt->m_instance));

        ASSERT(mutex_owned(&mpt->m_mutex));

        (void) mptsas_get_sas_io_unit_page(mpt);

        phy_mask_name = kmem_zalloc(MPTSAS_MAX_PHYS, KM_SLEEP);

        for (i = 0; i < mpt->m_num_phys; i++) {
                phy_mask = 0x00;

                if (mpt->m_phy_info[i].attached_devhdl == 0)
                        continue;
```

```
                bzero(phy_mask_name, sizeof (phy_mask_name));

                current_port = mpt->m_phy_info[i].port_num;

                if ((mask & (1 << i)) != 0)
                        continue;

                for (j = 0; j < mpt->m_num_phys; j++) {
                        if (mpt->m_phy_info[j].attached_devhdl &&
                            (mpt->m_phy_info[j].port_num == current_port)) {
                                phy_mask |= (1 << j);
                        }
                }
                mask = mask | phy_mask;

                for (j = 0; j < mpt->m_num_phys; j++) {
                        if ((phy_mask >> j) & 0x01) {
                                mpt->m_phy_info[j].phy_mask = phy_mask;
                        }
                }

                (void) sprintf(phy_mask_name, "%x", phy_mask);

                mutex_exit(&mpt->m_mutex);
                /*
                 * register a iport, if the port has already been existed
                 * SCSA will do nothing and just return.
                 */
                (void) scsi_hba_iport_register(mpt->m_dip, phy_mask_name);
                mutex_enter(&mpt->m_mutex);
        }
        kmem_free(phy_mask_name, MPTSAS_MAX_PHYS);
        NDBG20(("mptsas3%d update phymask return", mpt->m_instance));
}

/*
 * mptsas_handle_dr is a task handler for DR, the DR action includes:
 * 1. Directly attched Device Added/Removed.
 * 2. Expander Device Added/Removed.
 * 3. Indirectly Attached Device Added/Expander.
 * 4. LUNs of a existing device status change.
 * 5. RAID volume created/deleted.
 * 6. Member of RAID volume is released because of RAID deletion.
 * 7. Physical disks are removed because of RAID creation.
 */
static void
mptsas_handle_dr(void *args) {
        mptsas_topo_change_list_t       *topo_node = NULL;
        mptsas_topo_change_list_t       *save_node = NULL;
        mptsas_t                        *mpt;
        dev_info_t                      *parent = NULL;
        mptsas_phymask_t                phymask = 0;
        char                            phy_mask_name[MPTSAS_MAX_PHYS];
        uint8_t                         flags = 0, physport = 0xff;
        uint8_t                         port_update = 0;
        uint_t                          event;

        topo_node = (mptsas_topo_change_list_t *)args;

        mpt = topo_node->mpt;
        event = topo_node->event;
        flags = topo_node->flags;

        NDBG20(("mptsas3%d handle_dr enter", mpt->m_instance));

        switch (event) {
```



```
        case MPTSAS_DR_EVENT_RECONFIG_TARGET:
                if ((flags == MPTSAS_TOPO_FLAG_DIRECT_ATTACHED_DEVICE) ||
                    (flags == MPTSAS_TOPO_FLAG_EXPANDER_ATTACHED_DEVICE) ||
                    (flags == MPTSAS_TOPO_FLAG_RAID_PHYSDRV_ASSOCIATED)) {
                        /*
                         * Direct attached or expander attached device added
                         * into system or a Phys Disk that is being unhidden.
                         */
                        port_update = 1;
                }
                break;
        case MPTSAS_DR_EVENT_RECONFIG_SMP:
                /*
                 * New expander added into system, it must be the head
                 * of topo_change_list_t
                 */
                port_update = 1;
                break;
        default:
                port_update = 0;
                break;
        }
        /*
         * All cases port_update == 1 may cause initiator port form change
         */
        mutex_enter(&mpt->m_mutex);
        if (mpt->m_port_chng && port_update) {
                /*
                 * mpt->m_port_chng flag indicates some PHYs of initiator
                 * port have changed to online. So when expander added or
                 * directly attached device online event come, we force to
                 * update port information by issueing SAS IO Unit Page and
                 * update PHYMASKs.
                 */
                (void) mptsas_update_phymask(mpt);
                mpt->m_port_chng = 0;

        }
        mutex_exit(&mpt->m_mutex);

        while (topo_node) {
                phymask = 0;
                flags = topo_node->flags;
                event = topo_node->event;
                if (event == MPTSAS_DR_EVENT_REMOVE_HANDLE) {
                        goto handle_topo_change;
                }
                if ((event == MPTSAS_DR_EVENT_RECONFIG_TARGET) &&
                    (flags == MPTSAS_TOPO_FLAG_RAID_PHYSDRV_ASSOCIATED)) {
                        /*
                         * There is no any field in IR_CONFIG_CHANGE
                         * event indicate physport/phynum, let's get
                         * parent after SAS Device Page0 request.
                         */
                        goto handle_topo_change;
                }

                if (parent == NULL) {
                        physport = topo_node->un.physport;
                        if (event & (MPTSAS_DR_EVENT_OFFLINE_TARGET |
                            MPTSAS_DR_EVENT_OFFLINE_SMP)) {
                                /*
                                 * For all offline events, phymask is known
                                 */
                                phymask = topo_node->un.phymask;
                                goto find_parent;
```

```
                        }
                        if (flags & MPTSAS_TOPO_FLAG_LUN_ASSOCIATED) {
                                phymask = topo_node->un.phymask;
                                goto find_parent;
                        }

                        mutex_enter(&mpt->m_mutex);
                        if (flags == MPTSAS_TOPO_FLAG_DIRECT_ATTACHED_DEVICE) {
                                /*
                                 * If the direct attached device added or a
                                 * phys disk is being unhidden, argument
                                 * physport actually is PHY#, so we have to get
                                 * phymask according PHY#.
                                 */
                                physport = mpt->m_phy_info[physport].port_num;
                        }

                        /*
                         * Translate physport to phymask so that we can search
                         * parent dip.
                         */
                        phymask = mptsas_physport_to_phymask(mpt, physport);
                        mutex_exit(&mpt->m_mutex);

find_parent:
                        bzero(phy_mask_name, MPTSAS_MAX_PHYS);
                        /*
                         * For RAID topology change node, write the iport name
                         * as v0.
                         */
                        if (flags & MPTSAS_TOPO_FLAG_RAID_ASSOCIATED) {
                                (void) sprintf(phy_mask_name, "v0");
                        } else {
                                /*
                                 * phymask can be 0 if the drive has been
                                 * pulled by the time an add event is
                                 * processed.  If phymask is 0, just skip this
                                 * event and continue.
                                 */
                                if (phymask == 0) {
                                        save_node = topo_node;
                                        topo_node = topo_node->next;
                                        ASSERT(save_node);
                                        kmem_free(save_node,
                                            sizeof (mptsas_topo_change_list_t));
                                        parent = NULL;
                                        continue;
                                }
                                (void) sprintf(phy_mask_name, "%x", phymask);
                        }
                        parent = scsi_hba_iport_find(mpt->m_dip,
                            phy_mask_name);
                        if (parent == NULL) {
                                mptsas_log(mpt, CE_WARN, "Failed to find an "
                                    "iport for \"%s\", should not happen!",
                                    phy_mask_name);
                                save_node = topo_node;
                                topo_node = topo_node->next;
                                ASSERT(save_node);
                                kmem_free(save_node,
                                    sizeof (mptsas_topo_change_list_t));
                                continue;
                        }

                }
                ASSERT(parent);
```



```
handle_topo_change:

                mutex_enter(&mpt->m_mutex);
                /*
                 * If HBA is being reset, don't perform operations depending
                 * on the IOC. We must free the topo list, however.
                 */
                if (!mpt->m_in_reset)
                        mptsas_handle_topo_change(topo_node, parent);
                else
                        NDBG20(("skipping topo change received during reset"));
                mutex_exit(&mpt->m_mutex);
                save_node = topo_node;
                topo_node = topo_node->next;
                ASSERT(save_node);
                kmem_free(save_node, sizeof (mptsas_topo_change_list_t));

                if ((flags == MPTSAS_TOPO_FLAG_DIRECT_ATTACHED_DEVICE) ||
                    (flags == MPTSAS_TOPO_FLAG_RAID_PHYSDRV_ASSOCIATED) ||
                    (flags == MPTSAS_TOPO_FLAG_RAID_ASSOCIATED)) {
                        /*
                         * If direct attached device associated, make sure
                         * reset the parent before start the next one. But
                         * all devices associated with expander shares the
                         * parent.  Also, reset parent if this is for RAID.
                         */
                        parent = NULL;
                }
        }
}

static void
mptsas_offline_target(mptsas_t *mpt, mptsas_target_t *ptgt,
    uint8_t topo_flags, dev_info_t *parent)
{
        uint64_t        sas_wwn = 0;
        uint8_t         phy;
        char            wwn_str[MPTSAS_WWN_STRLEN];
        uint16_t        devhdl;
        int             circ = 0, circ1 = 0;
        int             rval = 0;

        sas_wwn = ptgt->m_addr.mta_wwn;
        phy = ptgt->m_phynum;
        devhdl = ptgt->m_devhdl;

        if (sas_wwn) {
                (void) sprintf(wwn_str, "w%016"PRIx64, sas_wwn);
        } else {
                (void) sprintf(wwn_str, "p%x", phy);
        }

        /*
         * Abort all outstanding command on the device
         */
        rval = mptsas_do_scsi_reset(mpt, devhdl);
        if (rval) {
                NDBG20(("mptsas3%d: mptsas_offline_target: reset target "
                    "before offline devhdl:%x, phymask:%x, rval:%x",
                    mpt->m_instance, ptgt->m_devhdl,
                    ptgt->m_addr.mta_phymask, rval));
        }

        mutex_exit(&mpt->m_mutex);

        ndi_devi_enter(scsi_vhci_dip, &circ);
```

```
        ndi_devi_enter(parent, &circ1);
        rval = mptsas_offline_targetdev(parent, wwn_str);
        ndi_devi_exit(parent, circ1);
        ndi_devi_exit(scsi_vhci_dip, circ);
        NDBG20(("mptsas3%d: mptsas_offline_target %s devhdl:%x, "
            "phymask:%x, rval:%x", mpt->m_instance, wwn_str,
            ptgt->m_devhdl, ptgt->m_addr.mta_phymask, rval));

        /*
         * Clear parent's props for SMHBA support
         */
        if (topo_flags == MPTSAS_TOPO_FLAG_DIRECT_ATTACHED_DEVICE) {
                if (ddi_prop_update_string(DDI_DEV_T_NONE, parent,
                    SCSI_ADDR_PROP_ATTACHED_PORT, "") !=
                    DDI_PROP_SUCCESS) {
                        (void) ddi_prop_remove(DDI_DEV_T_NONE, parent,
                            SCSI_ADDR_PROP_ATTACHED_PORT);
                        mptsas_log(mpt, CE_WARN, "mptsas attached port "
                            "prop update failed");
                }
                if (ddi_prop_update_int(DDI_DEV_T_NONE, parent,
                    MPTSAS_NUM_PHYS, 0) != DDI_PROP_SUCCESS) {
                        (void) ddi_prop_remove(DDI_DEV_T_NONE, parent,
                            MPTSAS_NUM_PHYS);
                        mptsas_log(mpt, CE_WARN, "mptsas num phys "
                            "prop update failed");
                }
                if (ddi_prop_update_int(DDI_DEV_T_NONE, parent,
                    MPTSAS_VIRTUAL_PORT, 1) != DDI_PROP_SUCCESS) {
                        (void) ddi_prop_remove(DDI_DEV_T_NONE, parent,
                            MPTSAS_VIRTUAL_PORT);
                        mptsas_log(mpt, CE_WARN, "mptsas virtual port "
                            "prop update failed");
                }
        }

        mutex_enter(&mpt->m_mutex);
        ptgt->m_led_status = 0;
        (void) mptsas_flush_led_status(mpt, ptgt);
        if (rval == DDI_SUCCESS) {
                mutex_destroy(&ptgt->m_t_mutex);
                refhash_remove(mpt->m_targets, ptgt);
                ptgt = NULL;
        } else {
                /*
                 * clean DR_INTRANSITION flag to allow I/O down to
                 * PHCI driver since failover finished.
                 * Invalidate the devhdl
                 */
                ptgt->m_devhdl = MPTSAS_INVALID_DEVHDL;
                ptgt->m_tgt_unconfigured = 0;
                ptgt->m_dr_flag = MPTSAS_DR_INACTIVE;
        }
}

static void
mptsas_handle_topo_change(mptsas_topo_change_list_t *topo_node,
    dev_info_t *parent)
{
        mptsas_target_t *ptgt = NULL;
        mptsas_smp_t    *psmp = NULL;
        mptsas_t        *mpt = (void *)topo_node->mpt;
        uint16_t        devhdl;
        uint16_t        attached_devhdl;
        int             rval = 0;
        uint32_t        page_address;
```



```
        uint8_t         flags;
        dev_info_t      *lundip;
        int             circ = 0, circ1 = 0;
        char            attached_wwnstr[MPTSAS_WWN_STRLEN];

        NDBG20(("mptsas3%d handle_topo_change enter, devhdl 0x%x,"
            "event 0x%x, flags 0x%x", mpt->m_instance, topo_node->devhdl,
            topo_node->event, topo_node->flags));

        ASSERT(mutex_owned(&mpt->m_mutex));

        switch (topo_node->event) {
        case MPTSAS_DR_EVENT_RECONFIG_TARGET:
        {
                char *phy_mask_name;
                mptsas_phymask_t phymask = 0;

                if (topo_node->flags == MPTSAS_TOPO_FLAG_RAID_ASSOCIATED) {
                        /*
                         * Get latest RAID info.
                         */
                        (void) mptsas_get_raid_info(mpt);
                        ptgt = refhash_linear_search(mpt->m_targets,
                            mptsas_target_eval_devhdl, &topo_node->devhdl);
                        if (ptgt == NULL)
                                break;
                } else {
                        ptgt = (void *)topo_node->object;
                }

                if (ptgt == NULL) {
                        /*
                         * If a Phys Disk was deleted, RAID info needs to be
                         * updated to reflect the new topology.
                         */
                        (void) mptsas_get_raid_info(mpt);

                        /*
                         * Get sas device page 0 by DevHandle to make sure if
                         * SSP/SATA end device exist.
                         */
                        page_address = (MPI2_SAS_DEVICE_PGAD_FORM_HANDLE &
                            MPI2_SAS_DEVICE_PGAD_FORM_MASK) |
                            topo_node->devhdl;

                        rval = mptsas_get_target_device_info(mpt, page_address,
                            &devhdl, &ptgt);
                        if (rval == DEV_INFO_WRONG_DEVICE_TYPE) {
                                mptsas_log(mpt, CE_NOTE,
                                    "mptsas_handle_topo_change: target %d is "
                                    "not a SAS/SATA device. \n",
                                    topo_node->devhdl);
                        } else if (rval == DEV_INFO_FAIL_ALLOC) {
                                mptsas_log(mpt, CE_NOTE,
                                    "mptsas_handle_topo_change: could not "
                                    "allocate memory. \n");
                        }
                        /*
                         * If rval is DEV_INFO_PHYS_DISK than there is nothing
                         * else to do, just leave.
                         */
                        if (rval != DEV_INFO_SUCCESS) {
                                return;
                        }
                }
```

```
                ASSERT(ptgt->m_devhdl == topo_node->devhdl);

                mutex_exit(&mpt->m_mutex);
                flags = topo_node->flags;

                if (flags == MPTSAS_TOPO_FLAG_RAID_PHYSDRV_ASSOCIATED) {
                        phymask = ptgt->m_addr.mta_phymask;
                        phy_mask_name = kmem_zalloc(MPTSAS_MAX_PHYS, KM_SLEEP);
                        (void) sprintf(phy_mask_name, "%x", phymask);
                        parent = scsi_hba_iport_find(mpt->m_dip,
                            phy_mask_name);
                        kmem_free(phy_mask_name, MPTSAS_MAX_PHYS);
                        if (parent == NULL) {
                                mptsas_log(mpt, CE_WARN, "Failed to find a "
                                    "iport for PD, should not happen!");
                                mutex_enter(&mpt->m_mutex);
                                break;
                        }
                }

                if (flags == MPTSAS_TOPO_FLAG_RAID_ASSOCIATED) {
                        ndi_devi_enter(parent, &circ1);
                        (void) mptsas_config_raid(parent, topo_node->devhdl,
                            &lundip);
                        ndi_devi_exit(parent, circ1);
                } else {
                        /*
                         * hold nexus for bus configure
                         */
                        ndi_devi_enter(scsi_vhci_dip, &circ);
                        ndi_devi_enter(parent, &circ1);
                        rval = mptsas_config_target(parent, ptgt);
                        /*
                         * release nexus for bus configure
                         */
                        ndi_devi_exit(parent, circ1);
                        ndi_devi_exit(scsi_vhci_dip, circ);

                        /*
                         * Add parent's props for SMHBA support
                         */
                        if (flags == MPTSAS_TOPO_FLAG_DIRECT_ATTACHED_DEVICE) {
                                bzero(attached_wwnstr,
                                    sizeof (attached_wwnstr));
                                (void) sprintf(attached_wwnstr, "w%016"PRIx64,
                                    ptgt->m_addr.mta_wwn);
                                if (ddi_prop_update_string(DDI_DEV_T_NONE,
                                    parent,
                                    SCSI_ADDR_PROP_ATTACHED_PORT,
                                    attached_wwnstr)
                                    != DDI_PROP_SUCCESS) {
                                        (void) ddi_prop_remove(DDI_DEV_T_NONE,
                                            parent,
                                            SCSI_ADDR_PROP_ATTACHED_PORT);
                                        mptsas_log(mpt, CE_WARN, "Failed to"
                                            "attached-port props");
                                        return;
                                }
                                if (ddi_prop_update_int(DDI_DEV_T_NONE, parent,
                                    MPTSAS_NUM_PHYS, 1) !=
                                    DDI_PROP_SUCCESS) {
                                        (void) ddi_prop_remove(DDI_DEV_T_NONE,
                                            parent, MPTSAS_NUM_PHYS);
                                        mptsas_log(mpt, CE_WARN, "Failed to"
                                            " create num-phys props");
                                        return;
```



```
                                }

                                /*
                                 * Update PHY info for smhba
                                 */
                                mutex_enter(&mpt->m_mutex);
                                if (mptsas_smhba_phy_init(mpt)) {
                                        mutex_exit(&mpt->m_mutex);
                                        mptsas_log(mpt, CE_WARN, "mptsas phy"
                                            " update failed");
                                        return;
                                }
                                mutex_exit(&mpt->m_mutex);

                                /*
                                 * topo_node->un.physport is really the PHY#
                                 * for direct attached devices
                                 */
                                mptsas_smhba_set_one_phy_props(mpt, parent,
                                    topo_node->un.physport, &attached_devhdl);

                                if (ddi_prop_update_int(DDI_DEV_T_NONE, parent,
                                    MPTSAS_VIRTUAL_PORT, 0) !=
                                    DDI_PROP_SUCCESS) {
                                        (void) ddi_prop_remove(DDI_DEV_T_NONE,
                                            parent, MPTSAS_VIRTUAL_PORT);
                                        mptsas_log(mpt, CE_WARN,
                                            "mptsas virtual-port"
                                            "port prop update failed");
                                        return;
                                }
                        }
                }
                mutex_enter(&mpt->m_mutex);

                NDBG20(("mptsas3%d handle_topo_change to online devhdl:%x, "
                    "phymask:%x.", mpt->m_instance, ptgt->m_devhdl,
                    ptgt->m_addr.mta_phymask));
                break;
        }
        case MPTSAS_DR_EVENT_OFFLINE_TARGET:
        {
                devhdl = topo_node->devhdl;
                ptgt = refhash_linear_search(mpt->m_targets,
                    mptsas_target_eval_devhdl, &devhdl);
                if (ptgt == NULL)
                        break;

                ASSERT(ptgt->m_devhdl == devhdl);

                if ((topo_node->flags == MPTSAS_TOPO_FLAG_RAID_ASSOCIATED) ||
                    (topo_node->flags ==
                    MPTSAS_TOPO_FLAG_RAID_PHYSDRV_ASSOCIATED)) {
                        /*
                         * Get latest RAID info if RAID volume status changes
                         * or Phys Disk status changes
                         */
                        (void) mptsas_get_raid_info(mpt);
                }

                mptsas_offline_target(mpt, ptgt, topo_node->flags, parent);

                /*
                 * Send SAS IO Unit Control to free the dev handle
                 */
                if ((flags == MPTSAS_TOPO_FLAG_DIRECT_ATTACHED_DEVICE) ||
```

```
                    (flags == MPTSAS_TOPO_FLAG_EXPANDER_ATTACHED_DEVICE)) {
                        rval = mptsas_free_devhdl(mpt, devhdl);

                        NDBG20(("mptsas3%d handle_topo_change to remove "
                            "devhdl:%x, rval:%x", mpt->m_instance, devhdl,
                            rval));
                }

                break;
        }
        case MPTSAS_DR_EVENT_REMOVE_HANDLE:
        {
                devhdl = topo_node->devhdl;

                /*
                 * Do a reset first.
                 */
                rval = mptsas_do_scsi_reset(mpt, devhdl);
                NDBG20(("mpt%d reset target before remove "
                    "devhdl:%x, rval:%x", mpt->m_instance, devhdl, rval));

                /*
                 * Send SAS IO Unit Control to free the dev handle
                 */
                rval = mptsas_free_devhdl(mpt, devhdl);
                NDBG20(("mptsas3%d handle_topo_change to remove "
                    "devhdl:%x, rval:%x", mpt->m_instance, devhdl,
                    rval));
                break;
        }
        case MPTSAS_DR_EVENT_RECONFIG_SMP:
        {
                mptsas_smp_t smp;
                dev_info_t *smpdip;

                devhdl = topo_node->devhdl;

                page_address = (MPI2_SAS_EXPAND_PGAD_FORM_HNDL &
                    MPI2_SAS_EXPAND_PGAD_FORM_MASK) | (uint32_t)devhdl;
                rval = mptsas_get_sas_expander_page0(mpt, page_address, &smp);
                if (rval != DDI_SUCCESS) {
                        mptsas_log(mpt, CE_WARN, "failed to online smp, "
                            "handle %x", devhdl);
                        return;
                }

                psmp = mptsas_smp_alloc(mpt, &smp);
                if (psmp == NULL) {
                        return;
                }

                mutex_exit(&mpt->m_mutex);
                ndi_devi_enter(parent, &circ1);
                (void) mptsas_online_smp(parent, psmp, &smpdip);
                ndi_devi_exit(parent, circ1);

                mutex_enter(&mpt->m_mutex);
                break;
        }
        case MPTSAS_DR_EVENT_OFFLINE_SMP:
        {
                devhdl = topo_node->devhdl;
                uint32_t dev_info;

                psmp = refhash_linear_search(mpt->m_smp_targets,
                    mptsas_smp_eval_devhdl, &devhdl);
```



```
                if (psmp == NULL)
                        break;
                /*
                 * The mptsas_smp_t data is released only if the dip is offlined
                 * successfully.
                 */
                mutex_exit(&mpt->m_mutex);

                ndi_devi_enter(parent, &circ1);
                rval = mptsas_offline_smp(parent, psmp, NDI_DEVI_REMOVE);
                ndi_devi_exit(parent, circ1);

                dev_info = psmp->m_deviceinfo;
                if ((dev_info & DEVINFO_DIRECT_ATTACHED) ==
                    DEVINFO_DIRECT_ATTACHED) {
                        if (ddi_prop_update_int(DDI_DEV_T_NONE, parent,
                            MPTSAS_VIRTUAL_PORT, 1) !=
                            DDI_PROP_SUCCESS) {
                                (void) ddi_prop_remove(DDI_DEV_T_NONE, parent,
                                    MPTSAS_VIRTUAL_PORT);
                                mptsas_log(mpt, CE_WARN, "mptsas virtual port "
                                    "prop update failed");
                                return;
                        }
                        /*
                         * Check whether the smp connected to the iport,
                         */
                        if (ddi_prop_update_int(DDI_DEV_T_NONE, parent,
                            MPTSAS_NUM_PHYS, 0) !=
                            DDI_PROP_SUCCESS) {
                                (void) ddi_prop_remove(DDI_DEV_T_NONE, parent,
                                    MPTSAS_NUM_PHYS);
                                mptsas_log(mpt, CE_WARN, "mptsas num phys"
                                    "prop update failed");
                                return;
                        }
                        /*
                         * Clear parent's attached-port props
                         */
                        bzero(attached_wwnstr, sizeof (attached_wwnstr));
                        if (ddi_prop_update_string(DDI_DEV_T_NONE, parent,
                            SCSI_ADDR_PROP_ATTACHED_PORT, attached_wwnstr) !=
                            DDI_PROP_SUCCESS) {
                                (void) ddi_prop_remove(DDI_DEV_T_NONE, parent,
                                    SCSI_ADDR_PROP_ATTACHED_PORT);
                                mptsas_log(mpt, CE_WARN, "mptsas attached port "
                                    "prop update failed");
                                return;
                        }
                }

                mutex_enter(&mpt->m_mutex);
                NDBG20(("mptsas3%d handle_topo_change to remove devhdl:%x, "
                    "rval:%x", mpt->m_instance, psmp->m_devhdl, rval));
                if (rval == DDI_SUCCESS) {
                        refhash_remove(mpt->m_smp_targets, psmp);
                } else {
                        psmp->m_devhdl = MPTSAS_INVALID_DEVHDL;
                }

                bzero(attached_wwnstr, sizeof (attached_wwnstr));

                break;
        }
        default:
                return;
```

```
        }
}

/*
 * Record the event if its type is enabled in mpt instance by ioctl.
 */
static void
mptsas_record_event(void *args)
{
        m_replyh_arg_t                  *replyh_arg;
        pMpi2EventNotificationReply_t   eventreply;
        uint32_t                        event, rfm;
        mptsas_t                        *mpt;
        int                             i, j;
        uint16_t                        event_data_len;
        boolean_t                       sendAEN = FALSE;

        replyh_arg = (m_replyh_arg_t *)args;
        rfm = replyh_arg->rfm;
        mpt = replyh_arg->mpt;

        eventreply = (pMpi2EventNotificationReply_t)
            (mpt->m_reply_frame + (rfm -
            (mpt->m_reply_frame_dma_addr&0xffffffffu)));
        event = ddi_get16(mpt->m_acc_reply_frame_hdl, &eventreply->Event);


        /*
         * Generate a system event to let anyone who cares know that a
         * LOG_ENTRY_ADDED event has occurred.  This is sent no matter what the
         * event mask is set to.
         */
        if (event == MPI2_EVENT_LOG_ENTRY_ADDED) {
                sendAEN = TRUE;
        }

        /*
         * Record the event only if it is not masked.  Determine which dword
         * and bit of event mask to test.
         */
        i = (uint8_t)(event / 32);
        j = (uint8_t)(event % 32);
        if ((i < 4) && ((1 << j) & mpt->m_event_mask[i])) {
                i = mpt->m_event_index;
                mpt->m_events[i].Type = event;
                mpt->m_events[i].Number = ++mpt->m_event_number;
                bzero(mpt->m_events[i].Data, MPTSAS_MAX_EVENT_DATA_LENGTH * 4);
                event_data_len = ddi_get16(mpt->m_acc_reply_frame_hdl,
                    &eventreply->EventDataLength);

                if (event_data_len > 0) {
                        /*
                         * Limit data to size in m_event entry
                         */
                        if (event_data_len > MPTSAS_MAX_EVENT_DATA_LENGTH) {
                                event_data_len = MPTSAS_MAX_EVENT_DATA_LENGTH;
                        }
                        for (j = 0; j < event_data_len; j++) {
                                mpt->m_events[i].Data[j] =
                                    ddi_get32(mpt->m_acc_reply_frame_hdl,
                                    &(eventreply->EventData[j]));
                        }

                        /*
                         * check for index wrap-around
                         */
```



```
                        if (++i == MPTSAS_EVENT_QUEUE_SIZE) {
                                i = 0;
                        }
                        mpt->m_event_index = (uint8_t)i;

                        /*
                         * Set flag to send the event.
                         */
                        sendAEN = TRUE;
                }
        }

        /*
         * Generate a system event if flag is set to let anyone who cares know
         * that an event has occurred.
         */
        if (sendAEN) {
                (void) ddi_log_sysevent(mpt->m_dip, DDI_VENDOR_LSI, "MPT_SAS",
                    "SAS", NULL, NULL, DDI_NOSLEEP);
        }
}

#define SMP_RESET_IN_PROGRESS MPI2_EVENT_SAS_TOPO_LR_SMP_RESET_IN_PROGRESS
/*
 * handle sync events from ioc in interrupt
 * return value:
 * DDI_SUCCESS: The event is handled by this func
 * DDI_FAILURE: Event is not handled
 */
static int
mptsas_handle_event_sync(void *args)
{
        m_replyh_arg_t                  *replyh_arg;
        pMpi2EventNotificationReply_t   eventreply;
        uint32_t                        event, rfm;
        mptsas_t                        *mpt;
        uint_t                          iocstatus;

        replyh_arg = (m_replyh_arg_t *)args;
        rfm = replyh_arg->rfm;
        mpt = replyh_arg->mpt;

        ASSERT(mutex_owned(&mpt->m_mutex));

        eventreply = (pMpi2EventNotificationReply_t)
            (mpt->m_reply_frame + (rfm -
            (mpt->m_reply_frame_dma_addr&0xffffffffu)));
        event = ddi_get16(mpt->m_acc_reply_frame_hdl, &eventreply->Event);

        if (iocstatus = ddi_get16(mpt->m_acc_reply_frame_hdl,
            &eventreply->IOCStatus)) {
                if (iocstatus == MPI2_IOCSTATUS_FLAG_LOG_INFO_AVAILABLE) {
                        mptsas_log(mpt, CE_WARN,
                            "!mptsas_handle_event_sync: event 0x%x, "
                            "IOCStatus=0x%x, "
                            "IOCLogInfo=0x%x", event, iocstatus,
                            ddi_get32(mpt->m_acc_reply_frame_hdl,
                            &eventreply->IOCLogInfo));
                } else {
                        mptsas_log(mpt, CE_WARN,
                            "mptsas_handle_event_sync: event 0x%x, "
                            "IOCStatus=0x%x, "
                            "(IOCLogInfo=0x%x)", event, iocstatus,
                            ddi_get32(mpt->m_acc_reply_frame_hdl,
                            &eventreply->IOCLogInfo));
                }
```

```
        }

        /*
         * figure out what kind of event we got and handle accordingly
         */
        switch (event) {
        case MPI2_EVENT_SAS_TOPOLOGY_CHANGE_LIST:
        {
                pMpi2EventDataSasTopologyChangeList_t   sas_topo_change_list;
                uint8_t                         num_entries, expstatus, phy;
                uint8_t                         phystatus, physport, state, i;
                uint8_t                         start_phy_num, link_rate;
                uint16_t                        dev_handle, reason_code;
                uint16_t                        enc_handle, expd_handle;
                char                            string[80], curr[80], prev[80];
                mptsas_topo_change_list_t       *topo_head = NULL;
                mptsas_topo_change_list_t       *topo_tail = NULL;
                mptsas_topo_change_list_t       *topo_node = NULL;
                mptsas_target_t                 *ptgt;
                mptsas_smp_t                    *psmp;
                uint8_t                         flags = 0, exp_flag;
                smhba_info_t                    *pSmhba = NULL;

                NDBG20(("mptsas_handle_event_sync: SAS topology change"));

                sas_topo_change_list = (pMpi2EventDataSasTopologyChangeList_t)
                    eventreply->EventData;

                enc_handle = ddi_get16(mpt->m_acc_reply_frame_hdl,
                    &sas_topo_change_list->EnclosureHandle);
                expd_handle = ddi_get16(mpt->m_acc_reply_frame_hdl,
                    &sas_topo_change_list->ExpanderDevHandle);
                num_entries = ddi_get8(mpt->m_acc_reply_frame_hdl,
                    &sas_topo_change_list->NumEntries);
                start_phy_num = ddi_get8(mpt->m_acc_reply_frame_hdl,
                    &sas_topo_change_list->StartPhyNum);
                expstatus = ddi_get8(mpt->m_acc_reply_frame_hdl,
                    &sas_topo_change_list->ExpStatus);
                physport = ddi_get8(mpt->m_acc_reply_frame_hdl,
                    &sas_topo_change_list->PhysicalPort);

                string[0] = 0;
                if (expd_handle) {
                        flags = MPTSAS_TOPO_FLAG_EXPANDER_ASSOCIATED;
                        switch (expstatus) {
                        case MPI2_EVENT_SAS_TOPO_ES_ADDED:
                                (void) sprintf(string, " added");
                                /*
                                 * New expander device added
                                 */
                                mpt->m_port_chng = 1;
                                topo_node = kmem_zalloc(
                                    sizeof (mptsas_topo_change_list_t),
                                    KM_SLEEP);
                                topo_node->mpt = mpt;
                                topo_node->event = MPTSAS_DR_EVENT_RECONFIG_SMP;
                                topo_node->un.physport = physport;
                                topo_node->devhdl = expd_handle;
                                topo_node->flags = flags;
                                topo_node->object = NULL;
                                if (topo_head == NULL) {
                                        topo_head = topo_tail = topo_node;
                                } else {
                                        topo_tail->next = topo_node;
                                        topo_tail = topo_node;
                                }
```



```
                                break;
                        case MPI2_EVENT_SAS_TOPO_ES_NOT_RESPONDING:
                                (void) sprintf(string, " not responding, "
                                    "removed");
                                psmp = refhash_linear_search(mpt->m_smp_targets,
                                    mptsas_smp_eval_devhdl, &expd_handle);
                                if (psmp == NULL)
                                        break;

                                topo_node = kmem_zalloc(
                                    sizeof (mptsas_topo_change_list_t),
                                    KM_SLEEP);
                                topo_node->mpt = mpt;
                                topo_node->un.phymask =
                                    psmp->m_addr.mta_phymask;
                                topo_node->event = MPTSAS_DR_EVENT_OFFLINE_SMP;
                                topo_node->devhdl = expd_handle;
                                topo_node->flags = flags;
                                topo_node->object = NULL;
                                if (topo_head == NULL) {
                                        topo_head = topo_tail = topo_node;
                                } else {
                                        topo_tail->next = topo_node;
                                        topo_tail = topo_node;
                                }
                                break;
                        case MPI2_EVENT_SAS_TOPO_ES_RESPONDING:
                                break;
                        case MPI2_EVENT_SAS_TOPO_ES_DELAY_NOT_RESPONDING:
                                (void) sprintf(string, " not responding, "
                                    "delaying removal");
                                break;
                        default:
                                break;
                        }
                } else {
                        flags = MPTSAS_TOPO_FLAG_DIRECT_ATTACHED_DEVICE;
                }

                NDBG20(("SAS TOPOLOGY CHANGE for enclosure %x expander %x%s\n",
                    enc_handle, expd_handle, string));
                for (i = 0; i < num_entries; i++) {
                        phy = i + start_phy_num;
                        phystatus = ddi_get8(mpt->m_acc_reply_frame_hdl,
                            &sas_topo_change_list->PHY[i].PhyStatus);
                        dev_handle = ddi_get16(mpt->m_acc_reply_frame_hdl,
                            &sas_topo_change_list->PHY[i].AttachedDevHandle);
                        reason_code = phystatus & MPI2_EVENT_SAS_TOPO_RC_MASK;
                        /*
                         * Filter out processing of Phy Vacant Status unless
                         * the reason code is "Not Responding".  Process all
                         * other combinations of Phy Status and Reason Codes.
                         */
                        if ((phystatus &
                            MPI2_EVENT_SAS_TOPO_PHYSTATUS_VACANT) &&
                            (reason_code !=
                            MPI2_EVENT_SAS_TOPO_RC_TARG_NOT_RESPONDING)) {
                                continue;
                        }
                        curr[0] = 0;
                        prev[0] = 0;
                        string[0] = 0;
                        switch (reason_code) {
                        case MPI2_EVENT_SAS_TOPO_RC_TARG_ADDED:
                        {
                                NDBG20(("mptsas3%d phy %d physical_port %d "
```

```
                                        "dev_handle %d added", mpt->m_instance, phy,
                                        physport, dev_handle));
                                link_rate = ddi_get8(mpt->m_acc_reply_frame_hdl,
                                        &sas_topo_change_list->PHY[i].LinkRate);
                                state = (link_rate &
                                        MPI2_EVENT_SAS_TOPO_LR_CURRENT_MASK) >>
                                        MPI2_EVENT_SAS_TOPO_LR_CURRENT_SHIFT;
                                switch (state) {
                                case MPI2_EVENT_SAS_TOPO_LR_PHY_DISABLED:
                                        (void) sprintf(curr, "is disabled");
                                        break;
                                case MPI2_EVENT_SAS_TOPO_LR_NEGOTIATION_FAILED:
                                        (void) sprintf(curr, "is offline, "
                                            "failed speed negotiation");
                                        break;
                                case MPI2_EVENT_SAS_TOPO_LR_SATA_OOB_COMPLETE:
                                        (void) sprintf(curr, "SATA OOB "
                                            "complete");
                                        break;
                                case SMP_RESET_IN_PROGRESS:
                                        (void) sprintf(curr, "SMP reset in "
                                            "progress");
                                        break;
                                case MPI2_EVENT_SAS_TOPO_LR_RATE_1_5:
                                        (void) sprintf(curr, "is online at "
                                            "1.5 Gbps");
                                        break;
                                case MPI2_EVENT_SAS_TOPO_LR_RATE_3_0:
                                        (void) sprintf(curr, "is online at 3.0 "
                                            "Gbps");
                                        break;
                                case MPI2_EVENT_SAS_TOPO_LR_RATE_6_0:
                                        (void) sprintf(curr, "is online at 6.0 "
                                            "Gbps");
                                        break;
                                case MPI25_EVENT_SAS_TOPO_LR_RATE_12_0:
                                        (void) sprintf(curr,
                                            "is online at 12.0 Gbps");
                                        break;
                                default:
                                        (void) sprintf(curr, "state is "
                                            "unknown");
                                        break;
                                }
                                /*
                                 * New target device added into the system.
                                 * Set association flag according to if an
                                 * expander is used or not.
                                 */
                                exp_flag =
                                        MPTSAS_TOPO_FLAG_EXPANDER_ATTACHED_DEVICE;
                                if (flags ==
                                        MPTSAS_TOPO_FLAG_EXPANDER_ASSOCIATED) {
                                        flags = exp_flag;
                                }
                                topo_node = kmem_zalloc(
                                        sizeof (mptsas_topo_change_list_t),
                                        KM_SLEEP);
                                topo_node->mpt = mpt;
                                topo_node->event =
                                        MPTSAS_DR_EVENT_RECONFIG_TARGET;
                                if (expd_handle == 0) {
                                        /*
                                         * Per MPI 2, if expander dev handle
                                         * is 0, it's a directly attached
                                         * device. So driver use PHY to decide
```



```
                                         * which iport is associated
                                         */
                                        physport = phy;
                                        mpt->m_port_chng = 1;
                                }
                                topo_node->un.physport = physport;
                                topo_node->devhdl = dev_handle;
                                topo_node->flags = flags;
                                topo_node->object = NULL;
                                if (topo_head == NULL) {
                                        topo_head = topo_tail = topo_node;
                                } else {
                                        topo_tail->next = topo_node;
                                        topo_tail = topo_node;
                                }
                                break;
                        }
                        case MPI2_EVENT_SAS_TOPO_RC_TARG_NOT_RESPONDING:
                        {
                                NDBG20(("mptsas3%d phy %d physical_port %d "
                                    "dev_handle %d removed", mpt->m_instance,
                                    phy, physport, dev_handle));
                                /*
                                 * Set association flag according to if an
                                 * expander is used or not.
                                 */
                                exp_flag =
                                    MPTSAS_TOPO_FLAG_EXPANDER_ATTACHED_DEVICE;
                                if (flags ==
                                    MPTSAS_TOPO_FLAG_EXPANDER_ASSOCIATED) {
                                        flags = exp_flag;
                                }
                                /*
                                 * Target device is removed from the system
                                 * Before the device is really offline from
                                 * from system.
                                 */
                                ptgt = refhash_linear_search(mpt->m_targets,
                                    mptsas_target_eval_devhdl, &dev_handle);
                                /*
                                 * If ptgt is NULL here, it means that the
                                 * DevHandle is not in the hash table.  This is
                                 * reasonable sometimes.  For example, if a
                                 * disk was pulled, then added, then pulled
                                 * again, the disk will not have been put into
                                 * the hash table because the add event will
                                 * have an invalid phymask.  BUT, this does not
                                 * mean that the DevHandle is invalid.  The
                                 * controller will still have a valid DevHandle
                                 * that must be removed.  To do this, use the
                                 * MPTSAS_DR_EVENT_REMOVE_HANDLE event.
                                 */
                                if (ptgt == NULL) {
                                        topo_node = kmem_zalloc(
                                            sizeof (mptsas_topo_change_list_t),
                                            KM_SLEEP);
                                        topo_node->mpt = mpt;
                                        topo_node->un.phymask = 0;
                                        topo_node->event =
                                            MPTSAS_DR_EVENT_REMOVE_HANDLE;
                                        topo_node->devhdl = dev_handle;
                                        topo_node->flags = flags;
                                        topo_node->object = NULL;
                                        if (topo_head == NULL) {
                                                topo_head = topo_tail =
                                                    topo_node;
```

```
                                        } else {
                                                topo_tail->next = topo_node;
                                                topo_tail = topo_node;
                                        }
                                        break;
                                }

                                /*
                                 * Update DR flag immediately avoid I/O failure
                                 * before failover finish. We won't add
                                 * any following commands into waitq, instead,
                                 * we need return TRAN_BUSY in the tran_start
                                 * context.
                                 */
                                ptgt->m_dr_flag = MPTSAS_DR_INTRANSITION;

                                topo_node = kmem_zalloc(
                                    sizeof (mptsas_topo_change_list_t),
                                    KM_SLEEP);
                                topo_node->mpt = mpt;
                                topo_node->un.phymask =
                                    ptgt->m_addr.mta_phymask;
                                topo_node->event =
                                    MPTSAS_DR_EVENT_OFFLINE_TARGET;
                                topo_node->devhdl = dev_handle;
                                topo_node->flags = flags;
                                topo_node->object = NULL;
                                if (topo_head == NULL) {
                                        topo_head = topo_tail = topo_node;
                                } else {
                                        topo_tail->next = topo_node;
                                        topo_tail = topo_node;
                                }
                                break;
                        }
                        case MPI2_EVENT_SAS_TOPO_RC_PHY_CHANGED:
                                link_rate = ddi_get8(mpt->m_acc_reply_frame_hdl,
                                    &sas_topo_change_list->PHY[i].LinkRate);
                                state = (link_rate &
                                    MPI2_EVENT_SAS_TOPO_LR_CURRENT_MASK) >>
                                    MPI2_EVENT_SAS_TOPO_LR_CURRENT_SHIFT;
                                pSmhba = &mpt->m_phy_info[i].smhba_info;
                                pSmhba->negotiated_link_rate = state;
                                switch (state) {
                                case MPI2_EVENT_SAS_TOPO_LR_PHY_DISABLED:
                                        (void) sprintf(curr, "is disabled");
                                        mptsas_smhba_log_sysevent(mpt,
                                            ESC_SAS_PHY_EVENT,
                                            SAS_PHY_REMOVE,
                                            &mpt->m_phy_info[i].smhba_info);
                                        mpt->m_phy_info[i].smhba_info.
                                            negotiated_link_rate
                                            = 0x1;
                                        break;
                                case MPI2_EVENT_SAS_TOPO_LR_NEGOTIATION_FAILED:
                                        (void) sprintf(curr, "is offline, "
                                            "failed speed negotiation");
                                        mptsas_smhba_log_sysevent(mpt,
                                            ESC_SAS_PHY_EVENT,
                                            SAS_PHY_OFFLINE,
                                            &mpt->m_phy_info[i].smhba_info);
                                        break;
                                case MPI2_EVENT_SAS_TOPO_LR_SATA_OOB_COMPLETE:
                                        (void) sprintf(curr, "SATA OOB "
                                            "complete");
                                        break;
```



```
                                case SMP_RESET_IN_PROGRESS:
                                        (void) sprintf(curr, "SMP reset in "
                                            "progress");
                                        break;
                                case MPI2_EVENT_SAS_TOPO_LR_RATE_1_5:
                                        (void) sprintf(curr, "is online at "
                                            "1.5 Gbps");
                                        if ((expd_handle == 0) &&
                                            (enc_handle == 1)) {
                                                mpt->m_port_chng = 1;
                                        }
                                        mptsas_smhba_log_sysevent(mpt,
                                            ESC_SAS_PHY_EVENT,
                                            SAS_PHY_ONLINE,
                                            &mpt->m_phy_info[i].smhba_info);
                                        break;
                                case MPI2_EVENT_SAS_TOPO_LR_RATE_3_0:
                                        (void) sprintf(curr, "is online at 3.0 "
                                            "Gbps");
                                        if ((expd_handle == 0) &&
                                            (enc_handle == 1)) {
                                                mpt->m_port_chng = 1;
                                        }
                                        mptsas_smhba_log_sysevent(mpt,
                                            ESC_SAS_PHY_EVENT,
                                            SAS_PHY_ONLINE,
                                            &mpt->m_phy_info[i].smhba_info);
                                        break;
                                case MPI2_EVENT_SAS_TOPO_LR_RATE_6_0:
                                        (void) sprintf(curr, "is online at "
                                            "6.0 Gbps");
                                        if ((expd_handle == 0) &&
                                            (enc_handle == 1)) {
                                                mpt->m_port_chng = 1;
                                        }
                                        mptsas_smhba_log_sysevent(mpt,
                                            ESC_SAS_PHY_EVENT,
                                            SAS_PHY_ONLINE,
                                            &mpt->m_phy_info[i].smhba_info);
                                        break;
                                case MPI25_EVENT_SAS_TOPO_LR_RATE_12_0:
                                        (void) sprintf(curr, "is online at "
                                            "12.0 Gbps");
                                        if ((expd_handle == 0) &&
                                            (enc_handle == 1)) {
                                                mpt->m_port_chng = 1;
                                        }
                                        mptsas_smhba_log_sysevent(mpt,
                                            ESC_SAS_PHY_EVENT,
                                            SAS_PHY_ONLINE,
                                            &mpt->m_phy_info[i].smhba_info);
                                        break;
                                default:
                                        (void) sprintf(curr, "state is "
                                            "unknown");
                                        break;
                                }

                                state = (link_rate &
                                    MPI2_EVENT_SAS_TOPO_LR_PREV_MASK) >>
                                    MPI2_EVENT_SAS_TOPO_LR_PREV_SHIFT;
                                switch (state) {
                                case MPI2_EVENT_SAS_TOPO_LR_PHY_DISABLED:
                                        (void) sprintf(prev, ", was disabled");
                                        break;
                                case MPI2_EVENT_SAS_TOPO_LR_NEGOTIATION_FAILED:
```

```
                                        (void) sprintf(prev, ", was offline, "
                                            "failed speed negotiation");
                                        break;
                                case MPI2_EVENT_SAS_TOPO_LR_SATA_OOB_COMPLETE:
                                        (void) sprintf(prev, ", was SATA OOB "
                                            "complete");
                                        break;
                                case SMP_RESET_IN_PROGRESS:
                                        (void) sprintf(prev, ", was SMP reset "
                                            "in progress");
                                        break;
                                case MPI2_EVENT_SAS_TOPO_LR_RATE_1_5:
                                        (void) sprintf(prev, ", was online at "
                                            "1.5 Gbps");
                                        break;
                                case MPI2_EVENT_SAS_TOPO_LR_RATE_3_0:
                                        (void) sprintf(prev, ", was online at "
                                            "3.0 Gbps");
                                        break;
                                case MPI2_EVENT_SAS_TOPO_LR_RATE_6_0:
                                        (void) sprintf(prev, ", was online at "
                                            "6.0 Gbps");
                                        break;
                                case MPI25_EVENT_SAS_TOPO_LR_RATE_12_0:
                                        (void) sprintf(prev, ", was online at "
                                            "12.0 Gbps");
                                        break;
                                default:
                                break;
                                }
                                (void) sprintf(&string[strlen(string)], "link "
                                    "changed, ");
                                break;
                        case MPI2_EVENT_SAS_TOPO_RC_NO_CHANGE:
                                continue;
                        case MPI2_EVENT_SAS_TOPO_RC_DELAY_NOT_RESPONDING:
                                (void) sprintf(&string[strlen(string)],
                                    "target not responding, delaying "
                                    "removal");
                                break;
                        }
                        NDBG20(("mptsas3%d phy %d DevHandle %x, %s%s%s\n",
                            mpt->m_instance, phy, dev_handle, string, curr,
                            prev));
                }
                if (topo_head != NULL) {
                        /*
                         * Launch DR taskq to handle topology change
                         */
                        if ((ddi_taskq_dispatch(mpt->m_dr_taskq,
                            mptsas_handle_dr, (void *)topo_head,
                            DDI_NOSLEEP)) != DDI_SUCCESS) {
                                while (topo_head != NULL) {
                                        topo_node = topo_head;
                                        topo_head = topo_head->next;
                                        kmem_free(topo_node,
                                            sizeof (mptsas_topo_change_list_t));
                                }
                                mptsas_log(mpt, CE_NOTE, "mptsas start taskq "
                                    "for handle SAS DR event failed. \n");
                        }
                }
                break;
        }
        case MPI2_EVENT_IR_CONFIGURATION_CHANGE_LIST:
        {
```



```
                Mpi2EventDataIrConfigChangeList_t       *irChangeList;
                mptsas_topo_change_list_t               *topo_head = NULL;
                mptsas_topo_change_list_t               *topo_tail = NULL;
                mptsas_topo_change_list_t               *topo_node = NULL;
                mptsas_target_t                         *ptgt;
                uint8_t                                 num_entries, i, reason;
                uint16_t                                volhandle, diskhandle;

                irChangeList = (pMpi2EventDataIrConfigChangeList_t)
                    eventreply->EventData;
                num_entries = ddi_get8(mpt->m_acc_reply_frame_hdl,
                    &irChangeList->NumElements);

                NDBG20(("mptsas3%d IR_CONFIGURATION_CHANGE_LIST event received",
                    mpt->m_instance));

                for (i = 0; i < num_entries; i++) {
                        reason = ddi_get8(mpt->m_acc_reply_frame_hdl,
                            &irChangeList->ConfigElement[i].ReasonCode);
                        volhandle = ddi_get16(mpt->m_acc_reply_frame_hdl,
                            &irChangeList->ConfigElement[i].VolDevHandle);
                        diskhandle = ddi_get16(mpt->m_acc_reply_frame_hdl,
                            &irChangeList->ConfigElement[i].PhysDiskDevHandle);

                        switch (reason) {
                        case MPI2_EVENT_IR_CHANGE_RC_ADDED:
                        case MPI2_EVENT_IR_CHANGE_RC_VOLUME_CREATED:
                        {
                                NDBG20(("mptsas %d volume added\n",
                                    mpt->m_instance));

                                topo_node = kmem_zalloc(
                                    sizeof (mptsas_topo_change_list_t),
                                    KM_SLEEP);

                                topo_node->mpt = mpt;
                                topo_node->event =
                                    MPTSAS_DR_EVENT_RECONFIG_TARGET;
                                topo_node->un.physport = 0xff;
                                topo_node->devhdl = volhandle;
                                topo_node->flags =
                                    MPTSAS_TOPO_FLAG_RAID_ASSOCIATED;
                                topo_node->object = NULL;
                                if (topo_head == NULL) {
                                        topo_head = topo_tail = topo_node;
                                } else {
                                        topo_tail->next = topo_node;
                                        topo_tail = topo_node;
                                }
                                break;
                        }
                        case MPI2_EVENT_IR_CHANGE_RC_REMOVED:
                        case MPI2_EVENT_IR_CHANGE_RC_VOLUME_DELETED:
                        {
                                NDBG20(("mptsas %d volume deleted\n",
                                    mpt->m_instance));
                                ptgt = refhash_linear_search(mpt->m_targets,
                                    mptsas_target_eval_devhdl, &volhandle);
                                if (ptgt == NULL)
                                        break;

                                /*
                                 * Clear any flags related to volume
                                 */
                                (void) mptsas_delete_volume(mpt, volhandle);
```

```
                                /*
                                 * Update DR flag immediately avoid I/O failure
                                 */
                                ptgt->m_dr_flag = MPTSAS_DR_INTRANSITION;

                                topo_node = kmem_zalloc(
                                    sizeof (mptsas_topo_change_list_t),
                                    KM_SLEEP);
                                topo_node->mpt = mpt;
                                topo_node->un.phymask =
                                    ptgt->m_addr.mta_phymask;
                                topo_node->event =
                                    MPTSAS_DR_EVENT_OFFLINE_TARGET;
                                topo_node->devhdl = volhandle;
                                topo_node->flags =
                                    MPTSAS_TOPO_FLAG_RAID_ASSOCIATED;
                                topo_node->object = (void *)ptgt;
                                if (topo_head == NULL) {
                                        topo_head = topo_tail = topo_node;
                                } else {
                                        topo_tail->next = topo_node;
                                        topo_tail = topo_node;
                                }
                                break;
                        }
                        case MPI2_EVENT_IR_CHANGE_RC_PD_CREATED:
                        case MPI2_EVENT_IR_CHANGE_RC_HIDE:
                        {
                                ptgt = refhash_linear_search(mpt->m_targets,
                                    mptsas_target_eval_devhdl, &diskhandle);
                                if (ptgt == NULL)
                                        break;

                                /*
                                 * Update DR flag immediately avoid I/O failure
                                 */
                                ptgt->m_dr_flag = MPTSAS_DR_INTRANSITION;

                                topo_node = kmem_zalloc(
                                    sizeof (mptsas_topo_change_list_t),
                                    KM_SLEEP);
                                topo_node->mpt = mpt;
                                topo_node->un.phymask =
                                    ptgt->m_addr.mta_phymask;
                                topo_node->event =
                                    MPTSAS_DR_EVENT_OFFLINE_TARGET;
                                topo_node->devhdl = diskhandle;
                                topo_node->flags =
                                    MPTSAS_TOPO_FLAG_RAID_PHYSDRV_ASSOCIATED;
                                topo_node->object = (void *)ptgt;
                                if (topo_head == NULL) {
                                        topo_head = topo_tail = topo_node;
                                } else {
                                        topo_tail->next = topo_node;
                                        topo_tail = topo_node;
                                }
                                break;
                        }
                        case MPI2_EVENT_IR_CHANGE_RC_UNHIDE:
                        case MPI2_EVENT_IR_CHANGE_RC_PD_DELETED:
                        {
                                /*
                                 * The physical drive is released by a IR
                                 * volume. But we cannot get the the physport
                                 * or phynum from the event data, so we only
                                 * can get the physport/phynum after SAS
```



```
                                 * Device Page0 request for the devhdl.
                                 */
                                topo_node = kmem_zalloc(
                                    sizeof (mptsas_topo_change_list_t),
                                    KM_SLEEP);
                                topo_node->mpt = mpt;
                                topo_node->un.phymask = 0;
                                topo_node->event =
                                    MPTSAS_DR_EVENT_RECONFIG_TARGET;
                                topo_node->devhdl = diskhandle;
                                topo_node->flags =
                                    MPTSAS_TOPO_FLAG_RAID_PHYSDRV_ASSOCIATED;
                                topo_node->object = NULL;
                                mpt->m_port_chng = 1;
                                if (topo_head == NULL) {
                                        topo_head = topo_tail = topo_node;
                                } else {
                                        topo_tail->next = topo_node;
                                        topo_tail = topo_node;
                                }
                                break;
                        }
                        default:
                                break;
                        }
                }

                if (topo_head != NULL) {
                        /*
                         * Launch DR taskq to handle topology change
                         */
                        if ((ddi_taskq_dispatch(mpt->m_dr_taskq,
                            mptsas_handle_dr, (void *)topo_head,
                            DDI_NOSLEEP)) != DDI_SUCCESS) {
                                while (topo_head != NULL) {
                                        topo_node = topo_head;
                                        topo_head = topo_head->next;
                                        kmem_free(topo_node,
                                            sizeof (mptsas_topo_change_list_t));
                                }
                                mptsas_log(mpt, CE_NOTE, "mptsas start taskq "
                                    "for handle SAS DR event failed. \n");
                        }
                }
                break;
        }
        default:
                return (DDI_FAILURE);
        }

        return (DDI_SUCCESS);
}

/*
 * handle events from ioc
 */
static void
mptsas_handle_event(void *args)
{
        m_replyh_arg_t                  *replyh_arg;
        pMpi2EventNotificationReply_t   eventreply;
        uint32_t                        event, iocloginfo, rfm;
        uint32_t                        status;
        uint8_t                         port;
        mptsas_t                        *mpt;
        uint_t                          iocstatus;
```

```
        replyh_arg = (m_replyh_arg_t *)args;
        rfm = replyh_arg->rfm;
        mpt = replyh_arg->mpt;

        mutex_enter(&mpt->m_mutex);
        /*
         * If HBA is being reset, drop incoming event.
         */
        if (mpt->m_in_reset) {
                NDBG20(("dropping event received prior to reset"));
                mutex_exit(&mpt->m_mutex);
                return;
        }

        eventreply = (pMpi2EventNotificationReply_t)
            (mpt->m_reply_frame + (rfm -
            (mpt->m_reply_frame_dma_addr&0xffffffffu)));
        event = ddi_get16(mpt->m_acc_reply_frame_hdl, &eventreply->Event);

        if (iocstatus = ddi_get16(mpt->m_acc_reply_frame_hdl,
            &eventreply->IOCStatus)) {
                if (iocstatus == MPI2_IOCSTATUS_FLAG_LOG_INFO_AVAILABLE) {
                        mptsas_log(mpt, CE_WARN,
                            "!mptsas_handle_event: IOCStatus=0x%x, "
                            "IOCLogInfo=0x%x", iocstatus,
                            ddi_get32(mpt->m_acc_reply_frame_hdl,
                            &eventreply->IOCLogInfo));
                } else {
                        mptsas_log(mpt, CE_WARN,
                            "mptsas_handle_event: IOCStatus=0x%x, "
                            "IOCLogInfo=0x%x", iocstatus,
                            ddi_get32(mpt->m_acc_reply_frame_hdl,
                            &eventreply->IOCLogInfo));
                }
        }

        /*
         * figure out what kind of event we got and handle accordingly
         */
        switch (event) {
        case MPI2_EVENT_LOG_ENTRY_ADDED:
                break;
        case MPI2_EVENT_LOG_DATA:
                iocloginfo = ddi_get32(mpt->m_acc_reply_frame_hdl,
                    &eventreply->IOCLogInfo);
                NDBG20(("mptsas %d log info %x received.\n", mpt->m_instance,
                    iocloginfo));
                break;
        case MPI2_EVENT_STATE_CHANGE:
                NDBG20(("mptsas3%d state change.", mpt->m_instance));
                break;
        case MPI2_EVENT_HARD_RESET_RECEIVED:
                NDBG20(("mptsas3%d event change.", mpt->m_instance));
                break;
        case MPI2_EVENT_SAS_DISCOVERY:
        {
                MPI2_EVENT_DATA_SAS_DISCOVERY   *sasdiscovery;
                char                            string[80];
                uint8_t                         rc;

                sasdiscovery =
                    (pMpi2EventDataSasDiscovery_t)eventreply->EventData;

                rc = ddi_get8(mpt->m_acc_reply_frame_hdl,
                    &sasdiscovery->ReasonCode);
```



```
                port = ddi_get8(mpt->m_acc_reply_frame_hdl,
                    &sasdiscovery->PhysicalPort);
                status = ddi_get32(mpt->m_acc_reply_frame_hdl,
                    &sasdiscovery->DiscoveryStatus);

                string[0] = 0;
                switch (rc) {
                case MPI2_EVENT_SAS_DISC_RC_STARTED:
                        (void) sprintf(string, "STARTING");
                        break;
                case MPI2_EVENT_SAS_DISC_RC_COMPLETED:
                        (void) sprintf(string, "COMPLETED");
                        break;
                default:
                        (void) sprintf(string, "UNKNOWN");
                        break;
                }

                NDBG20(("SAS DISCOVERY is %s for port %d, status %x", string,
                    port, status));

                break;
        }
        case MPI2_EVENT_EVENT_CHANGE:
                NDBG20(("mptsas3%d event change.", mpt->m_instance));
                break;
        case MPI2_EVENT_TASK_SET_FULL:
        {
                pMpi2EventDataTaskSetFull_t     taskfull;

                taskfull = (pMpi2EventDataTaskSetFull_t)eventreply->EventData;

                NDBG20(("TASK_SET_FULL received for mptsas3%d, depth %d\n",
                    mpt->m_instance,  ddi_get16(mpt->m_acc_reply_frame_hdl,
                    &taskfull->CurrentDepth)));
                break;
        }
        case MPI2_EVENT_SAS_TOPOLOGY_CHANGE_LIST:
        {
                /*
                 * SAS TOPOLOGY CHANGE LIST Event has already been handled
                 * in mptsas_handle_event_sync() of interrupt context
                 */
                break;
        }
        case MPI2_EVENT_SAS_ENCL_DEVICE_STATUS_CHANGE:
        {
                pMpi2EventDataSasEnclDevStatusChange_t  encstatus;
                uint8_t                                 rc;
                char                                    string[80];

                encstatus = (pMpi2EventDataSasEnclDevStatusChange_t)
                    eventreply->EventData;

                rc = ddi_get8(mpt->m_acc_reply_frame_hdl,
                    &encstatus->ReasonCode);
                switch (rc) {
                case MPI2_EVENT_SAS_ENCL_RC_ADDED:
                        (void) sprintf(string, "added");
                        break;
                case MPI2_EVENT_SAS_ENCL_RC_NOT_RESPONDING:
                        (void) sprintf(string, ", not responding");
                        break;
                default:
                break;
                }
```

```
                NDBG20(("mptsas3%d ENCLOSURE STATUS CHANGE for enclosure "
                    "%x%s\n", mpt->m_instance,
                    ddi_get16(mpt->m_acc_reply_frame_hdl,
                    &encstatus->EnclosureHandle), string));
                break;
        }

        /*
         * MPI2_EVENT_SAS_DEVICE_STATUS_CHANGE is handled by
         * mptsas_handle_event_sync,in here just send ack message.
         */
        case MPI2_EVENT_SAS_DEVICE_STATUS_CHANGE:
        {
                pMpi2EventDataSasDeviceStatusChange_t   statuschange;
                uint8_t                                 rc;
                uint16_t                                devhdl;
                uint64_t                                wwn = 0;
                uint32_t                                wwn_lo, wwn_hi;

                statuschange = (pMpi2EventDataSasDeviceStatusChange_t)
                    eventreply->EventData;
                rc = ddi_get8(mpt->m_acc_reply_frame_hdl,
                    &statuschange->ReasonCode);
                wwn_lo = ddi_get32(mpt->m_acc_reply_frame_hdl,
                    (uint32_t *)(void *)&statuschange->SASAddress);
                wwn_hi = ddi_get32(mpt->m_acc_reply_frame_hdl,
                    (uint32_t *)(void *)&statuschange->SASAddress + 1);
                wwn = ((uint64_t)wwn_hi << 32) | wwn_lo;
                devhdl =  ddi_get16(mpt->m_acc_reply_frame_hdl,
                    &statuschange->DevHandle);

                NDBG13(("MPI2_EVENT_SAS_DEVICE_STATUS_CHANGE wwn is %"PRIx64,
                    wwn));

                switch (rc) {
                case MPI2_EVENT_SAS_DEV_STAT_RC_SMART_DATA:
                        NDBG20(("SMART data received, ASC/ASCQ = %02x/%02x",
                            ddi_get8(mpt->m_acc_reply_frame_hdl,
                            &statuschange->ASC),
                            ddi_get8(mpt->m_acc_reply_frame_hdl,
                            &statuschange->ASCQ)));
                        break;

                case MPI2_EVENT_SAS_DEV_STAT_RC_UNSUPPORTED:
                        NDBG20(("Device not supported"));
                        break;

                case MPI2_EVENT_SAS_DEV_STAT_RC_INTERNAL_DEVICE_RESET:
                        NDBG20(("IOC internally generated the Target Reset "
                            "for devhdl:%x", devhdl));
                        break;

                case MPI2_EVENT_SAS_DEV_STAT_RC_CMP_INTERNAL_DEV_RESET:
                        NDBG20(("IOC’s internally generated Target Reset "
                            "completed for devhdl:%x", devhdl));
                        break;

                case MPI2_EVENT_SAS_DEV_STAT_RC_TASK_ABORT_INTERNAL:
                        NDBG20(("IOC internally generated Abort Task"));
                        break;

                case MPI2_EVENT_SAS_DEV_STAT_RC_CMP_TASK_ABORT_INTERNAL:
                        NDBG20(("IOC’s internally generated Abort Task "
                            "completed"));
                        break;
```



```
                case MPI2_EVENT_SAS_DEV_STAT_RC_ABORT_TASK_SET_INTERNAL:
                        NDBG20(("IOC internally generated Abort Task Set"));
                        break;

                case MPI2_EVENT_SAS_DEV_STAT_RC_CLEAR_TASK_SET_INTERNAL:
                        NDBG20(("IOC internally generated Clear Task Set"));
                        break;

                case MPI2_EVENT_SAS_DEV_STAT_RC_QUERY_TASK_INTERNAL:
                        NDBG20(("IOC internally generated Query Task"));
                        break;

                case MPI2_EVENT_SAS_DEV_STAT_RC_ASYNC_NOTIFICATION:
                        NDBG20(("Device sent an Asynchronous Notification"));
                        break;

                default:
                        break;
                }
                break;
        }
        case MPI2_EVENT_IR_CONFIGURATION_CHANGE_LIST:
        {
                /*
                 * IR TOPOLOGY CHANGE LIST Event has already been handled
                 * in mpt_handle_event_sync() of interrupt context
                 */
                break;
        }
        case MPI2_EVENT_IR_OPERATION_STATUS:
        {
                Mpi2EventDataIrOperationStatus_t        *irOpStatus;
                char                                    reason_str[80];
                uint8_t                                 rc, percent;
                uint16_t                                handle;

                irOpStatus = (pMpi2EventDataIrOperationStatus_t)
                    eventreply->EventData;
                rc = ddi_get8(mpt->m_acc_reply_frame_hdl,
                    &irOpStatus->RAIDOperation);
                percent = ddi_get8(mpt->m_acc_reply_frame_hdl,
                    &irOpStatus->PercentComplete);
                handle = ddi_get16(mpt->m_acc_reply_frame_hdl,
                    &irOpStatus->VolDevHandle);

                switch (rc) {
                        case MPI2_EVENT_IR_RAIDOP_RESYNC:
                                (void) sprintf(reason_str, "resync");
                                break;
                        case MPI2_EVENT_IR_RAIDOP_ONLINE_CAP_EXPANSION:
                                (void) sprintf(reason_str, "online capacity "
                                    "expansion");
                                break;
                        case MPI2_EVENT_IR_RAIDOP_CONSISTENCY_CHECK:
                                (void) sprintf(reason_str, "consistency check");
                                break;
                        default:
                                (void) sprintf(reason_str, "unknown reason %x",
                                    rc);
                }

                NDBG20(("mptsas3%d raid operational status: (%s)"
                    "\thandle(0x%04x), percent complete(%d)\n",
                    mpt->m_instance, reason_str, handle, percent));
                break;
        }
```

```
        case MPI2_EVENT_SAS_BROADCAST_PRIMITIVE:
        {
                pMpi2EventDataSasBroadcastPrimitive_t   sas_broadcast;
                uint8_t                                 phy_num;
                uint8_t                                 primitive;

                sas_broadcast = (pMpi2EventDataSasBroadcastPrimitive_t)
                    eventreply->EventData;

                phy_num = ddi_get8(mpt->m_acc_reply_frame_hdl,
                    &sas_broadcast->PhyNum);
                primitive = ddi_get8(mpt->m_acc_reply_frame_hdl,
                    &sas_broadcast->Primitive);

                switch (primitive) {
                case MPI2_EVENT_PRIMITIVE_CHANGE:
                        mptsas_smhba_log_sysevent(mpt,
                            ESC_SAS_HBA_PORT_BROADCAST,
                            SAS_PORT_BROADCAST_CHANGE,
                            &mpt->m_phy_info[phy_num].smhba_info);
                        break;
                case MPI2_EVENT_PRIMITIVE_SES:
                        mptsas_smhba_log_sysevent(mpt,
                            ESC_SAS_HBA_PORT_BROADCAST,
                            SAS_PORT_BROADCAST_SES,
                            &mpt->m_phy_info[phy_num].smhba_info);
                        break;
                case MPI2_EVENT_PRIMITIVE_EXPANDER:
                        mptsas_smhba_log_sysevent(mpt,
                            ESC_SAS_HBA_PORT_BROADCAST,
                            SAS_PORT_BROADCAST_D01_4,
                            &mpt->m_phy_info[phy_num].smhba_info);
                        break;
                case MPI2_EVENT_PRIMITIVE_ASYNCHRONOUS_EVENT:
                        mptsas_smhba_log_sysevent(mpt,
                            ESC_SAS_HBA_PORT_BROADCAST,
                            SAS_PORT_BROADCAST_D04_7,
                            &mpt->m_phy_info[phy_num].smhba_info);
                        break;
                case MPI2_EVENT_PRIMITIVE_RESERVED3:
                        mptsas_smhba_log_sysevent(mpt,
                            ESC_SAS_HBA_PORT_BROADCAST,
                            SAS_PORT_BROADCAST_D16_7,
                            &mpt->m_phy_info[phy_num].smhba_info);
                        break;
                case MPI2_EVENT_PRIMITIVE_RESERVED4:
                        mptsas_smhba_log_sysevent(mpt,
                            ESC_SAS_HBA_PORT_BROADCAST,
                            SAS_PORT_BROADCAST_D29_7,
                            &mpt->m_phy_info[phy_num].smhba_info);
                        break;
                case MPI2_EVENT_PRIMITIVE_CHANGE0_RESERVED:
                        mptsas_smhba_log_sysevent(mpt,
                            ESC_SAS_HBA_PORT_BROADCAST,
                            SAS_PORT_BROADCAST_D24_0,
                            &mpt->m_phy_info[phy_num].smhba_info);
                        break;
                case MPI2_EVENT_PRIMITIVE_CHANGE1_RESERVED:
                        mptsas_smhba_log_sysevent(mpt,
                            ESC_SAS_HBA_PORT_BROADCAST,
                            SAS_PORT_BROADCAST_D27_4,
                            &mpt->m_phy_info[phy_num].smhba_info);
                        break;
                default:
                        NDBG16(("mptsas3%d: unknown BROADCAST PRIMITIVE"
                            " %x received",
```



```
                            mpt->m_instance, primitive));
                        break;
                }
                NDBG16(("mptsas3%d sas broadcast primitive: "
                    "\tprimitive(0x%04x), phy(%d) complete\n",
                    mpt->m_instance, primitive, phy_num));
                break;
        }
        case MPI2_EVENT_IR_VOLUME:
        {
                Mpi2EventDataIrVolume_t         *irVolume;
                uint16_t                        devhandle;
                uint32_t                        state;
                int                             config, vol;
                uint8_t                         found = FALSE;

                irVolume = (pMpi2EventDataIrVolume_t)eventreply->EventData;
                state = ddi_get32(mpt->m_acc_reply_frame_hdl,
                    &irVolume->NewValue);
                devhandle = ddi_get16(mpt->m_acc_reply_frame_hdl,
                    &irVolume->VolDevHandle);

                NDBG20(("EVENT_IR_VOLUME event is received"));

                /*
                 * Get latest RAID info and then find the DevHandle for this
                 * event in the configuration.  If the DevHandle is not found
                 * just exit the event.
                 */
                (void) mptsas_get_raid_info(mpt);
                for (config = 0; (config < mpt->m_num_raid_configs) &&
                    (!found); config++) {
                        for (vol = 0; vol < MPTSAS_MAX_RAIDVOLS; vol++) {
                                if (mpt->m_raidconfig[config].m_raidvol[vol].
                                    m_raidhandle == devhandle) {
                                        found = TRUE;
                                        break;
                                }
                        }
                }
                if (!found) {
                        break;
                }

                switch (irVolume->ReasonCode) {
                case MPI2_EVENT_IR_VOLUME_RC_SETTINGS_CHANGED:
                {
                        uint32_t i;
                        mpt->m_raidconfig[config].m_raidvol[vol].m_settings =
                            state;

                        i = state & MPI2_RAIDVOL0_SETTING_MASK_WRITE_CACHING;
                        mptsas_log(mpt, CE_NOTE, " Volume %d settings changed"
                            ", auto-config of hot-swap drives is %s"
                            ", write caching is %s"
                            ", hot-spare pool mask is %02x\n",
                            vol, state &
                            MPI2_RAIDVOL0_SETTING_AUTO_CONFIG_HSWAP_DISABLE
                            ? "disabled" : "enabled",
                            i == MPI2_RAIDVOL0_SETTING_UNCHANGED
                            ? "controlled by member disks" :
                            i == MPI2_RAIDVOL0_SETTING_DISABLE_WRITE_CACHING
                            ? "disabled" :
                            i == MPI2_RAIDVOL0_SETTING_ENABLE_WRITE_CACHING
                            ? "enabled" :
                            "incorrectly set",
```

```
                            (state >> 16) & 0xff);
                                break;
                }
                case MPI2_EVENT_IR_VOLUME_RC_STATE_CHANGED:
                {
                        mpt->m_raidconfig[config].m_raidvol[vol].m_state =
                            (uint8_t)state;

                        mptsas_log(mpt, CE_NOTE,
                            "Volume %d is now %s\n", vol,
                            state == MPI2_RAID_VOL_STATE_OPTIMAL
                            ? "optimal" :
                            state == MPI2_RAID_VOL_STATE_DEGRADED
                            ? "degraded" :
                            state == MPI2_RAID_VOL_STATE_ONLINE
                            ? "online" :
                            state == MPI2_RAID_VOL_STATE_INITIALIZING
                            ? "initializing" :
                            state == MPI2_RAID_VOL_STATE_FAILED
                            ? "failed" :
                            state == MPI2_RAID_VOL_STATE_MISSING
                            ? "missing" :
                            "state unknown");
                        break;
                }
                case MPI2_EVENT_IR_VOLUME_RC_STATUS_FLAGS_CHANGED:
                {
                        mpt->m_raidconfig[config].m_raidvol[vol].
                            m_statusflags = state;

                        mptsas_log(mpt, CE_NOTE,
                            " Volume %d is now %s%s%s%s%s%s%s%s\n",
                            vol,
                            state & MPI2_RAIDVOL0_STATUS_FLAG_ENABLED
                            ? ", enabled" : ", disabled",
                            state & MPI2_RAIDVOL0_STATUS_FLAG_QUIESCED
                            ? ", quiesced" : "",
                            state & MPI2_RAIDVOL0_STATUS_FLAG_VOLUME_INACTIVE
                            ? ", inactive" : ", active",
                            state &
                            MPI2_RAIDVOL0_STATUS_FLAG_BAD_BLOCK_TABLE_FULL
                            ? ", bad block table is full" : "",
                            state &
                            MPI2_RAIDVOL0_STATUS_FLAG_RESYNC_IN_PROGRESS
                            ? ", resync in progress" : "",
                            state & MPI2_RAIDVOL0_STATUS_FLAG_BACKGROUND_INIT
                            ? ", background initialization in progress" : "",
                            state &
                            MPI2_RAIDVOL0_STATUS_FLAG_CAPACITY_EXPANSION
                            ? ", capacity expansion in progress" : "",
                            state &
                            MPI2_RAIDVOL0_STATUS_FLAG_CONSISTENCY_CHECK
                            ? ", consistency check in progress" : "",
                            state & MPI2_RAIDVOL0_STATUS_FLAG_DATA_SCRUB
                            ? ", data scrub in progress" : "");
                        break;
                }
                default:
                        break;
                }
                break;
        }
        case MPI2_EVENT_IR_PHYSICAL_DISK:
        {
                Mpi2EventDataIrPhysicalDisk_t   *irPhysDisk;
                uint16_t                        devhandle, enchandle, slot;
```



```
                uint32_t                        status, state;
                uint8_t                         physdisknum, reason;

                irPhysDisk = (Mpi2EventDataIrPhysicalDisk_t *)
                    eventreply->EventData;
                physdisknum = ddi_get8(mpt->m_acc_reply_frame_hdl,
                    &irPhysDisk->PhysDiskNum);
                devhandle = ddi_get16(mpt->m_acc_reply_frame_hdl,
                    &irPhysDisk->PhysDiskDevHandle);
                enchandle = ddi_get16(mpt->m_acc_reply_frame_hdl,
                    &irPhysDisk->EnclosureHandle);
                slot = ddi_get16(mpt->m_acc_reply_frame_hdl,
                    &irPhysDisk->Slot);
                state = ddi_get32(mpt->m_acc_reply_frame_hdl,
                    &irPhysDisk->NewValue);
                reason = ddi_get8(mpt->m_acc_reply_frame_hdl,
                    &irPhysDisk->ReasonCode);

                NDBG20(("EVENT_IR_PHYSICAL_DISK event is received"));

                switch (reason) {
                case MPI2_EVENT_IR_PHYSDISK_RC_SETTINGS_CHANGED:
                        mptsas_log(mpt, CE_NOTE,
                            " PhysDiskNum %d with DevHandle 0x%x in slot %d "
                            "for enclosure with handle 0x%x is now in hot "
                            "spare pool %d",
                            physdisknum, devhandle, slot, enchandle,
                            (state >> 16) & 0xff);
                        break;

                case MPI2_EVENT_IR_PHYSDISK_RC_STATUS_FLAGS_CHANGED:
                        status = state;
                        mptsas_log(mpt, CE_NOTE,
                            " PhysDiskNum %d with DevHandle 0x%x in slot %d "
                            "for enclosure with handle 0x%x is now "
                            "%s%s%s%s%s\n", physdisknum, devhandle, slot,
                            enchandle,
                            status & MPI2_PHYSDISK0_STATUS_FLAG_INACTIVE_VOLUME
                            ? ", inactive" : ", active",
                            status & MPI2_PHYSDISK0_STATUS_FLAG_OUT_OF_SYNC
                            ? ", out of sync" : "",
                            status & MPI2_PHYSDISK0_STATUS_FLAG_QUIESCED
                            ? ", quiesced" : "",
                            status &
                            MPI2_PHYSDISK0_STATUS_FLAG_WRITE_CACHE_ENABLED
                            ? ", write cache enabled" : "",
                            status & MPI2_PHYSDISK0_STATUS_FLAG_OCE_TARGET
                            ? ", capacity expansion target" : "");
                        break;

                case MPI2_EVENT_IR_PHYSDISK_RC_STATE_CHANGED:
                        mptsas_log(mpt, CE_NOTE,
                            " PhysDiskNum %d with DevHandle 0x%x in slot %d "
                            "for enclosure with handle 0x%x is now %s\n",
                            physdisknum, devhandle, slot, enchandle,
                            state == MPI2_RAID_PD_STATE_OPTIMAL
                            ? "optimal" :
                            state == MPI2_RAID_PD_STATE_REBUILDING
                            ? "rebuilding" :
                            state == MPI2_RAID_PD_STATE_DEGRADED
                            ? "degraded" :
                            state == MPI2_RAID_PD_STATE_HOT_SPARE
                            ? "a hot spare" :
                            state == MPI2_RAID_PD_STATE_ONLINE
                            ? "online" :
                            state == MPI2_RAID_PD_STATE_OFFLINE
```

```
                                ? "offline" :
                                state == MPI2_RAID_PD_STATE_NOT_COMPATIBLE
                                ? "not compatible" :
                                state == MPI2_RAID_PD_STATE_NOT_CONFIGURED
                                ? "not configured" :
                                "state unknown");
                        break;
                }
                break;
        }
        default:
                NDBG20(("mptsas3%d: unknown event %x received",
                    mpt->m_instance, event));
                break;
        }

        /*
         * Return the reply frame to the free queue.
         */
        ddi_put32(mpt->m_acc_free_queue_hdl,
            &((uint32_t *)(void *)mpt->m_free_queue)[mpt->m_free_index], rfm);
        (void) ddi_dma_sync(mpt->m_dma_free_queue_hdl, 0, 0,
            DDI_DMA_SYNC_FORDEV);
        if (++mpt->m_free_index == mpt->m_free_queue_depth) {
                mpt->m_free_index = 0;
        }
        ddi_put32(mpt->m_datap, &mpt->m_reg->ReplyFreeHostIndex,
            mpt->m_free_index);
        mutex_exit(&mpt->m_mutex);
}

/*
 * invoked from timeout() to restart qfull cmds with throttle == 0
 */
static void
mptsas_restart_cmd(void *arg)
{
        mptsas_t        *mpt = arg;
        mptsas_target_t *ptgt = NULL;

        mutex_enter(&mpt->m_mutex);

        mpt->m_restart_cmd_timeid = 0;

        for (ptgt = refhash_first(mpt->m_targets); ptgt != NULL;
            ptgt = refhash_next(mpt->m_targets, ptgt)) {
                mutex_enter(&ptgt->m_t_mutex);
                if (ptgt->m_reset_delay == 0) {
                        if (ptgt->m_t_throttle == QFULL_THROTTLE) {
                                mptsas_set_throttle(mpt, ptgt,
                                    MAX_THROTTLE);
                        }
                }
                mutex_exit(&ptgt->m_t_mutex);
        }
        mptsas_restart_hba(mpt);
        mutex_exit(&mpt->m_mutex);
}

/*
 * Assume some checks have been done prior to calling this
 * function so we don’t need to consider taking the m_mutex.
 */
static void
mptsas_remove_cmd_nomtx(mptsas_t *mpt, mptsas_cmd_t *cmd)
{
```



```
        int             slot;
        mptsas_slots_t  *slots = mpt->m_active;
        mptsas_target_t *ptgt = cmd->cmd_tgt_addr;

        ASSERT(cmd != NULL);
        ASSERT(cmd->cmd_queued == FALSE);
        ASSERT((cmd->cmd_flags & CFLAG_CMDIOC) == 0);

        slot = cmd->cmd_slot;

        /*
         * remove the cmd.
         */
        if (cmd == slots->m_slot[slot]) {
                NDBG31(("mptsas_remove_cmd_nomtx: removing cmd=0x%p, flags "
                    "0x%x", (void *)cmd, cmd->cmd_flags));
                slots->m_slot[slot] = NULL;
                ASSERT(mpt->m_ncmds != 0);
                atomic_dec_32(&mpt->m_ncmds);
                ASSERT(mpt->m_rep_post_queues[cmd->cmd_rpqidx].rpq_ncmds != 0);
                atomic_dec_32(
                    &mpt->m_rep_post_queues[cmd->cmd_rpqidx].rpq_ncmds);

                /*
                 * Decrement per target ncmds, we know this is not an
                 * IOC cmd and it therefore has a target associated with it.
                 */
                mutex_enter(&ptgt->m_t_mutex);
                ASSERT(ptgt->m_t_ncmds != 0);
                ptgt->m_t_ncmds--;

                /*
                 * reset throttle if we just ran an untagged command
                 * to a tagged target
                 */
                if ((ptgt->m_t_ncmds == 0) &&
                    ((cmd->cmd_pkt_flags & FLAG_TAGMASK) == 0)) {
                        mptsas_set_throttle(mpt, ptgt, MAX_THROTTLE);
                }

                /*
                 * Remove this command from the active queue.
                 */
                if (cmd->cmd_active_expiration != 0) {
                        TAILQ_REMOVE(&ptgt->m_active_cmdq, cmd,
                            cmd_active_link);
                        cmd->cmd_active_expiration = 0;
                }
                mutex_exit(&ptgt->m_t_mutex);
        }

        ASSERT(cmd != slots->m_slot[cmd->cmd_slot]);
}

void
mptsas_remove_cmd(mptsas_t *mpt, mptsas_cmd_t *cmd)
{
        int             slot;
        mptsas_slots_t  *slots = mpt->m_active;
        mptsas_target_t *ptgt = cmd->cmd_tgt_addr;

        ASSERT(cmd != NULL);
        ASSERT(cmd->cmd_queued == FALSE);

        /*
         * Task Management cmds are removed in their own routines.  Also,
```

```
         * we don't want to modify timeout based on TM cmds.
         */
        if (cmd->cmd_flags & CFLAG_TM_CMD) {
                return;
        }

        slot = cmd->cmd_slot;

        /*
         * remove the cmd.
         */
        if (cmd == slots->m_slot[slot]) {
                NDBG31(("mptsas_remove_cmd: removing cmd=0x%p, flags 0x%x",
                    (void *)cmd, cmd->cmd_flags));
                slots->m_slot[slot] = NULL;
                ASSERT(mpt->m_ncmds != 0);
                atomic_dec_32(&mpt->m_ncmds);
                ASSERT(mpt->m_rep_post_queues[cmd->cmd_rpqidx].rpq_ncmds != 0);
                atomic_dec_32(
                    &mpt->m_rep_post_queues[cmd->cmd_rpqidx].rpq_ncmds);

                /*
                 * only decrement per target ncmds if command
                 * has a target associated with it.
                 */
                if ((cmd->cmd_flags & CFLAG_CMDIOC) == 0) {
                        mutex_enter(&ptgt->m_t_mutex);
                        ASSERT(ptgt->m_t_ncmds != 0);
                        ptgt->m_t_ncmds--;

                        /*
                         * reset throttle if we just ran an untagged command
                         * to a tagged target
                         */
                        if ((ptgt->m_t_ncmds == 0) &&
                            ((cmd->cmd_pkt_flags & FLAG_TAGMASK) == 0)) {
                                mptsas_set_throttle(mpt, ptgt, MAX_THROTTLE);
                        }

                        /*
                         * Remove this command from the active queue.
                         */
                        if (cmd->cmd_active_expiration != 0) {
                                TAILQ_REMOVE(&ptgt->m_active_cmdq, cmd,
                                    cmd_active_link);
                                cmd->cmd_active_expiration = 0;
                        }
                        mutex_exit(&ptgt->m_t_mutex);
                }

        }

        /*
         * This is all we need to do for ioc commands.
         */
        if (cmd->cmd_flags & CFLAG_CMDIOC) {
                mptsas_return_to_pool(mpt, cmd);
                return;
        }

        ASSERT(cmd != slots->m_slot[cmd->cmd_slot]);
}

/*
 * accept all cmds on the tx_waitq if any and then
 * start a fresh request from the top of the device queue.
```



```
 *
 * since there are always cmds queued on the tx_waitq, and rare cmds on
 * the instance waitq, so this function should not be invoked in the ISR,
 * the mptsas_restart_waitq() is invoked in the ISR instead. otherwise, the
 * burden belongs to the IO dispatch CPUs is moved the interrupt CPU.
 */
static void
mptsas_restart_hba(mptsas_t *mpt)
{
        ASSERT(mutex_owned(&mpt->m_mutex));

        mptsas_accept_tx_waitqs(mpt);
        mptsas_restart_waitq(mpt);
}

/*
 * start a fresh request from the top of the device queue
 */
static void
mptsas_restart_waitq(mptsas_t *mpt)
{
        mptsas_cmd_t    *cmd, *next_cmd;
        mptsas_target_t *ptgt = NULL;

        NDBG1(("mptsas_restart_waitq: mpt=0x%p", (void *)mpt));

        ASSERT(mutex_owned(&mpt->m_mutex));

        /*
         * If there is a reset delay, don't start any cmds.  Otherwise, start
         * as many cmds as possible.
         * Since SMID 0 is reserved and the TM slot is reserved, the actual max
         * commands is m_max_requests - 2.
         */
        cmd = mpt->m_waitq;

        while (cmd != NULL) {
                next_cmd = cmd->cmd_linkp;
                if (cmd->cmd_flags & CFLAG_PASSTHRU) {
                        if (mptsas_save_cmd(mpt, cmd) == TRUE) {
                                /*
                                 * passthru command get slot need
                                 * set CFLAG_PREPARED.
                                 */
                                cmd->cmd_flags |= CFLAG_PREPARED;
                                mptsas_waitq_delete(mpt, cmd);
                                mptsas_start_passthru(mpt, cmd);
                        }
                        cmd = next_cmd;
                        continue;
                }
                if (cmd->cmd_flags & CFLAG_CONFIG) {
                        if (mptsas_save_cmd(mpt, cmd) == TRUE) {
                                /*
                                 * Send the config page request and delete it
                                 * from the waitq.
                                 */
                                cmd->cmd_flags |= CFLAG_PREPARED;
                                mptsas_waitq_delete(mpt, cmd);
                                mptsas_start_config_page_access(mpt, cmd);
                        }
                        cmd = next_cmd;
                        continue;
                }
                if (cmd->cmd_flags & CFLAG_FW_DIAG) {
                        if (mptsas_save_cmd(mpt, cmd) == TRUE) {
```

```
                                /*
                                 * Send the FW Diag request and delete if from
                                 * the waitq.
                                 */
                                cmd->cmd_flags |= CFLAG_PREPARED;
                                mptsas_waitq_delete(mpt, cmd);
                                mptsas_start_diag(mpt, cmd);
                        }
                        cmd = next_cmd;
                        continue;
                }

                ptgt = cmd->cmd_tgt_addr;
                if (ptgt) {
                        mutex_enter(&ptgt->m_t_mutex);
                        if ((ptgt->m_t_throttle == DRAIN_THROTTLE) &&
                            (ptgt->m_t_ncmds == 0)) {
                                mptsas_set_throttle(mpt, ptgt, MAX_THROTTLE);
                        }
                        if ((mpt->m_ncmds <= (mpt->m_max_requests - 2)) &&
                            (ptgt->m_reset_delay == 0) &&
                            (ptgt->m_t_ncmds < ptgt->m_t_throttle)) {
                                mutex_exit(&ptgt->m_t_mutex);

                                if (mptsas_save_cmd(mpt, cmd) == TRUE) {
                                        mptsas_waitq_delete(mpt, cmd);
                                        mutex_exit(&mpt->m_mutex);
                                        (void) mptsas_start_cmd(mpt, cmd);
                                        mutex_enter(&mpt->m_mutex);
                                        cmd = mpt->m_waitq;
                                        continue;
                                }
                        } else {
                                mutex_exit(&ptgt->m_t_mutex);
                        }
                }
                cmd = next_cmd;
        }
}

/*
 * Cmds are queued if scsi_start() doesn't get the m_mutex lock(no wait)
 * or if the decision has been made to always do that. Setting
 * mptsas_allow_txq_jumping to zero will allow higher performance on
 * a heavily loaded system as there is less disruption to the flow here.
 * There are 2 threads that handle one queue each. The idea is that
 * they take it in turn to grab the m_mutex to run the mptsas_accept_pkt()
 * function and then drop it while the cmd is started in mptsas_start_cmd().
 */
static void
mptsas_tx_waitq_thread(mptsas_thread_arg_t *arg)
{
        mptsas_t *mpt = arg->mpt;
        mptsas_tx_waitqueue_t *txwq = &mpt->m_tx_waitq[arg->t];

        mutex_enter(&txwq->txwq_mutex);
        while (txwq->txwq_active) {
                mptsas_drain_tx_waitq(mpt, txwq);
                if (txwq->txwq_wdrain) {
                        cv_signal(&txwq->txwq_drain_cv);
                }
                cv_wait(&txwq->txwq_cv, &txwq->txwq_mutex);
        }
        mutex_exit(&txwq->txwq_mutex);
        mutex_enter(&mpt->m_qthread_mutex);
        mpt->m_txwq_thread_n--;
```



```
        cv_broadcast(&mpt->m_qthread_cv);
        mutex_exit(&mpt->m_qthread_mutex);
}

/*
 * Set the draining flag, disconnect the list and process one at a time
 * so that the cmds are sent in order.
 */
static void
mptsas_drain_tx_waitq(mptsas_t *mpt, mptsas_tx_waitqueue_t *txwq)
{
        mptsas_cmd_t    *cmd, *ncmd;
        int             rval, start;
#ifdef MPTSAS_DEBUG
        uint32_t        qlen;
#endif

        txwq->txwq_draining = TRUE;
#ifndef __lock_lint
        _NOTE(CONSTCOND)
#endif
        while (TRUE) {

                /*
                 * A Bus Reset could occur at any time but it will have to
                 * wait for the main mutex before flushing the tx_waitq.
                 * Pull all commands at once, then follow the list in order to
                 * reduce txwq_mutex hold time. If there is a Bus Reset at
                 * some point the commands will get to the waitq and then be
                 * flushed.
                 */
                cmd = txwq->txwq_cmdq;

                if (cmd == NULL) {
                        txwq->txwq_draining = FALSE;
                        return;
                }
                txwq->txwq_cmdq = NULL;
                txwq->txwq_qtail = &txwq->txwq_cmdq;
#ifdef MPTSAS_DEBUG
                qlen = txwq->txwq_len;
#endif
                txwq->txwq_len = 0;
                mutex_exit(&txwq->txwq_mutex);

                while (cmd) {
                        ncmd = cmd->cmd_linkp;
                        cmd->cmd_linkp = NULL;
                        mutex_enter(&mpt->m_mutex);
                        start = mptsas_accept_pkt(mpt, cmd, &rval);
                        mutex_exit(&mpt->m_mutex);
                        if (start) {
                                (void) mptsas_start_cmd(mpt, cmd);
                        }
                        if (rval != TRAN_ACCEPT)
                                cmn_err(CE_WARN,
                                    "mpt: mptsas_drain_tx_waitq: failed "
                                    "(rval=0x%x) to accept cmd 0x%p on queue\n",
                                    rval, (void *)cmd);
                        cmd = ncmd;
#ifdef MPTSAS_DEBUG
                        qlen--;
#endif
                }
                ASSERT(qlen == 0);
                mutex_enter(&txwq->txwq_mutex);
```

```
        }
}

/*
 * 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.
 */
static void
mptsas_block_tx_waitqs(mptsas_t *mpt, int wait)
{
        int             i;
        uint8_t         wdrain = 0;
        mptsas_tx_waitqueue_t *txwq;

        ASSERT(mutex_owned(&mpt->m_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) {
                /*
                 * 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.
                 */
                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);
        }
}

static void
mptsas_unblock_tx_waitqs(mptsas_t *mpt)
{
        int                     i;
        mptsas_tx_waitqueue_t   *txwq;

        if (mpt->m_txwq_thread_n == 0) {
                return;
```



```
        }

        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);
        }

        mpt->m_txwq_enabled = FALSE;
}

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);
}

/*
 * 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;
        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;
        uint8_t                 MSIidx;
        uint64_t                request_desc;
        uint32_t                ars_dmaaddrlow;
        mptsas_cmd_t            *c;

        NDBG1(("mptsas_start_cmd: cmd=0x%p, flags 0x%x", (void *)cmd,
            cmd->cmd_flags));

        /*
         * Set SMID and increment index.  Rollover to 1 instead of 0 if index
         * is at the max.  0 is an invalid SMID, so we call the first index 1.
         */
        SMID = cmd->cmd_slot;
        MSIidx = cmd->cmd_rpqidx;

        /*
         * It is possible for back to back device reset to
         * happen before the reset delay has expired.  That's
         * ok, just let the device reset go out on the bus.
         */
        if ((cmd->cmd_pkt_flags & FLAG_NOINTR) == 0) {
                ASSERT(ptgt->m_reset_delay == 0);
        }
```

```
        /*
         * if a non-tagged cmd is submitted to an active tagged target
         * then drain before submitting this cmd; SCSI-2 allows RQSENSE
         * to be untagged
         */
        mutex_enter(&ptgt->m_t_mutex);
        if (((cmd->cmd_pkt_flags & FLAG_TAGMASK) == 0) &&
            (ptgt->m_t_ncmds > 1) &&
            ((cmd->cmd_flags & CFLAG_TM_CMD) == 0) &&
            (*(cmd->cmd_pkt->pkt_cdbp) != SCMD_REQUEST_SENSE)) {
                if ((cmd->cmd_pkt_flags & FLAG_NOINTR) == 0) {
                        NDBG23(("target=%d, untagged cmd, start draining\n",
                            ptgt->m_devhdl));

                        if (ptgt->m_reset_delay == 0) {
                                mptsas_set_throttle(mpt, ptgt, DRAIN_THROTTLE);
                        }
                        mutex_exit(&ptgt->m_t_mutex);

                        mutex_enter(&mpt->m_mutex);
                        mptsas_remove_cmd(mpt, cmd);
                        cmd->cmd_pkt_flags |= FLAG_HEAD;
                        mptsas_waitq_add(mpt, cmd);
                        mutex_exit(&mpt->m_mutex);
                } else {
                        mutex_exit(&ptgt->m_t_mutex);
                }
                return (DDI_FAILURE);
        }

        /*
         * Set correct tag bits.
         */
        if (cmd->cmd_pkt_flags & FLAG_TAGMASK) {
                switch (mptsas_tag_lookup[((cmd->cmd_pkt_flags &
                    FLAG_TAGMASK) >> 12)]) {
                case MSG_SIMPLE_QTAG:
                        control |= MPI2_SCSIIO_CONTROL_SIMPLEQ;
                        break;
                case MSG_HEAD_QTAG:
                        control |= MPI2_SCSIIO_CONTROL_HEADOFQ;
                        break;
                case MSG_ORDERED_QTAG:
                        control |= MPI2_SCSIIO_CONTROL_ORDEREDQ;
                        break;
                default:
                        mptsas_log(mpt, CE_WARN, "mpt: Invalid tag type\n");
                        break;
                }
        } else {
                if (*(cmd->cmd_pkt->pkt_cdbp) != SCMD_REQUEST_SENSE) {
                                ptgt->m_t_throttle = 1;
                }
                control |= MPI2_SCSIIO_CONTROL_SIMPLEQ;
        }

        /*
         * Set timeout.
         */
        cmd->cmd_active_expiration =
            gethrtime() + (hrtime_t)pkt->pkt_time * NANOSEC;

        c = TAILQ_FIRST(&ptgt->m_active_cmdq);
        if (c == NULL ||
            c->cmd_active_expiration < cmd->cmd_active_expiration) {
                /*
```



```
                 * Common case is that this is the last pending expiration
                 * (or queue is empty). Insert at head of the queue.
                 */
                TAILQ_INSERT_HEAD(&ptgt->m_active_cmdq, cmd, cmd_active_link);
        } else {
                /*
                 * Queue is not empty and first element expires later than
                 * this command. Search for element expiring sooner.
                 */
                while ((c = TAILQ_NEXT(c, cmd_active_link)) != NULL) {
                        if (c->cmd_active_expiration <
                            cmd->cmd_active_expiration) {
                                TAILQ_INSERT_BEFORE(c, cmd, cmd_active_link);
                                break;
                        }
                }
                if (c == NULL) {
                        /*
                         * No element found expiring sooner, append to
                         * non-empty queue.
                         */
                        TAILQ_INSERT_TAIL(&ptgt->m_active_cmdq, cmd,
                            cmd_active_link);
                }
        }

        mutex_exit(&ptgt->m_t_mutex);

        if (cmd->cmd_pkt_flags & FLAG_TLR) {
                control |= MPI2_SCSIIO_CONTROL_TLR_ON;
        }

        mem = mpt->m_req_frame + (mpt->m_req_frame_size * SMID);
        io_request = (pMpi2SCSIIORequest_t)mem;
        if (cmd->cmd_extrqslen != 0) {
                /*
                 * Mapping of the buffer was done in mptsas_pkt_alloc_extern().
                 * Calculate the DMA address with the same offset.
                 */
                arsbuf = cmd->cmd_arq_buf;
                ars_size = cmd->cmd_extrqslen;
                ars_dmaaddrlow = (mpt->m_req_sense_dma_addr +
                    ((uintptr_t)arsbuf - (uintptr_t)mpt->m_req_sense)) &
                    0xffffffffull;
        } else {
                arsbuf = mpt->m_req_sense + (mpt->m_req_sense_size * (SMID-1));
                cmd->cmd_arq_buf = arsbuf;
                ars_size = mpt->m_req_sense_size;
                ars_dmaaddrlow = (mpt->m_req_sense_dma_addr +
                    (mpt->m_req_sense_size * (SMID-1))) &
                    0xffffffffull;
        }
        bzero(io_request, sizeof (Mpi2SCSIIORequest_t));
        bzero(arsbuf, ars_size);

        ddi_put8(acc_hdl, &io_request->SGLOffset0, offsetof
            (MPI2_SCSI_IO_REQUEST, SGL) / 4);
        mptsas_init_std_hdr(acc_hdl, io_request, ptgt->m_devhdl, Lun(cmd), 0,
            MPI2_FUNCTION_SCSI_IO_REQUEST);

        (void) ddi_rep_put8(acc_hdl, (uint8_t *)pkt->pkt_cdbp,
            io_request->CDB.CDB32, cmd->cmd_cdblen, DDI_DEV_AUTOINCR);

        io_flags = cmd->cmd_cdblen;
        if (mptsas3_use_fastpath &&
            ptgt->m_io_flags & MPI25_SAS_DEVICE0_FLAGS_ENABLED_FAST_PATH) {
```

```
                io_flags |= MPI25_SCSIIO_IOFLAGS_FAST_PATH;
                request_desc = MPI25_REQ_DESCRIPT_FLAGS_FAST_PATH_SCSI_IO;
        } else {
                request_desc = MPI2_REQ_DESCRIPT_FLAGS_SCSI_IO;
        }
        ddi_put16(acc_hdl, &io_request->IoFlags, io_flags);
        /*
         * setup the Scatter/Gather DMA list for this request
         */
        if (cmd->cmd_cookiec > 0) {
                mptsas_sge_setup(mpt, cmd, &control, io_request, acc_hdl);
        } else {
                ddi_put32(acc_hdl, &io_request->SGL.MpiSimple.FlagsLength,
                    ((uint32_t)MPI2_SGE_FLAGS_LAST_ELEMENT |
                    MPI2_SGE_FLAGS_END_OF_BUFFER |
                    MPI2_SGE_FLAGS_SIMPLE_ELEMENT |
                    MPI2_SGE_FLAGS_END_OF_LIST) << MPI2_SGE_FLAGS_SHIFT);
        }

        /*
         * save ARQ information
         */
        ddi_put8(acc_hdl, &io_request->SenseBufferLength, cmd->cmd_rqslen);
        ddi_put32(acc_hdl, &io_request->SenseBufferLowAddress, ars_dmaaddrlow);

        ddi_put32(acc_hdl, &io_request->Control, control);

        NDBG31(("starting message=%d(0x%p), with cmd=0x%p",
            SMID, (void *)io_request, (void *)cmd));

        (void) ddi_dma_sync(dma_hdl, 0, 0, DDI_DMA_SYNC_FORDEV);

        /*
         * Build request descriptor and write it to the request desc post reg.
         */
        request_desc |= (SMID << 16) + (MSIidx << 8);
        request_desc |= ((uint64_t)ptgt->m_devhdl << 48);
        MPTSAS_START_CMD(mpt, request_desc);

#if 0
        /* Is this of any benefit here, what is it going to catch? */
        if ((mptsas_check_dma_handle(dma_hdl) != DDI_SUCCESS) ||
            (mptsas_check_acc_handle(acc_hdl) != DDI_SUCCESS)) {
                ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_UNAFFECTED);
                return (DDI_FAILURE);
        }
#endif
        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.
 */
static void
mptsas_deliver_doneq_thread(mptsas_t *mpt, mptsas_done_list_t *dlist)
{
        uint32_t                        t, i, j = mpt->m_doneq_next_thread;
        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.
         */
        for (i = 0; i < mpt->m_doneq_thread_n; i++) {
                item = &mpt->m_doneq_thread_id[j];

                /*
                 * If the completed command on help thread[i] less than
                 * doneq_thread_threshold, then pick the thread[j]. Otherwise
                 * pick a thread which has least completed command.
                 */
                if (item->dlist.dl_len < mpt->m_doneq_thread_threshold) {
                        t = j;
                        break;
                }
                if (item->dlist.dl_len < min) {
                        min = item->dlist.dl_len;
                        t = j;
                }
                if (++j == mpt->m_doneq_thread_n) {
                        j = 0;
                }
        }
        item = &mpt->m_doneq_thread_id[t];
        mutex_enter(&item->mutex);
        mptsas_doneq_mv(dlist, item);
        cv_signal(&item->cv);
        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;
        }
        mpt->m_doneq_next_thread = (uint16_t)t;
}

