

new/usr/src/cmd/mdb/Makefile.common

1

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

```
1 #
2 # CDDL HEADER START
3 #
4 # The contents of this file are subject to the terms of the
5 # Common Development and Distribution License (the "License").
6 # You may not use this file except in compliance with the License.
7 #
8 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9 # or http://www.opensolaris.org/os/licensing.
10 # See the License for the specific language governing permissions
11 # and limitations under the License.
12 #
13 # When distributing Covered Code, include this CDDL HEADER in each
14 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 # If applicable, add the following below this CDDL HEADER, with the
16 # fields enclosed by brackets "[]" replaced with your own identifying
17 # information: Portions Copyright [yyyy] [name of copyright owner]
18 #
19 # CDDL HEADER END
20 #
21 # Copyright (c) 1999, 2010, Oracle and/or its affiliates. All rights reserved.
22 #
23 # MDB modules used for debugging user processes that every ISA's build
24 # subdirectory will need to build.
25 #
26
27 COMMON_MODULES_PROC = \
28     dof \
29     libavl \
30     libc \
31     libcmdutils \
32     libnvpair \
33     libproc \
34     libpython2.6 \
35     libsysevent \
36     libtopo \
37     libumem \
38     libuutil \
39     libzpool \
40     mdb_ds \
41     mdb_test
42
43 #
44 # MDB modules used for debugging user processes which are only 32-bit
45 #
46 COMMON_MODULES_PROC_32BIT = \
47     svc.configd \
48     svc.startd
49
50 #
51 # MDB modules used for debugging kernels.
52 #
53 COMMON_MODULES_KVM = \
54     arp \
55     cpc \
56     crypto \
57     dtrace \
58     emlxs \
59     fcip \
60     fcp \
61     fctl \
```

new/usr/src/cmd/mdb/Makefile.common

2

```
62     genunix \
63     hook \
64     neti \
65     idm \
66     ii \
67     ip \
68     ipc \
69     ipp \
70     krtld \
71     lofs \
72     logindmux \
73     mac \
74     md \
75     mpt_sas \
76     mpt_sas3 \
77 #endif /* ! codereview */
78     mr_sas \
79     nca \
80     nsctl \
81     nsmb \
82     pmcs \
83     ptm \
84     qlc \
85     random \
86     rdc \
87     sl394 \
88     scsi_vhci \
89     sctp \
90     sd \
91     sdbc \
92     smbfs \
93     smbfsrv \
94     sockfs \
95     specfs \
96     spps \
97     srpt \
98     stmf \
99     stmf_sbd \
100     sv \
101     ufs \
102     usba \
103     zfs
104
105 CLOSED_COMMON_MODULES_KVM = \
106     mpt \
107     nfs
108
109 include $(SRC)/Makefile.master
```



```

378         mdb_printf(" OFFLINING ");
379         break;
380     case MPTSAS_DR_ONLINE_IN_PROGRESS:
381         mdb_printf(" ONLINING ");
382         break;
383     default:
384         mdb_printf(" UNKNOWN ");
385         break;
386 }
387 mdb_printf("%d\n", ptgt->m_dups);
388 mdb_printf("%3d/%-3d %d/%d\n",
389     s->m_target[i].m_dr_timeout, m.m_offline_delay,
390     s->m_target[i].m_dr_online_dups,
391     s->m_target[i].m_dr_offline_dups);
392
393 if (verbose) {
394     mdb_inc_indent(5);
395     if ((ptgt->m_deviceinfo &
396         if ((s->m_target[i].m_deviceinfo &
397             MPI2_SAS_DEVICE_INFO_MASK_DEVICE_TYPE) ==
398             MPI2_SAS_DEVICE_INFO_FANOUT_EXPANDER)
399                 mdb_printf("Fanout expander: ");
400         if ((ptgt->m_deviceinfo &
401             if ((s->m_target[i].m_deviceinfo &
402                 MPI2_SAS_DEVICE_INFO_MASK_DEVICE_TYPE) ==
403                 MPI2_SAS_DEVICE_INFO_EDGE_EXPANDER)
404                     mdb_printf("Edge expander: ");
405         if ((ptgt->m_deviceinfo &
406             if ((s->m_target[i].m_deviceinfo &
407                 MPI2_SAS_DEVICE_INFO_MASK_DEVICE_TYPE) ==
408                 MPI2_SAS_DEVICE_INFO_END_DEVICE)
409                     mdb_printf("End device: ");
410         if ((ptgt->m_deviceinfo &
411             if ((s->m_target[i].m_deviceinfo &
412                 MPI2_SAS_DEVICE_INFO_MASK_DEVICE_TYPE) ==
413                 MPI2_SAS_DEVICE_INFO_NO_DEVICE)
414                     mdb_printf("No device ");
415     }
416
417     for (loop = 0, comma = 0;
418         loop < (sizeof (devinfo_array) /
419             sizeof (devinfo_array[0])); loop++) {
420         if (ptgt->m_deviceinfo &
421             if (s->m_target[i].m_deviceinfo &
422                 devinfo_array[loop].value) {
423             mdb_printf("%s%s",
424                 (comma ? ", " : "" ),
425                 devinfo_array[loop].text);
426             comma++;
427         }
428     }
429     mdb_printf("\n");
430
431 #if 0
432     if (ptgt->m_tgt_dip) {
433         char target_path[PATH_MAX];
434
435         if (s->m_target[i].m_tgt_dip) {
436             *target_path = 0;
437             if (construct_path((uintptr_t)
438                 ptgt->m_tgt_dip,
439                 s->m_target[i].m_tgt_dip,
440                 target_path)
441                 == DCMD_OK)
442                 mdb_printf("%s\n", target_path);
443         }
444     }
445 #endif
446     mdi_info(mp, ptgt->m_slot_num);

```

```

432         mdi_info(m, i);
433         mdb_dec_indent(5);
434     }
435 }
436
437 mdb_printf("\n");
438 mdb_printf("The smp child information\n");
439 for (psmp = (mptsas_smp_t *)krefhash_first(
440     (uintptr_t)mp->m_smp_targets);
441     psmp != NULL;
442     psmp = krefhash_next((uintptr_t)mp->m_smp_targets, psmp)) {
443     mdb_printf("\n");
444     mdb_printf("devhdl %x, sasaddress %\"PRIx64\", phymask %x\n",
445         psmp->m_devhdl, psmp->m_addr.mta_wwn,
446         psmp->m_addr.mta_phymask);
447 }
448 #endif
449 }
450
451 int
452 display_slotinfo(struct mptsas *mp, struct mptsas_slots *s)
453 {
454     display_slotinfo()
455     {
456         #if 0
457         int i, nslots;
458         struct mptsas_cmd c, *q, *slots;
459         mptsas_target_t *ptgt;
460         #endif /* ! codereview */
461         int header_output = 0;
462         int rv = DCMD_OK;
463         int slots_in_use = 0;
464         int tcmds = 0;
465         int mismatch = 0;
466         int wq, dq;
467         int ncmds = 0;
468         int saved_indent;
469         ulong_t
470
471         nslots = s->m_n_normal;
472
473         slots = mdb_alloc(sizeof (mptsas_cmd_t) * nslots, UM_SLEEP);
474
475         for (i = 0; i < nslots; i++)
476             if (s->m_slot[i]) {
477                 slots_in_use++;
478                 if (mdb_vread(&slots[i], sizeof (mptsas_cmd_t),
479                     (uintptr_t)s->m_slot[i]) == -1) {
480                     mdb_warn("couldn't read slot");
481                     s->m_slot[i] = NULL;
482                 }
483                 if ((slots[i].cmd_flags & CFLAG_CMDIOC) == 0)
484                     tcmds++;
485                 if (i != slots[i].cmd_slot)
486                     mismatch++;
487             }
488
489         for (q = mp->m_waitq, wq = 0; q; q = c.cmd_linkp, wq++)
490             for (q = m.m_waitq, wq = 0; q; q = c.cmd_linkp, wq++)
491                 if (mdb_vread(&c, sizeof (mptsas_cmd_t), (uintptr_t)q) == -1) {
492                     mdb_warn("couldn't follow m_waitq");
493                     rv = DCMD_ERR;
494                     goto exit;
495                 }
496
497         for (q = mp->m_doneq, dq = 0; q; q = c.cmd_linkp, dq++)
498             for (q = m.m_doneq, dq = 0; q; q = c.cmd_linkp, dq++)

```

```

460         if (mdb_vread(&c, sizeof (mptsas_cmd_t), (uintptr_t)q) == -1) {
461             mdb_warn("couldn't follow m_doneq");
462             rv = DCMD_ERR;
463             goto exit;
464         }

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

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

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

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

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

515         /* print the wait queue */

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

```

```

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

536         /* print the done queue */

537         for (q = mp->m_doneq; q; q = c.cmd_linkp) {
538             if (q == mp->m_doneq)
539                 for (q = m.m_doneq; q; q = c.cmd_linkp) {
540                     if (q == m.m_doneq)
541                         mdb_printf("\n");
542                     if (mdb_vread(&c, sizeof (mptsas_cmd_t), (uintptr_t)q)
543                         == -1) {
544                         mdb_warn("couldn't follow m_doneq");
545                         rv = DCMD_ERR;
546                         goto exit;
547                     }
548                     mdb_printf("%16p done n/a %4d %8x          %8x %16p ",
549                         q, c.cmd_slot, c.cmd_flags, c.cmd_pkt_flags,
550                         c.cmd_pkt);
551                     print_cdb(&c);
552                 }

553             mdb_inc_indent(saved_indent);

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

558             if (tcmts != ncmts)
559                 mdb_printf("WARNING: the total of m_target[].m_t_ncmts does "
560                     "not match the slots in use\n");

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

564             /* now check for corruptions */

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

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

579             for (q = mp->m_doneq; q; q = c.cmd_linkp) {

```

```

585     for (q = m.m_doneq; q; q = c.cmd_linkp) {
586         for (i = 0; i < nslots; i++)
587             if (s->m_slot[i] == q)
588                 mdb_printf("WARNING: m_doneq entry "
589                     "(mptsas_cmd_t) %p is in m_slot[%i]\n", q, i);
590
591         if (mdb_vread(&c, sizeof (mptsas_cmd_t), (uintptr_t)q) == -1) {
592             mdb_warn("couldn't follow m_doneq");
593             rv = DCMD_ERR;
594             goto exit;
595         }
596         if ((c.cmd_flags & CFLAG_FINISHED) == 0)
597             mdb_printf("WARNING: m_doneq entry (mptsas_cmd_t) %p "
598                 "should have CFLAG_FINISHED set\n", q);
599         if (c.cmd_flags & CFLAG_IN_TRANSPORT)
600             mdb_printf("WARNING: m_doneq entry (mptsas_cmd_t) %p "
601                 "should not have CFLAG_IN_TRANSPORT set\n", q);
602         if (c.cmd_flags & CFLAG_CMDARQ)
603             mdb_printf("WARNING: m_doneq entry (mptsas_cmd_t) %p "
604                 "should not have CFLAG_CMDARQ set\n", q);
605         if (c.cmd_flags & CFLAG_COMPLETED)
606             mdb_printf("WARNING: m_doneq entry (mptsas_cmd_t) %p "
607                 "should not have CFLAG_COMPLETED set\n", q);
608     }
609
610 exit:
611     mdb_free(slots, sizeof (mptsas_cmd_t) * nslots);
612     return (rv);
613 #endif
614
615     mdb_printf("\n");
616     mdb_printf("The slot information is not implemented yet\n");
617     return (0);
618 }
619
620 void
621 display_deviceinfo(struct mptsas *mp)
622 {
623     char    device_path[PATH_MAX];
624
625     *device_path = 0;
626     if (construct_path((uintptr_t)mp->m_dip, device_path) != DCMD_OK) {
627         strcpy(device_path, "couldn't determine device path");
628     }
629
630     mdb_printf("\n");
631     mdb_printf("Path in device tree %s\n", device_path);
632 #if 0
633     mdb_printf("base_wwid      phys "
634         " prodid devid      revid ssid\n");
635     "mptid prodid devid      revid ssid\n";
636     mdb_printf("-----\n");
637     "-----\n");
638     mdb_printf("%"PRIx64"      %2d "
639         "0x%04x 0x%04x ", mp->un.m_base_wwid, mp->m_num_phys,
640         mp->m_productid, mp->m_devid);
641     switch (mp->m_devid) {
642         case MPI2_MFGPAGE_DEVID_SAS2004:
643             mdb_printf("(SAS2004) ");
644             break;
645         case MPI2_MFGPAGE_DEVID_SAS2008:
646             mdb_printf("(SAS2008) ");
647             break;
648         case MPI2_MFGPAGE_DEVID_SAS2108_1:
649         case MPI2_MFGPAGE_DEVID_SAS2108_2:
650         case MPI2_MFGPAGE_DEVID_SAS2108_3:
651             mdb_printf("(SAS2108) ");
652     }
653 #endif

```

```

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             break;
652     /* Same as 2308_1/2 ?? */
653     case MPI2_MFGPAGE_DEVID_SAS2208_7:
654     case MPI2_MFGPAGE_DEVID_SAS2208_8:
655         break;
656 #endif
657     mdb_printf("(SAS2208) ");
658     mdb_printf("%"PRIx64"      %2d %3d "
659         "0x%04x 0x%04x ", m.un.m_base_wwid, m.m_num_phys, m.m_mptid,
660         m.m_productid, m.m_devid);
661     switch (m.m_devid) {
662         case MPTSAS_909:
663             mdb_printf("(909) ");
664             break;
665         case MPTSAS_929:
666             mdb_printf("(929) ");
667             break;
668         case MPTSAS_919:
669             mdb_printf("(919) ");
670             break;
671         case MPTSAS_1030:
672             mdb_printf("(1030) ");
673             break;
674         case MPTSAS_1064:
675             mdb_printf("(1064) ");
676             break;
677         case MPTSAS_1068:
678             mdb_printf("(1068) ");
679             break;
680         case MPTSAS_1064E:
681             mdb_printf("(1064E) ");
682             break;
683         case MPTSAS_1068E:
684             mdb_printf("(1068E) ");
685             break;
686         default:
687             mdb_printf("(SAS????) ");
688             mdb_printf("(?????) ");
689             break;
690     }
691     mdb_printf("0x%02x 0x%04x\n", mp->m_revid, mp->m_ssid);
692     mdb_printf("0x%02x 0x%04x\n", m.m_revid, m.m_ssid);
693     mdb_printf("%s\n", device_path);
694
695 #if 0
696 #endif /* ! codereview */
697     for (i = 0; i < MAX_MPI2_PORTS; i++) {
698         if (i%4 == 0)
699             mdb_printf("\n");
700
701         mdb_printf("%d:", i);
702
703         switch (mp->m_port_type[i]) {
704             case MPI2_PORTFACTS_PORTTYPE_INACTIVE:
705                 mdb_printf("inactive ",

```

```

676         mp->m_protocol_flags[i]);
677         m.m_protocol_flags[i]);
678         break;
679         case MPI2_PORTFACTS_PORTTYPE_SCSI:
680             mdb_printf("SCSI (0x%lx) ",
681                 mp->m_protocol_flags[i]);
682             m.m_protocol_flags[i]);
683             break;
684             case MPI2_PORTFACTS_PORTTYPE_FC:
685                 mdb_printf("FC (0x%lx) ",
686                     mp->m_protocol_flags[i]);
687                     m.m_protocol_flags[i]);
688                     break;
689                     case MPI2_PORTFACTS_PORTTYPE_ISCSI:
690                         mdb_printf("iSCSI (0x%lx) ",
691                             mp->m_protocol_flags[i]);
692                             m.m_protocol_flags[i]);
693                             break;
694                             case MPI2_PORTFACTS_PORTTYPE_SAS:
695                                 mdb_printf("SAS (0x%lx) ",
696                                     mp->m_protocol_flags[i]);
697                                     m.m_protocol_flags[i]);
698                                     break;
699                                     default:
700                                         mdb_printf("unknown ");
701                                         }
702             }
703             mdb_printf("\n");
704 #endif /* ! codereview */
705 #endif
706 }
707
708 void
709 dump_debug_log(struct mptsas *mp)
710 {
711     uint_t idx;
712     char *logbuf;
713     int i;
714
715     if (mdb_readsym(&idx, sizeof (uint_t),
716         "mptsas_dbglog_idx") == -1) {
717         mdb_warn("No debug log buffer present");
718         return;
719     }
720     logbuf = mdb_alloc(16*256, UM_SLEEP);
721     if (idx == 0) {
722         mdb_warn("Logging turned off");
723         return;
724     }
725     if (mdb_readsym(logbuf, 16*256,
726         "mptsas_dbglog_bufs") == -1) {
727         mdb_warn("No debug log buffer present");
728         return;
729     }
730     for (i = 0; i < 16; i++) {
731         mdb_printf("%s\n", &logbuf[idx*256]);
732         idx = (idx+1) & 0xf;
733     }
734     mdb_free(logbuf, 16*256);
735 #endif /* ! codereview */
736 }

```

```

737 static int
738 mptsas_dcmd(uintptr_t addr, uint_t flags, int argc, const mdb_arg_t *argv)
739 {
740     struct mptsas m;
741     struct mptsas_slots *s;
742
743     int nslots;
744     int slot_size = 0;
745     uint_t verbose = FALSE;
746     uint_t target_info = FALSE;
747     uint_t slot_info = FALSE;
748     uint_t device_info = FALSE;
749     uint_t port_info = FALSE;
750     uint_t debug_log = FALSE;
751 #endif /* ! codereview */
752     int rv = DCMD_OK;
753     void *mptsas_state;
754
755     if (!(flags & DCMD_ADDRSPEC)) {
756         void *mptsas_state = NULL;
757
758         mptsas_state = NULL;
759         if (mdb_readvar(&mptsas_state, "mptsas_state") == -1) {
760             mdb_warn("can't read mptsas_state");
761             return (DCMD_ERR);
762         }
763         if (mdb_pwalk_dcmd("genunix'softstate",
764             "mpt_sas'mptsas", argc,
765             if (mdb_pwalk_dcmd("genunix'softstate", "mpt_sas'mptsas", argc,
766                 argv, (uintptr_t)mptsas_state) == -1) {
767                     mdb_warn("mdb_pwalk_dcmd failed");
768                     return (DCMD_ERR);
769                 }
770                 return (DCMD_OK);
771             }
772
773     if (mdb_getopts(argc, argv,
774         's', MDB_OPT_SETBITS, TRUE, &slot_info,
775         'd', MDB_OPT_SETBITS, TRUE, &device_info,
776         't', MDB_OPT_SETBITS, TRUE, &target_info,
777         'p', MDB_OPT_SETBITS, TRUE, &port_info,
778         'v', MDB_OPT_SETBITS, TRUE, &verbose,
779         'D', MDB_OPT_SETBITS, TRUE, &debug_log,
780 #endif /* ! codereview */
781         NULL) != argc)
782         return (DCMD_USAGE);
783
784     if (mdb_vread(&m, sizeof (m), addr) == -1) {
785         mdb_warn("couldn't read mpt struct at 0x%p", addr);
786         return (DCMD_ERR);
787     }
788
789     s = mdb_alloc(sizeof (mptsas_slots_t), UM_SLEEP);
790
791     if (mdb_vread(s, sizeof (mptsas_slots_t),
792         (uintptr_t)m.m_active) == -1) {
793         mdb_warn("couldn't read small mptsas_slots_t at 0x%p",
794             m.m_active);
795         mdb_free(s, sizeof (mptsas_slots_t));
796         return (DCMD_ERR);
797     }
798
799     nslots = s->m_n_normal;
800     mdb_free(s, sizeof (mptsas_slots_t));

```

```

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

804     s = mdb_alloc(slot_size, UM_SLEEP);

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

813     /* processing completed */

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

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

847     mdb_inc_indent(17);

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

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

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

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

861     if (debug_log)
862         dump_debug_log(&m);
863     display_slotinfo();

```

```

864         mdb_dec_indent(17);

866         mdb_free(s, slot_size);

868         return (rv);
869     }

871 void
872 mptsas_help()
873 {
874     mdb_printf("Prints summary information about each mpt_sas instance, "
875         "including warning\nmessages when slot usage doesn't match "
876         "summary information.\n"
877         "Without the address of a \"struct mptsas\", prints every "
878         "instance.\n\n"
879         "Switches:\n"
880         "  -t[v] includes information about targets, v = be more verbose\n"
881         "  -t includes information about targets\n"
882         "  -p includes information about port\n"
883         "  -s includes information about mpt slots\n"
884         "  -d includes information about the hardware\n"
885         "  -D print the mptsas specific debug log\n"
886         "  -d includes information about the hardware\n");
887 }

887 static const mdb_dcmd_t dcmds[] = {
888     { "mptsas", "?[-tpsdd]", "print mpt_sas information", mptsas_dcmd,
889     { "mptsas", "?[-tpd]", "print mpt_sas information", mptsas_dcmd,
890     mptsas_help, { NULL }
891     }
892 };

```

unchanged_portion_omitted

new/usr/src/cmd/mdb/common/modules/mpt_sas3/mpt_sas3.c

1

```
*****
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
*****

1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */
21 /*
22 * Copyright 2009 Sun Microsystems, Inc. All rights reserved.
23 * Use is subject to license terms.
24 */

26 /*
27 * Copyright 2014 Joyent, Inc. All rights reserved.
28 * Copyright (c) 2014, Tegile Systems Inc. All rights reserved.
29 */

31 #include <limits.h>
32 #include <sys/mdb_modapi.h>
33 #include <sys/sysinfo.h>
34 #include <sys/sunmdi.h>
35 #include <sys/list.h>
36 #include <sys/scsi/scsi.h>

38 #pragma pack(1)
39 #include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_type.h>
40 #include <sys/scsi/adapters/mpt_sas3/mpi/mpi2.h>
41 #include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_cnfg.h>
42 #include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_init.h>
43 #include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_ioc.h>
44 #include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_sas.h>
45 #include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_raid.h>
46 #include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_tool.h>
47 #pragma pack()

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

52 struct {
53     int     value;
54     char    *text;
55 } devinfo_array[] = {
56     { MPI2_SAS_DEVICE_INFO_SEP, "SEP" },
57     { MPI2_SAS_DEVICE_INFO_ATAPI_DEVICE, "ATAPI device" },
58     { MPI2_SAS_DEVICE_INFO_LSI_DEVICE, "LSI device" },
59     { MPI2_SAS_DEVICE_INFO_DIRECT_ATTACH, "direct attach" },
60     { MPI2_SAS_DEVICE_INFO_SSP_TARGET, "SSP tgt" },
61     { MPI2_SAS_DEVICE_INFO_STP_TARGET, "STP tgt" },
```

new/usr/src/cmd/mdb/common/modules/mpt_sas3/mpt_sas3.c

2

```
62     { MPI2_SAS_DEVICE_INFO_SMP_TARGET, "SMP tgt" },
63     { MPI2_SAS_DEVICE_INFO_SATA_DEVICE, "SATA dev" },
64     { MPI2_SAS_DEVICE_INFO_SSP_INITIATOR, "SSP init" },
65     { MPI2_SAS_DEVICE_INFO_STP_INITIATOR, "STP init" },
66     { MPI2_SAS_DEVICE_INFO_SMP_INITIATOR, "SMP init" },
67     { MPI2_SAS_DEVICE_INFO_SATA_HOST, "SATA host" }
68 };

70 int
71 construct_path(uintptr_t addr, char *result)
72 {
73     struct dev_info d;
74     char dev_node[PATH_MAX];
75     char dev_addr[PATH_MAX];

77     if (mdb_vread(&d, sizeof(d), addr) == -1) {
78         mdb_warn("couldn't read dev_info");
79         return (DCMD_ERR);
80     }

82     if (d.devi_parent) {
83         construct_path((uintptr_t)d.devi_parent, result);
84         mdb_readstr(devi_node, sizeof(devi_node),
85             (uintptr_t)d.devi_node_name);
86         mdb_readstr(devi_addr, sizeof(devi_addr),
87             (uintptr_t)d.devi_addr);
88         mdb_snprintf(result+strlen(result),
89             PATH_MAX-strlen(result),
90             "%s%s", devi_node, (*devi_addr ? "@" : ""),
91             devi_addr);
92     }
93     return (DCMD_OK);
94 }

96 /* ARGSUSED */
97 int
98 mdi_info_cb(uintptr_t addr, const void *data, void *cbdata)
99 {
100     struct mdi_pathinfo pi;
101     struct mdi_client c;
102     char dev_path[PATH_MAX];
103     char string[PATH_MAX];
104     int mdi_target = 0, mdi_lun = 0;
105     int target = *(int *)cbdata;

107     if (mdb_vread(&pi, sizeof(pi), addr) == -1) {
108         mdb_warn("couldn't read mdi_pathinfo");
109         return (DCMD_ERR);
110     }
111     mdb_readstr(string, sizeof(string), (uintptr_t)pi.pi_addr);
112     mdi_target = (int)mdb_strtoul(string);
113     mdi_lun = (int)mdb_strtoul(strchr(string, ',') + 1);
114     if (target != mdi_target)
115         return (0);

117     if (mdb_vread(&c, sizeof(c), (uintptr_t)pi.pi_client) == -1) {
118         mdb_warn("couldn't read mdi_client");
119         return (-1);
120     }

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

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



```

128     (pi.pi_preferred ? "preferred" : "");
129     switch (pi.pi_state & MDI_PATHINFO_STATE_MASK) {
130     case MDI_PATHINFO_STATE_INIT:
131         mdb_printf(" initializing");
132         break;
133     case MDI_PATHINFO_STATE_ONLINE:
134         mdb_printf(" online");
135         break;
136     case MDI_PATHINFO_STATE_STANDBY:
137         mdb_printf(" standby");
138         break;
139     case MDI_PATHINFO_STATE_FAULT:
140         mdb_printf(" fault");
141         break;
142     case MDI_PATHINFO_STATE_OFFLINE:
143         mdb_printf(" offline");
144         break;
145     default:
146         mdb_printf(" invalid state");
147         break;
148     }
149     mdb_printf("\n");
150     return (0);
151 }

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

159     if (mdb_vread(&d, sizeof (d), (uintptr_t)mp->m_dip) == -1) {
160         mdb_warn("couldn't read m_dip");
161         return;
162     }

164     if (MDI_PHCI(&d)) {
165         if (mdb_vread(&p, sizeof (p), (uintptr_t)d.devi_mdi_xhci)
166             == -1) {
167             mdb_warn("couldn't read m_dip.devi_mdi_xhci");
168             return;
169         }
170         if (p.ph_path_head)
171             mdb_pwalk("mdipi_phci_list", (mdb_walk_cb_t)mdi_info_cb,
172                 &target, (uintptr_t)p.ph_path_head);
173         return;
174     }
175 }

177 void
178 print_cdb(mptsas_cmd_t *m)
179 {
180     struct scsi_pkt      pkt;
181     uchar_t cdb[512];    /* an arbitrarily large number */
182     int          j;

184     if (mdb_vread(&pkt, sizeof (pkt), (uintptr_t)m->cmd_pkt) == -1) {
185         mdb_warn("couldn't read cmd_pkt");
186         return;
187     }

189     /*
190     * We use cmd_cdblen here because 5.10 doesn't
191     * have the cdb length in the pkt
192     */
193     if (mdb_vread(&cdb, m->cmd_cdblen, (uintptr_t)pkt.pkt_cdbp) == -1) {

```

```

194         mdb_warn("couldn't read pkt_cdbp");
195         return;
196     }

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

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

204     mdb_printf("]\n");
205 }

208 void
209 display_ports(struct mptsas *mp)
210 {
211     int i;
212     mdb_printf("\n");
213     mdb_printf("phy number and port mapping table\n");
214     for (i = 0; i < MPTSAS_MAX_PHYS; i++) {
215         if (mp->m_phy_info[i].attached_devhdl) {
216             mdb_printf("phy %x --> port %x, phymask %x,"
217                 "attached_devhdl %x\n", i, mp->m_phy_info[i].port_num,
218                 mp->m_phy_info[i].phy_mask,
219                 mp->m_phy_info[i].attached_devhdl);
220         }
221     }
222     mdb_printf("\n");
223 }

225 static uintptr_t
226 klist_head(list_t *lp, uintptr_t klp)
227 {
228     if ((uintptr_t)lp->list_head.list_next ==
229         klp + offsetof(struct list, list_head))
230         return (NULL);

232     return ((uintptr_t)(((char *)lp->list_head.list_next) -
233         lp->list_offset));
234 }

236 static uintptr_t
237 klist_next(list_t *lp, uintptr_t klp, void *op)
238 {
239     /* LINTED E_BAD_PTR_CAST_ALIG */
240     struct list_node *np = (struct list_node *)(((char *)op) +
241         lp->list_offset);

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

246     return (((uintptr_t)(np->list_next)) - lp->list_offset);
247 }

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

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

```

```

260         return (NULL);

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

268     return (rp);
269 }

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

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

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

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

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

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

316     mdb_printf("\n");
317     mdb_printf(" mptsas_target_t slot devhdl      wwn      ncmts throttle  "
318         "dr_flag dups\n");
319     mdb_printf("-----\n");
320     "-----\n");
321     for (ptgt = (mptsas_target_t *)krefhash_first(
322         (uintptr_t)mp->m_targets, &p_addr);
323         ptgt != NULL;
324         ptgt = krefhash_next((uintptr_t)mp->m_targets, ptgt, &p_addr)) {
325         if (ptgt->m_addr.mta_wnn ||

```

```

326     ptgt->m_deviceinfo) {
327         mdb_printf("%16p ", p_addr);
328         mdb_printf("%4d ", ptgt->m_slot_num);
329         mdb_printf("%4d ", ptgt->m_devhdl);
330         if (ptgt->m_addr.mta_wnn)
331             mdb_printf("%"PRIx64" ",
332                 ptgt->m_addr.mta_wnn);
333         mdb_printf("%3d", ptgt->m_t_ncmts);
334         switch (ptgt->m_t_throttle) {
335             case QFULL_THROTTLE:
336                 mdb_printf(" QFULL ");
337                 break;
338             case DRAIN_THROTTLE:
339                 mdb_printf(" DRAIN ");
340                 break;
341             case HOLD_THROTTLE:
342                 mdb_printf(" HOLD ");
343                 break;
344             case MAX_THROTTLE:
345                 mdb_printf(" MAX ");
346                 break;
347             default:
348                 mdb_printf("%8d ",
349                     ptgt->m_t_throttle);
350         }
351         switch (ptgt->m_dr_flag) {
352             case MPTSAS_DR_INACTIVE:
353                 mdb_printf(" INACTIVE ");
354                 break;
355             case MPTSAS_DR_INTRANSITION:
356                 mdb_printf("TRANSITION ");
357                 break;
358             default:
359                 mdb_printf(" UNKNOWN ");
360                 break;
361         }
362         mdb_printf("%d\n",
363             ptgt->m_dups);

365     if (verbose) {
366         mdb_inc_indent(5);
367         if ((ptgt->m_deviceinfo &
368             MPI2_SAS_DEVICE_INFO_MASK_DEVICE_TYPE) ==
369             MPI2_SAS_DEVICE_INFO_FANOUT_EXPANDER)
370             mdb_printf("Fanout expander: ");
371         if ((ptgt->m_deviceinfo &
372             MPI2_SAS_DEVICE_INFO_MASK_DEVICE_TYPE) ==
373             MPI2_SAS_DEVICE_INFO_EDGE_EXPANDER)
374             mdb_printf("Edge expander: ");
375         if ((ptgt->m_deviceinfo &
376             MPI2_SAS_DEVICE_INFO_MASK_DEVICE_TYPE) ==
377             MPI2_SAS_DEVICE_INFO_END_DEVICE)
378             mdb_printf("End device: ");
379         if ((ptgt->m_deviceinfo &
380             MPI2_SAS_DEVICE_INFO_MASK_DEVICE_TYPE) ==
381             MPI2_SAS_DEVICE_INFO_NO_DEVICE)
382             mdb_printf("No device ");

384     for (loop = 0, comma = 0;
385         loop < (sizeof (devinfo_array) /
386             sizeof (devinfo_array[0])); loop++) {
387         if (ptgt->m_deviceinfo &
388             devinfo_array[loop].value) {
389             mdb_printf("%s%s",
390                 (comma ? ", " : ""),
391                 devinfo_array[loop].text);

```

```

392                                     comma++;
393                                     }
394                                     }
395                                     mdb_printf("\n");
396                                     mdi_info(mp, ptgt->m_slot_num);
397                                     mdb_dec_indent(5);
398                             }
399                     }
400             }

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

434         for (loop = 0, comma = 0;
435             loop < (sizeof (devinfo_array) /
436                 sizeof (devinfo_array[0])); loop++) {
437             if (psmp->m_deviceinfo &
438                 devinfo_array[loop].value) {
439                 mdb_printf("%s%s",
440                     comma ? ", " : "",
441                     devinfo_array[loop].text);
442                 comma++;
443             }
444         }
445         mdb_printf("\n");
446         mdb_dec_indent(5);
447     }
448 }
449 }

451 int
452 display_slotinfo(struct mptsas *mp, struct mptsas_slots *s)
453 {
454     int i, nslots;
455     struct mptsas_cmd c, *q, *slots;
456     mptsas_target_t *ptgt;
457     int header_output = 0;

```

```

458     int rv = DCMD_OK;
459     int slots_in_use = 0;
460     int tcmds = 0;
461     int mismatch = 0;
462     int wq, dq;
463     int ncmds = 0;
464     ulong_t saved_indent;

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

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

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

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

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

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

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

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

```

```

525     for (i = 0; i < s->m_n_normal; i++)
526     if (s->m_slot[i]) {
527         if (!header_output) {
528             mdb_printf("\n");
529             mdb_printf("mptsas_cmd      slot cmd_slot "
530                 "cmd_flags cmd_pkt_flags scsi_pkt "
531                 "targ,lun [ pkt_cdbp ...]\n");
532             mdb_printf("-----\n");
533             mdb_printf("-----\n");
534             mdb_printf("-----\n");
535             header_output = 1;
536         }
537         mdb_printf("%16p %4d %s %4d %8x      %8x %16p ",
538             s->m_slot[i], i,
539             (i == slots[i].cmd_slot ? "BAD",
540             slots[i].cmd_slot,
541             slots[i].cmd_flags,
542             slots[i].cmd_pkt_flags,
543             slots[i].cmd_pkt);
544             (void) print_cdb(&slots[i]);
545         }
546
547 /* print the wait queue */
548
549 for (q = mp->m_waitq; q; q = c.cmd_linkp) {
550     if (q == mp->m_waitq)
551         mdb_printf("\n");
552     if (mdb_vread(&c, sizeof (mptsas_cmd_t), (uintptr_t)q)
553         == -1) {
554         mdb_warn("couldn't follow m_waitq");
555         rv = DCMD_ERR;
556         goto exit;
557     }
558     mdb_printf("%16p wait n/a %4d %8x      %8x %16p ",
559         q, c.cmd_slot, c.cmd_flags, c.cmd_pkt_flags,
560         c.cmd_pkt);
561     print_cdb(&c);
562 }
563
564 /* print the done queue */
565
566 for (q = mp->m_dlist.dl_q; q; q = c.cmd_linkp) {
567     if (q == mp->m_dlist.dl_q)
568         mdb_printf("\n");
569     if (mdb_vread(&c, sizeof (mptsas_cmd_t), (uintptr_t)q)
570         == -1) {
571         mdb_warn("couldn't follow m_doneq");
572         rv = DCMD_ERR;
573         goto exit;
574     }
575     mdb_printf("%16p done n/a %4d %8x      %8x %16p ",
576         q, c.cmd_slot, c.cmd_flags, c.cmd_pkt_flags,
577         c.cmd_pkt);
578     print_cdb(&c);
579 }
580
581 mdb_inc_indent(saved_indent);
582
583 if (mp->m_ncmds != slots_in_use)
584     mdb_printf("WARNING: mpt.m_ncmds does not match the number of "
585         "slots in use\n");
586
587 if (tcmds != ncmds)
588     mdb_printf("WARNING: the total of m_target[].m_t_ncmds does "

```

```

590         "not match the slots in use\n");
591
592 if (mismatch)
593     mdb_printf("WARNING: corruption in slot table, "
594         "m_slot[].cmd_slot incorrect\n");
595
596 /* now check for corruptions */
597
598 for (q = mp->m_waitq; q; q = c.cmd_linkp) {
599     for (i = 0; i < nslots; i++)
600         if (s->m_slot[i] == q)
601             mdb_printf("WARNING: m_waitq entry "
602                 "(mptsas_cmd_t) %p is in m_slot[%i]\n",
603                 q, i);
604
605     if (mdb_vread(&c, sizeof (mptsas_cmd_t), (uintptr_t)q) == -1) {
606         mdb_warn("couldn't follow m_waitq");
607         rv = DCMD_ERR;
608         goto exit;
609     }
610 }
611
612 for (q = mp->m_dlist.dl_q; q; q = c.cmd_linkp) {
613     for (i = 0; i < nslots; i++)
614         if (s->m_slot[i] == q)
615             mdb_printf("WARNING: m_doneq entry "
616                 "(mptsas_cmd_t) %p is in m_slot[%i]\n", q, i);
617
618     if (mdb_vread(&c, sizeof (mptsas_cmd_t), (uintptr_t)q) == -1) {
619         mdb_warn("couldn't follow m_doneq");
620         rv = DCMD_ERR;
621         goto exit;
622     }
623     if ((c.cmd_flags & CFLAG_FINISHED) == 0)
624         mdb_printf("WARNING: m_doneq entry (mptsas_cmd_t) %p "
625             "should have CFLAG_FINISHED set\n", q);
626     if (c.cmd_flags & CFLAG_IN_TRANSPORT)
627         mdb_printf("WARNING: m_doneq entry (mptsas_cmd_t) %p "
628             "should not have CFLAG_IN_TRANSPORT set\n", q);
629     if (c.cmd_flags & CFLAG_CMDARQ)
630         mdb_printf("WARNING: m_doneq entry (mptsas_cmd_t) %p "
631             "should not have CFLAG_CMDARQ set\n", q);
632     if (c.cmd_flags & CFLAG_COMPLETED)
633         mdb_printf("WARNING: m_doneq entry (mptsas_cmd_t) %p "
634             "should not have CFLAG_COMPLETED set\n", q);
635 }
636
637 exit:
638     mdb_free(slots, sizeof (mptsas_cmd_t) * nslots);
639     return (rv);
640 }
641
642 void
643 display_deviceinfo(struct mptsas *mp)
644 {
645     char    device_path[PATH_MAX];
646
647     *device_path = 0;
648     if (construct_path((uintptr_t)mp->m_dip, device_path) != DCMD_OK) {
649         strcpy(device_path, "couldn't determine device path");
650     }
651
652     mdb_printf("\n");
653     mdb_printf("base_wwid      phys "
654         "prodid devid      revid  ssid\n");
655     mdb_printf("-----\n");

```

```

656     "-----\n");
657     mdb_printf("%PRIx64"    %2d  "
658     "0x%04x 0x%04x ", mp->un.m_base_wwid, mp->m_num_phys,
659     mp->m_productid, mp->m_devid);
660     switch (mp->m_devid) {
661         case MPI2_MFGPAGE_DEVID_SAS2004:
662             mdb_printf("(SAS2004) ");
663             break;
664         case MPI2_MFGPAGE_DEVID_SAS2008:
665             mdb_printf("(SAS2008) ");
666             break;
667         case MPI2_MFGPAGE_DEVID_SAS2108_1:
668         case MPI2_MFGPAGE_DEVID_SAS2108_2:
669         case MPI2_MFGPAGE_DEVID_SAS2108_3:
670             mdb_printf("(SAS2108) ");
671             break;
672         case MPI2_MFGPAGE_DEVID_SAS2116_1:
673         case MPI2_MFGPAGE_DEVID_SAS2116_2:
674             mdb_printf("(SAS2116) ");
675             break;
676         case MPI2_MFGPAGE_DEVID_SSS6200:
677             mdb_printf("(SSS6200) ");
678             break;
679         case MPI2_MFGPAGE_DEVID_SAS2208_1:
680         case MPI2_MFGPAGE_DEVID_SAS2208_2:
681         case MPI2_MFGPAGE_DEVID_SAS2208_3:
682         case MPI2_MFGPAGE_DEVID_SAS2208_4:
683         case MPI2_MFGPAGE_DEVID_SAS2208_5:
684         case MPI2_MFGPAGE_DEVID_SAS2208_6:
685     #if 0
686         /* Same as 2308_1/2 ?? */
687         case MPI2_MFGPAGE_DEVID_SAS2208_7:
688         case MPI2_MFGPAGE_DEVID_SAS2208_8:
689     #endif
690             mdb_printf("(SAS2208) ");
691             break;
692         case MPI2_MFGPAGE_DEVID_SAS2308_1:
693         case MPI2_MFGPAGE_DEVID_SAS2308_2:
694         case MPI2_MFGPAGE_DEVID_SAS2308_3:
695             mdb_printf("(SAS2308) ");
696             break;
697         case MPI25_MFGPAGE_DEVID_SAS3004:
698             mdb_printf("(SAS3004) ");
699             break;
700         case MPI25_MFGPAGE_DEVID_SAS3008:
701             mdb_printf("(SAS3008) ");
702             break;
703         case MPI25_MFGPAGE_DEVID_SAS3108_1:
704         case MPI25_MFGPAGE_DEVID_SAS3108_2:
705         case MPI25_MFGPAGE_DEVID_SAS3108_3:
706         case MPI25_MFGPAGE_DEVID_SAS3108_4:
707         case MPI25_MFGPAGE_DEVID_SAS3108_5:
708         case MPI25_MFGPAGE_DEVID_SAS3108_6:
709             mdb_printf("(SAS3108) ");
710             break;
711         default:
712             mdb_printf("(SAS????) ");
713             break;
714     }
715     mdb_printf("0x%02x 0x%04x\n", mp->m_revid, mp->m_ssid);
716     mdb_printf("%s\n", device_path);
717 }

719 void
720 dump_debug_log(struct mptsas *mp)
721 {

```

```

722     uint_t  idx;
723     char    *logbuf;
724     int      i;

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

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

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

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

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

770         if (mdb_readvar(&mptsas3_state, "mptsas3_state") == -1) {
771             mdb_warn("can't read mptsas3_state");
772             return (DCMD_ERR);
773         }
774         if (mdb_pwalk_dcmd("genunix'softstate",
775         "mpt_sas3'mptsas3", argc,
776         argv, (uintptr_t)mptsas3_state) == -1) {
777             mdb_warn("mdb_pwalk_dcmd failed");
778             return (DCMD_ERR);
779         }
780         return (DCMD_OK);
781     }

783     if (mdb_getopts(argc, argv,
784     's', MDB_OPT_SETBITS, TRUE, &slot_info,
785     'd', MDB_OPT_SETBITS, TRUE, &device_info,
786     't', MDB_OPT_SETBITS, TRUE, &target_info,
787     'p', MDB_OPT_SETBITS, TRUE, &port_info,

```

```

788     'v', MDB_OPT_SETBITS, TRUE, &verbose,
789     'D', MDB_OPT_SETBITS, TRUE, &debug_log,
790     NULL) != argc)
791         return (DCMD_USAGE);

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

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

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

809     nslots = s->m_n_normal;

811     mdb_free(s, sizeof (mptsas_slots_t));

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

816     s = mdb_alloc(slot_size, UM_SLEEP);

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

825     /* processing completed */

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

839     mdb_printf("%16p %4d %5d ", addr, m.m_instance, m.m_ncmds);
840     mdb_printf("%7d", m.m_suspended);
841     switch (m.m_power_level) {
842     case PM_LEVEL_D0:
843         mdb_printf(" ON=D0 ");
844         break;
845     case PM_LEVEL_D1:
846         mdb_printf("  D1 ");
847         break;
848     case PM_LEVEL_D2:
849         mdb_printf("  D2 ");
850         break;
851     case PM_LEVEL_D3:
852         mdb_printf("OFF=D3 ");
853         break;

```

```

854         default:
855             mdb_printf("INVALID ");
856     }
857     mdb_printf("\n");

859     mdb_inc_indent(17);

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

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

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

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

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

876     mdb_dec_indent(17);

878     mdb_free(s, slot_size);

880     return (rv);
881 }

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

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

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

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

```

new/usr/src/cmd/mdb/intel/amd64/mpt_sas3/Makefile

1

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

```
1 #
2 # CDDL HEADER START
3 #
4 # The contents of this file are subject to the terms of the
5 # Common Development and Distribution License (the "License").
6 # You may not use this file except in compliance with the License.
7 #
8 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9 # or http://www.opensolaris.org/os/licensing.
10 # See the License for the specific language governing permissions
11 # and limitations under the License.
12 #
13 # When distributing Covered Code, include this CDDL HEADER in each
14 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 # If applicable, add the following below this CDDL HEADER, with the
16 # fields enclosed by brackets "[]" replaced with your own identifying
17 # information: Portions Copyright [yyyy] [name of copyright owner]
18 #
19 # CDDL HEADER END
20 #
21 #
22 # Copyright 2009 Sun Microsystems, Inc. All rights reserved.
23 # Use is subject to license terms.
24
25 MODULE = mpt_sas3.so
26 MDBTGT = kvm
27
28 MODSRCS = mpt_sas3.c
29
30 include ../../../../Makefile.cmd
31 include ../../../../Makefile.cmd.64
32 include ../../Makefile.amd64
33 include ../../../../Makefile.module
34
35 CPPFLAGS += -I$(SRC)/uts/common
36
37 CERRWARN += -_gcc=-Wno-trigraphs
38 #endif /* ! codereview */
```

new/usr/src/cmd/mdb/intel/ia32/mpt_sas3/Makefile

1

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

```
1 #
2 # CDDL HEADER START
3 #
4 # The contents of this file are subject to the terms of the
5 # Common Development and Distribution License (the "License").
6 # You may not use this file except in compliance with the License.
7 #
8 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9 # or http://www.opensolaris.org/os/licensing.
10 # See the License for the specific language governing permissions
11 # and limitations under the License.
12 #
13 # When distributing Covered Code, include this CDDL HEADER in each
14 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 # If applicable, add the following below this CDDL HEADER, with the
16 # fields enclosed by brackets "[]" replaced with your own identifying
17 # information: Portions Copyright [yyyy] [name of copyright owner]
18 #
19 # CDDL HEADER END
20 #
21 #
22 # Copyright 2009 Sun Microsystems, Inc. All rights reserved.
23 # Use is subject to license terms.

26 MODULE = mpt_sas3.so
27 MDBTGT = kvm

29 MODSRCS = mpt_sas3.c

31 include ../../../../Makefile.cmd
32 include ../../Makefile.ia32
33 include ../../../../Makefile.module

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

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


new/usr/src/cmd/mdb/sparc/v9/mpt_sas3/Makefile

1

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

```
1 #
2 # CDDL HEADER START
3 #
4 # The contents of this file are subject to the terms of the
5 # Common Development and Distribution License (the "License").
6 # You may not use this file except in compliance with the License.
7 #
8 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9 # or http://www.opensolaris.org/os/licensing.
10 # See the License for the specific language governing permissions
11 # and limitations under the License.
12 #
13 # When distributing Covered Code, include this CDDL HEADER in each
14 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 # If applicable, add the following below this CDDL HEADER, with the
16 # fields enclosed by brackets "[]" replaced with your own identifying
17 # information: Portions Copyright [yyyy] [name of copyright owner]
18 #
19 # CDDL HEADER END
20 #
21 #
22 # Copyright 2009 Sun Microsystems, Inc. All rights reserved.
23 # Use is subject to license terms.
24
25 MODULE = mpt_sas3.so
26 MDBTGT = kvm
27
28 MODSRCS = mpt_sas3.c
29
30 include ../../../../Makefile.cmd
31 include ../../../../Makefile.cmd.64
32 include ../../Makefile.sparcv9
33 include ../../../../Makefile.module
34
35 CPPFLAGS += -I$(SRC)/uts/common
36
37 CERRWARN += -_gcc=-Wno-trigraphs
38 #endif /* ! codereview */
```

new/usr/src/man/man7d/Makefile

1

4539 Thu Jun 12 17:28:21 2014

new/usr/src/man/man7d/Makefile

4546 mpt_sas needs enhancing to support LSI MPI2.5

```
1 #
2 # This file and its contents are supplied under the terms of the
3 # Common Development and Distribution License ("CDDL"), version 1.0.
4 # You may only use this file in accordance with the terms of version
5 # 1.0 of the CDDL.
6 #
7 # A full copy of the text of the CDDL should have accompanied this
8 # source. A copy of the CDDL is also available via the Internet
9 # at http://www.illumos.org/license/CDDL.
10 #
11 #
12 #
13 # Copyright 2011, Richard Lowe
14 # Copyright 2013 Nexenta Systems, Inc. All rights reserved.
15 # Copyright 2013 Garrett D'Amore <garrett@damore.org>
16 #
```

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

```
20 MANSECT= 7d
```

```
22 _MANFILES= aac.7d \
23 afe.7d \
24 audio.7d \
25 audiol575.7d \
26 audioens.7d \
27 audiols.7d \
28 audiopl6x.7d \
29 audiopci.7d \
30 audiot.7d \
31 avl394.7d \
32 bge.7d \
33 bscv.7d \
34 chxge.7d \
35 console.7d \
36 cpuid.7d \
37 dca.7d \
38 dcaml394.7d \
39 devinfo.7d \
40 dmfe.7d \
41 dtrace.7d \
42 ehci.7d \
43 fasttrap.7d \
44 fbt.7d \
45 fcip.7d \
46 fcoe.7d \
47 fcoei.7d \
48 fcoet.7d \
49 fcp.7d \
50 fctl.7d \
51 fd.7d \
52 fp.7d \
53 gld.7d \
54 hcil394.7d \
55 hermon.7d \
56 hid.7d \
57 hme.7d \
58 hubd.7d \
59 hwahc.7d \
60 hwarc.7d \
61 hxge.7d \
```

new/usr/src/man/man7d/Makefile

2

```
62 ib.7d \
63 ibcm.7d \
64 ibd.7d \
65 ibdm.7d \
66 ibdma.7d \
67 ibtl.7d \
68 ieee1394.7d \
69 igb.7d \
70 ii.7d \
71 ipnet.7d \
72 iscsi.7d \
73 iser.7d \
74 ixgbe.7d \
75 kmdb.7d \
76 kstat.7d \
77 ksyms.7d \
78 llcl.7d \
79 lockstat.7d \
80 lofi.7d \
81 log.7d \
82 md.7d \
83 mediator.7d \
84 mem.7d \
85 mpt_sas.7d \
86 mpt_sas3.7d \
87 #endif /* ! codereview */
88 mr_sas.7d \
89 msglog.7d \
90 mt.7d \
91 mxfe.7d \
92 myril0ge.7d \
93 null.7d \
94 nulldriver.7d \
95 nxge.7d \
96 ohci.7d \
97 openprom.7d \
98 pcic.7d \
99 pcmcia.7d \
100 physmem.7d \
101 pm.7d \
102 poll.7d \
103 profile.7d \
104 ptm.7d \
105 pts.7d \
106 pty.7d \
107 qlc.7d \
108 ramdisk.7d \
109 random.7d \
110 rge.7d \
111 sad.7d \
112 sata.7d \
113 scsa1394.7d \
114 scsa2usb.7d \
115 sd.7d \
116 sdp.7d \
117 sdt.7d \
118 ses.7d \
119 sfe.7d \
120 sgen.7d \
121 srpt.7d \
122 st.7d \
123 sv.7d \
124 sysmsg.7d \
125 systrace.7d \
126 ticlts.7d \
127 tty.7d \
```

```

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

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

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

```

```

194          asy.7d      \
195          ata.7d      \
196          atge.7d     \
197          ath.7d      \
198          atu.7d      \
199          audio810.7d \
200          audiocmi.7d \
201          audiocmihd.7d \
202          audioemu10k.7d \
203          audiohd.7d  \
204          audioixp.7d \
205          audiosolo.7d \
206          audiovia823x.7d \
207          audiovia97.7d \
208          bcm_sata.7d \
209          bfe.7d      \
210          cmdk.7d     \
211          cpqgary3.7d \
212          dnet.7d     \
213          ecpp.7d     \
214          heci.7d     \
215          i915.7d     \
216          ipmi.7d     \
217          ipw.7d      \
218          iwh.7d      \
219          iwi.7d      \
220          mega_sas.7d \
221          npe.7d      \
222          ntxn.7d     \
223          nv_sata.7d  \
224          pcn.7d      \
225          radeon.7d   \
226          ral.7d      \
227          rtw.7d      \
228          rum.7d      \
229          rwd.7d      \
230          rwn.7d      \
231          sda.7d      \
232          sdcards.7d  \
233          sdhost.7d   \
234          si3124.7d   \
235          smbios.7d   \
236          uath.7d     \
237          ural.7d     \
238          urtw.7d     \
239          wpi.7d      \
240          zyd.7d      \

242 _MANLINKS= 1394.7d \
243             allkmem.7d \
244             bscbus.7d \
245             fdc.7d     \
246             firewire.7d \
247             hwa1480_fw.7d \
248             i2bse.7d   \
249             kmem.7d    \
250             lo0.7d     \
251             ticots.7d  \
252             ticotsord.7d \
253             urandom.7d \
254             usb.7d     \
255             uwb.7d     \

257 sparc_MANLINKS= drmmach.7d \
258                 ngdr.7d \
259                 ngdrmmach.7d \

```

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

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

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

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

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

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

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

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

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

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

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

290 hwal480_fw.7d  := LINKSRC = wusb_df.7d

292 .KEEP_STATE:

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

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

new/usr/src/man/man7d/mpt_sas3.7d

1

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

```
1  \" te
2  .\" Copyright (c) 2009, Sun Microsystems, Inc. All Rights Reserved
3  .\" The contents of this file are subject to the terms of the Common Development
4  .\" or http://www.opensolaris.org/os/licensing. See the License for the specifi
5  .\" the following below this CDDL HEADER, with the fields enclosed by brackets "
6  .TH MPT_SAS3 7D "Jun 2, 2014"
7  .SH NAME
8  mpt_sas3 \- SAS-3 host bus adapter driver
9  .SH SYNOPSIS
10 .sp
11 .in +2
12 .nf
13 scsi@unit-address
14 .fi
15 .in -2

17 .SH DESCRIPTION
18 .sp
19 .LP
20 The \fBmpt_sas3\fR host bus adapter driver is a nexus driver that supports the
21 LSI SAS300x/3108 series of chips. These chips support SAS/SATA interfaces,
22 including tagged and untagged queuing, multiple MSI-X interrupts, LSI
23 fastpath and SATA 3G/6G/12G.
24 .SS "Configuration"
25 .sp
26 .LP
27 The \fBmpt_sas3\fR driver is configured by defining properties in
28 \fBmpt_sas3.conf\fR. These properties override the global SCSI settings. The
29 \fBmpt_sas3\fR driver supports one modifiable property:
30 .sp
31 .ne 2
32 .na
33 \fB\fBmpt_sas3\fR\fR
34 .ad
35 .sp .6
36 .RS 4n
37 Solaris I/O multipathing is enabled or disabled on SAS devices with the
38 \fBmpt_sas3\fR property. Specifying \fBmpt_sas3\fR activates I/O
39 multipathing, while \fBmpt_sas3\fR disables I/O multipathing.
40 .sp
41 Solaris I/O multipathing can be enabled or disabled on a per port basis. Per
42 port settings override the global setting for the specified ports.
43 .sp
44 The following example shows how to disable multipathing on port 0 whose parent
45 is \fBpci@0,0/pci8086,2940@1c/pci1000,90@0\fR:
46 .sp
47 .in +2
48 .nf
49 name="mpt_sas3" parent="/pci@0,0/pci8086,2940@1c/pci1000,90@0"
50 mpt_sas3-disable="yes";
51 .fi
52 .in -2

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

59 .RE

61 .SH EXAMPLES
```

new/usr/src/man/man7d/mpt_sas3.7d

2

```
62 .LP
63 \fBExample 1 \fRUsing the \fBmpt_sas3\fR Configuration File to Disable MPXIO
64 .sp
65 .LP
66 Create a file called \fBkernel/drv/mpt_sas3.conf\fR and add the following line:

68 .sp
69 .in +2
70 .nf
71 name="mpt_sas3" parent="/pci@0,0/pci8086,2940@1c/pci1000,90@0"
72 mpt_sas3-disable="yes";
73 .fi
74 .in -2

76 .SH FILES
77 .sp
78 .ne 2
79 .na
80 \fBkernel/drv/mpt_sas3\fR
81 .ad
82 .sp .6
83 .RS 4n
84 32-bit ELF kernel module
85 .RE

87 .sp
88 .ne 2
89 .na
90 \fBkernel/drv/sparcv9/mpt_sas3\fR
91 .ad
92 .sp .6
93 .RS 4n
94 64-bit SPARC ELF kernel module
95 .RE

97 .sp
98 .ne 2
99 .na
100 \fBkernel/drv/amd64/mpt_sas3\fR
101 .ad
102 .sp .6
103 .RS 4n
104 64-bit x86 ELF kernel module
105 .RE

107 .sp
108 .ne 2
109 .na
110 \fBmpt_sas3.conf\fR
111 .ad
112 .sp .6
113 .RS 4n
114 Optional configuration file
115 .RE

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

123 .sp
124 .TS
125 box;
126 1 | 1
127 1 | 1 .
```

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

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

```

*****
21467 Thu Jun 12 17:28:21 2014
new/usr/src/pkg/manifests/developer-debug-mdb.mf
4546 mpt_sas needs enhancing to support LSI MPI2.5
*****

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

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

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

```

```

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

```

```

128 file path=kernel/kmdb/$(ARCH64)/sppp group=sys mode=0555
129 $(sparc_ONLY)file path=kernel/kmdb/$(ARCH64)/ssd group=sys mode=0555
130 file path=kernel/kmdb/$(ARCH64)/ufs group=sys mode=0555
131 $(i386_ONLY)file path=kernel/kmdb/$(ARCH64)/uhci group=sys mode=0555
132 file path=kernel/kmdb/$(ARCH64)/usba group=sys mode=0555
133 $(i386_ONLY)file path=kernel/kmdb/arp group=sys mode=0555
134 $(i386_ONLY)file path=kernel/kmdb/cpc group=sys mode=0555
135 $(i386_ONLY)file path=kernel/kmdb/cpu.generic group=sys mode=0555
136 $(i386_ONLY)file path=kernel/kmdb/cpu_ms.AuthenticAMD.15 group=sys mode=0555
137 $(i386_ONLY)file path=kernel/kmdb/crypto group=sys mode=0555
138 $(i386_ONLY)file path=kernel/kmdb/genunix group=sys mode=0555
139 $(i386_ONLY)file path=kernel/kmdb/hook group=sys mode=0555
140 $(i386_ONLY)file path=kernel/kmdb/ip group=sys mode=0555
141 $(i386_ONLY)file path=kernel/kmdb/ipc group=sys mode=0555
142 $(i386_ONLY)file path=kernel/kmdb/ipp group=sys mode=0555
143 $(i386_ONLY)file path=kernel/kmdb/krtld group=sys mode=0555
144 $(i386_ONLY)file path=kernel/kmdb/lofs group=sys mode=0555
145 $(i386_ONLY)file path=kernel/kmdb/logindmux group=sys mode=0555
146 $(i386_ONLY)file path=kernel/kmdb/mac group=sys mode=0555
147 $(i386_ONLY)file path=kernel/kmdb/md group=sys mode=0555
148 $(i386_ONLY)file path=kernel/kmdb/mdb_ds group=sys mode=0555
149 $(i386_ONLY)file path=kernel/kmdb/mpt group=sys mode=0555
150 $(i386_ONLY)file path=kernel/kmdb/mpt_sas group=sys mode=0555
151 $(i386_ONLY)file path=kernel/kmdb/mpt_sas3 group=sys mode=0555
152 #endif /* ! codereview */
153 $(i386_ONLY)file path=kernel/kmdb/mr_sas group=sys mode=0555
154 $(i386_ONLY)file path=kernel/kmdb/nca group=sys mode=0555
155 $(i386_ONLY)file path=kernel/kmdb/neti group=sys mode=0555
156 $(i386_ONLY)file path=kernel/kmdb/nfs group=sys mode=0555
157 $(i386_ONLY)file path=kernel/kmdb/ptm group=sys mode=0555
158 $(i386_ONLY)file path=kernel/kmdb/random group=sys mode=0555
159 $(i386_ONLY)file path=kernel/kmdb/s1394 group=sys mode=0555
160 $(i386_ONLY)file path=kernel/kmdb/sata group=sys mode=0555
161 $(i386_ONLY)file path=kernel/kmdb/scsi_vhci group=sys mode=0555
162 $(i386_ONLY)file path=kernel/kmdb/sctp group=sys mode=0555
163 $(i386_ONLY)file path=kernel/kmdb/sd group=sys mode=0555
164 $(i386_ONLY)file path=kernel/kmdb/sockfs group=sys mode=0555
165 $(i386_ONLY)file path=kernel/kmdb/specfs group=sys mode=0555
166 $(i386_ONLY)file path=kernel/kmdb/sppp group=sys mode=0555
167 $(i386_ONLY)file path=kernel/kmdb/ufs group=sys mode=0555
168 $(i386_ONLY)file path=kernel/kmdb/uhci group=sys mode=0555
169 $(i386_ONLY)file path=kernel/kmdb/usba group=sys mode=0555
170 $(i386_ONLY)file path=platform/i86pc/kernel/kmdb/$(ARCH64)/apix group=sys \
171 mode=0555
172 $(i386_ONLY)file path=platform/i86pc/kernel/kmdb/$(ARCH64)/pcplusmp group=sys \
173 mode=0555
174 $(i386_ONLY)file path=platform/i86pc/kernel/kmdb/$(ARCH64)/unix group=sys \
175 mode=0555
176 $(i386_ONLY)file path=platform/i86pc/kernel/kmdb/$(ARCH64)/uppc group=sys \
177 mode=0555
178 $(i386_ONLY)file path=platform/i86pc/kernel/kmdb/apix group=sys mode=0555
179 $(i386_ONLY)file path=platform/i86pc/kernel/kmdb/pcplusmp group=sys mode=0555
180 $(i386_ONLY)file path=platform/i86pc/kernel/kmdb/unix group=sys mode=0555
181 $(i386_ONLY)file path=platform/i86pc/kernel/kmdb/uppc group=sys mode=0555
182 $(i386_ONLY)file path=platform/i86xpv/kernel/kmdb/$(ARCH64)/unix group=sys \
183 mode=0555
184 $(i386_ONLY)file path=platform/i86xpv/kernel/kmdb/$(ARCH64)/xpv_psm group=sys \
185 mode=0555
186 $(i386_ONLY)file path=platform/i86xpv/kernel/kmdb/$(ARCH64)/xpv_uppc group=sys \
187 mode=0555
188 $(i386_ONLY)file path=platform/i86xpv/kernel/kmdb/unix group=sys mode=0555
189 $(i386_ONLY)file path=platform/i86xpv/kernel/kmdb/xpv_psm group=sys mode=0555
190 $(i386_ONLY)file path=platform/i86xpv/kernel/kmdb/xpv_uppc group=sys mode=0555
191 $(sparc_ONLY)file path=platform/sun4u/kernel/kmdb/$(ARCH64)/oplhwd group=sys \
192 mode=0555
193 $(sparc_ONLY)file path=platform/sun4u/kernel/kmdb/$(ARCH64)/sgenv group=sys \

```

```

194 mode=0555
195 $(sparc_ONLY)file path=platform/sun4u/kernel/kmdb/$(ARCH64)/sgsbcc group=sys \
196 mode=0555
197 $(sparc_ONLY)file path=platform/sun4u/kernel/kmdb/$(ARCH64)/unix group=sys \
198 mode=0555
199 $(sparc_ONLY)file path=platform/sun4v/kernel/kmdb/$(ARCH64)/errh group=sys \
200 mode=0555
201 $(sparc_ONLY)file path=platform/sun4v/kernel/kmdb/$(ARCH64)/ldc group=sys \
202 mode=0555
203 $(sparc_ONLY)file path=platform/sun4v/kernel/kmdb/$(ARCH64)/mdesc group=sys \
204 mode=0555
205 $(sparc_ONLY)file path=platform/sun4v/kernel/kmdb/$(ARCH64)/unix group=sys \
206 mode=0555
207 $(sparc_ONLY)file path=platform/sun4v/kernel/kmdb/$(ARCH64)/vdsk group=sys \
208 mode=0555
209 file path=usr/bin/$(ARCH32)/mdb mode=0555
210 file path=usr/bin/$(ARCH64)/mdb mode=0555
211 file path=usr/include/sys/mdb_modapi.h
212 file path=usr/lib/mdb/kvm/$(ARCH64)/arp.so group=sys mode=0555
213 file path=usr/lib/mdb/kvm/$(ARCH64)/cpc.so group=sys mode=0555
214 $(i386_ONLY)file path=usr/lib/mdb/kvm/$(ARCH64)/cpu.generic.so group=sys \
215 mode=0555
216 $(i386_ONLY)file path=usr/lib/mdb/kvm/$(ARCH64)/cpu_ms.AuthenticAMD.15.so \
217 group=sys mode=0555
218 file path=usr/lib/mdb/kvm/$(ARCH64)/crypto.so group=sys mode=0555
219 file path=usr/lib/mdb/kvm/$(ARCH64)/genunix.so group=sys mode=0555
220 file path=usr/lib/mdb/kvm/$(ARCH64)/hook.so group=sys mode=0555
221 $(sparc_ONLY)file path=usr/lib/mdb/kvm/$(ARCH64)/intr.so group=sys mode=0555
222 file path=usr/lib/mdb/kvm/$(ARCH64)/ip.so group=sys mode=0555
223 file path=usr/lib/mdb/kvm/$(ARCH64)/ipc.so group=sys mode=0555
224 file path=usr/lib/mdb/kvm/$(ARCH64)/ipp.so group=sys mode=0555
225 file path=usr/lib/mdb/kvm/$(ARCH64)/krtld.so group=sys mode=0555
226 file path=usr/lib/mdb/kvm/$(ARCH64)/lofs.so group=sys mode=0555
227 file path=usr/lib/mdb/kvm/$(ARCH64)/logindmux.so group=sys mode=0555
228 file path=usr/lib/mdb/kvm/$(ARCH64)/mac.so group=sys mode=0555
229 file path=usr/lib/mdb/kvm/$(ARCH64)/md.so group=sys mode=0555
230 $(i386_ONLY)file path=usr/lib/mdb/kvm/$(ARCH64)/mdb_kb.so group=sys mode=0555
231 file path=usr/lib/mdb/kvm/$(ARCH64)/mdb_ks.so group=sys mode=0555
232 file path=usr/lib/mdb/kvm/$(ARCH64)/mpt.so group=sys mode=0555
233 file path=usr/lib/mdb/kvm/$(ARCH64)/mpt_sas.so group=sys mode=0555
234 file path=usr/lib/mdb/kvm/$(ARCH64)/mpt_sas3.so group=sys mode=0555
235 #endif /* ! codereview */
236 file path=usr/lib/mdb/kvm/$(ARCH64)/mr_sas.so group=sys mode=0555
237 file path=usr/lib/mdb/kvm/$(ARCH64)/nca.so group=sys mode=0555
238 file path=usr/lib/mdb/kvm/$(ARCH64)/neti.so group=sys mode=0555
239 file path=usr/lib/mdb/kvm/$(ARCH64)/nfs.so group=sys mode=0555
240 file path=usr/lib/mdb/kvm/$(ARCH64)/ptm.so group=sys mode=0555
241 file path=usr/lib/mdb/kvm/$(ARCH64)/random.so group=sys mode=0555
242 file path=usr/lib/mdb/kvm/$(ARCH64)/s1394.so group=sys mode=0555
243 $(i386_ONLY)file path=usr/lib/mdb/kvm/$(ARCH64)/sata.so group=sys mode=0555
244 file path=usr/lib/mdb/kvm/$(ARCH64)/scsi_vhci.so group=sys mode=0555
245 file path=usr/lib/mdb/kvm/$(ARCH64)/sctp.so group=sys mode=0555
246 file path=usr/lib/mdb/kvm/$(ARCH64)/sd.so group=sys mode=0555
247 file path=usr/lib/mdb/kvm/$(ARCH64)/sockfs.so group=sys mode=0555
248 file path=usr/lib/mdb/kvm/$(ARCH64)/specfs.so group=sys mode=0555
249 file path=usr/lib/mdb/kvm/$(ARCH64)/sppp.so group=sys mode=0555
250 $(sparc_ONLY)file path=usr/lib/mdb/kvm/$(ARCH64)/ssd.so group=sys mode=0555
251 file path=usr/lib/mdb/kvm/$(ARCH64)/ufs.so group=sys mode=0555
252 $(i386_ONLY)file path=usr/lib/mdb/kvm/$(ARCH64)/uhci.so group=sys mode=0555
253 file path=usr/lib/mdb/kvm/$(ARCH64)/usba.so group=sys mode=0555
254 $(i386_ONLY)file path=usr/lib/mdb/kvm/arp.so group=sys mode=0555
255 $(i386_ONLY)file path=usr/lib/mdb/kvm/cpc.so group=sys mode=0555
256 $(i386_ONLY)file path=usr/lib/mdb/kvm/cpu.generic.so group=sys mode=0555
257 $(i386_ONLY)file path=usr/lib/mdb/kvm/cpu_ms.AuthenticAMD.15.so group=sys \
258 mode=0555
259 $(i386_ONLY)file path=usr/lib/mdb/kvm/crypto.so group=sys mode=0555

```



```

260 $(i386_ONLY)file path=usr/lib/mdb/kvm/genunix.so group=sys mode=0555
261 $(i386_ONLY)file path=usr/lib/mdb/kvm/hook.so group=sys mode=0555
262 $(i386_ONLY)file path=usr/lib/mdb/kvm/ip.so group=sys mode=0555
263 $(i386_ONLY)file path=usr/lib/mdb/kvm/ipc.so group=sys mode=0555
264 $(i386_ONLY)file path=usr/lib/mdb/kvm/ipp.so group=sys mode=0555
265 $(i386_ONLY)file path=usr/lib/mdb/kvm/krtld.so group=sys mode=0555
266 $(i386_ONLY)file path=usr/lib/mdb/kvm/lofs.so group=sys mode=0555
267 $(i386_ONLY)file path=usr/lib/mdb/kvm/logindmux.so group=sys mode=0555
268 $(i386_ONLY)file path=usr/lib/mdb/kvm/mac.so group=sys mode=0555
269 $(i386_ONLY)file path=usr/lib/mdb/kvm/md.so group=sys mode=0555
270 $(i386_ONLY)file path=usr/lib/mdb/kvm/mdb_kb.so group=sys mode=0555
271 $(i386_ONLY)file path=usr/lib/mdb/kvm/mdb_ks.so group=sys mode=0555
272 $(i386_ONLY)file path=usr/lib/mdb/kvm/mpt.so group=sys mode=0555
273 $(i386_ONLY)file path=usr/lib/mdb/kvm/mpt_sas.so group=sys mode=0555
274 $(i386_ONLY)file path=usr/lib/mdb/kvm/mpt_sas3.so group=sys mode=0555
275 #endif /* ! codereview */
276 $(i386_ONLY)file path=usr/lib/mdb/kvm/mr_sas.so group=sys mode=0555
277 $(i386_ONLY)file path=usr/lib/mdb/kvm/nca.so group=sys mode=0555
278 $(i386_ONLY)file path=usr/lib/mdb/kvm/neti.so group=sys mode=0555
279 $(i386_ONLY)file path=usr/lib/mdb/kvm/nfs.so group=sys mode=0555
280 $(i386_ONLY)file path=usr/lib/mdb/kvm/ptm.so group=sys mode=0555
281 $(i386_ONLY)file path=usr/lib/mdb/kvm/random.so group=sys mode=0555
282 $(i386_ONLY)file path=usr/lib/mdb/kvm/sl394.so group=sys mode=0555
283 $(i386_ONLY)file path=usr/lib/mdb/kvm/sata.so group=sys mode=0555
284 $(i386_ONLY)file path=usr/lib/mdb/kvm/scsi_vhci.so group=sys mode=0555
285 $(i386_ONLY)file path=usr/lib/mdb/kvm/sctp.so group=sys mode=0555
286 $(i386_ONLY)file path=usr/lib/mdb/kvm/sd.so group=sys mode=0555
287 $(i386_ONLY)file path=usr/lib/mdb/kvm/sockfs.so group=sys mode=0555
288 $(i386_ONLY)file path=usr/lib/mdb/kvm/specfs.so group=sys mode=0555
289 $(i386_ONLY)file path=usr/lib/mdb/kvm/sppp.so group=sys mode=0555
290 $(i386_ONLY)file path=usr/lib/mdb/kvm/ufs.so group=sys mode=0555
291 $(i386_ONLY)file path=usr/lib/mdb/kvm/uhci.so group=sys mode=0555
292 $(i386_ONLY)file path=usr/lib/mdb/kvm/usba.so group=sys mode=0555
293 file path=usr/lib/mdb/proc/$(ARCH64)/ld.so group=sys mode=0555
294 file path=usr/lib/mdb/proc/$(ARCH64)/libavl.so group=sys mode=0555
295 file path=usr/lib/mdb/proc/$(ARCH64)/libc.so group=sys mode=0555
296 file path=usr/lib/mdb/proc/$(ARCH64)/libcmdutils.so group=sys mode=0555
297 file path=usr/lib/mdb/proc/$(ARCH64)/libnvpair.so group=sys mode=0555
298 file path=usr/lib/mdb/proc/$(ARCH64)/libproc.so group=sys mode=0555
299 file path=usr/lib/mdb/proc/$(ARCH64)/libpython2.6.so group=sys mode=0555
300 file path=usr/lib/mdb/proc/$(ARCH64)/libsysevent.so group=sys mode=0555
301 file path=usr/lib/mdb/proc/$(ARCH64)/libtopo.so group=sys mode=0555
302 file path=usr/lib/mdb/proc/$(ARCH64)/libumem.so group=sys mode=0555
303 file path=usr/lib/mdb/proc/$(ARCH64)/libutil.so group=sys mode=0555
304 file path=usr/lib/mdb/proc/$(ARCH64)/mdb_ds.so group=sys mode=0555
305 $(i386_ONLY)file path=usr/lib/mdb/proc/$(ARCH64)/mdb_test.so group=sys \
306 mode=0555
307 file path=usr/lib/mdb/proc/ld.so group=sys mode=0555
308 file path=usr/lib/mdb/proc/libavl.so group=sys mode=0555
309 file path=usr/lib/mdb/proc/libc.so group=sys mode=0555
310 file path=usr/lib/mdb/proc/libcmdutils.so group=sys mode=0555
311 file path=usr/lib/mdb/proc/libnvpair.so group=sys mode=0555
312 file path=usr/lib/mdb/proc/libproc.so group=sys mode=0555
313 file path=usr/lib/mdb/proc/libpython2.6.so group=sys mode=0555
314 file path=usr/lib/mdb/proc/libsysevent.so group=sys mode=0555
315 file path=usr/lib/mdb/proc/libtopo.so group=sys mode=0555
316 file path=usr/lib/mdb/proc/libumem.so group=sys mode=0555
317 file path=usr/lib/mdb/proc/libutil.so group=sys mode=0555
318 file path=usr/lib/mdb/proc/mdb_ds.so group=sys mode=0555
319 file path=usr/lib/mdb/proc/svc.configd.so group=sys mode=0555
320 file path=usr/lib/mdb/proc/svc.startd.so group=sys mode=0555
321 $(i386_ONLY)file path=usr/platform/i86pc/lib/mdb/kvm/$(ARCH64)/apix.so \
322 group=sys mode=0555
323 $(i386_ONLY)file path=usr/platform/i86pc/lib/mdb/kvm/$(ARCH64)/pcplusmp.so \
324 group=sys mode=0555
325 $(i386_ONLY)file path=usr/platform/i86pc/lib/mdb/kvm/$(ARCH64)/unix.so \

```

```

326 group=sys mode=0555
327 $(i386_ONLY)file path=usr/platform/i86pc/lib/mdb/kvm/$(ARCH64)/uppc.so \
328 group=sys mode=0555
329 $(i386_ONLY)file path=usr/platform/i86pc/lib/mdb/kvm/apix.so group=sys \
330 mode=0555
331 $(i386_ONLY)file path=usr/platform/i86pc/lib/mdb/kvm/pcplusmp.so group=sys \
332 mode=0555
333 $(i386_ONLY)file path=usr/platform/i86pc/lib/mdb/kvm/unix.so group=sys \
334 mode=0555
335 $(i386_ONLY)file path=usr/platform/i86pc/lib/mdb/kvm/uppc.so group=sys \
336 mode=0555
337 $(i386_ONLY)file path=usr/platform/i86pc/lib/mdb/kvm/$(ARCH64)/unix.so \
338 group=sys mode=0555
339 $(i386_ONLY)file path=usr/platform/i86pc/lib/mdb/kvm/$(ARCH64)/xpvm.so \
340 group=sys mode=0555
341 $(i386_ONLY)file path=usr/platform/i86pc/lib/mdb/kvm/$(ARCH64)/xpvm_psm.so \
342 group=sys mode=0555
343 $(i386_ONLY)file path=usr/platform/i86pc/lib/mdb/kvm/$(ARCH64)/xpvm_uppc.so \
344 group=sys mode=0555
345 $(i386_ONLY)file path=usr/platform/i86pc/lib/mdb/kvm/unix.so group=sys \
346 mode=0555
347 $(i386_ONLY)file path=usr/platform/i86pc/lib/mdb/kvm/xpvm.so group=sys \
348 mode=0555
349 $(i386_ONLY)file path=usr/platform/i86pc/lib/mdb/kvm/xpvm_psm.so group=sys \
350 mode=0555
351 $(i386_ONLY)file path=usr/platform/i86pc/lib/mdb/kvm/xpvm_uppc.so group=sys \
352 mode=0555
353 $(sparc_ONLY)file path=usr/platform/sun4u/lib/mdb/kvm/$(ARCH64)/oplhwd.so \
354 group=sys mode=0555
355 $(sparc_ONLY)file path=usr/platform/sun4u/lib/mdb/kvm/$(ARCH64)/sgenv.so \
356 group=sys mode=0555
357 $(sparc_ONLY)file path=usr/platform/sun4u/lib/mdb/kvm/$(ARCH64)/sgsbcc.so \
358 group=sys mode=0555
359 $(sparc_ONLY)file path=usr/platform/sun4u/lib/mdb/kvm/$(ARCH64)/unix.so \
360 group=sys mode=0555
361 $(sparc_ONLY)file path=usr/platform/sun4v/lib/mdb/kvm/$(ARCH64)/errh.so \
362 group=sys mode=0555
363 $(sparc_ONLY)file path=usr/platform/sun4v/lib/mdb/kvm/$(ARCH64)/ldc.so \
364 group=sys mode=0555
365 $(sparc_ONLY)file path=usr/platform/sun4v/lib/mdb/kvm/$(ARCH64)/mdesc.so \
366 group=sys mode=0555
367 $(sparc_ONLY)file path=usr/platform/sun4v/lib/mdb/kvm/$(ARCH64)/unix.so \
368 group=sys mode=0555
369 $(sparc_ONLY)file path=usr/platform/sun4v/lib/mdb/kvm/$(ARCH64)/vdsd.so \
370 group=sys mode=0555
371 file path=usr/share/man/man1/adb.1
372 file path=usr/share/man/man1/kmdb.1
373 file path=usr/share/man/man1/mdb.1
374 hardlink path=usr/bin/$(ARCH32)/adb target=../../../../usr/bin/$(ARCH32)/mdb
375 hardlink path=usr/bin/$(ARCH64)/adb target=../../../../usr/bin/$(ARCH64)/mdb
376 hardlink path=usr/bin/adb target=../../../../usr/lib/isaexec
377 hardlink path=usr/bin/mdb target=../../../../usr/lib/isaexec
378 legacy pkg=SUNWmdb desc="Modular Debugger (MDB)" name="Modular Debugger"
379 legacy pkg=SUNWmdbr desc="Modular Debugger (MDB) (Root)" \
380 name="Modular Debugger (Root)"
381 license cr_Sun license=cr_Sun
382 license lic_CDDL license=lic_CDDL
383 license usr/src/common/bzip2/LICENSE license=usr/src/common/bzip2/LICENSE
384 license usr/src/uts/common/io/mr_sas/THIRDPARTYLICENSE \
385 license=usr/src/uts/common/io/mr_sas/THIRDPARTYLICENSE
386 license usr/src/uts/common/zmod/THIRDPARTYLICENSE \
387 license=usr/src/uts/common/zmod/THIRDPARTYLICENSE
388 $(sparc_ONLY)link path=kernel/kmdb/$(ARCH64)/niuxm target=intr
389 $(sparc_ONLY)link path=kernel/kmdb/$(ARCH64)/pcipsy target=intr
390 $(sparc_ONLY)link path=kernel/kmdb/$(ARCH64)/pcisch target=intr
391 $(sparc_ONLY)link path=kernel/kmdb/$(ARCH64)/px target=intr

```

new/usr/src/pkg/manifests/developer-debug-mdb.mf

7

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

new/usr/src/pkg/manifests/driver-storage-mpt_sas3.mf

1

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

```
1 #
2 # CDDL HEADER START
3 #
4 # The contents of this file are subject to the terms of the
5 # Common Development and Distribution License (the "License").
6 # You may not use this file except in compliance with the License.
7 #
8 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9 # or http://www.opensolaris.org/os/licensing.
10 # See the License for the specific language governing permissions
11 # and limitations under the License.
12 #
13 # When distributing Covered Code, include this CDDL HEADER in each
14 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 # If applicable, add the following below this CDDL HEADER, with the
16 # fields enclosed by brackets "[]" replaced with your own identifying
17 # information: Portions Copyright [yyyy] [name of copyright owner]
18 #
19 # CDDL HEADER END
20 #
21 #
22 #
23 # Copyright (c) 2010, Oracle and/or its affiliates. All rights reserved.
24 #
25 #
26 #
27 # The default for payload-bearing actions in this package is to appear in the
28 # global zone only. See the include file for greater detail, as well as
29 # information about overriding the defaults.
30 #
31 <include global_zone_only_component>
32 set name=pkg.fmri value=pkg:/driver/storage/mpt_sas3@$(PKGVERS)
33 set name=pkg.description value="LSI MPT SAS 3.0 Controller HBA Driver"
34 set name=pkg.summary value="LSI MPT SAS 3.0 Controller HBA Driver"
35 set name=info.classification \
36     value=org.opensolaris.category.2008:Drivers/Storage
37 set name=variant.arch value=$(ARCH)
38 dir path=kernel group=sys
39 dir path=kernel/drv group=sys
40 dir path=kernel/drv/$(ARCH64) group=sys
41 dir path=usr/share/man
42 dir path=usr/share/man/man7d
43 driver name=mpt_sas3 class=scsi-self-identifying \
44     alias=pciex1000,90 \
45     alias=pciex1000,97
46 file path=kernel/drv/$(ARCH64)/mpt_sas3 group=sys
47 $(i386_ONLY)file path=kernel/drv/mpt_sas3 group=sys
48 file path=kernel/drv/mpt_sas3.conf group=sys preserve=true
49 file path=usr/share/man/man7d/mpt_sas3.7d
50 license lic_CDDL license=lic_CDDL
51 #endif /* ! codereview */
```

new/usr/src/uts/common/Makefile.files

1

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

```
1 #
2 # CDDL HEADER START
3 #
4 # The contents of this file are subject to the terms of the
5 # Common Development and Distribution License (the "License").
6 # You may not use this file except in compliance with the License.
7 #
8 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9 # or http://www.opensolaris.org/os/licensing.
10 # See the License for the specific language governing permissions
11 # and limitations under the License.
12 #
13 # When distributing Covered Code, include this CDDL HEADER in each
14 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 # If applicable, add the following below this CDDL HEADER, with the
16 # fields enclosed by brackets "[]" replaced with your own identifying
17 # information: Portions Copyright [yyyy] [name of copyright owner]
18 #
19 # CDDL HEADER END
20 #
21 #
22 #
23 # Copyright (c) 1991, 2010, Oracle and/or its affiliates. All rights reserved.
24 # Copyright (c) 2012 Nexenta Systems, Inc. All rights reserved.
25 # Copyright (c) 2013 by Delphix. All rights reserved.
26 # Copyright (c) 2013 by Saso Kiselkov. All rights reserved.
27 #
28 #
29 #
30 # This Makefile defines all file modules for the directory uts/common
31 # and its children. These are the source files which may be considered
32 # common to all SunOS systems.
33 #
34 i386_CORE_OBJS += \
35     atomic.o      \
36     avintr.o      \
37     pic.o
38 #
39 sparc_CORE_OBJS +=
40 #
41 COMMON_CORE_OBJS += \
42     beep.o        \
43     bitset.o      \
44     bp_map.o      \
45     brand.o       \
46     cpucaps.o     \
47     cmt.o         \
48     cmt_policy.o  \
49     cpu.o         \
50     cpu_event.o   \
51     cpu_intr.o    \
52     cpu_pm.o      \
53     cpupart.o     \
54     cap_util.o    \
55     disp.o        \
56     group.o       \
57     kstat_fr.o    \
58     iscsiboot_prop.o \
59     lgrp.o        \
60     lgrp_topo.o   \
61     mmapobj.o     \
```

new/usr/src/uts/common/Makefile.files

2

```
62     mutex.o       \
63     page_lock.o   \
64     page_retire.o \
65     panic.o       \
66     param.o       \
67     pg.o          \
68     pghw.o        \
69     putnext.o     \
70     rctl_proc.o   \
71     rwlock.o      \
72     seg_kmem.o    \
73     softint.o     \
74     string.o      \
75     strtol.o      \
76     strtoul.o     \
77     strtoll.o     \
78     strtoull.o    \
79     thread_intr.o \
80     vm_page.o     \
81     vm_pagelist.o \
82     zlib_obj.o    \
83     clock_tick.o
84 #
85 CORE_OBJS += $(COMMON_CORE_OBJS) $( $(MACH)_CORE_OBJS )
86 #
87 ZLIB_OBJS = zutil.o zmod.o zmod_subr.o \
88     adler32.o crc32.o deflate.o inffast.o \
89     inflate.o inftrees.o trees.o
90 #
91 GENUNIX_OBJS += \
92     access.o      \
93     acl.o         \
94     acl_common.o  \
95     adjtime.o     \
96     alarm.o       \
97     aio_subr.o    \
98     auditsys.o    \
99     audit_core.o  \
100     audit_zone.o  \
101     audit_memory.o \
102     autoconf.o    \
103     avl.o         \
104     bdev_dsort.o  \
105     bio.o         \
106     bitmap.o      \
107     blabel.o      \
108     brandsys.o    \
109     bz2blocksort.o \
110     bz2compress.o \
111     bz2decompress.o \
112     bz2randtable.o \
113     bz2bzip.o     \
114     bz2crctable.o \
115     bz2huffman.o  \
116     callb.o       \
117     callout.o     \
118     chdir.o       \
119     chmod.o       \
120     chown.o       \
121     cladm.o       \
122     class.o       \
123     clock.o       \
124     clock_highres.o \
125     clock_realtime.o \
126     close.o       \
127     compress.o    \
```

new/usr/src/uts/common/Makefile.files

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

3

new/usr/src/uts/common/Makefile.files

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

4

```

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

```

```

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

```

new/usr/src/uts/common/Makefile.files

7

```

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

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

412 i386_GENSTUBS_OBJS =

414 COMMON_GENSTUBS_OBJS =

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

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

423 SDT_OBJS += sdt_subr.o

425 PROFILE_OBJS += profile.o

427 SYSTRACE_OBJS += systrace.o

429 LOCKSTAT_OBJS += lockstat.o

431 FASTTRAP_OBJS += fasttrap.o fasttrap_isa.o

433 DCPC_OBJS += dcpc.o

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

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

445 AUDIOEMU10K_OBJS += audioemu10k.o

447 AUDIOENS_OBJS += audioens.o

449 AUDIOVIA823X_OBJS += audiovia823x.o

451 AUDIOVIA97_OBJS += audiovia97.o

453 AUDIO1575_OBJS += audio1575.o

455 AUDIO810_OBJS += audio810.o

457 AUDIOCMI_OBJS += audiocmi.o

```

new/usr/src/uts/common/Makefile.files

8

```

459 AUDIOCMIHD_OBJS += audiocmihd.o

461 AUDIOHD_OBJS += audiohd.o

463 AUDIOIXP_OBJS += audioixp.o

465 AUDIOLS_OBJS += audiol.s.o

467 AUDIOP16X_OBJS += audiop16x.o

469 AUDIOPCI_OBJS += audiopci.o

471 AUDIOSOLO_OBJS += audiosolo.o

473 AUDIOTS_OBJS += audiot.s.o

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

477 BLKDEV_OBJS += blkdev.o

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

481 CONSKBD_OBJS += conskbd.o

483 CONSMS_OBJS += consms.o

485 OLDPTY_OBJS += tty_ptyconf.o

487 PTC_OBJS += tty_pty.o

489 PTSL_OBJS += tty_pts.o

491 PTM_OBJS += ptm.o

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

496 PTS_OBJS += pts.o

498 PTY_OBJS += ptms_conf.o

500 SAD_OBJS += sad.o

502 MD4_OBJS += md4.o md4_mod.o

504 MD5_OBJS += md5.o md5_mod.o

506 SHA1_OBJS += sha1.o sha1_mod.o

508 SHA2_OBJS += sha2.o sha2_mod.o

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

513 DSCPMK_OBJS += dscpmk.o dscpmkddi.o

515 DLCOSMK_OBJS += dlcosmk.o dlcosmkddi.o

517 FLOWACCT_OBJS += flowacctddi.o flowacct.o

519 TOKENMT_OBJS += tokenmt.o tokenmtddi.o

521 TSWTCL_OBJS += tswtcl.o tswtclddi.o

523 ARP_OBJS += arpddi.o

```

```

525 ICMP_OBJS +=      icmpddi.o
527 ICMP6_OBJS +=     icmp6ddi.o
529 RTS_OBJS +=       rtsddi.o

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

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

563 IP6_OBJS +=          ip6ddi.o
565 HOOK_OBJS +=         hook.o
567 NETI_OBJS +=         neti_impl.o neti_mod.o neti_stack.o
569 KEYSOCK_OBJS +=      keysockddi.o keysock.o keysock_opt_data.o
571 IPNET_OBJS +=        ipnet.o ipnet_bpf.o
573 SPDSOCK_OBJS +=      spdsockddi.o spdsock.o spdsock_opt_data.o
575 IPSECESP_OBJS +=     ipsecespddi.o ipsecesp.o
577 IPSECAH_OBJS +=      ipsecahddi.o ipsecah.o sadb.o
579 SPPP_OBJS +=         sPPP.o sPPP_dlpi.o sPPP_mod.o s_common.o
581 SPPPTUN_OBJS +=      sppptun.o sppptun_mod.o
583 SPPPASYN_OBJS +=     spppasyn.o spppasyn_mod.o
585 SPPPCOMP_OBJS +=     sppppcomp.o sppppcomp_mod.o deflate.o bsd-comp.o vjcompress.o \
586                     zlib.o
588 TCP_OBJS +=          tcpddi.o

```

```

590 TCP6_OBJS +=         tcp6ddi.o
592 NCA_OBJS +=          ncaddi.o
594 SDP SOCK_MOD_OBJS += sockmod_sdp.o socksdp.o socksdpsubr.o
596 SCTP SOCK_MOD_OBJS += sockmod_sctp.o sockscctp.o socksctpsubr.o
598 PFP SOCK_MOD_OBJS += sockmod_pfp.o
600 RDS SOCK_MOD_OBJS += sockmod_rds.o
602 RDS_OBJS +=          rdsddi.o rdssubr.o rds_opt.o rds_ioctl.o
604 RDSIB_OBJS +=         rdsib.o rdsib_ib.o rdsib_cm.o rdsib_ep.o rdsib_buf.o \
605                       rdsib_debug.o rdsib_sc.o
607 RDSV3_OBJS +=         af_rds.o rds_v3_ddi.o bind.o loop.o threads.o connection.o \
608                       transport.o cong.o sysctl.o message.o rds_recv.o send.o \
609                       stats.o info.o page.o rdma_transport.o ib_ring.o ib_rdma.o \
610                       ib_recv.o ib.o ib_send.o ib_sysctl.o ib_stats.o ib_cm.o \
611                       rds_v3_sc.o rds_v3_debug.o rds_v3_impl.o rdma.o rds_v3_af_thr.o
613 ISER_OBJS +=          iser.o iser_cm.o iser_cq.o iser_ib.o iser_idm.o \
614                       iser_resource.o iser_xfer.o
616 UDP_OBJS +=          udpddi.o
618 UDP6_OBJS +=         udp6ddi.o
620 SY_OBJS +=           gentty.o
622 TCO_OBJS +=          ticots.o
624 TCOO_OBJS +=         ticotsord.o
626 TCL_OBJS +=          ticlts.o
628 TL_OBJS +=           tl.o
630 DUMP_OBJS +=         dump.o
632 BPF_OBJS +=          bpf.o bpf_filter.o bpf_mod.o bpf_dlt.o bpf_mac.o
634 CLONE_OBJS +=        clone.o
636 CN_OBJS +=           cons.o
638 DLD_OBJS +=          dld_drv.o dld_proto.o dld_str.o dld_flow.o
640 DLS_OBJS +=          dls.o dls_link.o dls_mod.o dls_stat.o dls_mgmt.o
642 GLD_OBJS +=          gld.o gldutil.o
644 MAC_OBJS +=          mac.o mac_bcast.o mac_client.o mac_datapath_setup.o mac_flow.o
645                       mac_hio.o mac_mod.o mac_ndd.o mac_provider.o mac_sched.o \
646                       mac_protect.o mac_soft_ring.o mac_stat.o mac_util.o
648 MAC_6TO4_OBJS +=      mac_6to4.o
650 MAC_ETHER_OBJS +=     mac_ether.o
652 MAC_IPV4_OBJS +=      mac_ipv4.o
654 MAC_IPV6_OBJS +=      mac_ipv6.o

```


new/usr/src/uts/common/Makefile.files

11

```
656 MAC_WIFI_OBJS +=      mac_wifi.o
658 MAC_IB_OBJS +=        mac_ib.o
660 IPTUN_OBJS +=         iptun_dev.o iptun_ctl.o iptun.o
662 AGGR_OBJS +=          aggr_dev.o aggr_ctl.o aggr_grp.o aggr_port.o \
663                        aggr_send.o aggr_recv.o aggr_lacp.o
665 SOFTMAC_OBJS +=        softmac_main.o softmac_ctl.o softmac_capab.o \
666                        softmac_dev.o softmac_stat.o softmac_pkt.o softmac_fp.o
668 NET80211_OBJS +=       net80211.o net80211_proto.o net80211_input.o \
669                        net80211_output.o net80211_node.o net80211_crypto.o \
670                        net80211_crypto_none.o net80211_crypto_wep.o net80211_ioctl.o \
671                        net80211_crypto_tkip.o net80211_crypto_ccmp.o \
672                        net80211_ht.o
674 VNIC_OBJS +=          vnic_ctl.o vnic_dev.o
676 SIMNET_OBJS +=        simnet.o
678 IB_OBJS +=            ibnex.o ibnex_ioctl.o ibnex_hca.o
680 IBCM_OBJS +=          ibcm_impl.o ibcm_sm.o ibcm_ti.o ibcm_utils.o ibcm_path.o \
681                        ibcm_arp.o ibcm_arp_link.o
683 IBDM_OBJS +=          ibdm.o
685 IBDMA_OBJS +=         ibdma.o
687 IBMF_OBJS +=          ibmf.o ibmf_impl.o ibmf_dr.o ibmf_wqe.o ibmf_ud_dest.o ibmf_mod.
688                        ibmf_send.o ibmf_recv.o ibmf_handlers.o ibmf_trans.o \
689                        ibmf_timers.o ibmf_msg.o ibmf_utils.o ibmf_rmpp.o \
690                        ibmf_saa.o ibmf_saa_impl.o ibmf_saa_utils.o ibmf_saa_events.o
692 IBTL_OBJS +=          ibtl_impl.o ibtl_util.o ibtl_mem.o ibtl_handlers.o ibtl_qp.o \
693                        ibtl_cq.o ibtl_wr.o ibtl_hca.o ibtl_chan.o ibtl_cm.o \
694                        ibtl_mcg.o ibtl_ibnex.o ibtl_srq.o ibtl_part.o
696 TAVOR_OBJS +=         tavor.o tavor_agents.o tavor_cfg.o tavor_ci.o tavor_cmd.o \
697                        tavor_cq.o tavor_event.o tavor_ioctl.o tavor_misc.o \
698                        tavor_mr.o tavor_qp.o tavor_qpmod.o tavor_rsrc.o \
699                        tavor_srq.o tavor_stats.o tavor_umap.o tavor_wr.o
701 HERMON_OBJS +=        hermon.o hermon_agents.o hermon_cfg.o hermon_ci.o hermon_cmd.o \
702                        hermon_cq.o hermon_event.o hermon_ioctl.o hermon_misc.o \
703                        hermon_mr.o hermon_qp.o hermon_qpmod.o hermon_rsrc.o \
704                        hermon_srq.o hermon_stats.o hermon_umap.o hermon_wr.o \
705                        hermon_fcoib.o hermon_fm.o
707 DAPLT_OBJS +=         daplt.o
709 SOL_OFS_OBJS +=        sol_cma.o sol_ib_cma.o sol_uobj.o \
710                        sol_ofs_debug_util.o sol_ofs_gen_util.o \
711                        sol_kverbs.o
713 SOL_UCMA_OBJS +=       sol_ucma.o
715 SOL_UVERBS_OBJS +=     sol_uverbs.o sol_uverbs_comp.o sol_uverbs_event.o \
716                        sol_uverbs_hca.o sol_uverbs_qp.o
718 SOL_UMAD_OBJS +=       sol_umad.o
720 KSTAT_OBJS +=          kstat.o
```

new/usr/src/uts/common/Makefile.files

12

```
722 KSYMS_OBJS +=         ksyms.o
724 INSTANCE_OBJS +=      inst_sync.o
726 IWSCN_OBJS +=         iwscons.o
728 LOFI_OBJS +=          lofi.o LzmaDec.o
730 FSSNAP_OBJS +=        fssnap.o
732 FSSNAPIF_OBJS +=      fssnap_if.o
734 MM_OBJS +=            mem.o
736 PHYSMEM_OBJS +=       physmem.o
738 OPTIONS_OBJS +=       options.o
740 WINLOCK_OBJS +=       winlockio.o
742 PM_OBJS +=            pm.o
743 SRN_OBJS +=            srn.o
745 PSEUDO_OBJS +=        pseudonex.o
747 RAMDISK_OBJS +=       ramdisk.o
749 LLC1_OBJS +=          llc1.o
751 USBKBM_OBJS +=        usbkbm.o
753 USBWCM_OBJS +=        usbwcm.o
755 BOFI_OBJS +=          bofi.o
757 HID_OBJS +=           hid.o
759 HWA_RC_OBJS +=        hwarc.o
761 USBSKEL_OBJS +=        usbskel.o
763 USBVC_OBJS +=          usbvc.o usbvc_v4l2.o
765 HIDPARSER_OBJS +=      hidparser.o
767 USB_AC_OBJS +=         usb_ac.o
769 USB_AS_OBJS +=         usb_as.o
771 USB_AH_OBJS +=         usb_ah.o
773 USBMS_OBJS +=          usbms.o
775 USBPRN_OBJS +=         usbprn.o
777 UGEN_OBJS +=           ugen.o
779 USBSER_OBJS +=         usbser.o usbser_rseq.o
781 USBSACM_OBJS +=        usb sacram.o
783 USBSER_KEYSPAN_OBJS += usbser_keyspan.o keyspan_dsd.o keyspan_pipe.o
785 USBS49_FW_OBJS +=      keyspan_49fw.o
787 USBSPRL_OBJS +=        usbser_pl2303.o pl2303_dsd.o
```

```

789 WUSB_CA_OBJS += wusb_ca.o
791 USBFTDI_OBJS += usbser_uftdi.o uftdi_dsd.o
793 USBECM_OBJS += usbecm.o
795 WC_OBJS += wscons.o vcons.o
797 VCONS_CONF_OBJS += vcons_conf.o

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

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

806 SCSI_VHCI_F_SYM_OBJS +=      sym.o

808 SCSI_VHCI_F_TPGS_OBJS +=      tpgs.o

810 SCSI_VHCI_F_ASYM_SUN_OBJS +=  asym_sun.o

812 SCSI_VHCI_F_SYM_HDS_OBJS +=  sym_hds.o

814 SCSI_VHCI_F_TAPE_OBJS +=      tape.o

816 SCSI_VHCI_F_TPGS_TAPE_OBJS += tpgs_tape.o

818 SGEN_OBJS +=      sgen.o

820 SMP_OBJS +=      smp.o

822 SATA_OBJS +=      sata.o

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

831 USBA10_OBJS +=      usba10.o

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

835 RSMOPS_OBJS +=      rsmops.o

837 S1394_OBJS +=      t1394.o t1394_errmsg.o s1394.o s1394_addr.o s1394_async.o \
838                  s1394_bus_reset.o s1394_cmp.o s1394_csr.o s1394_dev_disc.o \
839                  s1394_fa.o s1394_fcp.o \
840                  s1394_hotplug.o s1394_isoch.o s1394_misc.o h1394.o nx1394.o

842 HCI1394_OBJS +=      hcil1394.o hcil1394_async.o hcil1394_attach.o hcil1394_buf.o \
843                  hcil1394_csr.o hcil1394_detach.o hcil1394_extern.o \
844                  hcil1394_ioctl.o hcil1394_isoch.o hcil1394_isr.o \
845                  hcil1394_ixl_comp.o hcil1394_ixl_isr.o hcil1394_ixl_misc.o \
846                  hcil1394_ixl_update.o hcil1394_misc.o hcil1394_ohci.o \
847                  hcil1394_q.o hcil1394_s1394if.o hcil1394_tlabel.o \
848                  hcil1394_tlist.o hcil1394_vendor.o

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

```

```

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

858 SCAS1394_OBJS += hba.o sbp2_driver.o sbp2_bus.o

860 SBP2_OBJS +=      cfgrom.o sbp2.o

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

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

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

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

880 RDCSRV_OBJS +=      rdcsrv.o

882 RDCSTUB_OBJS +=      rdc_stub.o

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

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

```

```
920             spcs_s_k.o

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

928 SV_OBJS += sv.o

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

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

936 #
937 #      Build up defines and paths.

939 ST_OBJS += st.o st_conf.o

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

948 EMLXS_FW_OBJS += emlxs_fw.o

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

954 FCT_OBJS += discovery.o fct.o

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

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

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

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

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

966 ISCSIT_SHARED_OBJS += \
967             iscsit_common.o

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

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

976 STMF_OBJS += lun_map.o stmf.o

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

980 SYMSMSG_OBJS += sysmsg.o

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

984 TNF_OBJS += tn timer.o tn timer.o tn timer.o tn timer.o \
985            trace_funcs.o tn timer.o tn timer.o tn timer.o
```

```
987 LOGINDMUX_OBJS += loginmux.o

989 DEVINFO_OBJS += devinfo.o

991 DEVPOLL_OBJS += devpoll.o

993 DEVPOOL_OBJS += devpool.o

995 I8042_OBJS += i8042.o

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

1002 MOUSE8042_OBJS += mouse8042.o

1004 FDC_OBJS += fd.o

1006 ASY_OBJS += asy.o

1008 ECPP_OBJS += ecpp.o

1010 VUIDM3P_OBJS += vuidmice.o vuidm3p.o

1012 VUIDM4P_OBJS += vuidmice.o vuidm4p.o

1014 VUIDM5P_OBJS += vuidmice.o vuidm5p.o

1016 VUIDPS2_OBJS += vuidmice.o vuidps2.o

1018 HPCSV_OBJS += hpcsvc.o

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

1022 PCIHPNEXUS_OBJS += pcihp.o

1024 OPENEEP_OBJS += openprom.o

1026 RANDOM_OBJS += random.o

1028 PSHOT_OBJS += pshot.o

1030 GEN_DRV_OBJS += gen_drv.o

1032 TCLIENT_OBJS += tclient.o

1034 TPHCI_OBJS += tphci.o

1036 TVHCI_OBJS += tvhci.o

1038 EMUL64_OBJS += emul64.o emul64_bsd.o

1040 FCP_OBJS += fcp.o

1042 FCIP_OBJS += fcip.o

1044 FCSM_OBJS += fcsm.o

1046 FCTL_OBJS += fctl.o

1048 FP_OBJS += fp.o

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

```

1053 QLC_FW_2200_OBJS += ql_fw_2200.o
1055 QLC_FW_2300_OBJS += ql_fw_2300.o
1057 QLC_FW_2400_OBJS += ql_fw_2400.o
1059 QLC_FW_2500_OBJS += ql_fw_2500.o
1061 QLC_FW_6322_OBJS += ql_fw_6322.o
1063 QLC_FW_8100_OBJS += ql_fw_8100.o
1065 QLGE_OBJS += qlge.o qlge_dbg.o qlge_flash.o qlge_fm.o qlge_gld.o qlge_mpi.o
1067 ZCONS_OBJS += zcons.o
1069 NV_SATA_OBJS += nv_sata.o
1071 SI3124_OBJS += si3124.o
1073 AHCI_OBJS += ahci.o
1075 PCIIDE_OBJS += pci-ide.o
1077 PCEPP_OBJS += pcepp.o
1079 CPC_OBJS += cpc.o
1081 CPUID_OBJS += cpuid_drv.o
1083 SYSEVENT_OBJS += sysevent.o
1085 BL_OBJS += bl.o
1087 DRM_OBJS += drm_sunmod.o drm_kstat.o drm_agpsupport.o \
1088             drm_auth.o drm_bufs.o drm_context.o drm_dma.o \
1089             drm_drawable.o drm_drv.o drm_fops.o drm_ioctl.o drm_irq.o \
1090             drm_lock.o drm_memory.o drm_msg.o drm_pci.o drm_scatter.o \
1091             drm_cache.o drm_gem.o drm_mm.o ati_pcigart.o
1093 FM_OBJS += devfm.o devfm_machdep.o
1095 RTLS_OBJS += rtls.o
1097 #
1098 #             exec modules
1099 #
1100 AOUTEXEC_OBJS += aout.o
1102 ELFEXEC_OBJS += elf.o elf_notes.o old_notes.o
1104 INTPEXEC_OBJS += intp.o
1106 SHBINEXEC_OBJS += shbin.o
1108 JAVAEXEC_OBJS += java.o
1110 #
1111 #             file system modules
1112 #
1113 AUTOFS_OBJS += auto_vfsops.o auto_vnops.o auto_subr.o auto_xdr.o auto_sys.o
1115 CACHEFS_OBJS += cacheofs_cnode.o          cacheofs_cod.o \
1116                 cacheofs_dir.o             cacheofs_dlog.o cacheofs_filegrp.o \
1117                 cacheofs_fscache.o          cacheofs_ioctl.o cacheofs_log.o \

```

```

1118                 cacheofs_module.o \
1119                 cacheofs_noopc.o          cacheofs_resource.o \
1120                 cacheofs_strict.o \
1121                 cacheofs_subr.o            cacheofs_vfsops.o \
1122                 cacheofs_vnops.o
1124 DCFS_OBJS += dc_vnops.o
1126 DEVFS_OBJS += devfs_subr.o devfs_vfsops.o devfs_vnops.o
1128 DEV_OBJS += sdev_subr.o sdev_vfsops.o sdev_vnops.o \
1129             sdev_ptsops.o sdev_zvolops.o sdev_comm.o \
1130             sdev_profile.o sdev_ncache.o sdev_netops.o \
1131             sdev_ipnetops.o \
1132             sdev_vtops.o
1134 CTFS_OBJS += ctfs_all.o ctfs_cdir.o ctfs_ctl.o ctfs_event.o \
1135             ctfs_latest.o ctfs_root.o ctfs_sym.o ctfs_tdir.o ctfs_tmpl.o
1137 OBJFS_OBJS += objfs_vfs.o objfs_root.o objfs_common.o \
1138             objfs_odir.o objfs_data.o
1140 FDFS_OBJS += fdops.o
1142 FIFO_OBJS += fifosubr.o fifovnops.o
1144 PIPE_OBJS += pipe.o
1146 HSFS_OBJS += hsfs_node.o hsfs_subr.o hsfs_vfsops.o hsfs_vnops.o \
1147             hsfs_susp.o hsfs_rrip.o hsfs_susp_subr.o
1149 LOFS_OBJS += lofs_subr.o lofs_vfsops.o lofs_vnops.o
1151 NAMEFS_OBJS += namevfs.o namevno.o
1153 NFS_OBJS += nfs_client.o nfs_common.o nfs_dump.o \
1154             nfs_subr.o nfs_vfsops.o nfs_vnops.o \
1155             nfs_xdr.o nfs_sys.o nfs_strerror.o \
1156             nfs3_vfsops.o nfs3_vnops.o nfs3_xdr.o \
1157             nfs_acl_vnops.o nfs_acl_xdr.o nfs4_vfsops.o \
1158             nfs4_vnops.o nfs4_xdr.o nfs4_idmap.o \
1159             nfs4_shadow.o nfs4_subr.o \
1160             nfs4_attr.o nfs4_rnode.o nfs4_client.o \
1161             nfs4_acache.o nfs4_common.o nfs4_client_state.o \
1162             nfs4_callback.o nfs4_recovery.o nfs4_client_secinfo.o \
1163             nfs4_client_debug.o nfs_stats.o \
1164             nfs4_acl.o nfs4_stub_vnops.o nfs_cmd.o
1166 NFSSRV_OBJS += nfs_server.o nfs_srv.o nfs3_srv.o \
1167             nfs_acl_srv.o nfs_auth.o nfs_auth_xdr.o \
1168             nfs_export.o nfs_log.o nfs_log_xdr.o \
1169             nfs4_srv.o nfs4_state.o nfs4_srv_attr.o \
1170             nfs4_srv_ns.o nfs4_db.o nfs4_srv_deleg.o \
1171             nfs4_deleg_ops.o nfs4_srv_readdir.o nfs4_dispatch.o
1173 SMBSRV_SHARED_OBJS += \
1174             smb_inet.o \
1175             smb_match.o \
1176             smb_msgbuf.o \
1177             smb_oem.o \
1178             smb_string.o \
1179             smb_utf8.o \
1180             smb_door_legacy.o \
1181             smb_xdr.o \
1182             smb_token.o \
1183             smb_token_xdr.o \

```

new/usr/src/uts/common/Makefile.files

```

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

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

```

19

new/usr/src/uts/common/Makefile.files

```

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

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

1259 PROC_OBJS += prcontrol.o priotcl.o prsubr.o prusr.o \
1260         prvfsops.o prvnops.o

1262 MNTFS_OBJS += mntvfsops.o mntvnops.o

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

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

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

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

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

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

1290 VSCAN_OBJS += vscan_drv.o vscan_svc.o vscan_door.o

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

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

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

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

1313 MD_DERIVED_OBJS = metamed_xdr.o meta_basic_xdr.o

1315 SOFTPART_OBJS += sp.o sp_ioctl.o

```

20

```

1317 STRIPE_OBJS += stripe.o stripe_ioctl.o
1319 HOTSPARES_OBJS += hotspares.o
1321 RAID_OBJS += raid.o raid_ioctl.o raid_replay.o raid_resync.o raid_hotspare.o
1323 MIRROR_OBJS += mirror.o mirror_ioctl.o mirror_resync.o
1325 NOTIFY_OBJS += md_notify.o
1327 TRANS_OBJS += mdtrans.o trans_ioctl.o trans_log.o
1329 ZFS_COMMON_OBJS += \
1330     arc.o \
1331     blkptr.o \
1332     bplist.o \
1333     bpobj.o \
1334     bptree.o \
1335     dbuf.o \
1336     ddt.o \
1337     ddt_zap.o \
1338     dmu.o \
1339     dmu_diff.o \
1340     dmu_send.o \
1341     dmu_object.o \
1342     dmu_objset.o \
1343     dmu_traverse.o \
1344     dmu_tx.o \
1345     dnode.o \
1346     dnode_sync.o \
1347     dsl_bookmark.o \
1348     dsl_dir.o \
1349     dsl_dataset.o \
1350     dsl_deadlist.o \
1351     dsl_destroy.o \
1352     dsl_pool.o \
1353     dsl_synctask.o \
1354     dsl_userhold.o \
1355     dmu_zfetch.o \
1356     dsl_deleg.o \
1357     dsl_prop.o \
1358     dsl_scan.o \
1359     zfeature.o \
1360     gzip.o \
1361     lz4.o \
1362     lzjb.o \
1363     metaslab.o \
1364     range_tree.o \
1365     refcount.o \
1366     rrwlock.o \
1367     sa.o \
1368     sha256.o \
1369     spa.o \
1370     spa_config.o \
1371     spa_errlog.o \
1372     spa_history.o \
1373     spa_misc.o \
1374     space_map.o \
1375     space_reftree.o \
1376     txg.o \
1377     uberblock.o \
1378     unique.o \
1379     vdev.o \
1380     vdev_cache.o \
1381     vdev_file.o \

```

```

1382     vdev_label.o \
1383     vdev_mirror.o \
1384     vdev_missing.o \
1385     vdev_queue.o \
1386     vdev_raidz.o \
1387     vdev_root.o \
1388     zap.o \
1389     zap_leaf.o \
1390     zap_micro.o \
1391     zfs_byteswap.o \
1392     zfs_debug.o \
1393     zfs_fm.o \
1394     zfs_fuid.o \
1395     zfs_sa.o \
1396     zfs_znode.o \
1397     zil.o \
1398     zio.o \
1399     zio_checksum.o \
1400     zio_compress.o \
1401     zio_inject.o \
1402     zle.o \
1403     zrlock.o
1405 ZFS_SHARED_OBJS += \
1406     zfeature_common.o \
1407     zfs_comutil.o \
1408     zfs_deleg.o \
1409     zfs_fletcher.o \
1410     zfs_namecheck.o \
1411     zfs_prop.o \
1412     zpool_prop.o \
1413     zprop_common.o
1415 ZFS_OBJS += \
1416     $(ZFS_COMMON_OBJS) \
1417     $(ZFS_SHARED_OBJS) \
1418     vdev_disk.o \
1419     zfs_acl.o \
1420     zfs_ctldir.o \
1421     zfs_dir.o \
1422     zfs_ioctl.o \
1423     zfs_log.o \
1424     zfs_onexit.o \
1425     zfs_replay.o \
1426     zfs_rlock.o \
1427     zfs_vfsops.o \
1428     zfs_vnops.o \
1429     zvol.o
1431 ZUT_OBJS += \
1432     zut.o
1434 # \
1435 # streams modules
1436 #
1437 BUFMOD_OBJS += bufmod.o
1439 CONNLD_OBJS += connld.o
1441 DEDUMP_OBJS += dedump.o
1443 DRCOMPAT_OBJS += drcompat.o
1445 LDLINUX_OBJS += ldlinux.o
1447 LDTERM_OBJS += ldterm.o uwidth.o

```

```

1449 PKCT_OBJS +=      pckt.o
1451 PFMOD_OBJS +=      pfmod.o
1453 PTEM_OBJS +=      ptem.o
1455 REDIRMOD_OBJS +=  strredirm.o
1457 TIMOD_OBJS +=      timod.o
1459 TIRDWR_OBJS +=     tirdwr.o
1461 TTCOMPAT_OBJS +=   ttcompat.o
1463 LOG_OBJS +=         log.o
1465 PIPEMOD_OBJS +=    pipemod.o

1467 RPCMOD_OBJS +=      rpcmod.o      clnt_cots.o      clnt_clts.o \
1468                        clnt_gen.o      clnt_perr.o      mt_rpcinit.o      rpc_calmsg.o \
1469                        rpc_prot.o      rpc_sztypes.o   rpc_subr.o      rpcb_prot.o \
1470                        svc.o          svc_clts.o      svc_gen.o      svc_cots.o \
1471                        rpcsys.o      xdr_sizeof.o   clnt_rdma.o     svc_rdma.o \
1472                        xdr_rdma.o      rdma_subr.o     xdrdma_sizeof.o

1474 KLMMOD_OBJS +=      klmmod.o \
1475                        nlm_impl.o \
1476                        nlm_rpc_handle.o \
1477                        nlm_dispatch.o \
1478                        nlm_rpc_svc.o \
1479                        nlm_client.o \
1480                        nlm_service.o \
1481                        nlm_prot_clnt.o \
1482                        nlm_prot_xdr.o \
1483                        nlm_rpc_clnt.o \
1484                        nsm_addr_clnt.o \
1485                        nsm_addr_xdr.o \
1486                        sm_inter_clnt.o \
1487                        sm_inter_xdr.o

1489 KLMOPS_OBJS +=      klmops.o

1491 TLIMOD_OBJS +=      tlimod.o      t_kalloc.o      t_kbind.o      t_kclose.o \
1492                        t_kconnect.o  t_kfree.o      t_kgtstate.o   t_kopen.o \
1493                        t_krcvdat.o   t_ksndudat.o  t_kspoll.o     t_kunbind.o \
1494                        t_kutil.o

1496 RLMOD_OBJS +=      rlmmod.o
1498 TELMOD_OBJS +=      telmod.o
1500 CRYPTMOD_OBJS +=    cryptmod.o
1502 KB_OBJS +=          kbd.o          keytables.o

1504 #
1505 #                      ID mapping module
1506 #
1507 IDMAP_OBJS +=        idmap_mod.o     idmap_kapi.o     idmap_xdr.o     idmap_cache.o

1509 #
1510 #                      scheduling class modules
1511 #
1512 SDC_OBJS +=          sysdc.o

```

```

1514 RT_OBJS +=          rt.o
1515 RT_DPTBL_OBJS +=    rt_dptbl.o

1517 TS_OBJS +=          ts.o
1518 TS_DPTBL_OBJS +=    ts_dptbl.o

1520 IA_OBJS +=          ia.o

1522 FSS_OBJS +=          fss.o

1524 FX_OBJS +=          fx.o
1525 FX_DPTBL_OBJS +=    fx_dptbl.o

1527 #
1528 #                      Inter-Process Communication (IPC) modules
1529 #
1530 IPC_OBJS +=          ipc.o

1532 IPCMSG_OBJS +=       msg.o

1534 IPCSEM_OBJS +=       sem.o

1536 IPCSHM_OBJS +=       shm.o

1538 #
1539 #                      bignum module
1540 #
1541 COMMON_BIGNUM_OBJS += bignum_mod.o bignumimpl.o

1543 BIGNUM_OBJS += $(COMMON_BIGNUM_OBJS) $(BIGNUM_PSR_OBJS)

1545 #
1546 #                      kernel cryptographic framework
1547 #
1548 KCF_OBJS +=          kcf.o kcf_callprov.o kcf_cbufcall.o kcf_cipher.o kcf_crypto.o \
1549                        kcf_cryptoadm.o kcf_ctxops.o kcf_digest.o kcf_dual.o \
1550                        kcf_keys.o kcf_mac.o kcf_mech_tabs.o kcf_miscapi.o \
1551                        kcf_object.o kcf_policy.o kcf_prov_lib.o kcf_prov_tabs.o \
1552                        kcf_sched.o kcf_session.o kcf_sign.o kcf_spi.o kcf_verify.o \
1553                        kcf_random.o modes.o ecb.o cbc.o ctr.o ccm.o gcm.o \
1554                        fips_random.o

1556 CRYPTOADM_OBJS +=    cryptoadm.o

1558 CRYPTO_OBJS +=        crypto.o

1560 DPROV_OBJS +=          dprov.o

1562 DCA_OBJS +=            dca.o dca_3des.o dca_debug.o dca_dsa.o dca_kstat.o dca_rng.o \
1563                        dca_rsa.o

1565 AESPROV_OBJS +=        aes.o aes_impl.o aes_modes.o

1567 ARCFOURPROV_OBJS +=    arcfour.o arcfour_crypt.o

1569 BLOWFISHPROV_OBJS +=   blowfish.o blowfish_impl.o

1571 ECCPROV_OBJS +=        ecc.o ec.o ec2_163.o ec2_mont.o ecdecode.o ecl_mult.o \
1572                        ecp_384.o ecp_jac.o ec2_193.o ecl.o ecp_192.o ecp_521.o \
1573                        ecp_jm.o ec2_233.o ecl_curve.o ecp_224.o ecp_aff.o \
1574                        ecp_mont.o ec2_aff.o ec_naf.o ecl_gf.o ecp_256.o mp_gf2m.o \
1575                        mpi.o mplogic.o mpmontg.o mpprime.o oid.o \
1576                        secitem.o ec2_test.o ecp_test.o

1578 RSAPROV_OBJS +=        rsa.o rsa_impl.o pkcs1.o

```

```

1580 SWRANDPROV_OBJS += swrand.o

1582 #
1583 #             kernel SSL
1584 #
1585 KSSL_OBJS +=      kssl.o ksslioctl.o

1587 KSSL_SOCKETFIL_MOD_OBJS += ksslfilter.o ksslapi.o ksslrec.o

1589 #
1590 #             misc. modules
1591 #

1593 C2AUDIT_OBJS +=  adr.o audit.o audit_event.o audit_io.o \
1594                  audit_path.o audit_start.o audit_syscalls.o audit_token.o \
1595                  audit_mem.o

1597 PCIC_OBJS +=      pcic.o

1599 RPCSEC_OBJS +=      secmod.o      sec_clnt.o      sec_svc.o      sec_gen.o \
1600                  auth_des.o      auth_kern.o      auth_none.o      auth_loopb.o \
1601                  authdesprt.o      authdesubr.o      authu_prot.o \
1602                  key_call.o      key_prot.o      svc_authu.o      svcauthdes.o

1604 RPCSEC_GSS_OBJS +=      rpcsec_gssmod.o rpcsec_gss.o rpcsec_gss_misc.o \
1605                  rpcsec_gss_utils.o svc_rpcsec_gss.o

1607 CONSCONFIG_OBJS += consconfig.o

1609 CONSCONFIG_DACF_OBJS += consconfig_dacf.o consplat.o

1611 TEM_OBJS += tem.o tem_safe.o 6x10.o 7x14.o 12x22.o

1613 KBTRANS_OBJS +=      \
1614                  kbtrans.o      \
1615                  kbtrans_keytables.o \
1616                  kbtrans_polled.o \
1617                  kbtrans_streams.o \
1618                  usb_keytables.o

1620 KGSSD_OBJS +=      gssd_clnt_stubs.o gssd_handle.o gssd_prot.o \
1621                  gss_display_name.o gss_release_name.o gss_import_name.o \
1622                  gss_release_buffer.o gss_release_oid_set.o gen_oids.o gssdmod.o

1624 KGSSD_DERIVED_OBJS = gssd_xdr.o

1626 KGSS_DUMMY_OBJS += dmech.o

1628 KSOCKET_OBJS += ksocket.o ksocket_mod.o

1630 CRYPTO= cksumtypes.o decrypt.o encrypt.o encrypt_length.o etypes.o \
1631          nfold.o verify_checksum.o prng.o block_size.o make_checksum.o \
1632          checksum_length.o hmac.o default_state.o mandatory_sumtype.o

1634 # crypto/des
1635 CRYPTO_DES= f_cbc.o f_cksum.o f_parity.o weak_key.o d3_cbc.o ef_crypto.o

1637 CRYPTO_DK= checksum.o derive.o dk_decrypt.o dk_encrypt.o

1639 CRYPTO_ARCFOUR= k5_arcfour.o

1641 # crypto/enc_provider
1642 CRYPTO_ENC= des.o des3.o arcfour_provider.o aes_provider.o

1644 # crypto/hash_provider
1645 CRYPTO_HASH= hash_kef_generic.o hash_kmd5.o hash_crc32.o hash_kshal.o

```

```

1647 # crypto/keyhash_provider
1648 CRYPTO_KEYHASH= descbc.o k5_kmd5des.o k_hmac_md5.o

1650 # crypto/crc32
1651 CRYPTO_CRC32= crc32.o

1653 # crypto/old
1654 CRYPTO_OLD= old_decrypt.o old_encrypt.o

1656 # crypto/raw
1657 CRYPTO_RAW= raw_decrypt.o raw_encrypt.o

1659 K5_KRB= kfree.o copy_key.o \
1660         parse.o init_ctx.o \
1661         ser_adata.o ser_addr.o \
1662         ser_auth.o ser_cksum.o \
1663         ser_key.o ser_princ.o \
1664         serialize.o unparse.o \
1665         ser_actx.o

1667 K5_OS= timeofday.o toffset.o \
1668        init_os_ctx.o c_ustime.o

1670 SEAL= seal.o unseal.o

1672 MECH= delete_sec_context.o \
1673        import_sec_context.o \
1674        gssapi_krb5.o \
1675        k5seal.o k5unseal.o k5sealv3.o \
1676        ser_sctx.o \
1677        sign.o \
1678        util_crypt.o \
1679        util_validate.o util_ordering.o \
1680        util_seqnum.o util_set.o util_seed.o \
1681        wrap_size_limit.o verify.o

1685 MECH_GEN= util_token.o

1688 KGSS_KRB5_OBJS += krb5mech.o \
1689                  $(MECH) $(SEAL) $(MECH_GEN) \
1690                  $(CRYPTO) $(CRYPTO_DES) $(CRYPTO_DK) $(CRYPTO_ARCFOUR) \
1691                  $(CRYPTO_ENC) $(CRYPTO_HASH) \
1692                  $(CRYPTO_KEYHASH) $(CRYPTO_CRC32) \
1693                  $(CRYPTO_OLD) \
1694                  $(CRYPTO_RAW) $(K5_KRB) $(K5_OS)

1696 DES_OBJS +=      des_crypt.o des_impl.o des_ks.o des_soft.o

1698 DLBOOT_OBJS += bootparam_xdr.o nfs_dlinet.o scan.o

1700 KRTLD_OBJS +=      kobj_bootflags.o getoptstr.o \
1701                  kobj.o kobj_kdi.o kobj_lm.o kobj_subr.o

1703 MOD_OBJS +=      modctl.o modsubr.o modsysfile.o modconf.o modhash.o

1705 STRPLUMB_OBJS += strplumb.o

1707 CPR_OBJS +=      cpr_driver.o cpr_dump.o \
1708                  cpr_main.o cpr_misc.o cpr_mod.o cpr_stat.o \
1709                  cpr_uthread.o

1711 PROF_OBJS +=      prf.o

```



```
1713 SE_OBJS += se_driver.o
1715 SYSACCT_OBJS += acct.o
1717 ACCTCTL_OBJS += acctctl.o
1719 EXACCTSYS_OBJS += exacctsys.o
1721 KAIO_OBJS += aio.o
1723 PCMCIA_OBJS += pcmcia.o cs.o cis.o cis_callout.o cis_handlers.o cis_params.o
1725 BUSRA_OBJS += busra.o
1727 PCS_OBJS += pcs.o
1729 PSET_OBJS += pset.o
1731 OHCI_OBJS += ohci.o ohci_hub.o ohci_polled.o
1733 UHCI_OBJS += uhci.o uhciutil.o uhcitgt.o uhcihub.o uhci_polled.o
1735 EHCI_OBJS += ehci.o ehci_hub.o ehci_xfer.o ehci_intr.o ehci_util.o ehci_polled.o
1737 HUBD_OBJS += hubd.o
1739 USB_MID_OBJS += usb_mid.o
1741 USB_IA_OBJS += usb_ia.o
1743 UWBA_OBJS += uwba.o uwbai.o
1745 SCSA2USB_OBJS += scsa2usb.o usb_ms_bulkonly.o usb_ms_cbi.o
1747 HWAHC_OBJS += hwahc.o hwahc_util.o
1749 WUSB_DF_OBJS += wusb_df.o
1750 WUSB_FWMOD_OBJS += wusb_fwmod.o
1752 IPF_OBJS += ip_fil_solaris.o fil.o solaris.o ip_state.o ip_frag.o ip_nat.o \
1753 ip_proxy.o ip_auth.o ip_pool.o ip_htable.o ip_lookup.o \
1754 ip_log.o misc.o ip_compat.o ip_nat6.o drand48.o
1756 IPD_OBJS += ipd.o
1758 IBD_OBJS += ibd.o ibd_cm.o
1760 EIBNX_OBJS += enx_main.o enx_hdlrs.o enx_ibt.o enx_log.o enx_fip.o \
1761 enx_misc.o enx_q.o enx_ctl.o
1763 EOIB_OBJS += eib_adm.o eib_chan.o eib_cmnn.o eib_ctl.o eib_data.o \
1764 eib_fip.o eib_ibt.o eib_log.o eib_mac.o eib_main.o \
1765 eib_rsrc.o eib_svc.o eib_vnic.o
1767 DLPSTUB_OBJS += dlpistub.o
1769 SDP_OBJS += sdpddi.o
1771 TRILL_OBJS += trill.o
1773 CTF_OBJS += ctf_create.o ctf_decl.o ctf_error.o ctf_hash.o ctf_labels.o \
1774 ctf_lookup.o ctf_open.o ctf_types.o ctf_util.o ctf_subr.o ctf_mod.o
1776 SMBIOS_OBJS += smb_error.o smb_info.o smb_open.o smb_subr.o smb_dev.o
```

```
1778 RPCIB_OBJS += rpcib.o
1780 KMDB_OBJS += kdrv.o
1782 AFE_OBJS += afe.o
1784 BGE_OBJS += bge_main2.o bge_chip2.o bge_kstats.o bge_log.o bge_ndd.o \
1785 bge_atomic.o bge_mii.o bge_send.o bge_recv2.o bge_mii_5906.o
1787 DMFE_OBJS += dmfe_log.o dmfe_main.o dmfe_mii.o
1789 EFE_OBJS += efe.o
1791 ELXL_OBJS += elxl.o
1793 HME_OBJS += hme.o
1795 IXGB_OBJS += ixgb.o ixgb_atomic.o ixgb_chip.o ixgb_gld.o ixgb_kstats.o \
1796 ixgb_log.o ixgb_ndd.o ixgb_rx.o ixgb_tx.o ixgb_xmii.o
1798 NGE_OBJS += nge_main.o nge_atomic.o nge_chip.o nge_ndd.o nge_kstats.o \
1799 nge_log.o nge_rx.o nge_tx.o nge_xmii.o
1801 PCN_OBJS += pcn.o
1803 RGE_OBJS += rge_main.o rge_chip.o rge_ndd.o rge_kstats.o rge_log.o rge_rxtx.o
1805 URTW_OBJS += urtw.o
1807 ARN_OBJS += arn_hw.o arn_eeprom.o arn_mac.o arn_calib.o arn_anis.o arn_phy.o arn_
1808 arn_main.o arn_recv.o arn_xmit.o arn_rc.o
1810 ATH_OBJS += ath_aux.o ath_main.o ath_osdep.o ath_rate.o
1812 ATU_OBJS += atu.o
1814 IPW_OBJS += ipw2100_hw.o ipw2100.o
1816 IWI_OBJS += ipw2200_hw.o ipw2200.o
1818 IWH_OBJS += iwh.o
1820 IWK_OBJS += iwk2.o
1822 IWP_OBJS += iwp.o
1824 MWL_OBJS += mwl.o
1826 MWLFW_OBJS += mwlfw_mode.o
1828 WPI_OBJS += wpi.o
1830 RAL_OBJS += rt2560.o ral_rate.o
1832 RUM_OBJS += rum.o
1834 RWD_OBJS += rt2661.o
1836 RWN_OBJS += rt2860.o
1838 UATH_OBJS += uath.o
1840 UATHFW_OBJS += uathfw_mod.o
1842 URAL_OBJS += ural.o
```

```

1844 RTW_OBJS += rtw.o smc93cx6.o rtwphy.o rtwphyio.o
1846 ZYD_OBJS += zyd.o zyd_usb.o zyd_hw.o zyd_fw.o
1848 MXFE_OBJS += mxfe.o
1850 MPTSAS_OBJS += mptsas.o mptsas_hash.o mptsas_impl.o mptsas_init.o \
1851                mptsas_raid.o mptsas_smhba.o
1853 MPTSAS3_OBJS += mptsas3.o mptsas3_hash.o mptsas3_impl.o mptsas3_init.o \
1854                mptsas3_raid.o mptsas3_smhba.o
1856 #endif /* ! codereview */
1857 SFE_OBJS += sfe.o sfe_util.o
1859 BFE_OBJS += bfe.o
1861 BRIDGE_OBJS += bridge.o
1863 IDM_SHARED_OBJS += base64.o
1865 IDM_OBJS += $(IDM_SHARED_OBJS) \
1866             idm.o idm_impl.o idm_text.o idm_conn_sm.o idm_so.o
1868 VR_OBJS += vr.o
1870 ATGE_OBJS += atge_main.o atge_lle.o atge_mii.o atge_ll.o atge_llc.o
1872 YGE_OBJS = yge.o
1874 #
1875 #       Build up defines and paths.
1876 #
1877 LINT_DEFS      += -Dunix
1879 #
1880 #       This duality can be removed when the native and target compilers
1881 #       are the same (or at least recognize the same command line syntax!)
1882 #       It is a bug in the current compilation system that the assembler
1883 #       can't process the -Y I, flag.
1884 #
1885 NATIVE_INC_PATH += $(INC_PATH) $(CCYFLAG)$ (UTSBASE)/common
1886 AS_INC_PATH     += $(INC_PATH) -I$(UTSBASE)/common
1887 INCLUDE_PATH    += $(INC_PATH) $(CCYFLAG)$ (UTSBASE)/common
1889 PCIEB_OBJS += pcieb.o
1891 #       Chelsio N110 10G NIC driver module
1892 #
1893 CH_OBJS = ch.o glue.o pe.o sge.o
1895 CH_COM_OBJS = ch_mac.o ch_subr.o csapi.o espi.o ixfl1010.o mc3.o mc4.o mc5.o \
1896             mv88elxxx.o mv88x201x.o my3126.o pm3393.o tp.o ulp.o \
1897             vsc7321.o vsc7326.o xpak.o
1899 #
1900 #       Chelsio Terminator 4 10G NIC nexus driver module
1901 #
1902 CXGBE_FW_OBJS = t4_fw.o t4_cfg.o
1903 CXGBE_COM_OBJS = t4_hw.o common.o
1904 CXGBE_NEX_OBJS = t4_nexus.o t4_sge.o t4_mac.o t4_ioctl.o shared.o \
1905             t4_l2t.o adapter.o osdep.o
1907 #
1908 #       Chelsio Terminator 4 10G NIC driver module
1909 #

```

```

1910 CXGBE_OBJS = cxgbe.o
1912 #
1913 #       PCI strings file
1914 #
1915 PCI_STRING_OBJS = pci_strings.o
1917 NET_DACF_OBJS += net_dacf.o
1919 #
1920 #       Xframe 10G NIC driver module
1921 #
1922 XGE_OBJS = xge.o xgell.o
1924 XGE_HAL_OBJS = xgehal-channel.o xgehal-fifo.o xgehal-ring.o xgehal-config.o \
1925             xgehal-driver.o xgehal-mm.o xgehal-stats.o xgehal-device.o \
1926             xge-queue.o xgehal-mgmt.o xgehal-mgmtaux.o
1928 #
1929 #       e1000/igb common objs
1930 #
1931 #       Historically e1000g and igb had separate copies of all of the common
1932 #       code. At this time while they are now sharing the same copy of it, they
1933 #       are building it into their own modules which is due to the differences
1934 #       in the osdep and debug portions of their code.
1935 #
1936 E1000API_OBJS += e1000_80003es2lan.o e1000_82540.o e1000_82541.o e1000_82542.o \
1937             e1000_82543.o e1000_82571.o e1000_api.o e1000_ich8lan.o \
1938             e1000_mac.o e1000_manage.o e1000_nvmm.o e1000_phy.o \
1939             e1000_82575.o e1000_i210.o e1000_mbx.o e1000_vf.o
1941 #
1942 #       e1000g module
1943 #
1944 E1000G_OBJS += e1000g_debug.o e1000g_main.o e1000g_alloc.o \
1945             e1000g_tx.o e1000g_rx.o e1000g_stat.o \
1946             e1000g_osdep.o e1000g_workarounds.o
1947
1949 #
1950 #       Intel 82575 1G NIC driver module
1951 #
1952 IGB_OBJS = igb_buf.o igb_debug.o igb_gld.o igb_log.o igb_main.o \
1953             igb_rx.o igb_stat.o igb_tx.o igb_osdep.o
1955 #
1956 #       Intel Pro/100 NIC driver module
1957 #
1958 IPRB_OBJS = iprb.o
1960 #
1961 #       Intel 10GbE PCIE NIC driver module
1962 #
1963 IXGBE_OBJS = ixgbe_82598.o ixgbe_82599.o ixgbe_api.o \
1964             ixgbe_common.o ixgbe_phy.o \
1965             ixgbe_buf.o ixgbe_debug.o ixgbe_gld.o \
1966             ixgbe_log.o ixgbe_main.o \
1967             ixgbe_osdep.o ixgbe_rx.o ixgbe_stat.o \
1968             ixgbe_tx.o ixgbe_x540.o ixgbe_mbx.o
1970 #
1971 #       NIU 10G/1G driver module
1972 #
1973 NXGE_OBJS = nxge_mac.o nxge_ipp.o nxge_rxdma.o \
1974             nxge_txdma.o nxge_txc.o nxge_main.o \
1975             nxge_hw.o nxge_fzc.o nxge_virtual.o \

```

```

1976         nxge_send.o nxge_classify.o nxge_fflp.o      \
1977         nxge_fflp_hash.o nxge_ndd.o nxge_kstats.o     \
1978         nxge_zcp.o nxge_fm.o nxge_espc.o nxge_hv.o     \
1979         nxge_hio.o nxge_hio_guest.o nxge_intr.o

1981 NXGE_NPI_OBJS = \
1982         npi.o npi_mac.o npi_ipp.o                     \
1983         npi_txdma.o npi_rxdma.o npi_txc.o             \
1984         npi_zcp.o npi_espc.o npi_fflp.o               \
1985         npi_vir.o

1987 NXGE_HCALL_OBJS = \
1988         nxge_hcall.o

1990 #
1991 # Virtio modules
1992 #

1994 # Virtio core
1995 VIRTIO_OBJS = virtio.o

1997 # Virtio block driver
1998 VIOBLK_OBJS = vioblk.o

2000 #
2001 #         kiconv modules
2002 #
2003 KICONV_EMEA_OBJS += kiconv_emea.o

2005 KICONV_JA_OBJS += kiconv_ja.o

2007 KICONV_KO_OBJS += kiconv_ck_common.o kiconv_ko.o

2009 KICONV_SC_OBJS += kiconv_ck_common.o kiconv_sc.o

2011 KICONV_TC_OBJS += kiconv_ck_common.o kiconv_tc.o

2013 #
2014 #         AAC module
2015 #
2016 AAC_OBJS = aac.o aac_ioctl.o

2018 #
2019 #         sdcard modules
2020 #
2021 SDA_OBJS = sda_cmd.o sda_host.o sda_init.o sda_mem.o sda_mod.o sda_slot.o
2022 SDHOST_OBJS = sdhost.o

2024 #
2025 #         hxge 10G driver module
2026 #
2027 HXGE_OBJS = hxge_main.o hxge_vmac.o hxge_send.o      \
2028         hxge_txdma.o hxge_rxdma.o hxge_virtual.o     \
2029         hxge_fm.o hxge_fzc.o hxge_hw.o hxge_kstats.o \
2030         hxge_ndd.o hxge_pfc.o                       \
2031         hpi.o hpi_vmac.o hpi_rxdma.o hpi_txdma.o     \
2032         hpi_vir.o hpi_pfc.o

2034 #
2035 #         MEGARAID_SAS module
2036 #
2037 MEGA_SAS_OBJS = megaraidd_sas.o

2039 #
2040 #         MR_SAS module
2041 #

```

```

2042 MR_SAS_OBJS = ld_pd_map.o mr_sas.o mr_sas_tbolt.o mr_sas_list.o

2044 #
2045 #         CPQARY3 module
2046 #
2047 CPQARY3_OBJS = cpqary3.o cpqary3_noe.o cpqary3_talk2ctlr.o \
2048         cpqary3_isr.o cpqary3_transport.o cpqary3_mem.o \
2049         cpqary3_scsi.o cpqary3_util.o cpqary3_ioctl.o \
2050         cpqary3_bd.o

2052 #
2053 #         ISCSI_INITIATOR module
2054 #
2055 ISCSI_INITIATOR_OBJS = chap.o iscsi_io.o iscsi_thread.o \
2056         iscsi_ioctl.o iscsid.o iscsi.o \
2057         iscsi_login.o isns_client.o iscsiAuthClient.o \
2058         iscsi_lun.o iscsiAuthClientGlue.o \
2059         iscsi_net.o nvfile.o iscsi_cmd.o \
2060         iscsi_queue.o persistent.o iscsi_conn.o \
2061         iscsi_sess.o radius_auth.o iscsi_crc.o \
2062         iscsi_stats.o radius_packet.o iscsi_doorctl.o \
2063         iscsi_targetparam.o utils.o kifconf.o

2065 #
2066 #         ntxn 10Gb/1Gb NIC driver module
2067 #
2068 NTXN_OBJS = unnm_nic_init.o unnm_gem.o unnm_nic_hw.o unnm_ndd.o \
2069         unnm_nic_main.o unnm_nic_isr.o unnm_nic_ctx.o niu.o

2071 #
2072 #         Myricom 10Gb NIC driver module
2073 #
2074 MYRI10GE_OBJS = myri10ge.o myri10ge_lro.o

2076 #
2077 #         nulldriver module
2078 #
2079 NULLDRIVER_OBJS = nulldriver.o

2080 TPM_OBJS = tpm.o tpm_hcall.o

```

new/usr/src/uts/common/Makefile.rules

1

```
*****
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
*****

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

22 #
23 # Copyright (c) 1991, 2010, Oracle and/or its affiliates. All rights reserved.
24 # Copyright (c) 2012 Nexenta Systems, Inc. All rights reserved.
25 # Copyright 2013 Garrett D'Amore <garrett@damore.org>
26 #

28 #
29 # uts/common/Makefile.rules
30 #
31 # This Makefile defines all the file build rules for the directory
32 # uts/common and its children. These are the source files which may
33 # be considered common to all SunOS systems.
34 #
35 # The following two-level ordering must be maintained in this file.
36 # Lines are sorted first in order of decreasing specificity based on
37 # the first directory component. That is, sun4u rules come before
38 # sparc rules come before common rules.
39 #
40 # Lines whose initial directory components are equal are sorted
41 # alphabetically by the remaining components.

43 #
44 # Section 1a: C objects build rules
45 #
46 $(OBJSDIR)/%.o: $(COMMONBASE)/crypto/aes/%.c
47 $(COMPILE.c) -o $@ $<
48 $(CTFCONVERT_O)

50 $(OBJSDIR)/%.o: $(COMMONBASE)/crypto/arcfour/%.c
51 $(COMPILE.c) -o $@ $<
52 $(CTFCONVERT_O)

54 $(OBJSDIR)/%.o: $(COMMONBASE)/crypto/blowfish/%.c
55 $(COMPILE.c) -o $@ $<
56 $(CTFCONVERT_O)

58 $(OBJSDIR)/%.o: $(COMMONBASE)/crypto/ecc/%.c
59 $(COMPILE.c) -o $@ $<
60 $(CTFCONVERT_O)
```

new/usr/src/uts/common/Makefile.rules

2

```
62 $(OBJSDIR)/%.o: $(COMMONBASE)/crypto/modes/%.c
63 $(COMPILE.c) -o $@ $<
64 $(CTFCONVERT_O)

66 $(OBJSDIR)/%.o: $(COMMONBASE)/crypto/padding/%.c
67 $(COMPILE.c) -o $@ $<
68 $(CTFCONVERT_O)

70 $(OBJSDIR)/%.o: $(COMMONBASE)/crypto/rng/%.c
71 $(COMPILE.c) -o $@ $<
72 $(CTFCONVERT_O)

74 $(OBJSDIR)/%.o: $(COMMONBASE)/crypto/rsa/%.c
75 $(COMPILE.c) -o $@ $<
76 $(CTFCONVERT_O)

78 $(OBJSDIR)/%.o: $(COMMONBASE)/bignum/%.c
79 $(COMPILE.c) -o $@ $<
80 $(CTFCONVERT_O)

82 $(OBJSDIR)/%.o: $(UTSBASE)/common/bignum/%.c
83 $(COMPILE.c) -o $@ $<
84 $(CTFCONVERT_O)

86 $(OBJSDIR)/%.o: $(COMMONBASE)/mpi/%.c
87 $(COMPILE.c) -o $@ $<
88 $(CTFCONVERT_O)

90 $(OBJSDIR)/%.o: $(COMMONBASE)/acl/%.c
91 $(COMPILE.c) -o $@ $<
92 $(CTFCONVERT_O)

94 $(OBJSDIR)/%.o: $(COMMONBASE)/avl/%.c
95 $(COMPILE.c) -o $@ $<
96 $(CTFCONVERT_O)

98 $(OBJSDIR)/%.o: $(COMMONBASE)/ucode/%.c
99 $(COMPILE.c) -o $@ $<
100 $(CTFCONVERT_O)

102 $(OBJSDIR)/%.o: $(UTSBASE)/common/brand/snl/%.c
103 $(COMPILE.c) -o $@ $<
104 $(CTFCONVERT_O)

106 $(OBJSDIR)/%.o: $(UTSBASE)/common/brand/solaris10/%.c
107 $(COMPILE.c) -o $@ $<
108 $(CTFCONVERT_O)

110 $(OBJSDIR)/%.o: $(UTSBASE)/common/c2/%.c
111 $(COMPILE.c) -o $@ $<
112 $(CTFCONVERT_O)

114 $(OBJSDIR)/%.o: $(UTSBASE)/common/conf/%.c
115 $(COMPILE.c) -o $@ $<
116 $(CTFCONVERT_O)

118 $(OBJSDIR)/%.o: $(UTSBASE)/common/contract/%.c
119 $(COMPILE.c) -o $@ $<
120 $(CTFCONVERT_O)

122 $(OBJSDIR)/%.o: $(UTSBASE)/common/cpr/%.c
123 $(COMPILE.c) -o $@ $<
124 $(CTFCONVERT_O)

126 $(OBJSDIR)/%.o: $(UTSBASE)/common/ctf/%.c
127 $(COMPILE.c) -o $@ $<
```

new/usr/src/uts/common/Makefile.rules

```

128      $(CTFCONVERT_O)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```

3

new/usr/src/uts/common/Makefile.rules

```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```

4

new/usr/src/uts/common/Makefile.rules

```

260      $(CTFCONVERT_O)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```

5

new/usr/src/uts/common/Makefile.rules

```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

387 $(OBJSDIR)/%.o:      $(KMECHKRB5_BASE)/crypto/arcfour/%.c
388     $(COMPILE.c) $(KGSSDFFLAGS) -o $@ $<
389     $(CTFCONVERT_O)

391 $(OBJSDIR)/%.o:      $(KMECHKRB5_BASE)/crypto/dk/%.c

```

6

new/usr/src/uts/common/Makefile.rules

7

```
392 $(COMPILE.c) $(KGSSDFFLAGS) -o $$ $<
393 $(CTFCONVERT_O)

395 $(OBJSDIR)/%.o: $(KMECHKRB5_BASE)/crypto/enc_provider/%.c
396 $(COMPILE.c) $(KGSSDFFLAGS) -o $$ $<
397 $(CTFCONVERT_O)

399 $(OBJSDIR)/%.o: $(KMECHKRB5_BASE)/crypto/hash_provider/%.c
400 $(COMPILE.c) $(KGSSDFFLAGS) -o $$ $<
401 $(CTFCONVERT_O)

403 $(OBJSDIR)/%.o: $(KMECHKRB5_BASE)/crypto/keyhash_provider/%.c
404 $(COMPILE.c) $(KGSSDFFLAGS) -o $$ $<
405 $(CTFCONVERT_O)

407 $(OBJSDIR)/%.o: $(KMECHKRB5_BASE)/crypto/raw/%.c
408 $(COMPILE.c) $(KGSSDFFLAGS) -o $$ $<
409 $(CTFCONVERT_O)

411 $(OBJSDIR)/%.o: $(KMECHKRB5_BASE)/crypto/old/%.c
412 $(COMPILE.c) $(KGSSDFFLAGS) -o $$ $<
413 $(CTFCONVERT_O)

415 $(OBJSDIR)/%.o: $(KMECHKRB5_BASE)/krb5/krb/%.c
416 $(COMPILE.c) $(KGSSDFFLAGS) -o $$ $<
417 $(CTFCONVERT_O)

419 $(OBJSDIR)/%.o: $(KMECHKRB5_BASE)/krb5/os/%.c
420 $(COMPILE.c) $(KGSSDFFLAGS) -o $$ $<
421 $(CTFCONVERT_O)

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

425 $(OBJSDIR)/%.o: $(KMECHKRB5_BASE)/mech/%.c
426 $(COMPILE.c) $(KGSSDFFLAGS) -o $$ $<
427 $(CTFCONVERT_O)

429 $(OBJSDIR)/%.o: $(KMECHKRB5_BASE)/profile/%.c
430 $(COMPILE.c) $(KGSSDFFLAGS) -o $$ $<
431 $(CTFCONVERT_O)

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

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

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

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

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

453 $(OBJSDIR)/%.o: $(UTSBASE)/common/avs/ns/solaris/%.c
454 $(COMPILE.c) -o $$ $<
455 $(CTFCONVERT_O)

457 $(OBJSDIR)/%.o: $(UTSBASE)/common/avs/ns/sv/%.c
```

new/usr/src/uts/common/Makefile.rules

8

```
458 $(COMPILE.c) -o $$ $<
459 $(CTFCONVERT_O)

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

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

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

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

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

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

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

489 $(OBJSDIR)/%.o: $(UTSBASE)/common/inet/ksl/%.c
490 $(COMPILE.c) -o $$ $<
491 $(CTFCONVERT_O)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

553 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/1394/targets/scsal394/%.c
554     $(COMPILE.c) -o $@ $<
555     $(CTFCONVERT_O)

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

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

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

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

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

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

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

585 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/audio/impl/%.c
586     $(COMPILE.c) -o $@ $<
587     $(CTFCONVERT_O)

589 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/audio/ac97/%.c
```

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

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

597 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/audio/drv/audioemul0k/%.c
598     $(COMPILE.c) -o $@ $<
599     $(CTFCONVERT_O)

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

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

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

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

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

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

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

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

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

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

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

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

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

653 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/bfe/%.c
654     $(COMPILE.c) -o $@ $<
655     $(CTFCONVERT_O)
```



```

657 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/bge/%.c
658     $(COMPILE.c) -o $@ $<
659     $(CTFCONVERT_O)

661 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/blkdev/%.c
662     $(COMPILE.c) -o $@ $<
663     $(CTFCONVERT_O)

665 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/bpf/%.c
666     $(COMPILE.c) -o $@ $<
667     $(CTFCONVERT_O)

669 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/cardbus/%.c
670     $(COMPILE.c) -o $@ $<
671     $(CTFCONVERT_O)

673 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/comstar/stmf/%.c
674     $(COMPILE.c) -o $@ $<
675     $(CTFCONVERT_O)

677 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/comstar/port/fct/%.c
678     $(COMPILE.c) -o $@ $<
679     $(CTFCONVERT_O)

681 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/comstar/port/qlt/%.c
682     $(COMPILE.c) -o $@ $<
683     $(CTFCONVERT_O)

685 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/comstar/port/srpt/%.c
686     $(COMPILE.c) -o $@ $<
687     $(CTFCONVERT_O)

689 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/comstar/port/fcoet/%.c
690     $(COMPILE.c) -o $@ $<
691     $(CTFCONVERT_O)

693 $(OBJSDIR)/%.o: $(COMMONBASE)/iscsit/%.c
694     $(COMPILE.c) -o $@ $<
695     $(CTFCONVERT_O)

697 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/comstar/port/iscsit/%.c
698     $(COMPILE.c) -o $@ $<
699     $(CTFCONVERT_O)

701 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/comstar/port/pppt/%.c
702     $(COMPILE.c) -o $@ $<
703     $(CTFCONVERT_O)

705 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/comstar/lu/stmf_sbd/%.c
706     $(COMPILE.c) -o $@ $<
707     $(CTFCONVERT_O)

709 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/cpqary3/%.c
710     $(COMPILE.c) -o $@ $<
711     $(CTFCONVERT_O)

713 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/dld/%.c
714     $(COMPILE.c) -o $@ $<
715     $(CTFCONVERT_O)

717 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/dls/%.c
718     $(COMPILE.c) -o $@ $<
719     $(CTFCONVERT_O)

721 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/dmfe/%.c

```

```

722     $(COMPILE.c) -o $@ $<
723     $(CTFCONVERT_O)

725 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/drm/%.c
726     $(COMPILE.c) -o $@ $<
727     $(CTFCONVERT_O)

729 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/efe/%.c
730     $(COMPILE.c) -o $@ $<
731     $(CTFCONVERT_O)

733 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/elxl/%.c
734     $(COMPILE.c) -o $@ $<
735     $(CTFCONVERT_O)

737 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/fcoe/%.c
738     $(COMPILE.c) -o $@ $<
739     $(CTFCONVERT_O)

741 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/hme/%.c
742     $(COMPILE.c) -o $@ $<
743     $(CTFCONVERT_O)

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

749 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/hotplug/hpcsvc/%.c
750     $(COMPILE.c) -o $@ $<
751     $(CTFCONVERT_O)

753 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/pciex/hotplug/%.c
754     $(COMPILE.c) -o $@ $<
755     $(CTFCONVERT_O)

757 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/hotplug/pcihp/%.c
758     $(COMPILE.c) -o $@ $<
759     $(CTFCONVERT_O)

761 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/ib/clients/rds/%.c
762     $(COMPILE.c) -o $@ $<
763     $(CTFCONVERT_O)

765 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/ib/clients/rdsrv3/%.c
766     $(COMPILE.c) -o $@ $<
767     $(CTFCONVERT_O)

769 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/ib/clients/iser/%.c
770     $(COMPILE.c) -o $@ $<
771     $(CTFCONVERT_O)

773 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/ib/clients/ibd/%.c
774     $(COMPILE.c) -o $@ $<
775     $(CTFCONVERT_O)

777 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/ib/clients/eoib/%.c
778     $(COMPILE.c) -o $@ $<
779     $(CTFCONVERT_O)

781 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/ib/clients/of/sol_ofs/%.c
782     $(COMPILE.c) -o $@ $<
783     $(CTFCONVERT_O)

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

```

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

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

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

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

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

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

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

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

821 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/ib/ibt1/%.c
822     $(COMPILE.c) -o $@ $<
823     $(CTFCONVERT_O)

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

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

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

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

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

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

849 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/iwh/%.c
850     $(COMPILE.c) -o $@ $<
851     $(CTFCONVERT_O)

853 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/iwi/%.c
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

917 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/mac/%.c
918     $(COMPILE.c) -o $@ $<
919     $(CTFCONVERT_O)
```

```

921 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/mac/plugins/%.c
922     $(COMPILE.c) -o $@ $<
923     $(CTFCONVERT_O)

925 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/mega_sas/%.c
926     $(COMPILE.c) -o $@ $<
927     $(CTFCONVERT_O)

929 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/mii/%.c
930     $(COMPILE.c) -o $@ $<
931     $(CTFCONVERT_O)

933 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/mr_sas/%.c
934     $(COMPILE.c) -o $@ $<
935     $(CTFCONVERT_O)

937 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/scsi/adapters/mpt_sas/%.c
938     $(COMPILE.c) -o $@ $<
939     $(CTFCONVERT_O)

941 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/scsi/adapters/mpt_sas3/%.c
942     $(COMPILE.c) -o $@ $<
943     $(CTFCONVERT_O)

945 #endif /* ! codereview */
946 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/mxfe/%.c
947     $(COMPILE.c) -o $@ $<
948     $(CTFCONVERT_O)

950 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/mwl/%.c
951     $(COMPILE.c) -o $@ $<
952     $(CTFCONVERT_O)

954 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/mwl/mwl_fw/%.c
955     $(COMPILE.c) -o $@ $<
956     $(CTFCONVERT_O)

958 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/net80211/%.c
959     $(COMPILE.c) -o $@ $<
960     $(CTFCONVERT_O)

962 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/nge/%.c
963     $(COMPILE.c) -o $@ $<
964     $(CTFCONVERT_O)

966 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/nxge/%.c
967     $(COMPILE.c) -o $@ $<
968     $(CTFCONVERT_O)

970 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/nxge/npi/%.c
971     $(COMPILE.c) -o $@ $<
972     $(CTFCONVERT_O)

974 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/nxge/%.s
975     $(COMPILE.s) -o $@ $<

977 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/pci-ide/%.c
978     $(COMPILE.c) -o $@ $<
979     $(CTFCONVERT_O)

981 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/pcn/%.c
982     $(COMPILE.c) -o $@ $<
983     $(CTFCONVERT_O)

985 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/ppp/sppp/%.c

```

```

986     $(COMPILE.c) -o $@ $<
987     $(CTFCONVERT_O)

989 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/ppp/spppasyn/%.c
990     $(COMPILE.c) -o $@ $<
991     $(CTFCONVERT_O)

993 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/ppp/sppptun/%.c
994     $(COMPILE.c) -o $@ $<
995     $(CTFCONVERT_O)

997 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/ral/%.c
998     $(COMPILE.c) -o $@ $<
999     $(CTFCONVERT_O)

1001 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/rge/%.c
1002     $(COMPILE.c) -o $@ $<
1003     $(CTFCONVERT_O)

1005 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/rtls/%.c
1006     $(COMPILE.c) -o $@ $<
1007     $(CTFCONVERT_O)

1009 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/rsm/%.c
1010     $(COMPILE.c) -o $@ $<
1011     $(CTFCONVERT_O)

1013 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/rtw/%.c
1014     $(COMPILE.c) -o $@ $<
1015     $(CTFCONVERT_O)

1017 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/rum/%.c
1018     $(COMPILE.c) -o $@ $<
1019     $(CTFCONVERT_O)

1021 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/rwd/%.c
1022     $(COMPILE.c) -o $@ $<
1023     $(CTFCONVERT_O)

1025 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/rwn/%.c
1026     $(COMPILE.c) -o $@ $<
1027     $(CTFCONVERT_O)

1029 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/sata/adapters/ahci/%.c
1030     $(COMPILE.c) -o $@ $<
1031     $(CTFCONVERT_O)

1033 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/sata/adapters/nv_sata/%.c
1034     $(COMPILE.c) -o $@ $<
1035     $(CTFCONVERT_O)

1037 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/sata/adapters/si3124/%.c
1038     $(COMPILE.c) -o $@ $<
1039     $(CTFCONVERT_O)

1041 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/sata/impl/%.c
1042     $(COMPILE.c) -o $@ $<
1043     $(CTFCONVERT_O)

1045 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/scsi/conf/%.c
1046     $(COMPILE.c) -o $@ $<
1047     $(CTFCONVERT_O)

1049 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/scsi/impl/%.c
1050     $(COMPILE.c) -o $@ $<
1051     $(CTFCONVERT_O)

```

```

1053 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/scsi/targets/%.c
1054     $(COMPILE.c) -o $@ $<
1055     $(CTFCONVERT_O)

1057 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/scsi/adapters/%.c
1058     $(COMPILE.c) -o $@ $<
1059     $(CTFCONVERT_O)

1061 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/scsi/adapters/blk2scsa/%.c
1062     $(COMPILE.c) -o $@ $<
1063     $(CTFCONVERT_O)

1065 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/scsi/adapters/scsi_vhci/%.c
1066     $(COMPILE.c) -o $@ $<
1067     $(CTFCONVERT_O)

1069 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/scsi/adapters/scsi_vhci/fop
1070     $(COMPILE.c) -o $@ $<
1071     $(CTFCONVERT_O)

1073 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/fibre-channel/ulp/%.c
1074     $(COMPILE.c) -o $@ $<
1075     $(CTFCONVERT_O)

1077 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/fibre-channel/impl/%.c
1078     $(COMPILE.c) -o $@ $<
1079     $(CTFCONVERT_O)

1081 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/fibre-channel/fca/qlc/%.c
1082     $(COMPILE.c) -o $@ $<
1083     $(CTFCONVERT_O)

1085 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/fibre-channel/fca/qlge/%.c
1086     $(COMPILE.c) -o $@ $<
1087     $(CTFCONVERT_O)

1089 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/fibre-channel/fca/emlxs/%.c
1090     $(COMPILE.c) -o $@ $<
1091     $(CTFCONVERT_O)

1093 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/fibre-channel/fca/oce/%.c
1094     $(COMPILE.c) -o $@ $<
1095     $(CTFCONVERT_O)

1097 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/fibre-channel/fca/fcoe/%.c
1098     $(COMPILE.c) -o $@ $<
1099     $(CTFCONVERT_O)

1101 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/sdcard/adapters/sdhost/%.c
1102     $(COMPILE.c) -o $@ $<
1103     $(CTFCONVERT_O)

1105 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/sdcard/impl/%.c
1106     $(COMPILE.c) -o $@ $<
1107     $(CTFCONVERT_O)

1109 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/sdcard/targets/sdcard/%.c
1110     $(COMPILE.c) -o $@ $<
1111     $(CTFCONVERT_O)

1113 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/sfe/%.c
1114     $(COMPILE.c) -o $@ $<
1115     $(CTFCONVERT_O)

1117 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/simnet/%.c

```

```

1118     $(COMPILE.c) -o $@ $<
1119     $(CTFCONVERT_O)

1121 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/softmac/%.c
1122     $(COMPILE.c) -o $@ $<
1123     $(CTFCONVERT_O)

1125 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/uath/%.c
1126     $(COMPILE.c) -o $@ $<
1127     $(CTFCONVERT_O)

1129 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/uath/uath_fw/%.c
1130     $(COMPILE.c) -o $@ $<
1131     $(CTFCONVERT_O)

1133 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/ural/%.c
1134     $(COMPILE.c) -o $@ $<
1135     $(CTFCONVERT_O)

1137 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/urtw/%.c
1138     $(COMPILE.c) -o $@ $<
1139     $(CTFCONVERT_O)

1141 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/clients/audio/usb_ac/%.
1142     $(COMPILE.c) -o $@ $<
1143     $(CTFCONVERT_O)

1145 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/clients/audio/usb_as/%.
1146     $(COMPILE.c) -o $@ $<
1147     $(CTFCONVERT_O)

1149 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/clients/audio/usb_ah/%.
1150     $(COMPILE.c) -o $@ $<
1151     $(CTFCONVERT_O)

1153 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/clients/usbskel/%.c
1154     $(COMPILE.c) -o $@ $<
1155     $(CTFCONVERT_O)

1157 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/clients/video/usbvc/%.c
1158     $(COMPILE.c) -o $@ $<
1159     $(CTFCONVERT_O)

1161 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/clients/hwarc/%.c
1162     $(COMPILE.c) -o $@ $<
1163     $(CTFCONVERT_O)

1165 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/clients/hid/%.c
1166     $(COMPILE.c) -o $@ $<
1167     $(CTFCONVERT_O)

1169 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/clients/hidparser/%.c
1170     $(COMPILE.c) -o $@ $<
1171     $(CTFCONVERT_O)

1173 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/clients/printer/%.c
1174     $(COMPILE.c) -o $@ $<
1175     $(CTFCONVERT_O)

1177 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/clients/usbbkm/%.c
1178     $(COMPILE.c) -o $@ $<
1179     $(CTFCONVERT_O)

1181 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/clients/uslbs/%.c
1182     $(COMPILE.c) -o $@ $<
1183     $(CTFCONVERT_O)

```

```

1185 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/clients/usbinput/usbwcm
1186 $(COMPILE.c) -o $@ $<
1187 $(CTFCONVERT_O)

1189 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/clients/ugen/%.c
1190 $(COMPILE.c) -o $@ $<
1191 $(CTFCONVERT_O)

1193 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/clients/usbser/%.c
1194 $(COMPILE.c) -o $@ $<
1195 $(CTFCONVERT_O)

1197 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/clients/usbser/usbsacm/
1198 $(COMPILE.c) -o $@ $<
1199 $(CTFCONVERT_O)

1201 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/clients/usbser/usbftdi/
1202 $(COMPILE.c) -o $@ $<
1203 $(CTFCONVERT_O)

1205 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/clients/usbser/usbser_k
1206 $(COMPILE.c) -o $@ $<
1207 $(CTFCONVERT_O)

1209 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/clients/usbser/usbsprl/
1210 $(COMPILE.c) -o $@ $<
1211 $(CTFCONVERT_O)

1213 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/clients/wusb_df/%.c
1214 $(COMPILE.c) -o $@ $<
1215 $(CTFCONVERT_O)

1217 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/clients/hwa1480_fw/%.c
1218 $(COMPILE.c) -o $@ $<
1219 $(CTFCONVERT_O)

1221 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/clients/wusb_ca/%.c
1222 $(COMPILE.c) -o $@ $<
1223 $(CTFCONVERT_O)

1225 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/clients/usbecm/%.c
1226 $(COMPILE.c) -o $@ $<
1227 $(CTFCONVERT_O)

1229 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/hcd/openhci/%.c
1230 $(COMPILE.c) -o $@ $<
1231 $(CTFCONVERT_O)

1233 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/hcd/ehci/%.c
1234 $(COMPILE.c) -o $@ $<
1235 $(CTFCONVERT_O)

1237 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/hcd/uhci/%.c
1238 $(COMPILE.c) -I../common -o $@ $<
1239 $(CTFCONVERT_O)

1241 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/hubd/%.c
1242 $(COMPILE.c) -o $@ $<
1243 $(CTFCONVERT_O)

1245 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/scsa2usb/%.c
1246 $(COMPILE.c) -o $@ $<
1247 $(CTFCONVERT_O)

1249 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/usb_mid/%.c

```

```

1250 $(COMPILE.c) -o $@ $<
1251 $(CTFCONVERT_O)

1253 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/usb_ia/%.c
1254 $(COMPILE.c) -o $@ $<
1255 $(CTFCONVERT_O)

1257 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/usba/%.c
1258 $(COMPILE.c) -o $@ $<
1259 $(CTFCONVERT_O)

1261 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/usba10/%.c
1262 $(COMPILE.c) -o $@ $<
1263 $(CTFCONVERT_O)

1265 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/hwa/hwahc/%.c
1266 $(COMPILE.c) -o $@ $<
1267 $(CTFCONVERT_O)

1269 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/uwb/uwba/%.c
1270 $(COMPILE.c) -o $@ $<
1271 $(CTFCONVERT_O)

1273 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/vuidnice/%.c
1274 $(COMPILE.c) -o $@ $<
1275 $(CTFCONVERT_O)

1277 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/vnic/%.c
1278 $(COMPILE.c) -o $@ $<
1279 $(CTFCONVERT_O)

1281 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/wpi/%.c
1282 $(COMPILE.c) -o $@ $<
1283 $(CTFCONVERT_O)

1285 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/zyd/%.c
1286 $(COMPILE.c) -o $@ $<
1287 $(CTFCONVERT_O)

1289 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/chxge/com/%.c
1290 $(COMPILE.c) -o $@ $<
1291 $(CTFCONVERT_O)

1293 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/chxge/%.c
1294 $(COMPILE.c) -o $@ $<
1295 $(CTFCONVERT_O)

1297 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/cxgbe/common/%.c
1298 $(COMPILE.c) -o $@ $<
1299 $(CTFCONVERT_O)

1301 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/cxgbe/shared/%.c
1302 $(COMPILE.c) -o $@ $<
1303 $(CTFCONVERT_O)

1305 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/cxgbe/firmware/%.c
1306 $(COMPILE.c) -o $@ $<
1307 $(CTFCONVERT_O)

1309 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/cxgbe/t4nex/%.c
1310 $(COMPILE.c) -o $@ $<
1311 $(CTFCONVERT_O)

1313 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/cxgbe/cxgbe/%.c
1314 $(COMPILE.c) -o $@ $<
1315 $(CTFCONVERT_O)

```

```

1317 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/ixgb/%.c
1318 $(COMPILE.c) -o $@ $<
1319 $(CTFCONVERT_O)

1321 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/xge/drv/%.c
1322 $(COMPILE.c) -o $@ $<
1323 $(CTFCONVERT_O)

1325 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/xge/hal/xgehal/%.c
1326 $(COMPILE.c) -o $@ $<
1327 $(CTFCONVERT_O)

1329 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/e1000api/%.c
1330 $(COMPILE.c) -o $@ $<
1331 $(CTFCONVERT_O)

1333 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/e1000g/%.c
1334 $(COMPILE.c) -o $@ $<
1335 $(CTFCONVERT_O)

1337 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/igb/%.c
1338 $(COMPILE.c) -o $@ $<
1339 $(CTFCONVERT_O)

1341 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/iprb/%.c
1342 $(COMPILE.c) -o $@ $<
1343 $(CTFCONVERT_O)

1345 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/ixgbe/%.c
1346 $(COMPILE.c) -o $@ $<
1347 $(CTFCONVERT_O)

1349 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/ntxn/%.c
1350 $(COMPILE.c) -o $@ $<
1351 $(CTFCONVERT_O)

1353 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/myril0ge/drv/%.c
1354 $(COMPILE.c) -o $@ $<
1355 $(CTFCONVERT_O)

1357 $(OBJSDIR)/%.o: $(UTSBASE)/common/ipp/%.c
1358 $(COMPILE.c) -o $@ $<
1359 $(CTFCONVERT_O)

1361 $(OBJSDIR)/%.o: $(UTSBASE)/common/ipp/ipgpc/%.c
1362 $(COMPILE.c) -o $@ $<
1363 $(CTFCONVERT_O)

1365 $(OBJSDIR)/%.o: $(UTSBASE)/common/ipp/dlcosmk/%.c
1366 $(COMPILE.c) -o $@ $<
1367 $(CTFCONVERT_O)

1369 $(OBJSDIR)/%.o: $(UTSBASE)/common/ipp/flowacct/%.c
1370 $(COMPILE.c) -o $@ $<
1371 $(CTFCONVERT_O)

1373 $(OBJSDIR)/%.o: $(UTSBASE)/common/ipp/dscpmk/%.c
1374 $(COMPILE.c) -o $@ $<
1375 $(CTFCONVERT_O)

1377 $(OBJSDIR)/%.o: $(UTSBASE)/common/ipp/meters/%.c
1378 $(COMPILE.c) -o $@ $<
1379 $(CTFCONVERT_O)

1381 $(OBJSDIR)/%.o: $(UTSBASE)/common/kiconv/kiconv_emea/%.c

```

```

1382 $(COMPILE.c) -o $@ $<
1383 $(CTFCONVERT_O)

1385 $(OBJSDIR)/%.o: $(UTSBASE)/common/kiconv/kiconv_ja/%.c
1386 $(COMPILE.c) -o $@ $<
1387 $(CTFCONVERT_O)

1389 $(OBJSDIR)/%.o: $(UTSBASE)/common/kiconv/kiconv_ko/%.c
1390 $(COMPILE.c) -o $@ $<
1391 $(CTFCONVERT_O)

1393 $(OBJSDIR)/%.o: $(UTSBASE)/common/kiconv/kiconv_sc/%.c
1394 $(COMPILE.c) -o $@ $<
1395 $(CTFCONVERT_O)

1397 $(OBJSDIR)/%.o: $(UTSBASE)/common/kiconv/kiconv_tc/%.c
1398 $(COMPILE.c) -o $@ $<
1399 $(CTFCONVERT_O)

1401 $(OBJSDIR)/%.o: $(UTSBASE)/common/klm/%.c
1402 $(COMPILE.c) -o $@ $<
1403 $(CTFCONVERT_O)

1405 $(OBJSDIR)/%.o: $(UTSBASE)/common/kmdb/%.c
1406 $(COMPILE.c) -o $@ $<
1407 $(CTFCONVERT_O)

1409 $(OBJSDIR)/%.o: $(UTSBASE)/common/ktli/%.c
1410 $(COMPILE.c) -o $@ $<
1411 $(CTFCONVERT_O)

1413 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/scsi/adapters/iscsi/%.c
1414 $(COMPILE.c) -o $@ $<
1415 $(CTFCONVERT_O)

1417 $(OBJSDIR)/%.o: $(COMMONBASE)/iscsi/%.c
1418 $(COMPILE.c) -o $@ $<
1419 $(CTFCONVERT_O)

1421 $(OBJSDIR)/%.o: $(UTSBASE)/common/inet/kifconf/%.c
1422 $(COMPILE.c) -o $@ $<
1423 $(CTFCONVERT_O)

1425 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/vr/%.c
1426 $(COMPILE.c) -o $@ $<
1427 $(CTFCONVERT_O)

1429 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/yge/%.c
1430 $(COMPILE.c) -o $@ $<
1431 $(CTFCONVERT_O)

1433 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/virtio/%.c
1434 $(COMPILE.c) -o $@ $<
1435 $(CTFCONVERT_O)

1437 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/vioblk/%.c
1438 $(COMPILE.c) -o $@ $<
1439 $(CTFCONVERT_O)

1441 #
1442 # krtld must refer to its own bzero/bcopy until the kernel is fully linked
1443 #
1444 $(OBJSDIR)/bootrd.o := CPPFLAGS += -DKOBJ_OVERRIDES
1445 $(OBJSDIR)/doreloc.o := CPPFLAGS += -DKOBJ_OVERRIDES
1446 $(OBJSDIR)/kobj.o := CPPFLAGS += -DKOBJ_OVERRIDES
1447 $(OBJSDIR)/kobj_boot.o := CPPFLAGS += -DKOBJ_OVERRIDES

```

```

1448 $(OBJSDIR)/kobj_bootflags.o := CPPFLAGS += -DKOBJ_OVERRIDES
1449 $(OBJSDIR)/kobj_convrelstr.o := CPPFLAGS += -DKOBJ_OVERRIDES
1450 $(OBJSDIR)/kobj_isa.o := CPPFLAGS += -DKOBJ_OVERRIDES
1451 $(OBJSDIR)/kobj_kdi.o := CPPFLAGS += -DKOBJ_OVERRIDES
1452 $(OBJSDIR)/kobj_lm.o := CPPFLAGS += -DKOBJ_OVERRIDES
1453 $(OBJSDIR)/kobj_reloc.o := CPPFLAGS += -DKOBJ_OVERRIDES
1454 $(OBJSDIR)/kobj_stubs.o := CPPFLAGS += -DKOBJ_OVERRIDES
1455 $(OBJSDIR)/kobj_subr.o := CPPFLAGS += -DKOBJ_OVERRIDES

1457 $(OBJSDIR)/%.o: $(UTSBASE)/common/krtld/%.c
1458 $(COMPILE.c) -o $@ $<
1459 $(CTFCONVERT_O)

1461 $(OBJSDIR)/%.o: $(COMMONBASE)/list/%.c
1462 $(COMPILE.c) -o $@ $<
1463 $(CTFCONVERT_O)

1465 $(OBJSDIR)/%.o: $(COMMONBASE)/lvm/%.c
1466 $(COMPILE.c) -o $@ $<
1467 $(CTFCONVERT_O)

1469 $(OBJSDIR)/%.o: $(COMMONBASE)/lzma/%.c
1470 $(COMPILE.c) -o $@ $<
1471 $(CTFCONVERT_O)

1473 $(OBJSDIR)/%.o: $(COMMONBASE)/crypto/md4/%.c
1474 $(COMPILE.c) -o $@ $<
1475 $(CTFCONVERT_O)

1477 $(OBJSDIR)/%.o: $(COMMONBASE)/crypto/md5/%.c
1478 $(COMPILE.c) -o $@ $<
1479 $(CTFCONVERT_O)

1481 $(OBJSDIR)/%.o: $(COMMONBASE)/net/dhcp/%.c
1482 $(COMPILE.c) -o $@ $<
1483 $(CTFCONVERT_O)

1485 $(OBJSDIR)/%.o: $(COMMONBASE)/nvpair/%.c
1486 $(COMPILE.c) -o $@ $<
1487 $(CTFCONVERT_O)

1489 $(OBJSDIR)/%.o: $(UTSBASE)/common/os/%.c
1490 $(COMPILE.c) -o $@ $<
1491 $(CTFCONVERT_O)

1493 $(OBJSDIR)/%.o: $(UTSBASE)/common/pcmcia/cis/%.c
1494 $(COMPILE.c) -o $@ $<
1495 $(CTFCONVERT_O)

1497 $(OBJSDIR)/%.o: $(UTSBASE)/common/pcmcia/cs/%.c
1498 $(COMPILE.c) -o $@ $<
1499 $(CTFCONVERT_O)

1501 $(OBJSDIR)/%.o: $(UTSBASE)/common/pcmcia/nexus/%.c
1502 $(COMPILE.c) -o $@ $<
1503 $(CTFCONVERT_O)

1505 $(OBJSDIR)/%.o: $(UTSBASE)/common/pcmcia/pcs/%.c
1506 $(COMPILE.c) -o $@ $<
1507 $(CTFCONVERT_O)

1509 $(OBJSDIR)/%.o: $(UTSBASE)/common/rpc/%.c
1510 $(COMPILE.c) -o $@ $<
1511 $(CTFCONVERT_O)

1513 $(OBJSDIR)/%.o: $(UTSBASE)/common/rpc/sec/%.c

```

```

1514 $(COMPILE.c) -o $@ $<
1515 $(CTFCONVERT_O)

1517 $(OBJSDIR)/%.o: $(UTSBASE)/common/rpc/sec_gss/%.c
1518 $(COMPILE.c) -o $@ $<
1519 $(CTFCONVERT_O)

1521 $(OBJSDIR)/%.o: $(COMMONBASE)/crypto/sha1/%.c
1522 $(COMPILE.c) -o $@ $<
1523 $(CTFCONVERT_O)

1525 $(OBJSDIR)/%.o: $(COMMONBASE)/crypto/sha2/%.c
1526 $(COMPILE.c) -o $@ $<
1527 $(CTFCONVERT_O)

1529 $(OBJSDIR)/%.o: $(UTSBASE)/common/syscall/%.c
1530 $(COMPILE.c) -o $@ $<
1531 $(CTFCONVERT_O)

1533 $(OBJSDIR)/%.o: $(UTSBASE)/common/tnf/%.c
1534 $(COMPILE.c) -o $@ $<
1535 $(CTFCONVERT_O)

1537 $(OBJSDIR)/%.o: $(COMMONBASE)/tsol/%.c
1538 $(COMPILE.c) -o $@ $<
1539 $(CTFCONVERT_O)

1541 $(OBJSDIR)/%.o: $(COMMONBASE)/util/%.c
1542 $(COMPILE.c) -o $@ $<
1543 $(CTFCONVERT_O)

1545 $(OBJSDIR)/%.o: $(COMMONBASE)/unicode/%.c
1546 $(COMPILE.c) -o $@ $<
1547 $(CTFCONVERT_O)

1549 $(OBJSDIR)/%.o: $(UTSBASE)/common/vm/%.c
1550 $(COMPILE.c) -o $@ $<
1551 $(CTFCONVERT_O)

1553 $(OBJSDIR)/%.o: $(UTSBASE)/common/zmod/%.c
1554 $(COMPILE.c) -o $@ $<
1555 $(CTFCONVERT_O)

1557 $(OBJSDIR)/zlib_obj.o: $(ZLIB_OBJS:%=$(OBJSDIR)/%)
1558 $(LD) -r -Breduce -M$(UTSBASE)/common/zmod/mapfile -o $@ \
1559 $(ZLIB_OBJS:%=$(OBJSDIR)/%)
1560 $(CTFMERGE) -t -f -L VERSION -o $@ $(ZLIB_OBJS:%=$(OBJSDIR)/%)

1562 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/hxge/%.c
1563 $(COMPILE.c) -o $@ $<
1564 $(CTFCONVERT_O)

1566 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/tpm/%.c
1567 $(COMPILE.c) -o $@ $<
1568 $(CTFCONVERT_O)

1570 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/tpm/%.s
1571 $(COMPILE.s) -o $@ $<

1573 $(OBJSDIR)/bz2%.o: $(COMMONBASE)/bzip2/%.c
1574 $(COMPILE.c) -o $@ -I$(COMMONBASE)/bzip2 $<
1575 $(CTFCONVERT_O)

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

1579 $(LINTSDIR)/bz2%.ln: $(COMMONBASE)/bzip2/%.c

```

```

1580      @$(LHEAD) $(LINT.c) -C $(LINTS_DIR)/`basename $@ .ln` $(BZ2LINT) $< $(
1582 #
1583 #      Section lb:      Lint 'objects'
1584 #
1585 $(LINTS_DIR)/%.ln:      $(COMMONBASE)/crypto/aes/%.c
1586      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1588 $(LINTS_DIR)/%.ln:      $(COMMONBASE)/crypto/arcfour/%.c
1589      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1591 $(LINTS_DIR)/%.ln:      $(COMMONBASE)/crypto/blowfish/%.c
1592      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1594 $(LINTS_DIR)/%.ln:      $(COMMONBASE)/crypto/ecc/%.c
1595      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1597 $(LINTS_DIR)/%.ln:      $(COMMONBASE)/crypto/modes/%.c
1598      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1600 $(LINTS_DIR)/%.ln:      $(COMMONBASE)/crypto/padding/%.c
1601      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1603 $(LINTS_DIR)/%.ln:      $(COMMONBASE)/crypto/rng/%.c
1604      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1606 $(LINTS_DIR)/%.ln:      $(COMMONBASE)/crypto/rsa/%.c
1607      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1609 $(LINTS_DIR)/%.ln:      $(COMMONBASE)/bignum/%.c
1610      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1612 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/bignum/%.c
1613      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1615 $(LINTS_DIR)/%.ln:      $(COMMONBASE)/mpi/%.c
1616      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1618 $(LINTS_DIR)/%.ln:      $(COMMONBASE)/acl/%.c
1619      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1621 $(LINTS_DIR)/%.ln:      $(COMMONBASE)/avl/%.c
1622      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1624 $(LINTS_DIR)/%.ln:      $(COMMONBASE)/ucode/%.c
1625      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1627 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/brand/sn1/%.c
1628      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1630 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/brand/solaris10/%.c
1631      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1633 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/c2/%.c
1634      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1636 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/conf/%.c
1637      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1639 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/contract/%.c
1640      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1642 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/cpr/%.c
1643      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1645 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/ctf/%.c

```

```

1646      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1648 $(LINTS_DIR)/%.ln:      $(COMMONBASE)/ctf/%.c
1649      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1651 $(LINTS_DIR)/%.ln:      $(COMMONBASE)/pci/%.c
1652      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1654 $(LINTS_DIR)/%.ln:      $(COMMONBASE)/devide/%.c
1655      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1657 $(LINTS_DIR)/%.ln:      $(COMMONBASE)/crypto/des/%.c
1658      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1660 $(LINTS_DIR)/%.ln:      $(COMMONBASE)/smbios/%.c
1661      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1663 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/avs/ncall/%.c
1664      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1666 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/avs/ns/dsw/%.c
1667      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1669 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/avs/ns/nsctl/%.c
1670      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1672 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/avs/ns/rdc/%.c
1673      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1675 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/avs/ns/sdbc/%.c
1676      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1678 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/avs/ns/solaris/%.c
1679      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1681 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/avs/ns/sv/%.c
1682      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1684 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/avs/ns/unistat/%.c
1685      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1687 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/des/%.c
1688      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1690 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/crypto/api/%.c
1691      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1693 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/crypto/core/%.c
1694      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1696 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/crypto/io/%.c
1697      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1699 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/crypto/spi/%.c
1700      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1702 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/disp/%.c
1703      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1705 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/dtrace/%.c
1706      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1708 $(LINTS_DIR)/%.ln:      $(COMMONBASE)/exacct/%.c
1709      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1711 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/exec/aout/%.c

```



```

1712      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1714 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/exec/elf/%.c
1715      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1717 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/exec/intp/%.c
1718      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1720 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/exec/shbin/%.c
1721      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1723 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/exec/java/%.c
1724      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1726 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/%.c
1727      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1729 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/autofs/%.c
1730      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1732 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/cacheufs/%.c
1733      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1735 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/ctfs/%.c
1736      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1738 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/doorfs/%.c
1739      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1741 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/dcfis/%.c
1742      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1744 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/devfs/%.c
1745      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1747 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/dev/%.c
1748      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1750 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/fd/%.c
1751      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1753 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/fifofs/%.c
1754      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1756 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/hsfs/%.c
1757      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1759 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/lofs/%.c
1760      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1762 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/mntfs/%.c
1763      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1765 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/namefs/%.c
1766      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1768 $(LINTS_DIR)/%.ln:      $(COMMONBASE)/smbsrv/%.c
1769      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1771 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/smbsrv/%.c
1772      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1774 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/nfs/%.c
1775      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1777 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/objfs/%.c

```

```

1778      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1780 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/pcfs/%.c
1781      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1783 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/portfs/%.c
1784      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1786 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/proc/%.c
1787      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1789 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/sharefs/%.c
1790      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1792 $(LINTS_DIR)/%.ln:      $(COMMONBASE)/smbclnt/%.c
1793      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1795 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/smbclnt/net smb/%.c
1796      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1798 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/smbclnt/smbfs/%.c
1799      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1801 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/sockfs/%.c
1802      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1804 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/specfs/%.c
1805      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1807 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/swapfs/%.c
1808      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1810 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/tmpfs/%.c
1811      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1813 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/udfs/%.c
1814      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1816 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/ufs/%.c
1817      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1819 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/ufs_log/%.c
1820      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1822 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/vscan/%.c
1823      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1825 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/zfs/%.c
1826      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1828 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/zut/%.c
1829      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1831 $(LINTS_DIR)/%.ln:      $(COMMONBASE)/xattr/%.c
1832      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1834 $(LINTS_DIR)/%.ln:      $(COMMONBASE)/zfs/%.c
1835      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1837 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/gssapi/%.c
1838      @$(LHEAD) $(LINT.c) $(KGSSD_FLAGS) $< $(LTAIL))
1840 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/gssapi/mechs/dummy/%.c
1841      @$(LHEAD) $(LINT.c) $(KGSSD_FLAGS) $< $(LTAIL))
1843 $(LINTS_DIR)/%.ln:      $(KMECHKRB5_BASE)/%.c

```

```

1844      @$(LHEAD) $(LINT.c) $(KGSSD_FLAGS) $< $(LTAIL))
1846 $(LINTS_DIR)/%.ln:      $(KMECHKRB5_BASE)/crypto/%.c
1847   @$(LHEAD) $(LINT.c) $(KGSSD_FLAGS) $< $(LTAIL))
1849 $(LINTS_DIR)/%.ln:      $(KMECHKRB5_BASE)/crypto/des/%.c
1850   @$(LHEAD) $(LINT.c) $(KGSSD_FLAGS) $< $(LTAIL))
1852 $(LINTS_DIR)/%.ln:      $(KMECHKRB5_BASE)/crypto/dk/%.c
1853   @$(LHEAD) $(LINT.c) $(KGSSD_FLAGS) $< $(LTAIL))
1855 $(LINTS_DIR)/%.ln:      $(KMECHKRB5_BASE)/crypto/os/%.c
1856   @$(LHEAD) $(LINT.c) $(KGSSD_FLAGS) $< $(LTAIL))
1858 $(LINTS_DIR)/%.ln:      $(KMECHKRB5_BASE)/crypto/arcfour/%.c
1859   @$(LHEAD) $(LINT.c) $(KGSSD_FLAGS) $< $(LTAIL))
1861 $(LINTS_DIR)/%.ln:      $(KMECHKRB5_BASE)/crypto/enc_provider/%.c
1862   @$(LHEAD) $(LINT.c) $(KGSSD_FLAGS) $< $(LTAIL))
1864 $(LINTS_DIR)/%.ln:      $(KMECHKRB5_BASE)/crypto/hash_provider/%.c
1865   @$(LHEAD) $(LINT.c) $(KGSSD_FLAGS) $< $(LTAIL))
1867 $(LINTS_DIR)/%.ln:      $(KMECHKRB5_BASE)/crypto/keyhash_provider/%.c
1868   @$(LHEAD) $(LINT.c) $(KGSSD_FLAGS) $< $(LTAIL))
1870 $(LINTS_DIR)/%.ln:      $(KMECHKRB5_BASE)/crypto/raw/%.c
1871   @$(LHEAD) $(LINT.c) $(KGSSD_FLAGS) $< $(LTAIL))
1873 $(LINTS_DIR)/%.ln:      $(KMECHKRB5_BASE)/crypto/old/%.c
1874   @$(LHEAD) $(LINT.c) $(KGSSD_FLAGS) $< $(LTAIL))
1876 $(LINTS_DIR)/%.ln:      $(KMECHKRB5_BASE)/krb5/krb/%.c
1877   @$(LHEAD) $(LINT.c) $(KGSSD_FLAGS) $< $(LTAIL))
1879 $(LINTS_DIR)/%.ln:      $(KMECHKRB5_BASE)/krb5/os/%.c
1880   @$(LHEAD) $(LINT.c) $(KGSSD_FLAGS) $< $(LTAIL))
1882 $(LINTS_DIR)/%.ln:      $(KMECHKRB5_BASE)/mech/%.c
1883   @$(LHEAD) $(LINT.c) $(KGSSD_FLAGS) $< $(LTAIL))
1885 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/idmap/%.c
1886   @$(LHEAD) $(LINT.c) $< $(LTAIL))
1888 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/inet/%.c
1889   @$(LHEAD) $(LINT.c) $< $(LTAIL))
1891 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/inet/sockmods/%.c
1892   @$(LHEAD) $(LINT.c) $< $(LTAIL))
1894 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/inet/arp/%.c
1895   @$(LHEAD) $(LINT.c) $< $(LTAIL))
1897 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/inet/ip/%.c
1898   @$(LHEAD) $(LINT.c) $< $(LTAIL))
1900 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/inet/ipnet/%.c
1901   @$(LHEAD) $(LINT.c) $< $(LTAIL))
1903 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/inet/iptun/%.c
1904   @$(LHEAD) $(LINT.c) $< $(LTAIL))
1906 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/inet/ipd/%.c
1907   @$(LHEAD) $(LINT.c) $< $(LTAIL))
1909 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/inet/ipf/%.c

```

```

1910      @$(LHEAD) $(LINT.c) $(IPFFLAGS) $< $(LTAIL))
1912 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/inet/kssl/%.c
1913   @$(LHEAD) $(LINT.c) $< $(LTAIL))
1915 $(LINTS_DIR)/%.ln:      $(COMMONBASE)/net/patricia/%.c
1916   @$(LHEAD) $(LINT.c) $(IPFFLAGS) $< $(LTAIL))
1918 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/inet/udp/%.c
1919   @$(LHEAD) $(LINT.c) $< $(LTAIL))
1921 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/inet/sctp/%.c
1922   @$(LHEAD) $(LINT.c) $< $(LTAIL))
1924 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/inet/tcp/%.c
1925   @$(LHEAD) $(LINT.c) $< $(LTAIL))
1927 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/inet/ilb/%.c
1928   @$(LHEAD) $(LINT.c) $< $(LTAIL))
1930 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/inet/nca/%.c
1931   @$(LHEAD) $(LINT.c) $< $(LTAIL))
1933 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/inet/dlpistub/%.c
1934   @$(LHEAD) $(LINT.c) $< $(LTAIL))
1936 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/%.c
1937   @$(LHEAD) $(LINT.c) $< $(LTAIL))
1939 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/1394/%.c
1940   @$(LHEAD) $(LINT.c) $< $(LTAIL))
1942 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/1394/adapters/%.c
1943   @$(LHEAD) $(LINT.c) $< $(LTAIL))
1945 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/1394/targets/av1394/%.c
1946   @$(LHEAD) $(LINT.c) $< $(LTAIL))
1948 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/1394/targets/dcam1394/%.c
1949   @$(LHEAD) $(LINT.c) $< $(LTAIL))
1951 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/1394/targets/scsa1394/%.c
1952   @$(LHEAD) $(LINT.c) $< $(LTAIL))
1954 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/sbp2/%.c
1955   @$(LHEAD) $(LINT.c) $< $(LTAIL))
1957 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/aac/%.c
1958   @$(LHEAD) $(LINT.c) $< $(LTAIL))
1960 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/afe/%.c
1961   @$(LHEAD) $(LINT.c) $< $(LTAIL))
1963 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/atge/%.c
1964   @$(LHEAD) $(LINT.c) $< $(LTAIL))
1966 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/arn/%.c
1967   @$(LHEAD) $(LINT.c) $< $(LTAIL))
1969 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/ath/%.c
1970   @$(LHEAD) $(LINT.c) $< $(LTAIL))
1972 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/atu/%.c
1973   @$(LHEAD) $(LINT.c) $< $(LTAIL))
1975 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/audio/impl/%.c

```

```
1976      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1978 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/audio/ac97/%.c
1979      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1981 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/audio/drv/audio1575/%.c
1982      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1984 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/audio/drv/audio810/%.c
1985      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1987 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/audio/drv/audiocmi/%.c
1988      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1990 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/audio/drv/audiocmihd/%.c
1991      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1993 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/audio/drv/audioens/%.c
1994      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1996 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/audio/drv/audioemu10k/%.c
1997      @$(LHEAD) $(LINT.c) $< $(LTAIL))
1999 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/audio/drv/audiohd/%.c
2000      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2002 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/audio/drv/audioixp/%.c
2003      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2005 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/audio/drv/audiols/%.c
2006      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2008 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/audio/drv/audiopci/%.c
2009      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2011 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/audio/drv/audiopl6x/%.c
2012      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2014 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/audio/drv/audiosolo/%.c
2015      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2017 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/audio/drv/audiots/%.c
2018      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2020 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/audio/drv/udiovia823x/%.c
2021      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2023 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/audio/drv/udiovia97/%.c
2024      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2026 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/bfe/%.c
2027      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2029 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/bpf/%.c
2030      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2032 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/bge/%.c
2033      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2035 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/blkdev/%.c
2036      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2038 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/cardbus/%.c
2039      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2041 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/comstar/lu/stmf_sbd/%.c
```

```
2042      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2044 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/comstar/port/fct/%.c
2045      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2047 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/comstar/port/qlt/%.c
2048      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2050 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/comstar/port/srpt/%.c
2051      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2053 $(LINTS_DIR)/%.ln:      $(COMMONBASE)/iscsit/%.c
2054      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2056 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/comstar/port/fcoet/%.c
2057      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2059 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/comstar/port/iscsit/%.c
2060      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2062 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/comstar/port/pppt/%.c
2063      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2065 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/comstar/stmf/%.c
2066      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2068 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/cpqary3/%.c
2069      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2071 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/dld/%.c
2072      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2074 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/dls/%.c
2075      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2077 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/dmfe/%.c
2078      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2080 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/drm/%.c
2081      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2083 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/efe/%.c
2084      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2086 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/elxl/%.c
2087      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2089 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/fcoe/%.c
2090      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2092 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/hme/%.c
2093      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2095 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/pciex/%.c
2096      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2098 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/hotplug/hpcsvc/%.c
2099      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2101 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/pciex/hotplug/%.c
2102      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2104 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/hotplug/pcihp/%.c
2105      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2107 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/ib/clients/rds/%.c
```

```

2108      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2110 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/ib/clients/rds3/%.c
2111      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2113 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/ib/clients/iser/%.c
2114      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2116 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/ib/clients/ibd/%.c
2117      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2119 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/ib/clients/eoib/%.c
2120      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2122 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/ib/clients/of/sol_ofs/%.c
2123      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2125 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/ib/clients/of/sol_ucma/%.c
2126      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2128 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/ib/clients/of/sol_umad/%.c
2129      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2131 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/ib/clients/of/sol_uverbs/%.
2132      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2134 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/ib/clients/sdp/%.c
2135      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2137 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/ib/mgt/ibcm/%.c
2138      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2140 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/ib/mgt/ibdm/%.c
2141      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2143 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/ib/mgt/ibdma/%.c
2144      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2146 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/ib/mgt/ibmf/%.c
2147      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2149 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/ib/ibnex/%.c
2150      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2152 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/ib/ibt1/%.c
2153      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2155 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/ib/adapters/tavor/%.c
2156      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2158 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/ib/adapters/hermon/%.c
2159      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2161 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/ib/clients/daplt/%.c
2162      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2164 $(LINTS_DIR)/%.ln:      $(COMMONBASE)/iscsi/%.c
2165      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2167 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/idm/%.c
2168      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2170 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/ipw/%.c
2171      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2173 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/iwh/%.c

```

```

2174      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2176 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/iwi/%.c
2177      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2179 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/iwk/%.c
2180      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2182 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/iwp/%.c
2183      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2185 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/kb8042/%.c
2186      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2188 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/kbtrans/%.c
2189      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2191 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/ksocket/%.c
2192      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2194 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/aggr/%.c
2195      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2197 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/lp/%.c
2198      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2200 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/lvm/hotspares/%.c
2201      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2203 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/lvm/md/%.c
2204      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2206 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/lvm/mirror/%.c
2207      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2209 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/lvm/raid/%.c
2210      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2212 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/lvm/softpart/%.c
2213      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2215 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/lvm/stripe/%.c
2216      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2218 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/lvm/notify/%.c
2219      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2221 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/lvm/trans/%.c
2222      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2224 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/mac/%.c
2225      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2227 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/mac/plugins/%.c
2228      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2230 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/mega_sas/%.c
2231      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2233 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/mii/%.c
2234      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2236 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/mr_sas/%.c
2237      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2239 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/scsi/adapters/mpt_sas/%.c

```

```

2240      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2242 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/scsi/adapters/mpt_sas3/%.c
2243      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2245 #endif /* ! codereview */
2246 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/mxfe/%.c
2247      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2249 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/mwl/%.c
2250      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2252 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/mwl/mwl_fw/%.c
2253      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2255 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/net80211/%.c
2256      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2258 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/nge/%.c
2259      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2261 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/nxge/%.c
2262      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2264 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/nxge/%.s
2265      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2267 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/nxge/npi/%.c
2268      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2270 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/pci-ide/%.c
2271      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2273 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/pcn/%.c
2274      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2276 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/ppp/sppp/%.c
2277      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2279 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/ppp/spppasyn/%.c
2280      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2282 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/ppp/spptun/%.c
2283      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2285 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/ral/%.c
2286      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2288 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/rge/%.c
2289      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2291 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/rtls/%.c
2292      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2294 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/rsm/%.c
2295      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2297 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/rtw/%.c
2298      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2300 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/rum/%.c
2301      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2303 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/rwd/%.c
2304      @$(LHEAD) $(LINT.c) $< $(LTAIL))

```

```

2306 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/rwn/%.c
2307      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2309 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/sata/adapters/ahci/%.c
2310      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2312 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/sata/adapters/nv_sata/%.c
2313      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2315 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/sata/adapters/si3124/%.c
2316      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2318 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/sata/impl/%.c
2319      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2321 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/scsi/adapters/%.c
2322      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2324 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/scsi/adapters/blk2scsa/%.c
2325      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2327 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/scsi/adapters/pmcs/%.c
2328      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2330 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/scsi/adapters/scsi_vhci/%.c
2331      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2333 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/scsi/adapters/scsi_vhci/fop
2334      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2336 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/fibre-channel/ulp/%.c
2337      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2339 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/fibre-channel/impl/%.c
2340      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2342 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/fibre-channel/fca/qlc/%.c
2343      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2345 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/fibre-channel/fca/qlge/%.c
2346      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2348 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/fibre-channel/fca/emlxs/%.c
2349      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2351 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/fibre-channel/fca/oce/%.c
2352      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2354 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/fibre-channel/fca/fcoei/%.c
2355      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2357 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/scsi/conf/%.c
2358      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2360 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/scsi/impl/%.c
2361      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2363 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/scsi/targets/%.c
2364      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2366 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/sdcard/adapters/sdhost/%.c
2367      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2369 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/sdcard/impl/%.c
2370      @$(LHEAD) $(LINT.c) $< $(LTAIL))

```

```
2372 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/sdcard/targets/sdcard/%.c
2373     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2375 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/sfe/%.c
2376     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2378 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/simnet/%.c
2379     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2381 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/softmac/%.c
2382     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2384 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/uath/%.c
2385     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2387 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/uath/uath_fw/%.c
2388     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2390 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/ural/%.c
2391     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2393 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/urtw/%.c
2394     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2396 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/usb/clients/audio/usb_ac/%.
2397     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2399 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/usb/clients/audio/usb_as/%.
2400     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2402 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/usb/clients/audio/usb_ah/%.
2403     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2405 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/usb/clients/usbskel/%.c
2406     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2408 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/usb/clients/video/usbvc/%.c
2409     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2411 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/usb/clients/hwarc/%.c
2412     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2414 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/usb/clients/hid/%.c
2415     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2417 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/usb/clients/hidparser/%.c
2418     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2420 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/usb/clients/usbbkm/%.c
2421     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2423 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/usb/clients/uslbs/%.c
2424     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2426 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/usb/clients/usbinput/usbwcm
2427     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2429 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/usb/clients/ugen/%.c
2430     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2432 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/usb/clients/printer/%.c
2433     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2435 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/usb/clients/usbser/%.c
2436     @$(LHEAD) $(LINT.c) $< $(LTAIL))
```

```
2438 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/usb/clients/usbser/usbsacm/
2439     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2441 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/usb/clients/usbser/usbftdi/
2442     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2444 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/usb/clients/usbser/usbser_k
2445     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2447 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/usb/clients/usbser/usbsprl/
2448     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2450 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/usb/clients/wusb_df/%.c
2451     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2453 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/usb/clients/hwa1480_fw/%.c
2454     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2456 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/usb/clients/wusb_ca/%.c
2457     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2459 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/usb/clients/usbecm/%.c
2460     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2462 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/usb/hcd/openhci/%.c
2463     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2465 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/usb/hcd/ehci/%.c
2466     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2468 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/usb/hcd/uhci/%.c
2469     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2471 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/usb/hubd/%.c
2472     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2474 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/usb/scsa2usb/%.c
2475     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2477 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/usb/usb_mid/%.c
2478     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2480 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/usb/usb_ia/%.c
2481     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2483 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/usb/usba/%.c
2484     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2486 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/usb/usba10/%.c
2487     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2489 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/uwb/uwba/%.c
2490     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2492 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/usb/hwa/hwahc/%.c
2493     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2495 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/vuidmice/%.c
2496     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2498 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/vnic/%.c
2499     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2501 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/wpi/%.c
2502     @$(LHEAD) $(LINT.c) $< $(LTAIL))
```

```
2504 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/zyd/%.c
2505     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2507 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/chxge/com/%.c
2508     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2510 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/chxge/%.c
2511     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2513 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/cxgbe/common/%.c
2514     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2516 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/cxgbe/shared/%.c
2517     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2519 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/cxgbe/firmware/%.c
2520     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2522 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/cxgbe/t4nex/%.c
2523     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2525 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/cxgbe/cxgbe/%.c
2526     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2528 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/ixgb/%.c
2529     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2531 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/xge/drv/%.c
2532     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2534 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/xge/hal/xgehal/%.c
2535     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2537 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/e1000g/%.c
2538     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2540 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/e1000api/%.c
2541     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2543 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/igb/%.c
2544     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2546 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/iprb/%.c
2547     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2549 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/ixgbe/%.c
2550     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2552 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/ntxn/%.c
2553     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2555 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/myri10ge/drv/%.c
2556     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2558 $(LINTSDIR)/%.ln: $(UTSBASE)/common/ipp/%.c
2559     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2561 $(LINTSDIR)/%.ln: $(UTSBASE)/common/ipp/ipgpc/%.c
2562     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2564 $(LINTSDIR)/%.ln: $(UTSBASE)/common/ipp/dlcosmk/%.c
2565     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2567 $(LINTSDIR)/%.ln: $(UTSBASE)/common/ipp/flowacct/%.c
2568     @$(LHEAD) $(LINT.c) $< $(LTAIL))
```

```
2570 $(LINTSDIR)/%.ln: $(UTSBASE)/common/ipp/dscpmk/%.c
2571     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2573 $(LINTSDIR)/%.ln: $(UTSBASE)/common/ipp/meters/%.c
2574     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2576 $(LINTSDIR)/%.ln: $(UTSBASE)/common/kiconv/kiconv_emea/%.c
2577     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2579 $(LINTSDIR)/%.ln: $(UTSBASE)/common/kiconv/kiconv_ja/%.c
2580     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2582 $(LINTSDIR)/%.ln: $(UTSBASE)/common/kiconv/kiconv_ko/%.c
2583     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2585 $(LINTSDIR)/%.ln: $(UTSBASE)/common/kiconv/kiconv_sc/%.c
2586     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2588 $(LINTSDIR)/%.ln: $(UTSBASE)/common/kiconv/kiconv_tc/%.c
2589     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2591 $(LINTSDIR)/%.ln: $(UTSBASE)/common/klm/%.c
2592     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2594 $(LINTSDIR)/%.ln: $(UTSBASE)/common/kmdb/%.c
2595     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2597 $(LINTSDIR)/%.ln: $(UTSBASE)/common/krtld/%.c
2598     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2600 $(LINTSDIR)/%.ln: $(UTSBASE)/common/ktli/%.c
2601     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2603 $(LINTSDIR)/%.ln: $(COMMONBASE)/list/%.c
2604     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2606 $(LINTSDIR)/%.ln: $(COMMONBASE)/lvm/%.c
2607     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2609 $(LINTSDIR)/%.ln: $(COMMONBASE)/lzma/%.c
2610     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2612 $(LINTSDIR)/%.ln: $(COMMONBASE)/crypto/md4/%.c
2613     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2615 $(LINTSDIR)/%.ln: $(COMMONBASE)/crypto/md5/%.c
2616     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2618 $(LINTSDIR)/%.ln: $(COMMONBASE)/net/dhcp/%.c
2619     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2621 $(LINTSDIR)/%.ln: $(COMMONBASE)/nvpair/%.c
2622     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2624 $(LINTSDIR)/%.ln: $(UTSBASE)/common/os/%.c
2625     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2627 $(LINTSDIR)/%.ln: $(UTSBASE)/common/rpc/%.c
2628     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2630 $(LINTSDIR)/%.ln: $(UTSBASE)/common/pcmcia/cs/%.c
2631     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2633 $(LINTSDIR)/%.ln: $(UTSBASE)/common/pcmcia/cis/%.c
2634     @$(LHEAD) $(LINT.c) $< $(LTAIL))
```

```

2636 $(LINTSDIR)/%.ln:      $(UTSBASE)/common/pcmcia/nexus/%.c
2637     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2639 $(LINTSDIR)/%.ln:      $(UTSBASE)/common/pcmcia/pcs/%.c
2640     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2642 $(LINTSDIR)/%.ln:      $(UTSBASE)/common/rpc/%.c
2643     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2645 $(LINTSDIR)/%.ln:      $(UTSBASE)/common/rpc/sec/%.c
2646     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2648 $(LINTSDIR)/%.ln:      $(UTSBASE)/common/rpc/sec_gss/%.c
2649     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2651 $(LINTSDIR)/%.ln:      $(COMMONBASE)/crypto/sha1/%.c
2652     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2654 $(LINTSDIR)/%.ln:      $(COMMONBASE)/crypto/sha2/%.c
2655     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2657 $(LINTSDIR)/%.ln:      $(UTSBASE)/common/syscall/%.c
2658     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2660 $(LINTSDIR)/%.ln:      $(UTSBASE)/common/tnf/%.c
2661     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2663 $(LINTSDIR)/%.ln:      $(COMMONBASE)/tsol/%.c
2664     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2666 $(LINTSDIR)/%.ln:      $(COMMONBASE)/util/%.c
2667     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2669 $(LINTSDIR)/%.ln:      $(COMMONBASE)/unicode/%.c
2670     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2672 $(LINTSDIR)/%.ln:      $(UTSBASE)/common/vm/%.c
2673     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2675 $(LINTSDIR)/%.ln:      $(UTSBASE)/common/io/scsi/adapters/iscsi/%.c
2676     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2678 $(LINTSDIR)/%.ln:      $(COMMONBASE)/iscsi/%.c
2679     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2681 $(LINTSDIR)/%.ln:      $(UTSBASE)/common/inet/kifconf/%.c
2682     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2684 $(LINTSDIR)/%.ln:      $(UTSBASE)/common/io/virtio/%.c
2685     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2687 $(LINTSDIR)/%.ln:      $(UTSBASE)/common/io/vioblk/%.c
2688     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2690 ZMODLINTFLAGS = -erroff=E_CONSTANT_CONDITION

2692 $(LINTSDIR)/%.ln:      $(UTSBASE)/common/zmod/%.c
2693     @$(LHEAD) $(LINT.c) $(ZMODLINTFLAGS) $< $(LTAIL))

2695 $(LINTSDIR)/zlib_obj.ln: $(ZLIB_OBJS:%.o=$(LINTSDIR)/%.ln) \
2696     $(UTSBASE)/common/zmod/zlib_lint.c
2697     @$(LHEAD) $(LINT.c) -C $(LINTSDIR)/zlib_obj \
2698     $(UTSBASE)/common/zmod/zlib_lint.c $(LTAIL))

2700 $(LINTSDIR)/%.ln:      $(UTSBASE)/common/io/hxge/%.c
2701     @$(LHEAD) $(LINT.c) $< $(LTAIL))

```

```

2703 $(LINTSDIR)/%.ln:      $(UTSBASE)/common/io/tpm/%.c
2704     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2706 $(LINTSDIR)/%.ln:      $(UTSBASE)/common/io/tpm/%.s
2707     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2709 $(LINTSDIR)/%.ln:      $(UTSBASE)/common/io/vr/%.c
2710     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2712 $(LINTSDIR)/%.ln:      $(UTSBASE)/common/io/yge/%.c
2713     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2715 $(LINTSDIR)/%.ln:      $(COMMONBASE)/fsreparse/%.c
2716     @$(LHEAD) $(LINT.c) $< $(LTAIL))

```


new/usr/src/uts/common/io/scsi/adapters/mpt_sas3/mpt_sas3.conf

1

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

```
1 #
2 # CDDL HEADER START
3 #
4 # The contents of this file are subject to the terms of the
5 # Common Development and Distribution License (the "License").
6 # You may not use this file except in compliance with the License.
7 #
8 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9 # or http://www.opensolaris.org/os/licensing.
10 # See the License for the specific language governing permissions
11 # and limitations under the License.
12 #
13 # When distributing Covered Code, include this CDDL HEADER in each
14 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 # If applicable, add the following below this CDDL HEADER, with the
16 # fields enclosed by brackets "[]" replaced with your own identifying
17 # information: Portions Copyright [yyyy] [name of copyright owner]
18 #
19 # CDDL HEADER END
20 #
21 #
22 # Copyright 2009 Sun Microsystems, Inc. All rights reserved.
23 # Use is subject to license terms.
24 #
25 #
26 #
27 #
28 # The mpt_sas driver, as a PHCI driver, must specify the VHCI class it
29 # belongs to(scsi_vhci).
30 #
31 ddi-vhci-class="scsi_vhci";
32 #
33 # I/O multipathing feature (MPxIO) can be enabled or disabled using
34 # mpzio-disable property. Setting mpzio-disable="no" will activate
35 # I/O multipathing; setting mpzio-disable="yes" disables the feature.
36 #
37 # Global mpzio-disable property:
38 #
39 # To globally enable MPxIO on all LSI MPT SAS 2.0 controllers set:
40 # mpzio-disable="no";
41 #
42 # To globally disable MPxIO on all LSI MPT SAS 2.0 controllers set:
43 # mpzio-disable="yes";
44 #
45 # You can also enable or disable MPxIO on a per HBA basis.
46 # Per HBA settings override the global setting for the specified HBAs.
47 # To disable MPxIO on a controller whose parent is /pci@7c0/pci@0/pci@9
48 # and the unit-address is "0" set:
49 # name="mpt_sas" parent="/pci@7c0/pci@0/pci@9" unit-address="0" mpzio-disable="y
50 #
51 mpzio-disable="no";
52 #endif /* ! codereview */
```

new/usr/src/uts/common/io/scsi/adapters/mpt_sas3/mptsas3.c

1

```
*****
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
*****

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

22 /*
23  * Copyright (c) 2009, 2010, Oracle and/or its affiliates. All rights reserved.
24  * Copyright 2014 Nexenta Systems, Inc. All rights reserved.
25  * Copyright (c) 2014, Joyent, Inc. All rights reserved.
26  * Copyright (c) 2014, Tegile Systems Inc. All rights reserved.
27  * Copyright 2014 OmniTI Computer Consulting, Inc. All rights reserved.
28 */

30 /*
31  * Copyright (c) 2000 to 2010, LSI Corporation.
32  * All rights reserved.
33  *
34  * Redistribution and use in source and binary forms of all code within
35  * this file that is exclusively owned by LSI, with or without
36  * modification, is permitted provided that, in addition to the CDDL 1.0
37  * License requirements, the following conditions are met:
38  *
39  *   Neither the name of the author nor the names of its contributors may be
40  *   used to endorse or promote products derived from this software without
41  *   specific prior written permission.
42  *
43  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
44  * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
45  * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
46  * FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
47  * COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
48  * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
49  * BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
50  * OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
51  * AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
52  * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
53  * OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
54  * DAMAGE.
55 */

57 /*
58  * mptsas3 - This is a driver based on LSI Logic's MPT2.0/2.5 interface.
59  */
60 */
```

new/usr/src/uts/common/io/scsi/adapters/mpt_sas3/mptsas3.c

2

```
62 #if defined(lint) || defined(DEBUG)
63 #define MPTSAS_DEBUG
64 #endif

66 /*
67  * standard header files.
68 */
69 #include <sys/note.h>
70 #include <sys/scsi/scsi.h>
71 #include <sys/pci.h>
72 #include <sys/file.h>
73 #include <sys/policy.h>
74 #include <sys/model.h>
75 #include <sys/sysevent.h>
76 #include <sys/sysevent/eventdefs.h>
77 #include <sys/sysevent/dr.h>
78 #include <sys/sata/sata_defs.h>
79 #include <sys/scsi/generic/sas.h>
80 #include <sys/scsi/impl/scsi_sas.h>

82 #pragma pack(1)
83 #include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_type.h>
84 #include <sys/scsi/adapters/mpt_sas3/mpi/mpi2.h>
85 #include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_cnfg.h>
86 #include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_init.h>
87 #include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_ioc.h>
88 #include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_sas.h>
89 #include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_tool.h>
90 #include <sys/scsi/adapters/mpt_sas3/mpi/mpi2 RAID.h>
91 #pragma pack()

93 /*
94  * private header files.
95  */
96 #include <sys/scsi/impl/scsi_reset_notify.h>
97 #include <sys/scsi/adapters/mpt_sas3/mptsas3_var.h>
98 #include <sys/scsi/adapters/mpt_sas3/mptsas3_ioctl.h>
99 #include <sys/scsi/adapters/mpt_sas3/mptsas3_smhba.h>
100 #include <sys/scsi/adapters/mpt_sas3/mptsas3_hash.h>
101 #include <sys/raidioctl.h>

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

106 /*
107  * FMA header files
108 */
109 #include <sys/ddifm.h>
110 #include <sys/fm/protocol.h>
111 #include <sys/fm/util.h>
112 #include <sys/fm/io/ddi.h>

114 /*
115  * autoconfiguration data and routines.
116 */
117 static int mptsas_attach(dev_info_t *dip, ddi_attach_cmd_t cmd);
118 static int mptsas_detach(dev_info_t *devi, ddi_detach_cmd_t cmd);
119 static int mptsas_power(dev_info_t *dip, int component, int level);

121 /*
122  * cb_ops function
123 */
124 static int mptsas_ioctl(dev_t dev, int cmd, intptr_t data, int mode,
125 cred_t *credp, int *rval);
126 #ifdef __sparc
127 static int mptsas_reset(dev_info_t *devi, ddi_reset_cmd_t cmd);
```

```

128 #else /* __sparc */
129 static int mptsas_quiesce(dev_info_t *devi);
130 #endif /* __sparc */

132 /*
133  * Resource initialization for hardware
134  */
135 static void mptsas_setup_cmd_reg(mptsas_t *mpt);
136 static void mptsas_disable_bus_master(mptsas_t *mpt);
137 static void mptsas_hba_fini(mptsas_t *mpt);
138 static void mptsas_cfg_fini(mptsas_t *mptsas_blkp);
139 static int mptsas_hba_setup(mptsas_t *mpt);
140 static void mptsas_hba_teardown(mptsas_t *mpt);
141 static int mptsas_config_space_init(mptsas_t *mpt);
142 static void mptsas_config_space_fini(mptsas_t *mpt);
143 static void mptsas_iport_register(mptsas_t *mpt);
144 static int mptsas_smp_setup(mptsas_t *mpt);
145 static void mptsas_smp_teardown(mptsas_t *mpt);
146 static int mptsas_cache_create(mptsas_t *mpt);
147 static void mptsas_cache_destroy(mptsas_t *mpt);
148 static int mptsas_alloc_request_frames(mptsas_t *mpt);
149 static int mptsas_alloc_sense_bufs(mptsas_t *mpt);
150 static int mptsas_alloc_reply_frames(mptsas_t *mpt);
151 static int mptsas_alloc_free_queue(mptsas_t *mpt);
152 static int mptsas_alloc_post_queue(mptsas_t *mpt);
153 static void mptsas_free_post_queue(mptsas_t *mpt);
154 static void mptsas_alloc_reply_args(mptsas_t *mpt);
155 static int mptsas_alloc_extra_sgl_frame(mptsas_t *mpt, mptsas_cmd_t *cmd);
156 static void mptsas_free_extra_sgl_frame(mptsas_t *mpt, mptsas_cmd_t *cmd);
157 static int mptsas_init_chip(mptsas_t *mpt, int first_time);

159 /*
160  * SCSI function prototypes
161  */
162 static int mptsas_scsi_start(struct scsi_address *ap, struct scsi_pkt *pkt);
163 static int mptsas_scsi_reset(struct scsi_address *ap, int level);
164 static int mptsas_scsi_abort(struct scsi_address *ap, struct scsi_pkt *pkt);
165 static int mptsas_scsi_getcap(struct scsi_address *ap, char *cap, int tgtonly);
166 static int mptsas_scsi_setcap(struct scsi_address *ap, char *cap, int value,
167     int tgtonly);
168 static void mptsas_scsi_dmafree(struct scsi_address *ap, struct scsi_pkt *pkt);
169 static struct scsi_pkt *mptsas_scsi_init_pkt(struct scsi_address *ap,
170     struct scsi_pkt *pkt, struct buf *bp, int cmdlen, int statuslen,
171     int tgtlen, int flags, int (*callback)(), caddr_t arg);
172 static void mptsas_scsi_sync_pkt(struct scsi_address *ap, struct scsi_pkt *pkt);
173 static void mptsas_scsi_destroy_pkt(struct scsi_address *ap,
174     struct scsi_pkt *pkt);
175 static int mptsas_scsi_tgt_init(dev_info_t *hba_dip, dev_info_t *tgt_dip,
176     scsi_hba_tran_t *hba_tran, struct scsi_device *sd);
177 static void mptsas_scsi_tgt_free(dev_info_t *hba_dip, dev_info_t *tgt_dip,
178     scsi_hba_tran_t *hba_tran, struct scsi_device *sd);
179 static int mptsas_scsi_reset_notify(struct scsi_address *ap, int flag,
180     void (*callback)(caddr_t), caddr_t arg);
181 static int mptsas_get_name(struct scsi_device *sd, char *name, int len);
182 static void mptsas_get_bus_addr(struct scsi_device *sd, char *name, int len);
183 static int mptsas_scsi_quiesce(dev_info_t *dip);
184 static int mptsas_scsi_unquiesce(dev_info_t *dip);
185 static int mptsas_bus_config(dev_info_t *pdip, uint_t flags,
186     ddi_bus_config_op_t op, void *arg, dev_info_t **childp);

188 /*
189  * SMP functions
190  */
191 static int mptsas_smp_start(struct smp_pkt *smp_pkt);
193 */