/*
 * move one doneq to another.
 */
static void
mptsas_doneq_mv(mptsas_done_list_t *from, mptsas_doneq_thread_list_t *item)
{
        mptsas_done_list_t              *to = &item->dlist;
        mptsas_cmd_t                    *cmd;

        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;
        }
}

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) ||
            (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)) {
                ddi_fm_service_impact(mpt->m_dip,
                    DDI_SERVICE_UNAFFECTED);
                ddi_fm_acc_err_clear(mpt->m_config_handle,
                    DDI_FME_VER0);
                pkt->pkt_reason = CMD_TRAN_ERR;
                pkt->pkt_statistics = 0;
        }
        if ((mptsas_check_dma_handle(mpt->m_dma_req_frame_hdl) !=
            DDI_SUCCESS) ||
            (mptsas_check_dma_handle(mpt->m_dma_req_sense_hdl) !=
            DDI_SUCCESS) ||
            (mptsas_check_dma_handle(mpt->m_dma_reply_frame_hdl) !=
            DDI_SUCCESS) ||
            (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)) {
                ddi_fm_service_impact(mpt->m_dip,
                    DDI_SERVICE_UNAFFECTED);
                pkt->pkt_reason = CMD_TRAN_ERR;
                pkt->pkt_statistics = 0;
        }
        if (cmd->cmd_dmahandle &&
            (mptsas_check_dma_handle(cmd->cmd_dmahandle) != DDI_SUCCESS)) {
                ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_UNAFFECTED);
                pkt->pkt_reason = CMD_TRAN_ERR;
                pkt->pkt_statistics = 0;
        }
        if ((cmd->cmd_extra_frames &&
            ((mptsas_check_dma_handle(cmd->cmd_extra_frames->m_dma_hdl) !=
            DDI_SUCCESS) ||
            (mptsas_check_acc_handle(cmd->cmd_extra_frames->m_acc_hdl) !=
            DDI_SUCCESS)))) {
                ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_UNAFFECTED);
                pkt->pkt_reason = CMD_TRAN_ERR;
                pkt->pkt_statistics = 0;
        }
}

/*
 * These routines manipulate the queue of commands that
 * are waiting for their completion routines to be called.
 * The queue is usually in FIFO order but on an MP system
 * it's possible for the completion routines to get out
 * of order. If that's a problem you need to add a global
 * mutex around the code that calls the completion routine
 * in the interrupt handler.
```



```
 */
static void
mptsas_doneq_add(mptsas_t *mpt, mptsas_cmd_t *cmd)
{
        struct scsi_pkt *pkt = CMD2PKT(cmd);

        NDBG31(("mptsas_doneq_add: cmd=0x%p", (void *)cmd));

        ASSERT((cmd->cmd_flags & CFLAG_COMPLETED) == 0);
        cmd->cmd_linkp = NULL;
        cmd->cmd_flags |= CFLAG_FINISHED;
        cmd->cmd_flags &= ~CFLAG_IN_TRANSPORT;

        mptsas_fma_check(mpt, cmd);

        /*
         * only add scsi pkts that have completion routines to
         * the doneq.  no intr cmds do not have callbacks.
         */
        if (pkt && (pkt->pkt_comp)) {
                *mpt->m_dlist.dl_tail = cmd;
                mpt->m_dlist.dl_tail = &cmd->cmd_linkp;
                mpt->m_dlist.dl_len++;
        }
}

static void
mptsas_rpdoneq_add(mptsas_t *mpt, mptsas_reply_pqueue_t *rpqp,
    mptsas_cmd_t *cmd)
{
        struct scsi_pkt *pkt = CMD2PKT(cmd);

        NDBG31(("mptsas_rpdoneq_add: cmd=0x%p", (void *)cmd));

        ASSERT((cmd->cmd_flags & CFLAG_COMPLETED) == 0);
        cmd->cmd_linkp = NULL;
        cmd->cmd_flags |= CFLAG_FINISHED;
        cmd->cmd_flags &= ~CFLAG_IN_TRANSPORT;

        mptsas_fma_check(mpt, cmd);

        /*
         * only add scsi pkts that have completion routines to
         * the doneq.  no intr cmds do not have callbacks.
         */
        if (pkt && (pkt->pkt_comp)) {
                *rpqp->rpq_dlist.dl_tail = cmd;
                rpqp->rpq_dlist.dl_tail = &cmd->cmd_linkp;
                rpqp->rpq_dlist.dl_len++;
        }
}

static mptsas_cmd_t *
mptsas_doneq_thread_rm(mptsas_t *mpt, uint64_t t)
{
        mptsas_cmd_t                    *cmd;
        mptsas_doneq_thread_list_t      *item = &mpt->m_doneq_thread_id[t];

        /* pop one off the done queue */
        if ((cmd = item->dlist.dl_q) != NULL) {
                /* if the queue is now empty fix the tail pointer */
                NDBG31(("mptsas_doneq_thread_rm: cmd=0x%p", (void *)cmd));
                if ((item->dlist.dl_q = cmd->cmd_linkp) == NULL) {
                        item->dlist.dl_tail = &item->dlist.dl_q;
                }
                cmd->cmd_linkp = NULL;
```

```
                item->dlist.dl_len--;
        }
        return (cmd);
}

static void
mptsas_doneq_empty(mptsas_t *mpt)
{
        if (mpt->m_dlist.dl_q) {
                mptsas_cmd_t    *cmd, *next;
                struct scsi_pkt *pkt;

                cmd = mpt->m_dlist.dl_q;
                mpt->m_dlist.dl_q = NULL;
                mpt->m_dlist.dl_tail = &mpt->m_dlist.dl_q;
                mpt->m_dlist.dl_len = 0;

                mutex_exit(&mpt->m_mutex);
                /*
                 * run the completion routines of all the
                 * completed commands
                 */
                while (cmd != NULL) {
                        next = cmd->cmd_linkp;
                        cmd->cmd_linkp = NULL;
                        /* run this command's completion routine */
                        cmd->cmd_flags |= CFLAG_COMPLETED;
                        pkt = CMD2PKT(cmd);
                        mptsas_pkt_comp(pkt, cmd);
                        cmd = next;
                }
                mutex_enter(&mpt->m_mutex);
        }
}

static void
mptsas_rpdoneq_empty(mptsas_reply_pqueue_t *rpqp)
{
        if (rpqp->rpq_dlist.dl_q) {
                mptsas_cmd_t    *cmd, *next;
                struct scsi_pkt *pkt;

                cmd = rpqp->rpq_dlist.dl_q;
                rpqp->rpq_dlist.dl_q = NULL;
                rpqp->rpq_dlist.dl_tail = &rpqp->rpq_dlist.dl_q;
                rpqp->rpq_dlist.dl_len = 0;

                mutex_exit(&rpqp->rpq_mutex);
                /*
                 * run the completion routines of all the
                 * completed commands
                 */
                while (cmd != NULL) {
                        next = cmd->cmd_linkp;
                        cmd->cmd_linkp = NULL;
                        /* run this command's completion routine */
                        cmd->cmd_flags |= CFLAG_COMPLETED;
                        pkt = CMD2PKT(cmd);
                        mptsas_pkt_comp(pkt, cmd);
                        cmd = next;
                }
                mutex_enter(&rpqp->rpq_mutex);
        }
}

/*
```



```
 * These routines manipulate the target's queue of pending requests
 */
void
mptsas_waitq_add(mptsas_t *mpt, mptsas_cmd_t *cmd)
{
        NDBG7(("mptsas_waitq_add: cmd=0x%p", (void *)cmd));
        mptsas_target_t *ptgt = cmd->cmd_tgt_addr;
        cmd->cmd_queued = TRUE;
        if (ptgt)
                ptgt->m_t_nwait++;
        if (cmd->cmd_pkt_flags & FLAG_HEAD) {
                if ((cmd->cmd_linkp = mpt->m_waitq) == NULL) {
                        mpt->m_waitqtail = &cmd->cmd_linkp;
                }
                mpt->m_waitq = cmd;
        } else {
                cmd->cmd_linkp = NULL;
                *(mpt->m_waitqtail) = cmd;
                mpt->m_waitqtail = &cmd->cmd_linkp;
        }
}

static mptsas_cmd_t *
mptsas_waitq_rm(mptsas_t *mpt)
{
        mptsas_cmd_t    *cmd;
        mptsas_target_t *ptgt;
        NDBG7(("mptsas_waitq_rm"));

        MPTSAS_WAITQ_RM(mpt, cmd);

        NDBG7(("mptsas_waitq_rm: cmd=0x%p", (void *)cmd));
        if (cmd) {
                ptgt = cmd->cmd_tgt_addr;
                if (ptgt) {
                        ptgt->m_t_nwait--;
                        ASSERT(ptgt->m_t_nwait >= 0);
                }
        }
        return (cmd);
}

/*
 * remove specified cmd from the middle of the wait queue.
 */
static void
mptsas_waitq_delete(mptsas_t *mpt, mptsas_cmd_t *cmd)
{
        mptsas_cmd_t    *prevp = mpt->m_waitq;
        mptsas_target_t *ptgt = cmd->cmd_tgt_addr;

        NDBG7(("mptsas_waitq_delete: mpt=0x%p cmd=0x%p",
            (void *)mpt, (void *)cmd));
        if (ptgt) {
                ptgt->m_t_nwait--;
                ASSERT(ptgt->m_t_nwait >= 0);
        }

        if (prevp == cmd) {
                if ((mpt->m_waitq = cmd->cmd_linkp) == NULL)
                        mpt->m_waitqtail = &mpt->m_waitq;

                cmd->cmd_linkp = NULL;
                cmd->cmd_queued = FALSE;
                NDBG7(("mptsas_waitq_delete: mpt=0x%p cmd=0x%p",
                    (void *)mpt, (void *)cmd));
```

```
                return;
        }

        while (prevp != NULL) {
                if (prevp->cmd_linkp == cmd) {
                        if ((prevp->cmd_linkp = cmd->cmd_linkp) == NULL)
                                mpt->m_waitqtail = &prevp->cmd_linkp;

                        cmd->cmd_linkp = NULL;
                        cmd->cmd_queued = FALSE;
                        NDBG7(("mptsas_waitq_delete: mpt=0x%p cmd=0x%p",
                            (void *)mpt, (void *)cmd));
                        return;
                }
                prevp = prevp->cmd_linkp;
        }
        cmn_err(CE_PANIC, "mpt: mptsas_waitq_delete: queue botch");
}

/*
 * device and bus reset handling
 *
 * Notes:
 *      - RESET_ALL:    reset the controller
 *      - RESET_TARGET: reset the target specified in scsi_address
 */
static int
mptsas_scsi_reset(struct scsi_address *ap, int level)
{
        mptsas_t                *mpt = ADDR2MPT(ap);
        int                     rval;
        mptsas_tgt_private_t    *tgt_private;
        mptsas_target_t         *ptgt = NULL;

        tgt_private = (mptsas_tgt_private_t *)ap->a_hba_tran->tran_tgt_private;
        ptgt = tgt_private->t_private;
        if (ptgt == NULL) {
                return (FALSE);
        }
        NDBG22(("mptsas_scsi_reset: target=%d level=%d", ptgt->m_devhdl,
            level));

        mutex_enter(&mpt->m_mutex);
        /*
         * if we are not in panic set up a reset delay for this target
         */
        if (!ddi_in_panic()) {
                mptsas_setup_bus_reset_delay(mpt);
        } else {
                drv_usecwait(mpt->m_scsi_reset_delay * 1000);
        }
        rval = mptsas_do_scsi_reset(mpt, ptgt->m_devhdl);
        mutex_exit(&mpt->m_mutex);

        /*
         * The transport layer expect to only see TRUE and
         * FALSE. Therefore, we will adjust the return value
         * if mptsas_do_scsi_reset returns FAILED.
         */
        if (rval == FAILED)
                rval = FALSE;
        return (rval);
}

static int
mptsas_do_scsi_reset(mptsas_t *mpt, uint16_t devhdl)
```



```
{
        int             rval = FALSE;
        uint8_t         config, disk;

        ASSERT(mutex_owned(&mpt->m_mutex));

        if (mptsas_debug_resets) {
                mptsas_log(mpt, CE_WARN, "mptsas_do_scsi_reset: target=%d",
                    devhdl);
        }

        /*
         * Issue a Target Reset message to the target specified but not to a
         * disk making up a raid volume.  Just look through the RAID config
         * Phys Disk list of DevHandles.  If the target’s DevHandle is in this
         * list, then don’t reset this target.
         */
        for (config = 0; config < mpt->m_num_raid_configs; config++) {
                for (disk = 0; disk < MPTSAS_MAX_DISKS_IN_CONFIG; disk++) {
                        if (devhdl == mpt->m_raidconfig[config].
                            m_physdisk_devhdl[disk]) {
                                return (TRUE);
                        }
                }
        }

        rval = mptsas_ioc_task_management(mpt,
            MPI2_SCSITASKMGMT_TASKTYPE_TARGET_RESET, devhdl, 0, NULL, 0, 0);

        mptsas_doneq_empty(mpt);
        return (rval);
}

static int
mptsas_scsi_reset_notify(struct scsi_address *ap, int flag,
        void (*callback)(caddr_t), caddr_t arg)
{
        mptsas_t        *mpt = ADDR2MPT(ap);

        NDBG22(("mptsas_scsi_reset_notify: tgt=%d", ap->a_target));

        return (scsi_hba_reset_notify_setup(ap, flag, callback, arg,
            &mpt->m_mutex, &mpt->m_reset_notify_listf));
}

static int
mptsas_get_name(struct scsi_device *sd, char *name, int len)
{
        dev_info_t      *lun_dip = NULL;

        ASSERT(sd != NULL);
        ASSERT(name != NULL);
        lun_dip = sd->sd_dev;
        ASSERT(lun_dip != NULL);

        if (mptsas_name_child(lun_dip, name, len) == DDI_SUCCESS) {
                return (1);
        } else {
                return (0);
        }
}

static int
mptsas_get_bus_addr(struct scsi_device *sd, char *name, int len)
{
        return (mptsas_get_name(sd, name, len));
```

```
}

static void
mptsas_set_throttle(mptsas_t *mpt, mptsas_target_t *ptgt, int what)
{

        NDBG25(("mptsas_set_throttle: throttle=%x", what));

        /*
         * if the bus is draining/quiesced, no changes to the throttles
         * are allowed. Not allowing change of throttles during draining
         * limits error recovery but will reduce draining time
         *
         * all throttles should have been set to HOLD_THROTTLE
         */
        if (mpt->m_softstate & (MPTSAS_SS_QUIESCED | MPTSAS_SS_DRAINING)) {
                return;
        }

        if (what == HOLD_THROTTLE) {
                ptgt->m_t_throttle = HOLD_THROTTLE;
        } else if (ptgt->m_reset_delay == 0) {
                ptgt->m_t_throttle = what;
        }
}

static void
mptsas_set_throttle_mtx(mptsas_t *mpt, mptsas_target_t *ptgt, int what)
{
        if (mpt->m_softstate & (MPTSAS_SS_QUIESCED | MPTSAS_SS_DRAINING)) {
                return;
        }

        mutex_enter(&ptgt->m_t_mutex);
        mptsas_set_throttle(mpt, ptgt, what);
        mutex_exit(&ptgt->m_t_mutex);
}

/*
 * Find all commands in the tx_waitq's for target and lun (if lun not -1),
 * remove them from the queues and return the linked list.
 */
static mptsas_cmd_t *
mptsas_strip_targetlun_from_txwqs(mptsas_t *mpt, ushort_t target, int lun)
{
        mptsas_cmd_t            *cmd, *clist, **tailp, **prev_tailp;
        mptsas_tx_waitqueue_t   *txwq;
        int                     i;

        clist = NULL;
        tailp = &clist;

        for (i = 0; i < NUM_TX_WAITQ; i++) {
                txwq = &mpt->m_tx_waitq[i];
                mutex_enter(&txwq->txwq_mutex);
                prev_tailp = &txwq->txwq_cmdq;
                cmd = txwq->txwq_cmdq;
                while (cmd != NULL) {
                        if (Tgt(cmd) == target &&
                            (lun == -1 || (Lun(cmd) == lun))) {
                                *prev_tailp = cmd->cmd_linkp;
                                *tailp = cmd;
                                tailp = &cmd->cmd_linkp;
                                cmd = cmd->cmd_linkp;
                                *tailp = NULL;
                        } else {
```



```
                                prev_tailp = &cmd->cmd_linkp;
                                cmd = cmd->cmd_linkp;
                        }
                }
                txwq->txwq_qtail = prev_tailp;
                mutex_exit(&txwq->txwq_mutex);
        }
        return (clist);
}

/*
 * 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;

        NDBG25(("mptsas_flush_target: target=%d lun=%d", target, lun));

        timestamp = gethrtime();

        /*
         * 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;
                                }
                                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;
                                }

                                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;
                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);
                while (cmd != NULL) {
                        next_cmd = cmd->cmd_linkp;
                        mptsas_set_pkt_reason(mpt, cmd, reason, stat);
                        mptsas_doneq_add(mpt, cmd);
                        cmd = next_cmd;
                }
                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);
                                mptsas_doneq_add(mpt, cmd);
                        }
                        cmd = next_cmd;
                }
                cmd = mptsas_strip_targetlun_from_txwqs(mpt, target, lun);
                while (cmd != NULL) {
                        next_cmd = cmd->cmd_linkp;
```



```
                        mptsas_set_pkt_reason(mpt, cmd, reason, stat);
                        mptsas_doneq_add(mpt, cmd);
                        cmd = next_cmd;
                }
                break;
        default:
                mptsas_log(mpt, CE_WARN, "Unknown task management type %d.",
                    tasktype);
                break;
        }
}

/*
 * Clean up hba state, abort all outstanding command and commands in waitq
 * reset timeout of all targets.
 */
static void
mptsas_flush_hba(mptsas_t *mpt)
{
        mptsas_slots_t  *slots = mpt->m_active;
        mptsas_cmd_t    *cmd, *ncmd;
        int             slot, i;

        NDBG25(("mptsas_flush_hba"));

        /*
         * The I/O Controller should have already sent back
         * all commands via the scsi I/O reply frame.  Make
         * sure all commands have been flushed.
         * Account for TM request, which use the last SMID.
         */
        for (slot = 0; slot <= mpt->m_active->m_n_normal; slot++) {
                if ((cmd = slots->m_slot[slot]) == NULL)
                        continue;

                if (cmd->cmd_flags & CFLAG_CMDIOC) {
                        /*
                         * Need to make sure to tell everyone that might be
                         * waiting on this command that it's going to fail.  If
                         * we get here, this command will never timeout because
                         * the active command table is going to be re-allocated,
                         * so there will be nothing to check against a time out.
                         * Instead, mark the command as failed due to reset.
                         */
                        mptsas_set_pkt_reason(mpt, cmd, CMD_RESET,
                            STAT_BUS_RESET);
                        if ((cmd->cmd_flags &
                            (CFLAG_PASSTHRU | CFLAG_CONFIG | CFLAG_FW_DIAG))) {
                                cmd->cmd_flags |= CFLAG_FINISHED;
                                cv_broadcast(&mpt->m_passthru_cv);
                                cv_broadcast(&mpt->m_config_cv);
                                cv_broadcast(&mpt->m_fw_diag_cv);
                        }
                        continue;
                }

                NDBG25(("mptsas_flush_hba discovered non-NULL cmd in slot %d",
                    slot));
                mptsas_dump_cmd(mpt, cmd);

                mptsas_remove_cmd(mpt, cmd);
                mptsas_set_pkt_reason(mpt, cmd, CMD_RESET, STAT_BUS_RESET);
                mptsas_doneq_add(mpt, cmd);
        }

        /*
```

```
         * Flush the waitq.
         */
        while ((cmd = mptsas_waitq_rm(mpt)) != NULL) {
                mptsas_set_pkt_reason(mpt, cmd, CMD_RESET, STAT_BUS_RESET);
                if ((cmd->cmd_flags & CFLAG_PASSTHRU) ||
                    (cmd->cmd_flags & CFLAG_CONFIG) ||
                    (cmd->cmd_flags & CFLAG_FW_DIAG)) {
                        cmd->cmd_flags |= CFLAG_FINISHED;
                        cv_broadcast(&mpt->m_passthru_cv);
                        cv_broadcast(&mpt->m_config_cv);
                        cv_broadcast(&mpt->m_fw_diag_cv);
                } else {
                        mptsas_doneq_add(mpt, cmd);
                }
        }

        /*
         * Flush the tx_waitqs
         */
        for (i = 0; i < NUM_TX_WAITQ; i++) {
                mutex_enter(&mpt->m_tx_waitq[i].txwq_mutex);
                cmd = mpt->m_tx_waitq[i].txwq_cmdq;
                mpt->m_tx_waitq[i].txwq_cmdq = NULL;
                mpt->m_tx_waitq[i].txwq_qtail = &mpt->m_tx_waitq[i].txwq_cmdq;
                mutex_exit(&mpt->m_tx_waitq[i].txwq_mutex);
                while (cmd != NULL) {
                        ncmd = cmd->cmd_linkp;
                        mptsas_set_pkt_reason(mpt, cmd, CMD_RESET,
                            STAT_BUS_RESET);
                        mptsas_doneq_add(mpt, cmd);
                        cmd = ncmd;
                }
        }

        /*
         * Drain the taskqs prior to reallocating resources.
         */
        mutex_exit(&mpt->m_mutex);
        ddi_taskq_wait(mpt->m_event_taskq);
        ddi_taskq_wait(mpt->m_dr_taskq);
        mutex_enter(&mpt->m_mutex);
}

/*
 * set pkt_reason and OR in pkt_statistics flag
 */
static void
mptsas_set_pkt_reason(mptsas_t *mpt, mptsas_cmd_t *cmd, uchar_t reason,
    uint_t stat)
{
#ifndef __lock_lint
        _NOTE(ARGUNUSED(mpt))
#endif

        NDBG25(("mptsas_set_pkt_reason: cmd=0x%p reason=%x stat=%x",
            (void *)cmd, reason, stat));

        if (cmd) {
                if (cmd->cmd_pkt->pkt_reason == CMD_CMPLT) {
                        cmd->cmd_pkt->pkt_reason = reason;
                }
                cmd->cmd_pkt->pkt_statistics |= stat;
        }
}

static void
```



```
mptsas_start_watch_reset_delay()
{
        NDBG22(("mptsas_start_watch_reset_delay"));

        mutex_enter(&mptsas_global_mutex);
        if (mptsas_reset_watch == NULL && mptsas_timeouts_enabled) {
                mptsas_reset_watch = timeout(mptsas_watch_reset_delay, NULL,
                    drv_usectohz((clock_t)
                    MPTSAS_WATCH_RESET_DELAY_TICK * 1000));
                ASSERT(mptsas_reset_watch != NULL);
        }
        mutex_exit(&mptsas_global_mutex);
}

static void
mptsas_setup_bus_reset_delay(mptsas_t *mpt)
{
        mptsas_target_t *ptgt = NULL;

        ASSERT(MUTEX_HELD(&mpt->m_mutex));

        NDBG22(("mptsas_setup_bus_reset_delay"));
        for (ptgt = refhash_first(mpt->m_targets); ptgt != NULL;
            ptgt = refhash_next(mpt->m_targets, ptgt)) {
                mutex_enter(&ptgt->m_t_mutex);
                mptsas_set_throttle(mpt, ptgt, HOLD_THROTTLE);
                ptgt->m_reset_delay = mpt->m_scsi_reset_delay;
                mutex_exit(&ptgt->m_t_mutex);
        }

        mptsas_start_watch_reset_delay();
}

/*
 * mptsas_watch_reset_delay(_subr) is invoked by timeout() and checks every
 * mpt instance for active reset delays
 */
static void
mptsas_watch_reset_delay(void *arg)
{
#ifndef __lock_lint
        _NOTE(ARGUNUSED(arg))
#endif

        mptsas_t        *mpt;
        int             not_done = 0;

        NDBG22(("mptsas_watch_reset_delay"));

        mutex_enter(&mptsas_global_mutex);
        mptsas_reset_watch = 0;
        mutex_exit(&mptsas_global_mutex);
        rw_enter(&mptsas_global_rwlock, RW_READER);
        for (mpt = mptsas_head; mpt != NULL; mpt = mpt->m_next) {
                if (mpt->m_tran == 0) {
                        continue;
                }
                mutex_enter(&mpt->m_mutex);
                not_done += mptsas_watch_reset_delay_subr(mpt);
                mutex_exit(&mpt->m_mutex);
        }
        rw_exit(&mptsas_global_rwlock);

        if (not_done) {
                mptsas_start_watch_reset_delay();
        }
```

```
}

static int
mptsas_watch_reset_delay_subr(mptsas_t *mpt)
{
        int             done = 0;
        int             restart = 0;
        mptsas_target_t *ptgt = NULL;

        NDBG22(("mptsas_watch_reset_delay_subr: mpt=0x%p", (void *)mpt));

        ASSERT(mutex_owned(&mpt->m_mutex));

        for (ptgt = refhash_first(mpt->m_targets); ptgt != NULL;
            ptgt = refhash_next(mpt->m_targets, ptgt)) {
                mutex_enter(&ptgt->m_t_mutex);
                if (ptgt->m_reset_delay != 0) {
                        ptgt->m_reset_delay -=
                            MPTSAS_WATCH_RESET_DELAY_TICK;
                        if (ptgt->m_reset_delay <= 0) {
                                ptgt->m_reset_delay = 0;
                                mptsas_set_throttle(mpt, ptgt,
                                    MAX_THROTTLE);
                                restart++;
                        } else {
                                done = -1;
                        }
                }
                mutex_exit(&ptgt->m_t_mutex);
        }

        if (restart > 0) {
                mptsas_restart_hba(mpt);
        }
        return (done);
}

#ifdef MPTSAS_TEST
static void
mptsas_test_reset(mptsas_t *mpt, int target)
{
        mptsas_target_t    *ptgt = NULL;

        if (mptsas_rtest == target) {
                if (mptsas_do_scsi_reset(mpt, target) == TRUE) {
                        mptsas_rtest = -1;
                }
                if (mptsas_rtest == -1) {
                        NDBG22(("mptsas_test_reset success"));
                }
        }
}
#endif

/*
 * abort handling:
 *
 * Notes:
 *      - if pkt is not NULL, abort just that command
 *      - if pkt is NULL, abort all outstanding commands for target
 */
static int
mptsas_scsi_abort(struct scsi_address *ap, struct scsi_pkt *pkt)
{
        mptsas_t                *mpt = ADDR2MPT(ap);
        int                     rval;
```



```
        mptsas_tgt_private_t    *tgt_private;
        int                     target, lun;

        tgt_private = (mptsas_tgt_private_t *)ap->a_hba_tran->
            tran_tgt_private;
        ASSERT(tgt_private != NULL);
        target = tgt_private->t_private->m_devhdl;
        lun = tgt_private->t_lun;

        NDBG23(("mptsas_scsi_abort: target=%d.%d", target, lun));

        mutex_enter(&mpt->m_mutex);
        rval = mptsas_do_scsi_abort(mpt, target, lun, pkt);
        mutex_exit(&mpt->m_mutex);
        return (rval);
}

static int
mptsas_do_scsi_abort(mptsas_t *mpt, int target, int lun, struct scsi_pkt *pkt)
{
        mptsas_cmd_t    *sp = NULL;
        mptsas_slots_t  *slots = mpt->m_active;
        int             rval = FALSE;

        ASSERT(mutex_owned(&mpt->m_mutex));

        /*
         * Abort the command pkt on the target/lun in ap.  If pkt is
         * NULL, abort all outstanding commands on that target/lun.
         * If you can abort them, return 1, else return 0.
         * Each packet that's aborted should be sent back to the target
         * driver through the callback routine, with pkt_reason set to
         * CMD_ABORTED.
         *
         * abort cmd pkt on HBA hardware; clean out of outstanding
         * command lists, etc.
         */
        if (pkt != NULL) {
                /* abort the specified packet */
                sp = PKT2CMD(pkt);

                if (sp->cmd_queued) {
                        NDBG23(("mptsas_do_scsi_abort: queued sp=0x%p aborted",
                            (void *)sp));
                        mptsas_waitq_delete(mpt, sp);
                        mptsas_set_pkt_reason(mpt, sp, CMD_ABORTED,
                            STAT_ABORTED);
                        mptsas_doneq_add(mpt, sp);
                        rval = TRUE;
                        goto done;
                }

                /*
                 * Have mpt firmware abort this command
                 */

                if (slots->m_slot[sp->cmd_slot] != NULL) {
                        rval = mptsas_ioc_task_management(mpt,
                            MPI2_SCSITASKMGMT_TASKTYPE_ABORT_TASK, target,
                            lun, NULL, 0, 0);

                        /*
                         * The transport layer expects only TRUE and FALSE.
                         * Therefore, if mptsas_ioc_task_management returns
                         * FAILED we will return FALSE.
                         */
```

```
                        if (rval == FAILED)
                                rval = FALSE;
                        goto done;
                }
        }

        /*
         * If pkt is NULL then abort task set
         */
        rval = mptsas_ioc_task_management(mpt,
            MPI2_SCSITASKMGMT_TASKTYPE_ABRT_TASK_SET, target, lun, NULL, 0, 0);

        /*
         * The transport layer expects only TRUE and FALSE.
         * Therefore, if mptsas_ioc_task_management returns
         * FAILED we will return FALSE.
         */
        if (rval == FAILED)
                rval = FALSE;

#ifdef MPTSAS_TEST
        if (rval && mptsas_test_stop) {
                debug_enter("mptsas_do_scsi_abort");
        }
#endif

done:
        mptsas_doneq_empty(mpt);
        return (rval);
}

/*
 * capability handling:
 * (*tran_getcap).  Get the capability named, and return its value.
 */
static int
mptsas_scsi_getcap(struct scsi_address *ap, char *cap, int tgtonly)
{
        mptsas_t        *mpt = ADDR2MPT(ap);
        int             ckey;
        int             rval = FALSE;

        NDBG24(("mptsas_scsi_getcap: target=%d, cap=%s tgtonly=%x",
            ap->a_target, cap, tgtonly));

        mutex_enter(&mpt->m_mutex);

        if ((mptsas_scsi_capchk(cap, tgtonly, &ckey)) != TRUE) {
                mutex_exit(&mpt->m_mutex);
                return (UNDEFINED);
        }

        switch (ckey) {
        case SCSI_CAP_DMA_MAX:
                rval = (int)mpt->m_msg_dma_attr.dma_attr_maxxfer;
                break;
        case SCSI_CAP_ARQ:
                rval = TRUE;
                break;
        case SCSI_CAP_MSG_OUT:
        case SCSI_CAP_PARITY:
        case SCSI_CAP_UNTAGGED_QING:
                rval = TRUE;
                break;
        case SCSI_CAP_TAGGED_QING:
                rval = TRUE;
```



```
                break;
        case SCSI_CAP_RESET_NOTIFICATION:
                rval = TRUE;
                break;
        case SCSI_CAP_LINKED_CMDS:
                rval = FALSE;
                break;
        case SCSI_CAP_QFULL_RETRIES:
                rval = ((mptsas_tgt_private_t *)(ap->a_hba_tran->
                    tran_tgt_private))->t_private->m_qfull_retries;
                break;
        case SCSI_CAP_QFULL_RETRY_INTERVAL:
                rval = drv_hztousec(((mptsas_tgt_private_t *)
                    (ap->a_hba_tran->tran_tgt_private))->
                    t_private->m_qfull_retry_interval) / 1000;
                break;
        case SCSI_CAP_CDB_LEN:
                rval = CDB_GROUP4;
                break;
        case SCSI_CAP_INTERCONNECT_TYPE:
                rval = INTERCONNECT_SAS;
                break;
        case SCSI_CAP_TRAN_LAYER_RETRIES:
                if (mpt->m_ioc_capabilities &
                    MPI2_IOCFACTS_CAPABILITY_TLR)
                        rval = TRUE;
                else
                        rval = FALSE;
                break;
        default:
                rval = UNDEFINED;
                break;
        }

        NDBG24(("mptsas_scsi_getcap: %s, rval=%x", cap, rval));

        mutex_exit(&mpt->m_mutex);
        return (rval);
}

/*
 * (*tran_setcap).  Set the capability named to the value given.
 */
static int
mptsas_scsi_setcap(struct scsi_address *ap, char *cap, int value, int tgtonly)
{
        mptsas_t        *mpt = ADDR2MPT(ap);
        mptsas_target_t *ptgt;
        int             ckey;
        int             rval = FALSE;

        NDBG24(("mptsas_scsi_setcap: target=%d, cap=%s value=%x tgtonly=%x",
            ap->a_target, cap, value, tgtonly));

        if (!tgtonly) {
                return (rval);
        }

        mutex_enter(&mpt->m_mutex);

        if ((mptsas_scsi_capchk(cap, tgtonly, &ckey)) != TRUE) {
                mutex_exit(&mpt->m_mutex);
                return (UNDEFINED);
        }

        switch (ckey) {
```

```
        case SCSI_CAP_DMA_MAX:
        case SCSI_CAP_MSG_OUT:
        case SCSI_CAP_PARITY:
        case SCSI_CAP_INITIATOR_ID:
        case SCSI_CAP_LINKED_CMDS:
        case SCSI_CAP_UNTAGGED_QING:
        case SCSI_CAP_RESET_NOTIFICATION:
                /*
                 * None of these are settable via
                 * the capability interface.
                 */
                break;
        case SCSI_CAP_ARQ:
                /*
                 * We cannot turn off arq so return false if asked to
                 */
                if (value) {
                        rval = TRUE;
                } else {
                        rval = FALSE;
                }
                break;
        case SCSI_CAP_TAGGED_QING:
                ptgt = ((mptsas_tgt_private_t *)
                    (ap->a_hba_tran->tran_tgt_private))->t_private;
                mptsas_set_throttle_mtx(mpt, ptgt, MAX_THROTTLE);
                rval = TRUE;
                break;
        case SCSI_CAP_QFULL_RETRIES:
                ((mptsas_tgt_private_t *)(ap->a_hba_tran->tran_tgt_private))->
                    t_private->m_qfull_retries = (uchar_t)value;
                rval = TRUE;
                break;
        case SCSI_CAP_QFULL_RETRY_INTERVAL:
                ((mptsas_tgt_private_t *)(ap->a_hba_tran->tran_tgt_private))->
                    t_private->m_qfull_retry_interval =
                    drv_usectohz(value * 1000);
                rval = TRUE;
                break;
        default:
                rval = UNDEFINED;
                break;
        }
        mutex_exit(&mpt->m_mutex);
        return (rval);
}

/*
 * Utility routine for mptsas_ifsetcap/ifgetcap
 */
/*ARGSUSED*/
static int
mptsas_scsi_capchk(char *cap, int tgtonly, int *cidxp)
{
        NDBG24(("mptsas_scsi_capchk: cap=%s", cap));

        if (!cap)
                return (FALSE);

        *cidxp = scsi_hba_lookup_capstr(cap);
        return (TRUE);
}

static int
mptsas_alloc_active_slots(mptsas_t *mpt, int flag)
{
```



```
        mptsas_slots_t  *old_active = mpt->m_active;
        mptsas_slots_t  *new_active;
        size_t          size;

        /*
         * if there are active commands, then we cannot
         * change size of active slots array.
         */
        ASSERT(mpt->m_ncmds == 0);

        size = MPTSAS_SLOTS_SIZE(mpt);
        new_active = kmem_zalloc(size, flag);
        if (new_active == NULL) {
                NDBG1(("new active alloc failed"));
                return (-1);
        }
        /*
         * Since SMID 0 is reserved and the TM slot is reserved, the
         * number of slots that can be used at any one time is
         * m_max_requests - 2.
         */
        new_active->m_n_normal = (mpt->m_max_requests - 2);
        new_active->m_size = size;
        new_active->m_rotor = 1;
        if (old_active)
                mptsas_free_active_slots(mpt);
        mpt->m_active = new_active;

        return (0);
}

static void
mptsas_free_active_slots(mptsas_t *mpt)
{
        mptsas_slots_t  *active = mpt->m_active;
        size_t          size;

        if (active == NULL)
                return;
        size = active->m_size;
        kmem_free(active, size);
        mpt->m_active = NULL;
}

/*
 * Error logging, printing, and debug print routines.
 */
static char *mptsas_label = "mpt_sas3";

/*PRINTFLIKE3*/
void
mptsas_log(mptsas_t *mpt, int level, char *fmt, ...)
{
        dev_info_t      *dev;
        va_list         ap;

        if (mpt) {
                dev = mpt->m_dip;
        } else {
                dev = 0;
        }

        mutex_enter(&mptsas_log_mutex);

        va_start(ap, fmt);
        (void) vsprintf(mptsas_log_buf, fmt, ap);
```

```
        va_end(ap);

        if (level == CE_CONT) {
                scsi_log(dev, mptsas_label, level, "%s\n", mptsas_log_buf);
        } else {
                scsi_log(dev, mptsas_label, level, "%s", mptsas_log_buf);
        }

        mutex_exit(&mptsas_log_mutex);
}

#ifdef MPTSAS_DEBUG
/*
 * Use a circular buffer to log messages to private memory.
 * No mutexes, so there is the opportunity for this to miss lines.
 * But it’s fast and does not hold up the proceedings too much.
 */
static char mptsas_dbglog_bufs[32][256];
static uint32_t mptsas_dbglog_idx = 1;

/*PRINTFLIKE1*/
void
mptsas_debug_log(char *fmt, ...)
{
        va_list         ap;
        uint32_t        idx;

        if (!mptsas_dbglog_idx) {
                return;
        }
        idx = (mptsas_dbglog_idx++) & 0x1f;

        va_start(ap, fmt);
        (void) vsnprintf(mptsas_dbglog_bufs[idx],
            sizeof (mptsas_dbglog_bufs[0]), fmt, ap);
        va_end(ap);
}

/*PRINTFLIKE1*/
void
mptsas_printf(char *fmt, ...)
{
        dev_info_t      *dev = 0;
        va_list         ap;

        mutex_enter(&mptsas_log_mutex);

        va_start(ap, fmt);
        (void) vsprintf(mptsas_log_buf, fmt, ap);
        va_end(ap);

#ifdef PROM_PRINTF
        prom_printf("%s:\t%s\n", mptsas_label, mptsas_log_buf);
#else
        scsi_log(dev, mptsas_label, CE_CONT, "!%s\n", mptsas_log_buf);
#endif
        mutex_exit(&mptsas_log_mutex);
}
#endif

/*
 * timeout handling
 */
static void
mptsas_watch(void *arg)
{
```



```
#ifndef __lock_lint
        _NOTE(ARGUNUSED(arg))
#endif

        mptsas_t        *mpt;
        uint32_t        doorbell;

        NDBG30(("mptsas_watch"));

        rw_enter(&mptsas_global_rwlock, RW_READER);
        for (mpt = mptsas_head; mpt != (mptsas_t *)NULL; mpt = mpt->m_next) {

                mutex_enter(&mpt->m_mutex);

                /* Skip device if not powered on */
                if (mpt->m_options & MPTSAS_OPT_PM) {
                        if (mpt->m_power_level == PM_LEVEL_D0) {
                                (void) pm_busy_component(mpt->m_dip, 0);
                                mpt->m_busy = 1;
                        } else {
                                mutex_exit(&mpt->m_mutex);
                                continue;
                        }
                }

                /*
                 * Check if controller is in a FAULT state. If so, reset it.
                 */
                doorbell = ddi_get32(mpt->m_datap, &mpt->m_reg->Doorbell);
                if ((doorbell & MPI2_IOC_STATE_MASK) == MPI2_IOC_STATE_FAULT) {
                        doorbell &= MPI2_DOORBELL_DATA_MASK;
                        mptsas_log(mpt, CE_WARN, "MPT Firmware Fault, "
                            "code: %04x", doorbell);
                        mpt->m_softstate &= ~MPTSAS_SS_MSG_UNIT_RESET;
                        if ((mptsas_restart_ioc(mpt)) == DDI_FAILURE) {
                                mptsas_log(mpt, CE_WARN, "Reset failed"
                                    "after fault was detected");
                        }
                }

                /*
                 * For now, always call mptsas_watchsubr.
                 */
                mptsas_watchsubr(mpt);

                if (mpt->m_options & MPTSAS_OPT_PM) {
                        mpt->m_busy = 0;
                        (void) pm_idle_component(mpt->m_dip, 0);
                }

                mutex_exit(&mpt->m_mutex);
        }
        rw_exit(&mptsas_global_rwlock);

        mutex_enter(&mptsas_global_mutex);
        if (mptsas_timeouts_enabled)
                mptsas_timeout_id = timeout(mptsas_watch, NULL, mptsas_tick);
        mutex_exit(&mptsas_global_mutex);
}

int mptsas_monitor_for_txwqs = 1;
static void
mptsas_watchsubr(mptsas_t *mpt)
{
        int             i;
        mptsas_cmd_t    *cmd;
```

```
        mptsas_target_t *ptgt = NULL;
        hrtime_t        timestamp = gethrtime();
        boolean_t       restart_hba = B_FALSE;

        ASSERT(MUTEX_HELD(&mpt->m_mutex));

        NDBG30(("mptsas_watchsubr: mpt=0x%p, ncmds %d, nstarted %d",
            (void *)mpt, mpt->m_ncmds, mpt->m_ncstarted));

        mpt->m_lncstarted = mpt->m_ncstarted;
        if (mpt->m_txwq_thread_n != 0 && mpt->m_txwq_enabled != BLOCKED &&
            mptsas_monitor_for_txwqs) {
                i = mpt->m_ncstarted/mptsas_scsi_watchdog_tick;
                if (i > mpt->m_txwq_thread_threshold) {
                        mpt->m_txwq_enabled = TRUE;
                } else if (i < (mpt->m_txwq_thread_threshold>>1)) {
                        mpt->m_txwq_enabled = FALSE;
                }
        }
        mpt->m_ncstarted = 0;

        /*
         * Check for commands stuck in active slot
         * Account for TM requests, which use the last SMID.
         */
        for (i = 0; i <= mpt->m_active->m_n_normal; i++) {
                if ((cmd = mpt->m_active->m_slot[i]) != NULL) {
                        if (cmd->cmd_active_expiration <= timestamp) {
                                if ((cmd->cmd_flags & CFLAG_CMDIOC) == 0) {
                                        /*
                                         * There seems to be a command stuck
                                         * in the active slot.  Drain throttle.
                                         */
                                        ptgt = cmd->cmd_tgt_addr;
                                        mptsas_set_throttle_mtx(mpt, ptgt,
                                            DRAIN_THROTTLE);
                                } else if (cmd->cmd_flags &
                                    (CFLAG_PASSTHRU | CFLAG_CONFIG |
                                    CFLAG_FW_DIAG)) {
                                        /*
                                         * passthrough command timeout
                                         */
                                        cmd->cmd_flags |= (CFLAG_FINISHED |
                                            CFLAG_TIMEOUT);
                                        cv_broadcast(&mpt->m_passthru_cv);
                                        cv_broadcast(&mpt->m_config_cv);
                                        cv_broadcast(&mpt->m_fw_diag_cv);
                                }
                        }
                }
        }

        for (ptgt = refhash_first(mpt->m_targets); ptgt != NULL;
            ptgt = refhash_next(mpt->m_targets, ptgt)) {
                mutex_enter(&ptgt->m_t_mutex);
                /*
                 * If we were draining due to a qfull condition,
                 * go back to full throttle.
                 */
                if ((ptgt->m_t_throttle < MAX_THROTTLE) &&
                    (ptgt->m_t_throttle > HOLD_THROTTLE) &&
                    (ptgt->m_t_ncmds < ptgt->m_t_throttle)) {
                        mptsas_set_throttle(mpt, ptgt, MAX_THROTTLE);
                        restart_hba = B_TRUE;
                }
```



```
                cmd = TAILQ_LAST(&ptgt->m_active_cmdq, mptsas_active_cmdq);
                if (cmd != NULL) {
                        if (cmd->cmd_active_expiration <= timestamp) {
                                /*
                                 * Earliest command timeout expired.
                                 * Drain throttle.
                                 */
                                mptsas_set_throttle(mpt, ptgt, DRAIN_THROTTLE);

                                /*
                                 * Check for remaining commands.
                                 */
                                cmd = TAILQ_FIRST(&ptgt->m_active_cmdq);
                                if (cmd->cmd_active_expiration > timestamp) {
                                        /*
                                         * Wait for remaining commands to
                                         * complete or time out.
                                         */
                                        NDBG23(("command timed out, "
                                            "pending drain"));
                                } else {
                                        mutex_exit(&ptgt->m_t_mutex);

                                        /*
                                         * All command timeouts expired.
                                         */
                                        mptsas_log(mpt, CE_NOTE,
                                            "Timeout of %d seconds "
                                            "expired with %d commands on "
                                            "target %d lun %d.",
                                            cmd->cmd_pkt->pkt_time,
                                            ptgt->m_t_ncmds,
                                            ptgt->m_devhdl, Lun(cmd));

                                        mptsas_cmd_timeout(mpt, ptgt);
                                        continue;
                                }
                        } else if (cmd->cmd_active_expiration <= timestamp +
                            (hrtime_t)mptsas_scsi_watchdog_tick * NANOSEC) {
                                NDBG23(("pending timeout"));
                                mptsas_set_throttle(mpt, ptgt, DRAIN_THROTTLE);
                        }
                }
                mutex_exit(&ptgt->m_t_mutex);
        }
        if (restart_hba == B_TRUE) {
                mptsas_restart_hba(mpt);
        }
}

/*
 * timeout recovery
 */
static void
mptsas_cmd_timeout(mptsas_t *mpt, mptsas_target_t *ptgt)
{
        uint16_t        devhdl;
        uint64_t        sas_wwn;
        uint8_t         phy;
        char            wwn_str[MPTSAS_WWN_STRLEN];

        devhdl = ptgt->m_devhdl;
        sas_wwn = ptgt->m_addr.mta_wwn;
        phy = ptgt->m_phynum;
        if (sas_wwn == 0) {
                (void) sprintf(wwn_str, "p%x", phy);
```

```
        } else {
                (void) sprintf(wwn_str, "w%016"PRIx64, sas_wwn);
        }

        NDBG29(("mptsas_cmd_timeout: target=%d", devhdl));
        mptsas_log(mpt, CE_WARN, "Disconnected command timeout for "
            "target %d %s,  enclosure %u .", devhdl, wwn_str,
            ptgt->m_enclosure);

        /*
         * Abort all outstanding commands on the device.
         */
        NDBG29(("mptsas_cmd_timeout: device reset"));
        if (mptsas_do_scsi_reset(mpt, devhdl) != TRUE) {
                mptsas_log(mpt, CE_WARN, "Target %d reset for command timeout "
                    "recovery failed!", devhdl);
        }
}

/*
 * Device / Hotplug control
 */
static int
mptsas_scsi_quiesce(dev_info_t *dip)
{
        mptsas_t        *mpt;
        scsi_hba_tran_t *tran;

        tran = ddi_get_driver_private(dip);
        if (tran == NULL || (mpt = TRAN2MPT(tran)) == NULL)
                return (-1);

        return (mptsas_quiesce_bus(mpt));
}

static int
mptsas_scsi_unquiesce(dev_info_t *dip)
{
        mptsas_t                *mpt;
        scsi_hba_tran_t *tran;

        tran = ddi_get_driver_private(dip);
        if (tran == NULL || (mpt = TRAN2MPT(tran)) == NULL)
                return (-1);

        return (mptsas_unquiesce_bus(mpt));
}

static int
mptsas_quiesce_bus(mptsas_t *mpt)
{
        mptsas_target_t *ptgt = NULL;

        NDBG28(("mptsas_quiesce_bus"));
        mutex_enter(&mpt->m_mutex);

        /* Set all the throttles to zero */
        for (ptgt = refhash_first(mpt->m_targets); ptgt != NULL;
            ptgt = refhash_next(mpt->m_targets, ptgt)) {
                mptsas_set_throttle_mtx(mpt, ptgt, HOLD_THROTTLE);
        }

        /* If there are any outstanding commands in the queue */
        if (mpt->m_ncmds) {
                mpt->m_softstate |= MPTSAS_SS_DRAINING;
                mpt->m_quiesce_timeid = timeout(mptsas_ncmds_checkdrain,
```



```
                        mpt, (MPTSAS_QUIESCE_TIMEOUT * drv_usectohz(1000000)));
                if (cv_wait_sig(&mpt->m_cv, &mpt->m_mutex) == 0) {
                        /*
                         * Quiesce has been interrupted
                         */
                        mpt->m_softstate &= ~MPTSAS_SS_DRAINING;
                        for (ptgt = refhash_first(mpt->m_targets); ptgt != NULL;
                            ptgt = refhash_next(mpt->m_targets, ptgt)) {
                                mptsas_set_throttle_mtx(mpt, ptgt,
                                    MAX_THROTTLE);
                        }
                        mptsas_restart_hba(mpt);
                        if (mpt->m_quiesce_timeid != 0) {
                                timeout_id_t tid = mpt->m_quiesce_timeid;
                                mpt->m_quiesce_timeid = 0;
                                mutex_exit(&mpt->m_mutex);
                                (void) untimeout(tid);
                                return (-1);
                        }
                        mutex_exit(&mpt->m_mutex);
                        return (-1);
                } else {
                        /* Bus has been quiesced */
                        ASSERT(mpt->m_quiesce_timeid == 0);
                        mpt->m_softstate &= ~MPTSAS_SS_DRAINING;
                        mpt->m_softstate |= MPTSAS_SS_QUIESCED;
                        mutex_exit(&mpt->m_mutex);
                        return (0);
                }
        }
        /* Bus was not busy - QUIESCED */
        mutex_exit(&mpt->m_mutex);

        return (0);
}

static int
mptsas_unquiesce_bus(mptsas_t *mpt)
{
        mptsas_target_t *ptgt = NULL;

        NDBG28(("mptsas_unquiesce_bus"));
        mutex_enter(&mpt->m_mutex);
        mpt->m_softstate &= ~MPTSAS_SS_QUIESCED;
        for (ptgt = refhash_first(mpt->m_targets); ptgt != NULL;
            ptgt = refhash_next(mpt->m_targets, ptgt)) {
                mptsas_set_throttle_mtx(mpt, ptgt, MAX_THROTTLE);
        }
        mptsas_restart_hba(mpt);
        mutex_exit(&mpt->m_mutex);
        return (0);
}

static void
mptsas_ncmds_checkdrain(void *arg)
{
        mptsas_t        *mpt = arg;
        mptsas_target_t *ptgt = NULL;

        mutex_enter(&mpt->m_mutex);
        if (mpt->m_softstate & MPTSAS_SS_DRAINING) {
                mpt->m_quiesce_timeid = 0;
                if (mpt->m_ncmds == 0) {
                        /* Command queue has been drained */
                        cv_signal(&mpt->m_cv);
                } else {
```

```
                        /*
                         * The throttle may have been reset because
                         * of a SCSI bus reset
                         */
                        for (ptgt = refhash_first(mpt->m_targets); ptgt != NULL;
                            ptgt = refhash_next(mpt->m_targets, ptgt)) {
                                mptsas_set_throttle_mtx(mpt, ptgt,
                                    HOLD_THROTTLE);
                        }

                        mpt->m_quiesce_timeid = timeout(mptsas_ncmds_checkdrain,
                            mpt, (MPTSAS_QUIESCE_TIMEOUT *
                            drv_usectohz(1000000)));
                }
        }
        mutex_exit(&mpt->m_mutex);
}

/*ARGSUSED*/
static void
mptsas_dump_cmd(mptsas_t *mpt, mptsas_cmd_t *cmd)
{
        int     i;
        uint8_t *cp = (uchar_t *)cmd->cmd_pkt->pkt_cdbp;
        char    buf[128];

        buf[0] = '\0';
        NDBG25(("?Cmd (0x%p) dump for Target %d Lun %d:\n", (void *)cmd,
            Tgt(cmd), Lun(cmd)));
        (void) sprintf(&buf[0], "\tcdb=[");
        for (i = 0; i < (int)cmd->cmd_cdblen; i++) {
                (void) sprintf(&buf[strlen(buf)], " 0x%x", *cp++);
        }
        (void) sprintf(&buf[strlen(buf)], " ]");
        NDBG25(("?%s\n", buf));
        NDBG25(("?pkt_flags=0x%x pkt_statistics=0x%x pkt_state=0x%x\n",
            cmd->cmd_pkt->pkt_flags, cmd->cmd_pkt->pkt_statistics,
            cmd->cmd_pkt->pkt_state));
        NDBG25(("?pkt_scbp=0x%x cmd_flags=0x%x\n", cmd->cmd_pkt->pkt_scbp ?
            *(cmd->cmd_pkt->pkt_scbp) : 0, cmd->cmd_flags));
}

static void
mptsas_passthru_sge(ddi_acc_handle_t acc_hdl, mptsas_pt_request_t *pt,
    pMpi2SGESimple64_t sgep)
{
        uint32_t                sge_flags;
        uint32_t                data_size, dataout_size;
        ddi_dma_cookie_t        data_cookie;
        ddi_dma_cookie_t        dataout_cookie;

        data_size = pt->data_size;
        dataout_size = pt->dataout_size;
        data_cookie = pt->data_cookie;
        dataout_cookie = pt->dataout_cookie;

        if (dataout_size) {
                sge_flags = dataout_size |
                    ((uint32_t)(MPI2_SGE_FLAGS_SIMPLE_ELEMENT |
                    MPI2_SGE_FLAGS_END_OF_BUFFER |
                    MPI2_SGE_FLAGS_HOST_TO_IOC |
                    MPI2_SGE_FLAGS_64_BIT_ADDRESSING) <<
                    MPI2_SGE_FLAGS_SHIFT);
                ddi_put32(acc_hdl, &sgep->FlagsLength, sge_flags);
                ddi_put32(acc_hdl, &sgep->Address.Low,
                    (uint32_t)(dataout_cookie.dmac_laddress & 0xffffffffull));
```



```
                ddi_put32(acc_hdl, &sgep->Address.High,
                    (uint32_t)(dataout_cookie.dmac_laddress >> 32));
                sgep++;
        }
        sge_flags = data_size;
        sge_flags |= ((uint32_t)(MPI2_SGE_FLAGS_SIMPLE_ELEMENT |
            MPI2_SGE_FLAGS_LAST_ELEMENT |
            MPI2_SGE_FLAGS_END_OF_BUFFER |
            MPI2_SGE_FLAGS_END_OF_LIST |
            MPI2_SGE_FLAGS_64_BIT_ADDRESSING) <<
            MPI2_SGE_FLAGS_SHIFT);
        if (pt->direction == MPTSAS_PASS_THRU_DIRECTION_WRITE) {
                sge_flags |= ((uint32_t)(MPI2_SGE_FLAGS_HOST_TO_IOC) <<
                    MPI2_SGE_FLAGS_SHIFT);
        } else {
                sge_flags |= ((uint32_t)(MPI2_SGE_FLAGS_IOC_TO_HOST) <<
                    MPI2_SGE_FLAGS_SHIFT);
        }
        ddi_put32(acc_hdl, &sgep->FlagsLength, sge_flags);
        ddi_put32(acc_hdl, &sgep->Address.Low,
            (uint32_t)(data_cookie.dmac_laddress & 0xffffffffull));
        ddi_put32(acc_hdl, &sgep->Address.High,
            (uint32_t)(data_cookie.dmac_laddress >> 32));
}

static void
mptsas_passthru_ieee_sge(ddi_acc_handle_t acc_hdl, mptsas_pt_request_t *pt,
    pMpi2IeeeSgeSimple64_t ieeesgep)
{
        uint8_t                 sge_flags;
        uint32_t                data_size, dataout_size;
        ddi_dma_cookie_t        data_cookie;
        ddi_dma_cookie_t        dataout_cookie;

        data_size = pt->data_size;
        dataout_size = pt->dataout_size;
        data_cookie = pt->data_cookie;
        dataout_cookie = pt->dataout_cookie;

        sge_flags = (MPI2_IEEE_SGE_FLAGS_SIMPLE_ELEMENT |
            MPI2_IEEE_SGE_FLAGS_SYSTEM_ADDR);
        if (dataout_size) {
                ddi_put32(acc_hdl, &ieeesgep->Length, dataout_size);
                ddi_put32(acc_hdl, &ieeesgep->Address.Low,
                    (uint32_t)(dataout_cookie.dmac_laddress &
                    0xffffffffull));
                ddi_put32(acc_hdl, &ieeesgep->Address.High,
                    (uint32_t)(dataout_cookie.dmac_laddress >> 32));
                ddi_put8(acc_hdl, &ieeesgep->Flags, sge_flags);
                ieeesgep++;
        }
        sge_flags |= MPI25_IEEE_SGE_FLAGS_END_OF_LIST;
        ddi_put32(acc_hdl, &ieeesgep->Length, data_size);
        ddi_put32(acc_hdl, &ieeesgep->Address.Low,
            (uint32_t)(data_cookie.dmac_laddress & 0xffffffffull));
        ddi_put32(acc_hdl, &ieeesgep->Address.High,
            (uint32_t)(data_cookie.dmac_laddress >> 32));
        ddi_put8(acc_hdl, &ieeesgep->Flags, sge_flags);
}

static void
mptsas_start_passthru(mptsas_t *mpt, mptsas_cmd_t *cmd)
{
        caddr_t                 memp;
        pMPI2RequestHeader_t    request_hdrp;
        struct scsi_pkt         *pkt = cmd->cmd_pkt;
```

```
        mptsas_pt_request_t     *pt = pkt->pkt_ha_private;
        uint32_t                request_size;
        uint64_t                request_desc = 0;
        uint64_t                sense_bufp;
        uint8_t                 desc_type;
        uint8_t                 *request, function;
        ddi_dma_handle_t        dma_hdl = mpt->m_dma_req_frame_hdl;
        ddi_acc_handle_t        acc_hdl = mpt->m_acc_req_frame_hdl;

        desc_type = MPI2_REQ_DESCRIPT_FLAGS_DEFAULT_TYPE;

        request = pt->request;
        request_size = pt->request_size;

        /*
         * Store the passthrough message in memory location
         * corresponding to our slot number
         */
        memp = mpt->m_req_frame + (mpt->m_req_frame_size * cmd->cmd_slot);
        request_hdrp = (pMPI2RequestHeader_t)memp;
        bzero(memp, mpt->m_req_frame_size);

        bcopy(request, memp, request_size);

        NDBG15(("mptsas_start_passthru: Func 0x%x, MsgFlags 0x%x, "
            "size=%d, in %d, out %d", request_hdrp->Function,
            request_hdrp->MsgFlags, request_size,
            pt->data_size, pt->dataout_size));

        /*
         * Add an SGE, even if the length is zero.
         */
        if (mpt->m_MPI25 && pt->simple == 0) {
                mptsas_passthru_ieee_sge(acc_hdl, pt,
                    (pMpi2IeeeSgeSimple64_t)
                    ((uint8_t *)request_hdrp + pt->sgl_offset));
        } else {
                mptsas_passthru_sge(acc_hdl, pt,
                    (pMpi2SGESimple64_t)
                    ((uint8_t *)request_hdrp + pt->sgl_offset));
        }

        function = request_hdrp->Function;
        if ((function == MPI2_FUNCTION_SCSI_IO_REQUEST) ||
            (function == MPI2_FUNCTION_RAID_SCSI_IO_PASSTHROUGH)) {
                pMpi2SCSIIORequest_t    scsi_io_req;

                NDBG15(("mptsas_start_passthru: Is SCSI IO Req"));
                scsi_io_req = (pMpi2SCSIIORequest_t)request_hdrp;
                /*
                 * Put SGE for data and data_out buffer at the end of
                 * scsi_io_request message header.(64 bytes in total)
                 * Following above SGEs, the residual space will be
                 * used by sense data.
                 */
                ddi_put8(acc_hdl,
                    &scsi_io_req->SenseBufferLength,
                    (uint8_t)(request_size - 64));

                sense_bufp = (uint32_t)(mpt->m_req_frame_dma_addr +
                    (mpt->m_req_frame_size * cmd->cmd_slot) & 0xffffffffull);
                sense_bufp += 64;
                ddi_put32(acc_hdl,
                    &scsi_io_req->SenseBufferLowAddress, sense_bufp);

                /*
```



```
                 * Set SGLOffset0 value
                 */
                ddi_put8(acc_hdl, &scsi_io_req->SGLOffset0,
                    offsetof(MPI2_SCSI_IO_REQUEST, SGL) / 4);

                /*
                 * Setup descriptor info.  RAID passthrough must use the
                 * default request descriptor which is already set, so if this
                 * is a SCSI IO request, change the descriptor to SCSI IO.
                 */
                if (function == MPI2_FUNCTION_SCSI_IO_REQUEST) {
                        desc_type = MPI2_REQ_DESCRIPT_FLAGS_SCSI_IO;
                        request_desc = (((uint64_t)ddi_get16(acc_hdl,
                            &scsi_io_req->DevHandle)) << 48);
                }
        }

        /*
         * We must wait till the message has been completed before
         * beginning the next message so we wait for this one to
         * finish.
         */
        (void) ddi_dma_sync(dma_hdl, 0, 0, DDI_DMA_SYNC_FORDEV);
        request_desc |= ((cmd->cmd_slot << 16) | desc_type);
        cmd->cmd_rfm = NULL;
        MPTSAS_START_CMD(mpt, request_desc);
        if ((mptsas_check_dma_handle(dma_hdl) != DDI_SUCCESS) ||
            (mptsas_check_acc_handle(acc_hdl) != DDI_SUCCESS)) {
                ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_UNAFFECTED);
        }
}

typedef void (mps_pre_f)(mptsas_t *, mptsas_pt_request_t *);
static mps_pre_f        mpi_pre_ioc_facts;
static mps_pre_f        mpi_pre_port_facts;
static mps_pre_f        mpi_pre_fw_download;
static mps_pre_f        mpi_pre_fw_25_download;
static mps_pre_f        mpi_pre_fw_upload;
static mps_pre_f        mpi_pre_fw_25_upload;
static mps_pre_f        mpi_pre_sata_passthrough;
static mps_pre_f        mpi_pre_smp_passthrough;
static mps_pre_f        mpi_pre_config;
static mps_pre_f        mpi_pre_sas_io_unit_control;
static mps_pre_f        mpi_pre_scsi_io_req;

/*
 * Prepare the pt for a SAS2 FW_DOWNLOAD request.
 */
static void
mpi_pre_fw_download(mptsas_t *mpt, mptsas_pt_request_t *pt)
{
        pMpi2FWDownloadTCSGE_t tcsge;
        pMpi2FWDownloadRequest req;

        /*
         * If SAS3, call separate function.
         */
        if (mpt->m_MPI25) {
                mpi_pre_fw_25_download(mpt, pt);
                return;
        }

        /*
         * User requests should come in with the Transaction
         * context element where the SGL will go. Putting the
         * SGL after that seems to work, but don't really know
```

```
         * why. Other drivers tend to create an extra SGL and
         * refer to the TCE through that.
         */
        req = (pMpi2FWDownloadRequest)pt->request;
        tcsge = (pMpi2FWDownloadTCSGE_t)&req->SGL;
        if (tcsge->ContextSize != 0 || tcsge->DetailsLength != 12 ||
            tcsge->Flags != MPI2_SGE_FLAGS_TRANSACTION_ELEMENT) {
                mptsas_log(mpt, CE_WARN, "FW Download tce invalid!");
        }

        pt->sgl_offset = offsetof(MPI2_FW_DOWNLOAD_REQUEST, SGL) +
            sizeof (*tcsge);
        if (pt->request_size != pt->sgl_offset)
                NDBG15(("mpi_pre_fw_download(): Incorrect req size, "
                    "0x%x, should be 0x%x, dataoutsz 0x%x",
                    (int)pt->request_size, (int)pt->sgl_offset,
                    (int)pt->dataout_size));
        if (pt->data_size < sizeof (MPI2_FW_DOWNLOAD_REPLY))
                NDBG15(("mpi_pre_fw_download(): Incorrect rep size, "
                    "0x%x, should be 0x%x", pt->data_size,
                    (int)sizeof (MPI2_FW_DOWNLOAD_REPLY)));
}

/*
 * Prepare the pt for a SAS3 FW_DOWNLOAD request.
 */
static void
mpi_pre_fw_25_download(mptsas_t *mpt, mptsas_pt_request_t *pt)
{
        pMpi2FWDownloadTCSGE_t tcsge;
        pMpi2FWDownloadRequest req2;
        pMpi25FWDownloadRequest 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 = (pMpi2FWDownloadRequest)pt->request;
        req25 = (pMpi25FWDownloadRequest)pt->request;
        tcsge = (pMpi2FWDownloadTCSGE_t)&req2->SGL;
        if (tcsge->ContextSize != 0 || tcsge->DetailsLength != 12 ||
            tcsge->Flags != MPI2_SGE_FLAGS_TRANSACTION_ELEMENT) {
                mptsas_log(mpt, CE_WARN, "FW Download tce invalid!");
        }
        req25->ImageOffset = tcsge->ImageOffset;
        req25->ImageSize = tcsge->ImageSize;

        pt->sgl_offset = offsetof(MPI25_FW_DOWNLOAD_REQUEST, SGL);
        if (pt->request_size != pt->sgl_offset)
                NDBG15(("mpi_pre_fw_25_download(): 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_DOWNLOAD_REPLY))
                NDBG15(("mpi_pre_fw_25_download(): Incorrect rep size, "
                    "0x%x, should be 0x%x", pt->data_size,
                    (int)sizeof (MPI2_FW_UPLOAD_REPLY)));
}

/*
 * Prepare the pt for a SAS2 FW_UPLOAD request.
 */
static void
mpi_pre_fw_upload(mptsas_t *mpt, mptsas_pt_request_t *pt)
```



```
{
        pMpi2FWUploadTCSGE_t tcsge;
        pMpi2FWUploadRequest_t req;

        /*
         * If SAS3, call separate function.
         */
        if (mpt->m_MPI25) {
                mpi_pre_fw_25_upload(mpt, pt);
                return;
        }

        /*
         * User requests should come in with the Transaction
         * context element where the SGL will go. Putting the
         * SGL after that seems to work, but don’t really know
         * why. Other drivers tend to create an extra SGL and
         * refer to the TCE through that.
         */
        req = (pMpi2FWUploadRequest_t)pt->request;
        tcsge = (pMpi2FWUploadTCSGE_t)&req->SGL;
        if (tcsge->ContextSize != 0 || tcsge->DetailsLength != 12 ||
            tcsge->Flags != MPI2_SGE_FLAGS_TRANSACTION_ELEMENT) {
                mptsas_log(mpt, CE_WARN, "FW Upload tce invalid!");
        }

        pt->sgl_offset = offsetof(MPI2_FW_UPLOAD_REQUEST, SGL) +
            sizeof (*tcsge);
        if (pt->request_size != pt->sgl_offset)
                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;
}


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 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);
        /*
         * copy in the request buffer since it could be used by
         * another thread when the pt request into waitq
         */
        if (ddi_copyin(request, request_msg, request_size, mode)) {
                mutex_enter(&mpt->m_mutex);
                status = EFAULT;
                mptsas_log(mpt, CE_WARN, "failed to copy request data");
                goto out;
        }
        mutex_enter(&mpt->m_mutex);

        function = request_msg->Function;
        if (function == MPI2_FUNCTION_SCSI_TASK_MGMT) {
                pMpi2SCSITaskManagementRequest_t        task;
                task = (pMpi2SCSITaskManagementRequest_t)request_msg;
                mptsas_setup_bus_reset_delay(mpt);
                rv = mptsas_ioc_task_management(mpt, task->TaskType,
                    task->DevHandle, (int)task->LUN[1], reply, reply_size,
                    mode);

                if (rv != TRUE) {
                        status = EIO;
                        mptsas_log(mpt, CE_WARN, "task management failed");
                }
                goto out;
        }

        if (data_size != 0) {
                data_dma_state.size = data_size;
                if (mptsas_dma_alloc(mpt, &data_dma_state) != DDI_SUCCESS) {
                        status = ENOMEM;
                        mptsas_log(mpt, CE_WARN, "failed to alloc DMA "
                            "resource");
                        goto out;
                }
                pt_flags |= MPTSAS_DATA_ALLOCATED;
                if (direction == MPTSAS_PASS_THRU_DIRECTION_WRITE) {
                        mutex_exit(&mpt->m_mutex);
```

```
                        for (i = 0; i < data_size; i++) {
                                if (ddi_copyin(data + i, (uint8_t *)
                                    data_dma_state.memp + i, 1, mode)) {
                                        mutex_enter(&mpt->m_mutex);
                                        status = EFAULT;
                                        mptsas_log(mpt, CE_WARN, "failed to "
                                            "copy read data");
                                        goto out;
                                }
                        }
                        mutex_enter(&mpt->m_mutex);
                }
        }
        else
                bzero(&data_dma_state, sizeof (data_dma_state));

        if (dataout_size != 0) {
                dataout_dma_state.size = dataout_size;
                if (mptsas_dma_alloc(mpt, &dataout_dma_state) != DDI_SUCCESS) {
                        status = ENOMEM;
                        mptsas_log(mpt, CE_WARN, "failed to alloc DMA "
                            "resource");
                        goto out;
                }
                pt_flags |= MPTSAS_DATAOUT_ALLOCATED;
                mutex_exit(&mpt->m_mutex);
                for (i = 0; i < dataout_size; i++) {
                        if (ddi_copyin(dataout + i, (uint8_t *)
                            dataout_dma_state.memp + i, 1, mode)) {
                                mutex_enter(&mpt->m_mutex);
                                mptsas_log(mpt, CE_WARN, "failed to copy out"
                                    " data");
                                status = EFAULT;
                                goto out;
                        }
                }
                mutex_enter(&mpt->m_mutex);
        }
        else
                bzero(&dataout_dma_state, sizeof (dataout_dma_state));

        if ((rvalue = (mptsas_request_from_pool(mpt, &cmd, &pkt))) == -1) {
                status = EAGAIN;
                mptsas_log(mpt, CE_NOTE, "event ack command pool is full");
                goto out;
        }
        pt_flags |= MPTSAS_REQUEST_POOL_CMD;

        bzero((caddr_t)cmd, sizeof (*cmd));
        bzero((caddr_t)pkt, scsi_pkt_size());
        bzero((caddr_t)&pt, sizeof (pt));

        cmd->ioc_cmd_slot = (uint32_t)(rvalue);

        pt.request = (uint8_t *)request_msg;
        pt.direction = direction;
        pt.simple = 0;
        pt.request_size = request_size;
        pt.data_size = data_size;
        pt.dataout_size = dataout_size;
        pt.data_cookie = data_dma_state.cookie;
        pt.dataout_cookie = dataout_dma_state.cookie;
        mptsas_prep_sgl_offset(mpt, &pt);

        /*
         * Form a blank cmd/pkt to store the acknowledgement message
```



```
         */
        pkt->pkt_cdbp           = (opaque_t)&cmd->cmd_cdb[0];
        pkt->pkt_scbp           = (opaque_t)&cmd->cmd_scb;
        pkt->pkt_ha_private     = (opaque_t)&pt;
        pkt->pkt_flags          = FLAG_HEAD;
        pkt->pkt_time           = timeout;
        cmd->cmd_pkt            = pkt;
        cmd->cmd_flags          = CFLAG_CMDIOC | CFLAG_PASSTHRU;

        /*
         * Save the command in a slot
         */
        if (mptsas_save_cmd(mpt, cmd) == TRUE) {
                /*
                 * Once passthru command get slot, set cmd_flags
                 * CFLAG_PREPARED.
                 */
                cmd->cmd_flags |= CFLAG_PREPARED;
                mptsas_start_passthru(mpt, cmd);
        } else {
                mptsas_waitq_add(mpt, cmd);
        }

        while ((cmd->cmd_flags & CFLAG_FINISHED) == 0) {
                cv_wait(&mpt->m_passthru_cv, &mpt->m_mutex);
        }

        if (cmd->cmd_flags & CFLAG_PREPARED) {
                memp = mpt->m_req_frame + (mpt->m_req_frame_size *
                    cmd->cmd_slot);
                request_hdrp = (pMPI2RequestHeader_t)memp;
        }

        if (cmd->cmd_flags & CFLAG_TIMEOUT) {
                status = ETIMEDOUT;
                mptsas_log(mpt, CE_WARN, "passthrough command timeout");
                pt_flags |= MPTSAS_CMD_TIMEOUT;
                goto out;
        }

        if (cmd->cmd_rfm) {
                /*
                 * cmd_rfm is zero means the command reply is a CONTEXT
                 * reply and no PCI Write to post the free reply SMFA
                 * because no reply message frame is used.
                 * cmd_rfm is non-zero means the reply is a ADDRESS
                 * reply and reply message frame is used.
                 */
                pt_flags |= MPTSAS_ADDRESS_REPLY;
                (void) ddi_dma_sync(mpt->m_dma_reply_frame_hdl, 0, 0,
                    DDI_DMA_SYNC_FORCPU);
                reply_msg = (pMPI2DefaultReply_t)
                    (mpt->m_reply_frame + (cmd->cmd_rfm -
                    (mpt->m_reply_frame_dma_addr&0xffffffffu)));
        }

        mptsas_fma_check(mpt, cmd);
        if (pkt->pkt_reason == CMD_TRAN_ERR) {
                status = EAGAIN;
                mptsas_log(mpt, CE_WARN, "passthru fma error");
                goto out;
        }
        if (pkt->pkt_reason == CMD_RESET) {
                status = EAGAIN;
                mptsas_log(mpt, CE_WARN, "ioc reset abort passthru");
                goto out;
```

```
        }

        if (pkt->pkt_reason == CMD_INCOMPLETE) {
                status = EIO;
                mptsas_log(mpt, CE_WARN, "passthrough command incomplete");
                goto out;
        }

        mutex_exit(&mpt->m_mutex);
        if (cmd->cmd_flags & CFLAG_PREPARED) {
                function = request_hdrp->Function;
                if ((function == MPI2_FUNCTION_SCSI_IO_REQUEST) ||
                    (function == MPI2_FUNCTION_RAID_SCSI_IO_PASSTHROUGH)) {
                        reply_len = sizeof (MPI2_SCSI_IO_REPLY);
                        sense_len = reply_size - reply_len;
                } else {
                        reply_len = reply_size;
                        sense_len = 0;
                }

                for (i = 0; i < reply_len; i++) {
                        if (ddi_copyout((uint8_t *)reply_msg + i, reply + i, 1,
                            mode)) {
                                mutex_enter(&mpt->m_mutex);
                                status = EFAULT;
                                mptsas_log(mpt, CE_WARN, "failed to copy out "
                                    "reply data");
                                goto out;
                        }
                }
                for (i = 0; i < sense_len; i++) {
                        if (ddi_copyout((uint8_t *)request_hdrp + 64 + i,
                            reply + reply_len + i, 1, mode)) {
                                mutex_enter(&mpt->m_mutex);
                                status = EFAULT;
                                mptsas_log(mpt, CE_WARN, "failed to copy out "
                                    "sense data");
                                goto out;
                        }
                }
        }

        if (data_size) {
                if (direction != MPTSAS_PASS_THRU_DIRECTION_WRITE) {
                        (void) ddi_dma_sync(data_dma_state.handle, 0, 0,
                            DDI_DMA_SYNC_FORCPU);
                        for (i = 0; i < data_size; i++) {
                                if (ddi_copyout((uint8_t *)(
                                    data_dma_state.memp + i), data + i,  1,
                                    mode)) {
                                        mutex_enter(&mpt->m_mutex);
                                        status = EFAULT;
                                        mptsas_log(mpt, CE_WARN, "failed to "
                                            "copy out the reply data");
                                        goto out;
                                }
                        }
                }
        }
        mutex_enter(&mpt->m_mutex);
out:
        /*
         * Put the reply frame back on the free queue, increment the free
         * index, and write the new index to the free index register.  But only
         * if this reply is an ADDRESS reply.
         */
```



```
        if (pt_flags & MPTSAS_ADDRESS_REPLY) {
                ddi_put32(mpt->m_acc_free_queue_hdl,
                    &((uint32_t *)(void *)mpt->m_free_queue)[mpt->m_free_index],
                    cmd->cmd_rfm);
                (void) ddi_dma_sync(mpt->m_dma_free_queue_hdl, 0, 0,
                    DDI_DMA_SYNC_FORDEV);
                if (++mpt->m_free_index == mpt->m_free_queue_depth) {
                        mpt->m_free_index = 0;
                }
                ddi_put32(mpt->m_datap, &mpt->m_reg->ReplyFreeHostIndex,
                    mpt->m_free_index);
        }
        if (cmd && (cmd->cmd_flags & CFLAG_PREPARED)) {
                mptsas_remove_cmd(mpt, cmd);
                pt_flags &= (~MPTSAS_REQUEST_POOL_CMD);
        }
        if (pt_flags & MPTSAS_REQUEST_POOL_CMD)
                mptsas_return_to_pool(mpt, cmd);
        if (pt_flags & MPTSAS_DATA_ALLOCATED) {
                if (mptsas_check_dma_handle(data_dma_state.handle) !=
                    DDI_SUCCESS) {
                        ddi_fm_service_impact(mpt->m_dip,
                            DDI_SERVICE_UNAFFECTED);
                        status = EFAULT;
                }
                mptsas_dma_free(&data_dma_state);
        }
        if (pt_flags & MPTSAS_DATAOUT_ALLOCATED) {
                if (mptsas_check_dma_handle(dataout_dma_state.handle) !=
                    DDI_SUCCESS) {
                        ddi_fm_service_impact(mpt->m_dip,
                            DDI_SERVICE_UNAFFECTED);
                        status = EFAULT;
                }
                mptsas_dma_free(&dataout_dma_state);
        }
        if (pt_flags & MPTSAS_CMD_TIMEOUT) {
                if ((mptsas_restart_ioc(mpt)) == DDI_FAILURE) {
                        mptsas_log(mpt, CE_WARN, "mptsas_restart_ioc failed");
                }
        }
        if (request_msg)
                kmem_free(request_msg, request_size);

        return (status);
}

static int
mptsas_pass_thru(mptsas_t *mpt, mptsas_pass_thru_t *data, int mode)
{
        /*
         * If timeout is 0, set timeout to default of 60 seconds.
         */
        if (data->Timeout == 0) {
                data->Timeout = MPTSAS_PASS_THRU_TIME_DEFAULT;
        }

        if (((data->DataSize == 0) &&
            (data->DataDirection == MPTSAS_PASS_THRU_DIRECTION_NONE)) ||
            ((data->DataSize != 0) &&
            ((data->DataDirection == MPTSAS_PASS_THRU_DIRECTION_READ) ||
            (data->DataDirection == MPTSAS_PASS_THRU_DIRECTION_WRITE) ||
            ((data->DataDirection == MPTSAS_PASS_THRU_DIRECTION_BOTH) &&
            (data->DataOutSize != 0))))) {
                if (data->DataDirection == MPTSAS_PASS_THRU_DIRECTION_BOTH) {
                        data->DataDirection = MPTSAS_PASS_THRU_DIRECTION_READ;
```

```
                } else {
                        data->DataOutSize = 0;
                }
                /*
                 * Send passthru request messages
                 */
                return (mptsas_do_passthru(mpt,
                    (uint8_t *)((uintptr_t)data->PtrRequest),
                    (uint8_t *)((uintptr_t)data->PtrReply),
                    (uint8_t *)((uintptr_t)data->PtrData),
                    data->RequestSize, data->ReplySize,
                    data->DataSize, (uint8_t)data->DataDirection,
                    (uint8_t *)((uintptr_t)data->PtrDataOut),
                    data->DataOutSize, data->Timeout, mode));
        } else {
                return (EINVAL);
        }
}

static uint8_t
mptsas_get_fw_diag_buffer_number(mptsas_t *mpt, uint32_t unique_id)
{
        uint8_t index;

        for (index = 0; index < MPI2_DIAG_BUF_TYPE_COUNT; index++) {
                if (mpt->m_fw_diag_buffer_list[index].unique_id == unique_id) {
                        return (index);
                }
        }

        return (MPTSAS_FW_DIAGNOSTIC_UID_NOT_FOUND);
}

static void
mptsas_start_diag(mptsas_t *mpt, mptsas_cmd_t *cmd)
{
        pMpi2DiagBufferPostRequest_t    pDiag_post_msg;
        pMpi2DiagReleaseRequest_t       pDiag_release_msg;
        struct scsi_pkt                 *pkt = cmd->cmd_pkt;
        mptsas_diag_request_t           *diag = pkt->pkt_ha_private;
        uint32_t                        i;
        uint64_t                        request_desc;

        ASSERT(mutex_owned(&mpt->m_mutex));

        /*
         * Form the diag message depending on the post or release function.
         */
        if (diag->function == MPI2_FUNCTION_DIAG_BUFFER_POST) {
                pDiag_post_msg = (pMpi2DiagBufferPostRequest_t)
                    (mpt->m_req_frame + (mpt->m_req_frame_size *
                    cmd->cmd_slot));
                bzero(pDiag_post_msg, mpt->m_req_frame_size);
                ddi_put8(mpt->m_acc_req_frame_hdl, &pDiag_post_msg->Function,
                    diag->function);
                ddi_put8(mpt->m_acc_req_frame_hdl, &pDiag_post_msg->BufferType,
                    diag->pBuffer->buffer_type);
                ddi_put8(mpt->m_acc_req_frame_hdl,
                    &pDiag_post_msg->ExtendedType,
                    diag->pBuffer->extended_type);
                ddi_put32(mpt->m_acc_req_frame_hdl,
                    &pDiag_post_msg->BufferLength,
                    diag->pBuffer->buffer_data.size);
                for (i = 0; i < (sizeof (pDiag_post_msg->ProductSpecific) / 4);
                    i++) {
                        ddi_put32(mpt->m_acc_req_frame_hdl,
```



```
                            &pDiag_post_msg->ProductSpecific[i],
                            diag->pBuffer->product_specific[i]);
                }
                ddi_put32(mpt->m_acc_req_frame_hdl,
                    &pDiag_post_msg->BufferAddress.Low,
                    (uint32_t)(diag->pBuffer->buffer_data.cookie.dmac_laddress
                    & 0xffffffffull));
                ddi_put32(mpt->m_acc_req_frame_hdl,
                    &pDiag_post_msg->BufferAddress.High,
                    (uint32_t)(diag->pBuffer->buffer_data.cookie.dmac_laddress
                    >> 32));
        } else {
                pDiag_release_msg = (pMpi2DiagReleaseRequest_t)
                    (mpt->m_req_frame + (mpt->m_req_frame_size *
                    cmd->cmd_slot));
                bzero(pDiag_release_msg, mpt->m_req_frame_size);
                ddi_put8(mpt->m_acc_req_frame_hdl,
                    &pDiag_release_msg->Function, diag->function);
                ddi_put8(mpt->m_acc_req_frame_hdl,
                    &pDiag_release_msg->BufferType,
                    diag->pBuffer->buffer_type);
        }

        /*
         * Send the message
         */
        (void) ddi_dma_sync(mpt->m_dma_req_frame_hdl, 0, 0,
            DDI_DMA_SYNC_FORDEV);
        request_desc = (cmd->cmd_slot << 16) |
            MPI2_REQ_DESCRIPT_FLAGS_DEFAULT_TYPE;
        cmd->cmd_rfm = NULL;
        MPTSAS_START_CMD(mpt, request_desc);
        if ((mptsas_check_dma_handle(mpt->m_dma_req_frame_hdl) !=
            DDI_SUCCESS) ||
            (mptsas_check_acc_handle(mpt->m_acc_req_frame_hdl) !=
            DDI_SUCCESS)) {
                ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_UNAFFECTED);
        }
}

static int
mptsas_post_fw_diag_buffer(mptsas_t *mpt,
    mptsas_fw_diagnostic_buffer_t *pBuffer, uint32_t *return_code)
{
        mptsas_diag_request_t           diag;
        int                             status, slot_num, post_flags = 0;
        mptsas_cmd_t                    *cmd = NULL;
        struct scsi_pkt                 *pkt;
        pMpi2DiagBufferPostReply_t      reply;
        uint16_t                        iocstatus;
        uint32_t                        iocloginfo, transfer_length;

        /*
         * If buffer is not enabled, just leave.
         */
        *return_code = MPTSAS_FW_DIAG_ERROR_POST_FAILED;
        if (!pBuffer->enabled) {
                status = DDI_FAILURE;
                goto out;
        }

        /*
         * Clear some flags initially.
         */
        pBuffer->force_release = FALSE;
        pBuffer->valid_data = FALSE;
```

```
        pBuffer->owned_by_firmware = FALSE;

        /*
         * Get a cmd buffer from the cmd buffer pool
         */
        if ((slot_num = (mptsas_request_from_pool(mpt, &cmd, &pkt))) == -1) {
                status = DDI_FAILURE;
                mptsas_log(mpt, CE_NOTE, "command pool is full: Post FW Diag");
                goto out;
        }
        post_flags |= MPTSAS_REQUEST_POOL_CMD;

        bzero((caddr_t)cmd, sizeof (*cmd));
        bzero((caddr_t)pkt, scsi_pkt_size());

        cmd->ioc_cmd_slot = (uint32_t)(slot_num);

        diag.pBuffer = pBuffer;
        diag.function = MPI2_FUNCTION_DIAG_BUFFER_POST;

        /*
         * Form a blank cmd/pkt to store the acknowledgement message
         */
        pkt->pkt_ha_private     = (opaque_t)&diag;
        pkt->pkt_flags          = FLAG_HEAD;
        pkt->pkt_time           = 60;
        cmd->cmd_pkt            = pkt;
        cmd->cmd_flags          = CFLAG_CMDIOC | CFLAG_FW_DIAG;

        /*
         * Save the command in a slot
         */
        if (mptsas_save_cmd(mpt, cmd) == TRUE) {
                /*
                 * Once passthru command get slot, set cmd_flags
                 * CFLAG_PREPARED.
                 */
                cmd->cmd_flags |= CFLAG_PREPARED;
                mptsas_start_diag(mpt, cmd);
        } else {
                mptsas_waitq_add(mpt, cmd);
        }

        while ((cmd->cmd_flags & CFLAG_FINISHED) == 0) {
                cv_wait(&mpt->m_fw_diag_cv, &mpt->m_mutex);
        }

        if (cmd->cmd_flags & CFLAG_TIMEOUT) {
                status = DDI_FAILURE;
                mptsas_log(mpt, CE_WARN, "Post FW Diag command timeout");
                goto out;
        }

        /*
         * cmd_rfm points to the reply message if a reply was given.  Check the
         * IOCStatus to make sure everything went OK with the FW diag request
         * and set buffer flags.
         */
        if (cmd->cmd_rfm) {
                post_flags |= MPTSAS_ADDRESS_REPLY;
                (void) ddi_dma_sync(mpt->m_dma_reply_frame_hdl, 0, 0,
                    DDI_DMA_SYNC_FORCPU);
                reply = (pMpi2DiagBufferPostReply_t)(mpt->m_reply_frame +
                    (cmd->cmd_rfm -
                    (mpt->m_reply_frame_dma_addr&0xffffffful)));
```



```
                /*
                 * Get the reply message data
                 */
                iocstatus = ddi_get16(mpt->m_acc_reply_frame_hdl,
                    &reply->IOCStatus);
                iocloginfo = ddi_get32(mpt->m_acc_reply_frame_hdl,
                    &reply->IOCLogInfo);
                transfer_length = ddi_get32(mpt->m_acc_reply_frame_hdl,
                    &reply->TransferLength);

                /*
                 * If post failed quit.
                 */
                if (iocstatus != MPI2_IOCSTATUS_SUCCESS) {
                        status = DDI_FAILURE;
                        NDBG13(("post FW Diag Buffer failed: IOCStatus=0x%x, "
                            "IOCLogInfo=0x%x, TransferLength=0x%x", iocstatus,
                            iocloginfo, transfer_length));
                        goto out;
                }

                /*
                 * Post was successful.
                 */
                pBuffer->valid_data = TRUE;
                pBuffer->owned_by_firmware = TRUE;
                *return_code = MPTSAS_FW_DIAG_ERROR_SUCCESS;
                status = DDI_SUCCESS;
        }

out:
        /*
         * Put the reply frame back on the free queue, increment the free
         * index, and write the new index to the free index register.  But only
         * if this reply is an ADDRESS reply.
         */
        if (post_flags & MPTSAS_ADDRESS_REPLY) {
                ddi_put32(mpt->m_acc_free_queue_hdl,
                    &((uint32_t *)(void *)mpt->m_free_queue)[mpt->m_free_index],
                    cmd->cmd_rfm);
                (void) ddi_dma_sync(mpt->m_dma_free_queue_hdl, 0, 0,
                    DDI_DMA_SYNC_FORDEV);
                if (++mpt->m_free_index == mpt->m_free_queue_depth) {
                        mpt->m_free_index = 0;
                }
                ddi_put32(mpt->m_datap, &mpt->m_reg->ReplyFreeHostIndex,
                    mpt->m_free_index);
        }
        if (cmd && (cmd->cmd_flags & CFLAG_PREPARED)) {
                mptsas_remove_cmd(mpt, cmd);
                post_flags &= (~MPTSAS_REQUEST_POOL_CMD);
        }
        if (post_flags & MPTSAS_REQUEST_POOL_CMD) {
                mptsas_return_to_pool(mpt, cmd);
        }

        return (status);
}

static int
mptsas_release_fw_diag_buffer(mptsas_t *mpt,
    mptsas_fw_diagnostic_buffer_t *pBuffer, uint32_t *return_code,
    uint32_t diag_type)
{
        mptsas_diag_request_t   diag;
        int                     status, slot_num, rel_flags = 0;
```

```
        mptsas_cmd_t            *cmd = NULL;
        struct scsi_pkt         *pkt;
        pMpi2DiagReleaseReply_t reply;
        uint16_t                iocstatus;
        uint32_t                iocloginfo;

        /*
         * If buffer is not enabled, just leave.
         */
        *return_code = MPTSAS_FW_DIAG_ERROR_RELEASE_FAILED;
        if (!pBuffer->enabled) {
                mptsas_log(mpt, CE_NOTE, "This buffer type is not supported "
                    "by the IOC");
                status = DDI_FAILURE;
                goto out;
        }

        /*
         * Clear some flags initially.
         */
        pBuffer->force_release = FALSE;
        pBuffer->valid_data = FALSE;
        pBuffer->owned_by_firmware = FALSE;

        /*
         * Get a cmd buffer from the cmd buffer pool
         */
        if ((slot_num = (mptsas_request_from_pool(mpt, &cmd, &pkt))) == -1) {
                status = DDI_FAILURE;
                mptsas_log(mpt, CE_NOTE, "command pool is full: Release FW "
                    "Diag");
                goto out;
        }
        rel_flags |= MPTSAS_REQUEST_POOL_CMD;

        bzero((caddr_t)cmd, sizeof (*cmd));
        bzero((caddr_t)pkt, scsi_pkt_size());

        cmd->ioc_cmd_slot = (uint32_t)(slot_num);

        diag.pBuffer = pBuffer;
        diag.function = MPI2_FUNCTION_DIAG_RELEASE;

        /*
         * Form a blank cmd/pkt to store the acknowledgement message
         */
        pkt->pkt_ha_private     = (opaque_t)&diag;
        pkt->pkt_flags          = FLAG_HEAD;
        pkt->pkt_time           = 60;
        cmd->cmd_pkt            = pkt;
        cmd->cmd_flags          = CFLAG_CMDIOC | CFLAG_FW_DIAG;

        /*
         * Save the command in a slot
         */
        if (mptsas_save_cmd(mpt, cmd) == TRUE) {
                /*
                 * Once passthru command get slot, set cmd_flags
                 * CFLAG_PREPARED.
                 */
                cmd->cmd_flags |= CFLAG_PREPARED;
                mptsas_start_diag(mpt, cmd);
        } else {
                mptsas_waitq_add(mpt, cmd);
        }
```



```
        while ((cmd->cmd_flags & CFLAG_FINISHED) == 0) {
                cv_wait(&mpt->m_fw_diag_cv, &mpt->m_mutex);
        }

        if (cmd->cmd_flags & CFLAG_TIMEOUT) {
                status = DDI_FAILURE;
                mptsas_log(mpt, CE_WARN, "Release FW Diag command timeout");
                goto out;
        }

        /*
         * cmd_rfm points to the reply message if a reply was given.  Check the
         * IOCStatus to make sure everything went OK with the FW diag request
         * and set buffer flags.
         */
        if (cmd->cmd_rfm) {
                rel_flags |= MPTSAS_ADDRESS_REPLY;
                (void) ddi_dma_sync(mpt->m_dma_reply_frame_hdl, 0, 0,
                    DDI_DMA_SYNC_FORCPU);
                reply = (pMpi2DiagReleaseReply_t)(mpt->m_reply_frame +
                    (cmd->cmd_rfm -
                    (mpt->m_reply_frame_dma_addr&0xffffffffu)));

                /*
                 * Get the reply message data
                 */
                iocstatus = ddi_get16(mpt->m_acc_reply_frame_hdl,
                    &reply->IOCStatus);
                iocloginfo = ddi_get32(mpt->m_acc_reply_frame_hdl,
                    &reply->IOCLogInfo);

                /*
                 * If release failed quit.
                 */
                if ((iocstatus != MPI2_IOCSTATUS_SUCCESS) ||
                    pBuffer->owned_by_firmware) {
                        status = DDI_FAILURE;
                        NDBG13(("release FW Diag Buffer failed: "
                            "IOCStatus=0x%x, IOCLogInfo=0x%x", iocstatus,
                            iocloginfo));
                        goto out;
                }

                /*
                 * Release was successful.
                 */
                *return_code = MPTSAS_FW_DIAG_ERROR_SUCCESS;
                status = DDI_SUCCESS;

                /*
                 * If this was for an UNREGISTER diag type command, clear the
                 * unique ID.
                 */
                if (diag_type == MPTSAS_FW_DIAG_TYPE_UNREGISTER) {
                        pBuffer->unique_id = MPTSAS_FW_DIAG_INVALID_UID;
                }
        }

out:
        /*
         * Put the reply frame back on the free queue, increment the free
         * index, and write the new index to the free index register.  But only
         * if this reply is an ADDRESS reply.
         */
        if (rel_flags & MPTSAS_ADDRESS_REPLY) {
                ddi_put32(mpt->m_acc_free_queue_hdl,
```

```
                    &((uint32_t *)(void *)mpt->m_free_queue)[mpt->m_free_index],
                    cmd->cmd_rfm);
                (void) ddi_dma_sync(mpt->m_dma_free_queue_hdl, 0, 0,
                    DDI_DMA_SYNC_FORDEV);
                if (++mpt->m_free_index == mpt->m_free_queue_depth) {
                        mpt->m_free_index = 0;
                }
                ddi_put32(mpt->m_datap, &mpt->m_reg->ReplyFreeHostIndex,
                    mpt->m_free_index);
        }
        if (cmd && (cmd->cmd_flags & CFLAG_PREPARED)) {
                mptsas_remove_cmd(mpt, cmd);
                rel_flags &= (~MPTSAS_REQUEST_POOL_CMD);
        }
        if (rel_flags & MPTSAS_REQUEST_POOL_CMD) {
                mptsas_return_to_pool(mpt, cmd);
        }

        return (status);
}

static int
mptsas_diag_register(mptsas_t *mpt, mptsas_fw_diag_register_t *diag_register,
    uint32_t *return_code)
{
        mptsas_fw_diagnostic_buffer_t   *pBuffer;
        uint8_t                         extended_type, buffer_type, i;
        uint32_t                        buffer_size;
        uint32_t                        unique_id;
        int                             status;

        ASSERT(mutex_owned(&mpt->m_mutex));

        extended_type = diag_register->ExtendedType;
        buffer_type = diag_register->BufferType;
        buffer_size = diag_register->RequestedBufferSize;
        unique_id = diag_register->UniqueId;

        /*
         * Check for valid buffer type
         */
        if (buffer_type >= MPI2_DIAG_BUF_TYPE_COUNT) {
                *return_code = MPTSAS_FW_DIAG_ERROR_INVALID_PARAMETER;
                return (DDI_FAILURE);
        }

        /*
         * Get the current buffer and look up the unique ID.  The unique ID
         * should not be found.  If it is, the ID is already in use.
         */
        i = mptsas_get_fw_diag_buffer_number(mpt, unique_id);
        pBuffer = &mpt->m_fw_diag_buffer_list[buffer_type];
        if (i != MPTSAS_FW_DIAGNOSTIC_UID_NOT_FOUND) {
                *return_code = MPTSAS_FW_DIAG_ERROR_INVALID_UID;
                return (DDI_FAILURE);
        }

        /*
         * The buffer's unique ID should not be registered yet, and the given
         * unique ID cannot be 0.
         */
        if ((pBuffer->unique_id != MPTSAS_FW_DIAG_INVALID_UID) ||
            (unique_id == MPTSAS_FW_DIAG_INVALID_UID)) {
                *return_code = MPTSAS_FW_DIAG_ERROR_INVALID_UID;
                return (DDI_FAILURE);
        }
```



```
        /*
         * If this buffer is already posted as immediate, just change owner.
         */
        if (pBuffer->immediate && pBuffer->owned_by_firmware &&
            (pBuffer->unique_id == MPTSAS_FW_DIAG_INVALID_UID)) {
                pBuffer->immediate = FALSE;
                pBuffer->unique_id = unique_id;
                return (DDI_SUCCESS);
        }

        /*
         * Post a new buffer after checking if it's enabled.  The DMA buffer
         * that is allocated will be contiguous (sgl_len = 1).
         */
        if (!pBuffer->enabled) {
                *return_code = MPTSAS_FW_DIAG_ERROR_NO_BUFFER;
                return (DDI_FAILURE);
        }
        bzero(&pBuffer->buffer_data, sizeof (mptsas_dma_alloc_state_t));
        pBuffer->buffer_data.size = buffer_size;
        if (mptsas_dma_alloc(mpt, &pBuffer->buffer_data) != DDI_SUCCESS) {
                mptsas_log(mpt, CE_WARN, "failed to alloc DMA resource for "
                    "diag buffer: size = %d bytes", buffer_size);
                *return_code = MPTSAS_FW_DIAG_ERROR_NO_BUFFER;
                return (DDI_FAILURE);
        }

        /*
         * Copy the given info to the diag buffer and post the buffer.
         */
        pBuffer->buffer_type = buffer_type;
        pBuffer->immediate = FALSE;
        if (buffer_type == MPI2_DIAG_BUF_TYPE_TRACE) {
                for (i = 0; i < (sizeof (pBuffer->product_specific) / 4);
                    i++) {
                        pBuffer->product_specific[i] =
                            diag_register->ProductSpecific[i];
                }
        }
        pBuffer->extended_type = extended_type;
        pBuffer->unique_id = unique_id;
        status = mptsas_post_fw_diag_buffer(mpt, pBuffer, return_code);

        if (mptsas_check_dma_handle(pBuffer->buffer_data.handle) !=
            DDI_SUCCESS) {
                mptsas_log(mpt, CE_WARN, "Check of DMA handle failed in "
                    "mptsas_diag_register.");
                ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_UNAFFECTED);
                        status = DDI_FAILURE;
        }

        /*
         * In case there was a failure, free the DMA buffer.
         */
        if (status == DDI_FAILURE) {
                mptsas_dma_free(&pBuffer->buffer_data);
        }

        return (status);
}

static int
mptsas_diag_unregister(mptsas_t *mpt,
    mptsas_fw_diag_unregister_t *diag_unregister, uint32_t *return_code)
{
```

```
        mptsas_fw_diagnostic_buffer_t   *pBuffer;
        uint8_t                         i;
        uint32_t                        unique_id;
        int                             status;

        ASSERT(mutex_owned(&mpt->m_mutex));

        unique_id = diag_unregister->UniqueId;

        /*
         * Get the current buffer and look up the unique ID.  The unique ID
         * should be there.
         */
        i = mptsas_get_fw_diag_buffer_number(mpt, unique_id);
        if (i == MPTSAS_FW_DIAGNOSTIC_UID_NOT_FOUND) {
                *return_code = MPTSAS_FW_DIAG_ERROR_INVALID_UID;
                return (DDI_FAILURE);
        }

        pBuffer = &mpt->m_fw_diag_buffer_list[i];

        /*
         * Try to release the buffer from FW before freeing it.  If release
         * fails, don’t free the DMA buffer in case FW tries to access it
         * later.  If buffer is not owned by firmware, can’t release it.
         */
        if (!pBuffer->owned_by_firmware) {
                status = DDI_SUCCESS;
        } else {
                status = mptsas_release_fw_diag_buffer(mpt, pBuffer,
                    return_code, MPTSAS_FW_DIAG_TYPE_UNREGISTER);
        }

        /*
         * At this point, return the current status no matter what happens with
         * the DMA buffer.
         */
        pBuffer->unique_id = MPTSAS_FW_DIAG_INVALID_UID;
        if (status == DDI_SUCCESS) {
                if (mptsas_check_dma_handle(pBuffer->buffer_data.handle) !=
                    DDI_SUCCESS) {
                        mptsas_log(mpt, CE_WARN, "Check of DMA handle failed "
                            "in mptsas_diag_unregister.");
                        ddi_fm_service_impact(mpt->m_dip,
                            DDI_SERVICE_UNAFFECTED);
                }
                mptsas_dma_free(&pBuffer->buffer_data);
        }

        return (status);
}

static int
mptsas_diag_query(mptsas_t *mpt, mptsas_fw_diag_query_t *diag_query,
    uint32_t *return_code)
{
        mptsas_fw_diagnostic_buffer_t   *pBuffer;
        uint8_t                         i;
        uint32_t                        unique_id;

        ASSERT(mutex_owned(&mpt->m_mutex));

        unique_id = diag_query->UniqueId;

        /*
         * If ID is valid, query on ID.
```



```
         * If ID is invalid, query on buffer type.
         */
        if (unique_id == MPTSAS_FW_DIAG_INVALID_UID) {
                i = diag_query->BufferType;
                if (i >= MPI2_DIAG_BUF_TYPE_COUNT) {
                        *return_code = MPTSAS_FW_DIAG_ERROR_INVALID_UID;
                        return (DDI_FAILURE);
                }
        } else {
                i = mptsas_get_fw_diag_buffer_number(mpt, unique_id);
                if (i == MPTSAS_FW_DIAGNOSTIC_UID_NOT_FOUND) {
                        *return_code = MPTSAS_FW_DIAG_ERROR_INVALID_UID;
                        return (DDI_FAILURE);
                }
        }

        /*
         * Fill query structure with the diag buffer info.
         */
        pBuffer = &mpt->m_fw_diag_buffer_list[i];
        diag_query->BufferType = pBuffer->buffer_type;
        diag_query->ExtendedType = pBuffer->extended_type;
        if (diag_query->BufferType == MPI2_DIAG_BUF_TYPE_TRACE) {
                for (i = 0; i < (sizeof (diag_query->ProductSpecific) / 4);
                    i++) {
                        diag_query->ProductSpecific[i] =
                            pBuffer->product_specific[i];
                }
        }
        diag_query->TotalBufferSize = pBuffer->buffer_data.size;
        diag_query->DriverAddedBufferSize = 0;
        diag_query->UniqueId = pBuffer->unique_id;
        diag_query->ApplicationFlags = 0;
        diag_query->DiagnosticFlags = 0;

        /*
         * Set/Clear application flags
         */
        if (pBuffer->immediate) {
                diag_query->ApplicationFlags &= ~MPTSAS_FW_DIAG_FLAG_APP_OWNED;
        } else {
                diag_query->ApplicationFlags |= MPTSAS_FW_DIAG_FLAG_APP_OWNED;
        }
        if (pBuffer->valid_data || pBuffer->owned_by_firmware) {
                diag_query->ApplicationFlags |=
                    MPTSAS_FW_DIAG_FLAG_BUFFER_VALID;
        } else {
                diag_query->ApplicationFlags &=
                    ~MPTSAS_FW_DIAG_FLAG_BUFFER_VALID;
        }
        if (pBuffer->owned_by_firmware) {
                diag_query->ApplicationFlags |=
                    MPTSAS_FW_DIAG_FLAG_FW_BUFFER_ACCESS;
        } else {
                diag_query->ApplicationFlags &=
                    ~MPTSAS_FW_DIAG_FLAG_FW_BUFFER_ACCESS;
        }

        return (DDI_SUCCESS);
}

static int
mptsas_diag_read_buffer(mptsas_t *mpt,
    mptsas_diag_read_buffer_t *diag_read_buffer, uint8_t *ioctl_buf,
    uint32_t *return_code, int ioctl_mode)
{
```

```
        mptsas_fw_diagnostic_buffer_t   *pBuffer;
        uint8_t                         i, *pData;
        uint32_t                        unique_id, byte;
        int                             status;