```

```

194 * internal function prototypes.
195 */
196 static void mptsas_list_add(mptsas_t *mpt);
197 static void mptsas_list_del(mptsas_t *mpt);

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

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

205 static void mptsas_ncmds_checkdrain(void *arg);

207 static int mptsas_prepare_pkt(mptsas_cmd_t *cmd);
208 static void mptsas_retry_pkt(mptsas_t *mpt, mptsas_cmd_t *sp);
209 static int mptsas_save_cmd_to_slot(mptsas_t *mpt, mptsas_cmd_t *cmd);
210 static int mptsas_accept_pkt(mptsas_t *mpt, mptsas_cmd_t *sp,
211     int *tran_rval);
212 static void mptsas_accept_tx_waitqs(mptsas_t *mpt);
213 static void mptsas_unblock_tx_waitqs(mptsas_t *mpt);
214 static void mptsas_drain_tx_waitq(mptsas_t *mpt, mptsas_tx_waitqueue_t *txwq);
215 static int mptsas_check_targ_intxion(mptsas_target_t *ptgt, int cmd_pkt_flags);

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

223 static void mptsas_handle_qfull(mptsas_t *mpt, mptsas_cmd_t *cmd);
224 static void mptsas_handle_event(void *args);
225 static int mptsas_handle_event_sync(void *args);
226 static void mptsas_handle_dr(void *args);
227 static void mptsas_handle_topo_change(mptsas_topo_change_list_t *topo_node,
228     dev_info_t *pdip);

230 static void mptsas_restart_cmd(void *);

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

238 static uint_t mptsas_intr(caddr_t arg1, caddr_t arg2);
239 static void mptsas_process_intr(mptsas_t *mpt, mptsas_reply_pqueue_t *rpqp,
240     pMpi2ReplyDescriptorsUnion_t reply_desc_union);
241 static void mptsas_handle_scsi_io_success(mptsas_t *mpt,
242     mptsas_reply_pqueue_t *rpqp, pMpi2ReplyDescriptorsUnion_t reply_desc);
243 static void mptsas_handle_address_reply(mptsas_t *mpt,
244     pMpi2ReplyDescriptorsUnion_t reply_desc);
245 static int mptsas_wait_intr(mptsas_t *mpt, int polltime);
246 static void mptsas_sge_setup(mptsas_t *mpt, mptsas_cmd_t *cmd,
247     uint32_t *control, pMpi2SCSIIORequest_t frame, ddi_acc_handle_t acc_hdl);

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

253 static void mptsas_start_passthru(mptsas_t *mpt, mptsas_cmd_t *cmd);
254 static int mptsas_do_passthru(mptsas_t *mpt, uint8_t *request, uint8_t *reply,
255     uint8_t *data, uint32_t request_size, uint32_t reply_size,
256     uint32_t data_size, uint8_t direction, uint8_t *dataout,
257     uint32_t dataout_size, short timeout, int mode);
258 static int mptsas_free_devhdl(mptsas_t *mpt, uint16_t devhdl);

```

```

260 static uint8_t mptsas_get_fw_diag_buffer_number(mptsas_t *mpt,
261     uint32_t unique_id);
262 static void mptsas_start_diag(mptsas_t *mpt, mptsas_cmd_t *cmd);
263 static int mptsas_post_fw_diag_buffer(mptsas_t *mpt,
264     mptsas_fw_diagnostic_buffer_t *pBuffer, uint32_t *return_code);
265 static int mptsas_release_fw_diag_buffer(mptsas_t *mpt,
266     mptsas_fw_diagnostic_buffer_t *pBuffer, uint32_t *return_code,
267     uint32_t diag_type);
268 static int mptsas_diag_register(mptsas_t *mpt,
269     mptsas_fw_diag_register_t *diag_register, uint32_t *return_code);
270 static int mptsas_diag_unregister(mptsas_t *mpt,
271     mptsas_fw_diag_unregister_t *diag_unregister, uint32_t *return_code);
272 static int mptsas_diag_query(mptsas_t *mpt, mptsas_fw_diag_query_t *diag_query,
273     uint32_t *return_code);
274 static int mptsas_diag_read_buffer(mptsas_t *mpt,
275     mptsas_fw_diag_read_buffer_t *diag_read_buffer, uint8_t *ioctl_buf,
276     uint32_t *return_code, int ioctl_mode);
277 static int mptsas_diag_release(mptsas_t *mpt,
278     mptsas_fw_diag_release_t *diag_release, uint32_t *return_code);
279 static int mptsas_do_diag_action(mptsas_t *mpt, uint32_t action,
280     uint8_t *diag_action, uint32_t length, uint32_t *return_code,
281     int ioctl_mode);
282 static int mptsas_diag_action(mptsas_t *mpt, mptsas_diag_action_t *data,
283     int mode);

285 static int mptsas_pkt_alloc_extern(mptsas_t *mpt, mptsas_cmd_t *cmd,
286     int cmdlen, int tgtnlen, int statuslen, int kf);
287 static void mptsas_pkt_destroy_extern(mptsas_t *mpt, mptsas_cmd_t *cmd);

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

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

296 static void mptsas_check_scsi_io_error(mptsas_t *mpt, pMpi2SCSIIOReply_t reply,
297     mptsas_cmd_t *cmd);
298 static void mptsas_check_task_mgt(mptsas_t *mpt,
299     pMpi2SCSIManagementReply_t reply, mptsas_cmd_t *cmd);
300 static int mptsas_send_scsi_cmd(mptsas_t *mpt, struct scsi_address *ap,
301     mptsas_target_t *ptgt, uchar_t *cdb, int cdblen, struct buf *data_bp,
302     int *resid);

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

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

311 static void mptsas_deliver_doneq_thread(mptsas_t *mpt,
312     mptsas_done_list_t *dlist);
313 static void mptsas_doneq_add(mptsas_t *mpt, mptsas_cmd_t *cmd);
314 static void mptsas_rpdoneq_add(mptsas_t *mpt, mptsas_reply_pqueue_t *rpqp,
315     mptsas_cmd_t *cmd);
316 static void mptsas_doneq_mv(mptsas_done_list_t *from,
317     mptsas_doneq_thread_list_t *item);

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

325 static mptsas_cmd_t *mptsas_waitq_rm(mptsas_t *mpt);

```

```

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

328 static void mptsas_start_watch_reset_delay();
329 static void mptsas_setup_bus_reset_delay(mptsas_t *mpt);
330 static void mptsas_watch_reset_delay(void *arg);
331 static int mptsas_watch_reset_delay_subr(mptsas_t *mpt);
332 static void mptsas_set_throttle(struct mptsas *mpt, mptsas_target_t *ptgt,
333     int what);
334 static void mptsas_set_throttle_mtx(struct mptsas *mpt, mptsas_target_t *ptgt,
335     int what);
336 static void mptsas_remove_cmd_nmtx(mptsas_t *mpt, mptsas_cmd_t *cmd);

338 /*
339  * helper functions
340  */
341 static void mptsas_dump_cmd(mptsas_t *mpt, mptsas_cmd_t *cmd);

343 static dev_info_t *mptsas_find_child(dev_info_t *pdip, char *name);
344 static dev_info_t *mptsas_find_child_phy(dev_info_t *pdip, uint8_t phy);
345 static dev_info_t *mptsas_find_child_addr(dev_info_t *pdip, uint64_t sasaddr,
346     int lun);
347 static mdi_pathinfo_t *mptsas_find_path_addr(dev_info_t *pdip, uint64_t sasaddr,
348     int lun);
349 static mdi_pathinfo_t *mptsas_find_path_phy(dev_info_t *pdip, uint8_t phy);
350 static dev_info_t *mptsas_find_smp_child(dev_info_t *pdip, char *str_wwn);

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

356 static mptsas_target_t *mptsas_phy_to_tgt(mptsas_t *mpt,
357     mptsas_phymask_t phymask, uint8_t phy);
358 static mptsas_target_t *mptsas_wwid_to_ptgt(mptsas_t *mpt,
359     mptsas_phymask_t phymask, uint64_t wwid);
360 static mptsas_smp_t *mptsas_wwid_to_psmp(mptsas_t *mpt,
361     mptsas_phymask_t phymask, uint64_t wwid);

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

366 static int mptsas_get_target_device_info(mptsas_t *mpt, uint32_t page_address,
367     uint16_t *handle, mptsas_target_t *ptgt);
368 static void mptsas_update_phymask(mptsas_t *mpt);

370 static int mptsas_send_sep(mptsas_t *mpt, mptsas_target_t *ptgt,
371     uint32_t *status, uint8_t cmd);
372 static dev_info_t *mptsas_get_dip_from_dev(dev_t dev,
373     mptsas_phymask_t *phymask);
374 static mptsas_target_t *mptsas_addr_to_ptgt(mptsas_t *mpt, char *addr,
375     mptsas_phymask_t phymask);
376 static int mptsas_flush_led_status(mptsas_t *mpt, mptsas_target_t *ptgt);

379 /*
380  * Enumeration / DR functions
381  */
382 static void mptsas_config_all(dev_info_t *pdip);
383 static int mptsas_config_one_addr(dev_info_t *pdip, uint64_t sasaddr, int lun,
384     dev_info_t **lundi);
385 static int mptsas_config_one_phy(dev_info_t *pdip, uint8_t phy, int lun,
386     dev_info_t **lundi);

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

```

```

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

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

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

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

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

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

419 static int mptsas_event_query(mptsas_t *mpt, mptsas_event_query_t *data,
420     int mode, int *rval);
421 static int mptsas_event_enable(mptsas_t *mpt, mptsas_event_enable_t *data,
422     int mode, int *rval);
423 static int mptsas_event_report(mptsas_t *mpt, mptsas_event_report_t *data,
424     int mode, int *rval);
425 static void mptsas_record_event(void *args);
426 static int mptsas_reg_access(mptsas_t *mpt, mptsas_reg_access_t *data,
427     int mode);

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

435 /*
436  * Power management functions
437  */
438 static int mptsas_get_pci_cap(mptsas_t *mpt);
439 static int mptsas_init_pm(mptsas_t *mpt);

441 /*
442  * MPT MSI tunable:
443  *
444  * By default MSI is enabled on all supported platforms.
445  */
446 boolean_t mptsas_enable_msi = B_TRUE;
447 boolean_t mptsas_enable_msix = B_TRUE;
448 boolean_t mptsas_physical_bind_failed_page_83 = B_FALSE;

450 /*
451  * Global switch for use of MPI2.5 FAST PATH.
452  */
453 boolean_t mptsas3_use_fastpath = B_TRUE;

455 static int mptsas_register_intrs(mptsas_t *);
456 static void mptsas_unregister_intrs(mptsas_t *);
457 static int mptsas_add_intrs(mptsas_t *, int);

```

```

458 static void mptsas_rem_intrs(mptsas_t *);

460 /*
461  * FMA Prototypes
462  */
463 static void mptsas_fm_init(mptsas_t *mpt);
464 static void mptsas_fm_fini(mptsas_t *mpt);
465 static int mptsas_fm_error_cb(dev_info_t *, ddi_fm_error_t *, const void *);

467 extern pri_t minclsyspri, maxclsyspri;
468 /*
469  * NCPUS is used to determine some optimal configurations for number
470  * of threads created to perform specific jobs. If we are invoked because
471  * a disk is part of the root file system ncpus may still be 1 so check
472  * boot_ncpus as well.
473  */
474 extern int ncpus, boot_ncpus;
475 #define NCPUS    max(ncpus, boot_ncpus)

477 /*
478  * This device is created by the SCSI pseudo nexus driver (SCSI VHCI). It is
479  * under this device that the paths to a physical device are created when
480  * MPxIO is used.
481  */
482 extern dev_info_t      *scsi_vhci_dip;

484 /*
485  * Tunable timeout value for Inquiry VPD page 0x83
486  * By default the value is 30 seconds.
487  */
488 int mptsas_inq83_retry_timeout = 30;

490 /*
491  * Tunable for default SCSI pkt timeout. Defaults to 5 seconds, which should
492  * be plenty for INQUIRY and REPORT_LUNS, which are the only commands currently
493  * issued by mptsas directly.
494  */
495 int mptsas_scsi_pkt_time = 5;

497 /*
498  * This is used to allocate memory for message frame storage, not for
499  * data I/O DMA. All message frames must be stored in the first 4G of
500  * physical memory.
501  */
502 ddi_dma_attr_t mptsas_dma_attrs = {
503     DMA_ATTR_V0,      /* attribute layout version */
504     0x0ull,           /* address low - should be 0 (longlong) */
505     0xfffffffffull,   /* address high - 32-bit max range */
506     0x00fffffffull,   /* count max - max DMA object size */
507     4,                /* allocation alignment requirements */
508     0x78,             /* burstsizes - binary encoded values */
509     1,                /* minxfer - gran. of DMA engine */
510     0x00fffffffull,   /* maxxfer - gran. of DMA engine */
511     0xfffffffffull,   /* max segment size (DMA boundary) */
512     MPTSAS_MAX_DMA_SEGS, /* scatter/gather list length */
513     512,              /* granularity - device transfer size */
514     0,                /* flags, set to 0 */
515 };

517 /*
518  * This is used for data I/O DMA memory allocation. (full 64-bit DMA
519  * physical addresses are supported.)
520  */
521 ddi_dma_attr_t mptsas_dma_attrs64 = {
522     DMA_ATTR_V0,      /* attribute layout version */
523     0x0ull,           /* address low - should be 0 (longlong) */

```

```

524 0xffffffffffffffff, /* address high - 64-bit max */
525 0x0000000000000000, /* count max - max DMA object size */
526 4, /* allocation alignment requirements */
527 0x78, /* burstsizes - binary encoded values */
528 1, /* minxfer - gran. of DMA engine */
529 0x0000000000000000, /* maxxfer - gran. of DMA engine */
530 0xffffffff, /* max segment size (DMA boundary) */
531 MPTSAS_MAX_DMA_SEGS, /* scatter/gather list length */
532 512, /* granularity - device transfer size */
533 0 /* flags, set to 0 */
534 };

536 ddi_device_acc_attr_t mptsas_dev_attr = {
537     DDI_DEVICE_ATTR_V1,
538     DDI_STRUCTURE_LE_ACC,
539     DDI_STRICTORDER_ACC,
540     DDI_DEFAULT_ACC
541 };

543 static struct cb_ops mptsas_cb_ops = {
544     scsi_hba_open, /* open */
545     scsi_hba_close, /* close */
546     nodev, /* strategy */
547     nodev, /* print */
548     nodev, /* dump */
549     nodev, /* read */
550     nodev, /* write */
551     mptsas_ioctl, /* ioctl */
552     nodev, /* devmap */
553     nodev, /* mmap */
554     nodev, /* segmap */
555     nochpoll, /* chpoll */
556     ddi_prop_op, /* cb_prop_op */
557     NULL, /* streamtab */
558     D_MP, /* cb_flag */
559     CB_REV, /* rev */
560     nodev, /* aread */
561     nodev, /* awrite */
562 };

564 static struct dev_ops mptsas_ops = {
565     DEVO_REV, /* devo_rev */
566     0, /* refcnt */
567     ddi_no_info, /* info */
568     nulldev, /* identify */
569     nulldev, /* probe */
570     mptsas_attach, /* attach */
571     mptsas_detach, /* detach */
572 #ifdef __sparc
573     mptsas_reset,
574 #else
575     nodev, /* reset */
576 #endif /* __sparc */
577     &mptsas_cb_ops, /* driver operations */
578     NULL, /* bus operations */
579     mptsas_power, /* power management */
580 #ifdef __sparc
581     ddi_quiesce_not_needed
582 #else
583     mptsas_quiesce /* quiesce */
584 #endif /* __sparc */
585 };

588 #define MPTSAS_MOD_STRING "MPTSAS3 HBA Driver 00.00.01"

```

```

590 static struct modldrv modldrv = {
591     &mod_driverops, /* Type of module. This one is a driver */
592     MPTSAS_MOD_STRING, /* Name of the module. */
593     &mptsas_ops, /* driver ops */
594 };

596 static struct modlinkage modlinkage = {
597     MODREV_1, &modldrv, NULL
598 };
599 #define TARGET_PROP "target"
600 #define LUN_PROP "lun"
601 #define LUN64_PROP "lun64"
602 #define SAS_PROP "sas-mpt"
603 #define MDI_GUID "wwn"
604 #define NDI_GUID "guid"
605 #define MPTSAS_DEV_GONE "mptsas_dev_gone"

607 /*
608  * Local static data
609  */
610 #if defined(MPTSAS_DEBUG)
611 uint32_t mptsas_debug_flags = 0x0;
612 /*
613  * Flags to ignore these messages in local debug ring buffer.
614  * Default is to ignore the watchsubr() output which normally happens
615  * every second.
616  */
617 uint32_t mptsas_dbglog_imask = 0x40000000;
618 uint32_t mptsas_test_timeout = 0;
619 #endif /* defined(MPTSAS_DEBUG) */
620 uint32_t mptsas_debug_resets = 0;

622 static kmutex_t mptsas_global_mutex;
623 static void *mptsas3_state; /* soft state ptr */
624 static krwlock_t mptsas_global_rwlock;

626 static kmutex_t mptsas_log_mutex;
627 static char mptsas_log_buf[256];
628 _NOTE(MUTEX_PROTECTS_DATA(mptsas_log_mutex, mptsas_log_buf))

630 static mptsas_t *mptsas_head, *mptsas_tail;
631 static clock_t mptsas_scsi_watchdog_tick;
632 static clock_t mptsas_tick;
633 static timeout_id_t mptsas_reset_watch;
634 static timeout_id_t mptsas_timeout_id;
635 static int mptsas_timeouts_enabled = 0;

637 /*
638  * Maximum number of MSI-X interrupts any instance of mptsas3 can use.
639  * Note that if you want to increase this you may have to also bump the
640  * value of ddi_msix_alloc_limit which defaults to 8.
641  * Set to zero to fall back to other interrupt types.
642  */
643 int mptsas3_max_msix_intrs = 8;

645 /*
646  * Default length for extended auto request sense buffers.
647  * All sense buffers need to be under the same alloc because there
648  * is only one common top 32bits (of 64bits) address register.
649  * Most requests only require 32 bytes, but some request >256.
650  * We use rmalloc()/rmfree() on this additional memory to manage the
651  * "extended" requests.
652  */
653 int mptsas_extreq_sense_bufsize = 256*64;

655 /*

```

```

656 * Believe that all software restrictions of having to run with DMA
657 * attributes to limit allocation to the first 4G are removed.
658 * However, this flag remains to enable quick switchback should suspicious
659 * problems emerge.
660 * Note that scsi_alloc_consistent_buf() does still adhering to allocating
661 * 32 bit addressable memory, but we can cope if that is changed now.
662 */
663 int mptsas_use_64bit_msgaddr = 1;

665 /*
666  * warlock directives
667  */
668 _NOTE(SCHEME_PROTECTS_DATA("unique per pkt", scsi_pkt \
669   mptsas_cmd NcrTableIndirect buf scsi_cdb scsi_status))
670 _NOTE(SCHEME_PROTECTS_DATA("unique per pkt", smp_pkt))
671 _NOTE(SCHEME_PROTECTS_DATA("stable data", scsi_device scsi_address))
672 _NOTE(SCHEME_PROTECTS_DATA("No Mutex Needed", mptsas_tgt_private))
673 _NOTE(SCHEME_PROTECTS_DATA("No Mutex Needed", scsi_hba_tran::tran_tgt_private))

675 /*
676  * SM - HBA statics
677  */
678 char    *mptsas_driver_rev = MPTSAS_MOD_STRING;

680 #ifdef MPTSAS_DEBUG
681 void debug_enter(char *);
682 #endif

684 /*
685  * Notes:
686  * - scsi_hba_init(9F) initializes SCSI HBA modules
687  * - must call scsi_hba_fini(9F) if modload() fails
688  */
689 int
690 _init(void)
691 {
692     int status;
693     /* CONSTCOND */
694     ASSERT(NO_COMPETING_THREADS);

696     NDBG0(("_init"));

698     status = ddi_soft_state_init(&mptsas3_state, MPTSAS_SIZE,
699     MPTSAS_INITIAL_SOFT_SPACE);
700     if (status != 0) {
701         return (status);
702     }

704     if ((status = scsi_hba_init(&modlinkage)) != 0) {
705         ddi_soft_state_fini(&mptsas3_state);
706         return (status);
707     }

709     mutex_init(&mptsas_global_mutex, NULL, MUTEX_DRIVER, NULL);
710     rw_init(&mptsas_global_rwlock, NULL, RW_DRIVER, NULL);
711     mutex_init(&mptsas_log_mutex, NULL, MUTEX_DRIVER, NULL);

713     if ((status = mod_install(&modlinkage)) != 0) {
714         mutex_destroy(&mptsas_log_mutex);
715         rw_destroy(&mptsas_global_rwlock);
716         mutex_destroy(&mptsas_global_mutex);
717         ddi_soft_state_fini(&mptsas3_state);
718         scsi_hba_fini(&modlinkage);
719     }

721     return (status);

```

```

722 }

724 /*
725  * Notes:
726  * - scsi_hba_fini(9F) uninitializes SCSI HBA modules
727  */
728 int
729 _fini(void)
730 {
731     int status;
732     /* CONSTCOND */
733     ASSERT(NO_COMPETING_THREADS);

735     NDBG0(("_fini"));

737     if ((status = mod_remove(&modlinkage)) == 0) {
738         ddi_soft_state_fini(&mptsas3_state);
739         scsi_hba_fini(&modlinkage);
740         mutex_destroy(&mptsas_global_mutex);
741         rw_destroy(&mptsas_global_rwlock);
742         mutex_destroy(&mptsas_log_mutex);
743     }
744     return (status);
745 }

747 /*
748  * The loadable-module _info(9E) entry point
749  */
750 int
751 _info(struct modinfo *modinfo)
752 {
753     /* CONSTCOND */
754     ASSERT(NO_COMPETING_THREADS);
755     NDBG0(("mptsas _info"));

757     return (mod_info(&modlinkage, modinfo));
758 }

760 static int
761 mptsas_target_eval_devhdl(const void *op, void *arg)
762 {
763     uint16_t dh = *(uint16_t *)arg;
764     const mptsas_target_t *tp = op;

766     return ((int)tp->m_devhdl - (int)dh);
767 }

769 static int
770 mptsas_target_eval_slot(const void *op, void *arg)
771 {
772     mptsas_led_control_t *lcp = arg;
773     const mptsas_target_t *tp = op;

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

778     return ((int)tp->m_slot_num - (int)lcp->Slot);
779 }

781 static int
782 mptsas_target_eval_nowwn(const void *op, void *arg)
783 {
784     uint8_t phy = *(uint8_t *)arg;
785     const mptsas_target_t *tp = op;

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

```

```

788         return (-1);
790     return ((int)tp->m_phynum - (int)phy);
791 }
793 static int
794 mptsas_smp_eval_devhdl(const void *op, void *arg)
795 {
796     uint16_t dh = *(uint16_t *)arg;
797     const mptsas_smp_t *sp = op;
799     return ((int)sp->m_devhdl - (int)dh);
800 }
802 static uint64_t
803 mptsas_target_addr_hash(const void *tp)
804 {
805     const mptsas_target_addr_t *tap = tp;
807     return ((tap->mta_wwn & 0xffffffffFULL) |
808         ((uint64_t)tap->mta_phymask << 48));
809 }
811 static int
812 mptsas_target_addr_cmp(const void *a, const void *b)
813 {
814     const mptsas_target_addr_t *aap = a;
815     const mptsas_target_addr_t *bap = b;
817     if (aap->mta_wwn < bap->mta_wwn)
818         return (-1);
819     if (aap->mta_wwn > bap->mta_wwn)
820         return (1);
821     return ((int)bap->mta_phymask - (int)aap->mta_phymask);
822 }
824 static void
825 mptsas_target_free(void *op)
826 {
827     kmem_free(op, sizeof (mptsas_target_t));
828 }
830 static void
831 mptsas_smp_free(void *op)
832 {
833     kmem_free(op, sizeof (mptsas_smp_t));
834 }
836 static void
837 mptsas_destroy_hashes(mptsas_t *mpt)
838 {
839     mptsas_target_t *tp;
840     mptsas_smp_t *sp;
842     for (tp = rehash_first(mpt->m_targets); tp != NULL;
843         tp = rehash_next(mpt->m_targets, tp)) {
844         mutex_destroy(&tp->m_t_mutex);
845         rehash_remove(mpt->m_targets, tp);
846     }
847     for (sp = rehash_first(mpt->m_smp_targets); sp != NULL;
848         sp = rehash_next(mpt->m_smp_targets, sp)) {
849         rehash_remove(mpt->m_smp_targets, sp);
850     }
851     rehash_destroy(mpt->m_targets);
852     rehash_destroy(mpt->m_smp_targets);
853     mpt->m_targets = NULL;

```

```

854     mpt->m_smp_targets = NULL;
855 }
857 static int
858 mptsas_iport_attach(dev_info_t *dip, ddi_attach_cmd_t cmd)
859 {
860     dev_info_t      *pdip;
861     mptsas_t        *mpt;
862     scsi_hba_tran_t *hba_tran;
863     char             *iport = NULL;
864     char             phymask[MPTSAS_MAX_PHYS];
865     mptsas_phymask_t phy_mask = 0;
866     int              dynamic_port = 0;
867     uint32_t         page_address;
868     char             initiator_wwnstr[MPTSAS_WWN_STRLEN];
869     int              rval = DDI_FAILURE;
870     int              i = 0;
871     uint8_t          numphys = 0;
872     uint8_t          phy_id;
873     uint8_t          phy_port = 0;
874     uint16_t         attached_devhdl = 0;
875     uint32_t         dev_info;
876     uint64_t         attached_sas_wwn;
877     uint16_t         dev_hdl;
878     uint16_t         pdev_hdl;
879     uint16_t         bay_num;
880     char             enclosure, io_flags;
881     char             attached_wwnstr[MPTSAS_WWN_STRLEN];
882     /* CONSTCOND */
883     ASSERT(NO_COMPETING_THREADS);
885     switch (cmd) {
886     case DDI_ATTACH:
887         break;
889     case DDI_RESUME:
890         /*
891          * If this a scsi-iport node, nothing to do here.
892          */
893         return (DDI_SUCCESS);
895     default:
896         return (DDI_FAILURE);
897     }
899     pdip = ddi_get_parent(dip);
901     if ((hba_tran = ndi_flavorv_get(pdip, SCSI_FLAVOR_SCSI_DEVICE)) ==
902         NULL) {
903         cmn_err(CE_WARN, "Failed attach iport because fail to "
904             "get tran vector for the HBA node");
905         return (DDI_FAILURE);
906     }
908     mpt = TRAN2MPT(hba_tran);
909     ASSERT(mpt != NULL);
910     if (mpt == NULL)
911         return (DDI_FAILURE);
913     if ((hba_tran = ndi_flavorv_get(dip, SCSI_FLAVOR_SCSI_DEVICE)) ==
914         NULL) {
915         mptsas_log(mpt, CE_WARN, "Failed attach iport because fail to "
916             "get tran vector for the iport node");
917         return (DDI_FAILURE);
918     }

```



```

920  /*
921   * Overwrite parent's tran_hba_private to iport's tran vector
922   */
923  hba_tran->tran_hba_private = mpt;

925  ddi_report_dev(dip);

927  /*
928   * Get SAS address for initiator port according dev_handle
929   */
930  iport = ddi_get_name_addr(dip);
931  if (iport && strncmp(iport, "v0", 2) == 0) {
932      if (ddi_prop_update_int(DDI_DEV_T_NONE, dip,
933          MPTSAS_VIRTUAL_PORT, 1) !=
934          DDI_PROP_SUCCESS) {
935          (void) ddi_prop_remove(DDI_DEV_T_NONE, dip,
936              MPTSAS_VIRTUAL_PORT);
937          mptsas_log(mpt, CE_WARN, "mptsas virtual port "
938              "prop update failed");
939          return (DDI_FAILURE);
940      }
941      return (DDI_SUCCESS);
942  }

944  mutex_enter(&mpt->m_mutex);
945  for (i = 0; i < MPTSAS_MAX_PHYS; i++) {
946      bzero(phymask, sizeof (phymask));
947      (void) sprintf(phymask,
948          "%x", mpt->m_phy_info[i].phy_mask);
949      if (strcmp(phymask, iport) == 0) {
950          break;
951      }
952  }

954  if (i == MPTSAS_MAX_PHYS) {
955      mptsas_log(mpt, CE_WARN, "Failed attach port %s because port "
956          "seems not exist", iport);
957      mutex_exit(&mpt->m_mutex);
958      return (DDI_FAILURE);
959  }

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

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

968  /*
969   * Update PHY info for smhba
970   */
971  if (mptsas_smhba_phy_init(mpt)) {
972      mutex_exit(&mpt->m_mutex);
973      mptsas_log(mpt, CE_WARN, "mptsas phy update "
974          "failed");
975      return (DDI_FAILURE);
976  }

978  mutex_exit(&mpt->m_mutex);

980  numphys = 0;
981  for (i = 0; i < MPTSAS_MAX_PHYS; i++) {
982      if ((phy_mask >> i) & 0x01) {
983          numphys++;
984      }
985  }

```

```

987  bzero(initiator_wnnstr, sizeof (initiator_wnnstr));
988  (void) sprintf(initiator_wnnstr, "%016"PRIx64,
989      mpt->un.m_base_wwid);

991  if (ddi_prop_update_string(DDI_DEV_T_NONE, dip,
992      SCSI_ADDR_PROP_INITIATOR_PORT, initiator_wnnstr) !=
993      DDI_PROP_SUCCESS) {
994      (void) ddi_prop_remove(DDI_DEV_T_NONE,
995          dip, SCSI_ADDR_PROP_INITIATOR_PORT);
996      mptsas_log(mpt, CE_WARN, "mptsas Initiator port "
997          "prop update failed");
998      return (DDI_FAILURE);
999  }

1000  if (ddi_prop_update_int(DDI_DEV_T_NONE, dip,
1001      MPTSAS_NUM_PHYS, numphys) !=
1002      DDI_PROP_SUCCESS) {
1003      (void) ddi_prop_remove(DDI_DEV_T_NONE, dip, MPTSAS_NUM_PHYS);
1004      return (DDI_FAILURE);
1005  }

1007  if (ddi_prop_update_int(DDI_DEV_T_NONE, dip,
1008      "phymask", phy_mask) !=
1009      DDI_PROP_SUCCESS) {
1010      (void) ddi_prop_remove(DDI_DEV_T_NONE, dip, "phymask");
1011      mptsas_log(mpt, CE_WARN, "mptsas phy mask "
1012          "prop update failed");
1013      return (DDI_FAILURE);
1014  }

1016  if (ddi_prop_update_int(DDI_DEV_T_NONE, dip,
1017      "dynamic-port", dynamic_port) !=
1018      DDI_PROP_SUCCESS) {
1019      (void) ddi_prop_remove(DDI_DEV_T_NONE, dip, "dynamic-port");
1020      mptsas_log(mpt, CE_WARN, "mptsas dynamic port "
1021          "prop update failed");
1022      return (DDI_FAILURE);
1023  }

1024  if (ddi_prop_update_int(DDI_DEV_T_NONE, dip,
1025      MPTSAS_VIRTUAL_PORT, 0) !=
1026      DDI_PROP_SUCCESS) {
1027      (void) ddi_prop_remove(DDI_DEV_T_NONE, dip,
1028          MPTSAS_VIRTUAL_PORT);
1029      mptsas_log(mpt, CE_WARN, "mptsas virtual port "
1030          "prop update failed");
1031      return (DDI_FAILURE);
1032  }

1033  mptsas_smhba_set_all_phy_props(mpt, dip, numphys, phy_mask,
1034      &attached_devhdl);

1036  mutex_enter(&mpt->m_mutex);
1037  page_address = (MPI2_SAS_DEVICE_PGAD_FORM_HANDLE &
1038      MPI2_SAS_DEVICE_PGAD_FORM_MASK) | (uint32_t)attached_devhdl;
1039  rval = mptsas_get_sas_device_page0(mpt, page_address, &dev_hdl,
1040      &attached_sas_wnn, &dev_info, &phy_port, &phy_id,
1041      &pdev_hdl, &bay_num, &enclosure, &io_flags);
1042  if (rval != DDI_SUCCESS) {
1043      mptsas_log(mpt, CE_WARN,
1044          "Failed to get device page0 for handle:%d",
1045          attached_devhdl);
1046      mutex_exit(&mpt->m_mutex);
1047      return (DDI_FAILURE);
1048  }

1050  for (i = 0; i < MPTSAS_MAX_PHYS; i++) {
1051      bzero(phymask, sizeof (phymask));

```

```

1052         (void) sprintf(phymask, "%x", mpt->m_phy_info[i].phy_mask);
1053         if (strcmp(phymask, iport) == 0) {
1054             (void) sprintf(&mpt->m_phy_info[i].smhba_info.path[0],
1055                 "%x",
1056                 mpt->m_phy_info[i].phy_mask);
1057         }
1058     }
1059     mutex_exit(&mpt->m_mutex);

1061     bzero(attached_wnstr, sizeof (attached_wnstr));
1062     (void) sprintf(attached_wnstr, "w%016"PRIx64,
1063         attached_sas_wn);
1064     if (ddi_prop_update_string(DDI_DEV_T_NONE, dip,
1065         SCSI_ADDR_PROP_ATTACHED_PORT, attached_wnstr) !=
1066         DDI_PROP_SUCCESS) {
1067         (void) ddi_prop_remove(DDI_DEV_T_NONE,
1068             dip, SCSI_ADDR_PROP_ATTACHED_PORT);
1069         return (DDI_FAILURE);
1070     }

1072     /* Create kstats for each phy on this iport */

1074     mptsas_create_phy_stats(mpt, iport, dip);

1076     /*
1077      * register sas hba iport with mdi (MPxIO/vhci)
1078      */
1079     if (mdi_phci_register(MDI_HCI_CLASS_SCSI,
1080         dip, 0) == MDI_SUCCESS) {
1081         mpt->m_mpxio_enable = TRUE;
1082     }
1083     return (DDI_SUCCESS);
1084 }

1086 /*
1087  * Notes:
1088  *   Set up all device state and allocate data structures,
1089  *   mutexes, condition variables, etc. for device operation.
1090  *   Add interrupts needed.
1091  *   Return DDI_SUCCESS if device is ready, else return DDI_FAILURE.
1092  */
1093 static int
1094 mptsas_attach(dev_info_t *dip, ddi_attach_cmd_t cmd)
1095 {
1096     mptsas_t          *mpt = NULL;
1097     int                instance, i, j;
1098     int                q_thread_num;
1099     char               map_setup = 0;
1100     char               config_setup = 0;
1101     char               hba_attach_setup = 0;
1102     char               smp_attach_setup = 0;
1103     char               mutex_init_done = 0;
1104     char               event_taskq_create = 0;
1105     char               dr_taskq_create = 0;
1106     char               doneq_thread_create = 0;
1107     char               txwq_thread_create = 0;
1108     char               added_watchdog = 0;
1109     scsi_hba_tran_t    *hba_tran;
1110     uint_t             mem_bar = MEM_SPACE;
1111     int                rval = DDI_FAILURE;

1113     /* CONSTCOND */
1114     ASSERT(NO_COMPETING_THREADS);

1116     if (scsi_hba_iport_unit_address(dip)) {
1117         return (mptsas_iport_attach(dip, cmd));

```

```

1118     }

1120     switch (cmd) {
1121     case DDI_ATTACH:
1122         break;

1124     case DDI_RESUME:
1125         if ((hba_tran = ddi_get_driver_private(dip)) == NULL)
1126             return (DDI_FAILURE);

1128         mpt = TRAN2MPT(hba_tran);

1130         if (!mpt) {
1131             return (DDI_FAILURE);
1132         }

1134         /*
1135          * Reset hardware and softc to "no outstanding commands"
1136          * Note that a check condition can result on first command
1137          * to a target.
1138          */
1139         mutex_enter(&mpt->m_mutex);

1141         /*
1142          * raise power.
1143          */
1144         if (mpt->m_options & MPTSAS_OPT_PM) {
1145             mutex_exit(&mpt->m_mutex);
1146             (void) pm_busy_component(dip, 0);
1147             rval = pm_power_has_changed(dip, 0, PM_LEVEL_D0);
1148             if (rval == DDI_SUCCESS) {
1149                 mutex_enter(&mpt->m_mutex);
1150             } else {
1151                 /*
1152                  * The pm_raise_power() call above failed,
1153                  * and that can only occur if we were unable
1154                  * to reset the hardware. This is probably
1155                  * due to unhealthy hardware, and because
1156                  * important filesystems (such as the root
1157                  * filesystem) could be on the attached disks,
1158                  * it would not be a good idea to continue,
1159                  * as we won't be entirely certain we are
1160                  * writing correct data. So we panic() here
1161                  * to not only prevent possible data corruption,
1162                  * but to give developers or end users a hope
1163                  * of identifying and correcting any problems.
1164                  */
1165                 fm_panic("mptsas could not reset hardware "
1166                     "during resume");
1167             }
1168         }

1170         mpt->m_suspended = 0;

1172         /*
1173          * Reinitialize ioc
1174          */
1175         mpt->m_softstate |= MPTSAS_SS_MSG_UNIT_RESET;
1176         if (mptsas_init_chip(mpt, FALSE) == DDI_FAILURE) {
1177             mutex_exit(&mpt->m_mutex);
1178             if (mpt->m_options & MPTSAS_OPT_PM) {
1179                 (void) pm_idle_component(dip, 0);
1180             }
1181             fm_panic("mptsas init chip fail during resume");
1182         }
1183         /*

```

```

1184     * mptsas_update_driver_data needs interrupts so enable them
1185     * first.
1186     */
1187     MPTSAS_ENABLE_INTR(mpt);
1188     mptsas_update_driver_data(mpt);
1189
1190     /* start requests, if possible */
1191     mptsas_restart_hba(mpt);
1192
1193     mutex_exit(&mpt->m_mutex);
1194
1195     /*
1196     * Restart watch thread
1197     */
1198     mutex_enter(&mptsas_global_mutex);
1199     if (mptsas_timeout_id == 0) {
1200         mptsas_timeout_id = timeout(mptsas_watch, NULL,
1201             mptsas_tick);
1202         mptsas_timeouts_enabled = 1;
1203     }
1204     mutex_exit(&mptsas_global_mutex);
1205
1206     /* report idle status to pm framework */
1207     if (mpt->m_options & MPTSAS_OPT_PM) {
1208         (void) pm_idle_component(dip, 0);
1209     }
1210
1211     return (DDI_SUCCESS);
1212
1213 default:
1214     return (DDI_FAILURE);
1215 }
1216
1217 instance = ddi_get_instance(dip);
1218
1219 /*
1220 * Allocate softc information.
1221 */
1222 if (ddi_soft_state_zalloc(mptsas3_state, instance) != DDI_SUCCESS) {
1223     mptsas_log(NULL, CE_WARN,
1224         "mptsas%d: cannot allocate soft state", instance);
1225     goto fail;
1226 }
1227
1228 mpt = ddi_get_soft_state(mptsas3_state, instance);
1229
1230 if (mpt == NULL) {
1231     mptsas_log(NULL, CE_WARN,
1232         "mptsas%d: cannot get soft state", instance);
1233     goto fail;
1234 }
1235
1236 /* Indicate that we are 'sizeof (scsi_*(9S))' clean. */
1237 scsi_size_clean(dip);
1238
1239 mpt->m_dip = dip;
1240 mpt->m_instance = instance;
1241
1242 /* Make a per-instance copy of the structures */
1243 mpt->m_io_dma_attr = mptsas_dma_attrs64;
1244 if (mptsas_use_64bit_msgaddr) {
1245     mpt->m_msg_dma_attr = mptsas_dma_attrs64;
1246 } else {
1247     mpt->m_msg_dma_attr = mptsas_dma_attrs;
1248 }
1249 }

```

```

1250     mpt->m_reg_acc_attr = mptsas_dev_attr;
1251     mpt->m_dev_acc_attr = mptsas_dev_attr;
1252
1253     /*
1254     * Round down the arq sense buffer size to nearest 16 bytes.
1255     */
1256     mpt->m_req_sense_size = EXTCMDSTATUS_SIZE;
1257
1258     /*
1259     * Initialize FMA
1260     */
1261     mpt->m_fm_capabilities = ddi_getprop(DDI_DEV_T_ANY, mpt->m_dip,
1262         DDI_PROP_CANSLEEP | DDI_PROP_DONTPASS, "fm-capable",
1263         DDI_FM_EREPORCAPABLE | DDI_FM_ACCCHKCAPABLE |
1264         DDI_FM_DMACHKCAPABLE | DDI_FM_ERRRCBCAPABLE);
1265
1266     mptsas_fm_init(mpt);
1267
1268     if (mptsas_alloc_handshake_msg(mpt,
1269         sizeof (Mpi2SCSITaskManagementRequest_t)) == DDI_FAILURE) {
1270         mptsas_log(mpt, CE_WARN, "cannot initialize handshake msg.");
1271         goto fail;
1272     }
1273
1274     /*
1275     * Setup configuration space
1276     */
1277     if (mptsas_config_space_init(mpt) == FALSE) {
1278         mptsas_log(mpt, CE_WARN, "mptsas_config_space_init failed");
1279         goto fail;
1280     }
1281     config_setup++;
1282
1283     if (ddi_regs_map_setup(dip, mem_bar, (caddr_t *)&mpt->m_reg,
1284         0, 0, &mpt->m_reg_acc_attr, &mpt->m_datap) != DDI_SUCCESS) {
1285         mptsas_log(mpt, CE_WARN, "map setup failed");
1286         goto fail;
1287     }
1288     map_setup++;
1289
1290     /*
1291     * A taskq is created for dealing with the event handler
1292     */
1293     if ((mpt->m_event_taskq = ddi_taskq_create(dip, "mptsas_event_taskq",
1294         1, TASKQ_DEFAULTPRI, 0)) == NULL) {
1295         mptsas_log(mpt, CE_NOTE, "ddi_taskq_create failed");
1296         goto fail;
1297     }
1298     event_taskq_create++;
1299
1300     /*
1301     * A taskq is created for dealing with dr events
1302     */
1303     if ((mpt->m_dr_taskq = ddi_taskq_create(dip,
1304         "mptsas_dr_taskq",
1305         1, TASKQ_DEFAULTPRI, 0)) == NULL) {
1306         mptsas_log(mpt, CE_NOTE, "ddi_taskq_create for discovery "
1307             "failed");
1308         goto fail;
1309     }
1310     dr_taskq_create++;
1311
1312     cv_init(&mpt->m_qthread_cv, NULL, CV_DRIVER, NULL);
1313     mutex_init(&mpt->m_qthread_mutex, NULL, MUTEX_DRIVER, NULL);
1314
1315     i = ddi_prop_get_int(DDI_DEV_T_ANY, dip,

```

```

1316         0, "mptsas_enable_txdwq_prop", NCPUS > 1);
1317     if (i) {
1318         mpt->m_txdwq_thread_n = NUM_TX_WAITQ;
1319         mpt->m_txdwq_enabled = FALSE;
1320         if (ddi_prop_get_int(DDI_DEV_T_ANY, dip,
1321             0, "mptsas_allow_txdwq_jumping", 0)) {
1322             mpt->m_txdwq_allow_q_jumping = TRUE;
1323         }
1324         i = ddi_prop_get_int(DDI_DEV_T_ANY, dip,
1325             0, "mptsas_txdwq_threshold_prop", 80000);
1326         mpt->m_txdwq_thread_threshold = (uint16_t)i;
1327     } else {
1328         mpt->m_txdwq_thread_n = 0;
1329         mpt->m_txdwq_enabled = FALSE;
1330     }
1331
1332     if (mpt->m_txdwq_thread_n) {
1333         mutex_enter(&mpt->m_qthread_mutex);
1334         for (j = 0; j < NUM_TX_WAITQ; j++) {
1335             mutex_init(&mpt->m_tx_waitq[j].txdwq_mutex, NULL,
1336                 MUTEX_DRIVER,
1337                 NULL);
1338             cv_init(&mpt->m_tx_waitq[j].txdwq_cv, NULL, CV_DRIVER,
1339                 NULL);
1340             cv_init(&mpt->m_tx_waitq[j].txdwq_drain_cv, NULL,
1341                 CV_DRIVER, NULL);
1342             mpt->m_tx_waitq[j].txdwq_active = TRUE;
1343             mpt->m_tx_waitq[j].txdwq_draining = FALSE;
1344             mpt->m_tx_waitq[j].txdwq_cmdq = NULL;
1345             mpt->m_tx_waitq[j].txdwq_qtail =
1346                 &mpt->m_tx_waitq[j].txdwq_cmdq;
1347             mutex_enter(&mpt->m_tx_waitq[j].txdwq_mutex);
1348             mpt->m_tx_waitq[j].arg.mpt = mpt;
1349             mpt->m_tx_waitq[j].arg.t = j;
1350             mpt->m_tx_waitq[j].txdwq_threadp =
1351                 thread_create(NULL, 0, mptsas_tx_waitq_thread,
1352                 &mpt->m_tx_waitq[j].arg,
1353                 0, &p0, TS_RUN, maxclsyspri - 10);
1354             mutex_exit(&mpt->m_tx_waitq[j].txdwq_mutex);
1355         }
1356         mutex_exit(&mpt->m_qthread_mutex);
1357         txdwq_thread_create++;
1358     }
1359
1360     mpt->m_doneq_thread_threshold = ddi_prop_get_int(DDI_DEV_T_ANY, dip,
1361         0, "mptsas_doneq_thread_threshold_prop", 10);
1362     mpt->m_doneq_length_threshold = ddi_prop_get_int(DDI_DEV_T_ANY, dip,
1363         0, "mptsas_doneq_length_threshold_prop", 8);
1364     mpt->m_doneq_thread_n = ddi_prop_get_int(DDI_DEV_T_ANY, dip,
1365         0, "mptsas_doneq_thread_n_prop", min(NCPUS, 8));
1366
1367     if (mpt->m_doneq_thread_n) {
1368         mutex_enter(&mpt->m_qthread_mutex);
1369         mpt->m_doneq_thread_id =
1370             kmem_zalloc(sizeof (mptsas_doneq_thread_list_t)
1371                 * mpt->m_doneq_thread_n, KM_SLEEP);
1372
1373         for (j = 0; j < mpt->m_doneq_thread_n; j++) {
1374             cv_init(&mpt->m_doneq_thread_id[j].cv, NULL,
1375                 CV_DRIVER, NULL);
1376             mutex_init(&mpt->m_doneq_thread_id[j].mutex, NULL,
1377                 MUTEX_DRIVER, NULL);
1378             mutex_enter(&mpt->m_doneq_thread_id[j].mutex);
1379             mpt->m_doneq_thread_id[j].flag |=
1380                 MPTSAS_DONEQ_THREAD_ACTIVE;
1381             mpt->m_doneq_thread_id[j].arg.mpt = mpt;

```

```

1382         mpt->m_doneq_thread_id[j].arg.t = j;
1383         mpt->m_doneq_thread_id[j].threadp =
1384             thread_create(NULL, 0, mptsas_doneq_thread,
1385                 &mpt->m_doneq_thread_id[j].arg,
1386                 0, &p0, TS_RUN, maxclsyspri - 10);
1387         mpt->m_doneq_thread_id[j].dlist.dl_tail =
1388             &mpt->m_doneq_thread_id[j].dlist.dl_q;
1389         mutex_exit(&mpt->m_doneq_thread_id[j].mutex);
1390     }
1391     mutex_exit(&mpt->m_qthread_mutex);
1392     doneq_thread_create++;
1393 }
1394
1395 /*
1396  * Disable hardware interrupt since we're not ready to
1397  * handle it yet.
1398  */
1399 MPTSAS_DISABLE_INTR(mpt);
1400
1401 /*
1402  * Initialize mutex used in interrupt handler.
1403  * We don't support hi-level so the mutex's are all adaptive
1404  * and we don't want to register the interrupts until we get
1405  * the chip type information from init_chip() below.
1406  * Otherwise we would use DDI_INTR_PRI(mpt->m_intr_pri)
1407  * rather than NULL in the mutex_init() calls.
1408  */
1409 mutex_init(&mpt->m_mutex, NULL, MUTEX_DRIVER, NULL);
1410 mutex_init(&mpt->m_passthru_mutex, NULL, MUTEX_DRIVER, NULL);
1411 for (i = 0; i < MPTSAS_MAX_PHYS; i++) {
1412     mutex_init(&mpt->m_phy_info[i].smhba_info.phy_mutex,
1413         NULL, MUTEX_DRIVER, NULL);
1414 }
1415
1416 cv_init(&mpt->m_cv, NULL, CV_DRIVER, NULL);
1417 cv_init(&mpt->m_passthru_cv, NULL, CV_DRIVER, NULL);
1418 cv_init(&mpt->m_fw_cv, NULL, CV_DRIVER, NULL);
1419 cv_init(&mpt->m_config_cv, NULL, CV_DRIVER, NULL);
1420 cv_init(&mpt->m_fw_diag_cv, NULL, CV_DRIVER, NULL);
1421 mutex_init_done++;
1422
1423 mutex_enter(&mpt->m_mutex);
1424 /*
1425  * Initialize power management component
1426  */
1427 if (mpt->m_options & MPTSAS_OPT_PM) {
1428     if (mptsas_init_pm(mpt)) {
1429         mutex_exit(&mpt->m_mutex);
1430         mptsas_log(mpt, CE_WARN, "mptsas pm initialization "
1431             "failed");
1432         goto fail;
1433     }
1434 }
1435
1436 /*
1437  * Initialize chip using Message Unit Reset, if allowed
1438  */
1439 mpt->m_softstate |= MPTSAS_SS_MSG_UNIT_RESET;
1440 if (mptsas_init_chip(mpt, TRUE) == DDI_FAILURE) {
1441     mutex_exit(&mpt->m_mutex);
1442     mptsas_log(mpt, CE_WARN, "mptsas chip initialization failed");
1443     goto fail;
1444 }
1445
1446 /*
1447  * Fill in the phy_info structure and get the base WWID

```

```

1448     */
1449     if (mptsas_get_manufacture_page5(mpt) == DDI_FAILURE) {
1450         mptsas_log(mpt, CE_WARN,
1451             "mptsas_get_manufacture_page5 failed!");
1452         goto fail;
1453     }
1454
1455     if (mptsas_get_sas_io_unit_page_hndshk(mpt)) {
1456         mptsas_log(mpt, CE_WARN,
1457             "mptsas_get_sas_io_unit_page_hndshk failed!");
1458         goto fail;
1459     }
1460
1461     if (mptsas_get_manufacture_page0(mpt) == DDI_FAILURE) {
1462         mptsas_log(mpt, CE_WARN,
1463             "mptsas_get_manufacture_page0 failed!");
1464         goto fail;
1465     }
1466
1467     /*
1468     * If we only have one interrupt the default for doneq_thread_threshold
1469     * should be 0 so that all completion processing goes to the threads.
1470     * Only change it if it wasn't set from .conf file.
1471     */
1472     if (mpt->m_doneq_thread_n != 0 &&
1473         ddi_prop_exists(DDI_DEV_T_ANY, dip,
1474             0, "mptsas_doneq_length_threshold_prop") == 0 &&
1475         mpt->m_intr_cnt == 1) {
1476         mpt->m_doneq_length_threshold = 0;
1477     }
1478
1479     mutex_exit(&mpt->m_mutex);
1480
1481     /*
1482     * Register the iport for multiple port HBA
1483     */
1484     mptsas_iport_register(mpt);
1485
1486     /*
1487     * initialize SCSI HBA transport structure
1488     */
1489     if (mptsas_hba_setup(mpt) == FALSE)
1490         goto fail;
1491     hba_attach_setup++;
1492
1493     if (mptsas_smp_setup(mpt) == FALSE)
1494         goto fail;
1495     smp_attach_setup++;
1496
1497     if (mptsas_cache_create(mpt) == FALSE)
1498         goto fail;
1499
1500     mpt->m_scsi_reset_delay = ddi_prop_get_int(DDI_DEV_T_ANY,
1501         dip, 0, "scsi-reset-delay", SCSI_DEFAULT_RESET_DELAY);
1502     if (mpt->m_scsi_reset_delay == 0) {
1503         mptsas_log(mpt, CE_NOTE,
1504             "scsi_reset_delay of 0 is not recommended,"
1505             " resetting to SCSI_DEFAULT_RESET_DELAY\n");
1506         mpt->m_scsi_reset_delay = SCSI_DEFAULT_RESET_DELAY;
1507     }
1508
1509     /*
1510     * Initialize the wait and done FIFO queue
1511     */
1512     mpt->m_dlist.dl_tail = &mpt->m_dlist.dl_q;

```

```

1514     mpt->m_waitqtail = &mpt->m_waitq;
1515
1516     /*
1517     * ioc cmd queue initialize
1518     */
1519     mpt->m_ioc_event_cmdtail = &mpt->m_ioc_event_cmdq;
1520     mpt->m_dev_handle = 0xFFFFF;
1521
1522     MPTSAS_ENABLE_INTR(mpt);
1523
1524     /*
1525     * enable event notification
1526     */
1527     mutex_enter(&mpt->m_mutex);
1528     if (mptsas_ioc_enable_event_notification(mpt)) {
1529         mutex_exit(&mpt->m_mutex);
1530         goto fail;
1531     }
1532     mutex_exit(&mpt->m_mutex);
1533
1534     /*
1535     * used for mptsas_watch
1536     */
1537     mptsas_list_add(mpt);
1538
1539     mutex_enter(&mptsas_global_mutex);
1540     if (mptsas_timeouts_enabled == 0) {
1541         mptsas_scsi_watchdog_tick = ddi_prop_get_int(DDI_DEV_T_ANY,
1542             dip, 0, "scsi-watchdog-tick", DEFAULT_WD_TICK);
1543
1544         mptsas_tick = mptsas_scsi_watchdog_tick *
1545             drv_usectohz((clock_t)1000000);
1546
1547         mptsas_timeout_id = timeout(mptsas_watch, NULL, mptsas_tick);
1548         mptsas_timeouts_enabled = 1;
1549     }
1550     mutex_exit(&mptsas_global_mutex);
1551     added_watchdog++;
1552
1553     /*
1554     * Initialize PHY info for smhba.
1555     * This requires watchdog to be enabled otherwise if interrupts
1556     * don't work the system will hang.
1557     */
1558     if (mptsas_smhba_setup(mpt)) {
1559         mptsas_log(mpt, CE_WARN, "mptsas phy initialization "
1560             "failed");
1561         goto fail;
1562     }
1563
1564     /* Check all dma handles allocated in attach */
1565     if ((mptsas_check_dma_handle(mpt->m_dma_req_frame_hdl)
1566         != DDI_SUCCESS) ||
1567         (mptsas_check_dma_handle(mpt->m_dma_req_sense_hdl)
1568         != DDI_SUCCESS) ||
1569         (mptsas_check_dma_handle(mpt->m_dma_reply_frame_hdl)
1570         != DDI_SUCCESS) ||
1571         (mptsas_check_dma_handle(mpt->m_dma_free_queue_hdl)
1572         != DDI_SUCCESS) ||
1573         (mptsas_check_dma_handle(mpt->m_dma_post_queue_hdl)
1574         != DDI_SUCCESS) ||
1575         (mptsas_check_dma_handle(mpt->m_hshk_dma_hdl)
1576         != DDI_SUCCESS)) {
1577         goto fail;
1578     }

```

```

1580      /* Check all acc handles allocated in attach */
1581      if ((mptsas_check_acc_handle(mpt->m_datap) != DDI_SUCCESS) ||
1582          (mptsas_check_acc_handle(mpt->m_acc_req_frame_hdl)
1583              != DDI_SUCCESS) ||
1584          (mptsas_check_acc_handle(mpt->m_acc_req_sense_hdl)
1585              != DDI_SUCCESS) ||
1586          (mptsas_check_acc_handle(mpt->m_acc_reply_frame_hdl)
1587              != DDI_SUCCESS) ||
1588          (mptsas_check_acc_handle(mpt->m_acc_free_queue_hdl)
1589              != DDI_SUCCESS) ||
1590          (mptsas_check_acc_handle(mpt->m_acc_post_queue_hdl)
1591              != DDI_SUCCESS) ||
1592          (mptsas_check_acc_handle(mpt->m_hshk_acc_hdl)
1593              != DDI_SUCCESS) ||
1594          (mptsas_check_acc_handle(mpt->m_config_handle)
1595              != DDI_SUCCESS)) {
1596          goto fail;
1597      }

1599      /*
1600       * After this point, we are not going to fail the attach.
1601       */

1603      /* Print message of HBA present */
1604      ddi_report_dev(dip);

1606      /* report idle status to pm framework */
1607      if (mpt->m_options & MPTSAS_OPT_PM) {
1608          (void) pm_idle_component(dip, 0);
1609      }

1611      return (DDI_SUCCESS);

1613 fail:
1614      mptsas_log(mpt, CE_WARN, "attach failed");
1615      mptsas_fm_ereport(mpt, DDI_FM_DEVICE_NO_RESPONSE);
1616      ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_LOST);
1617      if (mpt) {
1618          /* deallocate in reverse order */
1619          if (added_watchdog) {
1620              mptsas_list_del(mpt);
1621              mutex_enter(&mptsas_global_mutex);

1623              if (mptsas_timeout_id && (mptsas_head == NULL)) {
1624                  timeout_id_t tid = mptsas_timeout_id;
1625                  mptsas_timeouts_enabled = 0;
1626                  mptsas_timeout_id = 0;
1627                  mutex_exit(&mptsas_global_mutex);
1628                  (void) untimeout(tid);
1629                  mutex_enter(&mptsas_global_mutex);
1630              }
1631              mutex_exit(&mptsas_global_mutex);
1632          }

1634      mptsas_cache_destroy(mpt);

1636      if (smp_attach_setup) {
1637          mptsas_smp_teardown(mpt);
1638      }
1639      if (hba_attach_setup) {
1640          mptsas_hba_teardown(mpt);
1641      }

1643      if (mpt->m_targets)
1644          rehash_destroy(mpt->m_targets);
1645      if (mpt->m_smp_targets)

```

```

1646          rehash_destroy(mpt->m_smp_targets);

1648      if (mpt->m_active) {
1649          mptsas_free_active_slots(mpt);
1650      }
1651      if (mpt->m_intr_cnt) {
1652          mptsas_unregister_intrs(mpt);
1653      }

1655      if (doneq_thread_create) {
1656          mutex_enter(&mpt->m_qthread_mutex);
1657          q_thread_num = mpt->m_doneq_thread_n;
1658          for (j = 0; j < q_thread_num; j++) {
1659              mutex_enter(&mpt->m_doneq_thread_id[j].mutex);
1660              mpt->m_doneq_thread_id[j].flag &=
1661                  (~MPTSAS_DONEQ_THREAD_ACTIVE);
1662              cv_signal(&mpt->m_doneq_thread_id[j].cv);
1663              mutex_exit(&mpt->m_doneq_thread_id[j].mutex);
1664          }
1665          while (mpt->m_doneq_thread_n) {
1666              cv_wait(&mpt->m_qthread_cv,
1667                  &mpt->m_qthread_mutex);
1668          }
1669          for (j = 0; j < q_thread_num; j++) {
1670              cv_destroy(&mpt->m_doneq_thread_id[j].cv);
1671              mutex_destroy(&mpt->m_doneq_thread_id[j].mutex);
1672          }
1673          kmem_free(mpt->m_doneq_thread_id,
1674              sizeof (mptsas_doneq_thread_list_t)
1675                  * q_thread_num);
1676          mutex_exit(&mpt->m_qthread_mutex);
1677      }
1678      if (txwq_thread_create) {
1679          mutex_enter(&mpt->m_qthread_mutex);
1680          q_thread_num = mpt->m_txwq_thread_n;
1681          for (j = 0; j < q_thread_num; j++) {
1682              mutex_enter(&mpt->m_tx_waitq[j].txwq_mutex);
1683              mpt->m_tx_waitq[j].txwq_active = FALSE;
1684              cv_signal(&mpt->m_tx_waitq[j].txwq_cv);
1685              mutex_exit(&mpt->m_tx_waitq[j].txwq_mutex);
1686          }
1687          while (mpt->m_txwq_thread_n) {
1688              cv_wait(&mpt->m_qthread_cv,
1689                  &mpt->m_qthread_mutex);
1690          }
1691          for (j = 0; j < q_thread_num; j++) {
1692              cv_destroy(&mpt->m_tx_waitq[j].txwq_cv);
1693              cv_destroy(&mpt->m_tx_waitq[j].txwq_drain_cv);
1694              mutex_destroy(&mpt->m_tx_waitq[j].txwq_mutex);
1695          }
1696      }
1697      if (event_taskq_create) {
1698          ddi_taskq_destroy(mpt->m_event_taskq);
1699      }
1700      if (dr_taskq_create) {
1701          ddi_taskq_destroy(mpt->m_dr_taskq);
1702      }
1703      if (mutex_init_done) {
1704          mutex_destroy(&mpt->m_qthread_mutex);
1705          mutex_destroy(&mpt->m_passthru_mutex);
1706          mutex_destroy(&mpt->m_mutex);
1707          for (i = 0; i < MPTSAS_MAX_PHYS; i++) {
1708              mutex_destroy(
1709                  &mpt->m_phy_info[i].smhba_info.phy_mutex);
1710          }
1711          cv_destroy(&mpt->m_qthread_cv);

```

```

1712         cv_destroy(&mpt->m_cv);
1713         cv_destroy(&mpt->m_passthru_cv);
1714         cv_destroy(&mpt->m_fw_cv);
1715         cv_destroy(&mpt->m_config_cv);
1716         cv_destroy(&mpt->m_fw_diag_cv);
1717     }
1718
1719     if (map_setup) {
1720         mptsas_cfg_fini(mpt);
1721     }
1722     if (config_setup) {
1723         mptsas_config_space_fini(mpt);
1724     }
1725     mptsas_free_handshake_msg(mpt);
1726     mptsas_hba_fini(mpt);
1727
1728     mptsas_fm_fini(mpt);
1729     ddi_soft_state_free(mptsas3_state, instance);
1730     ddi_prop_remove_all(dip);
1731 }
1732 return (DDI_FAILURE);
1733 }
1734
1735 static int
1736 mptsas_suspend(dev_info_t *devi)
1737 {
1738     mptsas_t      *mpt, *g;
1739     scsi_hba_tran_t *tran;
1740
1741     if (scsi_hba_iport_unit_address(devi)) {
1742         return (DDI_SUCCESS);
1743     }
1744
1745     if ((tran = ddi_get_driver_private(devi)) == NULL)
1746         return (DDI_SUCCESS);
1747
1748     mpt = TRAN2MPT(tran);
1749     if (!mpt) {
1750         return (DDI_SUCCESS);
1751     }
1752
1753     mutex_enter(&mpt->m_mutex);
1754
1755     if (mpt->m_suspended++) {
1756         mutex_exit(&mpt->m_mutex);
1757         return (DDI_SUCCESS);
1758     }
1759
1760     /*
1761      * Cancel timeout threads for this mpt
1762      */
1763     if (mpt->m_quiesce_timeout) {
1764         timeout_id_t tid = mpt->m_quiesce_timeout;
1765         mpt->m_quiesce_timeout = 0;
1766         mutex_exit(&mpt->m_mutex);
1767         (void) untimeout(tid);
1768         mutex_enter(&mpt->m_mutex);
1769     }
1770
1771     if (mpt->m_restart_cmd_timeout) {
1772         timeout_id_t tid = mpt->m_restart_cmd_timeout;
1773         mpt->m_restart_cmd_timeout = 0;
1774         mutex_exit(&mpt->m_mutex);
1775         (void) untimeout(tid);
1776         mutex_enter(&mpt->m_mutex);
1777     }

```

```

1779     mutex_exit(&mpt->m_mutex);
1780
1781     (void) pm_idle_component(mpt->m_dip, 0);
1782
1783     /*
1784      * Cancel watch threads if all mpts suspended
1785      */
1786     rw_enter(&mptsas_global_rwlock, RW_WRITER);
1787     for (g = mptsas_head; g != NULL; g = g->m_next) {
1788         if (!g->m_suspended)
1789             break;
1790     }
1791     rw_exit(&mptsas_global_rwlock);
1792
1793     mutex_enter(&mptsas_global_mutex);
1794     if (g == NULL) {
1795         timeout_id_t tid;
1796
1797         mptsas_timeouts_enabled = 0;
1798         if (mptsas_timeout_id) {
1799             tid = mptsas_timeout_id;
1800             mptsas_timeout_id = 0;
1801             mutex_exit(&mptsas_global_mutex);
1802             (void) untimeout(tid);
1803             mutex_enter(&mptsas_global_mutex);
1804         }
1805         if (mptsas_reset_watch) {
1806             tid = mptsas_reset_watch;
1807             mptsas_reset_watch = 0;
1808             mutex_exit(&mptsas_global_mutex);
1809             (void) untimeout(tid);
1810             mutex_enter(&mptsas_global_mutex);
1811         }
1812     }
1813     mutex_exit(&mptsas_global_mutex);
1814
1815     mutex_enter(&mpt->m_mutex);
1816
1817     /*
1818      * If this mpt is not in full power(PM_LEVEL_D0), just return.
1819      */
1820     if ((mpt->m_options & MPTSAS_OPT_PM) &&
1821         (mpt->m_power_level != PM_LEVEL_D0)) {
1822         mutex_exit(&mpt->m_mutex);
1823         return (DDI_SUCCESS);
1824     }
1825
1826     /* Disable HBA interrupts in hardware */
1827     MPTSAS_DISABLE_INTR(mpt);
1828     /*
1829      * Send RAID action system shutdown to sync IR
1830      */
1831     mptsas_raid_action_system_shutdown(mpt);
1832
1833     mutex_exit(&mpt->m_mutex);
1834
1835     /* drain the taskq */
1836     ddi_taskq_wait(mpt->m_event_taskq);
1837     ddi_taskq_wait(mpt->m_dr_taskq);
1838
1839     return (DDI_SUCCESS);
1840 }
1841
1842 #ifdef __sparc
1843 /* ARGSUSED */

```

```

1844 static int
1845 mptsas_reset(dev_info_t *devi, ddi_reset_cmd_t cmd)
1846 {
1847     mptsas_t      *mpt;
1848     scsi_hba_tran_t *tran;
1849
1850     /*
1851      * If this call is for iport, just return.
1852      */
1853     if (scsi_hba_iport_unit_address(devi))
1854         return (DDI_SUCCESS);
1855
1856     if ((tran = ddi_get_driver_private(devi)) == NULL)
1857         return (DDI_SUCCESS);
1858
1859     if ((mpt = TRAN2MPT(tran)) == NULL)
1860         return (DDI_SUCCESS);
1861
1862     /*
1863      * Send RAID action system shutdown to sync IR.  Disable HBA
1864      * interrupts in hardware first.
1865      */
1866     MPTSAS_DISABLE_INTR(mpt);
1867     mptsas_raid_action_system_shutdown(mpt);
1868
1869     return (DDI_SUCCESS);
1870 }
1871 #else /* __sparc */
1872 /*
1873  * quiesce(9E) entry point.
1874  */
1875 /* This function is called when the system is single-threaded at high
1876  * PIL with preemption disabled. Therefore, this function must not be
1877  * blocked.
1878  */
1879 /* This function returns DDI_SUCCESS on success, or DDI_FAILURE on failure.
1880  * DDI_FAILURE indicates an error condition and should almost never happen.
1881  */
1882 static int
1883 mptsas_quiesce(dev_info_t *devi)
1884 {
1885     mptsas_t      *mpt;
1886     scsi_hba_tran_t *tran;
1887
1888     /*
1889      * If this call is for iport, just return.
1890      */
1891     if (scsi_hba_iport_unit_address(devi))
1892         return (DDI_SUCCESS);
1893
1894     if ((tran = ddi_get_driver_private(devi)) == NULL)
1895         return (DDI_SUCCESS);
1896
1897     if ((mpt = TRAN2MPT(tran)) == NULL)
1898         return (DDI_SUCCESS);
1899
1900     /* Disable HBA interrupts in hardware */
1901     MPTSAS_DISABLE_INTR(mpt);
1902     /* Send RAID action system shutdown to sync IR */
1903     mptsas_raid_action_system_shutdown(mpt);
1904
1905     return (DDI_SUCCESS);
1906 }
1907 #endif /* __sparc */
1908
1909 /*

```

```

1910  * detach(9E). Remove all device allocations and system resources;
1911  * disable device interrupts.
1912  * Return DDI_SUCCESS if done; DDI_FAILURE if there's a problem.
1913  */
1914 static int
1915 mptsas_detach(dev_info_t *devi, ddi_detach_cmd_t cmd)
1916 {
1917     /* CONSTCOND */
1918     ASSERT(NO_COMPETING_THREADS);
1919     NDBG0(("mptsas_detach: dip=0x%p cmd=0x%p", (void *)devi, (void *)cmd));
1920
1921     switch (cmd) {
1922     case DDI_DETACH:
1923         return (mptsas_do_detach(devi));
1924
1925     case DDI_SUSPEND:
1926         return (mptsas_suspend(devi));
1927
1928     default:
1929         return (DDI_FAILURE);
1930     }
1931     /* NOTREACHED */
1932 }
1933
1934 static int
1935 mptsas_do_detach(dev_info_t *dip)
1936 {
1937     mptsas_t      *mpt;
1938     scsi_hba_tran_t *tran;
1939     int            circ = 0;
1940     int            circ1 = 0;
1941     mdi_pathinfo_t *pip = NULL;
1942     int            i;
1943     int            q_thread_num = 0;
1944
1945     NDBG0(("mptsas_do_detach: dip=0x%p", (void *)dip));
1946
1947     if ((tran = ndi_flavorv_get(dip, SCSI_FLAVOR SCSI_DEVICE)) == NULL)
1948         return (DDI_FAILURE);
1949
1950     mpt = TRAN2MPT(tran);
1951     if (!mpt) {
1952         return (DDI_FAILURE);
1953     }
1954     /*
1955      * Still have pathinfo child, should not detach mpt driver
1956      */
1957     if (scsi_hba_iport_unit_address(dip)) {
1958         if (mpt->m_mpxio_enable) {
1959             /*
1960              * MPxIO enabled for the iport
1961              */
1962             ndi_devi_enter(scsi_vhci_dip, &circ1);
1963             ndi_devi_enter(dip, &circ);
1964             while (pip = mdi_get_next_client_path(dip, NULL)) {
1965                 if (mdi_pi_free(pip, 0) == MDI_SUCCESS) {
1966                     continue;
1967                 }
1968                 ndi_devi_exit(dip, circ);
1969                 ndi_devi_exit(scsi_vhci_dip, circ1);
1970                 NDBG12(("detach failed because of "
1971                     "outstanding path info"));
1972                 return (DDI_FAILURE);
1973             }
1974             ndi_devi_exit(dip, circ);
1975             ndi_devi_exit(scsi_vhci_dip, circ1);

```



```

1976         (void) mdi_phci_unregister(dip, 0);
1977     }
1979     ddi_prop_remove_all(dip);
1981     return (DDI_SUCCESS);
1982 }

1984 /* Make sure power level is D0 before accessing registers */
1985 if (mpt->m_options & MPTSAS_OPT_PM) {
1986     (void) pm_busy_component(dip, 0);
1987     if (mpt->m_power_level != PM_LEVEL_D0) {
1988         if (pm_raise_power(dip, 0, PM_LEVEL_D0) !=
1989             DDI_SUCCESS) {
1990             mptsas_log(mpt, CE_WARN,
1991                 "mptsas3%d: Raise power request failed.",
1992                 mpt->m_instance);
1993             (void) pm_idle_component(dip, 0);
1994             return (DDI_FAILURE);
1995         }
1996     }
1997 }

1999 /*
2000  * Send RAID action system shutdown to sync IR. After action, send a
2001  * Message Unit Reset. Since after that DMA resource will be freed,
2002  * set ioc to READY state will avoid HBA initiated DMA operation.
2003  */
2004 mutex_enter(&mpt->m_mutex);
2005 MPTSAS_DISABLE_INTR(mpt);
2006 mptsas_raid_action_system_shutdown(mpt);
2007 mpt->m_softstate |= MPTSAS_SS_MSG_UNIT_RESET;
2008 (void) mptsas_ioc_reset(mpt, FALSE);
2009 mutex_exit(&mpt->m_mutex);
2010 mptsas_rem_intrs(mpt);
2011 ddi_taskq_destroy(mpt->m_event_taskq);
2012 ddi_taskq_destroy(mpt->m_dr_taskq);

2014 if (mpt->m_doneq_thread_n) {
2015     mutex_enter(&mpt->m_qthread_mutex);
2016     q_thread_num = mpt->m_doneq_thread_n;
2017     for (i = 0; i < mpt->m_doneq_thread_n; i++) {
2018         mutex_enter(&mpt->m_doneq_thread_id[i].mutex);
2019         mpt->m_doneq_thread_id[i].flag &=
2020             (~MPTSAS_DONEQ_THREAD_ACTIVE);
2021         cv_signal(&mpt->m_doneq_thread_id[i].cv);
2022         mutex_exit(&mpt->m_doneq_thread_id[i].mutex);
2023     }
2024     while (mpt->m_doneq_thread_n) {
2025         cv_wait(&mpt->m_qthread_cv,
2026             &mpt->m_qthread_mutex);
2027     }
2028     for (i = 0; i < q_thread_num; i++) {
2029         cv_destroy(&mpt->m_doneq_thread_id[i].cv);
2030         mutex_destroy(&mpt->m_doneq_thread_id[i].mutex);
2031     }
2032     kmem_free(mpt->m_doneq_thread_id,
2033         sizeof(mptsas_doneq_thread_list_t)
2034         * q_thread_num);
2035     mutex_exit(&mpt->m_qthread_mutex);
2036 }
2037 if (mpt->m_txwq_thread_n) {
2038     mutex_enter(&mpt->m_qthread_mutex);
2039     q_thread_num = mpt->m_txwq_thread_n;
2040     for (i = 0; i < q_thread_num; i++) {
2041         mutex_enter(&mpt->m_tx_waitq[i].txwq_mutex);

```

```

2042         mpt->m_tx_waitq[i].txwq_active = FALSE;
2043         cv_signal(&mpt->m_tx_waitq[i].txwq_cv);
2044         mutex_exit(&mpt->m_tx_waitq[i].txwq_mutex);
2045     }
2046     while (mpt->m_txwq_thread_n) {
2047         cv_wait(&mpt->m_qthread_cv,
2048             &mpt->m_qthread_mutex);
2049     }
2050     for (i = 0; i < q_thread_num; i++) {
2051         cv_destroy(&mpt->m_tx_waitq[i].txwq_cv);
2052         cv_destroy(&mpt->m_tx_waitq[i].txwq_drain_cv);
2053         mutex_destroy(&mpt->m_tx_waitq[i].txwq_mutex);
2054     }
2055 }

2057 scsi_hba_reset_notify_tear_down(mpt->m_reset_notify_listf);
2059 mptsas_list_del(mpt);

2061 /*
2062  * Cancel timeout threads for this mpt
2063  */
2064 mutex_enter(&mpt->m_mutex);
2065 if (mpt->m_quiesce_timeid) {
2066     timeout_id_t tid = mpt->m_quiesce_timeid;
2067     mpt->m_quiesce_timeid = 0;
2068     mutex_exit(&mpt->m_mutex);
2069     (void) untimeout(tid);
2070     mutex_enter(&mpt->m_mutex);
2071 }

2073 if (mpt->m_restart_cmd_timeid) {
2074     timeout_id_t tid = mpt->m_restart_cmd_timeid;
2075     mpt->m_restart_cmd_timeid = 0;
2076     mutex_exit(&mpt->m_mutex);
2077     (void) untimeout(tid);
2078     mutex_enter(&mpt->m_mutex);
2079 }

2081 mutex_exit(&mpt->m_mutex);

2083 /*
2084  * last mpt? ... if active, CANCEL watch threads.
2085  */
2086 mutex_enter(&mptsas_global_mutex);
2087 if (mptsas_head == NULL) {
2088     timeout_id_t tid;
2089     /*
2090      * Clear mptsas_timeouts_enable so that the watch thread
2091      * gets restarted on DDI_ATTACH
2092      */
2093     mptsas_timeouts_enabled = 0;
2094     if (mptsas_timeout_id) {
2095         tid = mptsas_timeout_id;
2096         mptsas_timeout_id = 0;
2097         mutex_exit(&mptsas_global_mutex);
2098         (void) untimeout(tid);
2099         mutex_enter(&mptsas_global_mutex);
2100     }
2101     if (mptsas_reset_watch) {
2102         tid = mptsas_reset_watch;
2103         mptsas_reset_watch = 0;
2104         mutex_exit(&mptsas_global_mutex);
2105         (void) untimeout(tid);
2106         mutex_enter(&mptsas_global_mutex);
2107     }

```

```

2108     }
2109     mutex_exit(&mptsas_global_mutex);

2111     /*
2112      * Delete Phy stats
2113      */
2114     mptsas_destroy_phy_stats(mpt);

2116     mptsas_destroy_hashes(mpt);

2118     /*
2119      * Delete nt_active.
2120      */
2121     mutex_enter(&mpt->m_mutex);
2122     mptsas_free_active_slots(mpt);
2123     mutex_exit(&mpt->m_mutex);

2125     /* deallocate everything that was allocated in mptsas_attach */
2126     mptsas_cache_destroy(mpt);

2128     mptsas_hba_fini(mpt);
2129     mptsas_cfg_fini(mpt);

2131     /* Lower the power informing PM Framework */
2132     if (mpt->m_options & MPTSAS_OPT_PM) {
2133         if (pm_lower_power(dip, 0, PM_LEVEL_D3) != DDI_SUCCESS)
2134             mptsas_log(mpt, CE_WARN,
2135                 "!mptsas3%d: Lower power request failed "
2136                 "during detach, ignoring.",
2137                 mpt->m_instance);
2138     }

2140     mutex_destroy(&mpt->m_qthread_mutex);
2141     mutex_destroy(&mpt->m_passthru_mutex);
2142     mutex_destroy(&mpt->m_mutex);
2143     for (i = 0; i < MPTSAS_MAX_PHYS; i++) {
2144         mutex_destroy(&mpt->m_phy_info[i].smhba_info.phy_mutex);
2145     }
2146     cv_destroy(&mpt->m_qthread_cv);
2147     cv_destroy(&mpt->m_cv);
2148     cv_destroy(&mpt->m_passthru_cv);
2149     cv_destroy(&mpt->m_fw_cv);
2150     cv_destroy(&mpt->m_config_cv);
2151     cv_destroy(&mpt->m_fw_diag_cv);

2154     mptsas_smp_teardown(mpt);
2155     mptsas_hba_teardown(mpt);

2157     mptsas_config_space_fini(mpt);

2159     mptsas_free_handshake_msg(mpt);

2161     mptsas_fm_fini(mpt);
2162     ddi_soft_state_free(mptsas3_state, ddi_get_instance(dip));
2163     ddi_prop_remove_all(dip);

2165     return (DDI_SUCCESS);
2166 }

2168 static void
2169 mptsas_list_add(mptsas_t *mpt)
2170 {
2171     rw_enter(&mptsas_global_rwlock, RW_WRITER);

2173     if (mptsas_head == NULL) {

```

```

2174         mptsas_head = mpt;
2175     } else {
2176         mptsas_tail->m_next = mpt;
2177     }
2178     mptsas_tail = mpt;
2179     rw_exit(&mptsas_global_rwlock);
2180 }

2182 static void
2183 mptsas_list_del(mptsas_t *mpt)
2184 {
2185     mptsas_t *m;
2186     /*
2187      * Remove device instance from the global linked list
2188      */
2189     rw_enter(&mptsas_global_rwlock, RW_WRITER);
2190     if (mptsas_head == mpt) {
2191         m = mptsas_head = mpt->m_next;
2192     } else {
2193         for (m = mptsas_head; m != NULL; m = m->m_next) {
2194             if (m->m_next == mpt) {
2195                 m->m_next = mpt->m_next;
2196                 break;
2197             }
2198         }
2199         if (m == NULL) {
2200             mptsas_log(mpt, CE_PANIC, "Not in softc list!");
2201         }
2202     }

2204     if (mptsas_tail == mpt) {
2205         mptsas_tail = m;
2206     }
2207     rw_exit(&mptsas_global_rwlock);
2208 }

2210 static int
2211 mptsas_alloc_handshake_msg(mptsas_t *mpt, size_t alloc_size)
2212 {
2213     ddi_dma_attr_t task_dma_attrs;

2215     mpt->m_hshk_dma_size = 0;
2216     task_dma_attrs = mpt->m_msg_dma_attr;
2217     task_dma_attrs.dma_attr_sgllen = 1;
2218     task_dma_attrs.dma_attr_granular = (uint32_t)(alloc_size);

2220     /* allocate Task Management ddi_dma resources */
2221     if (mptsas_dma_addr_create(mpt, task_dma_attrs,
2222         &mpt->m_hshk_dma_hdl, &mpt->m_hshk_acc_hdl, &mpt->m_hshk_memp,
2223         alloc_size, NULL) == FALSE) {
2224         return (DDI_FAILURE);
2225     }
2226     mpt->m_hshk_dma_size = alloc_size;

2228     return (DDI_SUCCESS);
2229 }

2231 static void
2232 mptsas_free_handshake_msg(mptsas_t *mpt)
2233 {
2234     if (mpt->m_hshk_dma_size == 0)
2235         return;
2236     mptsas_dma_addr_destroy(&mpt->m_hshk_dma_hdl, &mpt->m_hshk_acc_hdl);
2237     mpt->m_hshk_dma_size = 0;
2238 }

```

```

2240 static int
2241 mptsas_hba_setup(mptsas_t *mpt)
2242 {
2243     scsi_hba_tran_t      *hba_tran;
2244     int                   tran_flags;
2245
2246     /* Allocate a transport structure */
2247     hba_tran = mpt->m_tran = scsi_hba_tran_alloc(mpt->m_dip,
2248         SCSI_HBA_CANSLEEP);
2249     ASSERT(mpt->m_tran != NULL);
2250
2251     hba_tran->tran_hba_private = mpt;
2252     hba_tran->tran_tgt_private = NULL;
2253
2254     hba_tran->tran_tgt_init = mptsas_scsi_tgt_init;
2255     hba_tran->tran_tgt_free = mptsas_scsi_tgt_free;
2256
2257     hba_tran->tran_start = mptsas_scsi_start;
2258     hba_tran->tran_reset = mptsas_scsi_reset;
2259     hba_tran->tran_abort = mptsas_scsi_abort;
2260     hba_tran->tran_getcap = mptsas_scsi_getcap;
2261     hba_tran->tran_setcap = mptsas_scsi_setcap;
2262     hba_tran->tran_init_pkt = mptsas_scsi_init_pkt;
2263     hba_tran->tran_destroy_pkt = mptsas_scsi_destroy_pkt;
2264
2265     hba_tran->tran_dmafree = mptsas_scsi_dmafree;
2266     hba_tran->tran_sync_pkt = mptsas_scsi_sync_pkt;
2267     hba_tran->tran_reset_notify = mptsas_scsi_reset_notify;
2268
2269     hba_tran->tran_get_bus_addr = mptsas_get_bus_addr;
2270     hba_tran->tran_get_name = mptsas_get_name;
2271
2272     hba_tran->tran_quiesce = mptsas_scsi_quiesce;
2273     hba_tran->tran_unquiesce = mptsas_scsi_unquiesce;
2274     hba_tran->tran_bus_reset = NULL;
2275
2276     hba_tran->tran_add_eventcall = NULL;
2277     hba_tran->tran_get_eventcookie = NULL;
2278     hba_tran->tran_post_event = NULL;
2279     hba_tran->tran_remove_eventcall = NULL;
2280
2281     hba_tran->tran_bus_config = mptsas_bus_config;
2282
2283     hba_tran->tran_interconnect_type = INTERCONNECT_SAS;
2284
2285     /*
2286      * All children of the HBA are iports. We need tran was cloned.
2287      * So we pass the flags to SCSI_HBA_TRAN_CLONE will be
2288      * inherited to iport's tran vector.
2289      */
2290     tran_flags = (SCSI_HBA_HBA | SCSI_HBA_TRAN_CLONE);
2291
2292     if (scsi_hba_attach_setup(mpt->m_dip, &mpt->m_msg_dma_attr,
2293         hba_tran, tran_flags) != DDI_SUCCESS) {
2294         mptsas_log(mpt, CE_WARN, "hba attach setup failed");
2295         scsi_hba_tran_free(hba_tran);
2296         mpt->m_tran = NULL;
2297         return (FALSE);
2298     }
2299     return (TRUE);
2300 }
2301
2302 static void
2303 mptsas_hba_teardown(mptsas_t *mpt)
2304 {
2305     (void) scsi_hba_detach(mpt->m_dip);

```

```

2306     if (mpt->m_tran != NULL) {
2307         scsi_hba_tran_free(mpt->m_tran);
2308         mpt->m_tran = NULL;
2309     }
2310 }
2311
2312 static void
2313 mptsas_iport_register(mptsas_t *mpt)
2314 {
2315     int i, j;
2316     mptsas_phymask_t mask = 0x0;
2317     /*
2318      * initial value of mask is 0
2319      */
2320     mutex_enter(&mpt->m_mutex);
2321     for (i = 0; i < mpt->m_num_phys; i++) {
2322         mptsas_phymask_t phy_mask = 0x0;
2323         char phy_mask_name[MPTSAS_MAX_PHYS];
2324         uint8_t current_port;
2325
2326         if (mpt->m_phy_info[i].attached_devhdl == 0)
2327             continue;
2328
2329         bzero(phy_mask_name, sizeof (phy_mask_name));
2330
2331         current_port = mpt->m_phy_info[i].port_num;
2332
2333         if ((mask & (1 << i)) != 0)
2334             continue;
2335
2336         for (j = 0; j < mpt->m_num_phys; j++) {
2337             if (mpt->m_phy_info[j].attached_devhdl &&
2338                 (mpt->m_phy_info[j].port_num == current_port)) {
2339                 phy_mask |= (1 << j);
2340             }
2341         }
2342         mask = mask | phy_mask;
2343
2344         for (j = 0; j < mpt->m_num_phys; j++) {
2345             if ((phy_mask >> j) & 0x01) {
2346                 mpt->m_phy_info[j].phy_mask = phy_mask;
2347             }
2348         }
2349
2350         (void) sprintf(phy_mask_name, "%x", phy_mask);
2351
2352         mutex_exit(&mpt->m_mutex);
2353         /*
2354          * register a iport
2355          */
2356         (void) scsi_hba_iport_register(mpt->m_dip, phy_mask_name);
2357         mutex_enter(&mpt->m_mutex);
2358     }
2359     mutex_exit(&mpt->m_mutex);
2360     /*
2361      * register a virtual port for RAID volume always
2362      */
2363     (void) scsi_hba_iport_register(mpt->m_dip, "v0");
2364 }
2365
2366 static int
2367 mptsas_smp_setup(mptsas_t *mpt)
2368 {
2369     mpt->m_smptran = smp_hba_tran_alloc(mpt->m_dip);
2370     ASSERT(mpt->m_smptran != NULL);

```

```

2372     mpt->m_smptran->smp_tran_hba_private = mpt;
2373     mpt->m_smptran->smp_tran_start = mptsas_smp_start;
2374     if (smp_hba_attach_setup(mpt->m_dip, mpt->m_smptran) != DDI_SUCCESS) {
2375         mptsas_log(mpt, CE_WARN, "smp attach setup failed");
2376         smp_hba_tran_free(mpt->m_smptran);
2377         mpt->m_smptran = NULL;
2378         return (FALSE);
2379     }
2380     /*
2381      * Initialize smp hash table
2382      */
2383     mpt->m_smp_targets = rehash_create(MPTSAS_SMP_BUCKET_COUNT,
2384         mptsas_target_addr_hash, mptsas_target_addr_cmp,
2385         mptsas_smp_free, sizeof (mptsas_smp_t),
2386         offsetof(mptsas_smp_t, m_link), offsetof(mptsas_smp_t, m_addr),
2387         KM_SLEEP);
2388     mpt->m_smp_devhdl = 0xFFFF;

2390     return (TRUE);
2391 }

2393 static void
2394 mptsas_smp_tear_down(mptsas_t *mpt)
2395 {
2396     (void) smp_hba_detach(mpt->m_dip);
2397     if (mpt->m_smptran != NULL) {
2398         smp_hba_tran_free(mpt->m_smptran);
2399         mpt->m_smptran = NULL;
2400     }
2401     mpt->m_smp_devhdl = 0;
2402 }

2404 static int
2405 mptsas_cache_create(mptsas_t *mpt)
2406 {
2407     int instance = mpt->m_instance;
2408     char buf[64];

2410     /*
2411      * create kmem cache for packets
2412      */
2413     (void) sprintf(buf, "mptsas3%d_cache", instance);
2414     mpt->m_kmem_cache = kmem_cache_create(buf,
2415         sizeof (struct mptsas_cmd) + scsi_pkt_size(), 16,
2416         mptsas_kmem_cache_constructor, mptsas_kmem_cache_destructor,
2417         NULL, (void *)mpt, NULL, 0);

2419     if (mpt->m_kmem_cache == NULL) {
2420         mptsas_log(mpt, CE_WARN, "creating kmem cache failed");
2421         return (FALSE);
2422     }

2424     /*
2425      * create kmem cache for extra SGL frames if SGL cannot
2426      * be accommodated into main request frame.
2427      */
2428     (void) sprintf(buf, "mptsas3%d_cache_frames", instance);
2429     mpt->m_cache_frames = kmem_cache_create(buf,
2430         sizeof (mptsas_cache_frames_t), 16,
2431         mptsas_cache_frames_constructor, mptsas_cache_frames_destructor,
2432         NULL, (void *)mpt, NULL, 0);

2434     if (mpt->m_cache_frames == NULL) {
2435         mptsas_log(mpt, CE_WARN, "creating cache for frames failed");
2436         return (FALSE);
2437     }

```

```

2439         return (TRUE);
2440     }

2442 static void
2443 mptsas_cache_destroy(mptsas_t *mpt)
2444 {
2445     /* deallocate in reverse order */
2446     if (mpt->m_cache_frames) {
2447         kmem_cache_destroy(mpt->m_cache_frames);
2448         mpt->m_cache_frames = NULL;
2449     }
2450     if (mpt->m_kmem_cache) {
2451         kmem_cache_destroy(mpt->m_kmem_cache);
2452         mpt->m_kmem_cache = NULL;
2453     }
2454 }

2456 static int
2457 mptsas_power(dev_info_t *dip, int component, int level)
2458 {
2459     #ifndef __lock_lint
2460         __NOTE(ARGUNUSED(component))
2461     #endif
2462     mptsas_t *mpt;
2463     int rval = DDI_SUCCESS;
2464     int polls = 0;
2465     uint32_t ioc_status;

2467     if (scsi_hba_iport_unit_address(dip) != 0)
2468         return (DDI_SUCCESS);

2470     mpt = ddi_get_soft_state(mptsas3_state, ddi_get_instance(dip));
2471     if (mpt == NULL) {
2472         return (DDI_FAILURE);
2473     }

2475     mutex_enter(&mpt->m_mutex);

2477     /*
2478      * If the device is busy, don't lower its power level
2479      */
2480     if (mpt->m_busy && (mpt->m_power_level > level)) {
2481         mutex_exit(&mpt->m_mutex);
2482         return (DDI_FAILURE);
2483     }
2484     switch (level) {
2485     case PM_LEVEL_D0:
2486         NDBG11(("mptsas3%d: turning power ON.", mpt->m_instance));
2487         MPTSAS_POWER_ON(mpt);
2488         /*
2489          * Wait up to 30 seconds for IOC to come out of reset.
2490          */
2491         while (((ioc_status = ddi_get32(mpt->m_datap,
2492             &mpt->m_reg->Doorbell)) &
2493             MPI2_IOC_STATE_MASK) == MPI2_IOC_STATE_RESET) {
2494             if (polls++ > 3000) {
2495                 break;
2496             }
2497             delay(drv_usectohz(10000));
2498         }
2499         /*
2500          * If IOC is not in operational state, try to hard reset it.
2501          */
2502         if ((ioc_status & MPI2_IOC_STATE_MASK) !=
2503             MPI2_IOC_STATE_OPERATIONAL) {

```

```

2504         mpt->m_softstate &= ~MPTSAS_SS_MSG_UNIT_RESET;
2505         if (mptsas_restart_ioc(mpt) == DDI_FAILURE) {
2506             mptsas_log(mpt, CE_WARN,
2507                 "mptsas_power: hard reset failed");
2508             mutex_exit(&mpt->m_mutex);
2509             return (DDI_FAILURE);
2510         }
2511     }
2512     mpt->m_power_level = PM_LEVEL_D0;
2513     break;
2514 case PM_LEVEL_D3:
2515     NDBG11(("mptsas3%d: turning power OFF.", mpt->m_instance));
2516     MPTSAS_POWER_OFF(mpt);
2517     break;
2518 default:
2519     mptsas_log(mpt, CE_WARN, "mptsas3%d: unknown power level <%x>.",
2520         mpt->m_instance, level);
2521     rval = DDI_FAILURE;
2522     break;
2523 }
2524 mutex_exit(&mpt->m_mutex);
2525 return (rval);
2526 }

2528 /*
2529  * Initialize configuration space and figure out which
2530  * chip and revision of the chip the mpt driver is using.
2531  */
2532 static int
2533 mptsas_config_space_init(mptsas_t *mpt)
2534 {
2535     NDBG0(("mptsas_config_space_init"));

2537     if (mpt->m_config_handle != NULL)
2538         return (TRUE);

2540     if (pci_config_setup(mpt->m_dip,
2541         &mpt->m_config_handle) != DDI_SUCCESS) {
2542         mptsas_log(mpt, CE_WARN, "cannot map configuration space.");
2543         return (FALSE);
2544     }

2546     /*
2547      * This is a workaround for a XMITs ASIC bug which does not
2548      * drive the CBE upper bits.
2549      */
2550     if (pci_config_get16(mpt->m_config_handle, PCI_CONF_STAT) &
2551         PCI_STAT_ERROR) {
2552         pci_config_put16(mpt->m_config_handle, PCI_CONF_STAT,
2553             PCI_STAT_ERROR);
2554     }

2556     mptsas_setup_cmd_reg(mpt);

2558     /*
2559      * Get the chip device id:
2560      */
2561     mpt->m_devid = pci_config_get16(mpt->m_config_handle, PCI_CONF_DEVID);

2563     /*
2564      * Save the revision.
2565      */
2566     mpt->m_revid = pci_config_get8(mpt->m_config_handle, PCI_CONF_REVID);

2568     /*
2569      * Save the SubSystem Vendor and Device IDs

```

```

2570     */
2571     mpt->m_svid = pci_config_get16(mpt->m_config_handle, PCI_CONF_SUBVENID);
2572     mpt->m_ssid = pci_config_get16(mpt->m_config_handle, PCI_CONF_SUBSYSID);

2574     /*
2575      * Set the latency timer to 0x40 as specified by the upa -> pci
2576      * bridge chip design team. This may be done by the sparc pci
2577      * bus nexus driver, but the driver should make sure the latency
2578      * timer is correct for performance reasons.
2579      */
2580     pci_config_put8(mpt->m_config_handle, PCI_CONF_LATENCY_TIMER,
2581         MPTSAS_LATENCY_TIMER);

2583     (void) mptsas_get_pci_cap(mpt);
2584     return (TRUE);
2585 }

2587 static void
2588 mptsas_config_space_fini(mptsas_t *mpt)
2589 {
2590     if (mpt->m_config_handle != NULL) {
2591         mptsas_disable_bus_master(mpt);
2592         pci_config_tear_down(&mpt->m_config_handle);
2593         mpt->m_config_handle = NULL;
2594     }
2595 }

2597 static void
2598 mptsas_setup_cmd_reg(mptsas_t *mpt)
2599 {
2600     ushort_t    cmdreg;

2602     /*
2603      * Set the command register to the needed values.
2604      */
2605     cmdreg = pci_config_get16(mpt->m_config_handle, PCI_CONF_COMM);
2606     cmdreg |= (PCI_COMM_ME | PCI_COMM_SERR_ENABLE |
2607         PCI_COMM_PARITY_DETECT | PCI_COMM_MAE);
2608     cmdreg &= ~PCI_COMM_IO;
2609     pci_config_put16(mpt->m_config_handle, PCI_CONF_COMM, cmdreg);
2610 }

2612 static void
2613 mptsas_disable_bus_master(mptsas_t *mpt)
2614 {
2615     ushort_t    cmdreg;

2617     /*
2618      * Clear the master enable bit in the PCI command register.
2619      * This prevents any bus mastering activity like DMA.
2620      */
2621     cmdreg = pci_config_get16(mpt->m_config_handle, PCI_CONF_COMM);
2622     cmdreg &= ~PCI_COMM_ME;
2623     pci_config_put16(mpt->m_config_handle, PCI_CONF_COMM, cmdreg);
2624 }

2626 int
2627 mptsas_dma_alloc(mptsas_t *mpt, mptsas_dma_alloc_state_t *dma_statep)
2628 {
2629     ddi_dma_attr_t  attrs;

2631     attrs = mpt->m_io_dma_attr;
2632     attrs.dma_attr_sgllen = 1;

2634     ASSERT(dma_statep != NULL);

```

```

2636     if (mptsas_dma_addr_create(mpt, attrs, &dma_statep->handle,
2637         &dma_statep->accessp, &dma_statep->memp, dma_statep->size,
2638         &dma_statep->cookie) == FALSE) {
2639         return (DDI_FAILURE);
2640     }
2642     return (DDI_SUCCESS);
2643 }
2645 void
2646 mptsas_dma_free(mptsas_dma_alloc_state_t *dma_statep)
2647 {
2648     ASSERT(dma_statep != NULL);
2649     mptsas_dma_addr_destroy(&dma_statep->handle, &dma_statep->accessp);
2650     dma_statep->size = 0;
2651 }
2653 int
2654 mptsas_do_dma(mptsas_t *mpt, uint32_t size, int var, int (*callback)())
2655 {
2656     ddi_dma_attr_t      attrs;
2657     ddi_dma_handle_t    dma_handle;
2658     caddr_t             memp;
2659     ddi_acc_handle_t    accessp;
2660     int                 rval;
2662     ASSERT(mutex_owned(&mpt->m_mutex));
2664     attrs = mpt->m_msg_dma_attr;
2665     attrs.dma_attr_sgllen = 1;
2666     attrs.dma_attr_granular = size;
2668     if (mptsas_dma_addr_create(mpt, attrs, &dma_handle,
2669         &accessp, &memp, size, NULL) == FALSE) {
2670         return (DDI_FAILURE);
2671     }
2673     rval = (*callback) (mpt, memp, var, accessp);
2675     if ((mptsas_check_dma_handle(dma_handle) != DDI_SUCCESS) ||
2676         (mptsas_check_acc_handle(accessp) != DDI_SUCCESS)) {
2677         ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_UNAFFECTED);
2678         rval = DDI_FAILURE;
2679     }
2681     mptsas_dma_addr_destroy(&dma_handle, &accessp);
2682     return (rval);
2684 }
2686 static int
2687 mptsas_alloc_request_frames(mptsas_t *mpt)
2688 {
2689     ddi_dma_attr_t      frame_dma_attrs;
2690     caddr_t             memp;
2691     ddi_dma_cookie_t    cookie;
2692     size_t              mem_size;
2694     /*
2695      * re-alloc when it has already allocated
2696      */
2697     if (mpt->m_dma_flags & MPTSAS_REQ_FRAME) {
2698         mptsas_dma_addr_destroy(&mpt->m_dma_req_frame_hdl,
2699             &mpt->m_acc_req_frame_hdl);
2700         mpt->m_dma_flags &= ~MPTSAS_REQ_FRAME;
2701     }

```

```

2703     /*
2704      * The size of the request frame pool is:
2705      *   Number of Request Frames * Request Frame Size
2706      */
2707     mem_size = mpt->m_max_requests * mpt->m_req_frame_size;
2709     /*
2710      * set the DMA attributes. System Request Message Frames must be
2711      * aligned on a 16-byte boundary.
2712      */
2713     frame_dma_attrs = mpt->m_msg_dma_attr;
2714     frame_dma_attrs.dma_attr_align = 16;
2715     frame_dma_attrs.dma_attr_sgllen = 1;
2717     /*
2718      * allocate the request frame pool.
2719      */
2720     if (mptsas_dma_addr_create(mpt, frame_dma_attrs,
2721         &mpt->m_dma_req_frame_hdl, &mpt->m_acc_req_frame_hdl, &memp,
2722         mem_size, &cookie) == FALSE) {
2723         return (DDI_FAILURE);
2724     }
2726     /*
2727      * Store the request frame memory address. This chip uses this
2728      * address to dma to and from the driver's frame. The second
2729      * address is the address mpt uses to fill in the frame.
2730      */
2731     mpt->m_req_frame_dma_addr = cookie.dmac_laddress;
2732     mpt->m_req_frame = memp;
2734     /*
2735      * Clear the request frame pool.
2736      */
2737     bzero(mpt->m_req_frame, mem_size);
2739     mpt->m_dma_flags |= MPTSAS_REQ_FRAME;
2740     return (DDI_SUCCESS);
2741 }
2743 static int
2744 mptsas_alloc_sense_bufs(mptsas_t *mpt)
2745 {
2746     ddi_dma_attr_t      sense_dma_attrs;
2747     caddr_t             memp;
2748     ddi_dma_cookie_t    cookie;
2749     size_t              mem_size;
2750     int                 num_extrqsense_bufs;
2752     /*
2753      * re-alloc when it has already allocated
2754      */
2755     if (mpt->m_dma_flags & MPTSAS_REQ_SENSE) {
2756         rmfreesmap(mpt->m_ergsense_map);
2757         mptsas_dma_addr_destroy(&mpt->m_dma_req_sense_hdl,
2758             &mpt->m_acc_req_sense_hdl);
2759         mpt->m_dma_flags &= ~MPTSAS_REQ_SENSE;
2760     }
2762     /*
2763      * The size of the request sense pool is:
2764      *   (Number of Request Frames - 2) * Request Sense Size +
2765      *   extra memory for extended sense requests.
2766      */
2767     mem_size = ((mpt->m_max_requests - 2) * mpt->m_req_sense_size) +

```

```

2768     mptsas_extreq_sense_bufsize;

2770     /*
2771     * set the DMA attributes.  ARQ buffers
2772     * aligned on a 16-byte boundary.
2773     */
2774     sense_dma_attrs = mpt->m_msg_dma_attr;
2775     sense_dma_attrs.dma_attr_align = 16;
2776     sense_dma_attrs.dma_attr_sgllen = 1;

2778     /*
2779     * allocate the request sense buffer pool.
2780     */
2781     if (mptsas_dma_addr_create(mpt, sense_dma_attrs,
2782         &mpt->m_dma_req_sense_hdl, &mpt->m_acc_req_sense_hdl, &memp,
2783         mem_size, &cookie) == FALSE) {
2784         return (DDI_FAILURE);
2785     }

2787     /*
2788     * Store the request sense base memory address.  This chip uses this
2789     * address to dma the request sense data.  The second
2790     * address is the address mpt uses to access the data.
2791     * The third is the base for the extended rqsense buffers.
2792     */
2793     mpt->m_req_sense_dma_addr = cookie.dmac_laddress;
2794     mpt->m_req_sense = memp;
2795     memp += (mpt->m_max_requests - 2) * mpt->m_req_sense_size;
2796     mpt->m_extreq_sense = memp;

2798     /*
2799     * The extra memory is divided up into multiples of the base
2800     * buffer size in order to allocate via rmalloc().
2801     * Note that the rmallocmap cannot start at zero!
2802     */
2803     num_extrqsense_bufs = mptsas_extreq_sense_bufsize /
2804         mpt->m_req_sense_size;
2805     mpt->m_erqsense_map = rmallocmap_wait(num_extrqsense_bufs);
2806     rmfree(mpt->m_erqsense_map, num_extrqsense_bufs, 1);

2808     /*
2809     * Clear the pool.
2810     */
2811     bzero(mpt->m_req_sense, mem_size);

2813     mpt->m_dma_flags |= MPTSAS_REQ_SENSE;
2814     return (DDI_SUCCESS);
2815 }

2817 static int
2818 mptsas_alloc_reply_frames(mptsas_t *mpt)
2819 {
2820     ddi_dma_attr_t      frame_dma_attrs;
2821     caddr_t             memp;
2822     ddi_dma_cookie_t    cookie;
2823     size_t              mem_size;

2825     /*
2826     * re-alloc when it has already allocated
2827     */
2828     if (mpt->m_dma_flags & MPTSAS_REPLY_FRAME) {
2829         mptsas_dma_addr_destroy(&mpt->m_dma_reply_frame_hdl,
2830             &mpt->m_acc_reply_frame_hdl);
2831         mpt->m_dma_flags &= ~MPTSAS_REPLY_FRAME;
2832     }

```

```

2834     /*
2835     * The size of the reply frame pool is:
2836     *   Number of Reply Frames * Reply Frame Size
2837     */
2838     mem_size = mpt->m_max_replies * mpt->m_reply_frame_size;

2840     /*
2841     * set the DMA attributes.  System Reply Message Frames must be
2842     * aligned on a 4-byte boundary.  This is the default.
2843     */
2844     frame_dma_attrs = mpt->m_msg_dma_attr;
2845     frame_dma_attrs.dma_attr_sgllen = 1;

2847     /*
2848     * allocate the reply frame pool
2849     */
2850     if (mptsas_dma_addr_create(mpt, frame_dma_attrs,
2851         &mpt->m_dma_reply_frame_hdl, &mpt->m_acc_reply_frame_hdl, &memp,
2852         mem_size, &cookie) == FALSE) {
2853         return (DDI_FAILURE);
2854     }

2856     /*
2857     * Store the reply frame memory address.  This chip uses this
2858     * address to dma to and from the driver's frame.  The second
2859     * address is the address mpt uses to process the frame.
2860     */
2861     mpt->m_reply_frame_dma_addr = cookie.dmac_laddress;
2862     mpt->m_reply_frame = memp;

2864     /*
2865     * Clear the reply frame pool.
2866     */
2867     bzero(mpt->m_reply_frame, mem_size);

2869     mpt->m_dma_flags |= MPTSAS_REPLY_FRAME;
2870     return (DDI_SUCCESS);
2871 }

2873 static int
2874 mptsas_alloc_free_queue(mptsas_t *mpt)
2875 {
2876     ddi_dma_attr_t      frame_dma_attrs;
2877     caddr_t             memp;
2878     ddi_dma_cookie_t    cookie;
2879     size_t              mem_size;

2881     /*
2882     * re-alloc when it has already allocated
2883     */
2884     if (mpt->m_dma_flags & MPTSAS_FREE_QUEUE) {
2885         mptsas_dma_addr_destroy(&mpt->m_dma_free_queue_hdl,
2886             &mpt->m_acc_free_queue_hdl);
2887         mpt->m_dma_flags &= ~MPTSAS_FREE_QUEUE;
2888     }

2890     /*
2891     * The reply free queue size is:
2892     *   Reply Free Queue Depth * 4
2893     * The "4" is the size of one 32 bit address (low part of 64-bit
2894     *   address)
2895     */
2896     mem_size = mpt->m_free_queue_depth * 4;

2898     /*
2899     * set the DMA attributes  The Reply Free Queue must be aligned on a

```

```

2900     * 16-byte boundary.
2901     */
2902     frame_dma_attrs = mpt->m_msg_dma_attr;
2903     frame_dma_attrs.dma_attr_align = 16;
2904     frame_dma_attrs.dma_attr_sgllen = 1;
2905
2906     /*
2907     * allocate the reply free queue
2908     */
2909     if (mptsas_dma_addr_create(mpt, frame_dma_attrs,
2910         &mpt->m_dma_free_queue_hdl, &mpt->m_acc_free_queue_hdl, &memp,
2911         mem_size, &cookie) == FALSE) {
2912         return (DDI_FAILURE);
2913     }
2914
2915     /*
2916     * Store the reply free queue memory address. This chip uses this
2917     * address to read from the reply free queue. The second address
2918     * is the address mpt uses to manage the queue.
2919     */
2920     mpt->m_free_queue_dma_addr = cookie.dmac_laddress;
2921     mpt->m_free_queue = memp;
2922
2923     /*
2924     * Clear the reply free queue memory.
2925     */
2926     bzero(mpt->m_free_queue, mem_size);
2927
2928     mpt->m_dma_flags |= MPTSAS_FREE_QUEUE;
2929     return (DDI_SUCCESS);
2930 }
2931
2932 static void
2933 mptsas_free_post_queue(mptsas_t *mpt)
2934 {
2935     mptsas_reply_pqueue_t *rpqp;
2936     int i;
2937
2938     if (mpt->m_dma_flags & MPTSAS_POST_QUEUE) {
2939         mptsas_dma_addr_destroy(&mpt->m_dma_post_queue_hdl,
2940             &mpt->m_acc_post_queue_hdl);
2941         rpqp = mpt->m_rep_post_queues;
2942         for (i = 0; i < mpt->m_post_reply_qcount; i++) {
2943             mutex_destroy(&rpqp->rpq_mutex);
2944             rpqp++;
2945         }
2946         kmem_free(mpt->m_rep_post_queues,
2947             sizeof (mptsas_reply_pqueue_t) *
2948             mpt->m_post_reply_qcount);
2949         mpt->m_dma_flags &= ~MPTSAS_POST_QUEUE;
2950     }
2951 }
2952
2953 static int
2954 mptsas_alloc_post_queue(mptsas_t *mpt)
2955 {
2956     ddi_dma_attr_t      frame_dma_attrs;
2957     caddr_t             memp;
2958     ddi_dma_cookie_t    cookie;
2959     size_t              mem_size;
2960     mptsas_reply_pqueue_t *rpqp;
2961     int i;
2962
2963     /*
2964     * re-alloc when it has already allocated
2965     */

```

```

2966     mptsas_free_post_queue(mpt);
2967
2968     /*
2969     * The reply descriptor post queue size is:
2970     *   Reply Descriptor Post Queue Depth * 8
2971     * The "8" is the size of each descriptor (8 bytes or 64 bits).
2972     */
2973     mpt->m_post_reply_qcount = mpt->m_intr_cnt;
2974     mem_size = mpt->m_post_queue_depth * 8 * mpt->m_post_reply_qcount;
2975
2976     /*
2977     * set the DMA attributes. The Reply Descriptor Post Queue must be
2978     * aligned on a 16-byte boundary.
2979     */
2980     frame_dma_attrs = mpt->m_msg_dma_attr;
2981     frame_dma_attrs.dma_attr_align = 16;
2982     frame_dma_attrs.dma_attr_sgllen = 1;
2983
2984     /*
2985     * Allocate the reply post queue(s).
2986     * MPI2.5 introduces a method to allocate multiple queues
2987     * using a redirect table. For now stick to one contiguous
2988     * chunk. This can get as big as 1Mbyte for 16 queues.
2989     * The spec gives no indication that the queue size can be
2990     * reduced if you have many of them.
2991     */
2992     if (mptsas_dma_addr_create(mpt, frame_dma_attrs,
2993         &mpt->m_dma_post_queue_hdl, &mpt->m_acc_post_queue_hdl, &memp,
2994         mem_size, &cookie) == FALSE) {
2995         return (DDI_FAILURE);
2996     }
2997
2998     /*
2999     * Store the reply descriptor post queue memory address. This chip
3000     * uses this address to write to the reply descriptor post queue. The
3001     * second address is the address mpt uses to manage the queue.
3002     */
3003     mpt->m_post_queue_dma_addr = cookie.dmac_laddress;
3004     mpt->m_post_queue = memp;
3005
3006     mpt->m_rep_post_queues = kmem_zalloc(sizeof (mptsas_reply_pqueue_t) *
3007         mpt->m_post_reply_qcount, KM_SLEEP);
3008     rpqp = mpt->m_rep_post_queues;
3009     for (i = 0; i < mpt->m_post_reply_qcount; i++) {
3010         rpqp->rpq_queue = memp;
3011         mutex_init(&rpqp->rpq_mutex, NULL, MUTEX_DRIVER, NULL);
3012         rpqp->rpq_dlist.dl_tail = &rpqp->rpq_dlist.dl_q;
3013         rpqp->rpq_num = (uint8_t)i;
3014         memp += (mpt->m_post_queue_depth * 8);
3015         rpqp++;
3016     }
3017
3018     /*
3019     * Clear the reply post queue memory.
3020     */
3021     bzero(mpt->m_post_queue, mem_size);
3022
3023     mpt->m_dma_flags |= MPTSAS_POST_QUEUE;
3024     return (DDI_SUCCESS);
3025 }
3026
3027 static void
3028 mptsas_alloc_reply_args(mptsas_t *mpt)
3029 {
3030     if (mpt->m_replyh_args == NULL) {
3031         mpt->m_replyh_args = kmem_zalloc(sizeof (m_replyh_arg_t) *

```



```

3032         mpt->m_max_replies, KM_SLEEP);
3033     } else {
3034         bzero(mpt->m_replyh_args, sizeof (m_replyh_arg_t) *
3035             mpt->m_max_replies);
3036     }
3037 }

3039 static int
3040 mptsas_alloc_extra_sgl_frame(mptsas_t *mpt, mptsas_cmd_t *cmd)
3041 {
3042     mptsas_cache_frames_t *frames = NULL;
3043     if (cmd->cmd_extra_frames == NULL) {
3044         frames = kmem_cache_alloc(mpt->m_cache_frames, KM_NOSLEEP);
3045         if (frames == NULL) {
3046             return (DDI_FAILURE);
3047         }
3048         cmd->cmd_extra_frames = frames;
3049     }
3050     return (DDI_SUCCESS);
3051 }

3053 static void
3054 mptsas_free_extra_sgl_frame(mptsas_t *mpt, mptsas_cmd_t *cmd)
3055 {
3056     if (cmd->cmd_extra_frames) {
3057         kmem_cache_free(mpt->m_cache_frames,
3058             (void *)cmd->cmd_extra_frames);
3059         cmd->cmd_extra_frames = NULL;
3060     }
3061 }

3063 static void
3064 mptsas_cfg_fini(mptsas_t *mpt)
3065 {
3066     NDBG0(("mptsas_cfg_fini"));
3067     ddi_regs_map_free(&mpt->m_datap);
3068 }

3070 static void
3071 mptsas_hba_fini(mptsas_t *mpt)
3072 {
3073     NDBG0(("mptsas_hba_fini"));

3075     /*
3076      * Free up any allocated memory
3077      */
3078     if (mpt->m_dma_flags & MPTSAS_REQ_FRAME) {
3079         mptsas_dma_addr_destroy(&mpt->m_dma_req_frame_hdl,
3080             &mpt->m_acc_req_frame_hdl);
3081     }

3083     if (mpt->m_dma_flags & MPTSAS_REQ_SENSE) {
3084         rmfreemap(mpt->m_erqsense_map);
3085         mptsas_dma_addr_destroy(&mpt->m_dma_req_sense_hdl,
3086             &mpt->m_acc_req_sense_hdl);
3087     }

3089     if (mpt->m_dma_flags & MPTSAS_REPLY_FRAME) {
3090         mptsas_dma_addr_destroy(&mpt->m_dma_reply_frame_hdl,
3091             &mpt->m_acc_reply_frame_hdl);
3092     }

3094     if (mpt->m_dma_flags & MPTSAS_FREE_QUEUE) {
3095         mptsas_dma_addr_destroy(&mpt->m_dma_free_queue_hdl,
3096             &mpt->m_acc_free_queue_hdl);
3097     }

```

```

3099     mptsas_free_post_queue(mpt);

3101     if (mpt->m_replyh_args != NULL) {
3102         kmem_free(mpt->m_replyh_args, sizeof (m_replyh_arg_t)
3103             * mpt->m_max_replies);
3104     }
3105 }

3107 static int
3108 mptsas_name_child(dev_info_t *lun_dip, char *name, int len)
3109 {
3110     int lun = 0;
3111     char *sas_wwn = NULL;
3112     int phynum = -1;
3113     int reallen = 0;

3115     /* Get the target num */
3116     lun = ddi_prop_get_int(DDI_DEV_T_ANY, lun_dip, DDI_PROP_DONTPASS,
3117         LUN_PROP, 0);

3119     if ((phynum = ddi_prop_get_int(DDI_DEV_T_ANY, lun_dip,
3120         DDI_PROP_DONTPASS, "sata-phy", -1)) != -1) {
3121         /*
3122          * Stick in the address of form "pPHY,LUN"
3123          */
3124         reallen = snprintf(name, len, "p%x,%x", phynum, lun);
3125     } else if (ddi_prop_lookup_string(DDI_DEV_T_ANY, lun_dip,
3126         DDI_PROP_DONTPASS, SCSI_ADDR_PROP_TARGET_PORT, &sas_wwn)
3127         == DDI_PROP_SUCCESS) {
3128         /*
3129          * Stick in the address of the form "WWWN,LUN"
3130          */
3131         reallen = snprintf(name, len, "%s,%x", sas_wwn, lun);
3132         ddi_prop_free(sas_wwn);
3133     } else {
3134         return (DDI_FAILURE);
3135     }

3137     ASSERT(reallen < len);
3138     if (reallen >= len) {
3139         mptsas_log(0, CE_WARN, "!mptsas_get_name: name parameter "
3140             "length too small, it needs to be %d bytes", reallen + 1);
3141     }
3142     return (DDI_SUCCESS);
3143 }

3145 /*
3146  * tran_tgt_init(9E) - target device instance initialization
3147  */
3148 static int
3149 mptsas_scsi_tgt_init(dev_info_t *hba_dip, dev_info_t *tgt_dip,
3150     scsi_hba_tran_t *hba_tran, struct scsi_device *sd)
3151 {
3152     #ifndef __lock_lint
3153         _NOTE(ARGUNUSED(hba_tran))
3154     #endif

3156     /*
3157      * At this point, the scsi_device structure already exists
3158      * and has been initialized.
3159      *
3160      * Use this function to allocate target-private data structures,
3161      * if needed by this HBA. Add revised flow-control and queue
3162      * properties for child here, if desired and if you can tell they
3163      * support tagged queueing by now.

```

```

3164     */
3165     mptsas_t          *mpt;
3166     int               lun = sd->sd_address.a_lun;
3167     mdi_pathinfo_t    *pip = NULL;
3168     mptsas_tgt_private_t *tgt_private = NULL;
3169     mptsas_target_t    *tgt = NULL;
3170     char              *psas_wwn = NULL;
3171     mptsas_phymask_t    phymask = 0;
3172     uint64_t          sas_wwn = 0;
3173     mptsas_target_addr_t addr;
3174     mpt = SDEV2MPT(sd);

3176     ASSERT(scsi_hba_iport_unit_address(hba_dip) != 0);

3178     NDBG0(("mptsas_scsi_tgt_init: hbadip=0x%p tgt_dip=0x%p lun=%d",
3179         (void *)hba_dip, (void *)tgt_dip, lun));

3181     if (ndi_dev_is_persistent_node(tgt_dip) == 0) {
3182         (void) ndi_merge_node(tgt_dip, mptsas_name_child);
3183         ddi_set_name_addr(tgt_dip, NULL);
3184         return (DDI_FAILURE);
3185     }
3186     /*
3187     * phymask is 0 means the virtual port for RAID
3188     */
3189     phymask = (mptsas_phymask_t)ddi_prop_get_int(DDI_DEV_T_ANY, hba_dip, 0,
3190         "phymask", 0);
3191     if (mdi_component_is_client(tgt_dip, NULL) == MDI_SUCCESS) {
3192         if ((pip = (void *) (sd->sd_private)) == NULL) {
3193             /*
3194              * Very bad news if this occurs. Somehow scsi_vhci has
3195              * lost the pathinfo node for this target.
3196              */
3197             return (DDI_NOT_WELL_FORMED);
3198         }
3199     }
3200     if (mdi_prop_lookup_int(pip, LUN_PROP, &lun) !=
3201         DDI_PROP_SUCCESS) {
3202         mptsas_log(mpt, CE_WARN, "Get lun property failed\n");
3203         return (DDI_FAILURE);
3204     }

3206     if (mdi_prop_lookup_string(pip, SCSI_ADDR_PROP_TARGET_PORT,
3207         &psas_wwn) == MDI_SUCCESS) {
3208         if (scsi_wwnstr_to_wwn(psas_wwn, &sas_wwn)) {
3209             sas_wwn = 0;
3210         }
3211         (void) mdi_prop_free(psas_wwn);
3212     }
3213 } else {
3214     lun = ddi_prop_get_int(DDI_DEV_T_ANY, tgt_dip,
3215         DDI_PROP_DONTPASS, LUN_PROP, 0);
3216     if (ddi_prop_lookup_string(DDI_DEV_T_ANY, tgt_dip,
3217         DDI_PROP_DONTPASS, SCSI_ADDR_PROP_TARGET_PORT, &psas_wwn) ==
3218         DDI_PROP_SUCCESS) {
3219         if (scsi_wwnstr_to_wwn(psas_wwn, &sas_wwn)) {
3220             sas_wwn = 0;
3221         }
3222         ddi_prop_free(psas_wwn);
3223     } else {
3224         sas_wwn = 0;
3225     }
3226 }

3228     ASSERT((sas_wwn != 0) || (phymask != 0));
3229     addr.mta_wwn = sas_wwn;

```

```

3230     addr.mta_phymask = phymask;
3231     mutex_enter(&mpt->m_mutex);
3232     ptgt = reffhash_lookup(mpt->m_targets, &addr);
3233     mutex_exit(&mpt->m_mutex);
3234     if (ptgt == NULL) {
3235         mptsas_log(mpt, CE_WARN, "!tgt_init: target doesn't exist or "
3236             "gone already! phymask:%x, saswwn %"PRIx64, phymask,
3237             sas_wwn);
3238         return (DDI_FAILURE);
3239     }
3240     if (hba_tran->tran_tgt_private == NULL) {
3241         tgt_private = kmem_zalloc(sizeof (mptsas_tgt_private_t),
3242             KM_SLEEP);
3243         tgt_private->t_lun = lun;
3244         tgt_private->t_private = ptgt;
3245         hba_tran->tran_tgt_private = tgt_private;
3246     }

3248     if (mdi_component_is_client(tgt_dip, NULL) == MDI_SUCCESS) {
3249         return (DDI_SUCCESS);
3250     }
3251     mutex_enter(&mpt->m_mutex);

3253     if (ptgt->m_deviceinfo &
3254         (MPI2_SAS_DEVICE_INFO_SATA_DEVICE |
3255         MPI2_SAS_DEVICE_INFO_ATAPI_DEVICE)) {
3256         uchar_t *inq89 = NULL;
3257         int inq89_len = 0x238;
3258         int reallen = 0;
3259         int rval = 0;
3260         struct sata_id *sid = NULL;
3261         char model[SATA_ID_MODEL_LEN + 1];
3262         char fw[SATA_ID_FW_LEN + 1];
3263         char *vid, *pid;
3264         int i;

3266         mutex_exit(&mpt->m_mutex);
3267         /*
3268          * According SCSI/ATA Translation -2 (SAT-2) revision 01a
3269          * chapter 12.4.2 VPD page 89h includes 512 bytes ATA IDENTIFY
3270          * DEVICE data or ATA IDENTIFY PACKET DEVICE data.
3271          */
3272         inq89 = kmem_zalloc(inq89_len, KM_SLEEP);
3273         rval = mptsas_inquiry(mpt, ptgt, 0, 0x89,
3274             inq89, inq89_len, &reallen, 1);

3276         if (rval != 0) {
3277             if (inq89 != NULL) {
3278                 kmem_free(inq89, inq89_len);
3279             }

3281             mptsas_log(mpt, CE_WARN, "!mptsas request inquiry page "
3282                 "0x89 for SATA target:%x failed!", ptgt->m_devhdl);
3283             return (DDI_SUCCESS);
3284         }
3285         sid = (void *)(&inq89[60]);

3287         swab(sid->ai_model, model, SATA_ID_MODEL_LEN);
3288         swab(sid->ai_fw, fw, SATA_ID_FW_LEN);

3290         model[SATA_ID_MODEL_LEN] = 0;
3291         fw[SATA_ID_FW_LEN] = 0;

3293         /*
3294          * split model into into vid/pid
3295         */

```

```

3296         for (i = 0, pid = model; i < SATA_ID_MODEL_LEN; i++, pid++)
3297             if ((*pid == ' ') || (*pid == '\t'))
3298                 break;
3299         if (i < SATA_ID_MODEL_LEN) {
3300             vid = model;
3301             /*
3302              * terminate vid, establish pid
3303              */
3304             *pid++ = 0;
3305         } else {
3306             /*
3307              * vid will stay "ATA      ", the rule is same
3308              * as sata framework implementation.
3309              */
3310             vid = NULL;
3311             /*
3312              * model is all pid
3313              */
3314             pid = model;
3315         }
3317         /*
3318          * override SCSI "inquiry-" properties
3319          */
3320         if (vid)
3321             (void) scsi_device_prop_update_inqstring(sd,
3322             INQUIRY_VENDOR_ID, vid, strlen(vid));
3323         if (pid)
3324             (void) scsi_device_prop_update_inqstring(sd,
3325             INQUIRY_PRODUCT_ID, pid, strlen(pid));
3326         (void) scsi_device_prop_update_inqstring(sd,
3327             INQUIRY_REVISION_ID, fw, strlen(fw));
3329         if (inq89 != NULL) {
3330             kmem_free(inq89, inq89_len);
3331         }
3332     } else {
3333         mutex_exit(&mpt->m_mutex);
3334     }
3336     return (DDI_SUCCESS);
3337 }
3338 /*
3339  * tran_tgt_free(9E) - target device instance deallocation
3340  */
3341 static void
3342 mptsas_scsi_tgt_free(dev_info_t *hba_dip, dev_info_t *tgt_dip,
3343     scsi_hba_tran_t *hba_tran, struct scsi_device *sd)
3344 {
3345     #ifndef __lock_lint
3346         _NOTE(ARGUNUSED(hba_dip, tgt_dip, hba_tran, sd))
3347     #endif
3349     mptsas_tgt_private_t    *tgt_private = hba_tran->tran_tgt_private;
3351     if (tgt_private != NULL) {
3352         kmem_free(tgt_private, sizeof (mptsas_tgt_private_t));
3353         hba_tran->tran_tgt_private = NULL;
3354     }
3355 }
3357 /*
3358  * scsi_pkt handling
3359  *
3360  * Visible to the external world via the transport structure.
3361  */

```

```

3363 /*
3364  * Notes:
3365  *   - transport the command to the addressed SCSI target/lun device
3366  *   - normal operation is to schedule the command to be transported,
3367  *   - and return TRAN_ACCEPT if this is successful.
3368  *   - if NO_INTR, tran_start must poll device for command completion
3369  */
3370 static int
3371 mptsas_scsi_start(struct scsi_address *ap, struct scsi_pkt *pkt)
3372 {
3373     #ifndef __lock_lint
3374         _NOTE(ARGUNUSED(ap))
3375     #endif
3376     mptsas_t            *mpt = PKT2MPT(pkt);
3377     mptsas_cmd_t        *cmd = PKT2CMD(pkt);
3378     int                 rval, start;
3379     uint8_t             pref;
3380     mptsas_target_t     *tgt = cmd->cmd_tgt_addr;
3381     mptsas_tx_waitqueue_t *txwq;
3383     NDBG1(("mptsas_scsi_start: pkt=0x%p", (void *)pkt));
3384     ASSERT(tgt);
3385     if (tgt == NULL)
3386         return (TRAN_FATAL_ERROR);
3388     /*
3389      * prepare the pkt before taking mutex.
3390      */
3391     rval = mptsas_prepare_pkt(cmd);
3392     if (rval != TRAN_ACCEPT) {
3393         return (rval);
3394     }
3396     /*
3397      * Send the command to target/lun, however your HBA requires it.
3398      * If busy, return TRAN_BUSY; if there's some other formatting error
3399      * in the packet, return TRAN_BADPKT; otherwise, fall through to the
3400      * return of TRAN_ACCEPT.
3401      *
3402      * Remember that access to shared resources, including the mptsas_t
3403      * data structure and the HBA hardware registers, must be protected
3404      * with mutexes, here and everywhere.
3405      *
3406      * Also remember that at interrupt time, you'll get an argument
3407      * to the interrupt handler which is a pointer to your mptsas_t
3408      * structure; you'll have to remember which commands are outstanding
3409      * and which scsi_pkt is the currently-running command so the
3410      * interrupt handler can refer to the pkt to set completion
3411      * status, call the target driver back through pkt_comp, etc.
3412      *
3413      * If the instance lock is held by other thread, don't spin to wait
3414      * for it. Instead, queue the cmd and next time when the instance lock
3415      * is not held, accept all the queued cmd. A extra tx_waitq is
3416      * introduced to protect the queue.
3417      *
3418      * The polled cmd will not be queued and accepted as usual.
3419      *
3420      * Under the tx_waitq mutex, record whether a thread is draining
3421      * the tx_waitq. An IO requesting thread that finds the instance
3422      * mutex contended appends to the tx_waitq and while holding the
3423      * tx_wait mutex, if the draining flag is not set, sets it and then
3424      * proceeds to spin for the instance mutex. This scheme ensures that
3425      * the last cmd in a burst be processed.
3426      *
3427      * we enable this feature only when the helper threads are enabled,

```

```

3428      * at which we think the loads are heavy.
3429      *
3430      * per instance, per queue mutex m_tx_waitq[i].txwq_mutex is
3431      * introduced to protect the txwq_qtail, txwq_cmdq, txwq_len
3432      */
3433
3434      if (mpt->m_txwq_enabled == TRUE) {
3435          int gotmtx = 0;
3436
3437          if (mpt->m_txwq_allow_q_jumping) {
3438              gotmtx = mutex_tryenter(&mpt->m_mutex);
3439          }
3440          if (gotmtx == 0) {
3441              /* We didn't get the mutex or didn't try */
3442              if (cmd->cmd_pkt_flags & FLAG_NOINTR) {
3443                  mutex_enter(&mpt->m_mutex);
3444                  /* Polled commands queue jump */
3445                  mptsas_accept_tx_waitqs(mpt);
3446              } else {
3447                  rval = mptsas_check_targ_intxtn(
3448                      cmd->cmd_tgt_addr,
3449                      cmd->cmd_pkt_flags);
3450                  if (rval != TRAN_ACCEPT) {
3451                      return (rval);
3452                  }
3453
3454                  cmd->cmd_flags |= CFLAG_TXQ;
3455                  pref = mpt->m_pref_tx_waitq;
3456                  txwq = &mpt->m_tx_waitq[pref];
3457
3458                  if (mutex_tryenter(&txwq->txwq_mutex) == 0) {
3459                      txwq = &mpt->m_tx_waitq[pref^1];
3460                      mutex_enter(&txwq->txwq_mutex);
3461                  } else {
3462                      pref ^= 1;
3463                      mpt->m_pref_tx_waitq = pref;
3464                  }
3465
3466                  *txwq->txwq_qtail = cmd;
3467                  txwq->txwq_qtail = &cmd->cmd_linkp;
3468                  txwq->txwq_len++;
3469                  if (!txwq->txwq_draining) {
3470                      cv_signal(&txwq->txwq_cv);
3471                  }
3472                  mutex_exit(&txwq->txwq_mutex);
3473                  return (rval);
3474              }
3475          }
3476          } else {
3477              mutex_enter(&mpt->m_mutex);
3478          }
3479          rval = mptsas_check_targ_intxtn(cmd->cmd_tgt_addr,
3480              cmd->cmd_pkt_flags);
3481          if (rval != TRAN_ACCEPT) {
3482              mutex_exit(&mpt->m_mutex);
3483              return (rval);
3484          }
3485
3486          start = mptsas_accept_pkt(mpt, cmd, &rval);
3487          mutex_exit(&mpt->m_mutex);
3488          if (start) {
3489              (void) mptsas_start_cmd(mpt, cmd);
3490          }
3491
3492          return (rval);
3493      }

```

```

3495 static int
3496 mptsas_check_targ_intxtn(mptsas_target_t *ptgt, int cmd_pkt_flags)
3497 {
3498     /*
3499      * ptgt->m_dr_flag is a variable that is only ever changed by
3500      * direct write under the main m_mutex.
3501      * It doesn't need a mutex hold to protect this read.
3502      */
3503
3504     if (ptgt->m_dr_flag == MPTSAS_DR_INTRANSITION) {
3505         if (cmd_pkt_flags & FLAG_NOQUEUE) {
3506             /*
3507              * The command should be allowed to retry by returning
3508              * TRAN_BUSY to stall the I/O's which come from
3509              * scsi_vhci since the device/path is in unstable state
3510              * now.
3511              */
3512             return (TRAN_BUSY);
3513         } else {
3514             /*
3515              * The device is offline, just fail the command by
3516              * return TRAN_FATAL_ERROR.
3517              */
3518             return (TRAN_FATAL_ERROR);
3519         }
3520     }
3521     return (TRAN_ACCEPT);
3522 }
3523
3524 #if 0
3525 /*
3526  * Note that this function has a side effect of releasing the
3527  * per target mutex.
3528  */
3529 static void
3530 mptsas_offline_target_direct(mptsas_t *mpt, mptsas_target_t *ptgt)
3531 {
3532     char phy_mask_name[MPTSAS_MAX_PHYS];
3533     mptsas_phymask_t phymask = ptgt->m_addr.mta_phymask;
3534     dev_info_t *parent;
3535
3536     ASSERT(mutex_owned(&mpt->m_mutex));
3537
3538     ptgt->m_dr_flag = MPTSAS_DR_INTRANSITION;
3539     bzero(phy_mask_name, MPTSAS_MAX_PHYS);
3540     (void) sprintf(phy_mask_name, "%x", phymask);
3541     parent = scsi_hba_iport_find(mpt->m_dip, phy_mask_name);
3542
3543     if (parent != NULL) {
3544         mptsas_offline_target(mpt, ptgt,
3545             ptgt->m_deviceinfo & DEVINFO_DIRECT_ATTACHED ?
3546             MPTSAS_TOPO_FLAG_DIRECT_ATTACHED_DEVICE :
3547             MPTSAS_TOPO_FLAG_EXPANDER_ATTACHED_DEVICE,
3548             parent);
3549     } else {
3550         mptsas_log(mpt, CE_WARN, "Failed to find an "
3551             "iport for \"%s\", should not happen!",
3552             phy_mask_name);
3553     }
3554 }
3555 #endif
3556
3557 /*
3558  * In order to be efficient with the m_mutex (which can be dropped before
3559  * calling mptsas_start_cmd()) indicate if start_cmd should be called via the

```

```

3560 * returned value (FALSE or TRUE). Caller is then responsible for doing the
3561 * right thing with the m_mutex.
3562 */
3563 static int
3564 mptsas_accept_pkt(mptsas_t *mpt, mptsas_cmd_t *cmd, int *tran_rval)
3565 {
3566     int            rval = TRAN_ACCEPT;
3567     mptsas_target_t *ptgt = cmd->cmd_tgt_addr;
3568
3569     NDBG1(("mptsas_accept_pkt: cmd=0x%p", (void *)cmd));
3570
3571     ASSERT(mutex_owned(&mpt->m_mutex));
3572
3573     if ((cmd->cmd_flags & CFLAG_PREPARED) == 0) {
3574         rval = mptsas_prepare_pkt(cmd);
3575         if (rval != TRAN_ACCEPT) {
3576             cmd->cmd_flags &= ~CFLAG_TRANFLAG;
3577             goto set_tranrval;
3578         }
3579     }
3580
3581     /*
3582      * If the command came from the tx wait q it may have slipped
3583      * by the check for dr_flag before being added to the queue.
3584      * Fail here with abort status.
3585      */
3586     if (cmd->cmd_flags & CFLAG_TXQ) {
3587         rval = mptsas_check_targ_intxtion(cmd->cmd_tgt_addr,
3588             cmd->cmd_pkt_flags);
3589         if (rval != TRAN_ACCEPT) {
3590             mptsas_set_pkt_reason(mpt, cmd, CMD_ABORTED,
3591                 STAT_ABORTED);
3592             mptsas_doneq_add(mpt, cmd);
3593             mptsas_doneq_empty(mpt);
3594             goto set_tranrval;
3595         }
3596     }
3597     /*
3598      * If HBA is being reset, the DevHandles are being re-initialized,
3599      * which means that they could be invalid even if the target is still
3600      * attached. Check if being reset and if DevHandle is being
3601      * re-initialized. If this is the case, return BUSY so the I/O can be
3602      * retried later.
3603      */
3604     if ((ptgt->m_devhdl == MPTSAS_INVALID_DEVHDL) && mpt->m_in_reset) {
3605         mptsas_set_pkt_reason(mpt, cmd, CMD_RESET, STAT_BUS_RESET);
3606         if (cmd->cmd_flags & CFLAG_TXQ) {
3607             mptsas_doneq_add(mpt, cmd);
3608             mptsas_doneq_empty(mpt);
3609         } else {
3610             rval = TRAN_BUSY;
3611         }
3612         goto set_tranrval;
3613     }
3614
3615     mutex_enter(&ptgt->m_t_mutex);
3616     /*
3617      * reset the throttle if we were draining
3618      */
3619     if ((ptgt->m_t_ncmds == 0) &&
3620         (ptgt->m_t_throttle == DRAIN_THROTTLE)) {
3621         NDBG23(("reset throttle"));
3622         ASSERT(ptgt->m_reset_delay == 0);
3623         mptsas_set_throttle(mpt, ptgt, MAX_THROTTLE);
3624     }

```

```

3626     /*
3627      * If device handle has already been invalidated, just
3628      * fail the command. In theory, for a command from scsi_vhci
3629      * client it's impossible to receive a command with an invalid
3630      * devhdl since devhdl is set after path offline, target
3631      * driver is not supposed to select an offlined path.
3632      */
3633     if (ptgt->m_devhdl == MPTSAS_INVALID_DEVHDL) {
3634         NDBG3(("rejecting command, it might because invalid devhdl "
3635             "request."));
3636         mutex_exit(&ptgt->m_t_mutex);
3637         mptsas_set_pkt_reason(mpt, cmd, CMD_DEV_GONE, STAT_TERMINATED);
3638         if (cmd->cmd_flags & CFLAG_TXQ) {
3639             mptsas_doneq_add(mpt, cmd);
3640             mptsas_doneq_empty(mpt);
3641         } else {
3642             rval = TRAN_FATAL_ERROR;
3643         }
3644         goto set_tranrval;
3645     }
3646     /*
3647      * The first case is the normal case. mpt gets a command from the
3648      * target driver and starts it.
3649      * Since SMID 0 is reserved and the TM slot is reserved, the actual max
3650      * commands is m_max_requests - 2.
3651      */
3652     if ((mpt->m_ncmds <= (mpt->m_max_requests - 2)) &&
3653         (ptgt->m_t_throttle > HOLD_THROTTLE) &&
3654         (ptgt->m_t_ncmds < ptgt->m_t_throttle) &&
3655         (ptgt->m_reset_delay == 0) && (mpt->m_polled_intr == 0) &&
3656         (ptgt->m_t_nwait == 0) &&
3657         ((cmd->cmd_pkt_flags & FLAG_NOINTR) == 0)) {
3658         ASSERT((cmd->cmd_flags & CFLAG_CMDIOC) == 0);
3659         if (mptsas_save_cmd_to_slot(mpt, cmd) == TRUE) {
3660             ptgt->m_t_ncmds++;
3661             mutex_exit(&ptgt->m_t_mutex);
3662             cmd->cmd_active_expiration = 0;
3663             *tran_rval = rval;
3664             return (TRUE);
3665         } else {
3666             mutex_exit(&ptgt->m_t_mutex);
3667             mptsas_waitq_add(mpt, cmd);
3668         }
3669     } else {
3670         mutex_exit(&ptgt->m_t_mutex);
3671         /*
3672          * Add this pkt to the work queue
3673          */
3674         mptsas_waitq_add(mpt, cmd);
3675
3676         if (cmd->cmd_pkt_flags & FLAG_NOINTR) {
3677             (void) mptsas_poll(mpt, cmd, MPTSAS_POLL_TIME);
3678         }
3679         /*
3680          * Only flush the doneq if this is not a TM
3681          * cmd. For TM cmds the flushing of the
3682          * doneq will be done in those routines.
3683          */
3684         if ((cmd->cmd_flags & CFLAG_TM_CMD) == 0) {
3685             mptsas_doneq_empty(mpt);
3686         }
3687     }
3688     }
3689     set_tranrval:
3690     *tran_rval = rval;
3691     return (FALSE);

```

```

3692 }

3694 static void
3695 mptsas_retry_pkt(mptsas_t *mpt, mptsas_cmd_t *cmd)
3696 {
3697     int            rval;

3699     cmd->cmd_pkt_flags |= FLAG_HEAD;
3700     cmd->cmd_flags |= CFLAG_RETRY;
3701     cmd->cmd_flags &= ~CFLAG_TXQ;
3702     if (mptsas_accept_pkt(mpt, cmd, &rval)) {
3703         (void) mptsas_start_cmd(mpt, cmd);
3704     }

3706     /*
3707      * If there was a problem clear the retry flag so that the
3708      * command will be completed with error rather than get lost!
3709      */
3710     if (rval != TRAN_ACCEPT)
3711         cmd->cmd_flags &= ~CFLAG_RETRY;
3712 }

3714 static int
3715 mptsas_save_cmd_to_slot(mptsas_t *mpt, mptsas_cmd_t *cmd)
3716 {
3717     mptsas_slots_t *slots = mpt->m_active;
3718     uint_t slot, start_rotor, rotor, n_normal;

3720     /*
3721      * Account for reserved TM request slot and reserved SMID of 0.
3722      */
3723     ASSERT(slots->m_n_normal == (mpt->m_max_requests - 2));

3725     /*
3726      * Find the next available slot, beginning at m_rotor. If no slot is
3727      * available, we'll return FALSE to indicate that. This mechanism
3728      * considers only the normal slots, not the reserved slot 0 nor the
3729      * task management slot m_n_normal + 1. The rotor is left to point to
3730      * the normal slot after the one we select, unless we select the last
3731      * normal slot in which case it returns to slot 1.
3732      */
3733     start_rotor = rotor = slots->m_rotor;
3734     n_normal = slots->m_n_normal;
3735     do {
3736         slot = rotor++;
3737         if (rotor > n_normal)
3738             rotor = 1;

3740         if (rotor == start_rotor)
3741             break;
3742     } while (slots->m_slot[slot] != NULL);
3743     slots->m_rotor = rotor;

3745     if (slots->m_slot[slot] != NULL)
3746         return (FALSE);

3748     ASSERT(slot != 0 && slot <= slots->m_n_normal);

3750     cmd->cmd_slot = slot;
3751     slots->m_slot[slot] = cmd;
3752     atomic_inc_32(&mpt->m_ncmds);

3754     /*
3755      * Distribute the commands amongst the reply queues (Interrupt vectors).
3756      * Stick to 0 for polled.
3757      */

```

```

3758     if (!(cmd->cmd_pkt_flags & FLAG_NOINTR) &&
3759         !(cmd->cmd_flags & (CFLAG_PASSTHRU|CFLAG_CONFIG|CFLAG_FW_DIAG)) &&
3760         (mpt->m_post_reply_qcount > 1)) {
3761         cmd->cmd_rpqidx = slot % mpt->m_post_reply_qcount;
3762     }
3763     atomic_inc_32(&mpt->m_rep_post_queues[cmd->cmd_rpqidx].rpq_ncmds);
3764     return (TRUE);
3765 }

3767 int
3768 mptsas_save_cmd(mptsas_t *mpt, mptsas_cmd_t *cmd)
3769 {
3770     mptsas_target_t *ptgt = cmd->cmd_tgt_addr;

3772     ASSERT(MUTEX_HELD(&mpt->m_mutex));

3774     if (!mptsas_save_cmd_to_slot(mpt, cmd)) {
3775         return (FALSE);
3776     }

3778     /*
3779      * only increment per target ncmds if this is not a
3780      * command that has no target associated with it (i.e. a
3781      * event acknowledgement)
3782      */
3783     if ((cmd->cmd_flags & CFLAG_CMDIOC) == 0) {
3784         /*
3785          * Expiration time is set in mptsas_start_cmd
3786          */
3787         mutex_enter(&ptgt->m_t_mutex);
3788         ptgt->m_t_ncmds++;
3789         mutex_exit(&ptgt->m_t_mutex);
3790         cmd->cmd_active_expiration = 0;
3791     } else {
3792         /*
3793          * Initialize expiration time for passthrough commands,
3794          */
3795         cmd->cmd_active_expiration = gethrtime() +
3796             (hrtime_t)cmd->cmd_pkt->pkt_time * NANOSEC;
3797     }
3798     return (TRUE);
3799 }

3801 /*
3802  * prepare the pkt:
3803  * the pkt may have been resubmitted or just reused so
3804  * initialize some fields and do some checks.
3805  */
3806 static int
3807 mptsas_prepare_pkt(mptsas_cmd_t *cmd)
3808 {
3809     struct scsi_pkt *pkt = CMD2PKT(cmd);

3811     NDBG1(("mptsas_prepare_pkt: cmd=0x%p", (void *)cmd));

3813     /*
3814      * Reinitialize some fields that need it; the packet may
3815      * have been resubmitted
3816      */
3817     pkt->pkt_reason = CMD_CMPLT;
3818     pkt->pkt_state = 0;
3819     pkt->pkt_statistics = 0;
3820     pkt->pkt_resid = 0;
3821     cmd->cmd_age = 0;
3822     cmd->cmd_pkt_flags = pkt->pkt_flags;

```

```

3824 /*
3825  * zero status byte.
3826  */
3827 *(pkt->pkt_scbp) = 0;

3829 if (cmd->cmd_flags & CFLAG_DMAVALID) {
3830     pkt->pkt_resid = cmd->cmd_dmacount;

3832     /*
3833      * consistent packets need to be sync'ed first
3834      * (only for data going out)
3835      */
3836     if ((cmd->cmd_flags & CFLAG_CMDIOPB) &&
3837         (cmd->cmd_flags & CFLAG_DMASEND)) {
3838         (void) ddi_dma_sync(cmd->cmd_dmahandle, 0, 0,
3839             DDI_DMA_SYNC_FORDEV);
3840     }
3841 }

3843 cmd->cmd_flags =
3844     (cmd->cmd_flags & ~(CFLAG_TRANFLAG)) |
3845     CFLAG_PREPARED | CFLAG_IN_TRANSPORT;

3847 return (TRAN_ACCEPT);
3848 }

3850 /*
3851  * tran_init_pkt(9E) - allocate scsi_pkt(9S) for command
3852  *
3853  * One of three possibilities:
3854  *   - allocate scsi_pkt
3855  *   - allocate scsi_pkt and DMA resources
3856  *   - allocate DMA resources to an already-allocated pkt
3857  */
3858 static struct scsi_pkt *
3859 mptsas_scsi_init_pkt(struct scsi_address *ap, struct scsi_pkt *pkt,
3860     struct buf *bp, int cmdlen, int statuslen, int tgtlen, int flags,
3861     int (*callback)(), caddr_t arg)
3862 {
3863     mptsas_cmd_t      *cmd, *new_cmd;
3864     mptsas_t          *mpt = ADDR2MPT(ap);
3865     int                failure = 1;
3866     uint_t            oldcookiec;
3867     mptsas_target_t   *ptgt = NULL;
3868     int               rval;
3869     mptsas_tgt_private_t *tgt_private;
3870     int               kf;

3872     kf = (callback == SLEEP_FUNC)? KM_SLEEP: KM_NOSLEEP;

3874     tgt_private = (mptsas_tgt_private_t *)ap->a_hba_tran->
3875         tran_tgt_private;
3876     ASSERT(tgt_private != NULL);
3877     if (tgt_private == NULL) {
3878         return (NULL);
3879     }
3880     ptgt = tgt_private->t_private;
3881     ASSERT(ptgt != NULL);
3882     if (ptgt == NULL)
3883         return (NULL);
3884     ap->a_target = ptgt->m_devhdl;
3885     ap->a_lun = tgt_private->t_lun;

3887     ASSERT(callback == NULL_FUNC || callback == SLEEP_FUNC);
3888 #ifdef MPTSAS_TEST_EXTRN_ALLOC
3889     statuslen *= 100; tgtlen *= 4;

```

```

3890 #endif
3891 NDBG3(("mptsas_scsi_init_pkt:\n"
3892     "\ttgt=%d in=0x%p bp=0x%p clen=%d slen=%d tlen=%d flags=%x",
3893     ap->a_target, (void *)pkt, (void *)bp,
3894     cmdlen, statuslen, tgtlen, flags));

3896 /*
3897  * Allocate the new packet.
3898  */
3899 if (pkt == NULL) {
3900     ddi_dma_handle_t    save_dma_handle;

3902     cmd = kmem_cache_alloc(mpt->m_kmem_cache, kf);

3904     if (cmd) {
3905         save_dma_handle = cmd->cmd_dmahandle;
3906         bzero(cmd, sizeof (*cmd) + scsi_pkt_size());
3907         cmd->cmd_dmahandle = save_dma_handle;

3909         pkt = (void *)((uchar_t *)cmd +
3910             sizeof (struct mptsas_cmd));
3911         pkt->pkt_ha_private = (opaque_t)cmd;
3912         pkt->pkt_address = *ap;
3913         pkt->pkt_private = (opaque_t)cmd->cmd_pkt_private;
3914         pkt->pkt_scbp = (opaque_t)&cmd->cmd_scb;
3915         pkt->pkt_cdbp = (opaque_t)&cmd->cmd_cdb;
3916         cmd->cmd_pkt = (struct scsi_pkt *)pkt;
3917         cmd->cmd_cdblen = (uchar_t)cmdlen;
3918         cmd->cmd_scblen = statuslen;
3919         cmd->cmd_rqslen = SENSE_LENGTH;
3920         cmd->cmd_tgt_addr = ptgt;
3921         failure = 0;
3922     }

3924     if (failure || (cmdlen > sizeof (cmd->cmd_cdb)) ||
3925         (tgtlen > PKT_PRIV_LEN) ||
3926         (statuslen > EXTCMDSTATUS_SIZE)) {
3927         if (failure == 0) {
3928             /*
3929              * if extern alloc fails, all will be
3930              * deallocated, including cmd
3931              */
3932             failure = mptsas_pkt_alloc_extern(mpt, cmd,
3933                 cmdlen, tgtlen, statuslen, kf);
3934         }
3935         if (failure) {
3936             /*
3937              * if extern allocation fails, it will
3938              * deallocate the new pkt as well
3939              */
3940             return (NULL);
3941         }
3942     }
3943     new_cmd = cmd;

3945 } else {
3946     cmd = PKT2CMD(pkt);
3947     new_cmd = NULL;
3948 }

3951 /* grab cmd->cmd_cookiec here as oldcookiec */

3953 oldcookiec = cmd->cmd_cookiec;

3955 /*

```

```

3956      * If the dma was broken up into PARTIAL transfers cmd_nwin will be
3957      * greater than 0 and we'll need to grab the next dma window
3958      */
3959      /*
3960      * SLM-not doing extra command frame right now; may add later
3961      */

3963      if (cmd->cmd_nwin > 0) {

3965          /*
3966          * Make sure we havn't gone past the the total number
3967          * of windows
3968          */
3969          if (++cmd->cmd_winindex >= cmd->cmd_nwin) {
3970              return (NULL);
3971          }
3972          if (ddi_dma_getwin(cmd->cmd_dmahandle, cmd->cmd_winindex,
3973              &cmd->cmd_dma_offset, &cmd->cmd_dma_len,
3974              &cmd->cmd_cookie, &cmd->cmd_cookiec) == DDI_FAILURE) {
3975              return (NULL);
3976          }
3977          goto get_dma_cookies;
3978      }

3981      if (flags & PKT_XARQ) {
3982          cmd->cmd_flags |= CFLAG_XARQ;
3983      }

3985      /*
3986      * DMA resource allocation. This version assumes your
3987      * HBA has some sort of bus-mastering or onboard DMA capability, with a
3988      * scatter-gather list of length MPTSAS_MAX_DMA_SEGS, as given in the
3989      * ddi_dma_attr_t structure and passed to scsi_impl_dmaget.
3990      */
3991      if (bp && (bp->b_bcount != 0) &&
3992          (cmd->cmd_flags & CFLAG_DMAVALID) == 0) {

3994          int      cnt, dma_flags;
3995          mptti_t *dmap;          /* ptr to the S/G list */

3997          /*
3998          * Set up DMA memory and position to the next DMA segment.
3999          */
4000          ASSERT(cmd->cmd_dmahandle != NULL);

4002          if (bp->b_flags & B_READ) {
4003              dma_flags = DDI_DMA_READ;
4004              cmd->cmd_flags &= ~CFLAG_DMASEND;
4005          } else {
4006              dma_flags = DDI_DMA_WRITE;
4007              cmd->cmd_flags |= CFLAG_DMASEND;
4008          }
4009          if (flags & PKT_CONSISTENT) {
4010              cmd->cmd_flags |= CFLAG_CMDIOPB;
4011              dma_flags |= DDI_DMA_CONSISTENT;
4012          }

4014          if (flags & PKT_DMA_PARTIAL) {
4015              dma_flags |= DDI_DMA_PARTIAL;
4016          }

4018          /*
4019          * workaround for byte hole issue on psycho and
4020          * schizo pre 2.1
4021          */

```

```

4022      if ((bp->b_flags & B_READ) && ((bp->b_flags &
4023          (B_PAGEIO|B_REMAPPED)) != B_PAGEIO) &&
4024          ((uintptr_t)bp->b_un.b_addr & 0x7)) {
4025          dma_flags |= DDI_DMA_CONSISTENT;
4026      }

4028      rval = ddi_dma_buf_bind_handle(cmd->cmd_dmahandle, bp,
4029          dma_flags, callback, arg,
4030          &cmd->cmd_cookie, &cmd->cmd_cookiec);
4031      if (rval == DDI_DMA_PARTIAL_MAP) {
4032          (void) ddi_dma_numwin(cmd->cmd_dmahandle,
4033              &cmd->cmd_nwin);
4034          cmd->cmd_winindex = 0;
4035          (void) ddi_dma_getwin(cmd->cmd_dmahandle,
4036              cmd->cmd_winindex, &cmd->cmd_dma_offset,
4037              &cmd->cmd_dma_len, &cmd->cmd_cookie,
4038              &cmd->cmd_cookiec);
4039      } else if (rval && (rval != DDI_DMA_MAPPED)) {
4040          switch (rval) {
4041              case DDI_DMA_NORESOURCES:
4042                  bioerror(bp, 0);
4043                  break;
4044              case DDI_DMA_BADATTR:
4045              case DDI_DMA_NOMAPPING:
4046                  bioerror(bp, EFAULT);
4047                  break;
4048              case DDI_DMA_TOOBIG:
4049              default:
4050                  bioerror(bp, EINVAL);
4051                  break;
4052          }
4053          cmd->cmd_flags &= ~CFLAG_DMAVALID;
4054          if (new_cmd) {
4055              mptsas_scsi_destroy_pkt(ap, pkt);
4056          }
4057          return ((struct scsi_pkt *)NULL);
4058      }

4060      get_dma_cookies:
4061          cmd->cmd_flags |= CFLAG_DMAVALID;
4062          ASSERT(cmd->cmd_cookiec > 0);

4064          if (cmd->cmd_cookiec > MPTSAS_MAX_CMD_SEGS) {
4065              mptsas_log(mpt, CE_NOTE, "large cookiec received %d\n",
4066                  cmd->cmd_cookiec);
4067              bioerror(bp, EINVAL);
4068              if (new_cmd) {
4069                  mptsas_scsi_destroy_pkt(ap, pkt);
4070              }
4071              return ((struct scsi_pkt *)NULL);
4072          }

4074          /*
4075          * Allocate extra SGL buffer if needed.
4076          */
4077          if ((cmd->cmd_cookiec > MPTSAS_MAX_FRAME_SGES64(mpt)) &&
4078              (cmd->cmd_extra_frames == NULL)) {
4079              if (mptsas_alloc_extra_sgl_frame(mpt, cmd) ==
4080                  DDI_FAILURE) {
4081                  mptsas_log(mpt, CE_WARN, "MPT SGL mem alloc "
4082                      "failed");
4083                  bioerror(bp, ENOMEM);
4084                  if (new_cmd) {
4085                      mptsas_scsi_destroy_pkt(ap, pkt);
4086                  }
4087                  return ((struct scsi_pkt *)NULL);

```



```

4088     }
4089 }
4091 /*
4092  * Always use scatter-gather transfer
4093  * Use the loop below to store physical addresses of
4094  * DMA segments, from the DMA cookies, into your HBA's
4095  * scatter-gather list.
4096  * We need to ensure we have enough kmem alloc'd
4097  * for the sg entries since we are no longer using an
4098  * array inside mptsas_cmd_t.
4099  *
4100  * We check cmd->cmd_cookiec against oldcookiec so
4101  * the scatter-gather list is correctly allocated
4102  */
4104 if (oldcookiec != cmd->cmd_cookiec) {
4105     if (cmd->cmd_sg != (mptti_t *)NULL) {
4106         kmem_free(cmd->cmd_sg, sizeof (mptti_t) *
4107             oldcookiec);
4108         cmd->cmd_sg = NULL;
4109     }
4110 }
4112 if (cmd->cmd_sg == (mptti_t *)NULL) {
4113     cmd->cmd_sg = kmem_alloc((sizeof (mptti_t) *
4114         cmd->cmd_cookiec), kf);
4116     if (cmd->cmd_sg == (mptti_t *)NULL) {
4117         mptsas_log(mpt, CE_WARN,
4118             "unable to kmem_alloc enough memory "
4119             "for scatter/gather list");
4120     }
4121     /*
4122     * if we have an ENOMEM condition we need to behave
4123     * the same way as the rest of this routine
4124     */
4125     bioerror(bp, ENOMEM);
4126     if (new_cmd) {
4127         mptsas_scsi_destroy_pkt(ap, pkt);
4128     }
4129     return ((struct scsi_pkt *)NULL);
4130 }
4131 }
4133 dmap = cmd->cmd_sg;
4135 ASSERT(cmd->cmd_cookie.dmac_size != 0);
4137 /*
4138  * store the first segment into the S/G list
4139  */
4140 dmap->count = cmd->cmd_cookie.dmac_size;
4141 dmap->addr.address64.Low = (uint32_t)
4142     (cmd->cmd_cookie.dmac_laddress & 0xfffffffffull);
4143 dmap->addr.address64.High = (uint32_t)
4144     (cmd->cmd_cookie.dmac_laddress >> 32);
4146 /*
4147  * dmacount counts the size of the dma for this window
4148  * (if partial dma is being used). totaldmacount
4149  * keeps track of the total amount of dma we have
4150  * transferred for all the windows (needed to calculate
4151  * the resid value below).
4152  */
4153 cmd->cmd_dmacount = cmd->cmd_cookie.dmac_size;

```

```

4154 cmd->cmd_totaldmacount += cmd->cmd_cookie.dmac_size;
4156 /*
4157  * We already stored the first DMA scatter gather segment,
4158  * start at 1 if we need to store more.
4159  */
4160 for (cnt = 1; cnt < cmd->cmd_cookiec; cnt++) {
4161     /*
4162     * Get next DMA cookie
4163     */
4164     ddi_dma_nextcookie(cmd->cmd_dmahandle,
4165         &cmd->cmd_cookie);
4166     dmap++;
4168     cmd->cmd_dmacount += cmd->cmd_cookie.dmac_size;
4169     cmd->cmd_totaldmacount += cmd->cmd_cookie.dmac_size;
4171     /*
4172     * store the segment parms into the S/G list
4173     */
4174     dmap->count = cmd->cmd_cookie.dmac_size;
4175     dmap->addr.address64.Low = (uint32_t)
4176         (cmd->cmd_cookie.dmac_laddress & 0xfffffffffull);
4177     dmap->addr.address64.High = (uint32_t)
4178         (cmd->cmd_cookie.dmac_laddress >> 32);
4179 }
4181 /*
4182  * If this was partially allocated we set the resid
4183  * the amount of data NOT transferred in this window
4184  * If there is only one window, the resid will be 0
4185  */
4186 pkt->pkt_resid = (bp->b_bcount - cmd->cmd_totaldmacount);
4187 NDBG3(("mptsas_scsi_init_pkt: cmd_dmacount=%d.",
4188     cmd->cmd_dmacount));
4189 }
4190 return (pkt);
4191 }
4193 /*
4194  * tran_destroy_pkt(9E) - scsi_pkt(9s) deallocation
4195  *
4196  * Notes:
4197  * - also frees DMA resources if allocated
4198  * - implicit DMA synchronization
4199  */
4200 static void
4201 mptsas_scsi_destroy_pkt(struct scsi_address *ap, struct scsi_pkt *pkt)
4202 {
4203     mptsas_cmd_t *cmd = PKT2CMD(pkt);
4204     mptsas_t *mpt = ADDR2MPT(ap);
4206     NDBG3(("mptsas_scsi_destroy_pkt: target=%d pkt=0x%p",
4207         ap->a_target, (void *)pkt));
4209     if (cmd->cmd_flags & CFLAG_DMAVALID) {
4210         (void) ddi_dma_unbind_handle(cmd->cmd_dmahandle);
4211         cmd->cmd_flags &= ~CFLAG_DMAVALID;
4212     }
4214     if (cmd->cmd_sg) {
4215         kmem_free(cmd->cmd_sg, sizeof (mptti_t) * cmd->cmd_cookiec);
4216         cmd->cmd_sg = NULL;
4217     }
4219     mptsas_free_extra_sgl_frame(mpt, cmd);

```

```

4221     if ((cmd->cmd_flags &
4222          (CFLAG_FREE | CFLAG_CDBEXTERN | CFLAG_PRIVEXTERN |
4223           CFLAG_SCBEXTERN)) == 0) {
4224         cmd->cmd_flags = CFLAG_FREE;
4225         kmem_cache_free(mpt->m_kmem_cache, (void *)cmd);
4226     } else {
4227         mptsas_pkt_destroy_extern(mpt, cmd);
4228     }
4229 }

4231 /*
4232  * kmem cache constructor and destructor:
4233  * When constructing, we bzero the cmd and allocate the dma handle
4234  * When destructing, just free the dma handle
4235  */
4236 static int
4237 mptsas_kmem_cache_constructor(void *buf, void *cdrarg, int kmflags)
4238 {
4239     mptsas_cmd_t      *cmd = buf;
4240     mptsas_t          *mpt = cdrarg;
4241     int                (*callback)(caddr_t);

4243     callback = (kmflags == KM_SLEEP)? DDI_DMA_SLEEP: DDI_DMA_DONTWAIT;

4245     NDBG4(("mptsas_kmem_cache_constructor"));

4247     /*
4248      * allocate a dma handle
4249      */
4250     if ((ddi_dma_alloc_handle(mpt->m_dip, &mpt->m_io_dma_attr, callback,
4251                              NULL, &cmd->cmd_dmahandle)) != DDI_SUCCESS) {
4252         cmd->cmd_dmahandle = NULL;
4253         return (-1);
4254     }
4255     return (0);
4256 }

4258 static void
4259 mptsas_kmem_cache_destructor(void *buf, void *cdrarg)
4260 {
4261     #ifndef __lock_lint
4262         _NOTE(ARGUNUSED(cdrarg))
4263     #endif
4264     mptsas_cmd_t      *cmd = buf;

4266     NDBG4(("mptsas_kmem_cache_destructor"));

4268     if (cmd->cmd_dmahandle) {
4269         ddi_dma_free_handle(&cmd->cmd_dmahandle);
4270         cmd->cmd_dmahandle = NULL;
4271     }
4272 }

4274 static int
4275 mptsas_cache_frames_constructor(void *buf, void *cdrarg, int kmflags)
4276 {
4277     mptsas_cache_frames_t *p = buf;
4278     mptsas_t              *mpt = cdrarg;
4279     ddi_dma_attr_t         frame_dma_attr;
4280     size_t                 mem_size, alloc_len;
4281     ddi_dma_cookie_t        cookie;
4282     uint_t                 ncookie;
4283     int (*callback)(caddr_t) = (kmflags == KM_SLEEP)
4284         ? DDI_DMA_SLEEP: DDI_DMA_DONTWAIT;

```

```

4286     frame_dma_attr = mpt->m_msg_dma_attr;
4287     frame_dma_attr.dma_attr_align = 0x10;
4288     frame_dma_attr.dma_attr_sgllen = 1;

4290     if (ddi_dma_alloc_handle(mpt->m_dip, &frame_dma_attr, callback, NULL,
4291                             &p->m_dma_hdl) != DDI_SUCCESS) {
4292         mptsas_log(mpt, CE_WARN, "Unable to allocate dma handle for"
4293                  " extra SGL.");
4294         return (DDI_FAILURE);
4295     }

4297     mem_size = (mpt->m_max_request_frames - 1) * mpt->m_req_frame_size;

4299     if (ddi_dma_mem_alloc(p->m_dma_hdl, mem_size, &mpt->m_dev_acc_attr,
4300                          DDI_DMA_CONSISTENT, callback, NULL, (caddr_t *)&p->m_frames_addr,
4301                          &alloc_len, &p->m_acc_hdl) != DDI_SUCCESS) {
4302         ddi_dma_free_handle(&p->m_dma_hdl);
4303         p->m_dma_hdl = NULL;
4304         mptsas_log(mpt, CE_WARN, "Unable to allocate dma memory for"
4305                  " extra SGL.");
4306         return (DDI_FAILURE);
4307     }

4309     if (ddi_dma_addr_bind_handle(p->m_dma_hdl, NULL, p->m_frames_addr,
4310                                 alloc_len, DDI_DMA_RDWR | DDI_DMA_CONSISTENT, callback, NULL,
4311                                 &cookie, &ncookie) != DDI_DMA_MAPPED) {
4312         (void) ddi_dma_mem_free(&p->m_acc_hdl);
4313         ddi_dma_free_handle(&p->m_dma_hdl);
4314         p->m_dma_hdl = NULL;
4315         mptsas_log(mpt, CE_WARN, "Unable to bind DMA resources for"
4316                  " extra SGL.");
4317         return (DDI_FAILURE);
4318     }

4320     /*
4321      * Store the SGL memory address. This chip uses this
4322      * address to dma to and from the driver. The second
4323      * address is the address mpt uses to fill in the SGL.
4324      */
4325     p->m_phys_addr = cookie.dmac_laddress;

4327     return (DDI_SUCCESS);
4328 }

4330 static void
4331 mptsas_cache_frames_destructor(void *buf, void *cdrarg)
4332 {
4333     #ifndef __lock_lint
4334         _NOTE(ARGUNUSED(cdrarg))
4335     #endif
4336     mptsas_cache_frames_t *p = buf;
4337     if (p->m_dma_hdl != NULL) {
4338         (void) ddi_dma_unbind_handle(p->m_dma_hdl);
4339         (void) ddi_dma_mem_free(&p->m_acc_hdl);
4340         ddi_dma_free_handle(&p->m_dma_hdl);
4341         p->m_phys_addr = NULL;
4342         p->m_frames_addr = NULL;
4343         p->m_dma_hdl = NULL;
4344         p->m_acc_hdl = NULL;
4345     }

4347 }

4349 /*
4350  * allocate and deallocate external pkt space (ie. not part of mptsas_cmd)
4351  * for non-standard length cdb, pkt_private, status areas

```

```

4352 * if allocation fails, then deallocate all external space and the pkt
4353 */
4354 /* ARGSUSED */
4355 static int
4356 mptsas_pkt_alloc_extern(mptsas_t *mpt, mptsas_cmd_t *cmd,
4357     int cmdlen, int tgtlen, int statuslen, int kf)
4358 {
4359     caddr_t          cdbp, scbp, tgt;
4360     size_t           senselength;
4361
4362     NDBG3(("mptsas_pkt_alloc_extern: "
4363         "cmd=0x%p cmdlen=%d tgtlen=%d statuslen=%d kf=%x",
4364         (void *)cmd, cmdlen, tgtlen, statuslen, kf));
4365
4366     tgt = cdbp = scbp = NULL;
4367     cmd->cmd_scblen = statuslen;
4368     cmd->cmd_privlen = (uchar_t)tgtlen;
4369
4370     if (cmdlen > sizeof (cmd->cmd_cdb)) {
4371         if ((cdbp = kmem_zalloc((size_t)cmdlen, kf)) == NULL) {
4372             goto fail;
4373         }
4374         cmd->cmd_pkt->pkt_cdbp = (opaque_t)cdbp;
4375         cmd->cmd_flags |= CFLAG_CDBEXTERN;
4376     }
4377     if (tgtlen > PKT_PRIV_LEN) {
4378         if ((tgt = kmem_zalloc((size_t)tgtlen, kf)) == NULL) {
4379             goto fail;
4380         }
4381         cmd->cmd_flags |= CFLAG_PRIVEXTERN;
4382         cmd->cmd_pkt->pkt_private = tgt;
4383     }
4384     if (statuslen > EXTCMD_STATUS_SIZE) {
4385         if ((scbp = kmem_zalloc((size_t)statuslen, kf)) == NULL) {
4386             goto fail;
4387         }
4388         cmd->cmd_flags |= CFLAG_SCBEXTERN;
4389         cmd->cmd_pkt->pkt_scbp = (opaque_t)scbp;
4390
4391         /* allocate sense data buf for DMA */
4392
4393         senselength = statuslen - MPTSAS_GET_ITEM_OFF(
4394             struct scsi_arq_status, sts_sensedata);
4395         if (senselength > mpt->m_req_sense_size) {
4396             unsigned long i;
4397             cmd->cmd_extrqslen = (uint16_t)senselength;
4398             cmd->cmd_extrqschunks = (senselength +
4399                 (mpt->m_req_sense_size - 1)) / mpt->m_req_sense_size;
4400             i = rmalloc_wait(mpt->m_erqsense_map,
4401                 cmd->cmd_extrqschunks);
4402             ASSERT(i != 0);
4403             cmd->cmd_extrqsidx = i - 1;
4404             cmd->cmd_arq_buf = mpt->m_extreq_sense +
4405                 (cmd->cmd_extrqsidx * mpt->m_req_sense_size);
4406         } else {
4407             cmd->cmd_rqslen = (uchar_t)senselength;
4408         }
4409     }
4410     return (0);
4411 fail:
4412     mptsas_pkt_destroy_extern(mpt, cmd);
4413     return (1);
4414 }
4415
4416 /*
4417 * deallocate external pkt space and deallocate the pkt

```

```

4418 */
4419 static void
4420 mptsas_pkt_destroy_extern(mptsas_t *mpt, mptsas_cmd_t *cmd)
4421 {
4422     NDBG3(("mptsas_pkt_destroy_extern: cmd=0x%p", (void *)cmd));
4423
4424     if (cmd->cmd_flags & CFLAG_FREE) {
4425         mptsas_log(mpt, CE_PANIC,
4426             "mptsas_pkt_destroy_extern: freeing free packet");
4427         NOTE(NOT_REACHED)
4428         /* NOTREACHED */
4429     }
4430     if (cmd->cmd_extrqslen != 0) {
4431         rmfree(mpt->m_erqsense_map, cmd->cmd_extrqschunks,
4432             cmd->cmd_extrqsidx + 1);
4433     }
4434     if (cmd->cmd_flags & CFLAG_CDBEXTERN) {
4435         kmem_free(cmd->cmd_pkt->pkt_cdbp, (size_t)cmd->cmd_cdblen);
4436     }
4437     if (cmd->cmd_flags & CFLAG_SCBEXTERN) {
4438         kmem_free(cmd->cmd_pkt->pkt_scbp, (size_t)cmd->cmd_scblen);
4439     }
4440     if (cmd->cmd_flags & CFLAG_PRIVEXTERN) {
4441         kmem_free(cmd->cmd_pkt->pkt_private, (size_t)cmd->cmd_privlen);
4442     }
4443     cmd->cmd_flags = CFLAG_FREE;
4444     kmem_cache_free(mpt->m_kmem_cache, (void *)cmd);
4445 }
4446
4447 /*
4448 * tran_sync_pkt(9E) - explicit DMA synchronization
4449 */
4450 /* ARGSUSED */
4451 static void
4452 mptsas_scsi_sync_pkt(struct scsi_address *ap, struct scsi_pkt *pkt)
4453 {
4454     mptsas_cmd_t      *cmd = PKT2CMD(pkt);
4455
4456     NDBG3(("mptsas_scsi_sync_pkt: target=%d, pkt=0x%p",
4457         ap->a_target, (void *)pkt));
4458
4459     if (cmd->cmd_dmahandle) {
4460         (void) ddi_dma_sync(cmd->cmd_dmahandle, 0, 0,
4461             (cmd->cmd_flags & CFLAG_DMASEND) ?
4462             DDI_DMA_SYNC_FORDEV : DDI_DMA_SYNC_FORCPU);
4463     }
4464 }
4465
4466 /*
4467 * tran_dmafree(9E) - deallocate DMA resources allocated for command
4468 */
4469 /* ARGSUSED */
4470 static void
4471 mptsas_scsi_dmafree(struct scsi_address *ap, struct scsi_pkt *pkt)
4472 {
4473     mptsas_cmd_t      *cmd = PKT2CMD(pkt);
4474     mptsas_t           *mpt = ADDR2MPT(ap);
4475
4476     NDBG3(("mptsas_scsi_dmafree: target=%d pkt=0x%p",
4477         ap->a_target, (void *)pkt));
4478
4479     if (cmd->cmd_flags & CFLAG_DMAVALID) {
4480         (void) ddi_dma_unbind_handle(cmd->cmd_dmahandle);
4481         cmd->cmd_flags &= ~CFLAG_DMAVALID;
4482     }

```

```

4484     mptsas_free_extra_sgl_frame(mpt, cmd);
4485 }

4487 static void
4488 mptsas_pkt_comp(struct scsi_pkt *pkt, mptsas_cmd_t *cmd)
4489 {
4490     if ((cmd->cmd_flags & CFLAG_CMDIOPB) &&
4491         (! (cmd->cmd_flags & CFLAG_DMASEND))) {
4492         (void) ddi_dma_sync(cmd->cmd_dmahandle, 0, 0,
4493             DDI_DMA_SYNC_FORCPU);
4494     }
4495     (*pkt->pkt_comp)(pkt);
4496 }

4498 static void
4499 mptsas_sge_mainframe(mptsas_cmd_t *cmd, pMpi2SCSIIORequest_t frame,
4500     ddi_acc_handle_t acc_hdl, uint_t cookiec,
4501     uint32_t end_flags)
4502 {
4503     pMpi2SGESimple64_t sge;
4504     mptti_t *dmap;
4505     uint32_t flags;

4507     dmap = cmd->cmd_sg;

4509     sge = (pMpi2SGESimple64_t)(&frame->SGL);
4510     while (cookiec--) {
4511         ddi_put32(acc_hdl, &sge->Address.Low,
4512             dmap->addr.address64.Low);
4513         ddi_put32(acc_hdl, &sge->Address.High,
4514             dmap->addr.address64.High);
4515         ddi_put32(acc_hdl, &sge->FlagsLength, dmap->count);
4516         flags = ddi_get32(acc_hdl, &sge->FlagsLength);
4517         flags |= ((uint32_t)
4518             (MPI2_SGE_FLAGS_SIMPLE_ELEMENT |
4519             MPI2_SGE_FLAGS_SYSTEM_ADDRESS |
4520             MPI2_SGE_FLAGS_64_BIT_ADDRESSING) <<
4521             MPI2_SGE_FLAGS_SHIFT);

4523         /*
4524          * If this is the last cookie, we set the flags
4525          * to indicate so
4526          */
4527         if (cookiec == 0) {
4528             flags |= end_flags;
4529         }
4530         if (cmd->cmd_flags & CFLAG_DMASEND) {
4531             flags |= (MPI2_SGE_FLAGS_HOST_TO_IOC <<
4532                 MPI2_SGE_FLAGS_SHIFT);
4533         } else {
4534             flags |= (MPI2_SGE_FLAGS_IOC_TO_HOST <<
4535                 MPI2_SGE_FLAGS_SHIFT);
4536         }
4537         ddi_put32(acc_hdl, &sge->FlagsLength, flags);
4538         dmap++;
4539         sge++;
4540     }
4541 }

4543 static void
4544 mptsas_sge_chain(mptsas_t *mpt, mptsas_cmd_t *cmd,
4545     pMpi2SCSIIORequest_t frame, ddi_acc_handle_t acc_hdl)
4546 {
4547     pMpi2SGESimple64_t sge;
4548     pMpi2SGEChain64_t sgechain;
4549     uint64_t nframe_phys_addr;

```

```

4550     uint_t cookiec;
4551     mptti_t *dmap;
4552     uint32_t flags;
4553     int i, j, k, l, frames, sgemax;
4554     int temp, maxframe_sges;
4555     uint8_t chainflags;
4556     uint16_t chainlength;
4557     mptsas_cache_frames_t *p;

4559     cookiec = cmd->cmd_cookiec;

4561     /*
4562      * Hereby we start to deal with multiple frames.
4563      * The process is as follows:
4564      * 1. Determine how many frames are needed for SGL element
4565      *    storage; Note that all frames are stored in contiguous
4566      *    memory space and in 64-bit DMA mode each element is
4567      *    3 double-words (12 bytes) long.
4568      * 2. Fill up the main frame. We need to do this separately
4569      *    since it contains the SCSI IO request header and needs
4570      *    dedicated processing. Note that the last 4 double-words
4571      *    of the SCSI IO header is for SGL element storage
4572      *    (MPI2_SGE_IO_UNION).
4573      * 3. Fill the chain element in the main frame, so the DMA
4574      *    engine can use the following frames.
4575      * 4. Enter a loop to fill the remaining frames. Note that the
4576      *    last frame contains no chain element. The remaining
4577      *    frames go into the mpt SGL buffer allocated on the fly,
4578      *    not immediately following the main message frame, as in
4579      *    Gen1.
4580      * Some restrictions:
4581      * 1. For 64-bit DMA, the simple element and chain element
4582      *    are both of 3 double-words (12 bytes) in size, even
4583      *    though all frames are stored in the first 4G of mem
4584      *    range and the higher 32-bits of the address are always 0.
4585      * 2. On some controllers (like the 1064/1068), a frame can
4586      *    hold SGL elements with the last 1 or 2 double-words
4587      *    (4 or 8 bytes) un-used. On these controllers, we should
4588      *    recognize that there's not enough room for another SGL
4589      *    element and move the sge pointer to the next frame.
4590      */

4592     /*
4593      * Sgemax is the number of SGE's that will fit
4594      * each extra frame and frames is total
4595      * number of frames we'll need. 1 sge entry per
4596      * frame is reserved for the chain element thus the -1 below.
4597      */
4598     sgemax = ((mpt->m_req_frame_size / sizeof (MPI2_SGE_SIMPLE64)) - 1);
4599     maxframe_sges = MPTSAS_MAX_FRAME_SGES64(mpt);
4600     temp = (cookiec - (maxframe_sges - 1)) / sgemax;

4602     /*
4603      * A little check to see if we need to round up the number
4604      * of frames we need
4605      */
4606     if ((cookiec - (maxframe_sges - 1)) - (temp * sgemax) > 1) {
4607         frames = (temp + 1);
4608     } else {
4609         frames = temp;
4610     }
4611     dmap = cmd->cmd_sg;
4612     sge = (pMpi2SGESimple64_t)(&frame->SGL);

4614     /*
4615      * First fill in the main frame

```



```

4748         >> 2);
4749         ddi_put16(p->m_acc_hdl,
4750                 &sgechain->Length,
4751                 mpt->m_req_frame_size /
4752                 sizeof (MPI2_SGE_SIMPLE64) *
4753                 sizeof (MPI2_SGE_SIMPLE64));
4754     } else {
4755         /*
4756          * This is the last frame. Set
4757          * the NextChainOffset to 0 and
4758          * Length is the total size of
4759          * all remaining simple elements
4760          */
4761         ddi_put8(p->m_acc_hdl,
4762                 &sgechain->NextChainOffset,
4763                 0);
4764         ddi_put16(p->m_acc_hdl,
4765                 &sgechain->Length,
4766                 (cookiec - j) *
4767                 sizeof (MPI2_SGE_SIMPLE64));
4768     }
4769
4770     /* Jump to the next frame */
4771     sge = (pMpi2SGESimple64_t)
4772     ((char *)p->m_frames_addr +
4773      (int)mpt->m_req_frame_size * k);
4774
4775     continue;
4776 }
4777
4778 ddi_put32(p->m_acc_hdl,
4779          &sge->Address.Low,
4780          dmap->addr.address64.Low);
4781 ddi_put32(p->m_acc_hdl,
4782          &sge->Address.High,
4783          dmap->addr.address64.High);
4784 ddi_put32(p->m_acc_hdl,
4785          &sge->FlagsLength, dmap->count);
4786 flags = ddi_get32(p->m_acc_hdl,
4787                 &sge->FlagsLength);
4788 flags |= ((uint32_t){
4789     MPI2_SGE_FLAGS_SIMPLE_ELEMENT |
4790     MPI2_SGE_FLAGS_SYSTEM_ADDRESS |
4791     MPI2_SGE_FLAGS_64_BIT_ADDRESSING) <<
4792     MPI2_SGE_FLAGS_SHIFT);
4793
4794 /*
4795  * If we are at the end of the frame and
4796  * there is another frame to fill in
4797  * we set the last simple element as last
4798  * element
4799  */
4800 if ((l == sgemax) && (k != frames)) {
4801     flags |= ((uint32_t)
4802              (MPI2_SGE_FLAGS_LAST_ELEMENT) <<
4803              MPI2_SGE_FLAGS_SHIFT);
4804 }
4805
4806 /*
4807  * If this is the final cookie we
4808  * indicate it by setting the flags
4809  */
4810 if (j == i) {
4811     flags |= ((uint32_t)
4812              (MPI2_SGE_FLAGS_LAST_ELEMENT |
4813               MPI2_SGE_FLAGS_END_OF_BUFFER)

```

```

4814     MPI2_SGE_FLAGS_END_OF_LIST) <<
4815     MPI2_SGE_FLAGS_SHIFT);
4816     }
4817     if (cmd->cmd_flags & CFLAG_DMASEND) {
4818         flags |=
4819             (MPI2_SGE_FLAGS_HOST_TO_IOC <<
4820              MPI2_SGE_FLAGS_SHIFT);
4821     } else {
4822         flags |=
4823             (MPI2_SGE_FLAGS_IOC_TO_HOST <<
4824              MPI2_SGE_FLAGS_SHIFT);
4825     }
4826     ddi_put32(p->m_acc_hdl,
4827              &sge->FlagsLength, flags);
4828     dmap++;
4829     sge++;
4830 }
4831 }
4832
4833 /*
4834  * Sync DMA with the chain buffers that were just created
4835  */
4836 (void) ddi_dma_sync(p->m_dma_hdl, 0, 0, DDI_DMA_SYNC_FORDEV);
4837 }
4838
4839 static void
4840 mptsas_ieee_sge_mainframe(mptsas_cmd_t *cmd, pMpi2SCSIIORequest_t frame,
4841                          ddi_acc_handle_t acc_hdl, uint_t cookiec,
4842                          uint8_t end_flag)
4843 {
4844     pMpi2IeeeSgeSimple64_t ieeesge;
4845     mptti_t *dmap;
4846     uint8_t flags;
4847
4848     dmap = cmd->cmd_sg;
4849
4850     NDBG1(("mptsas_ieee_sge_mainframe: cookiec=%d, %s", cookiec,
4851           cmd->cmd_flags & CFLAG_DMASEND?"Out":"In"));
4852
4853     ieeesge = (pMpi2IeeeSgeSimple64_t)&frame->SGL;
4854     while (cookiec--) {
4855         ddi_put32(acc_hdl, &ieeesge->Address.Low,
4856                 dmap->addr.address64.Low);
4857         ddi_put32(acc_hdl, &ieeesge->Address.High,
4858                 dmap->addr.address64.High);
4859         ddi_put32(acc_hdl, &ieeesge->Length, dmap->count);
4860         NDBG1(("mptsas_ieee_sge_mainframe: len=%d, high=0x%x",
4861               dmap->count, dmap->addr.address64.High));
4862         flags = (MPI2_IEEE_SGE_FLAGS_SIMPLE_ELEMENT |
4863                 MPI2_IEEE_SGE_FLAGS_SYSTEM_ADDR);
4864
4865         /*
4866          * If this is the last cookie, we set the flags
4867          * to indicate so
4868          */
4869         if (cookiec == 0) {
4870             flags |= end_flag;
4871         }
4872
4873         /*
4874          * XXX: Hmmm, what about the direction based on
4875          * cmd->cmd_flags & CFLAG_DMASEND?
4876          */
4877         ddi_put8(acc_hdl, &ieeesge->Flags, flags);
4878         dmap++;
4879         ieeesge++;

```

```

4880     }
4881 }

4883 static void
4884 mptsas_ieee_sge_chain(mptsas_t *mpt, mptsas_cmd_t *cmd,
4885     pMpi2SCSIIORequest_t frame, ddi_acc_handle_t acc_hdl)
4886 {
4887     pMpi2IeeeSgeSimple64_t   ieeesge;
4888     pMpi2IeeeSgeChain64_t    ieeesgechain;
4889     uint64_t                 nframe_phys_addr;
4890     uint_t                   cookiec;
4891     mptti_t                  *dmap;
4892     uint8_t                  flags;
4893     int                       i, j, k, l, frames, sgemax;
4894     int                       temp, maxframe_sges;
4895     uint8_t                   chainflags;
4896     uint32_t                  chainlength;
4897     mptsas_cache_frames_t    *p;

4899     cookiec = cmd->cmd_cookiec;

4901     NDBG1(("mptsas_ieee_sge_chain: cookiec=%d", cookiec));

4903     /*
4904      * Hereby we start to deal with multiple frames.
4905      * The process is as follows:
4906      * 1. Determine how many frames are needed for SGL element
4907      *    storage; Note that all frames are stored in contiguous
4908      *    memory space and in 64-bit DMA mode each element is
4909      *    4 double-words (16 bytes) long.
4910      * 2. Fill up the main frame. We need to do this separately
4911      *    since it contains the SCSI IO request header and needs
4912      *    dedicated processing. Note that the last 4 double-words
4913      *    of the SCSI IO header is for SGL element storage
4914      *    (MPI2_SGE_IO_UNION).
4915      * 3. Fill the chain element in the main frame, so the DMA
4916      *    engine can use the following frames.
4917      * 4. Enter a loop to fill the remaining frames. Note that the
4918      *    last frame contains no chain element. The remaining
4919      *    frames go into the mpt SGL buffer allocated on the fly,
4920      *    not immediately following the main message frame, as in
4921      *    Genl.
4922      * Some restrictions:
4923      * 1. For 64-bit DMA, the simple element and chain element
4924      *    are both of 4 double-words (16 bytes) in size, even
4925      *    though all frames are stored in the first 4G of mem
4926      *    range and the higher 32-bits of the address are always 0.
4927      * 2. On some controllers (like the 1064/1068), a frame can
4928      *    hold SGL elements with the last 1 or 2 double-words
4929      *    (4 or 8 bytes) un-used. On these controllers, we should
4930      *    recognize that there's not enough room for another SGL
4931      *    element and move the sge pointer to the next frame.
4932      */

4934     /*
4935      * Sgemax is the number of SGE's that will fit
4936      * each extra frame and frames is total
4937      * number of frames we'll need. 1 sge entry per
4938      * frame is reserved for the chain element thus the -1 below.
4939      */
4940     sgemax = ((mpt->m_req_frame_size / sizeof (MPI2_IEEE_SGE_SIMPLE64))
4941         - 1);
4942     maxframe_sges = MPTSAS_MAX_FRAME_SGES64(mpt);
4943     temp = (cookiec - (maxframe_sges - 1)) / sgemax;
4944
4945     /*

```

```

4946      * A little check to see if we need to round up the number
4947      * of frames we need
4948      */
4949     if ((cookiec - (maxframe_sges - 1)) - (temp * sgemax) > 1) {
4950         frames = (temp + 1);
4951     } else {
4952         frames = temp;
4953     }
4954     NDBG1(("mptsas_ieee_sge_chain: temp=%d, frames=%d", temp, frames));
4955     dmap = cmd->cmd_sg;
4956     ieeesge = (pMpi2IeeeSgeSimple64_t)(&frame->SGL);

4958     /*
4959      * First fill in the main frame
4960      */
4961     j = maxframe_sges - 1;
4962     mptsas_ieee_sge_mainframe(cmd, frame, acc_hdl, j, 0);
4963     dmap += j;
4964     ieeesge += j;
4965     j++;

4967     /*
4968      * Fill in the chain element in the main frame.
4969      * About calculation on ChainOffset:
4970      * 1. Struct msg_scsi_io_request has 4 double-words (16 bytes)
4971      *    in the end reserved for SGL element storage
4972      *    (MPI2_SGE_IO_UNION); we should count it in our
4973      *    calculation. See its definition in the header file.
4974      * 2. Constant j is the counter of the current SGL element
4975      *    that will be processed, and (j - 1) is the number of
4976      *    SGL elements that have been processed (stored in the
4977      *    main frame).
4978      * 3. ChainOffset value should be in units of quad-words (16
4979      *    bytes) so the last value should be divided by 16.
4980      */
4981     ddi_put8(acc_hdl, &frame->ChainOffset,
4982         (sizeof (MPI2_SCSI_IO_REQUEST) -
4983         sizeof (MPI2_SGE_IO_UNION) +
4984         (j - 1) * sizeof (MPI2_IEEE_SGE_SIMPLE64)) >> 4);
4985     ieeesgechain = (pMpi2IeeeSgeChain64_t)ieeesge;
4986     chainflags = (MPI2_IEEE_SGE_FLAGS_CHAIN_ELEMENT |
4987         MPI2_IEEE_SGE_FLAGS_SYSTEM_ADDR);
4988     ddi_put8(acc_hdl, &ieeesgechain->Flags, chainflags);

4990     /*
4991      * The size of the next frame is the accurate size of space
4992      * (in bytes) used to store the SGL elements. j is the counter
4993      * of SGL elements. (j - 1) is the number of SGL elements that
4994      * have been processed (stored in frames).
4995      */
4996     if (frames >= 2) {
4997         chainlength = mpt->m_req_frame_size /
4998             sizeof (MPI2_IEEE_SGE_SIMPLE64) *
4999             sizeof (MPI2_IEEE_SGE_SIMPLE64);
5000     } else {
5001         chainlength = ((cookiec - (j - 1)) *
5002             sizeof (MPI2_IEEE_SGE_SIMPLE64));
5003     }

5005     p = cmd->cmd_extra_frames;

5007     ddi_put32(acc_hdl, &ieeesgechain->Length, chainlength);
5008     ddi_put32(acc_hdl, &ieeesgechain->Address.Low,
5009         p->m_phys_addr&0xfffffffffull);
5010     ddi_put32(acc_hdl, &ieeesgechain->Address.High, p->m_phys_addr>>32);

```

```

5012 /*
5013  * If there are more than 2 frames left we have to
5014  * fill in the next chain offset to the location of
5015  * the chain element in the next frame.
5016  * sgemax is the number of simple elements in an extra
5017  * frame. Note that the value NextChainOffset should be
5018  * in double-words (4 bytes).
5019  */
5020 if (frames >= 2) {
5021     ddi_put8(acc_hdl, &ieeesgechain->NextChainOffset,
5022             (sgemax * sizeof (MPI2_IEEE_SGE_SIMPLE64)) >> 4);
5023 } else {
5024     ddi_put8(acc_hdl, &ieeesgechain->NextChainOffset, 0);
5025 }
5026
5027 /*
5028  * Jump to next frame;
5029  * Starting here, chain buffers go into the per command SGL.
5030  * This buffer is allocated when chain buffers are needed.
5031  */
5032 ieeesge = (pMpi2IeeesSgeSimple64_t)p->m_frames_addr;
5033 i = cookiec;
5034
5035 /*
5036  * Start filling in frames with SGE's. If we
5037  * reach the end of frame and still have SGE's
5038  * to fill we need to add a chain element and
5039  * use another frame. j will be our counter
5040  * for what cookie we are at and i will be
5041  * the total cookiec. k is the current frame
5042  */
5043 for (k = 1; k <= frames; k++) {
5044     for (l = 1; l <= (sgemax + 1) && (j <= i); j++, l++) {
5045
5046         /*
5047          * If we have reached the end of frame
5048          * and we have more SGE's to fill in
5049          * we have to fill the final entry
5050          * with a chain element and then
5051          * continue to the next frame
5052          */
5053         if ((l == (sgemax + 1)) && (k != frames)) {
5054             ieeesgechain = (pMpi25IeeesSgeChain64_t)ieeesge;
5055             j--;
5056             chainflags =
5057                 MPI2_IEEE_SGE_FLAGS_CHAIN_ELEMENT |
5058                 MPI2_IEEE_SGE_FLAGS_SYSTEM_ADDR;
5059             ddi_put8(p->m_acc_hdl,
5060                     &ieeesgechain->Flags, chainflags);
5061             /*
5062              * k is the frame counter and (k + 1)
5063              * is the number of the next frame.
5064              * Note that frames are in contiguous
5065              * memory space.
5066              */
5067             nframe_phys_addr = p->m_phys_addr +
5068                 (mpt->m_req_frame_size * k);
5069             ddi_put32(p->m_acc_hdl,
5070                     &ieeesgechain->Address.Low,
5071                     nframe_phys_addr & 0xffffffff);
5072             ddi_put32(p->m_acc_hdl,
5073                     &ieeesgechain->Address.High,
5074                     nframe_phys_addr >> 32);
5075
5076             /*
5077              * If there are more than 2 frames left

```

```

5078         * we have to next chain offset to
5079         * the location of the chain element
5080         * in the next frame and fill in the
5081         * length of the next chain
5082         */
5083         if ((frames - k) >= 2) {
5084             ddi_put8(p->m_acc_hdl,
5085                     &ieeesgechain->NextChainOffset,
5086                     (sgemax *
5087                      sizeof (MPI2_IEEE_SGE_SIMPLE64))
5088                     >> 4);
5089             ddi_put32(p->m_acc_hdl,
5090                     &ieeesgechain->Length,
5091                     mpt->m_req_frame_size /
5092                     sizeof (MPI2_IEEE_SGE_SIMPLE64) *
5093                     sizeof (MPI2_IEEE_SGE_SIMPLE64));
5094         } else {
5095             /*
5096              * This is the last frame. Set
5097              * the NextChainOffset to 0 and
5098              * Length is the total size of
5099              * all remaining simple elements
5100              */
5101             ddi_put8(p->m_acc_hdl,
5102                     &ieeesgechain->NextChainOffset,
5103                     0);
5104             ddi_put32(p->m_acc_hdl,
5105                     &ieeesgechain->Length,
5106                     (cookiec - j) *
5107                     sizeof (MPI2_IEEE_SGE_SIMPLE64));
5108         }
5109
5110         /* Jump to the next frame */
5111         ieeesge = (pMpi2IeeesSgeSimple64_t)
5112             ((char *)p->m_frames_addr +
5113              (int)mpt->m_req_frame_size * k);
5114
5115         continue;
5116     }
5117
5118     ddi_put32(p->m_acc_hdl,
5119             &ieeesge->Address.Low,
5120             dmap->addr.address64.Low);
5121     ddi_put32(p->m_acc_hdl,
5122             &ieeesge->Address.High,
5123             dmap->addr.address64.High);
5124     ddi_put32(p->m_acc_hdl,
5125             &ieeesge->Length, dmap->count);
5126     flags = (MPI2_IEEE_SGE_FLAGS_SIMPLE_ELEMENT |
5127             MPI2_IEEE_SGE_FLAGS_SYSTEM_ADDR);
5128
5129     /*
5130      * If we are at the end of the frame and
5131      * there is another frame to fill in
5132      * do we need to do anything?
5133      * if ((l == sgemax) && (k != frames)) {
5134      * }
5135      */
5136
5137     /*
5138      * If this is the final cookie set end of list.
5139      */
5140     if (j == i) {
5141         flags |= MPI25_IEEE_SGE_FLAGS_END_OF_LIST;
5142     }

```



```

5144         ddi_put8(p->m_acc_hdl, &ieeesge->Flags, flags);
5145         dmap++;
5146         ieeesge++;
5147     }
5148 }

5150 /*
5151  * Sync DMA with the chain buffers that were just created
5152  */
5153 (void) ddi_dma_sync(p->m_dma_hdl, 0, 0, DDI_DMA_SYNC_FORDEV);
5154 }

5156 static void
5157 mptsas_sge_setup(mptsas_t *mpt, mptsas_cmd_t *cmd, uint32_t *control,
5158     pMpi2SCSIIORequest_t frame, ddi_acc_handle_t acc_hdl)
5159 {
5160     ASSERT(cmd->cmd_flags & CFLAG_DMAVALIDID);

5162     NDBG1(("mptsas_sge_setup: cookiec=%d", cmd->cmd_cookiec));

5164     /*
5165      * Set read/write bit in control.
5166      */
5167     if (cmd->cmd_flags & CFLAG_DMASEND) {
5168         *control |= MPI2_SCSIIO_CONTROL_WRITE;
5169     } else {
5170         *control |= MPI2_SCSIIO_CONTROL_READ;
5171     }

5173     ddi_put32(acc_hdl, &frame->DataLength, cmd->cmd_dmacount);

5175     /*
5176      * We have 4 cases here. First where we can fit all the
5177      * SG elements into the main frame, and the case
5178      * where we can't. The SG element is also different when using
5179      * MPI2.5 interface.
5180      * If we have more cookies than we can attach to a frame
5181      * we will need to use a chain element to point
5182      * a location of memory where the rest of the S/G
5183      * elements reside.
5184      */
5185     if (cmd->cmd_cookiec <= MPTSAS_MAX_FRAME_SGES64(mpt)) {
5186         if (mpt->m_MPI25) {
5187             mptsas_ieee_sge_mainframe(cmd, frame, acc_hdl,
5188                 cmd->cmd_cookiec,
5189                 MPI25_IEEE_SGE_FLAGS_END_OF_LIST);
5190         } else {
5191             mptsas_sge_mainframe(cmd, frame, acc_hdl,
5192                 cmd->cmd_cookiec,
5193                 ((uint32_t)(MPI2_SGE_FLAGS_LAST_ELEMENT
5194                     | MPI2_SGE_FLAGS_END_OF_BUFFER
5195                     | MPI2_SGE_FLAGS_END_OF_LIST) <<
5196                     MPI2_SGE_FLAGS_SHIFT));
5197         }
5198     } else {
5199         if (mpt->m_MPI25) {
5200             mptsas_ieee_sge_chain(mpt, cmd, frame, acc_hdl);
5201         } else {
5202             mptsas_sge_chain(mpt, cmd, frame, acc_hdl);
5203         }
5204     }
5205 }

5207 /*
5208  * Interrupt handling
5209  * Utility routine. Poll for status of a command sent to HBA

```

```

5210  * without interrupts (a FLAG_NOINTR command).
5211  */
5212 int
5213 mptsas_poll(mptsas_t *mpt, mptsas_cmd_t *poll_cmd, int polltime)
5214 {
5215     int rval = TRUE;
5216     uint32_t int_mask;

5218     NDBG5(("mptsas_poll: cmd=0x%p, flags 0x%x", (void *)poll_cmd,
5219         poll_cmd->cmd_flags));

5221     /*
5222      * Get the current interrupt mask and disable interrupts. When
5223      * re-enabling ints, set mask to saved value.
5224      */
5225     int_mask = ddi_get32(mpt->m_datap, &mpt->m_reg->HostInterruptMask);
5226     MPTSAS_DISABLE_INTR(mpt);

5228     mpt->m_polled_intr = 1;

5230     if ((poll_cmd->cmd_flags & CFLAG_TM_CMD) == 0) {
5231         mptsas_restart_hba(mpt);
5232     }

5234     /*
5235      * Wait, using drv_usecwait(), long enough for the command to
5236      * reasonably return from the target if the target isn't
5237      * "dead". A polled command may well be sent from scsi_poll, and
5238      * there are retries built in to scsi_poll if the transport
5239      * accepted the packet (TRAN_ACCEPT). scsi_poll waits 1 second
5240      * and retries the transport up to scsi_poll_buscnt times
5241      * (currently 60) if
5242      * 1. pkt_reason is CMD_INCOMPLETE and pkt_state is 0, or
5243      * 2. pkt_reason is CMD_CMPLT and *pkt_scbp has STATUS_BUSY
5244      *
5245      * limit the waiting to avoid a hang in the event that the
5246      * cmd never gets started but we are still receiving interrupts
5247      */
5248     while (!(poll_cmd->cmd_flags & CFLAG_FINISHED)) {
5249         if (mptsas_wait_intr(mpt, polltime) == FALSE) {
5250             NDBG5(("mptsas_poll: command incomplete"));
5251             rval = FALSE;
5252             break;
5253         }
5254     }

5256     if (rval == FALSE) {

5258         /*
5259          * this isn't supposed to happen, the hba must be wedged
5260          * Mark this cmd as a timeout.
5261          */
5262         mptsas_set_pkt_reason(mpt, poll_cmd, CMD_TIMEOUT,
5263             (STAT_TIMEOUT|STAT_ABORTED));

5265         if (poll_cmd->cmd_queued == FALSE) {

5267             NDBG5(("mptsas_poll: not on waitq"));

5269             poll_cmd->cmd_pkt->pkt_state |=
5270                 (STATE_GOT_BUS|STATE_GOT_TARGET|STATE_SENT_CMD);
5271         } else {

5273             /* find and remove it from the waitq */
5274             NDBG5(("mptsas_poll: delete from waitq"));
5275             mptsas_waitq_delete(mpt, poll_cmd);

```

```

5276     }
5277 }
5278 }
5279 mptsas_fma_check(mpt, poll_cmd);
5280
5281 /*
5282  * Clear polling flag, re-enable interrupts.
5283  */
5284 mpt->m_polled_intr = 0;
5285 ddi_put32(mpt->m_datap, &mpt->m_reg->HostInterruptMask, int_mask);
5286
5287 /*
5288  * If there are queued cmd, start them now.
5289  */
5290 if (mpt->m_waitq != NULL) {
5291     mptsas_restart_waitq(mpt);
5292 }
5293
5294 NDBG5(("mptsas_poll: done"));
5295 return (rval);
5296 }
5297
5298 /*
5299  * Used for polling cmds and TM function
5300  */
5301 static int
5302 mptsas_wait_intr(mptsas_t *mpt, int polltime)
5303 {
5304     int cnt, rval = FALSE;
5305     pMpi2ReplyDescriptorsUnion_t reply_desc_union;
5306     mptsas_reply_pqueue_t *rpqp;
5307
5308     NDBG5(("mptsas_wait_intr"));
5309     ASSERT(mutex_owned(&mpt->m_mutex));
5310
5311     /*
5312      * Keep polling for at least (polltime * 1000) seconds
5313      */
5314     rpqp = mpt->m_rep_post_queues;
5315
5316     /*
5317      * Drop the main mutex and grab the mutex for reply queue 0
5318      */
5319     mutex_exit(&mpt->m_mutex);
5320     mutex_enter(&rpqp->rpq_mutex);
5321     for (cnt = 0; cnt < polltime; cnt++) {
5322         (void) ddi_dma_sync(mpt->m_dma_post_queue_hdl, 0, 0,
5323             DDI_DMA_SYNC_FORCPU);
5324
5325         /*
5326          * Polled requests should only come back through
5327          * the first interrupt.
5328          */
5329         reply_desc_union = (pMpi2ReplyDescriptorsUnion_t)
5330             MPTSAS_GET_NEXT_REPLY(rpqp, rpqp->rpq_index);
5331
5332         if (ddi_get32(mpt->m_acc_post_queue_hdl,
5333             &reply_desc_union->Words.Low) == 0xFFFFFFFF ||
5334             ddi_get32(mpt->m_acc_post_queue_hdl,
5335             &reply_desc_union->Words.High) == 0xFFFFFFFF) {
5336             drv_usecwait(1000);
5337             continue;
5338         }
5339
5340         /*
5341          * The reply is valid, process it according to its

```

```

5342     * type.
5343     */
5344     mptsas_process_intr(mpt, rpqp, reply_desc_union);
5345
5346     /*
5347      * Clear the reply descriptor for re-use.
5348      */
5349     ddi_put64(mpt->m_acc_post_queue_hdl,
5350         &((uint64_t *) (void *) rpqp->rpq_queue)[rpqp->rpq_index],
5351         0xFFFFFFFFFFFFFFFF);
5352     (void) ddi_dma_sync(mpt->m_dma_post_queue_hdl, 0, 0,
5353         DDI_DMA_SYNC_FORDEV);
5354
5355     if (++rpqp->rpq_index == mpt->m_post_queue_depth) {
5356         rpqp->rpq_index = 0;
5357     }
5358
5359     /*
5360      * Update the reply index
5361      */
5362     ddi_put32(mpt->m_datap,
5363         &mpt->m_reg->ReplyPostHostIndex, rpqp->rpq_index);
5364     rval = TRUE;
5365     break;
5366 }
5367
5368 mutex_exit(&rpqp->rpq_mutex);
5369 mutex_enter(&mpt->m_mutex);
5370
5371 return (rval);
5372 }
5373
5374 static void
5375 mptsas_handle_scsi_io_success(mptsas_t *mpt,
5376     mptsas_reply_pqueue_t *rpqp,
5377     pMpi2ReplyDescriptorsUnion_t reply_desc)
5378 {
5379     pMpi2SCSIIOSuccessReplyDescriptor_t scsi_io_success;
5380     uint16_t SMID;
5381     mptsas_slots_t *slots = mpt->m_active;
5382     mptsas_cmd_t *cmd = NULL;
5383     struct scsi_pkt *pkt;
5384
5385     scsi_io_success = (pMpi2SCSIIOSuccessReplyDescriptor_t) reply_desc;
5386     SMID = ddi_get16(mpt->m_acc_post_queue_hdl, &scsi_io_success->SMID);
5387
5388     /*
5389      * This is a success reply so just complete the IO. First, do a sanity
5390      * check on the SMID. The final slot is used for TM requests, which
5391      * would not come into this reply handler.
5392      */
5393     if ((SMID == 0) || (SMID > slots->m_n_normal)) {
5394         mptsas_log(mpt, CE_WARN, "?Received invalid SMID of %d\n",
5395             SMID);
5396         ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_UNAFFECTED);
5397         return;
5398     }
5399
5400     cmd = slots->m_slot[SMID];
5401
5402     /*
5403      * print warning and return if the slot is empty
5404      */
5405     if (cmd == NULL) {
5406         mptsas_log(mpt, CE_WARN, "?NULL command for successful SCSI IO "
5407             "in slot %d", SMID);

```

```

5408         return;
5409     }
5410     ASSERT(cmd->cmd_rpqidx == rpqp->rpq_num);

5412     pkt = CMD2PKT(cmd);
5413     pkt->pkt_state |= (STATE_GOT_BUS | STATE_GOT_TARGET | STATE_SENT_CMD |
5414         STATE_GOT_STATUS);
5415     if (cmd->cmd_flags & CFLAG_DMAVALID) {
5416         pkt->pkt_state |= STATE_XFERRED_DATA;
5417     }
5418     pkt->pkt_resid = 0;

5420     if (cmd->cmd_flags & CFLAG_PASSTHRU) {
5421         cmd->cmd_flags |= CFLAG_FINISHED;
5422         cv_broadcast(&mpt->m_passthru_cv);
5423         return;
5424     }
5425     if (!(cmd->cmd_flags & CFLAG_TM_CMD)) {
5426         if (cmd->cmd_flags & CFLAG_CMDIOC) {
5427             mutex_enter(&mpt->m_mutex);
5428             mptsas_remove_cmd(mpt, cmd);
5429             mutex_exit(&mpt->m_mutex);
5430         } else {
5431             #ifdef MPTSAS_DEBUG
5432                 /*
5433                  * In order to test timeout for a command set
5434                  * mptsas_test_timeout via mdb to avoid completion
5435                  * processing here.
5436                  */
5437                 if (mptsas_test_timeout) {
5438                     mptsas_test_timeout = 0;
5439                     return;
5440                 }
5441             #endif
5442             /*
5443              * This is the normal path, avoid grabbing
5444              * the m_mutex.
5445              */
5446             mptsas_remove_cmd_nmtx(mpt, cmd);
5447         }
5448     }

5450     if (cmd->cmd_flags & CFLAG_RETRY) {
5451         /*
5452          * The target returned QFULL or busy, do not add this
5453          * pkt to the doneq since the hba will retry
5454          * this cmd.
5455          *
5456          * The pkt has already been resubmitted in
5457          * mptsas_handle_qfull() or in mptsas_check_scsi_io_error().
5458          * Remove this cmd_flag here.
5459          */
5460         cmd->cmd_flags &= ~CFLAG_RETRY;
5461     } else {
5462         mptsas_rpdoneq_add(mpt, rpqp, cmd);
5463     }
5464 }

5466 static void
5467 mptsas_handle_address_reply(mptsas_t *mpt,
5468     pMpi2ReplyDescriptorsUnion_t reply_desc)
5469 {
5470     pMpi2AddressReplyDescriptor_t address_reply;
5471     pMpi2DefaultReply_t reply;
5472     mptsas_fw_diagnostic_buffer_t *pBuffer;
5473     uint32_t reply_addr, reply_frame_dma_baseaddr;

```

```

5474     uint16_t SMID, iocstatus;
5475     mptsas_slots_t *slots = mpt->m_active;
5476     mptsas_cmd_t *cmd = NULL;
5477     uint8_t function, buffer_type;
5478     m_replyh_arg_t *args;
5479     int reply_frame_no;

5481     ASSERT(mutex_owned(&mpt->m_mutex));

5483     address_reply = (pMpi2AddressReplyDescriptor_t)reply_desc;
5484     reply_addr = ddi_get32(mpt->m_acc_post_queue_hdl,
5485         &address_reply->ReplyFrameAddress);
5486     SMID = ddi_get16(mpt->m_acc_post_queue_hdl, &address_reply->SMID);

5488     /*
5489      * If reply frame is not in the proper range we should ignore this
5490      * message and exit the interrupt handler.
5491      */
5492     reply_frame_dma_baseaddr = mpt->m_reply_frame_dma_addr & 0xfffffffful;
5493     if ((reply_addr < reply_frame_dma_baseaddr) ||
5494         (reply_addr >= (reply_frame_dma_baseaddr +
5495             (mpt->m_reply_frame_size * mpt->m_max_replies))) ||
5496         ((reply_addr - reply_frame_dma_baseaddr) %
5497             mpt->m_reply_frame_size != 0)) {
5498         mptsas_log(mpt, CE_WARN, "?Received invalid reply frame "
5499             "address 0x%x\n", reply_addr);
5500         ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_UNAFFECTED);
5501         return;
5502     }

5504     (void) ddi_dma_sync(mpt->m_dma_reply_frame_hdl, 0, 0,
5505         DDI_DMA_SYNC_FORCPU);
5506     reply = (pMpi2DefaultReply_t)(mpt->m_reply_frame + (reply_addr -
5507         reply_frame_dma_baseaddr));
5508     function = ddi_get8(mpt->m_acc_reply_frame_hdl, &reply->Function);

5510     NDBG31(("mptsas_handle_address_reply: function 0x%x, reply_addr=0x%x",
5511         function, reply_addr));

5513     /*
5514      * don't get slot information and command for events since these values
5515      * don't exist
5516      */
5517     if ((function != MPI2_FUNCTION_EVENT_NOTIFICATION) &&
5518         (function != MPI2_FUNCTION_DIAG_BUFFER_POST)) {
5519         /*
5520          * This could be a TM reply, which use the last allocated SMID,
5521          * so allow for that.
5522          */
5523         if ((SMID == 0) || (SMID > (slots->m_n_normal + 1))) {
5524             mptsas_log(mpt, CE_WARN, "?Received invalid SMID of "
5525                 "%d\n", SMID);
5526             ddi_fm_service_impact(mpt->m_dip,
5527                 DDI_SERVICE_UNAFFECTED);
5528             return;
5529         }

5531         cmd = slots->m_slot[SMID];

5533         /*
5534          * print warning and return if the slot is empty
5535          */
5536         if (cmd == NULL) {
5537             mptsas_log(mpt, CE_WARN, "?NULL command for address "
5538                 "reply in slot %d", SMID);
5539             return;