        ASSERT(mutex_owned(&mpt->m_mutex));

        unique_id = diag_read_buffer->UniqueId;

        /*
         * Get the current buffer and look up the unique ID.  The unique ID
         * should be there.
         */
        i = mptsas_get_fw_diag_buffer_number(mpt, unique_id);
        if (i == MPTSAS_FW_DIAGNOSTIC_UID_NOT_FOUND) {
                *return_code = MPTSAS_FW_DIAG_ERROR_INVALID_UID;
                return (DDI_FAILURE);
        }

        pBuffer = &mpt->m_fw_diag_buffer_list[i];

        /*
         * Make sure requested read is within limits
         */
        if (diag_read_buffer->StartingOffset + diag_read_buffer->BytesToRead >
            pBuffer->buffer_data.size) {
                *return_code = MPTSAS_FW_DIAG_ERROR_INVALID_PARAMETER;
                return (DDI_FAILURE);
        }

        /*
         * Copy the requested data from DMA to the diag_read_buffer.  The DMA
         * buffer that was allocated is one contiguous buffer.
         */
        pData = (uint8_t *)(pBuffer->buffer_data.memp +
            diag_read_buffer->StartingOffset);
        (void) ddi_dma_sync(pBuffer->buffer_data.handle, 0, 0,
            DDI_DMA_SYNC_FORCPU);
        for (byte = 0; byte < diag_read_buffer->BytesToRead; byte++) {
                if (ddi_copyout(pData + byte, ioctl_buf + byte, 1, ioctl_mode)
                    != 0) {
                        return (DDI_FAILURE);
                }
        }
        diag_read_buffer->Status = 0;

        /*
         * Set or clear the Force Release flag.
         */
        if (pBuffer->force_release) {
                diag_read_buffer->Flags |= MPTSAS_FW_DIAG_FLAG_FORCE_RELEASE;
        } else {
                diag_read_buffer->Flags &= ~MPTSAS_FW_DIAG_FLAG_FORCE_RELEASE;
        }

        /*
         * If buffer is to be reregistered, make sure it’s not already owned by
         * firmware first.
         */
        status = DDI_SUCCESS;
        if (!pBuffer->owned_by_firmware) {
                if (diag_read_buffer->Flags & MPTSAS_FW_DIAG_FLAG_REREGISTER) {
                        status = mptsas_post_fw_diag_buffer(mpt, pBuffer,
                            return_code);
                }
        }
```



```
        return (status);
}

static int
mptsas_diag_release(mptsas_t *mpt, mptsas_fw_diag_release_t *diag_release,
    uint32_t *return_code)
{
        mptsas_fw_diagnostic_buffer_t   *pBuffer;
        uint8_t                         i;
        uint32_t                        unique_id;
        int                             status;

        ASSERT(mutex_owned(&mpt->m_mutex));

        unique_id = diag_release->UniqueId;

        /*
         * Get the current buffer and look up the unique ID.  The unique ID
         * should be there.
         */
        i = mptsas_get_fw_diag_buffer_number(mpt, unique_id);
        if (i == MPTSAS_FW_DIAGNOSTIC_UID_NOT_FOUND) {
                *return_code = MPTSAS_FW_DIAG_ERROR_INVALID_UID;
                return (DDI_FAILURE);
        }

        pBuffer = &mpt->m_fw_diag_buffer_list[i];

        /*
         * If buffer is not owned by firmware, it’s already been released.
         */
        if (!pBuffer->owned_by_firmware) {
                *return_code = MPTSAS_FW_DIAG_ERROR_ALREADY_RELEASED;
                return (DDI_FAILURE);
        }

        /*
         * Release the buffer.
         */
        status = mptsas_release_fw_diag_buffer(mpt, pBuffer, return_code,
            MPTSAS_FW_DIAG_TYPE_RELEASE);
        return (status);
}

static int
mptsas_do_diag_action(mptsas_t *mpt, uint32_t action, uint8_t *diag_action,
    uint32_t length, uint32_t *return_code, int ioctl_mode)
{
        mptsas_fw_diag_register_t       diag_register;
        mptsas_fw_diag_unregister_t     diag_unregister;
        mptsas_fw_diag_query_t          diag_query;
        mptsas_diag_read_buffer_t       diag_read_buffer;
        mptsas_fw_diag_release_t        diag_release;
        int                             status = DDI_SUCCESS;
        uint32_t                        original_return_code, read_buf_len;

        ASSERT(mutex_owned(&mpt->m_mutex));

        original_return_code = *return_code;
        *return_code = MPTSAS_FW_DIAG_ERROR_SUCCESS;

        switch (action) {
                case MPTSAS_FW_DIAG_TYPE_REGISTER:
                        if (!length) {
                                *return_code =
```

```
                                MPTSAS_FW_DIAG_ERROR_INVALID_PARAMETER;
                        status = DDI_FAILURE;
                        break;
                }
                if (ddi_copyin(diag_action, &diag_register,
                    sizeof (diag_register), ioctl_mode) != 0) {
                        return (DDI_FAILURE);
                }
                status = mptsas_diag_register(mpt, &diag_register,
                    return_code);
                break;

        case MPTSAS_FW_DIAG_TYPE_UNREGISTER:
                if (length < sizeof (diag_unregister)) {
                        *return_code =
                                MPTSAS_FW_DIAG_ERROR_INVALID_PARAMETER;
                        status = DDI_FAILURE;
                        break;
                }
                if (ddi_copyin(diag_action, &diag_unregister,
                    sizeof (diag_unregister), ioctl_mode) != 0) {
                        return (DDI_FAILURE);
                }
                status = mptsas_diag_unregister(mpt, &diag_unregister,
                    return_code);
                break;

        case MPTSAS_FW_DIAG_TYPE_QUERY:
                if (length < sizeof (diag_query)) {
                        *return_code =
                                MPTSAS_FW_DIAG_ERROR_INVALID_PARAMETER;
                        status = DDI_FAILURE;
                        break;
                }
                if (ddi_copyin(diag_action, &diag_query,
                    sizeof (diag_query), ioctl_mode) != 0) {
                        return (DDI_FAILURE);
                }
                status = mptsas_diag_query(mpt, &diag_query,
                    return_code);
                if (status == DDI_SUCCESS) {
                        if (ddi_copyout(&diag_query, diag_action,
                            sizeof (diag_query), ioctl_mode) != 0) {
                                return (DDI_FAILURE);
                        }
                }
                break;

        case MPTSAS_FW_DIAG_TYPE_READ_BUFFER:
                if (ddi_copyin(diag_action, &diag_read_buffer,
                    sizeof (diag_read_buffer) - 4, ioctl_mode) != 0) {
                        return (DDI_FAILURE);
                }
                read_buf_len = sizeof (diag_read_buffer) -
                    sizeof (diag_read_buffer.DataBuffer) +
                    diag_read_buffer.BytesToRead;
                if (length < read_buf_len) {
                        *return_code =
                                MPTSAS_FW_DIAG_ERROR_INVALID_PARAMETER;
                        status = DDI_FAILURE;
                        break;
                }
                status = mptsas_diag_read_buffer(mpt,
                    &diag_read_buffer, diag_action +
                    sizeof (diag_read_buffer) - 4, return_code,
                    ioctl_mode);
```



```
                if (status == DDI_SUCCESS) {
                        if (ddi_copyout(&diag_read_buffer, diag_action,
                            sizeof (diag_read_buffer) - 4, ioctl_mode)
                            != 0) {
                                return (DDI_FAILURE);
                        }
                }
                break;

        case MPTSAS_FW_DIAG_TYPE_RELEASE:
                if (length < sizeof (diag_release)) {
                        *return_code =
                            MPTSAS_FW_DIAG_ERROR_INVALID_PARAMETER;
                        status = DDI_FAILURE;
                        break;
                }
                if (ddi_copyin(diag_action, &diag_release,
                    sizeof (diag_release), ioctl_mode) != 0) {
                        return (DDI_FAILURE);
                }
                status = mptsas_diag_release(mpt, &diag_release,
                    return_code);
                break;

        default:
                *return_code = MPTSAS_FW_DIAG_ERROR_INVALID_PARAMETER;
                status = DDI_FAILURE;
                break;
        }

        if ((status == DDI_FAILURE) &&
            (original_return_code == MPTSAS_FW_DIAG_NEW) &&
            (*return_code != MPTSAS_FW_DIAG_ERROR_SUCCESS)) {
                status = DDI_SUCCESS;
        }

        return (status);
}

static int
mptsas_diag_action(mptsas_t *mpt, mptsas_diag_action_t *user_data, int mode)
{
        int                     status;
        mptsas_diag_action_t    driver_data;

        ASSERT(mutex_owned(&mpt->m_mutex));

        /*
         * Copy the user data to a driver data buffer.
         */
        if (ddi_copyin(user_data, &driver_data, sizeof (mptsas_diag_action_t),
            mode) == 0) {
                /*
                 * Send diag action request if Action is valid
                 */
                if (driver_data.Action == MPTSAS_FW_DIAG_TYPE_REGISTER ||
                    driver_data.Action == MPTSAS_FW_DIAG_TYPE_UNREGISTER ||
                    driver_data.Action == MPTSAS_FW_DIAG_TYPE_QUERY ||
                    driver_data.Action == MPTSAS_FW_DIAG_TYPE_READ_BUFFER ||
                    driver_data.Action == MPTSAS_FW_DIAG_TYPE_RELEASE) {
                        status = mptsas_do_diag_action(mpt, driver_data.Action,
                            (void *)(uintptr_t)driver_data.PtrDiagAction,
                            driver_data.Length, &driver_data.ReturnCode,
                            mode);
                        if (status == DDI_SUCCESS) {
                                if (ddi_copyout(&driver_data.ReturnCode,
```

```
                                        &user_data->ReturnCode,
                                        sizeof (user_data->ReturnCode), mode)
                                        != 0) {
                                            status = EFAULT;
                                    } else {
                                            status = 0;
                                    }
                            } else {
                                    status = EIO;
                            }
                    } else {
                            status = EINVAL;
                    }
            } else {
                    status = EFAULT;
            }

            return (status);
}

/*
 * This routine handles the "event query" ioctl.
 */
static int
mptsas_event_query(mptsas_t *mpt, mptsas_event_query_t *data, int mode,
    int *rval)
{
            int                     status;
            mptsas_event_query_t    driverdata;
            uint8_t                 i;

            driverdata.Entries = MPTSAS_EVENT_QUEUE_SIZE;

            mutex_enter(&mpt->m_mutex);
            for (i = 0; i < 4; i++) {
                    driverdata.Types[i] = mpt->m_event_mask[i];
            }
            mutex_exit(&mpt->m_mutex);

            if (ddi_copyout(&driverdata, data, sizeof (driverdata), mode) != 0) {
                    status = EFAULT;
            } else {
                    *rval = MPTIOCTL_STATUS_GOOD;
                    status = 0;
            }

            return (status);
}

/*
 * This routine handles the "event enable" ioctl.
 */
static int
mptsas_event_enable(mptsas_t *mpt, mptsas_event_enable_t *data, int mode,
    int *rval)
{
            int                     status;
            mptsas_event_enable_t   driverdata;
            uint8_t                 i;

            if (ddi_copyin(data, &driverdata, sizeof (driverdata), mode) == 0) {
                    mutex_enter(&mpt->m_mutex);
                    for (i = 0; i < 4; i++) {
                            mpt->m_event_mask[i] = driverdata.Types[i];
                    }
                    mutex_exit(&mpt->m_mutex);
```



```
                    *rval = MPTIOCTL_STATUS_GOOD;
                    status = 0;
            } else {
                    status = EFAULT;
            }
            return (status);
}

/*
 * This routine handles the "event report" ioctl.
 */
static int
mptsas_event_report(mptsas_t *mpt, mptsas_event_report_t *data, int mode,
    int *rval)
{
            int                     status;
            mptsas_event_report_t   driverdata;

            mutex_enter(&mpt->m_mutex);

            if (ddi_copyin(&data->Size, &driverdata.Size, sizeof (driverdata.Size),
                mode) == 0) {
                    if (driverdata.Size >= sizeof (mpt->m_events)) {
                            if (ddi_copyout(mpt->m_events, data->Events,
                                sizeof (mpt->m_events), mode) != 0) {
                                    status = EFAULT;
                            } else {
                                    if (driverdata.Size > sizeof (mpt->m_events)) {
                                            driverdata.Size =
                                                sizeof (mpt->m_events);
                                            if (ddi_copyout(&driverdata.Size,
                                                &data->Size,
                                                sizeof (driverdata.Size),
                                                mode) != 0) {
                                                    status = EFAULT;
                                            } else {
                                                    *rval = MPTIOCTL_STATUS_GOOD;
                                                    status = 0;
                                            }
                                    } else {
                                            *rval = MPTIOCTL_STATUS_GOOD;
                                            status = 0;
                                    }
                            }
                    } else {
                            *rval = MPTIOCTL_STATUS_LEN_TOO_SHORT;
                            status = 0;
                    }
            } else {
                    status = EFAULT;
            }

            mutex_exit(&mpt->m_mutex);
            return (status);
}

static void
mptsas_lookup_pci_data(mptsas_t *mpt, mptsas_adapter_data_t *adapter_data)
{
            int     *reg_data;
            uint_t  reglen;

            /*
             * Lookup the 'reg' property and extract the other data
             */
```

```
        if (ddi_prop_lookup_int_array(DDI_DEV_T_ANY, mpt->m_dip,
            DDI_PROP_DONTPASS, "reg", &reg_data, &reglen) ==
            DDI_PROP_SUCCESS) {
                /*
                 * Extract the PCI data from the 'reg' property first DWORD.
                 * The entry looks like the following:
                 * First DWORD:
                 * Bits 0 - 7 8-bit Register number
                 * Bits 8 - 10 3-bit Function number
                 * Bits 11 - 15 5-bit Device number
                 * Bits 16 - 23 8-bit Bus number
                 * Bits 24 - 25 2-bit Address Space type identifier
                 *
                 */
                adapter_data->PciInformation.u.bits.BusNumber =
                    (reg_data[0] & 0x00FF0000) >> 16;
                adapter_data->PciInformation.u.bits.DeviceNumber =
                    (reg_data[0] & 0x0000F800) >> 11;
                adapter_data->PciInformation.u.bits.FunctionNumber =
                    (reg_data[0] & 0x00000700) >> 8;
                ddi_prop_free((void *)reg_data);
        } else {
                /*
                 * If we can't determine the PCI data then we fill in FF's for
                 * the data to indicate this.
                 */
                adapter_data->PCIDeviceHwId = 0xFFFFFFFF;
                adapter_data->MpiPortNumber = 0xFFFFFFFF;
                adapter_data->PciInformation.u.AsDWORD = 0xFFFFFFFF;
        }

        /*
         * Saved in the mpt->m_fwversion
         */
        adapter_data->MpiFirmwareVersion = mpt->m_fwversion;
}

static void
mptsas_read_adapter_data(mptsas_t *mpt, mptsas_adapter_data_t *adapter_data)
{
        char    *driver_verstr = MPTSAS_MOD_STRING;

        mptsas_lookup_pci_data(mpt, adapter_data);
        adapter_data->AdapterType = MPTIOCTL_ADAPTER_TYPE_SAS3;
        adapter_data->PCIDeviceHwId = (uint32_t)mpt->m_devid;
        adapter_data->PCIDeviceHwRev = (uint32_t)mpt->m_revid;
        adapter_data->SubSystemId = (uint32_t)mpt->m_ssid;
        adapter_data->SubsystemVendorId = (uint32_t)mpt->m_svid;
        (void) strcpy((char *)&adapter_data->DriverVersion[0], driver_verstr);
        adapter_data->BiosVersion = 0;
        (void) mptsas_get_bios_page3(mpt, &adapter_data->BiosVersion);
}

static void
mptsas_read_pci_info(mptsas_t *mpt, mptsas_pci_info_t *pci_info)
{
        int     *reg_data, i;
        uint_t  reglen;

        /*
         * Lookup the 'reg' property and extract the other data
         */
        if (ddi_prop_lookup_int_array(DDI_DEV_T_ANY, mpt->m_dip,
            DDI_PROP_DONTPASS, "reg", &reg_data, &reglen) ==
            DDI_PROP_SUCCESS) {
                /*
```



```
                 * Extract the PCI data from the 'reg' property first DWORD.
                 * The entry looks like the following:
                 * First DWORD:
                 * Bits 8 - 10 3-bit Function number
                 * Bits 11 - 15 5-bit Device number
                 * Bits 16 - 23 8-bit Bus number
                 */
                pci_info->BusNumber = (reg_data[0] & 0x00FF0000) >> 16;
                pci_info->DeviceNumber = (reg_data[0] & 0x0000F800) >> 11;
                pci_info->FunctionNumber = (reg_data[0] & 0x00000700) >> 8;
                ddi_prop_free((void *)reg_data);
        } else {
                /*
                 * If we can't determine the PCI info then we fill in FF's for
                 * the data to indicate this.
                 */
                pci_info->BusNumber = 0xFFFFFFFF;
                pci_info->DeviceNumber = 0xFF;
                pci_info->FunctionNumber = 0xFF;
        }

        /*
         * Now get the interrupt vector and the pci header.  The vector can
         * only be 0 right now.  The header is the first 256 bytes of config
         * space.
         */
        pci_info->InterruptVector = 0;
        for (i = 0; i < sizeof (pci_info->PciHeader); i++) {
                pci_info->PciHeader[i] = pci_config_get8(mpt->m_config_handle,
                    i);
        }
}

static int
mptsas_reg_access(mptsas_t *mpt, mptsas_reg_access_t *data, int mode)
{
        int                     status = 0;
        mptsas_reg_access_t     driverdata;

        mutex_enter(&mpt->m_mutex);
        if (ddi_copyin(data, &driverdata, sizeof (driverdata), mode) == 0) {
                switch (driverdata.Command) {
                        /*
                         * IO access is not supported.
                         */
                        case REG_IO_READ:
                        case REG_IO_WRITE:
                                mptsas_log(mpt, CE_WARN, "IO access is not "
                                    "supported.  Use memory access.");
                                status = EINVAL;
                                break;

                        case REG_MEM_READ:
                                driverdata.RegData = ddi_get32(mpt->m_datap,
                                    (uint32_t *)(void *)mpt->m_reg +
                                    driverdata.RegOffset);
                                if (ddi_copyout(&driverdata.RegData,
                                    &data->RegData,
                                    sizeof (driverdata.RegData), mode) != 0) {
                                        mptsas_log(mpt, CE_WARN, "Register "
                                            "Read Failed");
                                        status = EFAULT;
                                }
                                break;

                        case REG_MEM_WRITE:
```

```
                                ddi_put32(mpt->m_datap,
                                    (uint32_t *)(void *)mpt->m_reg +
                                    driverdata.RegOffset,
                                    driverdata.RegData);
                                break;

                        default:
                                status = EINVAL;
                                break;
                }
        } else {
                status = EFAULT;
        }

        mutex_exit(&mpt->m_mutex);
        return (status);
}

static int
led_control(mptsas_t *mpt, intptr_t data, int mode)
{
        int ret = 0;
        mptsas_led_control_t lc;
        mptsas_target_t *ptgt;

        if (ddi_copyin((void *)data, &lc, sizeof (lc), mode) != 0) {
                return (EFAULT);
        }

        if ((lc.Command != MPTSAS_LEDCTL_FLAG_SET &&
            lc.Command != MPTSAS_LEDCTL_FLAG_GET) ||
            lc.Led < MPTSAS_LEDCTL_LED_MIN ||
            lc.Led > MPTSAS_LEDCTL_LED_MAX ||
            (lc.Command == MPTSAS_LEDCTL_FLAG_SET && lc.LedStatus != 0 &&
            lc.LedStatus != 1)) {
                return (EINVAL);
        }

        if ((lc.Command == MPTSAS_LEDCTL_FLAG_SET && (mode & FWRITE) == 0) ||
            (lc.Command == MPTSAS_LEDCTL_FLAG_GET && (mode & FREAD) == 0))
                return (EACCES);

        /* Locate the target we're interrogating... */
        mutex_enter(&mpt->m_mutex);
        ptgt = refhash_linear_search(mpt->m_targets,
            mptsas_target_eval_slot, &lc);
        if (ptgt == NULL) {
                /* We could not find a target for that enclosure/slot. */
                mutex_exit(&mpt->m_mutex);
                return (ENOENT);
        }

        if (lc.Command == MPTSAS_LEDCTL_FLAG_SET) {
                /* Update our internal LED state. */
                ptgt->m_led_status &= ~(1 << (lc.Led - 1));
                ptgt->m_led_status |= lc.LedStatus << (lc.Led - 1);

                /* Flush it to the controller. */
                ret = mptsas_flush_led_status(mpt, ptgt);
                mutex_exit(&mpt->m_mutex);
                return (ret);
        }

        /* Return our internal LED state. */
        lc.LedStatus = (ptgt->m_led_status >> (lc.Led - 1)) & 1;
        mutex_exit(&mpt->m_mutex);
```



```
        if (ddi_copyout(&lc, (void *)data, sizeof (lc), mode) != 0) {
                return (EFAULT);
        }

        return (0);
}

static int
get_disk_info(mptsas_t *mpt, intptr_t data, int mode)
{
        uint16_t i = 0;
        uint16_t count = 0;
        int ret = 0;
        mptsas_target_t *ptgt;
        mptsas_disk_info_t *di;
        STRUCT_DECL(mptsas_get_disk_info, gdi);

        if ((mode & FREAD) == 0)
                return (EACCES);

        STRUCT_INIT(gdi, get_udatamodel());

        if (ddi_copyin((void *)data, STRUCT_BUF(gdi), STRUCT_SIZE(gdi),
            mode) != 0) {
                return (EFAULT);
        }

        /* Find out how many targets there are. */
        mutex_enter(&mpt->m_mutex);
        for (ptgt = refhash_first(mpt->m_targets); ptgt != NULL;
            ptgt = refhash_next(mpt->m_targets, ptgt)) {
                count++;
        }
        mutex_exit(&mpt->m_mutex);

        /*
         * If we haven't been asked to copy out information on each target,
         * then just return the count.
         */
        STRUCT_FSET(gdi, DiskCount, count);
        if (STRUCT_FGETP(gdi, PtrDiskInfoArray) == NULL)
                goto copy_out;

        /*
         * If we haven't been given a large enough buffer to copy out into,
         * let the caller know.
         */
        if (STRUCT_FGET(gdi, DiskInfoArraySize) <
            count * sizeof (mptsas_disk_info_t)) {
                ret = ENOSPC;
                goto copy_out;
        }

        di = kmem_zalloc(count * sizeof (mptsas_disk_info_t), KM_SLEEP);

        mutex_enter(&mpt->m_mutex);
        for (ptgt = refhash_first(mpt->m_targets); ptgt != NULL;
            ptgt = refhash_next(mpt->m_targets, ptgt)) {
                if (i >= count) {
                        /*
                         * The number of targets changed while we weren't
                         * looking, so give up.
                         */
                        refhash_rele(mpt->m_targets, ptgt);
                        mutex_exit(&mpt->m_mutex);
```

```
                        kmem_free(di, count * sizeof (mptsas_disk_info_t));
                        return (EAGAIN);
                }
                di[i].Instance = mpt->m_instance;
                di[i].Enclosure = ptgt->m_enclosure;
                di[i].Slot = ptgt->m_slot_num;
                di[i].SasAddress = ptgt->m_addr.mta_wwn;
                i++;
        }
        mutex_exit(&mpt->m_mutex);
        STRUCT_FSET(gdi, DiskCount, i);

        /* Copy out the disk information to the caller. */
        if (ddi_copyout((void *)di, STRUCT_FGETP(gdi, PtrDiskInfoArray),
            i * sizeof (mptsas_disk_info_t), mode) != 0) {
                ret = EFAULT;
        }

        kmem_free(di, count * sizeof (mptsas_disk_info_t));

copy_out:
        if (ddi_copyout(STRUCT_BUF(gdi), (void *)data, STRUCT_SIZE(gdi),
            mode) != 0) {
                ret = EFAULT;
        }

        return (ret);
}

static int
mptsas_ioctl(dev_t dev, int cmd, intptr_t data, int mode, cred_t *credp,
    int *rval)
{
        int                     status = 0;
        mptsas_t                *mpt;
        mptsas_update_flash_t   flashdata;
        mptsas_pass_thru_t      passthru_data;
        mptsas_adapter_data_t   adapter_data;
        mptsas_pci_info_t       pci_info;
        int                     copylen;

        int                     iport_flag = 0;
        dev_info_t              *dip = NULL;
        mptsas_phymask_t        phymask = 0;
        struct devctl_iocdata   *dcp = NULL;
        char                    *addr = NULL;
        mptsas_target_t         *ptgt = NULL;

        *rval = MPTIOCTL_STATUS_GOOD;
        if (secpolicy_sys_config(credp, B_FALSE) != 0) {
                return (EPERM);
        }

        mpt = ddi_get_soft_state(mptsas3_state, MINOR2INST(getminor(dev)));
        if (mpt == NULL) {
                /*
                 * Called from iport node, get the states
                 */
                iport_flag = 1;
                dip = mptsas_get_dip_from_dev(dev, &phymask);
                if (dip == NULL) {
                        return (ENXIO);
                }
                mpt = DIP2MPT(dip);
        }
        /* Make sure power level is D0 before accessing registers */
```



```
        mutex_enter(&mpt->m_mutex);
        if (mpt->m_options & MPTSAS_OPT_PM) {
                (void) pm_busy_component(mpt->m_dip, 0);
                if (mpt->m_power_level != PM_LEVEL_D0) {
                        mutex_exit(&mpt->m_mutex);
                        if (pm_raise_power(mpt->m_dip, 0, PM_LEVEL_D0) !=
                            DDI_SUCCESS) {
                                mptsas_log(mpt, CE_WARN,
                                    "mptsas3%d: mptsas_ioctl: Raise power "
                                    "request failed.", mpt->m_instance);
                                (void) pm_idle_component(mpt->m_dip, 0);
                                return (ENXIO);
                        }
                } else {
                        mutex_exit(&mpt->m_mutex);
                }
        } else {
                mutex_exit(&mpt->m_mutex);
        }

        if (iport_flag) {
                status = scsi_hba_ioctl(dev, cmd, data, mode, credp, rval);
                if (status != 0) {
                        goto out;
                }
                /*
                 * The following code control the OK2RM LED, it doesn't affect
                 * the ioctl return status.
                 */
                if ((cmd == DEVCTL_DEVICE_ONLINE) ||
                    (cmd == DEVCTL_DEVICE_OFFLINE)) {
                        if (ndi_dc_allochdl((void *)data, &dcp) !=
                            NDI_SUCCESS) {
                                goto out;
                        }
                        addr = ndi_dc_getaddr(dcp);
                        ptgt = mptsas_addr_to_ptgt(mpt, addr, phymask);
                        if (ptgt == NULL) {
                                NDBG14(("mptsas_ioctl led control: tgt %s not "
                                    "found", addr));
                                ndi_dc_freehdl(dcp);
                                goto out;
                        }
                        mutex_enter(&mpt->m_mutex);
                        if (cmd == DEVCTL_DEVICE_ONLINE) {
                                ptgt->m_tgt_unconfigured = 0;
                        } else if (cmd == DEVCTL_DEVICE_OFFLINE) {
                                ptgt->m_tgt_unconfigured = 1;
                        }
                        if (cmd == DEVCTL_DEVICE_OFFLINE) {
                                ptgt->m_led_status |=
                                    (1 << (MPTSAS_LEDCTL_LED_OK2RM - 1));
                        } else {
                                ptgt->m_led_status &=
                                    ~(1 << (MPTSAS_LEDCTL_LED_OK2RM - 1));
                        }
                        (void) mptsas_flush_led_status(mpt, ptgt);
                        mutex_exit(&mpt->m_mutex);
                        ndi_dc_freehdl(dcp);
                }
                goto out;
        }
        switch (cmd) {
                case MPTIOCTL_GET_DISK_INFO:
                        status = get_disk_info(mpt, data, mode);
                        break;
```

```
                case MPTIOCTL_LED_CONTROL:
                        status = led_control(mpt, data, mode);
                        break;
                case MPTIOCTL_UPDATE_FLASH:
                        if (ddi_copyin((void *)data, &flashdata,
                                sizeof (struct mptsas_update_flash), mode)) {
                                status = EFAULT;
                                break;
                        }

                        mutex_enter(&mpt->m_mutex);
                        if (mptsas_update_flash(mpt,
                            (caddr_t)(long)flashdata.PtrBuffer,
                            flashdata.ImageSize, flashdata.ImageType, mode)) {
                                status = EFAULT;
                        }

                        /*
                         * Reset the chip to start using the new
                         * firmware.  Reset if failed also.
                         */
                        mpt->m_softstate &= ~MPTSAS_SS_MSG_UNIT_RESET;
                        if (mptsas_restart_ioc(mpt) == DDI_FAILURE) {
                                status = EFAULT;
                        }
                        mutex_exit(&mpt->m_mutex);
                        break;
                case MPTIOCTL_PASS_THRU:
                        /*
                         * The user has requested to pass through a command to
                         * be executed by the MPT firmware.  Call our routine
                         * which does this.  Only allow one passthru IOCTL at
                         * one time. Other threads will block on
                         * m_passthru_mutex, which is of adaptive variant.
                         */
                        if (ddi_copyin((void *)data, &passthru_data,
                            sizeof (mptsas_pass_thru_t), mode)) {
                                status = EFAULT;
                                break;
                        }
                        mutex_enter(&mpt->m_passthru_mutex);
                        mutex_enter(&mpt->m_mutex);
                        status = mptsas_pass_thru(mpt, &passthru_data, mode);
                        mutex_exit(&mpt->m_mutex);
                        mutex_exit(&mpt->m_passthru_mutex);

                        break;
                case MPTIOCTL_GET_ADAPTER_DATA:
                        /*
                         * The user has requested to read adapter data.  Call
                         * our routine which does this.
                         */
                        bzero(&adapter_data, sizeof (mptsas_adapter_data_t));
                        if (ddi_copyin((void *)data, (void *)&adapter_data,
                            sizeof (mptsas_adapter_data_t), mode)) {
                                status = EFAULT;
                                break;
                        }
                        if (adapter_data.StructureLength >=
                            sizeof (mptsas_adapter_data_t)) {
                                adapter_data.StructureLength = (uint32_t)
                                    sizeof (mptsas_adapter_data_t);
                                copylen = sizeof (mptsas_adapter_data_t);
                                mutex_enter(&mpt->m_mutex);
                                mptsas_read_adapter_data(mpt, &adapter_data);
                                mutex_exit(&mpt->m_mutex);
```



```
                        } else {
                                adapter_data.StructureLength = (uint32_t)
                                    sizeof (mptsas_adapter_data_t);
                                copylen = sizeof (adapter_data.StructureLength);
                                *rval = MPTIOCTL_STATUS_LEN_TOO_SHORT;
                        }
                        if (ddi_copyout((void *)(&adapter_data), (void *)data,
                            copylen, mode) != 0) {
                                status = EFAULT;
                        }
                        break;
                case MPTIOCTL_GET_PCI_INFO:
                        /*
                         * The user has requested to read pci info.  Call
                         * our routine which does this.
                         */
                        bzero(&pci_info, sizeof (mptsas_pci_info_t));
                        mutex_enter(&mpt->m_mutex);
                        mptsas_read_pci_info(mpt, &pci_info);
                        mutex_exit(&mpt->m_mutex);
                        if (ddi_copyout((void *)(&pci_info), (void *)data,
                            sizeof (mptsas_pci_info_t), mode) != 0) {
                                status = EFAULT;
                        }
                        break;
                case MPTIOCTL_RESET_ADAPTER:
                        mutex_enter(&mpt->m_mutex);
                        mpt->m_softstate &= ~MPTSAS_SS_MSG_UNIT_RESET;
                        if ((mptsas_restart_ioc(mpt)) == DDI_FAILURE) {
                                mptsas_log(mpt, CE_WARN, "reset adapter IOCTL "
                                    "failed");
                                status = EFAULT;
                        }
                        mutex_exit(&mpt->m_mutex);
                        break;
                case MPTIOCTL_DIAG_ACTION:
                        /*
                         * The user has done a diag buffer action.  Call our
                         * routine which does this.  Only allow one diag action
                         * at one time.
                         */
                        mutex_enter(&mpt->m_mutex);
                        if (mpt->m_diag_action_in_progress) {
                                mutex_exit(&mpt->m_mutex);
                                return (EBUSY);
                        }
                        mpt->m_diag_action_in_progress = 1;
                        status = mptsas_diag_action(mpt,
                            (mptsas_diag_action_t *)data, mode);
                        mpt->m_diag_action_in_progress = 0;
                        mutex_exit(&mpt->m_mutex);
                        break;
                case MPTIOCTL_EVENT_QUERY:
                        /*
                         * The user has done an event query. Call our routine
                         * which does this.
                         */
                        status = mptsas_event_query(mpt,
                            (mptsas_event_query_t *)data, mode, rval);
                        break;
                case MPTIOCTL_EVENT_ENABLE:
                        /*
                         * The user has done an event enable. Call our routine
                         * which does this.
                         */
                        status = mptsas_event_enable(mpt,
```

```
                            (mptsas_event_enable_t *)data, mode, rval);
                        break;
                case MPTIOCTL_EVENT_REPORT:
                        /*
                         * The user has done an event report. Call our routine
                         * which does this.
                         */
                        status = mptsas_event_report(mpt,
                            (mptsas_event_report_t *)data, mode, rval);
                        break;
                case MPTIOCTL_REG_ACCESS:
                        /*
                         * The user has requested register access.  Call our
                         * routine which does this.
                         */
                        status = mptsas_reg_access(mpt,
                            (mptsas_reg_access_t *)data, mode);
                        break;
                default:
                        status = scsi_hba_ioctl(dev, cmd, data, mode, credp,
                            rval);
                        break;
        }

out:
        return (status);
}

int
mptsas_restart_ioc(mptsas_t *mpt)
{
        int             rval = DDI_SUCCESS;
        mptsas_target_t *ptgt = NULL;

        ASSERT(mutex_owned(&mpt->m_mutex));

        /*
         * Set a flag telling I/O path that we’re processing a reset.  This is
         * needed because after the reset is complete, the hash table still
         * needs to be rebuilt.  If I/Os are started before the hash table is
         * rebuilt, I/O errors will occur.  This flag allows I/Os to be marked
         * so that they can be retried.
         */
        mpt->m_in_reset = TRUE;

        /*
         * Set all throttles to HOLD
         */
        for (ptgt = refhash_first(mpt->m_targets); ptgt != NULL;
            ptgt = refhash_next(mpt->m_targets, ptgt)) {
                mptsas_set_throttle_mtx(mpt, ptgt, HOLD_THROTTLE);
        }

        /*
         * Disable interrupts
         */
        MPTSAS_DISABLE_INTR(mpt);

        /*
         * Abort all commands: outstanding commands, commands in waitq and
         * tx_waitq.
         */
        mptsas_flush_hba(mpt);

        /*
         * Reinitialize the chip.
```



```
         */
        if (mptsas_init_chip(mpt, FALSE) == DDI_FAILURE) {
                rval = DDI_FAILURE;
        }

        /*
         * Enable interrupts again
         */
        MPTSAS_ENABLE_INTR(mpt);

        /*
         * If mptsas_init_chip was successful, update the driver data.
         */
        if (rval == DDI_SUCCESS) {
                mptsas_update_driver_data(mpt);
        }

        /*
         * Reset the throttles
         */
        for (ptgt = refhash_first(mpt->m_targets); ptgt != NULL;
            ptgt = refhash_next(mpt->m_targets, ptgt)) {
                mptsas_set_throttle_mtx(mpt, ptgt, MAX_THROTTLE);
        }

        mptsas_doneq_empty(mpt);
        mptsas_restart_hba(mpt);

        if (rval != DDI_SUCCESS) {
                mptsas_fm_ereport(mpt, DDI_FM_DEVICE_NO_RESPONSE);
                ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_LOST);
        }

        /*
         * Clear the reset flag so that I/Os can continue.
         */
        mpt->m_in_reset = FALSE;

        return (rval);
}

static int
mptsas_init_chip(mptsas_t *mpt, int first_time)
{
        ddi_dma_cookie_t        cookie;
        mptsas_reply_pqueue_t   *rpqp;
        uint32_t                i, j;
        int                     rval;

        /*
         * Check to see if the firmware image is valid
         */
        if (ddi_get32(mpt->m_datap, &mpt->m_reg->HostDiagnostic) &
            MPI2_DIAG_FLASH_BAD_SIG) {
                mptsas_log(mpt, CE_WARN, "mptsas bad flash signature!");
                goto fail;
        }

        /*
         * Reset the chip
         */
        rval = mptsas_ioc_reset(mpt, first_time);
        if (rval == MPTSAS_RESET_FAIL) {
                mptsas_log(mpt, CE_WARN, "hard reset failed!");
                goto fail;
        }
```

```
        if ((rval == MPTSAS_SUCCESS_MUR) && (!first_time)) {
                goto mur;
        }
        /*
         * Setup configuration space
         */
        if (mptsas_config_space_init(mpt) == FALSE) {
                mptsas_log(mpt, CE_WARN, "mptsas_config_space_init "
                    "failed!");
                goto fail;
        }

        /*
         * IOC facts can change after a diag reset so all buffers that are
         * based on these numbers must be de-allocated and re-allocated.  Get
         * new IOC facts each time chip is initialized.
         */
        if (mptsas_ioc_get_facts(mpt) == DDI_FAILURE) {
                mptsas_log(mpt, CE_WARN, "mptsas_ioc_get_facts failed");
                goto fail;
        }

        /*
         * Now we know chip MSIX capabilitites and it's not been done
         * previously register interrupts accordingly. Need to know this
         * information before allocating the reply frames below.
         */
        if (mpt->m_intr_cnt == 0) {
                if (mptsas_register_intrs(mpt) == FALSE)
                        goto fail;
        }

        mpt->m_targets = refhash_create(MPTSAS_TARGET_BUCKET_COUNT,
            mptsas_target_addr_hash, mptsas_target_addr_cmp,
            mptsas_target_free, sizeof (mptsas_target_t),
            offsetof(mptsas_target_t, m_link),
            offsetof(mptsas_target_t, m_addr), KM_SLEEP);

        if (mptsas_alloc_active_slots(mpt, KM_SLEEP)) {
                goto fail;
        }
        /*
         * Allocate request message frames, reply free queue, reply descriptor
         * post queue, and reply message frames using latest IOC facts.
         */
        if (mptsas_alloc_request_frames(mpt) == DDI_FAILURE) {
                mptsas_log(mpt, CE_WARN, "mptsas_alloc_request_frames failed");
                goto fail;
        }
        if (mptsas_alloc_sense_bufs(mpt) == DDI_FAILURE) {
                mptsas_log(mpt, CE_WARN, "mptsas_alloc_sense_bufs failed");
                goto fail;
        }
        if (mptsas_alloc_free_queue(mpt) == DDI_FAILURE) {
                mptsas_log(mpt, CE_WARN, "mptsas_alloc_free_queue failed!");
                goto fail;
        }
        if (mptsas_alloc_post_queue(mpt) == DDI_FAILURE) {
                mptsas_log(mpt, CE_WARN, "mptsas_alloc_post_queue failed!");
                goto fail;
        }
        if (mptsas_alloc_reply_frames(mpt) == DDI_FAILURE) {
                mptsas_log(mpt, CE_WARN, "mptsas_alloc_reply_frames failed!");
                goto fail;
        }
```



```
mur:
        /*
         * Re-Initialize ioc to operational state
         */
        if (mptsas_ioc_init(mpt) == DDI_FAILURE) {
                mptsas_log(mpt, CE_WARN, "mptsas_ioc_init failed");
                goto fail;
        }

        mptsas_alloc_reply_args(mpt);

        /*
         * Initialize the Reply Free Queue with the physical addresses of our
         * reply frames.
         */
        cookie.dmac_address = mpt->m_reply_frame_dma_addr&0xffffffffu;
        for (i = 0; i < mpt->m_max_replies; i++) {
                ddi_put32(mpt->m_acc_free_queue_hdl,
                    &((uint32_t *)(void *)mpt->m_free_queue)[i],
                    cookie.dmac_address);
                cookie.dmac_address += mpt->m_reply_frame_size;
        }
        (void) ddi_dma_sync(mpt->m_dma_free_queue_hdl, 0, 0,
            DDI_DMA_SYNC_FORDEV);

        /*
         * Initialize the reply free index to one past the last frame on the
         * queue.  This will signify that the queue is empty to start with.
         */
        mpt->m_free_index = i;
        ddi_put32(mpt->m_datap, &mpt->m_reg->ReplyFreeHostIndex, i);

        /*
         * Initialize the reply post queue to 0xFFFFFFFF,0xFFFFFFFF's
         * and the indexes to 0.
         */
        rpqp = mpt->m_rep_post_queues;
        for (j = 0; j < mpt->m_post_reply_qcount; j++) {
                for (i = 0; i < mpt->m_post_queue_depth; i++) {
                        ddi_put64(mpt->m_acc_post_queue_hdl,
                            &((uint64_t *)(void *)rpqp->rpq_queue)[i],
                            0xFFFFFFFFFFFFFFFF);
                }
                rpqp->rpq_index = 0;
                rpqp++;
        }
        (void) ddi_dma_sync(mpt->m_dma_post_queue_hdl, 0, 0,
            DDI_DMA_SYNC_FORDEV);

        /*
         * Initialise all the reply post queue indexes.
         */
        for (j = 0; j < mpt->m_post_reply_qcount; j++) {
                ddi_put32(mpt->m_datap, &mpt->m_reg->ReplyPostHostIndex,
                    j << MPI2_RPHI_MSIX_INDEX_SHIFT);
        }

        /*
         * Enable ports
         */
        if (mptsas_ioc_enable_port(mpt) == DDI_FAILURE) {
                mptsas_log(mpt, CE_WARN, "mptsas_ioc_enable_port failed");
                goto fail;
        }
```

```
        /*
         * enable events
         */
        if (mptsas_ioc_enable_event_notification(mpt)) {
                mptsas_log(mpt, CE_WARN,
                        "mptsas_ioc_enable_event_notification failed");
                goto fail;
        }

        /*
         * We need checks in attach and these.
         * chip_init is called in mult. places
         */

        if ((mptsas_check_dma_handle(mpt->m_dma_req_frame_hdl) !=
            DDI_SUCCESS) ||
            (mptsas_check_dma_handle(mpt->m_dma_req_sense_hdl) !=
            DDI_SUCCESS) ||
            (mptsas_check_dma_handle(mpt->m_dma_reply_frame_hdl) !=
            DDI_SUCCESS) ||
            (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)) {
                ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_UNAFFECTED);
                goto fail;
        }

        /* Check all acc 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) ||
            (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)) {
                ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_UNAFFECTED);
                goto fail;
        }

        return (DDI_SUCCESS);

fail:
        return (DDI_FAILURE);
}

static int
mptsas_get_pci_cap(mptsas_t *mpt)
{
        ushort_t caps_ptr, cap, cap_count;

        if (mpt->m_config_handle == NULL)
                return (FALSE);
        /*
         * Check if capabilities list is supported and if so,
         * get initial capabilities pointer and clear bits 0,1.
```



```
         */
        if (pci_config_get16(mpt->m_config_handle, PCI_CONF_STAT)
            & PCI_STAT_CAP) {
                caps_ptr = P2ALIGN(pci_config_get8(mpt->m_config_handle,
                    PCI_CONF_CAP_PTR), 4);
        } else {
                caps_ptr = PCI_CAP_NEXT_PTR_NULL;
        }

        /*
         * Walk capabilities if supported.
         */
        for (cap_count = 0; caps_ptr != PCI_CAP_NEXT_PTR_NULL; ) {

                /*
                 * Check that we haven't exceeded the maximum number of
                 * capabilities and that the pointer is in a valid range.
                 */
                if (++cap_count > 48) {
                        mptsas_log(mpt, CE_WARN,
                            "too many device capabilities.\n");
                        break;
                }
                if (caps_ptr < 64) {
                        mptsas_log(mpt, CE_WARN,
                            "capabilities pointer 0x%x out of range.\n",
                            caps_ptr);
                        break;
                }

                /*
                 * Get next capability and check that it is valid.
                 * For now, we only support power management.
                 */
                cap = pci_config_get8(mpt->m_config_handle, caps_ptr);
                switch (cap) {
                        case PCI_CAP_ID_PM:
                                mptsas_log(mpt, CE_NOTE,
                                    "?mptsas3%d supports power management.\n",
                                    mpt->m_instance);
                                mpt->m_options |= MPTSAS_OPT_PM;

                                /* Save PMCSR offset */
                                mpt->m_pmcsr_offset = caps_ptr + PCI_PMCSR;
                                break;
                        case PCI_CAP_ID_MSI:
                                mptsas_log(mpt, CE_NOTE,
                                    "?mptsas3%d supports MSI.\n",
                                    mpt->m_instance);
                                mpt->m_options |= MPTSAS_OPT_MSI;
                                break;
                        case PCI_CAP_ID_MSI_X:
                                mptsas_log(mpt, CE_NOTE,
                                    "?mptsas3%d supports MSI-X.\n",
                                    mpt->m_instance);
                                mpt->m_options |= MPTSAS_OPT_MSI_X;
                                break;
                        /*
                         * The following capabilities are valid.  Any others
                         * will cause a message to be logged.
                         */
                        case PCI_CAP_ID_VPD:
                        case PCI_CAP_ID_PCIX:
                        case PCI_CAP_ID_PCI_E:
                                break;
                        default:
```

```
                                mptsas_log(mpt, CE_NOTE,
                                    "?mptsas3%d unrecognized capability "
                                    "0x%x.\n", mpt->m_instance, cap);
                                break;
                }

                /*
                 * Get next capabilities pointer and clear bits 0,1.
                 */
                caps_ptr = P2ALIGN(pci_config_get8(mpt->m_config_handle,
                    (caps_ptr + PCI_CAP_NEXT_PTR)), 4);
        }
        return (TRUE);
}

static int
mptsas_init_pm(mptsas_t *mpt)
{
        char            pmc_name[16];
        char            *pmc[] = {
                                NULL,
                                "0=Off (PCI D3 State)",
                                "3=On (PCI D0 State)",
                                NULL
                        };
        uint16_t        pmcsr_stat;

        /*
         * If PCI's capability does not support PM, then don't need
         * to registe the pm-components
         */
        if (!(mpt->m_options & MPTSAS_OPT_PM))
                return (DDI_SUCCESS);
        /*
         * If power management is supported by this chip, create
         * pm-components property for the power management framework
         */
        (void) sprintf(pmc_name, "NAME=mptsas3%d", mpt->m_instance);
        pmc[0] = pmc_name;
        if (ddi_prop_update_string_array(DDI_DEV_T_NONE, mpt->m_dip,
            "pm-components", pmc, 3) != DDI_PROP_SUCCESS) {
                mpt->m_options &= ~MPTSAS_OPT_PM;
                mptsas_log(mpt, CE_WARN,
                    "mptsas3%d: pm-component property creation failed.",
                    mpt->m_instance);
                return (DDI_FAILURE);
        }

        /*
         * Power on device.
         */
        (void) pm_busy_component(mpt->m_dip, 0);
        pmcsr_stat = pci_config_get16(mpt->m_config_handle,
            mpt->m_pmcsr_offset);
        if ((pmcsr_stat & PCI_PMCSR_STATE_MASK) != PCI_PMCSR_D0) {
                mptsas_log(mpt, CE_WARN, "mptsas3%d: Power up the device",
                    mpt->m_instance);
                pci_config_put16(mpt->m_config_handle, mpt->m_pmcsr_offset,
                    PCI_PMCSR_D0);
        }
        if (pm_power_has_changed(mpt->m_dip, 0, PM_LEVEL_D0) != DDI_SUCCESS) {
                mptsas_log(mpt, CE_WARN, "pm_power_has_changed failed");
                return (DDI_FAILURE);
        }
        mpt->m_power_level = PM_LEVEL_D0;
        /*
```



```
         * Set pm idle delay.
         */
        mpt->m_pm_idle_delay = ddi_prop_get_int(DDI_DEV_T_ANY,
            mpt->m_dip, 0, "mptsas-pm-idle-delay", MPTSAS_PM_IDLE_TIMEOUT);

        return (DDI_SUCCESS);
}

static int
mptsas_register_intrs(mptsas_t *mpt)
{
        dev_info_t *dip;
        int intr_types;

        dip = mpt->m_dip;

        /* Get supported interrupt types */
        if (ddi_intr_get_supported_types(dip, &intr_types) != DDI_SUCCESS) {
                mptsas_log(mpt, CE_WARN, "ddi_intr_get_supported_types "
                    "failed\n");
                return (FALSE);
        }

        NDBG6(("ddi_intr_get_supported_types() returned: 0x%x", intr_types));

        /*
         * Try MSIX first.
         */
        if (mptsas_enable_msix && (intr_types & DDI_INTR_TYPE_MSIX)) {
                if (mptsas_add_intrs(mpt, DDI_INTR_TYPE_MSIX) == DDI_SUCCESS) {
                        NDBG6(("Using MSI-X interrupt type"));
                        mpt->m_intr_type = DDI_INTR_TYPE_MSIX;
                        return (TRUE);
                }
        }

        /*
         * Try MSI, but fall back to FIXED
         */
        if (mptsas_enable_msi && (intr_types & DDI_INTR_TYPE_MSI)) {
                if (mptsas_add_intrs(mpt, DDI_INTR_TYPE_MSI) == DDI_SUCCESS) {
                        NDBG6(("Using MSI interrupt type"));
                        mpt->m_intr_type = DDI_INTR_TYPE_MSI;
                        return (TRUE);
                }
        }
        if (intr_types & DDI_INTR_TYPE_FIXED) {
                if (mptsas_add_intrs(mpt, DDI_INTR_TYPE_FIXED) == DDI_SUCCESS) {
                        NDBG6(("Using FIXED interrupt type"));
                        mpt->m_intr_type = DDI_INTR_TYPE_FIXED;
                        return (TRUE);
                } else {
                        NDBG6(("FIXED interrupt registration failed"));
                        return (FALSE);
                }
        }

        return (FALSE);
}

static void
mptsas_unregister_intrs(mptsas_t *mpt)
{
        mptsas_rem_intrs(mpt);
}
```

```
/*
 * mptsas_add_intrs:
 *
 * Register FIXED or MSI interrupts.
 */
static int
mptsas_add_intrs(mptsas_t *mpt, int intr_type)
{
        dev_info_t      *dip = mpt->m_dip;
        int             avail, actual, count = 0;
        int             i, flag, ret;

        NDBG6(("mptsas_add_intrs:interrupt type 0x%x", intr_type));

        /* Get number of interrupts */
        ret = ddi_intr_get_nintrs(dip, intr_type, &count);
        if ((ret != DDI_SUCCESS) || (count <= 0)) {
                mptsas_log(mpt, CE_WARN, "ddi_intr_get_nintrs() failed, "
                    "ret %d count %d\n", ret, count);

                return (DDI_FAILURE);
        }

        /* Get number of interrupts available to this device */
        ret = ddi_intr_get_navail(dip, intr_type, &avail);
        if ((ret != DDI_SUCCESS) || (avail == 0)) {
                mptsas_log(mpt, CE_WARN, "ddi_intr_get_navail() failed, "
                    "ret %d avail %d\n", ret, avail);

                return (DDI_FAILURE);
        }

        if (count < avail) {
                mptsas_log(mpt, CE_NOTE, "ddi_intr_get_nvail returned %d, "
                    "navail() returned %d", count, avail);
        }

        NDBG6(("mptsas_add_intrs:count %d, avail %d", count, avail));

        if (intr_type == DDI_INTR_TYPE_MSIX) {
                if (!mptsas3_max_msix_intrs) {
                        return (DDI_FAILURE);
                }

                /*
                 * Restrict the number of interrupts, firstly by
                 * the number returned from the IOCInfo, then by
                 * overall restriction.
                 */
                if (avail > mpt->m_max_msix_vectors) {
                        avail = mpt->m_max_msix_vectors?
                            mpt->m_max_msix_vectors:1;
                        NDBG6(("mptsas_add_intrs: mmmv avail %d", avail));
                }
                if (avail > mptsas3_max_msix_intrs) {
                        avail = mptsas3_max_msix_intrs;
                        NDBG6(("mptsas_add_intrs: m3mmi avail %d", avail));
                }
        }
        if (intr_type == DDI_INTR_TYPE_MSI) {
                NDBG6(("mptsas_add_intrs: MSI avail %d", avail));
                avail = 1;
        }

        /* Allocate an array of interrupt handles */
        mpt->m_intr_size = avail * sizeof (ddi_intr_handle_t);
```



```
        mpt->m_htable = kmem_alloc(mpt->m_intr_size, KM_SLEEP);

        flag = DDI_INTR_ALLOC_NORMAL;

        /* call ddi_intr_alloc() */
        ret = ddi_intr_alloc(dip, mpt->m_htable, intr_type, 0,
            avail, &actual, flag);

        if ((ret != DDI_SUCCESS) || (actual == 0)) {
                mptsas_log(mpt, CE_WARN, "ddi_intr_alloc() failed, ret %d\n",
                    ret);
                kmem_free(mpt->m_htable, mpt->m_intr_size);
                return (DDI_FAILURE);
        }

        NDBG6(("mptsas_add_intrs: actual %d, avail %d", actual, avail));
        /* use interrupt count returned or abort? */
        if (actual < avail) {
                mptsas_log(mpt, CE_NOTE,
                    "Interrupts requested: %d, received: %d\n",
                    avail, actual);
        }

        /*
         * Get priority for first msi, assume remaining are all the same
         */
        if ((ret = ddi_intr_get_pri(mpt->m_htable[0],
            &mpt->m_intr_pri)) != DDI_SUCCESS) {
                mptsas_log(mpt, CE_WARN, "ddi_intr_get_pri() failed %d\n", ret);

                /* Free already allocated intr */
                for (i = 0; i < actual; i++) {
                        (void) ddi_intr_free(mpt->m_htable[i]);
                }

                kmem_free(mpt->m_htable, mpt->m_intr_size);
                return (DDI_FAILURE);
        }

        /* Test for high level mutex */
        if (mpt->m_intr_pri >= ddi_intr_get_hilevel_pri()) {
                mptsas_log(mpt, CE_WARN, "mptsas_add_intrs: "
                    "Hi level interrupt not supported\n");

                /* Free already allocated intr */
                for (i = 0; i < actual; i++) {
                        (void) ddi_intr_free(mpt->m_htable[i]);
                }

                kmem_free(mpt->m_htable, mpt->m_intr_size);
                return (DDI_FAILURE);
        }

        /* Call ddi_intr_add_handler() */
        for (i = 0; i < actual; i++) {
                if ((ret = ddi_intr_add_handler(mpt->m_htable[i], mptsas_intr,
                    (caddr_t)mpt, (caddr_t)(uintptr_t)i)) != DDI_SUCCESS) {
                        mptsas_log(mpt, CE_WARN, "ddi_intr_add_handler() "
                            "failed %d\n", ret);

                        /* Free already allocated intr */
                        for (i = 0; i < actual; i++) {
                                (void) ddi_intr_free(mpt->m_htable[i]);
                        }

                        kmem_free(mpt->m_htable, mpt->m_intr_size);
```

```
                        return (DDI_FAILURE);
                }
        }

        if ((ret = ddi_intr_get_cap(mpt->m_htable[0], &mpt->m_intr_cap))
            != DDI_SUCCESS) {
                mptsas_log(mpt, CE_WARN, "ddi_intr_get_cap() failed %d\n", ret);

                /* Free already allocated intr */
                for (i = 0; i < actual; i++) {
                        (void) ddi_intr_free(mpt->m_htable[i]);
                }

                kmem_free(mpt->m_htable, mpt->m_intr_size);
                return (DDI_FAILURE);
        }

        mpt->m_intr_cnt = actual;

        /*
         * Enable interrupts
         */
        if (mpt->m_intr_cap & DDI_INTR_FLAG_BLOCK) {
                /* Call ddi_intr_block_enable() for MSI interrupts */
                (void) ddi_intr_block_enable(mpt->m_htable, mpt->m_intr_cnt);
        } else {
                /* Call ddi_intr_enable for MSI or FIXED interrupts */
                for (i = 0; i < mpt->m_intr_cnt; i++) {
                        (void) ddi_intr_enable(mpt->m_htable[i]);
                }
        }

        switch (intr_type) {
        case DDI_INTR_TYPE_MSIX:
                mptsas_log(mpt, CE_NOTE, "?Using %d MSI-X interrupt(s) "
                    "(Available sys %d, mpt %d, Requested %d)\n",
                    actual, count, mpt->m_max_msix_vectors, avail);
                break;
        case DDI_INTR_TYPE_MSI:
                mptsas_log(mpt, CE_NOTE, "Using single MSI interrupt\n");
                break;
        case DDI_INTR_TYPE_FIXED:
        default:
                mptsas_log(mpt, CE_NOTE, "Using single fixed interrupt\n");
                break;
        }

        return (DDI_SUCCESS);
}

/*
 * mptsas_rem_intrs:
 *
 * Unregister FIXED or MSI interrupts
 */
static void
mptsas_rem_intrs(mptsas_t *mpt)
{
        int     i;

        NDBG6(("mptsas_rem_intrs"));

        /* Disable all interrupts */
        if (mpt->m_intr_cap & DDI_INTR_FLAG_BLOCK) {
                /* Call ddi_intr_block_disable() */
                (void) ddi_intr_block_disable(mpt->m_htable, mpt->m_intr_cnt);
```



```
        } else {
                for (i = 0; i < mpt->m_intr_cnt; i++) {
                        (void) ddi_intr_disable(mpt->m_htable[i]);
                }
        }

        /* Call ddi_intr_remove_handler() */
        for (i = 0; i < mpt->m_intr_cnt; i++) {
                (void) ddi_intr_remove_handler(mpt->m_htable[i]);
                (void) ddi_intr_free(mpt->m_htable[i]);
        }
        kmem_free(mpt->m_htable, mpt->m_intr_size);
        mpt->m_intr_cnt = 0;
}

/*
 * The IO fault service error handling callback function
 */
/*ARGSUSED*/
static int
mptsas_fm_error_cb(dev_info_t *dip, ddi_fm_error_t *err, const void *impl_data)
{
        /*
         * as the driver can always deal with an error in any dma or
         * access handle, we can just return the fme_status value.
         */
        pci_ereport_post(dip, err, NULL);
        return (err->fme_status);
}

/*
 * mptsas_fm_init - initialize fma capabilities and register with IO
 *                  fault services.
 */
static void
mptsas_fm_init(mptsas_t *mpt)
{
        /*
         * Need to change iblock to priority for new MSI intr
         */
        ddi_iblock_cookie_t     fm_ibc;

        /* Only register with IO Fault Services if we have some capability */
        if (mpt->m_fm_capabilities) {
                /* Adjust access and dma attributes for FMA */
                mpt->m_reg_acc_attr.devacc_attr_access = DDI_FLAGERR_ACC;
                mpt->m_msg_dma_attr.dma_attr_flags |= DDI_DMA_FLAGERR;
                mpt->m_io_dma_attr.dma_attr_flags |= DDI_DMA_FLAGERR;

                /*
                 * Register capabilities with IO Fault Services.
                 * mpt->m_fm_capabilities will be updated to indicate
                 * capabilities actually supported (not requested.)
                 */
                ddi_fm_init(mpt->m_dip, &mpt->m_fm_capabilities, &fm_ibc);

                /*
                 * Initialize pci ereport capabilities if ereport
                 * capable (should always be.)
                 */
                if (DDI_FM_EREPORT_CAP(mpt->m_fm_capabilities) ||
                    DDI_FM_ERRCB_CAP(mpt->m_fm_capabilities)) {
                        pci_ereport_setup(mpt->m_dip);
                }

                /*
```

```
                 * Register error callback if error callback capable.
                 */
                if (DDI_FM_ERRCB_CAP(mpt->m_fm_capabilities)) {
                        ddi_fm_handler_register(mpt->m_dip,
                            mptsas_fm_error_cb, (void *) mpt);
                }
        }
}

/*
 * mptsas_fm_fini - Releases fma capabilities and un-registers with IO
 *                 fault services.
 *
 */
static void
mptsas_fm_fini(mptsas_t *mpt)
{
        /* Only unregister FMA capabilities if registered */
        if (mpt->m_fm_capabilities) {

                /*
                 * Un-register error callback if error callback capable.
                 */

                if (DDI_FM_ERRCB_CAP(mpt->m_fm_capabilities)) {
                        ddi_fm_handler_unregister(mpt->m_dip);
                }

                /*
                 * Release any resources allocated by pci_ereport_setup()
                 */

                if (DDI_FM_EREPORT_CAP(mpt->m_fm_capabilities) ||
                    DDI_FM_ERRCB_CAP(mpt->m_fm_capabilities)) {
                        pci_ereport_teardown(mpt->m_dip);
                }

                /* Unregister from IO Fault Services */
                ddi_fm_fini(mpt->m_dip);

                /* Adjust access and dma attributes for FMA */
                mpt->m_reg_acc_attr.devacc_attr_access = DDI_DEFAULT_ACC;
                mpt->m_msg_dma_attr.dma_attr_flags &= ~DDI_DMA_FLAGERR;
                mpt->m_io_dma_attr.dma_attr_flags &= ~DDI_DMA_FLAGERR;

        }
}

int
mptsas_check_acc_handle(ddi_acc_handle_t handle)
{
        ddi_fm_error_t  de;

        if (handle == NULL)
                return (DDI_FAILURE);
        ddi_fm_acc_err_get(handle, &de, DDI_FME_VER0);
        return (de.fme_status);
}

int
mptsas_check_dma_handle(ddi_dma_handle_t handle)
{
        ddi_fm_error_t  de;

        if (handle == NULL)
                return (DDI_FAILURE);
```



```
        ddi_fm_dma_err_get(handle, &de, DDI_FME_VER0);
        return (de.fme_status);
}

void
mptsas_fm_ereport(mptsas_t *mpt, char *detail)
{
        uint64_t        ena;
        char            buf[FM_MAX_CLASS];

        (void) snprintf(buf, FM_MAX_CLASS, "%s.%s", DDI_FM_DEVICE, detail);
        ena = fm_ena_generate(0, FM_ENA_FMT1);
        if (DDI_FM_EREPORT_CAP(mpt->m_fm_capabilities)) {
                ddi_fm_ereport_post(mpt->m_dip, buf, ena, DDI_NOSLEEP,
                    FM_VERSION, DATA_TYPE_UINT8, FM_EREPORT_VERS0, NULL);
        }
}

static int
mptsas_get_target_device_info(mptsas_t *mpt, uint32_t page_address,
    uint16_t *dev_handle, mptsas_target_t **pptgt)
{
        int             rval;
        uint32_t        dev_info;
        uint64_t        sas_wwn;
        mptsas_phymask_t phymask;
        uint8_t         physport, phynum, config, disk;
        uint64_t        devicename;
        uint16_t        pdev_hdl;
        mptsas_target_t *tmp_tgt = NULL;
        uint16_t        bay_num, enclosure, io_flags;

        ASSERT(*pptgt == NULL);

        rval = mptsas_get_sas_device_page0(mpt, page_address, dev_handle,
            &sas_wwn, &dev_info, &physport, &phynum, &pdev_hdl,
            &bay_num, &enclosure, &io_flags);
        if (rval != DDI_SUCCESS) {
                rval = DEV_INFO_FAIL_PAGE0;
                return (rval);
        }

        if ((dev_info & (MPI2_SAS_DEVICE_INFO_SSP_TARGET |
            MPI2_SAS_DEVICE_INFO_SATA_DEVICE |
            MPI2_SAS_DEVICE_INFO_ATAPI_DEVICE)) == NULL) {
                rval = DEV_INFO_WRONG_DEVICE_TYPE;
                return (rval);
        }

        /*
         * Check if the dev handle is for a Phys Disk. If so, set return value
         * and exit.  Don’t add Phys Disks to hash.
         */
        for (config = 0; config < mpt->m_num_raid_configs; config++) {
                for (disk = 0; disk < MPTSAS_MAX_DISKS_IN_CONFIG; disk++) {
                        if (*dev_handle == mpt->m_raidconfig[config].
                            m_physdisk_devhdl[disk]) {
                                rval = DEV_INFO_PHYS_DISK;
                                return (rval);
                        }
                }
        }

        /*
         * Get SATA Device Name from SAS device page0 for
         * sata device, if device name doesn’t exist, set mta_wwn to
```

```
 * 0 for direct attached SATA. For the device behind the expander
 * we still can use STP address assigned by expander.
 */
        if (dev_info & (MPI2_SAS_DEVICE_INFO_SATA_DEVICE |
            MPI2_SAS_DEVICE_INFO_ATAPI_DEVICE)) {
                mutex_exit(&mpt->m_mutex);
                /* alloc a tmp_tgt to send the cmd */
                tmp_tgt = kmem_zalloc(sizeof (struct mptsas_target),
                    KM_SLEEP);
                tmp_tgt->m_devhdl = *dev_handle;
                tmp_tgt->m_deviceinfo = dev_info;
                tmp_tgt->m_qfull_retries = QFULL_RETRIES;
                tmp_tgt->m_qfull_retry_interval =
                    drv_usectohz(QFULL_RETRY_INTERVAL * 1000);
                tmp_tgt->m_t_throttle = MAX_THROTTLE;
                mutex_init(&tmp_tgt->m_t_mutex, NULL, MUTEX_DRIVER, NULL);
                devicename = mptsas_get_sata_guid(mpt, tmp_tgt, 0);
                mutex_destroy(&tmp_tgt->m_t_mutex);
                kmem_free(tmp_tgt, sizeof (struct mptsas_target));
                mutex_enter(&mpt->m_mutex);
                if (devicename != 0 && (((devicename >> 56) & 0xf0) == 0x50)) {
                        sas_wwn = devicename;
                } else if (dev_info & MPI2_SAS_DEVICE_INFO_DIRECT_ATTACH) {
                        sas_wwn = 0;
                }
        }

        phymask = mptsas_physport_to_phymask(mpt, physport);
        *pptgt = mptsas_tgt_alloc(mpt, *dev_handle, sas_wwn,
            dev_info, phymask, phynum);
        if (*pptgt == NULL) {
                mptsas_log(mpt, CE_WARN, "Failed to allocated target"
                    "structure!");
                rval = DEV_INFO_FAIL_ALLOC;
                return (rval);
        }
        (*pptgt)->m_io_flags = io_flags;
        (*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;
        tran_clone->tran_tgt_private = tgt_private;
        ap->a_hba_tran = tran_clone;

        pktp = scsi_init_pkt(ap, (struct scsi_pkt *)NULL,
            data_bp, cdblen, sizeof (struct scsi_arq_status),
            0, PKT_CONSISTENT, NULL, NULL);
        if (pktp == NULL) {
                goto out;
        }
        bcopy(cdb, pktp->pkt_cdbp, cdblen);
        pktp->pkt_flags = FLAG_NOPARITY;
        pktp->pkt_time = mptsas_scsi_pkt_time;
        if (scsi_poll(pktp) < 0) {
                goto out;
        }
        if (((struct scsi_status *)pktp->pkt_scbp)->sts_chk) {
                goto out;
        }
        if (resid != NULL) {
                *resid = pktp->pkt_resid;
        }

        ret = DDI_SUCCESS;
out:
        if (pktp) {
                scsi_destroy_pkt(pktp);
        }
        if (tran_clone) {
                kmem_free(tran_clone, sizeof (scsi_hba_tran_t));
        }
        if (tgt_private) {
                kmem_free(tgt_private, sizeof (mptsas_tgt_private_t));
        }
        return (ret);
}
```



```
static int
mptsas_parse_address(char *name, uint64_t *wwid, uint8_t *phy, int *lun)
{
        char    *cp = NULL;
        char    *ptr = NULL;
        size_t  s = 0;
        char    *wwid_str = NULL;
        char    *lun_str = NULL;
        long    lunnum;
        long    phyid = -1;
        int     rc = DDI_FAILURE;

        ptr = name;
        ASSERT(ptr[0] == 'w' || ptr[0] == 'p');
        ptr++;
        if ((cp = strchr(ptr, ',')) == NULL) {
                return (DDI_FAILURE);
        }

        wwid_str = kmem_zalloc(SCSI_MAXNAMELEN, KM_SLEEP);
        s = (uintptr_t)cp - (uintptr_t)ptr;

        bcopy(ptr, wwid_str, s);
        wwid_str[s] = '\0';

        ptr = ++cp;

        if ((cp = strchr(ptr, '\0')) == NULL) {
                goto out;
        }
        lun_str =  kmem_zalloc(SCSI_MAXNAMELEN, KM_SLEEP);
        s = (uintptr_t)cp - (uintptr_t)ptr;

        bcopy(ptr, lun_str, s);
        lun_str[s] = '\0';

        if (name[0] == 'p') {
                rc = ddi_strtol(wwid_str, NULL, 0x10, &phyid);
        } else {
                rc = scsi_wwnstr_to_wwn(wwid_str, wwid);
        }
        if (rc != DDI_SUCCESS)
                goto out;

        if (phyid != -1) {
                ASSERT(phyid < MPTSAS_MAX_PHYS);
                *phy = (uint8_t)phyid;
        }
        rc = ddi_strtol(lun_str, NULL, 0x10, &lunnum);
        if (rc != 0)
                goto out;

        *lun = (int)lunnum;
        rc = DDI_SUCCESS;
out:
        if (wwid_str)
                kmem_free(wwid_str, SCSI_MAXNAMELEN);
        if (lun_str)
                kmem_free(lun_str, SCSI_MAXNAMELEN);

        return (rc);
}

/*
 * mptsas_parse_smp_name() is to parse sas wwn string
 * which format is "wWWN"
```

```
 */
static int
mptsas_parse_smp_name(char *name, uint64_t *wwn)
{
        char    *ptr = name;

        if (*ptr != 'w') {
                return (DDI_FAILURE);
        }

        ptr++;
        if (scsi_wwnstr_to_wwn(ptr, wwn)) {
                return (DDI_FAILURE);
        }
        return (DDI_SUCCESS);
}

static int
mptsas_bus_config(dev_info_t *pdip, uint_t flag,
    ddi_bus_config_op_t op, void *arg, dev_info_t **childp)
{
        int             ret = NDI_FAILURE;
        int             circ = 0;
        int             circ1 = 0;
        mptsas_t        *mpt;
        char            *ptr = NULL;
        char            *devnm = NULL;
        uint64_t        wwid = 0;
        uint8_t         phy = 0xFF;
        int             lun = 0;
        uint_t          mflags = flag;
        int             bconfig = TRUE;

        if (scsi_hba_iport_unit_address(pdip) == 0) {
                return (DDI_FAILURE);
        }

        mpt = DIP2MPT(pdip);
        if (!mpt) {
                return (DDI_FAILURE);
        }
        /*
         * Hold the nexus across the bus_config
         */
        ndi_devi_enter(scsi_vhci_dip, &circ);
        ndi_devi_enter(pdip, &circ1);
        switch (op) {
        case BUS_CONFIG_ONE:
                /* parse wwid/target name out of name given */
                if ((ptr = strchr((char *)arg, '@')) == NULL) {
                        ret = NDI_FAILURE;
                        break;
                }
                ptr++;
                if (strncmp((char *)arg, "smp", 3) == 0) {
                        /*
                         * This is a SMP target device
                         */
                        ret = mptsas_parse_smp_name(ptr, &wwid);
                        if (ret != DDI_SUCCESS) {
                                ret = NDI_FAILURE;
                                break;
                        }
                        ret = mptsas_config_smp(pdip, wwid, childp);
                } else if ((ptr[0] == 'w') || (ptr[0] == 'p')) {
                        /*
```



```
                         * OBP could pass down a non-canonical form
                         * bootpath without LUN part when LUN is 0.
                         * So driver need adjust the string.
                         */
                        if (strchr(ptr, ',') == NULL) {
                                devnm = kmem_zalloc(SCSI_MAXNAMELEN, KM_SLEEP);
                                (void) sprintf(devnm, "%s,0", (char *)arg);
                                ptr = strchr(devnm, '@');
                                ptr++;
                        }

                        /*
                         * The device path is wWWID format and the device
                         * is not SMP target device.
                         */
                        ret = mptsas_parse_address(ptr, &wwid, &phy, &lun);
                        if (ret != DDI_SUCCESS) {
                                ret = NDI_FAILURE;
                                break;
                        }
                        *childp = NULL;
                        if (ptr[0] == 'w') {
                                ret = mptsas_config_one_addr(pdip, wwid,
                                    lun, childp);
                        } else if (ptr[0] == 'p') {
                                ret = mptsas_config_one_phy(pdip, phy, lun,
                                    childp);
                        }

                        /*
                         * If this is CD/DVD device in OBP path, the
                         * ndi_busop_bus_config can be skipped as config one
                         * operation is done above.
                         */
                        if ((ret == NDI_SUCCESS) && (*childp != NULL) &&
                            (strcmp(ddi_node_name(*childp), "cdrom") == 0) &&
                            (strncmp((char *)arg, "disk", 4) == 0)) {
                                bconfig = FALSE;
                                ndi_hold_devi(*childp);
                        }
                } else {
                        ret = NDI_FAILURE;
                        break;
                }

                /*
                 * DDI group instructed us to use this flag.
                 */
                mflags |= NDI_MDI_FALLBACK;
                break;
        case BUS_CONFIG_DRIVER:
        case BUS_CONFIG_ALL:
                mptsas_config_all(pdip);
                ret = NDI_SUCCESS;
                break;
        }

        if ((ret == NDI_SUCCESS) && bconfig) {
                ret = ndi_busop_bus_config(pdip, mflags, op,
                    (devnm == NULL) ? arg : devnm, childp, 0);
        }

        ndi_devi_exit(pdip, circ1);
        ndi_devi_exit(scsi_vhci_dip, circ);
        if (devnm != NULL)
                kmem_free(devnm, SCSI_MAXNAMELEN);
```

```
        return (ret);
}

static int
mptsas_probe_lun(dev_info_t *pdip, int lun, dev_info_t **dip,
    mptsas_target_t *ptgt)
{
        int                     rval = DDI_FAILURE;
        struct scsi_inquiry     *sd_inq = NULL;
        mptsas_t                *mpt = DIP2MPT(pdip);

        sd_inq = (struct scsi_inquiry *)kmem_alloc(SUN_INQSIZE, KM_SLEEP);

        rval = mptsas_inquiry(mpt, ptgt, lun, 0, (uchar_t *)sd_inq,
            SUN_INQSIZE, 0, (uchar_t)0);

        if ((rval == DDI_SUCCESS) && MPTSAS_VALID_LUN(sd_inq)) {
                rval = mptsas_create_lun(pdip, sd_inq, dip, ptgt, lun);
        } else {
                rval = DDI_FAILURE;
        }

        kmem_free(sd_inq, SUN_INQSIZE);
        return (rval);
}

static int
mptsas_config_one_addr(dev_info_t *pdip, uint64_t sasaddr, int lun,
    dev_info_t **lundip)
{
        int             rval;
        mptsas_t                *mpt = DIP2MPT(pdip);
        int             phymask;
        mptsas_target_t *ptgt = NULL;

        /*
         * Get the physical port associated to the iport
         */
        phymask = ddi_prop_get_int(DDI_DEV_T_ANY, pdip, 0,
            "phymask", 0);

        ptgt = mptsas_wwid_to_ptgt(mpt, phymask, sasaddr);
        if (ptgt == NULL) {
                /*
                 * didn't match any device by searching
                 */
                return (DDI_FAILURE);
        }
        /*
         * If the LUN already exists and the status is online,
         * we just return the pointer to dev_info_t directly.
         * For the mdi_pathinfo node, we'll handle it in
         * mptsas_create_virt_lun()
         * TODO should be also in mptsas_handle_dr
         */

        *lundip = mptsas_find_child_addr(pdip, sasaddr, lun);
        if (*lundip != NULL) {
                /*
                 * TODO Another senario is, we hotplug the same disk
                 * on the same slot, the devhdl changed, is this
                 * possible?
                 * tgt_private->t_private != ptgt
                 */
                if (sasaddr != ptgt->m_addr.mta_wwn) {
                        /*
```



```
                         * The device has changed although the devhdl is the
                         * same (Enclosure mapping mode, change drive on the
                         * same slot)
                         */
                        return (DDI_FAILURE);
                }
                return (DDI_SUCCESS);
        }

        if (phymask == 0) {
                /*
                 * Configure IR volume
                 */
                rval =  mptsas_config_raid(pdip, ptgt->m_devhdl, lundip);
                return (rval);
        }
        rval = mptsas_probe_lun(pdip, lun, lundip, ptgt);

        return (rval);
}

static int
mptsas_config_one_phy(dev_info_t *pdip, uint8_t phy, int lun,
    dev_info_t **lundip)
{
        int             rval;
        mptsas_t        *mpt = DIP2MPT(pdip);
        mptsas_phymask_t phymask;
        mptsas_target_t *ptgt = NULL;

        /*
         * Get the physical port associated to the iport
         */
        phymask = (mptsas_phymask_t)ddi_prop_get_int(DDI_DEV_T_ANY, pdip, 0,
            "phymask", 0);

        ptgt = mptsas_phy_to_tgt(mpt, phymask, phy);
        if (ptgt == NULL) {
                /*
                 * didn't match any device by searching
                 */
                return (DDI_FAILURE);
        }

        /*
         * If the LUN already exists and the status is online,
         * we just return the pointer to dev_info_t directly.
         * For the mdi_pathinfo node, we'll handle it in
         * mptsas_create_virt_lun().
         */

        *lundip = mptsas_find_child_phy(pdip, phy);
        if (*lundip != NULL) {
                return (DDI_SUCCESS);
        }

        rval = mptsas_probe_lun(pdip, lun, lundip, ptgt);

        return (rval);
}

static int
mptsas_retrieve_lundata(int lun_cnt, uint8_t *buf, uint16_t *lun_num,
    uint8_t *lun_addr_type)
{
        uint32_t        lun_idx = 0;
```

```
        ASSERT(lun_num != NULL);
        ASSERT(lun_addr_type != NULL);

        lun_idx = (lun_cnt + 1) * MPTSAS_SCSI_REPORTLUNS_ADDRESS_SIZE;
        /* determine report luns addressing type */
        switch (buf[lun_idx] & MPTSAS_SCSI_REPORTLUNS_ADDRESS_MASK) {
                /*
                 * Vendors in the field have been found to be concatenating
                 * bus/target/lun to equal the complete lun value instead
                 * of switching to flat space addressing
                 */
                /* 00b - peripheral device addressing method */
        case MPTSAS_SCSI_REPORTLUNS_ADDRESS_PERIPHERAL:
                /* FALLTHRU */
                /* 10b - logical unit addressing method */
        case MPTSAS_SCSI_REPORTLUNS_ADDRESS_LOGICAL_UNIT:
                /* FALLTHRU */
                /* 01b - flat space addressing method */
        case MPTSAS_SCSI_REPORTLUNS_ADDRESS_FLAT_SPACE:
                /* byte0 bit0-5=msb lun byte1 bit0-7=lsb lun */
                *lun_addr_type = (buf[lun_idx] &
                    MPTSAS_SCSI_REPORTLUNS_ADDRESS_MASK) >> 6;
                *lun_num = (buf[lun_idx] & 0x3F) << 8;
                *lun_num |= buf[lun_idx + 1];
                return (DDI_SUCCESS);
        default:
                return (DDI_FAILURE);
        }
}

static int
mptsas_config_luns(dev_info_t *pdip, mptsas_target_t *ptgt)
{
        struct buf              *repluns_bp = NULL;
        struct scsi_address     ap;
        uchar_t                 cdb[CDB_GROUP5];
        int                     ret = DDI_FAILURE;
        int                     retry = 0;
        int                     lun_list_len = 0;
        uint16_t                lun_num = 0;
        uint8_t                 lun_addr_type = 0;
        uint32_t                lun_cnt = 0;
        uint32_t                lun_total = 0;
        dev_info_t              *cdip = NULL;
        uint16_t                *saved_repluns = NULL;
        char                    *buffer = NULL;
        int                     buf_len = 128;
        mptsas_t                *mpt = DIP2MPT(pdip);
        uint64_t                sas_wwn = 0;
        uint8_t                 phy = 0xFF;
        uint32_t                dev_info = 0;

        mutex_enter(&mpt->m_mutex);
        sas_wwn = ptgt->m_addr.mta_wwn;
        phy = ptgt->m_phynum;
        dev_info = ptgt->m_deviceinfo;
        mutex_exit(&mpt->m_mutex);

        if (sas_wwn == 0) {
                /*
                 * It’s a SATA without Device Name
                 * So don’t try multi-LUNs
                 */
                if (mptsas_find_child_phy(pdip, phy)) {
                        return (DDI_SUCCESS);
```



```
                } else {
                        /*
                         * need configure and create node
                         */
                        return (DDI_FAILURE);
                }
        }

        /*
         * WWN (SAS address or Device Name exist)
         */
        if (dev_info & (MPI2_SAS_DEVICE_INFO_SATA_DEVICE |
            MPI2_SAS_DEVICE_INFO_ATAPI_DEVICE)) {
                /*
                 * SATA device with Device Name
                 * So don’t try multi-LUNs
                 */
                if (mptsas_find_child_addr(pdip, sas_wwn, 0)) {
                        return (DDI_SUCCESS);
                } else {
                        return (DDI_FAILURE);
                }
        }

        do {
                ap.a_target = MPTSAS_INVALID_DEVHDL;
                ap.a_lun = 0;
                ap.a_hba_tran = mpt->m_tran;
                repluns_bp = scsi_alloc_consistent_buf(&ap,
                    (struct buf *)NULL, buf_len, B_READ, NULL_FUNC, NULL);
                if (repluns_bp == NULL) {
                        retry++;
                        continue;
                }
                bzero(cdb, CDB_GROUP5);
                cdb[0] = SCMD_REPORT_LUNS;
                cdb[6] = (buf_len & 0xff000000) >> 24;
                cdb[7] = (buf_len & 0x00ff0000) >> 16;
                cdb[8] = (buf_len & 0x0000ff00) >> 8;
                cdb[9] = (buf_len & 0x000000ff);

                ret = mptsas_send_scsi_cmd(mpt, &ap, ptgt, &cdb[0], CDB_GROUP5,
                    repluns_bp, NULL);
                if (ret != DDI_SUCCESS) {
                        scsi_free_consistent_buf(repluns_bp);
                        retry++;
                        continue;
                }
                lun_list_len = BE_32(*(int *)((void *)(
                    repluns_bp->b_un.b_addr)));
                if (buf_len >= lun_list_len + 8) {
                        ret = DDI_SUCCESS;
                        break;
                }
                scsi_free_consistent_buf(repluns_bp);
                buf_len = lun_list_len + 8;

        } while (retry < 3);

        if (ret != DDI_SUCCESS)
                return (ret);
        buffer = (char *)repluns_bp->b_un.b_addr;
        /*
         * find out the number of luns returned by the SCSI ReportLun call
         * and allocate buffer space
         */
```

```
        lun_total = lun_list_len / MPTSAS_SCSI_REPORTLUNS_ADDRESS_SIZE;
        saved_repluns = kmem_zalloc(sizeof (uint16_t) * lun_total, KM_SLEEP);
        if (saved_repluns == NULL) {
                scsi_free_consistent_buf(repluns_bp);
                return (DDI_FAILURE);
        }
        for (lun_cnt = 0; lun_cnt < lun_total; lun_cnt++) {
                if (mptsas_retrieve_lundata(lun_cnt, (uint8_t *)(buffer),
                    &lun_num, &lun_addr_type) != DDI_SUCCESS) {
                        continue;
                }
                saved_repluns[lun_cnt] = lun_num;
                if (cdip = mptsas_find_child_addr(pdip, sas_wwn, lun_num))
                        ret = DDI_SUCCESS;
                else
                        ret = mptsas_probe_lun(pdip, lun_num, &cdip,
                            ptgt);
                if ((ret == DDI_SUCCESS) && (cdip != NULL)) {
                        (void) ndi_prop_remove(DDI_DEV_T_NONE, cdip,
                            MPTSAS_DEV_GONE);
                }
        }
        mptsas_offline_missed_luns(pdip, saved_repluns, lun_total, ptgt);
        kmem_free(saved_repluns, sizeof (uint16_t) * lun_total);
        scsi_free_consistent_buf(repluns_bp);
        return (DDI_SUCCESS);
}

static int
mptsas_config_raid(dev_info_t *pdip, uint16_t target, dev_info_t **dip)
{
        int                     rval = DDI_FAILURE;
        struct scsi_inquiry     *sd_inq = NULL;
        mptsas_t                *mpt = DIP2MPT(pdip);
        mptsas_target_t         *ptgt = NULL;

        mutex_enter(&mpt->m_mutex);
        ptgt = refhash_linear_search(mpt->m_targets,
            mptsas_target_eval_devhdl, &target);
        mutex_exit(&mpt->m_mutex);
        if (ptgt == NULL) {
                mptsas_log(mpt, CE_WARN, "Volume with VolDevHandle of 0x%x "
                    "not found.", target);
                return (rval);
        }

        sd_inq = (struct scsi_inquiry *)kmem_alloc(SUN_INQSIZE, KM_SLEEP);
        rval = mptsas_inquiry(mpt, ptgt, 0, 0, (uchar_t *)sd_inq,
            SUN_INQSIZE, 0, (uchar_t)0);

        if ((rval == DDI_SUCCESS) && MPTSAS_VALID_LUN(sd_inq)) {
                rval = mptsas_create_phys_lun(pdip, sd_inq, NULL, dip, ptgt,
                    0);
        } else {
                rval = DDI_FAILURE;
        }

        kmem_free(sd_inq, SUN_INQSIZE);
        return (rval);
}

/*
 * configure all RAID volumes for virtual iport
 */
static void
mptsas_config_all_viport(dev_info_t *pdip)
```



```
{
        mptsas_t        *mpt = DIP2MPT(pdip);
        int             config, vol;
        int             target;
        dev_info_t      *lundip = NULL;

        /*
         * Get latest RAID info and search for any Volume DevHandles.  If any
         * are found, configure the volume.
         */
        mutex_enter(&mpt->m_mutex);
        for (config = 0; config < mpt->m_num_raid_configs; config++) {
                for (vol = 0; vol < MPTSAS_MAX_RAIDVOLS; vol++) {
                        if (mpt->m_raidconfig[config].m_raidvol[vol].m_israid
                            == 1) {
                                target = mpt->m_raidconfig[config].
                                    m_raidvol[vol].m_raidhandle;
                                mutex_exit(&mpt->m_mutex);
                                (void) mptsas_config_raid(pdip, target,
                                    &lundip);
                                mutex_enter(&mpt->m_mutex);
                        }
                }
        }
        mutex_exit(&mpt->m_mutex);
}

static void
mptsas_offline_missed_luns(dev_info_t *pdip, uint16_t *repluns,
    int lun_cnt, mptsas_target_t *ptgt)
{
        dev_info_t      *child = NULL, *savechild = NULL;
        mdi_pathinfo_t  *pip = NULL, *savepip = NULL;
        uint64_t        sas_wwn, wwid;
        uint8_t         phy;
        int             lun;
        int             i;
        int             find;
        char            *addr;
        char            *nodename;
        mptsas_t        *mpt = DIP2MPT(pdip);

        mutex_enter(&mpt->m_mutex);
        wwid = ptgt->m_addr.mta_wwn;
        mutex_exit(&mpt->m_mutex);

        child = ddi_get_child(pdip);
        while (child) {
                find = 0;
                savechild = child;
                child = ddi_get_next_sibling(child);

                nodename = ddi_node_name(savechild);
                if (strcmp(nodename, "smp") == 0) {
                        continue;
                }

                addr = ddi_get_name_addr(savechild);
                if (addr == NULL) {
                        continue;
                }

                if (mptsas_parse_address(addr, &sas_wwn, &phy, &lun) !=
                    DDI_SUCCESS) {
                        continue;
                }
```

```
                if (wwid == sas_wwn) {
                        for (i = 0; i < lun_cnt; i++) {
                                if (repluns[i] == lun) {
                                        find = 1;
                                        break;
                                }
                        }
                } else {
                        continue;
                }
                if (find == 0) {
                        /*
                         * The lun has not been there already
                         */
                        (void) mptsas_offline_lun(pdip, savechild, NULL,
                            NDI_DEVI_REMOVE);
                }
        }

        pip = mdi_get_next_client_path(pdip, NULL);
        while (pip) {
                find = 0;
                savepip = pip;
                addr = MDI_PI(pip)->pi_addr;

                pip = mdi_get_next_client_path(pdip, pip);

                if (addr == NULL) {
                        continue;
                }

                if (mptsas_parse_address(addr, &sas_wwn, &phy,
                    &lun) != DDI_SUCCESS) {
                        continue;
                }

                if (sas_wwn == wwid) {
                        for (i = 0; i < lun_cnt; i++) {
                                if (repluns[i] == lun) {
                                        find = 1;
                                        break;
                                }
                        }
                } else {
                        continue;
                }

                if (find == 0) {
                        /*
                         * The lun has not been there already
                         */
                        (void) mptsas_offline_lun(pdip, NULL, savepip,
                            NDI_DEVI_REMOVE);
                }
        }
}

void
mptsas_update_hashtab(struct mptsas *mpt)
{
        uint32_t        page_address;
        int             rval = 0;
        uint16_t        dev_handle;
        mptsas_target_t *ptgt = NULL;
        mptsas_smp_t    smp_node;
```



```
        /*
         * Get latest RAID info.
         */
        (void) mptsas_get_raid_info(mpt);

        dev_handle = mpt->m_smp_devhdl;
        for (; mpt->m_done_traverse_smp == 0; ) {
                page_address = (MPI2_SAS_EXPAND_PGAD_FORM_GET_NEXT_HNDL &
                    MPI2_SAS_EXPAND_PGAD_FORM_MASK) | (uint32_t)dev_handle;
                if (mptsas_get_sas_expander_page0(mpt, page_address, &smp_node)
                    != DDI_SUCCESS) {
                        break;
                }
                mpt->m_smp_devhdl = dev_handle = smp_node.m_devhdl;
                (void) mptsas_smp_alloc(mpt, &smp_node);
        }

        /*
         * Config target devices
         */
        dev_handle = mpt->m_dev_handle;

        /*
         * Do loop to get sas device page 0 by GetNextHandle till the
         * the last handle. If the sas device is a SATA/SSP target,
         * we try to config it.
         */
        for (; mpt->m_done_traverse_dev == 0; ) {
                ptgt = NULL;
                page_address =
                    (MPI2_SAS_DEVICE_PGAD_FORM_GET_NEXT_HANDLE &
                    MPI2_SAS_DEVICE_PGAD_FORM_MASK) |
                    (uint32_t)dev_handle;
                rval = mptsas_get_target_device_info(mpt, page_address,
                    &dev_handle, &ptgt);
                if ((rval == DEV_INFO_FAIL_PAGE0) ||
                    (rval == DEV_INFO_FAIL_ALLOC)) {
                        break;
                }

                mpt->m_dev_handle = dev_handle;
        }

}

void
mptsas_update_driver_data(struct mptsas *mpt)
{
        mptsas_target_t *tp;
        mptsas_smp_t *sp;

        ASSERT(MUTEX_HELD(&mpt->m_mutex));

        /*
         * TODO after hard reset, update the driver data structures
         * 1. update port/phymask mapping table mpt->m_phy_info
         * 2. invalid all the entries in hash table
         *    m_devhdl = 0xffff and m_deviceinfo = 0
         * 3. call sas_device_page/expander_page to update hash table
         */
        mptsas_update_phymask(mpt);
        /*
         * Invalid the existing entries
         *
         * XXX - It seems like we should just delete everything here.  We are
```

```
         * holding the lock and are about to refresh all the targets in both
         * hashes anyway.  Given the path we're in, what outstanding async
         * event could possibly be trying to reference one of these things
         * without taking the lock, and how would that be useful anyway?
         */
        for (tp = refhash_first(mpt->m_targets); tp != NULL;
            tp = refhash_next(mpt->m_targets, tp)) {
                tp->m_devhdl = MPTSAS_INVALID_DEVHDL;
                tp->m_deviceinfo = 0;
                tp->m_dr_flag = MPTSAS_DR_INACTIVE;
        }
        for (sp = refhash_first(mpt->m_smp_targets); sp != NULL;
            sp = refhash_next(mpt->m_smp_targets, sp)) {
                sp->m_devhdl = MPTSAS_INVALID_DEVHDL;
                sp->m_deviceinfo = 0;
        }
        mpt->m_done_traverse_dev = 0;
        mpt->m_done_traverse_smp = 0;
        mpt->m_dev_handle = mpt->m_smp_devhdl = MPTSAS_INVALID_DEVHDL;
        mptsas_update_hashtab(mpt);
}

static void
mptsas_config_all(dev_info_t *pdip)
{
        dev_info_t      *smpdip = NULL;
        mptsas_t        *mpt = DIP2MPT(pdip);
        int             phymask = 0;
        mptsas_phymask_t phy_mask;
        mptsas_target_t *ptgt = NULL;
        mptsas_smp_t    *psmp;

        /*
         * Get the phymask associated to the iport
         */
        phymask = ddi_prop_get_int(DDI_DEV_T_ANY, pdip, 0,
            "phymask", 0);

        /*
         * Enumerate RAID volumes here (phymask == 0).
         */
        if (phymask == 0) {
                mptsas_config_all_viport(pdip);
                return;
        }

        mutex_enter(&mpt->m_mutex);

        if (!mpt->m_done_traverse_dev || !mpt->m_done_traverse_smp) {
                mptsas_update_hashtab(mpt);
        }

        for (psmp = refhash_first(mpt->m_smp_targets); psmp != NULL;
            psmp = refhash_next(mpt->m_smp_targets, psmp)) {
                phy_mask = psmp->m_addr.mta_phymask;
                if (phy_mask == phymask) {
                        smpdip = NULL;
                        mutex_exit(&mpt->m_mutex);
                        (void) mptsas_online_smp(pdip, psmp, &smpdip);
                        mutex_enter(&mpt->m_mutex);
                }
        }

        for (ptgt = refhash_first(mpt->m_targets); ptgt != NULL;
            ptgt = refhash_next(mpt->m_targets, ptgt)) {
                phy_mask = ptgt->m_addr.mta_phymask;
```



```
                if (phy_mask == phymask) {
                        mutex_exit(&mpt->m_mutex);
                        (void) mptsas_config_target(pdip, ptgt);
                        mutex_enter(&mpt->m_mutex);
                }
        }
        mutex_exit(&mpt->m_mutex);
}

static int
mptsas_config_target(dev_info_t *pdip, mptsas_target_t *ptgt)
{
        int             rval = DDI_FAILURE;
        dev_info_t      *tdip;

        rval = mptsas_config_luns(pdip, ptgt);
        if (rval != DDI_SUCCESS) {
                /*
                 * The return value means the SCMD_REPORT_LUNS
                 * did not execute successfully. The target maybe
                 * doesn't support such command.
                 */
                rval = mptsas_probe_lun(pdip, 0, &tdip, ptgt);
        }
        return (rval);
}

/*
 * Return fail if not all the childs/paths are freed.
 * if there is any path under the HBA, the return value will be always fail
 * because we didn't call mdi_pi_free for path
 */
static int
mptsas_offline_targetdev(dev_info_t *pdip, char *name)
{
        dev_info_t              *child = NULL, *prechild = NULL;
        mdi_pathinfo_t          *pip = NULL, *savepip = NULL;
        int                     tmp_rval, rval = DDI_SUCCESS;
        char                    *addr, *cp;
        size_t                  s;
        mptsas_t                *mpt = DIP2MPT(pdip);

        child = ddi_get_child(pdip);
        while (child) {
                addr = ddi_get_name_addr(child);
                prechild = child;
                child = ddi_get_next_sibling(child);

                if (addr == NULL) {
                        continue;
                }
                if ((cp = strchr(addr, ',')) == NULL) {
                        continue;
                }

                s = (uintptr_t)cp - (uintptr_t)addr;

                if (strncmp(addr, name, s) != 0) {
                        continue;
                }

                tmp_rval = mptsas_offline_lun(pdip, prechild, NULL,
                    NDI_DEVI_REMOVE);
                if (tmp_rval != DDI_SUCCESS) {
                        rval = DDI_FAILURE;
                        if (ndi_prop_create_boolean(DDI_DEV_T_NONE,
```

```
                            prechild, MPTSAS_DEV_GONE) !=
                            DDI_PROP_SUCCESS) {
                                mptsas_log(mpt, CE_WARN,
                                    "unable to create property for "
                                    "SAS %s (MPTSAS_DEV_GONE)", addr);
                            }
                    }
        }

        pip = mdi_get_next_client_path(pdip, NULL);
        while (pip) {
                addr = MDI_PI(pip)->pi_addr;
                savepip = pip;
                pip = mdi_get_next_client_path(pdip, pip);
                if (addr == NULL) {
                        continue;
                }

                if ((cp = strchr(addr, ',')) == NULL) {
                        continue;
                }

                s = (uintptr_t)cp - (uintptr_t)addr;

                if (strncmp(addr, name, s) != 0) {
                        continue;
                }

                (void) mptsas_offline_lun(pdip, NULL, savepip,
                    NDI_DEVI_REMOVE);
                /*
                 * driver will not invoke mdi_pi_free, so path will not
                 * be freed forever, return DDI_FAILURE.
                 */
                rval = DDI_FAILURE;
        }
        return (rval);
}

static int
mptsas_offline_lun(dev_info_t *pdip, dev_info_t *rdip,
    mdi_pathinfo_t *rpip, uint_t flags)
{
        int             rval = DDI_FAILURE;
        char            *devname;
        dev_info_t      *cdip, *parent;

        if (rpip != NULL) {
                parent = scsi_vhci_dip;
                cdip = mdi_pi_get_client(rpip);
        } else if (rdip != NULL) {
                parent = pdip;
                cdip = rdip;
        } else {
                return (DDI_FAILURE);
        }

        /*
         * Make sure node is attached otherwise
         * it won't have related cache nodes to
         * clean up.  i_ddi_devi_attached is
         * similiar to i_ddi_node_state(cdip) >=
         * DS_ATTACHED.
         */
        if (i_ddi_devi_attached(cdip)) {
```



```
                /* Get full devname */
                devname = kmem_alloc(MAXNAMELEN + 1, KM_SLEEP);
                (void) ddi_deviname(cdip, devname);
                /* Clean cache */
                (void) devfs_clean(parent, devname + 1,
                    DV_CLEAN_FORCE);
                kmem_free(devname, MAXNAMELEN + 1);
        }
        if (rpip != NULL) {
                if (MDI_PI_IS_OFFLINE(rpip)) {
                        rval = DDI_SUCCESS;
                } else {
                        rval = mdi_pi_offline(rpip, 0);
                }
        } else {
                rval = ndi_devi_offline(cdip, flags);
        }

        return (rval);
}

static dev_info_t *
mptsas_find_smp_child(dev_info_t *parent, char *str_wwn)
{
        dev_info_t      *child = NULL;
        char            *smp_wwn = NULL;

        child = ddi_get_child(parent);
        while (child) {
                if (ddi_prop_lookup_string(DDI_DEV_T_ANY, child,
                    DDI_PROP_DONTPASS, SMP_WWN, &smp_wwn)
                    != DDI_SUCCESS) {
                        child = ddi_get_next_sibling(child);
                        continue;
                }

                if (strcmp(smp_wwn, str_wwn) == 0) {
                        ddi_prop_free(smp_wwn);
                        break;
                }
                child = ddi_get_next_sibling(child);
                ddi_prop_free(smp_wwn);
        }
        return (child);
}

static int
mptsas_offline_smp(dev_info_t *pdip, mptsas_smp_t *smp_node, uint_t flags)
{
        int             rval = DDI_FAILURE;
        char            *devname;
        char            wwn_str[MPTSAS_WWN_STRLEN];
        dev_info_t      *cdip;

        (void) sprintf(wwn_str, "%"PRIx64, smp_node->m_addr.mta_wwn);

        cdip = mptsas_find_smp_child(pdip, wwn_str);

        if (cdip == NULL)
                return (DDI_SUCCESS);

        /*
         * Make sure node is attached otherwise
         * it won't have related cache nodes to
         * clean up.  i_ddi_devi_attached is
         * similiar to i_ddi_node_state(cdip) >=
```

```
 * DS_ATTACHED.
 */
        if (i_ddi_devi_attached(cdip)) {

                /* Get full devname */
                devname = kmem_alloc(MAXNAMELEN + 1, KM_SLEEP);
                (void) ddi_deviname(cdip, devname);
                /* Clean cache */
                (void) devfs_clean(pdip, devname + 1,
                    DV_CLEAN_FORCE);
                kmem_free(devname, MAXNAMELEN + 1);
        }

        rval = ndi_devi_offline(cdip, flags);

        return (rval);
}

static dev_info_t *
mptsas_find_child(dev_info_t *pdip, char *name)
{
        dev_info_t      *child = NULL;
        char            *rname = NULL;
        int             rval = DDI_FAILURE;

        rname = kmem_zalloc(SCSI_MAXNAMELEN, KM_SLEEP);

        child = ddi_get_child(pdip);
        while (child) {
                rval = mptsas_name_child(child, rname, SCSI_MAXNAMELEN);
                if (rval != DDI_SUCCESS) {
                        child = ddi_get_next_sibling(child);
                        bzero(rname, SCSI_MAXNAMELEN);
                        continue;
                }

                if (strcmp(rname, name) == 0) {
                        break;
                }
                child = ddi_get_next_sibling(child);
                bzero(rname, SCSI_MAXNAMELEN);
        }

        kmem_free(rname, SCSI_MAXNAMELEN);

        return (child);
}


static dev_info_t *
mptsas_find_child_addr(dev_info_t *pdip, uint64_t sasaddr, int lun)
{
        dev_info_t      *child = NULL;
        char            *name = NULL;
        char            *addr = NULL;

        name = kmem_zalloc(SCSI_MAXNAMELEN, KM_SLEEP);
        addr = kmem_zalloc(SCSI_MAXNAMELEN, KM_SLEEP);
        (void) sprintf(name, "%016"PRIx64, sasaddr);
        (void) sprintf(addr, "w%s,%x", name, lun);
        child = mptsas_find_child(pdip, addr);
        kmem_free(name, SCSI_MAXNAMELEN);
        kmem_free(addr, SCSI_MAXNAMELEN);
        return (child);
}
```



```
static dev_info_t *
mptsas_find_child_phy(dev_info_t *pdip, uint8_t phy)
{
        dev_info_t      *child;
        char            *addr;

        addr = kmem_zalloc(SCSI_MAXNAMELEN, KM_SLEEP);
        (void) sprintf(addr, "p%x,0", phy);
        child = mptsas_find_child(pdip, addr);
        kmem_free(addr, SCSI_MAXNAMELEN);
        return (child);
}

static mdi_pathinfo_t *
mptsas_find_path_phy(dev_info_t *pdip, uint8_t phy)
{
        mdi_pathinfo_t  *path;
        char            *addr = NULL;

        addr = kmem_zalloc(SCSI_MAXNAMELEN, KM_SLEEP);
        (void) sprintf(addr, "p%x,0", phy);
        path = mdi_pi_find(pdip, NULL, addr);
        kmem_free(addr, SCSI_MAXNAMELEN);
        return (path);
}

static mdi_pathinfo_t *
mptsas_find_path_addr(dev_info_t *parent, uint64_t sasaddr, int lun)
{
        mdi_pathinfo_t  *path;
        char            *name = NULL;
        char            *addr = NULL;

        name = kmem_zalloc(SCSI_MAXNAMELEN, KM_SLEEP);
        addr = kmem_zalloc(SCSI_MAXNAMELEN, KM_SLEEP);
        (void) sprintf(name, "%016"PRIx64, sasaddr);
        (void) sprintf(addr, "w%s,%x", name, lun);
        path = mdi_pi_find(parent, NULL, addr);
        kmem_free(name, SCSI_MAXNAMELEN);
        kmem_free(addr, SCSI_MAXNAMELEN);

        return (path);
}

static int
mptsas_create_lun(dev_info_t *pdip, struct scsi_inquiry *sd_inq,
    dev_info_t **lun_dip, mptsas_target_t *ptgt, int lun)
{
        int                     i = 0;
        uchar_t                 *inq83 = NULL;
        int                     inq83_len1 = 0xFF;
        int                     inq83_len = 0;
        int                     rval = DDI_FAILURE;
        ddi_devid_t             devid;
        char                    *guid = NULL;
        int                     target = ptgt->m_devhdl;
        mdi_pathinfo_t          *pip = NULL;
        mptsas_t                *mpt = DIP2MPT(pdip);