```

```

5540     }
5541     if ((cmd->cmd_flags &
5542         (CFLAG_PASSTHRU | CFLAG_CONFIG | CFLAG_FW_DIAG))) {
5543         cmd->cmd_rfm = reply_addr;
5544         cmd->cmd_flags |= CFLAG_FINISHED;
5545         cv_broadcast(&mpt->m_passthru_cv);
5546         cv_broadcast(&mpt->m_config_cv);
5547         cv_broadcast(&mpt->m_fw_diag_cv);
5548         return;
5549     } else if (!(cmd->cmd_flags & CFLAG_FW_CMD)) {
5550         mptsas_remove_cmd(mpt, cmd);
5551     }
5552     NDBG31(("\\t\\tmptsas_process_intr: slot=%d", SMID));
5553 }
5554 /*
5555  * Depending on the function, we need to handle
5556  * the reply frame (and cmd) differently.
5557  */
5558 switch (function) {
5559 case MPI2_FUNCTION_SCSI_IO_REQUEST:
5560     mptsas_check_scsi_io_error(mpt, (pMpi2SCSIIOReply_t)reply, cmd);
5561     break;
5562 case MPI2_FUNCTION_SCSI_TASK_MGMT:
5563     cmd->cmd_rfm = reply_addr;
5564     mptsas_check_task_mgt(mpt, (pMpi2SCSIManagementReply_t)reply,
5565         cmd);
5566     break;
5567 case MPI2_FUNCTION_FW_DOWNLOAD:
5568     cmd->cmd_flags |= CFLAG_FINISHED;
5569     cv_signal(&mpt->m_fw_cv);
5570     break;
5571 case MPI2_FUNCTION_EVENT_NOTIFICATION:
5572     reply_frame_no = (reply_addr - reply_frame_dma_baseaddr) /
5573         mpt->m_reply_frame_size;
5574     args = &mpt->m_replyh_args[reply_frame_no];
5575     args->mpt = (void *)mpt;
5576     args->rfm = reply_addr;
5577
5578     /*
5579      * Record the event if its type is enabled in
5580      * this mpt instance by ioctl.
5581      */
5582     mptsas_record_event(args);
5583
5584     /*
5585      * Handle time critical events
5586      * NOT_RESPONDING/ADDED only now
5587      */
5588     if (mptsas_handle_event_sync(args) == DDI_SUCCESS) {
5589         /*
5590          * Would not return main process,
5591          * just let taskq resolve ack action
5592          * and ack would be sent in taskq thread
5593          */
5594         NDBG20(("send mptsas_handle_event_sync success"));
5595     }
5596
5597     if (mpt->m_in_reset) {
5598         NDBG20(("dropping event received during reset"));
5599         return;
5600     }
5601
5602     if ((ddi_taskq_dispatch(mpt->m_event_taskq, mptsas_handle_event,
5603         (void *)args, DDI_NOSLEEP)) != DDI_SUCCESS) {
5604         mptsas_log(mpt, CE_WARN, "No memory available"
5605             "for dispatch taskq");

```

```

5606     /*
5607      * Return the reply frame to the free queue.
5608      */
5609     ddi_put32(mpt->m_acc_free_queue_hdl,
5610         &((uint32_t *) (void *))
5611         mpt->m_free_queue)[mpt->m_free_index], reply_addr);
5612     (void) ddi_dma_sync(mpt->m_dma_free_queue_hdl, 0, 0,
5613         DDI_DMA_SYNC_FORDEV);
5614     if (++mpt->m_free_index == mpt->m_free_queue_depth) {
5615         mpt->m_free_index = 0;
5616     }
5617
5618     ddi_put32(mpt->m_datap,
5619         &mpt->m_reg->ReplyFreeHostIndex, mpt->m_free_index);
5620 }
5621 return;
5622 case MPI2_FUNCTION_DIAG_BUFFER_POST:
5623     /*
5624      * If SMID is 0, this implies that the reply is due to a
5625      * release function with a status that the buffer has been
5626      * released. Set the buffer flags accordingly.
5627      */
5628     if (SMID == 0) {
5629         iocstatus = ddi_get16(mpt->m_acc_reply_frame_hdl,
5630             &reply->IOCStatus);
5631         buffer_type = ddi_get8(mpt->m_acc_reply_frame_hdl,
5632             &(((pMpi2DiagBufferPostReply_t)reply)->BufferType));
5633         if (iocstatus == MPI2_IOCSTATUS_DIAGNOSTIC_RELEASED) {
5634             pBuffer =
5635                 &mpt->m_fw_diag_buffer_list[buffer_type];
5636             pBuffer->valid_data = TRUE;
5637             pBuffer->owned_by_firmware = FALSE;
5638             pBuffer->immediate = FALSE;
5639         }
5640     } else {
5641         /*
5642          * Normal handling of diag post reply with SMID.
5643          */
5644         cmd = slots->m_slot[SMID];
5645
5646         /*
5647          * print warning and return if the slot is empty
5648          */
5649         if (cmd == NULL) {
5650             mptsas_log(mpt, CE_WARN, "?NULL command for "
5651                 "address reply in slot %d", SMID);
5652             return;
5653         }
5654         cmd->cmd_rfm = reply_addr;
5655         cmd->cmd_flags |= CFLAG_FINISHED;
5656         cv_broadcast(&mpt->m_fw_diag_cv);
5657     }
5658     return;
5659 default:
5660     mptsas_log(mpt, CE_WARN, "Unknown function 0x%x ", function);
5661     break;
5662 }
5663
5664 /*
5665  * Return the reply frame to the free queue.
5666  */
5667 ddi_put32(mpt->m_acc_free_queue_hdl,
5668     &((uint32_t *) (void *))mpt->m_free_queue)[mpt->m_free_index],
5669     reply_addr);
5670 (void) ddi_dma_sync(mpt->m_dma_free_queue_hdl, 0, 0,
5671     DDI_DMA_SYNC_FORDEV);

```

```

5672     if (++mpt->m_free_index == mpt->m_free_queue_depth) {
5673         mpt->m_free_index = 0;
5674     }
5675     ddi_put32(mpt->m_datap, &mpt->m_reg->ReplyFreeHostIndex,
5676             mpt->m_free_index);

5678     if (cmd->cmd_flags & CFLAG_FW_CMD)
5679         return;

5681     if (cmd->cmd_flags & CFLAG_RETRY) {
5682         /*
5683          * The target returned QFULL or busy, do not add this
5684          * pkt to the doneq since the hba will retry
5685          * this cmd.
5686          *
5687          * The pkt has already been resubmitted in
5688          * mptsas_handle_qfull() or in mptsas_check_scsi_io_error().
5689          * Remove this cmd_flag here.
5690          */
5691         cmd->cmd_flags &= ~CFLAG_RETRY;
5692     } else {
5693         mptsas_doneq_add(mpt, cmd);
5694     }
5695 }

5697 #ifdef MPTSAS_DEBUG
5698 static uint8_t mptsas_last_sense[256];
5699 #endif

5701 static void
5702 mptsas_check_scsi_io_error(mptsas_t *mpt, pMpi2SCSIIOReply_t reply,
5703         mptsas_cmd_t *cmd)
5704 {
5705     uint8_t          scsi_status, scsi_state;
5706     uint16_t         ioc_status, cmd_rqs_len;
5707     uint32_t         xferred, sensecount, responsedata, loginfo = 0;
5708     struct scsi_pkt   *pkt;
5709     struct scsi_arq_status *arqstat;
5710     mptsas_target_t   *ptgt = cmd->cmd_tgt_addr;
5711     uint8_t           *sensedata = NULL;
5712     uint64_t          sas_wwn;
5713     uint8_t           phy;
5714     char              wwn_str[MPTSAS_WWN_STRLEN];

5716     scsi_status = ddi_get8(mpt->m_acc_reply_frame_hdl, &reply->SCSIStatus);
5717     ioc_status = ddi_get16(mpt->m_acc_reply_frame_hdl, &reply->IOCStatus);
5718     scsi_state = ddi_get8(mpt->m_acc_reply_frame_hdl, &reply->SCSIState);
5719     xferred = ddi_get32(mpt->m_acc_reply_frame_hdl, &reply->TransferCount);
5720     sensecount = ddi_get32(mpt->m_acc_reply_frame_hdl, &reply->SenseCount);
5721     responsedata = ddi_get32(mpt->m_acc_reply_frame_hdl,
5722             &reply->ResponseInfo);

5724     if (ioc_status & MPI2_IOCSTATUS_FLAG_LOG_INFO_AVAILABLE) {
5725         sas_wwn = ptgt->m_addr.mta_wwn;
5726         phy = ptgt->m_phynum;
5727         if (sas_wwn == 0) {
5728             (void) sprintf(wwn_str, "p%x", phy);
5729         } else {
5730             (void) sprintf(wwn_str, "w%016"PRIx64, sas_wwn);
5731         }
5732         loginfo = ddi_get32(mpt->m_acc_reply_frame_hdl,
5733                 &reply->IOCLogInfo);
5734         mptsas_log(mpt, CE_NOTE,
5735                 "?Log info 0x%x received for target %d %s.\n"
5736                 "\t\tscsi_status=0x%x, ioc_status=0x%x, scsi_state=0x%x",
5737                 loginfo, Tgt(cmd), wwn_str, scsi_status, ioc_status,

```

```

5738         scsi_state);
5739     }

5741     NDBG31(("t\tscsi_status=0x%x, ioc_status=0x%x, scsi_state=0x%x",
5742             scsi_status, ioc_status, scsi_state));

5744     pkt = CMD2PKT(cmd);
5745     *(pkt->pkt_scbp) = scsi_status;

5747     if (loginfo == 0x31170000) {
5748         /*
5749          * if loginfo PL_LOGININFO_CODE_IO_DEVICE_MISSING_DELAY_RETRY
5750          * 0x31170000 comes, that means the device missing delay
5751          * is in progressing, the command need retry later.
5752          */
5753         *(pkt->pkt_scbp) = STATUS_BUSY;
5754         return;
5755     }

5757     if ((scsi_state & MPI2_SCSI_STATE_NO_SCSI_STATUS) &&
5758         ((ioc_status & MPI2_IOCSTATUS_MASK) ==
5759         MPI2_IOCSTATUS_SCSI_DEVICE_NOT_THERE)) {
5760         pkt->pkt_reason = CMD_INCOMPLETE;
5761         pkt->pkt_state |= STATE_GOT_BUS;
5762         mutex_enter(&ptgt->m_t_mutex);
5763         if (ptgt->m_reset_delay == 0) {
5764             mptsas_set_throttle(mpt, ptgt,
5765                     DRAIN_THROTTLE);
5766         }
5767         mutex_exit(&ptgt->m_t_mutex);
5768         return;
5769     }

5771     if (scsi_state & MPI2_SCSI_STATE_RESPONSE_INFO_VALID) {
5772         responsedata &= 0x000000FF;
5773         if (responsedata & MPTSAS_SCSI_RESPONSE_CODE_TLR_OFF) {
5774             mptsas_log(mpt, CE_NOTE, "Do not support the TLR\n");
5775             pkt->pkt_reason = CMD_TLR_OFF;
5776             return;
5777         }
5778     }

5781     switch (scsi_status) {
5782     case MPI2_SCSI_STATUS_CHECK_CONDITION:
5783         (void) ddi_dma_sync(mpt->m_dma_req_sense_hdl, 0, 0,
5784                 DDI_DMA_SYNC_FORCPU);
5785         pkt->pkt_resid = (cmd->cmd_dmactount - xferred);
5786         arqstat = (void*)(pkt->pkt_scbp);
5787         arqstat->sts_rqpkt_status = *((struct scsi_status *)
5788                 (pkt->pkt_scbp));
5789         pkt->pkt_state |= (STATE_GOT_BUS | STATE_GOT_TARGET |
5790                 STATE_SENT_CMD | STATE_GOT_STATUS | STATE_ARQ_DONE);
5791         if (cmd->cmd_flags & CFLAG_XARQ) {
5792             pkt->pkt_state |= STATE_XARQ_DONE;
5793         }
5794         if (pkt->pkt_resid != cmd->cmd_dmactount) {
5795             pkt->pkt_state |= STATE_XFERRED_DATA;
5796         }
5797         arqstat->sts_rqpkt_reason = pkt->pkt_reason;
5798         arqstat->sts_rqpkt_state = pkt->pkt_state;
5799         arqstat->sts_rqpkt_state |= STATE_XFERRED_DATA;
5800         arqstat->sts_rqpkt_statistics = pkt->pkt_statistics;
5801         sensedata = (uint8_t *)&arqstat->sts_sensedata;
5802 #ifdef MPTSAS_DEBUG
5803         bcopy((uchar_t *)cmd->cmd_arq_buf, mptsas_last_sense,

```

```

5804         cmd->cmd_rqslen);
5805 #endif
5806         if (cmd->cmd_extrqslens != 0) {
5807             cmd_rqs_len = cmd->cmd_extrqslens;
5808         } else {
5809             cmd_rqs_len = cmd->cmd_rqslen;
5810         }
5811         bcopy((uchar_t *)cmd->cmd_arq_buf, sensedata,
5812             ((cmd->cmd_rqslen >= sensecount) ? sensecount :
5813              cmd_rqs_len));
5814         argstat->sts_rqpkt_resid = (cmd_rqs_len - sensecount);
5815         cmd->cmd_flags |= CFLAG_CMDARQ;
5816         /*
5817          * Set proper status for pkt if autosense was valid
5818          */
5819         if (scsi_state & MPI2_SCSI_STATE_AUTOTSENSE_VALID) {
5820             struct scsi_status zero_status = { 0 };
5821             argstat->sts_rqpkt_status = zero_status;
5822         }
5823
5824         /*
5825          * ASC=0x47 is parity error
5826          * ASC=0x48 is initiator detected error received
5827          */
5828         if ((scsi_sense_key(sensedata) == KEY_ABORTED_COMMAND) &&
5829             ((scsi_sense_asc(sensedata) == 0x47) ||
5830              (scsi_sense_asc(sensedata) == 0x48))) {
5831             mptsas_log(mpt, CE_NOTE, "Aborted command!");
5832         }
5833
5834         /*
5835          * ASC/ASCQ=0x3F/0x0E means report_luns data changed
5836          * ASC/ASCQ=0x25/0x00 means invalid lun
5837          */
5838         if (((scsi_sense_key(sensedata) == KEY_UNIT_ATTENTION) &&
5839             (scsi_sense_asc(sensedata) == 0x3F) &&
5840             (scsi_sense_ascq(sensedata) == 0x0E)) ||
5841             ((scsi_sense_key(sensedata) == KEY_ILLEGAL_REQUEST) &&
5842              (scsi_sense_asc(sensedata) == 0x25) &&
5843              (scsi_sense_ascq(sensedata) == 0x00))) {
5844             mptsas_topo_change_list_t *topo_node = NULL;
5845
5846             topo_node = kmem_zalloc(
5847                 sizeof (mptsas_topo_change_list_t),
5848                 KM_NOSLEEP);
5849             if (topo_node == NULL) {
5850                 mptsas_log(mpt, CE_NOTE, "No memory"
5851                     "resource for handle SAS dynamic"
5852                     "reconfigure.\n");
5853                 break;
5854             }
5855             topo_node->mpt = mpt;
5856             topo_node->event = MPTSAS_DR_EVENT_RECONFIG_TARGET;
5857             topo_node->un.phymask = ptgt->m_addr.mta_phymask;
5858             topo_node->devhdl = ptgt->m_devhdl;
5859             topo_node->object = (void *)ptgt;
5860             topo_node->flags = MPTSAS_TOPO_FLAG_LUN_ASSOCIATED;
5861
5862             if ((ddi_taskq_dispatch(mpt->m_dr_taskq,
5863                 mptsas_handle_dr,
5864                 (void *)topo_node,
5865                 DDI_NOSLEEP)) != DDI_SUCCESS) {
5866                 kmem_free(topo_node,
5867                     sizeof (mptsas_topo_change_list_t));
5868                 mptsas_log(mpt, CE_NOTE, "mptsas start taskq"
5869                     "for handle SAS dynamic reconfigure"

```

```

5870             "failed. \n");
5871         }
5872         break;
5873     case MPI2_SCSI_STATUS_GOOD:
5874         switch (ioc_status & MPI2_IOCSTATUS_MASK) {
5875             case MPI2_IOCSTATUS_SCSI_DEVICE_NOT_THERE:
5876                 pkt->pkt_reason = CMD_DEV_GONE;
5877                 pkt->pkt_state |= STATE_GOT_BUS;
5878                 mutex_enter(&ptgt->m_t_mutex);
5879                 if (ptgt->m_reset_delay == 0) {
5880                     mptsas_set_throttle(mpt, ptgt, DRAIN_THROTTLE);
5881                 }
5882                 mutex_exit(&ptgt->m_t_mutex);
5883                 NDBG31(("lost disk for target%d, command:%x",
5884                     Tgt(cmd), pkt->pkt_cdbp[0]));
5885                 break;
5886             case MPI2_IOCSTATUS_SCSI_DATA_OVERRUN:
5887                 NDBG31(("data overrun: xferred=%d", xferred));
5888                 NDBG31(("dmacount=%d", cmd->cmd_dmacount));
5889                 pkt->pkt_reason = CMD_DATA_OVR;
5890                 pkt->pkt_state |= (STATE_GOT_BUS | STATE_GOT_TARGET
5891                     | STATE_SENT_CMD | STATE_GOT_STATUS
5892                     | STATE_XFERRED_DATA);
5893                 pkt->pkt_resid = 0;
5894                 break;
5895             case MPI2_IOCSTATUS_SCSI_RESIDUAL_MISMATCH:
5896             case MPI2_IOCSTATUS_SCSI_DATA_UNDERRUN:
5897                 NDBG31(("data underrun: xferred=%d", xferred));
5898                 NDBG31(("dmacount=%d", cmd->cmd_dmacount));
5899                 pkt->pkt_state |= (STATE_GOT_BUS | STATE_GOT_TARGET
5900                     | STATE_SENT_CMD | STATE_GOT_STATUS);
5901                 pkt->pkt_resid = (cmd->cmd_dmacount - xferred);
5902                 if (pkt->pkt_resid != cmd->cmd_dmacount) {
5903                     pkt->pkt_state |= STATE_XFERRED_DATA;
5904                 }
5905                 break;
5906             case MPI2_IOCSTATUS_SCSI_TASK_TERMINATED:
5907                 if (cmd->cmd_active_expiration <= gethrtime()) {
5908                     /*
5909                      * When timeout requested, propagate
5910                      * proper reason and statistics to
5911                      * target drivers.
5912                      */
5913                     mptsas_set_pkt_reason(mpt, cmd, CMD_TIMEOUT,
5914                         STAT_BUS_RESET | STAT_TIMEOUT);
5915                 } else {
5916                     mptsas_set_pkt_reason(mpt, cmd, CMD_RESET,
5917                         STAT_BUS_RESET);
5918                 }
5919                 break;
5920             case MPI2_IOCSTATUS_SCSI_IOC_TERMINATED:
5921             case MPI2_IOCSTATUS_SCSI_EXT_TERMINATED:
5922                 mptsas_set_pkt_reason(mpt,
5923                     cmd, CMD_RESET, STAT_DEV_RESET);
5924                 break;
5925             case MPI2_IOCSTATUS_SCSI_IO_DATA_ERROR:
5926             case MPI2_IOCSTATUS_SCSI_PROTOCOL_ERROR:
5927                 pkt->pkt_state |= (STATE_GOT_BUS | STATE_GOT_TARGET);
5928                 mptsas_set_pkt_reason(mpt,
5929                     cmd, CMD_TERMINATED, STAT_TERMINATED);
5930                 break;
5931             case MPI2_IOCSTATUS_INSUFFICIENT_RESOURCES:
5932             case MPI2_IOCSTATUS_BUSY:
5933                 /*
5934                  * set throttles to drain

```

```

5936         /*
5937         for (ptgt = rehash_first(mpt->m_targets); ptgt != NULL;
5938             ptgt = rehash_next(mpt->m_targets, ptgt)) {
5939             mptsas_set_throttle_mtx(mpt, ptgt,
5940                 DRAIN_THROTTLE);
5941         }
5942
5943         /*
5944         * retry command
5945         */
5946         mptsas_retry_pkt(mpt, cmd);
5947         break;
5948     default:
5949         mptsas_log(mpt, CE_WARN,
5950             "unknown ioc_status = %x\n", ioc_status);
5951         mptsas_log(mpt, CE_CONT, "scsi_state = %x, transfer "
5952             "count = %x, scsi_status = %x", scsi_state,
5953             xferred, scsi_status);
5954         break;
5955     }
5956     break;
5957 case MPI2_SCSI_STATUS_TASK_SET_FULL:
5958     mptsas_handle_qfull(mpt, cmd);
5959     break;
5960 case MPI2_SCSI_STATUS_BUSY:
5961     NDBG31(("scsi_status busy received"));
5962     break;
5963 case MPI2_SCSI_STATUS_RESERVATION_CONFLICT:
5964     NDBG31(("scsi_status reservation conflict received"));
5965     break;
5966 default:
5967     mptsas_log(mpt, CE_WARN, "scsi_status=%x, ioc_status=%x\n",
5968         scsi_status, ioc_status);
5969     mptsas_log(mpt, CE_WARN,
5970         "mptsas_process_intr: invalid scsi status\n");
5971     break;
5972 }
5973 }
5974
5975 static void
5976 mptsas_check_task_mgt(mptsas_t *mpt, pMpi2SCSIManagementReply_t reply,
5977     mptsas_cmd_t *cmd)
5978 {
5979     uint8_t      task_type;
5980     uint16_t     ioc_status;
5981     uint32_t     log_info;
5982     uint16_t     dev_handle;
5983     struct scsi_pkt *pkt = CMD2PKT(cmd);
5984
5985     task_type = ddi_get8(mpt->m_acc_reply_frame_hdl, &reply->TaskType);
5986     ioc_status = ddi_get16(mpt->m_acc_reply_frame_hdl, &reply->IOCStatus);
5987     log_info = ddi_get32(mpt->m_acc_reply_frame_hdl, &reply->IOCLogInfo);
5988     dev_handle = ddi_get16(mpt->m_acc_reply_frame_hdl, &reply->DevHandle);
5989
5990     if (ioc_status != MPI2_IOCSTATUS_SUCCESS) {
5991         mptsas_log(mpt, CE_WARN, "mptsas_check_task_mgt: Task 0x%x "
5992             "failed. IOCStatus=0x%x IOCLogInfo=0x%x target=%d\n",
5993             task_type, ioc_status, log_info, dev_handle);
5994         pkt->pkt_reason = CMD_INCOMPLETE;
5995         return;
5996     }
5997
5998     switch (task_type) {
5999     case MPI2_SCSITASKMGMT_TASKTYPE_ABORT_TASK:
6000     case MPI2_SCSITASKMGMT_TASKTYPE_CLEAR_TASK_SET:
6001     case MPI2_SCSITASKMGMT_TASKTYPE_QUERY_TASK:

```

```

6002     case MPI2_SCSITASKMGMT_TASKTYPE_CLR_ACA:
6003     case MPI2_SCSITASKMGMT_TASKTYPE_QRY_TASK_SET:
6004     case MPI2_SCSITASKMGMT_TASKTYPE_QRY_UNIT_ATTENTION:
6005         break;
6006     case MPI2_SCSITASKMGMT_TASKTYPE_ABRT_TASK_SET:
6007     case MPI2_SCSITASKMGMT_TASKTYPE_LOGICAL_UNIT_RESET:
6008     case MPI2_SCSITASKMGMT_TASKTYPE_TARGET_RESET:
6009         /*
6010         * Check for invalid DevHandle of 0 in case application
6011         * sends bad command. DevHandle of 0 could cause problems.
6012         */
6013         if (dev_handle == 0) {
6014             mptsas_log(mpt, CE_WARN, "!Can't flush target with"
6015                 " DevHandle of 0.");
6016         } else {
6017             mptsas_flush_target(mpt, dev_handle, Lun(cmd),
6018                 task_type);
6019         }
6020         break;
6021     default:
6022         mptsas_log(mpt, CE_WARN, "Unknown task management type %d.",
6023             task_type);
6024         mptsas_log(mpt, CE_WARN, "ioc status = %x", ioc_status);
6025         break;
6026     }
6027 }
6028
6029 static void
6030 mptsas_doneq_thread(mptsas_thread_arg_t *arg)
6031 {
6032     mptsas_t      *mpt = arg->mpt;
6033     uint32_t      t = arg->t;
6034     mptsas_cmd_t  *cmd;
6035     struct scsi_pkt *pkt;
6036     mptsas_doneq_thread_list_t *item = &mpt->m_doneq_thread_id[t];
6037
6038     mutex_enter(&item->mutex);
6039     while (item->flag & MPTSAS_DONEQ_THREAD_ACTIVE) {
6040         if (!item->dlist.dl_g) {
6041             cv_wait(&item->cv, &item->mutex);
6042         }
6043         pkt = NULL;
6044         if ((cmd = mptsas_doneq_thread_rm(mpt, t)) != NULL) {
6045             cmd->cmd_flags |= CFLAG_COMPLETED;
6046             pkt = CMD2PKT(cmd);
6047         }
6048         mutex_exit(&item->mutex);
6049         if (pkt) {
6050             mptsas_pkt_comp(pkt, cmd);
6051         }
6052         mutex_enter(&item->mutex);
6053     }
6054     mutex_exit(&item->mutex);
6055     mutex_enter(&mpt->m_qthread_mutex);
6056     mpt->m_doneq_thread_n--;
6057     cv_broadcast(&mpt->m_qthread_cv);
6058     mutex_exit(&mpt->m_qthread_mutex);
6059 }
6060
6061 /*
6062 * mpt interrupt handler.
6063 */
6064 static uint_t
6065 mptsas_intr(caddr_t arg1, caddr_t arg2)
6066 {
6067 }

```

```

6068     mptsas_t          *mpt = (void *)arg1;
6069     mptsas_reply_pqueue_t *rpqp;
6070     int                reply_q = (int)(uintptr_t)arg2;
6071     pMpi2ReplyDescriptorsUnion_t reply_desc_union;
6072     int                found = 0, i, rpqidx;
6073     size_t            dma_sync_len;
6074     off_t             dma_sync_offset;
6075     uint32_t          istat;

6077     NDBG18(("mptsas_intr: arg1 0x%p reply_q 0x%d", (void *)arg1, reply_q));

6079     rpqp = &mpt->m_rep_post_queues[reply_q];

6081     /*
6082      * If interrupts are shared by two channels then check whether this
6083      * interrupt is genuinely for this channel by making sure first the
6084      * chip is in high power state.
6085      */
6086     if ((mpt->m_options & MPTSAS_OPT_PM) &&
6087         (mpt->m_power_level != PM_LEVEL_D0)) {
6088         mpt->m_unclaimed_pm_interrupt_count++;
6089         return (DDI_INTR_UNCLAIMED);
6090     }

6092     istat = MPTSAS_GET_ISTAT(mpt);
6093     if (!(istat & MPI2_HIS_REPLY_DESCRIPTOR_INTERRUPT)) {
6094         NDBG18(("Interrupt bit not set, istat 0x%x", istat));
6095         mpt->m_unclaimed_no_interrupt_count++;
6096         /*
6097          * Really need a good definition of when this is valid.
6098          * It appears not to be if you have multiple reply post
6099          * queues, there may be a better way - need LSI info.
6100          * For now just count them.
6101          */
6102     #if 0
6103         return (DDI_INTR_UNCLAIMED);
6104     #endif
6105     }

6107     /*
6108      * If polling, interrupt was triggered by some shared interrupt because
6109      * IOC interrupts are disabled during polling, so polling routine will
6110      * handle any replies. Considering this, if polling is happening,
6111      * return with interrupt unclaimed.
6112      */
6113     if (mpt->m_polled_intr) {
6114         mptsas_log(mpt, CE_WARN,
6115             "Unclaimed interrupt, rpq %d (Polling), istat 0x%x",
6116             reply_q, istat);
6117         mpt->m_unclaimed_polled_interrupt_count++;
6118         return (DDI_INTR_UNCLAIMED);
6119     }

6121     /*
6122      * At the moment this is the only place the mutex is grabbed.
6123      * So it should never fail!
6124      */
6125     if (mutex_tryenter(&rpqp->rpq_mutex) == 0) {
6126         mutex_enter(&rpqp->rpq_mutex);
6127         rpqp->rpq_intr_mutexbusy++;
6128     }

6130     dma_sync_len = mpt->m_post_queue_depth * 8;
6131     dma_sync_offset = dma_sync_len * reply_q;
6132     (void) ddi_dma_sync(mpt->m_dma_post_queue_hdl,
6133         dma_sync_offset, dma_sync_len, DDI_DMA_SYNC_FORCPU);

```

```

6135     /*
6136      * Go around the reply queue and process each descriptor until
6137      * we get to the next unused one.
6138      * It seems to be an occupational hazard that we get interrupts
6139      * with nothing to do. These are counted below.
6140      */
6141     rpqidx = rpqp->rpq_index;
6142     #ifndef __lock_lint
6143     _NOTE(CONSTCOND)
6144     #endif
6145     while (TRUE) {
6146         reply_desc_union = (pMpi2ReplyDescriptorsUnion_t)
6147             MPTSAS_GET_NEXT_REPLY(rpqp, rpqidx);

6149         if (ddi_get32(mpt->m_acc_post_queue_hdl,
6150             &reply_desc_union->Words.Low) == 0xFFFFFFFF ||
6151             ddi_get32(mpt->m_acc_post_queue_hdl,
6152             &reply_desc_union->Words.High) == 0xFFFFFFFF) {
6153             break;
6154         }

6156         found++;

6158         ASSERT(ddi_get8(mpt->m_acc_post_queue_hdl,
6159             &reply_desc_union->Default.MSIXIndex) == reply_q);

6161         /*
6162          * Process it according to its type.
6163          */
6164         mptsas_process_intr(mpt, rpqp, reply_desc_union);

6166         /*
6167          * Clear the reply descriptor for re-use.
6168          */
6169         ddi_put64(mpt->m_acc_post_queue_hdl,
6170             &((uint64_t *) (void *)rpqp->rpq_queue)[rpqidx],
6171             0xFFFFFFFFFFFFFFFF);

6173         /*
6174          * Increment post index and roll over if needed.
6175          */
6176         if (++rpqidx == mpt->m_post_queue_depth) {
6177             rpqidx = 0;
6178         }
6179     }

6181     if (found == 0) {
6182         rpqp->rpq_intr_unclaimed++;
6183         mutex_exit(&rpqp->rpq_mutex);
6184         mpt->m_unclaimed_nocmd_interrupt_count++;
6185         return (DDI_INTR_UNCLAIMED);
6186     }
6187     rpqp->rpq_index = rpqidx;

6189     rpqp->rpq_intr_count++;
6190     NDBG18(("mptsas_intr complete(%d), did %d loops", reply_q, found));

6192     (void) ddi_dma_sync(mpt->m_dma_post_queue_hdl,
6193         dma_sync_offset, dma_sync_len, DDI_DMA_SYNC_FORDEV);

6195     mpt->m_interrupt_count++;

6197     /*
6198      * Update the reply index if at least one reply was processed.
6199      * For more than 8 reply queues on SAS3 controllers we have to do

```



```

6200      * things a little different. See Chapter 20 in the MPI 2.5 spec.
6201      */
6202      if (mpt->m_post_reply_qcount > 8) {
6203          /*
6204           * The offsets from the base are multiples of 0x10.
6205           * We are indexing into 32 bit quantities so calculate
6206           * the index for that.
6207           */
6208          i = (reply_q&~0x7) >> 1;
6209          ddi_put32(mpt->m_datap,
6210                  &mpt->m_reg->SuppReplyPostHostIndex[i],
6211                  rpqp->rpq_index |
6212                  ((reply_q&0x7)<<MPI2_RPHI_MSIX_INDEX_SHIFT));
6213          (void) ddi_get32(mpt->m_datap,
6214                          &mpt->m_reg->SuppReplyPostHostIndex[i]);
6215      } else {
6216          ddi_put32(mpt->m_datap,
6217                  &mpt->m_reg->ReplyPostHostIndex,
6218                  rpqp->rpq_index | (reply_q<<MPI2_RPHI_MSIX_INDEX_SHIFT));
6219          (void) ddi_get32(mpt->m_datap,
6220                          &mpt->m_reg->ReplyPostHostIndex);
6221      }

6222      /*
6223       * If no helper threads are created, process the doneq in ISR. If
6224       * helpers are created, use the doneq length as a metric to measure the
6225       * load on the interrupt CPU. If it is long enough, which indicates the
6226       * load is heavy, then we deliver the IO completions to the helpers.
6227       * This measurement has some limitations, although it is simple and
6228       * straightforward and works well for most of the cases at present.
6229       * To always use the threads set mptsas_doneq_length_threshold_prop
6230       * to zero in the mpt_sas3.conf file.
6231       *
6232       * Check the current reply queue done queue.
6233       */
6234      if (rpqp->rpq_dlist.dl_len) {
6235          if (!mpt->m_doneq_thread_n ||
6236              (rpqp->rpq_dlist.dl_len <= mpt->m_doneq_length_threshold)) {
6237              mptsas_rpdoneq_empty(rpqp);
6238          } else {
6239              mptsas_deliver_doneq_thread(mpt, &rpqp->rpq_dlist);
6240          }
6241      }
6242  }

6244      mutex_exit(&rpqp->rpq_mutex);

6246      /*
6247       * Check the main done queue. If we find something
6248       * grab the mutex and check again before processing.
6249       */
6250      if (mpt->m_dlist.dl_len) {
6251          mutex_enter(&mpt->m_mutex);
6252          if (mpt->m_dlist.dl_len) {
6253              if (!mpt->m_doneq_thread_n ||
6254                  (mpt->m_dlist.dl_len <=
6255                   mpt->m_doneq_length_threshold)) {
6256                  mptsas_doneq_empty(mpt);
6257              } else {
6258                  mptsas_deliver_doneq_thread(mpt, &mpt->m_dlist);
6259              }
6260          }
6261          mutex_exit(&mpt->m_mutex);
6262      }

6264      /*
6265       * If there are queued cmd, start them now.

```

```

6266      */
6267      if (mpt->m_waitq != NULL) {
6268          mutex_enter(&mpt->m_mutex);
6269          if (mpt->m_waitq != NULL && mpt->m_polled_intr == 0) {
6270              mptsas_restart_waitq(mpt);
6271          }
6272          mutex_exit(&mpt->m_mutex);
6273      }
6274      return (DDI_INTR_CLAIMED);
6275  }

6277      static void
6278      mptsas_process_intr(mptsas_t *mpt, mptsas_reply_pqueue_t *rpqp,
6279                          pMpi2ReplyDescriptorsUnion_t reply_desc_union)
6280  {
6281      uint8_t reply_type;

6283      /*
6284       * Should get here with the reply queue mutex held, but not
6285       * the main mpt mutex. Want to avoid grabbing that during
6286       * normal operations if possible.
6287       */
6288      ASSERT(mutex_owned(&rpqp->rpq_mutex));

6290      /*
6291       * The reply is valid, process it according to its
6292       * type. Also, set a flag for updated the reply index
6293       * after they've all been processed.
6294       */
6295      reply_type = ddi_get8(mpt->m_acc_post_queue_hdl,
6296                           &reply_desc_union->Default.ReplyFlags);
6297      NDBG18(("mptsas_process_intr(rpqp %d) reply_type 0x%x", rpqp->rpq_num,
6298             reply_type));
6299      reply_type &= MPI2_RPY_DESCRIPTOR_FLAGS_TYPE_MASK;
6300      if (reply_type == MPI2_RPY_DESCRIPTOR_FLAGS_SCSI_IO_SUCCESS ||
6301          reply_type == MPI25_RPY_DESCRIPTOR_FLAGS_FAST_PATH_SCSI_IO_SUCCESS) {
6302          mptsas_handle_scsi_io_success(mpt, rpqp, reply_desc_union);
6303      } else if (reply_type == MPI2_RPY_DESCRIPTOR_FLAGS_ADDRESS_REPLY) {
6304          mutex_enter(&mpt->m_mutex);
6305          mptsas_handle_address_reply(mpt, reply_desc_union);
6306          mutex_exit(&mpt->m_mutex);
6307      } else {
6308          mptsas_log(mpt, CE_WARN, "?Bad reply type %x", reply_type);
6309          ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_UNAFFECTED);
6310      }
6311  }

6313      /*
6314       * handle qfull condition
6315       */
6316      static void
6317      mptsas_handle_qfull(mptsas_t *mpt, mptsas_cmd_t *cmd)
6318  {
6319      mptsas_target_t *ptgt = cmd->cmd_tgt_addr;

6321      mutex_enter(&ptgt->m_t_mutex);
6322      if ((++cmd->cmd_qfull_retries > ptgt->m_qfull_retries) ||
6323          (ptgt->m_qfull_retries == 0)) {
6324          /*
6325           * We have exhausted the retries on QFULL, or,
6326           * the target driver has indicated that it
6327           * wants to handle QFULL itself by setting
6328           * qfull-retries capability to 0. In either case
6329           * we want the target driver's QFULL handling
6330           * to kick in. We do this by having pkt_reason
6331           * as CMD_CMPLT and pkt_scbp as STATUS_QFULL.

```

```

6332     /*
6333     mptsas_set_throttle(mpt, ptgt, DRAIN_THROTTLE);
6334     } else {
6335         if (ptgt->m_reset_delay == 0) {
6336             ptgt->m_t_throttle =
6337                 max((ptgt->m_t_ncmds - 2), 0);
6338         }
6339         mutex_exit(&ptgt->m_t_mutex);

6341         cmd->cmd_flags &= ~(CFLAG_TRANFLAG);

6343         mptsas_retry_pkt(mpt, cmd);

6345         mutex_enter(&ptgt->m_t_mutex);
6346         /*
6347         * when target gives queue full status with no commands
6348         * outstanding (m_t_ncmds == 0), throttle is set to 0
6349         * (HOLD_THROTTLE), and the queue full handling start
6350         * (see psarc/1994/313); if there are commands outstanding,
6351         * throttle is set to (m_t_ncmds - 2)
6352         */
6353         if (ptgt->m_t_throttle == HOLD_THROTTLE) {
6354             /*
6355             * By setting throttle to QFULL_THROTTLE, we
6356             * avoid submitting new commands and in
6357             * mptsas_restart_cmd find out slots which need
6358             * their throttles to be cleared.
6359             */
6360             mptsas_set_throttle(mpt, ptgt, QFULL_THROTTLE);
6361             if (mpt->m_restart_cmd_timeid == 0) {
6362                 mpt->m_restart_cmd_timeid =
6363                     timeout(mptsas_restart_cmd, mpt,
6364                         ptgt->m_qfull_retry_interval);
6365             }
6366         }
6367     }
6368     mutex_exit(&ptgt->m_t_mutex);
6369 }

6371 mptsas_phymask_t
6372 mptsas_physport_to_phymask(mptsas_t *mpt, uint8_t physport)
6373 {
6374     mptsas_phymask_t    phy_mask = 0;
6375     uint8_t              i = 0;

6377     NDBG20(("mptsas3%d physport_to_phymask enter", mpt->m_instance));

6379     ASSERT(mutex_owned(&mpt->m_mutex));

6381     /*
6382     * If physport is 0xFF, this is a RAID volume. Use phymask of 0.
6383     */
6384     if (physport == 0xFF) {
6385         return (0);
6386     }

6388     for (i = 0; i < MPTSAS_MAX_PHYS; i++) {
6389         if (mpt->m_phy_info[i].attached_devhdl &&
6390             (mpt->m_phy_info[i].phy_mask != 0) &&
6391             (mpt->m_phy_info[i].port_num == physport)) {
6392             phy_mask = mpt->m_phy_info[i].phy_mask;
6393             break;
6394         }
6395     }
6396     NDBG20(("mptsas3%d physport_to_phymask:physport :%x phymask :%x, ",
6397         mpt->m_instance, physport, phy_mask));

```

```

6398         return (phy_mask);
6399     }

6401     /*
6402     * mpt free device handle after device gone, by use of passthrough
6403     */
6404     static int
6405     mptsas_free_devhdl(mptsas_t *mpt, uint16_t devhdl)
6406     {
6407         Mpi2SasIoUnitControlRequest_t    req;
6408         Mpi2SasIoUnitControlReply_t      rep;
6409         int                                ret;

6411         ASSERT(mutex_owned(&mpt->m_mutex));

6413         /*
6414         * Need to compose a SAS IO Unit Control request message
6415         * and call mptsas_do_passthru() function
6416         */
6417         bzero(&req, sizeof (req));
6418         bzero(&rep, sizeof (rep));

6420         req.Function = MPI2_FUNCTION_SAS_IO_UNIT_CONTROL;
6421         req.Operation = MPI2_SAS_OP_REMOVE_DEVICE;
6422         req.DevHandle = LE_16(devhdl);

6424         ret = mptsas_do_passthru(mpt, (uint8_t *)&req, (uint8_t *)&rep, NULL,
6425             sizeof (req), sizeof (rep), NULL, 0, NULL, 0, 60, FKIOCTL);
6426         if (ret != 0) {
6427             cmn_err(CE_WARN, "mptsas_free_devhdl: passthru SAS IO Unit "
6428                 "Control error %d", ret);
6429             return (DDI_FAILURE);
6430         }

6432         /* do passthrough success, check the ioc status */
6433         if (LE_16(rep.IOCStatus) != MPI2_IOCSTATUS_SUCCESS) {
6434             cmn_err(CE_WARN, "mptsas_free_devhdl: passthru SAS IO Unit "
6435                 "Control IOCStatus %d", LE_16(rep.IOCStatus));
6436             return (DDI_FAILURE);
6437         }

6439         return (DDI_SUCCESS);
6440     }

6442     static void
6443     mptsas_update_phymask(mptsas_t *mpt)
6444     {
6445         mptsas_phymask_t mask = 0, phy_mask;
6446         char                *phy_mask_name;
6447         uint8_t              current_port;
6448         int                   i, j;

6450         NDBG20(("mptsas3%d update phymask ", mpt->m_instance));

6452         ASSERT(mutex_owned(&mpt->m_mutex));

6454         (void) mptsas_get_sas_io_unit_page(mpt);

6456         phy_mask_name = kmem_zalloc(MPTSAS_MAX_PHYS, KM_SLEEP);

6458         for (i = 0; i < mpt->m_num_phys; i++) {
6459             phy_mask = 0x00;

6461             if (mpt->m_phy_info[i].attached_devhdl == 0)
6462                 continue;

```

```

6464         bzero(phy_mask_name, sizeof(phy_mask_name));
6466         current_port = mpt->m_phy_info[i].port_num;
6468         if ((mask & (1 << i)) != 0)
6469             continue;
6471         for (j = 0; j < mpt->m_num_phys; j++) {
6472             if (mpt->m_phy_info[j].attached_devhdl &&
6473                 (mpt->m_phy_info[j].port_num == current_port)) {
6474                 phy_mask |= (1 << j);
6475             }
6476         }
6477         mask = mask | phy_mask;
6479         for (j = 0; j < mpt->m_num_phys; j++) {
6480             if ((phy_mask >> j) & 0x01) {
6481                 mpt->m_phy_info[j].phy_mask = phy_mask;
6482             }
6483         }
6485         (void) sprintf(phy_mask_name, "%x", phy_mask);
6487         mutex_exit(&mpt->m_mutex);
6488         /*
6489          * register a iport, if the port has already been existed
6490          * SCSI will do nothing and just return.
6491          */
6492         (void) scsi_hba_iport_register(mpt->m_dip, phy_mask_name);
6493         mutex_enter(&mpt->m_mutex);
6494     }
6495     kmem_free(phy_mask_name, MPTSAS_MAX_PHYS);
6496     NDBG20(("mptsas3%d update phymask return", mpt->m_instance));
6497 }
6499 /*
6500  * mptsas_handle_dr is a task handler for DR, the DR action includes:
6501  * 1. Directly Attached Device Added/Removed.
6502  * 2. Expander Device Added/Removed.
6503  * 3. Indirectly Attached Device Added/Expander.
6504  * 4. LUNs of a existing device status change.
6505  * 5. RAID volume created/deleted.
6506  * 6. Member of RAID volume is released because of RAID deletion.
6507  * 7. Physical disks are removed because of RAID creation.
6508  */
6509 static void
6510 mptsas_handle_dr(void *args) {
6511     mptsas_topo_change_list_t    *topo_node = NULL;
6512     mptsas_topo_change_list_t    *save_node = NULL;
6513     mptsas_t                      *mpt;
6514     dev_info_t                    *parent = NULL;
6515     mptsas_phymask_t              phymask = 0;
6516     char                          phy_mask_name[MPTSAS_MAX_PHYS];
6517     uint8_t                       flags = 0, physport = 0xff;
6518     uint8_t                       port_update = 0;
6519     uint_t                        event;
6521     topo_node = (mptsas_topo_change_list_t *)args;
6523     mpt = topo_node->mpt;
6524     event = topo_node->event;
6525     flags = topo_node->flags;
6527     NDBG20(("mptsas3%d handle_dr enter", mpt->m_instance));
6529     switch (event) {

```

```

6530         case MPTSAS_DR_EVENT_RECONFIG_TARGET:
6531             if ((flags == MPTSAS_TOPO_FLAG_DIRECT_ATTACHED_DEVICE) ||
6532                 (flags == MPTSAS_TOPO_FLAG_EXPANDER_ATTACHED_DEVICE) ||
6533                 (flags == MPTSAS_TOPO_FLAG_RAID_PHYSDRV_ASSOCIATED)) {
6534                 /*
6535                  * Direct attached or expander attached device added
6536                  * into system or a Phys Disk that is being unhidden.
6537                  */
6538                 port_update = 1;
6539             }
6540             break;
6541         case MPTSAS_DR_EVENT_RECONFIG_SMP:
6542             /*
6543              * New expander added into system, it must be the head
6544              * of topo_change_list_t
6545              */
6546             port_update = 1;
6547             break;
6548         default:
6549             port_update = 0;
6550             break;
6551     }
6552     /*
6553      * All cases port_update == 1 may cause initiator port form change
6554      */
6555     mutex_enter(&mpt->m_mutex);
6556     if (mpt->m_port_chng && port_update) {
6557         /*
6558          * mpt->m_port_chng flag indicates some PHYSs of initiator
6559          * port have changed to online. So when expander added or
6560          * directly attached device online event come, we force to
6561          * update port information by issuing SAS IO Unit Page and
6562          * update PHYMASKs.
6563          */
6564         (void) mptsas_update_phymask(mpt);
6565         mpt->m_port_chng = 0;
6567     }
6568     mutex_exit(&mpt->m_mutex);
6570     while (topo_node) {
6571         phymask = 0;
6572         flags = topo_node->flags;
6573         event = topo_node->event;
6574         if (event == MPTSAS_DR_EVENT_REMOVE_HANDLE) {
6575             goto handle_topo_change;
6576         }
6577         if ((event == MPTSAS_DR_EVENT_RECONFIG_TARGET) &&
6578             (flags == MPTSAS_TOPO_FLAG_RAID_PHYSDRV_ASSOCIATED)) {
6579             /*
6580              * There is no any field in IR_CONFIG_CHANGE
6581              * event indicate physport/phynum, let's get
6582              * parent after SAS Device Page0 request.
6583              */
6584             goto handle_topo_change;
6585         }
6587         if (parent == NULL) {
6588             physport = topo_node->un.physport;
6589             if (event & (MPTSAS_DR_EVENT_OFFLINE_TARGET |
6590                 MPTSAS_DR_EVENT_OFFLINE_SMP)) {
6591                 /*
6592                  * For all offline events, phymask is known
6593                  */
6594                 phymask = topo_node->un.phymask;
6595                 goto find_parent;

```

```

6596     }
6597     if (flags & MPTSAS_TOPO_FLAG_LUN_ASSOCIATED) {
6598         phymask = topo_node->un.phymask;
6599         goto find_parent;
6600     }

6602     mutex_enter(&mpt->m_mutex);
6603     if (flags == MPTSAS_TOPO_FLAG_DIRECT_ATTACHED_DEVICE) {
6604         /*
6605          * If the direct attached device added or a
6606          * phys disk is being unhidden, argument
6607          * physport actually is PHY#, so we have to get
6608          * phymask according PHY#.
6609          */
6610         physport = mpt->m_phy_info[physport].port_num;
6611     }

6613     /*
6614      * Translate physport to phymask so that we can search
6615      * parent dip.
6616      */
6617     phymask = mptsas_physport_to_phymask(mpt, physport);
6618     mutex_exit(&mpt->m_mutex);

6620 find_parent:
6621     bzero(phy_mask_name, MPTSAS_MAX_PHYS);
6622     /*
6623      * For RAID topology change node, write the iport name
6624      * as v0.
6625      */
6626     if (flags & MPTSAS_TOPO_FLAG_RAID_ASSOCIATED) {
6627         (void) sprintf(phy_mask_name, "v0");
6628     } else {
6629         /*
6630          * phymask can be 0 if the drive has been
6631          * pulled by the time an add event is
6632          * processed. If phymask is 0, just skip this
6633          * event and continue.
6634          */
6635         if (phymask == 0) {
6636             save_node = topo_node;
6637             topo_node = topo_node->next;
6638             ASSERT(save_node);
6639             kmem_free(save_node,
6640                 sizeof (mptsas_topo_change_list_t));
6641             parent = NULL;
6642             continue;
6643         }
6644         (void) sprintf(phy_mask_name, "%x", phymask);
6645     }
6646     parent = scsi_hba_iport_find(mpt->m_dip,
6647         phy_mask_name);
6648     if (parent == NULL) {
6649         mptsas_log(mpt, CE_WARN, "Failed to find an "
6650             "iport for \"%s\", should not happen!",
6651             phy_mask_name);
6652         save_node = topo_node;
6653         topo_node = topo_node->next;
6654         ASSERT(save_node);
6655         kmem_free(save_node,
6656             sizeof (mptsas_topo_change_list_t));
6657         continue;
6658     }

6660 }
6661 ASSERT(parent);

```

```

6662 handle_topo_change:

6664         mutex_enter(&mpt->m_mutex);
6665         /*
6666          * If HBA is being reset, don't perform operations depending
6667          * on the IOC. We must free the topo list, however.
6668          */
6669         if (!mpt->m_in_reset)
6670             mptsas_handle_topo_change(topo_node, parent);
6671         else
6672             NDBG20(("skipping topo change received during reset"));
6673         mutex_exit(&mpt->m_mutex);
6674         save_node = topo_node;
6675         topo_node = topo_node->next;
6676         ASSERT(save_node);
6677         kmem_free(save_node, sizeof (mptsas_topo_change_list_t));

6679         if ((flags == MPTSAS_TOPO_FLAG_DIRECT_ATTACHED_DEVICE) ||
6680             (flags == MPTSAS_TOPO_FLAG_RAID_PHYSDRV_ASSOCIATED) ||
6681             (flags == MPTSAS_TOPO_FLAG_RAID_ASSOCIATED)) {
6682             /*
6683              * If direct attached device associated, make sure
6684              * reset the parent before start the next one. But
6685              * all devices associated with expander shares the
6686              * parent. Also, reset parent if this is for RAID.
6687              */
6688             parent = NULL;
6689         }
6690     }
6691 }

6693 static void
6694 mptsas_offline_target(mptsas_t *mpt, mptsas_target_t *ptgt,
6695     uint8_t topo_flags, dev_info_t *parent)
6696 {
6697     uint64_t    sas_wwn = 0;
6698     uint8_t     phy;
6699     char         wwn_str[MPTSAS_WWN_STRLEN];
6700     uint16_t     devhdl;
6701     int          circ = 0, circ1 = 0;
6702     int          rval = 0;

6704     sas_wwn = ptgt->m_addr.mta_wwn;
6705     phy = ptgt->m_phynum;
6706     devhdl = ptgt->m_devhdl;

6708     if (sas_wwn) {
6709         (void) sprintf(wwn_str, "w%016"PRIx64, sas_wwn);
6710     } else {
6711         (void) sprintf(wwn_str, "p%x", phy);
6712     }

6714     /*
6715      * Abort all outstanding command on the device
6716      */
6717     rval = mptsas_do_scsi_reset(mpt, devhdl);
6718     if (rval) {
6719         NDBG20(("mptsas3d: mptsas_offline_target: reset target "
6720             "before offline devhdl:%x, phymask:%x, rval:%x",
6721             mpt->m_instance, ptgt->m_devhdl,
6722             ptgt->m_addr.mta_phymask, rval));
6723     }

6725     mutex_exit(&mpt->m_mutex);
6727     ndi_devi_enter(scsi_vhci_dip, &circ);

```

```

6728     ndi_devi_enter(parent, &circ1);
6729     rval = mptsas_offline_targetdev(parent, wwn_str);
6730     ndi_devi_exit(parent, circ1);
6731     ndi_devi_exit(scsi_vhci_dip, circ);
6732     NDBG20(("mptsas3%d: mptsas_offline_target %s devhdl:%x, "
6733           "phymask:%x, rval:%x", mpt->m_instance, wwn_str,
6734           ptgt->m_devhdl, ptgt->m_addr.mta_phymask, rval));

6736     /*
6737     * Clear parent's props for SMHBA support
6738     */
6739     if (topo_flags == MPTSAS_TOPO_FLAG_DIRECT_ATTACHED_DEVICE) {
6740         if (ddi_prop_update_string(DDI_DEV_T_NONE, parent,
6741             SCSI_ADDR_PROP_ATTACHED_PORT, "") !=
6742             DDI_PROP_SUCCESS) {
6743             (void) ddi_prop_remove(DDI_DEV_T_NONE, parent,
6744                 SCSI_ADDR_PROP_ATTACHED_PORT);
6745             mptsas_log(mpt, CE_WARN, "mptsas attached port "
6746                 "prop update failed");
6747         }
6748         if (ddi_prop_update_int(DDI_DEV_T_NONE, parent,
6749             MPTSAS_NUM_PHYS, 0) != DDI_PROP_SUCCESS) {
6750             (void) ddi_prop_remove(DDI_DEV_T_NONE, parent,
6751                 MPTSAS_NUM_PHYS);
6752             mptsas_log(mpt, CE_WARN, "mptsas num phys "
6753                 "prop update failed");
6754         }
6755         if (ddi_prop_update_int(DDI_DEV_T_NONE, parent,
6756             MPTSAS_VIRTUAL_PORT, 1) != DDI_PROP_SUCCESS) {
6757             (void) ddi_prop_remove(DDI_DEV_T_NONE, parent,
6758                 MPTSAS_VIRTUAL_PORT);
6759             mptsas_log(mpt, CE_WARN, "mptsas virtual port "
6760                 "prop update failed");
6761         }
6762     }

6764     mutex_enter(&mpt->m_mutex);
6765     ptgt->m_led_status = 0;
6766     (void) mptsas_flush_led_status(mpt, ptgt);
6767     if (rval == DDI_SUCCESS) {
6768         mutex_destroy(&ptgt->m_t_mutex);
6769         rehash_remove(mpt->m_targets, ptgt);
6770         ptgt = NULL;
6771     } else {
6772         /*
6773         * clean DR_INTRANSITION flag to allow I/O down to
6774         * PHCI driver since failover finished.
6775         * Invalidate the devhdl
6776         */
6777         ptgt->m_devhdl = MPTSAS_INVALID_DEVHDL;
6778         ptgt->m_tgt_unconfigured = 0;
6779         ptgt->m_dr_flag = MPTSAS_DR_INACTIVE;
6780     }
6781 }

6783 static void
6784 mptsas_handle_topo_change(mptsas_topo_change_list_t *topo_node,
6785     dev_info_t *parent)
6786 {
6787     mptsas_target_t *ptgt = NULL;
6788     mptsas_smp_t *psmp = NULL;
6789     mptsas_t *mpt = (void *)topo_node->mpt;
6790     uint16_t devhdl;
6791     uint16_t attached_devhdl;
6792     int rval = 0;
6793     uint32_t page_address;

```

```

6794     uint8_t flags;
6795     dev_info_t *lundip;
6796     int circ = 0, circ1 = 0;
6797     char attached_wwnstr[MPTSAS_WWN_STRLEN];

6799     NDBG20(("mptsas3%d handle_topo_change enter, devhdl 0x%x, "
6800           "event 0x%x, flags 0x%x", mpt->m_instance, topo_node->devhdl,
6801           topo_node->event, topo_node->flags));

6803     ASSERT(mutex_owned(&mpt->m_mutex));

6805     switch (topo_node->event) {
6806     case MPTSAS_DR_EVENT_RECONFIG_TARGET:
6807     {
6808         char *phy_mask_name;
6809         mptsas_phymask_t phymask = 0;

6811         if (topo_node->flags == MPTSAS_TOPO_FLAG_RAID_ASSOCIATED) {
6812             /*
6813             * Get latest RAID info.
6814             */
6815             (void) mptsas_get_raid_info(mpt);
6816             ptgt = rehash_linear_search(mpt->m_targets,
6817                 mptsas_target_eval_devhdl, &topo_node->devhdl);
6818             if (ptgt == NULL)
6819                 break;
6820         } else {
6821             ptgt = (void *)topo_node->object;
6822         }

6824         if (ptgt == NULL) {
6825             /*
6826             * If a Phys Disk was deleted, RAID info needs to be
6827             * updated to reflect the new topology.
6828             */
6829             (void) mptsas_get_raid_info(mpt);

6831             /*
6832             * Get sas device page 0 by DevHandle to make sure if
6833             * SSP/SATA end device exist.
6834             */
6835             page_address = (MPI2_SAS_DEVICE_PGAD_FORM_HANDLE &
6836                 MPI2_SAS_DEVICE_PGAD_FORM_MASK) |
6837                 topo_node->devhdl;

6839             rval = mptsas_get_target_device_info(mpt, page_address,
6840                 &devhdl, &ptgt);
6841             if (rval == DEV_INFO_WRONG_DEVICE_TYPE) {
6842                 mptsas_log(mpt, CE_NOTE,
6843                     "mptsas_handle_topo_change: target %d is "
6844                     "not a SAS/SATA device. \n",
6845                     topo_node->devhdl);
6846             } else if (rval == DEV_INFO_FAIL_ALLOC) {
6847                 mptsas_log(mpt, CE_NOTE,
6848                     "mptsas_handle_topo_change: could not "
6849                     "allocate memory. \n");
6850             }
6851             /*
6852             * If rval is DEV_INFO_PHYS_DISK than there is nothing
6853             * else to do, just leave.
6854             */
6855             if (rval != DEV_INFO_SUCCESS) {
6856                 return;
6857             }
6858         }

```

```

6860     ASSERT(ptgt->m_devhdl == topo_node->devhdl);

6862     mutex_exit(&mpt->m_mutex);
6863     flags = topo_node->flags;

6865     if (flags == MPTSAS_TOPO_FLAG_RAID_PHYSDRV_ASSOCIATED) {
6866         phymask = ptgt->m_addr.mta_phymask;
6867         phy_mask_name = kmem_zalloc(MPTSAS_MAX_PHYS, KM_SLEEP);
6868         (void) sprintf(phy_mask_name, "%x", phymask);
6869         parent = scsi_hba_iport_find(mpt->m_dip,
6870             phy_mask_name);
6871         kmem_free(phy_mask_name, MPTSAS_MAX_PHYS);
6872         if (parent == NULL) {
6873             mptsas_log(mpt, CE_WARN, "Failed to find a "
6874                 "iport for PD, should not happen!");
6875             mutex_enter(&mpt->m_mutex);
6876             break;
6877         }
6878     }

6880     if (flags == MPTSAS_TOPO_FLAG_RAID_ASSOCIATED) {
6881         ndi_devi_enter(parent, &circl);
6882         (void) mptsas_config_raid(parent, topo_node->devhdl,
6883             &lundip);
6884         ndi_devi_exit(parent, circl);
6885     } else {
6886         /*
6887          * hold nexus for bus configure
6888          */
6889         ndi_devi_enter(scsi_vhci_dip, &circ);
6890         ndi_devi_enter(parent, &circl);
6891         rval = mptsas_config_target(parent, ptgt);
6892         /*
6893          * release nexus for bus configure
6894          */
6895         ndi_devi_exit(parent, circl);
6896         ndi_devi_exit(scsi_vhci_dip, circ);

6898         /*
6899          * Add parent's props for SMHBA support
6900          */
6901         if (flags == MPTSAS_TOPO_FLAG_DIRECT_ATTACHED_DEVICE) {
6902             bzero(attached_wwnstr,
6903                 sizeof(attached_wwnstr));
6904             (void) sprintf(attached_wwnstr, "%016"PRIx64,
6905                 ptgt->m_addr.mta_wwn);
6906             if (ddi_prop_update_string(DDI_DEV_T_NONE,
6907                 parent,
6908                 SCSI_ADDR_PROP_ATTACHED_PORT,
6909                 attached_wwnstr)
6910                 != DDI_PROP_SUCCESS) {
6911                 (void) ddi_prop_remove(DDI_DEV_T_NONE,
6912                     parent,
6913                     SCSI_ADDR_PROP_ATTACHED_PORT);
6914                 mptsas_log(mpt, CE_WARN, "Failed to "
6915                     "attached-port props");
6916                 return;
6917             }
6918             if (ddi_prop_update_int(DDI_DEV_T_NONE, parent,
6919                 MPTSAS_NUM_PHYS, 1) !=
6920                 DDI_PROP_SUCCESS) {
6921                 (void) ddi_prop_remove(DDI_DEV_T_NONE,
6922                     parent, MPTSAS_NUM_PHYS);
6923                 mptsas_log(mpt, CE_WARN, "Failed to "
6924                     "create num-phys props");
6925                 return;

```

```

6926     }

6928     /*
6929      * Update PHY info for smhba
6930      */
6931     mutex_enter(&mpt->m_mutex);
6932     if (mptsas_smhba_phy_init(mpt)) {
6933         mutex_exit(&mpt->m_mutex);
6934         mptsas_log(mpt, CE_WARN, "mptsas phy"
6935             " update failed");
6936         return;
6937     }
6938     mutex_exit(&mpt->m_mutex);

6940     /*
6941      * topo_node->un.physport is really the PHY#
6942      * for direct attached devices
6943      */
6944     mptsas_smhba_set_one_phy_props(mpt, parent,
6945         topo_node->un.physport, &attached_devhdl);

6947     if (ddi_prop_update_int(DDI_DEV_T_NONE, parent,
6948         MPTSAS_VIRTUAL_PORT, 0) !=
6949         DDI_PROP_SUCCESS) {
6950         (void) ddi_prop_remove(DDI_DEV_T_NONE,
6951             parent, MPTSAS_VIRTUAL_PORT);
6952         mptsas_log(mpt, CE_WARN,
6953             "mptsas virtual-port"
6954             "port prop update failed");
6955         return;
6956     }
6957 }
6958 }
6959 mutex_enter(&mpt->m_mutex);

6961     NDBG20(("mptsas3%d handle_topo_change to online devhdl:%x, "
6962         "phymask:%x.", mpt->m_instance, ptgt->m_devhdl,
6963         ptgt->m_addr.mta_phymask));
6964     break;
6965 }
6966 case MPTSAS_DR_EVENT_OFFLINE_TARGET:
6967 {
6968     devhdl = topo_node->devhdl;
6969     ptgt = rehash_linear_search(mpt->m_targets,
6970         mptsas_target_eval_devhdl, &devhdl);
6971     if (ptgt == NULL)
6972         break;

6974     ASSERT(ptgt->m_devhdl == devhdl);

6976     if ((topo_node->flags == MPTSAS_TOPO_FLAG_RAID_ASSOCIATED) ||
6977         (topo_node->flags ==
6978             MPTSAS_TOPO_FLAG_RAID_PHYSDRV_ASSOCIATED)) {
6979         /*
6980          * Get latest RAID info if RAID volume status changes
6981          * or Phys Disk status changes
6982          */
6983         (void) mptsas_get_raid_info(mpt);
6984     }

6986     mptsas_offline_target(mpt, ptgt, topo_node->flags, parent);

6988     /*
6989      * Send SAS IO Unit Control to free the dev handle
6990      */
6991     if ((flags == MPTSAS_TOPO_FLAG_DIRECT_ATTACHED_DEVICE) ||

```

```

6992         (flags == MPTSAS_TOPO_FLAG_EXPANDER_ATTACHED_DEVICE)) {
6993             rval = mptsas_free_devhdl(mpt, devhdl);

6995             NDBG20(("mptsas3%d handle_topo_change to remove "
6996                  "devhdl:%x, rval:%x", mpt->m_instance, devhdl,
6997                  rval));
6998         }

7000         break;
7001     }
7002     case MPTSAS_DR_EVENT_REMOVE_HANDLE:
7003     {
7004         devhdl = topo_node->devhdl;

7006         /*
7007          * Do a reset first.
7008          */
7009         rval = mptsas_do_scsi_reset(mpt, devhdl);
7010         NDBG20(("mpt%d reset target before remove "
7011              "devhdl:%x, rval:%x", mpt->m_instance, devhdl, rval));

7013         /*
7014          * Send SAS IO Unit Control to free the dev handle
7015          */
7016         rval = mptsas_free_devhdl(mpt, devhdl);
7017         NDBG20(("mptsas3%d handle_topo_change to remove "
7018              "devhdl:%x, rval:%x", mpt->m_instance, devhdl,
7019              rval));
7020         break;
7021     }
7022     case MPTSAS_DR_EVENT_RECONFIG_SMP:
7023     {
7024         mptsas_smp_t smp;
7025         dev_info_t *smpdip;

7027         devhdl = topo_node->devhdl;

7029         page_address = (MPI2_SAS_EXPAND_PGAD_FORM_HNDL &
7030                        MPI2_SAS_EXPAND_PGAD_FORM_MASK) | (uint32_t)devhdl;
7031         rval = mptsas_get_sas_expander_page0(mpt, page_address, &smp);
7032         if (rval != DDI_SUCCESS) {
7033             mptsas_log(mpt, CE_WARN, "failed to online smp, "
7034                  "handle %x", devhdl);
7035             return;
7036         }

7038         psm = mptsas_smp_alloc(mpt, &smp);
7039         if (psmp == NULL) {
7040             return;
7041         }

7043         mutex_exit(&mpt->m_mutex);
7044         ndi_devi_enter(parent, &circ1);
7045         (void) mptsas_online_smp(parent, psm, &smpdip);
7046         ndi_devi_exit(parent, circ1);

7048         mutex_enter(&mpt->m_mutex);
7049         break;
7050     }
7051     case MPTSAS_DR_EVENT_OFFLINE_SMP:
7052     {
7053         devhdl = topo_node->devhdl;
7054         uint32_t dev_info;

7056         psm = rehash_linear_search(mpt->m_smp_targets,
7057                                   mptsas_smp_eval_devhdl, &devhdl);

```

```

7058         if (psmp == NULL)
7059             break;
7060         /*
7061          * The mptsas_smp_t data is released only if the dip is offlined
7062          * successfully.
7063          */
7064         mutex_exit(&mpt->m_mutex);

7066         ndi_devi_enter(parent, &circ1);
7067         rval = mptsas_offline_smp(parent, psmp, NDI_DEVI_REMOVE);
7068         ndi_devi_exit(parent, circ1);

7070         dev_info = psmp->m_deviceinfo;
7071         if ((dev_info & DEVINFO_DIRECT_ATTACHED) ==
7072             DEVINFO_DIRECT_ATTACHED) {
7073             if (ddi_prop_update_int(DDI_DEV_T_NONE, parent,
7074                                     MPTSAS_VIRTUAL_PORT, 1) !=
7075                 DDI_PROP_SUCCESS) {
7076                 (void) ddi_prop_remove(DDI_DEV_T_NONE, parent,
7077                                         MPTSAS_VIRTUAL_PORT);
7078                 mptsas_log(mpt, CE_WARN, "mptsas virtual port "
7079                     "prop update failed");
7080                 return;
7081             }
7082             /*
7083              * Check whether the smp connected to the iport,
7084              */
7085             if (ddi_prop_update_int(DDI_DEV_T_NONE, parent,
7086                                     MPTSAS_NUM_PHYS, 0) !=
7087                 DDI_PROP_SUCCESS) {
7088                 (void) ddi_prop_remove(DDI_DEV_T_NONE, parent,
7089                                         MPTSAS_NUM_PHYS);
7090                 mptsas_log(mpt, CE_WARN, "mptsas num phys "
7091                     "prop update failed");
7092                 return;
7093             }
7094             /*
7095              * Clear parent's attached-port props
7096              */
7097             bzero(attached_wnvstr, sizeof (attached_wnvstr));
7098             if (ddi_prop_update_string(DDI_DEV_T_NONE, parent,
7099                                       SCSI_ADDR_PROP_ATTACHED_PORT, attached_wnvstr) !=
7100                 DDI_PROP_SUCCESS) {
7101                 (void) ddi_prop_remove(DDI_DEV_T_NONE, parent,
7102                                         SCSI_ADDR_PROP_ATTACHED_PORT);
7103                 mptsas_log(mpt, CE_WARN, "mptsas attached port "
7104                     "prop update failed");
7105                 return;
7106             }
7107         }

7109         mutex_enter(&mpt->m_mutex);
7110         NDBG20(("mptsas3%d handle_topo_change to remove devhdl:%x, "
7111              "rval:%x", mpt->m_instance, psmp->m_devhdl, rval));
7112         if (rval == DDI_SUCCESS) {
7113             rehash_remove(mpt->m_smp_targets, psmp);
7114         } else {
7115             psmp->m_devhdl = MPTSAS_INVALID_DEVHDL;
7116         }

7118         bzero(attached_wnvstr, sizeof (attached_wnvstr));

7120         break;
7121     }
7122     default:
7123         return;

```

```

7124     }
7125 }

7127 /*
7128  * Record the event if its type is enabled in mpt instance by ioctl.
7129  */
7130 static void
7131 mptsas_record_event(void *args)
7132 {
7133     m_replyh_arg_t      *replyh_arg;
7134     pMpi2EventNotificationReply_t eventreply;
7135     uint32_t             event, rfm;
7136     mptsas_t             *mpt;
7137     int                  i, j;
7138     uint16_t             event_data_len;
7139     boolean_t            sendAEN = FALSE;

7141     replyh_arg = (m_replyh_arg_t *)args;
7142     rfm = replyh_arg->rfm;
7143     mpt = replyh_arg->mpt;

7145     eventreply = (pMpi2EventNotificationReply_t)
7146         (mpt->m_reply_frame + (rfm -
7147             (mpt->m_reply_frame_dma_addr&0xfffffffful)));
7148     event = ddi_get16(mpt->m_acc_reply_frame_hdl, &eventreply->Event);

7151     /*
7152     * Generate a system event to let anyone who cares know that a
7153     * LOG_ENTRY_ADDED event has occurred. This is sent no matter what the
7154     * event mask is set to.
7155     */
7156     if (event == MPI2_EVENT_LOG_ENTRY_ADDED) {
7157         sendAEN = TRUE;
7158     }

7160     /*
7161     * Record the event only if it is not masked. Determine which dword
7162     * and bit of event mask to test.
7163     */
7164     i = (uint8_t)(event / 32);
7165     j = (uint8_t)(event % 32);
7166     if ((i < 4) && ((1 < j) & mpt->m_event_mask[i])) {
7167         i = mpt->m_event_index;
7168         mpt->m_events[i].Type = event;
7169         mpt->m_events[i].Number = ++mpt->m_event_number;
7170         bzero(mpt->m_events[i].Data, MPTSAS_MAX_EVENT_DATA_LENGTH * 4);
7171         event_data_len = ddi_get16(mpt->m_acc_reply_frame_hdl,
7172             &eventreply->EventDataLength);

7174         if (event_data_len > 0) {
7175             /*
7176             * Limit data to size in m_event entry
7177             */
7178             if (event_data_len > MPTSAS_MAX_EVENT_DATA_LENGTH) {
7179                 event_data_len = MPTSAS_MAX_EVENT_DATA_LENGTH;
7180             }
7181             for (j = 0; j < event_data_len; j++) {
7182                 mpt->m_events[i].Data[j] =
7183                     ddi_get32(mpt->m_acc_reply_frame_hdl,
7184                         &(eventreply->EventData[j]));
7185             }

7187             /*
7188             * check for index wrap-around
7189             */

```

```

7190         if (++i == MPTSAS_EVENT_QUEUE_SIZE) {
7191             i = 0;
7192         }
7193         mpt->m_event_index = (uint8_t)i;

7195         /*
7196         * Set flag to send the event.
7197         */
7198         sendAEN = TRUE;
7199     }
7200 }

7202     /*
7203     * Generate a system event if flag is set to let anyone who cares know
7204     * that an event has occurred.
7205     */
7206     if (sendAEN) {
7207         (void) ddi_log_sysevent(mpt->m_dip, DDI_VENDOR_LSI, "MPT_SAS",
7208             "SAS", NULL, NULL, DDI_NOSLEEP);
7209     }
7210 }

7212 #define SMP_RESET_IN_PROGRESS MPI2_EVENT_SAS_TOPO_LR_SMP_RESET_IN_PROGRESS
7213 /*
7214  * handle sync events from ioc in interrupt
7215  * return value:
7216  * DDI_SUCCESS: The event is handled by this func
7217  * DDI_FAILURE: Event is not handled
7218  */
7219 static int
7220 mptsas_handle_event_sync(void *args)
7221 {
7222     m_replyh_arg_t      *replyh_arg;
7223     pMpi2EventNotificationReply_t eventreply;
7224     uint32_t             event, rfm;
7225     mptsas_t             *mpt;
7226     uint_t               iocstatus;

7228     replyh_arg = (m_replyh_arg_t *)args;
7229     rfm = replyh_arg->rfm;
7230     mpt = replyh_arg->mpt;

7232     ASSERT(mutex_owned(&mpt->m_mutex));

7234     eventreply = (pMpi2EventNotificationReply_t)
7235         (mpt->m_reply_frame + (rfm -
7236             (mpt->m_reply_frame_dma_addr&0xfffffffful)));
7237     event = ddi_get16(mpt->m_acc_reply_frame_hdl, &eventreply->Event);

7239     if (iocstatus = ddi_get16(mpt->m_acc_reply_frame_hdl,
7240         &eventreply->IOCStatus)) {
7241         if (iocstatus == MPI2_IOCSTATUS_FLAG_LOG_INFO_AVAILABLE) {
7242             mptsas_log(mpt, CE_WARN,
7243                 "!mptsas_handle_event_sync: event 0x%x, "
7244                 "IOCStatus=0x%x, "
7245                 "IOCLogInfo=0x%x", event, iocstatus,
7246                 ddi_get32(mpt->m_acc_reply_frame_hdl,
7247                     &eventreply->IOCLogInfo));
7248         } else {
7249             mptsas_log(mpt, CE_WARN,
7250                 "mptsas_handle_event_sync: event 0x%x, "
7251                 "IOCStatus=0x%x, "
7252                 "(IOCLogInfo=0x%x)", event, iocstatus,
7253                 ddi_get32(mpt->m_acc_reply_frame_hdl,
7254                     &eventreply->IOCLogInfo));
7255         }

```



```

7256     }
7257
7258     /*
7259     * figure out what kind of event we got and handle accordingly
7260     */
7261     switch (event) {
7262     case MPI2_EVENT_SAS_TOPOLOGY_CHANGE_LIST:
7263     {
7264         pMpi2EventDataSasTopologyChangeList_t sas_topo_change_list;
7265         uint8_t num_entries, expstatus, phy;
7266         uint8_t phystatus, physport, state, i;
7267         uint8_t start_phy_num, link_rate;
7268         uint16_t dev_handle, reason_code;
7269         uint16_t enc_handle, expd_handle;
7270         char string[80], curr[80], prev[80];
7271         mptsas_topo_change_list_t *topo_head = NULL;
7272         mptsas_topo_change_list_t *topo_tail = NULL;
7273         mptsas_topo_change_list_t *topo_node = NULL;
7274         mptsas_target_t *ptgt;
7275         mptsas_smp_t *psmp;
7276         uint8_t flags = 0, exp_flag;
7277         smhba_info_t *psmhba = NULL;
7278
7279         NDBG20(("mptsas_handle_event_sync: SAS topology change"));
7280
7281         sas_topo_change_list = (pMpi2EventDataSasTopologyChangeList_t)
7282             eventreply->EventData;
7283
7284         enc_handle = ddi_get16(mpt->m_acc_reply_frame_hdl,
7285             &sas_topo_change_list->EnclosureHandle);
7286         expd_handle = ddi_get16(mpt->m_acc_reply_frame_hdl,
7287             &sas_topo_change_list->ExpanderDevHandle);
7288         num_entries = ddi_get8(mpt->m_acc_reply_frame_hdl,
7289             &sas_topo_change_list->NumEntries);
7290         start_phy_num = ddi_get8(mpt->m_acc_reply_frame_hdl,
7291             &sas_topo_change_list->StartPhyNum);
7292         expstatus = ddi_get8(mpt->m_acc_reply_frame_hdl,
7293             &sas_topo_change_list->ExpStatus);
7294         physport = ddi_get8(mpt->m_acc_reply_frame_hdl,
7295             &sas_topo_change_list->PhysicalPort);
7296
7297         string[0] = 0;
7298         if (expd_handle) {
7299             flags = MPTSAS_TOPO_FLAG_EXPANDER_ASSOCIATED;
7300             switch (expstatus) {
7301             case MPI2_EVENT_SAS_TOPO_ES_ADDED:
7302                 (void) sprintf(string, " added");
7303                 /*
7304                  * New expander device added
7305                  */
7306                 mpt->m_port_chng = 1;
7307                 topo_node = kmem_zalloc(
7308                     sizeof (mptsas_topo_change_list_t),
7309                     KM_SLEEP);
7310                 topo_node->mpt = mpt;
7311                 topo_node->event = MPTSAS_DR_EVENT_RECONFIG_SMP;
7312                 topo_node->un.physport = physport;
7313                 topo_node->devhdl = expd_handle;
7314                 topo_node->flags = flags;
7315                 topo_node->object = NULL;
7316                 if (topo_head == NULL) {
7317                     topo_head = topo_tail = topo_node;
7318                 } else {
7319                     topo_tail->next = topo_node;
7320                     topo_tail = topo_node;
7321                 }

```

```

7322         break;
7323         case MPI2_EVENT_SAS_TOPO_ES_NOT_RESPONDING:
7324             (void) sprintf(string, " not responding, "
7325                 "removed");
7326             psmp = rehash_linear_search(mpt->m_smp_targets,
7327                 mptsas_smp_eval_devhdl, &expd_handle);
7328             if (psmp == NULL)
7329                 break;
7330
7331             topo_node = kmem_zalloc(
7332                 sizeof (mptsas_topo_change_list_t),
7333                 KM_SLEEP);
7334             topo_node->mpt = mpt;
7335             topo_node->un.phymask =
7336                 psmp->m_addr.mta.phymask;
7337             topo_node->event = MPTSAS_DR_EVENT_OFFLINE_SMP;
7338             topo_node->devhdl = expd_handle;
7339             topo_node->flags = flags;
7340             topo_node->object = NULL;
7341             if (topo_head == NULL) {
7342                 topo_head = topo_tail = topo_node;
7343             } else {
7344                 topo_tail->next = topo_node;
7345                 topo_tail = topo_node;
7346             }
7347             break;
7348         case MPI2_EVENT_SAS_TOPO_ES_RESPONDING:
7349             break;
7350         case MPI2_EVENT_SAS_TOPO_ES_DELAY_NOT_RESPONDING:
7351             (void) sprintf(string, " not responding, "
7352                 "delaying removal");
7353             break;
7354         default:
7355             break;
7356     }
7357     } else {
7358         flags = MPTSAS_TOPO_FLAG_DIRECT_ATTACHED_DEVICE;
7359     }
7360
7361     NDBG20(("SAS TOPOLOGY CHANGE for enclosure %x expander %x%s\n",
7362         enc_handle, expd_handle, string));
7363     for (i = 0; i < num_entries; i++) {
7364         phy = i + start_phy_num;
7365         phystatus = ddi_get8(mpt->m_acc_reply_frame_hdl,
7366             &sas_topo_change_list->PHY[i].PhyStatus);
7367         dev_handle = ddi_get16(mpt->m_acc_reply_frame_hdl,
7368             &sas_topo_change_list->PHY[i].AttachedDevHandle);
7369         reason_code = phystatus & MPI2_EVENT_SAS_TOPO_RC_MASK;
7370         /*
7371          * Filter out processing of Phy Vacant Status unless
7372          * the reason code is "Not Responding". Process all
7373          * other combinations of Phy Status and Reason Codes.
7374          */
7375         if ((phystatus &
7376             MPI2_EVENT_SAS_TOPO_PHYSTATUS_VACANT) &&
7377             (reason_code !=
7378                 MPI2_EVENT_SAS_TOPO_RC_TARG_NOT_RESPONDING)) {
7379             continue;
7380         }
7381         curr[0] = 0;
7382         prev[0] = 0;
7383         string[0] = 0;
7384         switch (reason_code) {
7385         case MPI2_EVENT_SAS_TOPO_RC_TARG_ADDED:
7386             {
7387                 NDBG20(("mptsas3%d phy %d physical_port %d "

```

```

7388         "dev_handle %d added", mpt->m_instance, phy,
7389         physport, dev_handle));
7390     link_rate = ddi_get8(mpt->m_acc_reply_frame_hdl,
7391     &sas_topo_change_list->PHY[i].LinkRate);
7392     state = (link_rate &
7393     MPI2_EVENT_SAS_TOPO_LR_CURRENT_MASK) >>
7394     MPI2_EVENT_SAS_TOPO_LR_CURRENT_SHIFT;
7395     switch (state) {
7396     case MPI2_EVENT_SAS_TOPO_LR_PHY_DISABLED:
7397         (void) sprintf(curr, "is disabled");
7398         break;
7399     case MPI2_EVENT_SAS_TOPO_LR_NEGOTIATION_FAILED:
7400         (void) sprintf(curr, "is offline, "
7401         "failed speed negotiation");
7402         break;
7403     case MPI2_EVENT_SAS_TOPO_LR_SATA_OOB_COMPLETE:
7404         (void) sprintf(curr, "SATA OOB "
7405         "complete");
7406         break;
7407     case SMP_RESET_IN_PROGRESS:
7408         (void) sprintf(curr, "SMP reset in "
7409         "progress");
7410         break;
7411     case MPI2_EVENT_SAS_TOPO_LR_RATE_1_5:
7412         (void) sprintf(curr, "is online at "
7413         "1.5 Gbps");
7414         break;
7415     case MPI2_EVENT_SAS_TOPO_LR_RATE_3_0:
7416         (void) sprintf(curr, "is online at 3.0 "
7417         "Gbps");
7418         break;
7419     case MPI2_EVENT_SAS_TOPO_LR_RATE_6_0:
7420         (void) sprintf(curr, "is online at 6.0 "
7421         "Gbps");
7422         break;
7423     case MPI25_EVENT_SAS_TOPO_LR_RATE_12_0:
7424         (void) sprintf(curr,
7425         "is online at 12.0 Gbps");
7426         break;
7427     default:
7428         (void) sprintf(curr, "state is "
7429         "unknown");
7430         break;
7431     }
7432     /*
7433     * New target device added into the system.
7434     * Set association flag according to if an
7435     * expander is used or not.
7436     */
7437     exp_flag =
7438     MPTSAS_TOPO_FLAG_EXPANDER_ATTACHED_DEVICE;
7439     if (flags ==
7440     MPTSAS_TOPO_FLAG_EXPANDER_ASSOCIATED) {
7441         flags = exp_flag;
7442     }
7443     topo_node = kmem_zalloc(
7444     sizeof (mptsas_topo_change_list_t),
7445     KM_SLEEP);
7446     topo_node->mpt = mpt;
7447     topo_node->event =
7448     MPTSAS_DR_EVENT_RECONFIG_TARGET;
7449     if (expd_handle == 0) {
7450         /*
7451         * Per MPI 2, if expander dev handle
7452         * is 0, it's a directly attached
7453         * device. So driver use PHY to decide

```

```

7454         * which iport is associated
7455         */
7456         physport = phy;
7457         mpt->m_port_chng = 1;
7458     }
7459     topo_node->un.physport = physport;
7460     topo_node->devhdl = dev_handle;
7461     topo_node->flags = flags;
7462     topo_node->object = NULL;
7463     if (topo_head == NULL) {
7464         topo_head = topo_tail = topo_node;
7465     } else {
7466         topo_tail->next = topo_node;
7467         topo_tail = topo_node;
7468     }
7469     break;
7470 }
7471 case MPI2_EVENT_SAS_TOPO_RC_TARG_NOT_RESPONDING:
7472 {
7473     NDBG20(("mptsas3%d phy %d physical_port %d "
7474     "dev_handle %d removed", mpt->m_instance,
7475     phy, physport, dev_handle));
7476     /*
7477     * Set association flag according to if an
7478     * expander is used or not.
7479     */
7480     exp_flag =
7481     MPTSAS_TOPO_FLAG_EXPANDER_ATTACHED_DEVICE;
7482     if (flags ==
7483     MPTSAS_TOPO_FLAG_EXPANDER_ASSOCIATED) {
7484         flags = exp_flag;
7485     }
7486     /*
7487     * Target device is removed from the system
7488     * Before the device is really offline from
7489     * from system.
7490     */
7491     ptgt = rehash_linear_search(mpt->m_targets,
7492     mptsas_target_eval_devhdl, &dev_handle);
7493     /*
7494     * If ptgt is NULL here, it means that the
7495     * DevHandle is not in the hash table. This is
7496     * reasonable sometimes. For example, if a
7497     * disk was pulled, then added, then pulled
7498     * again, the disk will not have been put into
7499     * the hash table because the add event will
7500     * have an invalid phymask. BUT, this does not
7501     * mean that the DevHandle is invalid. The
7502     * controller will still have a valid DevHandle
7503     * that must be removed. To do this, use the
7504     * MPTSAS_DR_EVENT_REMOVE_HANDLE event.
7505     */
7506     if (ptgt == NULL) {
7507         topo_node = kmem_zalloc(
7508         sizeof (mptsas_topo_change_list_t),
7509         KM_SLEEP);
7510         topo_node->mpt = mpt;
7511         topo_node->un.phymask = 0;
7512         topo_node->event =
7513         MPTSAS_DR_EVENT_REMOVE_HANDLE;
7514         topo_node->devhdl = dev_handle;
7515         topo_node->flags = flags;
7516         topo_node->object = NULL;
7517         if (topo_head == NULL) {
7518             topo_head = topo_tail =
7519             topo_node;

```

```

7520         } else {
7521             topo_tail->next = topo_node;
7522             topo_tail = topo_node;
7523         }
7524         break;
7525     }
7526
7527     /*
7528     * Update DR flag immediately avoid I/O failure
7529     * before failover finish. We won't add
7530     * any following commands into waitq, instead,
7531     * we need return TRAN_BUSY in the tran_start
7532     * context.
7533     */
7534     ptgt->m_dr_flag = MPTSAS_DR_INTRANSITION;
7535
7536     topo_node = kmem_zalloc(
7537         sizeof (mptsas_topo_change_list_t),
7538         KM_SLEEP);
7539     topo_node->mpt = mpt;
7540     topo_node->un.phymask =
7541         ptgt->m_addr.mta.phymask;
7542     topo_node->event =
7543         MPTSAS_DR_EVENT_OFFLINE_TARGET;
7544     topo_node->devhdl = dev_hdl;
7545     topo_node->flags = flags;
7546     topo_node->object = NULL;
7547     if (topo_head == NULL) {
7548         topo_head = topo_tail = topo_node;
7549     } else {
7550         topo_tail->next = topo_node;
7551         topo_tail = topo_node;
7552     }
7553     break;
7554 }
7555 case MPI2_EVENT_SAS_TOPO_RC_PHY_CHANGED:
7556     link_rate = ddi_get8(mpt->m_acc_reply_frame_hdl,
7557         &sas_topo_change_list->PHY[i].LinkRate);
7558     state = (link_rate &
7559         MPI2_EVENT_SAS_TOPO_LR_CURRENT_MASK) >>
7560         MPI2_EVENT_SAS_TOPO_LR_CURRENT_SHIFT;
7561     pSmhba = &mpt->m_phy_info[i].smhba_info;
7562     pSmhba->negotiated_link_rate = state;
7563     switch (state) {
7564     case MPI2_EVENT_SAS_TOPO_LR_PHY_DISABLED:
7565         (void) sprintf(curr, "is disabled");
7566         mptsas_smhba_log_sysevent(mpt,
7567             ESC_SAS_PHY_EVENT,
7568             SAS_PHY_REMOVE,
7569             &mpt->m_phy_info[i].smhba_info);
7570         mpt->m_phy_info[i].smhba_info.
7571             negotiated_link_rate
7572             = 0x1;
7573         break;
7574     case MPI2_EVENT_SAS_TOPO_LR_NEGOTIATION_FAILED:
7575         (void) sprintf(curr, "is offline, "
7576             "failed speed negotiation");
7577         mptsas_smhba_log_sysevent(mpt,
7578             ESC_SAS_PHY_EVENT,
7579             SAS_PHY_OFFLINE,
7580             &mpt->m_phy_info[i].smhba_info);
7581         break;
7582     case MPI2_EVENT_SAS_TOPO_LR_SATA_OOB_COMPLETE:
7583         (void) sprintf(curr, "SATA OOB "
7584             "complete");
7585         break;

```

```

7586     case SMP_RESET_IN_PROGRESS:
7587         (void) sprintf(curr, "SMP reset in "
7588             "progress");
7589         break;
7590     case MPI2_EVENT_SAS_TOPO_LR_RATE_1_5:
7591         (void) sprintf(curr, "is online at "
7592             "1.5 Gbps");
7593         if ((expd_handle == 0) &&
7594             (enc_handle == 1)) {
7595             mpt->m_port_chng = 1;
7596         }
7597         mptsas_smhba_log_sysevent(mpt,
7598             ESC_SAS_PHY_EVENT,
7599             SAS_PHY_ONLINE,
7600             &mpt->m_phy_info[i].smhba_info);
7601         break;
7602     case MPI2_EVENT_SAS_TOPO_LR_RATE_3_0:
7603         (void) sprintf(curr, "is online at 3.0 "
7604             "Gbps");
7605         if ((expd_handle == 0) &&
7606             (enc_handle == 1)) {
7607             mpt->m_port_chng = 1;
7608         }
7609         mptsas_smhba_log_sysevent(mpt,
7610             ESC_SAS_PHY_EVENT,
7611             SAS_PHY_ONLINE,
7612             &mpt->m_phy_info[i].smhba_info);
7613         break;
7614     case MPI2_EVENT_SAS_TOPO_LR_RATE_6_0:
7615         (void) sprintf(curr, "is online at "
7616             "6.0 Gbps");
7617         if ((expd_handle == 0) &&
7618             (enc_handle == 1)) {
7619             mpt->m_port_chng = 1;
7620         }
7621         mptsas_smhba_log_sysevent(mpt,
7622             ESC_SAS_PHY_EVENT,
7623             SAS_PHY_ONLINE,
7624             &mpt->m_phy_info[i].smhba_info);
7625         break;
7626     case MPI25_EVENT_SAS_TOPO_LR_RATE_12_0:
7627         (void) sprintf(curr, "is online at "
7628             "12.0 Gbps");
7629         if ((expd_handle == 0) &&
7630             (enc_handle == 1)) {
7631             mpt->m_port_chng = 1;
7632         }
7633         mptsas_smhba_log_sysevent(mpt,
7634             ESC_SAS_PHY_EVENT,
7635             SAS_PHY_ONLINE,
7636             &mpt->m_phy_info[i].smhba_info);
7637         break;
7638     default:
7639         (void) sprintf(curr, "state is "
7640             "unknown");
7641         break;
7642 }
7643
7644 state = (link_rate &
7645     MPI2_EVENT_SAS_TOPO_LR_PREV_MASK) >>
7646     MPI2_EVENT_SAS_TOPO_LR_PREV_SHIFT;
7647 switch (state) {
7648 case MPI2_EVENT_SAS_TOPO_LR_PHY_DISABLED:
7649     (void) sprintf(prev, ", was disabled");
7650     break;
7651 case MPI2_EVENT_SAS_TOPO_LR_NEGOTIATION_FAILED:

```

```

7652         (void) sprintf(prev, ", was offline, "
7653             "failed speed negotiation");
7654         break;
7655     case MPI2_EVENT_SAS_TOPO_LR_SATA_OOB_COMPLETE:
7656         (void) sprintf(prev, ", was SATA OOB "
7657             "complete");
7658         break;
7659     case SMP_RESET_IN_PROGRESS:
7660         (void) sprintf(prev, ", was SMP reset "
7661             "in progress");
7662         break;
7663     case MPI2_EVENT_SAS_TOPO_LR_RATE_1_5:
7664         (void) sprintf(prev, ", was online at "
7665             "1.5 Gbps");
7666         break;
7667     case MPI2_EVENT_SAS_TOPO_LR_RATE_3_0:
7668         (void) sprintf(prev, ", was online at "
7669             "3.0 Gbps");
7670         break;
7671     case MPI2_EVENT_SAS_TOPO_LR_RATE_6_0:
7672         (void) sprintf(prev, ", was online at "
7673             "6.0 Gbps");
7674         break;
7675     case MPI25_EVENT_SAS_TOPO_LR_RATE_12_0:
7676         (void) sprintf(prev, ", was online at "
7677             "12.0 Gbps");
7678         break;
7679     default:
7680         break;
7681     }
7682     (void) sprintf(&string[strlen(string)], "link "
7683         "changed, ");
7684     break;
7685     case MPI2_EVENT_SAS_TOPO_RC_NO_CHANGE:
7686         continue;
7687     case MPI2_EVENT_SAS_TOPO_RC_DELAY_NOT_RESPONDING:
7688         (void) sprintf(&string[strlen(string)],
7689             "target not responding, delaying "
7690             "removal");
7691         break;
7692     }
7693     NDBG20(("mptsas3%d phy %d DevHandle %x, %s%s\n",
7694         mpt->m_instance, phy, dev_handle, string, curr,
7695         prev));
7696 }
7697 if (topo_head != NULL) {
7698     /*
7699      * Launch DR taskq to handle topology change
7700      */
7701     if ((ddi_taskq_dispatch(mpt->m_dr_taskq,
7702         mptsas_handle_dr, (void *)topo_head,
7703         DDI_NOSLEEP)) != DDI_SUCCESS) {
7704         while (topo_head != NULL) {
7705             topo_node = topo_head;
7706             topo_head = topo_head->next;
7707             kmem_free(topo_node,
7708                 sizeof (mptsas_topo_change_list_t));
7709         }
7710         mptsas_log(mpt, CE_NOTE, "mptsas start taskq "
7711             "for handle SAS DR event failed. \n");
7712     }
7713 }
7714 break;
7715 }
7716 case MPI2_EVENT_IR_CONFIGURATION_CHANGE_LIST:
7717 {

```

```

7718     Mpi2EventDataIrConfigChangeList_t      *irChangeList;
7719     mptsas_topo_change_list_t               *topo_head = NULL;
7720     mptsas_topo_change_list_t               *topo_tail = NULL;
7721     mptsas_topo_change_list_t               *topo_node = NULL;
7722     mptsas_target_t                         *ptgt;
7723     uint8_t                                 num_entries, i, reason;
7724     uint16_t                                volhandle, diskhandle;

7726     irChangeList = (pMpi2EventDataIrConfigChangeList_t)
7727         eventreply->EventData;
7728     num_entries = ddi_get8(mpt->m_acc_reply_frame_hdl,
7729         &irChangeList->NumElements);

7731     NDBG20(("mptsas3%d IR_CONFIGURATION_CHANGE_LIST event received",
7732         mpt->m_instance));

7734     for (i = 0; i < num_entries; i++) {
7735         reason = ddi_get8(mpt->m_acc_reply_frame_hdl,
7736             &irChangeList->ConfigElement[i].ReasonCode);
7737         volhandle = ddi_get16(mpt->m_acc_reply_frame_hdl,
7738             &irChangeList->ConfigElement[i].VolDevHandle);
7739         diskhandle = ddi_get16(mpt->m_acc_reply_frame_hdl,
7740             &irChangeList->ConfigElement[i].PhysDiskDevHandle);

7742         switch (reason) {
7743             case MPI2_EVENT_IR_CHANGE_RC_ADDED:
7744             case MPI2_EVENT_IR_CHANGE_RC_VOLUME_CREATED:
7745             {
7746                 NDBG20(("mptsas %d volume added\n",
7747                     mpt->m_instance));

7749                 topo_node = kmem_zalloc(
7750                     sizeof (mptsas_topo_change_list_t),
7751                     KM_SLEEP);

7753                 topo_node->mpt = mpt;
7754                 topo_node->event =
7755                     MPTSAS_DR_EVENT_RECONFIG_TARGET;
7756                 topo_node->un.physport = 0xff;
7757                 topo_node->devhdl = volhandle;
7758                 topo_node->flags =
7759                     MPTSAS_TOPO_FLAG_RAID_ASSOCIATED;
7760                 topo_node->object = NULL;
7761                 if (topo_head == NULL) {
7762                     topo_head = topo_node;
7763                 } else {
7764                     topo_tail->next = topo_node;
7765                     topo_tail = topo_node;
7766                 }
7767                 break;
7768             }
7769             case MPI2_EVENT_IR_CHANGE_RC_REMOVED:
7770             case MPI2_EVENT_IR_CHANGE_RC_VOLUME_DELETED:
7771             {
7772                 NDBG20(("mptsas %d volume deleted\n",
7773                     mpt->m_instance));
7774                 ptgt = rehash_linear_search(mpt->m_targets,
7775                     mptsas_target_eval_devhdl, &volhandle);
7776                 if (ptgt == NULL)
7777                     break;

7779                 /*
7780                  * Clear any flags related to volume
7781                  */
7782                 (void) mptsas_delete_volume(mpt, volhandle);

```

```

7784      /*
7785       * Update DR flag immediately avoid I/O failure
7786       */
7787      ptgt->m_dr_flag = MPTSAS_DR_INTRANSITION;

7789      topo_node = kmem_zalloc(
7790          sizeof(mptsas_topo_change_list_t),
7791          KM_SLEEP);
7792      topo_node->mpt = mpt;
7793      topo_node->un.phymask =
7794          ptgt->m_addr.mta_phymask;
7795      topo_node->event =
7796          MPTSAS_DR_EVENT_OFFLINE_TARGET;
7797      topo_node->devhdl = volhandle;
7798      topo_node->flags =
7799          MPTSAS_TOPO_FLAG_RAID_ASSOCIATED;
7800      topo_node->object = (void *)ptgt;
7801      if (topo_head == NULL) {
7802          topo_head = topo_tail = topo_node;
7803      } else {
7804          topo_tail->next = topo_node;
7805          topo_tail = topo_node;
7806      }
7807      break;
7808  }
7809  case MPI2_EVENT_IR_CHANGE_RC_PD_CREATED:
7810  case MPI2_EVENT_IR_CHANGE_RC_HIDE:
7811  {
7812      ptgt = rehash_linear_search(mpt->m_targets,
7813          mptsas_target_eval_devhdl, &diskhandle);
7814      if (ptgt == NULL)
7815          break;

7817      /*
7818       * Update DR flag immediately avoid I/O failure
7819       */
7820      ptgt->m_dr_flag = MPTSAS_DR_INTRANSITION;

7822      topo_node = kmem_zalloc(
7823          sizeof(mptsas_topo_change_list_t),
7824          KM_SLEEP);
7825      topo_node->mpt = mpt;
7826      topo_node->un.phymask =
7827          ptgt->m_addr.mta_phymask;
7828      topo_node->event =
7829          MPTSAS_DR_EVENT_OFFLINE_TARGET;
7830      topo_node->devhdl = diskhandle;
7831      topo_node->flags =
7832          MPTSAS_TOPO_FLAG_RAID_PHYSDRV_ASSOCIATED;
7833      topo_node->object = (void *)ptgt;
7834      if (topo_head == NULL) {
7835          topo_head = topo_tail = topo_node;
7836      } else {
7837          topo_tail->next = topo_node;
7838          topo_tail = topo_node;
7839      }
7840      break;
7841  }
7842  case MPI2_EVENT_IR_CHANGE_RC_UNHIDE:
7843  case MPI2_EVENT_IR_CHANGE_RC_PD_DELETED:
7844  {
7845      /*
7846       * The physical drive is released by a IR
7847       * volume. But we cannot get the the physport
7848       * or phnum from the event data, so we only
7849       * can get the physport/phnum after SAS

```

```

7850      * Device Page0 request for the devhdl.
7851      */
7852      topo_node = kmem_zalloc(
7853          sizeof(mptsas_topo_change_list_t),
7854          KM_SLEEP);
7855      topo_node->mpt = mpt;
7856      topo_node->un.phymask = 0;
7857      topo_node->event =
7858          MPTSAS_DR_EVENT_RECONFIG_TARGET;
7859      topo_node->devhdl = diskhandle;
7860      topo_node->flags =
7861          MPTSAS_TOPO_FLAG_RAID_PHYSDRV_ASSOCIATED;
7862      topo_node->object = NULL;
7863      mpt->m_port_chng = 1;
7864      if (topo_head == NULL) {
7865          topo_head = topo_tail = topo_node;
7866      } else {
7867          topo_tail->next = topo_node;
7868          topo_tail = topo_node;
7869      }
7870      break;
7871  }
7872  default:
7873      break;
7874  }
7875  }

7877      if (topo_head != NULL) {
7878          /*
7879           * Launch DR taskq to handle topology change
7880           */
7881          if ((ddi_taskq_dispatch(mpt->m_dr_taskq,
7882              mptsas_handle_dr, (void *)topo_head,
7883              DDI_NOSLEEP)) != DDI_SUCCESS) {
7884              while (topo_head != NULL) {
7885                  topo_node = topo_head;
7886                  topo_head = topo_head->next;
7887                  kmem_free(topo_node,
7888                      sizeof(mptsas_topo_change_list_t));
7889              }
7890              mptsas_log(mpt, CE_NOTE, "mptsas start taskq "
7891                  "for handle SAS DR event failed. \n");
7892          }
7893      }
7894      break;
7895  }
7896  default:
7897      return (DDI_FAILURE);
7898  }

7900      return (DDI_SUCCESS);
7901  }

7903  /*
7904   * handle events from ioc
7905   */
7906  static void
7907  mptsas_handle_event(void *args)
7908  {
7909      m_replyh_arg_t      *replyh_arg;
7910      pMpi2EventNotificationReply_t eventreply;
7911      uint32_t             event, iocloginfo, rfm;
7912      uint32_t             status;
7913      uint8_t              port;
7914      mptsas_t             *mpt;
7915      uint_t               iocstatus;

```

```

7917     replyh_arg = (m_replyh_arg_t *)args;
7918     rfm = replyh_arg->rfm;
7919     mpt = replyh_arg->mpt;

7921     mutex_enter(&mpt->m_mutex);
7922     /*
7923      * If HBA is being reset, drop incoming event.
7924      */
7925     if (mpt->m_in_reset) {
7926         NDBG20(("dropping event received prior to reset"));
7927         mutex_exit(&mpt->m_mutex);
7928         return;
7929     }

7931     eventreply = (pMpi2EventNotificationReply_t)
7932         (mpt->m_reply_frame + (rfm -
7933             (mpt->m_reply_frame_dma_addr&0xffffffff)));
7934     event = ddi_get16(mpt->m_acc_reply_frame_hdl, &eventreply->Event);

7936     if (iocstatus = ddi_get16(mpt->m_acc_reply_frame_hdl,
7937         &eventreply->IOCStatus)) {
7938         if (iocstatus == MPI2_IOCSTATUS_FLAG_LOG_INFO_AVAILABLE) {
7939             mptsas_log(mpt, CE_WARN,
7940                 "!mptsas_handle_event: IOCStatus=0x%x, "
7941                 "IOCLogInfo=0x%x", iocstatus,
7942                 ddi_get32(mpt->m_acc_reply_frame_hdl,
7943                     &eventreply->IOCLogInfo));
7944         } else {
7945             mptsas_log(mpt, CE_WARN,
7946                 "mptsas_handle_event: IOCStatus=0x%x, "
7947                 "IOCLogInfo=0x%x", iocstatus,
7948                 ddi_get32(mpt->m_acc_reply_frame_hdl,
7949                     &eventreply->IOCLogInfo));
7950         }
7951     }

7953     /*
7954      * figure out what kind of event we got and handle accordingly
7955      */
7956     switch (event) {
7957     case MPI2_EVENT_LOG_ENTRY_ADDED:
7958         break;
7959     case MPI2_EVENT_LOG_DATA:
7960         iocloginfo = ddi_get32(mpt->m_acc_reply_frame_hdl,
7961             &eventreply->IOCLogInfo);
7962         NDBG20(("mptsas %d log info %x received.\n", mpt->m_instance,
7963             iocloginfo));
7964         break;
7965     case MPI2_EVENT_STATE_CHANGE:
7966         NDBG20(("mptsas%d state change.", mpt->m_instance));
7967         break;
7968     case MPI2_EVENT_HARD_RESET_RECEIVED:
7969         NDBG20(("mptsas%d event change.", mpt->m_instance));
7970         break;
7971     case MPI2_EVENT_SAS_DISCOVERY:
7972     {
7973         MPI2_EVENT_DATA_SAS_DISCOVERY *sasdiscovery;
7974         char string[80];
7975         uint8_t rc;

7977         sasdiscovery =
7978             (pMpi2EventDataSasDiscovery_t)eventreply->EventData;

7980         rc = ddi_get8(mpt->m_acc_reply_frame_hdl,
7981             &sasdiscovery->ReasonCode);

```

```

7982         port = ddi_get8(mpt->m_acc_reply_frame_hdl,
7983             &sasdiscovery->PhysicalPort);
7984         status = ddi_get32(mpt->m_acc_reply_frame_hdl,
7985             &sasdiscovery->DiscoveryStatus);

7987         string[0] = 0;
7988         switch (rc) {
7989         case MPI2_EVENT_SAS_DISC_RC_STARTED:
7990             (void) sprintf(string, "STARTING");
7991             break;
7992         case MPI2_EVENT_SAS_DISC_RC_COMPLETED:
7993             (void) sprintf(string, "COMPLETED");
7994             break;
7995         default:
7996             (void) sprintf(string, "UNKNOWN");
7997             break;
7998         }

8000         NDBG20(("SAS DISCOVERY is %s for port %d, status %x", string,
8001             port, status));

8003         break;
8004     }
8005     case MPI2_EVENT_EVENT_CHANGE:
8006         NDBG20(("mptsas%d event change.", mpt->m_instance));
8007         break;
8008     case MPI2_EVENT_TASK_SET_FULL:
8009     {
8010         pMpi2EventDataTaskSetFull_t taskfull;

8012         taskfull = (pMpi2EventDataTaskSetFull_t)eventreply->EventData;

8014         NDBG20(("TASK_SET_FULL received for mptsas%d, depth %d\n",
8015             mpt->m_instance, ddi_get16(mpt->m_acc_reply_frame_hdl,
8016                 &taskfull->CurrentDepth)));
8017         break;
8018     }
8019     case MPI2_EVENT_SAS_TOPOLOGY_CHANGE_LIST:
8020     {
8021         /*
8022          * SAS TOPOLOGY CHANGE LIST Event has already been handled
8023          * in mptsas_handle_event_sync() of interrupt context
8024          */
8025         break;
8026     }
8027     case MPI2_EVENT_SAS_ENCL_DEVICE_STATUS_CHANGE:
8028     {
8029         pMpi2EventDataSasEnclDevStatusChange_t encstatus;
8030         uint8_t rc;
8031         char string[80];

8033         encstatus = (pMpi2EventDataSasEnclDevStatusChange_t)
8034             eventreply->EventData;

8036         rc = ddi_get8(mpt->m_acc_reply_frame_hdl,
8037             &encstatus->ReasonCode);
8038         switch (rc) {
8039         case MPI2_EVENT_SAS_ENCL_RC_ADDED:
8040             (void) sprintf(string, "added");
8041             break;
8042         case MPI2_EVENT_SAS_ENCL_RC_NOT_RESPONDING:
8043             (void) sprintf(string, ", not responding");
8044             break;
8045         default:
8046             break;
8047         }

```

```

8048         NDBG20(("mptsas3%d ENCLOSURE STATUS CHANGE for enclosure "
8049             "%x%s\n", mpt->m_instance,
8050             ddi_get16(mpt->m_acc_reply_frame_hdl,
8051                 &encstatus->EnclosureHandle), string));
8052         break;
8053     }

8055     /*
8056     * MPI2_EVENT_SAS_DEVICE_STATUS_CHANGE is handled by
8057     * mptsas_handle_event_sync, in here just send ack message.
8058     */
8059     case MPI2_EVENT_SAS_DEVICE_STATUS_CHANGE:
8060     {
8061         pMpi2EventDataSasDeviceStatusChange_t    statuschange;
8062         uint8_t                                    rc;
8063         uint16_t                                   devhdl;
8064         uint64_t                                   wwn = 0;
8065         uint32_t                                   wwn_lo, wwn_hi;

8067         statuschange = (pMpi2EventDataSasDeviceStatusChange_t)
8068             eventreply->EventData;
8069         rc = ddi_get8(mpt->m_acc_reply_frame_hdl,
8070             &statuschange->ReasonCode);
8071         wwn_lo = ddi_get32(mpt->m_acc_reply_frame_hdl,
8072             (uint32_t *) (void *) &statuschange->SASAddress);
8073         wwn_hi = ddi_get32(mpt->m_acc_reply_frame_hdl,
8074             (uint32_t *) (void *) &statuschange->SASAddress + 1);
8075         wwn = ((uint64_t) wwn_hi << 32) | wwn_lo;
8076         devhdl = ddi_get16(mpt->m_acc_reply_frame_hdl,
8077             &statuschange->DevHandle);

8079         NDBG13(("MPI2_EVENT_SAS_DEVICE_STATUS_CHANGE wwn is %"PRIx64,
8080             wwn));

8082         switch (rc) {
8083         case MPI2_EVENT_SAS_DEV_STAT_RC_SMART_DATA:
8084             NDBG20(("SMART data received, ASC/ASCQ = %02x/%02x",
8085                 ddi_get8(mpt->m_acc_reply_frame_hdl,
8086                     &statuschange->ASC),
8087                 ddi_get8(mpt->m_acc_reply_frame_hdl,
8088                     &statuschange->ASCQ)));
8089             break;

8091         case MPI2_EVENT_SAS_DEV_STAT_RC_UNSUPPORTED:
8092             NDBG20(("Device not supported"));
8093             break;

8095         case MPI2_EVENT_SAS_DEV_STAT_RC_INTERNAL_DEVICE_RESET:
8096             NDBG20(("IOC internally generated the Target Reset "
8097                 "for devhdl:%x", devhdl));
8098             break;

8100         case MPI2_EVENT_SAS_DEV_STAT_RC_CMP_INTERNAL_DEV_RESET:
8101             NDBG20(("IOC's internally generated Target Reset "
8102                 "completed for devhdl:%x", devhdl));
8103             break;

8105         case MPI2_EVENT_SAS_DEV_STAT_RC_TASK_ABORT_INTERNAL:
8106             NDBG20(("IOC internally generated Abort Task"));
8107             break;

8109         case MPI2_EVENT_SAS_DEV_STAT_RC_CMP_TASK_ABORT_INTERNAL:
8110             NDBG20(("IOC's internally generated Abort Task "
8111                 "completed"));
8112             break;

```

```

8114         case MPI2_EVENT_SAS_DEV_STAT_RC_ABORT_TASK_SET_INTERNAL:
8115             NDBG20(("IOC internally generated Abort Task Set"));
8116             break;

8118         case MPI2_EVENT_SAS_DEV_STAT_RC_CLEAR_TASK_SET_INTERNAL:
8119             NDBG20(("IOC internally generated Clear Task Set"));
8120             break;

8122         case MPI2_EVENT_SAS_DEV_STAT_RC_QUERY_TASK_INTERNAL:
8123             NDBG20(("IOC internally generated Query Task"));
8124             break;

8126         case MPI2_EVENT_SAS_DEV_STAT_RC_ASYNC_NOTIFICATION:
8127             NDBG20(("Device sent an Asynchronous Notification"));
8128             break;

8130         default:
8131             break;
8132     }
8133     break;
8134 }
8135 case MPI2_EVENT_IR_CONFIGURATION_CHANGE_LIST:
8136 {
8137     /*
8138     * IR TOPOLOGY CHANGE LIST Event has already been handled
8139     * in mpt_handle_event_sync() of interrupt context
8140     */
8141     break;
8142 }
8143 case MPI2_EVENT_IR_OPERATION_STATUS:
8144 {
8145     Mpi2EventDataIrOperationStatus_t    *irOpStatus;
8146     char                                reason_str[80];
8147     uint8_t                              rc, percent;
8148     uint16_t                             handle;

8150     irOpStatus = (pMpi2EventDataIrOperationStatus_t)
8151         eventreply->EventData;
8152     rc = ddi_get8(mpt->m_acc_reply_frame_hdl,
8153         &irOpStatus->RAIDOperation);
8154     percent = ddi_get8(mpt->m_acc_reply_frame_hdl,
8155         &irOpStatus->PercentComplete);
8156     handle = ddi_get16(mpt->m_acc_reply_frame_hdl,
8157         &irOpStatus->VolDevHandle);

8159     switch (rc) {
8160     case MPI2_EVENT_IR_RAIDOP_RESYNC:
8161         (void) sprintf(reason_str, "resync");
8162         break;
8163     case MPI2_EVENT_IR_RAIDOP_ONLINE_CAP_EXPANSION:
8164         (void) sprintf(reason_str, "online capacity "
8165             "expansion");
8166         break;
8167     case MPI2_EVENT_IR_RAIDOP_CONSISTENCY_CHECK:
8168         (void) sprintf(reason_str, "consistency check");
8169         break;
8170     default:
8171         (void) sprintf(reason_str, "unknown reason %x",
8172             rc);
8173     }

8175     NDBG20(("mptsas3%d raid operational status: (%s)"
8176         "\thandle(0x%04x), percent complete(%d)\n",
8177         mpt->m_instance, reason_str, handle, percent));
8178     break;
8179 }

```

```

8180     case MPI2_EVENT_SAS_BROADCAST_PRIMITIVE:
8181     {
8182         pMpi2EventDataSasBroadcastPrimitive_t    sas_broadcast;
8183         uint8_t                                   phy_num;
8184         uint8_t                                   primitive;

8186         sas_broadcast = (pMpi2EventDataSasBroadcastPrimitive_t)
8187             eventreply->EventData;

8189         phy_num = ddi_get8(mpt->m_acc_reply_frame_hdl,
8190             &sas_broadcast->PhyNum);
8191         primitive = ddi_get8(mpt->m_acc_reply_frame_hdl,
8192             &sas_broadcast->Primitive);

8194         switch (primitive) {
8195         case MPI2_EVENT_PRIMITIVE_CHANGE:
8196             mptsas_smhba_log_sysevent(mpt,
8197                 ESC_SAS_HBA_PORT_BROADCAST,
8198                 SAS_PORT_BROADCAST_CHANGE,
8199                 &mpt->m_phy_info[phy_num].smhba_info);
8200             break;
8201         case MPI2_EVENT_PRIMITIVE_SES:
8202             mptsas_smhba_log_sysevent(mpt,
8203                 ESC_SAS_HBA_PORT_BROADCAST,
8204                 SAS_PORT_BROADCAST_SES,
8205                 &mpt->m_phy_info[phy_num].smhba_info);
8206             break;
8207         case MPI2_EVENT_PRIMITIVE_EXPANDER:
8208             mptsas_smhba_log_sysevent(mpt,
8209                 ESC_SAS_HBA_PORT_BROADCAST,
8210                 SAS_PORT_BROADCAST_D01_4,
8211                 &mpt->m_phy_info[phy_num].smhba_info);
8212             break;
8213         case MPI2_EVENT_PRIMITIVE_ASYNCHRONOUS_EVENT:
8214             mptsas_smhba_log_sysevent(mpt,
8215                 ESC_SAS_HBA_PORT_BROADCAST,
8216                 SAS_PORT_BROADCAST_D04_7,
8217                 &mpt->m_phy_info[phy_num].smhba_info);
8218             break;
8219         case MPI2_EVENT_PRIMITIVE_RESERVED3:
8220             mptsas_smhba_log_sysevent(mpt,
8221                 ESC_SAS_HBA_PORT_BROADCAST,
8222                 SAS_PORT_BROADCAST_D16_7,
8223                 &mpt->m_phy_info[phy_num].smhba_info);
8224             break;
8225         case MPI2_EVENT_PRIMITIVE_RESERVED4:
8226             mptsas_smhba_log_sysevent(mpt,
8227                 ESC_SAS_HBA_PORT_BROADCAST,
8228                 SAS_PORT_BROADCAST_D29_7,
8229                 &mpt->m_phy_info[phy_num].smhba_info);
8230             break;
8231         case MPI2_EVENT_PRIMITIVE_CHANGE0_RESERVED:
8232             mptsas_smhba_log_sysevent(mpt,
8233                 ESC_SAS_HBA_PORT_BROADCAST,
8234                 SAS_PORT_BROADCAST_D24_0,
8235                 &mpt->m_phy_info[phy_num].smhba_info);
8236             break;
8237         case MPI2_EVENT_PRIMITIVE_CHANGE1_RESERVED:
8238             mptsas_smhba_log_sysevent(mpt,
8239                 ESC_SAS_HBA_PORT_BROADCAST,
8240                 SAS_PORT_BROADCAST_D27_4,
8241                 &mpt->m_phy_info[phy_num].smhba_info);
8242             break;
8243         default:
8244             NDBG16(("mptsas3%d: unknown BROADCAST PRIMITIVE
8245                 " %x received",

```

```

8246             mpt->m_instance, primitive));
8247             break;
8248         }
8249         NDBG16(("mptsas3%d sas broadcast primitive: "
8250             "\tprimitive(0x%04x), phy(%d) complete\n",
8251             mpt->m_instance, primitive, phy_num));
8252         break;
8253     }
8254     case MPI2_EVENT_IR_VOLUME:
8255     {
8256         Mpi2EventDataIrVolume_t    *irVolume;
8257         uint16_t                   devhandle;
8258         uint32_t                   state;
8259         int                        config, vol;
8260         uint8_t                    found = FALSE;

8262         irVolume = (pMpi2EventDataIrVolume_t)eventreply->EventData;
8263         state = ddi_get32(mpt->m_acc_reply_frame_hdl,
8264             &irVolume->NewValue);
8265         devhandle = ddi_get16(mpt->m_acc_reply_frame_hdl,
8266             &irVolume->VolDevHandle);

8268         NDBG20(("EVENT_IR_VOLUME event is received"));

8270         /*
8271          * Get latest RAID info and then find the DevHandle for this
8272          * event in the configuration. If the DevHandle is not found
8273          * just exit the event.
8274          */
8275         (void) mptsas_get_raid_info(mpt);
8276         for (config = 0; (config < mpt->m_num_raid_configs) &&
8277             (!found); config++) {
8278             for (vol = 0; vol < MPTSAS_MAX_RAIDVOL; vol++) {
8279                 if (mpt->m_raidconfig[config].m_raidvol[vol].
8280                     m_raidhandle == devhandle) {
8281                     found = TRUE;
8282                     break;
8283                 }
8284             }
8285         }
8286         if (!found) {
8287             break;
8288         }

8290         switch (irVolume->ReasonCode) {
8291         case MPI2_EVENT_IR_VOLUME_RC_SETTINGS_CHANGED:
8292         {
8293             uint32_t i;
8294             mpt->m_raidconfig[config].m_raidvol[vol].m_settings =
8295                 state;

8297             i = state & MPI2_RAIDVOL0_SETTING_MASK_WRITE_CACHING;
8298             mptsas_log(mpt, CE_NOTE, " Volume %d settings changed"
8299                 ", auto-config of hot-swap drives is %s"
8300                 ", write caching is %s"
8301                 ", hot-spare pool mask is %02x\n",
8302                 vol, state &
8303                 MPI2_RAIDVOL0_SETTING_AUTO_CONFIG_HSWAP_DISABLE
8304                 ? "disabled" : "enabled",
8305                 i == MPI2_RAIDVOL0_SETTING_UNCHANGED
8306                 ? "controlled by member disks" :
8307                 i == MPI2_RAIDVOL0_SETTING_DISABLE_WRITE_CACHING
8308                 ? "disabled" :
8309                 i == MPI2_RAIDVOL0_SETTING_ENABLE_WRITE_CACHING
8310                 ? "enabled" :
8311                 "incorrectly set",

```



```

8312         (state >> 16) & 0xff);
8313         break;
8314     }
8315     case MPI2_EVENT_IR_VOLUME_RC_STATE_CHANGED:
8316     {
8317         mpt->m_raidconfig[config].m_raidvol[vol].m_state =
8318             (uint8_t)state;

8320         mptsas_log(mpt, CE_NOTE,
8321             "Volume %d is now %s\n", vol,
8322             state == MPI2_RAID_VOL_STATE_OPTIMAL
8323             ? "optimal" :
8324             state == MPI2_RAID_VOL_STATE_DEGRADED
8325             ? "degraded" :
8326             state == MPI2_RAID_VOL_STATE_ONLINE
8327             ? "online" :
8328             state == MPI2_RAID_VOL_STATE_INITIALIZING
8329             ? "initializing" :
8330             state == MPI2_RAID_VOL_STATE_FAILED
8331             ? "failed" :
8332             state == MPI2_RAID_VOL_STATE_MISSING
8333             ? "missing" :
8334             "state unknown");
8335         break;
8336     }
8337     case MPI2_EVENT_IR_VOLUME_RC_STATUS_FLAGS_CHANGED:
8338     {
8339         mpt->m_raidconfig[config].m_raidvol[vol].
8340             m_statusflags = state;

8342         mptsas_log(mpt, CE_NOTE,
8343             " Volume %d is now %s%s%s%s%s%s%s\n",
8344             vol,
8345             state & MPI2_RAIDVOL0_STATUS_FLAG_ENABLED
8346             ? ", enabled" : ", disabled",
8347             state & MPI2_RAIDVOL0_STATUS_FLAG_QUIESCED
8348             ? ", quiesced" : "",
8349             state & MPI2_RAIDVOL0_STATUS_FLAG_VOLUME_INACTIVE
8350             ? ", inactive" : ", active",
8351             state &
8352             MPI2_RAIDVOL0_STATUS_FLAG_BAD_BLOCK_TABLE_FULL
8353             ? ", bad block table is full" : "",
8354             state &
8355             MPI2_RAIDVOL0_STATUS_FLAG_RESYNC_IN_PROGRESS
8356             ? ", resync in progress" : "",
8357             state & MPI2_RAIDVOL0_STATUS_FLAG_BACKGROUND_INIT
8358             ? ", background initialization in progress" : "",
8359             state &
8360             MPI2_RAIDVOL0_STATUS_FLAG_CAPACITY_EXPANSION
8361             ? ", capacity expansion in progress" : "",
8362             state &
8363             MPI2_RAIDVOL0_STATUS_FLAG_CONSISTENCY_CHECK
8364             ? ", consistency check in progress" : "",
8365             state & MPI2_RAIDVOL0_STATUS_FLAG_DATA_SCRUB
8366             ? ", data scrub in progress" : "");
8367         break;
8368     }
8369     default:
8370         break;
8371 }
8372 break;
8373 }
8374 case MPI2_EVENT_IR_PHYSICAL_DISK:
8375 {
8376     Mpi2EventDataIrPhysicalDisk_t *irPhysDisk;
8377     uint16_t

```

```

8378         uint32_t          status, state;
8379         uint8_t           physdisknum, reason;

8381         irPhysDisk = (Mpi2EventDataIrPhysicalDisk_t *)
8382             eventreply->EventData;
8383         physdisknum = ddi_get8(mpt->m_acc_reply_frame_hdl,
8384             &irPhysDisk->PhysDiskNum);
8385         devhandle = ddi_get16(mpt->m_acc_reply_frame_hdl,
8386             &irPhysDisk->PhysDiskDevHandle);
8387         enchandle = ddi_get16(mpt->m_acc_reply_frame_hdl,
8388             &irPhysDisk->EnclosureHandle);
8389         slot = ddi_get16(mpt->m_acc_reply_frame_hdl,
8390             &irPhysDisk->Slot);
8391         state = ddi_get32(mpt->m_acc_reply_frame_hdl,
8392             &irPhysDisk->NewValue);
8393         reason = ddi_get8(mpt->m_acc_reply_frame_hdl,
8394             &irPhysDisk->ReasonCode);

8396         NDBG20(("EVENT_IR_PHYSICAL_DISK event is received"));

8398         switch (reason) {
8399         case MPI2_EVENT_IR_PHYSDISK_RC_SETTINGS_CHANGED:
8400             mptsas_log(mpt, CE_NOTE,
8401                 " PhysDiskNum %d with DevHandle 0x%x in slot %d "
8402                 "for enclosure with handle 0x%x is now in hot "
8403                 "spare pool %d",
8404                 physdisknum, devhandle, slot, enchandle,
8405                 (state >> 16) & 0xff);
8406             break;

8408         case MPI2_EVENT_IR_PHYSDISK_RC_STATUS_FLAGS_CHANGED:
8409             status = state;
8410             mptsas_log(mpt, CE_NOTE,
8411                 " PhysDiskNum %d with DevHandle 0x%x in slot %d "
8412                 "for enclosure with handle 0x%x is now "
8413                 "%s%s%s%s\n", physdisknum, devhandle, slot,
8414                 enchandle,
8415                 status & MPI2_PHYSDISK0_STATUS_FLAG_INACTIVE_VOLUME
8416                 ? ", inactive" : ", active",
8417                 status & MPI2_PHYSDISK0_STATUS_FLAG_OUT_OF_SYNC
8418                 ? ", out of sync" : "",
8419                 status & MPI2_PHYSDISK0_STATUS_FLAG_QUIESCED
8420                 ? ", quiesced" : "",
8421                 status &
8422                 MPI2_PHYSDISK0_STATUS_FLAG_WRITE_CACHE_ENABLED
8423                 ? ", write cache enabled" : "",
8424                 status & MPI2_PHYSDISK0_STATUS_FLAG_OCE_TARGET
8425                 ? ", capacity expansion target" : "");
8426             break;

8428         case MPI2_EVENT_IR_PHYSDISK_RC_STATE_CHANGED:
8429             mptsas_log(mpt, CE_NOTE,
8430                 " PhysDiskNum %d with DevHandle 0x%x in slot %d "
8431                 "for enclosure with handle 0x%x is now %s\n",
8432                 physdisknum, devhandle, slot, enchandle,
8433                 state == MPI2_RAID_PD_STATE_OPTIMAL
8434                 ? "optimal" :
8435                 state == MPI2_RAID_PD_STATE_REBUILDING
8436                 ? "rebuilding" :
8437                 state == MPI2_RAID_PD_STATE_DEGRADED
8438                 ? "degraded" :
8439                 state == MPI2_RAID_PD_STATE_HOT_SPARE
8440                 ? "a hot spare" :
8441                 state == MPI2_RAID_PD_STATE_ONLINE
8442                 ? "online" :
8443                 state == MPI2_RAID_PD_STATE_OFFLINE

```

```

8444         ? "offline" :
8445         state == MPI2_RAID_PD_STATE_NOT_COMPATIBLE
8446         ? "not compatible" :
8447         state == MPI2_RAID_PD_STATE_NOT_CONFIGURED
8448         ? "not configured" :
8449         "state unknown");
8450         break;
8451     }
8452     break;
8453 }
8454 default:
8455     NDBG20(("mptsas3%d: unknown event %x received",
8456     mpt->m_instance, event));
8457     break;
8458 }
8459
8460 /*
8461  * Return the reply frame to the free queue.
8462  */
8463 ddi_put32(mpt->m_acc_free_queue_hdl,
8464 &((uint32_t *) (void *) mpt->m_free_queue)[mpt->m_free_index], rfm);
8465 (void) ddi_dma_sync(mpt->m_dma_free_queue_hdl, 0, 0,
8466     DDI_DMA_SYNC_FORDEV);
8467 if (++mpt->m_free_index == mpt->m_free_queue_depth) {
8468     mpt->m_free_index = 0;
8469 }
8470 ddi_put32(mpt->m_datap, &mpt->m_reg->ReplyFreeHostIndex,
8471     mpt->m_free_index);
8472 mutex_exit(&mpt->m_mutex);
8473 }
8474
8475 /*
8476  * invoked from timeout() to restart qfull cmds with throttle == 0
8477  */
8478 static void
8479 mptsas_restart_cmd(void *arg)
8480 {
8481     mptsas_t      *mpt = arg;
8482     mptsas_target_t *ptgt = NULL;
8483
8484     mutex_enter(&mpt->m_mutex);
8485
8486     mpt->m_restart_cmd_timeid = 0;
8487
8488     for (ptgt = rehash_first(mpt->m_targets); ptgt != NULL;
8489         ptgt = rehash_next(mpt->m_targets, ptgt)) {
8490         mutex_enter(&ptgt->m_t_mutex);
8491         if (ptgt->m_reset_delay == 0) {
8492             if (ptgt->m_t_throttle == QFULL_THROTTLE) {
8493                 mptsas_set_throttle(mpt, ptgt,
8494                     MAX_THROTTLE);
8495             }
8496         }
8497         mutex_exit(&ptgt->m_t_mutex);
8498     }
8499     mptsas_restart_hba(mpt);
8500     mutex_exit(&mpt->m_mutex);
8501 }
8502
8503 /*
8504  * Assume some checks have been done prior to calling this
8505  * function so we don't need to consider taking the m_mutex.
8506  */
8507 static void
8508 mptsas_remove_cmd_nontx(mptsas_t *mpt, mptsas_cmd_t *cmd)
8509 {

```

```

8510     int          slot;
8511     mptsas_slots_t *slots = mpt->m_active;
8512     mptsas_target_t *ptgt = cmd->cmd_tgt_addr;
8513
8514     ASSERT(cmd != NULL);
8515     ASSERT(cmd->cmd_queued == FALSE);
8516     ASSERT((cmd->cmd_flags & CFLAG_CMDIOC) == 0);
8517
8518     slot = cmd->cmd_slot;
8519
8520     /*
8521      * remove the cmd.
8522      */
8523     if (cmd == slots->m_slot[slot]) {
8524         NDBG31(("mptsas_remove_cmd_nontx: removing cmd=0x%p, flags "
8525         "0x%x", (void *) cmd, cmd->cmd_flags));
8526         slots->m_slot[slot] = NULL;
8527         ASSERT(mpt->m_ncmds != 0);
8528         atomic_dec_32(&mpt->m_ncmds);
8529         ASSERT(mpt->m_rep_post_queues[cmd->cmd_rpqidx].rpq_ncmds != 0);
8530         atomic_dec_32(
8531             &mpt->m_rep_post_queues[cmd->cmd_rpqidx].rpq_ncmds);
8532
8533         /*
8534          * Decrement per target ncmds, we know this is not an
8535          * IOC cmd and it therefore has a target associated with it.
8536          */
8537         mutex_enter(&ptgt->m_t_mutex);
8538         ASSERT(ptgt->m_t_ncmds != 0);
8539         ptgt->m_t_ncmds--;
8540
8541         /*
8542          * reset throttle if we just ran an untagged command
8543          * to a tagged target
8544          */
8545         if ((ptgt->m_t_ncmds == 0) &&
8546             ((cmd->cmd_pkt_flags & FLAG_TAGMASK) == 0)) {
8547             mptsas_set_throttle(mpt, ptgt, MAX_THROTTLE);
8548         }
8549
8550         /*
8551          * Remove this command from the active queue.
8552          */
8553         if (cmd->cmd_active_expiration != 0) {
8554             TAILQ_REMOVE(&ptgt->m_active_cmdq, cmd,
8555                 cmd_active_link);
8556             cmd->cmd_active_expiration = 0;
8557         }
8558         mutex_exit(&ptgt->m_t_mutex);
8559     }
8560
8561     ASSERT(cmd != slots->m_slot[cmd->cmd_slot]);
8562 }
8563
8564 void
8565 mptsas_remove_cmd(mptsas_t *mpt, mptsas_cmd_t *cmd)
8566 {
8567     int          slot;
8568     mptsas_slots_t *slots = mpt->m_active;
8569     mptsas_target_t *ptgt = cmd->cmd_tgt_addr;
8570
8571     ASSERT(cmd != NULL);
8572     ASSERT(cmd->cmd_queued == FALSE);
8573
8574     /*
8575      * Task Management cmds are removed in their own routines. Also,

```

```

8576     * we don't want to modify timeout based on TM cmds.
8577     */
8578     if (cmd->cmd_flags & CFLAG_TM_CMD) {
8579         return;
8580     }

8582     slot = cmd->cmd_slot;

8584     /*
8585     * remove the cmd.
8586     */
8587     if (cmd == slots->m_slot[slot]) {
8588         NDBG31(("mptsas_remove_cmd: removing cmd=0x%p, flags 0x%x",
8589             (void *)cmd, cmd->cmd_flags));
8590         slots->m_slot[slot] = NULL;
8591         ASSERT(mpt->m_ncmds != 0);
8592         atomic_dec_32(&mpt->m_ncmds);
8593         ASSERT(mpt->m_rep_post_queues[cmd->cmd_rpqidx].rpq_ncmds != 0);
8594         atomic_dec_32(
8595             &mpt->m_rep_post_queues[cmd->cmd_rpqidx].rpq_ncmds);

8597         /*
8598         * only decrement per target ncmds if command
8599         * has a target associated with it.
8600         */
8601         if ((cmd->cmd_flags & CFLAG_CMDIOC) == 0) {
8602             mutex_enter(&ptgt->m_t_mutex);
8603             ASSERT(ptgt->m_t_ncmds != 0);
8604             ptgt->m_t_ncmds--;

8606             /*
8607             * reset throttle if we just ran an untagged command
8608             * to a tagged target
8609             */
8610             if ((ptgt->m_t_ncmds == 0) &&
8611                 ((cmd->cmd_pkt_flags & FLAG_TAGMASK) == 0)) {
8612                 mptsas_set_throttle(mpt, ptgt, MAX_THROTTLE);
8613             }

8615             /*
8616             * Remove this command from the active queue.
8617             */
8618             if (cmd->cmd_active_expiration != 0) {
8619                 TAILQ_REMOVE(&ptgt->m_active_cmdq, cmd,
8620                     cmd_active_link);
8621                 cmd->cmd_active_expiration = 0;
8622             }
8623             mutex_exit(&ptgt->m_t_mutex);
8624         }

8626     }

8628     /*
8629     * This is all we need to do for ioc commands.
8630     */
8631     if (cmd->cmd_flags & CFLAG_CMDIOC) {
8632         mptsas_return_to_pool(mpt, cmd);
8633         return;
8634     }

8636     ASSERT(cmd != slots->m_slot[cmd->cmd_slot]);
8637 }

8639 /*
8640 * accept all cmds on the tx_waitq if any and then
8641 * start a fresh request from the top of the device queue.

```

```

8642     *
8643     * since there are always cmds queued on the tx_waitq, and rare cmds on
8644     * the instance waitq, so this function should not be invoked in the ISR,
8645     * the mptsas_restart_waitq() is invoked in the ISR instead. otherwise, the
8646     * burden belongs to the IO dispatch CPUs is moved the interrupt CPU.
8647     */
8648     static void
8649     mptsas_restart_hba(mptsas_t *mpt)
8650     {
8651         ASSERT(mutex_owned(&mpt->m_mutex));

8653         mptsas_accept_tx_waitqs(mpt);
8654         mptsas_restart_waitq(mpt);
8655     }

8657     /*
8658     * start a fresh request from the top of the device queue
8659     */
8660     static void
8661     mptsas_restart_waitq(mptsas_t *mpt)
8662     {
8663         mptsas_cmd_t *cmd, *next_cmd;
8664         mptsas_target_t *ptgt = NULL;

8666         NDBG1(("mptsas_restart_waitq: mpt=0x%p", (void *)mpt));

8668         ASSERT(mutex_owned(&mpt->m_mutex));

8670         /*
8671         * If there is a reset delay, don't start any cmds. Otherwise, start
8672         * as many cmds as possible.
8673         * Since SMID 0 is reserved and the TM slot is reserved, the actual max
8674         * commands is m_max_requests - 2.
8675         */
8676         cmd = mpt->m_waitq;

8678         while (cmd != NULL) {
8679             next_cmd = cmd->cmd_linkp;
8680             if (cmd->cmd_flags & CFLAG_PASSTHRU) {
8681                 if (mptsas_save_cmd(mpt, cmd) == TRUE) {
8682                     /*
8683                     * passthru command get slot need
8684                     * set CFLAG_PREPARED.
8685                     */
8686                     cmd->cmd_flags |= CFLAG_PREPARED;
8687                     mptsas_waitq_delete(mpt, cmd);
8688                     mptsas_start_passthru(mpt, cmd);
8689                 }
8690                 cmd = next_cmd;
8691                 continue;
8692             }
8693             if (cmd->cmd_flags & CFLAG_CONFIG) {
8694                 if (mptsas_save_cmd(mpt, cmd) == TRUE) {
8695                     /*
8696                     * Send the config page request and delete it
8697                     * from the waitq.
8698                     */
8699                     cmd->cmd_flags |= CFLAG_PREPARED;
8700                     mptsas_waitq_delete(mpt, cmd);
8701                     mptsas_start_config_page_access(mpt, cmd);
8702                 }
8703                 cmd = next_cmd;
8704                 continue;
8705             }
8706             if (cmd->cmd_flags & CFLAG_FW_DIAG) {
8707                 if (mptsas_save_cmd(mpt, cmd) == TRUE) {

```

```

8708         /*
8709          * Send the FW Diag request and delete if from
8710          * the waitq.
8711          */
8712         cmd->cmd_flags |= CFLAG_PREPARED;
8713         mptsas_waitq_delete(mpt, cmd);
8714         mptsas_start_diag(mpt, cmd);
8715     }
8716     cmd = next_cmd;
8717     continue;
8718 }

8720 ptgt = cmd->cmd_tgt_addr;
8721 if (ptgt) {
8722     mutex_enter(&ptgt->m_t_mutex);
8723     if ((ptgt->m_t_throttle == DRAIN_THROTTLE) &&
8724         (ptgt->m_t_ncmds == 0)) {
8725         mptsas_set_throttle(mpt, ptgt, MAX_THROTTLE);
8726     }
8727     if ((mpt->m_ncmds <= (mpt->m_max_requests - 2)) &&
8728         (ptgt->m_reset_delay == 0) &&
8729         (ptgt->m_t_ncmds < ptgt->m_t_throttle)) {
8730         mutex_exit(&ptgt->m_t_mutex);

8732         if (mptsas_save_cmd(mpt, cmd) == TRUE) {
8733             mptsas_waitq_delete(mpt, cmd);
8734             mutex_exit(&mpt->m_mutex);
8735             (void) mptsas_start_cmd(mpt, cmd);
8736             mutex_enter(&mpt->m_mutex);
8737             cmd = mpt->m_waitq;
8738             continue;
8739         }
8740     } else {
8741         mutex_exit(&ptgt->m_t_mutex);
8742     }
8743 }
8744 cmd = next_cmd;
8745 }
8746 }

8748 /*
8749 * Cmds are queued if scsi_start() doesn't get the m_mutex lock(no wait)
8750 * or if the decision has been made to always do that. Setting
8751 * mptsas_allow_txq_jumping to zero will allow higher performance on
8752 * a heavily loaded system as there is less disruption to the flow here.
8753 * There are 2 threads that handle one queue each. The idea is that
8754 * they take it in turn to grab the m_mutex to run the mptsas_accept_pkt()
8755 * function and then drop it while the cmd is started in mptsas_start_cmd().
8756 */
8757 static void
8758 mptsas_tx_waitq_thread(mptsas_thread_arg_t *arg)
8759 {
8760     mptsas_t *mpt = arg->mpt;
8761     mptsas_tx_waitqueue_t *txwq = &mpt->m_tx_waitq[arg->t];

8763     mutex_enter(&txwq->txwq_mutex);
8764     while (txwq->txwq_active) {
8765         mptsas_drain_tx_waitq(mpt, txwq);
8766         if (txwq->txwq_wdrain) {
8767             cv_signal(&txwq->txwq_drain_cv);
8768         }
8769         cv_wait(&txwq->txwq_cv, &txwq->txwq_mutex);
8770     }
8771     mutex_exit(&txwq->txwq_mutex);
8772     mutex_enter(&mpt->m_qthread_mutex);
8773     mpt->m_txwq_thread_n--;

```

```

8774     cv_broadcast(&mpt->m_qthread_cv);
8775     mutex_exit(&mpt->m_qthread_mutex);
8776 }

8778 /*
8779 * Set the draining flag, disconnect the list and process one at a time
8780 * so that the cmds are sent in order.
8781 */
8782 static void
8783 mptsas_drain_tx_waitq(mptsas_t *mpt, mptsas_tx_waitqueue_t *txwq)
8784 {
8785     mptsas_cmd_t *cmd, *ncmd;
8786     int rval, start;
8787 #ifdef MPTSAS_DEBUG
8788     uint32_t qlen;
8789 #endif

8791     txwq->txwq_draining = TRUE;
8792 #ifndef __lock_lint
8793     _NOTE(CONSTCOND)
8794 #endif
8795     while (TRUE) {

8797         /*
8798          * A Bus Reset could occur at any time but it will have to
8799          * wait for the main mutex before flushing the tx_waitq.
8800          * Pull all commands at once, then follow the list in order to
8801          * reduce txwq_mutex hold time. If there is a Bus Reset at
8802          * some point the commands will get to the waitq and then be
8803          * flushed.
8804          */
8805         cmd = txwq->txwq_cmdq;

8807         if (cmd == NULL) {
8808             txwq->txwq_draining = FALSE;
8809             return;
8810         }
8811         txwq->txwq_cmdq = NULL;
8812         txwq->txwq_qtail = &txwq->txwq_cmdq;
8813 #ifdef MPTSAS_DEBUG
8814         qlen = txwq->txwq_len;
8815 #endif
8816         txwq->txwq_len = 0;
8817         mutex_exit(&txwq->txwq_mutex);

8819         while (cmd) {
8820             ncmd = cmd->cmd_linkp;
8821             cmd->cmd_linkp = NULL;
8822             mutex_enter(&mpt->m_mutex);
8823             start = mptsas_accept_pkt(mpt, cmd, &rval);
8824             mutex_exit(&mpt->m_mutex);
8825             if (start) {
8826                 (void) mptsas_start_cmd(mpt, cmd);
8827             }
8828             if (rval != TRAN_ACCEPT)
8829                 cmn_err(CE_WARN,
8830                     "mpt: mptsas_drain_tx_waitq: failed "
8831                     "(rval=0x%x) to accept cmd 0x%p on queue\n",
8832                     rval, (void *)cmd);
8833             cmd = ncmd;
8834 #ifdef MPTSAS_DEBUG
8835             qlen--;
8836 #endif
8837         }
8838         ASSERT(qlen == 0);
8839         mutex_enter(&txwq->txwq_mutex);

```

```

8840     }
8841 }

8843 /*
8844  * Stop the drain threads from picking up a new list.
8845  * Optionally wait for the current list being processed to drain through.
8846  * Add to and processing the tx waitq is now on hold until unblock is called.
8847  */
8848 static void
8849 mptsas_block_tx_waitqs(mptsas_t *mpt, int wait)
8850 {
8851     int i;
8852     uint8_t wdrain = 0;
8853     mptsas_tx_waitqueue_t *txwq;

8855     ASSERT(mutex_owned(&mpt->m_mutex));

8857     if (mpt->m_txwq_thread_n == 0) {
8858         return;
8859     }

8861     /*
8862      * Turn off the use of the tx wait queues by scsi_start().
8863      * This is just a dynamic flag no need for a mutex.
8864      */
8865     mpt->m_txwq_enabled = BLOCKED;

8867     for (i = 0; i < NUM_TX_WAITQ; i++) {
8868         txwq = &mpt->m_tx_waitq[i];
8869         mutex_enter(&txwq->txwq_mutex);
8870         txwq->txwq_wdrain = TRUE;
8871         if (txwq->txwq_draining && wait)
8872             wdrain |= (1<<i);
8873         mutex_exit(&txwq->txwq_mutex);
8874     }

8876     if (wdrain) {
8877         /*
8878          * Because the threads disconnect the entire queue each time
8879          * round in order to drain to completely drain we have to
8880          * drop the main mutex otherwise the drain threads get stuck.
8881          */
8882         mutex_exit(&mpt->m_mutex);
8883         for (i = 0; i < NUM_TX_WAITQ; i++) {
8884             if (wdrain & (1<<i)) {
8885                 txwq = &mpt->m_tx_waitq[i];
8886                 mutex_enter(&txwq->txwq_mutex);
8887                 while (txwq->txwq_draining) {
8888                     cv_wait(&txwq->txwq_drain_cv,
8889                             &txwq->txwq_mutex);
8890                 }
8891                 mutex_exit(&txwq->txwq_mutex);
8892             }
8893         }
8894         mutex_enter(&mpt->m_mutex);
8895     }
8896 }

8898 static void
8899 mptsas_unblock_tx_waitqs(mptsas_t *mpt)
8900 {
8901     int i;
8902     mptsas_tx_waitqueue_t *txwq;

8904     if (mpt->m_txwq_thread_n == 0) {
8905         return;

```

```

8906     }

8908     for (i = 0; i < NUM_TX_WAITQ; i++) {
8909         txwq = &mpt->m_tx_waitq[i];
8910         mutex_enter(&txwq->txwq_mutex);
8911         txwq->txwq_wdrain = FALSE;
8912         cv_signal(&txwq->txwq_cv);
8913         mutex_exit(&txwq->txwq_mutex);
8914     }

8916     mpt->m_txwq_enabled = FALSE;
8917 }

8919 static void
8920 mptsas_accept_tx_waitqs(mptsas_t *mpt)
8921 {
8922     /*
8923      * Block with drain and unblock will leave us in a state where
8924      * we have the main mutex, there is nothing on the tx wait queues
8925      * and they are not in use until watch notices high activity again.
8926      */
8927     mptsas_block_tx_waitqs(mpt, 1);
8928     mptsas_unblock_tx_waitqs(mpt);
8929 }

8931 /*
8932  * mpt tag type lookup
8933  */
8934 static char mptsas_tag_lookup[] =
8935     {0, MSG_HEAD_QTAG, MSG_ORDERED_QTAG, 0, MSG_SIMPLE_QTAG};

8937 static int
8938 mptsas_start_cmd(mptsas_t *mpt, mptsas_cmd_t *cmd)
8939 {
8940     struct scsi_pkt *pkt = CMD2PKT(cmd);
8941     uint32_t control = 0;
8942     caddr_t mem, arsbuff;
8943     pMpi2SCSIIORequest_t io_request;
8944     ddi_dma_handle_t dma_hdl = mpt->m_dma_req_frame_hdl;
8945     ddi_acc_handle_t acc_hdl = mpt->m_acc_req_frame_hdl;
8946     mptsas_target_t *tgt = cmd->cmd_tgt_addr;
8947     uint16_t SMID, io_flags = 0, ars_size;
8948     uint8_t MSIidx;
8949     uint64_t request_desc;
8950     uint32_t ars_dmaaddr_low;
8951     mptsas_cmd_t *c;

8953     NDBG1(("mptsas_start_cmd: cmd=0x%p, flags 0x%x", (void *)cmd,
8954           cmd->cmd_flags));

8956     /*
8957      * Set SMID and increment index. Rollover to 1 instead of 0 if index
8958      * is at the max. 0 is an invalid SMID, so we call the first index 1.
8959      */
8960     SMID = cmd->cmd_slot;
8961     MSIidx = cmd->cmd_rpqidx;

8963     /*
8964      * It is possible for back to back device reset to
8965      * happen before the reset delay has expired. That's
8966      * ok, just let the device reset go out on the bus.
8967      */
8968     if ((cmd->cmd_pkt_flags & FLAG_NOINTR) == 0) {
8969         ASSERT(tgt->m_reset_delay == 0);
8970     }

```

```

8972  /*
8973  * if a non-tagged cmd is submitted to an active tagged target
8974  * then drain before submitting this cmd; SCSI-2 allows RQSENSE
8975  * to be untagged
8976  */
8977  mutex_enter(&ptgt->m_t_mutex);
8978  if (((cmd->cmd_pkt_flags & FLAG_TAGMASK) == 0) &&
8979      (ptgt->m_t_ncmds > 1) &&
8980      ((cmd->cmd_flags & CFLAG_TM_CMD) == 0) &&
8981      (*(cmd->cmd_pkt->pkt_cdbp) != SCMD_REQUEST_SENSE)) {
8982      if ((cmd->cmd_pkt_flags & FLAG_NOINTR) == 0) {
8983          NDBG23(("target=%d, untagged cmd, start draining\n",
8984              ptgt->m_devhdl));
8986          if (ptgt->m_reset_delay == 0) {
8987              mptsas_set_throttle(mpt, ptgt, DRAIN_THROTTLE);
8988          }
8989          mutex_exit(&ptgt->m_t_mutex);
8991          mutex_enter(&mpt->m_mutex);
8992          mptsas_remove_cmd(mpt, cmd);
8993          cmd->cmd_pkt_flags |= FLAG_HEAD;
8994          mptsas_waitq_add(mpt, cmd);
8995          mutex_exit(&mpt->m_mutex);
8996      } else {
8997          mutex_exit(&ptgt->m_t_mutex);
8998      }
8999      return (DDI_FAILURE);
9000  }
9002  /*
9003  * Set correct tag bits.
9004  */
9005  if (cmd->cmd_pkt_flags & FLAG_TAGMASK) {
9006      switch (mptsas_tag_lookup(((cmd->cmd_pkt_flags &
9007          FLAG_TAGMASK) >> 12))) {
9008          case MSG_SIMPLE_QTAG:
9009              control |= MPI2_SCSIIO_CONTROL_SIMPLEQ;
9010              break;
9011          case MSG_HEAD_QTAG:
9012              control |= MPI2_SCSIIO_CONTROL_HEADOFQ;
9013              break;
9014          case MSG_ORDERED_QTAG:
9015              control |= MPI2_SCSIIO_CONTROL_ORDEREDQ;
9016              break;
9017          default:
9018              mptsas_log(mpt, CE_WARN, "mpt: Invalid tag type\n");
9019              break;
9020      }
9021  } else {
9022      if (*(cmd->cmd_pkt->pkt_cdbp) != SCMD_REQUEST_SENSE) {
9023          ptgt->m_t_throttle = 1;
9024      }
9025      control |= MPI2_SCSIIO_CONTROL_SIMPLEQ;
9026  }
9028  /*
9029  * Set timeout.
9030  */
9031  cmd->cmd_active_expiration =
9032      gethrtime() + (hrtime_t)pkt->pkt_time * NANOSec;
9034  c = TAILQ_FIRST(&ptgt->m_active_cmdq);
9035  if (c == NULL ||
9036      c->cmd_active_expiration < cmd->cmd_active_expiration) {
9037      /*

```

```

9038      * Common case is that this is the last pending expiration
9039      * (or queue is empty). Insert at head of the queue.
9040      */
9041      TAILQ_INSERT_HEAD(&ptgt->m_active_cmdq, cmd, cmd_active_link);
9042  } else {
9043      /*
9044      * Queue is not empty and first element expires later than
9045      * this command. Search for element expiring sooner.
9046      */
9047      while ((c = TAILQ_NEXT(c, cmd_active_link)) != NULL) {
9048          if (c->cmd_active_expiration <
9049              cmd->cmd_active_expiration) {
9050              TAILQ_INSERT_BEFORE(c, cmd, cmd_active_link);
9051              break;
9052          }
9053      }
9054      if (c == NULL) {
9055          /*
9056          * No element found expiring sooner, append to
9057          * non-empty queue.
9058          */
9059          TAILQ_INSERT_TAIL(&ptgt->m_active_cmdq, cmd,
9060              cmd_active_link);
9061      }
9062  }
9064  mutex_exit(&ptgt->m_t_mutex);
9066  if (cmd->cmd_pkt_flags & FLAG_TLR) {
9067      control |= MPI2_SCSIIO_CONTROL_TLR_ON;
9068  }
9070  mem = mpt->m_req_frame + (mpt->m_req_frame_size * SMID);
9071  io_request = (pMpi2SCSIIORequest_t)mem;
9072  if (cmd->cmd_extrqslens != 0) {
9073      /*
9074      * Mapping of the buffer was done in mptsas_pkt_alloc_extern().
9075      * Calculate the DMA address with the same offset.
9076      */
9077      arsbuff = cmd->cmd_arq_buf;
9078      ars_size = cmd->cmd_extrqslens;
9079      ars_dmaaddrlow = (mpt->m_req_sense_dma_addr +
9080          ((uintptr_t)arsbuff - (uintptr_t)mpt->m_req_sense)) &
9081          0xfffffffffull;
9082  } else {
9083      arsbuff = mpt->m_req_sense + (mpt->m_req_sense_size * (SMID-1));
9084      cmd->cmd_arq_buf = arsbuff;
9085      ars_size = mpt->m_req_sense_size;
9086      ars_dmaaddrlow = (mpt->m_req_sense_dma_addr +
9087          (mpt->m_req_sense_size * (SMID-1))) &
9088          0xfffffffffull;
9089  }
9090  bzero(io_request, sizeof (Mpi2SCSIIORequest_t));
9091  bzero(arsbuff, ars_size);
9093  ddi_put8(acc_hdl, &io_request->SGLOffset0, offsetof
9094      (MPI2_SCSI_IO_REQUEST, SGL) / 4);
9095  mptsas_init_std_hdr(acc_hdl, io_request, ptgt->m_devhdl, Lun(cmd), 0,
9096      MPI2_FUNCTION_SCSI_IO_REQUEST);
9098  (void) ddi_rep_put8(acc_hdl, (uint8_t *)pkt->pkt_cdbp,
9099      io_request->CDB.CDB32, cmd->cmd_cdblen, DDI_DEV_AUTOINCR);
9101  io_flags = cmd->cmd_cdblen;
9102  if (mptsas3_use_fastpath &&
9103      ptgt->m_io_flags & MPI25_SAS_DEVICE0_FLAGS_ENABLED_FAST_PATH) {

```

```

9104         io_flags |= MPI25_SCSIIO_IOFLAGS_FAST_PATH;
9105         request_desc = MPI25_REQ_DESCRIPTOR_FLAGS_FAST_PATH_SCSI_IO;
9106     } else {
9107         request_desc = MPI2_REQ_DESCRIPTOR_FLAGS_SCSI_IO;
9108     }
9109     ddi_put16(acc_hdl, &io_request->IoFlags, io_flags);
9110     /*
9111      * setup the Scatter/Gather DMA list for this request
9112      */
9113     if (cmd->cmd_cookiec > 0) {
9114         mptsas_sge_setup(mpt, cmd, &control, io_request, acc_hdl);
9115     } else {
9116         ddi_put32(acc_hdl, &io_request->SGL.MpiSimple.FlagsLength,
9117             ((uint32_t)MPI2_SGE_FLAGS_LAST_ELEMENT |
9118             MPI2_SGE_FLAGS_END_OF_BUFFER |
9119             MPI2_SGE_FLAGS_SIMPLE_ELEMENT |
9120             MPI2_SGE_FLAGS_END_OF_LIST) << MPI2_SGE_FLAGS_SHIFT);
9121     }
9122
9123     /*
9124      * save ARQ information
9125      */
9126     ddi_put8(acc_hdl, &io_request->SenseBufferLength, cmd->cmd_rqslen);
9127     ddi_put32(acc_hdl, &io_request->SenseBufferLowAddress, ars_dmaaddr_low);
9128
9129     ddi_put32(acc_hdl, &io_request->Control, control);
9130
9131     NDBG31(("starting message=%d(0x%p), with cmd=0x%p",
9132         SMID, (void *)io_request, (void *)cmd));
9133
9134     (void) ddi_dma_sync(dma_hdl, 0, 0, DDI_DMA_SYNC_FORDEV);
9135
9136     /*
9137      * Build request descriptor and write it to the request desc post reg.
9138      */
9139     request_desc |= (SMID << 16) + (MSIidx << 8);
9140     request_desc |= ((uint64_t)ptgt->m_devhdl << 48);
9141     MPTSAS_START_CMD(mpt, request_desc);
9142
9143 #if 0
9144     /* Is this of any benefit here, what is it going to catch? */
9145     if ((mptsas_check_dma_handle(dma_hdl) != DDI_SUCCESS) ||
9146         (mptsas_check_acc_handle(acc_hdl) != DDI_SUCCESS)) {
9147         ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_UNAFFECTED);
9148         return (DDI_FAILURE);
9149     }
9150 #endif
9151     return (DDI_SUCCESS);
9152 }
9153
9154 /*
9155  * Select a helper thread to handle given doneq.
9156  * Note that we don't require to have the main m_mutex here, but worst case
9157  * is that we wont follow the thread rotation to the letter.
9158  * However must ensure we have the mutex that covers the source dlist when
9159  * we actually hand off.
9160  */
9161 static void
9162 mptsas_deliver_doneq_thread(mptsas_t *mpt, mptsas_done_list_t *dlist)
9163 {
9164     uint32_t t, i, j = mpt->m_doneq_next_thread;
9165     uint32_t min = 0xffffffff;
9166     mptsas_doneq_thread_list_t *item;
9167
9168     /*
9169      * No need to take individual list mutex's during the loop.

```

```

9170     * We are only reading values and the worst that will happen is that
9171     * we pick the wrong thread.
9172     */
9173     for (i = 0; i < mpt->m_doneq_thread_n; i++) {
9174         item = &mpt->m_doneq_thread_id[j];
9175
9176         /*
9177          * If the completed command on help thread[i] less than
9178          * doneq_thread_threshold, then pick the thread[j]. Otherwise
9179          * pick a thread which has least completed command.
9180          */
9181         if (item->dlist.dl_len < mpt->m_doneq_thread_threshold) {
9182             t = j;
9183             break;
9184         }
9185         if (item->dlist.dl_len < min) {
9186             min = item->dlist.dl_len;
9187             t = j;
9188         }
9189         if (++j == mpt->m_doneq_thread_n) {
9190             j = 0;
9191         }
9192     }
9193     item = &mpt->m_doneq_thread_id[t];
9194     mutex_enter(&item->mutex);
9195     mptsas_doneq_mv(dlist, item);
9196     cv_signal(&item->cv);
9197     mutex_exit(&item->mutex);
9198
9199     /*
9200      * Next time start at the next thread.
9201      * This will minimize the potential of grabbing a lock
9202      * for a thread that is busy, either on a very busy systems
9203      * or on one that is configured to do all command completion
9204      * processing through threads.
9205      */
9206     if (++t == mpt->m_doneq_thread_n) {
9207         t = 0;
9208     }
9209     mpt->m_doneq_next_thread = (uint16_t)t;
9210 }
9211
9212 /*
9213  * move one doneq to another.
9214  */
9215 static void
9216 mptsas_doneq_mv(mptsas_done_list_t *from, mptsas_doneq_thread_list_t *item)
9217 {
9218     mptsas_done_list_t *to = &item->dlist;
9219     mptsas_cmd_t *cmd;
9220
9221     if ((cmd = from->dl_q) != NULL) {
9222         *to->dl_tail = cmd;
9223         to->dl_tail = from->dl_tail;
9224         to->dl_len += from->dl_len;
9225         from->dl_q = NULL;
9226         from->dl_tail = &from->dl_q;
9227         from->dl_len = 0;
9228     }
9229 }
9230
9231 void
9232 mptsas_fma_check(mptsas_t *mpt, mptsas_cmd_t *cmd)
9233 {
9234     struct scsi_pkt *pkt = CMD2PKT(cmd);

```

```

9236 /* Check all acc and dma handles */
9237 if ((mptsas_check_acc_handle(mpt->m_datap) !=
9238     DDI_SUCCESS) ||
9239     (mptsas_check_acc_handle(mpt->m_acc_req_frame_hdl) !=
9240     DDI_SUCCESS) ||
9241     (mptsas_check_acc_handle(mpt->m_acc_req_sense_hdl) !=
9242     DDI_SUCCESS) ||
9243     (mptsas_check_acc_handle(mpt->m_acc_reply_frame_hdl) !=
9244     DDI_SUCCESS) ||
9245     (mptsas_check_acc_handle(mpt->m_acc_free_queue_hdl) !=
9246     DDI_SUCCESS) ||
9247     (mptsas_check_acc_handle(mpt->m_acc_post_queue_hdl) !=
9248     DDI_SUCCESS) ||
9249     (mptsas_check_acc_handle(mpt->m_hshk_acc_hdl) !=
9250     DDI_SUCCESS) ||
9251     (mptsas_check_acc_handle(mpt->m_config_handle) !=
9252     DDI_SUCCESS)) {
9253     ddi_fm_service_impact(mpt->m_dip,
9254         DDI_SERVICE_UNAFFECTED);
9255     ddi_fm_acc_err_clear(mpt->m_config_handle,
9256         DDI_FME_VER0);
9257     pkt->pkt_reason = CMD_TRAN_ERR;
9258     pkt->pkt_statistics = 0;
9259 }
9260 if ((mptsas_check_dma_handle(mpt->m_dma_req_frame_hdl) !=
9261     DDI_SUCCESS) ||
9262     (mptsas_check_dma_handle(mpt->m_dma_req_sense_hdl) !=
9263     DDI_SUCCESS) ||
9264     (mptsas_check_dma_handle(mpt->m_dma_reply_frame_hdl) !=
9265     DDI_SUCCESS) ||
9266     (mptsas_check_dma_handle(mpt->m_dma_free_queue_hdl) !=
9267     DDI_SUCCESS) ||
9268     (mptsas_check_dma_handle(mpt->m_dma_post_queue_hdl) !=
9269     DDI_SUCCESS) ||
9270     (mptsas_check_dma_handle(mpt->m_hshk_dma_hdl) !=
9271     DDI_SUCCESS)) {
9272     ddi_fm_service_impact(mpt->m_dip,
9273         DDI_SERVICE_UNAFFECTED);
9274     pkt->pkt_reason = CMD_TRAN_ERR;
9275     pkt->pkt_statistics = 0;
9276 }
9277 if (cmd->cmd_dmahandle &&
9278     (mptsas_check_dma_handle(cmd->cmd_dmahandle) != DDI_SUCCESS)) {
9279     ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_UNAFFECTED);
9280     pkt->pkt_reason = CMD_TRAN_ERR;
9281     pkt->pkt_statistics = 0;
9282 }
9283 if ((cmd->cmd_extra_frames &&
9284     ((mptsas_check_dma_handle(cmd->cmd_extra_frames->m_dma_hdl) !=
9285     DDI_SUCCESS) ||
9286     (mptsas_check_acc_handle(cmd->cmd_extra_frames->m_acc_hdl) !=
9287     DDI_SUCCESS)))) {
9288     ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_UNAFFECTED);
9289     pkt->pkt_reason = CMD_TRAN_ERR;
9290     pkt->pkt_statistics = 0;
9291 }
9292 }
9294 /*
9295  * These routines manipulate the queue of commands that
9296  * are waiting for their completion routines to be called.
9297  * The queue is usually in FIFO order but on an MP system
9298  * it's possible for the completion routines to get out
9299  * of order. If that's a problem you need to add a global
9300  * mutex around the code that calls the completion routine
9301  * in the interrupt handler.

```

```

9302 */
9303 static void
9304 mptsas_doneq_add(mptsas_t *mpt, mptsas_cmd_t *cmd)
9305 {
9306     struct scsi_pkt *pkt = CMD2PKT(cmd);
9307
9308     NDBG31(("mptsas_doneq_add: cmd=0x%p", (void *)cmd));
9309
9310     ASSERT((cmd->cmd_flags & CFLAG_COMPLETED) == 0);
9311     cmd->cmd_linkp = NULL;
9312     cmd->cmd_flags |= CFLAG_FINISHED;
9313     cmd->cmd_flags &= ~CFLAG_IN_TRANSPORT;
9314
9315     mptsas_fma_check(mpt, cmd);
9316
9317     /*
9318      * only add scsi pkts that have completion routines to
9319      * the doneq. no intr cmds do not have callbacks.
9320      */
9321     if (pkt && (pkt->pkt_comp)) {
9322         *mpt->m_dlist.dl_tail = cmd;
9323         mpt->m_dlist.dl_tail = &cmd->cmd_linkp;
9324         mpt->m_dlist.dl_len++;
9325     }
9326 }
9327
9328 static void
9329 mptsas_rpdoneq_add(mptsas_t *mpt, mptsas_reply_pqueue_t *rpqp,
9330     mptsas_cmd_t *cmd)
9331 {
9332     struct scsi_pkt *pkt = CMD2PKT(cmd);
9333
9334     NDBG31(("mptsas_rpdoneq_add: cmd=0x%p", (void *)cmd));
9335
9336     ASSERT((cmd->cmd_flags & CFLAG_COMPLETED) == 0);
9337     cmd->cmd_linkp = NULL;
9338     cmd->cmd_flags |= CFLAG_FINISHED;
9339     cmd->cmd_flags &= ~CFLAG_IN_TRANSPORT;
9340
9341     mptsas_fma_check(mpt, cmd);
9342
9343     /*
9344      * only add scsi pkts that have completion routines to
9345      * the doneq. no intr cmds do not have callbacks.
9346      */
9347     if (pkt && (pkt->pkt_comp)) {
9348         *rpqp->rpq_dlist.dl_tail = cmd;
9349         rpqp->rpq_dlist.dl_tail = &cmd->cmd_linkp;
9350         rpqp->rpq_dlist.dl_len++;
9351     }
9352 }
9353
9354 static mptsas_cmd_t *
9355 mptsas_doneq_thread_rm(mptsas_t *mpt, uint64_t t)
9356 {
9357     mptsas_cmd_t *cmd;
9358     mptsas_doneq_thread_list_t *item = &mpt->m_doneq_thread_id[t];
9359
9360     /* pop one off the done queue */
9361     if ((cmd = item->dlist.dl_q) != NULL) {
9362         /* if the queue is now empty fix the tail pointer */
9363         NDBG31(("mptsas_doneq_thread_rm: cmd=0x%p", (void *)cmd));
9364         if ((item->dlist.dl_q = cmd->cmd_linkp) == NULL) {
9365             item->dlist.dl_tail = &item->dlist.dl_q;
9366         }
9367         cmd->cmd_linkp = NULL;

```



```

9368         item->dlist.dl_len--;
9369     }
9370     return (cmd);
9371 }

9373 static void
9374 mptsas_doneq_empty(mptsas_t *mpt)
9375 {
9376     if (mpt->m_dlist.dl_q) {
9377         mptsas_cmd_t *cmd, *next;
9378         struct scsi_pkt *pkt;

9380         cmd = mpt->m_dlist.dl_q;
9381         mpt->m_dlist.dl_q = NULL;
9382         mpt->m_dlist.dl_tail = &mpt->m_dlist.dl_q;
9383         mpt->m_dlist.dl_len = 0;

9385         mutex_exit(&mpt->m_mutex);
9386         /*
9387          * run the completion routines of all the
9388          * completed commands
9389          */
9390         while (cmd != NULL) {
9391             next = cmd->cmd_linkp;
9392             cmd->cmd_linkp = NULL;
9393             /* run this command's completion routine */
9394             cmd->cmd_flags |= CFLAG_COMPLETED;
9395             pkt = CMD2PKT(cmd);
9396             mptsas_pkt_comp(pkt, cmd);
9397             cmd = next;
9398         }
9399         mutex_enter(&mpt->m_mutex);
9400     }
9401 }

9403 static void
9404 mptsas_rpdoneq_empty(mptsas_reply_pqueue_t *rpqp)
9405 {
9406     if (rpqp->rpq_dlist.dl_q) {
9407         mptsas_cmd_t *cmd, *next;
9408         struct scsi_pkt *pkt;

9410         cmd = rpqp->rpq_dlist.dl_q;
9411         rpqp->rpq_dlist.dl_q = NULL;
9412         rpqp->rpq_dlist.dl_tail = &rpqp->rpq_dlist.dl_q;
9413         rpqp->rpq_dlist.dl_len = 0;

9415         mutex_exit(&rpqp->rpq_mutex);
9416         /*
9417          * run the completion routines of all the
9418          * completed commands
9419          */
9420         while (cmd != NULL) {
9421             next = cmd->cmd_linkp;
9422             cmd->cmd_linkp = NULL;
9423             /* run this command's completion routine */
9424             cmd->cmd_flags |= CFLAG_COMPLETED;
9425             pkt = CMD2PKT(cmd);
9426             mptsas_pkt_comp(pkt, cmd);
9427             cmd = next;
9428         }
9429         mutex_enter(&rpqp->rpq_mutex);
9430     }
9431 }

9433 /*

```

```

9434 * These routines manipulate the target's queue of pending requests
9435 */
9436 void
9437 mptsas_waitq_add(mptsas_t *mpt, mptsas_cmd_t *cmd)
9438 {
9439     NDBG7(("mptsas_waitq_add: cmd=0x%p", (void *)cmd));
9440     mptsas_target_t *ptgt = cmd->cmd_tgt_addr;
9441     cmd->cmd_queued = TRUE;
9442     if (ptgt)
9443         ptgt->m_t_nwait++;
9444     if (cmd->cmd_pkt_flags & FLAG_HEAD) {
9445         if ((cmd->cmd_linkp = mpt->m_waitq) == NULL) {
9446             mpt->m_waitqtail = &cmd->cmd_linkp;
9447         }
9448         mpt->m_waitq = cmd;
9449     } else {
9450         cmd->cmd_linkp = NULL;
9451         *(mpt->m_waitqtail) = cmd;
9452         mpt->m_waitqtail = &cmd->cmd_linkp;
9453     }
9454 }

9456 static mptsas_cmd_t *
9457 mptsas_waitq_rm(mptsas_t *mpt)
9458 {
9459     mptsas_cmd_t *cmd;
9460     mptsas_target_t *ptgt;
9461     NDBG7(("mptsas_waitq_rm"));

9463     MPTSAS_WAITQ_RM(mpt, cmd);

9465     NDBG7(("mptsas_waitq_rm: cmd=0x%p", (void *)cmd));
9466     if (cmd) {
9467         ptgt = cmd->cmd_tgt_addr;
9468         if (ptgt) {
9469             ptgt->m_t_nwait--;
9470             ASSERT(ptgt->m_t_nwait >= 0);
9471         }
9472     }
9473     return (cmd);
9474 }

9476 /*
9477 * remove specified cmd from the middle of the wait queue.
9478 */
9479 static void
9480 mptsas_waitq_delete(mptsas_t *mpt, mptsas_cmd_t *cmd)
9481 {
9482     mptsas_cmd_t *prevp = mpt->m_waitq;
9483     mptsas_target_t *ptgt = cmd->cmd_tgt_addr;

9485     NDBG7(("mptsas_waitq_delete: mpt=0x%p cmd=0x%p",
9486         (void *)mpt, (void *)cmd));
9487     if (ptgt) {
9488         ptgt->m_t_nwait--;
9489         ASSERT(ptgt->m_t_nwait >= 0);
9490     }

9492     if (prevp == cmd) {
9493         if ((mpt->m_waitq = cmd->cmd_linkp) == NULL)
9494             mpt->m_waitqtail = &mpt->m_waitq;

9496         cmd->cmd_linkp = NULL;
9497         cmd->cmd_queued = FALSE;
9498         NDBG7(("mptsas_waitq_delete: mpt=0x%p cmd=0x%p",
9499             (void *)mpt, (void *)cmd));

```

```

9500         return;
9501     }

9503     while (prevp != NULL) {
9504         if (prevp->cmd_linkp == cmd) {
9505             if ((prevp->cmd_linkp == cmd->cmd_linkp) == NULL)
9506                 mpt->m_waitqtail = &prevp->cmd_linkp;

9508             cmd->cmd_linkp = NULL;
9509             cmd->cmd_queued = FALSE;
9510             NDBG7(("mptsas_waitq_delete: mpt=0x%p cmd=0x%p",
9511                 (void *)mpt, (void *)cmd));
9512             return;
9513         }
9514         prevp = prevp->cmd_linkp;
9515     }
9516     cmn_err(CE_PANIC, "mpt: mptsas_waitq_delete: queue botch");
9517 }

9519 /*
9520  * device and bus reset handling
9521  */
9522  * Notes:
9523  * - RESET_ALL: reset the controller
9524  * - RESET_TARGET: reset the target specified in scsi_address
9525  */
9526 static int
9527 mptsas_scsi_reset(struct scsi_address *ap, int level)
9528 {
9529     mptsas_t          *mpt = ADDR2MPT(ap);
9530     int                rval;
9531     mptsas_tgt_private_t *tgt_private;
9532     mptsas_target_t    *tgt = NULL;

9534     tgt_private = (mptsas_tgt_private_t *)ap->a_hba_tran->tran_tgt_private;
9535     ptgt = tgt_private->t_private;
9536     if (ptgt == NULL) {
9537         return (FALSE);
9538     }
9539     NDBG22(("mptsas_scsi_reset: target=%d level=%d", ptgt->m_devhdl,
9540         level));

9542     mutex_enter(&mpt->m_mutex);
9543     /*
9544      * if we are not in panic set up a reset delay for this target
9545      */
9546     if (!ddi_in_panic()) {
9547         mptsas_setup_bus_reset_delay(mpt);
9548     } else {
9549         drv_usecwait(mpt->m_scsi_reset_delay * 1000);
9550     }
9551     rval = mptsas_do_scsi_reset(mpt, ptgt->m_devhdl);
9552     mutex_exit(&mpt->m_mutex);

9554     /*
9555      * The transport layer expect to only see TRUE and
9556      * FALSE. Therefore, we will adjust the return value
9557      * if mptsas_do_scsi_reset returns FAILED.
9558      */
9559     if (rval == FAILED)
9560         rval = FALSE;
9561     return (rval);
9562 }

9564 static int
9565 mptsas_do_scsi_reset(mptsas_t *mpt, uint16_t devhdl)

```

```

9566 {
9567     int                rval = FALSE;
9568     uint8_t            config, disk;

9570     ASSERT(mutex_owned(&mpt->m_mutex));

9572     if (mptsas_debug_resets) {
9573         mptsas_log(mpt, CE_WARN, "mptsas_do_scsi_reset: target=%d",
9574             devhdl);
9575     }

9577     /*
9578      * Issue a Target Reset message to the target specified but not to a
9579      * disk making up a raid volume. Just look through the RAID config
9580      * Phys Disk list of DevHandles. If the target's DevHandle is in this
9581      * list, then don't reset this target.
9582      */
9583     for (config = 0; config < mpt->m_num_raid_configs; config++) {
9584         for (disk = 0; disk < MPTSAS_MAX_DISKS_IN_CONFIG; disk++) {
9585             if (devhdl == mpt->m_raidconfig[config].
9586                 m_physdisk_devhdl[disk]) {
9587                 return (TRUE);
9588             }
9589         }
9590     }

9592     rval = mptsas_ioc_task_management(mpt,
9593         MPI2_SCSITASKMGMT_TASKTYPE_TARGET_RESET, devhdl, 0, NULL, 0, 0);

9595     mptsas_doneq_empty(mpt);
9596     return (rval);
9597 }

9599 static int
9600 mptsas_scsi_reset_notify(struct scsi_address *ap, int flag,
9601     void (*callback)(caddr_t), caddr_t arg)
9602 {
9603     mptsas_t          *mpt = ADDR2MPT(ap);

9605     NDBG22(("mptsas_scsi_reset_notify: tgt=%d", ap->a_target));

9607     return (scsi_hba_reset_notify_setup(ap, flag, callback, arg,
9608         &mpt->m_mutex, &mpt->m_reset_notify_listf));
9609 }

9611 static int
9612 mptsas_get_name(struct scsi_device *sd, char *name, int len)
9613 {
9614     dev_info_t          *lun_dip = NULL;

9616     ASSERT(sd != NULL);
9617     ASSERT(name != NULL);
9618     lun_dip = sd->sd_dev;
9619     ASSERT(lun_dip != NULL);

9621     if (mptsas_name_child(lun_dip, name, len) == DDI_SUCCESS) {
9622         return (1);
9623     } else {
9624         return (0);
9625     }
9626 }

9628 static int
9629 mptsas_get_bus_addr(struct scsi_device *sd, char *name, int len)
9630 {
9631     return (mptsas_get_name(sd, name, len));

```

```

9632 }

9634 static void
9635 mptsas_set_throttle(mptsas_t *mpt, mptsas_target_t *ptgt, int what)
9636 {

9638     NDBG25(("mptsas_set_throttle: throttle=%x", what));

9640     /*
9641      * if the bus is draining/quiesced, no changes to the throttles
9642      * are allowed. Not allowing change of throttles during draining
9643      * limits error recovery but will reduce draining time
9644      *
9645      * all throttles should have been set to HOLD_THROTTLE
9646      */
9647     if (mpt->m_softstate & (MPTSAS_SS_QUIESCED | MPTSAS_SS_DRAINING)) {
9648         return;
9649     }

9651     if (what == HOLD_THROTTLE) {
9652         ptgt->m_t_throttle = HOLD_THROTTLE;
9653     } else if (ptgt->m_reset_delay == 0) {
9654         ptgt->m_t_throttle = what;
9655     }
9656 }

9658 static void
9659 mptsas_set_throttle_mtx(mptsas_t *mpt, mptsas_target_t *ptgt, int what)
9660 {
9661     if (mpt->m_softstate & (MPTSAS_SS_QUIESCED | MPTSAS_SS_DRAINING)) {
9662         return;
9663     }

9665     mutex_enter(&ptgt->m_t_mutex);
9666     mptsas_set_throttle(mpt, ptgt, what);
9667     mutex_exit(&ptgt->m_t_mutex);
9668 }

9670 /*
9671  * Find all commands in the tx_waitq's for target and lun (if lun not -1),
9672  * remove them from the queues and return the linked list.
9673  */
9674 static mptsas_cmd_t *
9675 mptsas_strip_targetlun_from_txwqs(mptsas_t *mpt, ushort_t target, int lun)
9676 {
9677     mptsas_cmd_t      *cmd, *clist, **tailp, **prev_tailp;
9678     mptsas_tx_waitqueue_t *txwq;
9679     int                i;

9681     clist = NULL;
9682     tailp = &clist;

9684     for (i = 0; i < NUM_TX_WAITQ; i++) {
9685         txwq = &mpt->m_tx_waitq[i];
9686         mutex_enter(&txwq->txwq_mutex);
9687         prev_tailp = &txwq->txwq_cmdq;
9688         cmd = txwq->txwq_cmdq;
9689         while (cmd != NULL) {
9690             if (Tgt(cmd) == target &&
9691                 (lun == -1 || (Lun(cmd) == lun))) {
9692                 *prev_tailp = cmd->cmd_linkp;
9693                 *tailp = cmd;
9694                 tailp = &cmd->cmd_linkp;
9695                 cmd = cmd->cmd_linkp;
9696                 *tailp = NULL;
9697             } else {

```

```

9698         prev_tailp = &cmd->cmd_linkp;
9699         cmd = cmd->cmd_linkp;
9700     }
9701     }
9702     txwq->txwq_qtail = prev_tailp;
9703     mutex_exit(&txwq->txwq_mutex);
9704 }
9705     return (clist);
9706 }

9708 /*
9709  * Clean up from a device reset.
9710  * For the case of target reset, this function clears the waitq of all
9711  * commands for a particular target. For the case of abort task set, this
9712  * function clears the waitq of all commands for a particular target/lun.
9713  */
9714 static void
9715 mptsas_flush_target(mptsas_t *mpt, ushort_t target, int lun, uint8_t tasktype)
9716 {
9717     mptsas_slots_t      *slots = mpt->m_active;
9718     mptsas_cmd_t        *cmd, *next_cmd;
9719     int                 slot;
9720     uchar_t             reason;
9721     uint_t              stat;
9722     hrttime_t           timestamp;

9724     NDBG25(("mptsas_flush_target: target=%d lun=%d", target, lun));

9726     timestamp = gethrtime();

9728     /*
9729      * Make sure the I/O Controller has flushed all cmds
9730      * that are associated with this target for a target reset
9731      * and target/lun for abort task set.
9732      * Account for TM requests, which use the last SMID.
9733      */
9734     for (slot = 0; slot <= mpt->m_active->m_n_normal; slot++) {
9735         if ((cmd = slots->m_slot[slot]) == NULL)
9736             continue;
9737         reason = CMD_RESET;
9738         stat = STAT_DEV_RESET;
9739         switch (tasktype) {
9740             case MPI2_SCSITASKMGMT_TASKTYPE_TARGET_RESET:
9741                 if (Tgt(cmd) == target) {
9742                     if (cmd->cmd_active_expiration <= timestamp) {
9743                         /*
9744                          * When timeout requested, propagate
9745                          * proper reason and statistics to
9746                          * target drivers.
9747                          */
9748                         reason = CMD_TIMEOUT;
9749                         stat |= STAT_TIMEOUT;
9750                     }
9751                     NDBG25(("mptsas_flush_target discovered non-
9752                        \"NULL cmd in slot %d, tasktype 0x%x\", slot,
9753                        tasktype));
9754                     mptsas_dump_cmd(mpt, cmd);
9755                     mptsas_remove_cmd(mpt, cmd);
9756                     mptsas_set_pkt_reason(mpt, cmd, reason, stat);
9757                     mptsas_doneq_add(mpt, cmd);
9758                 }
9759                 break;
9760             case MPI2_SCSITASKMGMT_TASKTYPE_ABORT_TASK_SET:
9761                 reason = CMD_ABORTED;
9762                 stat = STAT_ABORTED;
9763                 /*FALLTHROUGH*/

```

```

9764         case MPI2_SCSITASKMGMT_TASKTYPE_LOGICAL_UNIT_RESET:
9765             if ((Tgt(cmd) == target) && (Lun(cmd) == lun)) {
9766                 if (cmd->cmd_active_expiration <= timestamp) {
9767                     stat |= STAT_TIMEOUT;
9768                 }
9769
9770                 NDBG25(("mptsas_flush_target discovered non-
9771                     \"NULL cmd in slot %d, tasktype 0x%x\", slot,
9772                     tasktype));
9773                 mptsas_dump_cmd(mpt, cmd);
9774                 mptsas_remove_cmd(mpt, cmd);
9775                 mptsas_set_pkt_reason(mpt, cmd, reason, stat);
9776                 mptsas_doneq_add(mpt, cmd);
9777             }
9778             break;
9779         default:
9780             break;
9781     }
9782 }
9783
9784 /*
9785  * Flush the waitq and tx_waitq of this target's cmds
9786  */
9787 cmd = mpt->m_waitq;
9788
9789 reason = CMD_RESET;
9790 stat = STAT_DEV_RESET;
9791
9792 switch (tasktype) {
9793 case MPI2_SCSITASKMGMT_TASKTYPE_TARGET_RESET:
9794     while (cmd != NULL) {
9795         next_cmd = cmd->cmd_linkp;
9796         if (Tgt(cmd) == target) {
9797             mptsas_waitq_delete(mpt, cmd);
9798             mptsas_set_pkt_reason(mpt, cmd,
9799                 reason, stat);
9800             mptsas_doneq_add(mpt, cmd);
9801         }
9802         cmd = next_cmd;
9803     }
9804     cmd = mptsas_strip_targetlun_from_txxqs(mpt, target, -1);
9805     while (cmd != NULL) {
9806         next_cmd = cmd->cmd_linkp;
9807         mptsas_set_pkt_reason(mpt, cmd, reason, stat);
9808         mptsas_doneq_add(mpt, cmd);
9809         cmd = next_cmd;
9810     }
9811     break;
9812 case MPI2_SCSITASKMGMT_TASKTYPE_ABORT_TASK_SET:
9813     reason = CMD_ABORTED;
9814     stat = STAT_ABORTED;
9815     /*FALLTHROUGH*/
9816 case MPI2_SCSITASKMGMT_TASKTYPE_LOGICAL_UNIT_RESET:
9817     while (cmd != NULL) {
9818         next_cmd = cmd->cmd_linkp;
9819         if ((Tgt(cmd) == target) && (Lun(cmd) == lun)) {
9820             mptsas_waitq_delete(mpt, cmd);
9821             mptsas_set_pkt_reason(mpt, cmd,
9822                 reason, stat);
9823             mptsas_doneq_add(mpt, cmd);
9824         }
9825         cmd = next_cmd;
9826     }
9827     cmd = mptsas_strip_targetlun_from_txxqs(mpt, target, lun);
9828     while (cmd != NULL) {
9829         next_cmd = cmd->cmd_linkp;

```

```

9830         mptsas_set_pkt_reason(mpt, cmd, reason, stat);
9831         mptsas_doneq_add(mpt, cmd);
9832         cmd = next_cmd;
9833     }
9834     break;
9835     default:
9836         mptsas_log(mpt, CE_WARN, "Unknown task management type %d.",
9837             tasktype);
9838         break;
9839     }
9840 }
9841
9842 /*
9843  * Clean up hba state, abort all outstanding command and commands in waitq
9844  * reset timeout of all targets.
9845  */
9846 static void
9847 mptsas_flush_hba(mptsas_t *mpt)
9848 {
9849     mptsas_slots_t *slots = mpt->m_active;
9850     mptsas_cmd_t *cmd, *ncmd;
9851     int slot, i;
9852
9853     NDBG25(("mptsas_flush_hba"));
9854
9855     /*
9856      * The I/O Controller should have already sent back
9857      * all commands via the scsi I/O reply frame. Make
9858      * sure all commands have been flushed.
9859      * Account for TM request, which use the last SMID.
9860      */
9861     for (slot = 0; slot <= mpt->m_active->m_n_normal; slot++) {
9862         if ((cmd = slots->m_slot[slot]) == NULL)
9863             continue;
9864
9865         if (cmd->cmd_flags & CFLAG_CMDIOC) {
9866             /*
9867              * Need to make sure to tell everyone that might be
9868              * waiting on this command that it's going to fail. If
9869              * we get here, this command will never timeout because
9870              * the active command table is going to be re-allocated,
9871              * so there will be nothing to check against a time out.
9872              * Instead, mark the command as failed due to reset.
9873              */
9874             mptsas_set_pkt_reason(mpt, cmd, CMD_RESET,
9875                 STAT_BUS_RESET);
9876             if ((cmd->cmd_flags &
9877                 (CFLAG_PASSTHRU | CFLAG_CONFIG | CFLAG_FW_DIAG))) {
9878                 cmd->cmd_flags |= CFLAG_FINISHED;
9879                 cv_broadcast(&mpt->m_passthru_cv);
9880                 cv_broadcast(&mpt->m_config_cv);
9881                 cv_broadcast(&mpt->m_fw_diag_cv);
9882             }
9883             continue;
9884         }
9885
9886         NDBG25(("mptsas_flush_hba discovered non-NULL cmd in slot %d",
9887             slot));
9888         mptsas_dump_cmd(mpt, cmd);
9889
9890         mptsas_remove_cmd(mpt, cmd);
9891         mptsas_set_pkt_reason(mpt, cmd, CMD_RESET, STAT_BUS_RESET);
9892         mptsas_doneq_add(mpt, cmd);
9893     }
9894
9895     /*

```

```

9896     * Flush the waitq.
9897     */
9898     while ((cmd = mptsas_waitq_rm(mpt)) != NULL) {
9899         mptsas_set_pkt_reason(mpt, cmd, CMD_RESET, STAT_BUS_RESET);
9900         if ((cmd->cmd_flags & CFLAG_PASSTHRU) ||
9901             (cmd->cmd_flags & CFLAG_CONFIG) ||
9902             (cmd->cmd_flags & CFLAG_FW_DIAG)) {
9903             cmd->cmd_flags |= CFLAG_FINISHED;
9904             cv_broadcast(&mpt->m_passthru_cv);
9905             cv_broadcast(&mpt->m_config_cv);
9906             cv_broadcast(&mpt->m_fw_diag_cv);
9907         } else {
9908             mptsas_doneq_add(mpt, cmd);
9909         }
9910     }
9911
9912     /*
9913     * Flush the tx_waitqs
9914     */
9915     for (i = 0; i < NUM_TX_WAITQ; i++) {
9916         mutex_enter(&mpt->m_tx_waitq[i].txwq_mutex);
9917         cmd = mpt->m_tx_waitq[i].txwq_cmdq;
9918         mpt->m_tx_waitq[i].txwq_cmdq = NULL;
9919         mpt->m_tx_waitq[i].txwq_qtail = &mpt->m_tx_waitq[i].txwq_cmdq;
9920         mutex_exit(&mpt->m_tx_waitq[i].txwq_mutex);
9921         while (cmd != NULL) {
9922             ncmd = cmd->cmd_linkp;
9923             mptsas_set_pkt_reason(mpt, cmd, CMD_RESET,
9924                 STAT_BUS_RESET);
9925             mptsas_doneq_add(mpt, cmd);
9926             cmd = ncmd;
9927         }
9928     }
9929
9930     /*
9931     * Drain the taskqs prior to reallocating resources.
9932     */
9933     mutex_exit(&mpt->m_mutex);
9934     ddi_taskq_wait(mpt->m_event_taskq);
9935     ddi_taskq_wait(mpt->m_dr_taskq);
9936     mutex_enter(&mpt->m_mutex);
9937 }
9938
9939 /*
9940 * set pkt_reason and OR in pkt_statistics flag
9941 */
9942 static void
9943 mptsas_set_pkt_reason(mptsas_t *mpt, mptsas_cmd_t *cmd, uchar_t reason,
9944     uint_t stat)
9945 {
9946     #ifndef __lock_lint
9947     _NOTE(ARGUNUSED(mpt))
9948     #endif
9949
9950     NDBG25(("mptsas_set_pkt_reason: cmd=0x%p reason=%x stat=%x",
9951         (void *)cmd, reason, stat));
9952
9953     if (cmd) {
9954         if (cmd->cmd_pkt->pkt_reason == CMD_CMPLT) {
9955             cmd->cmd_pkt->pkt_reason = reason;
9956         }
9957         cmd->cmd_pkt->pkt_statistics |= stat;
9958     }
9959 }
9960
9961 static void

```

```

9962 mptsas_start_watch_reset_delay()
9963 {
9964     NDBG22(("mptsas_start_watch_reset_delay"));
9965
9966     mutex_enter(&mptsas_global_mutex);
9967     if (mptsas_reset_watch == NULL && mptsas_timeouts_enabled) {
9968         mptsas_reset_watch = timeout(mptsas_watch_reset_delay, NULL,
9969             drv_usectohz((clock_t)
9970                 MPTSAS_WATCH_RESET_DELAY_TICK * 1000));
9971         ASSERT(mptsas_reset_watch != NULL);
9972     }
9973     mutex_exit(&mptsas_global_mutex);
9974 }
9975
9976 static void
9977 mptsas_setup_bus_reset_delay(mptsas_t *mpt)
9978 {
9979     mptsas_target_t *ptgt = NULL;
9980
9981     ASSERT(MUTEX_HELD(&mpt->m_mutex));
9982
9983     NDBG22(("mptsas_setup_bus_reset_delay"));
9984     for (ptgt = rehash_first(mpt->m_targets); ptgt != NULL;
9985         ptgt = rehash_next(mpt->m_targets, ptgt)) {
9986         mutex_enter(&ptgt->m_t_mutex);
9987         mptsas_set_throttle(mpt, ptgt, HOLD_THROTTLE);
9988         ptgt->m_reset_delay = mpt->m_scsi_reset_delay;
9989         mutex_exit(&ptgt->m_t_mutex);
9990     }
9991
9992     mptsas_start_watch_reset_delay();
9993 }
9994
9995 /*
9996 * mptsas_watch_reset_delay(_subr) is invoked by timeout() and checks every
9997 * mpt instance for active reset delays
9998 */
9999 static void
10000 mptsas_watch_reset_delay(void *arg)
10001 {
10002     #ifndef __lock_lint
10003     _NOTE(ARGUNUSED(arg))
10004     #endif
10005
10006     mptsas_t      *mpt;
10007     int            not_done = 0;
10008
10009     NDBG22(("mptsas_watch_reset_delay"));
10010
10011     mutex_enter(&mptsas_global_mutex);
10012     mptsas_reset_watch = 0;
10013     mutex_exit(&mptsas_global_mutex);
10014     rw_enter(&mptsas_global_rwlock, RW_READER);
10015     for (mpt = mptsas_head; mpt != NULL; mpt = mpt->m_next) {
10016         if (mpt->m_tran == 0) {
10017             continue;
10018         }
10019         mutex_enter(&mpt->m_mutex);
10020         not_done += mptsas_watch_reset_delay_subr(mpt);
10021         mutex_exit(&mpt->m_mutex);
10022     }
10023     rw_exit(&mptsas_global_rwlock);
10024
10025     if (not_done) {
10026         mptsas_start_watch_reset_delay();
10027     }

```

```

10028 }

10030 static int
10031 mptsas_watch_reset_delay_subr(mptsas_t *mpt)
10032 {
10033     int         done = 0;
10034     int         restart = 0;
10035     mptsas_target_t *ptgt = NULL;

10037     NDBG22(("mptsas_watch_reset_delay_subr: mpt=0x%p", (void *)mpt));

10039     ASSERT(mutex_owned(&mpt->m_mutex));

10041     for (ptgt = rehash_first(mpt->m_targets); ptgt != NULL;
10042          ptgt = rehash_next(mpt->m_targets, ptgt)) {
10043         mutex_enter(&ptgt->m_t_mutex);
10044         if (ptgt->m_reset_delay != 0) {
10045             ptgt->m_reset_delay -=
10046                 MPTSAS_WATCH_RESET_DELAY_TICK;
10047             if (ptgt->m_reset_delay <= 0) {
10048                 ptgt->m_reset_delay = 0;
10049                 mptsas_set_throttle(mpt, ptgt,
10050                     MAX_THROTTLE);
10051                 restart++;
10052             } else {
10053                 done = -1;
10054             }
10055         }
10056         mutex_exit(&ptgt->m_t_mutex);
10057     }

10059     if (restart > 0) {
10060         mptsas_restart_hba(mpt);
10061     }
10062     return (done);
10063 }

10065 #ifdef MPTSAS_TEST
10066 static void
10067 mptsas_test_reset(mptsas_t *mpt, int target)
10068 {
10069     mptsas_target_t *ptgt = NULL;

10071     if (mptsas_rtest == target) {
10072         if (mptsas_do_scsi_reset(mpt, target) == TRUE) {
10073             mptsas_rtest = -1;
10074         }
10075         if (mptsas_rtest == -1) {
10076             NDBG22(("mptsas_test_reset success"));
10077         }
10078     }
10079 }
10080 #endif

10082 /*
10083  * abort handling:
10084  *
10085  * Notes:
10086  *   - if pkt is not NULL, abort just that command
10087  *   - if pkt is NULL, abort all outstanding commands for target
10088  */
10089 static int
10090 mptsas_scsi_abort(struct scsi_address *ap, struct scsi_pkt *pkt)
10091 {
10092     mptsas_t *mpt = ADDR2MPT(ap);
10093     int rval;

```

```

10094     mptsas_tgt_private_t *tgt_private;
10095     int target, lun;

10097     tgt_private = (mptsas_tgt_private_t *)ap->a_hba_tran->
10098         tran_tgt_private;
10099     ASSERT(tgt_private != NULL);
10100     target = tgt_private->t_private->m_devhdl;
10101     lun = tgt_private->t_lun;

10103     NDBG23(("mptsas_scsi_abort: target=%d.%d", target, lun));

10105     mutex_enter(&mpt->m_mutex);
10106     rval = mptsas_do_scsi_abort(mpt, target, lun, pkt);
10107     mutex_exit(&mpt->m_mutex);
10108     return (rval);
10109 }

10111 static int
10112 mptsas_do_scsi_abort(mptsas_t *mpt, int target, int lun, struct scsi_pkt *pkt)
10113 {
10114     mptsas_cmd_t *sp = NULL;
10115     mptsas_slots_t *slots = mpt->m_active;
10116     int rval = FALSE;

10118     ASSERT(mutex_owned(&mpt->m_mutex));

10120     /*
10121      * Abort the command pkt on the target/lun in ap. If pkt is
10122      * NULL, abort all outstanding commands on that target/lun.
10123      * If you can abort them, return 1, else return 0.
10124      * Each packet that's aborted should be sent back to the target
10125      * driver through the callback routine, with pkt_reason set to
10126      * CMD_ABORTED.
10127      *
10128      * abort cmd pkt on HBA hardware; clean out of outstanding
10129      * command lists, etc.
10130      */
10131     if (pkt != NULL) {
10132         /* abort the specified packet */
10133         sp = PKT2CMD(pkt);

10135         if (sp->cmd_queued) {
10136             NDBG23(("mptsas_do_scsi_abort: queued sp=0x%p aborted",
10137                 (void *)sp));
10138             mptsas_waitq_delete(mpt, sp);
10139             mptsas_set_pkt_reason(mpt, sp, CMD_ABORTED,
10140                 STAT_ABORTED);
10141             mptsas_doneq_add(mpt, sp);
10142             rval = TRUE;
10143             goto done;
10144         }

10146         /*
10147          * Have mpt firmware abort this command
10148          */

10150         if (slots->m_slot[sp->cmd_slot] != NULL) {
10151             rval = mptsas_ioc_task_management(mpt,
10152                 MPI2_SCSITASKMGMT_TASKTYPE_ABORT_TASK, target,
10153                 lun, NULL, 0, 0);

10155             /*
10156              * The transport layer expects only TRUE and FALSE.
10157              * Therefore, if mptsas_ioc_task_management returns
10158              * FAILED we will return FALSE.
10159              */

```

```

10160         if (rval == FAILED)
10161             rval = FALSE;
10162         goto done;
10163     }
10164 }
10165
10166 /*
10167  * If pkt is NULL then abort task set
10168  */
10169 rval = mptsas_ioc_task_management(mpt,
10170     MPI2_SCSITASKMGMT_TASKTYPE_ABRT_TASK_SET, target, lun, NULL, 0, 0);
10171
10172 /*
10173  * The transport layer expects only TRUE and FALSE.
10174  * Therefore, if mptsas_ioc_task_management returns
10175  * FAILED we will return FALSE.
10176  */
10177 if (rval == FAILED)
10178     rval = FALSE;
10179
10180 #ifdef MPTSAS_TEST
10181 if (rval && mptsas_test_stop) {
10182     debug_enter("mptsas_do_scsi_abort");
10183 }
10184 #endif
10185
10186 done:
10187     mptsas_doneq_empty(mpt);
10188     return (rval);
10189 }
10190
10191 /*
10192  * capability handling:
10193  * (*tran_getcap). Get the capability named, and return its value.
10194  */
10195 static int
10196 mptsas_scsi_getcap(struct scsi_address *ap, char *cap, int tgtonly)
10197 {
10198     mptsas_t      *mpt = ADDR2MPT(ap);
10199     int            ckey;
10200     int            rval = FALSE;
10201
10202     NDBG24(("mptsas_scsi_getcap: target=%d, cap=%s tgtonly=%x",
10203         ap->a_target, cap, tgtonly));
10204
10205     mutex_enter(&mpt->m_mutex);
10206
10207     if ((mptsas_scsi_capchk(cap, tgtonly, &ckey)) != TRUE) {
10208         mutex_exit(&mpt->m_mutex);
10209         return (UNDEFINED);
10210     }
10211
10212     switch (ckey) {
10213     case SCSI_CAP_DMA_MAX:
10214         rval = (int)mpt->m_msg_dma_attr.dma_attr_maxxfer;
10215         break;
10216     case SCSI_CAP_ARQ:
10217         rval = TRUE;
10218         break;
10219     case SCSI_CAP_MSG_OUT:
10220     case SCSI_CAP_PARITY:
10221     case SCSI_CAP_UNTAGGED_QING:
10222         rval = TRUE;
10223         break;
10224     case SCSI_CAP_TAGGED_QING:
10225         rval = TRUE;

```

```

10226         break;
10227     case SCSI_CAP_RESET_NOTIFICATION:
10228         rval = TRUE;
10229         break;
10230     case SCSI_CAP_LINKED_CMDS:
10231         rval = FALSE;
10232         break;
10233     case SCSI_CAP_QFULL_RETRIES:
10234         rval = ((mptsas_tgt_private_t *) (ap->a_hba_tran->
10235             tran_tgt_private))->t_private->m_qfull_retries;
10236         break;
10237     case SCSI_CAP_QFULL_RETRY_INTERVAL:
10238         rval = drv_hztousec(((mptsas_tgt_private_t *)
10239             (ap->a_hba_tran->tran_tgt_private))->
10240             t_private->m_qfull_retry_interval) / 1000;
10241         break;
10242     case SCSI_CAP_CDB_LEN:
10243         rval = CDB_GROUP4;
10244         break;
10245     case SCSI_CAP_INTERCONNECT_TYPE:
10246         rval = INTERCONNECT_SAS;
10247         break;
10248     case SCSI_CAP_TRAN_LAYER_RETRIES:
10249         if (mpt->m_ioc_capabilities &
10250             MPI2_IOCFACTS_CAPABILITY_TLR)
10251             rval = TRUE;
10252         else
10253             rval = FALSE;
10254         break;
10255     default:
10256         rval = UNDEFINED;
10257         break;
10258 }
10259
10260     NDBG24(("mptsas_scsi_getcap: %s, rval=%x", cap, rval));
10261
10262     mutex_exit(&mpt->m_mutex);
10263     return (rval);
10264 }
10265
10266 /*
10267  * (*tran_setcap). Set the capability named to the value given.
10268  */
10269 static int
10270 mptsas_scsi_setcap(struct scsi_address *ap, char *cap, int value, int tgtonly)
10271 {
10272     mptsas_t      *mpt = ADDR2MPT(ap);
10273     mptsas_target_t *tgt;
10274     int            ckey;
10275     int            rval = FALSE;
10276
10277     NDBG24(("mptsas_scsi_setcap: target=%d, cap=%s value=%x tgtonly=%x",
10278         ap->a_target, cap, value, tgtonly));
10279
10280     if (!tgtonly) {
10281         return (rval);
10282     }
10283
10284     mutex_enter(&mpt->m_mutex);
10285
10286     if ((mptsas_scsi_capchk(cap, tgtonly, &ckey)) != TRUE) {
10287         mutex_exit(&mpt->m_mutex);
10288         return (UNDEFINED);
10289     }
10290
10291     switch (ckey) {

```

```

10292     case SCSI_CAP_DMA_MAX:
10293     case SCSI_CAP_MSG_OUT:
10294     case SCSI_CAP_PARITY:
10295     case SCSI_CAP_INITIATOR_ID:
10296     case SCSI_CAP_LINKED_CMDS:
10297     case SCSI_CAP_UNTAGGED_QING:
10298     case SCSI_CAP_RESET_NOTIFICATION:
10299         /*
10300          * None of these are settable via
10301          * the capability interface.
10302          */
10303         break;
10304     case SCSI_CAP_ARQ:
10305         /*
10306          * We cannot turn off arq so return false if asked to
10307          */
10308         if (value) {
10309             rval = TRUE;
10310         } else {
10311             rval = FALSE;
10312         }
10313         break;
10314     case SCSI_CAP_TAGGED_QING:
10315         ptgt = ((mptsas_tgt_private_t *)
10316             (ap->a_hba_tran->tran_tgt_private))->t_private;
10317         mptsas_set_throttle_mtx(mpt, ptgt, MAX_THROTTLE);
10318         rval = TRUE;
10319         break;
10320     case SCSI_CAP_QFULL_RETRIES:
10321         ((mptsas_tgt_private_t *) (ap->a_hba_tran->tran_tgt_private))->
10322             t_private->m_qfull_retries = (uchar_t) value;
10323         rval = TRUE;
10324         break;
10325     case SCSI_CAP_QFULL_RETRY_INTERVAL:
10326         ((mptsas_tgt_private_t *) (ap->a_hba_tran->tran_tgt_private))->
10327             t_private->m_qfull_retry_interval =
10328             drv_usectoh(value * 1000);
10329         rval = TRUE;
10330         break;
10331     default:
10332         rval = UNDEFINED;
10333         break;
10334     }
10335     mutex_exit(&mpt->m_mutex);
10336     return (rval);
10337 }

10339 /*
10340  * Utility routine for mptsas_ifsetcap/ifgetcap
10341  */
10342 /* ARGSUSED */
10343 static int
10344 mptsas_scsi_capchk(char *cap, int tgtonly, int *cidxp)
10345 {
10346     NDBG24(("mptsas_scsi_capchk: cap=%s", cap));

10348     if (!cap)
10349         return (FALSE);

10351     *cidxp = scsi_hba_lookup_capstr(cap);
10352     return (TRUE);
10353 }

10355 static int
10356 mptsas_alloc_active_slots(mptsas_t *mpt, int flag)
10357 {

```

```

10358     mptsas_slots_t *old_active = mpt->m_active;
10359     mptsas_slots_t *new_active;
10360     size_t size;

10362     /*
10363      * if there are active commands, then we cannot
10364      * change size of active slots array.
10365      */
10366     ASSERT(mpt->m_ncmds == 0);

10368     size = MPTSAS_SLOTS_SIZE(mpt);
10369     new_active = kmem_zalloc(size, flag);
10370     if (new_active == NULL) {
10371         NDBG1(("new active alloc failed"));
10372         return (-1);
10373     }
10374     /*
10375      * Since SMID 0 is reserved and the TM slot is reserved, the
10376      * number of slots that can be used at any one time is
10377      * m_max_requests - 2.
10378      */
10379     new_active->m_n_normal = (mpt->m_max_requests - 2);
10380     new_active->m_size = size;
10381     new_active->m_rotor = 1;
10382     if (old_active)
10383         mptsas_free_active_slots(mpt);
10384     mpt->m_active = new_active;

10386     return (0);
10387 }

10389 static void
10390 mptsas_free_active_slots(mptsas_t *mpt)
10391 {
10392     mptsas_slots_t *active = mpt->m_active;
10393     size_t size;

10395     if (active == NULL)
10396         return;
10397     size = active->m_size;
10398     kmem_free(active, size);
10399     mpt->m_active = NULL;
10400 }

10402 /*
10403  * Error logging, printing, and debug print routines.
10404  */
10405 static char *mptsas_label = "mpt_sas3";

10407 /* PRINTFLIKE3 */
10408 void
10409 mptsas_log(mptsas_t *mpt, int level, char *fmt, ...)
10410 {
10411     dev_info_t *dev;
10412     va_list ap;

10414     if (mpt) {
10415         dev = mpt->m_dip;
10416     } else {
10417         dev = 0;
10418     }

10420     mutex_enter(&mptsas_log_mutex);

10422     va_start(ap, fmt);
10423     (void) vsprintf(mptsas_log_buf, fmt, ap);

```



```

10424     va_end(ap);

10426     if (level == CE_CONT) {
10427         scsi_log(dev, mptsas_label, level, "%s\n", mptsas_log_buf);
10428     } else {
10429         scsi_log(dev, mptsas_label, level, "%s", mptsas_log_buf);
10430     }

10432     mutex_exit(&mptsas_log_mutex);
10433 }

10435 #ifdef MPTSAS_DEBUG
10436 /*
10437  * Use a circular buffer to log messages to private memory.
10438  * No mutexes, so there is the opportunity for this to miss lines.
10439  * But it's fast and does not hold up the proceedings too much.
10440  */
10441 static char mptsas_dbglog_bufs[32][256];
10442 static uint32_t mptsas_dbglog_idx = 1;

10444 /*PRINTFLIKE1*/
10445 void
10446 mptsas_debug_log(char *fmt, ...)
10447 {
10448     va_list      ap;
10449     uint32_t      idx;

10451     if (!mptsas_dbglog_idx) {
10452         return;
10453     }
10454     idx = (mptsas_dbglog_idx++) & 0x1f;

10456     va_start(ap, fmt);
10457     (void) vsnprintf(mptsas_dbglog_bufs[idx],
10458         sizeof (mptsas_dbglog_bufs[0]), fmt, ap);
10459     va_end(ap);
10460 }

10462 /*PRINTFLIKE1*/
10463 void
10464 mptsas_printf(char *fmt, ...)
10465 {
10466     dev_info_t    *dev = 0;
10467     va_list      ap;

10469     mutex_enter(&mptsas_log_mutex);

10471     va_start(ap, fmt);
10472     (void) vsprintf(mptsas_log_buf, fmt, ap);
10473     va_end(ap);

10475 #ifdef PROM_PRINTF
10476     prom_printf("%s:\t%s\n", mptsas_label, mptsas_log_buf);
10477 #else
10478     scsi_log(dev, mptsas_label, CE_CONT, "!%s\n", mptsas_log_buf);
10479 #endif
10480     mutex_exit(&mptsas_log_mutex);
10481 }
10482 #endif

10484 /*
10485  * timeout handling
10486  */
10487 static void
10488 mptsas_watch(void *arg)
10489 {

```

```

10490 #ifndef __lock_lint
10491     _NOTE(ARGUNUSED(arg))
10492 #endif

10494     mptsas_t      *mpt;
10495     uint32_t      doorbell;

10497     NDBG30(("mptsas_watch"));

10499     rw_enter(&mptsas_global_rwlock, RW_READER);
10500     for (mpt = mptsas_head; mpt != (mptsas_t *)NULL; mpt = mpt->m_next) {

10502         mutex_enter(&mpt->m_mutex);

10504         /* Skip device if not powered on */
10505         if (mpt->m_options & MPTSAS_OPT_PM) {
10506             if (mpt->m_power_level == PM_LEVEL_D0) {
10507                 (void) pm_busy_component(mpt->m_dip, 0);
10508                 mpt->m_busy = 1;
10509             } else {
10510                 mutex_exit(&mpt->m_mutex);
10511                 continue;
10512             }
10513         }

10515         /*
10516          * Check if controller is in a FAULT state. If so, reset it.
10517          */
10518         doorbell = ddi_get32(mpt->m_datap, &mpt->m_reg->Doorbell);
10519         if ((doorbell & MPI2_IOC_STATE_MASK) == MPI2_IOC_STATE_FAULT) {
10520             doorbell &= MPI2_DOORBELL_DATA_MASK;
10521             mptsas_log(mpt, CE_WARN, "MPT Firmware Fault, "
10522                 "code: %04x", doorbell);
10523             mpt->m_softstate &= ~MPTSAS_SS_MSG_UNIT_RESET;
10524             if ((mptsas_restart_ioc(mpt)) == DDI_FAILURE) {
10525                 mptsas_log(mpt, CE_WARN, "Reset failed"
10526                     "after fault was detected");
10527             }
10528         }

10530         /*
10531          * For now, always call mptsas_watchsubr.
10532          */
10533         mptsas_watchsubr(mpt);

10535         if (mpt->m_options & MPTSAS_OPT_PM) {
10536             mpt->m_busy = 0;
10537             (void) pm_idle_component(mpt->m_dip, 0);
10538         }

10540         mutex_exit(&mpt->m_mutex);
10541     }
10542     rw_exit(&mptsas_global_rwlock);

10544     mutex_enter(&mptsas_global_mutex);
10545     if (mptsas_timeouts_enabled)
10546         mptsas_timeout_id = timeout(mptsas_watch, NULL, mptsas_tick);
10547     mutex_exit(&mptsas_global_mutex);
10548 }

10550 int mptsas_monitor_for_txwqs = 1;
10551 static void
10552 mptsas_watchsubr(mptsas_t *mpt)
10553 {
10554     int      i;
10555     mptsas_cmd_t *cmd;

```

```

10556     mptsas_target_t *ptgt = NULL;
10557     hrttime_t        timestamp = gethrtime();
10558     boolean_t        restart_hba = B_FALSE;

10560     ASSERT(MUTEX_HELD(&mpt->m_mutex));

10562     NDBG30(("mptsas_watchsubr: mpt=0x%p, ncmts %d, nstarted %d",
10563         (void *)mpt, mpt->m_ncmts, mpt->m_nstarted));

10565     mpt->m_lncstarted = mpt->m_nstarted;
10566     if (mpt->m_txwq_thread_n != 0 && mpt->m_txwq_enabled != BLOCKED &&
10567         mptsas_monitor_for_txwqs) {
10568         i = mpt->m_nstarted/mptsas_scsi_watchdog_tick;
10569         if (i > mpt->m_txwq_thread_threshold) {
10570             mpt->m_txwq_enabled = TRUE;
10571         } else if (i < (mpt->m_txwq_thread_threshold>>1)) {
10572             mpt->m_txwq_enabled = FALSE;
10573         }
10574     }
10575     mpt->m_nstarted = 0;

10577     /*
10578      * Check for commands stuck in active slot
10579      * Account for TM requests, which use the last SMID.
10580      */
10581     for (i = 0; i <= mpt->m_active->m_n_normal; i++) {
10582         if ((cmd = mpt->m_active->m_slot[i]) != NULL) {
10583             if (cmd->cmd_active_expiration <= timestamp) {
10584                 if ((cmd->cmd_flags & CFLAG_CMDIOC) == 0) {
10585                     /*
10586                      * There seems to be a command stuck
10587                      * in the active slot. Drain throttle.
10588                      */
10589                     ptgt = cmd->cmd_tgt_addr;
10590                     mptsas_set_throttle_mtx(mpt, ptgt,
10591                         DRAIN_THROTTLE);
10592                 } else if (cmd->cmd_flags &
10593                     (CFLAG_PASSTHRU | CFLAG_CONFIG |
10594                     CFLAG_FW_DIAG)) {
10595                     /*
10596                      * passthrough command timeout
10597                      */
10598                     cmd->cmd_flags |= (CFLAG_FINISHED |
10599                         CFLAG_TIMEOUT);
10600                     cv_broadcast(&mpt->m_passthru_cv);
10601                     cv_broadcast(&mpt->m_config_cv);
10602                     cv_broadcast(&mpt->m_fw_diag_cv);
10603                 }
10604             }
10605         }
10606     }

10608     for (ptgt = rehash_first(mpt->m_targets); ptgt != NULL;
10609         ptgt = rehash_next(mpt->m_targets, ptgt)) {
10610         mutex_enter(&ptgt->m_t_mutex);
10611         /*
10612          * If we were draining due to a qfull condition,
10613          * go back to full throttle.
10614          */
10615         if ((ptgt->m_t_throttle < MAX_THROTTLE) &&
10616             (ptgt->m_t_throttle > HOLD_THROTTLE) &&
10617             (ptgt->m_t_ncmts < ptgt->m_t_throttle)) {
10618             mptsas_set_throttle(mpt, ptgt, MAX_THROTTLE);
10619             restart_hba = B_TRUE;
10620         }