        /*
         * For DVD/CD ROM and tape devices and optical
         * devices, we won't try to enumerate them under
         * scsi_vhci, so no need to try page83
         */
        if (sd_inq && (sd_inq->inq_dtype == DTYPE_RODIRECT ||
            sd_inq->inq_dtype == DTYPE_OPTICAL ||
```

```
            sd_inq->inq_dtype == DTYPE_ESI))
                goto create_lun;

        /*
         * The LCA returns good SCSI status, but corrupt page 83 data the first
         * time it is queried. The solution is to keep trying to request page83
         * and verify the GUID is not (DDI_NOT_WELL_FORMED) in
         * mptsas_inq83_retry_timeout seconds. If the timeout expires, driver
         * give up to get VPD page at this stage and fail the enumeration.
         */

        inq83   = kmem_zalloc(inq83_len1, KM_SLEEP);

        for (i = 0; i < mptsas_inq83_retry_timeout; i++) {
                rval = mptsas_inquiry(mpt, ptgt, lun, 0x83, inq83,
                    inq83_len1, &inq83_len, 1);
                if (rval != 0) {
                        mptsas_log(mpt, CE_WARN, "!mptsas request inquiry page "
                            "0x83 for target:%x, lun:%x failed!", target, lun);
                        if (mptsas_physical_bind_failed_page_83 != B_FALSE)
                                goto create_lun;
                        goto out;
                }
                /*
                 * create DEVID from inquiry data
                 */
                if ((rval = ddi_devid_scsi_encode(
                    DEVID_SCSI_ENCODE_VERSION_LATEST, NULL, (uchar_t *)sd_inq,
                    sizeof (struct scsi_inquiry), NULL, 0, inq83,
                    (size_t)inq83_len, &devid)) == DDI_SUCCESS) {
                        /*
                         * extract GUID from DEVID
                         */
                        guid = ddi_devid_to_guid(devid);

                        /*
                         * Do not enable MPXIO if the strlen(guid) is greater
                         * than MPTSAS_MAX_GUID_LEN, this constrain would be
                         * handled by framework later.
                         */
                        if (guid && (strlen(guid) > MPTSAS_MAX_GUID_LEN)) {
                                ddi_devid_free_guid(guid);
                                guid = NULL;
                                if (mpt->m_mpxio_enable == TRUE) {
                                        mptsas_log(mpt, CE_NOTE, "!Target:%x, "
                                            "lun:%x doesn't have a valid GUID, "
                                            "multipathing for this drive is "
                                            "not enabled", target, lun);
                                }
                        }

                        /*
                         * devid no longer needed
                         */
                        ddi_devid_free(devid);
                        break;
                } else if (rval == DDI_NOT_WELL_FORMED) {
                        /*
                         * return value of ddi_devid_scsi_encode equal to
                         * DDI_NOT_WELL_FORMED means DEVID_RETRY, it worth
                         * to retry inquiry page 0x83 and get GUID.
                         */
                        NDBG20(("Not well formed devid, retry..."));
                        delay(1 * drv_usectohz(1000000));
                        continue;
                } else {
```



```
                                mptsas_log(mpt, CE_WARN, "!Encode devid failed for "
                                    "path target:%x, lun:%x", target, lun);
                                rval = DDI_FAILURE;
                                goto create_lun;
                        }
                }

                if (i == mptsas_inq83_retry_timeout) {
                        mptsas_log(mpt, CE_WARN, "!Repeated page83 requests timeout "
                            "for path target:%x, lun:%x", target, lun);
                }

                rval = DDI_FAILURE;

create_lun:
                if ((guid != NULL) && (mpt->m_mpxio_enable == TRUE)) {
                        rval = mptsas_create_virt_lun(pdip, sd_inq, guid, lun_dip, &pip,
                            ptgt, lun);
                }
                if (rval != DDI_SUCCESS) {
                        rval = mptsas_create_phys_lun(pdip, sd_inq, guid, lun_dip,
                            ptgt, lun);

                }
out:
                if (guid != NULL) {
                        /*
                         * guid no longer needed
                         */
                        ddi_devid_free_guid(guid);
                }
                if (inq83 != NULL)
                        kmem_free(inq83, inq83_len1);
                return (rval);
}

static int
mptsas_create_virt_lun(dev_info_t *pdip, struct scsi_inquiry *inq, char *guid,
    dev_info_t **lun_dip, mdi_pathinfo_t **pip, mptsas_target_t *ptgt, int lun)
{
        int                     target;
        char                    *nodename = NULL;
        char                    **compatible = NULL;
        int                     ncompatible     = 0;
        int                     mdi_rtn = MDI_FAILURE;
        int                     rval = DDI_FAILURE;
        char                    *old_guid = NULL;
        mptsas_t                *mpt = DIP2MPT(pdip);
        char                    *lun_addr = NULL;
        char                    wwn_str[MPTSAS_WWN_STRLEN];
        char                    *component = NULL;
        uint8_t                 phy = 0xFF;
        uint64_t                sas_wwn;
        int64_t                 lun64 = 0;
        uint32_t                devinfo;
        uint16_t                dev_hdl;
        uint16_t                pdev_hdl;
        uint64_t                dev_sas_wwn;
        uint64_t                pdev_sas_wwn;
        uint32_t                pdev_info;
        uint8_t                 physport;
        uint8_t                 phy_id;
        uint32_t                page_address;
        uint16_t                bay_num, enclosure, io_flags;
        char                    pdev_wwn_str[MPTSAS_WWN_STRLEN];
        uint32_t                dev_info;
```

```
        mutex_enter(&mpt->m_mutex);
        target = ptgt->m_devhdl;
        sas_wwn = ptgt->m_addr.mta_wwn;
        devinfo = ptgt->m_deviceinfo;
        phy = ptgt->m_phynum;
        mutex_exit(&mpt->m_mutex);

        if (sas_wwn) {
                *pip = mptsas_find_path_addr(pdip, sas_wwn, lun);
        } else {
                *pip = mptsas_find_path_phy(pdip, phy);
        }

        if (*pip != NULL) {
                *lun_dip = MDI_PI(*pip)->pi_client->ct_dip;
                ASSERT(*lun_dip != NULL);
                if (ddi_prop_lookup_string(DDI_DEV_T_ANY, *lun_dip,
                    (DDI_PROP_DONTPASS | DDI_PROP_NOTPROM),
                    MDI_CLIENT_GUID_PROP, &old_guid) == DDI_SUCCESS) {
                        if (strncmp(guid, old_guid, strlen(guid)) == 0) {
                                /*
                                 * Same path back online again.
                                 */
                                (void) ddi_prop_free(old_guid);
                                if ((!MDI_PI_IS_ONLINE(*pip)) &&
                                    (!MDI_PI_IS_STANDBY(*pip)) &&
                                    (ptgt->m_tgt_unconfigured == 0)) {
                                        rval = mdi_pi_online(*pip, 0);
                                        mutex_enter(&mpt->m_mutex);
                                        ptgt->m_led_status = 0;
                                        (void) mptsas_flush_led_status(mpt,
                                            ptgt);
                                        mutex_exit(&mpt->m_mutex);
                                } else {
                                        rval = DDI_SUCCESS;
                                }
                                if (rval != DDI_SUCCESS) {
                                        mptsas_log(mpt, CE_WARN, "path:target: "
                                            "%x, lun:%x online failed!", target,
                                            lun);
                                        *pip = NULL;
                                        *lun_dip = NULL;
                                }
                                return (rval);
                        } else {
                                /*
                                 * The GUID of the LUN has changed which maybe
                                 * because customer mapped another volume to the
                                 * same LUN.
                                 */
                                mptsas_log(mpt, CE_WARN, "The GUID of the "
                                    "target:%x, lun:%x was changed, maybe "
                                    "because someone mapped another volume "
                                    "to the same LUN", target, lun);
                                (void) ddi_prop_free(old_guid);
                                if (!MDI_PI_IS_OFFLINE(*pip)) {
                                        rval = mdi_pi_offline(*pip, 0);
                                        if (rval != MDI_SUCCESS) {
                                                mptsas_log(mpt, CE_WARN, "path:"
                                                    "target:%x, lun:%x offline "
                                                    "failed!", target, lun);
                                                *pip = NULL;
                                                *lun_dip = NULL;
                                                return (DDI_FAILURE);
                                        }
```



```
                                }
                                if (mdi_pi_free(*pip, 0) != MDI_SUCCESS) {
                                        mptsas_log(mpt, CE_WARN, "path:target:"
                                            "%x, lun:%x free failed!", target,
                                            lun);
                                        *pip = NULL;
                                        *lun_dip = NULL;
                                        return (DDI_FAILURE);
                                }
                        }
                } else {
                        mptsas_log(mpt, CE_WARN, "Can't get client-guid "
                            "property for path:target:%x, lun:%x", target, lun);
                        *pip = NULL;
                        *lun_dip = NULL;
                        return (DDI_FAILURE);
                }
        }
        scsi_hba_nodename_compatible_get(inq, NULL,
            inq->inq_dtype, NULL, &nodename, &compatible, &ncompatible);

        /*
         * if nodename can't be determined then print a message and skip it
         */
        if (nodename == NULL) {
                mptsas_log(mpt, CE_WARN, "found no compatible "
                    "driver for target%d lun %d dtype:0x%02x", target, lun,
                    inq->inq_dtype);
                return (DDI_FAILURE);
        }

        /* The property is needed by MPAPI */
        (void) sprintf(wwn_str, "%016"PRIx64, sas_wwn);

        lun_addr = kmem_zalloc(SCSI_MAXNAMELEN, KM_SLEEP);
        if (guid) {
                (void) sprintf(lun_addr, "w%s,%x", wwn_str, lun);
                (void) sprintf(wwn_str, "w%016"PRIx64, sas_wwn);
        } else {
                (void) sprintf(lun_addr, "p%x,%x", phy, lun);
                (void) sprintf(wwn_str, "p%x", phy);
        }

        mdi_rtn = mdi_pi_alloc_compatible(pdip, nodename,
            guid, lun_addr, compatible, ncompatible,
            0, pip);
        if (mdi_rtn == MDI_SUCCESS) {

                if (mdi_prop_update_string(*pip, MDI_GUID,
                    guid) != DDI_SUCCESS) {
                        mptsas_log(mpt, CE_WARN, "unable to "
                            "create prop for target %d lun %d (MDI_GUID)",
                            target, lun);
                        mdi_rtn = MDI_FAILURE;
                        goto virt_create_done;
                }

                if (mdi_prop_update_int(*pip, LUN_PROP,
                    lun) != DDI_SUCCESS) {
                        mptsas_log(mpt, CE_WARN, "unable to "
                            "create prop for target %d lun %d (LUN_PROP)",
                            target, lun);
                        mdi_rtn = MDI_FAILURE;
                        goto virt_create_done;
                }
                lun64 = (int64_t)lun;
```

```
                if (mdi_prop_update_int64(*pip, LUN64_PROP,
                    lun64) != DDI_SUCCESS) {
                        mptsas_log(mpt, CE_WARN, "unable to "
                            "create prop for target %d (LUN64_PROP)",
                            target);
                        mdi_rtn = MDI_FAILURE;
                        goto virt_create_done;
                }
                if (mdi_prop_update_string_array(*pip, "compatible",
                    compatible, ncompatible) !=
                    DDI_PROP_SUCCESS) {
                        mptsas_log(mpt, CE_WARN, "unable to "
                            "create prop for target %d lun %d (COMPATIBLE)",
                            target, lun);
                        mdi_rtn = MDI_FAILURE;
                        goto virt_create_done;
                }
                if (sas_wwn && (mdi_prop_update_string(*pip,
                    SCSI_ADDR_PROP_TARGET_PORT, wwn_str) != DDI_PROP_SUCCESS)) {
                        mptsas_log(mpt, CE_WARN, "unable to "
                            "create prop for target %d lun %d "
                            "(target-port)", target, lun);
                        mdi_rtn = MDI_FAILURE;
                        goto virt_create_done;
                } else if ((sas_wwn == 0) && (mdi_prop_update_int(*pip,
                    "sata-phy", phy) != DDI_PROP_SUCCESS)) {
                        /*
                         * Direct attached SATA device without DeviceName
                         */
                        mptsas_log(mpt, CE_WARN, "unable to "
                            "create prop for SAS target %d lun %d "
                            "(sata-phy)", target, lun);
                        mdi_rtn = MDI_FAILURE;
                        goto virt_create_done;
                }
                mutex_enter(&mpt->m_mutex);

                page_address = (MPI2_SAS_DEVICE_PGAD_FORM_HANDLE &
                    MPI2_SAS_DEVICE_PGAD_FORM_MASK) |
                    (uint32_t)ptgt->m_devhdl;
                rval = mptsas_get_sas_device_page0(mpt, page_address,
                    &dev_hdl, &dev_sas_wwn, &dev_info, &physport,
                    &phy_id, &pdev_hdl, &bay_num, &enclosure, &io_flags);
                if (rval != DDI_SUCCESS) {
                        mutex_exit(&mpt->m_mutex);
                        mptsas_log(mpt, CE_WARN, "mptsas unable to get "
                            "parent device for handle %d", page_address);
                        mdi_rtn = MDI_FAILURE;
                        goto virt_create_done;
                }

                page_address = (MPI2_SAS_DEVICE_PGAD_FORM_HANDLE &
                    MPI2_SAS_DEVICE_PGAD_FORM_MASK) | (uint32_t)pdev_hdl;
                rval = mptsas_get_sas_device_page0(mpt, page_address,
                    &dev_hdl, &pdev_sas_wwn, &pdev_info, &physport,
                    &phy_id, &pdev_hdl, &bay_num, &enclosure, &io_flags);
                if (rval != DDI_SUCCESS) {
                        mutex_exit(&mpt->m_mutex);
                        mptsas_log(mpt, CE_WARN, "mptsas unable to get"
                            "device info for handle %d", page_address);
                        mdi_rtn = MDI_FAILURE;
                        goto virt_create_done;
                }

                mutex_exit(&mpt->m_mutex);
```



```
                /*
                 * If this device direct attached to the controller
                 * set the attached-port to the base wwid
                 */
                if ((ptgt->m_deviceinfo & DEVINFO_DIRECT_ATTACHED)
                    != DEVINFO_DIRECT_ATTACHED) {
                        (void) sprintf(pdev_wwn_str, "w%016"PRIx64,
                            pdev_sas_wwn);
                } else {
                        /*
                         * Update the iport's attached-port to guid
                         */
                        if (sas_wwn == 0) {
                                (void) sprintf(wwn_str, "p%x", phy);
                        } else {
                                (void) sprintf(wwn_str, "w%016"PRIx64, sas_wwn);
                        }
                        if (ddi_prop_update_string(DDI_DEV_T_NONE,
                            pdip, SCSI_ADDR_PROP_ATTACHED_PORT, wwn_str) !=
                            DDI_PROP_SUCCESS) {
                                mptsas_log(mpt, CE_WARN,
                                    "mptsas unable to create "
                                    "property for iport target-port"
                                    " %s (sas_wwn)",
                                    wwn_str);
                                mdi_rtn = MDI_FAILURE;
                                goto virt_create_done;
                        }

                        (void) sprintf(pdev_wwn_str, "w%016"PRIx64,
                            mpt->un.m_base_wwid);
                }

                if (mdi_prop_update_string(*pip,
                    SCSI_ADDR_PROP_ATTACHED_PORT, pdev_wwn_str) !=
                    DDI_PROP_SUCCESS) {
                        mptsas_log(mpt, CE_WARN, "unable to create "
                            "property for iport attached-port %s (sas_wwn)",
                            pdev_wwn_str);
                        mdi_rtn = MDI_FAILURE;
                        goto virt_create_done;
                }


                if (inq->inq_dtype == 0) {
                        component = kmem_zalloc(MAXPATHLEN, KM_SLEEP);
                        /*
                         * set obp path for pathinfo
                         */
                        (void) snprintf(component, MAXPATHLEN,
                            "disk@%s", lun_addr);

                        if (mdi_pi_pathname_obp_set(*pip, component) !=
                            DDI_SUCCESS) {
                                mptsas_log(mpt, CE_WARN,
                                    "unable to set obp-path for object %s",
                                    component);
                                mdi_rtn = MDI_FAILURE;
                                goto virt_create_done;
                        }
                }

                *lun_dip = MDI_PI(*pip)->pi_client->ct_dip;
                if (devinfo & (MPI2_SAS_DEVICE_INFO_SATA_DEVICE |
                    MPI2_SAS_DEVICE_INFO_ATAPI_DEVICE)) {
                        if ((ndi_prop_update_int(DDI_DEV_T_NONE, *lun_dip,
```

```
                            "pm-capable", 1)) !=
                            DDI_PROP_SUCCESS) {
                                mptsas_log(mpt, CE_WARN,
                                    "failed to create pm-capable "
                                    "property, target %d", target);
                                mdi_rtn = MDI_FAILURE;
                                goto virt_create_done;
                        }
                }
                /*
                 * Create the phy-num property
                 */
                if (mdi_prop_update_int(*pip, "phy-num",
                    ptgt->m_phynum) != DDI_SUCCESS) {
                        mptsas_log(mpt, CE_WARN, "unable to "
                            "create phy-num property for target %d lun %d",
                            target, lun);
                        mdi_rtn = MDI_FAILURE;
                        goto virt_create_done;
                }
                NDBG20(("new path:%s onlining,", MDI_PI(*pip)->pi_addr));
                mdi_rtn = mdi_pi_online(*pip, 0);
                if (mdi_rtn == MDI_SUCCESS) {
                        mutex_enter(&mpt->m_mutex);
                        ptgt->m_led_status = 0;
                        (void) mptsas_flush_led_status(mpt, ptgt);
                        mutex_exit(&mpt->m_mutex);
                }
                if (mdi_rtn == MDI_NOT_SUPPORTED) {
                        mdi_rtn = MDI_FAILURE;
                }
virt_create_done:
                if (*pip && mdi_rtn != MDI_SUCCESS) {
                        (void) mdi_pi_free(*pip, 0);
                        *pip = NULL;
                        *lun_dip = NULL;
                }
        }

        scsi_hba_nodename_compatible_free(nodename, compatible);
        if (lun_addr != NULL) {
                kmem_free(lun_addr, SCSI_MAXNAMELEN);
        }
        if (component != NULL) {
                kmem_free(component, MAXPATHLEN);
        }

        return ((mdi_rtn == MDI_SUCCESS) ? DDI_SUCCESS : DDI_FAILURE);
}

static int
mptsas_create_phys_lun(dev_info_t *pdip, struct scsi_inquiry *inq,
    char *guid, dev_info_t **lun_dip, mptsas_target_t *ptgt, int lun)
{
        int                     target;
        int                     rval;
        int                     ndi_rtn = NDI_FAILURE;
        uint64_t                be_sas_wwn;
        char                    *nodename = NULL;
        char                    **compatible = NULL;
        int                     ncompatible = 0;
        int                     instance = 0;
        mptsas_t                *mpt = DIP2MPT(pdip);
        char                    wwn_str[MPTSAS_WWN_STRLEN];
        char                    component[MAXPATHLEN];
        uint8_t                 phy = 0xFF;
```



```
        uint64_t                sas_wwn;
        uint32_t                devinfo;
        uint16_t                dev_hdl;
        uint16_t                pdev_hdl;
        uint64_t                pdev_sas_wwn;
        uint64_t                dev_sas_wwn;
        uint32_t                pdev_info;
        uint8_t                 physport;
        uint8_t                 phy_id;
        uint32_t                page_address;
        uint16_t                bay_num, enclosure, io_flags;
        char                    pdev_wwn_str[MPTSAS_WWN_STRLEN];
        uint32_t                dev_info;
        int64_t                 lun64 = 0;

        mutex_enter(&mpt->m_mutex);
        target = ptgt->m_devhdl;
        sas_wwn = ptgt->m_addr.mta_wwn;
        devinfo = ptgt->m_deviceinfo;
        phy = ptgt->m_phynum;
        mutex_exit(&mpt->m_mutex);

        /*
         * generate compatible property with binding-set "mpt"
         */
        scsi_hba_nodename_compatible_get(inq, NULL, inq->inq_dtype, NULL,
            &nodename, &compatible, &ncompatible);

        /*
         * if nodename can’t be determined then print a message and skip it
         */
        if (nodename == NULL) {
                mptsas_log(mpt, CE_WARN, "mptsas found no compatible driver "
                    "for target %d lun %d", target, lun);
                return (DDI_FAILURE);
        }

        ndi_rtn = ndi_devi_alloc(pdip, nodename,
            DEVI_SID_NODEID, lun_dip);

        /*
         * if lun alloc success, set props
         */
        if (ndi_rtn == NDI_SUCCESS) {

                if (ndi_prop_update_int(DDI_DEV_T_NONE,
                    *lun_dip, LUN_PROP, lun) !=
                    DDI_PROP_SUCCESS) {
                        mptsas_log(mpt, CE_WARN, "mptsas unable to create "
                            "property for target %d lun %d (LUN_PROP)",
                            target, lun);
                        ndi_rtn = NDI_FAILURE;
                        goto phys_create_done;
                }

                lun64 = (int64_t)lun;
                if (ndi_prop_update_int64(DDI_DEV_T_NONE,
                    *lun_dip, LUN64_PROP, lun64) !=
                    DDI_PROP_SUCCESS) {
                        mptsas_log(mpt, CE_WARN, "mptsas unable to create "
                            "property for target %d lun64 %d (LUN64_PROP)",
                            target, lun);
                        ndi_rtn = NDI_FAILURE;
                        goto phys_create_done;
                }
                if (ndi_prop_update_string_array(DDI_DEV_T_NONE,
```

```
                    *lun_dip, "compatible", compatible, ncompatible)
                    != DDI_PROP_SUCCESS) {
                        mptsas_log(mpt, CE_WARN, "mptsas unable to create "
                            "property for target %d lun %d (COMPATIBLE)",
                            target, lun);
                        ndi_rtn = NDI_FAILURE;
                        goto phys_create_done;
                }

                /*
                 * We need the SAS WWN for non-multipath devices, so
                 * we'll use the same property as that multipathing
                 * devices need to present for MPAPI. If we don't have
                 * a WWN (e.g. parallel SCSI), don't create the prop.
                 */
                (void) sprintf(wwn_str, "w%016"PRIx64, sas_wwn);
                if (sas_wwn && ndi_prop_update_string(DDI_DEV_T_NONE,
                    *lun_dip, SCSI_ADDR_PROP_TARGET_PORT, wwn_str)
                    != DDI_PROP_SUCCESS) {
                        mptsas_log(mpt, CE_WARN, "mptsas unable to "
                            "create property for SAS target %d lun %d "
                            "(target-port)", target, lun);
                        ndi_rtn = NDI_FAILURE;
                        goto phys_create_done;
                }

                be_sas_wwn = BE_64(sas_wwn);
                if (sas_wwn && ndi_prop_update_byte_array(
                    DDI_DEV_T_NONE, *lun_dip, "port-wwn",
                    (uchar_t *)&be_sas_wwn, 8) != DDI_PROP_SUCCESS) {
                        mptsas_log(mpt, CE_WARN, "mptsas unable to "
                            "create property for SAS target %d lun %d "
                            "(port-wwn)", target, lun);
                        ndi_rtn = NDI_FAILURE;
                        goto phys_create_done;
                } else if ((sas_wwn == 0) && (ndi_prop_update_int(
                    DDI_DEV_T_NONE, *lun_dip, "sata-phy", phy) !=
                    DDI_PROP_SUCCESS)) {
                        /*
                         * Direct attached SATA device without DeviceName
                         */
                        mptsas_log(mpt, CE_WARN, "mptsas unable to "
                            "create property for SAS target %d lun %d "
                            "(sata-phy)", target, lun);
                        ndi_rtn = NDI_FAILURE;
                        goto phys_create_done;
                }

                if (ndi_prop_create_boolean(DDI_DEV_T_NONE,
                    *lun_dip, SAS_PROP) != DDI_PROP_SUCCESS) {
                        mptsas_log(mpt, CE_WARN, "mptsas unable to"
                            "create property for SAS target %d lun %d"
                            " (SAS_PROP)", target, lun);
                        ndi_rtn = NDI_FAILURE;
                        goto phys_create_done;
                }
                if (guid && (ndi_prop_update_string(DDI_DEV_T_NONE,
                    *lun_dip, NDI_GUID, guid) != DDI_SUCCESS)) {
                        mptsas_log(mpt, CE_WARN, "mptsas unable "
                            "to create guid property for target %d "
                            "lun %d", target, lun);
                        ndi_rtn = NDI_FAILURE;
                        goto phys_create_done;
                }

                /*
```



```
                 * The following code is to set properties for SM-HBA support,
                 * it doesn't apply to RAID volumes
                 */
                if (ptgt->m_addr.mta_phymask == 0)
                        goto phys_raid_lun;

                mutex_enter(&mpt->m_mutex);

                page_address = (MPI2_SAS_DEVICE_PGAD_FORM_HANDLE &
                    MPI2_SAS_DEVICE_PGAD_FORM_MASK) |
                    (uint32_t)ptgt->m_devhdl;
                rval = mptsas_get_sas_device_page0(mpt, page_address,
                    &dev_hdl, &dev_sas_wwn, &dev_info,
                    &physport, &phy_id, &pdev_hdl,
                    &bay_num, &enclosure, &io_flags);
                if (rval != DDI_SUCCESS) {
                        mutex_exit(&mpt->m_mutex);
                        mptsas_log(mpt, CE_WARN, "mptsas unable to get"
                            "parent device for handle %d.", page_address);
                        ndi_rtn = NDI_FAILURE;
                        goto phys_create_done;
                }

                page_address = (MPI2_SAS_DEVICE_PGAD_FORM_HANDLE &
                    MPI2_SAS_DEVICE_PGAD_FORM_MASK) | (uint32_t)pdev_hdl;
                rval = mptsas_get_sas_device_page0(mpt, page_address,
                    &dev_hdl, &pdev_sas_wwn, &pdev_info, &physport,
                    &phy_id, &pdev_hdl, &bay_num, &enclosure, &io_flags);
                if (rval != DDI_SUCCESS) {
                        mutex_exit(&mpt->m_mutex);
                        mptsas_log(mpt, CE_WARN, "mptsas unable to create "
                            "device for handle %d.", page_address);
                        ndi_rtn = NDI_FAILURE;
                        goto phys_create_done;
                }

                mutex_exit(&mpt->m_mutex);

                /*
                 * If this device direct attached to the controller
                 * set the attached-port to the base wwid
                 */
                if ((ptgt->m_deviceinfo & DEVINFO_DIRECT_ATTACHED)
                    != DEVINFO_DIRECT_ATTACHED) {
                        (void) sprintf(pdev_wwn_str, "w%016"PRIx64,
                            pdev_sas_wwn);
                } else {
                        /*
                         * Update the iport's attached-port to guid
                         */
                        if (sas_wwn == 0) {
                                (void) sprintf(wwn_str, "p%x", phy);
                        } else {
                                (void) sprintf(wwn_str, "w%016"PRIx64, sas_wwn);
                        }
                        if (ddi_prop_update_string(DDI_DEV_T_NONE,
                            pdip, SCSI_ADDR_PROP_ATTACHED_PORT, wwn_str) !=
                            DDI_PROP_SUCCESS) {
                                mptsas_log(mpt, CE_WARN,
                                    "mptsas unable to create "
                                    "property for iport target-port"
                                    " %s (sas_wwn)",
                                    wwn_str);
                                ndi_rtn = NDI_FAILURE;
                                goto phys_create_done;
                        }
```

```
                        (void) sprintf(pdev_wwn_str, "w%016"PRIx64,
                            mpt->un.m_base_wwid);
                }

                if (ndi_prop_update_string(DDI_DEV_T_NONE,
                    *lun_dip, SCSI_ADDR_PROP_ATTACHED_PORT, pdev_wwn_str) !=
                    DDI_PROP_SUCCESS) {
                        mptsas_log(mpt, CE_WARN,
                            "mptsas unable to create "
                            "property for iport attached-port %s (sas_wwn)",
                            pdev_wwn_str);
                        ndi_rtn = NDI_FAILURE;
                        goto phys_create_done;
                }

                if (IS_SATA_DEVICE(dev_info)) {
                        if (ndi_prop_update_string(DDI_DEV_T_NONE,
                            *lun_dip, MPTSAS_VARIANT, "sata") !=
                            DDI_PROP_SUCCESS) {
                                mptsas_log(mpt, CE_WARN,
                                    "mptsas unable to create "
                                    "property for device variant ");
                                ndi_rtn = NDI_FAILURE;
                                goto phys_create_done;
                        }
                }

                if (IS_ATAPI_DEVICE(dev_info)) {
                        if (ndi_prop_update_string(DDI_DEV_T_NONE,
                            *lun_dip, MPTSAS_VARIANT, "atapi") !=
                            DDI_PROP_SUCCESS) {
                                mptsas_log(mpt, CE_WARN,
                                    "mptsas unable to create "
                                    "property for device variant ");
                                ndi_rtn = NDI_FAILURE;
                                goto phys_create_done;
                        }
                }

phys_raid_lun:
                /*
                 * if this is a SAS controller, and the target is a SATA
                 * drive, set the 'pm-capable' property for sd and if on
                 * an OPL platform, also check if this is an ATAPI
                 * device.
                 */
                instance = ddi_get_instance(mpt->m_dip);
                if (devinfo & (MPI2_SAS_DEVICE_INFO_SATA_DEVICE |
                    MPI2_SAS_DEVICE_INFO_ATAPI_DEVICE)) {
                        NDBG2(("mptsas3%d: creating pm-capable property, "
                            "target %d", instance, target));

                        if ((ndi_prop_update_int(DDI_DEV_T_NONE,
                            *lun_dip, "pm-capable", 1)) !=
                            DDI_PROP_SUCCESS) {
                                mptsas_log(mpt, CE_WARN, "mptsas "
                                    "failed to create pm-capable "
                                    "property, target %d", target);
                                ndi_rtn = NDI_FAILURE;
                                goto phys_create_done;
                        }

                }

                if ((inq->inq_dtype == 0) || (inq->inq_dtype == 5)) {
```



```
                        /*
                         * add 'obp-path' properties for devinfo
                         */
                        bzero(wwn_str, sizeof (wwn_str));
                        (void) sprintf(wwn_str, "%016"PRIx64, sas_wwn);
                        if (guid) {
                                (void) snprintf(component, MAXPATHLEN,
                                    "disk@w%s,%x", wwn_str, lun);
                        } else {
                                (void) snprintf(component, MAXPATHLEN,
                                    "disk@p%x,%x", phy, lun);
                        }
                        if (ddi_pathname_obp_set(*lun_dip, component)
                            != DDI_SUCCESS) {
                                mptsas_log(mpt, CE_WARN, "mpt_sas driver "
                                    "unable to set obp-path for SAS "
                                    "object %s", component);
                                ndi_rtn = NDI_FAILURE;
                                goto phys_create_done;
                        }
                }
                /*
                 * Create the phy-num property for non-raid disk
                 */
                if (ptgt->m_addr.mta_phymask != 0) {
                        if (ndi_prop_update_int(DDI_DEV_T_NONE,
                            *lun_dip, "phy-num", ptgt->m_phynum) !=
                            DDI_PROP_SUCCESS) {
                                mptsas_log(mpt, CE_WARN,
                                    "failed to create phy-num property for "
                                    "target %d", target);
                                ndi_rtn = NDI_FAILURE;
                                goto phys_create_done;
                        }
                }
phys_create_done:
                /*
                 * If props were setup ok, online the lun
                 */
                if (ndi_rtn == NDI_SUCCESS) {
                        /*
                         * Try to online the new node
                         */
                        ndi_rtn = ndi_devi_online(*lun_dip, NDI_ONLINE_ATTACH);
                }
                if (ndi_rtn == NDI_SUCCESS) {
                        mutex_enter(&mpt->m_mutex);
                        ptgt->m_led_status = 0;
                        (void) mptsas_flush_led_status(mpt, ptgt);
                        mutex_exit(&mpt->m_mutex);
                }

                /*
                 * If success set rtn flag, else unwire alloc'd lun
                 */
                if (ndi_rtn != NDI_SUCCESS) {
                        NDBG12(("unable to online "
                            "target %d lun %d", target, lun));
                        ndi_prop_remove_all(*lun_dip);
                        (void) ndi_devi_free(*lun_dip);
                        *lun_dip = NULL;
                }
        }

        scsi_hba_nodename_compatible_free(nodename, compatible);
```

```
        return ((ndi_rtn == NDI_SUCCESS) ? DDI_SUCCESS : DDI_FAILURE);
}

static int
mptsas_probe_smp(dev_info_t *pdip, uint64_t wwn)
{
        mptsas_t        *mpt = DIP2MPT(pdip);
        struct smp_device smp_sd;

        /* XXX An HBA driver should not be allocating an smp_device. */
        bzero(&smp_sd, sizeof (struct smp_device));
        smp_sd.smp_sd_address.smp_a_hba_tran = mpt->m_smptran;
        bcopy(&wwn, smp_sd.smp_sd_address.smp_a_wwn, SAS_WWN_BYTE_SIZE);

        if (smp_probe(&smp_sd) != DDI_PROBE_SUCCESS)
                return (NDI_FAILURE);
        return (NDI_SUCCESS);
}

static int
mptsas_config_smp(dev_info_t *pdip, uint64_t sas_wwn, dev_info_t **smp_dip)
{
        mptsas_t        *mpt = DIP2MPT(pdip);
        mptsas_smp_t    *psmp = NULL;
        int             rval;
        int             phymask;

        /*
         * Get the physical port associated to the iport
         * PHYMASK TODO
         */
        phymask = ddi_prop_get_int(DDI_DEV_T_ANY, pdip, 0,
            "phymask", 0);
        /*
         * Find the smp node in hash table with specified sas address and
         * physical port
         */
        psmp = mptsas_wwid_to_psmp(mpt, phymask, sas_wwn);
        if (psmp == NULL) {
                return (DDI_FAILURE);
        }

        rval = mptsas_online_smp(pdip, psmp, smp_dip);

        return (rval);
}

static int
mptsas_online_smp(dev_info_t *pdip, mptsas_smp_t *smp_node,
    dev_info_t **smp_dip)
{
        char            wwn_str[MPTSAS_WWN_STRLEN];
        char            attached_wwn_str[MPTSAS_WWN_STRLEN];
        int             ndi_rtn = NDI_FAILURE;
        int             rval = 0;
        mptsas_smp_t    dev_info;
        uint32_t        page_address;
        mptsas_t        *mpt = DIP2MPT(pdip);
        uint16_t        dev_hdl;
        uint64_t        sas_wwn;
        uint64_t        smp_sas_wwn;
        uint8_t         physport;
        uint8_t         phy_id;
        uint16_t        pdev_hdl;
        uint8_t         numphys = 0;
        uint16_t        i = 0;
```



```
        char            phymask[MPTSAS_MAX_PHYS];
        char            *iport = NULL;
        mptsas_phymask_t        phy_mask = 0;
        uint16_t        attached_devhdl;
        uint16_t        bay_num, enclosure, io_flags;

        (void) sprintf(wwn_str, "%"PRIx64, smp_node->m_addr.mta_wwn);

        /*
         * Probe smp device, prevent the node of removed device from being
         * configured succesfully
         */
        if (mptsas_probe_smp(pdip, smp_node->m_addr.mta_wwn) != NDI_SUCCESS) {
                return (DDI_FAILURE);
        }

        if ((*smp_dip = mptsas_find_smp_child(pdip, wwn_str)) != NULL) {
                return (DDI_SUCCESS);
        }

        ndi_rtn = ndi_devi_alloc(pdip, "smp", DEVI_SID_NODEID, smp_dip);

        /*
         * if lun alloc success, set props
         */
        if (ndi_rtn == NDI_SUCCESS) {
                /*
                 * Set the flavor of the child to be SMP flavored
                 */
                ndi_flavor_set(*smp_dip, SCSA_FLAVOR_SMP);

                if (ndi_prop_update_string(DDI_DEV_T_NONE,
                    *smp_dip, SMP_WWN, wwn_str) !=
                    DDI_PROP_SUCCESS) {
                        mptsas_log(mpt, CE_WARN, "mptsas unable to create "
                            "property for smp device %s (sas_wwn)",
                            wwn_str);
                        ndi_rtn = NDI_FAILURE;
                        goto smp_create_done;
                }
                (void) sprintf(wwn_str, "w%"PRIx64, smp_node->m_addr.mta_wwn);
                if (ndi_prop_update_string(DDI_DEV_T_NONE,
                    *smp_dip, SCSI_ADDR_PROP_TARGET_PORT, wwn_str) !=
                    DDI_PROP_SUCCESS) {
                        mptsas_log(mpt, CE_WARN, "mptsas unable to create "
                            "property for iport target-port %s (sas_wwn)",
                            wwn_str);
                        ndi_rtn = NDI_FAILURE;
                        goto smp_create_done;
                }

                mutex_enter(&mpt->m_mutex);

                page_address = (MPI2_SAS_EXPAND_PGAD_FORM_HNDL &
                    MPI2_SAS_EXPAND_PGAD_FORM_MASK) | smp_node->m_devhdl;
                rval = mptsas_get_sas_expander_page0(mpt, page_address,
                    &dev_info);
                if (rval != DDI_SUCCESS) {
                        mutex_exit(&mpt->m_mutex);
                        mptsas_log(mpt, CE_WARN,
                            "mptsas unable to get expander "
                            "parent device info for %x", page_address);
                        ndi_rtn = NDI_FAILURE;
                        goto smp_create_done;
                }
```

```
                smp_node->m_pdevhdl = dev_info.m_pdevhdl;
                page_address = (MPI2_SAS_DEVICE_PGAD_FORM_HANDLE &
                    MPI2_SAS_DEVICE_PGAD_FORM_MASK) |
                    (uint32_t)dev_info.m_pdevhdl;
                rval = mptsas_get_sas_device_page0(mpt, page_address,
                    &dev_hdl, &sas_wwn, &smp_node->m_pdevinfo, &physport,
                    &phy_id, &pdev_hdl, &bay_num, &enclosure, &io_flags);
                if (rval != DDI_SUCCESS) {
                        mutex_exit(&mpt->m_mutex);
                        mptsas_log(mpt, CE_WARN, "mptsas unable to get "
                            "device info for %x", page_address);
                        ndi_rtn = NDI_FAILURE;
                        goto smp_create_done;
                }

                page_address = (MPI2_SAS_DEVICE_PGAD_FORM_HANDLE &
                    MPI2_SAS_DEVICE_PGAD_FORM_MASK) |
                    (uint32_t)dev_info.m_devhdl;
                rval = mptsas_get_sas_device_page0(mpt, page_address,
                    &dev_hdl, &smp_sas_wwn, &smp_node->m_deviceinfo,
                    &physport, &phy_id, &pdev_hdl, &bay_num, &enclosure,
                    &io_flags);
                if (rval != DDI_SUCCESS) {
                        mutex_exit(&mpt->m_mutex);
                        mptsas_log(mpt, CE_WARN, "mptsas unable to get "
                            "device info for %x", page_address);
                        ndi_rtn = NDI_FAILURE;
                        goto smp_create_done;
                }
                mutex_exit(&mpt->m_mutex);

                /*
                 * If this smp direct attached to the controller
                 * set the attached-port to the base wwid
                 */
                if ((smp_node->m_deviceinfo & DEVINFO_DIRECT_ATTACHED)
                    != DEVINFO_DIRECT_ATTACHED) {
                        (void) sprintf(attached_wwn_str, "w%016"PRIx64,
                            sas_wwn);
                } else {
                        (void) sprintf(attached_wwn_str, "w%016"PRIx64,
                            mpt->un.m_base_wwid);
                }

                if (ndi_prop_update_string(DDI_DEV_T_NONE,
                    *smp_dip, SCSI_ADDR_PROP_ATTACHED_PORT, attached_wwn_str) !=
                    DDI_PROP_SUCCESS) {
                        mptsas_log(mpt, CE_WARN, "mptsas unable to create "
                            "property for smp attached-port %s (sas_wwn)",
                            attached_wwn_str);
                        ndi_rtn = NDI_FAILURE;
                        goto smp_create_done;
                }

                if (ndi_prop_create_boolean(DDI_DEV_T_NONE,
                    *smp_dip, SMP_PROP) != DDI_PROP_SUCCESS) {
                        mptsas_log(mpt, CE_WARN, "mptsas unable to "
                            "create property for SMP %s (SMP_PROP) ",
                            wwn_str);
                        ndi_rtn = NDI_FAILURE;
                        goto smp_create_done;
                }

                /*
                 * check the smp to see whether it direct
                 * attached to the controller
```



```
                 */
                if ((smp_node->m_deviceinfo & DEVINFO_DIRECT_ATTACHED)
                    != DEVINFO_DIRECT_ATTACHED) {
                        goto smp_create_done;
                }
                numphys = ddi_prop_get_int(DDI_DEV_T_ANY, pdip,
                    DDI_PROP_DONTPASS, MPTSAS_NUM_PHYS, -1);
                if (numphys > 0) {
                        goto smp_create_done;
                }
                /*
                 * this iport is an old iport, we need to
                 * reconfig the props for it.
                 */
                if (ddi_prop_update_int(DDI_DEV_T_NONE, pdip,
                    MPTSAS_VIRTUAL_PORT, 0) !=
                    DDI_PROP_SUCCESS) {
                        (void) ddi_prop_remove(DDI_DEV_T_NONE, pdip,
                            MPTSAS_VIRTUAL_PORT);
                        mptsas_log(mpt, CE_WARN, "mptsas virtual port "
                            "prop update failed");
                        goto smp_create_done;
                }

                mutex_enter(&mpt->m_mutex);
                numphys = 0;
                iport = ddi_get_name_addr(pdip);
                for (i = 0; i < MPTSAS_MAX_PHYS; i++) {
                        bzero(phymask, sizeof (phymask));
                        (void) sprintf(phymask,
                            "%x", mpt->m_phy_info[i].phy_mask);
                        if (strcmp(phymask, iport) == 0) {
                                phy_mask = mpt->m_phy_info[i].phy_mask;
                                break;
                        }
                }

                for (i = 0; i < MPTSAS_MAX_PHYS; i++) {
                        if ((phy_mask >> i) & 0x01) {
                                numphys++;
                        }
                }
                /*
                 * Update PHY info for smhba
                 */
                if (mptsas_smhba_phy_init(mpt)) {
                        mutex_exit(&mpt->m_mutex);
                        mptsas_log(mpt, CE_WARN, "mptsas phy update "
                            "failed");
                        goto smp_create_done;
                }
                mutex_exit(&mpt->m_mutex);

                mptsas_smhba_set_all_phy_props(mpt, pdip, numphys, phy_mask,
                    &attached_devhdl);

                if (ddi_prop_update_int(DDI_DEV_T_NONE, pdip,
                    MPTSAS_NUM_PHYS, numphys) !=
                    DDI_PROP_SUCCESS) {
                        (void) ddi_prop_remove(DDI_DEV_T_NONE, pdip,
                            MPTSAS_NUM_PHYS);
                        mptsas_log(mpt, CE_WARN, "mptsas update "
                            "num phys props failed");
                        goto smp_create_done;
                }
                /*
```

```
                 * Add parent's props for SMHBA support
                 */
                if (ddi_prop_update_string(DDI_DEV_T_NONE, pdip,
                    SCSI_ADDR_PROP_ATTACHED_PORT, wwn_str) !=
                    DDI_PROP_SUCCESS) {
                        (void) ddi_prop_remove(DDI_DEV_T_NONE, pdip,
                            SCSI_ADDR_PROP_ATTACHED_PORT);
                        mptsas_log(mpt, CE_WARN, "mptsas update iport"
                            "attached-port failed");
                        goto smp_create_done;
                }

smp_create_done:
                /*
                 * If props were setup ok, online the lun
                 */
                if (ndi_rtn == NDI_SUCCESS) {
                        /*
                         * Try to online the new node
                         */
                        ndi_rtn = ndi_devi_online(*smp_dip, NDI_ONLINE_ATTACH);
                }

                /*
                 * If success set rtn flag, else unwire alloc'd lun
                 */
                if (ndi_rtn != NDI_SUCCESS) {
                        NDBG12(("mptsas unable to online "
                            "SMP target %s", wwn_str));
                        ndi_prop_remove_all(*smp_dip);
                        (void) ndi_devi_free(*smp_dip);
                }
        }

        return ((ndi_rtn == NDI_SUCCESS) ? DDI_SUCCESS : DDI_FAILURE);
}

/* smp transport routine */
static int mptsas_smp_start(struct smp_pkt *smp_pkt)
{
        uint64_t                        wwn;
        Mpi2SmpPassthroughRequest_t     req;
        Mpi2SmpPassthroughReply_t       rep;
        uint8_t                         direction = 0;
        mptsas_t                        *mpt;
        int                             ret;
        uint64_t                        tmp64;

        mpt = (mptsas_t *)smp_pkt->smp_pkt_address->
            smp_a_hba_tran->smp_tran_hba_private;

        bcopy(smp_pkt->smp_pkt_address->smp_a_wwn, &wwn, SAS_WWN_BYTE_SIZE);
        /*
         * Need to compose a SMP request message
         * and call mptsas_do_passthru() function
         */
        bzero(&req, sizeof (req));
        bzero(&rep, sizeof (rep));
        req.PassthroughFlags = 0;
        req.PhysicalPort = 0xff;
        req.ChainOffset = 0;
        req.Function = MPI2_FUNCTION_SMP_PASSTHROUGH;

        if ((smp_pkt->smp_pkt_reqsize & 0xffff0000ul) != 0) {
                smp_pkt->smp_pkt_reason = ERANGE;
                return (DDI_FAILURE);
```



```
        }
        req.RequestDataLength = LE_16((uint16_t)(smp_pkt->smp_pkt_reqsize - 4));

        req.MsgFlags = 0;
        tmp64 = LE_64(wwn);
        bcopy(&tmp64, &req.SASAddress, SAS_WWN_BYTE_SIZE);
        if (smp_pkt->smp_pkt_rspsize > 0) {
                direction |= MPTSAS_PASS_THRU_DIRECTION_READ;
        }
        if (smp_pkt->smp_pkt_reqsize > 0) {
                direction |= MPTSAS_PASS_THRU_DIRECTION_WRITE;
        }

        mutex_enter(&mpt->m_mutex);
        ret = mptsas_do_passthru(mpt, (uint8_t *)&req, (uint8_t *)&rep,
            (uint8_t *)smp_pkt->smp_pkt_rsp,
            offsetof(Mpi2SmpPassthroughRequest_t, SGL), sizeof (rep),
            smp_pkt->smp_pkt_rspsize - 4, direction,
            (uint8_t *)smp_pkt->smp_pkt_req, smp_pkt->smp_pkt_reqsize - 4,
            smp_pkt->smp_pkt_timeout, FKIOCTL);
        mutex_exit(&mpt->m_mutex);
        if (ret != 0) {
                cmn_err(CE_WARN, "smp_start do passthru error %d", ret);
                smp_pkt->smp_pkt_reason = (uchar_t)(ret);
                return (DDI_FAILURE);
        }
        /* do passthrough success, check the smp status */
        if (LE_16(rep.IOCStatus) != MPI2_IOCSTATUS_SUCCESS) {
                switch (LE_16(rep.IOCStatus)) {
                case MPI2_IOCSTATUS_SCSI_DEVICE_NOT_THERE:
                        smp_pkt->smp_pkt_reason = ENODEV;
                        break;
                case MPI2_IOCSTATUS_SAS_SMP_DATA_OVERRUN:
                        smp_pkt->smp_pkt_reason = EOVERFLOW;
                        break;
                case MPI2_IOCSTATUS_SAS_SMP_REQUEST_FAILED:
                        smp_pkt->smp_pkt_reason = EIO;
                        break;
                default:
                        mptsas_log(mpt, CE_NOTE, "smp_start: get unknown ioc"
                            "status:%x", LE_16(rep.IOCStatus));
                        smp_pkt->smp_pkt_reason = EIO;
                        break;
                }
                return (DDI_FAILURE);
        }
        if (rep.SASStatus != MPI2_SASSTATUS_SUCCESS) {
                mptsas_log(mpt, CE_NOTE, "smp_start: get error SAS status:%x",
                    rep.SASStatus);
                smp_pkt->smp_pkt_reason = EIO;
                return (DDI_FAILURE);
        }

        return (DDI_SUCCESS);
}

/*
 * If we didn't get a match, we need to get sas page0 for each device, and
 * untill we get a match. If failed, return NULL
 */
static mptsas_target_t *
mptsas_phy_to_tgt(mptsas_t *mpt, mptsas_phymask_t phymask, uint8_t phy)
{
        int             i, j = 0;
        int             rval = 0;
        uint16_t        cur_handle;
```

```
        uint32_t        page_address;
        mptsas_target_t *ptgt = NULL;

        /*
         * PHY named device must be direct attached and attaches to
         * narrow port, if the iport is not parent of the device which
         * we are looking for.
         */
        for (i = 0; i < MPTSAS_MAX_PHYS; i++) {
                if ((1 << i) & phymask)
                        j++;
        }

        if (j > 1)
                return (NULL);

        /*
         * Must be a narrow port and single device attached to the narrow port
         * So the physical port num of device  which is equal to the iport's
         * port num is the device what we are looking for.
         */

        if (mpt->m_phy_info[phy].phy_mask != phymask)
                return (NULL);

        mutex_enter(&mpt->m_mutex);

        ptgt = refhash_linear_search(mpt->m_targets, mptsas_target_eval_nowwn,
            &phy);
        if (ptgt != NULL) {
                mutex_exit(&mpt->m_mutex);
                return (ptgt);
        }

        if (mpt->m_done_traverse_dev) {
                mutex_exit(&mpt->m_mutex);
                return (NULL);
        }

        /* If didn't get a match, come here */
        cur_handle = mpt->m_dev_handle;
        for (; ; ) {
                ptgt = NULL;
                page_address = (MPI2_SAS_DEVICE_PGAD_FORM_GET_NEXT_HANDLE &
                    MPI2_SAS_DEVICE_PGAD_FORM_MASK) | (uint32_t)cur_handle;
                rval = mptsas_get_target_device_info(mpt, page_address,
                    &cur_handle, &ptgt);
                if ((rval == DEV_INFO_FAIL_PAGE0) ||
                    (rval == DEV_INFO_FAIL_ALLOC)) {
                        break;
                }
                if ((rval == DEV_INFO_WRONG_DEVICE_TYPE) ||
                    (rval == DEV_INFO_PHYS_DISK)) {
                        continue;
                }
                mpt->m_dev_handle = cur_handle;

                if ((ptgt->m_addr.mta_wwn == 0) && (ptgt->m_phynum == phy)) {
                        break;
                }
        }

        mutex_exit(&mpt->m_mutex);
        return (ptgt);
}
```



```
/*
 * The ptgt->m_addr.mta_wwn contains the wwid for each disk.
 * For Raid volumes, we need to check m_raidvol[x].m_raidwwid
 * If we didn't get a match, we need to get sas page0 for each device, and
 * untill we get a match
 * If failed, return NULL
 */
static mptsas_target_t *
mptsas_wwid_to_ptgt(mptsas_t *mpt, mptsas_phymask_t phymask, uint64_t wwid)
{
        int             rval = 0;
        uint16_t        cur_handle;
        uint32_t        page_address;
        mptsas_target_t *tmp_tgt = NULL;
        mptsas_target_addr_t addr;

        addr.mta_wwn = wwid;
        addr.mta_phymask = phymask;
        mutex_enter(&mpt->m_mutex);
        tmp_tgt = refhash_lookup(mpt->m_targets, &addr);
        if (tmp_tgt != NULL) {
                mutex_exit(&mpt->m_mutex);
                return (tmp_tgt);
        }

        if (phymask == 0) {
                /*
                 * It's IR volume
                 */
                rval = mptsas_get_raid_info(mpt);
                if (rval) {
                        tmp_tgt = refhash_lookup(mpt->m_targets, &addr);
                }
                mutex_exit(&mpt->m_mutex);
                return (tmp_tgt);
        }

        if (mpt->m_done_traverse_dev) {
                mutex_exit(&mpt->m_mutex);
                return (NULL);
        }

        /* If didn't get a match, come here */
        cur_handle = mpt->m_dev_handle;
        for (;;) {
                tmp_tgt = NULL;
                page_address = (MPI2_SAS_DEVICE_PGAD_FORM_GET_NEXT_HANDLE &
                    MPI2_SAS_DEVICE_PGAD_FORM_MASK) | cur_handle;
                rval = mptsas_get_target_device_info(mpt, page_address,
                    &cur_handle, &tmp_tgt);
                if ((rval == DEV_INFO_FAIL_PAGE0) ||
                    (rval == DEV_INFO_FAIL_ALLOC)) {
                        tmp_tgt = NULL;
                        break;
                }
                if ((rval == DEV_INFO_WRONG_DEVICE_TYPE) ||
                    (rval == DEV_INFO_PHYS_DISK)) {
                        continue;
                }
                mpt->m_dev_handle = cur_handle;
                if ((tmp_tgt->m_addr.mta_wwn) &&
                    (tmp_tgt->m_addr.mta_wwn == wwid) &&
                    (tmp_tgt->m_addr.mta_phymask == phymask)) {
                        break;
                }
        }
```

```
        mutex_exit(&mpt->m_mutex);
        return (tmp_tgt);
}

static mptsas_smp_t *
mptsas_wwid_to_psmp(mptsas_t *mpt, mptsas_phymask_t phymask, uint64_t wwid)
{
        int             rval = 0;
        uint16_t        cur_handle;
        uint32_t        page_address;
        mptsas_smp_t    smp_node, *psmp = NULL;
        mptsas_target_addr_t addr;

        addr.mta_wwn = wwid;
        addr.mta_phymask = phymask;
        mutex_enter(&mpt->m_mutex);
        psmp = refhash_lookup(mpt->m_smp_targets, &addr);
        if (psmp != NULL) {
                mutex_exit(&mpt->m_mutex);
                return (psmp);
        }

        if (mpt->m_done_traverse_smp) {
                mutex_exit(&mpt->m_mutex);
                return (NULL);
        }

        /* If didn't get a match, come here */
        cur_handle = mpt->m_smp_devhdl;
        for (;;) {
                psmp = NULL;
                page_address = (MPI2_SAS_EXPAND_PGAD_FORM_GET_NEXT_HNDL &
                    MPI2_SAS_EXPAND_PGAD_FORM_MASK) | (uint32_t)cur_handle;
                rval = mptsas_get_sas_expander_page0(mpt, page_address,
                    &smp_node);
                if (rval != DDI_SUCCESS) {
                        break;
                }
                mpt->m_smp_devhdl = cur_handle = smp_node.m_devhdl;
                psmp = mptsas_smp_alloc(mpt, &smp_node);
                ASSERT(psmp);
                if ((psmp->m_addr.mta_wwn) && (psmp->m_addr.mta_wwn == wwid) &&
                    (psmp->m_addr.mta_phymask == phymask)) {
                        break;
                }
        }

        mutex_exit(&mpt->m_mutex);
        return (psmp);
}

mptsas_target_t *
mptsas_tgt_alloc(mptsas_t *mpt, uint16_t devhdl, uint64_t wwid,
    uint32_t devinfo, mptsas_phymask_t phymask, uint8_t phynum)
{
        mptsas_target_t *tmp_tgt = NULL;
        mptsas_target_addr_t addr;

        addr.mta_wwn = wwid;
        addr.mta_phymask = phymask;
        tmp_tgt = refhash_lookup(mpt->m_targets, &addr);
        if (tmp_tgt != NULL) {
                NDBG20(("Hash item already exist"));
                tmp_tgt->m_deviceinfo = devinfo;
                tmp_tgt->m_devhdl = devhdl;     /* XXX - duplicate? */
```



```
                return (tmp_tgt);
        }
        tmp_tgt = kmem_zalloc(sizeof (struct mptsas_target), KM_SLEEP);
        if (tmp_tgt == NULL) {
                cmn_err(CE_WARN, "Fatal, allocated tgt failed");
                return (NULL);
        }
        tmp_tgt->m_devhdl = devhdl;
        tmp_tgt->m_addr.mta_wwn = wwid;
        tmp_tgt->m_deviceinfo = devinfo;
        tmp_tgt->m_addr.mta_phymask = phymask;
        tmp_tgt->m_phynum = phynum;
        /* Initialized the tgt structure */
        tmp_tgt->m_qfull_retries = QFULL_RETRIES;
        tmp_tgt->m_qfull_retry_interval =
            drv_usectohz(QFULL_RETRY_INTERVAL * 1000);
        tmp_tgt->m_t_throttle = MAX_THROTTLE;
        mutex_init(&tmp_tgt->m_t_mutex, NULL, MUTEX_DRIVER, NULL);
        TAILQ_INIT(&tmp_tgt->m_active_cmdq);

        refhash_insert(mpt->m_targets, tmp_tgt);

        return (tmp_tgt);
}

static void
mptsas_smp_target_copy(mptsas_smp_t *src, mptsas_smp_t *dst)
{
        dst->m_devhdl = src->m_devhdl;
        dst->m_deviceinfo = src->m_deviceinfo;
        dst->m_pdevhdl = src->m_pdevhdl;
        dst->m_pdevinfo = src->m_pdevinfo;
}

static mptsas_smp_t *
mptsas_smp_alloc(mptsas_t *mpt, mptsas_smp_t *data)
{
        mptsas_target_addr_t addr;
        mptsas_smp_t *ret_data;

        addr.mta_wwn = data->m_addr.mta_wwn;
        addr.mta_phymask = data->m_addr.mta_phymask;
        ret_data = refhash_lookup(mpt->m_smp_targets, &addr);
        /*
         * If there's already a matching SMP target, update its fields
         * in place.  Since the address is not changing, it's safe to do
         * this.  We cannot just bcopy() here because the structure we've
         * been given has invalid hash links.
         */
        if (ret_data != NULL) {
                mptsas_smp_target_copy(data, ret_data);
                return (ret_data);
        }

        ret_data = kmem_alloc(sizeof (mptsas_smp_t), KM_SLEEP);
        bcopy(data, ret_data, sizeof (mptsas_smp_t));
        refhash_insert(mpt->m_smp_targets, ret_data);
        return (ret_data);
}

/*
 * Functions for SGPIO LED support
 */
static dev_info_t *
mptsas_get_dip_from_dev(dev_t dev, mptsas_phymask_t *phymask)
{
```

```
        dev_info_t      *dip;
        int             prop;
        dip = e_ddi_hold_devi_by_dev(dev, 0);
        if (dip == NULL)
                return (dip);
        prop = ddi_prop_get_int(DDI_DEV_T_ANY, dip, 0,
            "phymask", 0);
        *phymask = (mptsas_phymask_t)prop;
        ddi_release_devi(dip);
        return (dip);
}
static mptsas_target_t *
mptsas_addr_to_ptgt(mptsas_t *mpt, char *addr, mptsas_phymask_t phymask)
{
        uint8_t                 phynum;
        uint64_t                wwn;
        int                     lun;
        mptsas_target_t         *ptgt = NULL;

        if (mptsas_parse_address(addr, &wwn, &phynum, &lun) != DDI_SUCCESS) {
                return (NULL);
        }
        if (addr[0] == 'w') {
                ptgt = mptsas_wwid_to_ptgt(mpt, (int)phymask, wwn);
        } else {
                ptgt = mptsas_phy_to_tgt(mpt, (int)phymask, phynum);
        }
        return (ptgt);
}

static int
mptsas_flush_led_status(mptsas_t *mpt, mptsas_target_t *ptgt)
{
        uint32_t slotstatus = 0;

        /* Build an MPI2 Slot Status based on our view of the world */
        if (ptgt->m_led_status & (1 << (MPTSAS_LEDCTL_LED_IDENT - 1)))
                slotstatus |= MPI2_SEP_REQ_SLOTSTATUS_IDENTIFY_REQUEST;
        if (ptgt->m_led_status & (1 << (MPTSAS_LEDCTL_LED_FAIL - 1)))
                slotstatus |= MPI2_SEP_REQ_SLOTSTATUS_PREDICTED_FAULT;
        if (ptgt->m_led_status & (1 << (MPTSAS_LEDCTL_LED_OK2RM - 1)))
                slotstatus |= MPI2_SEP_REQ_SLOTSTATUS_REQUEST_REMOVE;

        /* Write it to the controller */
        NDBG14(("mptsas_ioctl: set LED status %x for slot %x",
            slotstatus, ptgt->m_slot_num));
        return (mptsas_send_sep(mpt, ptgt, &slotstatus,
            MPI2_SEP_REQ_ACTION_WRITE_STATUS));
}

/*
 *  send sep request, use enclosure/slot addressing
 */
static int
mptsas_send_sep(mptsas_t *mpt, mptsas_target_t *ptgt,
    uint32_t *status, uint8_t act)
{
        Mpi2SepRequest_t        req;
        Mpi2SepReply_t          rep;
        int                     ret;

        ASSERT(mutex_owned(&mpt->m_mutex));

        /*
         * We only support SEP control of directly-attached targets, in which
         * case the "SEP" we're talking to is a virtual one contained within
```



```
         * the HBA itself.  This is necessary because DA targets typically have
         * no other mechanism for LED control.  Targets for which a separate
         * enclosure service processor exists should be controlled via ses(7d)
         * or sgen(7d).  Furthermore, since such requests can time out, they
         * should be made in user context rather than in response to
         * asynchronous fabric changes.
         *
         * In addition, we do not support this operation for RAID volumes,
         * since there is no slot associated with them.
         */
        if (!(ptgt->m_deviceinfo & DEVINFO_DIRECT_ATTACHED) ||
            ptgt->m_addr.mta_phymask == 0) {
                return (ENOTTY);
        }

        bzero(&req, sizeof (req));
        bzero(&rep, sizeof (rep));

        req.Function = MPI2_FUNCTION_SCSI_ENCLOSURE_PROCESSOR;
        req.Action = act;
        req.Flags = MPI2_SEP_REQ_FLAGS_ENCLOSURE_SLOT_ADDRESS;
        req.EnclosureHandle = LE_16(ptgt->m_enclosure);
        req.Slot = LE_16(ptgt->m_slot_num);
        if (act == MPI2_SEP_REQ_ACTION_WRITE_STATUS) {
                req.SlotStatus = LE_32(*status);
        }
        ret = mptsas_do_passthru(mpt, (uint8_t *)&req, (uint8_t *)&rep, NULL,
            sizeof (req), sizeof (rep), NULL, 0, NULL, 0, 60, FKIOCTL);
        if (ret != 0) {
                mptsas_log(mpt, CE_NOTE, "mptsas_send_sep: passthru SEP "
                    "Processor Request message error %d", ret);
                return (ret);
        }
        /* do passthrough success, check the ioc status */
        if (LE_16(rep.IOCStatus) != MPI2_IOCSTATUS_SUCCESS) {
                mptsas_log(mpt, CE_NOTE, "send_sep act %x: ioc "
                    "status:%x loginfo %x", act, LE_16(rep.IOCStatus),
                    LE_32(rep.IOCLogInfo));
                switch (LE_16(rep.IOCStatus) & MPI2_IOCSTATUS_MASK) {
                case MPI2_IOCSTATUS_INVALID_FUNCTION:
                case MPI2_IOCSTATUS_INVALID_VPID:
                case MPI2_IOCSTATUS_INVALID_FIELD:
                case MPI2_IOCSTATUS_INVALID_STATE:
                case MPI2_IOCSTATUS_OP_STATE_NOT_SUPPORTED:
                case MPI2_IOCSTATUS_CONFIG_INVALID_ACTION:
                case MPI2_IOCSTATUS_CONFIG_INVALID_TYPE:
                case MPI2_IOCSTATUS_CONFIG_INVALID_PAGE:
                case MPI2_IOCSTATUS_CONFIG_INVALID_DATA:
                case MPI2_IOCSTATUS_CONFIG_NO_DEFAULTS:
                        return (EINVAL);
                case MPI2_IOCSTATUS_BUSY:
                        return (EBUSY);
                case MPI2_IOCSTATUS_INSUFFICIENT_RESOURCES:
                        return (EAGAIN);
                case MPI2_IOCSTATUS_INVALID_SGL:
                case MPI2_IOCSTATUS_INTERNAL_ERROR:
                case MPI2_IOCSTATUS_CONFIG_CANT_COMMIT:
                default:
                        return (EIO);
                }
        }
        if (act != MPI2_SEP_REQ_ACTION_WRITE_STATUS) {
                *status = LE_32(rep.SlotStatus);
        }

        return (0);
```

```
}

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)
{
        ddi_dma_cookie_t        new_cookie;
        size_t                  alloc_len;
        uint_t                  ncookie;

        if (cookiep == NULL)
                cookiep = &new_cookie;

        if (ddi_dma_alloc_handle(mpt->m_dip, &dma_attr, DDI_DMA_SLEEP,
            NULL, dma_hdp) != DDI_SUCCESS) {
                return (FALSE);
        }

        if (ddi_dma_mem_alloc(*dma_hdp, alloc_size, &mpt->m_dev_acc_attr,
            DDI_DMA_CONSISTENT, DDI_DMA_SLEEP, NULL, dma_memp, &alloc_len,
            acc_hdp) != DDI_SUCCESS) {
                ddi_dma_free_handle(dma_hdp);
                return (FALSE);
        }

        if (ddi_dma_addr_bind_handle(*dma_hdp, NULL, *dma_memp, alloc_len,
            (DDI_DMA_RDWR | DDI_DMA_CONSISTENT), DDI_DMA_SLEEP, NULL,
            cookiep, &ncookie) != DDI_DMA_MAPPED) {
                (void) ddi_dma_mem_free(acc_hdp);
                ddi_dma_free_handle(dma_hdp);
                return (FALSE);
        }

        return (TRUE);
}

void
mptsas_dma_addr_destroy(ddi_dma_handle_t *dma_hdp, ddi_acc_handle_t *acc_hdp)
{
        if (*dma_hdp == NULL)
                return;

        (void) ddi_dma_unbind_handle(*dma_hdp);
        (void) ddi_dma_mem_free(acc_hdp);
        ddi_dma_free_handle(dma_hdp);
}
#endif /* ! codereview */
```

```
**********************************************************
    4802 Thu Jun 12 17:28:22 2014
new/usr/src/uts/common/io/scsi/adapters/mpt_sas3/mptsas3_hash.c
4546 mpt_sas needs enhancing to support LSI MPI2.5
**********************************************************
/*
 * 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.
 */

#include <sys/scsi/adapters/mpt_sas3/mptsas3_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);
}

void
refhash_destroy(refhash_t *hp)
{
        ASSERT(list_is_empty(&hp->rh_objs));

        kmem_free(hp->rh_buckets, hp->rh_bucket_count * sizeof (list_t));
        kmem_free(hp, sizeof (refhash_t));
}

void
refhash_insert(refhash_t *hp, void *op)
{
        uint_t bucket;
        refhash_link_t *lp = obj_to_link(hp, op);

        bucket = hp->rh_hash(obj_to_tag(hp, op)) % hp->rh_bucket_count;
        list_link_init(&lp->rhl_chain_link);
        list_link_init(&lp->rhl_global_link);
        lp->rhl_flags = 0;
        lp->rhl_refcnt = 0;
        list_insert_tail(&hp->rh_buckets[bucket], lp);
        list_insert_tail(&hp->rh_objs, lp);
}

static void
refhash_delete(refhash_t *hp, void *op)
{
        refhash_link_t *lp = obj_to_link(hp, op);
        uint_t bucket;

        bucket = hp->rh_hash(obj_to_tag(hp, op)) % hp->rh_bucket_count;
        list_remove(&hp->rh_buckets[bucket], lp);
        list_remove(&hp->rh_objs, lp);
        hp->rh_dtor(op);
}

void
refhash_remove(refhash_t *hp, void *op)
{
        refhash_link_t *lp = obj_to_link(hp, op);

        if (lp->rhl_refcnt > 0) {
                lp->rhl_flags |= RHL_F_DEAD;
        } else {
                refhash_delete(hp, op);
        }
}

void *
refhash_lookup(refhash_t *hp, const void *tp)
{
        uint_t bucket;
        refhash_link_t *lp;
        void *op;

        bucket = hp->rh_hash(tp) % hp->rh_bucket_count;
```

```
        for (lp = list_head(&hp->rh_buckets[bucket]); lp != NULL;
            lp = list_next(&hp->rh_buckets[bucket], lp)) {
                op = link_to_obj(hp, lp);
                if (hp->rh_cmp(obj_to_tag(hp, op), tp) == 0 &&
                    !(lp->rhl_flags & RHL_F_DEAD)) {
                        return (op);
                }
        }

        return (NULL);
}

void *
refhash_linear_search(refhash_t *hp, refhash_eval_f eval, void *arg)
{
        void *op;
        refhash_link_t *lp;

        for (lp = list_head(&hp->rh_objs); lp != NULL;
            lp = list_next(&hp->rh_objs, lp)) {
                op = link_to_obj(hp, lp);
                if (eval(op, arg) == 0)
                        return (op);
        }

        return (NULL);
}

void
refhash_hold(refhash_t *hp, void *op)
{
        refhash_link_t *lp = obj_to_link(hp, op);

        ++lp->rhl_refcnt;
}

void
refhash_rele(refhash_t *hp, void *op)
{
        refhash_link_t *lp = obj_to_link(hp, op);

        ASSERT(lp->rhl_refcnt > 0);

        if (--lp->rhl_refcnt == 0 && (lp->rhl_flags & RHL_F_DEAD))
                refhash_remove(hp, op);
}

void *
refhash_first(refhash_t *hp)
{
        refhash_link_t *lp;

        lp = list_head(&hp->rh_objs);
        if (lp == NULL)
                return (NULL);

        ++lp->rhl_refcnt;

        return (link_to_obj(hp, lp));
}

void *
refhash_next(refhash_t *hp, void *op)
{
        refhash_link_t *lp;
```



```
        lp = obj_to_link(hp, op);
        while ((lp = list_next(&hp->rh_objs, lp)) != NULL) {
                if (!(lp->rhl_flags & RHL_F_DEAD))
                        break;
        }

        refhash_rele(hp, op);
        if (lp == NULL)
                return (NULL);

        ++lp->rhl_refcnt;

        return (link_to_obj(hp, lp));
}

boolean_t
refhash_obj_valid(refhash_t *hp, const void *op)
{
        /* LINTED - E_ARG_INCOMPATIBLE_WITH_ARG_L */
        const refhash_link_t *lp = obj_to_link(hp, op);

        return ((lp->rhl_flags & RHL_F_DEAD) != 0);
}
#endif /* ! codereview */
```

```
**********************************************************
   83656 Thu Jun 12 17:28:22 2014
new/usr/src/uts/common/io/scsi/adapters/mpt_sas3/mptsas3_impl.c
4546 mpt_sas needs enhancing to support LSI MPI2.5
**********************************************************
/*
 * 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.
 */

/*
 * 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>
#pragma pack()

/*
 * private header files.
 */
#include <sys/scsi/adapters/mpt_sas3/mptsas3_var.h>
#include <sys/scsi/adapters/mpt_sas3/mptsas3_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

/*
 *  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;

        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);
        direction = MPI2_SGE_FLAGS_IOC_TO_HOST;
        length = 0;
        sge = (pMpi2SGESimple64_t)&request->PageBufferSGE;
        if (config->action == MPI2_CONFIG_ACTION_PAGE_HEADER) {
                if (config->page_type > MPI2_CONFIG_PAGETYPE_MASK) {
                        ddi_put8(mpt->m_acc_req_frame_hdl,
                            &request->Header.PageType,
                            MPI2_CONFIG_PAGETYPE_EXTENDED);
                        ddi_put8(mpt->m_acc_req_frame_hdl,
                            &request->ExtPageType, config->page_type);
                } else {
                        ddi_put8(mpt->m_acc_req_frame_hdl,
                            &request->Header.PageType, config->page_type);
                }
        } else {
                ddi_put8(mpt->m_acc_req_frame_hdl, &request->ExtPageType,
                    config->ext_page_type);
                ddi_put16(mpt->m_acc_req_frame_hdl, &request->ExtPageLength,
                    config->ext_page_length);
                ddi_put8(mpt->m_acc_req_frame_hdl, &request->Header.PageType,
                    config->page_type);
                ddi_put8(mpt->m_acc_req_frame_hdl, &request->Header.PageLength,
                    config->page_length);
                ddi_put8(mpt->m_acc_req_frame_hdl,
                    &request->Header.PageVersion, config->page_version);
                if ((config->page_type & MPI2_CONFIG_PAGETYPE_MASK) ==
                    MPI2_CONFIG_PAGETYPE_EXTENDED) {
                        length = config->ext_page_length * 4;
                } else {
                        length = config->page_length * 4;
```

```
                }

                if (config->action == MPI2_CONFIG_ACTION_PAGE_WRITE_NVRAM) {
                        direction = MPI2_SGE_FLAGS_HOST_TO_IOC;
                }
                ddi_put32(mpt->m_acc_req_frame_hdl, &sge->Address.Low,
                    (uint32_t)(cmd->cmd_dma_addr&0xffffffffful));
                ddi_put32(mpt->m_acc_req_frame_hdl, &sge->Address.High,
                    (uint32_t)(cmd->cmd_dma_addr >> 32));
        }
        ddi_put8(mpt->m_acc_req_frame_hdl, &request->Header.PageNumber,
            config->page_number);
        ddi_put32(mpt->m_acc_req_frame_hdl, &request->PageAddress,
            config->page_address);
        flagslength = ((uint32_t)(MPI2_SGE_FLAGS_LAST_ELEMENT |
            MPI2_SGE_FLAGS_END_OF_BUFFER |
            MPI2_SGE_FLAGS_SIMPLE_ELEMENT |
            MPI2_SGE_FLAGS_SYSTEM_ADDRESS |
            MPI2_SGE_FLAGS_64_BIT_ADDRESSING |
            direction |
            MPI2_SGE_FLAGS_END_OF_LIST) << MPI2_SGE_FLAGS_SHIFT);
        flagslength |= length;
        ddi_put32(mpt->m_acc_req_frame_hdl, &sge->FlagsLength, flagslength);

        (void) ddi_dma_sync(mpt->m_dma_req_frame_hdl, 0, 0,
            DDI_DMA_SYNC_FORDEV);
        request_desc = (cmd->cmd_slot << 16) +
            MPI2_REQ_DESCRIPT_FLAGS_DEFAULT_TYPE;
        cmd->cmd_rfm = NULL;
        MPTSAS_START_CMD(mpt, request_desc);
        if ((mptsas_check_dma_handle(mpt->m_dma_req_frame_hdl) !=
            DDI_SUCCESS) ||
            (mptsas_check_acc_handle(mpt->m_acc_req_frame_hdl) !=
            DDI_SUCCESS)) {
                ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_UNAFFECTED);
        }
}

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), ...)
{
        va_list                 ap;
        ddi_dma_attr_t          attrs;
        ddi_dma_cookie_t        cookie;
        ddi_acc_handle_t        accessp;
        size_t                  len = 0;
        mptsas_config_request_t config;
        int                     rval = DDI_SUCCESS, config_flags = 0;
        mptsas_cmd_t            *cmd;
        struct scsi_pkt         *pkt;
        pMpi2ConfigReply_t      reply;
        uint16_t                iocstatus = 0;
        uint32_t                iocloginfo;
        caddr_t                 page_memp;
        boolean_t               free_dma = B_FALSE;

        va_start(ap, callback);
        ASSERT(mutex_owned(&mpt->m_mutex));

        /*
         * Get a command from the pool.
         */
        if ((rval = (mptsas_request_from_pool(mpt, &cmd, &pkt))) == -1) {
                mptsas_log(mpt, CE_NOTE, "command pool is full for config "
```



```
                    "page request");
                rval = DDI_FAILURE;
                goto page_done;
        }
        config_flags |= MPTSAS_REQUEST_POOL_CMD;

        bzero((caddr_t)cmd, sizeof (*cmd));
        bzero((caddr_t)pkt, scsi_pkt_size());
        bzero((caddr_t)&config, sizeof (config));

        /*
         * Save the data for this request to be used in the call to start the
         * config header request.
         */
        config.action = MPI2_CONFIG_ACTION_PAGE_HEADER;
        config.page_type = page_type;
        config.page_number = page_number;
        config.page_address = page_address;

        /*
         * Form a blank cmd/pkt to store the acknowledgement message
         */
        pkt->pkt_ha_private     = (opaque_t)&config;
        pkt->pkt_flags          = FLAG_HEAD;
        pkt->pkt_time           = 60;
        cmd->cmd_pkt            = pkt;
        cmd->cmd_flags          = CFLAG_CMDIOC | CFLAG_CONFIG;

        /*
         * Save the config header request message in a slot.
         */
        if (mptsas_save_cmd(mpt, cmd) == TRUE) {
                cmd->cmd_flags |= CFLAG_PREPARED;
                mptsas_start_config_page_access(mpt, cmd);
        } else {
                mptsas_waitq_add(mpt, cmd);
        }

        /*
         * If this is a request for a RAID info page, or any page called during
         * the RAID info page request, poll because these config page requests
         * are nested.  Poll to avoid data corruption due to one page's data
         * overwriting the outer page request's data.  This can happen when
         * the mutex is released in cv_wait.
         */
        if ((page_type == MPI2_CONFIG_EXTPAGETYPE_RAID_CONFIG) ||
            (page_type == MPI2_CONFIG_PAGETYPE_RAID_VOLUME) ||
            (page_type == MPI2_CONFIG_PAGETYPE_RAID_PHYSDISK)) {
                (void) mptsas_poll(mpt, cmd, pkt->pkt_time * 1000);
        } else {
                while ((cmd->cmd_flags & CFLAG_FINISHED) == 0) {
                        cv_wait(&mpt->m_config_cv, &mpt->m_mutex);
                }
        }

        /*
         * Check if the header request completed without timing out
         */
        if (cmd->cmd_flags & CFLAG_TIMEOUT) {
                mptsas_log(mpt, CE_WARN, "config header request timeout");
                rval = DDI_FAILURE;
                goto page_done;
        }

        /*
         * cmd_rfm points to the reply message if a reply was given.  Check the
```

```
 * IOCStatus to make sure everything went OK with the header request.
 */
if (cmd->cmd_rfm) {
        config_flags |= MPTSAS_ADDRESS_REPLY;
        (void) ddi_dma_sync(mpt->m_dma_reply_frame_hdl, 0, 0,
            DDI_DMA_SYNC_FORCPU);
        reply = (pMpi2ConfigReply_t)(mpt->m_reply_frame + (cmd->cmd_rfm
            - (mpt->m_reply_frame_dma_addr&0xfffffffful)));
        config.page_type = ddi_get8(mpt->m_acc_reply_frame_hdl,
            &reply->Header.PageType);
        config.page_number = ddi_get8(mpt->m_acc_reply_frame_hdl,
            &reply->Header.PageNumber);
        config.page_length = ddi_get8(mpt->m_acc_reply_frame_hdl,
            &reply->Header.PageLength);
        config.page_version = ddi_get8(mpt->m_acc_reply_frame_hdl,
            &reply->Header.PageVersion);
        config.ext_page_type = ddi_get8(mpt->m_acc_reply_frame_hdl,
            &reply->ExtPageType);
        config.ext_page_length = ddi_get16(mpt->m_acc_reply_frame_hdl,
            &reply->ExtPageLength);

        iocstatus = ddi_get16(mpt->m_acc_reply_frame_hdl,
            &reply->IOCStatus);
        iocloginfo = ddi_get32(mpt->m_acc_reply_frame_hdl,
            &reply->IOCLogInfo);

        if (iocstatus) {
                NDBG13(("mptsas_access_config_page header: "
                    "IOCStatus=0x%x, IOCLogInfo=0x%x", iocstatus,
                    iocloginfo));
                rval = DDI_FAILURE;
                goto page_done;
        }

        if ((config.page_type & MPI2_CONFIG_PAGETYPE_MASK) ==
            MPI2_CONFIG_PAGETYPE_EXTENDED)
                len = (config.ext_page_length * 4);
        else
                len = (config.page_length * 4);

}

if (pkt->pkt_reason == CMD_RESET) {
        mptsas_log(mpt, CE_WARN, "ioc reset abort config header "
            "request");
        rval = DDI_FAILURE;
        goto page_done;
}

/*
 * Put the reply frame back on the free queue, increment the free
 * index, and write the new index to the free index register.  But only
 * if this reply is an ADDRESS reply.
 */
if (config_flags & MPTSAS_ADDRESS_REPLY) {
        ddi_put32(mpt->m_acc_free_queue_hdl,
            &((uint32_t *)(void *)mpt->m_free_queue)[mpt->m_free_index],
            cmd->cmd_rfm);
        (void) ddi_dma_sync(mpt->m_dma_free_queue_hdl, 0, 0,
            DDI_DMA_SYNC_FORDEV);
        if (++mpt->m_free_index == mpt->m_free_queue_depth) {
                mpt->m_free_index = 0;
        }
        ddi_put32(mpt->m_datap, &mpt->m_reg->ReplyFreeHostIndex,
            mpt->m_free_index);
        config_flags &= (~MPTSAS_ADDRESS_REPLY);
```



```
}

/*
 * Allocate DMA buffer here.  Store the info regarding this buffer in
 * the cmd struct so that it can be used for this specific command and
 * de-allocated after the command completes.  The size of the reply
 * will not be larger than the reply frame size.
 */
attrs = mpt->m_msg_dma_attr;
attrs.dma_attr_sgllen = 1;
attrs.dma_attr_granular = (uint32_t)len;

if (mptsas_dma_addr_create(mpt, attrs,
    &cmd->cmd_dmahandle, &accessp, &page_memp,
    len, &cookie) == FALSE) {
        mptsas_log(mpt, CE_WARN,
            "mptsas_dma_addr_create(len=0x%x) failed", (int)len);
        rval = DDI_FAILURE;
        goto page_done;
}
/* NOW we can safely call mptsas_dma_addr_destroy(). */
free_dma = B_TRUE;

cmd->cmd_dma_addr = cookie.dmac_laddress;
bzero(page_memp, len);

/*
 * Save the data for this request to be used in the call to start the
 * config page read
 */
config.action = action;
config.page_address = page_address;

/*
 * Re-use the cmd that was used to get the header.  Reset some of the
 * values.
 */
bzero((caddr_t)pkt, scsi_pkt_size());
pkt->pkt_ha_private     = (opaque_t)&config;
pkt->pkt_flags          = FLAG_HEAD;
pkt->pkt_time           = 60;
cmd->cmd_flags          = CFLAG_PREPARED | CFLAG_CMDIOC | CFLAG_CONFIG;

/*
 * Send the config page request.  cmd is re-used from header request.
 */
mptsas_start_config_page_access(mpt, cmd);

/*
 * If this is a request for a RAID info page, or any page called during
 * the RAID info page request, poll because these config page requests
 * are nested.  Poll to avoid data corruption due to one page's data
 * overwriting the outer page request's data.  This can happen when
 * the mutex is released in cv_wait.
 */
if ((page_type == MPI2_CONFIG_EXTPAGETYPE_RAID_CONFIG) ||
    (page_type == MPI2_CONFIG_PAGETYPE_RAID_VOLUME) ||
    (page_type == MPI2_CONFIG_PAGETYPE_RAID_PHYSDISK)) {
        (void) mptsas_poll(mpt, cmd, pkt->pkt_time * 1000);
} else {
        while ((cmd->cmd_flags & CFLAG_FINISHED) == 0) {
                cv_wait(&mpt->m_config_cv, &mpt->m_mutex);
        }
}

/*
```

```
         * Check if the request completed without timing out
         */
        if (cmd->cmd_flags & CFLAG_TIMEOUT) {
                mptsas_log(mpt, CE_WARN, "config page request timeout");
                rval = DDI_FAILURE;
                goto page_done;
        }

        /*
         * cmd_rfm points to the reply message if a reply was given.  The reply
         * frame and the config page are returned from this function in the
         * param list.
         */
        if (cmd->cmd_rfm) {
                config_flags |= MPTSAS_ADDRESS_REPLY;
                (void) ddi_dma_sync(mpt->m_dma_reply_frame_hdl, 0, 0,
                    DDI_DMA_SYNC_FORCPU);
                (void) ddi_dma_sync(cmd->cmd_dmahandle, 0, 0,
                    DDI_DMA_SYNC_FORCPU);
                reply = (pMpi2ConfigReply_t)(mpt->m_reply_frame + (cmd->cmd_rfm
                    - (mpt->m_reply_frame_dma_addr&0xffffffffu)));
                iocstatus = ddi_get16(mpt->m_acc_reply_frame_hdl,
                    &reply->IOCStatus);
                iocstatus = MPTSAS_IOCSTATUS(iocstatus);
                iocloginfo = ddi_get32(mpt->m_acc_reply_frame_hdl,
                    &reply->IOCLogInfo);
        }

        if (callback(mpt, page_memp, accessp, iocstatus, iocloginfo, ap)) {
                rval = DDI_FAILURE;
                goto page_done;
        }

        mptsas_fma_check(mpt, cmd);
        /*
         * Check the DMA/ACC handles and then free the DMA buffer.
         */
        if ((mptsas_check_dma_handle(cmd->cmd_dmahandle) != DDI_SUCCESS) ||
            (mptsas_check_acc_handle(accessp) != DDI_SUCCESS)) {
                ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_UNAFFECTED);
                rval = DDI_FAILURE;
        }

        if (pkt->pkt_reason == CMD_TRAN_ERR) {
                mptsas_log(mpt, CE_WARN, "config fma error");
                rval = DDI_FAILURE;
                goto page_done;
        }
        if (pkt->pkt_reason == CMD_RESET) {
                mptsas_log(mpt, CE_WARN, "ioc reset abort config request");
                rval = DDI_FAILURE;
                goto page_done;
        }

page_done:
        va_end(ap);
        /*
         * Put the reply frame back on the free queue, increment the free
         * index, and write the new index to the free index register.  But only
         * if this reply is an ADDRESS reply.
         */
        if (config_flags & MPTSAS_ADDRESS_REPLY) {
                ddi_put32(mpt->m_acc_free_queue_hdl,
                    &((uint32_t *)(void *)mpt->m_free_queue)[mpt->m_free_index],
                    cmd->cmd_rfm);
                (void) ddi_dma_sync(mpt->m_dma_free_queue_hdl, 0, 0,
```



```
                    DDI_DMA_SYNC_FORDEV);
                if (++mpt->m_free_index == mpt->m_free_queue_depth) {
                        mpt->m_free_index = 0;
                }
                ddi_put32(mpt->m_datap, &mpt->m_reg->ReplyFreeHostIndex,
                    mpt->m_free_index);
        }

        if (free_dma)
                mptsas_dma_addr_destroy(&cmd->cmd_dmahandle, &accessp);

        if (cmd && (cmd->cmd_flags & CFLAG_PREPARED)) {
                mptsas_remove_cmd(mpt, cmd);
                config_flags &= (~MPTSAS_REQUEST_POOL_CMD);
        }
        if (config_flags & MPTSAS_REQUEST_POOL_CMD)
                mptsas_return_to_pool(mpt, cmd);

        if (config_flags & MPTSAS_CMD_TIMEOUT) {
                mpt->m_softstate &= ~MPTSAS_SS_MSG_UNIT_RESET;
                if ((mptsas_restart_ioc(mpt)) == DDI_FAILURE) {
                        mptsas_log(mpt, CE_WARN, "mptsas_restart_ioc failed");
                }
        }

        return (rval);
}

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)
{
        pMpi2ConfigRequest_t    config;
        int                     send_numbytes;

        bzero(mpt->m_hshk_memp, sizeof (MPI2_CONFIG_REQUEST));
        config = (pMpi2ConfigRequest_t)mpt->m_hshk_memp;
        ddi_put8(mpt->m_hshk_acc_hdl, &config->Function, MPI2_FUNCTION_CONFIG);
        ddi_put8(mpt->m_hshk_acc_hdl, &config->Action, action);
        ddi_put8(mpt->m_hshk_acc_hdl, &config->Header.PageNumber, pagenumber);
        ddi_put8(mpt->m_hshk_acc_hdl, &config->Header.PageType, pagetype);
        ddi_put32(mpt->m_hshk_acc_hdl, &config->PageAddress, pageaddress);
        ddi_put8(mpt->m_hshk_acc_hdl, &config->Header.PageVersion, pageversion);
        ddi_put8(mpt->m_hshk_acc_hdl, &config->Header.PageLength, pagelength);
        ddi_put32(mpt->m_hshk_acc_hdl,
            &config->PageBufferSGE.MpiSimple.FlagsLength, SGEflagslength);
        ddi_put32(mpt->m_hshk_acc_hdl,
            &config->PageBufferSGE.MpiSimple.u.Address64.Low,
            SGEaddress&0xffffffffu);
        ddi_put32(mpt->m_hshk_acc_hdl,
            &config->PageBufferSGE.MpiSimple.u.Address64.High,
            SGEaddress >> 32);
        send_numbytes = sizeof (MPI2_CONFIG_REQUEST);

        /*
         * Post message via handshake
         */
        if (mptsas_send_handshake_msg(mpt, (caddr_t)config, send_numbytes,
            mpt->m_hshk_acc_hdl)) {
                return (-1);
        }
        return (0);
}

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)
{
        pMpi2ConfigRequest_t    config;
        int                     send_numbytes;

        bzero(mpt->m_hshk_memp, sizeof (MPI2_CONFIG_REQUEST));
        config = (pMpi2ConfigRequest_t)mpt->m_hshk_memp;
        ddi_put8(mpt->m_hshk_acc_hdl, &config->Function, MPI2_FUNCTION_CONFIG);
        ddi_put8(mpt->m_hshk_acc_hdl, &config->Action, action);
        ddi_put8(mpt->m_hshk_acc_hdl, &config->Header.PageNumber, pagenumber);
        ddi_put8(mpt->m_hshk_acc_hdl, &config->Header.PageType,
            MPI2_CONFIG_PAGETYPE_EXTENDED);
        ddi_put8(mpt->m_hshk_acc_hdl, &config->ExtPageType, extpagetype);
        ddi_put32(mpt->m_hshk_acc_hdl, &config->PageAddress, pageaddress);
        ddi_put8(mpt->m_hshk_acc_hdl, &config->Header.PageVersion, pageversion);
        ddi_put16(mpt->m_hshk_acc_hdl, &config->ExtPageLength, extpagelength);
        ddi_put32(mpt->m_hshk_acc_hdl,
            &config->PageBufferSGE.MpiSimple.FlagsLength, SGEflagslength);
        ddi_put32(mpt->m_hshk_acc_hdl,
            &config->PageBufferSGE.MpiSimple.u.Address64.Low,
            SGEaddress&0xfffffffful);
        ddi_put32(mpt->m_hshk_acc_hdl,
            &config->PageBufferSGE.MpiSimple.u.Address64.High,
            SGEaddress >> 32);
        send_numbytes = sizeof (MPI2_CONFIG_REQUEST);

        /*
         * Post message via handshake
         */
        if (mptsas_send_handshake_msg(mpt, (caddr_t)config, send_numbytes,
            mpt->m_hshk_acc_hdl)) {
                return (-1);
        }
        return (0);
}

int
mptsas_ioc_wait_for_response(mptsas_t *mpt)
{
        int     polls = 0;

        while ((ddi_get32(mpt->m_datap,
            &mpt->m_reg->HostInterruptStatus) & MPI2_HIS_IOP_DOORBELL_STATUS)) {
                drv_usecwait(1000);
                if (polls++ > 60000) {
                        return (-1);
                }
        }
        return (0);
}

int
mptsas_ioc_wait_for_doorbell(mptsas_t *mpt)
{
        int     polls = 0;

        while ((ddi_get32(mpt->m_datap,
            &mpt->m_reg->HostInterruptStatus) & MPI2_HIM_DIM) == 0) {
                drv_usecwait(1000);
                if (polls++ > 300000) {
                        return (-1);
                }
        }
```



```
        return (0);
}

int
mptsas_send_handshake_msg(mptsas_t *mpt, caddr_t memp, int numbytes,
        ddi_acc_handle_t accessp)
{
        int     i;

        /*
         * clean pending doorbells
         */
        ddi_put32(mpt->m_datap, &mpt->m_reg->HostInterruptStatus, 0);
        ddi_put32(mpt->m_datap, &mpt->m_reg->Doorbell,
            ((MPI2_FUNCTION_HANDSHAKE << MPI2_DOORBELL_FUNCTION_SHIFT) |
            ((numbytes / 4) << MPI2_DOORBELL_ADD_DWORDS_SHIFT)));

        if (mptsas_ioc_wait_for_doorbell(mpt)) {
                NDBG19(("mptsas_send_handshake failed.  Doorbell not ready\n"));
                return (-1);
        }

        /*
         * clean pending doorbells again
         */
        ddi_put32(mpt->m_datap, &mpt->m_reg->HostInterruptStatus, 0);

        if (mptsas_ioc_wait_for_response(mpt)) {
                NDBG19(("mptsas_send_handshake failed.  Doorbell not "
                    "cleared\n"));
                return (-1);
        }

        /*
         * post handshake message
         */
        for (i = 0; (i < numbytes / 4); i++, memp += 4) {
                ddi_put32(mpt->m_datap, &mpt->m_reg->Doorbell,
                    ddi_get32(accessp, (uint32_t *)((void *)(memp))));
                if (mptsas_ioc_wait_for_response(mpt)) {
                        NDBG19(("mptsas_send_handshake failed posting "
                            "message\n"));
                        return (-1);
                }
        }

        if (mptsas_check_acc_handle(mpt->m_datap) != DDI_SUCCESS) {
                ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_UNAFFECTED);
                ddi_fm_acc_err_clear(mpt->m_datap, DDI_FME_VER0);
                return (-1);
        }

        return (0);
}

int
mptsas_get_handshake_msg(mptsas_t *mpt, caddr_t memp, int numbytes,
        ddi_acc_handle_t accessp)
{
        int             i, totalbytes, bytesleft;
        uint16_t        val;

        /*
         * wait for doorbell
         */
        if (mptsas_ioc_wait_for_doorbell(mpt)) {
```

```
                NDBG19(("mptsas_get_handshake failed.  Doorbell not ready\n"));
                return (-1);
        }

        /*
         * get first 2 bytes of handshake message to determine how much
         * data we will be getting
         */
        for (i = 0; i < 2; i++, memp += 2) {
                val = (ddi_get32(mpt->m_datap,
                    &mpt->m_reg->Doorbell) & MPI2_DOORBELL_DATA_MASK);
                ddi_put32(mpt->m_datap, &mpt->m_reg->HostInterruptStatus, 0);
                if (mptsas_ioc_wait_for_doorbell(mpt)) {
                        NDBG19(("mptsas_get_handshake failure getting initial"
                            " data\n"));
                        return (-1);
                }
                ddi_put16(accessp, (uint16_t *)((void *)(memp)), val);
                if (i == 1) {
                        totalbytes = (val & 0xFF) * 2;
                }
        }

        /*
         * If we are expecting less bytes than the message wants to send
         * we simply save as much as we expected and then throw out the rest
         * later
         */
        if (totalbytes > (numbytes / 2)) {
                bytesleft = ((numbytes / 2) - 2);
        } else {
                bytesleft = (totalbytes - 2);
        }

        /*
         * Get the rest of the data
         */
        for (i = 0; i < bytesleft; i++, memp += 2) {
                val = (ddi_get32(mpt->m_datap,
                    &mpt->m_reg->Doorbell) & MPI2_DOORBELL_DATA_MASK);
                ddi_put32(mpt->m_datap, &mpt->m_reg->HostInterruptStatus, 0);
                if (mptsas_ioc_wait_for_doorbell(mpt)) {
                        NDBG19(("mptsas_get_handshake failure getting"
                            " main data\n"));
                        return (-1);
                }
                ddi_put16(accessp, (uint16_t *)((void *)(memp)), val);
        }

        /*
         * Sometimes the device will send more data than is expected
         * This data is not used by us but needs to be cleared from
         * ioc doorbell.  So we just read the values and throw
         * them out.
         */
        if (totalbytes > (numbytes / 2)) {
                for (i = (numbytes / 2); i < totalbytes; i++) {
                        val = (ddi_get32(mpt->m_datap,
                            &mpt->m_reg->Doorbell) &
                            MPI2_DOORBELL_DATA_MASK);
                        ddi_put32(mpt->m_datap,
                            &mpt->m_reg->HostInterruptStatus, 0);
                        if (mptsas_ioc_wait_for_doorbell(mpt)) {
                                NDBG19(("mptsas_get_handshake failure getting "
                                    "extra garbage data\n"));
                                return (-1);
```



```
                        }
                }
        }

        ddi_put32(mpt->m_datap, &mpt->m_reg->HostInterruptStatus, 0);

        if (mptsas_check_acc_handle(mpt->m_datap) != DDI_SUCCESS) {
                ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_UNAFFECTED);
                ddi_fm_acc_err_clear(mpt->m_datap, DDI_FME_VER0);
                return (-1);
        }

        return (0);
}

int
mptsas_kick_start(mptsas_t *mpt)
{
        int             polls = 0;
        uint32_t        diag_reg, ioc_state, saved_HCB_size;

        /*
         * Start a hard reset.  Write magic number and wait 500 mSeconds.
         */
        MPTSAS_ENABLE_DRWE(mpt);
        drv_usecwait(500000);

        /*
         * Read the current Diag Reg and save the Host Controlled Boot size.
         */
        diag_reg = ddi_get32(mpt->m_datap, &mpt->m_reg->HostDiagnostic);
        saved_HCB_size = ddi_get32(mpt->m_datap, &mpt->m_reg->HCBSize);

        /*
         * Set Reset Adapter bit and wait 50 mSeconds.
         */
        diag_reg |= MPI2_DIAG_RESET_ADAPTER;
        ddi_put32(mpt->m_datap, &mpt->m_reg->HostDiagnostic, diag_reg);
        drv_usecwait(50000);

        /*
         * Poll, waiting for Reset Adapter bit to clear.  300 Seconds max
         * (600000 * 500 = 300,000,000 uSeconds, 300 seconds).
         * If no more adapter (all FF’s), just return failure.
         */
        for (polls = 0; polls < 600000; polls++) {
                diag_reg = ddi_get32(mpt->m_datap,
                    &mpt->m_reg->HostDiagnostic);
                if (diag_reg == 0xFFFFFFFF) {
                        mptsas_fm_ereport(mpt, DDI_FM_DEVICE_NO_RESPONSE);
                        ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_LOST);
                        return (DDI_FAILURE);
                }
                if (!(diag_reg & MPI2_DIAG_RESET_ADAPTER)) {
                        break;
                }
                drv_usecwait(500);
        }
        if (polls == 600000) {
                mptsas_fm_ereport(mpt, DDI_FM_DEVICE_NO_RESPONSE);
                ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_LOST);
                return (DDI_FAILURE);
        }

        /*
         * Check if adapter is in Host Boot Mode.  If so, restart adapter
```

```
 * assuming the HCB points to good FW.
 * Set BootDeviceSel to HCDW (Host Code and Data Window).
 */
if (diag_reg & MPI2_DIAG_HCB_MODE) {
        diag_reg &= ~MPI2_DIAG_BOOT_DEVICE_SELECT_MASK;
        diag_reg |= MPI2_DIAG_BOOT_DEVICE_SELECT_HCDW;
        ddi_put32(mpt->m_datap, &mpt->m_reg->HostDiagnostic, diag_reg);

        /*
         * Re-enable the HCDW.
         */
        ddi_put32(mpt->m_datap, &mpt->m_reg->HCBSize,
            (saved_HCB_size | MPI2_HCB_SIZE_HCB_ENABLE));
}

/*
 * Restart the adapter.
 */
diag_reg &= ~MPI2_DIAG_HOLD_IOC_RESET;
ddi_put32(mpt->m_datap, &mpt->m_reg->HostDiagnostic, diag_reg);

/*
 * Disable writes to the Host Diag register.
 */
ddi_put32(mpt->m_datap, &mpt->m_reg->WriteSequence,
    MPI2_WRSEQ_FLUSH_KEY_VALUE);

/*
 * Wait 60 seconds max for FW to come to ready state.
 */
for (polls = 0; polls < 60000; polls++) {
        ioc_state = ddi_get32(mpt->m_datap, &mpt->m_reg->Doorbell);
        if (ioc_state == 0xFFFFFFFF) {
                mptsas_fm_ereport(mpt, DDI_FM_DEVICE_NO_RESPONSE);
                ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_LOST);
                return (DDI_FAILURE);
        }
        if ((ioc_state & MPI2_IOC_STATE_MASK) ==
            MPI2_IOC_STATE_READY) {
                break;
        }
        drv_usecwait(1000);
}
if (polls == 60000) {
        mptsas_fm_ereport(mpt, DDI_FM_DEVICE_NO_RESPONSE);
        ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_LOST);
        return (DDI_FAILURE);
}

/*
 * Clear the ioc ack events queue.
 */
mptsas_destroy_ioc_event_cmd(mpt);

return (DDI_SUCCESS);
}

int
mptsas_ioc_reset(mptsas_t *mpt, int first_time)
{
        int             polls = 0;
        uint32_t        reset_msg;
        uint32_t        ioc_state;

        ioc_state = ddi_get32(mpt->m_datap, &mpt->m_reg->Doorbell);
        /*
```



```
 * If chip is already in ready state then there is nothing to do.
 */
if (ioc_state == MPI2_IOC_STATE_READY) {
        return (MPTSAS_NO_RESET);
}
/*
 * If the chip is already operational, we just need to send
 * it a message unit reset to put it back in the ready state
 */
if (ioc_state & MPI2_IOC_STATE_OPERATIONAL) {
        /*
         * If the first time, try MUR anyway, because we haven't even
         * queried the card for m_event_replay and other capabilities.
         * Other platforms do it this way, we can still do a hard
         * reset if we need to, MUR takes less time than a full
         * adapter reset, and there are reports that some HW
         * combinations will lock up when receiving a hard reset.
         */
        if ((first_time || mpt->m_event_replay) &&
            (mpt->m_softstate & MPTSAS_SS_MSG_UNIT_RESET)) {
                mpt->m_softstate &= ~MPTSAS_SS_MSG_UNIT_RESET;
                reset_msg = MPI2_FUNCTION_IOC_MESSAGE_UNIT_RESET;
                ddi_put32(mpt->m_datap, &mpt->m_reg->Doorbell,
                    (reset_msg << MPI2_DOORBELL_FUNCTION_SHIFT));
                if (mptsas_ioc_wait_for_response(mpt)) {
                        NDBG19(("mptsas_ioc_reset failure sending "
                            "message_unit_reset\n"));
                        goto hard_reset;
                }

                /*
                 * Wait no more than 60 seconds for chip to become
                 * ready.
                 */
                while ((ddi_get32(mpt->m_datap, &mpt->m_reg->Doorbell) &
                    MPI2_IOC_STATE_READY) == 0x0) {
                        drv_usecwait(1000);
                        if (polls++ > 60000) {
                                goto hard_reset;
                        }
                }

                /*
                 * Save the last reset mode done on IOC which will be
                 * helpful while resuming from suspension.
                 */
                mpt->m_softstate |= MPTSAS_DID_MSG_UNIT_RESET;

                /*
                 * the message unit reset would do reset operations
                 * clear reply and request queue, so we should clear
                 * ACK event cmd.
                 */
                mptsas_destroy_ioc_event_cmd(mpt);
                return (MPTSAS_SUCCESS_MUR);
        }
}
hard_reset:
        mpt->m_softstate &= ~MPTSAS_DID_MSG_UNIT_RESET;
        if (mptsas_kick_start(mpt) == DDI_FAILURE) {
                mptsas_fm_ereport(mpt, DDI_FM_DEVICE_NO_RESPONSE);
                ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_LOST);
                return (MPTSAS_RESET_FAIL);
        }
        return (MPTSAS_SUCCESS_HARDRESET);
}
```

```
int
mptsas_request_from_pool(mptsas_t *mpt, mptsas_cmd_t **cmd,
    struct scsi_pkt **pkt)
{
        m_event_struct_t        *ioc_cmd = NULL;

        ioc_cmd = kmem_zalloc(M_EVENT_STRUCT_SIZE, KM_SLEEP);
        if (ioc_cmd == NULL) {
                return (DDI_FAILURE);
        }
        ioc_cmd->m_event_linkp = NULL;
        mptsas_ioc_event_cmdq_add(mpt, ioc_cmd);
        *cmd = &(ioc_cmd->m_event_cmd);
        *pkt = &(ioc_cmd->m_event_pkt);

        return (DDI_SUCCESS);
}

void
mptsas_return_to_pool(mptsas_t *mpt, mptsas_cmd_t *cmd)
{
        m_event_struct_t        *ioc_cmd = NULL;

        ioc_cmd = mptsas_ioc_event_find_by_cmd(mpt, cmd);
        if (ioc_cmd == NULL) {
                return;
        }

        mptsas_ioc_event_cmdq_delete(mpt, ioc_cmd);
        kmem_free(ioc_cmd, M_EVENT_STRUCT_SIZE);
        ioc_cmd = NULL;
}

/*
 * NOTE: We should be able to queue TM requests in the controller to make this
 * a lot faster.  If resetting all targets, for example, we can load the hi
 * priority queue with its limit and the controller will reply as they are
 * completed.  This way, we don't have to poll for one reply at a time.
 * Think about enhancing this later.
 */
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)
{
        /*
         * In order to avoid allocating variables on the stack,
         * we make use of the pre-existing mptsas_cmd_t and
         * scsi_pkt which are included in the mptsas_t which
         * is passed to this routine.
         */

        pMpi2SCSITaskManagementRequest_t        task;
        int                                     rval = FALSE;
        mptsas_cmd_t                            *cmd;
        struct scsi_pkt                         *pkt;
        mptsas_slots_t                          *slots = mpt->m_active;
        uint32_t                                i;
        uint64_t                                request_desc;
        pMPI2DefaultReply_t                     reply_msg;

        /*
         * Can't start another task management routine.
         */
        if (slots->m_slot[MPTSAS_TM_SLOT(mpt)] != NULL) {
```



```
                mptsas_log(mpt, CE_WARN, "Can only start 1 task management"
                    " command at a time\n");
                return (FALSE);
        }

        cmd = &(mpt->m_event_task_mgmt.m_event_cmd);
        pkt = &(mpt->m_event_task_mgmt.m_event_pkt);

        bzero((caddr_t)cmd, sizeof (*cmd));
        bzero((caddr_t)pkt, scsi_pkt_size());

        pkt->pkt_cdbp           = (opaque_t)&cmd->cmd_cdb[0];
        pkt->pkt_scbp           = (opaque_t)&cmd->cmd_scb;
        pkt->pkt_ha_private     = (opaque_t)cmd;
        pkt->pkt_flags          = (FLAG_NOINTR | FLAG_HEAD);
        pkt->pkt_time           = 60;
        pkt->pkt_address.a_target = dev_handle;
        pkt->pkt_address.a_lun = (uchar_t)lun;
        cmd->cmd_pkt            = pkt;
        cmd->cmd_scblen         = 1;
        cmd->cmd_flags          = CFLAG_TM_CMD;
        cmd->cmd_slot           = MPTSAS_TM_SLOT(mpt);

        slots->m_slot[MPTSAS_TM_SLOT(mpt)] = cmd;

        /*
         * Store the TM message in memory location corresponding to the TM slot
         * number.
         */
        task = (pMpi2SCSITaskManagementRequest_t)(mpt->m_req_frame +
            (mpt->m_req_frame_size * cmd->cmd_slot));
        bzero(task, mpt->m_req_frame_size);

        /*
         * form message for requested task
         */
        mptsas_init_std_hdr(mpt->m_acc_req_frame_hdl, task, dev_handle, lun, 0,
            MPI2_FUNCTION_SCSI_TASK_MGMT);

        /*
         * Set the task type
         */
        ddi_put8(mpt->m_acc_req_frame_hdl, &task->TaskType, task_type);

        /*
         * Send TM request using High Priority Queue.
         */
        (void) ddi_dma_sync(mpt->m_dma_req_frame_hdl, 0, 0,
            DDI_DMA_SYNC_FORDEV);
        request_desc = (cmd->cmd_slot << 16) +
            MPI2_REQ_DESCRIPT_FLAGS_HIGH_PRIORITY;
        MPTSAS_START_CMD(mpt, request_desc);
        rval = mptsas_poll(mpt, cmd, MPTSAS_POLL_TIME);

        if (pkt->pkt_reason == CMD_INCOMPLETE)
                rval = FALSE;

        /*
         * If a reply frame was used and there is a reply buffer to copy the
         * reply data into, copy it.  If this fails, log a message, but don't
         * fail the TM request.
         */
        if (cmd->cmd_rfm && reply) {
                (void) ddi_dma_sync(mpt->m_dma_reply_frame_hdl, 0, 0,
                    DDI_DMA_SYNC_FORCPU);
                reply_msg = (pMPI2DefaultReply_t)
```

```
                    (mpt->m_reply_frame + (cmd->cmd_rfm -
                    (mpt->m_reply_frame_dma_addr&0xffffffffu)));
                if (reply_size > sizeof (MPI2_SCSI_TASK_MANAGE_REPLY)) {
                        reply_size = sizeof (MPI2_SCSI_TASK_MANAGE_REPLY);
                }
                mutex_exit(&mpt->m_mutex);
                for (i = 0; i < reply_size; i++) {
                        if (ddi_copyout((uint8_t *)reply_msg + i, reply + i, 1,
                            mode)) {
                                mptsas_log(mpt, CE_WARN, "failed to copy out "
                                    "reply data for TM request");
                                break;
                        }
                }
                mutex_enter(&mpt->m_mutex);
        }

        /*
         * clear the TM slot before returning
         */
        slots->m_slot[MPTSAS_TM_SLOT(mpt)] = NULL;

        /*
         * If we lost our task management command
         * we need to reset the ioc
         */
        if (rval == FALSE) {
                mptsas_log(mpt, CE_WARN, "mptsas_ioc_task_management failed "
                    "try to reset ioc to recovery!");
                mpt->m_softstate &= ~MPTSAS_SS_MSG_UNIT_RESET;
                if (mptsas_restart_ioc(mpt)) {
                        mptsas_log(mpt, CE_WARN, "mptsas_restart_ioc failed");
                        rval = FAILED;
                }
        }

        return (rval);
}

/*
 * Complete firmware download frame for v2.0 cards.
 */
static void
mptsas_uflash2(pMpi2FWDownloadRequest fwdownload,
    ddi_acc_handle_t acc_hdl, uint32_t size, uint8_t type,
    ddi_dma_cookie_t flsh_cookie)
{
        pMpi2FWDownloadTCSGE_t  tcsge;
        pMpi2SGESimple64_t      sge;
        uint32_t                flagslength;

        ddi_put8(acc_hdl, &fwdownload->Function,
            MPI2_FUNCTION_FW_DOWNLOAD);
        ddi_put8(acc_hdl, &fwdownload->ImageType, type);
        ddi_put8(acc_hdl, &fwdownload->MsgFlags,
            MPI2_FW_DOWNLOAD_MSGFLGS_LAST_SEGMENT);
        ddi_put32(acc_hdl, &fwdownload->TotalImageSize, size);

        tcsge = (pMpi2FWDownloadTCSGE_t)&fwdownload->SGL;
        ddi_put8(acc_hdl, &tcsge->ContextSize, 0);
        ddi_put8(acc_hdl, &tcsge->DetailsLength, 12);
        ddi_put8(acc_hdl, &tcsge->Flags, 0);
        ddi_put32(acc_hdl, &tcsge->ImageOffset, 0);
        ddi_put32(acc_hdl, &tcsge->ImageSize, size);

        sge = (pMpi2SGESimple64_t)(tcsge + 1);
```



```
        flagslength = size;
        flagslength |= ((uint32_t)(MPI2_SGE_FLAGS_LAST_ELEMENT |
            MPI2_SGE_FLAGS_END_OF_BUFFER |
            MPI2_SGE_FLAGS_SIMPLE_ELEMENT |
            MPI2_SGE_FLAGS_SYSTEM_ADDRESS |
            MPI2_SGE_FLAGS_64_BIT_ADDRESSING |
            MPI2_SGE_FLAGS_HOST_TO_IOC |
            MPI2_SGE_FLAGS_END_OF_LIST) << MPI2_SGE_FLAGS_SHIFT);
        ddi_put32(acc_hdl, &sge->FlagsLength, flagslength);
        ddi_put32(acc_hdl, &sge->Address.Low,
            flsh_cookie.dmac_address);
        ddi_put32(acc_hdl, &sge->Address.High,
            (uint32_t)(flsh_cookie.dmac_laddress >> 32));
}

/*
 * Complete firmware download frame for v2.5 cards.
 */
static void
mptsas_uflash25(pMpi25FWDownloadRequest fwdownload,
    ddi_acc_handle_t acc_hdl, uint32_t size, uint8_t type,
    ddi_dma_cookie_t flsh_cookie)
{
        pMpi2IeeeSgeSimple64_t  sge;
        uint8_t                 flags;

        ddi_put8(acc_hdl, &fwdownload->Function,
            MPI2_FUNCTION_FW_DOWNLOAD);
        ddi_put8(acc_hdl, &fwdownload->ImageType, type);
        ddi_put8(acc_hdl, &fwdownload->MsgFlags,
            MPI2_FW_DOWNLOAD_MSGFLGS_LAST_SEGMENT);
        ddi_put32(acc_hdl, &fwdownload->TotalImageSize, size);

        ddi_put32(acc_hdl, &fwdownload->ImageOffset, 0);
        ddi_put32(acc_hdl, &fwdownload->ImageSize, size);

        sge = (pMpi2IeeeSgeSimple64_t)&fwdownload->SGL;
        flags = MPI2_IEEE_SGE_FLAGS_SIMPLE_ELEMENT |
            MPI2_IEEE_SGE_FLAGS_SYSTEM_ADDR |
            MPI25_IEEE_SGE_FLAGS_END_OF_LIST;
        ddi_put8(acc_hdl, &sge->Flags, flags);
        ddi_put32(acc_hdl, &sge->Length, size);
        ddi_put32(acc_hdl, &sge->Address.Low,
            flsh_cookie.dmac_address);
        ddi_put32(acc_hdl, &sge->Address.High,
            (uint32_t)(flsh_cookie.dmac_laddress >> 32));
}

static int mptsas_enable_mpi25_flashupdate = 0;

int
mptsas_update_flash(mptsas_t *mpt, caddr_t ptrbuffer, uint32_t size,
    uint8_t type, int mode)
{

        /*
         * In order to avoid allocating variables on the stack,
         * we make use of the pre-existing mptsas_cmd_t and
         * scsi_pkt which are included in the mptsas_t which
         * is passed to this routine.
         */

        ddi_dma_attr_t          flsh_dma_attrs;
        ddi_dma_cookie_t        flsh_cookie;
        ddi_dma_handle_t        flsh_dma_handle;
        ddi_acc_handle_t        flsh_accessp;
```

```
        caddr_t                 memp, flsh_memp;
        mptsas_cmd_t            *cmd;
        struct scsi_pkt         *pkt;
        int                     i;
        int                     rvalue = 0;
        uint64_t                request_desc;

        if (mpt->m_MPI25) {
                /*
                 * The code is there but not tested yet.
                 * User has to know there are risks here.
                 */
                mptsas_log(mpt, CE_WARN, "mptsas_update_flash(): "
                    "Updating firmware through MPI 2.5 has not been "
                    "tested yet!\n"
                    "To enable set mptsas_enable_mpi25_flashupdate to 1\n");
                if (!mptsas_enable_mpi25_flashupdate)
                        return (-1);
        }

        if ((rvalue = (mptsas_request_from_pool(mpt, &cmd, &pkt))) == -1) {
                mptsas_log(mpt, CE_WARN, "mptsas_update_flash(): allocation "
                    "failed. event ack command pool is full\n");
                return (rvalue);
        }

        bzero((caddr_t)cmd, sizeof (*cmd));
        bzero((caddr_t)pkt, scsi_pkt_size());
        cmd->ioc_cmd_slot = (uint32_t)rvalue;

        /*
         * dynamically create a customized dma attribute structure
         * that describes the flash file.
         */
        flsh_dma_attrs = mpt->m_msg_dma_attr;
        flsh_dma_attrs.dma_attr_sgllen = 1;

        if (mptsas_dma_addr_create(mpt, flsh_dma_attrs, &flsh_dma_handle,
            &flsh_accessp, &flsh_memp, size, &flsh_cookie) == FALSE) {
                mptsas_log(mpt, CE_WARN,
                    "(unable to allocate dma resource.");
                mptsas_return_to_pool(mpt, cmd);
                return (-1);
        }

        bzero(flsh_memp, size);

        for (i = 0; i < size; i++) {
                (void) ddi_copyin(ptrbuffer + i, flsh_memp + i, 1, mode);
        }
        (void) ddi_dma_sync(flsh_dma_handle, 0, 0, DDI_DMA_SYNC_FORDEV);

        /*
         * form a cmd/pkt to store the fw download message
         */
        pkt->pkt_cdbp           = (opaque_t)&cmd->cmd_cdb[0];
        pkt->pkt_scbp           = (opaque_t)&cmd->cmd_scb;
        pkt->pkt_ha_private     = (opaque_t)cmd;
        pkt->pkt_flags          = FLAG_HEAD;
        pkt->pkt_time           = 60;
        cmd->cmd_pkt            = pkt;
        cmd->cmd_scblen         = 1;
        cmd->cmd_flags          = CFLAG_CMDIOC | CFLAG_FW_CMD;

        /*
         * Save the command in a slot
```



```
         */
        if (mptsas_save_cmd(mpt, cmd) == FALSE) {
                mptsas_dma_addr_destroy(&flsh_dma_handle, &flsh_accessp);
                mptsas_return_to_pool(mpt, cmd);
                return (-1);
        }

        /*
         * Fill in fw download message
         */
        ASSERT(cmd->cmd_slot != 0);
        memp = mpt->m_req_frame + (mpt->m_req_frame_size * cmd->cmd_slot);
        bzero(memp, mpt->m_req_frame_size);

        if (mpt->m_MPI25)
                mptsas_uflash25((pMpi25FWDownloadRequest)memp,
                    mpt->m_acc_req_frame_hdl, size, type, flsh_cookie);
        else
                mptsas_uflash2((pMpi2FWDownloadRequest)memp,
                    mpt->m_acc_req_frame_hdl, size, type, flsh_cookie);

        /*
         * Start command
         */
        (void) ddi_dma_sync(mpt->m_dma_req_frame_hdl, 0, 0,
            DDI_DMA_SYNC_FORDEV);
        request_desc = (cmd->cmd_slot << 16) +
            MPI2_REQ_DESCRIPT_FLAGS_DEFAULT_TYPE;
        cmd->cmd_rfm = NULL;
        MPTSAS_START_CMD(mpt, request_desc);

        rvalue = 0;
        (void) cv_reltimedwait(&mpt->m_fw_cv, &mpt->m_mutex,
            drv_usectohz(60 * MICROSEC), TR_CLOCK_TICK);
        if (!(cmd->cmd_flags & CFLAG_FINISHED)) {
                mpt->m_softstate &= ~MPTSAS_SS_MSG_UNIT_RESET;
                if ((mptsas_restart_ioc(mpt)) == DDI_FAILURE) {
                        mptsas_log(mpt, CE_WARN, "mptsas_restart_ioc failed");
                }
                rvalue = -1;
        }
        mptsas_remove_cmd(mpt, cmd);
        mptsas_dma_addr_destroy(&flsh_dma_handle, &flsh_accessp);

        return (rvalue);
}

static int
mptsas_sasdevpage_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
        pMpi2SasDevicePage0_t   sasdevpage;
        int                     rval = DDI_SUCCESS, i;
        uint8_t                 *sas_addr = NULL;
        uint8_t                 tmp_sas_wwn[SAS_WWN_BYTE_SIZE];
        uint16_t                *devhdl, *bay_num, *enclosure;
        uint64_t                *sas_wwn;
        uint32_t                *dev_info;
        uint8_t                 *physport, *phynum;
        uint16_t                *pdevhdl, *io_flags;
        uint32_t                page_address;
```

```
        if ((iocstatus != MPI2_IOCSTATUS_SUCCESS) &&
            (iocstatus != MPI2_IOCSTATUS_CONFIG_INVALID_PAGE)) {
                mptsas_log(mpt, CE_WARN, "mptsas_get_sas_device_page0 "
                    "header: IOCStatus=0x%x, IOCLogInfo=0x%x",
                    iocstatus, iocloginfo);
                rval = DDI_FAILURE;
                return (rval);
        }
        page_address = va_arg(ap, uint32_t);
        /*
         * The INVALID_PAGE status is normal if using GET_NEXT_HANDLE and there
         * are no more pages.  If everything is OK up to this point but the
         * status is INVALID_PAGE, change rval to FAILURE and quit.  Also,
         * signal that device traversal is complete.
         */
        if (iocstatus == MPI2_IOCSTATUS_CONFIG_INVALID_PAGE) {
                if ((page_address & MPI2_SAS_DEVICE_PGAD_FORM_MASK) ==
                    MPI2_SAS_DEVICE_PGAD_FORM_GET_NEXT_HANDLE) {
                        mpt->m_done_traverse_dev = 1;
                }
                rval = DDI_FAILURE;
                return (rval);
        }
        devhdl = va_arg(ap, uint16_t *);
        sas_wwn = va_arg(ap, uint64_t *);
        dev_info = va_arg(ap, uint32_t *);
        physport = va_arg(ap, uint8_t *);
        phynum = va_arg(ap, uint8_t *);
        pdevhdl = va_arg(ap, uint16_t *);
        bay_num = va_arg(ap, uint16_t *);
        enclosure = va_arg(ap, uint16_t *);
        io_flags = va_arg(ap, uint16_t *);

        sasdevpage = (pMpi2SasDevicePage0_t)page_memp;

        *dev_info = ddi_get32(accessp, &sasdevpage->DeviceInfo);
        *devhdl = ddi_get16(accessp, &sasdevpage->DevHandle);
        sas_addr = (uint8_t *)(&sasdevpage->SASAddress);
        for (i = 0; i < SAS_WWN_BYTE_SIZE; i++) {
                tmp_sas_wwn[i] = ddi_get8(accessp, sas_addr + i);
        }
        bcopy(tmp_sas_wwn, sas_wwn, SAS_WWN_BYTE_SIZE);
        *sas_wwn = LE_64(*sas_wwn);
        *physport = ddi_get8(accessp, &sasdevpage->PhysicalPort);
        *phynum = ddi_get8(accessp, &sasdevpage->PhyNum);
        *pdevhdl = ddi_get16(accessp, &sasdevpage->ParentDevHandle);
        *bay_num = ddi_get16(accessp, &sasdevpage->Slot);
        *enclosure = ddi_get16(accessp, &sasdevpage->EnclosureHandle);
        *io_flags = ddi_get16(accessp, &sasdevpage->Flags);

        if (*io_flags & MPI25_SAS_DEVICE0_FLAGS_FAST_PATH_CAPABLE) {
                /*
                 * Leave a messages about FP cabability in the log.
                 */
                mptsas_log(mpt, CE_CONT,
                    "!w%016"PRIx64" FastPath Capable%s", *sas_wwn,
                    (*io_flags &
                    MPI25_SAS_DEVICE0_FLAGS_ENABLED_FAST_PATH)?
                    " and Enabled":" but Disabled");
        }

        return (rval);
}

/*
 * Request MPI configuration page SAS device page 0 to get DevHandle, device
```



```
 * info and SAS address.
 */
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 *pdev_handle,
    uint16_t *bay_num, uint16_t *enclosure, uint16_t *io_flags)
{
        int rval = DDI_SUCCESS;

        ASSERT(mutex_owned(&mpt->m_mutex));

        /*
         * Get the header and config page.  reply contains the reply frame,
         * which holds status info for the request.
         */
        rval = mptsas_access_config_page(mpt,
            MPI2_CONFIG_ACTION_PAGE_READ_CURRENT,
            MPI2_CONFIG_EXTPAGETYPE_SAS_DEVICE, 0, page_address,
            mptsas_sasdevpage_0_cb, page_address, dev_handle, sas_wwn,
            dev_info, physport, phynum, pdev_handle,
            bay_num, enclosure, io_flags);

        return (rval);
}

static int
mptsas_sasexpdpage_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
        pMpi2ExpanderPage0_t    expddevpage;
        int                     rval = DDI_SUCCESS, i;
        uint8_t                 *sas_addr = NULL;
        uint8_t                 tmp_sas_wwn[SAS_WWN_BYTE_SIZE];
        uint16_t                *devhdl;
        uint64_t                *sas_wwn;
        uint8_t                 physport;
        mptsas_phymask_t        *phymask;
        uint16_t                *pdevhdl;
        uint32_t                page_address;

        if ((iocstatus != MPI2_IOCSTATUS_SUCCESS) &&
            (iocstatus != MPI2_IOCSTATUS_CONFIG_INVALID_PAGE)) {
                mptsas_log(mpt, CE_WARN, "mptsas_get_sas_expander_page0 "
                    "config: IOCStatus=0x%x, IOCLogInfo=0x%x",
                    iocstatus, iocloginfo);
                rval = DDI_FAILURE;
                return (rval);
        }
        page_address = va_arg(ap, uint32_t);
        /*
         * The INVALID_PAGE status is normal if using GET_NEXT_HANDLE and there
         * are no more pages.  If everything is OK up to this point but the
         * status is INVALID_PAGE, change rval to FAILURE and quit.  Also,
         * signal that device traversal is complete.
         */
        if (iocstatus == MPI2_IOCSTATUS_CONFIG_INVALID_PAGE) {
                if ((page_address & MPI2_SAS_EXPAND_PGAD_FORM_MASK) ==
                    MPI2_SAS_EXPAND_PGAD_FORM_GET_NEXT_HNDL) {
                        mpt->m_done_traverse_smp = 1;
                }
                rval = DDI_FAILURE;
```

```
                return (rval);
        }
        devhdl = va_arg(ap, uint16_t *);
        sas_wwn = va_arg(ap, uint64_t *);
        phymask = va_arg(ap, mptsas_phymask_t *);
        pdevhdl = va_arg(ap, uint16_t *);

        expddevpage = (pMpi2ExpanderPage0_t)page_memp;

        *devhdl = ddi_get16(accessp, &expddevpage->DevHandle);
        physport = ddi_get8(accessp, &expddevpage->PhysicalPort);
        *phymask = mptsas_physport_to_phymask(mpt, physport);
        *pdevhdl = ddi_get16(accessp, &expddevpage->ParentDevHandle);
        sas_addr = (uint8_t *)(&expddevpage->SASAddress);
        for (i = 0; i < SAS_WWN_BYTE_SIZE; i++) {
                tmp_sas_wwn[i] = ddi_get8(accessp, sas_addr + i);
        }
        bcopy(tmp_sas_wwn, sas_wwn, SAS_WWN_BYTE_SIZE);
        *sas_wwn = LE_64(*sas_wwn);

        return (rval);
}

/*
 * Request MPI configuration page SAS device page 0 to get DevHandle, phymask
 * and SAS address.
 */
int
mptsas_get_sas_expander_page0(mptsas_t *mpt, uint32_t page_address,
    mptsas_smp_t *info)
{
        int                     rval = DDI_SUCCESS;

        ASSERT(mutex_owned(&mpt->m_mutex));

        /*
         * Get the header and config page.  reply contains the reply frame,
         * which holds status info for the request.
         */
        rval = mptsas_access_config_page(mpt,
            MPI2_CONFIG_ACTION_PAGE_READ_CURRENT,
            MPI2_CONFIG_EXTPAGETYPE_SAS_EXPANDER, 0, page_address,
            mptsas_sasexpdpage_0_cb, page_address, &info->m_devhdl,
            &info->m_addr.mta_wwn, &info->m_addr.mta_phymask, &info->m_pdevhdl);

        return (rval);
}

static int
mptsas_sasportpage_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
        int     rval = DDI_SUCCESS, i;
        uint8_t *sas_addr = NULL;
        uint64_t *sas_wwn;
        uint8_t tmp_sas_wwn[SAS_WWN_BYTE_SIZE];
        uint8_t *portwidth;
        pMpi2SasPortPage0_t sasportpage;

        if (iocstatus != MPI2_IOCSTATUS_SUCCESS) {
                mptsas_log(mpt, CE_WARN, "mptsas_get_sas_port_page0 "
                    "config: IOCStatus=0x%x, IOCLogInfo=0x%x",
```



```
                    iocstatus, iocloginfo);
                rval = DDI_FAILURE;
                return (rval);
        }
        sas_wwn = va_arg(ap, uint64_t *);
        portwidth = va_arg(ap, uint8_t *);

        sasportpage = (pMpi2SasPortPage0_t)page_memp;
        sas_addr = (uint8_t *)(&sasportpage->SASAddress);
        for (i = 0; i < SAS_WWN_BYTE_SIZE; i++) {
                tmp_sas_wwn[i] = ddi_get8(accessp, sas_addr + i);
        }
        bcopy(tmp_sas_wwn, sas_wwn, SAS_WWN_BYTE_SIZE);
        *sas_wwn = LE_64(*sas_wwn);
        *portwidth = ddi_get8(accessp, &sasportpage->PortWidth);
        return (rval);
}

/*
 * Request MPI configuration page SAS port page 0 to get initiator SAS address
 * and port width.
 */
int
mptsas_get_sas_port_page0(mptsas_t *mpt, uint32_t page_address,
    uint64_t *sas_wwn, uint8_t *portwidth)
{
        int rval = DDI_SUCCESS;

        ASSERT(mutex_owned(&mpt->m_mutex));

        /*
         * Get the header and config page.  reply contains the reply frame,
         * which holds status info for the request.
         */
        rval = mptsas_access_config_page(mpt,
            MPI2_CONFIG_ACTION_PAGE_READ_CURRENT,
            MPI2_CONFIG_EXTPAGETYPE_SAS_PORT, 0, page_address,
            mptsas_sasportpage_0_cb, sas_wwn, portwidth);

        return (rval);
}

static int
mptsas_sasiou_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
        int rval = DDI_SUCCESS;
        pMpi2SasIOUnitPage0_t sasioupage0;
        int i, num_phys;
        uint32_t cpdi[MPTSAS_MAX_PHYS], *retrypage0, *readpage1;
        uint8_t port_flags;

        if (iocstatus != MPI2_IOCSTATUS_SUCCESS) {
                mptsas_log(mpt, CE_WARN, "mptsas_get_sas_io_unit_page0 "
                    "config: IOCStatus=0x%x, IOCLogInfo=0x%x",
                    iocstatus, iocloginfo);
                rval = DDI_FAILURE;
                return (rval);
        }
        readpage1 = va_arg(ap, uint32_t *);
        retrypage0 = va_arg(ap, uint32_t *);
```

```
        sasioupage0 = (pMpi2SasIOUnitPage0_t)page_memp;

        num_phys = ddi_get8(accessp, &sasioupage0->NumPhys);
        /*
         * ASSERT that the num_phys value in SAS IO Unit Page 0 is the same as
         * was initially set.  This should never change throughout the life of
         * the driver.
         */
        ASSERT(num_phys == mpt->m_num_phys);
        for (i = 0; i < num_phys; i++) {
                cpdi[i] = ddi_get32(accessp,
                    &sasioupage0->PhyData[i].
                    ControllerPhyDeviceInfo);
                port_flags = ddi_get8(accessp,
                    &sasioupage0->PhyData[i].PortFlags);
                mpt->m_phy_info[i].port_num =
                    ddi_get8(accessp,
                    &sasioupage0->PhyData[i].Port);
                mpt->m_phy_info[i].ctrl_devhdl =
                    ddi_get16(accessp, &sasioupage0->
                    PhyData[i].ControllerDevHandle);
                mpt->m_phy_info[i].attached_devhdl =
                    ddi_get16(accessp, &sasioupage0->
                    PhyData[i].AttachedDevHandle);
                mpt->m_phy_info[i].phy_device_type = cpdi[i];
                mpt->m_phy_info[i].port_flags = port_flags;

                if (port_flags & DISCOVERY_IN_PROGRESS) {
                        *retrypage0 = *retrypage0 + 1;
                        break;
                } else {
                        *retrypage0 = 0;
                }
                if (!(port_flags & AUTO_PORT_CONFIGURATION)) {
                        /*
                         * some PHY configuration described in
                         * SAS IO Unit Page1
                         */
                        *readpage1 = 1;
                }
        }

        return (rval);
}

static int
mptsas_sasiou_page_1_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
        int rval = DDI_SUCCESS;
        pMpi2SasIOUnitPage1_t sasioupage1;
        int i, num_phys;
        uint32_t cpdi[MPTSAS_MAX_PHYS];
        uint8_t port_flags;

        if (iocstatus != MPI2_IOCSTATUS_SUCCESS) {
                mptsas_log(mpt, CE_WARN, "mptsas_get_sas_io_unit_page1 "
                    "config: IOCStatus=0x%x, IOCLogInfo=0x%x",
                    iocstatus, iocloginfo);
                rval = DDI_FAILURE;
                return (rval);
        }
```



```
        sasioupage1 = (pMpi2SasIOUnitPage1_t)page_memp;
        num_phys = ddi_get8(accessp, &sasioupage1->NumPhys);
        /*
         * ASSERT that the num_phys value in SAS IO Unit Page 1 is the same as
         * was initially set.  This should never change throughout the life of
         * the driver.
         */
        ASSERT(num_phys == mpt->m_num_phys);
        for (i = 0; i < num_phys; i++) {
                cpdi[i] = ddi_get32(accessp, &sasioupage1->PhyData[i].
                    ControllerPhyDeviceInfo);
                port_flags = ddi_get8(accessp,
                    &sasioupage1->PhyData[i].PortFlags);
                mpt->m_phy_info[i].port_num =
                    ddi_get8(accessp,
                    &sasioupage1->PhyData[i].Port);
                mpt->m_phy_info[i].port_flags = port_flags;
                mpt->m_phy_info[i].phy_device_type = cpdi[i];
        }
        return (rval);
}

/*
 * Read IO unit page 0 to get information for each PHY. If needed, Read IO Unit
 * page1 to update the PHY information.  This is the message passing method of
 * this function which should be called except during initialization.
 */
int
mptsas_get_sas_io_unit_page(mptsas_t *mpt)
{
        int rval = DDI_SUCCESS, state;
        uint32_t readpage1 = 0, retrypage0 = 0;

        ASSERT(mutex_owned(&mpt->m_mutex));

        /*
         * Now we cycle through the state machine.  Here's what happens:
         * 1. Read IO unit page 0 and set phy information
         * 2. See if Read IO unit page1 is needed because of port configuration
         * 3. Read IO unit page 1 and update phy information.
         */
        state = IOUC_READ_PAGE0;
        while (state != IOUC_DONE) {
                if (state == IOUC_READ_PAGE0) {
                        rval = mptsas_access_config_page(mpt,
                            MPI2_CONFIG_ACTION_PAGE_READ_CURRENT,
                            MPI2_CONFIG_EXTPAGETYPE_SAS_IO_UNIT, 0, 0,
                            mptsas_sasiou_page_0_cb, &readpage1,
                            &retrypage0);
                } else if (state == IOUC_READ_PAGE1) {
                        rval = mptsas_access_config_page(mpt,
                            MPI2_CONFIG_ACTION_PAGE_READ_CURRENT,
                            MPI2_CONFIG_EXTPAGETYPE_SAS_IO_UNIT, 1, 0,
                            mptsas_sasiou_page_1_cb);
                }

                if (rval == DDI_SUCCESS) {
                        switch (state) {
                        case IOUC_READ_PAGE0:
                                /*
                                 * retry 30 times if discovery is in process
                                 */
                                if (retrypage0 && (retrypage0 < 30)) {
                                        drv_usecwait(1000 * 100);
                                        state = IOUC_READ_PAGE0;
```

```
                                        break;
                                } else if (retrypage0 == 30) {
                                        mptsas_log(mpt, CE_WARN,
                                            "!Discovery in progress, can't "
                                            "verify IO unit config, then "
                                            "after 30 times retry, give "
                                            "up!");
                                        state = IOUC_DONE;
                                        rval = DDI_FAILURE;
                                        break;
                                }

                                if (readpage1 == 0) {
                                        state = IOUC_DONE;
                                        rval = DDI_SUCCESS;
                                        break;
                                }

                                state = IOUC_READ_PAGE1;
                                break;

                        case IOUC_READ_PAGE1:
                                state = IOUC_DONE;
                                rval = DDI_SUCCESS;
                                break;
                        }
                } else {
                        return (rval);
                }
        }

        return (rval);
}

static int
mptsas_biospage_3_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
        pMpi2BiosPage3_t        sasbiospage;
        int                     rval = DDI_SUCCESS;
        uint32_t                *bios_version;

        if ((iocstatus != MPI2_IOCSTATUS_SUCCESS) &&
            (iocstatus != MPI2_IOCSTATUS_CONFIG_INVALID_PAGE)) {
                mptsas_log(mpt, CE_WARN, "mptsas_get_bios_page3 header: "
                    "IOCStatus=0x%x, IOCLogInfo=0x%x", iocstatus, iocloginfo);
                rval = DDI_FAILURE;
                return (rval);
        }
        bios_version = va_arg(ap, uint32_t *);
        sasbiospage = (pMpi2BiosPage3_t)page_memp;
        *bios_version = ddi_get32(accessp, &sasbiospage->BiosVersion);

        return (rval);
}

/*
 * Request MPI configuration page BIOS page 3 to get BIOS version.  Since all
 * other information in this page is not needed, just ignore it.
 */
int
mptsas_get_bios_page3(mptsas_t *mpt, uint32_t *bios_version)
```



```
{
        int rval = DDI_SUCCESS;

        ASSERT(mutex_owned(&mpt->m_mutex));

        /*
         * Get the header and config page.  reply contains the reply frame,
         * which holds status info for the request.
         */
        rval = mptsas_access_config_page(mpt,
            MPI2_CONFIG_ACTION_PAGE_READ_CURRENT, MPI2_CONFIG_PAGETYPE_BIOS, 3,
            0, mptsas_biospage_3_cb, bios_version);

        return (rval);
}

/*
 * Read IO unit page 0 to get information for each PHY. If needed, Read IO Unit
 * page1 to update the PHY information.  This is the handshaking version of
 * this function, which should be called during initialization only.
 */
int
mptsas_get_sas_io_unit_page_hndshk(mptsas_t *mpt)
{
        ddi_dma_attr_t          recv_dma_attrs, page_dma_attrs;
        ddi_dma_cookie_t        page_cookie;
        ddi_dma_handle_t        recv_dma_handle, page_dma_handle;
        ddi_acc_handle_t        recv_accessp, page_accessp;
        pMpi2ConfigReply_t      configreply;
        pMpi2SasIOUnitPage0_t   sasioupage0;
        pMpi2SasIOUnitPage1_t   sasioupage1;
        int                     recv_numbytes;
        caddr_t                 recv_memp, page_memp;
        int                     i, num_phys, start_phy = 0;
        int                     page0_size =
            sizeof (MPI2_CONFIG_PAGE_SASIOUNIT_0) +
            (sizeof (MPI2_SAS_IO_UNIT0_PHY_DATA) * (MPTSAS_MAX_PHYS - 1));
        int                     page1_size =
            sizeof (MPI2_CONFIG_PAGE_SASIOUNIT_1) +
            (sizeof (MPI2_SAS_IO_UNIT1_PHY_DATA) * (MPTSAS_MAX_PHYS - 1));
        uint32_t                flags_length;
        uint32_t                cpdi[MPTSAS_MAX_PHYS];
        uint32_t                readpage1 = 0, retrypage0 = 0;
        uint16_t                iocstatus;
        uint8_t                 port_flags, page_number, action;
        uint32_t                reply_size = 256; /* Big enough for any page */
        uint_t                  state;
        int                     rval = DDI_FAILURE;
        boolean_t               free_recv = B_FALSE, free_page = B_FALSE;

        /*
         * Initialize our "state machine".  This is a bit convoluted,
         * but it keeps us from having to do the ddi allocations numerous
         * times.
         */

        NDBG20(("mptsas_get_sas_io_unit_page_hndshk enter"));
        ASSERT(mutex_owned(&mpt->m_mutex));
        state = IOUC_READ_PAGE0;

        /*
         * dynamically create a customized dma attribute structure
         * that describes mpt's config reply page request structure.
         */
        recv_dma_attrs = mpt->m_msg_dma_attr;
        recv_dma_attrs.dma_attr_sgllen = 1;
```

```
        recv_dma_attrs.dma_attr_granular = (sizeof (MPI2_CONFIG_REPLY));

        if (mptsas_dma_addr_create(mpt, recv_dma_attrs,
            &recv_dma_handle, &recv_accessp, &recv_memp,
            (sizeof (MPI2_CONFIG_REPLY)), NULL) == FALSE) {
                mptsas_log(mpt, CE_WARN,
                    "mptsas_get_sas_io_unit_page_hndshk: recv dma failed");
                goto cleanup;
        }
        /* Now safe to call mptsas_dma_addr_destroy(recv_dma_handle). */
        free_recv = B_TRUE;

        page_dma_attrs = mpt->m_msg_dma_attr;
        page_dma_attrs.dma_attr_sgllen = 1;
        page_dma_attrs.dma_attr_granular = reply_size;

        if (mptsas_dma_addr_create(mpt, page_dma_attrs,
            &page_dma_handle, &page_accessp, &page_memp,
            reply_size, &page_cookie) == FALSE) {
                mptsas_log(mpt, CE_WARN,
                    "mptsas_get_sas_io_unit_page_hndshk: page dma failed");
                goto cleanup;
        }
        /* Now safe to call mptsas_dma_addr_destroy(page_dma_handle). */
        free_page = B_TRUE;

        /*
         * Now we cycle through the state machine.  Here's what happens:
         * 1. Read IO unit page 0 and set phy information
         * 2. See if Read IO unit page1 is needed because of port configuration
         * 3. Read IO unit page 1 and update phy information.
         */

        sasioupage0 = (pMpi2SasIOUnitPage0_t)page_memp;
        sasioupage1 = (pMpi2SasIOUnitPage1_t)page_memp;

        while (state != IOUC_DONE) {
                switch (state) {
                case IOUC_READ_PAGE0:
                        page_number = 0;
                        action = MPI2_CONFIG_ACTION_PAGE_READ_CURRENT;
                        flags_length = (uint32_t)page0_size;
                        flags_length |= ((uint32_t)(
                            MPI2_SGE_FLAGS_LAST_ELEMENT |
                            MPI2_SGE_FLAGS_END_OF_BUFFER |
                            MPI2_SGE_FLAGS_SIMPLE_ELEMENT |
                            MPI2_SGE_FLAGS_SYSTEM_ADDRESS |
                            MPI2_SGE_FLAGS_64_BIT_ADDRESSING |
                            MPI2_SGE_FLAGS_IOC_TO_HOST |
                            MPI2_SGE_FLAGS_END_OF_LIST) <<
                            MPI2_SGE_FLAGS_SHIFT);

                        break;

                case IOUC_READ_PAGE1:
                        page_number = 1;
                        action = MPI2_CONFIG_ACTION_PAGE_READ_CURRENT;
                        flags_length = (uint32_t)page1_size;
                        flags_length |= ((uint32_t)(
                            MPI2_SGE_FLAGS_LAST_ELEMENT |
                            MPI2_SGE_FLAGS_END_OF_BUFFER |
                            MPI2_SGE_FLAGS_SIMPLE_ELEMENT |
                            MPI2_SGE_FLAGS_SYSTEM_ADDRESS |
                            MPI2_SGE_FLAGS_64_BIT_ADDRESSING |
                            MPI2_SGE_FLAGS_IOC_TO_HOST |
                            MPI2_SGE_FLAGS_END_OF_LIST) <<
```



```
                            MPI2_SGE_FLAGS_SHIFT);

                        break;
                default:
                        break;
                }

                bzero(recv_memp, sizeof (MPI2_CONFIG_REPLY));
                configreply = (pMpi2ConfigReply_t)recv_memp;
                recv_numbytes = sizeof (MPI2_CONFIG_REPLY);

                if (mptsas_send_extended_config_request_msg(mpt,
                    MPI2_CONFIG_ACTION_PAGE_HEADER,
                    MPI2_CONFIG_EXTPAGETYPE_SAS_IO_UNIT,
                    0, page_number, 0, 0, 0, 0)) {
                        goto cleanup;
                }

                if (mptsas_get_handshake_msg(mpt, recv_memp, recv_numbytes,
                    recv_accessp)) {
                        goto cleanup;
                }

                iocstatus = ddi_get16(recv_accessp, &configreply->IOCStatus);
                iocstatus = MPTSAS_IOCSTATUS(iocstatus);

                if (iocstatus != MPI2_IOCSTATUS_SUCCESS) {
                        mptsas_log(mpt, CE_WARN,
                            "mptsas_get_sas_io_unit_page_hndshk: read page "
                            "header iocstatus = 0x%x", iocstatus);
                        goto cleanup;
                }

                if (action != MPI2_CONFIG_ACTION_PAGE_WRITE_NVRAM) {
                        bzero(page_memp, reply_size);
                }

                if (mptsas_send_extended_config_request_msg(mpt, action,
                    MPI2_CONFIG_EXTPAGETYPE_SAS_IO_UNIT, 0, page_number,
                    ddi_get8(recv_accessp, &configreply->Header.PageVersion),
                    ddi_get16(recv_accessp, &configreply->ExtPageLength),
                    flags_length, page_cookie.dmac_laddress)) {
                        goto cleanup;
                }

                if (mptsas_get_handshake_msg(mpt, recv_memp, recv_numbytes,
                    recv_accessp)) {
                        goto cleanup;
                }

                iocstatus = ddi_get16(recv_accessp, &configreply->IOCStatus);
                iocstatus = MPTSAS_IOCSTATUS(iocstatus);

                if (iocstatus != MPI2_IOCSTATUS_SUCCESS) {
                        mptsas_log(mpt, CE_WARN,
                            "mptsas_get_sas_io_unit_page_hndshk: IO unit "
                            "config failed for action %d, iocstatus = 0x%x",
                            action, iocstatus);
                        goto cleanup;
                }

                switch (state) {
                case IOUC_READ_PAGE0:
                        if ((ddi_dma_sync(page_dma_handle, 0, 0,
                            DDI_DMA_SYNC_FORCPU)) != DDI_SUCCESS) {
                                goto cleanup;
```

```
                        }

                        num_phys = ddi_get8(page_accessp,
                            &sasioupage0->NumPhys);
                        ASSERT(num_phys == mpt->m_num_phys);
                        if (num_phys > MPTSAS_MAX_PHYS) {
                                mptsas_log(mpt, CE_WARN, "Number of phys "
                                    "supported by HBA (%d) is more than max "
                                    "supported by driver (%d).  Driver will "
                                    "not attach.", num_phys,
                                    MPTSAS_MAX_PHYS);
                                rval = DDI_FAILURE;
                                goto cleanup;
                        }
                        for (i = start_phy; i < num_phys; i++, start_phy = i) {
                                cpdi[i] = ddi_get32(page_accessp,
                                    &sasioupage0->PhyData[i].
                                    ControllerPhyDeviceInfo);
                                port_flags = ddi_get8(page_accessp,
                                    &sasioupage0->PhyData[i].PortFlags);

                                mpt->m_phy_info[i].port_num =
                                    ddi_get8(page_accessp,
                                    &sasioupage0->PhyData[i].Port);
                                mpt->m_phy_info[i].ctrl_devhdl =
                                    ddi_get16(page_accessp, &sasioupage0->
                                    PhyData[i].ControllerDevHandle);
                                mpt->m_phy_info[i].attached_devhdl =
                                    ddi_get16(page_accessp, &sasioupage0->
                                    PhyData[i].AttachedDevHandle);
                                mpt->m_phy_info[i].phy_device_type = cpdi[i];
                                mpt->m_phy_info[i].port_flags = port_flags;

                                if (port_flags & DISCOVERY_IN_PROGRESS) {
                                        retrypage0++;
                                        NDBG20(("Discovery in progress, can't "
                                            "verify IO unit config, then NO.%d"
                                            " times retry", retrypage0));
                                        break;
                                } else {
                                        retrypage0 = 0;
                                }
                                if (!(port_flags & AUTO_PORT_CONFIGURATION)) {
                                        /*
                                         * some PHY configuration described in
                                         * SAS IO Unit Page1
                                         */
                                        readpage1 = 1;
                                }
                        }

                        /*
                         * retry 30 times if discovery is in process
                         */
                        if (retrypage0 && (retrypage0 < 30)) {
                                drv_usecwait(1000 * 100);
                                state = IOUC_READ_PAGE0;
                                break;
                        } else if (retrypage0 == 30) {
                                mptsas_log(mpt, CE_WARN,
                                    "!Discovery in progress, can't "
                                    "verify IO unit config, then after"
                                    " 30 times retry, give up!");
                                state = IOUC_DONE;
                                rval = DDI_FAILURE;
                                break;
```



```
                        }

                        if (readpage1 == 0) {
                                state = IOUC_DONE;
                                rval = DDI_SUCCESS;
                                break;
                        }

                        state = IOUC_READ_PAGE1;
                        break;

                case IOUC_READ_PAGE1:
                        if ((ddi_dma_sync(page_dma_handle, 0, 0,
                            DDI_DMA_SYNC_FORCPU)) != DDI_SUCCESS) {
                                goto cleanup;
                        }

                        num_phys = ddi_get8(page_accessp,
                            &sasioupage1->NumPhys);
                        ASSERT(num_phys == mpt->m_num_phys);
                        if (num_phys > MPTSAS_MAX_PHYS) {
                                mptsas_log(mpt, CE_WARN, "Number of phys "
                                    "supported by HBA (%d) is more than max "
                                    "supported by driver (%d).  Driver will "
                                    "not attach.", num_phys,
                                    MPTSAS_MAX_PHYS);
                                rval = DDI_FAILURE;
                                goto cleanup;
                        }
                        for (i = 0; i < num_phys; i++) {
                                cpdi[i] = ddi_get32(page_accessp,
                                    &sasioupage1->PhyData[i].
                                    ControllerPhyDeviceInfo);
                                port_flags = ddi_get8(page_accessp,
                                    &sasioupage1->PhyData[i].PortFlags);
                                mpt->m_phy_info[i].port_num =
                                    ddi_get8(page_accessp,
                                    &sasioupage1->PhyData[i].Port);
                                mpt->m_phy_info[i].port_flags = port_flags;
                                mpt->m_phy_info[i].phy_device_type = cpdi[i];

                        }

                        state = IOUC_DONE;
                        rval = DDI_SUCCESS;
                        break;
                }
        }
        if ((mptsas_check_dma_handle(recv_dma_handle) != DDI_SUCCESS) ||
            (mptsas_check_dma_handle(page_dma_handle) != DDI_SUCCESS)) {
                ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_UNAFFECTED);
                rval = DDI_FAILURE;
                goto cleanup;
        }
        if ((mptsas_check_acc_handle(recv_accessp) != DDI_SUCCESS) ||
            (mptsas_check_acc_handle(page_accessp) != DDI_SUCCESS)) {
                ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_UNAFFECTED);
                rval = DDI_FAILURE;
                goto cleanup;
        }

cleanup:
        if (free_recv)
                mptsas_dma_addr_destroy(&recv_dma_handle, &recv_accessp);
        if (free_page)
                mptsas_dma_addr_destroy(&page_dma_handle, &page_accessp);
```

```
        if (rval != DDI_SUCCESS) {
                mptsas_fm_ereport(mpt, DDI_FM_DEVICE_NO_RESPONSE);
                ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_LOST);
        }
        return (rval);
}

/*
 * mptsas_get_manufacture_page5
 *
 * This function will retrieve the base WWID from the adapter.  Since this
 * function is only called during the initialization process, use handshaking.
 */
int
mptsas_get_manufacture_page5(mptsas_t *mpt)
{
        ddi_dma_attr_t                  recv_dma_attrs, page_dma_attrs;
        ddi_dma_cookie_t                page_cookie;
        ddi_dma_handle_t                recv_dma_handle, page_dma_handle;
        ddi_acc_handle_t                recv_accessp, page_accessp;
        pMpi2ConfigReply_t              configreply;
        caddr_t                         recv_memp, page_memp;
        int                             recv_numbytes;
        pMpi2ManufacturingPage5_t       m5;
        uint32_t                        flagslength;
        int                             rval = DDI_SUCCESS;
        uint_t                          iocstatus;
        boolean_t               free_recv = B_FALSE, free_page = B_FALSE;

        MPTSAS_DISABLE_INTR(mpt);

        if (mptsas_send_config_request_msg(mpt, MPI2_CONFIG_ACTION_PAGE_HEADER,
            MPI2_CONFIG_PAGETYPE_MANUFACTURING, 0, 5, 0, 0, 0, 0)) {
                rval = DDI_FAILURE;
                goto done;
        }

        /*
         * dynamically create a customized dma attribute structure
         * that describes the MPT's config reply page request structure.
         */
        recv_dma_attrs = mpt->m_msg_dma_attr;
        recv_dma_attrs.dma_attr_sgllen = 1;
        recv_dma_attrs.dma_attr_granular = (sizeof (MPI2_CONFIG_REPLY));

        if (mptsas_dma_addr_create(mpt, recv_dma_attrs,
            &recv_dma_handle, &recv_accessp, &recv_memp,
            (sizeof (MPI2_CONFIG_REPLY)), NULL) == FALSE) {
                rval = DDI_FAILURE;
                goto done;
        }
        /* Now safe to call mptsas_dma_addr_destroy(recv_dma_handle). */
        free_recv = B_TRUE;

        bzero(recv_memp, sizeof (MPI2_CONFIG_REPLY));
        configreply = (pMpi2ConfigReply_t)recv_memp;
        recv_numbytes = sizeof (MPI2_CONFIG_REPLY);

        /*
         * get config reply message
         */
        if (mptsas_get_handshake_msg(mpt, recv_memp, recv_numbytes,
            recv_accessp)) {
                rval = DDI_FAILURE;
                goto done;
        }
```



```
        if (iocstatus = ddi_get16(recv_accessp, &configreply->IOCStatus)) {
                mptsas_log(mpt, CE_WARN, "mptsas_get_manufacture_page5 update: "
                    "IOCStatus=0x%x, IOCLogInfo=0x%x", iocstatus,
                    ddi_get32(recv_accessp, &configreply->IOCLogInfo));
                goto done;
        }

        /*
         * dynamically create a customized dma attribute structure
         * that describes the MPT's config page structure.
         */
        page_dma_attrs = mpt->m_msg_dma_attr;
        page_dma_attrs.dma_attr_sgllen = 1;
        page_dma_attrs.dma_attr_granular = (sizeof (MPI2_CONFIG_PAGE_MAN_5));

        if (mptsas_dma_addr_create(mpt, page_dma_attrs, &page_dma_handle,
            &page_accessp, &page_memp, (sizeof (MPI2_CONFIG_PAGE_MAN_5)),
            &page_cookie) == FALSE) {
                rval = DDI_FAILURE;
                goto done;
        }
        /* Now safe to call mptsas_dma_addr_destroy(page_dma_handle). */
        free_page = B_TRUE;

        bzero(page_memp, sizeof (MPI2_CONFIG_PAGE_MAN_5));
        m5 = (pMpi2ManufacturingPage5_t)page_memp;
        NDBG20(("mptsas_get_manufacture_page5: paddr 0x%x%08x",
            (uint32_t)(page_cookie.dmac_laddress>>32),
            (uint32_t)(page_cookie.dmac_laddress&0xffffffffUL)));

        /*
         * Give reply address to IOC to store config page in and send
         * config request out.
         */

        flagslength = sizeof (MPI2_CONFIG_PAGE_MAN_5);
        flagslength |= ((uint32_t)(MPI2_SGE_FLAGS_LAST_ELEMENT |
            MPI2_SGE_FLAGS_END_OF_BUFFER | MPI2_SGE_FLAGS_SIMPLE_ELEMENT |
            MPI2_SGE_FLAGS_SYSTEM_ADDRESS | MPI2_SGE_FLAGS_64_BIT_ADDRESSING |
            MPI2_SGE_FLAGS_IOC_TO_HOST |
            MPI2_SGE_FLAGS_END_OF_LIST) << MPI2_SGE_FLAGS_SHIFT);

        if (mptsas_send_config_request_msg(mpt,
            MPI2_CONFIG_ACTION_PAGE_READ_CURRENT,
            MPI2_CONFIG_PAGETYPE_MANUFACTURING, 0, 5,
            ddi_get8(recv_accessp, &configreply->Header.PageVersion),
            ddi_get8(recv_accessp, &configreply->Header.PageLength),
            flagslength, page_cookie.dmac_laddress)) {
                rval = DDI_FAILURE;
                goto done;
        }

        /*
         * get reply view handshake
         */
        if (mptsas_get_handshake_msg(mpt, recv_memp, recv_numbytes,
            recv_accessp)) {
                rval = DDI_FAILURE;
                goto done;
        }

        if (iocstatus = ddi_get16(recv_accessp, &configreply->IOCStatus)) {
                mptsas_log(mpt, CE_WARN, "mptsas_get_manufacture_page5 config: "
                    "IOCStatus=0x%x, IOCLogInfo=0x%x", iocstatus,
                    ddi_get32(recv_accessp, &configreply->IOCLogInfo));
```

```
                goto done;
        }

        (void) ddi_dma_sync(page_dma_handle, 0, 0, DDI_DMA_SYNC_FORCPU);

        /*
         * Fusion-MPT stores fields in little-endian format.  This is
         * why the low-order 32 bits are stored first.
         */
        mpt->un.sasaddr.m_base_wwid_lo =
            ddi_get32(page_accessp, (uint32_t *)(void *)&m5->Phy[0].WWID);
        mpt->un.sasaddr.m_base_wwid_hi =
            ddi_get32(page_accessp, (uint32_t *)(void *)&m5->Phy[0].WWID + 1);

        if (ddi_prop_update_int64(DDI_DEV_T_NONE, mpt->m_dip,
            "base-wwid", mpt->un.m_base_wwid) != DDI_PROP_SUCCESS) {
                NDBG2(("%s%d: failed to create base-wwid property",
                    ddi_driver_name(mpt->m_dip), ddi_get_instance(mpt->m_dip)));
        }

        /*
         * Set the number of PHYs present.
         */
        mpt->m_num_phys = ddi_get8(page_accessp, (uint8_t *)&m5->NumPhys);

        if (ddi_prop_update_int(DDI_DEV_T_NONE, mpt->m_dip,
            "num-phys", mpt->m_num_phys) != DDI_PROP_SUCCESS) {
                NDBG2(("%s%d: failed to create num-phys property",
                    ddi_driver_name(mpt->m_dip), ddi_get_instance(mpt->m_dip)));
        }

        mptsas_log(mpt, CE_NOTE, "!mpt%d: Initiator WWNs: 0x%016llx-0x%016llx",
            mpt->m_instance, (unsigned long long)mpt->un.m_base_wwid,
            (unsigned long long)mpt->un.m_base_wwid + mpt->m_num_phys - 1);

        if ((mptsas_check_dma_handle(recv_dma_handle) != DDI_SUCCESS) ||
            (mptsas_check_dma_handle(page_dma_handle) != DDI_SUCCESS)) {
                ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_UNAFFECTED);
                rval = DDI_FAILURE;
                goto done;
        }
        if ((mptsas_check_acc_handle(recv_accessp) != DDI_SUCCESS) ||
            (mptsas_check_acc_handle(page_accessp) != DDI_SUCCESS)) {
                ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_UNAFFECTED);
                rval = DDI_FAILURE;
        }
done:
        /*
         * free up memory
         */
        if (free_recv)
                mptsas_dma_addr_destroy(&recv_dma_handle, &recv_accessp);
        if (free_page)
                mptsas_dma_addr_destroy(&page_dma_handle, &page_accessp);
        MPTSAS_ENABLE_INTR(mpt);

        return (rval);
}

static int
mptsas_sasphypage_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
        pMpi2SasPhyPage0_t      sasphypage;
        int                     rval = DDI_SUCCESS;
        uint16_t                *owner_devhdl, *attached_devhdl;
        uint8_t                 *attached_phy_identify;
        uint32_t                *attached_phy_info;
        uint8_t                 *programmed_link_rate;
        uint8_t                 *hw_link_rate;
        uint8_t                 *change_count;
        uint32_t                *phy_info;
        uint8_t                 *negotiated_link_rate;
        uint32_t                page_address;

        if ((iocstatus != MPI2_IOCSTATUS_SUCCESS) &&
            (iocstatus != MPI2_IOCSTATUS_CONFIG_INVALID_PAGE)) {
                mptsas_log(mpt, CE_WARN, "mptsas_get_sas_expander_page0 "
                    "config: IOCStatus=0x%x, IOCLogInfo=0x%x",
                    iocstatus, iocloginfo);
                rval = DDI_FAILURE;
                return (rval);
        }
        page_address = va_arg(ap, uint32_t);
        /*
         * The INVALID_PAGE status is normal if using GET_NEXT_HANDLE and there
         * are no more pages.  If everything is OK up to this point but the
         * status is INVALID_PAGE, change rval to FAILURE and quit.  Also,
         * signal that device traversal is complete.
         */
        if (iocstatus == MPI2_IOCSTATUS_CONFIG_INVALID_PAGE) {
                if ((page_address & MPI2_SAS_EXPAND_PGAD_FORM_MASK) ==
                    MPI2_SAS_EXPAND_PGAD_FORM_GET_NEXT_HNDL) {
                        mpt->m_done_traverse_smp = 1;
                }
                rval = DDI_FAILURE;
                return (rval);
        }
        owner_devhdl = va_arg(ap, uint16_t *);
        attached_devhdl = va_arg(ap, uint16_t *);
        attached_phy_identify = va_arg(ap, uint8_t *);
        attached_phy_info = va_arg(ap, uint32_t *);
        programmed_link_rate = va_arg(ap, uint8_t *);
        hw_link_rate = va_arg(ap, uint8_t *);
        change_count = va_arg(ap, uint8_t *);
        phy_info = va_arg(ap, uint32_t *);
        negotiated_link_rate = va_arg(ap, uint8_t *);

        sasphypage = (pMpi2SasPhyPage0_t)page_memp;

        *owner_devhdl =
            ddi_get16(accessp, &sasphypage->OwnerDevHandle);
        *attached_devhdl =
            ddi_get16(accessp, &sasphypage->AttachedDevHandle);
        *attached_phy_identify =
            ddi_get8(accessp, &sasphypage->AttachedPhyIdentifier);
        *attached_phy_info =
            ddi_get32(accessp, &sasphypage->AttachedPhyInfo);
        *programmed_link_rate =
            ddi_get8(accessp, &sasphypage->ProgrammedLinkRate);
        *hw_link_rate =
            ddi_get8(accessp, &sasphypage->HwLinkRate);
        *change_count =
            ddi_get8(accessp, &sasphypage->ChangeCount);
        *phy_info =
            ddi_get32(accessp, &sasphypage->PhyInfo);
        *negotiated_link_rate =
            ddi_get8(accessp, &sasphypage->NegotiatedLinkRate);
```

```
        return (rval);
}

/*
 * Request MPI configuration page SAS phy page 0 to get DevHandle, phymask
 * and SAS address.
 */
int
mptsas_get_sas_phy_page0(mptsas_t *mpt, uint32_t page_address,
    smhba_info_t *info)
{
        int                     rval = DDI_SUCCESS;

        ASSERT(mutex_owned(&mpt->m_mutex));

        /*
         * Get the header and config page.  reply contains the reply frame,
         * which holds status info for the request.
         */
        rval = mptsas_access_config_page(mpt,
            MPI2_CONFIG_ACTION_PAGE_READ_CURRENT,
            MPI2_CONFIG_EXTPAGETYPE_SAS_PHY, 0, page_address,
            mptsas_sasphypage_0_cb, page_address, &info->owner_devhdl,
            &info->attached_devhdl, &info->attached_phy_identify,
            &info->attached_phy_info, &info->programmed_link_rate,
            &info->hw_link_rate, &info->change_count,
            &info->phy_info, &info->negotiated_link_rate);

        return (rval);
}

static int
mptsas_sasphypage_1_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
        pMpi2SasPhyPage1_t      sasphypage;
        int                     rval = DDI_SUCCESS;

        uint32_t                *invalid_dword_count;
        uint32_t                *running_disparity_error_count;
        uint32_t                *loss_of_dword_sync_count;
        uint32_t                *phy_reset_problem_count;
        uint32_t                page_address;

        if ((iocstatus != MPI2_IOCSTATUS_SUCCESS) &&
            (iocstatus != MPI2_IOCSTATUS_CONFIG_INVALID_PAGE)) {
                mptsas_log(mpt, CE_WARN, "mptsas_get_sas_expander_page1 "
                    "config: IOCStatus=0x%x, IOCLogInfo=0x%x",
                    iocstatus, iocloginfo);
                rval = DDI_FAILURE;
                return (rval);
        }
        page_address = va_arg(ap, uint32_t);
        /*
         * The INVALID_PAGE status is normal if using GET_NEXT_HANDLE and there
         * are no more pages.  If everything is OK up to this point but the
         * status is INVALID_PAGE, change rval to FAILURE and quit.  Also,
         * signal that device traversal is complete.
         */
        if (iocstatus == MPI2_IOCSTATUS_CONFIG_INVALID_PAGE) {
                if ((page_address & MPI2_SAS_EXPAND_PGAD_FORM_MASK) ==
```



```
                        MPI2_SAS_EXPAND_PGAD_FORM_GET_NEXT_HNDL) {
                            mpt->m_done_traverse_smp = 1;
                    }
                    rval = DDI_FAILURE;
                    return (rval);
        }

        invalid_dword_count = va_arg(ap, uint32_t *);
        running_disparity_error_count = va_arg(ap, uint32_t *);
        loss_of_dword_sync_count = va_arg(ap, uint32_t *);
        phy_reset_problem_count = va_arg(ap, uint32_t *);

        sasphypage = (pMpi2SasPhyPage1_t)page_memp;

        *invalid_dword_count =
            ddi_get32(accessp, &sasphypage->InvalidDwordCount);
        *running_disparity_error_count =
            ddi_get32(accessp, &sasphypage->RunningDisparityErrorCount);
        *loss_of_dword_sync_count =
            ddi_get32(accessp, &sasphypage->LossDwordSynchCount);
        *phy_reset_problem_count =
            ddi_get32(accessp, &sasphypage->PhyResetProblemCount);

        return (rval);
}

/*
 * Request MPI configuration page SAS phy page 0 to get DevHandle, phymask
 * and SAS address.
 */
int
mptsas_get_sas_phy_page1(mptsas_t *mpt, uint32_t page_address,
    smhba_info_t *info)
{
        int                     rval = DDI_SUCCESS;

        ASSERT(mutex_owned(&mpt->m_mutex));

        /*
         * Get the header and config page.  reply contains the reply frame,
         * which holds status info for the request.
         */
        rval = mptsas_access_config_page(mpt,
            MPI2_CONFIG_ACTION_PAGE_READ_CURRENT,
            MPI2_CONFIG_EXTPAGETYPE_SAS_PHY, 1, page_address,
            mptsas_sasphypage_1_cb, page_address,
            &info->invalid_dword_count,
            &info->running_disparity_error_count,
            &info->loss_of_dword_sync_count,
            &info->phy_reset_problem_count);

        return (rval);
}
/*
 * mptsas_get_manufacture_page0
 *
 * This function will retrieve the base
 * Chip name, Board Name,Board Trace number from the adapter.
 * Since this function is only called during the
 * initialization process, use handshaking.
 */
int
mptsas_get_manufacture_page0(mptsas_t *mpt)
{
        ddi_dma_attr_t                  recv_dma_attrs, page_dma_attrs;
        ddi_dma_cookie_t                page_cookie;
```

```
        ddi_dma_handle_t                recv_dma_handle, page_dma_handle;
        ddi_acc_handle_t                recv_accessp, page_accessp;
        pMpi2ConfigReply_t              configreply;
        caddr_t                         recv_memp, page_memp;
        int                             recv_numbytes;
        pMpi2ManufacturingPage0_t       m0;
        uint32_t                        flagslength;
        int                             rval = DDI_SUCCESS;
        uint_t                          iocstatus;
        uint8_t                         i = 0;
        boolean_t               free_recv = B_FALSE, free_page = B_FALSE;

        MPTSAS_DISABLE_INTR(mpt);

        if (mptsas_send_config_request_msg(mpt, MPI2_CONFIG_ACTION_PAGE_HEADER,
            MPI2_CONFIG_PAGETYPE_MANUFACTURING, 0, 0, 0, 0, 0, 0)) {
                rval = DDI_FAILURE;
                goto done;
        }

        /*
         * dynamically create a customized dma attribute structure
         * that describes the MPT's config reply page request structure.
         */
        recv_dma_attrs = mpt->m_msg_dma_attr;
        recv_dma_attrs.dma_attr_sgllen = 1;
        recv_dma_attrs.dma_attr_granular = (sizeof (MPI2_CONFIG_REPLY));

        if (mptsas_dma_addr_create(mpt, recv_dma_attrs, &recv_dma_handle,
            &recv_accessp, &recv_memp, (sizeof (MPI2_CONFIG_REPLY)),
            NULL) == FALSE) {
                rval = DDI_FAILURE;
                goto done;
        }
        /* Now safe to call mptsas_dma_addr_destroy(recv_dma_handle). */
        free_recv = B_TRUE;

        bzero(recv_memp, sizeof (MPI2_CONFIG_REPLY));
        configreply = (pMpi2ConfigReply_t)recv_memp;
        recv_numbytes = sizeof (MPI2_CONFIG_REPLY);

        /*
         * get config reply message
         */
        if (mptsas_get_handshake_msg(mpt, recv_memp, recv_numbytes,
            recv_accessp)) {
                rval = DDI_FAILURE;
                goto done;
        }

        if (iocstatus = ddi_get16(recv_accessp, &configreply->IOCStatus)) {
                mptsas_log(mpt, CE_WARN, "mptsas_get_manufacture_page5 update: "
                    "IOCStatus=0x%x, IOCLogInfo=0x%x", iocstatus,
                    ddi_get32(recv_accessp, &configreply->IOCLogInfo));
                goto done;
        }

        /*
         * dynamically create a customized dma attribute structure
         * that describes the MPT's config page structure.
         */
        page_dma_attrs = mpt->m_msg_dma_attr;
        page_dma_attrs.dma_attr_sgllen = 1;
        page_dma_attrs.dma_attr_granular = (sizeof (MPI2_CONFIG_PAGE_MAN_0));

        if (mptsas_dma_addr_create(mpt, page_dma_attrs, &page_dma_handle,
```



```
            &page_accessp, &page_memp, (sizeof (MPI2_CONFIG_PAGE_MAN_0)),
            &page_cookie) == FALSE) {
                rval = DDI_FAILURE;
                goto done;
        }
        /* Now safe to call mptsas_dma_addr_destroy(page_dma_handle). */
        free_page = B_TRUE;

        bzero(page_memp, sizeof (MPI2_CONFIG_PAGE_MAN_0));
        m0 = (pMpi2ManufacturingPage0_t)page_memp;

        /*
         * Give reply address to IOC to store config page in and send
         * config request out.
         */

        flagslength = sizeof (MPI2_CONFIG_PAGE_MAN_0);
        flagslength |= ((uint32_t)(MPI2_SGE_FLAGS_LAST_ELEMENT |
            MPI2_SGE_FLAGS_END_OF_BUFFER | MPI2_SGE_FLAGS_SIMPLE_ELEMENT |
            MPI2_SGE_FLAGS_SYSTEM_ADDRESS | MPI2_SGE_FLAGS_64_BIT_ADDRESSING |
            MPI2_SGE_FLAGS_IOC_TO_HOST |
            MPI2_SGE_FLAGS_END_OF_LIST) << MPI2_SGE_FLAGS_SHIFT);

        if (mptsas_send_config_request_msg(mpt,
            MPI2_CONFIG_ACTION_PAGE_READ_CURRENT,
            MPI2_CONFIG_PAGETYPE_MANUFACTURING, 0, 0,
            ddi_get8(recv_accessp, &configreply->Header.PageVersion),
            ddi_get8(recv_accessp, &configreply->Header.PageLength),
            flagslength, page_cookie.dmac_laddress)) {
                rval = DDI_FAILURE;
                goto done;
        }

        /*
         * get reply view handshake
         */
        if (mptsas_get_handshake_msg(mpt, recv_memp, recv_numbytes,
            recv_accessp)) {
                rval = DDI_FAILURE;
                goto done;
        }

        if (iocstatus = ddi_get16(recv_accessp, &configreply->IOCStatus)) {
                mptsas_log(mpt, CE_WARN, "mptsas_get_manufacture_page0 config: "
                    "IOCStatus=0x%x, IOCLogInfo=0x%x", iocstatus,
                    ddi_get32(recv_accessp, &configreply->IOCLogInfo));
                goto done;
        }

        (void) ddi_dma_sync(page_dma_handle, 0, 0, DDI_DMA_SYNC_FORCPU);

        /*
         * Fusion-MPT stores fields in little-endian format.  This is
         * why the low-order 32 bits are stored first.
         */

        for (i = 0; i < 16; i++) {
                mpt->m_MANU_page0.ChipName[i] =
                    ddi_get8(page_accessp,
                    (uint8_t *)(void *)&m0->ChipName[i]);
        }

        for (i = 0; i < 8; i++) {
                mpt->m_MANU_page0.ChipRevision[i] =
                    ddi_get8(page_accessp,
                    (uint8_t *)(void *)&m0->ChipRevision[i]);
```

```
        }

        for (i = 0; i < 16; i++) {
                mpt->m_MANU_page0.BoardName[i] =
                    ddi_get8(page_accessp,
                    (uint8_t *)(void *)&m0->BoardName[i]);
        }

        for (i = 0; i < 16; i++) {
                mpt->m_MANU_page0.BoardAssembly[i] =
                    ddi_get8(page_accessp,
                    (uint8_t *)(void *)&m0->BoardAssembly[i]);
        }

        for (i = 0; i < 16; i++) {
                mpt->m_MANU_page0.BoardTracerNumber[i] =
                    ddi_get8(page_accessp,
                    (uint8_t *)(void *)&m0->BoardTracerNumber[i]);
        }

        if ((mptsas_check_dma_handle(recv_dma_handle) != DDI_SUCCESS) ||
            (mptsas_check_dma_handle(page_dma_handle) != DDI_SUCCESS)) {
                ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_UNAFFECTED);
                rval = DDI_FAILURE;
                goto done;
        }
        if ((mptsas_check_acc_handle(recv_accessp) != DDI_SUCCESS) ||
            (mptsas_check_acc_handle(page_accessp) != DDI_SUCCESS)) {
                ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_UNAFFECTED);
                rval = DDI_FAILURE;
        }
done:
        /*
         * free up memory
         */
        if (free_recv)
                mptsas_dma_addr_destroy(&recv_dma_handle, &recv_accessp);
        if (free_page)
                mptsas_dma_addr_destroy(&page_dma_handle, &page_accessp);
        MPTSAS_ENABLE_INTR(mpt);

        return (rval);
}
#endif /* ! codereview */
```

```
**********************************************************
   21016 Thu Jun 12 17:28:22 2014
new/usr/src/uts/common/io/scsi/adapters/mpt_sas3/mptsas3_init.c
4546 mpt_sas needs enhancing to support LSI MPI2.5
**********************************************************
/*
 * 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.
 */

/*
 * 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>
#pragma pack()
/*
 * private header files.
 */
#include <sys/scsi/adapters/mpt_sas3/mptsas3_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)
{
        switch (mpt->m_devid) {
        case MPI2_MFGPAGE_DEVID_SAS2008:
                return ("SAS2008");
        case MPI2_MFGPAGE_DEVID_SAS2004:
                return ("SAS2004");
        case MPI2_MFGPAGE_DEVID_SAS2108_1:
        case MPI2_MFGPAGE_DEVID_SAS2108_2:
        case MPI2_MFGPAGE_DEVID_SAS2108_3:
                return ("SAS2108");
        case MPI2_MFGPAGE_DEVID_SAS2116_1:
        case MPI2_MFGPAGE_DEVID_SAS2116_2:
                return ("SAS2116");
        case MPI2_MFGPAGE_DEVID_SAS2208_1:
        case MPI2_MFGPAGE_DEVID_SAS2208_2:
        case MPI2_MFGPAGE_DEVID_SAS2208_3:
        case MPI2_MFGPAGE_DEVID_SAS2208_4:
        case MPI2_MFGPAGE_DEVID_SAS2208_5:
        case MPI2_MFGPAGE_DEVID_SAS2208_6:
#if 0
        /* These are the same as the next 2?? */
        case MPI2_MFGPAGE_DEVID_SAS2208_7:
```

```
        case MPI2_MFGPAGE_DEVID_SAS2208_8:
#endif
                return ("SAS2208");
        case MPI2_MFGPAGE_DEVID_SAS2308_1:
        case MPI2_MFGPAGE_DEVID_SAS2308_2:
        case MPI2_MFGPAGE_DEVID_SAS2308_3:
                return ("SAS2308");
        case MPI25_MFGPAGE_DEVID_SAS3004:
                return ("SAS3004");
        case MPI25_MFGPAGE_DEVID_SAS3008:
                return ("SAS3008");
        case MPI25_MFGPAGE_DEVID_SAS3108_1:
        case MPI25_MFGPAGE_DEVID_SAS3108_2:
        case MPI25_MFGPAGE_DEVID_SAS3108_3:
        case MPI25_MFGPAGE_DEVID_SAS3108_4:
        case MPI25_MFGPAGE_DEVID_SAS3108_5:
        case MPI25_MFGPAGE_DEVID_SAS3108_6:
                return ("SAS3108");
        default:
                return ("?");
        }
}

int
mptsas_ioc_get_facts(mptsas_t *mpt)
{
        /*
         * Send get facts messages
         */
        if (mptsas_do_dma(mpt, sizeof (MPI2_IOC_FACTS_REQUEST), NULL,
            mptsas_ioc_do_get_facts)) {
                return (DDI_FAILURE);
        }

        /*
         * Get facts reply messages
         */
        if (mptsas_do_dma(mpt, sizeof (MPI2_IOC_FACTS_REPLY), NULL,
            mptsas_ioc_do_get_facts_reply)) {
                return (DDI_FAILURE);
        }

        return (DDI_SUCCESS);
}

static int
mptsas_ioc_do_get_facts(mptsas_t *mpt, caddr_t memp, int var,
                ddi_acc_handle_t accessp)
{
#ifndef __lock_lint
        _NOTE(ARGUNUSED(var))
#endif
        pMpi2IOCFactsRequest_t  facts;
        int                     numbytes;

        bzero(memp, sizeof (*facts));
        facts = (void *)memp;
        ddi_put8(accessp, &facts->Function, MPI2_FUNCTION_IOC_FACTS);
        numbytes = sizeof (*facts);

        /*
         * Post message via handshake
         */
        if (mptsas_send_handshake_msg(mpt, memp, numbytes, accessp)) {
                return (DDI_FAILURE);
        }
```



```
        return (DDI_SUCCESS);
}

static int
mptsas_ioc_do_get_facts_reply(mptsas_t *mpt, caddr_t memp, int var,
                ddi_acc_handle_t accessp)
{
#ifndef __lock_lint
        _NOTE(ARGUNUSED(var))
#endif

        pMpi2IOCFactsReply_t    factsreply;
        int                     numbytes;
        uint_t                  iocstatus;
        char                    buf[32];
        uint16_t                numReplyFrames;
        uint16_t                queueSize, queueDiff;
        int                     simple_sge_main;
        int                     simple_sge_next;
        uint32_t                capabilities;
        uint16_t                msgversion;

        bzero(memp, sizeof (*factsreply));
        factsreply = (void *)memp;
        numbytes = sizeof (*factsreply);

        /*
         * get ioc facts reply message
         */
        if (mptsas_get_handshake_msg(mpt, memp, numbytes, accessp)) {
                return (DDI_FAILURE);
        }

        if (iocstatus = ddi_get16(accessp, &factsreply->IOCStatus)) {
                mptsas_log(mpt, CE_WARN, "mptsas_ioc_do_get_facts_reply: "
                    "IOCStatus=0x%x, IOCLogInfo=0x%x", iocstatus,
                    ddi_get32(accessp, &factsreply->IOCLogInfo));
                return (DDI_FAILURE);
        }

        /*
         * store key values from reply to mpt structure
         */
        mpt->m_fwversion = ddi_get32(accessp, &factsreply->FWVersion.Word);
        mpt->m_productid = ddi_get16(accessp, &factsreply->ProductID);


        (void) sprintf(buf, "%u.%u.%u.%u",
            ddi_get8(accessp, &factsreply->FWVersion.Struct.Major),
            ddi_get8(accessp, &factsreply->FWVersion.Struct.Minor),
            ddi_get8(accessp, &factsreply->FWVersion.Struct.Unit),
            ddi_get8(accessp, &factsreply->FWVersion.Struct.Dev));
        mptsas_log(mpt, CE_NOTE, "?MPT Firmware version v%s (%s)\n",
            buf, mptsas_devid_type_string(mpt));
        (void) ddi_prop_update_string(DDI_DEV_T_NONE, mpt->m_dip,
            "firmware-version", buf);

        /*
         * Set up request info.
         */
        mpt->m_max_requests = ddi_get16(accessp,
            &factsreply->RequestCredit) - 1;
        mpt->m_req_frame_size = ddi_get16(accessp,
            &factsreply->IOCRequestFrameSize) * 4;
```

```
        /*
         * Size of reply free queue should be the number of requests
         * plus some additional for events (32).  Make sure number of
         * reply frames is not a multiple of 16 so that the queue sizes
         * are calculated correctly later to be a multiple of 16.
         */
        mpt->m_reply_frame_size = ddi_get8(accessp,
            &factsreply->ReplyFrameSize) * 4;
        numReplyFrames = mpt->m_max_requests + 32;
        if (!(numReplyFrames % 16)) {
                numReplyFrames--;
        }
        mpt->m_max_replies = numReplyFrames;
        queueSize = numReplyFrames;
        queueSize += 16 - (queueSize % 16);
        mpt->m_free_queue_depth = queueSize;

        /*
         * Size of reply descriptor post queue should be the number of
         * request frames + the number of reply frames + 1 and needs to
         * be a multiple of 16.  This size can be no larger than
         * MaxReplyDescriptorPostQueueDepth from IOCFacts.  If the
         * calculated queue size is larger than allowed, subtract a
         * multiple of 16 from m_max_requests, m_max_replies, and
         * m_reply_free_depth.
         *
         * There is no indication in the spec that you can reduce the
         * queue size if you have many.
         */
        queueSize = mpt->m_max_requests + numReplyFrames + 1;
        if (queueSize % 16) {
                queueSize += 16 - (queueSize % 16);
        }
        mpt->m_post_queue_depth = ddi_get16(accessp,
            &factsreply->MaxReplyDescriptorPostQueueDepth);
        if (queueSize > mpt->m_post_queue_depth) {
                queueDiff = queueSize - mpt->m_post_queue_depth;
                if (queueDiff % 16) {
                        queueDiff += 16 - (queueDiff % 16);
                }
                mpt->m_max_requests -= queueDiff;
                mpt->m_max_replies -= queueDiff;
                mpt->m_free_queue_depth -= queueDiff;
                queueSize -= queueDiff;
        }
        mpt->m_post_queue_depth = queueSize;

        /*
         * Set up max chain depth.
         */
        mpt->m_max_chain_depth = ddi_get8(accessp,
            &factsreply->MaxChainDepth);
        mpt->m_ioc_capabilities = ddi_get32(accessp,
            &factsreply->IOCCapabilities);
        if (mpt->m_ioc_capabilities & MPI2_IOCFACTS_CAPABILITY_MSI_X_INDEX) {
                mpt->m_max_msix_vectors = ddi_get8(accessp,
                    &factsreply->MaxMSIxVectors);
        }

        /*
         * Set flag to check for SAS3 support.
         */
        msgversion = ddi_get16(accessp, &factsreply->MsgVersion);
        if (msgversion == MPI2_VERSION_02_05) {
                mptsas_log(mpt, CE_NOTE, "?mpt_sas3%d SAS 3 Supported\n",
                    mpt->m_instance);
```



```
                mpt->m_MPI25 = TRUE;
        } else {
                mptsas_log(mpt, CE_NOTE, "?mpt_sas3%d MPI Version 0x%x\n",
                    mpt->m_instance, msgversion);
        }

        /*
         * Calculate max frames per request based on DMA S/G length.
         */
        simple_sge_main = MPTSAS_MAX_FRAME_SGES64(mpt) - 1;
        simple_sge_next = mpt->m_req_frame_size /
            (mpt->m_MPI25 ? sizeof (MPI2_IEEE_SGE_SIMPLE64) :
            sizeof (MPI2_SGE_SIMPLE64)) - 1;

        mpt->m_max_request_frames = (MPTSAS_MAX_DMA_SEGS -
            simple_sge_main) / simple_sge_next + 1;
        if (((MPTSAS_MAX_DMA_SEGS - simple_sge_main) %
            simple_sge_next) > 1) {
                mpt->m_max_request_frames++;
        }

        /*
         * Check if controller supports FW diag buffers and set flag to enable
         * each type.
         */
        capabilities = mpt->m_ioc_capabilities;
        if (capabilities & MPI2_IOCFACTS_CAPABILITY_DIAG_TRACE_BUFFER) {
                mpt->m_fw_diag_buffer_list[MPI2_DIAG_BUF_TYPE_TRACE].enabled =
                    TRUE;
        }
        if (capabilities & MPI2_IOCFACTS_CAPABILITY_SNAPSHOT_BUFFER) {
                mpt->m_fw_diag_buffer_list[MPI2_DIAG_BUF_TYPE_SNAPSHOT].
                    enabled = TRUE;
        }
        if (capabilities & MPI2_IOCFACTS_CAPABILITY_EXTENDED_BUFFER) {
                mpt->m_fw_diag_buffer_list[MPI2_DIAG_BUF_TYPE_EXTENDED].
                    enabled = TRUE;
        }

        /*
         * Check if controller supports replaying events when issuing Message
         * Unit Reset and set flag to enable MUR.
         */
        if (capabilities & MPI2_IOCFACTS_CAPABILITY_EVENT_REPLAY) {
                mpt->m_event_replay = TRUE;
        }

        /*
         * Check if controller supports IR.
         */
        if (capabilities & MPI2_IOCFACTS_CAPABILITY_INTEGRATED_RAID) {
                mpt->m_ir_capable = TRUE;
        }

        return (DDI_SUCCESS);
}

int
mptsas_ioc_get_port_facts(mptsas_t *mpt, int port)
{
        /*
         * Send get port facts message
         */
        if (mptsas_do_dma(mpt, sizeof (MPI2_PORT_FACTS_REQUEST), port,
            mptsas_ioc_do_get_port_facts)) {
                return (DDI_FAILURE);
```

```
        }

        /*
         * Get port facts reply message
         */
        if (mptsas_do_dma(mpt, sizeof (MPI2_PORT_FACTS_REPLY), port,
            mptsas_ioc_do_get_port_facts_reply)) {
                return (DDI_FAILURE);
        }

        return (DDI_SUCCESS);
}

static int
mptsas_ioc_do_get_port_facts(mptsas_t *mpt, caddr_t memp, int var,
                        ddi_acc_handle_t accessp)
{
        pMpi2PortFactsRequest_t facts;
        int                     numbytes;

        bzero(memp, sizeof (*facts));
        facts = (void *)memp;
        ddi_put8(accessp, &facts->Function, MPI2_FUNCTION_PORT_FACTS);
        ddi_put8(accessp, &facts->PortNumber, var);
        numbytes = sizeof (*facts);

        /*
         * Send port facts message via handshake
         */
        if (mptsas_send_handshake_msg(mpt, memp, numbytes, accessp)) {
                return (DDI_FAILURE);
        }

        return (DDI_SUCCESS);
}

static int
mptsas_ioc_do_get_port_facts_reply(mptsas_t *mpt, caddr_t memp, int var,
                                ddi_acc_handle_t accessp)
{
#ifndef __lock_lint
        _NOTE(ARGUNUSED(var))
#endif
        pMpi2PortFactsReply_t   factsreply;
        int                     numbytes;
        uint_t                  iocstatus;

        bzero(memp, sizeof (*factsreply));
        factsreply = (void *)memp;
        numbytes = sizeof (*factsreply);

        /*
         * Get port facts reply message via handshake
         */
        if (mptsas_get_handshake_msg(mpt, memp, numbytes, accessp)) {
                return (DDI_FAILURE);
        }

        if (iocstatus = ddi_get16(accessp, &factsreply->IOCStatus)) {
                mptsas_log(mpt, CE_WARN, "mptsas_ioc_do_get_port_facts_reply: "
                    "IOCStatus=0x%x, IOCLogInfo=0x%x", iocstatus,
                    ddi_get32(accessp, &factsreply->IOCLogInfo));
                return (DDI_FAILURE);
        }

        return (DDI_SUCCESS);
```



```
}

int
mptsas_ioc_enable_port(mptsas_t *mpt)
{
        /*
         * Send enable port message
         */
        if (mptsas_do_dma(mpt, sizeof (MPI2_PORT_ENABLE_REQUEST), 0,
            mptsas_ioc_do_enable_port)) {
                return (DDI_FAILURE);
        }

        /*
         * Get enable port reply message
         */
        if (mptsas_do_dma(mpt, sizeof (MPI2_PORT_ENABLE_REPLY), 0,
            mptsas_ioc_do_enable_port_reply)) {
                return (DDI_FAILURE);
        }

        return (DDI_SUCCESS);
}

static int
mptsas_ioc_do_enable_port(mptsas_t *mpt, caddr_t memp, int var,
        ddi_acc_handle_t accessp)
{
#ifndef __lock_lint
        _NOTE(ARGUNUSED(var))
#endif
        pMpi2PortEnableRequest_t        enable;
        int                             numbytes;

        bzero(memp, sizeof (*enable));
        enable = (void *)memp;
        ddi_put8(accessp, &enable->Function, MPI2_FUNCTION_PORT_ENABLE);
        numbytes = sizeof (*enable);

        /*
         * Send message via handshake
         */
        if (mptsas_send_handshake_msg(mpt, memp, numbytes, accessp)) {
                return (DDI_FAILURE);
        }

        return (DDI_SUCCESS);
}

static int
mptsas_ioc_do_enable_port_reply(mptsas_t *mpt, caddr_t memp, int var,
        ddi_acc_handle_t accessp)
{
#ifndef __lock_lint
        _NOTE(ARGUNUSED(var))
#endif

        int                     numbytes;
        uint_t                  iocstatus;
        pMpi2PortEnableReply_t  portreply;

        numbytes = sizeof (MPI2_PORT_ENABLE_REPLY);
        bzero(memp, numbytes);
        portreply = (void *)memp;

        /*
```

```
         * Get message via handshake
         */
        if (mptsas_get_handshake_msg(mpt, memp, numbytes, accessp)) {
                return (DDI_FAILURE);
        }

        if (iocstatus = ddi_get16(accessp, &portreply->IOCStatus)) {
                mptsas_log(mpt, CE_WARN, "mptsas_ioc_do_enable_port_reply: "
                    "IOCStatus=0x%x, IOCLogInfo=0x%x", iocstatus,
                    ddi_get32(accessp, &portreply->IOCLogInfo));
                return (DDI_FAILURE);
        }

        return (DDI_SUCCESS);
}

int
mptsas_ioc_enable_event_notification(mptsas_t *mpt)
{
        ASSERT(mutex_owned(&mpt->m_mutex));

        /*
         * Send enable event notification message
         */
        if (mptsas_do_dma(mpt, sizeof (MPI2_EVENT_NOTIFICATION_REQUEST), NULL,
            mptsas_ioc_do_enable_event_notification)) {
                return (DDI_FAILURE);
        }

        /*
         * Get enable event reply message
         */
        if (mptsas_do_dma(mpt, sizeof (MPI2_EVENT_NOTIFICATION_REPLY), NULL,
            mptsas_ioc_do_enable_event_notification_reply)) {
                return (DDI_FAILURE);
        }

        return (DDI_SUCCESS);
}

static int
mptsas_ioc_do_enable_event_notification(mptsas_t *mpt, caddr_t memp, int var,
        ddi_acc_handle_t accessp)
{
#ifndef __lock_lint
        _NOTE(ARGUNUSED(var))
#endif

        pMpi2EventNotificationRequest_t event;
        int                             numbytes;

        bzero(memp, sizeof (*event));
        event = (void *)memp;
        ddi_put8(accessp, &event->Function, MPI2_FUNCTION_EVENT_NOTIFICATION);
        numbytes = sizeof (*event);

        /*
         * Send message via handshake
         */
        if (mptsas_send_handshake_msg(mpt, memp, numbytes, accessp)) {
                return (DDI_FAILURE);
        }

        return (DDI_SUCCESS);
}
```



```
static int
mptsas_ioc_do_enable_event_notification_reply(mptsas_t *mpt, caddr_t memp,
    int var, ddi_acc_handle_t accessp)
{
#ifndef __lock_lint
        _NOTE(ARGUNUSED(var))
#endif
        int                             numbytes;
        uint_t                          iocstatus;
        pMpi2EventNotificationReply_t   eventsreply;

        numbytes = sizeof (MPI2_EVENT_NOTIFICATION_REPLY);
        bzero(memp, numbytes);
        eventsreply = (void *)memp;

        /*
         * Get message via handshake
         */
        if (mptsas_get_handshake_msg(mpt, memp, numbytes, accessp)) {
                return (DDI_FAILURE);
        }

        if (iocstatus = ddi_get16(accessp, &eventsreply->IOCStatus)) {
                mptsas_log(mpt, CE_WARN,
                    "mptsas_ioc_do_enable_event_notification_reply: "
                    "IOCStatus=0x%x, IOCLogInfo=0x%x", iocstatus,
                    ddi_get32(accessp, &eventsreply->IOCLogInfo));
                return (DDI_FAILURE);
        }

        return (DDI_SUCCESS);
}

int
mptsas_ioc_init(mptsas_t *mpt)
{
        /*
         * Send ioc init message
         */
        if (mptsas_do_dma(mpt, sizeof (MPI2_IOC_INIT_REQUEST), NULL,
            mptsas_do_ioc_init)) {
                return (DDI_FAILURE);
        }

        /*
         * Get ioc init reply message
         */
        if (mptsas_do_dma(mpt, sizeof (MPI2_IOC_INIT_REPLY), NULL,
            mptsas_do_ioc_init_reply)) {
                return (DDI_FAILURE);
        }

        return (DDI_SUCCESS);
}

static int
mptsas_do_ioc_init(mptsas_t *mpt, caddr_t memp, int var,
    ddi_acc_handle_t accessp)
{
#ifndef __lock_lint
        _NOTE(ARGUNUSED(var))
#endif

        pMpi2IOCInitRequest_t   init;
        int                     numbytes;
        timespec_t              time;
```

```
        uint64_t                mSec;

        bzero(memp, sizeof (*init));
        init = (void *)memp;
        ddi_put8(accessp, &init->Function, MPI2_FUNCTION_IOC_INIT);
        ddi_put8(accessp, &init->WhoInit, MPI2_WHOINIT_HOST_DRIVER);
        ddi_put16(accessp, &init->MsgVersion, MPI2_VERSION);
        ddi_put16(accessp, &init->HeaderVersion, MPI2_HEADER_VERSION);
        if (mpt->m_intr_type == DDI_INTR_TYPE_MSIX) {
                ddi_put8(accessp, &init->HostMSIxVectors, mpt->m_intr_cnt);
        }
        ddi_put16(accessp, &init->SystemRequestFrameSize,
            mpt->m_req_frame_size / 4);
        ddi_put16(accessp, &init->ReplyDescriptorPostQueueDepth,
            mpt->m_post_queue_depth);
        ddi_put16(accessp, &init->ReplyFreeQueueDepth,
            mpt->m_free_queue_depth);

        /*
         * These addresses are set using the DMA cookie addresses from when the
         * memory was allocated.  Sense buffer hi address should be 0.
         */
        ddi_put32(accessp, &init->SenseBufferAddressHigh,
            (uint32_t)(mpt->m_req_sense_dma_addr >> 32));
        ddi_put32(accessp, &init->SystemReplyAddressHigh,
            (uint32_t)(mpt->m_reply_frame_dma_addr >> 32));
        ddi_put32(accessp, &init->SystemRequestFrameBaseAddress.High,
            (uint32_t)(mpt->m_req_frame_dma_addr >> 32));
        ddi_put32(accessp, &init->SystemRequestFrameBaseAddress.Low,
            (uint32_t)mpt->m_req_frame_dma_addr);
        ddi_put32(accessp, &init->ReplyDescriptorPostQueueAddress.High,
            (uint32_t)(mpt->m_post_queue_dma_addr >> 32));
        ddi_put32(accessp, &init->ReplyDescriptorPostQueueAddress.Low,
            (uint32_t)mpt->m_post_queue_dma_addr);
        ddi_put32(accessp, &init->ReplyFreeQueueAddress.High,
            (uint32_t)(mpt->m_free_queue_dma_addr >> 32));
        ddi_put32(accessp, &init->ReplyFreeQueueAddress.Low,
            (uint32_t)mpt->m_free_queue_dma_addr);

        /*
         * Fill in the timestamp with the number of milliseconds since midnight
         * of January 1, 1970 UT (Greenwich Mean Time).  Time is returned in
         * seconds and nanoseconds.  Translate both to milliseconds and add
         * them together to get total milliseconds.
         */
        gethrestime(&time);
        mSec = time.tv_sec * MILLISEC;
        mSec += (time.tv_nsec / MICROSEC);
        ddi_put32(accessp, &init->TimeStamp.High, (uint32_t)(mSec >> 32));
        ddi_put32(accessp, &init->TimeStamp.Low, (uint32_t)mSec);

        numbytes = sizeof (*init);

        /*
         * Post message via handshake
         */
        if (mptsas_send_handshake_msg(mpt, memp, numbytes, accessp)) {
                return (DDI_FAILURE);
        }

        return (DDI_SUCCESS);
}

static int
mptsas_do_ioc_init_reply(mptsas_t *mpt, caddr_t memp, int var,
                ddi_acc_handle_t accessp)
```



```
{
#ifndef __lock_lint
        _NOTE(ARGUNUSED(var))
#endif

        pMpi2IOCInitReply_t     initreply;
        int                     numbytes;
        uint_t                  iocstatus;

        numbytes = sizeof (MPI2_IOC_INIT_REPLY);
        bzero(memp, numbytes);
        initreply = (void *)memp;

        /*
         * Get reply message via handshake
         */
        if (mptsas_get_handshake_msg(mpt, memp, numbytes, accessp)) {
                return (DDI_FAILURE);
        }

        if (iocstatus = ddi_get16(accessp, &initreply->IOCStatus)) {
                mptsas_log(mpt, CE_WARN, "mptsas_do_ioc_init_reply: "
                    "IOCStatus=0x%x, IOCLogInfo=0x%x", iocstatus,
                    ddi_get32(accessp, &initreply->IOCLogInfo));
                return (DDI_FAILURE);
        }

        if ((ddi_get32(mpt->m_datap, &mpt->m_reg->Doorbell)) &
            MPI2_IOC_STATE_OPERATIONAL) {
                mptsas_log(mpt, CE_NOTE,
                    "?mpt%d: IOC Operational.\n", mpt->m_instance);
        } else {
                return (DDI_FAILURE);
        }

        return (DDI_SUCCESS);
}
#endif /* ! codereview */
```

```
**********************************************************
   22084 Thu Jun 12 17:28:22 2014
new/usr/src/uts/common/io/scsi/adapters/mpt_sas3/mptsas3_raid.c
4546 mpt_sas needs enhancing to support LSI MPI2.5
**********************************************************
/*
 * 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.
 */

/*
 * 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>

#pragma pack()

/*
 * private header files.
 */
#include <sys/scsi/adapters/mpt_sas3/mptsas3_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;
        int rval = DDI_SUCCESS, i;
        uint8_t numelements, vol, disk;
        uint16_t elementtype, voldevhandle;
        uint16_t etype_vol, etype_pd, etype_hs;
        uint16_t etype_oce;
        m_raidconfig_t *raidconfig;
        uint64_t raidwwn;
        uint32_t native;
        mptsas_target_t *ptgt;
        uint32_t configindex;

        if (iocstatus == MPI2_IOCSTATUS_CONFIG_INVALID_PAGE) {
                return (DDI_FAILURE);
        }

        if (iocstatus != MPI2_IOCSTATUS_SUCCESS) {
                mptsas_log(mpt, CE_WARN, "mptsas_get_raid_conf_page0 "
                    "config: IOCStatus=0x%x, IOCLogInfo=0x%x",
                    iocstatus, iocloginfo);
```

```
                rval = DDI_FAILURE;
                return (rval);
        }
        confignum = va_arg(ap,  uint32_t *);
        configindex = va_arg(ap, uint32_t);
        raidconfig_page0 = (pMpi2RaidConfigurationPage0_t)page_memp;
        /*
         * Get all RAID configurations.
         */
        etype_vol = MPI2_RAIDCONFIG0_EFLAGS_VOLUME_ELEMENT;
        etype_pd = MPI2_RAIDCONFIG0_EFLAGS_VOL_PHYS_DISK_ELEMENT;
        etype_hs = MPI2_RAIDCONFIG0_EFLAGS_HOT_SPARE_ELEMENT;
        etype_oce = MPI2_RAIDCONFIG0_EFLAGS_OCE_ELEMENT;
        /*
         * Set up page address for next time through.
         */
        *confignum =  ddi_get8(accessp,
            &raidconfig_page0->ConfigNum);

        /*
         * Point to the right config in the structure.
         * Increment the number of valid RAID configs.
         */
        raidconfig = &mpt->m_raidconfig[configindex];
        mpt->m_num_raid_configs++;

        /*
         * Set the native flag if this is not a foreign
         * configuration.
         */
        native = ddi_get32(accessp, &raidconfig_page0->Flags);
        if (native & MPI2_RAIDCONFIG0_FLAG_FOREIGN_CONFIG) {
                native = FALSE;
        } else {
                native = TRUE;
        }
        raidconfig->m_native = (uint8_t)native;

        /*
         * Get volume information for the volumes in the
         * config.
         */
        numelements = ddi_get8(accessp, &raidconfig_page0->NumElements);
        vol = 0;
        disk = 0;
        element = (pMpi2RaidConfig0ConfigElement_t)
            &raidconfig_page0->ConfigElement;

        for (i = 0; ((i < numelements) && native); i++, element++) {
                /*
                 * Get the element type.  Could be Volume,
                 * PhysDisk, Hot Spare, or Online Capacity
                 * Expansion PhysDisk.
                 */
                elementtype = ddi_get16(accessp, &element->ElementFlags);
                elementtype &= MPI2_RAIDCONFIG0_EFLAGS_MASK_ELEMENT_TYPE;

                /*
                 * For volumes, get the RAID settings and the
                 * WWID.
                 */
                if (elementtype == etype_vol) {
                        voldevhandle = ddi_get16(accessp,
                            &element->VolDevHandle);
                        raidconfig->m_raidvol[vol].m_israid = 1;
                        raidconfig->m_raidvol[vol].
```



```
                            m_raidhandle = voldevhandle;
                        /*
                         * Get the settings for the raid
                         * volume.  This includes the
                         * DevHandles for the disks making up
                         * the raid volume.
                         */
                        if (mptsas_get_raid_settings(mpt,
                            &raidconfig->m_raidvol[vol]))
                                continue;

                        /*
                         * Get the WWID of the RAID volume for
                         * SAS HBA
                         */
                        if (mptsas_get_raid_wwid(mpt,
                            &raidconfig->m_raidvol[vol]))
                                continue;

                        raidwwn = raidconfig->m_raidvol[vol].
                            m_raidwwid;

                        /*
                         * RAID uses phymask of 0.
                         */
                        ptgt = mptsas_tgt_alloc(mpt,
                            voldevhandle, raidwwn, 0, 0, 0);

                        raidconfig->m_raidvol[vol].m_raidtgt =
                            ptgt;

                        /*
                         * Increment volume index within this
                         * raid config.
                         */
                        vol++;
                } else if ((elementtype == etype_pd) ||
                    (elementtype == etype_hs) ||
                    (elementtype == etype_oce)) {
                        /*
                         * For all other element types, put
                         * their DevHandles in the phys disk
                         * list of the config.  These are all
                         * some variation of a Phys Disk and
                         * this list is used to keep these
                         * disks from going online.
                         */
                        raidconfig->m_physdisk_devhdl[disk] = ddi_get16(accessp,
                            &element->PhysDiskDevHandle);

                        /*
                         * Increment disk index within this
                         * raid config.
                         */
                        disk++;
                }
        }

        return (rval);
}

int
mptsas_get_raid_info(mptsas_t *mpt)
{
        int rval = DDI_SUCCESS;
        uint32_t confignum, pageaddress;
```

```
        uint8_t configindex;

        ASSERT(mutex_owned(&mpt->m_mutex));

        /*
         * Clear all RAID info before starting.
         */
        bzero(mpt->m_raidconfig, sizeof (mpt->m_raidconfig));
        mpt->m_num_raid_configs = 0;

        configindex = 0;
        confignum = 0xff;
        pageaddress = MPI2_RAID_PGAD_FORM_GET_NEXT_CONFIGNUM | confignum;
        while (rval == DDI_SUCCESS) {
                /*
                 * Get the header and config page.  reply contains the reply
                 * frame, which holds status info for the request.
                 */
                rval = mptsas_access_config_page(mpt,
                    MPI2_CONFIG_ACTION_PAGE_READ_CURRENT,
                    MPI2_CONFIG_EXTPAGETYPE_RAID_CONFIG, 0, pageaddress,
                    mptsas_raidconf_page_0_cb, &confignum, configindex);
                configindex++;
                pageaddress = MPI2_RAID_PGAD_FORM_GET_NEXT_CONFIGNUM |
                    confignum;
        }

        return (rval);
}

static int
mptsas_raidvol_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
        pMpi2RaidVolPage0_t raidpage;
        int rval = DDI_SUCCESS, i;
        mptsas_raidvol_t *raidvol;
        uint8_t numdisks, volstate, voltype, physdisknum;
        uint32_t volsetting;
        uint32_t statusflags, resync_flag;

        if (iocstatus == MPI2_IOCSTATUS_CONFIG_INVALID_PAGE)
                return (DDI_FAILURE);

        if (iocstatus != MPI2_IOCSTATUS_SUCCESS) {
                mptsas_log(mpt, CE_WARN, "mptsas_raidvol_page0_cb "
                    "config: IOCStatus=0x%x, IOCLogInfo=0x%x",
                    iocstatus, iocloginfo);
                rval = DDI_FAILURE;
                return (rval);
        }

        raidvol = va_arg(ap,  mptsas_raidvol_t *);

        raidpage = (pMpi2RaidVolPage0_t)page_memp;
        volstate = ddi_get8(accessp, &raidpage->VolumeState);
        volsetting = ddi_get32(accessp,
            (uint32_t *)(void *)&raidpage->VolumeSettings);
        statusflags = ddi_get32(accessp, &raidpage->VolumeStatusFlags);
        voltype = ddi_get8(accessp, &raidpage->VolumeType);

        raidvol->m_state = volstate;
```



```
        raidvol->m_statusflags = statusflags;
        /*
         * Volume size is not used right now. Set to 0.
         */
        raidvol->m_raidsize = 0;
        raidvol->m_settings = volsetting;
        raidvol->m_raidlevel = voltype;

        if (statusflags & MPI2_RAIDVOL0_STATUS_FLAG_QUIESCED) {
                mptsas_log(mpt, CE_NOTE, "?Volume %d is quiesced\n",
                    raidvol->m_raidhandle);
        }

        if (statusflags &
            MPI2_RAIDVOL0_STATUS_FLAG_RESYNC_IN_PROGRESS) {
                mptsas_log(mpt, CE_NOTE, "?Volume %d is resyncing\n",
                    raidvol->m_raidhandle);
        }

        resync_flag = MPI2_RAIDVOL0_STATUS_FLAG_RESYNC_IN_PROGRESS;
        switch (volstate) {
        case MPI2_RAID_VOL_STATE_OPTIMAL:
                mptsas_log(mpt, CE_NOTE, "?Volume %d is "
                    "optimal\n", raidvol->m_raidhandle);
                break;
        case MPI2_RAID_VOL_STATE_DEGRADED:
                if ((statusflags & resync_flag) == 0) {
                        mptsas_log(mpt, CE_WARN, "Volume %d "
                            "is degraded\n",
                            raidvol->m_raidhandle);
                }
                break;
        case MPI2_RAID_VOL_STATE_FAILED:
                mptsas_log(mpt, CE_WARN, "Volume %d is "
                    "failed\n", raidvol->m_raidhandle);
                break;
        case MPI2_RAID_VOL_STATE_MISSING:
                mptsas_log(mpt, CE_WARN, "Volume %d is "
                    "missing\n", raidvol->m_raidhandle);
                break;
        default:
                break;
        }
        numdisks = raidpage->NumPhysDisks;
        raidvol->m_ndisks = numdisks;
        for (i = 0; i < numdisks; i++) {
                physdisknum = raidpage->PhysDisk[i].PhysDiskNum;
                raidvol->m_disknum[i] = physdisknum;
                if (mptsas_get_physdisk_settings(mpt, raidvol,
                    physdisknum))
                        break;
        }
        return (rval);
}

int
mptsas_get_raid_settings(mptsas_t *mpt, mptsas_raidvol_t *raidvol)
{
        int rval = DDI_SUCCESS;
        uint32_t page_address;