```

```

10622         cmd = TAILQ_LAST(&ptgt->m_active_cmdq, mptsas_active_cmdq);
10623         if (cmd != NULL) {
10624             if (cmd->cmd_active_expiration <= timestamp) {
10625                 /*
10626                  * Earliest command timeout expired.
10627                  * Drain throttle.
10628                  */
10629                 mptsas_set_throttle(mpt, ptgt, DRAIN_THROTTLE);

10631                 /*
10632                  * Check for remaining commands.
10633                  */
10634                 cmd = TAILQ_FIRST(&ptgt->m_active_cmdq);
10635                 if (cmd->cmd_active_expiration > timestamp) {
10636                     /*
10637                      * Wait for remaining commands to
10638                      * complete or time out.
10639                      */
10640                     NDBG23(("command timed out, "
10641                         "pending drain"));
10642                 } else {
10643                     mutex_exit(&ptgt->m_t_mutex);

10645                     /*
10646                      * All command timeouts expired.
10647                      */
10648                     mptsas_log(mpt, CE_NOTE,
10649                         "Timeout of %d seconds "
10650                         "expired with %d commands on "
10651                         "target %d lun %d.",
10652                         cmd->cmd_pkt->pkt_time,
10653                         ptgt->m_t_ncmts,
10654                         ptgt->m_devhdl, Lun(cmd));

10656                     mptsas_cmd_timeout(mpt, ptgt);
10657                     continue;
10658                 }
10659             } else if (cmd->cmd_active_expiration <= timestamp +
10660                 (hrtime_t)mptsas_scsi_watchdog_tick * NANOSEC) {
10661                 NDBG23(("pending timeout"));
10662                 mptsas_set_throttle(mpt, ptgt, DRAIN_THROTTLE);
10663             }
10664         }
10665         mutex_exit(&ptgt->m_t_mutex);
10666     }
10667     if (restart_hba == B_TRUE) {
10668         mptsas_restart_hba(mpt);
10669     }
10670 }

10672 /*
10673  * timeout recovery
10674  */
10675 static void
10676 mptsas_cmd_timeout(mptsas_t *mpt, mptsas_target_t *ptgt)
10677 {
10678     uint16_t    devhdl;
10679     uint64_t    sas_wnn;
10680     uint8_t     phy;
10681     char        wwn_str[MPTSAS_WWN_STRLEN];

10683     devhdl = ptgt->m_devhdl;
10684     sas_wnn = ptgt->m_addr.mta_wnn;
10685     phy = ptgt->m_phnum;
10686     if (sas_wnn == 0) {
10687         (void) sprintf(wwn_str, "p%x", phy);

```

```

10688     } else {
10689         (void) sprintf(wnn_str, "w%016"PRIx64, sas_wnn);
10690     }

10692     NDBG29(("mptsas_cmd_timeout: target=%d", devhdl));
10693     mptsas_log(mpt, CE_WARN, "Disconnected command timeout for "
10694         "target %d %s, enclosure %u .", devhdl, wnn_str,
10695         ptgt->m_enclosure);

10697     /*
10698      * Abort all outstanding commands on the device.
10699      */
10700     NDBG29(("mptsas_cmd_timeout: device reset"));
10701     if (mptsas_do_scsi_reset(mpt, devhdl) != TRUE) {
10702         mptsas_log(mpt, CE_WARN, "Target %d reset for command timeout "
10703             "recovery failed!", devhdl);
10704     }
10705 }

10707 /*
10708  * Device / Hotplug control
10709  */
10710 static int
10711 mptsas_scsi_quiesce(dev_info_t *dip)
10712 {
10713     mptsas_t      *mpt;
10714     scsi_hba_tran_t *tran;

10716     tran = ddi_get_driver_private(dip);
10717     if (tran == NULL || (mpt = TRAN2MPT(tran)) == NULL)
10718         return (-1);

10720     return (mptsas_quiesce_bus(mpt));
10721 }

10723 static int
10724 mptsas_scsi_unquiesce(dev_info_t *dip)
10725 {
10726     mptsas_t      *mpt;
10727     scsi_hba_tran_t *tran;

10729     tran = ddi_get_driver_private(dip);
10730     if (tran == NULL || (mpt = TRAN2MPT(tran)) == NULL)
10731         return (-1);

10733     return (mptsas_unquiesce_bus(mpt));
10734 }

10736 static int
10737 mptsas_quiesce_bus(mptsas_t *mpt)
10738 {
10739     mptsas_target_t *ptgt = NULL;

10741     NDBG28(("mptsas_quiesce_bus"));
10742     mutex_enter(&mpt->m_mutex);

10744     /* Set all the throttles to zero */
10745     for (ptgt = rehash_first(mpt->m_targets); ptgt != NULL;
10746         ptgt = rehash_next(mpt->m_targets, ptgt)) {
10747         mptsas_set_throttle_mtx(mpt, ptgt, HOLD_THROTTLE);
10748     }

10750     /* If there are any outstanding commands in the queue */
10751     if (mpt->m_ncmds) {
10752         mpt->m_softstate |= MPTSAS_SS_DRAINING;
10753         mpt->m_quiesce_timeid = timeout(mptsas_ncmds_checkdrain,

```

```

10754         mpt, (MPTSAS_QUIESCE_TIMEOUT * drv_usectohz(1000000)));
10755         if (cv_wait_sig(&mpt->m_cv, &mpt->m_mutex) == 0) {
10756             /*
10757              * Quiesce has been interrupted
10758              */
10759             mpt->m_softstate &= ~MPTSAS_SS_DRAINING;
10760             for (ptgt = rehash_first(mpt->m_targets); ptgt != NULL;
10761                 ptgt = rehash_next(mpt->m_targets, ptgt)) {
10762                 mptsas_set_throttle_mtx(mpt, ptgt,
10763                     MAX_THROTTLE);
10764             }
10765             mptsas_restart_hba(mpt);
10766             if (mpt->m_quiesce_timeid != 0) {
10767                 timeout_id_t tid = mpt->m_quiesce_timeid;
10768                 mpt->m_quiesce_timeid = 0;
10769                 mutex_exit(&mpt->m_mutex);
10770                 (void) untimedout(tid);
10771                 return (-1);
10772             }
10773             mutex_exit(&mpt->m_mutex);
10774             return (-1);
10775         } else {
10776             /* Bus has been quiesced */
10777             ASSERT(mpt->m_quiesce_timeid == 0);
10778             mpt->m_softstate &= ~MPTSAS_SS_DRAINING;
10779             mpt->m_softstate |= MPTSAS_SS_QUIESCED;
10780             mutex_exit(&mpt->m_mutex);
10781             return (0);
10782         }
10783     }
10784     /* Bus was not busy - QUIESCED */
10785     mutex_exit(&mpt->m_mutex);

10787     return (0);
10788 }

10790 static int
10791 mptsas_unquiesce_bus(mptsas_t *mpt)
10792 {
10793     mptsas_target_t *ptgt = NULL;

10795     NDBG28(("mptsas_unquiesce_bus"));
10796     mutex_enter(&mpt->m_mutex);
10797     mpt->m_softstate &= ~MPTSAS_SS_QUIESCED;
10798     for (ptgt = rehash_first(mpt->m_targets); ptgt != NULL;
10799         ptgt = rehash_next(mpt->m_targets, ptgt)) {
10800         mptsas_set_throttle_mtx(mpt, ptgt, MAX_THROTTLE);
10801     }
10802     mptsas_restart_hba(mpt);
10803     mutex_exit(&mpt->m_mutex);
10804     return (0);
10805 }

10807 static void
10808 mptsas_ncmds_checkdrain(void *arg)
10809 {
10810     mptsas_t      *mpt = arg;
10811     mptsas_target_t *ptgt = NULL;

10813     mutex_enter(&mpt->m_mutex);
10814     if (mpt->m_softstate & MPTSAS_SS_DRAINING) {
10815         mpt->m_quiesce_timeid = 0;
10816         if (mpt->m_ncmds == 0) {
10817             /* Command queue has been drained */
10818             cv_signal(&mpt->m_cv);
10819         } else {

```

```

10820      /*
10821       * The throttle may have been reset because
10822       * of a SCSI bus reset
10823       */
10824      for (ptgt = rehash_first(mpt->m_targets); ptgt != NULL;
10825           ptgt = rehash_next(mpt->m_targets, ptgt)) {
10826          mptsas_set_throttle_mtx(mpt, ptgt,
10827                                HOLD_THROTTLE);
10828      }

10830      mpt->m_quiesce_timeid = timeout(mptsas_ncmds_checkdrain,
10831                                     mpt, (MPTSAS_QUIESCE_TIMEOUT *
10832                                             drv_usectoh(1000000)));
10833      }
10834  }
10835  mutex_exit(&mpt->m_mutex);
10836  }

10838  /*ARGSUSED*/
10839  static void
10840  mptsas_dump_cmd(mptsas_t *mpt, mptsas_cmd_t *cmd)
10841  {
10842      int i;
10843      uint8_t *cp = (uchar_t *)cmd->cmd_pkt->pkt_cdbp;
10844      char buf[128];

10846      buf[0] = '\0';
10847      NDBG25(("?Cmd (0x%p) dump for Target %d Lun %d:\n", (void *)cmd,
10848            Tgt(cmd), Lun(cmd)));
10849      (void) sprintf(&buf[0], "\tcdb=[");
10850      for (i = 0; i < (int)cmd->cmd_cdblen; i++) {
10851          (void) sprintf(&buf[strlen(buf)], " 0x%x", *cp++);
10852      }
10853      (void) sprintf(&buf[strlen(buf)], " ]");
10854      NDBG25(("?%s\n", buf));
10855      NDBG25(("?pkt_flags=0x%x pkt_statistics=0x%x pkt_state=0x%x\n",
10856            cmd->cmd_pkt->pkt_flags, cmd->cmd_pkt->pkt_statistics,
10857            cmd->cmd_pkt->pkt_state));
10858      NDBG25(("?pkt_scbp=0x%x cmd_flags=0x%x\n", cmd->cmd_pkt->pkt_scbp ?
10859            *(cmd->cmd_pkt->pkt_scbp) : 0, cmd->cmd_flags));
10860  }

10862  static void
10863  mptsas_passthru_sge(ddi_acc_handle_t acc_hdl, mptsas_pt_request_t *pt,
10864                     pMpi2SGESimple64_t sgep)
10865  {
10866      uint32_t sge_flags;
10867      uint32_t data_size, dataout_size;
10868      ddi_dma_cookie_t data_cookie;
10869      ddi_dma_cookie_t dataout_cookie;

10871      data_size = pt->data_size;
10872      dataout_size = pt->dataout_size;
10873      data_cookie = pt->data_cookie;
10874      dataout_cookie = pt->dataout_cookie;

10876      if (dataout_size) {
10877          sge_flags = dataout_size |
10878              ((uint32_t)(MPI2_SGE_FLAGS_SIMPLE_ELEMENT |
10879                        MPI2_SGE_FLAGS_END_OF_BUFFER |
10880                        MPI2_SGE_FLAGS_HOST_TO_IOC |
10881                        MPI2_SGE_FLAGS_64_BIT_ADDRESSING) <<
10882              MPI2_SGE_FLAGS_SHIFT);
10883          ddi_put32(acc_hdl, &sgep->FlagsLength, sge_flags);
10884          ddi_put32(acc_hdl, &sgep->Address.Low,
10885                    (uint32_t)(dataout_cookie.dmac_laddress & 0xfffffffffull));

```

```

10886          ddi_put32(acc_hdl, &sgep->Address.High,
10887                    (uint32_t)(dataout_cookie.dmac_laddress >> 32));
10888          sgep++;
10889      }
10890      sge_flags = data_size;
10891      sge_flags |= ((uint32_t)(MPI2_SGE_FLAGS_SIMPLE_ELEMENT |
10892                        MPI2_SGE_FLAGS_LAST_ELEMENT |
10893                        MPI2_SGE_FLAGS_END_OF_BUFFER |
10894                        MPI2_SGE_FLAGS_END_OF_LIST |
10895                        MPI2_SGE_FLAGS_64_BIT_ADDRESSING) <<
10896                  MPI2_SGE_FLAGS_SHIFT);
10897      if (pt->direction == MPTSAS_PASS_THRU_DIRECTION_WRITE) {
10898          sge_flags |= ((uint32_t)(MPI2_SGE_FLAGS_HOST_TO_IOC) <<
10899                      MPI2_SGE_FLAGS_SHIFT);
10900      } else {
10901          sge_flags |= ((uint32_t)(MPI2_SGE_FLAGS_IOC_TO_HOST) <<
10902                      MPI2_SGE_FLAGS_SHIFT);
10903      }
10904      ddi_put32(acc_hdl, &sgep->FlagsLength, sge_flags);
10905      ddi_put32(acc_hdl, &sgep->Address.Low,
10906                (uint32_t)(data_cookie.dmac_laddress & 0xfffffffffull));
10907      ddi_put32(acc_hdl, &sgep->Address.High,
10908                (uint32_t)(data_cookie.dmac_laddress >> 32));
10909  }

10911  static void
10912  mptsas_passthru_ieee_sge(ddi_acc_handle_t acc_hdl, mptsas_pt_request_t *pt,
10913                           pMpi2IeeesgeSimple64_t ieeesgep)
10914  {
10915      uint8_t sge_flags;
10916      uint32_t data_size, dataout_size;
10917      ddi_dma_cookie_t data_cookie;
10918      ddi_dma_cookie_t dataout_cookie;

10920      data_size = pt->data_size;
10921      dataout_size = pt->dataout_size;
10922      data_cookie = pt->data_cookie;
10923      dataout_cookie = pt->dataout_cookie;

10925      sge_flags = (MPI2_IEEE_SGE_FLAGS_SIMPLE_ELEMENT |
10926                  MPI2_IEEE_SGE_FLAGS_SYSTEM_ADDR);
10927      if (dataout_size) {
10928          ddi_put32(acc_hdl, &ieeesgep->Length, dataout_size);
10929          ddi_put32(acc_hdl, &ieeesgep->Address.Low,
10930                    (uint32_t)(dataout_cookie.dmac_laddress &
10931                                0xfffffffffull));
10932          ddi_put32(acc_hdl, &ieeesgep->Address.High,
10933                    (uint32_t)(dataout_cookie.dmac_laddress >> 32));
10934          ddi_put8(acc_hdl, &ieeesgep->Flags, sge_flags);
10935          ieeesgep++;
10936      }
10937      sge_flags |= MPI25_IEEE_SGE_FLAGS_END_OF_LIST;
10938      ddi_put32(acc_hdl, &ieeesgep->Length, data_size);
10939      ddi_put32(acc_hdl, &ieeesgep->Address.Low,
10940                (uint32_t)(data_cookie.dmac_laddress & 0xfffffffffull));
10941      ddi_put32(acc_hdl, &ieeesgep->Address.High,
10942                (uint32_t)(data_cookie.dmac_laddress >> 32));
10943      ddi_put8(acc_hdl, &ieeesgep->Flags, sge_flags);
10944  }

10946  static void
10947  mptsas_start_passthru(mptsas_t *mpt, mptsas_cmd_t *cmd)
10948  {
10949      caddr_t memptr;
10950      pMpi2RequestHeader_t request_hdrp;
10951      struct scsi_pkt *pkt = cmd->cmd_pkt;

```

```

10952     mptsas_pt_request_t      *pt = pkt->pkt_ha_private;
10953     uint32_t                  request_size;
10954     uint64_t                  request_desc = 0;
10955     uint64_t                  sense_bufp;
10956     uint8_t                   desc_type;
10957     uint8_t                   *request, function;
10958     ddi_dma_handle_t          dma_hdl = mpt->m_dma_req_frame_hdl;
10959     ddi_acc_handle_t          acc_hdl = mpt->m_acc_req_frame_hdl;

10961     desc_type = MPI2_REQ_DESCRIPTOR_FLAGS_DEFAULT_TYPE;

10963     request = pt->request;
10964     request_size = pt->request_size;

10966     /*
10967     * Store the passthrough message in memory location
10968     * corresponding to our slot number
10969     */
10970     memp = mpt->m_req_frame + (mpt->m_req_frame_size * cmd->cmd_slot);
10971     request_hdrp = (pMPI2RequestHeader_t)memp;
10972     bzero(memp, mpt->m_req_frame_size);

10974     bcopy(request, memp, request_size);

10976     NDBG15(("mptsas_start_passthru: Func 0x%x, MsgFlags 0x%x, "
10977           "size=%d, in %d, out %d", request_hdrp->Function,
10978           request_hdrp->MsgFlags, request_size,
10979           pt->data_size, pt->dataout_size));

10981     /*
10982     * Add an SGE, even if the length is zero.
10983     */
10984     if (mpt->m_MPI25 && pt->simple == 0) {
10985         mptsas_passthru_ieee_sge(acc_hdl, pt,
10986             (pMpi2IeeeSgeSimple64_t)
10987             ((uint8_t *)request_hdrp + pt->sgl_offset));
10988     } else {
10989         mptsas_passthru_sge(acc_hdl, pt,
10990             (pMpi2SGESimple64_t)
10991             ((uint8_t *)request_hdrp + pt->sgl_offset));
10992     }

10994     function = request_hdrp->Function;
10995     if ((function == MPI2_FUNCTION_SCSI_IO_REQUEST) ||
10996         (function == MPI2_FUNCTION_RAID_SCSI_IO_PASSTHROUGH)) {
10997         pMpi2SCSIIORequest_t    scsi_io_req;

10999         NDBG15(("mptsas_start_passthru: Is SCSI IO Req"));
11000         scsi_io_req = (pMpi2SCSIIORequest_t)request_hdrp;
11001         /*
11002         * Put SGE for data and data_out buffer at the end of
11003         * scsi_io_request message header.(64 bytes in total)
11004         * Following above SGEs, the residual space will be
11005         * used by sense data.
11006         */
11007         ddi_put8(acc_hdl,
11008             &scsi_io_req->SenseBufferLength,
11009             (uint8_t)(request_size - 64));

11011         sense_bufp = (uint32_t)(mpt->m_req_frame_dma_addr +
11012             (mpt->m_req_frame_size * cmd->cmd_slot) & 0xfffffffffull);
11013         sense_bufp += 64;
11014         ddi_put32(acc_hdl,
11015             &scsi_io_req->SenseBufferLowAddress, sense_bufp);

11017         /*

```

```

11018         * Set SGLOffset0 value
11019         */
11020         ddi_put8(acc_hdl, &scsi_io_req->SGLOffset0,
11021             offsetof(MPI2_SCSI_IO_REQUEST, SGL) / 4);

11023         /*
11024         * Setup descriptor info. RAID passthrough must use the
11025         * default request descriptor which is already set, so if this
11026         * is a SCSI IO request, change the descriptor to SCSI IO.
11027         */
11028         if (function == MPI2_FUNCTION_SCSI_IO_REQUEST) {
11029             desc_type = MPI2_REQ_DESCRIPTOR_FLAGS_SCSI_IO;
11030             request_desc = (((uint64_t)ddi_get16(acc_hdl,
11031                 &scsi_io_req->DevHandle)) << 48);
11032         }
11033     }

11035     /*
11036     * We must wait till the message has been completed before
11037     * beginning the next message so we wait for this one to
11038     * finish.
11039     */
11040     (void) ddi_dma_sync(dma_hdl, 0, 0, DDI_DMA_SYNC_FORDEV);
11041     request_desc |= ((cmd->cmd_slot << 16) | desc_type);
11042     cmd->cmd_rfm = NULL;
11043     MPTSAS_START_CMD(mpt, request_desc);
11044     if ((mptsas_check_dma_handle(dma_hdl) != DDI_SUCCESS) ||
11045         (mptsas_check_acc_handle(acc_hdl) != DDI_SUCCESS)) {
11046         ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_UNAFFECTED);
11047     }
11048 }

11050 typedef void (mps_pre_f)(mptsas_t *, mptsas_pt_request_t *);
11051 static mps_pre_f    mpi_pre_ioc_facts;
11052 static mps_pre_f    mpi_pre_port_facts;
11053 static mps_pre_f    mpi_pre_fw_download;
11054 static mps_pre_f    mpi_pre_fw_25_download;
11055 static mps_pre_f    mpi_pre_fw_upload;
11056 static mps_pre_f    mpi_pre_fw_25_upload;
11057 static mps_pre_f    mpi_pre_sata_passthrough;
11058 static mps_pre_f    mpi_pre_smp_passthrough;
11059 static mps_pre_f    mpi_pre_config;
11060 static mps_pre_f    mpi_pre_sas_io_unit_control;
11061 static mps_pre_f    mpi_pre_scsi_io_req;

11063 /*
11064 * Prepare the pt for a SAS2 FW_DOWNLOAD request.
11065 */
11066 static void
11067 mpi_pre_fw_download(mptsas_t *mpt, mptsas_pt_request_t *pt)
11068 {
11069     pMpi2FWDownloadTCSGE_t tcsge;
11070     pMpi2FWDownloadRequest req;

11072     /*
11073     * If SAS3, call separate function.
11074     */
11075     if (mpt->m_MPI25) {
11076         mpi_pre_fw_25_download(mpt, pt);
11077         return;
11078     }

11080     /*
11081     * User requests should come in with the Transaction
11082     * context element where the SGL will go. Putting the
11083     * SGL after that seems to work, but don't really know

```

```

11084      * why. Other drivers tend to create an extra SGL and
11085      * refer to the TCE through that.
11086      */
11087      req = (pMpi2FWDownloadRequest)pt->request;
11088      tcsge = (pMpi2FWDownloadTCSGE_t)&req->SGL;
11089      if (tcsge->ContextSize != 0 || tcsge->DetailsLength != 12 ||
11090          tcsge->Flags != MPI2_SGE_FLAGS_TRANSACTION_ELEMENT) {
11091          mptsas_log(mpt, CE_WARN, "FW Download tce invalid!");
11092      }

11094      pt->sgl_offset = offsetof(MPI2_FW_DOWNLOAD_REQUEST, SGL) +
11095          sizeof (*tcsge);
11096      if (pt->request_size != pt->sgl_offset)
11097          NDBG15(("mpi_pre_fw_download(): Incorrect req size, "
11098              "0x%x, should be 0x%x, dataoutsz 0x%x",
11099              (int)pt->request_size, (int)pt->sgl_offset,
11100              (int)pt->dataout_size));
11101      if (pt->data_size < sizeof (MPI2_FW_DOWNLOAD_REPLY))
11102          NDBG15(("mpi_pre_fw_download(): Incorrect rep size, "
11103              "0x%x, should be 0x%x", pt->data_size,
11104              (int)sizeof (MPI2_FW_DOWNLOAD_REPLY)));
11105  }

11107 /*
11108  * Prepare the pt for a SAS3 FW_DOWNLOAD request.
11109  */
11110  static void
11111  mpi_pre_fw_25_download(mptsas_t *mpt, mptsas_pt_request_t *pt)
11112  {
11113      pMpi2FWDownloadTCSGE_t tcsge;
11114      pMpi2FWDownloadRequest req2;
11115      pMpi25FWDownloadRequest req25;

11117      /*
11118      * User requests should come in with the Transaction
11119      * context element where the SGL will go. The new firmware
11120      * Doesn't use TCE and has space in the main request for
11121      * this information. So move to the right place.
11122      */
11123      req2 = (pMpi2FWDownloadRequest)pt->request;
11124      req25 = (pMpi25FWDownloadRequest)pt->request;
11125      tcsge = (pMpi2FWDownloadTCSGE_t)&req2->SGL;
11126      if (tcsge->ContextSize != 0 || tcsge->DetailsLength != 12 ||
11127          tcsge->Flags != MPI2_SGE_FLAGS_TRANSACTION_ELEMENT) {
11128          mptsas_log(mpt, CE_WARN, "FW Download tce invalid!");
11129      }
11130      req25->ImageOffset = tcsge->ImageOffset;
11131      req25->ImageSize = tcsge->ImageSize;

11133      pt->sgl_offset = offsetof(MPI25_FW_DOWNLOAD_REQUEST, SGL);
11134      if (pt->request_size != pt->sgl_offset)
11135          NDBG15(("mpi_pre_fw_25_download(): Incorrect req size, "
11136              "0x%x, should be 0x%x, dataoutsz 0x%x",
11137              pt->request_size, pt->sgl_offset,
11138              pt->dataout_size));
11139      if (pt->data_size < sizeof (MPI25_FW_DOWNLOAD_REPLY))
11140          NDBG15(("mpi_pre_fw_25_download(): Incorrect rep size, "
11141              "0x%x, should be 0x%x", pt->data_size,
11142              (int)sizeof (MPI25_FW_UPLOAD_REPLY)));
11143  }

11145 /*
11146  * Prepare the pt for a SAS2 FW_UPLOAD request.
11147  */
11148  static void
11149  mpi_pre_fw_upload(mptsas_t *mpt, mptsas_pt_request_t *pt)

```

```

11150  {
11151      pMpi2FWUploadTCSGE_t tcsge;
11152      pMpi2FWUploadRequest_t req;

11154      /*
11155      * If SAS3, call separate function.
11156      */
11157      if (mpt->m_MPI25) {
11158          mpi_pre_fw_25_upload(mpt, pt);
11159          return;
11160      }

11162      /*
11163      * User requests should come in with the Transaction
11164      * context element where the SGL will go. Putting the
11165      * SGL after that seems to work, but don't really know
11166      * why. Other drivers tend to create an extra SGL and
11167      * refer to the TCE through that.
11168      */
11169      req = (pMpi2FWUploadRequest_t)pt->request;
11170      tcsge = (pMpi2FWUploadTCSGE_t)&req->SGL;
11171      if (tcsge->ContextSize != 0 || tcsge->DetailsLength != 12 ||
11172          tcsge->Flags != MPI2_SGE_FLAGS_TRANSACTION_ELEMENT) {
11173          mptsas_log(mpt, CE_WARN, "FW Upload tce invalid!");
11174      }

11176      pt->sgl_offset = offsetof(MPI2_FW_UPLOAD_REQUEST, SGL) +
11177          sizeof (*tcsge);
11178      if (pt->request_size != pt->sgl_offset)
11179          NDBG15(("mpi_pre_fw_upload(): Incorrect req size, "
11180              "0x%x, should be 0x%x, dataoutsz 0x%x",
11181              pt->request_size, pt->sgl_offset,
11182              pt->dataout_size));
11183      if (pt->data_size < sizeof (MPI2_FW_UPLOAD_REPLY))
11184          NDBG15(("mpi_pre_fw_upload(): Incorrect rep size, "
11185              "0x%x, should be 0x%x", pt->data_size,
11186              (int)sizeof (MPI2_FW_UPLOAD_REPLY)));
11187  }

11189 /*
11190  * Prepare the pt a SAS3 FW_UPLOAD request.
11191  */
11192  static void
11193  mpi_pre_fw_25_upload(mptsas_t *mpt, mptsas_pt_request_t *pt)
11194  {
11195      pMpi2FWUploadTCSGE_t tcsge;
11196      pMpi2FWUploadRequest_t req2;
11197      pMpi25FWUploadRequest_t req25;

11199      /*
12000      * User requests should come in with the Transaction
12001      * context element where the SGL will go. The new firmware
12002      * Doesn't use TCE and has space in the main request for
12003      * this information. So move to the right place.
12004      */
12005      req2 = (pMpi2FWUploadRequest_t)pt->request;
12006      req25 = (pMpi25FWUploadRequest_t)pt->request;
12007      tcsge = (pMpi2FWUploadTCSGE_t)&req2->SGL;
12008      if (tcsge->ContextSize != 0 || tcsge->DetailsLength != 12 ||
12009          tcsge->Flags != MPI2_SGE_FLAGS_TRANSACTION_ELEMENT) {
12010          mptsas_log(mpt, CE_WARN, "FW Upload tce invalid!");
12011      }
12012      req25->ImageOffset = tcsge->ImageOffset;
12013      req25->ImageSize = tcsge->ImageSize;

12015      pt->sgl_offset = offsetof(MPI25_FW_UPLOAD_REQUEST, SGL);

```

```

11216     if (pt->request_size != pt->sgl_offset)
11217         NDBG15(("mpi_pre_fw_25_upload(): Incorrect req size, "
11218             "0x%x, should be 0x%x, dataoutasz 0x%x",
11219             pt->request_size, pt->sgl_offset,
11220             pt->dataout_size));
11221     if (pt->data_size < sizeof (MPI2_FW_UPLOAD_REPLY))
11222         NDBG15(("mpi_pre_fw_25_upload(): Incorrect rep size, "
11223             "0x%x, should be 0x%x", pt->data_size,
11224             (int)sizeof (MPI2_FW_UPLOAD_REPLY)));
11225 }

11227 /*
11228  * Prepare the pt for an IOC_FACTS request.
11229  */
11230 static void
11231 mpi_pre_ioc_facts(mptsas_t *mpt, mptsas_pt_request_t *pt)
11232 {
11233     #ifndef __lock_lint
11234         _NOTE(ARGUNUSED(mpt))
11235     #endif
11236     if (pt->request_size != sizeof (MPI2_IOC_FACTS_REQUEST))
11237         NDBG15(("mpi_pre_ioc_facts(): Incorrect req size, "
11238             "0x%x, should be 0x%x, dataoutasz 0x%x",
11239             pt->request_size,
11240             (int)sizeof (MPI2_IOC_FACTS_REQUEST),
11241             pt->dataout_size));
11242     if (pt->data_size != sizeof (MPI2_IOC_FACTS_REPLY))
11243         NDBG15(("mpi_pre_ioc_facts(): Incorrect rep size, "
11244             "0x%x, should be 0x%x", pt->data_size,
11245             (int)sizeof (MPI2_IOC_FACTS_REPLY)));
11246     pt->sgl_offset = (uint16_t)pt->request_size;
11247 }

11249 /*
11250  * Prepare the pt for a PORT_FACTS request.
11251  */
11252 static void
11253 mpi_pre_port_facts(mptsas_t *mpt, mptsas_pt_request_t *pt)
11254 {
11255     #ifndef __lock_lint
11256         _NOTE(ARGUNUSED(mpt))
11257     #endif
11258     if (pt->request_size != sizeof (MPI2_PORT_FACTS_REQUEST))
11259         NDBG15(("mpi_pre_port_facts(): Incorrect req size, "
11260             "0x%x, should be 0x%x, dataoutasz 0x%x",
11261             pt->request_size,
11262             (int)sizeof (MPI2_PORT_FACTS_REQUEST),
11263             pt->dataout_size));
11264     if (pt->data_size != sizeof (MPI2_PORT_FACTS_REPLY))
11265         NDBG15(("mpi_pre_port_facts(): Incorrect rep size, "
11266             "0x%x, should be 0x%x", pt->data_size,
11267             (int)sizeof (MPI2_PORT_FACTS_REPLY)));
11268     pt->sgl_offset = (uint16_t)pt->request_size;
11269 }

11271 /*
11272  * Prepare pt for a SATA_PASSTHROUGH request.
11273  */
11274 static void
11275 mpi_pre_sata_passthrough(mptsas_t *mpt, mptsas_pt_request_t *pt)
11276 {
11277     #ifndef __lock_lint
11278         _NOTE(ARGUNUSED(mpt))
11279     #endif
11280     pt->sgl_offset = offsetof(MPI2_SATA_PASSTHROUGH_REQUEST, SGL);
11281     if (pt->request_size != pt->sgl_offset)

```

```

11282         NDBG15(("mpi_pre_sata_passthrough(): Incorrect req size, "
11283             "0x%x, should be 0x%x, dataoutasz 0x%x",
11284             pt->request_size, pt->sgl_offset,
11285             pt->dataout_size));
11286     if (pt->data_size != sizeof (MPI2_SATA_PASSTHROUGH_REPLY))
11287         NDBG15(("mpi_pre_sata_passthrough(): Incorrect rep size, "
11288             "0x%x, should be 0x%x", pt->data_size,
11289             (int)sizeof (MPI2_SATA_PASSTHROUGH_REPLY)));
11290 }

11292 static void
11293 mpi_pre_smp_passthrough(mptsas_t *mpt, mptsas_pt_request_t *pt)
11294 {
11295     #ifndef __lock_lint
11296         _NOTE(ARGUNUSED(mpt))
11297     #endif
11298     pt->sgl_offset = offsetof(MPI2_SMP_PASSTHROUGH_REQUEST, SGL);
11299     if (pt->request_size != pt->sgl_offset)
11300         NDBG15(("mpi_pre_smp_passthrough(): Incorrect req size, "
11301             "0x%x, should be 0x%x, dataoutasz 0x%x",
11302             pt->request_size, pt->sgl_offset,
11303             pt->dataout_size));
11304     if (pt->data_size != sizeof (MPI2_SMP_PASSTHROUGH_REPLY))
11305         NDBG15(("mpi_pre_smp_passthrough(): Incorrect rep size, "
11306             "0x%x, should be 0x%x", pt->data_size,
11307             (int)sizeof (MPI2_SMP_PASSTHROUGH_REPLY)));
11308 }

11310 /*
11311  * Prepare pt for a CONFIG request.
11312  */
11313 static void
11314 mpi_pre_config(mptsas_t *mpt, mptsas_pt_request_t *pt)
11315 {
11316     #ifndef __lock_lint
11317         _NOTE(ARGUNUSED(mpt))
11318     #endif
11319     pt->sgl_offset = offsetof(MPI2_CONFIG_REQUEST, PageBufferSGE);
11320     if (pt->request_size != pt->sgl_offset)
11321         NDBG15(("mpi_pre_config(): Incorrect req size, 0x%x, "
11322             "should be 0x%x, dataoutasz 0x%x", pt->request_size,
11323             pt->sgl_offset, pt->dataout_size));
11324     if (pt->data_size != sizeof (MPI2_CONFIG_REPLY))
11325         NDBG15(("mpi_pre_config(): Incorrect rep size, 0x%x, "
11326             "should be 0x%x", pt->data_size,
11327             (int)sizeof (MPI2_CONFIG_REPLY)));
11328     pt->simple = 1;
11329 }

11331 /*
11332  * Prepare pt for a SCSI_IO_REQ request.
11333  */
11334 static void
11335 mpi_pre_scsi_io_req(mptsas_t *mpt, mptsas_pt_request_t *pt)
11336 {
11337     #ifndef __lock_lint
11338         _NOTE(ARGUNUSED(mpt))
11339     #endif
11340     pt->sgl_offset = offsetof(MPI2 SCSI_IO_REQUEST, SGL);
11341     if (pt->request_size != pt->sgl_offset)
11342         NDBG15(("mpi_pre_config(): Incorrect req size, 0x%x, "
11343             "should be 0x%x, dataoutasz 0x%x", pt->request_size,
11344             pt->sgl_offset,
11345             pt->dataout_size));
11346     if (pt->data_size != sizeof (MPI2 SCSI_IO_REPLY))
11347         NDBG15(("mpi_pre_config(): Incorrect rep size, 0x%x, "

```

```

11348         "should be 0x%x", pt->data_size,
11349         (int)sizeof (MPI2_SCSI_IO_REPLY));
11350 }

11352 /*
11353  * Prepare the mps_command for a SAS_IO_UNIT_CONTROL request.
11354  */
11355 static void
11356 mpi_pre_sas_io_unit_control(mptsas_t *mpt, mptsas_pt_request_t *pt)
11357 {
11358     #ifndef __lock_lint
11359     _NOTE(ARGUNUSED(mpt))
11360     #endif
11361     pt->sgl_offset = (uint16_t)pt->request_size;
11362 }

11364 /*
11365  * A set of functions to prepare an mps_command for the various
11366  * supported requests.
11367  */
11368 struct mps_func {
11369     U8          Function;
11370     char        *Name;
11371     mps_pre_f   *f_pre;
11372 } mps_func_list[] = {
11373     { MPI2_FUNCTION_IOC_FACTS, "IOC_FACTS",          mpi_pre_ioc_facts },
11374     { MPI2_FUNCTION_PORT_FACTS, "PORT_FACTS",        mpi_pre_port_facts },
11375     { MPI2_FUNCTION_FW_DOWNLOAD, "FW_DOWNLOAD",      mpi_pre_fw_download },
11376     { MPI2_FUNCTION_FW_UPLOAD, "FW_UPLOAD",          mpi_pre_fw_upload },
11377     { MPI2_FUNCTION_SATA_PASSTHROUGH, "SATA_PASSTHROUGH",
11378       mpi_pre_sata_passthrough },
11379     { MPI2_FUNCTION_SMP_PASSTHROUGH, "SMP_PASSTHROUGH",
11380       mpi_pre_smp_passthrough },
11381     { MPI2_FUNCTION_SCSI_IO_REQUEST, "SCSI_IO_REQUEST",
11382       mpi_pre_scsi_io_req },
11383     { MPI2_FUNCTION_CONFIG, "CONFIG",                mpi_pre_config },
11384     { MPI2_FUNCTION_SAS_IO_UNIT_CONTROL, "SAS_IO_UNIT_CONTROL",
11385       mpi_pre_sas_io_unit_control },
11386     { 0xFF, NULL, NULL } /* list end */
11387 };

11389 static void
11390 mptsas_prep_sgl_offset(mptsas_t *mpt, mptsas_pt_request_t *pt)
11391 {
11392     pMPI2RequestHeader_t  hdr;
11393     struct mps_func       *f;

11395     hdr = (pMPI2RequestHeader_t)pt->request;

11397     for (f = mps_func_list; f->f_pre != NULL; f++) {
11398         if (hdr->Function == f->Function) {
11399             f->f_pre(mpt, pt);
11400             NDBG15(("mptsas_prep_sgl_offset: Function %s",
11401                  " sgl_offset 0x%x", f->Name,
11402                  pt->sgl_offset));
11403             return;
11404         }
11405     }
11406     NDBG15(("mptsas_prep_sgl_offset: Unknown Function 0x%02x",
11407            " returning req_size 0x%x for sgl_offset",
11408            hdr->Function, pt->request_size));
11409     pt->sgl_offset = (uint16_t)pt->request_size;
11410 }

11413 static int

```

```

11414 mptsas_do_passthru(mptsas_t *mpt, uint8_t *request, uint8_t *reply,
11415     uint8_t *data, uint32_t request_size, uint32_t reply_size,
11416     uint32_t data_size, uint8_t direction, uint8_t *dataout,
11417     uint32_t dataout_size, short timeout, int mode)
11418 {
11419     mptsas_pt_request_t      pt;
11420     mptsas_dma_alloc_state_t data_dma_state;
11421     mptsas_dma_alloc_state_t dataout_dma_state;
11422     caddr_t                  memp;
11423     mptsas_cmd_t             *cmd = NULL;
11424     struct scsi_pkt          *pkt;
11425     uint32_t                 reply_len = 0, sense_len = 0;
11426     pMPI2RequestHeader_t     request_hdrp;
11427     pMPI2RequestHeader_t     request_msg;
11428     pMPI2DefaultReply_t      reply_msg;
11429     rep_msg;
11430     int                      i, status = 0, pt_flags = 0, rv = 0;
11431     int                      rvalue;
11432     uint8_t                  function;

11434     ASSERT(mutex_owned(&mpt->m_mutex));

11436     reply_msg = (pMPI2DefaultReply_t)&rep_msg;
11437     bzero(reply_msg, sizeof (MPI2_DEFAULT_REPLY));
11438     request_msg = kmem_zalloc(request_size, KM_SLEEP);

11440     mutex_exit(&mpt->m_mutex);
11441     /*
11442      * copy in the request buffer since it could be used by
11443      * another thread when the pt request into waitq
11444      */
11445     if (ddi_copyin(request, request_msg, request_size, mode)) {
11446         mutex_enter(&mpt->m_mutex);
11447         status = EFAULT;
11448         mptsas_log(mpt, CE_WARN, "failed to copy request data");
11449         goto out;
11450     }
11451     mutex_enter(&mpt->m_mutex);

11453     function = request_msg->Function;
11454     if (function == MPI2_FUNCTION_SCSI_TASK_MGMT) {
11455         pMpi2SCSITaskManagementRequest_t task;
11456         task = (pMpi2SCSITaskManagementRequest_t)request_msg;
11457         mptsas_setup_bus_reset_delay(mpt);
11458         rv = mptsas_ioc_task_management(mpt, task->TaskType,
11459             task->DevHandle, (int)task->LUN[1], reply, reply_size,
11460             mode);

11462         if (rv != TRUE) {
11463             status = EIO;
11464             mptsas_log(mpt, CE_WARN, "task management failed");
11465         }
11466         goto out;
11467     }

11469     if (data_size != 0) {
11470         data_dma_state.size = data_size;
11471         if (mptsas_dma_alloc(mpt, &data_dma_state) != DDI_SUCCESS) {
11472             status = ENOMEM;
11473             mptsas_log(mpt, CE_WARN, "failed to alloc DMA "
11474                 "resource");
11475             goto out;
11476         }
11477         pt_flags |= MPTSAS_DATA_ALLOCATED;
11478         if (direction == MPTSAS_PASS_THRU_DIRECTION_WRITE) {
11479             mutex_exit(&mpt->m_mutex);

```



```

11480         for (i = 0; i < data_size; i++) {
11481             if (ddi_copyin(data + i, (uint8_t *)
11482                 data_dma_state.memp + i, 1, mode)) {
11483                 mutex_enter(&mpt->m_mutex);
11484                 status = EFAULT;
11485                 mptsas_log(mpt, CE_WARN, "failed to "
11486                     "copy read data");
11487                 goto out;
11488             }
11489             mutex_enter(&mpt->m_mutex);
11490         }
11491     }
11492 }
11493 else
11494     bzero(&data_dma_state, sizeof (data_dma_state));
11495
11496 if (dataout_size != 0) {
11497     dataout_dma_state.size = dataout_size;
11498     if (mptsas_dma_alloc(mpt, &dataout_dma_state) != DDI_SUCCESS) {
11499         status = ENOMEM;
11500         mptsas_log(mpt, CE_WARN, "failed to alloc DMA "
11501             "resource");
11502         goto out;
11503     }
11504     pt_flags |= MPTSAS_DATAOUT_ALLOCATED;
11505     mutex_exit(&mpt->m_mutex);
11506     for (i = 0; i < dataout_size; i++) {
11507         if (ddi_copyin(dataout + i, (uint8_t *)
11508             dataout_dma_state.memp + i, 1, mode)) {
11509             mutex_enter(&mpt->m_mutex);
11510             mptsas_log(mpt, CE_WARN, "failed to copy out"
11511                 " data");
11512             status = EFAULT;
11513             goto out;
11514         }
11515     }
11516     mutex_enter(&mpt->m_mutex);
11517 }
11518 else
11519     bzero(&dataout_dma_state, sizeof (dataout_dma_state));
11520
11521 if ((rvalue = (mptsas_request_from_pool(mpt, &cmd, &pkt))) == -1) {
11522     status = EAGAIN;
11523     mptsas_log(mpt, CE_NOTE, "event ack command pool is full");
11524     goto out;
11525 }
11526 pt_flags |= MPTSAS_REQUEST_POOL_CMD;
11527
11528 bzero((caddr_t)cmd, sizeof (*cmd));
11529 bzero((caddr_t)pkt, scsi_pkt_size());
11530 bzero((caddr_t)&pt, sizeof (pt));
11531
11532 cmd->ioc_cmd_slot = (uint32_t)(rvalue);
11533
11534 pt.request = (uint8_t *)request_msg;
11535 pt.direction = direction;
11536 pt.simple = 0;
11537 pt.request_size = request_size;
11538 pt.data_size = data_size;
11539 pt.dataout_size = dataout_size;
11540 pt.data_cookie = data_dma_state.cookie;
11541 pt.dataout_cookie = dataout_dma_state.cookie;
11542 mptsas_prep_sg1_offset(mpt, &pt);
11543
11544 /*
11545  * Form a blank cmd/pkt to store the acknowledgement message

```

```

11546     /*
11547     pkt->pkt_cdbp         = (opaque_t)&cmd->cmd_cdb[0];
11548     pkt->pkt_scbp         = (opaque_t)&cmd->cmd_scb;
11549     pkt->pkt_ha_private   = (opaque_t)&pt;
11550     pkt->pkt_flags        = FLAG_HEAD;
11551     pkt->pkt_time         = timeout;
11552     cmd->cmd_pkt          = pkt;
11553     cmd->cmd_flags        = CFLAG_CMDDIOC | CFLAG_PASSTHRU;
11554
11555     /*
11556     * Save the command in a slot
11557     */
11558     if (mptsas_save_cmd(mpt, cmd) == TRUE) {
11559         /*
11560         * Once passthru command get slot, set cmd_flags
11561         * CFLAG_PREPARED.
11562         */
11563         cmd->cmd_flags |= CFLAG_PREPARED;
11564         mptsas_start_passthru(mpt, cmd);
11565     } else {
11566         mptsas_waitq_add(mpt, cmd);
11567     }
11568
11569 while ((cmd->cmd_flags & CFLAG_FINISHED) == 0) {
11570     cv_wait(&mpt->m_passthru_cv, &mpt->m_mutex);
11571 }
11572
11573 if (cmd->cmd_flags & CFLAG_PREPARED) {
11574     memp = mpt->m_req_frame + (mpt->m_req_frame_size *
11575         cmd->cmd_slot);
11576     request_hdrp = (pMPI2RequestHeader_t)memp;
11577 }
11578
11579 if (cmd->cmd_flags & CFLAG_TIMEOUT) {
11580     status = ETIMEDOUT;
11581     mptsas_log(mpt, CE_WARN, "passthrough command timeout");
11582     pt_flags |= MPTSAS_CMD_TIMEOUT;
11583     goto out;
11584 }
11585
11586 if (cmd->cmd_rfm) {
11587     /*
11588     * cmd_rfm is zero means the command reply is a CONTEXT
11589     * reply and no PCI Write to post the free reply SMFA
11590     * because no reply message frame is used.
11591     * cmd_rfm is non-zero means the reply is a ADDRESS
11592     * reply and reply message frame is used.
11593     */
11594     pt_flags |= MPTSAS_ADDRESS_REPLY;
11595     (void) ddi_dma_sync(mpt->m_dma_reply_frame_hdl, 0, 0,
11596         DDI_DMA_SYNC_FORCPU);
11597     reply_msg = (pMPI2DefaultReply_t)
11598         (mpt->m_reply_frame + (cmd->cmd_rfm -
11599             (mpt->m_reply_frame_dma_addr&0xffffffffful)));
11600 }
11601
11602 mptsas_fma_check(mpt, cmd);
11603 if (pkt->pkt_reason == CMD_TRAN_ERR) {
11604     status = EAGAIN;
11605     mptsas_log(mpt, CE_WARN, "passthru fma error");
11606     goto out;
11607 }
11608 if (pkt->pkt_reason == CMD_RESET) {
11609     status = EAGAIN;
11610     mptsas_log(mpt, CE_WARN, "ioc reset abort passthru");
11611     goto out;

```

```

11612     }
11614     if (pkt->pkt_reason == CMD_INCOMPLETE) {
11615         status = EIO;
11616         mptsas_log(mpt, CE_WARN, "passthrough command incomplete");
11617         goto out;
11618     }
11620     mutex_exit(&mpt->m_mutex);
11621     if (cmd->cmd_flags & CFLAG_PREPARED) {
11622         function = request_hdrp->Function;
11623         if ((function == MPI2_FUNCTION_SCSI_IO_REQUEST) ||
11624             (function == MPI2_FUNCTION_RAID_SCSI_IO_PASSTHROUGH)) {
11625             reply_len = sizeof (MPI2_SCSI_IO_REPLY);
11626             sense_len = reply_size - reply_len;
11627         } else {
11628             reply_len = reply_size;
11629             sense_len = 0;
11630         }
11632         for (i = 0; i < reply_len; i++) {
11633             if (ddi_copyout((uint8_t *)reply_msg + i, reply + i, 1,
11634                 mode)) {
11635                 mutex_enter(&mpt->m_mutex);
11636                 status = EFAULT;
11637                 mptsas_log(mpt, CE_WARN, "failed to copy out "
11638                     "reply data");
11639                 goto out;
11640             }
11642             for (i = 0; i < sense_len; i++) {
11643                 if (ddi_copyout((uint8_t *)request_hdrp + 64 + i,
11644                     reply + reply_len + i, 1, mode)) {
11645                     mutex_enter(&mpt->m_mutex);
11646                     status = EFAULT;
11647                     mptsas_log(mpt, CE_WARN, "failed to copy out "
11648                         "sense data");
11649                     goto out;
11650                 }
11651             }
11652         }
11654         if (data_size) {
11655             if (direction != MPTSAS_PASS_THRU_DIRECTION_WRITE) {
11656                 (void) ddi_dma_sync(data_dma_state.handle, 0, 0,
11657                     DDI_DMA_SYNC_FORCPU);
11658                 for (i = 0; i < data_size; i++) {
11659                     if (ddi_copyout((uint8_t *)
11660                         data_dma_state.memp + i), data + i, 1,
11661                         mode)) {
11662                         mutex_enter(&mpt->m_mutex);
11663                         status = EFAULT;
11664                         mptsas_log(mpt, CE_WARN, "failed to "
11665                             "copy out the reply data");
11666                         goto out;
11667                     }
11668                 }
11669             }
11670         }
11671         mutex_enter(&mpt->m_mutex);
11672     out:
11673     /*
11674     * Put the reply frame back on the free queue, increment the free
11675     * index, and write the new index to the free index register. But only
11676     * if this reply is an ADDRESS reply.
11677     */

```

```

11678     if (pt_flags & MPTSAS_ADDRESS_REPLY) {
11679         ddi_put32(mpt->m_acc_free_queue_hdl,
11680             &((uint32_t *) (void *)mpt->m_free_queue)[mpt->m_free_index],
11681             cmd->cmd_rfm);
11682         (void) ddi_dma_sync(mpt->m_dma_free_queue_hdl, 0, 0,
11683             DDI_DMA_SYNC_FORDEV);
11684         if (++mpt->m_free_index == mpt->m_free_queue_depth) {
11685             mpt->m_free_index = 0;
11686         }
11687         ddi_put32(mpt->m_datap, &mpt->m_reg->ReplyFreeHostIndex,
11688             mpt->m_free_index);
11689     }
11690     if (cmd && (cmd->cmd_flags & CFLAG_PREPARED)) {
11691         mptsas_remove_cmd(mpt, cmd);
11692         pt_flags &= (~MPTSAS_REQUEST_POOL_CMD);
11693     }
11694     if (pt_flags & MPTSAS_REQUEST_POOL_CMD)
11695         mptsas_return_to_pool(mpt, cmd);
11696     if (pt_flags & MPTSAS_DATA_ALLOCATED) {
11697         if (mptsas_check_dma_handle(data_dma_state.handle) !=
11698             DDI_SUCCESS) {
11699             ddi_fm_service_impact(mpt->m_dip,
11700                 DDI_SERVICE_UNAFFECTED);
11701             status = EFAULT;
11702         }
11703         mptsas_dma_free(&data_dma_state);
11704     }
11705     if (pt_flags & MPTSAS_DATAOUT_ALLOCATED) {
11706         if (mptsas_check_dma_handle(dataout_dma_state.handle) !=
11707             DDI_SUCCESS) {
11708             ddi_fm_service_impact(mpt->m_dip,
11709                 DDI_SERVICE_UNAFFECTED);
11710             status = EFAULT;
11711         }
11712         mptsas_dma_free(&dataout_dma_state);
11713     }
11714     if (pt_flags & MPTSAS_CMD_TIMEOUT) {
11715         if ((mptsas_restart_ioc(mpt)) == DDI_FAILURE) {
11716             mptsas_log(mpt, CE_WARN, "mptsas_restart_ioc failed");
11717         }
11718     }
11719     if (request_msg)
11720         kmem_free(request_msg, request_size);
11722     return (status);
11723 }
11725 static int
11726 mptsas_pass_thru(mptsas_t *mpt, mptsas_pass_thru_t *data, int mode)
11727 {
11728     /*
11729     * If timeout is 0, set timeout to default of 60 seconds.
11730     */
11731     if (data->Timeout == 0) {
11732         data->Timeout = MPTSAS_PASS_THRU_TIME_DEFAULT;
11733     }
11735     if (((data->DataSize == 0) &&
11736         (data->DataDirection == MPTSAS_PASS_THRU_DIRECTION_NONE)) ||
11737         ((data->DataSize != 0) &&
11738         ((data->DataDirection == MPTSAS_PASS_THRU_DIRECTION_READ) ||
11739         (data->DataDirection == MPTSAS_PASS_THRU_DIRECTION_WRITE) ||
11740         (data->DataDirection == MPTSAS_PASS_THRU_DIRECTION_BOTH) &&
11741         (data->DataOutSize != 0)))) {
11742         if (data->DataDirection == MPTSAS_PASS_THRU_DIRECTION_BOTH) {
11743             data->DataDirection = MPTSAS_PASS_THRU_DIRECTION_READ;

```

```

11744         } else {
11745             data->DataOutSize = 0;
11746         }
11747         /*
11748          * Send passthru request messages
11749          */
11750         return (mptsas_do_passthru(mpt,
11751             (uint8_t *)((uintptr_t)data->PtrRequest),
11752             (uint8_t *)((uintptr_t)data->PtrReply),
11753             (uint8_t *)((uintptr_t)data->PtrData),
11754             data->RequestSize, data->ReplySize,
11755             data->DataSize, (uint8_t)data->DataDirection,
11756             (uint8_t *)((uintptr_t)data->PtrDataOut),
11757             data->DataOutSize, data->Timeout, mode));
11758     } else {
11759         return (EINVAL);
11760     }
11761 }

11763 static uint8_t
11764 mptsas_get_fw_diag_buffer_number(mptsas_t *mpt, uint32_t unique_id)
11765 {
11766     uint8_t index;

11768     for (index = 0; index < MPI2_DIAG_BUF_TYPE_COUNT; index++) {
11769         if (mpt->m_fw_diag_buffer_list[index].unique_id == unique_id) {
11770             return (index);
11771         }
11772     }

11774     return (MPTSAS_FW_DIAGNOSTIC_UID_NOT_FOUND);
11775 }

11777 static void
11778 mptsas_start_diag(mptsas_t *mpt, mptsas_cmd_t *cmd)
11779 {
11780     pMpi2DiagBufferPostRequest_t    pDiag_post_msg;
11781     pMpi2DiagReleaseRequest_t        pDiag_release_msg;
11782     struct scsi_pkt                  *pkt = cmd->cmd_pkt;
11783     mptsas_diag_request_t            *diag = pkt->pkt_ha_private;
11784     uint32_t                          i;
11785     uint64_t                          request_desc;

11787     ASSERT(mutex_owned(&mpt->m_mutex));

11789     /*
11790      * Form the diag message depending on the post or release function.
11791      */
11792     if (diag->function == MPI2_FUNCTION_DIAG_BUFFER_POST) {
11793         pDiag_post_msg = (pMpi2DiagBufferPostRequest_t) {
11794             (mpt->m_req_frame + (mpt->m_req_frame_size *
11795                 cmd->cmd_slot));
11796             bzero(pDiag_post_msg, mpt->m_req_frame_size);
11797             ddi_put8(mpt->m_acc_req_frame_hdl, &pDiag_post_msg->Function,
11798                 diag->function);
11799             ddi_put8(mpt->m_acc_req_frame_hdl, &pDiag_post_msg->BufferType,
11800                 diag->pBuffer->buffer_type);
11801             ddi_put8(mpt->m_acc_req_frame_hdl, &pDiag_post_msg->ExtendedType,
11802                 diag->pBuffer->extended_type);
11803             ddi_put32(mpt->m_acc_req_frame_hdl,
11804                 &pDiag_post_msg->BufferLength,
11805                 diag->pBuffer->buffer_data.size);
11806             for (i = 0; i < (sizeof (pDiag_post_msg->ProductSpecific) / 4);
11807                 i++) {
11808                 ddi_put32(mpt->m_acc_req_frame_hdl,

```

```

11810             &pDiag_post_msg->ProductSpecific[i],
11811             diag->pBuffer->product_specific[i]);
11812         }
11813         ddi_put32(mpt->m_acc_req_frame_hdl,
11814             &pDiag_post_msg->BufferAddress.Low,
11815             (uint32_t)(diag->pBuffer->buffer_data.cookie.dmac_laddress
11816                 & 0xffffffffull));
11817         ddi_put32(mpt->m_acc_req_frame_hdl,
11818             &pDiag_post_msg->BufferAddress.High,
11819             (uint32_t)(diag->pBuffer->buffer_data.cookie.dmac_laddress
11820                 >> 32));
11821     } else {
11822         pDiag_release_msg = (pMpi2DiagReleaseRequest_t)
11823             (mpt->m_req_frame + (mpt->m_req_frame_size *
11824                 cmd->cmd_slot));
11825         bzero(pDiag_release_msg, mpt->m_req_frame_size);
11826         ddi_put8(mpt->m_acc_req_frame_hdl,
11827             &pDiag_release_msg->Function, diag->function);
11828         ddi_put8(mpt->m_acc_req_frame_hdl,
11829             &pDiag_release_msg->BufferType,
11830             diag->pBuffer->buffer_type);
11831     }

11833     /*
11834      * Send the message
11835      */
11836     (void) ddi_dma_sync(mpt->m_dma_req_frame_hdl, 0, 0,
11837         DDI_DMA_SYNC_FORDEV);
11838     request_desc = (cmd->cmd_slot << 16) |
11839         MPI2_REQ_DESCRIPTOR_FLAGS_DEFAULT_TYPE;
11840     cmd->cmd_rfm = NULL;
11841     MPTSAS_START_CMD(mpt, request_desc);
11842     if ((mptsas_check_dma_handle(mpt->m_dma_req_frame_hdl) !=
11843         DDI_SUCCESS) ||
11844         (mptsas_check_acc_handle(mpt->m_acc_req_frame_hdl) !=
11845         DDI_SUCCESS)) {
11846         ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_UNAFFECTED);
11847     }
11848 }

11850 static int
11851 mptsas_post_fw_diag_buffer(mptsas_t *mpt,
11852     mptsas_fw_diagnostic_buffer_t *pBuffer, uint32_t *return_code)
11853 {
11854     mptsas_diag_request_t    diag;
11855     int                        status, slot_num, post_flags = 0;
11856     mptsas_cmd_t              *cmd = NULL;
11857     struct scsi_pkt            *pkt;
11858     pMpi2DiagBufferPostReply_t reply;
11859     uint16_t                   iocstatus;
11860     uint32_t                   iocloginfo, transfer_length;

11862     /*
11863      * If buffer is not enabled, just leave.
11864      */
11865     *return_code = MPTSAS_FW_DIAG_ERROR_POST_FAILED;
11866     if (!pBuffer->enabled) {
11867         status = DDI_FAILURE;
11868         goto out;
11869     }

11871     /*
11872      * Clear some flags initially.
11873      */
11874     pBuffer->force_release = FALSE;
11875     pBuffer->valid_data = FALSE;

```

```

11876     pBuffer->owned_by_firmware = FALSE;
11877
11878     /*
11879     * Get a cmd buffer from the cmd buffer pool
11880     */
11881     if ((slot_num = (mptsas_request_from_pool(mpt, &cmd, &pkt))) == -1) {
11882         status = DDI_FAILURE;
11883         mptsas_log(mpt, CE_NOTE, "command pool is full: Post FW Diag");
11884         goto out;
11885     }
11886     post_flags |= MPTSAS_REQUEST_POOL_CMD;
11887
11888     bzero((caddr_t)cmd, sizeof (*cmd));
11889     bzero((caddr_t)pkt, scsi_pkt_size());
11890
11891     cmd->ioc_cmd_slot = (uint32_t)(slot_num);
11892
11893     diag.pBuffer = pBuffer;
11894     diag.function = MPI2_FUNCTION_DIAG_BUFFER_POST;
11895
11896     /*
11897     * Form a blank cmd/pkt to store the acknowledgement message
11898     */
11899     pkt->pkt_ha_private = (opaque_t)&diag;
11900     pkt->pkt_flags = FLAG_HEAD;
11901     pkt->pkt_time = 60;
11902     cmd->cmd_pkt = pkt;
11903     cmd->cmd_flags = CFLAG_CMDIOC | CFLAG_FW_DIAG;
11904
11905     /*
11906     * Save the command in a slot
11907     */
11908     if (mptsas_save_cmd(mpt, cmd) == TRUE) {
11909         /*
11910         * Once passthru command get slot, set cmd_flags
11911         * CFLAG_PREPARED.
11912         */
11913         cmd->cmd_flags |= CFLAG_PREPARED;
11914         mptsas_start_diag(mpt, cmd);
11915     } else {
11916         mptsas_waitq_add(mpt, cmd);
11917     }
11918
11919     while ((cmd->cmd_flags & CFLAG_FINISHED) == 0) {
11920         cv_wait(&mpt->m_fw_diag_cv, &mpt->m_mutex);
11921     }
11922
11923     if (cmd->cmd_flags & CFLAG_TIMEOUT) {
11924         status = DDI_FAILURE;
11925         mptsas_log(mpt, CE_WARN, "Post FW Diag command timeout");
11926         goto out;
11927     }
11928
11929     /*
11930     * cmd_rfm points to the reply message if a reply was given. Check the
11931     * IOCStatus to make sure everything went OK with the FW diag request
11932     * and set buffer flags.
11933     */
11934     if (cmd->cmd_rfm) {
11935         post_flags |= MPTSAS_ADDRESS_REPLY;
11936         (void) ddi_dma_sync(mpt->m_dma_reply_frame_hdl, 0, 0,
11937             DDI_DMA_SYNC_FORCPU);
11938         reply = (pMpi2DiagBufferPostReply_t)(mpt->m_reply_frame +
11939             (cmd->cmd_rfm -
11940             (mpt->m_reply_frame_dma_addr&0xffffffff)));

```

```

11942     /*
11943     * Get the reply message data
11944     */
11945     iocstatus = ddi_get16(mpt->m_acc_reply_frame_hdl,
11946         &reply->IOCStatus);
11947     iocloginfo = ddi_get32(mpt->m_acc_reply_frame_hdl,
11948         &reply->IOCLogInfo);
11949     transfer_length = ddi_get32(mpt->m_acc_reply_frame_hdl,
11950         &reply->TransferLength);
11951
11952     /*
11953     * If post failed quit.
11954     */
11955     if (iocstatus != MPI2_IOCSTATUS_SUCCESS) {
11956         status = DDI_FAILURE;
11957         NDBG13(("post FW Diag Buffer failed: IOCStatus=0x%x, "
11958             "IOCLogInfo=0x%x, TransferLength=0x%x", iocstatus,
11959             iocloginfo, transfer_length));
11960         goto out;
11961     }
11962
11963     /*
11964     * Post was successful.
11965     */
11966     pBuffer->valid_data = TRUE;
11967     pBuffer->owned_by_firmware = TRUE;
11968     *return_code = MPTSAS_FW_DIAG_ERROR_SUCCESS;
11969     status = DDI_SUCCESS;
11970 }
11971
11972 out:
11973     /*
11974     * Put the reply frame back on the free queue, increment the free
11975     * index, and write the new index to the free index register. But only
11976     * if this reply is an ADDRESS reply.
11977     */
11978     if (post_flags & MPTSAS_ADDRESS_REPLY) {
11979         ddi_put32(mpt->m_acc_free_queue_hdl,
11980             &((uint32_t *) (void *) mpt->m_free_queue)[mpt->m_free_index],
11981             cmd->cmd_rfm);
11982         (void) ddi_dma_sync(mpt->m_dma_free_queue_hdl, 0, 0,
11983             DDI_DMA_SYNC_FORDEV);
11984         if (++mpt->m_free_index == mpt->m_free_queue_depth) {
11985             mpt->m_free_index = 0;
11986         }
11987         ddi_put32(mpt->m_datap, &mpt->m_reg->ReplyFreeHostIndex,
11988             mpt->m_free_index);
11989     }
11990     if (cmd && (cmd->cmd_flags & CFLAG_PREPARED)) {
11991         mptsas_remove_cmd(mpt, cmd);
11992         post_flags &= (~MPTSAS_REQUEST_POOL_CMD);
11993     }
11994     if (post_flags & MPTSAS_REQUEST_POOL_CMD) {
11995         mptsas_return_to_pool(mpt, cmd);
11996     }
11997
11998     return (status);
11999 }
12000
12001 static int
12002 mptsas_release_fw_diag_buffer(mptsas_t *mpt,
12003     mptsas_fw_diagnostic_buffer_t *pBuffer, uint32_t *return_code,
12004     uint32_t diag_type)
12005 {
12006     mptsas_diag_request_t diag;
12007     int status, slot_num, rel_flags = 0;

```

```

12008     mptsas_cmd_t      *cmd = NULL;
12009     struct scsi_pkt    *pkt;
12010     pMpi2DiagReleaseReply_t reply;
12011     uint16_t           iocstatus;
12012     uint32_t           iocloginfo;

12014     /*
12015      * If buffer is not enabled, just leave.
12016      */
12017     *return_code = MPTSAS_FW_DIAG_ERROR_RELEASE_FAILED;
12018     if (!pBuffer->enabled) {
12019         mptsas_log(mpt, CE_NOTE, "This buffer type is not supported "
12020             "by the IOC");
12021         status = DDI_FAILURE;
12022         goto out;
12023     }

12025     /*
12026      * Clear some flags initially.
12027      */
12028     pBuffer->force_release = FALSE;
12029     pBuffer->valid_data = FALSE;
12030     pBuffer->owned_by_firmware = FALSE;

12032     /*
12033      * Get a cmd buffer from the cmd buffer pool
12034      */
12035     if ((slot_num = (mptsas_request_from_pool(mpt, &cmd, &pkt))) == -1) {
12036         status = DDI_FAILURE;
12037         mptsas_log(mpt, CE_NOTE, "command pool is full: Release FW "
12038             "Diag");
12039         goto out;
12040     }
12041     rel_flags |= MPTSAS_REQUEST_POOL_CMD;

12043     bzero((caddr_t)cmd, sizeof (*cmd));
12044     bzero((caddr_t)pkt, scsi_pkt_size());

12046     cmd->ioc_cmd_slot = (uint32_t)(slot_num);

12048     diag.pBuffer = pBuffer;
12049     diag.function = MPI2_FUNCTION_DIAG_RELEASE;

12051     /*
12052      * Form a blank cmd/pkt to store the acknowledgement message
12053      */
12054     pkt->pkt_ha_private = (opaque_t)&diag;
12055     pkt->pkt_flags = FLAG_HEAD;
12056     pkt->pkt_time = 60;
12057     cmd->cmd_pkt = pkt;
12058     cmd->cmd_flags = CFLAG_CMDIOC | CFLAG_FW_DIAG;

12060     /*
12061      * Save the command in a slot
12062      */
12063     if (mptsas_save_cmd(mpt, cmd) == TRUE) {
12064         /*
12065          * Once passthru command get slot, set cmd_flags
12066          * CFLAG_PREPARED.
12067          */
12068         cmd->cmd_flags |= CFLAG_PREPARED;
12069         mptsas_start_diag(mpt, cmd);
12070     } else {
12071         mptsas_waitq_add(mpt, cmd);
12072     }

```

```

12074     while ((cmd->cmd_flags & CFLAG_FINISHED) == 0) {
12075         cv_wait(&mpt->m_fw_diag_cv, &mpt->m_mutex);
12076     }

12078     if (cmd->cmd_flags & CFLAG_TIMEOUT) {
12079         status = DDI_FAILURE;
12080         mptsas_log(mpt, CE_WARN, "Release FW Diag command timeout");
12081         goto out;
12082     }

12084     /*
12085      * cmd_rfm points to the reply message if a reply was given. Check the
12086      * IOCStatus to make sure everything went OK with the FW diag request
12087      * and set buffer flags.
12088      */
12089     if (cmd->cmd_rfm) {
12090         rel_flags |= MPTSAS_ADDRESS_REPLY;
12091         (void) ddi_dma_sync(mpt->m_dma_reply_frame_hdl, 0, 0,
12092             DDI_DMA_SYNC_FORCPU);
12093         reply = (pMpi2DiagReleaseReply_t)(mpt->m_reply_frame +
12094             (cmd->cmd_rfm -
12095                 (mpt->m_reply_frame_dma_addr&0xfffffffful)));

12097         /*
12098          * Get the reply message data
12099          */
12100         iocstatus = ddi_get16(mpt->m_acc_reply_frame_hdl,
12101             &reply->IOCStatus);
12102         iocloginfo = ddi_get32(mpt->m_acc_reply_frame_hdl,
12103             &reply->IOCLogInfo);

12105         /*
12106          * If release failed quit.
12107          */
12108         if ((iocstatus != MPI2_IOCSTATUS_SUCCESS) ||
12109             pBuffer->owned_by_firmware) {
12110             status = DDI_FAILURE;
12111             NDBG13(("release FW Diag Buffer failed: "
12112                 "IOCStatus=0x%x, IOCLogInfo=0x%x", iocstatus,
12113                 iocloginfo));
12114             goto out;
12115         }

12117         /*
12118          * Release was successful.
12119          */
12120         *return_code = MPTSAS_FW_DIAG_ERROR_SUCCESS;
12121         status = DDI_SUCCESS;

12123         /*
12124          * If this was for an UNREGISTER diag type command, clear the
12125          * unique ID.
12126          */
12127         if (diag_type == MPTSAS_FW_DIAG_TYPE_UNREGISTER) {
12128             pBuffer->unique_id = MPTSAS_FW_DIAG_INVALID_UID;
12129         }

12130     }

12132 out:
12133     /*
12134      * Put the reply frame back on the free queue, increment the free
12135      * index, and write the new index to the free index register. But only
12136      * if this reply is an ADDRESS reply.
12137      */
12138     if (rel_flags & MPTSAS_ADDRESS_REPLY) {
12139         ddi_put32(mpt->m_acc_free_queue_hdl,

```

```

12140         &((uint32_t *) (void *) mpt->m_free_queue)[mpt->m_free_index],
12141         cmd->cmd_rfm);
12142         (void) ddi_dma_sync(mpt->m_dma_free_queue_hdl, 0, 0,
12143         DDI_DMA_SYNC_FORDEV);
12144         if (++mpt->m_free_index == mpt->m_free_queue_depth) {
12145             mpt->m_free_index = 0;
12146         }
12147         ddi_put32(mpt->m_datap, &mpt->m_reg->ReplyFreeHostIndex,
12148         mpt->m_free_index);
12149     }
12150     if (cmd && (cmd->cmd_flags & CFLAG_PREPARED)) {
12151         mptsas_remove_cmd(mpt, cmd);
12152         rel_flags &= (~MPTSAS_REQUEST_POOL_CMD);
12153     }
12154     if (rel_flags & MPTSAS_REQUEST_POOL_CMD) {
12155         mptsas_return_to_pool(mpt, cmd);
12156     }
12157
12158     return (status);
12159 }
12160
12161 static int
12162 mptsas_diag_register(mptsas_t *mpt, mptsas_fw_diag_register_t *diag_register,
12163         uint32_t *return_code)
12164 {
12165     mptsas_fw_diagnostic_buffer_t *pBuffer;
12166     uint8_t extended_type, buffer_type, i;
12167     uint32_t buffer_size;
12168     uint32_t unique_id;
12169     int status;
12170
12171     ASSERT(mutex_owned(&mpt->m_mutex));
12172
12173     extended_type = diag_register->ExtendedType;
12174     buffer_type = diag_register->BufferType;
12175     buffer_size = diag_register->RequestedBufferSize;
12176     unique_id = diag_register->UniqueID;
12177
12178     /*
12179     * Check for valid buffer type
12180     */
12181     if (buffer_type >= MPI2_DIAG_BUF_TYPE_COUNT) {
12182         *return_code = MPTSAS_FW_DIAG_ERROR_INVALID_PARAMETER;
12183         return (DDI_FAILURE);
12184     }
12185
12186     /*
12187     * Get the current buffer and look up the unique ID. The unique ID
12188     * should not be found. If it is, the ID is already in use.
12189     */
12190     i = mptsas_get_fw_diag_buffer_number(mpt, unique_id);
12191     pBuffer = &mpt->m_fw_diag_buffer_list[buffer_type];
12192     if (i != MPTSAS_FW_DIAGNOSTIC_UID_NOT_FOUND) {
12193         *return_code = MPTSAS_FW_DIAG_ERROR_INVALID_UID;
12194         return (DDI_FAILURE);
12195     }
12196
12197     /*
12198     * The buffer's unique ID should not be registered yet, and the given
12199     * unique ID cannot be 0.
12200     */
12201     if ((pBuffer->unique_id != MPTSAS_FW_DIAG_INVALID_UID) ||
12202         (unique_id == MPTSAS_FW_DIAG_INVALID_UID)) {
12203         *return_code = MPTSAS_FW_DIAG_ERROR_INVALID_UID;
12204         return (DDI_FAILURE);
12205     }

```

```

12207     /*
12208     * If this buffer is already posted as immediate, just change owner.
12209     */
12210     if (pBuffer->immediate && pBuffer->owned_by_firmware &&
12211         (pBuffer->unique_id == MPTSAS_FW_DIAG_INVALID_UID)) {
12212         pBuffer->immediate = FALSE;
12213         pBuffer->unique_id = unique_id;
12214         return (DDI_SUCCESS);
12215     }
12216
12217     /*
12218     * Post a new buffer after checking if it's enabled. The DMA buffer
12219     * that is allocated will be contiguous (sgl_len = 1).
12220     */
12221     if (!pBuffer->enabled) {
12222         *return_code = MPTSAS_FW_DIAG_ERROR_NO_BUFFER;
12223         return (DDI_FAILURE);
12224     }
12225     bzero(&pBuffer->buffer_data, sizeof (mptsas_dma_alloc_state_t));
12226     pBuffer->buffer_data.size = buffer_size;
12227     if (mptsas_dma_alloc(mpt, &pBuffer->buffer_data) != DDI_SUCCESS) {
12228         mptsas_log(mpt, CE_WARN, "failed to alloc DMA resource for "
12229         "diag buffer: size = %d bytes", buffer_size);
12230         *return_code = MPTSAS_FW_DIAG_ERROR_NO_BUFFER;
12231         return (DDI_FAILURE);
12232     }
12233
12234     /*
12235     * Copy the given info to the diag buffer and post the buffer.
12236     */
12237     pBuffer->buffer_type = buffer_type;
12238     pBuffer->immediate = FALSE;
12239     if (buffer_type == MPI2_DIAG_BUF_TYPE_TRACE) {
12240         for (i = 0; i < (sizeof (pBuffer->product_specific) / 4);
12241             i++) {
12242             pBuffer->product_specific[i] =
12243                 diag_register->ProductSpecific[i];
12244         }
12245     }
12246     pBuffer->extended_type = extended_type;
12247     pBuffer->unique_id = unique_id;
12248     status = mptsas_post_fw_diag_buffer(mpt, pBuffer, return_code);
12249
12250     if (mptsas_check_dma_handle(pBuffer->buffer_data.handle) !=
12251         DDI_SUCCESS) {
12252         mptsas_log(mpt, CE_WARN, "Check of DMA handle failed in "
12253         "mptsas_diag_register.");
12254         ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_UNAFFECTED);
12255         status = DDI_FAILURE;
12256     }
12257
12258     /*
12259     * In case there was a failure, free the DMA buffer.
12260     */
12261     if (status == DDI_FAILURE) {
12262         mptsas_dma_free(&pBuffer->buffer_data);
12263     }
12264
12265     return (status);
12266 }
12267
12268 static int
12269 mptsas_diag_unregister(mptsas_t *mpt,
12270         mptsas_fw_diag_unregister_t *diag_unregister, uint32_t *return_code)
12271 {

```

```

12272 mptsas_fw_diagnostic_buffer_t *pBuffer;
12273 uint8_t i;
12274 uint32_t unique_id;
12275 int status;

12277 ASSERT(mutex_owned(&mpt->m_mutex));

12279 unique_id = diag_unregister->UniqueId;

12281 /*
12282  * Get the current buffer and look up the unique ID. The unique ID
12283  * should be there.
12284  */
12285 i = mptsas_get_fw_diag_buffer_number(mpt, unique_id);
12286 if (i == MPTSAS_FW_DIAGNOSTIC_UID_NOT_FOUND) {
12287     *return_code = MPTSAS_FW_DIAG_ERROR_INVALID_UID;
12288     return (DDI_FAILURE);
12289 }

12291 pBuffer = &mpt->m_fw_diag_buffer_list[i];

12293 /*
12294  * Try to release the buffer from FW before freeing it. If release
12295  * fails, don't free the DMA buffer in case FW tries to access it
12296  * later. If buffer is not owned by firmware, can't release it.
12297  */
12298 if (!pBuffer->owned_by_firmware) {
12299     status = DDI_SUCCESS;
12300 } else {
12301     status = mptsas_release_fw_diag_buffer(mpt, pBuffer,
12302     return_code, MPTSAS_FW_DIAG_TYPE_UNREGISTER);
12303 }

12305 /*
12306  * At this point, return the current status no matter what happens with
12307  * the DMA buffer.
12308  */
12309 pBuffer->unique_id = MPTSAS_FW_DIAG_INVALID_UID;
12310 if (status == DDI_SUCCESS) {
12311     if (mptsas_check_dma_handle(pBuffer->buffer_data.handle) !=
12312         DDI_SUCCESS) {
12313         mptsas_log(mpt, CE_WARN, "Check of DMA handle failed "
12314             "in mptsas_diag_unregister.");
12315         ddi_fm_service_impact(mpt->m_dip,
12316             DDI_SERVICE_UNAFFECTED);
12317     }
12318     mptsas_dma_free(&pBuffer->buffer_data);
12319 }

12321 return (status);
12322 }

12324 static int
12325 mptsas_diag_query(mptsas_t *mpt, mptsas_fw_diag_query_t *diag_query,
12326     uint32_t *return_code)
12327 {
12328     mptsas_fw_diagnostic_buffer_t *pBuffer;
12329     uint8_t i;
12330     uint32_t unique_id;

12332     ASSERT(mutex_owned(&mpt->m_mutex));

12334     unique_id = diag_query->UniqueId;

12336     /*
12337      * If ID is valid, query on ID.

```

```

12338     * If ID is invalid, query on buffer type.
12339     */
12340     if (unique_id == MPTSAS_FW_DIAG_INVALID_UID) {
12341         i = diag_query->BufferType;
12342         if (i >= MPI2_DIAG_BUF_TYPE_COUNT) {
12343             *return_code = MPTSAS_FW_DIAG_ERROR_INVALID_UID;
12344             return (DDI_FAILURE);
12345         }
12346     } else {
12347         i = mptsas_get_fw_diag_buffer_number(mpt, unique_id);
12348         if (i == MPTSAS_FW_DIAGNOSTIC_UID_NOT_FOUND) {
12349             *return_code = MPTSAS_FW_DIAG_ERROR_INVALID_UID;
12350             return (DDI_FAILURE);
12351         }
12352     }

12354     /*
12355      * Fill query structure with the diag buffer info.
12356      */
12357     pBuffer = &mpt->m_fw_diag_buffer_list[i];
12358     diag_query->BufferType = pBuffer->buffer_type;
12359     diag_query->ExtendedType = pBuffer->extended_type;
12360     if (diag_query->BufferType == MPI2_DIAG_BUF_TYPE_TRACE) {
12361         for (i = 0; i < (sizeof (diag_query->ProductSpecific) / 4);
12362             i++) {
12363             diag_query->ProductSpecific[i] =
12364                 pBuffer->product_specific[i];
12365         }
12366     }
12367     diag_query->TotalBufferSize = pBuffer->buffer_data.size;
12368     diag_query->DriverAddedBufferSize = 0;
12369     diag_query->UniqueId = pBuffer->unique_id;
12370     diag_query->ApplicationFlags = 0;
12371     diag_query->DiagnosticFlags = 0;

12373     /*
12374      * Set/Clear application flags
12375      */
12376     if (pBuffer->immediate) {
12377         diag_query->ApplicationFlags &= ~MPTSAS_FW_DIAG_FLAG_APP_OWNED;
12378     } else {
12379         diag_query->ApplicationFlags |= MPTSAS_FW_DIAG_FLAG_APP_OWNED;
12380     }
12381     if (pBuffer->valid_data || pBuffer->owned_by_firmware) {
12382         diag_query->ApplicationFlags |=
12383             MPTSAS_FW_DIAG_FLAG_BUFFER_VALID;
12384     } else {
12385         diag_query->ApplicationFlags &=
12386             ~MPTSAS_FW_DIAG_FLAG_BUFFER_VALID;
12387     }
12388     if (pBuffer->owned_by_firmware) {
12389         diag_query->ApplicationFlags |=
12390             MPTSAS_FW_DIAG_FLAG_FW_BUFFER_ACCESS;
12391     } else {
12392         diag_query->ApplicationFlags &=
12393             ~MPTSAS_FW_DIAG_FLAG_FW_BUFFER_ACCESS;
12394     }

12396     return (DDI_SUCCESS);
12397 }

12399 static int
12400 mptsas_diag_read_buffer(mptsas_t *mpt,
12401     mptsas_diag_read_buffer_t *diag_read_buffer, uint8_t *ioctl_buf,
12402     uint32_t *return_code, int ioctl_mode)
12403 {

```

```

12404     mptsas_fw_diagnostic_buffer_t    *pBuffer;
12405     uint8_t                          i, *pData;
12406     uint32_t                        unique_id, byte;
12407     int                             status;

12409     ASSERT(mutex_owned(&mpt->m_mutex));

12411     unique_id = diag_read_buffer->UniqueId;

12413     /*
12414      * Get the current buffer and look up the unique ID. The unique ID
12415      * should be there.
12416      */
12417     i = mptsas_get_fw_diag_buffer_number(mpt, unique_id);
12418     if (i == MPTSAS_FW_DIAGNOSTIC_UID_NOT_FOUND) {
12419         *return_code = MPTSAS_FW_DIAG_ERROR_INVALID_UID;
12420         return (DDI_FAILURE);
12421     }

12423     pBuffer = &mpt->m_fw_diag_buffer_list[i];

12425     /*
12426      * Make sure requested read is within limits
12427      */
12428     if (diag_read_buffer->StartingOffset + diag_read_buffer->BytesToRead >
12429         pBuffer->buffer_data.size) {
12430         *return_code = MPTSAS_FW_DIAG_ERROR_INVALID_PARAMETER;
12431         return (DDI_FAILURE);
12432     }

12434     /*
12435      * Copy the requested data from DMA to the diag_read_buffer. The DMA
12436      * buffer that was allocated is one contiguous buffer.
12437      */
12438     pData = (uint8_t *) (pBuffer->buffer_data.memp +
12439         diag_read_buffer->StartingOffset);
12440     (void) ddi_dma_sync(pBuffer->buffer_data.handle, 0, 0,
12441         DDI_DMA_SYNC_FORCPU);
12442     for (byte = 0; byte < diag_read_buffer->BytesToRead; byte++) {
12443         if (ddi_copyout(pData + byte, iocctl_buf + byte, 1, iocctl_mode)
12444             != 0) {
12445             return (DDI_FAILURE);
12446         }
12447     }
12448     diag_read_buffer->Status = 0;

12450     /*
12451      * Set or clear the Force Release flag.
12452      */
12453     if (pBuffer->force_release) {
12454         diag_read_buffer->Flags |= MPTSAS_FW_DIAG_FLAG_FORCE_RELEASE;
12455     } else {
12456         diag_read_buffer->Flags &= ~MPTSAS_FW_DIAG_FLAG_FORCE_RELEASE;
12457     }

12459     /*
12460      * If buffer is to be reregistered, make sure it's not already owned by
12461      * firmware first.
12462      */
12463     status = DDI_SUCCESS;
12464     if (!pBuffer->owned_by_firmware) {
12465         if (diag_read_buffer->Flags & MPTSAS_FW_DIAG_FLAG_REREGISTER) {
12466             status = mptsas_post_fw_diag_buffer(mpt, pBuffer,
12467                 return_code);
12468         }
12469     }

```

```

12471         return (status);
12472     }

12474     static int
12475     mptsas_diag_release(mptsas_t *mpt, mptsas_fw_diag_release_t *diag_release,
12476         uint32_t *return_code)
12477     {
12478         mptsas_fw_diagnostic_buffer_t    *pBuffer;
12479         uint8_t                          i;
12480         uint32_t                        unique_id;
12481         int                             status;

12483         ASSERT(mutex_owned(&mpt->m_mutex));

12485         unique_id = diag_release->UniqueId;

12487         /*
12488          * Get the current buffer and look up the unique ID. The unique ID
12489          * should be there.
12490          */
12491         i = mptsas_get_fw_diag_buffer_number(mpt, unique_id);
12492         if (i == MPTSAS_FW_DIAGNOSTIC_UID_NOT_FOUND) {
12493             *return_code = MPTSAS_FW_DIAG_ERROR_INVALID_UID;
12494             return (DDI_FAILURE);
12495         }

12497         pBuffer = &mpt->m_fw_diag_buffer_list[i];

12499         /*
12500          * If buffer is not owned by firmware, it's already been released.
12501          */
12502         if (!pBuffer->owned_by_firmware) {
12503             *return_code = MPTSAS_FW_DIAG_ERROR_ALREADY_RELEASED;
12504             return (DDI_FAILURE);
12505         }

12507         /*
12508          * Release the buffer.
12509          */
12510         status = mptsas_release_fw_diag_buffer(mpt, pBuffer, return_code,
12511             MPTSAS_FW_DIAG_TYPE_RELEASE);
12512         return (status);
12513     }

12515     static int
12516     mptsas_do_diag_action(mptsas_t *mpt, uint32_t action, uint8_t *diag_action,
12517         uint32_t length, uint32_t *return_code, int iocctl_mode)
12518     {
12519         mptsas_fw_diag_register_t        diag_register;
12520         mptsas_fw_diag_unregister_t      diag_unregister;
12521         mptsas_fw_diag_query_t           diag_query;
12522         mptsas_diag_read_buffer_t        diag_read_buffer;
12523         mptsas_fw_diag_release_t         diag_release;
12524         int                             status = DDI_SUCCESS;
12525         uint32_t                          original_return_code, read_buf_len;

12527         ASSERT(mutex_owned(&mpt->m_mutex));

12529         original_return_code = *return_code;
12530         *return_code = MPTSAS_FW_DIAG_ERROR_SUCCESS;

12532         switch (action) {
12533             case MPTSAS_FW_DIAG_TYPE_REGISTER:
12534                 if (!length) {
12535                     *return_code =

```



```

12536         MPTSAS_FW_DIAG_ERROR_INVALID_PARAMETER;
12537         status = DDI_FAILURE;
12538         break;
12539     }
12540     if (ddi_copyin(diag_action, &diag_register,
12541         sizeof (diag_register), ioctl_mode) != 0) {
12542         return (DDI_FAILURE);
12543     }
12544     status = mptsas_diag_register(mpt, &diag_register,
12545         return_code);
12546     break;
12547
12548 case MPTSAS_FW_DIAG_TYPE_UNREGISTER:
12549     if (length < sizeof (diag_unregister)) {
12550         *return_code =
12551             MPTSAS_FW_DIAG_ERROR_INVALID_PARAMETER;
12552         status = DDI_FAILURE;
12553         break;
12554     }
12555     if (ddi_copyin(diag_action, &diag_unregister,
12556         sizeof (diag_unregister), ioctl_mode) != 0) {
12557         return (DDI_FAILURE);
12558     }
12559     status = mptsas_diag_unregister(mpt, &diag_unregister,
12560         return_code);
12561     break;
12562
12563 case MPTSAS_FW_DIAG_TYPE_QUERY:
12564     if (length < sizeof (diag_query)) {
12565         *return_code =
12566             MPTSAS_FW_DIAG_ERROR_INVALID_PARAMETER;
12567         status = DDI_FAILURE;
12568         break;
12569     }
12570     if (ddi_copyin(diag_action, &diag_query,
12571         sizeof (diag_query), ioctl_mode) != 0) {
12572         return (DDI_FAILURE);
12573     }
12574     status = mptsas_diag_query(mpt, &diag_query,
12575         return_code);
12576     if (status == DDI_SUCCESS) {
12577         if (ddi_copyout(&diag_query, diag_action,
12578             sizeof (diag_query), ioctl_mode) != 0) {
12579             return (DDI_FAILURE);
12580         }
12581     }
12582     break;
12583
12584 case MPTSAS_FW_DIAG_TYPE_READ_BUFFER:
12585     if (ddi_copyin(diag_action, &diag_read_buffer,
12586         sizeof (diag_read_buffer) - 4, ioctl_mode) != 0) {
12587         return (DDI_FAILURE);
12588     }
12589     read_buf_len = sizeof (diag_read_buffer) -
12590         sizeof (diag_read_buffer.DataBuffer) +
12591         diag_read_buffer.BytesToRead;
12592     if (length < read_buf_len) {
12593         *return_code =
12594             MPTSAS_FW_DIAG_ERROR_INVALID_PARAMETER;
12595         status = DDI_FAILURE;
12596         break;
12597     }
12598     status = mptsas_diag_read_buffer(mpt,
12599         &diag_read_buffer, diag_action +
12600         sizeof (diag_read_buffer) - 4, return_code,
12601         ioctl_mode);

```

```

12602         if (status == DDI_SUCCESS) {
12603             if (ddi_copyout(&diag_read_buffer, diag_action,
12604                 sizeof (diag_read_buffer) - 4, ioctl_mode)
12605                 != 0) {
12606                 return (DDI_FAILURE);
12607             }
12608         }
12609         break;
12610
12611 case MPTSAS_FW_DIAG_TYPE_RELEASE:
12612     if (length < sizeof (diag_release)) {
12613         *return_code =
12614             MPTSAS_FW_DIAG_ERROR_INVALID_PARAMETER;
12615         status = DDI_FAILURE;
12616         break;
12617     }
12618     if (ddi_copyin(diag_action, &diag_release,
12619         sizeof (diag_release), ioctl_mode) != 0) {
12620         return (DDI_FAILURE);
12621     }
12622     status = mptsas_diag_release(mpt, &diag_release,
12623         return_code);
12624     break;
12625
12626 default:
12627     *return_code = MPTSAS_FW_DIAG_ERROR_INVALID_PARAMETER;
12628     status = DDI_FAILURE;
12629     break;
12630 }
12631
12632 if ((status == DDI_FAILURE) &&
12633     (original_return_code == MPTSAS_FW_DIAG_NEW) &&
12634     (*return_code != MPTSAS_FW_DIAG_ERROR_SUCCESS)) {
12635     status = DDI_SUCCESS;
12636 }
12637
12638 return (status);
12639 }
12640
12641 static int
12642 mptsas_diag_action(mptsas_t *mpt, mptsas_diag_action_t *user_data, int mode)
12643 {
12644     int status;
12645     mptsas_diag_action_t driver_data;
12646
12647     ASSERT(mutex_owned(&mpt->m_mutex));
12648
12649     /*
12650      * Copy the user data to a driver data buffer.
12651      */
12652     if (ddi_copyin(user_data, &driver_data, sizeof (mptsas_diag_action_t),
12653         mode) == 0) {
12654         /*
12655          * Send diag action request if Action is valid
12656          */
12657         if (driver_data.Action == MPTSAS_FW_DIAG_TYPE_REGISTER ||
12658             driver_data.Action == MPTSAS_FW_DIAG_TYPE_UNREGISTER ||
12659             driver_data.Action == MPTSAS_FW_DIAG_TYPE_QUERY ||
12660             driver_data.Action == MPTSAS_FW_DIAG_TYPE_READ_BUFFER ||
12661             driver_data.Action == MPTSAS_FW_DIAG_TYPE_RELEASE) {
12662             status = mptsas_do_diag_action(mpt, driver_data.Action,
12663                 (void *) (uintptr_t) driver_data.PtrDiagAction,
12664                 driver_data.Length, &driver_data.ReturnCode,
12665                 mode);
12666             if (status == DDI_SUCCESS) {
12667                 if (ddi_copyout(&driver_data.ReturnCode,

```

```

12668         &user_data->ReturnCode,
12669         sizeof (user_data->ReturnCode), mode)
12670         != 0) {
12671             status = EFAULT;
12672         } else {
12673             status = 0;
12674         }
12675     } else {
12676         status = EIO;
12677     }
12678 } else {
12679     status = EINVAL;
12680 }
12681 } else {
12682     status = EFAULT;
12683 }
12684
12685 return (status);
12686 }
12687
12688 /*
12689  * This routine handles the "event query" ioctl.
12690  */
12691 static int
12692 mptsas_event_query(mptsas_t *mpt, mptsas_event_query_t *data, int mode,
12693 int *rval)
12694 {
12695     int status;
12696     mptsas_event_query_t driverdata;
12697     uint8_t i;
12698
12699     driverdata.Entries = MPTSAS_EVENT_QUEUE_SIZE;
12700
12701     mutex_enter(&mpt->m_mutex);
12702     for (i = 0; i < 4; i++) {
12703         driverdata.Types[i] = mpt->m_event_mask[i];
12704     }
12705     mutex_exit(&mpt->m_mutex);
12706
12707     if (ddi_copyout(&driverdata, data, sizeof (driverdata), mode) != 0) {
12708         status = EFAULT;
12709     } else {
12710         *rval = MPTIOCTL_STATUS_GOOD;
12711         status = 0;
12712     }
12713
12714     return (status);
12715 }
12716
12717 /*
12718  * This routine handles the "event enable" ioctl.
12719  */
12720 static int
12721 mptsas_event_enable(mptsas_t *mpt, mptsas_event_enable_t *data, int mode,
12722 int *rval)
12723 {
12724     int status;
12725     mptsas_event_enable_t driverdata;
12726     uint8_t i;
12727
12728     if (ddi_copyin(data, &driverdata, sizeof (driverdata), mode) == 0) {
12729         mutex_enter(&mpt->m_mutex);
12730         for (i = 0; i < 4; i++) {
12731             mpt->m_event_mask[i] = driverdata.Types[i];
12732         }
12733         mutex_exit(&mpt->m_mutex);

```

```

12735         *rval = MPTIOCTL_STATUS_GOOD;
12736         status = 0;
12737     } else {
12738         status = EFAULT;
12739     }
12740     return (status);
12741 }
12742
12743 /*
12744  * This routine handles the "event report" ioctl.
12745  */
12746 static int
12747 mptsas_event_report(mptsas_t *mpt, mptsas_event_report_t *data, int mode,
12748 int *rval)
12749 {
12750     int status;
12751     mptsas_event_report_t driverdata;
12752
12753     mutex_enter(&mpt->m_mutex);
12754
12755     if (ddi_copyin(&data->Size, &driverdata.Size, sizeof (driverdata.Size),
12756 mode) == 0) {
12757         if (driverdata.Size >= sizeof (mpt->m_events)) {
12758             if (ddi_copyout(mpt->m_events, data->Events,
12759 sizeof (mpt->m_events), mode) != 0) {
12760                 status = EFAULT;
12761             } else {
12762                 if (driverdata.Size > sizeof (mpt->m_events)) {
12763                     driverdata.Size =
12764                         sizeof (mpt->m_events);
12765                     if (ddi_copyout(&driverdata.Size,
12766 &data->Size,
12767 sizeof (driverdata.Size),
12768 mode) != 0) {
12769                         status = EFAULT;
12770                     } else {
12771                         *rval = MPTIOCTL_STATUS_GOOD;
12772                         status = 0;
12773                     }
12774                 } else {
12775                     *rval = MPTIOCTL_STATUS_GOOD;
12776                     status = 0;
12777                 }
12778             }
12779         } else {
12780             *rval = MPTIOCTL_STATUS_LEN_TOO_SHORT;
12781             status = 0;
12782         }
12783     } else {
12784         status = EFAULT;
12785     }
12786
12787     mutex_exit(&mpt->m_mutex);
12788     return (status);
12789 }
12790
12791 static void
12792 mptsas_lookup_pci_data(mptsas_t *mpt, mptsas_adapter_data_t *adapter_data)
12793 {
12794     int *reg_data;
12795     uint_t reglen;
12796
12797     /*
12798      * Lookup the 'reg' property and extract the other data
12799      */

```

```

12800     if (ddi_prop_lookup_int_array(DDI_DEV_T_ANY, mpt->m_dip,
12801         DDI_PROP_DONTPASS, "reg", &reg_data, &reglen) ==
12802         DDI_PROP_SUCCESS) {
12803         /*
12804          * Extract the PCI data from the 'reg' property first DWORD.
12805          * The entry looks like the following:
12806          * First DWORD:
12807          * Bits 0 - 7 8-bit Register number
12808          * Bits 8 - 10 3-bit Function number
12809          * Bits 11 - 15 5-bit Device number
12810          * Bits 16 - 23 8-bit Bus number
12811          * Bits 24 - 25 2-bit Address Space type identifier
12812          */
12813         adapter_data->PciInformation.u.bits.BusNumber =
12814             (reg_data[0] & 0x00FF0000) >> 16;
12815         adapter_data->PciInformation.u.bits.DeviceNumber =
12816             (reg_data[0] & 0x0000F800) >> 11;
12817         adapter_data->PciInformation.u.bits.FunctionNumber =
12818             (reg_data[0] & 0x00000700) >> 8;
12819         ddi_prop_free((void *)reg_data);
12820     } else {
12821         /*
12822          * If we can't determine the PCI data then we fill in FF's for
12823          * the data to indicate this.
12824          */
12825         adapter_data->PCIDeviceHwId = 0xFFFFFFFF;
12826         adapter_data->MpiPortNumber = 0xFFFFFFFF;
12827         adapter_data->PciInformation.u.AsDWORD = 0xFFFFFFFF;
12828     }
12829 }
12830
12831 /*
12832  * Saved in the mpt->m_fwversion
12833  */
12834 adapter_data->MpiFirmwareVersion = mpt->m_fwversion;
12835 }
12836
12837 static void
12838 mptsas_read_adapter_data(mptsas_t *mpt, mptsas_adapter_data_t *adapter_data)
12839 {
12840     char    *driver_verstr = MPTSAS_MOD_STRING;
12841
12842     mptsas_lookup_pci_data(mpt, adapter_data);
12843     adapter_data->AdapterType = MPTIOCTL_ADAPTER_TYPE_SAS3;
12844     adapter_data->PCIDeviceHwId = (uint32_t)mpt->m_devid;
12845     adapter_data->PCIDeviceHwRev = (uint32_t)mpt->m_revid;
12846     adapter_data->SubSystemId = (uint32_t)mpt->m_ssid;
12847     adapter_data->SubsystemVendorId = (uint32_t)mpt->m_svid;
12848     (void) strcpy((char *)&adapter_data->DriverVersion[0], driver_verstr);
12849     adapter_data->BiosVersion = 0;
12850     (void) mptsas_get_bios_page3(mpt, &adapter_data->BiosVersion);
12851 }
12852
12853 static void
12854 mptsas_read_pci_info(mptsas_t *mpt, mptsas_pci_info_t *pci_info)
12855 {
12856     int      *reg_data, i;
12857     uint_t    reglen;
12858
12859     /*
12860      * Lookup the 'reg' property and extract the other data
12861      */
12862     if (ddi_prop_lookup_int_array(DDI_DEV_T_ANY, mpt->m_dip,
12863         DDI_PROP_DONTPASS, "reg", &reg_data, &reglen) ==
12864         DDI_PROP_SUCCESS) {
12865         /*

```

```

12866         * Extract the PCI data from the 'reg' property first DWORD.
12867         * The entry looks like the following:
12868         * First DWORD:
12869         * Bits 8 - 10 3-bit Function number
12870         * Bits 11 - 15 5-bit Device number
12871         * Bits 16 - 23 8-bit Bus number
12872         */
12873         pci_info->BusNumber = (reg_data[0] & 0x00FF0000) >> 16;
12874         pci_info->DeviceNumber = (reg_data[0] & 0x0000F800) >> 11;
12875         pci_info->FunctionNumber = (reg_data[0] & 0x00000700) >> 8;
12876         ddi_prop_free((void *)reg_data);
12877     } else {
12878         /*
12879          * If we can't determine the PCI info then we fill in FF's for
12880          * the data to indicate this.
12881          */
12882         pci_info->BusNumber = 0xFFFFFFFF;
12883         pci_info->DeviceNumber = 0xFF;
12884         pci_info->FunctionNumber = 0xFF;
12885     }
12886
12887     /*
12888      * Now get the interrupt vector and the pci header. The vector can
12889      * only be 0 right now. The header is the first 256 bytes of config
12890      * space.
12891      */
12892     pci_info->InterruptVector = 0;
12893     for (i = 0; i < sizeof(pci_info->PciHeader); i++) {
12894         pci_info->PciHeader[i] = pci_config_get8(mpt->m_config_handle,
12895             i);
12896     }
12897 }
12898
12899 static int
12900 mptsas_reg_access(mptsas_t *mpt, mptsas_reg_access_t *data, int mode)
12901 {
12902     int      status = 0;
12903     mptsas_reg_access_t    driverdata;
12904
12905     mutex_enter(&mpt->m_mutex);
12906     if (ddi_copyin(data, &driverdata, sizeof(driverdata), mode) == 0) {
12907         switch (driverdata.Command) {
12908             /*
12909              * IO access is not supported.
12910              */
12911             case REG_IO_READ:
12912             case REG_IO_WRITE:
12913                 mptsas_log(mpt, CE_WARN, "IO access is not "
12914                     "supported. Use memory access.");
12915                 status = EINVAL;
12916                 break;
12917
12918             case REG_MEM_READ:
12919                 driverdata.RegData = ddi_get32(mpt->m_datap,
12920                     (uint32_t *) (void *) mpt->m_reg +
12921                     driverdata.RegOffset);
12922                 if (ddi_copyout(&driverdata.RegData,
12923                     &data->RegData,
12924                     sizeof(driverdata.RegData), mode) != 0) {
12925                     mptsas_log(mpt, CE_WARN, "Register "
12926                         "Read Failed");
12927                     status = EFAULT;
12928                 }
12929                 break;
12930
12931             case REG_MEM_WRITE:

```

```

12932         ddi_put32(mpt->m_datap,
12933             (uint32_t *) (void *) mpt->m_reg +
12934             driverdata.RegOffset,
12935             driverdata.RegData);
12936         break;

12938     default:
12939         status = EINVAL;
12940         break;
12941     }
12942 } else {
12943     status = EFAULT;
12944 }

12946 mutex_exit(&mpt->m_mutex);
12947 return (status);
12948 }

12950 static int
12951 led_control(mptsas_t *mpt, intptr_t data, int mode)
12952 {
12953     int ret = 0;
12954     mptsas_led_control_t lc;
12955     mptsas_target_t *ptgt;

12957     if (ddi_copyin((void *) data, &lc, sizeof (lc), mode) != 0) {
12958         return (EFAULT);
12959     }

12961     if ((lc.Command != MPTSAS_LEDCTL_FLAG_SET &&
12962         lc.Command != MPTSAS_LEDCTL_FLAG_GET) ||
12963         lc.Led < MPTSAS_LEDCTL_LED_MIN ||
12964         lc.Led > MPTSAS_LEDCTL_LED_MAX ||
12965         (lc.Command == MPTSAS_LEDCTL_FLAG_SET && lc.LedStatus != 0 &&
12966         lc.LedStatus != 1)) {
12967         return (EINVAL);
12968     }

12970     if ((lc.Command == MPTSAS_LEDCTL_FLAG_SET && (mode & FWRITE) == 0) ||
12971         (lc.Command == MPTSAS_LEDCTL_FLAG_GET && (mode & FREAD) == 0))
12972         return (EACCES);

12974     /* Locate the target we're interrogating... */
12975     mutex_enter(&mpt->m_mutex);
12976     ptgt = rehash_linear_search(mpt->m_targets,
12977         mptsas_target_eval_slot, &lc);
12978     if (ptgt == NULL) {
12979         /* We could not find a target for that enclosure/slot. */
12980         mutex_exit(&mpt->m_mutex);
12981         return (ENOENT);
12982     }

12984     if (lc.Command == MPTSAS_LEDCTL_FLAG_SET) {
12985         /* Update our internal LED state. */
12986         ptgt->m_led_status &= ~(1 << (lc.Led - 1));
12987         ptgt->m_led_status |= lc.LedStatus << (lc.Led - 1);

12989         /* Flush it to the controller. */
12990         ret = mptsas_flush_led_status(mpt, ptgt);
12991         mutex_exit(&mpt->m_mutex);
12992         return (ret);
12993     }

12995     /* Return our internal LED state. */
12996     lc.LedStatus = (ptgt->m_led_status >> (lc.Led - 1)) & 1;
12997     mutex_exit(&mpt->m_mutex);

```

```

12999     if (ddi_copyout(&lc, (void *) data, sizeof (lc), mode) != 0) {
13000         return (EFAULT);
13001     }

13003     return (0);
13004 }

13006 static int
13007 get_disk_info(mptsas_t *mpt, intptr_t data, int mode)
13008 {
13009     uint16_t i = 0;
13010     uint16_t count = 0;
13011     int ret = 0;
13012     mptsas_target_t *ptgt;
13013     mptsas_disk_info_t *di;
13014     STRUCT_DECL(mptsas_get_disk_info, gdi);

13016     if ((mode & FREAD) == 0)
13017         return (EACCES);

13019     STRUCT_INIT(gdi, get_udatamodel());

13021     if (ddi_copyin((void *) data, STRUCT_BUF(gdi), STRUCT_SIZE(gdi),
13022         mode) != 0) {
13023         return (EFAULT);
13024     }

13026     /* Find out how many targets there are. */
13027     mutex_enter(&mpt->m_mutex);
13028     for (ptgt = rehash_first(mpt->m_targets); ptgt != NULL;
13029         ptgt = rehash_next(mpt->m_targets, ptgt)) {
13030         count++;
13031     }
13032     mutex_exit(&mpt->m_mutex);

13034     /*
13035      * If we haven't been asked to copy out information on each target,
13036      * then just return the count.
13037      */
13038     STRUCT_FSET(gdi, DiskCount, count);
13039     if (STRUCT_FGETP(gdi, PtrDiskInfoArray) == NULL)
13040         goto copy_out;

13042     /*
13043      * If we haven't been given a large enough buffer to copy out into,
13044      * let the caller know.
13045      */
13046     if (STRUCT_FGET(gdi, DiskInfoArraySize) <
13047         count * sizeof (mptsas_disk_info_t)) {
13048         ret = ENOSPC;
13049         goto copy_out;
13050     }

13052     di = kmem_zalloc(count * sizeof (mptsas_disk_info_t), KM_SLEEP);

13054     mutex_enter(&mpt->m_mutex);
13055     for (ptgt = rehash_first(mpt->m_targets); ptgt != NULL;
13056         ptgt = rehash_next(mpt->m_targets, ptgt)) {
13057         if (i >= count) {
13058             /*
13059              * The number of targets changed while we weren't
13060              * looking, so give up.
13061              */
13062             rehash_rele(mpt->m_targets, ptgt);
13063             mutex_exit(&mpt->m_mutex);

```

```

13064         kmem_free(di, count * sizeof (mptsas_disk_info_t));
13065         return (EAGAIN);
13066     }
13067     di[i].Instance = mpt->m_instance;
13068     di[i].Enclosure = ptgt->m_enclosure;
13069     di[i].Slot = ptgt->m_slot_num;
13070     di[i].SasAddress = ptgt->m_addr.mta_wwn;
13071     i++;
13072 }
13073 mutex_exit(&mpt->m_mutex);
13074 STRUCT_FSET(gdi, DiskCount, i);

13076 /* Copy out the disk information to the caller. */
13077 if (ddi_copyout((void *)di, STRUCT_FGETP(gdi, PtrDiskInfoArray),
13078     i * sizeof (mptsas_disk_info_t), mode) != 0) {
13079     ret = EFAULT;
13080 }

13082 kmem_free(di, count * sizeof (mptsas_disk_info_t));

13084 copy_out:
13085 if (ddi_copyout(STRUCT_BUF(gdi), (void *)data, STRUCT_SIZE(gdi),
13086     mode) != 0) {
13087     ret = EFAULT;
13088 }

13090 return (ret);
13091 }

13093 static int
13094 mptsas_ioctl(dev_t dev, int cmd, intptr_t data, int mode, cred_t *credp,
13095     int *rval)
13096 {
13097     int                status = 0;
13098     mptsas_t           *mpt;
13099     mptsas_update_flash_t flashdata;
13100     mptsas_pass_thru_t  passthru_data;
13101     mptsas_adapter_data_t adapter_data;
13102     mptsas_pci_info_t   pci_info;
13103     int                copylen;

13105     int                iport_flag = 0;
13106     dev_info_t         *dip = NULL;
13107     mptsas_phymask_t    phymask = 0;
13108     struct devctl_iocdata *dcp = NULL;
13109     char                *addr = NULL;
13110     mptsas_target_t     *ptgt = NULL;

13112     *rval = MPTIOCTL_STATUS_GOOD;
13113     if (secpolicy_sys_config(credp, B_FALSE) != 0) {
13114         return (EPERM);
13115     }

13117     mpt = ddi_get_soft_state(mptsas3_state, MINOR2INST(getminor(dev)));
13118     if (mpt == NULL) {
13119         /*
13120          * Called from iport node, get the states
13121          */
13122         iport_flag = 1;
13123         dip = mptsas_get_dip_from_dev(dev, &phymask);
13124         if (dip == NULL) {
13125             return (ENXIO);
13126         }
13127         mpt = DIP2MPT(dip);
13128     }
13129     /* Make sure power level is D0 before accessing registers */

```

```

13130     mutex_enter(&mpt->m_mutex);
13131     if (mpt->m_options & MPTSAS_OPT_PM) {
13132         (void) pm_busy_component(mpt->m_dip, 0);
13133         if (mpt->m_power_level != PM_LEVEL_D0) {
13134             mutex_exit(&mpt->m_mutex);
13135             if (pm_raise_power(mpt->m_dip, 0, PM_LEVEL_D0) !=
13136                 DDI_SUCCESS) {
13137                 mptsas_log(mpt, CE_WARN,
13138                     "mptsas3%d: mptsas_ioctl: Raise power "
13139                     "request failed.", mpt->m_instance);
13140                 (void) pm_idle_component(mpt->m_dip, 0);
13141                 return (ENXIO);
13142             }
13143         } else {
13144             mutex_exit(&mpt->m_mutex);
13145         }
13146     } else {
13147         mutex_exit(&mpt->m_mutex);
13148     }

13150     if (iport_flag) {
13151         status = scsi_hba_ioctl(dev, cmd, data, mode, credp, rval);
13152         if (status != 0) {
13153             goto out;
13154         }
13155         /*
13156          * The following code control the OK2RM LED, it doesn't affect
13157          * the ioctl return status.
13158          */
13159         if ((cmd == DEVCTL_DEVICE_ONLINE) ||
13160             (cmd == DEVCTL_DEVICE_OFFLINE)) {
13161             if (ndi_dc_allochdl((void *)data, &dcp) !=
13162                 NDI_SUCCESS) {
13163                 goto out;
13164             }
13165             addr = ndi_dc_getaddr(dcp);
13166             ptgt = mptsas_addr_to_ptgt(mpt, addr, phymask);
13167             if (ptgt == NULL) {
13168                 NDBG14(("mptsas_ioctl led control: tgt %s not "
13169                     "found", addr));
13170                 ndi_dc_freehdl(dcp);
13171                 goto out;
13172             }
13173             mutex_enter(&mpt->m_mutex);
13174             if (cmd == DEVCTL_DEVICE_ONLINE) {
13175                 ptgt->m_tgt_unconfigured = 0;
13176             } else if (cmd == DEVCTL_DEVICE_OFFLINE) {
13177                 ptgt->m_tgt_unconfigured = 1;
13178             }
13179             if (cmd == DEVCTL_DEVICE_OFFLINE) {
13180                 ptgt->m_led_status |=
13181                     (1 << (MPTSAS_LEDCTL_LED_OK2RM - 1));
13182             } else {
13183                 ptgt->m_led_status &=
13184                     ~(1 << (MPTSAS_LEDCTL_LED_OK2RM - 1));
13185             }
13186             (void) mptsas_flush_led_status(mpt, ptgt);
13187             mutex_exit(&mpt->m_mutex);
13188             ndi_dc_freehdl(dcp);
13189         }
13190         goto out;
13191     }
13192     switch (cmd) {
13193     case MPTIOCTL_GET_DISK_INFO:
13194         status = get_disk_info(mpt, data, mode);
13195         break;

```

```

13196         case MPTIOCTL_LED_CONTROL:
13197             status = led_control(mpt, data, mode);
13198             break;
13199         case MPTIOCTL_UPDATE_FLASH:
13200             if (ddi_copyin((void *)data, &flashdata,
13201                 sizeof (struct mptsas_update_flash), mode)) {
13202                 status = EFAULT;
13203                 break;
13204             }
13205
13206             mutex_enter(&mpt->m_mutex);
13207             if (mptsas_update_flash(mpt,
13208                 (caddr_t)(long)flashdata.PtrBuffer,
13209                 flashdata.ImageSize, flashdata.ImageType, mode)) {
13210                 status = EFAULT;
13211             }
13212
13213             /*
13214              * Reset the chip to start using the new
13215              * firmware. Reset if failed also.
13216              */
13217             mpt->m_softstate &= ~MPTSAS_SS_MSG_UNIT_RESET;
13218             if (mptsas_restart_ioc(mpt) == DDI_FAILURE) {
13219                 status = EFAULT;
13220             }
13221             mutex_exit(&mpt->m_mutex);
13222             break;
13223         case MPTIOCTL_PASS_THRU:
13224             /*
13225              * The user has requested to pass through a command to
13226              * be executed by the MPT firmware. Call our routine
13227              * which does this. Only allow one passthru IOCTL at
13228              * one time. Other threads will block on
13229              * m_passthru_mutex, which is of adaptive variant.
13230              */
13231             if (ddi_copyin((void *)data, &passthru_data,
13232                 sizeof (mptsas_pass_thru_t), mode)) {
13233                 status = EFAULT;
13234                 break;
13235             }
13236             mutex_enter(&mpt->m_passthru_mutex);
13237             mutex_enter(&mpt->m_mutex);
13238             status = mptsas_pass_thru(mpt, &passthru_data, mode);
13239             mutex_exit(&mpt->m_mutex);
13240             mutex_exit(&mpt->m_passthru_mutex);
13241
13242             break;
13243         case MPTIOCTL_GET_ADAPTER_DATA:
13244             /*
13245              * The user has requested to read adapter data. Call
13246              * our routine which does this.
13247              */
13248             bzero(&adapter_data, sizeof (mptsas_adapter_data_t));
13249             if (ddi_copyin((void *)data, (void *)&adapter_data,
13250                 sizeof (mptsas_adapter_data_t), mode)) {
13251                 status = EFAULT;
13252                 break;
13253             }
13254             if (adapter_data.StructureLength >=
13255                 sizeof (mptsas_adapter_data_t)) {
13256                 adapter_data.StructureLength = (uint32_t)
13257                     sizeof (mptsas_adapter_data_t);
13258                 copylen = sizeof (mptsas_adapter_data_t);
13259                 mutex_enter(&mpt->m_mutex);
13260                 mptsas_read_adapter_data(mpt, &adapter_data);
13261                 mutex_exit(&mpt->m_mutex);

```

```

13262             } else {
13263                 adapter_data.StructureLength = (uint32_t)
13264                     sizeof (mptsas_adapter_data_t);
13265                 copylen = sizeof (adapter_data.StructureLength);
13266                 *rval = MPTIOCTL_STATUS_LEN_TOO_SHORT;
13267             }
13268             if (ddi_copyout((void *)&adapter_data, (void *)data,
13269                 copylen, mode) != 0) {
13270                 status = EFAULT;
13271             }
13272             break;
13273         case MPTIOCTL_GET_PCI_INFO:
13274             /*
13275              * The user has requested to read pci info. Call
13276              * our routine which does this.
13277              */
13278             bzero(&pci_info, sizeof (mptsas_pci_info_t));
13279             mutex_enter(&mpt->m_mutex);
13280             mptsas_read_pci_info(mpt, &pci_info);
13281             mutex_exit(&mpt->m_mutex);
13282             if (ddi_copyout((void *)&pci_info, (void *)data,
13283                 sizeof (mptsas_pci_info_t), mode) != 0) {
13284                 status = EFAULT;
13285             }
13286             break;
13287         case MPTIOCTL_RESET_ADAPTER:
13288             mutex_enter(&mpt->m_mutex);
13289             mpt->m_softstate &= ~MPTSAS_SS_MSG_UNIT_RESET;
13290             if ((mptsas_restart_ioc(mpt)) == DDI_FAILURE) {
13291                 mptsas_log(mpt, CE_WARN, "reset adapter IOCTL "
13292                     "failed");
13293                 status = EFAULT;
13294             }
13295             mutex_exit(&mpt->m_mutex);
13296             break;
13297         case MPTIOCTL_DIAG_ACTION:
13298             /*
13299              * The user has done a diag buffer action. Call our
13300              * routine which does this. Only allow one diag action
13301              * at one time.
13302              */
13303             mutex_enter(&mpt->m_mutex);
13304             if (mpt->m_diag_action_in_progress) {
13305                 mutex_exit(&mpt->m_mutex);
13306                 return (EBUSY);
13307             }
13308             mpt->m_diag_action_in_progress = 1;
13309             status = mptsas_diag_action(mpt,
13310                 (mptsas_diag_action_t *)data, mode);
13311             mpt->m_diag_action_in_progress = 0;
13312             mutex_exit(&mpt->m_mutex);
13313             break;
13314         case MPTIOCTL_EVENT_QUERY:
13315             /*
13316              * The user has done an event query. Call our routine
13317              * which does this.
13318              */
13319             status = mptsas_event_query(mpt,
13320                 (mptsas_event_query_t *)data, mode, rval);
13321             break;
13322         case MPTIOCTL_EVENT_ENABLE:
13323             /*
13324              * The user has done an event enable. Call our routine
13325              * which does this.
13326              */
13327             status = mptsas_event_enable(mpt,

```

```

13328         (mptsas_event_enable_t *)data, mode, rval);
13329         break;
13330     case MPTIOCTL_EVENT_REPORT:
13331         /*
13332          * The user has done an event report. Call our routine
13333          * which does this.
13334          */
13335         status = mptsas_event_report(mpt,
13336         (mptsas_event_report_t *)data, mode, rval);
13337         break;
13338     case MPTIOCTL_REG_ACCESS:
13339         /*
13340          * The user has requested register access. Call our
13341          * routine which does this.
13342          */
13343         status = mptsas_reg_access(mpt,
13344         (mptsas_reg_access_t *)data, mode);
13345         break;
13346     default:
13347         status = scsi_hba_ioctl(dev, cmd, data, mode, credp,
13348         rval);
13349         break;
13350 }

13352 out:
13353     return (status);
13354 }

13356 int
13357 mptsas_restart_ioc(mptsas_t *mpt)
13358 {
13359     int         rval = DDI_SUCCESS;
13360     mptsas_target_t *ptgt = NULL;

13362     ASSERT(mutex_owned(&mpt->m_mutex));

13364     /*
13365      * Set a flag telling I/O path that we're processing a reset. This is
13366      * needed because after the reset is complete, the hash table still
13367      * needs to be rebuilt. If I/Os are started before the hash table is
13368      * rebuilt, I/O errors will occur. This flag allows I/Os to be marked
13369      * so that they can be retried.
13370      */
13371     mpt->m_in_reset = TRUE;

13373     /*
13374      * Set all throttles to HOLD
13375      */
13376     for (ptgt = rehash_first(mpt->m_targets); ptgt != NULL;
13377          ptgt = rehash_next(mpt->m_targets, ptgt)) {
13378         mptsas_set_throttle_mtx(mpt, ptgt, HOLD_THROTTLE);
13379     }

13381     /*
13382      * Disable interrupts
13383      */
13384     MPTSAS_DISABLE_INTR(mpt);

13386     /*
13387      * Abort all commands: outstanding commands, commands in waitq and
13388      * tx_waitq.
13389      */
13390     mptsas_flush_hba(mpt);

13392     /*
13393      * Reinitialize the chip.