        ASSERT(mutex_owned(&mpt->m_mutex));

        /*
         * Get the header and config page.  reply contains the reply frame,
         * which holds status info for the request.
```

```
 */
        page_address = (MPI2_RAID_VOLUME_PGAD_FORM_MASK &
            MPI2_RAID_VOLUME_PGAD_FORM_HANDLE) | raidvol->m_raidhandle;
        rval = mptsas_access_config_page(mpt,
            MPI2_CONFIG_ACTION_PAGE_READ_CURRENT,
            MPI2_CONFIG_PAGETYPE_RAID_VOLUME, 0, page_address,
            mptsas_raidvol_page_0_cb, raidvol);

        return (rval);
}

static int
mptsas_raidvol_page_1_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
        pMpi2RaidVolPage1_t     raidpage;
        int                     rval = DDI_SUCCESS, i;
        uint8_t                 *sas_addr = NULL;
        uint8_t                 tmp_sas_wwn[SAS_WWN_BYTE_SIZE];
        uint64_t                *sas_wwn;

        if (iocstatus != MPI2_IOCSTATUS_SUCCESS) {
                mptsas_log(mpt, CE_WARN, "mptsas_raidvol_page_1_cb "
                    "config: IOCStatus=0x%x, IOCLogInfo=0x%x",
                    iocstatus, iocloginfo);
                rval = DDI_FAILURE;
                return (rval);
        }
        sas_wwn = va_arg(ap, uint64_t *);

        raidpage = (pMpi2RaidVolPage1_t)page_memp;
        sas_addr = (uint8_t *)(&raidpage->WWID);
        for (i = 0; i < SAS_WWN_BYTE_SIZE; i++) {
                tmp_sas_wwn[i] = ddi_get8(accessp, sas_addr + i);
        }
        bcopy(tmp_sas_wwn, sas_wwn, SAS_WWN_BYTE_SIZE);
        *sas_wwn = LE_64(*sas_wwn);
        return (rval);
}

static int
mptsas_get_raid_wwid(mptsas_t *mpt, mptsas_raidvol_t *raidvol)
{
        int rval = DDI_SUCCESS;
        uint32_t page_address;
        uint64_t sas_wwn;

        ASSERT(mutex_owned(&mpt->m_mutex));

        /*
         * Get the header and config page.  reply contains the reply frame,
         * which holds status info for the request.
         */
        page_address = (MPI2_RAID_VOLUME_PGAD_FORM_MASK &
            MPI2_RAID_VOLUME_PGAD_FORM_HANDLE) | raidvol->m_raidhandle;
        rval = mptsas_access_config_page(mpt,
            MPI2_CONFIG_ACTION_PAGE_READ_CURRENT,
            MPI2_CONFIG_PAGETYPE_RAID_VOLUME, 1, page_address,
            mptsas_raidvol_page_1_cb, &sas_wwn);

        /*
         * Get the required information from the page.
```



```
         */
        if (rval == DDI_SUCCESS) {

                /*
                 * replace top nibble of WWID of RAID to '3' for OBP
                 */
                sas_wwn = MPTSAS_RAID_WWID(sas_wwn);
                raidvol->m_raidwwid = sas_wwn;
        }

done:
        return (rval);
}

static int
mptsas_raidphydsk_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
        pMpi2RaidPhysDiskPage0_t        diskpage;
        int                     rval = DDI_SUCCESS;
        uint16_t                *devhdl;
        uint8_t                 *state;

        if (iocstatus == MPI2_IOCSTATUS_CONFIG_INVALID_PAGE)
                return (DDI_FAILURE);

        if (iocstatus != MPI2_IOCSTATUS_SUCCESS) {
                mptsas_log(mpt, CE_WARN, "mptsas_raidphydsk_page0_cb "
                    "config: IOCStatus=0x%x, IOCLogInfo=0x%x",
                    iocstatus, iocloginfo);
                rval = DDI_FAILURE;
                return (rval);
        }
        devhdl = va_arg(ap, uint16_t *);
        state = va_arg(ap, uint8_t *);
        diskpage = (pMpi2RaidPhysDiskPage0_t)page_memp;
        *devhdl = ddi_get16(accessp, &diskpage->DevHandle);
        *state = ddi_get8(accessp, &diskpage->PhysDiskState);
        return (rval);
}

int
mptsas_get_physdisk_settings(mptsas_t *mpt, mptsas_raidvol_t *raidvol,
    uint8_t physdisknum)
{
        int                     rval = DDI_SUCCESS, i;
        uint8_t                 state;
        uint16_t                devhdl;
        uint32_t                page_address;

        ASSERT(mutex_owned(&mpt->m_mutex));

        /*
         * Get the header and config page.  reply contains the reply frame,
         * which holds status info for the request.
         */
        page_address = (MPI2_PHYSDISK_PGAD_FORM_MASK &
            MPI2_PHYSDISK_PGAD_FORM_PHYSDISKNUM) | physdisknum;
        rval = mptsas_access_config_page(mpt,
            MPI2_CONFIG_ACTION_PAGE_READ_CURRENT,
            MPI2_CONFIG_PAGETYPE_RAID_PHYSDISK, 0, page_address,
            mptsas_raidphydsk_page_0_cb, &devhdl, &state);
```

```
        /*
         * Get the required information from the page.
         */
        if (rval == DDI_SUCCESS) {
                for (i = 0; i < MPTSAS_MAX_DISKS_IN_VOL; i++) {
                        /* find the correct position in the arrays */
                        if (raidvol->m_disknum[i] == physdisknum)
                                break;
                }
                raidvol->m_devhdl[i] = devhdl;

                switch (state) {
                        case MPI2_RAID_PD_STATE_OFFLINE:
                                raidvol->m_diskstatus[i] =
                                    RAID_DISKSTATUS_FAILED;
                                break;

                        case MPI2_RAID_PD_STATE_HOT_SPARE:
                        case MPI2_RAID_PD_STATE_NOT_CONFIGURED:
                        case MPI2_RAID_PD_STATE_NOT_COMPATIBLE:
                                break;

                        case MPI2_RAID_PD_STATE_DEGRADED:
                        case MPI2_RAID_PD_STATE_OPTIMAL:
                        case MPI2_RAID_PD_STATE_REBUILDING:
                        case MPI2_RAID_PD_STATE_ONLINE:
                        default:
                                raidvol->m_diskstatus[i] =
                                    RAID_DISKSTATUS_GOOD;
                                break;
                }
        }

        return (rval);
}

/*
 * RAID Action for System Shutdown. This request uses the dedicated TM slot to
 * avoid a call to mptsas_save_cmd.  Since Solaris requires that the mutex is
 * not held during the mptsas_quiesce function, this RAID action must not use
 * the normal code path of requests and replies.
 */
void
mptsas_raid_action_system_shutdown(mptsas_t *mpt)
{
        mptsas_reply_pqueue_t           *rpqp;
        pMpi2RaidActionRequest_t        action;
        uint8_t                         ir_active = FALSE, reply_type;
        uint8_t                         function, found_reply = FALSE;
        uint16_t                        SMID, action_type;
        mptsas_slots_t                  *slots = mpt->m_active;
        int                             config, vol;
        mptsas_cmd_t                    *cmd;
        uint32_t                        reply_addr;
        uint64_t                        request_desc;
        int                             cnt;
        pMpi2ReplyDescriptorsUnion_t    reply_desc_union;
        pMPI2DefaultReply_t             reply;
        pMpi2AddressReplyDescriptor_t   address_reply;

        /*
         * Before doing the system shutdown RAID Action, make sure that the IOC
         * supports IR and make sure there is a valid volume for the request.
         */
        if (mpt->m_ir_capable) {
```



```
                for (config = 0; (config < mpt->m_num_raid_configs) &&
                    (!ir_active); config++) {
                        for (vol = 0; vol < MPTSAS_MAX_RAIDVOLS; vol++) {
                                if (mpt->m_raidconfig[config].m_raidvol[vol].
                                    m_israid) {
                                        ir_active = TRUE;
                                        break;
                                }
                        }
                }
        }
        if (!ir_active) {
                return;
        }

        /*
         * If TM slot is already being used (highly unlikely), show message and
         * don't issue the RAID action.
         */
        if (slots->m_slot[MPTSAS_TM_SLOT(mpt)] != NULL) {
                mptsas_log(mpt, CE_WARN, "RAID Action slot in use.  Cancelling"
                    " System Shutdown RAID Action.\n");
                return;
        }

        /*
         * Create the cmd and put it in the dedicated TM slot.
         */
        cmd = &(mpt->m_event_task_mgmt.m_event_cmd);
        bzero((caddr_t)cmd, sizeof (*cmd));
        cmd->cmd_pkt = NULL;
        cmd->cmd_slot = MPTSAS_TM_SLOT(mpt);
        slots->m_slot[MPTSAS_TM_SLOT(mpt)] = cmd;

        /*
         * Form message for raid action.
         */
        action = (pMpi2RaidActionRequest_t)(mpt->m_req_frame +
            (mpt->m_req_frame_size * cmd->cmd_slot));
        bzero(action, mpt->m_req_frame_size);
        action->Function = MPI2_FUNCTION_RAID_ACTION;
        action->Action = MPI2_RAID_ACTION_SYSTEM_SHUTDOWN_INITIATED;

        /*
         * Send RAID Action.
         * Defaults to MSIxIndex of 0, so check reply q 0 below.
         */
        (void) ddi_dma_sync(mpt->m_dma_req_frame_hdl, 0, 0,
            DDI_DMA_SYNC_FORDEV);
        request_desc = (cmd->cmd_slot << 16) +
            MPI2_REQ_DESCRIPT_FLAGS_DEFAULT_TYPE;
        MPTSAS_START_CMD(mpt, request_desc);

        /*
         * Even though reply does not matter because the system is shutting
         * down, wait no more than 5 seconds here to get the reply just because
         * we don't want to leave it hanging if it's coming.  Poll because
         * interrupts are disabled when this function is called.
         */
        rpqp = mpt->m_rep_post_queues;
        for (cnt = 0; cnt < 5000; cnt++) {
                /*
                 * Check for a reply.
                 */
                (void) ddi_dma_sync(mpt->m_dma_post_queue_hdl, 0, 0,
                    DDI_DMA_SYNC_FORCPU);
```

```
                reply_desc_union = (pMpi2ReplyDescriptorsUnion_t)
                    MPTSAS_GET_NEXT_REPLY(rpqp, rpqp->rpq_index);

                if (ddi_get32(mpt->m_acc_post_queue_hdl,
                    &reply_desc_union->Words.Low) == 0xFFFFFFFF ||
                    ddi_get32(mpt->m_acc_post_queue_hdl,
                    &reply_desc_union->Words.High) == 0xFFFFFFFF) {
                        drv_usecwait(1000);
                        continue;
                }

                /*
                 * There is a reply.  If it’s not an address reply, ignore it.
                 */
                reply_type = ddi_get8(mpt->m_acc_post_queue_hdl,
                    &reply_desc_union->Default.ReplyFlags);
                reply_type &= MPI2_RPY_DESCRIPT_FLAGS_TYPE_MASK;
                if (reply_type != MPI2_RPY_DESCRIPT_FLAGS_ADDRESS_REPLY) {
                        goto clear_and_continue;
                }

                /*
                 * SMID must be the TM slot since that’s what we’re using for
                 * this RAID action.  If not, ignore this reply.
                 */
                address_reply =
                    (pMpi2AddressReplyDescriptor_t)reply_desc_union;
                SMID = ddi_get16(mpt->m_acc_post_queue_hdl,
                    &address_reply->SMID);
                if (SMID != MPTSAS_TM_SLOT(mpt)) {
                        goto clear_and_continue;
                }

                /*
                 * If reply frame is not in the proper range ignore it.
                 */
                reply_addr = ddi_get32(mpt->m_acc_post_queue_hdl,
                    &address_reply->ReplyFrameAddress);
                if ((reply_addr < mpt->m_reply_frame_dma_addr) ||
                    (reply_addr >= (mpt->m_reply_frame_dma_addr +
                    (mpt->m_reply_frame_size * mpt->m_free_queue_depth))) ||
                    ((reply_addr - mpt->m_reply_frame_dma_addr) %
                    mpt->m_reply_frame_size != 0)) {
                        goto clear_and_continue;
                }

                /*
                 * If not a RAID action reply ignore it.
                 */
                (void) ddi_dma_sync(mpt->m_dma_reply_frame_hdl, 0, 0,
                    DDI_DMA_SYNC_FORCPU);
                reply = (pMPI2DefaultReply_t)(mpt->m_reply_frame +
                    (reply_addr - mpt->m_reply_frame_dma_addr));
                function = ddi_get8(mpt->m_acc_reply_frame_hdl,
                    &reply->Function);
                if (function != MPI2_FUNCTION_RAID_ACTION) {
                        goto clear_and_continue;
                }

                /*
                 * Finally, make sure this is the System Shutdown RAID action.
                 * If not, ignore reply.
                 */
                action_type = ddi_get16(mpt->m_acc_reply_frame_hdl,
                    &reply->FunctionDependent1);
```



```
                if (action_type !=
                    MPI2_RAID_ACTION_SYSTEM_SHUTDOWN_INITIATED) {
                        goto clear_and_continue;
                }
                found_reply = TRUE;

clear_and_continue:
                /*
                 * Clear the reply descriptor for re-use and increment index.
                 */
                ddi_put64(mpt->m_acc_post_queue_hdl,
                    &((uint64_t *)(void *)rpqp->rpq_queue)[rpqp->rpq_index],
                    0xFFFFFFFFFFFFFFFF);
                (void) ddi_dma_sync(mpt->m_dma_post_queue_hdl, 0, 0,
                    DDI_DMA_SYNC_FORDEV);

                /*
                 * Update the reply index and keep looking for the
                 * reply if not found yet.
                 */
                if (++rpqp->rpq_index == mpt->m_post_queue_depth) {
                        rpqp->rpq_index = 0;
                }

                ddi_put32(mpt->m_datap, &mpt->m_reg->ReplyPostHostIndex,
                    rpqp->rpq_index);
                if (!found_reply) {
                        continue;
                }

                break;
        }

        /*
         * clear the used slot as the last step.
         */
        slots->m_slot[MPTSAS_TM_SLOT(mpt)] = NULL;
}

int
mptsas_delete_volume(mptsas_t *mpt, uint16_t volid)
{
        int             config, i = 0, vol = (-1);

        for (config = 0; (config < mpt->m_num_raid_configs) && (vol != i);
            config++) {
                for (i = 0; i < MPTSAS_MAX_RAIDVOLS; i++) {
                        if (mpt->m_raidconfig[config].m_raidvol[i].
                            m_raidhandle == volid) {
                                vol = i;
                                break;
                        }
                }
        }

        if (vol < 0) {
                mptsas_log(mpt, CE_WARN, "raid doesn’t exist at specified "
                    "target.");
                return (-1);
        }

        mpt->m_raidconfig[config].m_raidvol[vol].m_israid = 0;
        mpt->m_raidconfig[config].m_raidvol[vol].m_ndisks = 0;
        for (i = 0; i < MPTSAS_MAX_DISKS_IN_VOL; i++) {
                mpt->m_raidconfig[config].m_raidvol[vol].m_disknum[i] = 0;
                mpt->m_raidconfig[config].m_raidvol[vol].m_devhdl[i] = 0;
```

```
        }

        return (0);
}
#endif /* ! codereview */
```

```
**********************************************************
   14649 Thu Jun 12 17:28:23 2014
new/usr/src/uts/common/io/scsi/adapters/mpt_sas3/mptsas3_smhba.c
4546 mpt_sas needs enhancing to support LSI MPI2.5
**********************************************************
/*
 * 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.
 */
/*
 * This file contains SM-HBA support for MPT SAS3 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>
#pragma pack()

/*
 * private header files.
 */
#include <sys/scsi/adapters/mpt_sas3/mptsas3_var.h>
#include <sys/scsi/adapters/mpt_sas3/mptsas3_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);
        }
}

void
mptsas_smhba_show_phy_info(mptsas_t *mpt)
{
        int i;

        ASSERT(mpt != NULL);

        for (i = 0; i < MPTSAS_MAX_PHYS; i++) {
                mptsas_log(mpt, CE_WARN,
                    "phy %d, Owner hdl:0x%x, attached hdl: 0x%x,"
                    "attached phy identifier %d,Program link rate 0x%x,"
                    "hw link rate 0x%x, negotiator link rate 0x%x, path %s",
                    i, mpt->m_phy_info[i].smhba_info.owner_devhdl,
                    mpt->m_phy_info[i].smhba_info.attached_devhdl,
                    mpt->m_phy_info[i].smhba_info.attached_phy_identify,
                    mpt->m_phy_info[i].smhba_info.programmed_link_rate,
                    mpt->m_phy_info[i].smhba_info.hw_link_rate,
                    mpt->m_phy_info[i].smhba_info.negotiated_link_rate,
                    mpt->m_phy_info[i].smhba_info.path);
        }
}

static void
mptsas_smhba_create_phy_props(nvlist_t **phy_props, smhba_info_t *pSmhba,
```

```
    uint8_t phy_id, uint16_t *attached_devhdl)
{
        (void) nvlist_alloc(phy_props, NV_UNIQUE_NAME, KM_SLEEP);
        (void) nvlist_add_uint8(*phy_props, SAS_PHY_ID, phy_id);
        (void) nvlist_add_uint8(*phy_props, "phyState",
            (pSmhba->negotiated_link_rate & 0x0f));
        (void) nvlist_add_int8(*phy_props, SAS_NEG_LINK_RATE,
            (pSmhba->negotiated_link_rate & 0x0f));
        (void) nvlist_add_int8(*phy_props, SAS_PROG_MIN_LINK_RATE,
            (pSmhba->programmed_link_rate & 0x0f));
        (void) nvlist_add_int8(*phy_props, SAS_HW_MIN_LINK_RATE,
            (pSmhba->hw_link_rate & 0x0f));
        (void) nvlist_add_int8(*phy_props, SAS_PROG_MAX_LINK_RATE,
            ((pSmhba->programmed_link_rate & 0xf0) >> 4));
        (void) nvlist_add_int8(*phy_props, SAS_HW_MAX_LINK_RATE,
            ((pSmhba->hw_link_rate & 0xf0) >> 4));

        if (pSmhba->attached_devhdl && (attached_devhdl != NULL))
                *attached_devhdl = pSmhba->attached_devhdl;
}

static void
mptsas_smhba_update_phy_props(mptsas_t *mpt, dev_info_t *dip,
    nvlist_t **phy_props, uint8_t phy_nums)
{
        int             rval;
        size_t          packed_size;
        char            *packed_data = NULL;
        nvlist_t        *nvl;

        if (nvlist_alloc(&nvl, NV_UNIQUE_NAME, KM_SLEEP) != 0) {
                mptsas_log(mpt, CE_WARN, "%s: nvlist_alloc() failed", __func__);
                return;
        }

        rval = nvlist_add_nvlist_array(nvl, SAS_PHY_INFO_NVL, phy_props,
            phy_nums);
        if (rval) {
                mptsas_log(mpt, CE_WARN,
                    " nv list array add failed, return value %d.", rval);
                goto exit;
        }
        (void) nvlist_size(nvl, &packed_size, NV_ENCODE_NATIVE);
        packed_data = kmem_zalloc(packed_size, KM_SLEEP);
        (void) nvlist_pack(nvl, &packed_data, &packed_size,
            NV_ENCODE_NATIVE, 0);

        (void) ddi_prop_update_byte_array(DDI_DEV_T_NONE, dip,
            SAS_PHY_INFO, (uchar_t *)packed_data, packed_size);

exit:
        nvlist_free(nvl);

        if (packed_data != NULL) {
                kmem_free(packed_data, packed_size);
        }
}

void
mptsas_smhba_set_one_phy_props(mptsas_t *mpt, dev_info_t *dip, uint8_t phy_id,
    uint16_t *attached_devhdl)
{
        nvlist_t        *phy_props;

        ASSERT(phy_id < mpt->m_num_phys);
```



```
        mptsas_smhba_create_phy_props(&phy_props,
            &mpt->m_phy_info[phy_id].smhba_info, phy_id, attached_devhdl);

        mptsas_smhba_update_phy_props(mpt, dip, &phy_props, 1);

        nvlist_free(phy_props);
}

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)
{
        int             i, j;
        nvlist_t        **phy_props;

        if (phy_nums == 0)
                return;

        phy_props = kmem_zalloc(sizeof (nvlist_t *) * phy_nums, KM_SLEEP);

        for (i = 0, j = 0; i < mpt->m_num_phys && j < phy_nums; i++)
                if (phy_mask == mpt->m_phy_info[i].phy_mask)
                        mptsas_smhba_create_phy_props(&phy_props[j++],
                            &mpt->m_phy_info[i].smhba_info, i, attached_devhdl);

        mptsas_smhba_update_phy_props(mpt, dip, phy_props, j);

        for (i = 0; i < j && phy_props[i] != NULL; i++)
                nvlist_free(phy_props[i]);

        kmem_free(phy_props, sizeof (nvlist_t *) * phy_nums);
}

/*
 * Called with PHY lock held on phyp
 */
void
mptsas_smhba_log_sysevent(mptsas_t *mpt, char *subclass, char *etype,
    smhba_info_t *phyp)
{
        nvlist_t        *attr_list;
        char            *pname;
        char            sas_addr[MPTSAS_WWN_STRLEN];
        uint8_t         phynum = 0;
        uint8_t         lrate = 0;

        if (mpt->m_dip == NULL)
                return;
        if (phyp == NULL)
                return;

        pname = kmem_zalloc(MAXPATHLEN, KM_NOSLEEP);
        if (pname == NULL)
                return;

        if ((strcmp(subclass, ESC_SAS_PHY_EVENT) == 0) ||
            (strcmp(subclass, ESC_SAS_HBA_PORT_BROADCAST) == 0)) {
                ASSERT(phyp != NULL);
                (void) strncpy(pname, phyp->path, strlen(phyp->path));
                phynum = phyp->phy_id;
                bzero(sas_addr, sizeof (sas_addr));
                (void) sprintf(sas_addr, "w%016"PRIx64, phyp->sas_addr);
                if (strcmp(etype, SAS_PHY_ONLINE) == 0) {
                        lrate = phyp->negotiated_link_rate;
                }
        }
```

```
        if (strcmp(subclass, ESC_SAS_HBA_PORT_BROADCAST) == 0) {
                (void) ddi_pathname(mpt->m_dip, pname);
        }

        if (nvlist_alloc(&attr_list, NV_UNIQUE_NAME_TYPE, 0) != 0) {
                mptsas_log(mpt, CE_WARN,
                    "%s: Failed to post sysevent", __func__);
                kmem_free(pname, MAXPATHLEN);
                return;
        }

        if (nvlist_add_int32(attr_list, SAS_DRV_INST,
            ddi_get_instance(mpt->m_dip)) != 0)
                goto fail;

        if (nvlist_add_string(attr_list, SAS_PORT_ADDR, sas_addr) != 0)
                goto fail;

        if (nvlist_add_string(attr_list, SAS_DEVFS_PATH, pname) != 0)
                goto fail;

        if (nvlist_add_uint8(attr_list, SAS_PHY_ID, phynum) != 0)
                goto fail;

        if (strcmp(etype, SAS_PHY_ONLINE) == 0) {
                if (nvlist_add_uint8(attr_list, SAS_LINK_RATE, lrate) != 0)
                        goto fail;
        }

        if (nvlist_add_string(attr_list, SAS_EVENT_TYPE, etype) != 0)
                goto fail;

        (void) ddi_log_sysevent(mpt->m_dip, DDI_VENDOR_SUNW, EC_HBA, subclass,
            attr_list, NULL, DDI_NOSLEEP);

fail:
        kmem_free(pname, MAXPATHLEN);
        nvlist_free(attr_list);
}

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];
        char                    phymask[MPTSAS_MAX_PHYS];
        int                     i;

        ASSERT(iport != NULL);
        ASSERT(mpt != NULL);

        for (i = 0; i < mpt->m_num_phys; i++) {

                bzero(phymask, sizeof (phymask));
                (void) sprintf(phymask, "%x", mpt->m_phy_info[i].phy_mask);
                if (strcmp(phymask, iport) == 0) {

                        phyp = &mpt->m_phy_info[i].smhba_info;
                        mutex_enter(&phyp->phy_mutex);

                        if (phyp->phy_stats != NULL) {
                                mutex_exit(&phyp->phy_mutex);
                                /* We’ve already created this kstat instance */
                                continue;
```



```
                        }

                        ndata = (sizeof (sas_phy_stats_t)/
                            sizeof (kstat_named_t));
                        (void) snprintf(ks_name, sizeof (ks_name),
                            "%s.%llx.%d.%d", ddi_driver_name(dip),
                            (longlong_t)mpt->un.m_base_wwid,
                            ddi_get_instance(dip), i);

                        phyp->phy_stats = kstat_create("mptsas3",
                            ddi_get_instance(dip), ks_name, KSTAT_SAS_PHY_CLASS,
                            KSTAT_TYPE_NAMED, ndata, 0);

                        if (phyp->phy_stats == NULL) {
                                mutex_exit(&phyp->phy_mutex);
                                mptsas_log(mpt, CE_WARN,
                                    "%s: Failed to create %s kstats", __func__,
                                    ks_name);
                                continue;
                        }

                        ps = (sas_phy_stats_t *)phyp->phy_stats->ks_data;

                        kstat_named_init(&ps->seconds_since_last_reset,
                            "SecondsSinceLastReset", KSTAT_DATA_ULONGLONG);
                        kstat_named_init(&ps->tx_frames,
                            "TxFrames", KSTAT_DATA_ULONGLONG);
                        kstat_named_init(&ps->rx_frames,
                            "RxFrames", KSTAT_DATA_ULONGLONG);
                        kstat_named_init(&ps->tx_words,
                            "TxWords", KSTAT_DATA_ULONGLONG);
                        kstat_named_init(&ps->rx_words,
                            "RxWords", KSTAT_DATA_ULONGLONG);
                        kstat_named_init(&ps->invalid_dword_count,
                            "InvalidDwordCount", KSTAT_DATA_ULONGLONG);
                        kstat_named_init(&ps->running_disparity_error_count,
                            "RunningDisparityErrorCount", KSTAT_DATA_ULONGLONG);
                        kstat_named_init(&ps->loss_of_dword_sync_count,
                            "LossofDwordSyncCount", KSTAT_DATA_ULONGLONG);
                        kstat_named_init(&ps->phy_reset_problem_count,
                            "PhyResetProblemCount", KSTAT_DATA_ULONGLONG);

                        phyp->phy_stats->ks_private = phyp;
                        phyp->phy_stats->ks_update = mptsas_update_phy_stats;
                        kstat_install(phyp->phy_stats);
                        mutex_exit(&phyp->phy_mutex);
                }
        }
}

int
mptsas_update_phy_stats(kstat_t *ks, int rw)
{
        int                     ret = DDI_FAILURE;
        smhba_info_t            *pptr = NULL;
        sas_phy_stats_t         *ps = ks->ks_data;
        uint32_t                page_address;
        mptsas_t                *mpt;

        _NOTE(ARGUNUSED(rw));

        pptr = (smhba_info_t *)ks->ks_private;
        ASSERT((pptr != NULL));
        mpt = (mptsas_t *)pptr->mpt;
        ASSERT((mpt != NULL));
        page_address = (MPI2_SAS_PHY_PGAD_FORM_PHY_NUMBER | pptr->phy_id);
```

```
        /*
         * We just want to lock against other invocations of kstat;
         * we don't need to pmcs_lock_phy() for this.
         */
        mutex_enter(&mpt->m_mutex);

        ret = mptsas_get_sas_phy_page1(pptr->mpt, page_address, pptr);

        if (ret == DDI_FAILURE)
                goto fail;

        ps->invalid_dword_count.value.ull =
            (unsigned long long)pptr->invalid_dword_count;

        ps->running_disparity_error_count.value.ull =
            (unsigned long long)pptr->running_disparity_error_count;

        ps->loss_of_dword_sync_count.value.ull =
            (unsigned long long)pptr->loss_of_dword_sync_count;

        ps->phy_reset_problem_count.value.ull =
            (unsigned long long)pptr->phy_reset_problem_count;

        ret = DDI_SUCCESS;
fail:
        mutex_exit(&mpt->m_mutex);

        return (ret);
}

void
mptsas_destroy_phy_stats(mptsas_t *mpt)
{
        smhba_info_t    *phyp;
        int                     i = 0;

        ASSERT(mpt != NULL);

        for (i = 0; i < mpt->m_num_phys; i++) {
                phyp = &mpt->m_phy_info[i].smhba_info;
                if (phyp == NULL) {
                        continue;
                }

                mutex_enter(&phyp->phy_mutex);
                if (phyp->phy_stats != NULL) {
                        kstat_delete(phyp->phy_stats);
                        phyp->phy_stats = NULL;
                }
                mutex_exit(&phyp->phy_mutex);
        }
}

int
mptsas_smhba_phy_init(mptsas_t *mpt)
{
        int             i = 0;
        int             rval = DDI_SUCCESS;
        uint32_t        page_address;

        ASSERT(mutex_owned(&mpt->m_mutex));

        for (i = 0; i < mpt->m_num_phys; i++) {
                page_address =
                    (MPI2_SAS_PHY_PGAD_FORM_PHY_NUMBER |
```



```
                        (MPI2_SAS_PHY_PGAD_PHY_NUMBER_MASK & i));
                rval = mptsas_get_sas_phy_page0(mpt,
                    page_address, &mpt->m_phy_info[i].smhba_info);
                if (rval != DDI_SUCCESS) {
                        mptsas_log(mpt, CE_WARN,
                            "Failed to get sas phy page 0"
                            " for each phy");
                        return (DDI_FAILURE);
                }
                mpt->m_phy_info[i].smhba_info.phy_id = (uint8_t)i;
                mpt->m_phy_info[i].smhba_info.sas_addr =
                    mpt->un.m_base_wwid + i;
                mpt->m_phy_info[i].smhba_info.mpt = mpt;
        }
        return (DDI_SUCCESS);
}

int
mptsas_smhba_setup(mptsas_t *mpt)
{
        int             sm_hba = 1;
        char            chiprev, hw_rev[24];
        char            serial_number[72];
        int             protocol = 0;

        mutex_enter(&mpt->m_mutex);
        if (mptsas_smhba_phy_init(mpt)) {
                mutex_exit(&mpt->m_mutex);
                return (DDI_FAILURE);
        }
        mutex_exit(&mpt->m_mutex);

        /* SM-HBA support */
        mptsas_smhba_add_hba_prop(mpt, DATA_TYPE_INT32, MPTSAS_SMHBA_SUPPORTED,
            &sm_hba);

        /* SM-HBA driver version */
        mptsas_smhba_add_hba_prop(mpt, DATA_TYPE_STRING, MPTSAS_DRV_VERSION,
            mptsas_driver_rev);

        /* SM-HBA hardware version */
        chiprev = 'A' + mpt->m_revid;
        (void) snprintf(hw_rev, 2, "%s", &chiprev);
        mptsas_smhba_add_hba_prop(mpt, DATA_TYPE_STRING, MPTSAS_HWARE_VERSION,
            hw_rev);

        /* SM-HBA phy number per HBA */
        mptsas_smhba_add_hba_prop(mpt, DATA_TYPE_INT32, MPTSAS_NUM_PHYS_HBA,
            &(mpt->m_num_phys));

        /* SM-HBA protocal support */
        protocol = SAS_SSP_SUPPORT | SAS_SATA_SUPPORT | SAS_SMP_SUPPORT;
        mptsas_smhba_add_hba_prop(mpt, DATA_TYPE_INT32,
            MPTSAS_SUPPORTED_PROTOCOL, &protocol);

        mptsas_smhba_add_hba_prop(mpt, DATA_TYPE_STRING, MPTSAS_MANUFACTURER,
            mpt->m_MANU_page0.ChipName);

        mptsas_smhba_add_hba_prop(mpt, DATA_TYPE_STRING, MPTSAS_MODEL_NAME,
            mpt->m_MANU_page0.BoardName);

        /*
         * VPD data is not available, we make a serial number for this.
         */

        (void) sprintf(serial_number, "%s%s%s%s%s",
```

```
            mpt->m_MANU_page0.ChipName,
            mpt->m_MANU_page0.ChipRevision,
            mpt->m_MANU_page0.BoardName,
            mpt->m_MANU_page0.BoardAssembly,
            mpt->m_MANU_page0.BoardTracerNumber);

        mptsas_smhba_add_hba_prop(mpt, DATA_TYPE_STRING, MPTSAS_SERIAL_NUMBER,
            &serial_number[0]);

        return (DDI_SUCCESS);
}
#endif /* ! codereview */
```

```
**********************************************************
   52532 Thu Jun 12 17:28:23 2014
new/usr/src/uts/common/sys/scsi/adapters/mpt_sas3/mpi/mpi2.h
4546 mpt_sas needs enhancing to support LSI MPI2.5
**********************************************************
/*
 * 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.
 *
 * 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.
 *
```



```
 *  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_MASK             (0xFF00)
#define MPI2_VERSION_MAJOR_SHIFT            (8)
#define MPI2_VERSION_MINOR_MASK             (0x00FF)
#define MPI2_VERSION_MINOR_SHIFT            (0)

/* major version for all MPI v2.x */
#define MPI2_VERSION_MAJOR                  (0x02)

/* minor version for MPI v2.0 compatible products */
#define MPI2_VERSION_MINOR                  (0x00)
```

```
#define MPI2_VERSION ((MPI2_VERSION_MAJOR << MPI2_VERSION_MAJOR_SHIFT) |   \
                                      MPI2_VERSION_MINOR)
#define MPI2_VERSION_02_00                  (0x0200)


/* minor version for MPI v2.5 compatible products */
#define MPI25_VERSION_MINOR                 (0x05)
#define MPI25_VERSION ((MPI2_VERSION_MAJOR << MPI2_VERSION_MAJOR_SHIFT) |   \
                                      MPI25_VERSION_MINOR)
#define MPI2_VERSION_02_05                  (0x0205)


/* Unit and Dev versioning for this MPI header set */
#define MPI2_HEADER_VERSION_UNIT            (0x12)
#define MPI2_HEADER_VERSION_DEV             (0x00)
#define MPI2_HEADER_VERSION_UNIT_MASK       (0xFF00)
#define MPI2_HEADER_VERSION_UNIT_SHIFT      (8)
#define MPI2_HEADER_VERSION_DEV_MASK        (0x00FF)
#define MPI2_HEADER_VERSION_DEV_SHIFT       (0)
#define MPI2_HEADER_VERSION ((MPI2_HEADER_VERSION_UNIT << 8) | MPI2_HEADER_VERSI


/*****************************************************************************
*
*        IOC State Definitions
*
*****************************************************************************/

#define MPI2_IOC_STATE_RESET               (0x00000000)
#define MPI2_IOC_STATE_READY               (0x10000000)
#define MPI2_IOC_STATE_OPERATIONAL         (0x20000000)
#define MPI2_IOC_STATE_FAULT               (0x40000000)

#define MPI2_IOC_STATE_MASK                (0xF0000000)
#define MPI2_IOC_STATE_SHIFT               (28)

/* Fault state range for prodcut specific codes */
#define MPI2_FAULT_PRODUCT_SPECIFIC_MIN                 (0x0000)
#define MPI2_FAULT_PRODUCT_SPECIFIC_MAX                 (0xEFFF)


/*****************************************************************************
*
*        System Interface Register Definitions
*
*****************************************************************************/

typedef volatile struct _MPI2_SYSTEM_INTERFACE_REGS
{
    U32         Doorbell;                   /* 0x00 */
    U32         WriteSequence;              /* 0x04 */
    U32         HostDiagnostic;             /* 0x08 */
    U32         Reserved1;                  /* 0x0C */
    U32         DiagRWData;                 /* 0x10 */
    U32         DiagRWAddressLow;           /* 0x14 */
    U32         DiagRWAddressHigh;          /* 0x18 */
    U32         Reserved2[5];               /* 0x1C */
    U32         HostInterruptStatus;        /* 0x30 */
    U32         HostInterruptMask;          /* 0x34 */
    U32         DCRData;                    /* 0x38 */
    U32         DCRAddress;                 /* 0x3C */
    U32         Reserved3[2];               /* 0x40 */
    U32         ReplyFreeHostIndex;         /* 0x48 */
    U32         Reserved4[8];               /* 0x4C */
    U32         ReplyPostHostIndex;         /* 0x6C */
    U32         Reserved5;                  /* 0x70 */
```



```
    U32         HCBSize;                    /* 0x74 */
    U32         HCBAddressLow;              /* 0x78 */
    U32         HCBAddressHigh;             /* 0x7C */
    U32         Reserved6[16];              /* 0x80 */
    U32         RequestDescriptorPostLow;   /* 0xC0 */
    U32         RequestDescriptorPostHigh;  /* 0xC4 */
    U32         Reserved7[14];              /* 0xC8 */
    U32         Reserved8[128];             /* 0x100 */
    U32         Reserved10[3];              /* 0x300 */
    U32         SuppReplyPostHostIndex[32]; /* 0x30C */
} MPI2_SYSTEM_INTERFACE_REGS, MPI2_POINTER PTR_MPI2_SYSTEM_INTERFACE_REGS,
  Mpi2SystemInterfaceRegs_t, MPI2_POINTER pMpi2SystemInterfaceRegs_t;

/*
 * Defines for working with the Doorbell register.
 */
#define MPI2_DOORBELL_OFFSET                    (0x00000000)

/* IOC --> System values */
#define MPI2_DOORBELL_USED                      (0x08000000)
#define MPI2_DOORBELL_WHO_INIT_MASK             (0x07000000)
#define MPI2_DOORBELL_WHO_INIT_SHIFT            (24)
#define MPI2_DOORBELL_FAULT_CODE_MASK           (0x0000FFFF)
#define MPI2_DOORBELL_DATA_MASK                 (0x0000FFFF)

/* System --> IOC values */
#define MPI2_DOORBELL_FUNCTION_MASK             (0xFF000000)
#define MPI2_DOORBELL_FUNCTION_SHIFT            (24)
#define MPI2_DOORBELL_ADD_DWORDS_MASK           (0x00FF0000)
#define MPI2_DOORBELL_ADD_DWORDS_SHIFT          (16)


/*
 * Defines for the WriteSequence register
 */
#define MPI2_WRITE_SEQUENCE_OFFSET              (0x00000004)
#define MPI2_WRSEQ_KEY_VALUE_MASK               (0x0000000F)
#define MPI2_WRSEQ_FLUSH_KEY_VALUE              (0x0)
#define MPI2_WRSEQ_1ST_KEY_VALUE                (0xF)
#define MPI2_WRSEQ_2ND_KEY_VALUE                (0x4)
#define MPI2_WRSEQ_3RD_KEY_VALUE                (0xB)
#define MPI2_WRSEQ_4TH_KEY_VALUE                (0x2)
#define MPI2_WRSEQ_5TH_KEY_VALUE                (0x7)
#define MPI2_WRSEQ_6TH_KEY_VALUE                (0xD)

/*
 * Defines for the HostDiagnostic register
 */
#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)

/*
 * 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)
```



```
/*****************************************************************************
*
*        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)

#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;


/* SCSI IO Request Descriptor */
typedef struct _MPI2_SCSI_IO_REQUEST_DESCRIPTOR
{
    U8              RequestFlags;               /* 0x00 */
    U8              MSIxIndex;                  /* 0x01 */
    U16             SMID;                       /* 0x02 */
    U16             LMID;                       /* 0x04 */
    U16             DevHandle;                  /* 0x06 */
} MPI2_SCSI_IO_REQUEST_DESCRIPTOR,
  MPI2_POINTER PTR_MPI2_SCSI_IO_REQUEST_DESCRIPTOR,
  Mpi2SCSIIORequestDescriptor_t, MPI2_POINTER pMpi2SCSIIORequestDescriptor_t;


/* SCSI Target Request Descriptor */
typedef struct _MPI2_SCSI_TARGET_REQUEST_DESCRIPTOR
{
    U8              RequestFlags;               /* 0x00 */
    U8              MSIxIndex;                  /* 0x01 */
```

```
    U16             SMID;                       /* 0x02 */
    U16             LMID;                       /* 0x04 */
    U16             IoIndex;                    /* 0x06 */
} MPI2_SCSI_TARGET_REQUEST_DESCRIPTOR,
  MPI2_POINTER PTR_MPI2_SCSI_TARGET_REQUEST_DESCRIPTOR,
  Mpi2SCSITargetRequestDescriptor_t,
  MPI2_POINTER pMpi2SCSITargetRequestDescriptor_t;


/* RAID Accelerator Request Descriptor */
typedef struct _MPI2_RAID_ACCEL_REQUEST_DESCRIPTOR
{
    U8              RequestFlags;               /* 0x00 */
    U8              MSIxIndex;                  /* 0x01 */
    U16             SMID;                       /* 0x02 */
    U16             LMID;                       /* 0x04 */
    U16             Reserved;                   /* 0x06 */
} MPI2_RAID_ACCEL_REQUEST_DESCRIPTOR,
  MPI2_POINTER PTR_MPI2_RAID_ACCEL_REQUEST_DESCRIPTOR,
  Mpi2RAIDAcceleratorRequestDescriptor_t,
  MPI2_POINTER pMpi2RAIDAcceleratorRequestDescriptor_t;


/* Fast Path SCSI IO Request Descriptor */
typedef MPI2_SCSI_IO_REQUEST_DESCRIPTOR
    MPI25_FP_SCSI_IO_REQUEST_DESCRIPTOR,
    MPI2_POINTER PTR_MPI25_FP_SCSI_IO_REQUEST_DESCRIPTOR,
    Mpi25FastPathSCSIIORequestDescriptor_t,
    MPI2_POINTER pMpi25FastPathSCSIIORequestDescriptor_t;


/* union of Request Descriptors */
typedef union _MPI2_REQUEST_DESCRIPTOR_UNION
{
    MPI2_DEFAULT_REQUEST_DESCRIPTOR             Default;
    MPI2_HIGH_PRIORITY_REQUEST_DESCRIPTOR       HighPriority;
    MPI2_SCSI_IO_REQUEST_DESCRIPTOR             SCSIIO;
    MPI2_SCSI_TARGET_REQUEST_DESCRIPTOR         SCSITarget;
    MPI2_RAID_ACCEL_REQUEST_DESCRIPTOR          RAIDAccelerator;
    MPI25_FP_SCSI_IO_REQUEST_DESCRIPTOR         FastPathSCSIIO;
    U64                                         Words;
} MPI2_REQUEST_DESCRIPTOR_UNION, MPI2_POINTER PTR_MPI2_REQUEST_DESCRIPTOR_UNION,
  Mpi2RequestDescriptorUnion_t, MPI2_POINTER pMpi2RequestDescriptorUnion_t;


/* Reply Descriptors */

/* Default Reply Descriptor */
typedef struct _MPI2_DEFAULT_REPLY_DESCRIPTOR
{
    U8              ReplyFlags;                 /* 0x00 */
    U8              MSIxIndex;                  /* 0x01 */
    U16             DescriptorTypeDependent1;   /* 0x02 */
    U32             DescriptorTypeDependent2;   /* 0x04 */
} MPI2_DEFAULT_REPLY_DESCRIPTOR, MPI2_POINTER PTR_MPI2_DEFAULT_REPLY_DESCRIPTOR,
  Mpi2DefaultReplyDescriptor_t, MPI2_POINTER pMpi2DefaultReplyDescriptor_t;

/* defines for the ReplyFlags field */
#define MPI2_RPY_DESCRIPT_FLAGS_TYPE_MASK                   (0x0F)
#define MPI2_RPY_DESCRIPT_FLAGS_SCSI_IO_SUCCESS             (0x00)
#define MPI2_RPY_DESCRIPT_FLAGS_ADDRESS_REPLY               (0x01)
#define MPI2_RPY_DESCRIPT_FLAGS_TARGETASSIST_SUCCESS        (0x02)
#define MPI2_RPY_DESCRIPT_FLAGS_TARGET_COMMAND_BUFFER       (0x03)
#define MPI2_RPY_DESCRIPT_FLAGS_RAID_ACCELERATOR_SUCCESS    (0x05)
#define MPI25_RPY_DESCRIPT_FLAGS_FAST_PATH_SCSI_IO_SUCCESS  (0x06)
#define MPI2_RPY_DESCRIPT_FLAGS_UNUSED                      (0x0F)
```



```
/* values for marking a reply descriptor as unused */
#define MPI2_RPY_DESCRIPT_UNUSED_WORD0_MARK             (0xFFFFFFFF)
#define MPI2_RPY_DESCRIPT_UNUSED_WORD1_MARK             (0xFFFFFFFF)

/* Address Reply Descriptor */
typedef struct _MPI2_ADDRESS_REPLY_DESCRIPTOR
{
    U8              ReplyFlags;                 /* 0x00 */
    U8              MSIxIndex;                  /* 0x01 */
    U16             SMID;                       /* 0x02 */
    U32             ReplyFrameAddress;          /* 0x04 */
} MPI2_ADDRESS_REPLY_DESCRIPTOR, MPI2_POINTER PTR_MPI2_ADDRESS_REPLY_DESCRIPTOR,
  Mpi2AddressReplyDescriptor_t, MPI2_POINTER pMpi2AddressReplyDescriptor_t;

#define MPI2_ADDRESS_REPLY_SMID_INVALID                 (0x00)


/* SCSI IO Success Reply Descriptor */
typedef struct _MPI2_SCSI_IO_SUCCESS_REPLY_DESCRIPTOR
{
    U8              ReplyFlags;                 /* 0x00 */
    U8              MSIxIndex;                  /* 0x01 */
    U16             SMID;                       /* 0x02 */
    U16             TaskTag;                    /* 0x04 */
    U16             Reserved1;                  /* 0x06 */
} MPI2_SCSI_IO_SUCCESS_REPLY_DESCRIPTOR,
  MPI2_POINTER PTR_MPI2_SCSI_IO_SUCCESS_REPLY_DESCRIPTOR,
  Mpi2SCSIIOSuccessReplyDescriptor_t,
  MPI2_POINTER pMpi2SCSIIOSuccessReplyDescriptor_t;


/* TargetAssist Success Reply Descriptor */
typedef struct _MPI2_TARGETASSIST_SUCCESS_REPLY_DESCRIPTOR
{
    U8              ReplyFlags;                 /* 0x00 */
    U8              MSIxIndex;                  /* 0x01 */
    U16             SMID;                       /* 0x02 */
    U8              SequenceNumber;             /* 0x04 */
    U8              Reserved1;                  /* 0x05 */
    U16             IoIndex;                    /* 0x06 */
} MPI2_TARGETASSIST_SUCCESS_REPLY_DESCRIPTOR,
  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;


/* 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;
    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



/* Doorbell functions */
#define MPI2_FUNCTION_IOC_MESSAGE_UNIT_RESET        (0x40)
#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)

/****************************************************************************
*  Config IOCStatus values
****************************************************************************/

#define MPI2_IOCSTATUS_CONFIG_INVALID_ACTION        (0x0020)
#define MPI2_IOCSTATUS_CONFIG_INVALID_TYPE          (0x0021)
#define MPI2_IOCSTATUS_CONFIG_INVALID_PAGE          (0x0022)
#define MPI2_IOCSTATUS_CONFIG_INVALID_DATA          (0x0023)
#define MPI2_IOCSTATUS_CONFIG_NO_DEFAULTS           (0x0024)
#define MPI2_IOCSTATUS_CONFIG_CANT_COMMIT           (0x0025)

/****************************************************************************
*  SCSI IO Reply
****************************************************************************/

#define MPI2_IOCSTATUS_SCSI_RECOVERED_ERROR         (0x0040)
#define MPI2_IOCSTATUS_SCSI_INVALID_DEVHANDLE       (0x0042)
#define MPI2_IOCSTATUS_SCSI_DEVICE_NOT_THERE        (0x0043)
#define MPI2_IOCSTATUS_SCSI_DATA_OVERRUN            (0x0044)
#define MPI2_IOCSTATUS_SCSI_DATA_UNDERRUN           (0x0045)
#define MPI2_IOCSTATUS_SCSI_IO_DATA_ERROR           (0x0046)
#define MPI2_IOCSTATUS_SCSI_PROTOCOL_ERROR          (0x0047)
#define MPI2_IOCSTATUS_SCSI_TASK_TERMINATED         (0x0048)
#define MPI2_IOCSTATUS_SCSI_RESIDUAL_MISMATCH       (0x0049)
#define MPI2_IOCSTATUS_SCSI_TASK_MGMT_FAILED        (0x004A)
#define MPI2_IOCSTATUS_SCSI_IOC_TERMINATED          (0x004B)
```

```
#define MPI2_IOCSTATUS_SCSI_EXT_TERMINATED          (0x004C)

/*****************************************************************************
*  For use by SCSI Initiator and SCSI Target end-to-end data protection
*****************************************************************************/

#define MPI2_IOCSTATUS_EEDP_GUARD_ERROR             (0x004D)
#define MPI2_IOCSTATUS_EEDP_REF_TAG_ERROR           (0x004E)
#define MPI2_IOCSTATUS_EEDP_APP_TAG_ERROR           (0x004F)

/*****************************************************************************
*  SCSI Target values
*****************************************************************************/

#define MPI2_IOCSTATUS_TARGET_INVALID_IO_INDEX      (0x0062)
#define MPI2_IOCSTATUS_TARGET_ABORTED               (0x0063)
#define MPI2_IOCSTATUS_TARGET_NO_CONN_RETRYABLE     (0x0064)
#define MPI2_IOCSTATUS_TARGET_NO_CONNECTION         (0x0065)
#define MPI2_IOCSTATUS_TARGET_XFER_COUNT_MISMATCH   (0x006A)
#define MPI2_IOCSTATUS_TARGET_DATA_OFFSET_ERROR     (0x006D)
#define MPI2_IOCSTATUS_TARGET_TOO_MUCH_WRITE_DATA   (0x006E)
#define MPI2_IOCSTATUS_TARGET_IU_TOO_SHORT          (0x006F)
#define MPI2_IOCSTATUS_TARGET_ACK_NAK_TIMEOUT       (0x0070)
#define MPI2_IOCSTATUS_TARGET_NAK_RECEIVED          (0x0071)

/*****************************************************************************
*  Serial Attached SCSI values
*****************************************************************************/

#define MPI2_IOCSTATUS_SAS_SMP_REQUEST_FAILED       (0x0090)
#define MPI2_IOCSTATUS_SAS_SMP_DATA_OVERRUN         (0x0091)

/*****************************************************************************
*  Diagnostic Buffer Post / Diagnostic Release values
*****************************************************************************/

#define MPI2_IOCSTATUS_DIAGNOSTIC_RELEASED          (0x00A0)

/*****************************************************************************
*  RAID Accelerator values
*****************************************************************************/

#define MPI2_IOCSTATUS_RAID_ACCEL_ERROR             (0x00B0)

/*****************************************************************************
*  IOCStatus flag to indicate that log info is available
*****************************************************************************/

#define MPI2_IOCSTATUS_FLAG_LOG_INFO_AVAILABLE      (0x8000)

/*****************************************************************************
*  IOCLogInfo Types
*****************************************************************************/

#define MPI2_IOCLOGINFO_TYPE_MASK               (0xF0000000)
#define MPI2_IOCLOGINFO_TYPE_SHIFT              (28)
#define MPI2_IOCLOGINFO_TYPE_NONE               (0x0)
#define MPI2_IOCLOGINFO_TYPE_SCSI               (0x1)
#define MPI2_IOCLOGINFO_TYPE_FC                 (0x2)
#define MPI2_IOCLOGINFO_TYPE_SAS                (0x3)
#define MPI2_IOCLOGINFO_TYPE_ISCSI              (0x4)
#define MPI2_IOCLOGINFO_LOG_DATA_MASK           (0x0FFFFFFF)


/*****************************************************************************
*
```



```
*        Standard Message Structures
*
*****************************************************************************/

/*****************************************************************************
* Request Message Header for all request messages
*****************************************************************************/

typedef struct _MPI2_REQUEST_HEADER
{
    U16             FunctionDependent1;         /* 0x00 */
    U8              ChainOffset;                /* 0x02 */
    U8              Function;                   /* 0x03 */
    U16             FunctionDependent2;         /* 0x04 */
    U8              FunctionDependent3;         /* 0x06 */
    U8              MsgFlags;                   /* 0x07 */
    U8              VP_ID;                      /* 0x08 */
    U8              VF_ID;                      /* 0x09 */
    U16             Reserved1;                  /* 0x0A */
} MPI2_REQUEST_HEADER, MPI2_POINTER PTR_MPI2_REQUEST_HEADER,
  MPI2RequestHeader_t, MPI2_POINTER pMPI2RequestHeader_t;


/*****************************************************************************
*  Default Reply
*****************************************************************************/

typedef struct _MPI2_DEFAULT_REPLY
{
    U16             FunctionDependent1;         /* 0x00 */
    U8              MsgLength;                  /* 0x02 */
    U8              Function;                   /* 0x03 */
    U16             FunctionDependent2;         /* 0x04 */
    U8              FunctionDependent3;         /* 0x06 */
    U8              MsgFlags;                   /* 0x07 */
    U8              VP_ID;                      /* 0x08 */
    U8              VF_ID;                      /* 0x09 */
    U16             Reserved1;                  /* 0x0A */
    U16             FunctionDependent5;         /* 0x0C */
    U16             IOCStatus;                  /* 0x0E */
    U32             IOCLogInfo;                 /* 0x10 */
} MPI2_DEFAULT_REPLY, MPI2_POINTER PTR_MPI2_DEFAULT_REPLY,
  MPI2DefaultReply_t, MPI2_POINTER pMPI2DefaultReply_t;


/* common version structure/union used in messages and configuration pages */

typedef struct _MPI2_VERSION_STRUCT
{
    U8                      Dev;                        /* 0x00 */
    U8                      Unit;                       /* 0x01 */
    U8                      Minor;                      /* 0x02 */
    U8                      Major;                      /* 0x03 */
} MPI2_VERSION_STRUCT;

typedef union _MPI2_VERSION_UNION
{
    MPI2_VERSION_STRUCT     Struct;
    U32                     Word;
} MPI2_VERSION_UNION;


/* LUN field defines, common to many structures */
#define MPI2_LUN_FIRST_LEVEL_ADDRESSING             (0x0000FFFF)
#define MPI2_LUN_SECOND_LEVEL_ADDRESSING            (0xFFFF0000)
#define MPI2_LUN_THIRD_LEVEL_ADDRESSING             (0x0000FFFF)
```

```
#define MPI2_LUN_FOURTH_LEVEL_ADDRESSING            (0xFFFF0000)
#define MPI2_LUN_LEVEL_1_WORD                       (0xFF00)
#define MPI2_LUN_LEVEL_1_DWORD                      (0x0000FF00)


/*****************************************************************************
*
*        Fusion-MPT MPI Scatter Gather Elements
*
*****************************************************************************/

/****************************************************************************
*  MPI Simple Element structures
****************************************************************************/

typedef struct _MPI2_SGE_SIMPLE32
{
    U32                     FlagsLength;
    U32                     Address;
} MPI2_SGE_SIMPLE32, MPI2_POINTER PTR_MPI2_SGE_SIMPLE32,
  Mpi2SGESimple32_t, MPI2_POINTER pMpi2SGESimple32_t;

typedef struct _MPI2_SGE_SIMPLE64
{
    U32                     FlagsLength;
    U64                     Address;
} MPI2_SGE_SIMPLE64, MPI2_POINTER PTR_MPI2_SGE_SIMPLE64,
  Mpi2SGESimple64_t, MPI2_POINTER pMpi2SGESimple64_t;

typedef struct _MPI2_SGE_SIMPLE_UNION
{
    U32                     FlagsLength;
    union
    {
        U32                 Address32;
        U64                 Address64;
    } u;
} MPI2_SGE_SIMPLE_UNION, MPI2_POINTER PTR_MPI2_SGE_SIMPLE_UNION,
  Mpi2SGESimpleUnion_t, MPI2_POINTER pMpi2SGESimpleUnion_t;


/****************************************************************************
*  MPI Chain Element structures - for MPI v2.0 products only
****************************************************************************/

typedef struct _MPI2_SGE_CHAIN32
{
    U16                     Length;
    U8                      NextChainOffset;
    U8                      Flags;
    U32                     Address;
} MPI2_SGE_CHAIN32, MPI2_POINTER PTR_MPI2_SGE_CHAIN32,
  Mpi2SGEChain32_t, MPI2_POINTER pMpi2SGEChain32_t;

typedef struct _MPI2_SGE_CHAIN64
{
    U16                     Length;
    U8                      NextChainOffset;
    U8                      Flags;
    U64                     Address;
} MPI2_SGE_CHAIN64, MPI2_POINTER PTR_MPI2_SGE_CHAIN64,
  Mpi2SGEChain64_t, MPI2_POINTER pMpi2SGEChain64_t;

typedef struct _MPI2_SGE_CHAIN_UNION
{
    U16                     Length;
```



```
    U8                      NextChainOffset;
    U8                      Flags;
    union
    {
        U32                 Address32;
        U64                 Address64;
    } u;
} MPI2_SGE_CHAIN_UNION, MPI2_POINTER PTR_MPI2_SGE_CHAIN_UNION,
  Mpi2SGEChainUnion_t, MPI2_POINTER pMpi2SGEChainUnion_t;


/****************************************************************************
*  MPI Transaction Context Element structures - for MPI v2.0 products only
****************************************************************************/

typedef struct _MPI2_SGE_TRANSACTION32
{
    U8                      Reserved;
    U8                      ContextSize;
    U8                      DetailsLength;
    U8                      Flags;
    U32                     TransactionContext[1];
    U32                     TransactionDetails[1];
} MPI2_SGE_TRANSACTION32, MPI2_POINTER PTR_MPI2_SGE_TRANSACTION32,
  Mpi2SGETransaction32_t, MPI2_POINTER pMpi2SGETransaction32_t;

typedef struct _MPI2_SGE_TRANSACTION64
{
    U8                      Reserved;
    U8                      ContextSize;
    U8                      DetailsLength;
    U8                      Flags;
    U32                     TransactionContext[2];
    U32                     TransactionDetails[1];
} MPI2_SGE_TRANSACTION64, MPI2_POINTER PTR_MPI2_SGE_TRANSACTION64,
  Mpi2SGETransaction64_t, MPI2_POINTER pMpi2SGETransaction64_t;

typedef struct _MPI2_SGE_TRANSACTION96
{
    U8                      Reserved;
    U8                      ContextSize;
    U8                      DetailsLength;
    U8                      Flags;
    U32                     TransactionContext[3];
    U32                     TransactionDetails[1];
} MPI2_SGE_TRANSACTION96, MPI2_POINTER PTR_MPI2_SGE_TRANSACTION96,
  Mpi2SGETransaction96_t, MPI2_POINTER pMpi2SGETransaction96_t;

typedef struct _MPI2_SGE_TRANSACTION128
{
    U8                      Reserved;
    U8                      ContextSize;
    U8                      DetailsLength;
    U8                      Flags;
    U32                     TransactionContext[4];
    U32                     TransactionDetails[1];
} MPI2_SGE_TRANSACTION128, MPI2_POINTER PTR_MPI2_SGE_TRANSACTION128,
  Mpi2SGETransaction_t128, MPI2_POINTER pMpi2SGETransaction_t128;

typedef struct _MPI2_SGE_TRANSACTION_UNION
{
    U8                      Reserved;
    U8                      ContextSize;
    U8                      DetailsLength;
    U8                      Flags;
    union
```

```
    {
        U32                 TransactionContext32[1];
        U32                 TransactionContext64[2];
        U32                 TransactionContext96[3];
        U32                 TransactionContext128[4];
    } u;
    U32                     TransactionDetails[1];
} MPI2_SGE_TRANSACTION_UNION, MPI2_POINTER PTR_MPI2_SGE_TRANSACTION_UNION,
  Mpi2SGETransactionUnion_t, MPI2_POINTER pMpi2SGETransactionUnion_t;


/*****************************************************************************
*  MPI SGE union for IO SGL's - for MPI v2.0 products only
*****************************************************************************/

typedef struct _MPI2_MPI_SGE_IO_UNION
{
    union
    {
        MPI2_SGE_SIMPLE_UNION   Simple;
        MPI2_SGE_CHAIN_UNION    Chain;
    } u;
} MPI2_MPI_SGE_IO_UNION, MPI2_POINTER PTR_MPI2_MPI_SGE_IO_UNION,
  Mpi2MpiSGEIOUnion_t, MPI2_POINTER pMpi2MpiSGEIOUnion_t;


/*****************************************************************************
*  MPI SGE union for SGL's with Simple and Transaction elements - for MPI v2.0 p
*****************************************************************************/

typedef struct _MPI2_SGE_TRANS_SIMPLE_UNION
{
    union
    {
        MPI2_SGE_SIMPLE_UNION       Simple;
        MPI2_SGE_TRANSACTION_UNION  Transaction;
    } u;
} MPI2_SGE_TRANS_SIMPLE_UNION, MPI2_POINTER PTR_MPI2_SGE_TRANS_SIMPLE_UNION,
  Mpi2SGETransSimpleUnion_t, MPI2_POINTER pMpi2SGETransSimpleUnion_t;


/*****************************************************************************
*  All MPI SGE types union
*****************************************************************************/

typedef struct _MPI2_MPI_SGE_UNION
{
    union
    {
        MPI2_SGE_SIMPLE_UNION       Simple;
        MPI2_SGE_CHAIN_UNION        Chain;
        MPI2_SGE_TRANSACTION_UNION  Transaction;
    } u;
} MPI2_MPI_SGE_UNION, MPI2_POINTER PTR_MPI2_MPI_SGE_UNION,
  Mpi2MpiSgeUnion_t, MPI2_POINTER pMpi2MpiSgeUnion_t;


/*****************************************************************************
*  MPI SGE field definition and masks
*****************************************************************************/

/* Flags field bit definitions */

#define MPI2_SGE_FLAGS_LAST_ELEMENT             (0x80)
#define MPI2_SGE_FLAGS_END_OF_BUFFER            (0x40)
#define MPI2_SGE_FLAGS_ELEMENT_TYPE_MASK        (0x30)
```



```
#define MPI2_SGE_FLAGS_LOCAL_ADDRESS            (0x08)
#define MPI2_SGE_FLAGS_DIRECTION                (0x04)
#define MPI2_SGE_FLAGS_ADDRESS_SIZE             (0x02)
#define MPI2_SGE_FLAGS_END_OF_LIST              (0x01)

#define MPI2_SGE_FLAGS_SHIFT                    (24)

#define MPI2_SGE_LENGTH_MASK                    (0x00FFFFFF)
#define MPI2_SGE_CHAIN_LENGTH_MASK              (0x0000FFFF)

/* Element Type */

#define MPI2_SGE_FLAGS_TRANSACTION_ELEMENT      (0x00) /* for MPI v2.0 products
#define MPI2_SGE_FLAGS_SIMPLE_ELEMENT           (0x10)
#define MPI2_SGE_FLAGS_CHAIN_ELEMENT            (0x30) /* for MPI v2.0 products
#define MPI2_SGE_FLAGS_ELEMENT_MASK             (0x30)

/* Address location */

#define MPI2_SGE_FLAGS_SYSTEM_ADDRESS           (0x00)

/* Direction */

#define MPI2_SGE_FLAGS_IOC_TO_HOST              (0x00)
#define MPI2_SGE_FLAGS_HOST_TO_IOC              (0x04)

#define MPI2_SGE_FLAGS_DEST                     (MPI2_SGE_FLAGS_IOC_TO_HOST)
#define MPI2_SGE_FLAGS_SOURCE                   (MPI2_SGE_FLAGS_HOST_TO_IOC)

/* Address Size */

#define MPI2_SGE_FLAGS_32_BIT_ADDRESSING        (0x00)
#define MPI2_SGE_FLAGS_64_BIT_ADDRESSING        (0x02)

/* Context Size */

#define MPI2_SGE_FLAGS_32_BIT_CONTEXT           (0x00)
#define MPI2_SGE_FLAGS_64_BIT_CONTEXT           (0x02)
#define MPI2_SGE_FLAGS_96_BIT_CONTEXT           (0x04)
#define MPI2_SGE_FLAGS_128_BIT_CONTEXT          (0x06)

#define MPI2_SGE_CHAIN_OFFSET_MASK              (0x00FF0000)
#define MPI2_SGE_CHAIN_OFFSET_SHIFT             (16)

/*****************************************************************************
*  MPI SGE operation Macros
*****************************************************************************/

/* SIMPLE FlagsLength manipulations... */
#define MPI2_SGE_SET_FLAGS(f)          ((U32)(f) << MPI2_SGE_FLAGS_SHIFT)
#define MPI2_SGE_GET_FLAGS(f)          (((f) & ~MPI2_SGE_LENGTH_MASK) >> MPI2_SG
#define MPI2_SGE_LENGTH(f)             ((f) & MPI2_SGE_LENGTH_MASK)
#define MPI2_SGE_CHAIN_LENGTH(f)       ((f) & MPI2_SGE_CHAIN_LENGTH_MASK)

#define MPI2_SGE_SET_FLAGS_LENGTH(f,l) (MPI2_SGE_SET_FLAGS(f) | MPI2_SGE_LENGTH(

#define MPI2_pSGE_GET_FLAGS(psg)           MPI2_SGE_GET_FLAGS((psg)->FlagsLengt
#define MPI2_pSGE_GET_LENGTH(psg)          MPI2_SGE_LENGTH((psg)->FlagsLength)
#define MPI2_pSGE_SET_FLAGS_LENGTH(psg,f,l) (psg)->FlagsLength = MPI2_SGE_SET_FL

/* CAUTION - The following are READ-MODIFY-WRITE! */
#define MPI2_pSGE_SET_FLAGS(psg,f)      (psg)->FlagsLength |= MPI2_SGE_SET_FLAGS
#define MPI2_pSGE_SET_LENGTH(psg,l)     (psg)->FlagsLength |= MPI2_SGE_LENGTH(l)

#define MPI2_GET_CHAIN_OFFSET(x)    ((x & MPI2_SGE_CHAIN_OFFSET_MASK) >> MPI2_SG
```

```
/*****************************************************************************
*
*        Fusion-MPT IEEE Scatter Gather Elements
*
*****************************************************************************/

/****************************************************************************
*  IEEE Simple Element structures
****************************************************************************/

/* MPI2_IEEE_SGE_SIMPLE32 is for MPI v2.0 products only */
typedef struct _MPI2_IEEE_SGE_SIMPLE32
{
    U32                     Address;
    U32                     FlagsLength;
} MPI2_IEEE_SGE_SIMPLE32, MPI2_POINTER PTR_MPI2_IEEE_SGE_SIMPLE32,
  Mpi2IeeeSgeSimple32_t, MPI2_POINTER pMpi2IeeeSgeSimple32_t;

typedef struct _MPI2_IEEE_SGE_SIMPLE64
{
    U64                     Address;
    U32                     Length;
    U16                     Reserved1;
    U8                      Reserved2;
    U8                      Flags;
} MPI2_IEEE_SGE_SIMPLE64, MPI2_POINTER PTR_MPI2_IEEE_SGE_SIMPLE64,
  Mpi2IeeeSgeSimple64_t, MPI2_POINTER pMpi2IeeeSgeSimple64_t;

typedef union _MPI2_IEEE_SGE_SIMPLE_UNION
{
    MPI2_IEEE_SGE_SIMPLE32  Simple32;
    MPI2_IEEE_SGE_SIMPLE64  Simple64;
} MPI2_IEEE_SGE_SIMPLE_UNION, MPI2_POINTER PTR_MPI2_IEEE_SGE_SIMPLE_UNION,
  Mpi2IeeeSgeSimpleUnion_t, MPI2_POINTER pMpi2IeeeSgeSimpleUnion_t;


/****************************************************************************
*  IEEE Chain Element structures
****************************************************************************/

/* MPI2_IEEE_SGE_CHAIN32 is for MPI v2.0 products only */
typedef MPI2_IEEE_SGE_SIMPLE32  MPI2_IEEE_SGE_CHAIN32;

/* MPI2_IEEE_SGE_CHAIN64 is for MPI v2.0 products only */
typedef MPI2_IEEE_SGE_SIMPLE64  MPI2_IEEE_SGE_CHAIN64;

typedef union _MPI2_IEEE_SGE_CHAIN_UNION
{
    MPI2_IEEE_SGE_CHAIN32   Chain32;
    MPI2_IEEE_SGE_CHAIN64   Chain64;
} MPI2_IEEE_SGE_CHAIN_UNION, MPI2_POINTER PTR_MPI2_IEEE_SGE_CHAIN_UNION,
  Mpi2IeeeSgeChainUnion_t, MPI2_POINTER pMpi2IeeeSgeChainUnion_t;

/* MPI25_IEEE_SGE_CHAIN64 is for MPI v2.5 products only */
typedef struct _MPI25_IEEE_SGE_CHAIN64
{
    U64                     Address;
    U32                     Length;
    U16                     Reserved1;
    U8                      NextChainOffset;
    U8                      Flags;
} MPI25_IEEE_SGE_CHAIN64, MPI2_POINTER PTR_MPI25_IEEE_SGE_CHAIN64,
  Mpi25IeeeSgeChain64_t, MPI2_POINTER pMpi25IeeeSgeChain64_t;
```



```
/****************************************************************************
*  All IEEE SGE types union
****************************************************************************/

/* MPI2_IEEE_SGE_UNION is for MPI v2.0 products only */
typedef struct _MPI2_IEEE_SGE_UNION
{
    union
    {
        MPI2_IEEE_SGE_SIMPLE_UNION  Simple;
        MPI2_IEEE_SGE_CHAIN_UNION   Chain;
    } u;
} MPI2_IEEE_SGE_UNION, MPI2_POINTER PTR_MPI2_IEEE_SGE_UNION,
  Mpi2IeeeSgeUnion_t, MPI2_POINTER pMpi2IeeeSgeUnion_t;


/****************************************************************************
*  IEEE SGE union for IO SGL's
****************************************************************************/

typedef union _MPI25_SGE_IO_UNION
{
    MPI2_IEEE_SGE_SIMPLE64      IeeeSimple;
    MPI25_IEEE_SGE_CHAIN64      IeeeChain;
} MPI25_SGE_IO_UNION, MPI2_POINTER PTR_MPI25_SGE_IO_UNION,
  Mpi25SGEIOUnion_t, MPI2_POINTER pMpi25SGEIOUnion_t;


/****************************************************************************
*  IEEE SGE field definitions and masks
****************************************************************************/

/* Flags field bit definitions */

#define MPI2_IEEE_SGE_FLAGS_ELEMENT_TYPE_MASK   (0x80)
#define MPI25_IEEE_SGE_FLAGS_END_OF_LIST        (0x40)

#define MPI2_IEEE32_SGE_FLAGS_SHIFT             (24)

#define MPI2_IEEE32_SGE_LENGTH_MASK             (0x00FFFFFF)

/* Element Type */

#define MPI2_IEEE_SGE_FLAGS_SIMPLE_ELEMENT      (0x00)
#define MPI2_IEEE_SGE_FLAGS_CHAIN_ELEMENT       (0x80)

/* Data Location Address Space */

#define MPI2_IEEE_SGE_FLAGS_ADDR_MASK           (0x03)
#define MPI2_IEEE_SGE_FLAGS_SYSTEM_ADDR         (0x00) /* IEEE Simple Element on
#define MPI2_IEEE_SGE_FLAGS_IOCDDR_ADDR         (0x01) /* IEEE Simple Element on
#define MPI2_IEEE_SGE_FLAGS_IOCPLB_ADDR         (0x02)
#define MPI2_IEEE_SGE_FLAGS_IOCPLBNTA_ADDR      (0x03) /* IEEE Simple Element on
#define MPI2_IEEE_SGE_FLAGS_SYSTEMPLBCPI_ADDR   (0x03) /* IEEE Chain Element onl


/****************************************************************************
*  IEEE SGE operation Macros
****************************************************************************/

/* SIMPLE FlagsLength manipulations... */
#define MPI2_IEEE32_SGE_SET_FLAGS(f)     ((U32)(f) << MPI2_IEEE32_SGE_FLAGS_SHIF
#define MPI2_IEEE32_SGE_GET_FLAGS(f)     (((f) & ~MPI2_IEEE32_SGE_LENGTH_MASK) >
#define MPI2_IEEE32_SGE_LENGTH(f)        ((f) & MPI2_IEEE32_SGE_LENGTH_MASK)

#define MPI2_IEEE32_SGE_SET_FLAGS_LENGTH(f, l)      (MPI2_IEEE32_SGE_SET_FLAGS(f
```

```
#define MPI2_IEEE32_pSGE_GET_FLAGS(psg)             MPI2_IEEE32_SGE_GET_FLAGS((p
#define MPI2_IEEE32_pSGE_GET_LENGTH(psg)            MPI2_IEEE32_SGE_LENGTH((psg)
#define MPI2_IEEE32_pSGE_SET_FLAGS_LENGTH(psg,f,l)  (psg)->FlagsLength = MPI2_IE

/* CAUTION - The following are READ-MODIFY-WRITE! */
#define MPI2_IEEE32_pSGE_SET_FLAGS(psg,f)    (psg)->FlagsLength |= MPI2_IEEE32_S
#define MPI2_IEEE32_pSGE_SET_LENGTH(psg,l)   (psg)->FlagsLength |= MPI2_IEEE32_S




/*****************************************************************************
*
*        Fusion-MPT MPI/IEEE Scatter Gather Unions
*
*****************************************************************************/

typedef union _MPI2_SIMPLE_SGE_UNION
{
    MPI2_SGE_SIMPLE_UNION       MpiSimple;
    MPI2_IEEE_SGE_SIMPLE_UNION  IeeeSimple;
} MPI2_SIMPLE_SGE_UNION, MPI2_POINTER PTR_MPI2_SIMPLE_SGE_UNION,
  Mpi2SimpleSgeUntion_t, MPI2_POINTER pMpi2SimpleSgeUntion_t;


typedef union _MPI2_SGE_IO_UNION
{
    MPI2_SGE_SIMPLE_UNION       MpiSimple;
    MPI2_SGE_CHAIN_UNION        MpiChain;
    MPI2_IEEE_SGE_SIMPLE_UNION  IeeeSimple;
    MPI2_IEEE_SGE_CHAIN_UNION   IeeeChain;
} MPI2_SGE_IO_UNION, MPI2_POINTER PTR_MPI2_SGE_IO_UNION,
  Mpi2SGEIOUnion_t, MPI2_POINTER pMpi2SGEIOUnion_t;


/*****************************************************************************
*
*  Values for SGLFlags field, used in many request messages with an SGL
*
*****************************************************************************/

/* values for MPI SGL Data Location Address Space subfield */
#define MPI2_SGLFLAGS_ADDRESS_SPACE_MASK            (0x0C)
#define MPI2_SGLFLAGS_SYSTEM_ADDRESS_SPACE          (0x00)
#define MPI2_SGLFLAGS_IOCDDR_ADDRESS_SPACE          (0x04)
#define MPI2_SGLFLAGS_IOCPLB_ADDRESS_SPACE          (0x08)
#define MPI2_SGLFLAGS_IOCPLBNTA_ADDRESS_SPACE       (0x0C)
/* values for SGL Type subfield */
#define MPI2_SGLFLAGS_SGL_TYPE_MASK                 (0x03)
#define MPI2_SGLFLAGS_SGL_TYPE_MPI                  (0x00)
#define MPI2_SGLFLAGS_SGL_TYPE_IEEE32               (0x01)
#define MPI2_SGLFLAGS_SGL_TYPE_IEEE64               (0x02)


#endif

#endif /* ! codereview */
```

```
**********************************************************
  144624 Thu Jun 12 17:28:23 2014
new/usr/src/uts/common/sys/scsi/adapters/mpt_sas3/mpi/mpi2_cnfg.h
4546 mpt_sas needs enhancing to support LSI MPI2.5
**********************************************************
/*
 * 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.
 *
 * 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.
 *
 *  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.
 *  --------------------------------------------------------------------------
 */

#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)


/*****************************************************************************
*   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 */
#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)

#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,
  MPI2_POINTER PTR_MPI2_CONFIG_PAGE_MAN_0,
  Mpi2ManufacturingPage0_t, MPI2_POINTER pMpi2ManufacturingPage0_t;

#define MPI2_MANUFACTURING0_PAGEVERSION                (0x00)


/* Manufacturing Page 1 */

typedef struct _MPI2_CONFIG_PAGE_MAN_1
{
    MPI2_CONFIG_PAGE_HEADER Header;                     /* 0x00 */
    U8                      VPD[256];                   /* 0x04 */
} MPI2_CONFIG_PAGE_MAN_1,
  MPI2_POINTER PTR_MPI2_CONFIG_PAGE_MAN_1,
  Mpi2ManufacturingPage1_t, MPI2_POINTER pMpi2ManufacturingPage1_t;

#define MPI2_MANUFACTURING1_PAGEVERSION                (0x00)


typedef struct _MPI2_CHIP_REVISION_ID
{
    U16 DeviceID;                                       /* 0x00 */
    U8  PCIRevisionID;                                  /* 0x02 */
    U8  Reserved;                                       /* 0x03 */
} MPI2_CHIP_REVISION_ID, MPI2_POINTER PTR_MPI2_CHIP_REVISION_ID,
  Mpi2ChipRevisionId_t, MPI2_POINTER pMpi2ChipRevisionId_t;


/* Manufacturing Page 2 */

/*
 * Host code (drivers, BIOS, utilities, etc.) should leave this define set to
 * one and check Header.PageLength at runtime.
 */
#ifndef MPI2_MAN_PAGE_2_HW_SETTINGS_WORDS
#define MPI2_MAN_PAGE_2_HW_SETTINGS_WORDS   (1)
#endif

typedef struct _MPI2_CONFIG_PAGE_MAN_2
{
    MPI2_CONFIG_PAGE_HEADER Header;                     /* 0x00 */
    MPI2_CHIP_REVISION_ID   ChipId;                     /* 0x04 */
    U32                     HwSettings[MPI2_MAN_PAGE_2_HW_SETTINGS_WORDS];/* 0x0
} MPI2_CONFIG_PAGE_MAN_2,
```

```
  MPI2_POINTER PTR_MPI2_CONFIG_PAGE_MAN_2,
  Mpi2ManufacturingPage2_t, MPI2_POINTER pMpi2ManufacturingPage2_t;

#define MPI2_MANUFACTURING2_PAGEVERSION                 (0x00)


/* Manufacturing Page 3 */

/*
 * Host code (drivers, BIOS, utilities, etc.) should leave this define set to
 * one and check Header.PageLength at runtime.
 */
#ifndef MPI2_MAN_PAGE_3_INFO_WORDS
#define MPI2_MAN_PAGE_3_INFO_WORDS          (1)
#endif

typedef struct _MPI2_CONFIG_PAGE_MAN_3
{
    MPI2_CONFIG_PAGE_HEADER             Header;         /* 0x00 */
    MPI2_CHIP_REVISION_ID               ChipId;         /* 0x04 */
    U32                                 Info[MPI2_MAN_PAGE_3_INFO_WORDS];/* 0x08
} MPI2_CONFIG_PAGE_MAN_3,
  MPI2_POINTER PTR_MPI2_CONFIG_PAGE_MAN_3,
  Mpi2ManufacturingPage3_t, MPI2_POINTER pMpi2ManufacturingPage3_t;

#define MPI2_MANUFACTURING3_PAGEVERSION                 (0x00)


/* Manufacturing Page 4 */

typedef struct _MPI2_MANPAGE4_PWR_SAVE_SETTINGS
{
    U8                          PowerSaveFlags;                 /* 0x00 */
    U8                          InternalOperationsSleepTime;    /* 0x01 */
    U8                          InternalOperationsRunTime;      /* 0x02 */
    U8                          HostIdleTime;                   /* 0x03 */
} MPI2_MANPAGE4_PWR_SAVE_SETTINGS,
  MPI2_POINTER PTR_MPI2_MANPAGE4_PWR_SAVE_SETTINGS,
  Mpi2ManPage4PwrSaveSettings_t, MPI2_POINTER pMpi2ManPage4PwrSaveSettings_t;

/* defines for the PowerSaveFlags field */
#define MPI2_MANPAGE4_MASK_POWERSAVE_MODE               (0x03)
#define MPI2_MANPAGE4_POWERSAVE_MODE_DISABLED           (0x00)
#define MPI2_MANPAGE4_CUSTOM_POWERSAVE_MODE             (0x01)
#define MPI2_MANPAGE4_FULL_POWERSAVE_MODE               (0x02)

typedef struct _MPI2_CONFIG_PAGE_MAN_4
{
    MPI2_CONFIG_PAGE_HEADER             Header;                 /* 0x00 */
    U32                                 Reserved1;              /* 0x04 */
    U32                                 Flags;                  /* 0x08 */
    U8                                  InquirySize;            /* 0x0C */
    U8                                  Reserved2;              /* 0x0D */
    U16                                 Reserved3;              /* 0x0E */
    U8                                  InquiryData[56];        /* 0x10 */
    U32                                 RAID0VolumeSettings;    /* 0x48 */
    U32                                 RAID1EVolumeSettings;   /* 0x4C */
    U32                                 RAID1VolumeSettings;    /* 0x50 */
    U32                                 RAID10VolumeSettings;   /* 0x54 */
    U32                                 Reserved4;              /* 0x58 */
    U32                                 Reserved5;              /* 0x5C */
    MPI2_MANPAGE4_PWR_SAVE_SETTINGS     PowerSaveSettings;      /* 0x60 */
    U8                                  MaxOCEDisks;            /* 0x64 */
    U8                                  ResyncRate;             /* 0x65 */
    U16                                 DataScrubDuration;      /* 0x66 */
    U8                                  MaxHotSpares;           /* 0x68 */
```



```
    U8                                  MaxPhysDisksPerVol;     /* 0x69 */
    U8                                  MaxPhysDisks;           /* 0x6A */
    U8                                  MaxVolumes;             /* 0x6B */
} MPI2_CONFIG_PAGE_MAN_4,
  MPI2_POINTER PTR_MPI2_CONFIG_PAGE_MAN_4,
  Mpi2ManufacturingPage4_t, MPI2_POINTER pMpi2ManufacturingPage4_t;

#define MPI2_MANUFACTURING4_PAGEVERSION                 (0x0A)

/* Manufacturing Page 4 Flags field */
#define MPI2_MANPAGE4_METADATA_SIZE_MASK                (0x00030000)
#define MPI2_MANPAGE4_METADATA_512MB                    (0x00000000)

#define MPI2_MANPAGE4_MIX_SSD_SAS_SATA                  (0x00008000)
#define MPI2_MANPAGE4_MIX_SSD_AND_NON_SSD               (0x00004000)
#define MPI2_MANPAGE4_HIDE_PHYSDISK_NON_IR              (0x00002000)

#define MPI2_MANPAGE4_MASK_PHYSDISK_COERCION            (0x00001C00)
#define MPI2_MANPAGE4_PHYSDISK_COERCION_1GB             (0x00000000)
#define MPI2_MANPAGE4_PHYSDISK_128MB_COERCION           (0x00000400)
#define MPI2_MANPAGE4_PHYSDISK_ADAPTIVE_COERCION        (0x00000800)
#define MPI2_MANPAGE4_PHYSDISK_ZERO_COERCION            (0x00000C00)

#define MPI2_MANPAGE4_MASK_BAD_BLOCK_MARKING            (0x00000300)
#define MPI2_MANPAGE4_DEFAULT_BAD_BLOCK_MARKING         (0x00000000)
#define MPI2_MANPAGE4_TABLE_BAD_BLOCK_MARKING           (0x00000100)
#define MPI2_MANPAGE4_WRITE_LONG_BAD_BLOCK_MARKING      (0x00000200)

#define MPI2_MANPAGE4_FORCE_OFFLINE_FAILOVER            (0x00000080)
#define MPI2_MANPAGE4_RAID10_DISABLE                    (0x00000040)
#define MPI2_MANPAGE4_RAID1E_DISABLE                    (0x00000020)
#define MPI2_MANPAGE4_RAID1_DISABLE                     (0x00000010)
#define MPI2_MANPAGE4_RAID0_DISABLE                     (0x00000008)
#define MPI2_MANPAGE4_IR_MODEPAGE8_DISABLE              (0x00000004)
#define MPI2_MANPAGE4_IM_RESYNC_CACHE_ENABLE            (0x00000002)
#define MPI2_MANPAGE4_IR_NO_MIX_SAS_SATA                (0x00000001)


/* Manufacturing Page 5 */

/*
 * Host code (drivers, BIOS, utilities, etc.) should leave this define set to
 * one and check Header.PageLength or NumPhys at runtime.
 */
#ifndef MPI2_MAN_PAGE_5_PHY_ENTRIES
#define MPI2_MAN_PAGE_5_PHY_ENTRIES         (1)
#endif

typedef struct _MPI2_MANUFACTURING5_ENTRY
{
    U64                                 WWID;           /* 0x00 */
    U64                                 DeviceName;     /* 0x08 */
} MPI2_MANUFACTURING5_ENTRY, MPI2_POINTER PTR_MPI2_MANUFACTURING5_ENTRY,
  Mpi2Manufacturing5Entry_t, MPI2_POINTER pMpi2Manufacturing5Entry_t;

typedef struct _MPI2_CONFIG_PAGE_MAN_5
{
    MPI2_CONFIG_PAGE_HEADER             Header;         /* 0x00 */
    U8                                  NumPhys;        /* 0x04 */
    U8                                  Reserved1;      /* 0x05 */
    U16                                 Reserved2;      /* 0x06 */
    U32                                 Reserved3;      /* 0x08 */
    U32                                 Reserved4;      /* 0x0C */
    MPI2_MANUFACTURING5_ENTRY           Phy[MPI2_MAN_PAGE_5_PHY_ENTRIES];/* 0x08
} MPI2_CONFIG_PAGE_MAN_5,
  MPI2_POINTER PTR_MPI2_CONFIG_PAGE_MAN_5,
```

```
  Mpi2ManufacturingPage5_t, MPI2_POINTER pMpi2ManufacturingPage5_t;

#define MPI2_MANUFACTURING5_PAGEVERSION                 (0x03)


/* Manufacturing Page 6 */

typedef struct _MPI2_CONFIG_PAGE_MAN_6
{
    MPI2_CONFIG_PAGE_HEADER         Header;             /* 0x00 */
    U32                             ProductSpecificInfo;/* 0x04 */
} MPI2_CONFIG_PAGE_MAN_6,
  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 */
    U16                         Slot;                   /* 0x16 */
    U32                         Reserved2;              /* 0x18 */
} MPI2_MANPAGE7_CONNECTOR_INFO, MPI2_POINTER PTR_MPI2_MANPAGE7_CONNECTOR_INFO,
  Mpi2ManPage7ConnectorInfo_t, MPI2_POINTER pMpi2ManPage7ConnectorInfo_t;

/* defines for the Pinout field */
#define MPI2_MANPAGE7_PINOUT_SFF_8484_L4                (0x00080000)
#define MPI2_MANPAGE7_PINOUT_SFF_8484_L3                (0x00040000)
#define MPI2_MANPAGE7_PINOUT_SFF_8484_L2                (0x00020000)
#define MPI2_MANPAGE7_PINOUT_SFF_8484_L1                (0x00010000)
#define MPI2_MANPAGE7_PINOUT_SFF_8470_L4                (0x00000800)
#define MPI2_MANPAGE7_PINOUT_SFF_8470_L3                (0x00000400)
#define MPI2_MANPAGE7_PINOUT_SFF_8470_L2                (0x00000200)
#define MPI2_MANPAGE7_PINOUT_SFF_8470_L1                (0x00000100)
#define MPI2_MANPAGE7_PINOUT_SFF_8482                   (0x00000002)
#define MPI2_MANPAGE7_PINOUT_CONNECTION_UNKNOWN         (0x00000001)

/* defines for the Location field */
#define MPI2_MANPAGE7_LOCATION_UNKNOWN                  (0x01)
#define MPI2_MANPAGE7_LOCATION_INTERNAL                 (0x02)
#define MPI2_MANPAGE7_LOCATION_EXTERNAL                 (0x04)
#define MPI2_MANPAGE7_LOCATION_SWITCHABLE               (0x08)
#define MPI2_MANPAGE7_LOCATION_AUTO                     (0x10)
#define MPI2_MANPAGE7_LOCATION_NOT_PRESENT              (0x20)
#define MPI2_MANPAGE7_LOCATION_NOT_CONNECTED            (0x80)

/*
 * Host code (drivers, BIOS, utilities, etc.) should leave this define set to
 * one and check NumPhys at runtime.
 */
#ifndef MPI2_MANPAGE7_CONNECTOR_INFO_MAX
#define MPI2_MANPAGE7_CONNECTOR_INFO_MAX  (1)
#endif

typedef struct _MPI2_CONFIG_PAGE_MAN_7
{
    MPI2_CONFIG_PAGE_HEADER         Header;             /* 0x00 */
    U32                             Reserved1;          /* 0x04 */
    U32                             Reserved2;          /* 0x08 */
    U32                             Flags;              /* 0x0C */
```



```
    U8                              EnclosureName[16];  /* 0x10 */
    U8                              NumPhys;            /* 0x20 */
    U8                              Reserved3;          /* 0x21 */
    U16                             Reserved4;          /* 0x22 */
    MPI2_MANPAGE7_CONNECTOR_INFO    ConnectorInfo[MPI2_MANPAGE7_CONNECTOR_INFO_M
} MPI2_CONFIG_PAGE_MAN_7,
  MPI2_POINTER PTR_MPI2_CONFIG_PAGE_MAN_7,
  Mpi2ManufacturingPage7_t, MPI2_POINTER pMpi2ManufacturingPage7_t;

#define MPI2_MANUFACTURING7_PAGEVERSION                 (0x01)

/* defines for the Flags field */
#define MPI2_MANPAGE7_FLAG_USE_SLOT_INFO                (0x00000001)


/*
 * Generic structure to use for product-specific manufacturing pages
 * (currently Manufacturing Page 8 through Manufacturing Page 31).
 */

typedef struct _MPI2_CONFIG_PAGE_MAN_PS
{
    MPI2_CONFIG_PAGE_HEADER         Header;             /* 0x00 */
    U32                             ProductSpecificInfo;/* 0x04 */
} MPI2_CONFIG_PAGE_MAN_PS,
  MPI2_POINTER PTR_MPI2_CONFIG_PAGE_MAN_PS,
  Mpi2ManufacturingPagePS_t, MPI2_POINTER pMpi2ManufacturingPagePS_t;

#define MPI2_MANUFACTURING8_PAGEVERSION                 (0x00)
#define MPI2_MANUFACTURING9_PAGEVERSION                 (0x00)
#define MPI2_MANUFACTURING10_PAGEVERSION                (0x00)
#define MPI2_MANUFACTURING11_PAGEVERSION                (0x00)
#define MPI2_MANUFACTURING12_PAGEVERSION                (0x00)
#define MPI2_MANUFACTURING13_PAGEVERSION                (0x00)
#define MPI2_MANUFACTURING14_PAGEVERSION                (0x00)
#define MPI2_MANUFACTURING15_PAGEVERSION                (0x00)
#define MPI2_MANUFACTURING16_PAGEVERSION                (0x00)
#define MPI2_MANUFACTURING17_PAGEVERSION                (0x00)
#define MPI2_MANUFACTURING18_PAGEVERSION                (0x00)
#define MPI2_MANUFACTURING19_PAGEVERSION                (0x00)
#define MPI2_MANUFACTURING20_PAGEVERSION                (0x00)
#define MPI2_MANUFACTURING21_PAGEVERSION                (0x00)
#define MPI2_MANUFACTURING22_PAGEVERSION                (0x00)
#define MPI2_MANUFACTURING23_PAGEVERSION                (0x00)
#define MPI2_MANUFACTURING24_PAGEVERSION                (0x00)
#define MPI2_MANUFACTURING25_PAGEVERSION                (0x00)
#define MPI2_MANUFACTURING26_PAGEVERSION                (0x00)
#define MPI2_MANUFACTURING27_PAGEVERSION                (0x00)
#define MPI2_MANUFACTURING28_PAGEVERSION                (0x00)
#define MPI2_MANUFACTURING29_PAGEVERSION                (0x00)
#define MPI2_MANUFACTURING30_PAGEVERSION                (0x00)
#define MPI2_MANUFACTURING31_PAGEVERSION                (0x00)


/*****************************************************************************
*   IO Unit Config Pages
*****************************************************************************/

/* IO Unit Page 0 */

typedef struct _MPI2_CONFIG_PAGE_IO_UNIT_0
{
    MPI2_CONFIG_PAGE_HEADER Header;                     /* 0x00 */
    U64                     UniqueValue;                /* 0x04 */
    MPI2_VERSION_UNION      NvdataVersionDefault;       /* 0x08 */
    MPI2_VERSION_UNION      NvdataVersionPersistent;    /* 0x0A */
```

```
} MPI2_CONFIG_PAGE_IO_UNIT_0, MPI2_POINTER PTR_MPI2_CONFIG_PAGE_IO_UNIT_0,
  Mpi2IOUnitPage0_t, MPI2_POINTER pMpi2IOUnitPage0_t;

#define MPI2_IOUNITPAGE0_PAGEVERSION                    (0x02)


/* IO Unit Page 1 */

typedef struct _MPI2_CONFIG_PAGE_IO_UNIT_1
{
    MPI2_CONFIG_PAGE_HEADER Header;                     /* 0x00 */
    U32                     Flags;                      /* 0x04 */
} MPI2_CONFIG_PAGE_IO_UNIT_1, MPI2_POINTER PTR_MPI2_CONFIG_PAGE_IO_UNIT_1,
  Mpi2IOUnitPage1_t, MPI2_POINTER pMpi2IOUnitPage1_t;

#define MPI2_IOUNITPAGE1_PAGEVERSION                    (0x04)

/* IO Unit Page 1 Flags defines */
#define MPI25_IOUNITPAGE1_NEW_DEVICE_FAST_PATH_DISABLE  (0x00002000)
#define MPI25_IOUNITPAGE1_DISABLE_FAST_PATH             (0x00001000)
#define MPI2_IOUNITPAGE1_ENABLE_HOST_BASED_DISCOVERY    (0x00000800)
#define MPI2_IOUNITPAGE1_MASK_SATA_WRITE_CACHE          (0x00000600)
#define MPI2_IOUNITPAGE1_SATA_WRITE_CACHE_SHIFT         (9)
#define MPI2_IOUNITPAGE1_ENABLE_SATA_WRITE_CACHE        (0x00000000)
#define MPI2_IOUNITPAGE1_DISABLE_SATA_WRITE_CACHE       (0x00000200)
#define MPI2_IOUNITPAGE1_UNCHANGED_SATA_WRITE_CACHE     (0x00000400)
#define MPI2_IOUNITPAGE1_NATIVE_COMMAND_Q_DISABLE       (0x00000100)
#define MPI2_IOUNITPAGE1_DISABLE_IR                     (0x00000040)
#define MPI2_IOUNITPAGE1_DISABLE_TASK_SET_FULL_HANDLING (0x00000020)
#define MPI2_IOUNITPAGE1_IR_USE_STATIC_VOLUME_ID        (0x00000004)
#define MPI2_IOUNITPAGE1_MULTI_PATHING                  (0x00000002)
#define MPI2_IOUNITPAGE1_SINGLE_PATHING                 (0x00000000)


/* IO Unit Page 3 */

/*
 * Host code (drivers, BIOS, utilities, etc.) should leave this define set to
 * one and check Header.PageLength at runtime.
 */
#ifndef MPI2_IO_UNIT_PAGE_3_GPIO_VAL_MAX
#define MPI2_IO_UNIT_PAGE_3_GPIO_VAL_MAX    (1)
#endif

typedef struct _MPI2_CONFIG_PAGE_IO_UNIT_3
{
    MPI2_CONFIG_PAGE_HEADER Header;                             /* 0x00 */
    U8                      GPIOCount;                          /* 0x04 */
    U8                      Reserved1;                          /* 0x05 */
    U16                     Reserved2;                          /* 0x06 */
    U16                     GPIOVal[MPI2_IO_UNIT_PAGE_3_GPIO_VAL_MAX];/* 0x08 */
} MPI2_CONFIG_PAGE_IO_UNIT_3, MPI2_POINTER PTR_MPI2_CONFIG_PAGE_IO_UNIT_3,
  Mpi2IOUnitPage3_t, MPI2_POINTER pMpi2IOUnitPage3_t;

#define MPI2_IOUNITPAGE3_PAGEVERSION                    (0x01)

/* defines for IO Unit Page 3 GPIOVal field */
#define MPI2_IOUNITPAGE3_GPIO_FUNCTION_MASK             (0xFFFC)
#define MPI2_IOUNITPAGE3_GPIO_FUNCTION_SHIFT            (2)
#define MPI2_IOUNITPAGE3_GPIO_SETTING_OFF               (0x0000)
#define MPI2_IOUNITPAGE3_GPIO_SETTING_ON                (0x0001)


/* IO Unit Page 5 */

/*
```



```
 * Upper layer code (drivers, utilities, etc.) should leave this define set to
 * one and check Header.PageLength or NumDmaEngines at runtime.
 */
#ifndef MPI2_IOUNITPAGE5_DMAENGINE_ENTRIES
#define MPI2_IOUNITPAGE5_DMAENGINE_ENTRIES      (1)
#endif

typedef struct _MPI2_CONFIG_PAGE_IO_UNIT_5
{
    MPI2_CONFIG_PAGE_HEADER Header;                                     /* 0x00
    U64                     RaidAcceleratorBufferBaseAddress;           /* 0x04
    U64                     RaidAcceleratorBufferSize;                  /* 0x0C
    U64                     RaidAcceleratorControlBaseAddress;          /* 0x14
    U8                      RAControlSize;                              /* 0x1C
    U8                      NumDmaEngines;                              /* 0x1D
    U8                      RAMinControlSize;                           /* 0x1E
    U8                      RAMaxControlSize;                           /* 0x1F
    U32                     Reserved1;                                  /* 0x20
    U32                     Reserved2;                                  /* 0x24
    U32                     Reserved3;                                  /* 0x28
    U32                     DmaEngineCapabilities[MPI2_IOUNITPAGE5_DMAENGINE_ENT
} MPI2_CONFIG_PAGE_IO_UNIT_5, MPI2_POINTER PTR_MPI2_CONFIG_PAGE_IO_UNIT_5,
  Mpi2IOUnitPage5_t, MPI2_POINTER pMpi2IOUnitPage5_t;

#define MPI2_IOUNITPAGE5_PAGEVERSION                    (0x00)

/* defines for IO Unit Page 5 DmaEngineCapabilities field */
#define MPI2_IOUNITPAGE5_DMA_CAP_MASK_MAX_REQUESTS      (0xFF00)
#define MPI2_IOUNITPAGE5_DMA_CAP_SHIFT_MAX_REQUESTS     (16)

#define MPI2_IOUNITPAGE5_DMA_CAP_EEDP                   (0x0008)
#define MPI2_IOUNITPAGE5_DMA_CAP_PARITY_GENERATION      (0x0004)
#define MPI2_IOUNITPAGE5_DMA_CAP_HASHING                (0x0002)
#define MPI2_IOUNITPAGE5_DMA_CAP_ENCRYPTION             (0x0001)


/* IO Unit Page 6 */

typedef struct _MPI2_CONFIG_PAGE_IO_UNIT_6
{
    MPI2_CONFIG_PAGE_HEADER Header;                             /* 0x00 */
    U16                     Flags;                              /* 0x04 */
    U8                      RAHostControlSize;                  /* 0x06 */
    U8                      Reserved0;                          /* 0x07 */
    U64                     RaidAcceleratorHostControlBaseAddress;  /* 0x08 */
    U32                     Reserved1;                          /* 0x10 */
    U32                     Reserved2;                          /* 0x14 */
    U32                     Reserved3;                          /* 0x18 */
} MPI2_CONFIG_PAGE_IO_UNIT_6, MPI2_POINTER PTR_MPI2_CONFIG_PAGE_IO_UNIT_6,
  Mpi2IOUnitPage6_t, MPI2_POINTER pMpi2IOUnitPage6_t;

#define MPI2_IOUNITPAGE6_PAGEVERSION                    (0x00)

/* defines for IO Unit Page 6 Flags field */
#define MPI2_IOUNITPAGE6_FLAGS_ENABLE_RAID_ACCELERATOR  (0x0001)


/* IO Unit Page 7 */

typedef struct _MPI2_CONFIG_PAGE_IO_UNIT_7
{
    MPI2_CONFIG_PAGE_HEADER Header;                             /* 0x00 */
    U16                     Reserved1;                          /* 0x04 */
    U8                      PCIeWidth;                          /* 0x06 */
    U8                      PCIeSpeed;                          /* 0x07 */
    U32                     ProcessorState;                     /* 0x08 */
```

```
    U32                     PowerManagementCapabilities;            /* 0x0C */
    U16                     IOCTemperature;                         /* 0x10 */
    U8                      IOCTemperatureUnits;                    /* 0x12 */
    U8                      IOCSpeed;                               /* 0x13 */
    U16                     BoardTemperature;                       /* 0x14 */
    U8                      BoardTemperatureUnits;                  /* 0x16 */
    U8                      Reserved3;                              /* 0x17 */
} 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)

/* 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)

/* 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)


/*****************************************************************************
*   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)
#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)
#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)
#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)
#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)


/* 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)
#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)


/* 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)



/*****************************************************************************
*   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)
#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)

/* 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 */
    U16         InactivityTimerExponent;    /* 0x02 */
    U8          SATAPartialTimeout;         /* 0x04 */
    U8          Reserved2;                  /* 0x05 */
    U8          SATASlumberTimeout;         /* 0x06 */
    U8          Reserved3;                  /* 0x07 */
    U8          SASPartialTimeout;          /* 0x08 */
    U8          Reserved4;                  /* 0x09 */
    U8          SASSlumberTimeout;          /* 0x0A */
    U8          Reserved5;                  /* 0x0B */
} MPI2_SAS_IO_UNIT5_PHY_PM_SETTINGS,
  MPI2_POINTER PTR_MPI2_SAS_IO_UNIT5_PHY_PM_SETTINGS,
  Mpi2SasIOUnit5PhyPmSettings_t, MPI2_POINTER pMpi2SasIOUnit5PhyPmSettings_t;

/* defines for ControlFlags field */
#define MPI2_SASIOUNIT5_CONTROL_SAS_SLUMBER_ENABLE      (0x08)
#define MPI2_SASIOUNIT5_CONTROL_SAS_PARTIAL_ENABLE      (0x04)
#define MPI2_SASIOUNIT5_CONTROL_SATA_SLUMBER_ENABLE     (0x02)
#define MPI2_SASIOUNIT5_CONTROL_SATA_PARTIAL_ENABLE     (0x01)

/* defines for PortWidthModeGroup field */
```

```
#define MPI2_SASIOUNIT5_PWMG_DISABLE                    (0xFF)

/* defines for InactivityTimerExponent field */
#define MPI2_SASIOUNIT5_ITE_MASK_SAS_SLUMBER            (0x7000)
#define MPI2_SASIOUNIT5_ITE_SHIFT_SAS_SLUMBER           (12)
#define MPI2_SASIOUNIT5_ITE_MASK_SAS_PARTIAL            (0x0700)
#define MPI2_SASIOUNIT5_ITE_SHIFT_SAS_PARTIAL           (8)
#define MPI2_SASIOUNIT5_ITE_MASK_SATA_SLUMBER           (0x0070)
#define MPI2_SASIOUNIT5_ITE_SHIFT_SATA_SLUMBER          (4)
#define MPI2_SASIOUNIT5_ITE_MASK_SATA_PARTIAL           (0x0007)
#define MPI2_SASIOUNIT5_ITE_SHIFT_SATA_PARTIAL          (0)

#define MPI2_SASIOUNIT5_ITE_TEN_SECONDS                 (7)
#define MPI2_SASIOUNIT5_ITE_ONE_SECOND                  (6)
#define MPI2_SASIOUNIT5_ITE_HUNDRED_MILLISECONDS        (5)
#define MPI2_SASIOUNIT5_ITE_TEN_MILLISECONDS            (4)
#define MPI2_SASIOUNIT5_ITE_ONE_MILLISECOND             (3)
#define MPI2_SASIOUNIT5_ITE_HUNDRED_MICROSECONDS        (2)
#define MPI2_SASIOUNIT5_ITE_TEN_MICROSECONDS            (1)
#define MPI2_SASIOUNIT5_ITE_ONE_MICROSECOND             (0)

/*
 * Host code (drivers, BIOS, utilities, etc.) should leave this define set to
 * one and check Header.ExtPageLength or NumPhys at runtime.
 */
#ifndef MPI2_SAS_IOUNIT5_PHY_MAX
#define MPI2_SAS_IOUNIT5_PHY_MAX        (1)
#endif

typedef struct _MPI2_CONFIG_PAGE_SASIOUNIT_5
{
    MPI2_CONFIG_EXTENDED_PAGE_HEADER    Header;                             /* 0
    U8                                  NumPhys;                            /* 0
    U8                                  Reserved1;                          /* 0
    U16                                 Reserved2;                          /* 0
    U32                                 Reserved3;                          /* 0
    MPI2_SAS_IO_UNIT5_PHY_PM_SETTINGS   SASPhyPowerManagementSettings[MPI2_SAS_I
} MPI2_CONFIG_PAGE_SASIOUNIT_5,
  MPI2_POINTER PTR_MPI2_CONFIG_PAGE_SASIOUNIT_5,
  Mpi2SasIOUnitPage5_t, MPI2_POINTER pMpi2SasIOUnitPage5_t;

#define MPI2_SASIOUNITPAGE5_PAGEVERSION     (0x01)


/* SAS IO Unit Page 6 */

typedef struct _MPI2_SAS_IO_UNIT6_PORT_WIDTH_MOD_GROUP_STATUS
{
    U8          CurrentStatus;              /* 0x00 */
    U8          CurrentModulation;          /* 0x01 */
    U8          CurrentUtilization;         /* 0x02 */
    U8          Reserved1;                  /* 0x03 */
    U32         Reserved2;                  /* 0x04 */
} MPI2_SAS_IO_UNIT6_PORT_WIDTH_MOD_GROUP_STATUS,
  MPI2_POINTER PTR_MPI2_SAS_IO_UNIT6_PORT_WIDTH_MOD_GROUP_STATUS,
  Mpi2SasIOUnit6PortWidthModGroupStatus_t,
  MPI2_POINTER pMpi2SasIOUnit6PortWidthModGroupStatus_t;

/* defines for CurrentStatus field */
#define MPI2_SASIOUNIT6_STATUS_UNAVAILABLE                      (0x00)
#define MPI2_SASIOUNIT6_STATUS_UNCONFIGURED                     (0x01)
#define MPI2_SASIOUNIT6_STATUS_INVALID_CONFIG                   (0x02)
#define MPI2_SASIOUNIT6_STATUS_LINK_DOWN                        (0x03)
#define MPI2_SASIOUNIT6_STATUS_OBSERVATION_ONLY                 (0x04)
#define MPI2_SASIOUNIT6_STATUS_INACTIVE                         (0x05)
#define MPI2_SASIOUNIT6_STATUS_ACTIVE_IOUNIT                    (0x06)
```



```
#define MPI2_SASIOUNIT6_STATUS_ACTIVE_HOST                      (0x07)

/* defines for CurrentModulation field */
#define MPI2_SASIOUNIT6_MODULATION_25_PERCENT                   (0x00)
#define MPI2_SASIOUNIT6_MODULATION_50_PERCENT                   (0x01)
#define MPI2_SASIOUNIT6_MODULATION_75_PERCENT                   (0x02)
#define MPI2_SASIOUNIT6_MODULATION_100_PERCENT                  (0x03)

/*
 * Host code (drivers, BIOS, utilities, etc.) should leave this define set to
 * one and check the value returned for NumGroups at runtime.
 */
#ifndef MPI2_SAS_IOUNIT6_GROUP_MAX
#define MPI2_SAS_IOUNIT6_GROUP_MAX      (1)
#endif

typedef struct _MPI2_CONFIG_PAGE_SASIOUNIT_6
{
    MPI2_CONFIG_EXTENDED_PAGE_HEADER    Header;                     /* 0x00 */
    U32                                 Reserved1;                  /* 0x08 */
    U32                                 Reserved2;                  /* 0x0C */
    U8                                  NumGroups;                  /* 0x10 */
    U8                                  Reserved3;                  /* 0x11 */
    U16                                 Reserved4;                  /* 0x12 */
    MPI2_SAS_IO_UNIT6_PORT_WIDTH_MOD_GROUP_STATUS
        PortWidthModulationGroupStatus[MPI2_SAS_IOUNIT6_GROUP_MAX]; /* 0x14 */
} MPI2_CONFIG_PAGE_SASIOUNIT_6,
  MPI2_POINTER PTR_MPI2_CONFIG_PAGE_SASIOUNIT_6,
  Mpi2SasIOUnitPage6_t, MPI2_POINTER pMpi2SasIOUnitPage6_t;

#define MPI2_SASIOUNITPAGE6_PAGEVERSION     (0x00)


/* SAS IO Unit Page 7 */

typedef struct _MPI2_SAS_IO_UNIT7_PORT_WIDTH_MOD_GROUP_SETTINGS
{
    U8          Flags;                      /* 0x00 */
    U8          Reserved1;                  /* 0x01 */
    U16         Reserved2;                  /* 0x02 */
    U8          Threshold75Pct;             /* 0x04 */
    U8          Threshold50Pct;             /* 0x05 */
    U8          Threshold25Pct;             /* 0x06 */
    U8          Reserved3;                  /* 0x07 */
} MPI2_SAS_IO_UNIT7_PORT_WIDTH_MOD_GROUP_SETTINGS,
  MPI2_POINTER PTR_MPI2_SAS_IO_UNIT7_PORT_WIDTH_MOD_GROUP_SETTINGS,
  Mpi2SasIOUnit7PortWidthModGroupSettings_t,
  MPI2_POINTER pMpi2SasIOUnit7PortWidthModGroupSettings_t;

/* defines for Flags field */
#define MPI2_SASIOUNIT7_FLAGS_ENABLE_PORT_WIDTH_MODULATION  (0x01)


/*
 * Host code (drivers, BIOS, utilities, etc.) should leave this define set to
 * one and check the value returned for NumGroups at runtime.
 */
#ifndef MPI2_SAS_IOUNIT7_GROUP_MAX
#define MPI2_SAS_IOUNIT7_GROUP_MAX      (1)
#endif

typedef struct _MPI2_CONFIG_PAGE_SASIOUNIT_7
{
    MPI2_CONFIG_EXTENDED_PAGE_HEADER            Header;             /* 0x00 */
    U8                                          SamplingInterval;   /* 0x08 */
    U8                                          WindowLength;       /* 0x09 */
```

```
    U16                                         Reserved1;          /* 0x0A */
    U32                                         Reserved2;          /* 0x0C */
    U32                                         Reserved3;          /* 0x10 */
    U8                                          NumGroups;          /* 0x14 */
    U8                                          Reserved4;          /* 0x15 */
    U16                                         Reserved5;          /* 0x16 */
    MPI2_SAS_IO_UNIT7_PORT_WIDTH_MOD_GROUP_SETTINGS
        PortWidthModulationGroupSettings[MPI2_SAS_IOUNIT7_GROUP_MAX]; /* 0x18 */
} MPI2_CONFIG_PAGE_SASIOUNIT_7,
  MPI2_POINTER PTR_MPI2_CONFIG_PAGE_SASIOUNIT_7,
  Mpi2SasIOUnitPage7_t, MPI2_POINTER pMpi2SasIOUnitPage7_t;

#define MPI2_SASIOUNITPAGE7_PAGEVERSION     (0x00)


/* SAS IO Unit Page 8 */

typedef struct _MPI2_CONFIG_PAGE_SASIOUNIT_8
{
    MPI2_CONFIG_EXTENDED_PAGE_HEADER    Header;                         /* 0
    U32                                 Reserved1;                      /* 0
    U32                                 PowerManagementCapabilities;    /* 0
    U32                                 Reserved2;                      /* 0
} MPI2_CONFIG_PAGE_SASIOUNIT_8,
  MPI2_POINTER PTR_MPI2_CONFIG_PAGE_SASIOUNIT_8,
  Mpi2SasIOUnitPage8_t, MPI2_POINTER pMpi2SasIOUnitPage8_t;

#define MPI2_SASIOUNITPAGE8_PAGEVERSION     (0x00)

/* defines for PowerManagementCapabilities field */
#define MPI2_SASIOUNIT8_PM_HOST_PORT_WIDTH_MOD          (0x000001000)
#define MPI2_SASIOUNIT8_PM_HOST_SAS_SLUMBER_MODE        (0x000000800)
#define MPI2_SASIOUNIT8_PM_HOST_SAS_PARTIAL_MODE        (0x000000400)
#define MPI2_SASIOUNIT8_PM_HOST_SATA_SLUMBER_MODE       (0x000000200)
#define MPI2_SASIOUNIT8_PM_HOST_SATA_PARTIAL_MODE       (0x000000100)
#define MPI2_SASIOUNIT8_PM_IOUNIT_PORT_WIDTH_MOD        (0x000000010)
#define MPI2_SASIOUNIT8_PM_IOUNIT_SAS_SLUMBER_MODE      (0x000000008)
#define MPI2_SASIOUNIT8_PM_IOUNIT_SAS_PARTIAL_MODE      (0x000000004)
#define MPI2_SASIOUNIT8_PM_IOUNIT_SATA_SLUMBER_MODE     (0x000000002)
#define MPI2_SASIOUNIT8_PM_IOUNIT_SATA_PARTIAL_MODE     (0x000000001)




/*****************************************************************************
*   SAS Expander Config Pages
*****************************************************************************/

/* SAS Expander Page 0 */

typedef struct _MPI2_CONFIG_PAGE_EXPANDER_0
{
    MPI2_CONFIG_EXTENDED_PAGE_HEADER    Header;                     /* 0x00 */
    U8                                  PhysicalPort;               /* 0x08 */
    U8                                  ReportGenLength;            /* 0x09 */
    U16                                 EnclosureHandle;            /* 0x0A */
    U64                                 SASAddress;                 /* 0x0C */
    U32                                 DiscoveryStatus;            /* 0x14 */
    U16                                 DevHandle;                  /* 0x18 */
    U16                                 ParentDevHandle;            /* 0x1A */
    U16                                 ExpanderChangeCount;        /* 0x1C */
    U16                                 ExpanderRouteIndexes;       /* 0x1E */
    U8                                  NumPhys;                    /* 0x20 */
    U8                                  SASLevel;                   /* 0x21 */
    U16                                 Flags;                      /* 0x22 */
    U16                                 STPBusInactivityTimeLimit;  /* 0x24 */
```



```
    U16                                 STPMaxConnectTimeLimit;     /* 0x26 */
    U16                                 STP_SMP_NexusLossTime;      /* 0x28 */
    U16                                 MaxNumRoutedSasAddresses;   /* 0x2A */
    U64                                 ActiveZoneManagerSASAddress;/* 0x2C */
    U16                                 ZoneLockInactivityLimit;    /* 0x34 */
    U16                                 Reserved1;                  /* 0x36 */
    U8                                  TimeToReducedFunc;          /* 0x38 */
    U8                                  InitialTimeToReducedFunc;   /* 0x39 */
    U8                                  MaxReducedFuncTime;         /* 0x3A */
    U8                                  Reserved2;                  /* 0x3B */
} MPI2_CONFIG_PAGE_EXPANDER_0, MPI2_POINTER PTR_MPI2_CONFIG_PAGE_EXPANDER_0,
  Mpi2ExpanderPage0_t, MPI2_POINTER pMpi2ExpanderPage0_t;

#define MPI2_SASEXPANDER0_PAGEVERSION       (0x06)

/* values for SAS Expander Page 0 DiscoveryStatus field */
#define MPI2_SAS_EXPANDER0_DS_MAX_ENCLOSURES_EXCEED         (0x80000000)
#define MPI2_SAS_EXPANDER0_DS_MAX_EXPANDERS_EXCEED          (0x40000000)
#define MPI2_SAS_EXPANDER0_DS_MAX_DEVICES_EXCEED            (0x20000000)
#define MPI2_SAS_EXPANDER0_DS_MAX_TOPO_PHYS_EXCEED          (0x10000000)
#define MPI2_SAS_EXPANDER0_DS_DOWNSTREAM_INITIATOR          (0x08000000)
#define MPI2_SAS_EXPANDER0_DS_MULTI_SUBTRACTIVE_SUBTRACTIVE (0x00008000)
#define MPI2_SAS_EXPANDER0_DS_EXP_MULTI_SUBTRACTIVE         (0x00004000)
#define MPI2_SAS_EXPANDER0_DS_MULTI_PORT_DOMAIN             (0x00002000)
#define MPI2_SAS_EXPANDER0_DS_TABLE_TO_SUBTRACTIVE_LINK     (0x00001000)
#define MPI2_SAS_EXPANDER0_DS_UNSUPPORTED_DEVICE            (0x00000800)
#define MPI2_SAS_EXPANDER0_DS_TABLE_LINK                    (0x00000400)
#define MPI2_SAS_EXPANDER0_DS_SUBTRACTIVE_LINK              (0x00000200)
#define MPI2_SAS_EXPANDER0_DS_SMP_CRC_ERROR                 (0x00000100)
#define MPI2_SAS_EXPANDER0_DS_SMP_FUNCTION_FAILED           (0x00000080)
#define MPI2_SAS_EXPANDER0_DS_INDEX_NOT_EXIST               (0x00000040)
#define MPI2_SAS_EXPANDER0_DS_OUT_ROUTE_ENTRIES             (0x00000020)
#define MPI2_SAS_EXPANDER0_DS_SMP_TIMEOUT                   (0x00000010)
#define MPI2_SAS_EXPANDER0_DS_MULTIPLE_PORTS                (0x00000004)
#define MPI2_SAS_EXPANDER0_DS_UNADDRESSABLE_DEVICE          (0x00000002)
#define MPI2_SAS_EXPANDER0_DS_LOOP_DETECTED                 (0x00000001)

/* values for SAS Expander Page 0 Flags field */
#define MPI2_SAS_EXPANDER0_FLAGS_REDUCED_FUNCTIONALITY      (0x2000)
#define MPI2_SAS_EXPANDER0_FLAGS_ZONE_LOCKED                (0x1000)
#define MPI2_SAS_EXPANDER0_FLAGS_SUPPORTED_PHYSICAL_PRES    (0x0800)
#define MPI2_SAS_EXPANDER0_FLAGS_ASSERTED_PHYSICAL_PRES     (0x0400)
#define MPI2_SAS_EXPANDER0_FLAGS_ZONING_SUPPORT             (0x0200)
#define MPI2_SAS_EXPANDER0_FLAGS_ENABLED_ZONING             (0x0100)
#define MPI2_SAS_EXPANDER0_FLAGS_TABLE_TO_TABLE_SUPPORT     (0x0080)
#define MPI2_SAS_EXPANDER0_FLAGS_CONNECTOR_END_DEVICE       (0x0010)
#define MPI2_SAS_EXPANDER0_FLAGS_OTHERS_CONFIG              (0x0004)
#define MPI2_SAS_EXPANDER0_FLAGS_CONFIG_IN_PROGRESS         (0x0002)
#define MPI2_SAS_EXPANDER0_FLAGS_ROUTE_TABLE_CONFIG         (0x0001)


/* SAS Expander Page 1 */

typedef struct _MPI2_CONFIG_PAGE_EXPANDER_1
{
    MPI2_CONFIG_EXTENDED_PAGE_HEADER    Header;                     /* 0x00 */
    U8                                  PhysicalPort;               /* 0x08 */
    U8                                  Reserved1;                  /* 0x09 */
    U16                                 Reserved2;                  /* 0x0A */
    U8                                  NumPhys;                    /* 0x0C */
    U8                                  Phy;                        /* 0x0D */
    U16                                 NumTableEntriesProgrammed;  /* 0x0E */
    U8                                  ProgrammedLinkRate;         /* 0x10 */
    U8                                  HwLinkRate;                 /* 0x11 */
    U16                                 AttachedDevHandle;          /* 0x12 */
    U32                                 PhyInfo;                    /* 0x14 */
```

```
    U32                                 AttachedDeviceInfo;         /* 0x18 */
    U16                                 ExpanderDevHandle;          /* 0x1C */
    U8                                  ChangeCount;                /* 0x1E */
    U8                                  NegotiatedLinkRate;         /* 0x1F */
    U8                                  PhyIdentifier;              /* 0x20 */
    U8                                  AttachedPhyIdentifier;      /* 0x21 */
    U8                                  Reserved3;                  /* 0x22 */
    U8                                  DiscoveryInfo;              /* 0x23 */
    U32                                 AttachedPhyInfo;            /* 0x24 */
    U8                                  ZoneGroup;                  /* 0x28 */
    U8                                  SelfConfigStatus;           /* 0x29 */
    U16                                 Reserved4;                  /* 0x2A */
} MPI2_CONFIG_PAGE_EXPANDER_1, MPI2_POINTER PTR_MPI2_CONFIG_PAGE_EXPANDER_1,
  Mpi2ExpanderPage1_t, MPI2_POINTER pMpi2ExpanderPage1_t;

#define MPI2_SASEXPANDER1_PAGEVERSION       (0x02)

/* use MPI2_SAS_PRATE_ defines for the ProgrammedLinkRate field */

/* use MPI2_SAS_HWRATE_ defines for the HwLinkRate field */

/* use MPI2_SAS_PHYINFO_ for the PhyInfo field */

/* see mpi2_sas.h for the MPI2_SAS_DEVICE_INFO_ defines used for the AttachedDev

/* use MPI2_SAS_NEG_LINK_RATE_ defines for the NegotiatedLinkRate field */

/* use MPI2_SAS_APHYINFO_ defines for AttachedPhyInfo field */

/* values for SAS Expander Page 1 DiscoveryInfo field */
#define MPI2_SAS_EXPANDER1_DISCINFO_BAD_PHY_DISABLED    (0x04)
#define MPI2_SAS_EXPANDER1_DISCINFO_LINK_STATUS_CHANGE  (0x02)
#define MPI2_SAS_EXPANDER1_DISCINFO_NO_ROUTING_ENTRIES  (0x01)


/*****************************************************************************
*   SAS Device Config Pages
*****************************************************************************/

/* SAS Device Page 0 */

typedef struct _MPI2_CONFIG_PAGE_SAS_DEV_0
{
    MPI2_CONFIG_EXTENDED_PAGE_HEADER    Header;                 /* 0x00 */
    U16                                 Slot;                   /* 0x08 */
    U16                                 EnclosureHandle;        /* 0x0A */
    U64                                 SASAddress;             /* 0x0C */
    U16                                 ParentDevHandle;        /* 0x14 */
    U8                                  PhyNum;                 /* 0x16 */
    U8                                  AccessStatus;           /* 0x17 */
    U16                                 DevHandle;              /* 0x18 */
    U8                                  AttachedPhyIdentifier;  /* 0x1A */
    U8                                  ZoneGroup;              /* 0x1B */
    U32                                 DeviceInfo;             /* 0x1C */
    U16                                 Flags;                  /* 0x20 */
    U8                                  PhysicalPort;           /* 0x22 */
    U8                                  MaxPortConnections;     /* 0x23 */
    U64                                 DeviceName;             /* 0x24 */
    U8                                  PortGroups;             /* 0x2C */
    U8                                  DmaGroup;               /* 0x2D */
    U8                                  ControlGroup;           /* 0x2E */
    U8                                  Reserved1;              /* 0x2F */
    U32                                 Reserved2;              /* 0x30 */
    U32                                 Reserved3;              /* 0x34 */
} MPI2_CONFIG_PAGE_SAS_DEV_0, MPI2_POINTER PTR_MPI2_CONFIG_PAGE_SAS_DEV_0,
  Mpi2SasDevicePage0_t, MPI2_POINTER pMpi2SasDevicePage0_t;
```



```
#define MPI2_SASDEVICE0_PAGEVERSION         (0x08)

/* values for SAS Device Page 0 AccessStatus field */
#define MPI2_SAS_DEVICE0_ASTATUS_NO_ERRORS                  (0x00)
#define MPI2_SAS_DEVICE0_ASTATUS_SATA_INIT_FAILED           (0x01)
#define MPI2_SAS_DEVICE0_ASTATUS_SATA_CAPABILITY_FAILED     (0x02)
#define MPI2_SAS_DEVICE0_ASTATUS_SATA_AFFILIATION_CONFLICT  (0x03)
#define MPI2_SAS_DEVICE0_ASTATUS_SATA_NEEDS_INITIALIZATION  (0x04)
#define MPI2_SAS_DEVICE0_ASTATUS_ROUTE_NOT_ADDRESSABLE      (0x05)
#define MPI2_SAS_DEVICE0_ASTATUS_SMP_ERROR_NOT_ADDRESSABLE  (0x06)
#define MPI2_SAS_DEVICE0_ASTATUS_DEVICE_BLOCKED             (0x07)
/* specific values for SATA Init failures */
#define MPI2_SAS_DEVICE0_ASTATUS_SIF_UNKNOWN                (0x10)
#define MPI2_SAS_DEVICE0_ASTATUS_SIF_AFFILIATION_CONFLICT   (0x11)
#define MPI2_SAS_DEVICE0_ASTATUS_SIF_DIAG                   (0x12)
#define MPI2_SAS_DEVICE0_ASTATUS_SIF_IDENTIFICATION         (0x13)
#define MPI2_SAS_DEVICE0_ASTATUS_SIF_CHECK_POWER            (0x14)
#define MPI2_SAS_DEVICE0_ASTATUS_SIF_PIO_SN                 (0x15)
#define MPI2_SAS_DEVICE0_ASTATUS_SIF_MDMA_SN                (0x16)
#define MPI2_SAS_DEVICE0_ASTATUS_SIF_UDMA_SN                (0x17)
#define MPI2_SAS_DEVICE0_ASTATUS_SIF_ZONING_VIOLATION       (0x18)
#define MPI2_SAS_DEVICE0_ASTATUS_SIF_NOT_ADDRESSABLE        (0x19)
#define MPI2_SAS_DEVICE0_ASTATUS_SIF_MAX                    (0x1F)

/* see mpi2_sas.h for values for SAS Device Page 0 DeviceInfo values */

/* values for SAS Device Page 0 Flags field */
#define MPI25_SAS_DEVICE0_FLAGS_ENABLED_FAST_PATH           (0x4000)
#define MPI25_SAS_DEVICE0_FLAGS_FAST_PATH_CAPABLE           (0x2000)
#define MPI2_SAS_DEVICE0_FLAGS_SLUMBER_PM_CAPABLE           (0x1000)
#define MPI2_SAS_DEVICE0_FLAGS_PARTIAL_PM_CAPABLE           (0x0800)
#define MPI2_SAS_DEVICE0_FLAGS_SATA_ASYNCHRONOUS_NOTIFY     (0x0400)
#define MPI2_SAS_DEVICE0_FLAGS_SATA_SW_PRESERVE             (0x0200)
#define MPI2_SAS_DEVICE0_FLAGS_UNSUPPORTED_DEVICE           (0x0100)
#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)
```



```
/*****************************************************************************
*   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

#endif /* ! codereview */
```

```
**********************************************************
   29921 Thu Jun 12 17:28:23 2014
new/usr/src/uts/common/sys/scsi/adapters/mpt_sas3/mpi/mpi2_init.h
4546 mpt_sas needs enhancing to support LSI MPI2.5
**********************************************************
/*
 * 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.
 *
 * 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.
 *
 *  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.
 *  ----------------------------------------------------------------------------
 */

#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;

typedef union
{
    U8                      CDB32[32];
    MPI2_SCSI_IO_CDB_EEDP32 EEDP32;
    MPI2_SGE_SIMPLE_UNION   SGE;
} MPI2_SCSI_IO_CDB_UNION, MPI2_POINTER PTR_MPI2_SCSI_IO_CDB_UNION,
  Mpi2ScsiIoCdb_t, MPI2_POINTER pMpi2ScsiIoCdb_t;

/* MPI v2.0 SCSI IO Request Message */
typedef struct _MPI2_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 */
    U16                     SGLFlags;                       /* 0x10 */
    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 */
    U32                     EEDPBlockSize;                  /* 0x28 */
    U32                     SecondaryReferenceTag;          /* 0x2C */
    U16                     SecondaryApplicationTag;        /* 0x30 */
    U16                     ApplicationTagTranslationMask;  /* 0x32 */
    U8                      LUN[8];                         /* 0x34 */
    U32                     Control;                        /* 0x3C */
    MPI2_SCSI_IO_CDB_UNION  CDB;                            /* 0x40 */
    MPI2_SGE_IO_UNION       SGL;                            /* 0x60 */
} MPI2_SCSI_IO_REQUEST, MPI2_POINTER PTR_MPI2_SCSI_IO_REQUEST,
  Mpi2SCSIIORequest_t, MPI2_POINTER pMpi2SCSIIORequest_t;

/* SCSI IO MsgFlags bits */

/* MsgFlags for SenseBufferAddressSpace */
#define MPI2_SCSIIO_MSGFLAGS_MASK_SENSE_ADDR        (0x0C)
#define MPI2_SCSIIO_MSGFLAGS_SYSTEM_SENSE_ADDR      (0x00)
#define MPI2_SCSIIO_MSGFLAGS_IOCDDR_SENSE_ADDR      (0x04)
#define MPI2_SCSIIO_MSGFLAGS_IOCPLB_SENSE_ADDR      (0x08)
#define MPI2_SCSIIO_MSGFLAGS_IOCPLBNTA_SENSE_ADDR   (0x0C)

/* SCSI IO SGLFlags bits */

/* base values for Data Location Address Space */
#define MPI2_SCSIIO_SGLFLAGS_ADDR_MASK              (0x0C)
#define MPI2_SCSIIO_SGLFLAGS_SYSTEM_ADDR            (0x00)
#define MPI2_SCSIIO_SGLFLAGS_IOCDDR_ADDR            (0x04)
#define MPI2_SCSIIO_SGLFLAGS_IOCPLB_ADDR            (0x08)
#define MPI2_SCSIIO_SGLFLAGS_IOCPLBNTA_ADDR         (0x0C)

/* base values for Type */
#define MPI2_SCSIIO_SGLFLAGS_TYPE_MASK              (0x03)
#define MPI2_SCSIIO_SGLFLAGS_TYPE_MPI               (0x00)
#define MPI2_SCSIIO_SGLFLAGS_TYPE_IEEE32            (0x01)
#define MPI2_SCSIIO_SGLFLAGS_TYPE_IEEE64            (0x02)

/* shift values for each sub-field */
#define MPI2_SCSIIO_SGLFLAGS_SGL3_SHIFT             (12)
#define MPI2_SCSIIO_SGLFLAGS_SGL2_SHIFT             (8)
#define MPI2_SCSIIO_SGLFLAGS_SGL1_SHIFT             (4)
#define MPI2_SCSIIO_SGLFLAGS_SGL0_SHIFT             (0)

/* number of SGLOffset fields */
#define MPI2_SCSIIO_NUM_SGLOFFSETS                  (4)

/* SCSI IO IoFlags bits */

/* Large CDB Address Space */
#define MPI2_SCSIIO_CDB_ADDR_MASK                   (0x6000)
#define MPI2_SCSIIO_CDB_ADDR_SYSTEM                 (0x0000)
```



```
#define MPI2_SCSIIO_CDB_ADDR_IOCDDR                 (0x2000)
#define MPI2_SCSIIO_CDB_ADDR_IOCPLB                 (0x4000)
#define MPI2_SCSIIO_CDB_ADDR_IOCPLBNTA              (0x6000)

#define MPI2_SCSIIO_IOFLAGS_LARGE_CDB               (0x1000)
#define MPI2_SCSIIO_IOFLAGS_BIDIRECTIONAL           (0x0800)
#define MPI2_SCSIIO_IOFLAGS_MULTICAST               (0x0400)
#define MPI2_SCSIIO_IOFLAGS_CMD_DETERMINES_DATA_DIR (0x0200)
#define MPI2_SCSIIO_IOFLAGS_CDBLENGTH_MASK          (0x01FF)

/* SCSI IO EEDPFlags bits */

#define MPI2_SCSIIO_EEDPFLAGS_INC_PRI_REFTAG        (0x8000)
#define MPI2_SCSIIO_EEDPFLAGS_INC_SEC_REFTAG        (0x4000)
#define MPI2_SCSIIO_EEDPFLAGS_INC_PRI_APPTAG        (0x2000)
#define MPI2_SCSIIO_EEDPFLAGS_INC_SEC_APPTAG        (0x1000)

#define MPI2_SCSIIO_EEDPFLAGS_CHECK_REFTAG          (0x0400)
#define MPI2_SCSIIO_EEDPFLAGS_CHECK_APPTAG          (0x0200)
#define MPI2_SCSIIO_EEDPFLAGS_CHECK_GUARD           (0x0100)

#define MPI2_SCSIIO_EEDPFLAGS_PASSTHRU_REFTAG       (0x0008)

#define MPI2_SCSIIO_EEDPFLAGS_MASK_OP               (0x0007)
#define MPI2_SCSIIO_EEDPFLAGS_NOOP_OP               (0x0000)
#define MPI2_SCSIIO_EEDPFLAGS_CHECK_OP              (0x0001)
#define MPI2_SCSIIO_EEDPFLAGS_STRIP_OP              (0x0002)
#define MPI2_SCSIIO_EEDPFLAGS_CHECK_REMOVE_OP       (0x0003)
#define MPI2_SCSIIO_EEDPFLAGS_INSERT_OP             (0x0004)
#define MPI2_SCSIIO_EEDPFLAGS_REPLACE_OP            (0x0006)
#define MPI2_SCSIIO_EEDPFLAGS_CHECK_REGEN_OP        (0x0007)

/* SCSI IO LUN fields: use MPI2_LUN_ from mpi2.h */

/* SCSI IO Control bits */
#define MPI2_SCSIIO_CONTROL_ADDCDBLEN_MASK      (0xFC000000)
#define MPI2_SCSIIO_CONTROL_ADDCDBLEN_SHIFT     (26)

#define MPI2_SCSIIO_CONTROL_DATADIRECTION_MASK  (0x03000000)
#define MPI2_SCSIIO_CONTROL_NODATATRANSFER      (0x00000000)
#define MPI2_SCSIIO_CONTROL_WRITE               (0x01000000)
#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.
 */

/* SCSI IO Error Reply Message */
typedef struct _MPI2_SCSI_IO_REPLY
{
    U16                     DevHandle;                      /* 0x00 */
    U8                      MsgLength;                      /* 0x02 */
    U8                      Function;                       /* 0x03 */
    U16                     Reserved1;                      /* 0x04 */
    U8                      Reserved2;                      /* 0x06 */
    U8                      MsgFlags;                       /* 0x07 */
    U8                      VP_ID;                          /* 0x08 */
    U8                      VF_ID;                          /* 0x09 */
    U16                     Reserved3;                      /* 0x0A */
    U8                      SCSIStatus;                     /* 0x0C */
    U8                      SCSIState;                      /* 0x0D */
    U16                     IOCStatus;                      /* 0x0E */
    U32                     IOCLogInfo;                     /* 0x10 */
    U32                     TransferCount;                  /* 0x14 */
    U32                     SenseCount;                     /* 0x18 */
    U32                     ResponseInfo;                   /* 0x1C */
    U16                     TaskTag;                        /* 0x20 */
    U16                     Reserved4;                      /* 0x22 */
    U32                     BidirectionalTransferCount;     /* 0x24 */
    U32                     Reserved5;                      /* 0x28 */
    U32                     Reserved6;                      /* 0x2C */
} MPI2_SCSI_IO_REPLY, MPI2_POINTER PTR_MPI2_SCSI_IO_REPLY,
```

```
  Mpi2SCSIIOReply_t, MPI2_POINTER pMpi2SCSIIOReply_t;

/* SCSI IO Reply SCSIStatus values (SAM-4 status codes) */

#define MPI2_SCSI_STATUS_GOOD                   (0x00)
#define MPI2_SCSI_STATUS_CHECK_CONDITION        (0x02)
#define MPI2_SCSI_STATUS_CONDITION_MET          (0x04)
#define MPI2_SCSI_STATUS_BUSY                   (0x08)
#define MPI2_SCSI_STATUS_INTERMEDIATE           (0x10)
#define MPI2_SCSI_STATUS_INTERMEDIATE_CONDMET   (0x14)
#define MPI2_SCSI_STATUS_RESERVATION_CONFLICT   (0x18)
#define MPI2_SCSI_STATUS_COMMAND_TERMINATED     (0x22) /* obsolete */
#define MPI2_SCSI_STATUS_TASK_SET_FULL          (0x28)
#define MPI2_SCSI_STATUS_ACA_ACTIVE             (0x30)
#define MPI2_SCSI_STATUS_TASK_ABORTED           (0x40)

/* SCSI IO Reply SCSIState flags */

#define MPI2_SCSI_STATE_RESPONSE_INFO_VALID     (0x10)
#define MPI2_SCSI_STATE_TERMINATED              (0x08)
#define MPI2_SCSI_STATE_NO_SCSI_STATUS          (0x04)
#define MPI2_SCSI_STATE_AUTOSENSE_FAILED        (0x02)
#define MPI2_SCSI_STATE_AUTOSENSE_VALID         (0x01)

/* masks and shifts for the ResponseInfo field */

#define MPI2_SCSI_RI_MASK_REASONCODE            (0x000000FF)
#define MPI2_SCSI_RI_SHIFT_REASONCODE           (0)

#define MPI2_SCSI_TASKTAG_UNKNOWN               (0xFFFF)


/*****************************************************************************
*  SCSI Task Management messages
*****************************************************************************/

/* SCSI Task Management Request Message */
typedef struct _MPI2_SCSI_TASK_MANAGE_REQUEST
{
    U16                     DevHandle;                      /* 0x00 */
    U8                      ChainOffset;                    /* 0x02 */
    U8                      Function;                       /* 0x03 */
    U8                      Reserved1;                      /* 0x04 */
    U8                      TaskType;                       /* 0x05 */
    U8                      Reserved2;                      /* 0x06 */
    U8                      MsgFlags;                       /* 0x07 */
    U8                      VP_ID;                          /* 0x08 */
    U8                      VF_ID;                          /* 0x09 */
    U16                     Reserved3;                      /* 0x0A */
    U8                      LUN[8];                         /* 0x0C */
    U32                     Reserved4[7];                   /* 0x14 */
    U16                     TaskMID;                        /* 0x30 */
    U16                     Reserved5;                      /* 0x32 */
} MPI2_SCSI_TASK_MANAGE_REQUEST,
  MPI2_POINTER PTR_MPI2_SCSI_TASK_MANAGE_REQUEST,
  Mpi2SCSITaskManagementRequest_t,
  MPI2_POINTER pMpi2SCSITaskManagementRequest_t;

/* TaskType values */

#define MPI2_SCSITASKMGMT_TASKTYPE_ABORT_TASK           (0x01)
#define MPI2_SCSITASKMGMT_TASKTYPE_ABRT_TASK_SET        (0x02)
#define MPI2_SCSITASKMGMT_TASKTYPE_TARGET_RESET         (0x03)
#define MPI2_SCSITASKMGMT_TASKTYPE_LOGICAL_UNIT_RESET   (0x05)
#define MPI2_SCSITASKMGMT_TASKTYPE_CLEAR_TASK_SET       (0x06)
#define MPI2_SCSITASKMGMT_TASKTYPE_QUERY_TASK           (0x07)
```



```
#define MPI2_SCSITASKMGMT_TASKTYPE_CLR_ACA              (0x08)
#define MPI2_SCSITASKMGMT_TASKTYPE_QRY_TASK_SET         (0x09)
#define MPI2_SCSITASKMGMT_TASKTYPE_QRY_ASYNC_EVENT      (0x0A)

/* obsolete TaskType name */
#define MPI2_SCSITASKMGMT_TASKTYPE_QRY_UNIT_ATTENTION   (MPI2_SCSITASKMGMT_TASKT

/* MsgFlags bits */

#define MPI2_SCSITASKMGMT_MSGFLAGS_MASK_TARGET_RESET    (0x18)
#define MPI2_SCSITASKMGMT_MSGFLAGS_LINK_RESET           (0x00)
#define MPI2_SCSITASKMGMT_MSGFLAGS_NEXUS_RESET_SRST     (0x08)
#define MPI2_SCSITASKMGMT_MSGFLAGS_SAS_HARD_LINK_RESET  (0x10)

#define MPI2_SCSITASKMGMT_MSGFLAGS_DO_NOT_SEND_TASK_IU  (0x01)



/* SCSI Task Management Reply Message */
typedef struct _MPI2_SCSI_TASK_MANAGE_REPLY
{
    U16                     DevHandle;                      /* 0x00 */
    U8                      MsgLength;                      /* 0x02 */
    U8                      Function;                       /* 0x03 */
    U8                      ResponseCode;                   /* 0x04 */
    U8                      TaskType;                       /* 0x05 */
    U8                      Reserved1;                      /* 0x06 */
    U8                      MsgFlags;                       /* 0x07 */
    U8                      VP_ID;                          /* 0x08 */
    U8                      VF_ID;                          /* 0x09 */
    U16                     Reserved2;                      /* 0x0A */
    U16                     Reserved3;                      /* 0x0C */
    U16                     IOCStatus;                      /* 0x0E */
    U32                     IOCLogInfo;                     /* 0x10 */
    U32                     TerminationCount;               /* 0x14 */
    U32                     ResponseInfo;                   /* 0x18 */
} MPI2_SCSI_TASK_MANAGE_REPLY,
  MPI2_POINTER PTR_MPI2_SCSI_TASK_MANAGE_REPLY,
  Mpi2SCSITaskManagementReply_t, MPI2_POINTER pMpi2SCSIManagementReply_t;

/* ResponseCode values */

#define MPI2_SCSITASKMGMT_RSP_TM_COMPLETE               (0x00)
#define MPI2_SCSITASKMGMT_RSP_INVALID_FRAME             (0x02)
#define MPI2_SCSITASKMGMT_RSP_TM_NOT_SUPPORTED          (0x04)
#define MPI2_SCSITASKMGMT_RSP_TM_FAILED                 (0x05)
#define MPI2_SCSITASKMGMT_RSP_TM_SUCCEEDED              (0x08)
#define MPI2_SCSITASKMGMT_RSP_TM_INVALID_LUN            (0x09)
#define MPI2_SCSITASKMGMT_RSP_TM_OVERLAPPED_TAG         (0x0A)
#define MPI2_SCSITASKMGMT_RSP_IO_QUEUED_ON_IOC          (0x80)

/* masks and shifts for the ResponseInfo field */

#define MPI2_SCSITASKMGMT_RI_MASK_REASONCODE            (0x000000FF)
#define MPI2_SCSITASKMGMT_RI_SHIFT_REASONCODE           (0)
#define MPI2_SCSITASKMGMT_RI_MASK_ARI2                  (0x0000FF00)
#define MPI2_SCSITASKMGMT_RI_SHIFT_ARI2                 (8)
#define MPI2_SCSITASKMGMT_RI_MASK_ARI1                  (0x00FF0000)
#define MPI2_SCSITASKMGMT_RI_SHIFT_ARI1                 (16)
#define MPI2_SCSITASKMGMT_RI_MASK_ARI0                  (0xFF000000)
#define MPI2_SCSITASKMGMT_RI_SHIFT_ARI0                 (24)


/*****************************************************************************
*  SCSI Enclosure Processor messages
*****************************************************************************/
```

```
/* SCSI Enclosure Processor Request Message */
typedef struct _MPI2_SEP_REQUEST
{
    U16                     DevHandle;          /* 0x00 */
    U8                      ChainOffset;        /* 0x02 */
    U8                      Function;           /* 0x03 */
    U8                      Action;             /* 0x04 */
    U8                      Flags;              /* 0x05 */
    U8                      Reserved1;          /* 0x06 */
    U8                      MsgFlags;           /* 0x07 */
    U8                      VP_ID;              /* 0x08 */
    U8                      VF_ID;              /* 0x09 */
    U16                     Reserved2;          /* 0x0A */
    U32                     SlotStatus;         /* 0x0C */
    U32                     Reserved3;          /* 0x10 */
    U32                     Reserved4;          /* 0x14 */
    U32                     Reserved5;          /* 0x18 */
    U16                     Slot;               /* 0x1C */
    U16                     EnclosureHandle;    /* 0x1E */
} MPI2_SEP_REQUEST, MPI2_POINTER PTR_MPI2_SEP_REQUEST,
  Mpi2SepRequest_t, MPI2_POINTER pMpi2SepRequest_t;

/* Action defines */
#define MPI2_SEP_REQ_ACTION_WRITE_STATUS                (0x00)
#define MPI2_SEP_REQ_ACTION_READ_STATUS                 (0x01)

/* Flags defines */
#define MPI2_SEP_REQ_FLAGS_DEVHANDLE_ADDRESS            (0x00)
#define MPI2_SEP_REQ_FLAGS_ENCLOSURE_SLOT_ADDRESS       (0x01)

/* SlotStatus defines */
#define MPI2_SEP_REQ_SLOTSTATUS_REQUEST_REMOVE          (0x00040000)
#define MPI2_SEP_REQ_SLOTSTATUS_IDENTIFY_REQUEST        (0x00020000)
#define MPI2_SEP_REQ_SLOTSTATUS_REBUILD_STOPPED         (0x00000200)
#define MPI2_SEP_REQ_SLOTSTATUS_HOT_SPARE               (0x00000100)
#define MPI2_SEP_REQ_SLOTSTATUS_UNCONFIGURED            (0x00000080)
#define MPI2_SEP_REQ_SLOTSTATUS_PREDICTED_FAULT         (0x00000040)
#define MPI2_SEP_REQ_SLOTSTATUS_IN_CRITICAL_ARRAY       (0x00000010)
#define MPI2_SEP_REQ_SLOTSTATUS_IN_FAILED_ARRAY         (0x00000008)
#define MPI2_SEP_REQ_SLOTSTATUS_DEV_REBUILDING          (0x00000004)
#define MPI2_SEP_REQ_SLOTSTATUS_DEV_FAULTY              (0x00000002)
#define MPI2_SEP_REQ_SLOTSTATUS_NO_ERROR                (0x00000001)


/* SCSI Enclosure Processor Reply Message */
typedef struct _MPI2_SEP_REPLY
{
    U16                     DevHandle;          /* 0x00 */
    U8                      MsgLength;          /* 0x02 */
    U8                      Function;           /* 0x03 */
    U8                      Action;             /* 0x04 */
    U8                      Flags;              /* 0x05 */
    U8                      Reserved1;          /* 0x06 */
    U8                      MsgFlags;           /* 0x07 */
    U8                      VP_ID;              /* 0x08 */
    U8                      VF_ID;              /* 0x09 */
    U16                     Reserved2;          /* 0x0A */
    U16                     Reserved3;          /* 0x0C */
    U16                     IOCStatus;          /* 0x0E */
    U32                     IOCLogInfo;         /* 0x10 */
    U32                     SlotStatus;         /* 0x14 */
    U32                     Reserved4;          /* 0x18 */
    U16                     Slot;               /* 0x1C */
    U16                     EnclosureHandle;    /* 0x1E */
} MPI2_SEP_REPLY, MPI2_POINTER PTR_MPI2_SEP_REPLY,
```



```
  Mpi2SepReply_t, MPI2_POINTER pMpi2SepReply_t;

/* SlotStatus defines */
#define MPI2_SEP_REPLY_SLOTSTATUS_REMOVE_READY          (0x00040000)
#define MPI2_SEP_REPLY_SLOTSTATUS_IDENTIFY_REQUEST      (0x00020000)
#define MPI2_SEP_REPLY_SLOTSTATUS_REBUILD_STOPPED       (0x00000200)
#define MPI2_SEP_REPLY_SLOTSTATUS_HOT_SPARE             (0x00000100)
#define MPI2_SEP_REPLY_SLOTSTATUS_UNCONFIGURED          (0x00000080)
#define MPI2_SEP_REPLY_SLOTSTATUS_PREDICTED_FAULT       (0x00000040)
#define MPI2_SEP_REPLY_SLOTSTATUS_IN_CRITICAL_ARRAY     (0x00000010)
#define MPI2_SEP_REPLY_SLOTSTATUS_IN_FAILED_ARRAY       (0x00000008)
#define MPI2_SEP_REPLY_SLOTSTATUS_DEV_REBUILDING        (0x00000004)
#define MPI2_SEP_REPLY_SLOTSTATUS_DEV_FAULTY            (0x00000002)
#define MPI2_SEP_REPLY_SLOTSTATUS_NO_ERROR              (0x00000001)


#endif


#endif /* ! codereview */
```

```
**********************************************************
   83013 Thu Jun 12 17:28:23 2014
new/usr/src/uts/common/sys/scsi/adapters/mpt_sas3/mpi/mpi2_ioc.h
4546 mpt_sas needs enhancing to support LSI MPI2.5
**********************************************************
/*
 * 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.
 *
 * 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.
 *
 *  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.
 *  ----------------------------------------------------------------------------
 */

#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 */
    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,
  Mpi2IOCInitRequest_t, MPI2_POINTER pMpi2IOCInitRequest_t;

/* WhoInit values */
#define MPI2_WHOINIT_NOT_INITIALIZED            (0x00)
#define MPI2_WHOINIT_SYSTEM_BIOS                (0x01)
#define MPI2_WHOINIT_ROM_BIOS                   (0x02)
#define MPI2_WHOINIT_PCI_PEER                   (0x03)
#define MPI2_WHOINIT_HOST_DRIVER                (0x04)
#define MPI2_WHOINIT_MANUFACTURER               (0x05)

/* MsgVersion */
#define MPI2_IOCINIT_MSGVERSION_MAJOR_MASK      (0xFF00)
#define MPI2_IOCINIT_MSGVERSION_MAJOR_SHIFT     (8)
#define MPI2_IOCINIT_MSGVERSION_MINOR_MASK      (0x00FF)
#define MPI2_IOCINIT_MSGVERSION_MINOR_SHIFT     (0)

/* HeaderVersion */
#define MPI2_IOCINIT_HDRVERSION_UNIT_MASK       (0xFF00)
#define MPI2_IOCINIT_HDRVERSION_UNIT_SHIFT      (8)
#define MPI2_IOCINIT_HDRVERSION_DEV_MASK        (0x00FF)
#define MPI2_IOCINIT_HDRVERSION_DEV_SHIFT       (0)

/* minimum depth for the Reply Descriptor Post Queue */
#define MPI2_RDPQ_DEPTH_MIN                     (16)


/* IOCInit Reply message */
typedef struct _MPI2_IOC_INIT_REPLY
{
    U8                      WhoInit;                        /* 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_IOC_INIT_REPLY, MPI2_POINTER PTR_MPI2_IOC_INIT_REPLY,
  Mpi2IOCInitReply_t, MPI2_POINTER pMpi2IOCInitReply_t;


/****************************************************************************
*  IOCFacts message
****************************************************************************/

/* IOCFacts Request message */
typedef struct _MPI2_IOC_FACTS_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 */
} MPI2_IOC_FACTS_REQUEST, MPI2_POINTER PTR_MPI2_IOC_FACTS_REQUEST,
  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 */
    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 */
} MPI2_IOC_FACTS_REPLY, MPI2_POINTER PTR_MPI2_IOC_FACTS_REPLY,
  Mpi2IOCFactsReply_t, MPI2_POINTER pMpi2IOCFactsReply_t;

/* MsgVersion */
#define MPI2_IOCFACTS_MSGVERSION_MAJOR_MASK             (0xFF00)
#define MPI2_IOCFACTS_MSGVERSION_MAJOR_SHIFT            (8)
#define MPI2_IOCFACTS_MSGVERSION_MINOR_MASK             (0x00FF)
#define MPI2_IOCFACTS_MSGVERSION_MINOR_SHIFT            (0)

/* HeaderVersion */
#define MPI2_IOCFACTS_HDRVERSION_UNIT_MASK              (0xFF00)
#define MPI2_IOCFACTS_HDRVERSION_UNIT_SHIFT             (8)
#define MPI2_IOCFACTS_HDRVERSION_DEV_MASK               (0x00FF)
#define MPI2_IOCFACTS_HDRVERSION_DEV_SHIFT              (0)

/* IOCExceptions */
#define MPI2_IOCFACTS_EXCEPT_IR_FOREIGN_CONFIG_MAX      (0x0100)

#define MPI2_IOCFACTS_EXCEPT_BOOTSTAT_MASK              (0x00E0)
#define MPI2_IOCFACTS_EXCEPT_BOOTSTAT_GOOD              (0x0000)
```



```
#define MPI2_IOCFACTS_EXCEPT_BOOTSTAT_BACKUP            (0x0020)
#define MPI2_IOCFACTS_EXCEPT_BOOTSTAT_RESTORED          (0x0040)
#define MPI2_IOCFACTS_EXCEPT_BOOTSTAT_CORRUPT_BACKUP    (0x0060)

#define MPI2_IOCFACTS_EXCEPT_METADATA_UNSUPPORTED       (0x0010)
#define MPI2_IOCFACTS_EXCEPT_MANUFACT_CHECKSUM_FAIL     (0x0008)
#define MPI2_IOCFACTS_EXCEPT_FW_CHECKSUM_FAIL           (0x0004)
#define MPI2_IOCFACTS_EXCEPT_RAID_CONFIG_INVALID        (0x0002)
#define MPI2_IOCFACTS_EXCEPT_CONFIG_CHECKSUM_FAIL       (0x0001)

/* defines for WhoInit field are after the IOCInit Request */

/* ProductID field uses MPI2_FW_HEADER_PID_ */

/* IOCCapabilities */
#define MPI25_IOCFACTS_CAPABILITY_FAST_PATH_CAPABLE     (0x00020000)
#define MPI2_IOCFACTS_CAPABILITY_HOST_BASED_DISCOVERY   (0x00010000)
#define MPI2_IOCFACTS_CAPABILITY_MSI_X_INDEX            (0x00008000)
#define MPI2_IOCFACTS_CAPABILITY_RAID_ACCELERATOR       (0x00004000)
#define MPI2_IOCFACTS_CAPABILITY_EVENT_REPLAY           (0x00002000)
#define MPI2_IOCFACTS_CAPABILITY_INTEGRATED_RAID        (0x00001000)
#define MPI2_IOCFACTS_CAPABILITY_TLR                    (0x00000800)
#define MPI2_IOCFACTS_CAPABILITY_MULTICAST              (0x00000100)
#define MPI2_IOCFACTS_CAPABILITY_BIDIRECTIONAL_TARGET   (0x00000080)
#define MPI2_IOCFACTS_CAPABILITY_EEDP                   (0x00000040)
#define MPI2_IOCFACTS_CAPABILITY_EXTENDED_BUFFER        (0x00000020)
#define MPI2_IOCFACTS_CAPABILITY_SNAPSHOT_BUFFER        (0x00000010)
#define MPI2_IOCFACTS_CAPABILITY_DIAG_TRACE_BUFFER      (0x00000008)
#define MPI2_IOCFACTS_CAPABILITY_TASK_SET_FULL_HANDLING (0x00000004)

/* ProtocolFlags */
#define MPI2_IOCFACTS_PROTOCOL_SCSI_TARGET              (0x0001)
#define MPI2_IOCFACTS_PROTOCOL_SCSI_INITIATOR           (0x0002)


/*****************************************************************************
*  PortFacts message
*****************************************************************************/

/* PortFacts Request message */
typedef struct _MPI2_PORT_FACTS_REQUEST
{
    U16                     Reserved1;                      /* 0x00 */
    U8                      ChainOffset;                    /* 0x02 */
    U8                      Function;                       /* 0x03 */
    U16                     Reserved2;                      /* 0x04 */
    U8                      PortNumber;                     /* 0x06 */
    U8                      MsgFlags;                       /* 0x07 */
    U8                      VP_ID;                          /* 0x08 */
    U8                      VF_ID;                          /* 0x09 */
    U16                     Reserved3;                      /* 0x0A */
} MPI2_PORT_FACTS_REQUEST, MPI2_POINTER PTR_MPI2_PORT_FACTS_REQUEST,
  Mpi2PortFactsRequest_t, MPI2_POINTER pMpi2PortFactsRequest_t;

/* PortFacts Reply message */
typedef struct _MPI2_PORT_FACTS_REPLY
{
    U16                     Reserved1;                      /* 0x00 */
    U8                      MsgLength;                      /* 0x02 */
    U8                      Function;                       /* 0x03 */
    U16                     Reserved2;                      /* 0x04 */
    U8                      PortNumber;                     /* 0x06 */
    U8                      MsgFlags;                       /* 0x07 */
    U8                      VP_ID;                          /* 0x08 */
    U8                      VF_ID;                          /* 0x09 */
    U16                     Reserved3;                      /* 0x0A */
```

```
    U16                     Reserved4;                      /* 0x0C */
    U16                     IOCStatus;                      /* 0x0E */
    U32                     IOCLogInfo;                     /* 0x10 */
    U8                      Reserved5;                      /* 0x14 */
    U8                      PortType;                       /* 0x15 */
    U16                     Reserved6;                      /* 0x16 */
    U16                     MaxPostedCmdBuffers;            /* 0x18 */
    U16                     Reserved7;                      /* 0x1A */
} MPI2_PORT_FACTS_REPLY, MPI2_POINTER PTR_MPI2_PORT_FACTS_REPLY,
  Mpi2PortFactsReply_t, MPI2_POINTER pMpi2PortFactsReply_t;

/* PortType values */
#define MPI2_PORTFACTS_PORTTYPE_INACTIVE            (0x00)
#define MPI2_PORTFACTS_PORTTYPE_FC                  (0x10)
#define MPI2_PORTFACTS_PORTTYPE_ISCSI               (0x20)
#define MPI2_PORTFACTS_PORTTYPE_SAS_PHYSICAL        (0x30)
#define MPI2_PORTFACTS_PORTTYPE_SAS_VIRTUAL         (0x31)


/*****************************************************************************
*  PortEnable message
*****************************************************************************/

/* PortEnable Request message */
typedef struct _MPI2_PORT_ENABLE_REQUEST
{
    U16                     Reserved1;                      /* 0x00 */
    U8                      ChainOffset;                    /* 0x02 */
    U8                      Function;                       /* 0x03 */
    U8                      Reserved2;                      /* 0x04 */
    U8                      PortFlags;                      /* 0x05 */
    U8                      Reserved3;                      /* 0x06 */
    U8                      MsgFlags;                       /* 0x07 */
    U8                      VP_ID;                          /* 0x08 */
    U8                      VF_ID;                          /* 0x09 */
    U16                     Reserved4;                      /* 0x0A */
} MPI2_PORT_ENABLE_REQUEST, MPI2_POINTER PTR_MPI2_PORT_ENABLE_REQUEST,
  Mpi2PortEnableRequest_t, MPI2_POINTER pMpi2PortEnableRequest_t;


/* PortEnable Reply message */
typedef struct _MPI2_PORT_ENABLE_REPLY
{
    U16                     Reserved1;                      /* 0x00 */
    U8                      MsgLength;                      /* 0x02 */
    U8                      Function;                       /* 0x03 */
    U8                      Reserved2;                      /* 0x04 */
    U8                      PortFlags;                      /* 0x05 */
    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_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)


/* 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,
  MPI2_POINTER pMpi2EventDataHardResetReceived_t;

/* Task Set Full Event data */
/*   this event is obsolete */

typedef struct _MPI2_EVENT_DATA_TASK_SET_FULL
{
    U16                     DevHandle;                      /* 0x00 */
    U16                     CurrentDepth;                   /* 0x02 */
} MPI2_EVENT_DATA_TASK_SET_FULL, MPI2_POINTER PTR_MPI2_EVENT_DATA_TASK_SET_FULL,
  Mpi2EventDataTaskSetFull_t, MPI2_POINTER pMpi2EventDataTaskSetFull_t;


/* SAS Device Status Change Event data */

typedef struct _MPI2_EVENT_DATA_SAS_DEVICE_STATUS_CHANGE
{
    U16                     TaskTag;                        /* 0x00 */
    U8                      ReasonCode;                     /* 0x02 */
```



```
    U8                      Reserved1;                      /* 0x03 */
    U8                      ASC;                            /* 0x04 */
    U8                      ASCQ;                           /* 0x05 */
    U16                     DevHandle;                      /* 0x06 */
    U32                     Reserved2;                      /* 0x08 */
    U64                     SASAddress;                     /* 0x0C */
    U8                      LUN[8];                         /* 0x14 */
} MPI2_EVENT_DATA_SAS_DEVICE_STATUS_CHANGE,
  MPI2_POINTER PTR_MPI2_EVENT_DATA_SAS_DEVICE_STATUS_CHANGE,
  Mpi2EventDataSasDeviceStatusChange_t,
  MPI2_POINTER pMpi2EventDataSasDeviceStatusChange_t;

/* SAS Device Status Change Event data ReasonCode values */
#define MPI2_EVENT_SAS_DEV_STAT_RC_SMART_DATA                           (0x05)
#define MPI2_EVENT_SAS_DEV_STAT_RC_UNSUPPORTED                          (0x07)
#define MPI2_EVENT_SAS_DEV_STAT_RC_INTERNAL_DEVICE_RESET                (0x08)
#define MPI2_EVENT_SAS_DEV_STAT_RC_TASK_ABORT_INTERNAL                  (0x09)
#define MPI2_EVENT_SAS_DEV_STAT_RC_ABORT_TASK_SET_INTERNAL              (0x0A)
#define MPI2_EVENT_SAS_DEV_STAT_RC_CLEAR_TASK_SET_INTERNAL              (0x0B)
#define MPI2_EVENT_SAS_DEV_STAT_RC_QUERY_TASK_INTERNAL                  (0x0C)
#define MPI2_EVENT_SAS_DEV_STAT_RC_ASYNC_NOTIFICATION                   (0x0D)
#define MPI2_EVENT_SAS_DEV_STAT_RC_CMP_INTERNAL_DEV_RESET               (0x0E)
#define MPI2_EVENT_SAS_DEV_STAT_RC_CMP_TASK_ABORT_INTERNAL              (0x0F)
#define MPI2_EVENT_SAS_DEV_STAT_RC_SATA_INIT_FAILURE                    (0x10)
#define MPI2_EVENT_SAS_DEV_STAT_RC_EXPANDER_REDUCED_FUNCTIONALITY       (0x11)
#define MPI2_EVENT_SAS_DEV_STAT_RC_CMP_EXPANDER_REDUCED_FUNCTIONALITY   (0x12)


/* Integrated RAID Operation Status Event data */

typedef struct _MPI2_EVENT_DATA_IR_OPERATION_STATUS
{
    U16                     VolDevHandle;               /* 0x00 */
    U16                     Reserved1;                  /* 0x02 */
    U8                      RAIDOperation;              /* 0x04 */
    U8                      PercentComplete;            /* 0x05 */
    U16                     Reserved2;                  /* 0x06 */
    U32                     Resereved3;                 /* 0x08 */
} MPI2_EVENT_DATA_IR_OPERATION_STATUS,
  MPI2_POINTER PTR_MPI2_EVENT_DATA_IR_OPERATION_STATUS,
  Mpi2EventDataIrOperationStatus_t,
  MPI2_POINTER pMpi2EventDataIrOperationStatus_t;

/* Integrated RAID Operation Status Event data RAIDOperation values */
#define MPI2_EVENT_IR_RAIDOP_RESYNC                     (0x00)
#define MPI2_EVENT_IR_RAIDOP_ONLINE_CAP_EXPANSION       (0x01)
#define MPI2_EVENT_IR_RAIDOP_CONSISTENCY_CHECK          (0x02)
#define MPI2_EVENT_IR_RAIDOP_BACKGROUND_INIT            (0x03)
#define MPI2_EVENT_IR_RAIDOP_MAKE_DATA_CONSISTENT       (0x04)


/* Integrated RAID Volume Event data */

typedef struct _MPI2_EVENT_DATA_IR_VOLUME
{
    U16                     VolDevHandle;               /* 0x00 */
    U8                      ReasonCode;                 /* 0x02 */
    U8                      Reserved1;                  /* 0x03 */
    U32                     NewValue;                   /* 0x04 */
    U32                     PreviousValue;              /* 0x08 */
} MPI2_EVENT_DATA_IR_VOLUME, MPI2_POINTER PTR_MPI2_EVENT_DATA_IR_VOLUME,
  Mpi2EventDataIrVolume_t, MPI2_POINTER pMpi2EventDataIrVolume_t;

/* Integrated RAID Volume Event data ReasonCode values */
#define MPI2_EVENT_IR_VOLUME_RC_SETTINGS_CHANGED        (0x01)
#define MPI2_EVENT_IR_VOLUME_RC_STATUS_FLAGS_CHANGED    (0x02)
```

```
#define MPI2_EVENT_IR_VOLUME_RC_STATE_CHANGED           (0x03)


/* Integrated RAID Physical Disk Event data */

typedef struct _MPI2_EVENT_DATA_IR_PHYSICAL_DISK
{
    U16                     Reserved1;                  /* 0x00 */
    U8                      ReasonCode;                 /* 0x02 */
    U8                      PhysDiskNum;                /* 0x03 */
    U16                     PhysDiskDevHandle;          /* 0x04 */
    U16                     Reserved2;                  /* 0x06 */
    U16                     Slot;                       /* 0x08 */
    U16                     EnclosureHandle;            /* 0x0A */
    U32                     NewValue;                   /* 0x0C */
    U32                     PreviousValue;              /* 0x10 */
} MPI2_EVENT_DATA_IR_PHYSICAL_DISK,
  MPI2_POINTER PTR_MPI2_EVENT_DATA_IR_PHYSICAL_DISK,
  Mpi2EventDataIrPhysicalDisk_t, MPI2_POINTER pMpi2EventDataIrPhysicalDisk_t;

/* Integrated RAID Physical Disk Event data ReasonCode values */
#define MPI2_EVENT_IR_PHYSDISK_RC_SETTINGS_CHANGED      (0x01)
#define MPI2_EVENT_IR_PHYSDISK_RC_STATUS_FLAGS_CHANGED  (0x02)
#define MPI2_EVENT_IR_PHYSDISK_RC_STATE_CHANGED         (0x03)


/* Integrated RAID Configuration Change List Event data */

/*
 * Host code (drivers, BIOS, utilities, etc.) should leave this define set to
 * one and check NumElements at runtime.
 */
#ifndef MPI2_EVENT_IR_CONFIG_ELEMENT_COUNT
#define MPI2_EVENT_IR_CONFIG_ELEMENT_COUNT          (1)
#endif

typedef struct _MPI2_EVENT_IR_CONFIG_ELEMENT
{
    U16                     ElementFlags;               /* 0x00 */
    U16                     VolDevHandle;               /* 0x02 */
    U8                      ReasonCode;                 /* 0x04 */
    U8                      PhysDiskNum;                /* 0x05 */
    U16                     PhysDiskDevHandle;          /* 0x06 */
} MPI2_EVENT_IR_CONFIG_ELEMENT, MPI2_POINTER PTR_MPI2_EVENT_IR_CONFIG_ELEMENT,
  Mpi2EventIrConfigElement_t, MPI2_POINTER pMpi2EventIrConfigElement_t;

/* IR Configuration Change List Event data ElementFlags values */
#define MPI2_EVENT_IR_CHANGE_EFLAGS_ELEMENT_TYPE_MASK   (0x000F)
#define MPI2_EVENT_IR_CHANGE_EFLAGS_VOLUME_ELEMENT      (0x0000)
#define MPI2_EVENT_IR_CHANGE_EFLAGS_VOLPHYSDISK_ELEMENT (0x0001)
#define MPI2_EVENT_IR_CHANGE_EFLAGS_HOTSPARE_ELEMENT    (0x0002)

/* IR Configuration Change List Event data ReasonCode values */
#define MPI2_EVENT_IR_CHANGE_RC_ADDED                   (0x01)
#define MPI2_EVENT_IR_CHANGE_RC_REMOVED                 (0x02)
#define MPI2_EVENT_IR_CHANGE_RC_NO_CHANGE               (0x03)
#define MPI2_EVENT_IR_CHANGE_RC_HIDE                    (0x04)
#define MPI2_EVENT_IR_CHANGE_RC_UNHIDE                  (0x05)
#define MPI2_EVENT_IR_CHANGE_RC_VOLUME_CREATED          (0x06)
#define MPI2_EVENT_IR_CHANGE_RC_VOLUME_DELETED          (0x07)
#define MPI2_EVENT_IR_CHANGE_RC_PD_CREATED              (0x08)
#define MPI2_EVENT_IR_CHANGE_RC_PD_DELETED              (0x09)

typedef struct _MPI2_EVENT_DATA_IR_CONFIG_CHANGE_LIST
{
    U8                              NumElements;        /* 0x00 */
```



```
    U8                              Reserved1;          /* 0x01 */
    U8                              Reserved2;          /* 0x02 */
    U8                              ConfigNum;          /* 0x03 */
    U32                             Flags;              /* 0x04 */
    MPI2_EVENT_IR_CONFIG_ELEMENT    ConfigElement[MPI2_EVENT_IR_CONFIG_ELEMENT_C
} MPI2_EVENT_DATA_IR_CONFIG_CHANGE_LIST,
  MPI2_POINTER PTR_MPI2_EVENT_DATA_IR_CONFIG_CHANGE_LIST,
  Mpi2EventDataIrConfigChangeList_t,
  MPI2_POINTER pMpi2EventDataIrConfigChangeList_t;

/* IR Configuration Change List Event data Flags values */
#define MPI2_EVENT_IR_CHANGE_FLAGS_FOREIGN_CONFIG   (0x00000001)


/* SAS Discovery Event data */

typedef struct _MPI2_EVENT_DATA_SAS_DISCOVERY
{
    U8                      Flags;                      /* 0x00 */
    U8                      ReasonCode;                 /* 0x01 */
    U8                      PhysicalPort;               /* 0x02 */
    U8                      Reserved1;                  /* 0x03 */
    U32                     DiscoveryStatus;            /* 0x04 */
} MPI2_EVENT_DATA_SAS_DISCOVERY,
  MPI2_POINTER PTR_MPI2_EVENT_DATA_SAS_DISCOVERY,
  Mpi2EventDataSasDiscovery_t, MPI2_POINTER pMpi2EventDataSasDiscovery_t;

/* SAS Discovery Event data Flags values */
#define MPI2_EVENT_SAS_DISC_DEVICE_CHANGE                   (0x02)
#define MPI2_EVENT_SAS_DISC_IN_PROGRESS                     (0x01)

/* SAS Discovery Event data ReasonCode values */
#define MPI2_EVENT_SAS_DISC_RC_STARTED                      (0x01)
#define MPI2_EVENT_SAS_DISC_RC_COMPLETED                    (0x02)

/* SAS Discovery Event data DiscoveryStatus values */
#define MPI2_EVENT_SAS_DISC_DS_MAX_ENCLOSURES_EXCEED            (0x80000000)
#define MPI2_EVENT_SAS_DISC_DS_MAX_EXPANDERS_EXCEED             (0x40000000)
#define MPI2_EVENT_SAS_DISC_DS_MAX_DEVICES_EXCEED               (0x20000000)
#define MPI2_EVENT_SAS_DISC_DS_MAX_TOPO_PHYS_EXCEED             (0x10000000)
#define MPI2_EVENT_SAS_DISC_DS_DOWNSTREAM_INITIATOR             (0x08000000)
#define MPI2_EVENT_SAS_DISC_DS_MULTI_SUBTRACTIVE_SUBTRACTIVE    (0x00008000)
#define MPI2_EVENT_SAS_DISC_DS_EXP_MULTI_SUBTRACTIVE            (0x00004000)
#define MPI2_EVENT_SAS_DISC_DS_MULTI_PORT_DOMAIN                (0x00002000)
#define MPI2_EVENT_SAS_DISC_DS_TABLE_TO_SUBTRACTIVE_LINK        (0x00001000)
#define MPI2_EVENT_SAS_DISC_DS_UNSUPPORTED_DEVICE               (0x00000800)
#define MPI2_EVENT_SAS_DISC_DS_TABLE_LINK                       (0x00000400)
#define MPI2_EVENT_SAS_DISC_DS_SUBTRACTIVE_LINK                 (0x00000200)
#define MPI2_EVENT_SAS_DISC_DS_SMP_CRC_ERROR                    (0x00000100)
#define MPI2_EVENT_SAS_DISC_DS_SMP_FUNCTION_FAILED              (0x00000080)
#define MPI2_EVENT_SAS_DISC_DS_INDEX_NOT_EXIST                  (0x00000040)
#define MPI2_EVENT_SAS_DISC_DS_OUT_ROUTE_ENTRIES                (0x00000020)
#define MPI2_EVENT_SAS_DISC_DS_SMP_TIMEOUT                      (0x00000010)
#define MPI2_EVENT_SAS_DISC_DS_MULTIPLE_PORTS                   (0x00000004)
#define MPI2_EVENT_SAS_DISC_DS_UNADDRESSABLE_DEVICE             (0x00000002)
#define MPI2_EVENT_SAS_DISC_DS_LOOP_DETECTED                    (0x00000001)


/* SAS Broadcast Primitive Event data */

typedef struct _MPI2_EVENT_DATA_SAS_BROADCAST_PRIMITIVE
{
    U8                      PhyNum;                     /* 0x00 */
    U8                      Port;                       /* 0x01 */
    U8                      PortWidth;                  /* 0x02 */
    U8                      Primitive;                  /* 0x03 */
```

```
} MPI2_EVENT_DATA_SAS_BROADCAST_PRIMITIVE,
  MPI2_POINTER PTR_MPI2_EVENT_DATA_SAS_BROADCAST_PRIMITIVE,
  Mpi2EventDataSasBroadcastPrimitive_t,
  MPI2_POINTER pMpi2EventDataSasBroadcastPrimitive_t;

/* defines for the Primitive field */
#define MPI2_EVENT_PRIMITIVE_CHANGE                        (0x01)
#define MPI2_EVENT_PRIMITIVE_SES                           (0x02)
#define MPI2_EVENT_PRIMITIVE_EXPANDER                      (0x03)
#define MPI2_EVENT_PRIMITIVE_ASYNCHRONOUS_EVENT            (0x04)
#define MPI2_EVENT_PRIMITIVE_RESERVED3                     (0x05)
#define MPI2_EVENT_PRIMITIVE_RESERVED4                     (0x06)
#define MPI2_EVENT_PRIMITIVE_CHANGE0_RESERVED              (0x07)
#define MPI2_EVENT_PRIMITIVE_CHANGE1_RESERVED              (0x08)


/* SAS Initiator Device Status Change Event data */

typedef struct _MPI2_EVENT_DATA_SAS_INIT_DEV_STATUS_CHANGE
{
    U8                      ReasonCode;                 /* 0x00 */
    U8                      PhysicalPort;               /* 0x01 */
    U16                     DevHandle;                  /* 0x02 */
    U64                     SASAddress;                 /* 0x04 */
} MPI2_EVENT_DATA_SAS_INIT_DEV_STATUS_CHANGE,
  MPI2_POINTER PTR_MPI2_EVENT_DATA_SAS_INIT_DEV_STATUS_CHANGE,
  Mpi2EventDataSasInitDevStatusChange_t,
  MPI2_POINTER pMpi2EventDataSasInitDevStatusChange_t;

/* SAS Initiator Device Status Change event ReasonCode values */
#define MPI2_EVENT_SAS_INIT_RC_ADDED                (0x01)
#define MPI2_EVENT_SAS_INIT_RC_NOT_RESPONDING       (0x02)


/* SAS Initiator Device Table Overflow Event data */

typedef struct _MPI2_EVENT_DATA_SAS_INIT_TABLE_OVERFLOW
{
    U16                     MaxInit;                    /* 0x00 */
    U16                     CurrentInit;                /* 0x02 */
    U64                     SASAddress;                 /* 0x04 */
} MPI2_EVENT_DATA_SAS_INIT_TABLE_OVERFLOW,
  MPI2_POINTER PTR_MPI2_EVENT_DATA_SAS_INIT_TABLE_OVERFLOW,
  Mpi2EventDataSasInitTableOverflow_t,
  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)
#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)
#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)

/* 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)



/*****************************************************************************
*  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 */
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)
#define MPI2_FW_DOWNLOAD_ITYPE_COMMON_BOOT_BLOCK    (0x0B)
#define MPI2_FW_DOWNLOAD_ITYPE_MIN_PRODUCT_SPECIFIC (0xF0)

/* MPI v2.0 FWDownload TransactionContext Element */
typedef struct _MPI2_FW_DOWNLOAD_TCSGE
{
    U8                      Reserved1;                  /* 0x00 */
    U8                      ContextSize;                /* 0x01 */
    U8                      DetailsLength;              /* 0x02 */
    U8                      Flags;                      /* 0x03 */
    U32                     Reserved2;                  /* 0x04 */
    U32                     ImageOffset;                /* 0x08 */
    U32                     ImageSize;                  /* 0x0C */
} MPI2_FW_DOWNLOAD_TCSGE, MPI2_POINTER PTR_MPI2_FW_DOWNLOAD_TCSGE,
  Mpi2FWDownloadTCSGE_t, MPI2_POINTER pMpi2FWDownloadTCSGE_t;


/* MPI v2.5 FWDownload Request message */
typedef struct _MPI25_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 */
    U32                     Reserved6;                  /* 0x14 */
    U32                     ImageOffset;                /* 0x18 */
    U32                     ImageSize;                  /* 0x1C */
    MPI25_SGE_IO_UNION      SGL;                        /* 0x20 */
} MPI25_FW_DOWNLOAD_REQUEST, MPI2_POINTER PTR_MPI25_FW_DOWNLOAD_REQUEST,
  Mpi25FWDownloadRequest, MPI2_POINTER pMpi25FWDownloadRequest;


/* FWDownload Reply message */
typedef struct _MPI2_FW_DOWNLOAD_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 */
} MPI2_FW_DOWNLOAD_REPLY, MPI2_POINTER PTR_MPI2_FW_DOWNLOAD_REPLY,
  Mpi2FWDownloadReply_t, MPI2_POINTER pMpi2FWDownloadReply_t;


/*****************************************************************************
*  FWUpload message
*****************************************************************************/

/* MPI v2.0 FWUpload Request message */
typedef struct _MPI2_FW_UPLOAD_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                     Reserved5;                  /* 0x0C */
    U32                     Reserved6;                  /* 0x10 */
    MPI2_MPI_SGE_UNION      SGL;                        /* 0x14 */
} MPI2_FW_UPLOAD_REQUEST, MPI2_POINTER PTR_MPI2_FW_UPLOAD_REQUEST,
  Mpi2FWUploadRequest_t, MPI2_POINTER pMpi2FWUploadRequest_t;

#define MPI2_FW_UPLOAD_ITYPE_FW_CURRENT         (0x00)
#define MPI2_FW_UPLOAD_ITYPE_FW_FLASH           (0x01)
#define MPI2_FW_UPLOAD_ITYPE_BIOS_FLASH         (0x02)
#define MPI2_FW_UPLOAD_ITYPE_FW_BACKUP          (0x05)
#define MPI2_FW_UPLOAD_ITYPE_MANUFACTURING      (0x06)
#define MPI2_FW_UPLOAD_ITYPE_CONFIG_1           (0x07)
#define MPI2_FW_UPLOAD_ITYPE_CONFIG_2           (0x08)
#define MPI2_FW_UPLOAD_ITYPE_MEGARAID           (0x09)
#define MPI2_FW_UPLOAD_ITYPE_COMPLETE           (0x0A)
#define MPI2_FW_UPLOAD_ITYPE_COMMON_BOOT_BLOCK  (0x0B)

/* MPI v2.0 FWUpload TransactionContext Element */
```

```
typedef struct _MPI2_FW_UPLOAD_TCSGE
{
    U8                      Reserved1;                  /* 0x00 */
    U8                      ContextSize;                /* 0x01 */
    U8                      DetailsLength;              /* 0x02 */
    U8                      Flags;                      /* 0x03 */
    U32                     Reserved2;                  /* 0x04 */
    U32                     ImageOffset;                /* 0x08 */
    U32                     ImageSize;                  /* 0x0C */
} MPI2_FW_UPLOAD_TCSGE, MPI2_POINTER PTR_MPI2_FW_UPLOAD_TCSGE,
  Mpi2FWUploadTCSGE_t, MPI2_POINTER pMpi2FWUploadTCSGE_t;


/* MPI v2.5 FWUpload Request message */
typedef struct _MPI25_FW_UPLOAD_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                     Reserved5;                  /* 0x0C */
    U32                     Reserved6;                  /* 0x10 */
    U32                     Reserved7;                  /* 0x14 */
    U32                     ImageOffset;                /* 0x18 */
    U32                     ImageSize;                  /* 0x1C */
    MPI25_SGE_IO_UNION      SGL;                        /* 0x20 */
} MPI25_FW_UPLOAD_REQUEST, MPI2_POINTER PTR_MPI25_FW_UPLOAD_REQUEST,
  Mpi25FWUploadRequest_t, MPI2_POINTER pMpi25FWUploadRequest_t;


/* 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)

#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)

/* 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 */
    U32                     Checksum;                   /* 0x04 */
    U32                     ImageSize;                  /* 0x08 */
    U32                     NextImageHeaderOffset;      /* 0x0C */
    U32                     PackageVersion;             /* 0x10 */
    U32                     Reserved3;                  /* 0x14 */
    U32                     Reserved4;                  /* 0x18 */
    U32                     Reserved5;                  /* 0x1C */
    U8                      IdentifyString[32];         /* 0x20 */
} MPI2_EXT_IMAGE_HEADER, MPI2_POINTER PTR_MPI2_EXT_IMAGE_HEADER,
  Mpi2ExtImageHeader_t, MPI2_POINTER pMpi2ExtImageHeader_t;

/* useful offsets */
#define MPI2_EXT_IMAGE_IMAGETYPE_OFFSET         (0x00)
#define MPI2_EXT_IMAGE_IMAGESIZE_OFFSET         (0x08)
#define MPI2_EXT_IMAGE_NEXTIMAGE_OFFSET         (0x0C)

#define MPI2_EXT_IMAGE_HEADER_SIZE              (0x40)

/* defines for the ImageType field */
#define MPI2_EXT_IMAGE_TYPE_UNSPECIFIED         (0x00)
#define MPI2_EXT_IMAGE_TYPE_FW                  (0x01)
#define MPI2_EXT_IMAGE_TYPE_NVDATA              (0x03)
#define MPI2_EXT_IMAGE_TYPE_BOOTLOADER          (0x04)
#define MPI2_EXT_IMAGE_TYPE_INITIALIZATION      (0x05)
#define MPI2_EXT_IMAGE_TYPE_FLASH_LAYOUT        (0x06)
#define MPI2_EXT_IMAGE_TYPE_SUPPORTED_DEVICES   (0x07)
#define MPI2_EXT_IMAGE_TYPE_MEGARAID            (0x08)

#define MPI2_EXT_IMAGE_TYPE_MAX                 (MPI2_EXT_IMAGE_TYPE_MEGARAID)



/* FLASH Layout Extended Image Data */

/*
 * Host code (drivers, BIOS, utilities, etc.) should leave this define set to
```



```
 * one and check RegionsPerLayout at runtime.
 */
#ifndef MPI2_FLASH_NUMBER_OF_REGIONS
#define MPI2_FLASH_NUMBER_OF_REGIONS        (1)
#endif

/*
 * Host code (drivers, BIOS, utilities, etc.) should leave this define set to
 * one and check NumberOfLayouts at runtime.
 */
#ifndef MPI2_FLASH_NUMBER_OF_LAYOUTS
#define MPI2_FLASH_NUMBER_OF_LAYOUTS        (1)
#endif

typedef struct _MPI2_FLASH_REGION
{
    U8                      RegionType;                 /* 0x00 */
    U8                      Reserved1;                  /* 0x01 */
    U16                     Reserved2;                  /* 0x02 */
    U32                     RegionOffset;               /* 0x04 */
    U32                     RegionSize;                 /* 0x08 */
    U32                     Reserved3;                  /* 0x0C */
} MPI2_FLASH_REGION, MPI2_POINTER PTR_MPI2_FLASH_REGION,
  Mpi2FlashRegion_t, MPI2_POINTER pMpi2FlashRegion_t;

typedef struct _MPI2_FLASH_LAYOUT
{
    U32                     FlashSize;                  /* 0x00 */
    U32                     Reserved1;                  /* 0x04 */
    U32                     Reserved2;                  /* 0x08 */
    U32                     Reserved3;                  /* 0x0C */
    MPI2_FLASH_REGION       Region[MPI2_FLASH_NUMBER_OF_REGIONS];/* 0x10 */
} MPI2_FLASH_LAYOUT, MPI2_POINTER PTR_MPI2_FLASH_LAYOUT,
  Mpi2FlashLayout_t, MPI2_POINTER pMpi2FlashLayout_t;

typedef struct _MPI2_FLASH_LAYOUT_DATA
{
    U8                      ImageRevision;              /* 0x00 */
    U8                      Reserved1;                  /* 0x01 */
    U8                      SizeOfRegion;               /* 0x02 */
    U8                      Reserved2;                  /* 0x03 */
    U16                     NumberOfLayouts;            /* 0x04 */
    U16                     RegionsPerLayout;           /* 0x06 */
    U16                     MinimumSectorAlignment;     /* 0x08 */
    U16                     Reserved3;                  /* 0x0A */
    U32                     Reserved4;                  /* 0x0C */
    MPI2_FLASH_LAYOUT       Layout[MPI2_FLASH_NUMBER_OF_LAYOUTS];/* 0x10 */
} MPI2_FLASH_LAYOUT_DATA, MPI2_POINTER PTR_MPI2_FLASH_LAYOUT_DATA,
  Mpi2FlashLayoutData_t, MPI2_POINTER pMpi2FlashLayoutData_t;

/* defines for the RegionType field */
#define MPI2_FLASH_REGION_UNUSED                (0x00)
#define MPI2_FLASH_REGION_FIRMWARE              (0x01)
#define MPI2_FLASH_REGION_BIOS                  (0x02)
#define MPI2_FLASH_REGION_NVDATA                (0x03)
#define MPI2_FLASH_REGION_FIRMWARE_BACKUP       (0x05)
#define MPI2_FLASH_REGION_MFG_INFORMATION       (0x06)
#define MPI2_FLASH_REGION_CONFIG_1              (0x07)
#define MPI2_FLASH_REGION_CONFIG_2              (0x08)
#define MPI2_FLASH_REGION_MEGARAID              (0x09)
#define MPI2_FLASH_REGION_INIT                  (0x0A)

/* ImageRevision */
#define MPI2_FLASH_LAYOUT_IMAGE_REVISION        (0x00)
```

```
/* Supported Devices Extended Image Data */

/*
 * Host code (drivers, BIOS, utilities, etc.) should leave this define set to
 * one and check NumberOfDevices at runtime.
 */
#ifndef MPI2_SUPPORTED_DEVICES_IMAGE_NUM_DEVICES
#define MPI2_SUPPORTED_DEVICES_IMAGE_NUM_DEVICES    (1)
#endif

typedef struct _MPI2_SUPPORTED_DEVICE
{
    U16                     DeviceID;                   /* 0x00 */
    U16                     VendorID;                   /* 0x02 */
    U16                     DeviceIDMask;               /* 0x04 */
    U16                     Reserved1;                  /* 0x06 */
    U8                      LowPCIRev;                  /* 0x08 */
    U8                      HighPCIRev;                 /* 0x09 */
    U16                     Reserved2;                  /* 0x0A */
    U32                     Reserved3;                  /* 0x0C */
} MPI2_SUPPORTED_DEVICE, MPI2_POINTER PTR_MPI2_SUPPORTED_DEVICE,
  Mpi2SupportedDevice_t, MPI2_POINTER pMpi2SupportedDevice_t;

typedef struct _MPI2_SUPPORTED_DEVICES_DATA
{
    U8                      ImageRevision;              /* 0x00 */
    U8                      Reserved1;                  /* 0x01 */
    U8                      NumberOfDevices;            /* 0x02 */
    U8                      Reserved2;                  /* 0x03 */
    U32                     Reserved3;                  /* 0x04 */
    MPI2_SUPPORTED_DEVICE   SupportedDevice[MPI2_SUPPORTED_DEVICES_IMAGE_NUM_DEV
} MPI2_SUPPORTED_DEVICES_DATA, MPI2_POINTER PTR_MPI2_SUPPORTED_DEVICES_DATA,
  Mpi2SupportedDevicesData_t, MPI2_POINTER pMpi2SupportedDevicesData_t;

/* ImageRevision */
#define MPI2_SUPPORTED_DEVICES_IMAGE_REVISION   (0x00)


/* Init Extended Image Data */

typedef struct _MPI2_INIT_IMAGE_FOOTER

{
    U32                     BootFlags;                  /* 0x00 */
    U32                     ImageSize;                  /* 0x04 */
    U32                     Signature0;                 /* 0x08 */
    U32                     Signature1;                 /* 0x0C */
    U32                     Signature2;                 /* 0x10 */
    U32                     ResetVector;                /* 0x14 */
} MPI2_INIT_IMAGE_FOOTER, MPI2_POINTER PTR_MPI2_INIT_IMAGE_FOOTER,
  Mpi2InitImageFooter_t, MPI2_POINTER pMpi2InitImageFooter_t;

/* defines for the BootFlags field */
#define MPI2_INIT_IMAGE_BOOTFLAGS_OFFSET        (0x00)

/* defines for the ImageSize field */
#define MPI2_INIT_IMAGE_IMAGESIZE_OFFSET        (0x04)

/* defines for the Signature0 field */
#define MPI2_INIT_IMAGE_SIGNATURE0_OFFSET       (0x08)
#define MPI2_INIT_IMAGE_SIGNATURE0              (0x5AA55AEA)

/* defines for the Signature1 field */
#define MPI2_INIT_IMAGE_SIGNATURE1_OFFSET       (0x0C)
#define MPI2_INIT_IMAGE_SIGNATURE1              (0xA55AEAA5)
```



```
/* defines for the Signature2 field */
#define MPI2_INIT_IMAGE_SIGNATURE2_OFFSET       (0x10)
#define MPI2_INIT_IMAGE_SIGNATURE2              (0x5AEAA55A)

/* Signature fields as individual bytes */
#define MPI2_INIT_IMAGE_SIGNATURE_BYTE_0        (0xEA)
#define MPI2_INIT_IMAGE_SIGNATURE_BYTE_1        (0x5A)
#define MPI2_INIT_IMAGE_SIGNATURE_BYTE_2        (0xA5)
#define MPI2_INIT_IMAGE_SIGNATURE_BYTE_3        (0x5A)

#define MPI2_INIT_IMAGE_SIGNATURE_BYTE_4        (0xA5)
#define MPI2_INIT_IMAGE_SIGNATURE_BYTE_5        (0xEA)
#define MPI2_INIT_IMAGE_SIGNATURE_BYTE_6        (0x5A)
#define MPI2_INIT_IMAGE_SIGNATURE_BYTE_7        (0xA5)

#define MPI2_INIT_IMAGE_SIGNATURE_BYTE_8        (0x5A)
#define MPI2_INIT_IMAGE_SIGNATURE_BYTE_9        (0xA5)
#define MPI2_INIT_IMAGE_SIGNATURE_BYTE_A        (0xEA)
#define MPI2_INIT_IMAGE_SIGNATURE_BYTE_B        (0x5A)

/* defines for the ResetVector field */
#define MPI2_INIT_IMAGE_RESETVECTOR_OFFSET      (0x14)


/*****************************************************************************
*  PowerManagementControl message
*****************************************************************************/

/* PowerManagementControl Request message */
typedef struct _MPI2_PWR_MGMT_CONTROL_REQUEST
{
    U8                      Feature;                    /* 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                      Parameter1;                 /* 0x0C */
    U8                      Parameter2;                 /* 0x0D */
    U8                      Parameter3;                 /* 0x0E */
    U8                      Parameter4;                 /* 0x0F */
    U32                     Reserved5;                  /* 0x10 */
    U32                     Reserved6;                  /* 0x14 */
} MPI2_PWR_MGMT_CONTROL_REQUEST, MPI2_POINTER PTR_MPI2_PWR_MGMT_CONTROL_REQUEST,
  Mpi2PwrMgmtControlRequest_t, MPI2_POINTER pMpi2PwrMgmtControlRequest_t;

/* defines for the Feature field */
#define MPI2_PM_CONTROL_FEATURE_DA_PHY_POWER_COND       (0x01)
#define MPI2_PM_CONTROL_FEATURE_PORT_WIDTH_MODULATION   (0x02)
#define MPI2_PM_CONTROL_FEATURE_PCIE_LINK               (0x03)
#define MPI2_PM_CONTROL_FEATURE_IOC_SPEED               (0x04)
#define MPI2_PM_CONTROL_FEATURE_MIN_PRODUCT_SPECIFIC    (0x80)
#define MPI2_PM_CONTROL_FEATURE_MAX_PRODUCT_SPECIFIC    (0xFF)

/* parameter usage for the MPI2_PM_CONTROL_FEATURE_DA_PHY_POWER_COND Feature */
/* Parameter1 contains a PHY number */
/* Parameter2 indicates power condition action using these defines */
#define MPI2_PM_CONTROL_PARAM2_PARTIAL                  (0x01)
#define MPI2_PM_CONTROL_PARAM2_SLUMBER                  (0x02)
#define MPI2_PM_CONTROL_PARAM2_EXIT_PWR_MGMT            (0x03)
/* Parameter3 and Parameter4 are reserved */
```

```
/* parameter usage for the MPI2_PM_CONTROL_FEATURE_PORT_WIDTH_MODULATION Feature
/* Parameter1 contains SAS port width modulation group number */
/* Parameter2 indicates IOC action using these defines */
#define MPI2_PM_CONTROL_PARAM2_REQUEST_OWNERSHIP        (0x01)
#define MPI2_PM_CONTROL_PARAM2_CHANGE_MODULATION        (0x02)
#define MPI2_PM_CONTROL_PARAM2_RELINQUISH_OWNERSHIP     (0x03)
/* Parameter3 indicates desired modulation level using these defines */
#define MPI2_PM_CONTROL_PARAM3_25_PERCENT               (0x00)
#define MPI2_PM_CONTROL_PARAM3_50_PERCENT               (0x01)
#define MPI2_PM_CONTROL_PARAM3_75_PERCENT               (0x02)
#define MPI2_PM_CONTROL_PARAM3_100_PERCENT              (0x03)
/* Parameter4 is reserved */

/* parameter usage for the MPI2_PM_CONTROL_FEATURE_PCIE_LINK Feature */
/* Parameter1 indicates desired PCIe link speed using these defines */
#define MPI2_PM_CONTROL_PARAM1_PCIE_2_5_GBPS            (0x00)
#define MPI2_PM_CONTROL_PARAM1_PCIE_5_0_GBPS            (0x01)
#define MPI2_PM_CONTROL_PARAM1_PCIE_8_0_GBPS            (0x02)
/* Parameter2 indicates desired PCIe link width using these defines */
#define MPI2_PM_CONTROL_PARAM2_WIDTH_X1                 (0x01)
#define MPI2_PM_CONTROL_PARAM2_WIDTH_X2                 (0x02)
#define MPI2_PM_CONTROL_PARAM2_WIDTH_X4                 (0x04)
#define MPI2_PM_CONTROL_PARAM2_WIDTH_X8                 (0x08)
/* Parameter3 and Parameter4 are reserved */

/* parameter usage for the MPI2_PM_CONTROL_FEATURE_IOC_SPEED Feature */
/* Parameter1 indicates desired IOC hardware clock speed using these defines */
#define MPI2_PM_CONTROL_PARAM1_FULL_IOC_SPEED           (0x01)
#define MPI2_PM_CONTROL_PARAM1_HALF_IOC_SPEED           (0x02)
#define MPI2_PM_CONTROL_PARAM1_QUARTER_IOC_SPEED        (0x04)
#define MPI2_PM_CONTROL_PARAM1_EIGHTH_IOC_SPEED         (0x08)
/* Parameter2, Parameter3, and Parameter4 are reserved */


/* PowerManagementControl Reply message */
typedef struct _MPI2_PWR_MGMT_CONTROL_REPLY
{
    U8                      Feature;                    /* 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_PWR_MGMT_CONTROL_REPLY, MPI2_POINTER PTR_MPI2_PWR_MGMT_CONTROL_REPLY,
  Mpi2PwrMgmtControlReply_t, MPI2_POINTER pMpi2PwrMgmtControlReply_t;


#endif

#endif /* ! codereview */
```

```
**********************************************************
   16251 Thu Jun 12 17:28:23 2014
new/usr/src/uts/common/sys/scsi/adapters/mpt_sas3/mpi/mpi2_raid.h
4546 mpt_sas needs enhancing to support LSI MPI2.5
**********************************************************
/*
 * 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.
 *
 * 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
 *
 *  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.
 *  --------------------------------------------------------------------------
 */

#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,
  Mpi2RaidVolumePhysDisk_t, MPI2_POINTER pMpi2RaidVolumePhysDisk_t;

/* defines for the PhysDiskMap field */
#define MPI2_RAIDACTION_PHYSDISK_PRIMARY            (0x01)
#define MPI2_RAIDACTION_PHYSDISK_SECONDARY          (0x02)

typedef struct _MPI2_RAID_VOLUME_CREATION_STRUCT
{
    U8                          NumPhysDisks;               /* 0x00 */
    U8                          VolumeType;                 /* 0x01 */
    U16                         Reserved1;                  /* 0x02 */
    U32                         VolumeCreationFlags;        /* 0x04 */
    U32                         VolumeSettings;             /* 0x08 */
    U8                          Reserved2;                  /* 0x0C */
    U8                          ResyncRate;                 /* 0x0D */
    U16                         DataScrubDuration;          /* 0x0E */
    U64                         VolumeMaxLBA;               /* 0x10 */
    U32                         StripeSize;                 /* 0x18 */
    U8                          Name[16];                   /* 0x1C */
    MPI2_RAID_VOLUME_PHYSDISK   PhysDisk[MPI2_RAID_VOL_CREATION_NUM_PHYSDISKS];/
```

```
} MPI2_RAID_VOLUME_CREATION_STRUCT,
  MPI2_POINTER PTR_MPI2_RAID_VOLUME_CREATION_STRUCT,
  Mpi2RaidVolumeCreationStruct_t, MPI2_POINTER pMpi2RaidVolumeCreationStruct_t;

/* use MPI2_RAID_VOL_TYPE_ defines from mpi2_cnfg.h for VolumeType */

/* defines for the VolumeCreationFlags field */
#define MPI2_RAID_VOL_CREATION_DEFAULT_SETTINGS     (0x80000000)
#define MPI2_RAID_VOL_CREATION_BACKGROUND_INIT      (0x00000004)
#define MPI2_RAID_VOL_CREATION_LOW_LEVEL_INIT       (0x00000002)
#define MPI2_RAID_VOL_CREATION_MIGRATE_DATA         (0x00000001)
/* The following is an obsolete define.
 * It must be shifted left 24 bits in order to set the proper bit.
 */
#define MPI2_RAID_VOL_CREATION_USE_DEFAULT_SETTINGS (0x80)


/* RAID Online Capacity Expansion Structure */

typedef struct _MPI2_RAID_ONLINE_CAPACITY_EXPANSION
{
    U32                     Flags;                          /* 0x00 */
    U16                     DevHandle0;                     /* 0x04 */
    U16                     Reserved1;                      /* 0x06 */
    U16                     DevHandle1;                     /* 0x08 */
    U16                     Reserved2;                      /* 0x0A */
} MPI2_RAID_ONLINE_CAPACITY_EXPANSION,
  MPI2_POINTER PTR_MPI2_RAID_ONLINE_CAPACITY_EXPANSION,
  Mpi2RaidOnlineCapacityExpansion_t,
  MPI2_POINTER pMpi2RaidOnlineCapacityExpansion_t;


/* RAID Volume Indicator Structure */

typedef struct _MPI2_RAID_VOL_INDICATOR
{
    U64                     TotalBlocks;                    /* 0x00 */
    U64                     BlocksRemaining;                /* 0x08 */
    U32                     Flags;                          /* 0x10 */
} MPI2_RAID_VOL_INDICATOR, MPI2_POINTER PTR_MPI2_RAID_VOL_INDICATOR,
  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)


/* RAID Action Reply ActionData union */
typedef union _MPI2_RAID_ACTION_REPLY_DATA
{
    U32                     Word[5];
    MPI2_RAID_VOL_INDICATOR RaidVolumeIndicator;
    U16                     VolDevHandle;
    U8                      VolumeState;
    U8                      PhysDiskNum;
} MPI2_RAID_ACTION_REPLY_DATA, MPI2_POINTER PTR_MPI2_RAID_ACTION_REPLY_DATA,
  Mpi2RaidActionReplyData_t, MPI2_POINTER pMpi2RaidActionReplyData_t;

/* use MPI2_RAIDVOL0_SETTING_ defines from mpi2_cnfg.h for MPI2_RAID_ACTION_CHAN


/* RAID Action Reply Message */
```



```
typedef struct _MPI2_RAID_ACTION_REPLY
{
    U8                          Action;                     /* 0x00 */
    U8                          Reserved1;                  /* 0x01 */
    U8                          MsgLength;                  /* 0x02 */
    U8                          Function;                   /* 0x03 */
    U16                         VolDevHandle;               /* 0x04 */
    U8                          PhysDiskNum;                /* 0x06 */
    U8                          MsgFlags;                   /* 0x07 */
    U8                          VP_ID;                      /* 0x08 */
    U8                          VF_ID;                      /* 0x09 */
    U16                         Reserved2;                  /* 0x0A */
    U16                         Reserved3;                  /* 0x0C */
    U16                         IOCStatus;                  /* 0x0E */
    U32                         IOCLogInfo;                 /* 0x10 */
    MPI2_RAID_ACTION_REPLY_DATA ActionData;                 /* 0x14 */
} MPI2_RAID_ACTION_REPLY, MPI2_POINTER PTR_MPI2_RAID_ACTION_REPLY,
  Mpi2RaidActionReply_t, MPI2_POINTER pMpi2RaidActionReply_t;


#endif

#endif /* ! codereview */
```

```
**********************************************************
   16332 Thu Jun 12 17:28:23 2014
new/usr/src/uts/common/sys/scsi/adapters/mpt_sas3/mpi/mpi2_sas.h
4546 mpt_sas needs enhancing to support LSI MPI2.5
**********************************************************
/*
 * 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.
 *
 * 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.
 *
 *  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.
 *  --------------------------------------------------------------------------
 */

#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                      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_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 */


/* 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                      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_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 */


/* 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 */
    U32                     IOCLogInfo;         /* 0x10 */
    U8                      StatusFIS[20];      /* 0x14 */
    U32                     StatusControlRegisters; /* 0x28 */
    U32                     TransferCount;      /* 0x2C */
} MPI2_SATA_PASSTHROUGH_REPLY, MPI2_POINTER PTR_MPI2_SATA_PASSTHROUGH_REPLY,
  Mpi2SataPassthroughReply_t, MPI2_POINTER pMpi2SataPassthroughReply_t;

/* values for SASStatus field are at the top of this file */


/*****************************************************************************
*  SAS IO Unit Control messages
*****************************************************************************/
```

```
/* SAS IO Unit Control Request Message */
typedef struct _MPI2_SAS_IOUNIT_CONTROL_REQUEST
{
    U8                      Operation;          /* 0x00 */
    U8                      Reserved1;          /* 0x01 */
    U8                      ChainOffset;        /* 0x02 */
    U8                      Function;           /* 0x03 */
    U16                     DevHandle;          /* 0x04 */
    U8                      IOCParameter;       /* 0x06 */
    U8                      MsgFlags;           /* 0x07 */
    U8                      VP_ID;              /* 0x08 */
    U8                      VF_ID;              /* 0x09 */
    U16                     Reserved3;          /* 0x0A */
    U16                     Reserved4;          /* 0x0C */
    U8                      PhyNum;             /* 0x0E */
    U8                      PrimFlags;          /* 0x0F */
    U32                     Primitive;          /* 0x10 */
    U8                      LookupMethod;       /* 0x14 */
    U8                      Reserved5;          /* 0x15 */
    U16                     SlotNumber;         /* 0x16 */
    U64                     LookupAddress;      /* 0x18 */
    U32                     IOCParameterValue;  /* 0x20 */
    U32                     Reserved7;          /* 0x24 */
    U32                     Reserved8;          /* 0x28 */
} MPI2_SAS_IOUNIT_CONTROL_REQUEST,
  MPI2_POINTER PTR_MPI2_SAS_IOUNIT_CONTROL_REQUEST,
  Mpi2SasIoUnitControlRequest_t, MPI2_POINTER pMpi2SasIoUnitControlRequest_t;

/* values for the Operation field */
#define MPI2_SAS_OP_CLEAR_ALL_PERSISTENT        (0x02)
#define MPI2_SAS_OP_PHY_LINK_RESET              (0x06)
#define MPI2_SAS_OP_PHY_HARD_RESET              (0x07)
#define MPI2_SAS_OP_PHY_CLEAR_ERROR_LOG         (0x08)
#define MPI2_SAS_OP_SEND_PRIMITIVE              (0x0A)
#define MPI2_SAS_OP_FORCE_FULL_DISCOVERY        (0x0B)
#define MPI2_SAS_OP_TRANSMIT_PORT_SELECT_SIGNAL (0x0C)
#define MPI2_SAS_OP_REMOVE_DEVICE               (0x0D)
#define MPI2_SAS_OP_LOOKUP_MAPPING              (0x0E)
#define MPI2_SAS_OP_SET_IOC_PARAMETER           (0x0F)
#define MPI25_SAS_OP_ENABLE_FP_DEVICE           (0x10)
#define MPI25_SAS_OP_DISABLE_FP_DEVICE          (0x11)
#define MPI25_SAS_OP_ENABLE_FP_ALL              (0x12)
#define MPI25_SAS_OP_DISABLE_FP_ALL             (0x13)
#define MPI2_SAS_OP_DEV_ENABLE_NCQ              (0x14)
#define MPI2_SAS_OP_DEV_DISABLE_NCQ             (0x15)
#define MPI2_SAS_OP_PRODUCT_SPECIFIC_MIN        (0x80)

/* values for the PrimFlags field */
#define MPI2_SAS_PRIMFLAGS_SINGLE               (0x08)
#define MPI2_SAS_PRIMFLAGS_TRIPLE               (0x02)
#define MPI2_SAS_PRIMFLAGS_REDUNDANT            (0x01)

/* values for the LookupMethod field */
#define MPI2_SAS_LOOKUP_METHOD_SAS_ADDRESS          (0x01)
#define MPI2_SAS_LOOKUP_METHOD_SAS_ENCLOSURE_SLOT   (0x02)
#define MPI2_SAS_LOOKUP_METHOD_SAS_DEVICE_NAME      (0x03)


/* SAS IO Unit Control Reply Message */
typedef struct _MPI2_SAS_IOUNIT_CONTROL_REPLY
{
    U8                      Operation;          /* 0x00 */
    U8                      Reserved1;          /* 0x01 */
    U8                      MsgLength;          /* 0x02 */
    U8                      Function;           /* 0x03 */
    U16                     DevHandle;          /* 0x04 */
```



```
    U8                      IOCParameter;       /* 0x06 */
    U8                      MsgFlags;           /* 0x07 */
    U8                      VP_ID;              /* 0x08 */
    U8                      VF_ID;              /* 0x09 */
    U16                     Reserved3;          /* 0x0A */
    U16                     Reserved4;          /* 0x0C */
    U16                     IOCStatus;          /* 0x0E */
    U32                     IOCLogInfo;         /* 0x10 */
} MPI2_SAS_IOUNIT_CONTROL_REPLY,
  MPI2_POINTER PTR_MPI2_SAS_IOUNIT_CONTROL_REPLY,
  Mpi2SasIoUnitControlReply_t, MPI2_POINTER pMpi2SasIoUnitControlReply_t;


#endif


#endif /* ! codereview */
```

```
**********************************************************
   24463 Thu Jun 12 17:28:23 2014
new/usr/src/uts/common/sys/scsi/adapters/mpt_sas3/mpi/mpi2_tool.h
4546 mpt_sas needs enhancing to support LSI MPI2.5
**********************************************************
/*
 * 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.
 *
 * 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
 *
 *  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.
 *  ---------------------------------------------------------------------------
 */

#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)
#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;


/*****************************************************************************
*  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 */


/* 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,
  MPI2_POINTER PTR_MPI2_TOOLBOX_DIAG_CLI_REPLY,
  Mpi2ToolboxDiagnosticCliReply_t,
  MPI2_POINTER pMpi2ToolboxDiagnosticCliReply_t;


/*****************************************************************************
*
*       Diagnostic Buffer Messages
*
*****************************************************************************/


/*****************************************************************************
*  Diagnostic Buffer Post request
*****************************************************************************/

typedef struct _MPI2_DIAG_BUFFER_POST_REQUEST
{
    U8                      ExtendedType;               /* 0x00 */
    U8                      BufferType;                 /* 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 */
    U64                     BufferAddress;              /* 0x0C */
    U32                     BufferLength;               /* 0x14 */
```

```
    U32                     Reserved5;                  /* 0x18 */
    U32                     Reserved6;                  /* 0x1C */
    U32                     Flags;                      /* 0x20 */
    U32                     ProductSpecific[23];        /* 0x24 */
} MPI2_DIAG_BUFFER_POST_REQUEST, MPI2_POINTER PTR_MPI2_DIAG_BUFFER_POST_REQUEST,
  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)

/******************************************************************************
*  Diagnostic Buffer Post reply
******************************************************************************/

typedef struct _MPI2_DIAG_BUFFER_POST_REPLY
{
    U8                      ExtendedType;               /* 0x00 */
    U8                      BufferType;                 /* 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                     TransferLength;             /* 0x14 */
} MPI2_DIAG_BUFFER_POST_REPLY, MPI2_POINTER PTR_MPI2_DIAG_BUFFER_POST_REPLY,
  Mpi2DiagBufferPostReply_t, MPI2_POINTER pMpi2DiagBufferPostReply_t;


/******************************************************************************
*  Diagnostic Release request
******************************************************************************/

typedef struct _MPI2_DIAG_RELEASE_REQUEST
{
    U8                      Reserved1;                  /* 0x00 */
    U8                      BufferType;                 /* 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_DIAG_RELEASE_REQUEST, MPI2_POINTER PTR_MPI2_DIAG_RELEASE_REQUEST,
  Mpi2DiagReleaseRequest_t, MPI2_POINTER pMpi2DiagReleaseRequest_t;


/******************************************************************************
```



```
*  Diagnostic Buffer Post reply
******************************************************************************/

typedef struct _MPI2_DIAG_RELEASE_REPLY
{
    U8                      Reserved1;                  /* 0x00 */
    U8                      BufferType;                 /* 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_DIAG_RELEASE_REPLY, MPI2_POINTER PTR_MPI2_DIAG_RELEASE_REPLY,
  Mpi2DiagReleaseReply_t, MPI2_POINTER pMpi2DiagReleaseReply_t;


#endif

#endif /* ! codereview */
```

```
**********************************************************
    4215 Thu Jun 12 17:28:23 2014
new/usr/src/uts/common/sys/scsi/adapters/mpt_sas3/mpi/mpi2_type.h
4546 mpt_sas needs enhancing to support LSI MPI2.5
**********************************************************
/*
 * 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.
 *
 * 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;

typedef struct _U64
{
    U32          Low;
    U32          High;
} U64;


/******************************************************************************
*
*               Pointer Types
*
******************************************************************************/

typedef S8      *PS8;
typedef U8      *PU8;
typedef S16     *PS16;
typedef U16     *PU16;
```

```
typedef S32     *PS32;
typedef U32     *PU32;
typedef S64     *PS64;
typedef U64     *PU64;

#endif

#endif

#endif /* ! codereview */
```

```
**********************************************************
    1986 Thu Jun 12 17:28:23 2014
new/usr/src/uts/common/sys/scsi/adapters/mpt_sas3/mptsas3_hash.h
4546 mpt_sas needs enhancing to support LSI MPI2.5
**********************************************************
/*
 * 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.
 */

#ifndef _SYS_SCSI_ADAPTERS_MPTSAS3_HASH_H
#define _SYS_SCSI_ADAPTERS_MPTSAS3_HASH_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;

typedef uint64_t (*refhash_hash_f)(const void *);
typedef int (*refhash_cmp_f)(const void *, const void *);
typedef void (*refhash_dtor_f)(void *);
typedef int (*refhash_eval_f)(const void *, void *);

typedef struct refhash {
        list_t *rh_buckets;
        uint_t rh_bucket_count;
        list_t rh_objs;
        size_t rh_obj_size;     /* used by mdb */
        size_t rh_link_off;
        size_t rh_tag_off;
        refhash_hash_f rh_hash;
        refhash_cmp_f rh_cmp;
        refhash_dtor_f rh_dtor;
} refhash_t;

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 /* ! codereview */
```

```
**********************************************************
    9415 Thu Jun 12 17:28:23 2014
new/usr/src/uts/common/sys/scsi/adapters/mpt_sas3/mptsas3_ioctl.h
4546 mpt_sas needs enhancing to support LSI MPI2.5
**********************************************************
/*
 * 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.
 */

/*
 * 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

#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
 *
 *  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
typedef struct mptsas_adapter_data
{
        uint32_t                StructureLength;
        uint32_t                AdapterType;
        uint32_t                MpiPortNumber;
        uint32_t                PCIDeviceHwId;
        uint32_t                PCIDeviceHwRev;
        uint32_t                SubSystemId;
        uint32_t                SubsystemVendorId;
        uint32_t                Reserved1;
        uint32_t                MpiFirmwareVersion;
        uint32_t                BiosVersion;
        uint8_t                 DriverVersion[32];
        uint8_t                 Reserved2;
```

```
        uint8_t                 ScsiId;
        uint16_t                Reserved3;
        mptsas_pci_bits_t       PciInformation;
} mptsas_adapter_data_t;


typedef struct mptsas_update_flash
{
        uint64_t        PtrBuffer;
        uint32_t        ImageChecksum;
        uint32_t        ImageOffset;
        uint32_t        ImageSize;
        uint32_t        ImageType;
} mptsas_update_flash_t;


#define MPTSAS_PASS_THRU_DIRECTION_NONE         0
#define MPTSAS_PASS_THRU_DIRECTION_READ         1
#define MPTSAS_PASS_THRU_DIRECTION_WRITE        2
#define MPTSAS_PASS_THRU_DIRECTION_BOTH         3

typedef struct mptsas_pass_thru
{
        uint64_t        PtrRequest;
        uint64_t        PtrReply;
        uint64_t        PtrData;
        uint32_t        RequestSize;
        uint32_t        ReplySize;
        uint32_t        DataSize;
        uint32_t        DataDirection;
        uint64_t        PtrDataOut;
        uint32_t        DataOutSize;
        uint32_t        Timeout;
} mptsas_pass_thru_t;


/*
 * Event queue defines
 */
#define MPTSAS_EVENT_QUEUE_SIZE         (50) /* Max Events stored in driver */
#define MPTSAS_MAX_EVENT_DATA_LENGTH    (48) /* Size of each event in Dwords */

typedef struct mptsas_event_query
{
        uint16_t        Entries;
        uint16_t        Reserved;
        uint32_t        Types[4];
} mptsas_event_query_t;

typedef struct mptsas_event_enable
{
        uint32_t        Types[4];
} mptsas_event_enable_t;

/*
 * Event record entry for ioctl.
 */
typedef struct mptsas_event_entry
{
        uint32_t        Type;
        uint32_t        Number;
        uint32_t        Data[MPTSAS_MAX_EVENT_DATA_LENGTH];
} mptsas_event_entry_t;

typedef struct mptsas_event_report
{
```



```
        uint32_t                Size;
        mptsas_event_entry_t    Events[1];
} mptsas_event_report_t;


typedef struct mptsas_pci_info
{
        uint32_t        BusNumber;
        uint8_t         DeviceNumber;
        uint8_t         FunctionNumber;
        uint16_t        InterruptVector;
        uint8_t         PciHeader[256];
} mptsas_pci_info_t;


typedef struct mptsas_diag_action
{
        uint32_t        Action;
        uint32_t        Length;
        uint64_t        PtrDiagAction;
        uint32_t        ReturnCode;
} mptsas_diag_action_t;

#define MPTSAS_FW_DIAGNOSTIC_UID_NOT_FOUND      (0xFF)

#define MPTSAS_FW_DIAG_NEW                      (0x806E6577)

#define MPTSAS_FW_DIAG_TYPE_REGISTER            (0x00000001)
#define MPTSAS_FW_DIAG_TYPE_UNREGISTER          (0x00000002)
#define MPTSAS_FW_DIAG_TYPE_QUERY               (0x00000003)
#define MPTSAS_FW_DIAG_TYPE_READ_BUFFER         (0x00000004)
#define MPTSAS_FW_DIAG_TYPE_RELEASE             (0x00000005)

#define MPTSAS_FW_DIAG_INVALID_UID              (0x00000000)

#define MPTSAS_FW_DIAG_ERROR_SUCCESS            (0x00000000)
#define MPTSAS_FW_DIAG_ERROR_FAILURE            (0x00000001)
#define MPTSAS_FW_DIAG_ERROR_INVALID_PARAMETER  (0x00000002)
#define MPTSAS_FW_DIAG_ERROR_POST_FAILED        (0x00000010)
#define MPTSAS_FW_DIAG_ERROR_INVALID_UID        (0x00000011)
#define MPTSAS_FW_DIAG_ERROR_RELEASE_FAILED     (0x00000012)
#define MPTSAS_FW_DIAG_ERROR_NO_BUFFER          (0x00000013)
#define MPTSAS_FW_DIAG_ERROR_ALREADY_RELEASED   (0x00000014)


typedef struct mptsas_fw_diag_register
{
        uint8_t         ExtendedType;
        uint8_t         BufferType;
        uint16_t        ApplicationFlags;
        uint32_t        DiagnosticFlags;
        uint32_t        ProductSpecific[23];
        uint32_t        RequestedBufferSize;
        uint32_t        UniqueId;
} mptsas_fw_diag_register_t;

typedef struct mptsas_fw_diag_unregister
{
        uint32_t        UniqueId;
} mptsas_fw_diag_unregister_t;

#define MPTSAS_FW_DIAG_FLAG_APP_OWNED           (0x0001)
#define MPTSAS_FW_DIAG_FLAG_BUFFER_VALID        (0x0002)
#define MPTSAS_FW_DIAG_FLAG_FW_BUFFER_ACCESS    (0x0004)

typedef struct mptsas_fw_diag_query
```

```
{
        uint8_t         ExtendedType;
        uint8_t         BufferType;
        uint16_t        ApplicationFlags;
        uint32_t        DiagnosticFlags;
        uint32_t        ProductSpecific[23];
        uint32_t        TotalBufferSize;
        uint32_t        DriverAddedBufferSize;
        uint32_t        UniqueId;
} mptsas_fw_diag_query_t;

typedef struct mptsas_fw_diag_release
{
        uint32_t        UniqueId;
} mptsas_fw_diag_release_t;

#define MPTSAS_FW_DIAG_FLAG_REREGISTER          (0x0001)
#define MPTSAS_FW_DIAG_FLAG_FORCE_RELEASE       (0x0002)

typedef struct mptsas_diag_read_buffer
{
        uint8_t         Status;
        uint8_t         Reserved;
        uint16_t        Flags;
        uint32_t        StartingOffset;
        uint32_t        BytesToRead;
        uint32_t        UniqueId;
        uint32_t        DataBuffer[1];
} mptsas_diag_read_buffer_t;

/*
 * Register Access
 */
#define REG_IO_READ     1
#define REG_IO_WRITE    2
#define REG_MEM_READ    3
#define REG_MEM_WRITE   4

typedef struct mptsas_reg_access
{
        uint32_t        Command;
        uint32_t        RegOffset;
        uint32_t        RegData;
} mptsas_reg_access_t;

/*
 * Disk Toplogy Information
 */
typedef struct mptsas_disk_info
{
        uint64_t        SasAddress;
        uint16_t        Instance;
        uint16_t        Enclosure;
        uint16_t        Slot;
} mptsas_disk_info_t;

typedef struct mptsas_get_disk_info
{
        uint16_t                DiskCount;
        mptsas_disk_info_t      *PtrDiskInfoArray;
        uint64_t                DiskInfoArraySize;
} mptsas_get_disk_info_t;

#ifdef _KERNEL

typedef struct mptsas_get_disk_info32
```



```
{
        uint16_t                DiskCount;
        caddr32_t               PtrDiskInfoArray;
        uint64_t                DiskInfoArraySize;
} mptsas_get_disk_info32_t;

#endif /* _KERNEL */

/*
 * LED Control
 */

typedef struct mptsas_led_control
{
        uint8_t         Command;
        uint16_t        Enclosure;
        uint16_t        Slot;
        uint8_t         Led;
        uint8_t         LedStatus;
} mptsas_led_control_t;

#define MPTSAS_LEDCTL_FLAG_SET          1
#define MPTSAS_LEDCTL_FLAG_GET          2

#define MPTSAS_LEDCTL_LED_IDENT         1
#define MPTSAS_LEDCTL_LED_FAIL          2
#define MPTSAS_LEDCTL_LED_OK2RM         3

#define MPTSAS_LEDCTL_LED_MIN           MPTSAS_LEDCTL_LED_IDENT
#define MPTSAS_LEDCTL_LED_MAX           MPTSAS_LEDCTL_LED_OK2RM

#ifdef  __cplusplus
}
#endif

#endif  /* _MPTSAS3_IOCTL_H */
#endif /* ! codereview */
```

```
**********************************************************
    2958 Thu Jun 12 17:28:24 2014
new/usr/src/uts/common/sys/scsi/adapters/mpt_sas3/mptsas3_smhba.h
4546 mpt_sas needs enhancing to support LSI MPI2.5
**********************************************************
/*
 * 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.
 */

/*
 * SM-HBA interfaces/definitions for MPT SAS driver.
 */

#ifndef _MPTSAS3_SMHBA_H
#define _MPTSAS3_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>

#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 /* ! codereview */
```

```
**********************************************************
   46778 Thu Jun 12 17:28:24 2014
new/usr/src/uts/common/sys/scsi/adapters/mpt_sas3/mptsas3_var.h
4546 mpt_sas needs enhancing to support LSI MPI2.5
**********************************************************
/*
 * 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 (c) 2013, Joyent, Inc. All rights reserved.
 * Copyright (c) 2014, Tegile Systems Inc. All rights reserved.
 */

/*
 * 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

#include <sys/byteorder.h>
#include <sys/queue.h>
#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>

#ifdef  __cplusplus
extern "C" {
#endif

/*
 * Compile options
 */
#ifdef DEBUG
#define MPTSAS_DEBUG            /* turn on debugging code */
#endif  /* DEBUG */

#define MPTSAS_INITIAL_SOFT_SPACE       4

#define MAX_MPI_PORTS           16

/*
 * Note below macro definition and data type definition
 * are used for phy mask handling, it should be changed
 * simultaneously.
 */
#define MPTSAS_MAX_PHYS         16
typedef uint16_t                mptsas_phymask_t;

#define MPTSAS_INVALID_DEVHDL   0xffff
#define MPTSAS_SATA_GUID        "sata-guid"

/*
 * Hash table sizes for SMP targets (i.e., expanders) and ordinary SSP/STP
 * targets.  There's no need to go overboard here, as the ordinary paths for
 * I/O do not normally require hashed target lookups.  These should be good
 * enough and then some for any fabric within the hardware's capabilities.
 */
#define MPTSAS_SMP_BUCKET_COUNT         23
#define MPTSAS_TARGET_BUCKET_COUNT      97

/*
 * MPT HW defines
 */
#define MPTSAS_MAX_DISKS_IN_CONFIG      14
#define MPTSAS_MAX_DISKS_IN_VOL         10
#define MPTSAS_MAX_HOTSPARES            2
#define MPTSAS_MAX_RAIDVOLS             2
#define MPTSAS_MAX_RAIDCONFIGS          5

/*
 * 64-bit SAS WWN is displayed as 16 characters as HEX characters,
 * plus two means the prefix 'w' and end of the string '\0'.
 */
#define MPTSAS_WWN_STRLEN       (16 + 2)
#define MPTSAS_MAX_GUID_LEN     64

/*
 * DMA routine flags
 */
#define MPTSAS_DMA_HANDLE_ALLOCD        0x2
#define MPTSAS_DMA_MEMORY_ALLOCD        0x4
#define MPTSAS_DMA_HANDLE_BOUND 0x8

/*
```

```
 * If the HBA supports DMA or bus-mastering, you may have your own
 * scatter-gather list for physically non-contiguous memory in one
 * I/O operation; if so, there's probably a size for that list.
 * It must be placed in the ddi_dma_lim_t structure, so that the system
 * DMA-support routines can use it to break up the I/O request, so we
 * define it here.
 */
#if defined(__sparc)
#define MPTSAS_MAX_DMA_SEGS     1
#define MPTSAS_MAX_CMD_SEGS     1
#else
#define MPTSAS_MAX_DMA_SEGS     256
#define MPTSAS_MAX_CMD_SEGS     257
#endif
#define MPTSAS_MAX_FRAME_SGES(mpt) \
        (((mpt->m_req_frame_size - (sizeof (MPI2_SCSI_IO_REQUEST))) / 8) + 1)

/*
 * Calculating how many 64-bit DMA simple elements can be stored in the first
 * frame. Note that msg_scsi_io_request contains 2 double-words (8 bytes) for
 * element storage.  And 64-bit dma element is 3 double-words (12 bytes) in
 * size. IEEE 64-bit dma element used for SAS3 controllers is 4 double-words
 * (16 bytes).
 */
#define MPTSAS_MAX_FRAME_SGES64(mpt) \
        ((mpt->m_req_frame_size - \
        sizeof (MPI2_SCSI_IO_REQUEST) + sizeof (MPI2_SGE_IO_UNION)) / \
        (mpt->m_MPI25 ? sizeof (MPI2_IEEE_SGE_SIMPLE64) : \
        sizeof (MPI2_SGE_SIMPLE64)))

/*
 * Scatter-gather list structure defined by HBA hardware
 */
typedef struct NcrTableIndirect {       /* Table Indirect entries */
        uint32_t count;         /* 24 bit count */
        union {
                uint32_t address32;     /* 32 bit address */
                struct {
                        uint32_t Low;
                        uint32_t High;
                } address64;            /* 64 bit address */
        } addr;
} mptti_t;

/*
 * preferred pkt_private length in 64-bit quantities
 */
#ifdef  _LP64
#define PKT_PRIV_SIZE   2
#define PKT_PRIV_LEN    16      /* in bytes */
#else /* _ILP32 */
#define PKT_PRIV_SIZE   1
#define PKT_PRIV_LEN    8       /* in bytes */
#endif

#define PKT2CMD(pkt)    ((struct mptsas_cmd *)((pkt)->pkt_ha_private))
#define CMD2PKT(cmdp)   ((struct scsi_pkt *)((cmdp)->cmd_pkt))
#define EXTCMDS_STATUS_SIZE (sizeof (struct scsi_arq_status))

/*
 * get offset of item in structure
 */
#define MPTSAS_GET_ITEM_OFF(type, member) ((size_t)(&((type *)0)->member))

/*
 * WWID provided by LSI firmware is generated by firmware but the WWID is not
```



```
 * IEEE NAA standard format, OBP has no chance to distinguish format of unit
 * address. According LSI's confirmation, the top nibble of RAID WWID is
 * meanless, so the consensus between Solaris and OBP is to replace top nibble
 * of WWID provided by LSI to "3" always to hint OBP that this is a RAID WWID
 * format unit address.
 */
#define MPTSAS_RAID_WWID(wwid) \
        ((wwid & 0x0FFFFFFFFFFFFFFF) | 0x3000000000000000)

typedef struct mptsas_target_addr {
        uint64_t mta_wwn;
        mptsas_phymask_t mta_phymask;
} mptsas_target_addr_t;

TAILQ_HEAD(mptsas_active_cmdq, mptsas_cmd);
typedef struct mptsas_active_cmdq mptsas_active_cmdq_t;

typedef struct mptsas_target {
                kmutex_t                m_t_mutex;
                mptsas_target_addr_t    m_addr;
                refhash_link_t          m_link;
                uint16_t                m_devhdl;
                uint32_t                m_deviceinfo;
                uint32_t                m_dups;
                uint8_t                 m_phynum;
                mptsas_active_cmdq_t    m_active_cmdq;
                int32_t                 m_t_throttle;
                int32_t                 m_t_ncmds;
                int32_t                 m_reset_delay;
                int32_t                 m_t_nwait;
                uint16_t                m_io_flags;
                uint16_t                m_enclosure;
                uint16_t                m_slot_num;
                uint16_t                m_qfull_retry_interval;
                uint8_t                 m_qfull_retries;
                uint8_t                 m_tgt_unconfigured;
                uint8_t                 m_led_status;
                uint8_t                 m_dr_flag;
} mptsas_target_t;

/*
 * If you change this structure, be sure that mptsas_smp_target_copy()
 * does the right thing.
 */
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;
} 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;

        int                     cmd_pkt_flags;

        /* pending expiration time for command in active slot */
        hrtime_t                cmd_active_expiration;
        TAILQ_ENTRY(mptsas_cmd) cmd_active_link;

        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 */
        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;

        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_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

#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
#define MPTSAS_SCSI_REPORTLUNS_ADDRESS_LOGICAL_UNIT             0x80
#define MPTSAS_SCSI_REPORTLUNS_ADDRESS_EXTENDED_UNIT            0xC0
#define MPTSAS_SCSI_REPORTLUNS_ADDRESS_LOGICAL_UNIT_2B          0x00
#define MPTSAS_SCSI_REPORTLUNS_ADDRESS_LOGICAL_UNIT_4B          0x01
#define MPTSAS_SCSI_REPORTLUNS_ADDRESS_LOGICAL_UNIT_6B          0x10
#define MPTSAS_SCSI_REPORTLUNS_ADDRESS_LOGICAL_UNIT_8B          0x20
#define MPTSAS_SCSI_REPORTLUNS_ADDRESS_LOGICAL_UNIT_SIZE                0x30

#define MPTSAS_HASH_ARRAY_SIZE  16
/*
 * hash table definition
 */

#define MPTSAS_HASH_FIRST       0xffff
#define MPTSAS_HASH_NEXT        0x0000

typedef struct mptsas_dma_alloc_state
{
        ddi_dma_handle_t        handle;
        caddr_t                 memp;
        size_t                  size;
        ddi_acc_handle_t        accessp;
        ddi_dma_cookie_t        cookie;
} mptsas_dma_alloc_state_t;

/*
 * passthrough request structure
 */
typedef struct mptsas_pt_request {
        uint8_t *request;
        uint32_t request_size;
        uint32_t data_size;
        uint32_t dataout_size;
        uint8_t direction;
        uint8_t simple;
        uint16_t sgl_offset;
        ddi_dma_cookie_t data_cookie;
        ddi_dma_cookie_t dataout_cookie;
} mptsas_pt_request_t;

/*
 * config page request structure
```

```
 */
typedef struct mptsas_config_request {
        uint32_t        page_address;
        uint8_t         action;
        uint8_t         page_type;
        uint8_t         page_number;
        uint8_t         page_length;
        uint8_t         page_version;
        uint8_t         ext_page_type;
        uint16_t        ext_page_length;
} mptsas_config_request_t;

typedef struct mptsas_fw_diagnostic_buffer {
        mptsas_dma_alloc_state_t        buffer_data;
        uint8_t                         extended_type;
        uint8_t                         buffer_type;
        uint8_t                         force_release;
        uint32_t                        product_specific[23];
        uint8_t                         immediate;
        uint8_t                         enabled;
        uint8_t                         valid_data;
        uint8_t                         owned_by_firmware;
        uint32_t                        unique_id;
} mptsas_fw_diagnostic_buffer_t;

/*
 * FW diag request structure
 */
typedef struct mptsas_diag_request {
        mptsas_fw_diagnostic_buffer_t   *pBuffer;
        uint8_t                         function;
} mptsas_diag_request_t;

typedef struct mptsas_hash_node {
        void *data;
        struct mptsas_hash_node *next;
} mptsas_hash_node_t;

typedef struct mptsas_hash_table {
        struct mptsas_hash_node *head[MPTSAS_HASH_ARRAY_SIZE];
        /*
         * last position in traverse
         */
        struct mptsas_hash_node *cur;
        uint16_t line;

} mptsas_hash_table_t;

/*
 * RAID volume information
 */
typedef struct mptsas_raidvol {
        ushort_t        m_israid;
        uint16_t        m_raidhandle;
        uint64_t        m_raidwwid;
        uint8_t         m_state;
        uint32_t        m_statusflags;
        uint32_t        m_settings;
        uint16_t        m_devhdl[MPTSAS_MAX_DISKS_IN_VOL];
        uint8_t         m_disknum[MPTSAS_MAX_DISKS_IN_VOL];
        ushort_t        m_diskstatus[MPTSAS_MAX_DISKS_IN_VOL];
        uint64_t        m_raidsize;
        int             m_raidlevel;
        int             m_ndisks;
        mptsas_target_t *m_raidtgt;
} mptsas_raidvol_t;
```



```
/*
 * RAID configurations
 */
typedef struct mptsas_raidconfig {
                mptsas_raidvol_t        m_raidvol[MPTSAS_MAX_RAIDVOLS];
                uint16_t                m_physdisk_devhdl[
                                            MPTSAS_MAX_DISKS_IN_CONFIG];
                uint8_t                 m_native;
} m_raidconfig_t;

/*
 * Track outstanding commands.  The index into the m_slot array is the SMID
 * (system message ID) of the outstanding command.  SMID 0 is reserved by the
 * software/firmware protocol and is never used for any command we generate;
 * as such, the assertion m_slot[0] == NULL is universally true.  The last
 * entry in the array is slot number MPTSAS_TM_SLOT(mpt) and is used ONLY for
 * task management commands.  No normal SCSI or ATA command will ever occupy
 * that slot.  Finally, the relationship m_slot[X]->cmd_slot == X holds at any
 * time that a consistent view of the target array is obtainable.
 *
 * As such, m_n_normal is the maximum number of slots available to ordinary
 * commands, and the relationship:
 * mpt->m_active->m_n_normal == mpt->m_max_requests - 2
 * always holds after initialisation.
 */
typedef struct mptsas_slots {
        size_t                  m_size;         /* size of struct, bytes */
        uint_t                  m_n_normal;     /* see above */
        uint_t                  m_rotor;        /* next slot idx to consider */
        mptsas_cmd_t            *m_slot[1];
} mptsas_slots_t;

/*
 * Structure to hold command and packets for event ack
 * and task management commands.
 */
typedef struct  m_event_struct {
        struct mptsas_cmd               m_event_cmd;
        struct m_event_struct   *m_event_linkp;
        /*
         * event member record the failure event and eventcntx
         * event member would be used in send ack pending process
         */
        uint32_t                m_event;
        uint32_t                m_eventcntx;
        uint_t                  in_use;
        struct scsi_pkt         m_event_pkt;    /* must be last */
                                                /* ... scsi_pkt_size() */
} m_event_struct_t;
#define M_EVENT_STRUCT_SIZE     (sizeof (m_event_struct_t) - \
                                sizeof (struct scsi_pkt) + scsi_pkt_size())

#define MAX_IOC_COMMANDS        8

/*
 * A pool of MAX_IOC_COMMANDS is maintained for event ack commands.
 * A new event ack command requests mptsas_cmd and scsi_pkt structures
 * from this pool, and returns it back when done.
 */

typedef struct m_replyh_arg {
        void *mpt;
        uint32_t rfm;
} m_replyh_arg_t;
_NOTE(DATA_READABLE_WITHOUT_LOCK(m_replyh_arg_t::mpt))
```

```
_NOTE(DATA_READABLE_WITHOUT_LOCK(m_replyh_arg_t::rfm))

/*
 * Flags for DR handler topology change
 */
#define MPTSAS_TOPO_FLAG_DIRECT_ATTACHED_DEVICE         0x0
#define MPTSAS_TOPO_FLAG_EXPANDER_ASSOCIATED            0x1
#define MPTSAS_TOPO_FLAG_LUN_ASSOCIATED                 0x2
#define MPTSAS_TOPO_FLAG_RAID_ASSOCIATED                0x4
#define MPTSAS_TOPO_FLAG_RAID_PHYSDRV_ASSOCIATED        0x8
#define MPTSAS_TOPO_FLAG_EXPANDER_ATTACHED_DEVICE       0x10

typedef struct mptsas_topo_change_list {
        void *mpt;
        uint_t  event;
        union {
                uint8_t physport;
                mptsas_phymask_t phymask;
        } un;
        uint16_t devhdl;
        void *object;
        uint8_t flags;
        struct mptsas_topo_change_list *next;
} mptsas_topo_change_list_t;


_NOTE(DATA_READABLE_WITHOUT_LOCK(mptsas_topo_change_list_t::mpt))
_NOTE(DATA_READABLE_WITHOUT_LOCK(mptsas_topo_change_list_t::event))
_NOTE(DATA_READABLE_WITHOUT_LOCK(mptsas_topo_change_list_t::physport))
_NOTE(DATA_READABLE_WITHOUT_LOCK(mptsas_topo_change_list_t::devhdl))
_NOTE(DATA_READABLE_WITHOUT_LOCK(mptsas_topo_change_list_t::object))
_NOTE(DATA_READABLE_WITHOUT_LOCK(mptsas_topo_change_list_t::flags))

/*
 * Status types when calling mptsas_get_target_device_info
 */
#define DEV_INFO_SUCCESS                0x0
#define DEV_INFO_FAIL_PAGE0             0x1
#define DEV_INFO_WRONG_DEVICE_TYPE      0x2
#define DEV_INFO_PHYS_DISK              0x3
#define DEV_INFO_FAIL_ALLOC             0x4

/*
 * mpt hotplug event defines
 */
#define MPTSAS_DR_EVENT_RECONFIG_TARGET 0x01
#define MPTSAS_DR_EVENT_OFFLINE_TARGET  0x02
#define MPTSAS_DR_EVENT_REMOVE_HANDLE   0x04

/*
 * SMP target hotplug events
 */
#define MPTSAS_DR_EVENT_RECONFIG_SMP    0x10
#define MPTSAS_DR_EVENT_OFFLINE_SMP     0x20
#define MPTSAS_DR_EVENT_MASK            0x3F

/*
 * mpt hotplug status definition for m_dr_flag
 */

/*
 * MPTSAS_DR_INACTIVE
 *
 * The target is in a normal operating state.
 * No dynamic reconfiguration operation is in progress.
 */
```



```
#define MPTSAS_DR_INACTIVE                              0x0
/*
 * MPTSAS_DR_INTRANSITION
 *
 * The target is in a transition mode since
 * hotplug event happens and offline procedure has not
 * been finished
 */
#define MPTSAS_DR_INTRANSITION                  0x1

typedef struct mptsas_tgt_private {
        int t_lun;
        struct mptsas_target *t_private;
} mptsas_tgt_private_t;

/*
 * The following defines are used in mptsas_set_init_mode to track the current
 * state as we progress through reprogramming the HBA from target mode into
 * initiator mode.
 */

#define IOUC_READ_PAGE0         0x00000100
#define IOUC_READ_PAGE1         0x00000200
#define IOUC_WRITE_PAGE1        0x00000400
#define IOUC_DONE               0x00000800
#define DISCOVERY_IN_PROGRESS   MPI2_SASIOUNIT0_PORTFLAGS_DISCOVERY_IN_PROGRESS
#define AUTO_PORT_CONFIGURATION MPI2_SASIOUNIT0_PORTFLAGS_AUTO_PORT_CONFIG

/*
 * Last allocated slot is used for TM requests.  Since only m_max_requests
 * frames are allocated, the last SMID will be m_max_requests - 1.
 */
#define MPTSAS_SLOTS_SIZE(mpt) \
        (sizeof (struct mptsas_slots) + (sizeof (struct mptsas_cmd *) * \
                mpt->m_max_requests))
#define MPTSAS_TM_SLOT(mpt)     (mpt->m_max_requests - 1)

/*
 * Macro for phy_flags
 */

typedef struct smhba_info {
        kmutex_t        phy_mutex;
        uint8_t         phy_id;
        uint64_t        sas_addr;
        char            path[8];
        uint16_t        owner_devhdl;
        uint16_t        attached_devhdl;
        uint8_t         attached_phy_identify;
        uint32_t        attached_phy_info;
        uint8_t         programmed_link_rate;
        uint8_t         hw_link_rate;
        uint8_t         change_count;
        uint32_t        phy_info;
        uint8_t         negotiated_link_rate;
        uint8_t         port_num;
        kstat_t         *phy_stats;
        uint32_t        invalid_dword_count;
        uint32_t        running_disparity_error_count;
        uint32_t        loss_of_dword_sync_count;
        uint32_t        phy_reset_problem_count;
        void            *mpt;
} smhba_info_t;

typedef struct mptsas_phy_info {
        uint8_t                 port_num;
```

```
        uint8_t                 port_flags;
        uint16_t                ctrl_devhdl;
        uint32_t                phy_device_type;
        uint16_t                attached_devhdl;
        mptsas_phymask_t        phy_mask;
        smhba_info_t            smhba_info;
} mptsas_phy_info_t;


typedef struct mptsas_thread_arg {
        void            *mpt;
        uint32_t        t;
} mptsas_thread_arg_t;

typedef struct mptsas_done_list {
        mptsas_cmd_t            *dl_q;
        mptsas_cmd_t            **dl_tail;
        uint32_t                dl_len;
} mptsas_done_list_t;

#define MPTSAS_DONEQ_THREAD_ACTIVE      0x1
typedef struct mptsas_doneq_thread_list {
        mptsas_done_list_t      dlist;
        kthread_t               *threadp;
        kcondvar_t              cv;
        ushort_t                reserv1;
        uint32_t                reserv2;
        kmutex_t                mutex;
        uint32_t                flag;
        mptsas_thread_arg_t     arg;
} mptsas_doneq_thread_list_t;

typedef struct mptsas_reply_pqueue {
        kmutex_t                rpq_mutex;
        uint8_t                 rpq_num;
        caddr_t                 rpq_queue; /* Pointer to this queue base */
        uint32_t                rpq_index; /* Index of next replyq entry */
        uint32_t                rpq_ncmds; /* Number of outstanding commands */
        mptsas_done_list_t      rpq_dlist;
        uint32_t                rpq_intr_count;
        uint32_t                rpq_intr_unclaimed;
        uint32_t                rpq_intr_mutexbusy;
} mptsas_reply_pqueue_t;


typedef struct mptsas_tx_waitqueue {
        kmutex_t        txwq_mutex;
        kcondvar_t      txwq_cv;
        kcondvar_t      txwq_drain_cv;
        kthread_t       *txwq_threadp;
        mptsas_cmd_t    *txwq_cmdq;     /* TX cmd queue for active request */
        mptsas_cmd_t    **txwq_qtail;   /* tx_wait queue tail ptr */
        uint32_t        txwq_len;       /* TX queue length */
        mptsas_thread_arg_t     arg;
        uint8_t         txwq_active;    /* Thread active flag */
        uint8_t         txwq_draining;  /* TX queue draining flag */
        uint8_t         txwq_wdrain;
} mptsas_tx_waitqueue_t;

#define NUM_TX_WAITQ    2

typedef struct mptsas {
        int             m_instance;

        struct mptsas *m_next;
```



```
        scsi_hba_tran_t         *m_tran;
        smp_hba_tran_t          *m_smptran;
        kmutex_t                m_mutex;
        kmutex_t                m_passthru_mutex;
        kcondvar_t              m_cv;
        kcondvar_t              m_passthru_cv;
        kcondvar_t              m_fw_cv;
        kcondvar_t              m_config_cv;
        kcondvar_t              m_fw_diag_cv;
        dev_info_t              *m_dip;

        /*
         * soft state flags
         */
        uint_t          m_softstate;

        refhash_t       *m_targets;
        refhash_t       *m_smp_targets;

        m_raidconfig_t  m_raidconfig[MPTSAS_MAX_RAIDCONFIGS];
        uint8_t         m_num_raid_configs;
        uint8_t         m_pref_tx_waitq;

        struct mptsas_slots *m_active;  /* outstanding cmds */

        mptsas_cmd_t    *m_waitq;       /* cmd queue for active request */
        mptsas_cmd_t    **m_waitqtail;  /* wait queue tail ptr */
        mptsas_tx_waitqueue_t   m_tx_waitq[NUM_TX_WAITQ];
        uint16_t                m_txwq_thread_threshold;
        uint16_t                m_txwq_thread_n;
        uint8_t                 m_txwq_enabled;
        uint8_t                 m_txwq_allow_q_jumping;
        mptsas_done_list_t      m_dlist; /* List of completed commands */

        /*
         * variables for helper threads (fan-out interrupts)
         */
        mptsas_doneq_thread_list_t      *m_doneq_thread_id;
        uint16_t                m_doneq_thread_n;
        uint16_t                m_doneq_next_thread;
        uint32_t                m_doneq_thread_threshold;
        uint32_t                m_doneq_length_threshold;
        kcondvar_t              m_qthread_cv;
        kmutex_t                m_qthread_mutex;

        uint32_t                m_ncmds; /* number of outstanding commands */
        uint32_t                m_ncstarted; /* ncmds started per interval */
        uint32_t                m_lncstarted; /* record of last value */
        m_event_struct_t *m_ioc_event_cmdq;     /* cmd queue for ioc event */
        m_event_struct_t **m_ioc_event_cmdtail; /* ioc cmd queue tail */

        ddi_acc_handle_t m_datap;       /* operating regs data access handle */

        struct _MPI2_SYSTEM_INTERFACE_REGS      *m_reg;

        ushort_t        m_devid;        /* device id of chip. */
        uchar_t         m_revid;        /* revision of chip. */
        uint16_t        m_svid;         /* subsystem Vendor ID of chip */
        uint16_t        m_ssid;         /* subsystem Device ID of chip */

        uchar_t         m_sync_offset;  /* default offset for this chip. */

        timeout_id_t    m_quiesce_timeid;

        ddi_dma_handle_t m_dma_req_frame_hdl;
        ddi_acc_handle_t m_acc_req_frame_hdl;
```

```
        ddi_dma_handle_t m_dma_req_sense_hdl;
        ddi_acc_handle_t m_acc_req_sense_hdl;
        ddi_dma_handle_t m_dma_reply_frame_hdl;
        ddi_acc_handle_t m_acc_reply_frame_hdl;
        ddi_dma_handle_t m_dma_free_queue_hdl;
        ddi_acc_handle_t m_acc_free_queue_hdl;
        ddi_dma_handle_t m_dma_post_queue_hdl;
        ddi_acc_handle_t m_acc_post_queue_hdl;
        uint8_t          m_dma_flags;

        /*
         * list of reset notification requests
         */
        struct scsi_reset_notify_entry  *m_reset_notify_listf;

        /*
         * qfull handling
         */
        timeout_id_t    m_restart_cmd_timeid;

        /*
         * scsi reset delay per bus
         */
        uint_t          m_scsi_reset_delay;

        int             m_pm_idle_delay;

        uchar_t         m_polled_intr;  /* intr was polled. */
        uchar_t         m_suspended;    /* true if driver is suspended */

        struct kmem_cache *m_kmem_cache;
        struct kmem_cache *m_cache_frames;

        /*
         * hba options.
         */
        uint_t          m_options;

        int             m_power_level;  /* current power level */

        int             m_busy;         /* power management busy state */

        off_t           m_pmcsr_offset; /* PMCSR offset */

        ddi_acc_handle_t m_config_handle;

        ddi_dma_attr_t          m_io_dma_attr;  /* Used for data I/O */
        ddi_dma_attr_t          m_msg_dma_attr; /* Used for message frames */
        ddi_device_acc_attr_t   m_dev_acc_attr;
        ddi_device_acc_attr_t   m_reg_acc_attr;

        /*
         * request/reply variables
         */
        caddr_t         m_req_frame;
        uint64_t        m_req_frame_dma_addr;
        caddr_t         m_req_sense;
        caddr_t         m_extreq_sense;
        uint64_t        m_req_sense_dma_addr;
        caddr_t         m_reply_frame;
        uint64_t        m_reply_frame_dma_addr;
        caddr_t         m_free_queue;
        uint64_t        m_free_queue_dma_addr;
        caddr_t         m_post_queue;
        uint64_t        m_post_queue_dma_addr;
        struct map      *m_erqsense_map;
```



```
        mptsas_reply_pqueue_t   *m_rep_post_queues;

        m_replyh_arg_t *m_replyh_args;

        uint16_t        m_max_requests;
        uint16_t        m_req_frame_size;
        uint16_t        m_req_sense_size;

        /*
         * Max frames per request reported in IOC Facts
         */
        uint8_t         m_max_chain_depth;
        /*
         * Max frames per request which is used in reality. It’s adjusted
         * according DMA SG length attribute, and shall not exceed the
         * m_max_chain_depth.
         */
        uint8_t         m_max_request_frames;
        uint8_t         m_max_msix_vectors;
        uint8_t         m_reply_frame_size;
        uint8_t         m_post_reply_qcount;

        uint16_t        m_free_queue_depth;
        uint16_t        m_post_queue_depth;
        uint16_t        m_max_replies;
        uint32_t        m_free_index;
        uint32_t        m_ioc_capabilities;

        /*
         * Housekeeping.
         */
        uint64_t        m_interrupt_count;
        uint32_t        m_unclaimed_pm_interrupt_count;
        uint32_t        m_unclaimed_polled_interrupt_count;
        uint32_t        m_unclaimed_no_interrupt_count;
        uint32_t        m_unclaimed_nocmd_interrupt_count;

        /*
         * indicates if the firmware was upload by the driver
         * at boot time
         */
        ushort_t        m_fwupload;

        uint16_t        m_productid;

        /*
         * per instance data structures for dma memory resources for
         * MPI handshake protocol. only one handshake cmd can run at a time.
         */
        ddi_dma_handle_t        m_hshk_dma_hdl;
        ddi_acc_handle_t        m_hshk_acc_hdl;
        caddr_t                 m_hshk_memp;
        size_t                  m_hshk_dma_size;

        /* Firmware version on the card at boot time */
        uint32_t                m_fwversion;

        /* MSI specific fields */
        ddi_intr_handle_t       *m_htable;      /* For array of interrupts */
        int                     m_intr_type;    /* What type of interrupt */
        int                     m_intr_cnt;     /* # of intrs count returned */
        size_t                  m_intr_size;    /* Size of intr array */
        uint_t                  m_intr_pri;     /* Interrupt priority   */
        int                     m_intr_cap;     /* Interrupt capabilities */
        ddi_taskq_t             *m_event_taskq;
```

```
        /* SAS specific information */

        union {
                uint64_t        m_base_wwid;    /* Base WWID */
                struct {
#ifdef _BIG_ENDIAN
                        uint32_t        m_base_wwid_hi;
                        uint32_t        m_base_wwid_lo;
#else
                        uint32_t        m_base_wwid_lo;
                        uint32_t        m_base_wwid_hi;
#endif
                } sasaddr;
        } un;

        uint8_t                 m_num_phys;             /* # of PHYs */
        mptsas_phy_info_t       m_phy_info[MPTSAS_MAX_PHYS];
        uint8_t                 m_port_chng;    /* initiator port changes */
        MPI2_CONFIG_PAGE_MAN_0   m_MANU_page0;   /* Manufactor page 0 info */
        MPI2_CONFIG_PAGE_MAN_1   m_MANU_page1;   /* Manufactor page 1 info */

        /* FMA Capabilities */
        int                     m_fm_capabilities;
        ddi_taskq_t             *m_dr_taskq;
        int                     m_mpxio_enable;
        uint8_t                 m_done_traverse_dev;
        uint8_t                 m_done_traverse_smp;
        int                     m_diag_action_in_progress;
        uint16_t                m_dev_handle;
        uint16_t                m_smp_devhdl;

        /*
         * Event recording
         */
        uint8_t                 m_event_index;
        uint32_t                m_event_number;
        uint32_t                m_event_mask[4];
        mptsas_event_entry_t    m_events[MPTSAS_EVENT_QUEUE_SIZE];

        /*
         * FW diag Buffer List
         */
        mptsas_fw_diagnostic_buffer_t
                m_fw_diag_buffer_list[MPI2_DIAG_BUF_TYPE_COUNT];

        /* GEN3 support */
        uint8_t                 m_MPI25;

        /*
         * Event Replay flag (MUR support)
         */
        uint8_t                 m_event_replay;

        /*
         * IR Capable flag
         */
        uint8_t                 m_ir_capable;

        /*
         * Is HBA processing a diag reset?
         */
        uint8_t                 m_in_reset;

        /*
         * per instance cmd data structures for task management cmds
         */
```



```
        m_event_struct_t        m_event_task_mgmt;      /* must be last */
                                                        /* ... scsi_pkt_size */
} mptsas_t;
#define MPTSAS_SIZE     (sizeof (struct mptsas) - \
                        sizeof (struct scsi_pkt) + scsi_pkt_size())
/*
 * Only one of below two conditions is satisfied, we
 * think the target is associated to the iport and
 * allow call into mptsas_probe_lun().
 * 1. physicalsport == physport
 * 2. (phymask & (1 << physport)) == 0
 * The condition #2 is because LSI uses lowest PHY
 * number as the value of physical port when auto port
 * configuration.
 */
#define IS_SAME_PORT(physicalport, physport, phymask, dynamicport) \
        ((physicalport == physport) || (dynamicport && (phymask & \
        (1 << physport))))

_NOTE(MUTEX_PROTECTS_DATA(mptsas::m_mutex, mptsas))
_NOTE(SCHEME_PROTECTS_DATA("safe sharing", mptsas::m_next))
_NOTE(SCHEME_PROTECTS_DATA("stable data", mptsas::m_dip mptsas::m_tran))
_NOTE(SCHEME_PROTECTS_DATA("stable data", mptsas::m_kmem_cache))
_NOTE(DATA_READABLE_WITHOUT_LOCK(mptsas::m_io_dma_attr.dma_attr_sgllen))
_NOTE(DATA_READABLE_WITHOUT_LOCK(mptsas::m_devid))
_NOTE(DATA_READABLE_WITHOUT_LOCK(mptsas::m_productid))
_NOTE(DATA_READABLE_WITHOUT_LOCK(mptsas::m_mpxio_enable))
_NOTE(DATA_READABLE_WITHOUT_LOCK(mptsas::m_instance))

/*
 * These should eventually migrate into the mpt header files
 * that may become the /kernel/misc/mpt module...
 */
#define mptsas_init_std_hdr(hdl, mp, DevHandle, Lun, ChainOffset, Function) \
        mptsas_put_msg_DevHandle(hdl, mp, DevHandle); \
        mptsas_put_msg_ChainOffset(hdl, mp, ChainOffset); \
        mptsas_put_msg_Function(hdl, mp, Function); \
        mptsas_put_msg_Lun(hdl, mp, Lun)

#define mptsas_put_msg_DevHandle(hdl, mp, val) \
        ddi_put16(hdl, &(mp)->DevHandle, (val))
#define mptsas_put_msg_ChainOffset(hdl, mp, val) \
        ddi_put8(hdl, &(mp)->ChainOffset, (val))
#define mptsas_put_msg_Function(hdl, mp, val) \
        ddi_put8(hdl, &(mp)->Function, (val))
#define mptsas_put_msg_Lun(hdl, mp, val) \
        ddi_put8(hdl, &(mp)->LUN[1], (val))

#define mptsas_get_msg_Function(hdl, mp) \
        ddi_get8(hdl, &(mp)->Function)

#define mptsas_get_msg_MsgFlags(hdl, mp) \
        ddi_get8(hdl, &(mp)->MsgFlags)

#define MPTSAS_ENABLE_DRWE(hdl) \
        ddi_put32(hdl->m_datap, &hdl->m_reg->WriteSequence, \
                MPI2_WRSEQ_FLUSH_KEY_VALUE); \
        ddi_put32(hdl->m_datap, &hdl->m_reg->WriteSequence, \
                MPI2_WRSEQ_1ST_KEY_VALUE); \
        ddi_put32(hdl->m_datap, &hdl->m_reg->WriteSequence, \
                MPI2_WRSEQ_2ND_KEY_VALUE); \
        ddi_put32(hdl->m_datap, &hdl->m_reg->WriteSequence, \
                MPI2_WRSEQ_3RD_KEY_VALUE); \
        ddi_put32(hdl->m_datap, &hdl->m_reg->WriteSequence, \
                MPI2_WRSEQ_4TH_KEY_VALUE); \
        ddi_put32(hdl->m_datap, &hdl->m_reg->WriteSequence, \
```

```
                MPI2_WRSEQ_5TH_KEY_VALUE); \
        ddi_put32(hdl->m_datap, &hdl->m_reg->WriteSequence, \
                MPI2_WRSEQ_6TH_KEY_VALUE);

/*
 * m_options flags
 */
#define MPTSAS_OPT_PM           0x01    /* Power Management */
#define MPTSAS_OPT_MSI          0x02    /* PCI MSI Interrupts */
#define MPTSAS_OPT_MSI_X        0x04    /* PCI MSI-X Interrupts */

/*
 * m_softstate flags
 */
#define MPTSAS_SS_DRAINING              0x02
#define MPTSAS_SS_QUIESCED              0x04
#define MPTSAS_SS_MSG_UNIT_RESET        0x08
#define MPTSAS_DID_MSG_UNIT_RESET       0x10

/*
 * m_dma_flags (allocated).
 */
#define MPTSAS_REQ_FRAME        0x01
#define MPTSAS_REQ_SENSE        0x02
#define MPTSAS_REPLY_FRAME      0x04
#define MPTSAS_FREE_QUEUE       0x08
#define MPTSAS_POST_QUEUE       0x10

/*
 * regspec defines.
 */
#define CONFIG_SPACE    0       /* regset[0] - configuration space */
#define IO_SPACE        1       /* regset[1] - used for i/o mapped device */
#define MEM_SPACE       2       /* regset[2] - used for memory mapped device */
#define BASE_REG2       3       /* regset[3] - used for 875 scripts ram */

/*
 * Handy constants
 */
#define FALSE           0
#define TRUE            1
#define BLOCKED         2
#define UNDEFINED       -1
#define FAILED          -2

/*
 * power management.
 */
#define MPTSAS_POWER_ON(mpt) { \
        pci_config_put16(mpt->m_config_handle, mpt->m_pmcsr_offset, \
            PCI_PMCSR_D0); \
        delay(drv_usectohz(10000)); \
        (void) pci_restore_config_regs(mpt->m_dip); \
        mptsas_setup_cmd_reg(mpt); \
}

#define MPTSAS_POWER_OFF(mpt) { \
        (void) pci_save_config_regs(mpt->m_dip); \
        pci_config_put16(mpt->m_config_handle, mpt->m_pmcsr_offset, \
            PCI_PMCSR_D3HOT); \
        mpt->m_power_level = PM_LEVEL_D3; \
}

/*
 * inq_dtype:
 * Bits 5 through 7 are the Peripheral Device Qualifier
```



```
 * 001b: device not connected to the LUN
 * Bits 0 through 4 are the Peripheral Device Type
 * 1fh: Unknown or no device type
 *
 * Although the inquiry may return success, the following value
 * means no valid LUN connected.
 */
#define MPTSAS_VALID_LUN(sd_inq) \
        (((sd_inq->inq_dtype & 0xe0) != 0x20) && \
        ((sd_inq->inq_dtype & 0x1f) != 0x1f))

/*
 * Default is to have 10 retries on receiving QFULL status and
 * each retry to be after 100 ms.
 */
#define QFULL_RETRIES           10
#define QFULL_RETRY_INTERVAL    100

/*
 * Handy macros
 */
#define Tgt(sp) ((sp)->cmd_pkt->pkt_address.a_target)
#define Lun(sp) ((sp)->cmd_pkt->pkt_address.a_lun)

#define IS_HEX_DIGIT(n) (((n) >= '0' && (n) <= '9') || \
        ((n) >= 'a' && (n) <= 'f') || ((n) >= 'A' && (n) <= 'F'))

/*
 * poll time for mptsas_pollret() and mptsas_wait_intr()
 */
#define MPTSAS_POLL_TIME        30000   /* 30 seconds */

/*
 * default time for mptsas_do_passthru
 */
#define MPTSAS_PASS_THRU_TIME_DEFAULT   60      /* 60 seconds */

/*
 * macro to return the effective address of a given per-target field
 */
#define EFF_ADDR(start, offset)         ((start) + (offset))

#define SDEV2ADDR(devp)         (&((devp)->sd_address))
#define SDEV2TRAN(devp)         ((devp)->sd_address.a_hba_tran)
#define PKT2TRAN(pkt)           ((pkt)->pkt_address.a_hba_tran)
#define ADDR2TRAN(ap)           ((ap)->a_hba_tran)
#define DIP2TRAN(dip)           (ddi_get_driver_private(dip))


#define TRAN2MPT(hba)           ((mptsas_t *)(hba)->tran_hba_private)
#define DIP2MPT(dip)            (TRAN2MPT((scsi_hba_tran_t *)DIP2TRAN(dip)))
#define SDEV2MPT(sd)            (TRAN2MPT(SDEV2TRAN(sd)))
#define PKT2MPT(pkt)            (TRAN2MPT(PKT2TRAN(pkt)))

#define ADDR2MPT(ap)            (TRAN2MPT(ADDR2TRAN(ap)))

#define POLL_TIMEOUT            (2 * SCSI_POLL_TIMEOUT * 1000000)
#define SHORT_POLL_TIMEOUT      (1000000)       /* in usec, about 1 secs */
#define MPTSAS_QUIESCE_TIMEOUT  1               /* 1 sec */
#define MPTSAS_PM_IDLE_TIMEOUT  60              /* 60 seconds */

#define MPTSAS_GET_ISTAT(mpt)  (ddi_get32((mpt)->m_datap, \
                        &(mpt)->m_reg->HostInterruptStatus))

#define MPTSAS_SET_SIGP(P) \
                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 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_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

/*
 * 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_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);
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);

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);
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);
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);
void
mptsas_create_phy_stats(mptsas_t *mpt, char *iport, dev_info_t *dip);
void mptsas_destroy_phy_stats(mptsas_t *mpt);
int mptsas_smhba_phy_init(mptsas_t *mpt);
/*
 * RAID functions
 */
int mptsas_get_raid_settings(mptsas_t *mpt, mptsas_raidvol_t *raidvol);
int mptsas_get_raid_info(mptsas_t *mpt);
int mptsas_get_physdisk_settings(mptsas_t *mpt, mptsas_raidvol_t *raidvol,
    uint8_t physdisknum);
int mptsas_delete_volume(mptsas_t *mpt, uint16_t volid);
void mptsas_raid_action_system_shutdown(mptsas_t *mpt);

#define MPTSAS_IOCSTATUS(status) (status & MPI2_IOCSTATUS_MASK)
/*
 * debugging.
 */
#if defined(MPTSAS_DEBUG)
```

```
void mptsas_printf(char *fmt, ...);
void mptsas_debug_log(char *fmt, ...);

#define MPTSAS_DBGPR(m, args)   \
        if (mptsas_debug_flags & (m)) \
                mptsas_printf args;   \
        if (~mptsas_dbglog_imask & (m)) \
                mptsas_debug_log args
#else   /* ! defined(MPTSAS_DEBUG) */
#define MPTSAS_DBGPR(m, args)
#endif  /* defined(MPTSAS_DEBUG) */

#define NDBG0(args)     MPTSAS_DBGPR(0x01, args)        /* init */
#define NDBG1(args)     MPTSAS_DBGPR(0x02, args)        /* normal running */
#define NDBG2(args)     MPTSAS_DBGPR(0x04, args)        /* property handling */
#define NDBG3(args)     MPTSAS_DBGPR(0x08, args)        /* pkt handling */

#define NDBG4(args)     MPTSAS_DBGPR(0x10, args)        /* kmem alloc/free */
#define NDBG5(args)     MPTSAS_DBGPR(0x20, args)        /* polled cmds */
#define NDBG6(args)     MPTSAS_DBGPR(0x40, args)        /* interrupt setup */
#define NDBG7(args)     MPTSAS_DBGPR(0x80, args)        /* queue handling */

#define NDBG8(args)     MPTSAS_DBGPR(0x0100, args)      /* arq */
#define NDBG9(args)     MPTSAS_DBGPR(0x0200, args)      /* Tagged Q'ing */
#define NDBG10(args)    MPTSAS_DBGPR(0x0400, args)      /* halting chip */
#define NDBG11(args)    MPTSAS_DBGPR(0x0800, args)      /* power management */

#define NDBG12(args)    MPTSAS_DBGPR(0x1000, args)      /* enumeration */
#define NDBG13(args)    MPTSAS_DBGPR(0x2000, args)      /* configuration page */
#define NDBG14(args)    MPTSAS_DBGPR(0x4000, args)      /* LED control */
#define NDBG15(args)    MPTSAS_DBGPR(0x8000, args)      /* Passthrough */

#define NDBG16(args)    MPTSAS_DBGPR(0x010000, args)    /* SAS Broadcasts */
#define NDBG17(args)    MPTSAS_DBGPR(0x020000, args)    /* scatter/gather */
#define NDBG18(args)    MPTSAS_DBGPR(0x040000, args)    /* Interrupts */
#define NDBG19(args)    MPTSAS_DBGPR(0x080000, args)    /* handshaking */

#define NDBG20(args)    MPTSAS_DBGPR(0x100000, args)    /* events */
#define NDBG21(args)    MPTSAS_DBGPR(0x200000, args)    /* dma */
#define NDBG22(args)    MPTSAS_DBGPR(0x400000, args)    /* reset */
#define NDBG23(args)    MPTSAS_DBGPR(0x800000, args)    /* abort */

#define NDBG24(args)    MPTSAS_DBGPR(0x1000000, args)   /* capabilities */
#define NDBG25(args)    MPTSAS_DBGPR(0x2000000, args)   /* flushing */
#define NDBG26(args)    MPTSAS_DBGPR(0x4000000, args)
#define NDBG27(args)    MPTSAS_DBGPR(0x8000000, args)

#define NDBG28(args)    MPTSAS_DBGPR(0x10000000, args)  /* hotplug */
#define NDBG29(args)    MPTSAS_DBGPR(0x20000000, args)  /* timeouts */
#define NDBG30(args)    MPTSAS_DBGPR(0x40000000, args)  /* mptsas_watch */
#define NDBG31(args)    MPTSAS_DBGPR(0x80000000, args)  /* negotations */

/*
 * auto request sense
 */
#define RQ_MAKECOM_COMMON(pkt, flag, cmd) \
        (pkt)->pkt_flags = (flag), \
        ((union scsi_cdb *)(pkt)->pkt_cdbp)->scc_cmd = (cmd), \
        ((union scsi_cdb *)(pkt)->pkt_cdbp)->scc_lun = \
            (pkt)->pkt_address.a_lun

#define RQ_MAKECOM_G0(pkt, flag, cmd, addr, cnt) \
        RQ_MAKECOM_COMMON((pkt), (flag), (cmd)), \
        FORMG0ADDR(((union scsi_cdb *)(pkt)->pkt_cdbp), (addr)), \
        FORMG0COUNT(((union scsi_cdb *)(pkt)->pkt_cdbp), (cnt))
```



```
#ifdef  __cplusplus
}
#endif

#endif  /* _SYS_SCSI_ADAPTERS_MPTSAS3_VAR_H */
#endif /* ! codereview */
```

```
**********************************************************
   16031 Thu Jun 12 17:28:24 2014
new/usr/src/uts/intel/Makefile.intel
4546 mpt_sas needs enhancing to support LSI MPI2.5
**********************************************************
# 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) 2005, 2010, Oracle and/or its affiliates. All rights reserved.
# Copyright (c) 2012 Nexenta Systems, Inc. All rights reserved.
# Copyright (c) 2013 Andrew Stormont.  All rights reserved.

#
#       This makefile contains the common definitions for all intel
#       implementation architecture independent modules.
#

#
#       Machine type (implementation architecture):
#
PLATFORM         = i86pc

#
#       Everybody needs to know how to build modstubs.o and to locate unix.o.
#       Note that unix.o must currently be selected from among the possible
#       "implementation architectures". Note further, that unix.o is only
#       used as an optional error check for undefines so (theoretically)
#       any "implementation architectures" could be used. We choose i86pc
#       because it is the reference port.
#
UNIX_DIR         = $(UTSBASE)/i86pc/unix
GENLIB_DIR       = $(UTSBASE)/intel/genunix
IPDRV_DIR        = $(UTSBASE)/intel/ip
MODSTUBS_DIR     = $(UNIX_DIR)
DSF_DIR          = $(UTSBASE)/$(PLATFORM)/genassym
LINTS_DIR        = $(OBJS_DIR)
LINT_LIB_DIR     = $(UTSBASE)/intel/lint-libs/$(OBJS_DIR)

UNIX_O           = $(UNIX_DIR)/$(OBJS_DIR)/unix.o
GENLIB           = $(GENLIB_DIR)/$(OBJS_DIR)/libgenunix.so
MODSTUBS_O       = $(MODSTUBS_DIR)/$(OBJS_DIR)/modstubs.o
LINT_LIB         = $(UTSBASE)/i86pc/lint-libs/$(OBJS_DIR)/llib-lunix.ln
GEN_LINT_LIB     = $(UTSBASE)/intel/lint-libs/$(OBJS_DIR)/llib-lgenunix.ln

#
#       Include the makefiles which define build rule templates, the
#       collection of files per module, and a few specific flags. Note
#       that order is significant, just as with an include path. The
#       first build rule template which matches the files name will be
```



```
#       used. By including these in order from most machine dependent
#       to most machine independent, we allow a machine dependent file
#       to be used in preference over a machine independent version
#       (Such as a machine specific optimization, which preserves the
#       interfaces.)
#
include $(UTSBASE)/intel/Makefile.files
include $(UTSBASE)/common/Makefile.files

#
# ----- TRANSITIONAL SECTION ---------------------------------------------------
#

#
#       Not everything which *should* be a module is a module yet. The
#       following is a list of such objects which are currently part of
#       genunix but which might someday become kmods.  This must be
#       defined before we include Makefile.uts, or else genunix's build
#       won't be as parallel as we might like.
#
NOT_YET_KMODS    = $(OLDPTY_OBJS) $(PTY_OBJS) $(VCONS_CONF_OBJS) $(MOD_OBJS)

#
# ----- END OF TRANSITIONAL SECTION --------------------------------------------
#
#       Include machine independent rules. Note that this does not imply
#       that the resulting module from rules in Makefile.uts is machine
#       independent. Only that the build rules are machine independent.
#
include $(UTSBASE)/Makefile.uts

#
#       The following must be defined for all implementations:
#
MODSTUBS                = $(UTSBASE)/intel/ia32/ml/modstubs.s

#
#       Define supported builds
#
DEF_BUILDS              = $(DEF_BUILDS64) $(DEF_BUILDS32)
ALL_BUILDS              = $(ALL_BUILDS64) $(ALL_BUILDS32)

#
#       x86 or amd64 inline templates
#
INLINES_32              = $(UTSBASE)/intel/ia32/ml/ia32.il
INLINES_64              = $(UTSBASE)/intel/amd64/ml/amd64.il
INLINES                 += $(INLINES_$(CLASS))

#
#       kernel-specific optimizations; override default in Makefile.master
#

CFLAGS_XARCH_32         = $(i386_CFLAGS)
CFLAGS_XARCH_64         = $(amd64_CFLAGS)
CFLAGS_XARCH            = $(CFLAGS_XARCH_$(CLASS))

COPTFLAG_32             = $(COPTFLAG)
COPTFLAG_64             = $(COPTFLAG64)
COPTIMIZE               = $(COPTFLAG_$(CLASS))

CFLAGS                  = $(CFLAGS_XARCH)
CFLAGS                  += $(COPTIMIZE)
CFLAGS                  += $(INLINES) -D_ASM_INLINES
CFLAGS                  += $(CCMODE)
CFLAGS                  += $(SPACEFLAG)
```

```
CFLAGS                  += $(CCUNBOUND)
CFLAGS                  += $(CFLAGS_uts)
CFLAGS                  += -xstrconst

ASFLAGS_XARCH_32        = $(i386_ASFLAGS)
ASFLAGS_XARCH_64        = $(amd64_ASFLAGS)
ASFLAGS_XARCH           = $(ASFLAGS_XARCH_$(CLASS))

ASFLAGS                 += $(ASFLAGS_XARCH)

#
#       Define the base directory for installation.
#
BASE_INS_DIR    = $(ROOT)

#
#       Debugging level
#
#       Special knowledge of which special debugging options affect which
#       file is used to optimize the build if these flags are changed.
#
DEBUG_DEFS_OBJ32        =
DEBUG_DEFS_DBG32        = -DDEBUG
DEBUG_DEFS_OBJ64        =
DEBUG_DEFS_DBG64        = -DDEBUG
DEBUG_DEFS              = $(DEBUG_DEFS_$(BUILD_TYPE))

DEBUG_COND_OBJ32        = $(POUND_SIGN)
DEBUG_COND_DBG32        =
DEBUG_COND_OBJ64        = $(POUND_SIGN)
DEBUG_COND_DBG64        =
IF_DEBUG_OBJ            = $(DEBUG_COND_$(BUILD_TYPE))$(OBJS_DIR)/

$(IF_DEBUG_OBJ)syscall.o        :=      DEBUG_DEFS      += -DSYSCALLTRACE
$(IF_DEBUG_OBJ)clock.o          :=      DEBUG_DEFS      += -DKSLICE=1

#
#       Collect the preprocessor definitions to be associated with *all*
#       files.
#
ALL_DEFS         = $(DEBUG_DEFS) $(OPTION_DEFS)

#
#       The kernels modules which are "implementation architecture"
#       specific for this machine are enumerated below. Note that most
#       of these modules must exist (in one form or another) for each
#       architecture.
#
#       Common Drivers (usually pseudo drivers) (/kernel/drv)
#       DRV_KMODS are built both 32-bit and 64-bit
#       DRV_KMODS_32 are built only 32-bit
#       DRV_KMODS_64 are built only 64-bit
#
DRV_KMODS       += aac
DRV_KMODS       += aggr
DRV_KMODS       += ahci
DRV_KMODS       += amd64_gart
DRV_KMODS       += amr
DRV_KMODS       += agpgart
DRV_KMODS       += srn
DRV_KMODS       += agptarget
DRV_KMODS       += arn
DRV_KMODS       += arp
DRV_KMODS       += asy
DRV_KMODS       += ata
DRV_KMODS       += ath
```



```
DRV_KMODS       += atu
DRV_KMODS       += audio
DRV_KMODS       += audio1575
DRV_KMODS       += audio810
DRV_KMODS       += audiocmi
DRV_KMODS       += audiocmihd
DRV_KMODS       += audioemu10k
DRV_KMODS       += audioens
DRV_KMODS       += audiohd
DRV_KMODS       += audioixp
DRV_KMODS       += audiols
DRV_KMODS       += audiop16x
DRV_KMODS       += audiopci
DRV_KMODS       += audiosolo
DRV_KMODS       += audiots
DRV_KMODS       += audiovia823x
DRV_KMODS_32    += audiovia97
DRV_KMODS       += bl
DRV_KMODS       += blkdev
DRV_KMODS       += bge
DRV_KMODS       += bofi
DRV_KMODS       += bpf
DRV_KMODS       += bridge
DRV_KMODS       += bscbus
DRV_KMODS       += bscv
DRV_KMODS       += chxge
DRV_KMODS       += cxgbe
DRV_KMODS       += ntxn
DRV_KMODS       += myri10ge
DRV_KMODS       += clone
DRV_KMODS       += cmdk
DRV_KMODS       += cn
DRV_KMODS       += conskbd
DRV_KMODS       += consms
DRV_KMODS       += cpqary3
DRV_KMODS       += cpuid
DRV_KMODS       += cpunex
DRV_KMODS       += crypto
DRV_KMODS       += cryptoadm
DRV_KMODS       += dca
DRV_KMODS       += devinfo
DRV_KMODS       += dld
DRV_KMODS       += dlpistub
DRV_KMODS_32    += dnet
DRV_KMODS       += dump
DRV_KMODS       += ecpp
DRV_KMODS       += emlxs
DRV_KMODS       += fd
DRV_KMODS       += fdc
DRV_KMODS       += fm
DRV_KMODS       += fssnap
DRV_KMODS       += hxge
DRV_KMODS       += i8042
DRV_KMODS       += i915
DRV_KMODS       += icmp
DRV_KMODS       += icmp6
DRV_KMODS       += intel_nb5000
DRV_KMODS       += intel_nhm
DRV_KMODS       += ip
DRV_KMODS       += ip6
DRV_KMODS       += ipd
DRV_KMODS       += ipf
DRV_KMODS       += ipnet
DRV_KMODS       += ippctl
DRV_KMODS       += ipsecah
DRV_KMODS       += ipsecesp
```

```
DRV_KMODS       += ipw
DRV_KMODS       += iwh
DRV_KMODS       += iwi
DRV_KMODS       += iwk
DRV_KMODS       += iwp
DRV_KMODS       += iwscn
DRV_KMODS       += kb8042
DRV_KMODS       += keysock
DRV_KMODS       += kssl
DRV_KMODS       += kstat
DRV_KMODS       += ksyms
DRV_KMODS       += kmdb
DRV_KMODS       += llc1
DRV_KMODS       += lofi
DRV_KMODS       += log
DRV_KMODS       += logindmux
DRV_KMODS       += mega_sas
DRV_KMODS       += mc-amd
DRV_KMODS       += mm
DRV_KMODS       += mouse8042
DRV_KMODS       += mpt_sas3
#endif /* ! codereview */
DRV_KMODS       += mpt_sas
DRV_KMODS       += mr_sas
DRV_KMODS       += mwl
DRV_KMODS       += nca
DRV_KMODS       += nsmb
DRV_KMODS       += nulldriver
DRV_KMODS       += nv_sata
DRV_KMODS       += nxge
DRV_KMODS       += oce
DRV_KMODS       += openeepr
DRV_KMODS       += pci_pci
DRV_KMODS       += pcic
DRV_KMODS       += pcieb
DRV_KMODS       += physmem
DRV_KMODS       += pit_beep
DRV_KMODS       += pm
DRV_KMODS       += poll
DRV_KMODS       += pool
DRV_KMODS       += power
DRV_KMODS       += pseudo
DRV_KMODS       += ptc
DRV_KMODS       += ptm
DRV_KMODS       += pts
DRV_KMODS       += ptsl
DRV_KMODS       += qlge
DRV_KMODS       += radeon
DRV_KMODS       += ral
DRV_KMODS       += ramdisk
DRV_KMODS       += random
DRV_KMODS       += rds
DRV_KMODS       += rdsv3
DRV_KMODS       += rpcib
DRV_KMODS       += rsm
DRV_KMODS       += rts
DRV_KMODS       += rtw
DRV_KMODS       += rum
DRV_KMODS       += rwd
DRV_KMODS       += rwn
DRV_KMODS       += sad
DRV_KMODS       += sd
DRV_KMODS       += sdhost
DRV_KMODS       += sgen
DRV_KMODS       += si3124
DRV_KMODS       += smbios
```



```
DRV_KMODS       += softmac
DRV_KMODS       += spdsock
DRV_KMODS       += smbsrv
DRV_KMODS       += smp
DRV_KMODS       += sppp
DRV_KMODS       += spppptun
DRV_KMODS       += srpt
DRV_KMODS       += st
DRV_KMODS       += sy
DRV_KMODS       += sysevent
DRV_KMODS       += sysmsg
DRV_KMODS       += tcp
DRV_KMODS       += tcp6
DRV_KMODS       += tl
DRV_KMODS       += tnf
DRV_KMODS       += tpm
DRV_KMODS       += trill
DRV_KMODS       += udp
DRV_KMODS       += udp6
DRV_KMODS       += ucode
DRV_KMODS       += ural
DRV_KMODS       += uath
DRV_KMODS       += urtw
DRV_KMODS       += vgatext
DRV_KMODS       += heci
DRV_KMODS       += vnic
DRV_KMODS       += vscan
DRV_KMODS       += wc
DRV_KMODS       += winlock
DRV_KMODS       += wpi
DRV_KMODS       += xge
DRV_KMODS       += yge
DRV_KMODS       += zcons
DRV_KMODS       += zyd
DRV_KMODS       += simnet
DRV_KMODS       += stmf
DRV_KMODS       += stmf_sbd
DRV_KMODS       += fct
DRV_KMODS       += fcoe
DRV_KMODS       += fcoet
DRV_KMODS       += fcoei
DRV_KMODS       += qlt
DRV_KMODS       += iscsit
DRV_KMODS       += pppt
DRV_KMODS       += ncall nsctl sdbc nskern sv
DRV_KMODS       += ii rdc rdcsrv rdcstub
DRV_KMODS       += iptun

#
# Common code drivers
#

DRV_KMODS       += afe
DRV_KMODS       += atge
DRV_KMODS       += bfe
DRV_KMODS       += dmfe
DRV_KMODS       += e1000g
DRV_KMODS       += efe
DRV_KMODS       += elxl
DRV_KMODS       += hme
DRV_KMODS       += mxfe
DRV_KMODS       += nge
DRV_KMODS       += pcn
DRV_KMODS       += rge
DRV_KMODS       += rtls
DRV_KMODS       += sfe
```

```
DRV_KMODS       += amd8111s
DRV_KMODS       += igb
DRV_KMODS       += ipmi
DRV_KMODS       += iprb
DRV_KMODS       += ixgbe
DRV_KMODS       += vr

#
# Virtio drivers
#

# Virtio core
DRV_KMODS       += virtio

# Virtio block driver
DRV_KMODS       += vioblk

#
#       DTrace and DTrace Providers
#
DRV_KMODS       += dtrace
DRV_KMODS       += fbt
DRV_KMODS       += lockstat
DRV_KMODS       += profile
DRV_KMODS       += sdt
DRV_KMODS       += systrace
DRV_KMODS       += fasttrap
DRV_KMODS       += dcpc

#
#       I/O framework test drivers
#
DRV_KMODS       += pshot
DRV_KMODS       += gen_drv
DRV_KMODS       += tvhci tphci tclient
DRV_KMODS       += emul64

#
#       Machine Specific Driver Modules (/kernel/drv):
#
DRV_KMODS       += options
DRV_KMODS       += scsi_vhci
DRV_KMODS       += pmcs
DRV_KMODS       += pmcs8001fw
DRV_KMODS       += arcmsr
DRV_KMODS       += fcp
DRV_KMODS       += fcip
DRV_KMODS       += fcsm
DRV_KMODS       += fp
DRV_KMODS       += qlc
DRV_KMODS       += iscsi

#
#       PCMCIA specific module(s)
#
DRV_KMODS       += pcs
MISC_KMODS      += cardbus

#
#       SCSI Enclosure Services driver
#
DRV_KMODS       += ses

#
#       USB specific modules
#
```



```
DRV_KMODS       += hid
DRV_KMODS       += hwarc hwahc
DRV_KMODS       += hubd
DRV_KMODS       += uhci
DRV_KMODS       += ehci
DRV_KMODS       += ohci
DRV_KMODS       += usb_mid
DRV_KMODS       += usb_ia
DRV_KMODS       += scsa2usb
DRV_KMODS       += usbprn
DRV_KMODS       += ugen
DRV_KMODS       += usbser
DRV_KMODS       += usbsacm
DRV_KMODS       += usbsksp
DRV_KMODS       += usbsprl
DRV_KMODS       += usb_ac
DRV_KMODS       += usb_as
DRV_KMODS       += usbskel
DRV_KMODS       += usbvc
DRV_KMODS       += usbftdi
DRV_KMODS       += wusb_df
DRV_KMODS       += wusb_ca
DRV_KMODS       += usbecm

#
#       1394 modules
#
MISC_KMODS      += s1394 sbp2
DRV_KMODS       += hci1394 scsa1394
DRV_KMODS       += av1394
DRV_KMODS       += dcam1394

#
#       InfiniBand pseudo drivers
#
DRV_KMODS       += ib ibp eibnx eoib rdsib sdp iser daplt hermon tavor sol_ucma
DRV_KMODS       += sol_umad

#
#       LVM modules
#
DRV_KMODS       += md
MISC_KMODS      += md_stripe md_hotspares md_mirror md_raid md_trans md_notify
MISC_KMODS      += md_sp

#
#       Brand modules
#
BRAND_KMODS     += sn1_brand s10_brand

#
#       Exec Class Modules (/kernel/exec):
#
EXEC_KMODS      += elfexec intpexec shbinexec javaexec

#
#       Scheduling Class Modules (/kernel/sched):
#
SCHED_KMODS     += IA RT TS RT_DPTBL TS_DPTBL FSS FX FX_DPTBL SDC

#
#       File System Modules (/kernel/fs):
#
FS_KMODS        += autofs cachefs ctfs dcfs dev devfs fdfs fifofs hsfs lofs
FS_KMODS        += mntfs namefs nfs objfs zfs zut
FS_KMODS        += pcfs procfs sockfs specfs tmpfs udfs ufs sharefs
```

```
FS_KMODS        += smbfs

#
#       Streams Modules (/kernel/strmod):
#
STRMOD_KMODS    += bufmod connld dedump ldterm pckt pfmod pipemod
STRMOD_KMODS    += ptem redirmod rpcmod rlmod telmod timod
STRMOD_KMODS    += spppasyn spppcomp
STRMOD_KMODS    += tirdwr ttcompat
STRMOD_KMODS    += usbkbm
STRMOD_KMODS    += usbms
STRMOD_KMODS    += usbwcm
STRMOD_KMODS    += usb_ah
STRMOD_KMODS    += drcompat
STRMOD_KMODS    += cryptmod
STRMOD_KMODS    += vuid2ps2
STRMOD_KMODS    += vuid3ps2
STRMOD_KMODS    += vuidm3p
STRMOD_KMODS    += vuidm4p
STRMOD_KMODS    += vuidm5p

#
#       'System' Modules (/kernel/sys):
#
SYS_KMODS       += c2audit
SYS_KMODS       += doorfs
SYS_KMODS       += exacctsys
SYS_KMODS       += inst_sync
SYS_KMODS       += kaio
SYS_KMODS       += msgsys
SYS_KMODS       += pipe
SYS_KMODS       += portfs
SYS_KMODS       += pset
SYS_KMODS       += semsys
SYS_KMODS       += shmsys
SYS_KMODS       += sysacct
SYS_KMODS       += acctctl

#
#       'Misc' Modules (/kernel/misc)
#       MISC_KMODS are built both 32-bit and 64-bit
#       MISC_KMODS_32 are built only 32-bit
#       MISC_KMODS_64 are built only 64-bit
#
MISC_KMODS      += ac97
MISC_KMODS      += acpica
MISC_KMODS      += agpmaster
MISC_KMODS      += bignum
MISC_KMODS      += bootdev
MISC_KMODS      += busra
MISC_KMODS      += cmlb
MISC_KMODS      += consconfig
MISC_KMODS      += ctf
MISC_KMODS      += dadk
MISC_KMODS      += dcopy
MISC_KMODS      += dls
MISC_KMODS      += drm
MISC_KMODS      += fssnap_if
MISC_KMODS      += gda
MISC_KMODS      += gld
MISC_KMODS      += hidparser
MISC_KMODS      += hook
MISC_KMODS      += hpcsvc
MISC_KMODS      += ibcm
MISC_KMODS      += ibdm
MISC_KMODS      += ibdma
```



```
MISC_KMODS      += ibmf
MISC_KMODS      += ibtl
MISC_KMODS      += idm
MISC_KMODS      += idmap
MISC_KMODS      += iommulib
MISC_KMODS      += ipc
MISC_KMODS      += kbtrans
MISC_KMODS      += kcf
MISC_KMODS      += kgssapi
MISC_KMODS      += kmech_dummy
MISC_KMODS      += kmech_krb5
MISC_KMODS      += ksocket
MISC_KMODS      += mac
MISC_KMODS      += mii
MISC_KMODS      += mwlfw
MISC_KMODS      += net80211
MISC_KMODS      += nfs_dlboot
MISC_KMODS      += nfssrv
MISC_KMODS      += neti
MISC_KMODS      += pci_autoconfig
MISC_KMODS      += pcicfg
MISC_KMODS      += pcihp
MISC_KMODS      += pcmcia
MISC_KMODS      += rpcsec
MISC_KMODS      += rpcsec_gss
MISC_KMODS      += rsmops
MISC_KMODS      += sata
MISC_KMODS      += scsi
MISC_KMODS      += sda
MISC_KMODS      += sol_ofs
MISC_KMODS      += spuni
MISC_KMODS      += strategy
MISC_KMODS      += strplumb
MISC_KMODS      += tem
MISC_KMODS      += tlimod
MISC_KMODS      += usba usba10 usbs49_fw
MISC_KMODS      += scsi_vhci_f_sym_hds
MISC_KMODS      += scsi_vhci_f_sym
MISC_KMODS      += scsi_vhci_f_tpgs
MISC_KMODS      += scsi_vhci_f_asym_sun
MISC_KMODS      += scsi_vhci_f_tape
MISC_KMODS      += scsi_vhci_f_tpgs_tape
MISC_KMODS      += fctl
MISC_KMODS      += emlxs_fw
MISC_KMODS      += qlc_fw_2200
MISC_KMODS      += qlc_fw_2300
MISC_KMODS      += qlc_fw_2400
MISC_KMODS      += qlc_fw_2500
MISC_KMODS      += qlc_fw_6322
MISC_KMODS      += qlc_fw_8100
MISC_KMODS      += hwa1480_fw
MISC_KMODS      += uathfw
MISC_KMODS      += uwba

MISC_KMODS      += klmmod klmops

#
#       Software Cryptographic Providers (/kernel/crypto):
#
CRYPTO_KMODS    += aes
CRYPTO_KMODS    += arcfour
CRYPTO_KMODS    += blowfish
CRYPTO_KMODS    += des
CRYPTO_KMODS    += ecc
CRYPTO_KMODS    += md4
CRYPTO_KMODS    += md5
```

```
CRYPTO_KMODS    += rsa
CRYPTO_KMODS    += sha1
CRYPTO_KMODS    += sha2
CRYPTO_KMODS    += swrand

#
#       IP Policy Modules (/kernel/ipp)
#
IPP_KMODS       += dlcosmk
IPP_KMODS       += flowacct
IPP_KMODS       += ipgpc
IPP_KMODS       += dscpmk
IPP_KMODS       += tokenmt
IPP_KMODS       += tswtclmt

#
#       generic-unix module (/kernel/genunix):
#
GENUNIX_KMODS   += genunix

#
#       Modules eXcluded from the product:
#

#
#       'Dacf' Modules (/kernel/dacf):
#

#
#       Performance Counter BackEnd modules (/usr/kernel/pcbe)
#
PCBE_KMODS      += p123_pcbe p4_pcbe opteron_pcbe core_pcbe

#
#       MAC-Type Plugin Modules (/kernel/mac)
#
MAC_KMODS       += mac_6to4
MAC_KMODS       += mac_ether
MAC_KMODS       += mac_ipv4
MAC_KMODS       += mac_ipv6
MAC_KMODS       += mac_wifi
MAC_KMODS       += mac_ib

#
# socketmod (kernel/socketmod)
#
SOCKET_KMODS    += sockpfp
SOCKET_KMODS    += socksctp
SOCKET_KMODS    += socksdp
SOCKET_KMODS    += sockrds
SOCKET_KMODS    += ksslf

#
#       kiconv modules (/kernel/kiconv):
#
KICONV_KMODS    += kiconv_emea kiconv_ja kiconv_ko kiconv_sc kiconv_tc

#
#       'Dacf' Modules (/kernel/dacf):
#
DACF_KMODS      += net_dacf

#
# Ensure that the variable member of the cpu_t (cpu_m) is defined
# for the lint builds so as not to cause lint errors during the
# global cross check.
```



```
#
LINTFLAGS       += -D_MACHDEP -I$(UTSBASE)/i86pc
```

```
**********************************************************
    2972 Thu Jun 12 17:28:24 2014
new/usr/src/uts/intel/mpt_sas3/Makefile
4546 mpt_sas needs enhancing to support LSI MPI2.5
**********************************************************
#
# 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.
#
#       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)
ROOTMODULE      = $(ROOT_DRV_DIR)/$(MODULE)
CONF_SRCDIR     = $(UTSBASE)/common/io/scsi/adapters/mpt_sas3/
WARLOCK_OUT     = $(MPTSAS3_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
        $(WLCC) $(CPPFLAGS) -DDEBUG -o $@ $<

warlock_ddi.files:
        @cd $(UTSBASE)/intel/warlock; pwd; $(MAKE) warlock

scsi.files:
        @cd $(UTSBASE)/intel/scsi; pwd; $(MAKE) warlock
```

```
#endif /* ! codereview */
```

```
**********************************************************
   13361 Thu Jun 12 17:28:24 2014
new/usr/src/uts/sparc/Makefile.sparc
4546 mpt_sas needs enhancing to support LSI MPI2.5
**********************************************************
#
# 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) 2005, 2010, Oracle and/or its affiliates. All rights reserved.
# Copyright (c) 2013 Andrew Stormont.  All rights reserved.


#
#       This makefile contains the common definitions for all sparc
#       implementation architecture independent modules.
#

#
#       Define supported builds
#
DEF_BUILDS      = $(DEF_BUILDS64)
ALL_BUILDS      = $(ALL_BUILDS64)

#
#       Everybody needs to know how to build modstubs.o and to locate unix.o.
#       Note that unix.o must currently be selected from among the possible
#       "implementation architectures". Note further, that unix.o is only
#       used as an optional error check for undefines so (theoretically)
#       any "implementation architectures" could be used. We choose sun4u
#       because it is the reference port.
#
UNIX_DIR         = $(UTSBASE)/sun4u/unix
GENLIB_DIR       = $(UTSBASE)/sun4u/genunix
IPDRV_DIR        = $(UTSBASE)/sparc/ip
MODSTUBS_DIR     = $(UNIX_DIR)
DSF_DIR          = $(UNIX_DIR)
LINTS_DIR        = $(OBJS_DIR)
LINT_LIB_DIR     = $(UTSBASE)/sparc/lint-libs/$(OBJS_DIR)

UNIX_O           = $(UNIX_DIR)/$(OBJS_DIR)/unix.o
MODSTUBS_O       = $(MODSTUBS_DIR)/$(OBJS_DIR)/modstubs.o
GENLIB           = $(UTSBASE)/sun4u/lint-libs/$(OBJS_DIR)/libgenunix.so

LINT_LIB_32      = $(UTSBASE)/sun4u/lint-libs/$(OBJS_DIR)/llib-lunix.ln
GEN_LINT_LIB_32  = $(UTSBASE)/sun4u/lint-libs/$(OBJS_DIR)/llib-lgenunix.ln

LINT_LIB_64      = $(UTSBASE)/sun4u/lint-libs/$(OBJS_DIR)/llib-lunix.ln
GEN_LINT_LIB_64  = $(UTSBASE)/sun4u/lint-libs/$(OBJS_DIR)/llib-lgenunix.ln
```



```
LINT_LIB         = $(LINT_LIB_$(CLASS))
GEN_LINT_LIB     = $(GEN_LINT_LIB_$(CLASS))

LINT32_DIRS      = $(LINT32_BUILDS:%=$(UTSBASE)/sparc/lint-libs/%)
LINT32_FILES     = $(LINT32_DIRS:%=%/llib-l$(MODULE).ln)

LINT64_DIRS      = $(LINT64_BUILDS:%=$(UTSBASE)/sparc/lint-libs/%)
LINT64_FILES     = $(LINT64_DIRS:%=%/llib-l$(MODULE).ln)

#
#       Include the makefiles which define build rule templates, the
#       collection of files per module, and a few specific flags. Note
#       that order is significant, just as with an include path. The
#       first build rule template which matches the files name will be
#       used. By including these in order from most machine dependent
#       to most machine independent, we allow a machine dependent file
#       to be used in preference over a machine independent version
#       (Such as a machine specific optimization, which preserves the
#       interfaces.)
#
include $(UTSBASE)/sparc/Makefile.files
include $(UTSBASE)/sparc/v9/Makefile.files
include $(UTSBASE)/sun/Makefile.files
include $(UTSBASE)/common/Makefile.files

#
# ----- TRANSITIONAL SECTION ---------------------------------------------------
#

#
#       Not everything which *should* be a module is a module yet. The
#       following is a list of such objects which are currently part of
#       genunix but which might someday become kmods.   This must be
#       defined before we include Makefile.uts, or else genunix's build
#       won't be as parallel as we might like.
#
NOT_YET_KMODS    = $(OLDPTY_OBJS) $(PTY_OBJS) $(VCONS_CONF_OBJS) $(MOD_OBJS)

#
# ----- END OF TRANSITIONAL SECTION --------------------------------------------
#
#       Include machine independent rules. Note that this does not imply
#       that the resulting module from rules in Makefile.uts is machine
#       independent. Only that the build rules are machine independent.
#
include $(UTSBASE)/Makefile.uts

#
#       machine specific optimization, override default in Makefile.master
#
XARCH_32        = -xarch=v8
XARCH_64        = -m64
XARCH           = $(XARCH_$(CLASS))

COPTIMIZE_32    = -xO3
COPTIMIZE_64    = -xO3
COPTIMIZE       = $(COPTIMIZE_$(CLASS))

CCMODE          = -Xa

CFLAGS_32       = -xcg92
CFLAGS_64       = -xchip=ultra $(CCABS32) $(CCREGSYM)
CFLAGS          = $(CFLAGS_$(CLASS))

CFLAGS          += $(XARCH)
```

```
CFLAGS          += $(COPTIMIZE)
CFLAGS          += $(EXTRA_CFLAGS)
CFLAGS          += $(XAOPT)
CFLAGS          += $(INLINES) -D_ASM_INLINES
CFLAGS          += $(CCMODE)
CFLAGS          += $(SPACEFLAG)
CFLAGS          += $(CERRWARN)
CFLAGS          += $(CTF_FLAGS_$(CLASS))
CFLAGS          += $(C99MODE)
CFLAGS          += $(CCUNBOUND)
CFLAGS          += $(CCSTATICSYM)
CFLAGS          += $(CC32BITCALLERS)
CFLAGS          += $(CCNOAUTOINLINE)
CFLAGS          += $(IROPTFLAG)
CFLAGS          += $(CGLOBALSTATIC)
CFLAGS          += -xregs=no%float
CFLAGS          += -xstrconst
CFLAGS          += $(CSOURCEDEBUGFLAGS)
CFLAGS          += $(CUSERFLAGS)

ASFLAGS         += $(XARCH)

LINT_DEFS_32    =
LINT_DEFS_64    = -m64
LINT_DEFS       += $(LINT_DEFS_$(CLASS))

#
#       The following must be defined for all implementations:
#
#       MODSTUBS:       Module stubs source file.
#
MODSTUBS         = $(UTSBASE)/sparc/ml/modstubs.s

#
#       Define the actual specific platforms - obviously none.
#
MACHINE_DEFS     =

#
#       Debugging level
#
#       Special knowledge of which special debugging options effect which
#       file is used to optimize the build if these flags are changed.
#
#       XXX: The above could possibly be done for more flags and files, but
#            is left as an experiment to the interested reader. Be forewarned,
#            that excessive use could lead to maintenance difficulties.
#
DEBUG_DEFS_OBJ32        =
DEBUG_DEFS_DBG32        = -DDEBUG
DEBUG_DEFS_OBJ64        =
DEBUG_DEFS_DBG64        = -DDEBUG
DEBUG_DEFS              = $(DEBUG_DEFS_$(BUILD_TYPE))

DEBUG_COND_OBJ32        = $(POUND_SIGN)
DEBUG_COND_DBG32        =
DEBUG_COND_OBJ64        = $(POUND_SIGN)
DEBUG_COND_DBG64        =
IF_DEBUG_OBJ            = $(DEBUG_COND_$(BUILD_TYPE))$(OBJS_DIR)/

$(IF_DEBUG_OBJ)syscall.o        :=      DEBUG_DEFS      += -DSYSCALLTRACE
$(IF_DEBUG_OBJ)clock.o          :=      DEBUG_DEFS      += -DKSLICE=1

# Comment these out if you don't want dispatcher lock statistics.

# $(IF_DEBUG_OBJ)disp_lock.o    := DEBUG_DEFS   += -DDISP_LOCK_STATS
```



```
#
#       Collect the preprocessor definitions to be associated with *all*
#       files.
#
ALL_DEFS         = $(MACHINE_DEFS) $(DEBUG_DEFS) $(OPTION_DEFS)
#
#
#       The kernels modules which are "implementation architecture"
#       specific for this machine are enumerated below. Note that most
#       of these modules must exist (in one form or another) for each
#       architecture.
#
#       Common Drivers (usually pseudo drivers) (/kernel/drv):
#
DRV_KMODS       += aggr arp audio bl blkdev bofi clone cn conskbd consms cpuid
DRV_KMODS       += crypto cryptoadm devinfo dump
DRV_KMODS       += dtrace fasttrap fbt lockstat profile sdt systrace dcpc
DRV_KMODS       += fssnap icmp icmp6 ip ip6 ipnet ipsecah
DRV_KMODS       += ipsecesp iptun iwscn keysock kmdb kstat ksyms llc1
DRV_KMODS       += lofi
DRV_KMODS       += log logindmux kssl mm nca physmem pm poll pool
DRV_KMODS       += pseudo ptc ptm pts ptsl ramdisk random rsm rts sad
DRV_KMODS       += simnet softmac sppp spppptun sy sysevent sysmsg
DRV_KMODS       += spdsock
DRV_KMODS       += tcp tcp6 tl tnf ttymux udp udp6 wc winlock zcons
DRV_KMODS       += ippctl
DRV_KMODS       += dld
DRV_KMODS       += ipd
DRV_KMODS       += ipf
DRV_KMODS       += rpcib
DRV_KMODS       += dlpistub
DRV_KMODS       += vnic
DRV_KMODS       += xge
DRV_KMODS       += rds
DRV_KMODS       += rdsv3
DRV_KMODS       += chxge
DRV_KMODS       += smbsrv
DRV_KMODS       += vscan
DRV_KMODS       += nsmb
DRV_KMODS       += fm
DRV_KMODS       += nulldriver
DRV_KMODS       += bridge trill
DRV_KMODS       += bpf
DRV_KMODS       += dca

#
#       Hardware Drivers in common space
#

DRV_KMODS       += afe
DRV_KMODS       += audio1575
DRV_KMODS       += audioens
DRV_KMODS       += audiols
DRV_KMODS       += audiop16x
DRV_KMODS       += audiopci
DRV_KMODS       += audiots
DRV_KMODS       += e1000g
DRV_KMODS       += efe
DRV_KMODS       += hxge
DRV_KMODS       += mxfe
DRV_KMODS       += rge
DRV_KMODS       += rtls
DRV_KMODS       += sfe
DRV_KMODS       += aac
DRV_KMODS       += igb
```

```
DRV_KMODS       += ixgbe
DRV_KMODS       += vr
DRV_KMODS       += mr_sas
DRV_KMODS       += yge

#
#       Machine Specific Driver Modules (/kernel/drv):
#
DRV_KMODS       += audiocs
DRV_KMODS       += bge dmfe eri fas hme qfe
DRV_KMODS       += openeepr options sd ses st
DRV_KMODS       += ssd
DRV_KMODS       += ecpp
DRV_KMODS       += hid hubd ehci ohci uhci usb_mid usb_ia scsa2usb usbprn ugen
DRV_KMODS       += usbser usbsacm usbsksp usbsprl
DRV_KMODS       += usb_as usb_ac
DRV_KMODS       += usbskel
DRV_KMODS       += usbvc
DRV_KMODS       += usbftdi
DRV_KMODS       += wusb_df hwahc hwarc wusb_ca
DRV_KMODS       += usbecm
DRV_KMODS       += hci1394 av1394 scsa1394 dcam1394
DRV_KMODS       += sbp2
DRV_KMODS       += ib ibp eibnx eoib rdsib sdp iser daplt hermon tavor sol_ucma
DRV_KMODS       += sol_umad
DRV_KMODS       += pci_pci pcieb pcieb_bcm
DRV_KMODS       += i8042 kb8042 mouse8042
DRV_KMODS       += fcode
DRV_KMODS       += mpt_sas3
#endif /* ! codereview */
DRV_KMODS       += mpt_sas
DRV_KMODS       += socal
DRV_KMODS       += sgen
DRV_KMODS       += myri10ge
DRV_KMODS       += smp
DRV_KMODS       += dad
DRV_KMODS       += scsi_vhci
DRV_KMODS       += fcp
DRV_KMODS       += fcip
DRV_KMODS       += fcsm
DRV_KMODS       += fp
DRV_KMODS       += qlc
DRV_KMODS       += qlge
DRV_KMODS       += stmf
DRV_KMODS       += stmf_sbd
DRV_KMODS       += fct
DRV_KMODS       += fcoe
DRV_KMODS       += fcoet
DRV_KMODS       += fcoei
DRV_KMODS       += qlt
DRV_KMODS       += iscsit
DRV_KMODS       += pppt
DRV_KMODS       += ncall nsctl sdbc nskern sv
DRV_KMODS       += ii rdc rdcsrv rdcstub
DRV_KMODS       += iscsi
DRV_KMODS       += emlxs
DRV_KMODS       += oce
DRV_KMODS       += srpt
DRV_KMODS       += pmcs
DRV_KMODS       += pmcs8001fw

#
#       I/O framework test drivers
#
DRV_KMODS       += pshot
DRV_KMODS       += gen_drv
```



```
DRV_KMODS       += tvhci tphci tclient
DRV_KMODS       += emul64

#
# PCMCIA specific module(s)
#
DRV_KMODS       += pcs
MISC_KMODS      += busra cardbus dada pcmcia
DRV_KMODS       += pcic

# Add lvm
#
DRV_KMODS       += md
MISC_KMODS      += md_mirror md_stripe md_hotspares md_raid md_trans md_notify
MISC_KMODS      += md_sp

#
#       Exec Class Modules (/kernel/exec):
#
EXEC_KMODS      += aoutexec elfexec intpexec shbinexec javaexec

#
#       Scheduling Class Modules (/kernel/sched):
#
SCHED_KMODS     += RT TS RT_DPTBL TS_DPTBL IA FSS FX FX_DPTBL SDC

#
#       File System Modules (/kernel/fs):
#
FS_KMODS        += dev devfs fdfs fifofs hsfs lofs namefs nfs pcfs tmpfs zfs
FS_KMODS        += zut specfs udfs ufs autofs cachefs procfs sockfs mntfs
FS_KMODS        += ctfs objfs sharefs dcfs smbfs

#
#       Streams Modules (/kernel/strmod):
#
STRMOD_KMODS    += bufmod connld dedump ldterm ms pckt pfmod
STRMOD_KMODS    += pipemod ptem redirmod rpcmod rlmod telmod timod
STRMOD_KMODS    += spppasyn spppcomp
STRMOD_KMODS    += tirdwr ttcompat
STRMOD_KMODS    += usbkbm usbms usbwcm usb_ah
STRMOD_KMODS    += drcompat
STRMOD_KMODS    += cryptmod
STRMOD_KMODS    += vuid3ps2

#
#       'System' Modules (/kernel/sys):
#
SYS_KMODS       += c2audit
SYS_KMODS       += exacctsys
SYS_KMODS       += inst_sync kaio msgsys semsys shmsys sysacct pipe
SYS_KMODS       += doorfs pset acctctl portfs

#
#       'User' Modules (/kernel/misc):
#
MISC_KMODS      += ac97
MISC_KMODS      += bignum
MISC_KMODS      += consconfig gld ipc nfs_dlboot nfssrv scsi
MISC_KMODS      += strplumb swapgeneric tlimod
MISC_KMODS      += rpcsec rpcsec_gss kgssapi kmech_dummy
MISC_KMODS      += kmech_krb5
MISC_KMODS      += fssnap_if
MISC_KMODS      += hidparser kbtrans usba usba10 usbs49_fw
MISC_KMODS      += s1394
MISC_KMODS      += hpcsvc pcihp
```

```
MISC_KMODS      += rsmops
MISC_KMODS      += kcf
MISC_KMODS      += ksocket
MISC_KMODS      += ibcm
MISC_KMODS      += ibdm
MISC_KMODS      += ibdma
MISC_KMODS      += ibmf
MISC_KMODS      += ibtl
MISC_KMODS      += sol_ofs
MISC_KMODS      += idm
MISC_KMODS      += idmap
MISC_KMODS      += hook
MISC_KMODS      += neti
MISC_KMODS      += ctf
MISC_KMODS      += mac dls
MISC_KMODS      += cmlb
MISC_KMODS      += tem
MISC_KMODS      += pcicfg fcodem fcpci
MISC_KMODS      += scsi_vhci_f_sym scsi_vhci_f_tpgs scsi_vhci_f_asym_sun
MISC_KMODS      += scsi_vhci_f_sym_hds
MISC_KMODS      += scsi_vhci_f_tape scsi_vhci_f_tpgs_tape
MISC_KMODS      += fctl
MISC_KMODS      += emlxs_fw
MISC_KMODS      += qlc_fw_2200
MISC_KMODS      += qlc_fw_2300
MISC_KMODS      += qlc_fw_2400
MISC_KMODS      += qlc_fw_2500
MISC_KMODS      += qlc_fw_6322
MISC_KMODS      += qlc_fw_8100
MISC_KMODS      += spuni
MISC_KMODS      += hwa1480_fw uwba
MISC_KMODS      += mii

MISC_KMODS      += klmmod klmops

#
#       Software Cryptographic Providers (/kernel/crypto):
#
CRYPTO_KMODS    += aes
CRYPTO_KMODS    += arcfour
CRYPTO_KMODS    += blowfish
CRYPTO_KMODS    += des
CRYPTO_KMODS    += md4
CRYPTO_KMODS    += md5
CRYPTO_KMODS    += ecc
CRYPTO_KMODS    += rsa
CRYPTO_KMODS    += sha1
CRYPTO_KMODS    += sha2
CRYPTO_KMODS    += swrand

#
# IP Policy Modules (/kernel/ipp):
#
IPP_KMODS       += dlcosmk
IPP_KMODS       += flowacct
IPP_KMODS       += ipgpc
IPP_KMODS       += dscpmk
IPP_KMODS       += tokenmt
IPP_KMODS       += tswtclmt

#
# 'Dacf' modules (/kernel/dacf)
DACF_KMODS      += consconfig_dacf

#
#       SVVS Testing Modules (/kernel/strmod):
```



```
#
#       These are streams and driver modules which are not to be
#       delivered with a released system. However, during development
#       it is convenient to build and install the SVVS kernel modules.
#
SVVS_KMODS      += lmodb lmode lmodr lmodt svvslo tidg tivc tmux

#
#       Modules eXcluded from the product:
#
XMODS           +=

#
#       'Dacf' Modules (/kernel/dacf):
#
DACF_KMODS      += net_dacf

#
#       MAC-Type Plugin Modules (/kernel/mac)
#
MAC_KMODS       += mac_6to4
MAC_KMODS       += mac_ether
MAC_KMODS       += mac_ipv4
MAC_KMODS       += mac_ipv6
MAC_KMODS       += mac_wifi
MAC_KMODS       += mac_ib

#
# socketmod (kernel/socketmod)
#
SOCKET_KMODS    += sockpfp
SOCKET_KMODS    += socksctp
SOCKET_KMODS    += socksdp
SOCKET_KMODS    += sockrds
SOCKET_KMODS    += ksslf

#
#       kiconv modules (/kernel/kiconv):
#
KICONV_KMODS    += kiconv_emea kiconv_ja kiconv_ko kiconv_sc kiconv_tc

#
# Ensure that the variable member of the cpu_t (cpu_m) is defined
# for the lint builds so as not to cause lint errors during the
# global cross check.
#
$(LINTFLAGSUPPRESS)LINTFLAGS    += -D_MACHDEP -I$(UTSBASE)/sun4 \
                                   -I$(UTSBASE)/sun4u -I$(UTSBASE)/sfmmu
```

```
**********************************************************
    3055 Thu Jun 12 17:28:24 2014
new/usr/src/uts/sparc/mpt_sas3/Makefile
4546 mpt_sas needs enhancing to support LSI MPI2.5
**********************************************************
#
# 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.
#
#       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)
ROOTMODULE      = $(ROOT_DRV_DIR)/$(MODULE)
CONF_SRCDIR     = $(UTSBASE)/common/io/scsi/adapters/mpt_sas3
WARLOCK_OUT     = $(MPTSAS3_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) \
        $(UTSBASE)/sparc/warlock/scsi.ll \
        $(SCSI_FILES) \
        -l $(UTSBASE)/sparc/warlock/ddi_dki_impl.ll
        $(TOUCH) $@

%.ll: $(UTSBASE)/common/io/scsi/adapters/mpt_sas3/%.c
        $(WLCC) $(CPPFLAGS) -DDEBUG -o $@ $<
```

```
warlock_ddi.files:
        @cd $(UTSBASE)/sparc/warlock; pwd; $(MAKE) warlock

scsi.files:
        @cd $(UTSBASE)/sparc/scsi; pwd; $(MAKE) warlock

#endif /* ! codereview */
```