```

```

13394     /*
13395     if (mptsas_init_chip(mpt, FALSE) == DDI_FAILURE) {
13396         rval = DDI_FAILURE;
13397     }

13399     /*
13400     * Enable interrupts again
13401     */
13402     MPTSAS_ENABLE_INTR(mpt);

13404     /*
13405     * If mptsas_init_chip was successful, update the driver data.
13406     */
13407     if (rval == DDI_SUCCESS) {
13408         mptsas_update_driver_data(mpt);
13409     }

13411     /*
13412     * Reset the throttles
13413     */
13414     for (ptgt = rehash_first(mpt->m_targets); ptgt != NULL;
13415          ptgt = rehash_next(mpt->m_targets, ptgt)) {
13416         mptsas_set_throttle_mtx(mpt, ptgt, MAX_THROTTLE);
13417     }

13419     mptsas_doneq_empty(mpt);
13420     mptsas_restart_hba(mpt);

13422     if (rval != DDI_SUCCESS) {
13423         mptsas_fm_ereport(mpt, DDI_FM_DEVICE_NO_RESPONSE);
13424         ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_LOST);
13425     }

13427     /*
13428     * Clear the reset flag so that I/Os can continue.
13429     */
13430     mpt->m_in_reset = FALSE;

13432     return (rval);
13433 }

13435 static int
13436 mptsas_init_chip(mptsas_t *mpt, int first_time)
13437 {
13438     ddi_dma_cookie_t    cookie;
13439     mptsas_reply_pqueue_t *rpqp;
13440     uint32_t            i, j;
13441     int                 rval;

13443     /*
13444     * Check to see if the firmware image is valid
13445     */
13446     if (ddi_get32(mpt->m_datap, &mpt->m_reg->HostDiagnostic) &
13447         MPI2_DIAG_FLASH_BAD_SIG) {
13448         mptsas_log(mpt, CE_WARN, "mptsas bad flash signature!");
13449         goto fail;
13450     }

13452     /*
13453     * Reset the chip
13454     */
13455     rval = mptsas_ioc_reset(mpt, first_time);
13456     if (rval == MPTSAS_RESET_FAIL) {
13457         mptsas_log(mpt, CE_WARN, "hard reset failed!");
13458         goto fail;
13459     }

```

```

13461     if ((rval == MPTSAS_SUCCESS_MUR) && (!first_time)) {
13462         goto mur;
13463     }
13464     /*
13465      * Setup configuration space
13466      */
13467     if (mptsas_config_space_init(mpt) == FALSE) {
13468         mptsas_log(mpt, CE_WARN, "mptsas_config_space_init "
13469             "failed!");
13470         goto fail;
13471     }
13472
13473     /*
13474      * IOC facts can change after a diag reset so all buffers that are
13475      * based on these numbers must be de-allocated and re-allocated. Get
13476      * new IOC facts each time chip is initialized.
13477      */
13478     if (mptsas_ioc_get_facts(mpt) == DDI_FAILURE) {
13479         mptsas_log(mpt, CE_WARN, "mptsas_ioc_get_facts failed");
13480         goto fail;
13481     }
13482
13483     /*
13484      * Now we know chip MSIX capabilities and it's not been done
13485      * previously register interrupts accordingly. Need to know this
13486      * information before allocating the reply frames below.
13487      */
13488     if (mpt->m_intr_cnt == 0) {
13489         if (mptsas_register_intrs(mpt) == FALSE)
13490             goto fail;
13491     }
13492
13493     mpt->m_targets = rehash_create(MPTSAS_TARGET_BUCKET_COUNT,
13494         mptsas_target_addr_hash, mptsas_target_addr_cmp,
13495         mptsas_target_free, sizeof(mptsas_target_t),
13496         offsetof(mptsas_target_t, m_link),
13497         offsetof(mptsas_target_t, m_addr), KM_SLEEP);
13498
13499     if (mptsas_alloc_active_slots(mpt, KM_SLEEP)) {
13500         goto fail;
13501     }
13502     /*
13503      * Allocate request message frames, reply free queue, reply descriptor
13504      * post queue, and reply message frames using latest IOC facts.
13505      */
13506     if (mptsas_alloc_request_frames(mpt) == DDI_FAILURE) {
13507         mptsas_log(mpt, CE_WARN, "mptsas_alloc_request_frames failed");
13508         goto fail;
13509     }
13510     if (mptsas_alloc_sense_bufs(mpt) == DDI_FAILURE) {
13511         mptsas_log(mpt, CE_WARN, "mptsas_alloc_sense_bufs failed");
13512         goto fail;
13513     }
13514     if (mptsas_alloc_free_queue(mpt) == DDI_FAILURE) {
13515         mptsas_log(mpt, CE_WARN, "mptsas_alloc_free_queue failed!");
13516         goto fail;
13517     }
13518     if (mptsas_alloc_post_queue(mpt) == DDI_FAILURE) {
13519         mptsas_log(mpt, CE_WARN, "mptsas_alloc_post_queue failed!");
13520         goto fail;
13521     }
13522     if (mptsas_alloc_reply_frames(mpt) == DDI_FAILURE) {
13523         mptsas_log(mpt, CE_WARN, "mptsas_alloc_reply_frames failed!");
13524         goto fail;
13525     }

```

```

13527 mur:
13528     /*
13529      * Re-Initialize ioc to operational state
13530      */
13531     if (mptsas_ioc_init(mpt) == DDI_FAILURE) {
13532         mptsas_log(mpt, CE_WARN, "mptsas_ioc_init failed");
13533         goto fail;
13534     }
13535
13536     mptsas_alloc_reply_args(mpt);
13537
13538     /*
13539      * Initialize the Reply Free Queue with the physical addresses of our
13540      * reply frames.
13541      */
13542     cookie.dmac_address = mpt->m_reply_frame_dma_addr&0xfffffffful;
13543     for (i = 0; i < mpt->m_max_replies; i++) {
13544         ddi_put32(mpt->m_acc_free_queue_hdl,
13545             &((uint32_t *) (void *) mpt->m_free_queue)[i],
13546             cookie.dmac_address);
13547         cookie.dmac_address += mpt->m_reply_frame_size;
13548     }
13549     (void) ddi_dma_sync(mpt->m_dma_free_queue_hdl, 0, 0,
13550         DDI_DMA_SYNC_FORDEV);
13551
13552     /*
13553      * Initialize the reply free index to one past the last frame on the
13554      * queue. This will signify that the queue is empty to start with.
13555      */
13556     mpt->m_free_index = i;
13557     ddi_put32(mpt->m_datap, &mpt->m_reg->ReplyFreeHostIndex, i);
13558
13559     /*
13560      * Initialize the reply post queue to 0xFFFFFFFF, 0xFFFFFFFF's
13561      * and the indexes to 0.
13562      */
13563     rpqp = mpt->m_rep_post_queues;
13564     for (j = 0; j < mpt->m_post_reply_qcount; j++) {
13565         for (i = 0; i < mpt->m_post_queue_depth; i++) {
13566             ddi_put64(mpt->m_acc_post_queue_hdl,
13567                 &((uint64_t *) (void *) rpqp->rpq_queue)[i],
13568                 0xFFFFFFFFFFFFFFFF);
13569         }
13570         rpqp->rpq_index = 0;
13571         rpqp++;
13572     }
13573     (void) ddi_dma_sync(mpt->m_dma_post_queue_hdl, 0, 0,
13574         DDI_DMA_SYNC_FORDEV);
13575
13576     /*
13577      * Initialize all the reply post queue indexes.
13578      */
13579     for (j = 0; j < mpt->m_post_reply_qcount; j++) {
13580         ddi_put32(mpt->m_datap, &mpt->m_reg->ReplyPostHostIndex,
13581             j << MPI2_RPHI_MSIX_INDEX_SHIFT);
13582     }
13583
13584     /*
13585      * Enable ports
13586      */
13587     if (mptsas_ioc_enable_port(mpt) == DDI_FAILURE) {
13588         mptsas_log(mpt, CE_WARN, "mptsas_ioc_enable_port failed");
13589         goto fail;
13590     }

```



```

13592      /*
13593       * enable events
13594       */
13595      if (mptsas_ioc_enable_event_notification(mpt)) {
13596          mptsas_log(mpt, CE_WARN,
13597              "mptsas_ioc_enable_event_notification failed");
13598          goto fail;
13599      }

13601      /*
13602       * We need checks in attach and these.
13603       * chip_init is called in mult. places
13604       */

13606      if ((mptsas_check_dma_handle(mpt->m_dma_req_frame_hdl) !=
13607          DDI_SUCCESS) ||
13608          (mptsas_check_dma_handle(mpt->m_dma_req_sense_hdl) !=
13609          DDI_SUCCESS) ||
13610          (mptsas_check_dma_handle(mpt->m_dma_reply_frame_hdl) !=
13611          DDI_SUCCESS) ||
13612          (mptsas_check_dma_handle(mpt->m_dma_free_queue_hdl) !=
13613          DDI_SUCCESS) ||
13614          (mptsas_check_dma_handle(mpt->m_dma_post_queue_hdl) !=
13615          DDI_SUCCESS) ||
13616          (mptsas_check_dma_handle(mpt->m_hshk_dma_hdl) !=
13617          DDI_SUCCESS)) {
13618          ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_UNAFFECTED);
13619          goto fail;
13620      }

13622      /* Check all acc handles */
13623      if ((mptsas_check_acc_handle(mpt->m_datap) != DDI_SUCCESS) ||
13624          (mptsas_check_acc_handle(mpt->m_acc_req_frame_hdl) !=
13625          DDI_SUCCESS) ||
13626          (mptsas_check_acc_handle(mpt->m_acc_req_sense_hdl) !=
13627          DDI_SUCCESS) ||
13628          (mptsas_check_acc_handle(mpt->m_acc_reply_frame_hdl) !=
13629          DDI_SUCCESS) ||
13630          (mptsas_check_acc_handle(mpt->m_acc_free_queue_hdl) !=
13631          DDI_SUCCESS) ||
13632          (mptsas_check_acc_handle(mpt->m_acc_post_queue_hdl) !=
13633          DDI_SUCCESS) ||
13634          (mptsas_check_acc_handle(mpt->m_hshk_acc_hdl) !=
13635          DDI_SUCCESS) ||
13636          (mptsas_check_acc_handle(mpt->m_config_handle) !=
13637          DDI_SUCCESS)) {
13638          ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_UNAFFECTED);
13639          goto fail;
13640      }

13642      return (DDI_SUCCESS);

13644 fail:
13645      return (DDI_FAILURE);
13646 }

13648 static int
13649 mptsas_get_pci_cap(mptsas_t *mpt)
13650 {
13651     ushort_t caps_ptr, cap, cap_count;

13653     if (mpt->m_config_handle == NULL)
13654         return (FALSE);
13655     /*
13656      * Check if capabilities list is supported and if so,
13657      * get initial capabilities pointer and clear bits 0,1.

```

```

13658     /*
13659     if (pci_config_get16(mpt->m_config_handle, PCI_CONF_STAT)
13660         & PCI_STAT_CAP) {
13661         caps_ptr = P2ALIGN(pci_config_get8(mpt->m_config_handle,
13662             PCI_CONF_CAP_PTR), 4);
13663     } else {
13664         caps_ptr = PCI_CAP_NEXT_PTR_NULL;
13665     }

13667     /*
13668      * Walk capabilities if supported.
13669      */
13670     for (cap_count = 0; caps_ptr != PCI_CAP_NEXT_PTR_NULL; ) {

13672         /*
13673          * Check that we haven't exceeded the maximum number of
13674          * capabilities and that the pointer is in a valid range.
13675          */
13676         if (++cap_count > 48) {
13677             mptsas_log(mpt, CE_WARN,
13678                 "too many device capabilities.\n");
13679             break;
13680         }
13681         if (caps_ptr < 64) {
13682             mptsas_log(mpt, CE_WARN,
13683                 "capabilities pointer 0x%x out of range.\n",
13684                 caps_ptr);
13685             break;
13686         }

13688         /*
13689          * Get next capability and check that it is valid.
13690          * For now, we only support power management.
13691          */
13692         cap = pci_config_get8(mpt->m_config_handle, caps_ptr);
13693         switch (cap) {
13694             case PCI_CAP_ID_PM:
13695                 mptsas_log(mpt, CE_NOTE,
13696                     "?mptsas3%d supports power management.\n",
13697                     mpt->m_instance);
13698                 mpt->m_options |= MPTSAS_OPT_PM;

13700                 /* Save PMCSR offset */
13701                 mpt->m_pmcsr_offset = caps_ptr + PCI_PMCSR;
13702                 break;
13703             case PCI_CAP_ID_MSI:
13704                 mptsas_log(mpt, CE_NOTE,
13705                     "?mptsas3%d supports MSI.\n",
13706                     mpt->m_instance);
13707                 mpt->m_options |= MPTSAS_OPT_MSI;
13708                 break;
13709             case PCI_CAP_ID_MSI_X:
13710                 mptsas_log(mpt, CE_NOTE,
13711                     "?mptsas3%d supports MSI-X.\n",
13712                     mpt->m_instance);
13713                 mpt->m_options |= MPTSAS_OPT_MSI_X;
13714                 break;
13715             /*
13716              * The following capabilities are valid. Any others
13717              * will cause a message to be logged.
13718              */
13719             case PCI_CAP_ID_VPD:
13720             case PCI_CAP_ID_PCIX:
13721             case PCI_CAP_ID_PCI_E:
13722                 break;
13723             default:

```

```

13724         mptsas_log(mpt, CE_NOTE,
13725                     "?mptsas3%d unrecognized capability "
13726                     "0x%x.\n", mpt->m_instance, cap);
13727         break;
13728     }
13729
13730     /*
13731      * Get next capabilities pointer and clear bits 0,1.
13732      */
13733     caps_ptr = P2ALIGN(pci_config_get8(mpt->m_config_handle,
13734                                     (caps_ptr + PCI_CAP_NEXT_PTR)), 4);
13735 }
13736 return (TRUE);
13737 }
13738
13739 static int
13740 mptsas_init_pm(mptsas_t *mpt)
13741 {
13742     char        pmc_name[16];
13743     char        *pmc[] = {
13744         NULL,
13745         "0=Off (PCI D3 State)",
13746         "3=On (PCI D0 State)",
13747         NULL
13748     };
13749     uint16_t    pmcsr_stat;
13750
13751     /*
13752      * If PCI's capability does not support PM, then don't need
13753      * to register the pm-components
13754      */
13755     if (!(mpt->m_options & MPTSAS_OPT_PM))
13756         return (DDI_SUCCESS);
13757
13758     /*
13759      * If power management is supported by this chip, create
13760      * pm-components property for the power management framework
13761      */
13762     (void) sprintf(pmc_name, "NAME=mptsas3%d", mpt->m_instance);
13763     pmc[0] = pmc_name;
13764     if (ddi_prop_update_string_array(DDI_DEV_T_NONE, mpt->m_dip,
13765                                     "pm-components", pmc, 3) != DDI_PROP_SUCCESS) {
13766         mpt->m_options &= ~MPTSAS_OPT_PM;
13767         mptsas_log(mpt, CE_WARN,
13768                 "mptsas3%d: pm-component property creation failed.",
13769                 mpt->m_instance);
13770         return (DDI_FAILURE);
13771     }
13772
13773     /*
13774      * Power on device.
13775      */
13776     (void) pm_busy_component(mpt->m_dip, 0);
13777     pmcsr_stat = pci_config_get16(mpt->m_config_handle,
13778                                 mpt->m_pmcsr_offset);
13779     if ((pmcsr_stat & PCI_PMCSR_STATE_MASK) != PCI_PMCSR_D0) {
13780         mptsas_log(mpt, CE_WARN, "mptsas3%d: Power up the device",
13781                 mpt->m_instance);
13782         pci_config_put16(mpt->m_config_handle, mpt->m_pmcsr_offset,
13783                         PCI_PMCSR_D0);
13784     }
13785     if (pm_power_has_changed(mpt->m_dip, 0, PM_LEVEL_D0) != DDI_SUCCESS) {
13786         mptsas_log(mpt, CE_WARN, "pm_power_has_changed failed");
13787         return (DDI_FAILURE);
13788     }
13789     mpt->m_power_level = PM_LEVEL_D0;
13790 }

```

```

13790     * Set pm idle delay.
13791     */
13792     mpt->m_pm_idle_delay = ddi_prop_get_int(DDI_DEV_T_ANY,
13793     mpt->m_dip, 0, "mptsas-pm-idle-delay", MPTSAS_PM_IDLE_TIMEOUT);
13794
13795     return (DDI_SUCCESS);
13796 }
13797
13798 static int
13799 mptsas_register_intrs(mptsas_t *mpt)
13800 {
13801     dev_info_t *dip;
13802     int intr_types;
13803
13804     dip = mpt->m_dip;
13805
13806     /* Get supported interrupt types */
13807     if (ddi_intr_get_supported_types(dip, &intr_types) != DDI_SUCCESS) {
13808         mptsas_log(mpt, CE_WARN, "ddi_intr_get_supported_types "
13809                 "failed\n");
13810         return (FALSE);
13811     }
13812
13813     NDBG6(("ddi_intr_get_supported_types() returned: 0x%x", intr_types));
13814
13815     /*
13816      * Try MSIX first.
13817      */
13818     if (mptsas_enable_msix && (intr_types & DDI_INTR_TYPE_MSIX)) {
13819         if (mptsas_add_intrs(mpt, DDI_INTR_TYPE_MSIX) == DDI_SUCCESS) {
13820             NDBG6(("Using MSI-X interrupt type"));
13821             mpt->m_intr_type = DDI_INTR_TYPE_MSIX;
13822             return (TRUE);
13823         }
13824     }
13825
13826     /*
13827      * Try MSI, but fall back to FIXED
13828      */
13829     if (mptsas_enable_msi && (intr_types & DDI_INTR_TYPE_MSI)) {
13830         if (mptsas_add_intrs(mpt, DDI_INTR_TYPE_MSI) == DDI_SUCCESS) {
13831             NDBG6(("Using MSI interrupt type"));
13832             mpt->m_intr_type = DDI_INTR_TYPE_MSI;
13833             return (TRUE);
13834         }
13835     }
13836     if (intr_types & DDI_INTR_TYPE_FIXED) {
13837         if (mptsas_add_intrs(mpt, DDI_INTR_TYPE_FIXED) == DDI_SUCCESS) {
13838             NDBG6(("Using FIXED interrupt type"));
13839             mpt->m_intr_type = DDI_INTR_TYPE_FIXED;
13840             return (TRUE);
13841         } else {
13842             NDBG6(("FIXED interrupt registration failed"));
13843             return (FALSE);
13844         }
13845     }
13846
13847     return (FALSE);
13848 }
13849
13850 static void
13851 mptsas_unregister_intrs(mptsas_t *mpt)
13852 {
13853     mptsas_rem_intrs(mpt);
13854 }

```

```

13856 /*
13857  * mptsas_add_intrs:
13858  *
13859  * Register FIXED or MSI interrupts.
13860  */
13861 static int
13862 mptsas_add_intrs(mptsas_t *mpt, int intr_type)
13863 {
13864     dev_info_t      *dip = mpt->m_dip;
13865     int              avail, actual, count = 0;
13866     int              i, flag, ret;

13868     NDBG6(("mptsas_add_intrs: interrupt type 0x%x", intr_type));

13870     /* Get number of interrupts */
13871     ret = ddi_intr_get_nintrs(dip, intr_type, &count);
13872     if ((ret != DDI_SUCCESS) || (count == 0)) {
13873         mptsas_log(mpt, CE_WARN, "ddi_intr_get_nintrs() failed, "
13874             "ret %d count %d\n", ret, count);

13876         return (DDI_FAILURE);
13877     }

13879     /* Get number of interrupts available to this device */
13880     ret = ddi_intr_get_navail(dip, intr_type, &avail);
13881     if ((ret != DDI_SUCCESS) || (avail == 0)) {
13882         mptsas_log(mpt, CE_WARN, "ddi_intr_get_navail() failed, "
13883             "ret %d avail %d\n", ret, avail);

13885         return (DDI_FAILURE);
13886     }

13888     if (count < avail) {
13889         mptsas_log(mpt, CE_NOTE, "ddi_intr_get_nvail returned %d, "
13890             "navail() returned %d", count, avail);
13891     }

13893     NDBG6(("mptsas_add_intrs: count %d, avail %d", count, avail));

13895     if (intr_type == DDI_INTR_TYPE_MSIX) {
13896         if (!mptsas3_max_msix_intrs) {
13897             return (DDI_FAILURE);
13898         }
13899     }

13900     /*
13901      * Restrict the number of interrupts, firstly by
13902      * the number returned from the IOCInfo, then by
13903      * overall restriction.
13904      */
13905     if (avail > mpt->m_max_msix_vectors) {
13906         avail = mpt->m_max_msix_vectors;
13907         mpt->m_max_msix_vectors = 1;
13908         NDBG6(("mptsas_add_intrs: mmmv avail %d", avail));
13909     }
13910     if (avail > mptsas3_max_msix_intrs) {
13911         avail = mptsas3_max_msix_intrs;
13912         NDBG6(("mptsas_add_intrs: m3mmi avail %d", avail));
13913     }
13914 }
13915 if (intr_type == DDI_INTR_TYPE_MSI) {
13916     NDBG6(("mptsas_add_intrs: MSI avail %d", avail));
13917     avail = 1;
13918 }

13920 /* Allocate an array of interrupt handles */
13921 mpt->m_intr_size = avail * sizeof (ddi_intr_handle_t);

```

```

13922     mpt->m_htable = kmem_alloc(mpt->m_intr_size, KM_SLEEP);

13924     flag = DDI_INTR_ALLOC_NORMAL;

13926     /* call ddi_intr_alloc() */
13927     ret = ddi_intr_alloc(dip, mpt->m_htable, intr_type, 0,
13928         avail, &actual, flag);

13930     if ((ret != DDI_SUCCESS) || (actual == 0)) {
13931         mptsas_log(mpt, CE_WARN, "ddi_intr_alloc() failed, ret %d\n",
13932             ret);
13933         kmem_free(mpt->m_htable, mpt->m_intr_size);
13934         return (DDI_FAILURE);
13935     }

13937     NDBG6(("mptsas_add_intrs: actual %d, avail %d", actual, avail));
13938     /* use interrupt count returned or abort? */
13939     if (actual < avail) {
13940         mptsas_log(mpt, CE_NOTE,
13941             "Interrupts requested: %d, received: %d\n",
13942             avail, actual);
13943     }

13945     /*
13946      * Get priority for first msi, assume remaining are all the same
13947      */
13948     if ((ret = ddi_intr_get_pri(mpt->m_htable[0],
13949         &mpt->m_intr_pri)) != DDI_SUCCESS) {
13950         mptsas_log(mpt, CE_WARN, "ddi_intr_get_pri() failed %d\n", ret);

13952         /* Free already allocated intr */
13953         for (i = 0; i < actual; i++) {
13954             (void) ddi_intr_free(mpt->m_htable[i]);
13955         }

13957         kmem_free(mpt->m_htable, mpt->m_intr_size);
13958         return (DDI_FAILURE);
13959     }

13961     /* Test for high level mutex */
13962     if (mpt->m_intr_pri >= ddi_intr_get_hilevel_pri()) {
13963         mptsas_log(mpt, CE_WARN, "mptsas_add_intrs: "
13964             "Hi level interrupt not supported\n");

13966         /* Free already allocated intr */
13967         for (i = 0; i < actual; i++) {
13968             (void) ddi_intr_free(mpt->m_htable[i]);
13969         }

13971         kmem_free(mpt->m_htable, mpt->m_intr_size);
13972         return (DDI_FAILURE);
13973     }

13975     /* Call ddi_intr_add_handler() */
13976     for (i = 0; i < actual; i++) {
13977         if ((ret = ddi_intr_add_handler(mpt->m_htable[i], mptsas_intr,
13978             (caddr_t)mpt, (caddr_t)(uintptr_t)i)) != DDI_SUCCESS) {
13979             mptsas_log(mpt, CE_WARN, "ddi_intr_add_handler() "
13980                 "failed %d\n", ret);

13982             /* Free already allocated intr */
13983             for (i = 0; i < actual; i++) {
13984                 (void) ddi_intr_free(mpt->m_htable[i]);
13985             }

13987             kmem_free(mpt->m_htable, mpt->m_intr_size);

```

```

13988         return (DDI_FAILURE);
13989     }
13990 }

13992 if ((ret = ddi_intr_get_cap(mpt->m_htable[0], &mpt->m_intr_cap))
13993     != DDI_SUCCESS) {
13994     mptsas_log(mpt, CE_WARN, "ddi_intr_get_cap() failed %d\n", ret);

13996     /* Free already allocated intr */
13997     for (i = 0; i < actual; i++) {
13998         (void) ddi_intr_free(mpt->m_htable[i]);
13999     }

14001     kmem_free(mpt->m_htable, mpt->m_intr_size);
14002     return (DDI_FAILURE);
14003 }

14005 mpt->m_intr_cnt = actual;

14007 /*
14008  * Enable interrupts
14009  */
14010 if (mpt->m_intr_cap & DDI_INTR_FLAG_BLOCK) {
14011     /* Call ddi_intr_block_enable() for MSI interrupts */
14012     (void) ddi_intr_block_enable(mpt->m_htable, mpt->m_intr_cnt);
14013 } else {
14014     /* Call ddi_intr_enable for MSI or FIXED interrupts */
14015     for (i = 0; i < mpt->m_intr_cnt; i++) {
14016         (void) ddi_intr_enable(mpt->m_htable[i]);
14017     }
14018 }

14020 switch (intr_type) {
14021 case DDI_INTR_TYPE_MSIX:
14022     mptsas_log(mpt, CE_NOTE, "?Using %d MSI-X interrupt(s) "
14023         "(Available sys %d, mpt %d, Requested %d)\n",
14024         actual, count, mpt->m_max_msix_vectors, avail);
14025     break;
14026 case DDI_INTR_TYPE_MSI:
14027     mptsas_log(mpt, CE_NOTE, "Using single MSI interrupt\n");
14028     break;
14029 case DDI_INTR_TYPE_FIXED:
14030     default:
14031         mptsas_log(mpt, CE_NOTE, "Using single fixed interrupt\n");
14032         break;
14033 }

14035 return (DDI_SUCCESS);
14036 }

14038 /*
14039  * mptsas_rem_intrs:
14040  *
14041  * Unregister FIXED or MSI interrupts
14042  */
14043 static void
14044 mptsas_rem_intrs(mptsas_t *mpt)
14045 {
14046     int    i;

14048     NDBG6(("mptsas_rem_intrs"));

14050     /* Disable all interrupts */
14051     if (mpt->m_intr_cap & DDI_INTR_FLAG_BLOCK) {
14052         /* Call ddi_intr_block_disable() */
14053         (void) ddi_intr_block_disable(mpt->m_htable, mpt->m_intr_cnt);

```

```

14054     } else {
14055         for (i = 0; i < mpt->m_intr_cnt; i++) {
14056             (void) ddi_intr_disable(mpt->m_htable[i]);
14057         }
14058     }

14060     /* Call ddi_intr_remove_handler() */
14061     for (i = 0; i < mpt->m_intr_cnt; i++) {
14062         (void) ddi_intr_remove_handler(mpt->m_htable[i]);
14063         (void) ddi_intr_free(mpt->m_htable[i]);
14064     }
14065     kmem_free(mpt->m_htable, mpt->m_intr_size);
14066     mpt->m_intr_cnt = 0;
14067 }

14069 /*
14070  * The IO fault service error handling callback function
14071  */
14072 /*ARGSUSED*/
14073 static int
14074 mptsas_fm_error_cb(dev_info_t *dip, ddi_fm_error_t *err, const void *impl_data)
14075 {
14076     /*
14077      * as the driver can always deal with an error in any dma or
14078      * access handle, we can just return the fme_status value.
14079      */
14080     pci_ereport_post(dip, err, NULL);
14081     return (err->fme_status);
14082 }

14084 /*
14085  * mptsas_fm_init - initialize fma capabilities and register with IO
14086  *                  fault services.
14087  */
14088 static void
14089 mptsas_fm_init(mptsas_t *mpt)
14090 {
14091     /*
14092      * Need to change iblock to priority for new MSI intr
14093      */
14094     ddi_iblock_cookie_t    fm_ibc;

14096     /* Only register with IO Fault Services if we have some capability */
14097     if (mpt->m_fm_capabilities) {
14098         /* Adjust access and dma attributes for FMA */
14099         mpt->m_reg_acc_attr.devacc_attr.access = DDI_FLAGERR_ACC;
14100         mpt->m_msg_dma_attr.dma_attr.flags |= DDI_DMA_FLAGERR;
14101         mpt->m_io_dma_attr.dma_attr.flags |= DDI_DMA_FLAGERR;

14103         /*
14104          * Register capabilities with IO Fault Services.
14105          * mpt->m_fm_capabilities will be updated to indicate
14106          * capabilities actually supported (not requested.)
14107          */
14108         ddi_fm_init(mpt->m_dip, &mpt->m_fm_capabilities, &fm_ibc);

14110         /*
14111          * Initialize pci ereport capabilities if ereport
14112          * capable (should always be.)
14113          */
14114         if (DDI_FM_EREPORT_CAP(mpt->m_fm_capabilities) ||
14115             DDI_FM_ERRCB_CAP(mpt->m_fm_capabilities)) {
14116             pci_ereport_setup(mpt->m_dip);
14117         }

14119         /*

```

```

14120         * Register error callback if error callback capable.
14121         */
14122         if (DDI_FM_ERRCB_CAP(mpt->m_fm_capabilities)) {
14123             ddi_fm_handler_register(mpt->m_dip,
14124                                     mptsas_fm_error_cb, (void *) mpt);
14125         }
14126     }
14127 }

14129 /*
14130  * mptsas_fm_fini - Releases fma capabilities and un-registers with IO
14131  * fault services.
14132  */
14133 static void
14134 mptsas_fm_fini(mptsas_t *mpt)
14135 {
14136     /* Only unregister FMA capabilities if registered */
14137     if (mpt->m_fm_capabilities) {
14138
14139         /*
14140          * Un-register error callback if error callback capable.
14141          */
14142         if (DDI_FM_ERRCB_CAP(mpt->m_fm_capabilities)) {
14143             ddi_fm_handler_unregister(mpt->m_dip);
14144         }
14145
14146         /*
14147          * Release any resources allocated by pci_ereport_setup()
14148          */
14149         if (DDI_FM_EREPORCAP(mpt->m_fm_capabilities) ||
14150             DDI_FM_ERRCB_CAP(mpt->m_fm_capabilities)) {
14151             pci_ereport_tear_down(mpt->m_dip);
14152         }
14153
14154         /* Unregister from IO Fault Services */
14155         ddi_fm_fini(mpt->m_dip);
14156
14157         /* Adjust access and dma attributes for FMA */
14158         mpt->m_reg_acc_attr.devacc_attr.access = DDI_DEFAULT_ACC;
14159         mpt->m_msg_dma_attr.dma_attr.flags &= ~DDI_DMA_FLAGERR;
14160         mpt->m_io_dma_attr.dma_attr.flags &= ~DDI_DMA_FLAGERR;
14161     }
14162 }

14163 int
14164 mptsas_check_acc_handle(ddi_acc_handle_t handle)
14165 {
14166     ddi_fm_error_t de;
14167
14168     if (handle == NULL)
14169         return (DDI_FAILURE);
14170     ddi_fm_acc_err_get(handle, &de, DDI_FME_VER0);
14171     return (de.fme_status);
14172 }

14173 int
14174 mptsas_check_dma_handle(ddi_dma_handle_t handle)
14175 {
14176     ddi_fm_error_t de;
14177
14178     if (handle == NULL)
14179         return (DDI_FAILURE);

```

```

14186     ddi_fm_dma_err_get(handle, &de, DDI_FME_VER0);
14187     return (de.fme_status);
14188 }

14190 void
14191 mptsas_fm_ereport(mptsas_t *mpt, char *detail)
14192 {
14193     uint64_t ena;
14194     char buf[FM_MAX_CLASS];
14195
14196     (void) snprintf(buf, FM_MAX_CLASS, "%s.%s", DDI_FM_DEVICE, detail);
14197     ena = fm_ena_generate(0, FM_ENA_FMT1);
14198     if (DDI_FM_EREPORCAP(mpt->m_fm_capabilities)) {
14199         ddi_fm_ereport_post(mpt->m_dip, buf, ena, DDI_NOSLEEP,
14200                             FM_VERSION, DATA_TYPE_UINT8, FM_EREPORCAP, NULL);
14201     }
14202 }

14204 static int
14205 mptsas_get_target_device_info(mptsas_t *mpt, uint32_t page_address,
14206                               uint16_t *dev_handle, mptsas_target_t **pptgt)
14207 {
14208     int rval;
14209     uint32_t dev_info;
14210     uint64_t sas_wnn;
14211     mptsas_phymask_t phymask;
14212     uint8_t physport, phynum, config, disk;
14213     uint64_t devicename;
14214     uint16_t pdev_hdl;
14215     mptsas_target_t *tmp_tgt = NULL;
14216     uint16_t bay_num, enclosure, io_flags;
14217
14218     ASSERT(*pptgt == NULL);
14219
14220     rval = mptsas_get_sas_device_page0(mpt, page_address, dev_handle,
14221                                         &sas_wnn, &dev_info, &physport, &phynum, &pdev_hdl,
14222                                         &bay_num, &enclosure, &io_flags);
14223     if (rval != DDI_SUCCESS) {
14224         rval = DEV_INFO_FAIL_PAGE0;
14225         return (rval);
14226     }
14227
14228     if ((dev_info & (MPI2_SAS_DEVICE_INFO_SSP_TARGET |
14229                     MPI2_SAS_DEVICE_INFO_SATA_DEVICE |
14230                     MPI2_SAS_DEVICE_INFO_ATAPI_DEVICE)) == NULL) {
14231         rval = DEV_INFO_WRONG_DEVICE_TYPE;
14232         return (rval);
14233     }
14234
14235     /*
14236      * Check if the dev handle is for a Phys Disk. If so, set return value
14237      * and exit. Don't add Phys Disks to hash.
14238      */
14239     for (config = 0; config < mpt->m_num_raid_configs; config++) {
14240         for (disk = 0; disk < MPTSAS_MAX_DISKS_IN_CONFIG; disk++) {
14241             if (*dev_handle == mpt->m_raidconfig[config].
14242                 m_physdisk_devhdl[disk]) {
14243                 rval = DEV_INFO_PHYS_DISK;
14244                 return (rval);
14245             }
14246         }
14247     }
14248
14249     /*
14250      * Get SATA Device Name from SAS device page0 for
14251      * sata device, if device name doesn't exist, set mta_wnn to

```

```

14252      * 0 for direct attached SATA. For the device behind the expander
14253      * we still can use STP address assigned by expander.
14254      */
14255      if (dev_info & (MPI2_SAS_DEVICE_INFO_SATA_DEVICE |
14256        MPI2_SAS_DEVICE_INFO_ATAPI_DEVICE)) {
14257          mutex_exit(&mpt->m_mutex);
14258          /* alloc a tmp_tgt to send the cmd */
14259          tmp_tgt = kmem_zalloc(sizeof (struct mptsas_target),
14260            KM_SLEEP);
14261          tmp_tgt->m_devhdl = *dev_handle;
14262          tmp_tgt->m_deviceinfo = dev_info;
14263          tmp_tgt->m_qfull_retries = QFULL_RETRIES;
14264          tmp_tgt->m_qfull_retry_interval =
14265            drv_usectohz(QFULL_RETRY_INTERVAL * 1000);
14266          tmp_tgt->m_t_throttle = MAX_THROTTLE;
14267          mutex_init(&tmp_tgt->m_t_mutex, NULL, MUTEX_DRIVER, NULL);
14268          devicename = mptsas_get_sata_guid(mpt, tmp_tgt, 0);
14269          mutex_destroy(&tmp_tgt->m_t_mutex);
14270          kmem_free(tmp_tgt, sizeof (struct mptsas_target));
14271          mutex_enter(&mpt->m_mutex);
14272          if (devicename != 0 && (((devicename >> 56) & 0xf0) == 0x50)) {
14273              sas_wwn = devicename;
14274          } else if (dev_info & MPI2_SAS_DEVICE_INFO_DIRECT_ATTACH) {
14275              sas_wwn = 0;
14276          }
14277      }

14279      phymask = mptsas_physport_to_phymask(mpt, physport);
14280      *pptgt = mptsas_tgt_alloc(mpt, *dev_handle, sas_wwn,
14281        dev_info, phymask, phynum);
14282      if (*pptgt == NULL) {
14283          mptsas_log(mpt, CE_WARN, "Failed to allocated target"
14284            "structure!");
14285          rval = DEV_INFO_FAIL_ALLOC;
14286          return (rval);
14287      }
14288      (*pptgt)->m_io_flags = io_flags;
14289      (*pptgt)->m_enclosure = enclosure;
14290      (*pptgt)->m_slot_num = bay_num;
14291      return (DEV_INFO_SUCCESS);
14292  }

14294  uint64_t
14295  mptsas_get_sata_guid(mptsas_t *mpt, mptsas_target_t *ptgt, int lun)
14296  {
14297      uint64_t      sata_guid = 0, *pwwn = NULL;
14298      int            target = ptgt->m_devhdl;
14299      uchar_t        *inq83 = NULL;
14300      int            inq83_len = 0xFF;
14301      uchar_t        *dblk = NULL;
14302      int            inq83_retry = 3;
14303      int            rval = DDI_FAILURE;

14305      inq83 = kmem_zalloc(inq83_len, KM_SLEEP);

14307  inq83_retry:
14308      rval = mptsas_inquiry(mpt, ptgt, lun, 0x83, inq83,
14309        inq83_len, NULL, 1);
14310      if (rval != DDI_SUCCESS) {
14311          mptsas_log(mpt, CE_WARN, "!mptsas request inquiry page "
14312            "0x83 for target:%x, lun:%x failed!", target, lun);
14313          goto out;
14314      }
14315      /* According to SAT2, the first descriptor is logic unit name */
14316      dblk = &inq83[4];
14317      if ((dblk[1] & 0x30) != 0) {

```

```

14318          mptsas_log(mpt, CE_WARN, "!Descriptor is not lun associated.");
14319          goto out;
14320      }
14321      pwwn = (uint64_t *) (void *) (&dblk[4]);
14322      if ((dblk[4] & 0xf0) == 0x50) {
14323          sata_guid = BE_64(*pwwn);
14324          goto out;
14325      } else if (dblk[4] == 'A') {
14326          NDBG20(("SATA drive has no NAA format GUID."));
14327          goto out;
14328      } else {
14329          /* The data is not ready, wait and retry */
14330          inq83_retry--;
14331          if (inq83_retry <= 0) {
14332              goto out;
14333          }
14334          NDBG20(("The GUID is not ready, retry..."));
14335          delay(1 * drv_usectohz(1000000));
14336          goto inq83_retry;
14337      }
14338  out:
14339      kmem_free(inq83, inq83_len);
14340      return (sata_guid);
14341  }

14343  static int
14344  mptsas_inquiry(mptsas_t *mpt, mptsas_target_t *ptgt, int lun, uchar_t page,
14345    unsigned char *buf, int len, int *reallen, uchar_t evpd)
14346  {
14347      uchar_t        cdb[CDB_GROUP0];
14348      struct scsi_address ap;
14349      struct buf      *data_bp = NULL;
14350      int            resid = 0;
14351      int            ret = DDI_FAILURE;

14353      ASSERT(len <= 0xffff);

14355      ap.a_target = MPTSAS_INVALID_DEVDHL;
14356      ap.a_lun = (uchar_t) (lun);
14357      ap.a_hba_tran = mpt->m_tran;

14359      data_bp = scsi_alloc_consistent_buf(&ap,
14360        (struct buf *) NULL, len, B_READ, NULL_FUNC, NULL);
14361      if (data_bp == NULL) {
14362          return (ret);
14363      }
14364      bzero(cdb, CDB_GROUP0);
14365      cdb[0] = SCMD_INQUIRY;
14366      cdb[1] = evpd;
14367      cdb[2] = page;
14368      cdb[3] = (len & 0xff00) >> 8;
14369      cdb[4] = (len & 0x00ff);
14370      cdb[5] = 0;

14372      ret = mptsas_send_scsi_cmd(mpt, &ap, ptgt, &cdb[0], CDB_GROUP0, data_bp,
14373        &resid);
14374      if (ret == DDI_SUCCESS) {
14375          if (reallen) {
14376              *reallen = len - resid;
14377          }
14378          bcopy((caddr_t) data_bp->b_un.b_addr, buf, len);
14379      }
14380      if (data_bp) {
14381          scsi_free_consistent_buf(data_bp);
14382      }
14383      return (ret);

```

```

14384 }

14386 static int
14387 mptsas_send_scsi_cmd(mptsas_t *mpt, struct scsi_address *ap,
14388     mptsas_target_t *ptgt, uchar_t *cdb, int cdblen, struct buf *data_bp,
14389     int *resid)
14390 {
14391     struct scsi_pkt      *pktp = NULL;
14392     scsi_hba_tran_t      *tran_clone = NULL;
14393     mptsas_tgt_private_t *tgt_private = NULL;
14394     int                  ret = DDI_FAILURE;

14396     /*
14397      * scsi_hba_tran_t->tran_tgt_private is used to pass the address
14398      * information to scsi_init_pkt, allocate a scsi_hba_tran structure
14399      * to simulate the cmds from sd
14400      */
14401     tran_clone = kmem_alloc(
14402         sizeof (scsi_hba_tran_t), KM_SLEEP);
14403     if (tran_clone == NULL) {
14404         goto out;
14405     }
14406     bcopy((caddr_t)mpt->m_tran,
14407         (caddr_t)tran_clone, sizeof (scsi_hba_tran_t));
14408     tgt_private = kmem_alloc(
14409         sizeof (mptsas_tgt_private_t), KM_SLEEP);
14410     if (tgt_private == NULL) {
14411         goto out;
14412     }
14413     tgt_private->t_lun = ap->a_lun;
14414     tgt_private->t_private = ptgt;
14415     tran_clone->tran_tgt_private = tgt_private;
14416     ap->a_hba_tran = tran_clone;

14418     pktp = scsi_init_pkt(ap, (struct scsi_pkt *)NULL,
14419         data_bp, cdblen, sizeof (struct scsi_arq_status),
14420         0, PKT_CONSISTENT, NULL, NULL);
14421     if (pktp == NULL) {
14422         goto out;
14423     }
14424     bcopy(cdb, pktp->pkt_cdbp, cdblen);
14425     pktp->pkt_flags = FLAG_NOPARITY;
14426     pktp->pkt_time = mptsas_scsi_pkt_time;
14427     if (scsi_poll(pktp) < 0) {
14428         goto out;
14429     }
14430     if (((struct scsi_status *)pktp->pkt_scbp)->sts_chk) {
14431         goto out;
14432     }
14433     if (resid != NULL) {
14434         *resid = pktp->pkt_resid;
14435     }

14437     ret = DDI_SUCCESS;
14438 out:
14439     if (pktp) {
14440         scsi_destroy_pkt(pktp);
14441     }
14442     if (tran_clone) {
14443         kmem_free(tran_clone, sizeof (scsi_hba_tran_t));
14444     }
14445     if (tgt_private) {
14446         kmem_free(tgt_private, sizeof (mptsas_tgt_private_t));
14447     }
14448     return (ret);
14449 }

```

```

14450 static int
14451 mptsas_parse_address(char *name, uint64_t *wwid, uint8_t *phy, int *lun)
14452 {
14453     char      *cp = NULL;
14454     char      *ptr = NULL;
14455     size_t    s = 0;
14456     char      *wwid_str = NULL;
14457     char      *lun_str = NULL;
14458     long      lunnum;
14459     long      phyid = -1;
14460     int       rc = DDI_FAILURE;

14462     ptr = name;
14463     ASSERT(ptr[0] == 'w' || ptr[0] == 'p');
14464     ptr++;
14465     if ((cp = strchr(ptr, ',')) == NULL) {
14466         return (DDI_FAILURE);
14467     }

14469     wwid_str = kmem_zalloc(SCSI_MAXNAMELEN, KM_SLEEP);
14470     s = (uintptr_t)cp - (uintptr_t)ptr;

14472     bcopy(ptr, wwid_str, s);
14473     wwid_str[s] = '\0';

14475     ptr = ++cp;

14477     if ((cp = strchr(ptr, '\0')) == NULL) {
14478         goto out;
14479     }
14480     lun_str = kmem_zalloc(SCSI_MAXNAMELEN, KM_SLEEP);
14481     s = (uintptr_t)cp - (uintptr_t)ptr;

14483     bcopy(ptr, lun_str, s);
14484     lun_str[s] = '\0';

14486     if (name[0] == 'p') {
14487         rc = ddi_strtol(wwid_str, NULL, 0x10, &phyid);
14488     } else {
14489         rc = scsi_wwnstr_to_wwn(wwid_str, wwid);
14490     }
14491     if (rc != DDI_SUCCESS)
14492         goto out;

14494     if (phyid != -1) {
14495         ASSERT(phyid < MPTSAS_MAX_PHYS);
14496         *phy = (uint8_t)phyid;
14497     }
14498     rc = ddi_strtol(lun_str, NULL, 0x10, &lunnum);
14499     if (rc != 0)
14500         goto out;

14502     *lun = (int)lunnum;
14503     rc = DDI_SUCCESS;
14504 out:
14505     if (wwid_str)
14506         kmem_free(wwid_str, SCSI_MAXNAMELEN);
14507     if (lun_str)
14508         kmem_free(lun_str, SCSI_MAXNAMELEN);

14510     return (rc);
14511 }

14513 /*
14514  * mptsas_parse_smp_name() is to parse sas wwn string
14515  * which format is "wwwn"

```

```

14516 */
14517 static int
14518 mptsas_parse_smp_name(char *name, uint64_t *wwn)
14519 {
14520     char    *ptr = name;
14521
14522     if (*ptr != 'w') {
14523         return (DDI_FAILURE);
14524     }
14525
14526     ptr++;
14527     if (scsi_wwnstr_to_wwn(ptr, wwn)) {
14528         return (DDI_FAILURE);
14529     }
14530     return (DDI_SUCCESS);
14531 }
14532
14533 static int
14534 mptsas_bus_config(dev_info_t *pdip, uint_t flag,
14535     ddi_bus_config_op_t op, void *arg, dev_info_t **childp)
14536 {
14537     int        ret = NDI_FAILURE;
14538     int        circ = 0;
14539     int        circ1 = 0;
14540     mptsas_t   *mpt;
14541     char        *ptr = NULL;
14542     char        *devnm = NULL;
14543     uint64_t    wwid = 0;
14544     uint8_t     phy = 0xFF;
14545     int         lun = 0;
14546     uint_t      mflags = flag;
14547     int         bconfig = TRUE;
14548
14549     if (scsi_hba_iport_unit_address(pdip) == 0) {
14550         return (DDI_FAILURE);
14551     }
14552
14553     mpt = DIP2MPT(pdip);
14554     if (!mpt) {
14555         return (DDI_FAILURE);
14556     }
14557     /*
14558      * Hold the nexus across the bus_config
14559      */
14560     ndi_devi_enter(scsi_vhci_dip, &circ);
14561     ndi_devi_enter(pdip, &circ1);
14562     switch (op) {
14563     case BUS_CONFIG_ONE:
14564         /* parse wwid/target name out of name given */
14565         if ((ptr = strchr((char *)arg, '@')) == NULL) {
14566             ret = NDI_FAILURE;
14567             break;
14568         }
14569         ptr++;
14570         if (strncmp((char *)arg, "smp", 3) == 0) {
14571             /*
14572              * This is a SMP target device
14573              */
14574             ret = mptsas_parse_smp_name(ptr, &wwid);
14575             if (ret != DDI_SUCCESS) {
14576                 ret = NDI_FAILURE;
14577                 break;
14578             }
14579             ret = mptsas_config_smp(pdip, wwid, childp);
14580         } else if ((ptr[0] == 'w') || (ptr[0] == 'p')) {
14581             /*

```

```

14582         * OBP could pass down a non-canonical form
14583         * bootpath without LUN part when LUN is 0.
14584         * So driver need adjust the string.
14585         */
14586         if (strchr(ptr, ',') == NULL) {
14587             devnm = kmem_zalloc(SCSI_MAXNAMELEN, KM_SLEEP);
14588             (void) sprintf(devnm, "%s,0", (char *)arg);
14589             ptr = strchr(devnm, '@');
14590             ptr++;
14591         }
14592
14593     /*
14594      * The device path is wwid format and the device
14595      * is not SMP target device.
14596      */
14597     ret = mptsas_parse_address(ptr, &wwid, &phy, &lun);
14598     if (ret != DDI_SUCCESS) {
14599         ret = NDI_FAILURE;
14600         break;
14601     }
14602     *childp = NULL;
14603     if (ptr[0] == 'w') {
14604         ret = mptsas_config_one_addr(pdip, wwid,
14605             lun, childp);
14606     } else if (ptr[0] == 'p') {
14607         ret = mptsas_config_one_phy(pdip, phy, lun,
14608             childp);
14609     }
14610
14611     /*
14612      * If this is CD/DVD device in OBP path, the
14613      * ndi_busop_bus_config can be skipped as config one
14614      * operation is done above.
14615      */
14616     if ((ret == NDI_SUCCESS) && (*childp != NULL) &&
14617         (strcmp(ddd_node_name(*childp), "cdrom") == 0) &&
14618         (strncmp((char *)arg, "disk", 4) == 0)) {
14619         bconfig = FALSE;
14620         ndi_hold_devi(*childp);
14621     }
14622     } else {
14623         ret = NDI_FAILURE;
14624         break;
14625     }
14626
14627     /*
14628      * DDI group instructed us to use this flag.
14629      */
14630     mflags |= NDI_MDI_FALLBACK;
14631     break;
14632 case BUS_CONFIG_DRIVER:
14633 case BUS_CONFIG_ALL:
14634     mptsas_config_all(pdip);
14635     ret = NDI_SUCCESS;
14636     break;
14637 }
14638
14639 if ((ret == NDI_SUCCESS) && bconfig) {
14640     ret = ndi_busop_bus_config(pdip, mflags, op,
14641         (devnm == NULL) ? arg : devnm, childp, 0);
14642 }
14643
14644 ndi_devi_exit(pdip, circ1);
14645 ndi_devi_exit(scsi_vhci_dip, circ);
14646 if (devnm != NULL)
14647     kmem_free(devnm, SCSI_MAXNAMELEN);

```



```

14648     return (ret);
14649 }

14651 static int
14652 mptsas_probe_lun(dev_info_t *pdip, int lun, dev_info_t **dip,
14653 mptsas_target_t *ptgt)
14654 {
14655     int rval = DDI_FAILURE;
14656     struct scsi_inquiry *sd_inq = NULL;
14657     mptsas_t *mpt = DIP2MPT(pdip);

14659     sd_inq = (struct scsi_inquiry *)kmem_alloc(SUN_INQSIZE, KM_SLEEP);

14661     rval = mptsas_inquiry(mpt, ptgt, lun, 0, (uchar_t *)sd_inq,
14662     SUN_INQSIZE, 0, (uchar_t)0);

14664     if ((rval == DDI_SUCCESS) && MPTSAS_VALID_LUN(sd_inq)) {
14665         rval = mptsas_create_lun(pdip, sd_inq, dip, ptgt, lun);
14666     } else {
14667         rval = DDI_FAILURE;
14668     }

14670     kmem_free(sd_inq, SUN_INQSIZE);
14671     return (rval);
14672 }

14674 static int
14675 mptsas_config_one_addr(dev_info_t *pdip, uint64_t sasaddr, int lun,
14676 dev_info_t **lundip)
14677 {
14678     int rval;
14679     mptsas_t *mpt = DIP2MPT(pdip);
14680     int phymask;
14681     mptsas_target_t *ptgt = NULL;

14683     /*
14684      * Get the physical port associated to the iport
14685      */
14686     phymask = ddi_prop_get_int(DDI_DEV_T_ANY, pdip, 0,
14687     "phymask", 0);

14689     ptgt = mptsas_wwid_to_ptgt(mpt, phymask, sasaddr);
14690     if (ptgt == NULL) {
14691         /*
14692          * didn't match any device by searching
14693          */
14694         return (DDI_FAILURE);
14695     }
14696     /*
14697      * If the LUN already exists and the status is online,
14698      * we just return the pointer to dev_info_t directly.
14699      * For the mdi_pathinfo node, we'll handle it in
14700      * mptsas_create_virt_lun()
14701      * TODO should be also in mptsas_handle_dr
14702      */

14704     *lundip = mptsas_find_child_addr(pdip, sasaddr, lun);
14705     if (*lundip != NULL) {
14706         /*
14707          * TODO Another senario is, we hotplug the same disk
14708          * on the same slot, the devhdl changed, is this
14709          * possible?
14710          * tgt_private->t_private != ptgt
14711          */
14712         if (sasaddr != ptgt->m_addr.mta_wwn) {
14713

```

```

14714         * The device has changed although the devhdl is the
14715         * same (Enclosure mapping mode, change drive on the
14716         * same slot)
14717         */
14718         return (DDI_FAILURE);
14719     }
14720     return (DDI_SUCCESS);
14721 }

14723     if (phymask == 0) {
14724         /*
14725          * Configure IR volume
14726          */
14727         rval = mptsas_config_raid(pdip, ptgt->m_devhdl, lundip);
14728         return (rval);
14729     }
14730     rval = mptsas_probe_lun(pdip, lun, lundip, ptgt);

14732     return (rval);
14733 }

14735 static int
14736 mptsas_config_one_phy(dev_info_t *pdip, uint8_t phy, int lun,
14737 dev_info_t **lundip)
14738 {
14739     int rval;
14740     mptsas_t *mpt = DIP2MPT(pdip);
14741     mptsas_phymask_t phymask;
14742     mptsas_target_t *ptgt = NULL;

14744     /*
14745      * Get the physical port associated to the iport
14746      */
14747     phymask = (mptsas_phymask_t)ddi_prop_get_int(DDI_DEV_T_ANY, pdip, 0,
14748     "phymask", 0);

14750     ptgt = mptsas_phy_to_tgt(mpt, phymask, phy);
14751     if (ptgt == NULL) {
14752         /*
14753          * didn't match any device by searching
14754          */
14755         return (DDI_FAILURE);
14756     }

14758     /*
14759      * If the LUN already exists and the status is online,
14760      * we just return the pointer to dev_info_t directly.
14761      * For the mdi_pathinfo node, we'll handle it in
14762      * mptsas_create_virt_lun().
14763      */

14765     *lundip = mptsas_find_child_phy(pdip, phy);
14766     if (*lundip != NULL) {
14767         return (DDI_SUCCESS);
14768     }

14770     rval = mptsas_probe_lun(pdip, lun, lundip, ptgt);

14772     return (rval);
14773 }

14775 static int
14776 mptsas_retrieve_lundata(int lun_cnt, uint8_t *buf, uint16_t *lun_num,
14777 uint8_t *lun_addr_type)
14778 {
14779     uint32_t lun_idx = 0;

```

```

14781     ASSERT(lun_num != NULL);
14782     ASSERT(lun_addr_type != NULL);

14784     lun_idx = (lun_cnt + 1) * MPTSAS_SCSI_REPORTLUNS_ADDRESS_SIZE;
14785     /* determine report luns addressing type */
14786     switch (buf[lun_idx] & MPTSAS_SCSI_REPORTLUNS_ADDRESS_MASK) {
14787         /*
14788          * Vendors in the field have been found to be concatenating
14789          * bus/target/lun to equal the complete lun value instead
14790          * of switching to flat space addressing
14791          */
14792         /* 00b - peripheral device addressing method */
14793         case MPTSAS_SCSI_REPORTLUNS_ADDRESS_PERIPHERAL:
14794             /* FALLTHRU */
14795             /* 10b - logical unit addressing method */
14796         case MPTSAS_SCSI_REPORTLUNS_ADDRESS_LOGICAL_UNIT:
14797             /* FALLTHRU */
14798             /* 01b - flat space addressing method */
14799         case MPTSAS_SCSI_REPORTLUNS_ADDRESS_FLAT_SPACE:
14800             /* byte0 bit0-5=msb lun byte1 bit0-7=lsb lun */
14801             *lun_addr_type = (buf[lun_idx] &
14802                 MPTSAS_SCSI_REPORTLUNS_ADDRESS_MASK) >> 6;
14803             *lun_num = (buf[lun_idx] & 0x3F) << 8;
14804             *lun_num |= buf[lun_idx + 1];
14805             return (DDI_SUCCESS);
14806         default:
14807             return (DDI_FAILURE);
14808     }
14809 }

14811 static int
14812 mptsas_config_luns(dev_info_t *pdip, mptsas_target_t *ptgt)
14813 {
14814     struct buf          *repluns_bp = NULL;
14815     struct scsi_address  ap;
14816     uchar_t             cdb[CDB_GROUP5];
14817     int                 ret = DDI_FAILURE;
14818     int                 retry = 0;
14819     int                 lun_list_len = 0;
14820     uint16_t            lun_num = 0;
14821     uint8_t             lun_addr_type = 0;
14822     uint32_t            lun_cnt = 0;
14823     uint32_t            lun_total = 0;
14824     dev_info_t          *cdip = NULL;
14825     uint16_t            *saved_repluns = NULL;
14826     char                *buffer = NULL;
14827     int                 buf_len = 128;
14828     mptsas_t            *mpt = DIP2MPT(pdip);
14829     uint64_t            sas_wnn = 0;
14830     uint8_t             phy = 0xFF;
14831     uint32_t            dev_info = 0;

14833     mutex_enter(&mpt->m_mutex);
14834     sas_wnn = ptgt->m_addr.mta_wnn;
14835     phy = ptgt->m_phynum;
14836     dev_info = ptgt->m_deviceinfo;
14837     mutex_exit(&mpt->m_mutex);

14839     if (sas_wnn == 0) {
14840         /*
14841          * It's a SATA without Device Name
14842          * So don't try multi-LUNS
14843          */
14844         if (mptsas_find_child_phy(pdip, phy)) {
14845             return (DDI_SUCCESS);

```

```

14846     } else {
14847         /*
14848          * need configure and create node
14849          */
14850         return (DDI_FAILURE);
14851     }
14852 }

14854 /*
14855  * WWN (SAS address or Device Name exist)
14856  */
14857 if (dev_info & (MPI2_SAS_DEVICE_INFO_SATA_DEVICE |
14858     MPI2_SAS_DEVICE_INFO_ATAPI_DEVICE)) {
14859     /*
14860      * SATA device with Device Name
14861      * So don't try multi-LUNS
14862      */
14863     if (mptsas_find_child_addr(pdip, sas_wnn, 0)) {
14864         return (DDI_SUCCESS);
14865     } else {
14866         return (DDI_FAILURE);
14867     }
14868 }

14870 do {
14871     ap.a_target = MPTSAS_INVALID_DEVHDL;
14872     ap.a_lun = 0;
14873     ap.a_hba_tran = mpt->m_tran;
14874     repluns_bp = scsi_alloc_consistent_buf(&ap,
14875         (struct buf *)NULL, buf_len, B_READ, NULL_FUNC, NULL);
14876     if (repluns_bp == NULL) {
14877         retry++;
14878         continue;
14879     }
14880     bzero(cdb, CDB_GROUP5);
14881     cdb[0] = SCMD_REPORT_LUNS;
14882     cdb[6] = (buf_len & 0xff000000) >> 24;
14883     cdb[7] = (buf_len & 0x00ff0000) >> 16;
14884     cdb[8] = (buf_len & 0x0000ff00) >> 8;
14885     cdb[9] = (buf_len & 0x000000ff);

14887     ret = mptsas_send_scsi_cmd(mpt, &ap, ptgt, &cdb[0], CDB_GROUP5,
14888         repluns_bp, NULL);
14889     if (ret != DDI_SUCCESS) {
14890         scsi_free_consistent_buf(repluns_bp);
14891         retry++;
14892         continue;
14893     }
14894     lun_list_len = BE_32(*(int *)((void *) (
14895         repluns_bp->b_un.b_addr)));
14896     if (buf_len >= lun_list_len + 8) {
14897         ret = DDI_SUCCESS;
14898         break;
14899     }
14900     scsi_free_consistent_buf(repluns_bp);
14901     buf_len = lun_list_len + 8;

14903 } while (retry < 3);

14905 if (ret != DDI_SUCCESS)
14906     return (ret);
14907 buffer = (char *)repluns_bp->b_un.b_addr;
14908 /*
14909  * find out the number of luns returned by the SCSI ReportLun call
14910  * and allocate buffer space
14911  */

```

```

14912     lun_total = lun_list_len / MPTSAS SCSI_REPORTLUNS_ADDRESS_SIZE;
14913     saved_repluns = kmem_zalloc(sizeof (uint16_t) * lun_total, KM_SLEEP);
14914     if (saved_repluns == NULL) {
14915         scsi_free_consistent_buf(repluns_bp);
14916         return (DDI_FAILURE);
14917     }
14918     for (lun_cnt = 0; lun_cnt < lun_total; lun_cnt++) {
14919         if (mptsas_retrieve_lundata(lun_cnt, (uint8_t *) (buffer),
14920             &lun_num, &lun_addr_type) != DDI_SUCCESS) {
14921             continue;
14922         }
14923         saved_repluns[lun_cnt] = lun_num;
14924         if (cdip = mptsas_find_child_addr(pdip, sas_wwn, lun_num))
14925             ret = DDI_SUCCESS;
14926         else
14927             ret = mptsas_probe_lun(pdip, lun_num, &cdip,
14928                 ptgt);
14929         if ((ret == DDI_SUCCESS) && (cdip != NULL)) {
14930             (void) ndi_prop_remove(DDI_DEV_T_NONE, cdip,
14931                 MPTSAS_DEV_GONE);
14932         }
14933     }
14934     mptsas_offline_missed_luns(pdip, saved_repluns, lun_total, ptgt);
14935     kmem_free(saved_repluns, sizeof (uint16_t) * lun_total);
14936     scsi_free_consistent_buf(repluns_bp);
14937     return (DDI_SUCCESS);
14938 }

14940 static int
14941 mptsas_config_raid(dev_info_t *pdip, uint16_t target, dev_info_t **dip)
14942 {
14943     int             rval = DDI_FAILURE;
14944     struct scsi_inquiry *sd_inq = NULL;
14945     mptsas_t        *mpt = DIP2MPT(pdip);
14946     mptsas_target_t *ptgt = NULL;

14948     mutex_enter(&mpt->m_mutex);
14949     ptgt = rehash_linear_search(mpt->m_targets,
14950         mptsas_target_eval_devhdl, &target);
14951     mutex_exit(&mpt->m_mutex);
14952     if (ptgt == NULL) {
14953         mptsas_log(mpt, CE_WARN, "Volume with VolDevHandle of 0x%x "
14954             "not found.", target);
14955         return (rval);
14956     }

14958     sd_inq = (struct scsi_inquiry *) kmem_alloc(SUN_INQSIZE, KM_SLEEP);
14959     rval = mptsas_inquiry(mpt, ptgt, 0, 0, (uchar_t *) sd_inq,
14960         SUN_INQSIZE, 0, (uchar_t) 0);

14962     if ((rval == DDI_SUCCESS) && MPTSAS_VALID_LUN(sd_inq)) {
14963         rval = mptsas_create_phys_lun(pdip, sd_inq, NULL, dip, ptgt,
14964             0);
14965     } else {
14966         rval = DDI_FAILURE;
14967     }

14969     kmem_free(sd_inq, SUN_INQSIZE);
14970     return (rval);
14971 }

14973 /*
14974  * configure all RAID volumes for virtual iport
14975  */
14976 static void
14977 mptsas_config_all_vport(dev_info_t *pdip)

```

```

14978 {
14979     mptsas_t        *mpt = DIP2MPT(pdip);
14980     int             config, vol;
14981     int             target;
14982     dev_info_t      *lundip = NULL;

14984     /*
14985      * Get latest RAID info and search for any Volume DevHandles. If any
14986      * are found, configure the volume.
14987      */
14988     mutex_enter(&mpt->m_mutex);
14989     for (config = 0; config < mpt->m_num_raid_configs; config++) {
14990         for (vol = 0; vol < MPTSAS_MAX_RAIDVOLS; vol++) {
14991             if (mpt->m_raidconfig[config].m_raidvol[vol].m_israid
14992                 == 1) {
14993                 target = mpt->m_raidconfig[config].
14994                     m_raidvol[vol].m_raidhandle;
14995                 mutex_exit(&mpt->m_mutex);
14996                 (void) mptsas_config_raid(pdip, target,
14997                     &lundip);
14998                 mutex_enter(&mpt->m_mutex);
14999             }
15000         }
15001     }
15002     mutex_exit(&mpt->m_mutex);
15003 }

15005 static void
15006 mptsas_offline_missed_luns(dev_info_t *pdip, uint16_t *repluns,
15007     int lun_cnt, mptsas_target_t *ptgt)
15008 {
15009     dev_info_t      *child = NULL, *savechild = NULL;
15010     mdi_pathinfo_t  *pip = NULL, *savepip = NULL;
15011     uint64_t        sas_wwn, wwid;
15012     uint8_t         phy;
15013     int             lun;
15014     int             i;
15015     int             find;
15016     char             *addr;
15017     char             *nodename;
15018     mptsas_t        *mpt = DIP2MPT(pdip);

15020     mutex_enter(&mpt->m_mutex);
15021     wwid = ptgt->m_addr.mta_wwn;
15022     mutex_exit(&mpt->m_mutex);

15024     child = ddi_get_child(pdip);
15025     while (child) {
15026         find = 0;
15027         savechild = child;
15028         child = ddi_get_next_sibling(child);

15030         nodename = ddi_node_name(savechild);
15031         if (strcmp(nodename, "smp") == 0) {
15032             continue;
15033         }

15035         addr = ddi_get_name_addr(savechild);
15036         if (addr == NULL) {
15037             continue;
15038         }

15040         if (mptsas_parse_address(addr, &sas_wwn, &phy, &lun) !=
15041             DDI_SUCCESS) {
15042             continue;
15043         }

```

```

15045         if (wwid == sas_wnw) {
15046             for (i = 0; i < lun_cnt; i++) {
15047                 if (repluns[i] == lun) {
15048                     find = 1;
15049                     break;
15050                 }
15051             }
15052         } else {
15053             continue;
15054         }
15055         if (find == 0) {
15056             /*
15057              * The lun has not been there already
15058              */
15059             (void) mptsas_offline_lun(pdip, savechild, NULL,
15060                                     NDI_DEVI_REMOVE);
15061         }
15062     }
15064     pip = mdi_get_next_client_path(pdip, NULL);
15065     while (pip) {
15066         find = 0;
15067         savepip = pip;
15068         addr = MDI_PI(pip)->pi_addr;
15070
15072         pip = mdi_get_next_client_path(pdip, pip);
15074
15076         if (addr == NULL) {
15077             continue;
15078         }
15079
15081         if (mptsas_parse_address(addr, &sas_wnw, &phy,
15082                                 &lun) != DDI_SUCCESS) {
15083             continue;
15084         }
15085
15086         if (sas_wnw == wwid) {
15087             for (i = 0; i < lun_cnt; i++) {
15088                 if (repluns[i] == lun) {
15089                     find = 1;
15090                     break;
15091                 }
15092             }
15093         } else {
15094             continue;
15095         }
15096
15097         if (find == 0) {
15098             /*
15099              * The lun has not been there already
15100              */
15101             (void) mptsas_offline_lun(pdip, NULL, savepip,
15102                                     NDI_DEVI_REMOVE);
15103         }
15104     }
15105 }
15106
15107 void
15108 mptsas_update_hashtab(struct mptsas *mpt)
15109 {
15110     uint32_t    page_address;
15111     int         rval = 0;
15112     uint16_t    dev_handle;
15113     mptsas_target_t *ptgt = NULL;
15114     mptsas_smp_t smp_node;

```

```

15111     /*
15112      * Get latest RAID info.
15113      */
15114     (void) mptsas_get_raid_info(mpt);
15116
15117     dev_handle = mpt->m_smp_devhdl;
15118     for (; mpt->m_done_traverse_smp == 0; ) {
15119         page_address = (MPI2_SAS_EXPAND_PGAD_FORM_GET_NEXT_HNDL &
15120                        MPI2_SAS_EXPAND_PGAD_FORM_MASK) | (uint32_t)dev_handle;
15121         if (mptsas_get_sas_expander_page0(mpt, page_address, &smp_node)
15122             != DDI_SUCCESS) {
15123             break;
15124         }
15125         mpt->m_smp_devhdl = dev_handle = smp_node.m_devhdl;
15126         (void) mptsas_smp_alloc(mpt, &smp_node);
15127     }
15128
15129     /*
15130      * Config target devices
15131      */
15132     dev_handle = mpt->m_dev_handle;
15134
15135     /*
15136      * Do loop to get sas device page 0 by GetNextHandle till the
15137      * the last handle. If the sas device is a SATA/SSP target,
15138      * we try to config it.
15139      */
15140     for (; mpt->m_done_traverse_dev == 0; ) {
15141         ptgt = NULL;
15142         page_address =
15143             (MPI2_SAS_DEVICE_PGAD_FORM_GET_NEXT_HANDLE &
15144              MPI2_SAS_DEVICE_PGAD_FORM_MASK) |
15145             (uint32_t)dev_handle;
15146         rval = mptsas_get_target_device_info(mpt, page_address,
15147                                              &dev_handle, &ptgt);
15148         if ((rval == DEV_INFO_FAIL_PAGE0) ||
15149             (rval == DEV_INFO_FAIL_ALLOC)) {
15150             break;
15151         }
15152         mpt->m_dev_handle = dev_handle;
15153     }
15154 }
15156 void
15157 mptsas_update_driver_data(struct mptsas *mpt)
15158 {
15159     mptsas_target_t *tp;
15160     mptsas_smp_t *sp;
15162     ASSERT(MUTEX_HELD(&mpt->m_mutex));
15164
15165     /*
15166      * TODO after hard reset, update the driver data structures
15167      * 1. update port/phymask mapping table mpt->m_phy_info
15168      * 2. invalid all the entries in hash table
15169      *    m_devhdl = 0xffff and m_deviceinfo = 0
15170      * 3. call sas_device_page/expander_page to update hash table
15171      */
15172     mptsas_update_phymask(mpt);
15173     /*
15174      * Invalid the existing entries
15175      * XXX - It seems like we should just delete everything here. We are

```

```

15176     * holding the lock and are about to refresh all the targets in both
15177     * hashes anyway. Given the path we're in, what outstanding async
15178     * event could possibly be trying to reference one of these things
15179     * without taking the lock, and how would that be useful anyway?
15180     */
15181     for (tp = rehash_first(mpt->m_targets); tp != NULL;
15182          tp = rehash_next(mpt->m_targets, tp)) {
15183         tp->m_devhdl = MPTSAS_INVALID_DEVHDL;
15184         tp->m_deviceinfo = 0;
15185         tp->m_dr_flag = MPTSAS_DR_INACTIVE;
15186     }
15187     for (sp = rehash_first(mpt->m_smp_targets); sp != NULL;
15188          sp = rehash_next(mpt->m_smp_targets, sp)) {
15189         sp->m_devhdl = MPTSAS_INVALID_DEVHDL;
15190         sp->m_deviceinfo = 0;
15191     }
15192     mpt->m_done_traverse_dev = 0;
15193     mpt->m_done_traverse_smp = 0;
15194     mpt->m_dev_handle = mpt->m_smp_devhdl = MPTSAS_INVALID_DEVHDL;
15195     mptsas_update_hashtab(mpt);
15196 }

15198 static void
15199 mptsas_config_all(dev_info_t *pdip)
15200 {
15201     dev_info_t      *smpdip = NULL;
15202     mptsas_t        *mpt = DIP2MPT(pdip);
15203     int              phymask = 0;
15204     mptsas_phymask_t phy_mask;
15205     mptsas_target_t *ptgt = NULL;
15206     mptsas_smp_t     *psmp;

15208     /*
15209      * Get the phymask associated to the iport
15210      */
15211     phymask = ddi_prop_get_int(DDI_DEV_T_ANY, pdip, 0,
15212                               "phymask", 0);

15214     /*
15215      * Enumerate RAID volumes here (phymask == 0).
15216      */
15217     if (phymask == 0) {
15218         mptsas_config_all_vport(pdip);
15219         return;
15220     }

15222     mutex_enter(&mpt->m_mutex);

15224     if (!mpt->m_done_traverse_dev || !mpt->m_done_traverse_smp) {
15225         mptsas_update_hashtab(mpt);
15226     }

15228     for (psmp = rehash_first(mpt->m_smp_targets); psmp != NULL;
15229          psmp = rehash_next(mpt->m_smp_targets, psmp)) {
15230         phy_mask = psmp->m_addr.mta_phymask;
15231         if (phy_mask == phymask) {
15232             smpdip = NULL;
15233             mutex_exit(&mpt->m_mutex);
15234             (void) mptsas_online_smp(pdip, psmp, &smpdip);
15235             mutex_enter(&mpt->m_mutex);
15236         }
15237     }

15239     for (ptgt = rehash_first(mpt->m_targets); ptgt != NULL;
15240          ptgt = rehash_next(mpt->m_targets, ptgt)) {
15241         phy_mask = ptgt->m_addr.mta_phymask;

```

```

15242         if (phy_mask == phymask) {
15243             mutex_exit(&mpt->m_mutex);
15244             (void) mptsas_config_target(pdip, ptgt);
15245             mutex_enter(&mpt->m_mutex);
15246         }
15247     }
15248     mutex_exit(&mpt->m_mutex);
15249 }

15251 static int
15252 mptsas_config_target(dev_info_t *pdip, mptsas_target_t *ptgt)
15253 {
15254     int              rval = DDI_FAILURE;
15255     dev_info_t       *tdip;

15257     rval = mptsas_config_luns(pdip, ptgt);
15258     if (rval != DDI_SUCCESS) {
15259         /*
15260          * The return value means the SCMD_REPORT_LUNS
15261          * did not execute successfully. The target maybe
15262          * doesn't support such command.
15263          */
15264         rval = mptsas_probe_lun(pdip, 0, &tdip, ptgt);
15265     }
15266     return (rval);
15267 }

15269 /*
15270  * Return fail if not all the childs/paths are freed.
15271  * if there is any path under the HBA, the return value will be always fail
15272  * because we didn't call mdi_pi_free for path
15273  */
15274 static int
15275 mptsas_offline_targetdev(dev_info_t *pdip, char *name)
15276 {
15277     dev_info_t      *child = NULL, *prechild = NULL;
15278     mdi_pathinfo_t   *pip = NULL, *savepip = NULL;
15279     int              tmp_rval, rval = DDI_SUCCESS;
15280     char             *addr, *cp;
15281     size_t           s;
15282     mptsas_t         *mpt = DIP2MPT(pdip);

15284     child = ddi_get_child(pdip);
15285     while (child) {
15286         addr = ddi_get_name_addr(child);
15287         prechild = child;
15288         child = ddi_get_next_sibling(child);

15290         if (addr == NULL) {
15291             continue;
15292         }
15293         if ((cp = strchr(addr, ',')) == NULL) {
15294             continue;
15295         }

15297         s = (uintptr_t)cp - (uintptr_t)addr;

15299         if (strncmp(addr, name, s) != 0) {
15300             continue;
15301         }

15303         tmp_rval = mptsas_offline_lun(pdip, prechild, NULL,
15304                                       NDI_DEVI_REMOVE);
15305         if (tmp_rval != DDI_SUCCESS) {
15306             rval = DDI_FAILURE;
15307             if (ndi_prop_create_boolean(DDI_DEV_T_NONE,

```

```

15308         prechild, MPTSAS_DEV_GONE) !=
15309         DDI_PROP_SUCCESS) {
15310             mptsas_log(mpt, CE_WARN,
15311                 "unable to create property for "
15312                 "SAS %s (MPTSAS_DEV_GONE)", addr);
15313         }
15314     }
15315 }

15317 pip = mdi_get_next_client_path(pdip, NULL);
15318 while (pip) {
15319     addr = MDI_PI(pip)->pi_addr;
15320     savepip = pip;
15321     pip = mdi_get_next_client_path(pdip, pip);
15322     if (addr == NULL) {
15323         continue;
15324     }

15326     if ((cp = strchr(addr, ',')) == NULL) {
15327         continue;
15328     }

15330     s = (uintptr_t)cp - (uintptr_t)addr;

15332     if (strncmp(addr, name, s) != 0) {
15333         continue;
15334     }

15336     (void) mptsas_offline_lun(pdip, NULL, savepip,
15337         NDI_DEVI_REMOVE);
15338     /*
15339      * driver will not invoke mdi_pi_free, so path will not
15340      * be freed forever, return DDI_FAILURE.
15341      */
15342     rval = DDI_FAILURE;
15343 }
15344 return (rval);
15345 }

15347 static int
15348 mptsas_offline_lun(dev_info_t *pdip, dev_info_t *rdip,
15349     mdi_pathinfo_t *rpip, uint_t flags)
15350 {
15351     int         rval = DDI_FAILURE;
15352     char        *devname;
15353     dev_info_t  *cdip, *parent;

15355     if (rpip != NULL) {
15356         parent = scsi_vhci_dip;
15357         cdip = mdi_pi_get_client(rpip);
15358     } else if (rdip != NULL) {
15359         parent = pdip;
15360         cdip = rdip;
15361     } else {
15362         return (DDI_FAILURE);
15363     }

15365     /*
15366      * Make sure node is attached otherwise
15367      * it won't have related cache nodes to
15368      * clean up. i_ddi_devi_attached is
15369      * similiar to i_ddi_node_state(cdip) >=
15370      * DS_ATTACHED.
15371      */
15372     if (i_ddi_devi_attached(cdip)) {

```

```

15374         /* Get full devname */
15375         devname = kmem_alloc(MAXNAMELEN + 1, KM_SLEEP);
15376         (void) ddi_devname(cdip, devname);
15377         /* Clean cache */
15378         (void) devfs_clean(parent, devname + 1,
15379             DV_CLEAN_FORCE);
15380         kmem_free(devname, MAXNAMELEN + 1);
15381     }
15382     if (rpip != NULL) {
15383         if (MDI_PI_IS_OFFLINE(rpip)) {
15384             rval = DDI_SUCCESS;
15385         } else {
15386             rval = mdi_pi_offline(rpip, 0);
15387         }
15388     } else {
15389         rval = ndi_devi_offline(cdip, flags);
15390     }

15392     return (rval);
15393 }

15395 static dev_info_t *
15396 mptsas_find_smp_child(dev_info_t *parent, char *str_wnn)
15397 {
15398     dev_info_t  *child = NULL;
15399     char         *smp_wnn = NULL;

15401     child = ddi_get_child(parent);
15402     while (child) {
15403         if (ddi_prop_lookup_string(DDI_DEV_T_ANY, child,
15404             DDI_PROP_DONTPASS, SMP_WWN, &smp_wnn)
15405             != DDI_SUCCESS) {
15406             child = ddi_get_next_sibling(child);
15407             continue;
15408         }

15410         if (strcmp(smp_wnn, str_wnn) == 0) {
15411             ddi_prop_free(smp_wnn);
15412             break;
15413         }
15414         child = ddi_get_next_sibling(child);
15415         ddi_prop_free(smp_wnn);
15416     }
15417     return (child);
15418 }

15420 static int
15421 mptsas_offline_smp(dev_info_t *pdip, mptsas_smp_t *smp_node, uint_t flags)
15422 {
15423     int         rval = DDI_FAILURE;
15424     char        *devname;
15425     char        wwn_str[MPTSAS_WWN_STRLEN];
15426     dev_info_t  *cdip;

15428     (void) sprintf(wwn_str, "%PRiX64, smp_node->m_addr.mta_wnn);

15430     cdip = mptsas_find_smp_child(pdip, wwn_str);

15432     if (cdip == NULL)
15433         return (DDI_SUCCESS);

15435     /*
15436      * Make sure node is attached otherwise
15437      * it won't have related cache nodes to
15438      * clean up. i_ddi_devi_attached is
15439      * similiar to i_ddi_node_state(cdip) >=

```

```

15440      * DS_ATTACHED.
15441      */
15442      if (i_ddi_devi_attached(cdip)) {

15444          /* Get full devname */
15445          devname = kmem_alloc(MAXNAMELEN + 1, KM_SLEEP);
15446          (void) ddi_devname(cdip, devname);
15447          /* Clean cache */
15448          (void) devfs_clean(pdip, devname + 1,
15449              DV_CLEAN_FORCE);
15450          kmem_free(devname, MAXNAMELEN + 1);
15451      }

15453      rval = ndi_devi_offline(cdip, flags);

15455      return (rval);
15456 }

15458 static dev_info_t *
15459 mptsas_find_child(dev_info_t *pdip, char *name)
15460 {
15461     dev_info_t    *child = NULL;
15462     char          *rname = NULL;
15463     int           rval = DDI_FAILURE;

15465     rname = kmem_zalloc(SCSI_MAXNAMELEN, KM_SLEEP);

15467     child = ddi_get_child(pdip);
15468     while (child) {
15469         rval = mptsas_name_child(child, rname, SCSI_MAXNAMELEN);
15470         if (rval != DDI_SUCCESS) {
15471             child = ddi_get_next_sibling(child);
15472             bzero(rname, SCSI_MAXNAMELEN);
15473             continue;
15474         }

15476         if (strcmp(rname, name) == 0) {
15477             break;
15478         }
15479         child = ddi_get_next_sibling(child);
15480         bzero(rname, SCSI_MAXNAMELEN);
15481     }

15483     kmem_free(rname, SCSI_MAXNAMELEN);

15485     return (child);
15486 }

15489 static dev_info_t *
15490 mptsas_find_child_addr(dev_info_t *pdip, uint64_t sasaddr, int lun)
15491 {
15492     dev_info_t    *child = NULL;
15493     char          *name = NULL;
15494     char          *addr = NULL;

15496     name = kmem_zalloc(SCSI_MAXNAMELEN, KM_SLEEP);
15497     addr = kmem_zalloc(SCSI_MAXNAMELEN, KM_SLEEP);
15498     (void) sprintf(name, "%016"PRIx64, sasaddr);
15499     (void) sprintf(addr, "%ws,%x", name, lun);
15500     child = mptsas_find_child(pdip, addr);
15501     kmem_free(name, SCSI_MAXNAMELEN);
15502     kmem_free(addr, SCSI_MAXNAMELEN);
15503     return (child);
15504 }

```

```

15506 static dev_info_t *
15507 mptsas_find_child_phy(dev_info_t *pdip, uint8_t phy)
15508 {
15509     dev_info_t    *child;
15510     char          *addr;

15512     addr = kmem_zalloc(SCSI_MAXNAMELEN, KM_SLEEP);
15513     (void) sprintf(addr, "p%x,0", phy);
15514     child = mptsas_find_child(pdip, addr);
15515     kmem_free(addr, SCSI_MAXNAMELEN);
15516     return (child);
15517 }

15519 static mdi_pathinfo_t *
15520 mptsas_find_path_phy(dev_info_t *pdip, uint8_t phy)
15521 {
15522     mdi_pathinfo_t *path;
15523     char          *addr = NULL;

15525     addr = kmem_zalloc(SCSI_MAXNAMELEN, KM_SLEEP);
15526     (void) sprintf(addr, "p%x,0", phy);
15527     path = mdi_pi_find(pdip, NULL, addr);
15528     kmem_free(addr, SCSI_MAXNAMELEN);
15529     return (path);
15530 }

15532 static mdi_pathinfo_t *
15533 mptsas_find_path_addr(dev_info_t *parent, uint64_t sasaddr, int lun)
15534 {
15535     mdi_pathinfo_t *path;
15536     char          *name = NULL;
15537     char          *addr = NULL;

15539     name = kmem_zalloc(SCSI_MAXNAMELEN, KM_SLEEP);
15540     addr = kmem_zalloc(SCSI_MAXNAMELEN, KM_SLEEP);
15541     (void) sprintf(name, "%016"PRIx64, sasaddr);
15542     (void) sprintf(addr, "%ws,%x", name, lun);
15543     path = mdi_pi_find(parent, NULL, addr);
15544     kmem_free(name, SCSI_MAXNAMELEN);
15545     kmem_free(addr, SCSI_MAXNAMELEN);

15547     return (path);
15548 }

15550 static int
15551 mptsas_create_lun(dev_info_t *pdip, struct scsi_inquiry *sd_inq,
15552     dev_info_t **lun_dip, mptsas_target_t *ptgt, int lun)
15553 {
15554     int           i = 0;
15555     uchar_t      *inq83 = NULL;
15556     int          inq83_len1 = 0xFF;
15557     int          inq83_len = 0;
15558     int          rval = DDI_FAILURE;
15559     ddi_devid_t  devid;
15560     char         *guid = NULL;
15561     int          target = ptgt->m_devhdl;
15562     mdi_pathinfo_t *pip = NULL;
15563     mptsas_t     *mpt = DIP2MPT(pdip);

15565     /*
15566      * For DVD/CD ROM and tape devices and optical
15567      * devices, we won't try to enumerate them under
15568      * scsi_vhci, so no need to try page83
15569      */
15570     if (sd_inq && (sd_inq->inq_dtype == DTYPE_RODIRECT ||
15571         sd_inq->inq_dtype == DTYPE_OPTICAL ||

```

```

15572         sd_inq->inq_dtype == DTYPE_ESI))
15573         goto create_lun;

15575     /*
15576     * The LCA returns good SCSI status, but corrupt page 83 data the first
15577     * time it is queried. The solution is to keep trying to request page83
15578     * and verify the GUID is not (DDI_NOT_WELL_FORMED) in
15579     * mptsas_inq83_retry_timeout seconds. If the timeout expires, driver
15580     * give up to get VPD page at this stage and fail the enumeration.
15581     */

15583     inq83 = kmem_zalloc(inq83_len1, KM_SLEEP);

15585     for (i = 0; i < mptsas_inq83_retry_timeout; i++) {
15586         rval = mptsas_inquiry(mpt, ptgt, lun, 0x83, inq83,
15587             inq83_len1, &inq83_len, 1);
15588         if (rval != 0) {
15589             mptsas_log(mpt, CE_WARN, "!mptsas request inquiry page "
15590                 "0x83 for target:%x, lun:%x failed!", target, lun);
15591             if (mptsas_physical_bind_failed_page_83 != B_FALSE)
15592                 goto create_lun;
15593             goto out;
15594         }
15595         /*
15596         * create DEVID from inquiry data
15597         */
15598         if ((rval = ddi_devid_scsi_encode(
15599             DEVID_SCSI_ENCODE_VERSION_LATEST, NULL, (uchar_t *)sd_inq,
15600             sizeof (struct scsi_inquiry), NULL, 0, inq83,
15601             (size_t)inq83_len, &devid)) == DDI_SUCCESS) {
15602             /*
15603             * extract GUID from DEVID
15604             */
15605             guid = ddi_devid_to_guid(devid);

15607             /*
15608             * Do not enable MPXIO if the strlen(guid) is greater
15609             * than MPTSAS_MAX_GUID_LEN, this constrain would be
15610             * handled by framework later.
15611             */
15612             if (guid && (strlen(guid) > MPTSAS_MAX_GUID_LEN)) {
15613                 ddi_devid_free_guid(guid);
15614                 guid = NULL;
15615                 if (mpt->m_mpxio_enable == TRUE) {
15616                     mptsas_log(mpt, CE_NOTE, "!Target:%x, "
15617                         "lun:%x doesn't have a valid GUID, "
15618                         "multipathing for this drive is "
15619                         "not enabled", target, lun);
15620                 }
15621             }

15623             /*
15624             * devid no longer needed
15625             */
15626             ddi_devid_free(devid);
15627             break;
15628         } else if (rval == DDI_NOT_WELL_FORMED) {
15629             /*
15630             * return value of ddi_devid_scsi_encode equal to
15631             * DDI_NOT_WELL_FORMED means DEVID_RETRY, it worth
15632             * to retry inquiry page 0x83 and get GUID.
15633             */
15634             NDBG20(("Not well formed devid, retry..."));
15635             delay(1 * drv_usectoh(1000000));
15636             continue;
15637         } else {

```

```

15638             mptsas_log(mpt, CE_WARN, "!Encode devid failed for "
15639                 "path target:%x, lun:%x", target, lun);
15640             rval = DDI_FAILURE;
15641             goto create_lun;
15642         }
15643     }

15645     if (i == mptsas_inq83_retry_timeout) {
15646         mptsas_log(mpt, CE_WARN, "!Repeated page83 requests timeout "
15647             "for path target:%x, lun:%x", target, lun);
15648     }

15650     rval = DDI_FAILURE;

15652 create_lun:
15653     if ((guid != NULL) && (mpt->m_mpxio_enable == TRUE)) {
15654         rval = mptsas_create_virt_lun(pdip, sd_inq, guid, lun_dip, &pip,
15655             ptgt, lun);
15656     }
15657     if (rval != DDI_SUCCESS) {
15658         rval = mptsas_create_phys_lun(pdip, sd_inq, guid, lun_dip,
15659             ptgt, lun);
15660     }

15661 out:
15662     if (guid != NULL) {
15663         /*
15664         * guid no longer needed
15665         */
15666         ddi_devid_free_guid(guid);
15667     }
15668     if (inq83 != NULL)
15669         kmem_free(inq83, inq83_len1);
15670     return (rval);
15671 }
15672 }

15674 static int
15675 mptsas_create_virt_lun(dev_info_t *pdip, struct scsi_inquiry *inq, char *guid,
15676     dev_info_t **lun_dip, mdi_pathinfo_t **pip, mptsas_target_t *ptgt, int lun)
15677 {
15678     int target;
15679     char *nodename = NULL;
15680     char **compatible = NULL;
15681     int ncompatible = 0;
15682     int mdi_rtn = MDI_FAILURE;
15683     int rval = DDI_FAILURE;
15684     char *old_guid = NULL;
15685     mptsas_t *mpt = DIP2MPT(pdip);
15686     char *lun_addr = NULL;
15687     char wwn_str[MPTSAS_WWN_STRLEN];
15688     char *component = NULL;
15689     phy = 0xFF;
15690     sas_wwn;
15691     lun64 = 0;
15692     devinfo;
15693     dev_hdl;
15694     pdev_hdl;
15695     dev_sas_wwn;
15696     pdev_sas_wwn;
15697     pdev_info;
15698     physport;
15699     phy_id;
15700     page_address;
15701     bay_num, enclosure, io_flags;
15702     pdev_wwn_str[MPTSAS_WWN_STRLEN];
15703     dev_info;

```



```

15705     mutex_enter(&mpt->m_mutex);
15706     target = ptgt->m_devhdl;
15707     sas_wnn = ptgt->m_addr.mta_wnn;
15708     devinfo = ptgt->m_deviceinfo;
15709     phy = ptgt->m_phynum;
15710     mutex_exit(&mpt->m_mutex);

15712     if (sas_wnn) {
15713         *pip = mptsas_find_path_addr(pdip, sas_wnn, lun);
15714     } else {
15715         *pip = mptsas_find_path_phy(pdip, phy);
15716     }

15718     if (*pip != NULL) {
15719         *lun_dip = MDI_PI(*pip)->pi_client->ct_dip;
15720         ASSERT(*lun_dip != NULL);
15721         if (ddi_prop_lookup_string(DDI_DEV_T_ANY, *lun_dip,
15722             (DDI_PROP_DONTPASS | DDI_PROP_NOTPROM),
15723             MDI_CLIENT_GUID_PROP, &old_guid) == DDI_SUCCESS) {
15724             if (strcmp(guid, old_guid, strlen(guid)) == 0) {
15725                 /*
15726                  * Same path back online again.
15727                  */
15728                 (void) ddi_prop_free(old_guid);
15729                 if (!MDI_PI_IS_ONLINE(*pip) &&
15730                     (!MDI_PI_IS_STANDBY(*pip)) &&
15731                     (ptgt->m_tgt_unconfigured == 0)) {
15732                     rval = mdi_pi_online(*pip, 0);
15733                     mutex_enter(&mpt->m_mutex);
15734                     ptgt->m_led_status = 0;
15735                     (void) mptsas_flush_led_status(mpt,
15736                         ptgt);
15737                     mutex_exit(&mpt->m_mutex);
15738                 } else {
15739                     rval = DDI_SUCCESS;
15740                 }
15741                 if (rval != DDI_SUCCESS) {
15742                     mptsas_log(mpt, CE_WARN, "path:target: "
15743                         "%x, lun:%x online failed!", target,
15744                         lun);
15745                     *pip = NULL;
15746                     *lun_dip = NULL;
15747                 }
15748                 return (rval);
15749             } else {
15750                 /*
15751                  * The GUID of the LUN has changed which maybe
15752                  * because customer mapped another volume to the
15753                  * same LUN.
15754                  */
15755                 mptsas_log(mpt, CE_WARN, "The GUID of the "
15756                     "target:%x, lun:%x was changed, maybe "
15757                     "because someone mapped another volume "
15758                     "to the same LUN", target, lun);
15759                 (void) ddi_prop_free(old_guid);
15760                 if (!MDI_PI_IS_OFFLINE(*pip)) {
15761                     rval = mdi_pi_offline(*pip, 0);
15762                     if (rval != MDI_SUCCESS) {
15763                         mptsas_log(mpt, CE_WARN, "path:"
15764                             "target:%x, lun:%x offline "
15765                             "failed!", target, lun);
15766                         *pip = NULL;
15767                         *lun_dip = NULL;
15768                         return (DDI_FAILURE);
15769                     }

```

```

15770     }
15771     if (mdi_pi_free(*pip, 0) != MDI_SUCCESS) {
15772         mptsas_log(mpt, CE_WARN, "path:target:"
15773             "%x, lun:%x free failed!", target,
15774             lun);
15775         *pip = NULL;
15776         *lun_dip = NULL;
15777         return (DDI_FAILURE);
15778     }
15779     } else {
15780         mptsas_log(mpt, CE_WARN, "Can't get client-guid "
15781             "property for path:target:%x, lun:%x", target, lun);
15782         *pip = NULL;
15783         *lun_dip = NULL;
15784         return (DDI_FAILURE);
15785     }
15786     }
15787     scsi_hba_nodename_compatible_get(inq, NULL,
15788         inq->inq_dtype, NULL, &nodename, &compatible, &ncompatible);
15789
15790     /*
15791     * if nodename can't be determined then print a message and skip it
15792     */
15793     if (nodename == NULL) {
15794         mptsas_log(mpt, CE_WARN, "found no compatible "
15795             "driver for target%d lun %d dtype:0x%02x", target, lun,
15796             inq->inq_dtype);
15797         return (DDI_FAILURE);
15798     }
15799
15800     /* The property is needed by MPAPI */
15801     (void) sprintf(wnn_str, "%016PRIx64, sas_wnn");
15802
15803     lun_addr = kmem_zalloc(SCSI_MAXNAMELEN, KM_SLEEP);
15804     if (guid) {
15805         (void) sprintf(lun_addr, "w%s,%x", wnn_str, lun);
15806         (void) sprintf(wnn_str, "w%016PRIx64, sas_wnn");
15807     } else {
15808         (void) sprintf(lun_addr, "p%x,%x", phy, lun);
15809         (void) sprintf(wnn_str, "p%x", phy);
15810     }
15811
15812     mdi_rtn = mdi_pi_alloc_compatible(pdip, nodename,
15813         guid, lun_addr, compatible, ncompatible,
15814         0, pip);
15815     if (mdi_rtn == MDI_SUCCESS) {
15816         if (mdi_prop_update_string(*pip, MDI_GUID,
15817             guid) != DDI_SUCCESS) {
15818             mptsas_log(mpt, CE_WARN, "unable to "
15819                 "create prop for target %d lun %d (MDI_GUID)",
15820                 target, lun);
15821             mdi_rtn = MDI_FAILURE;
15822             goto virt_create_done;
15823         }
15824         if (mdi_prop_update_int(*pip, LUN_PROP,
15825             lun) != DDI_SUCCESS) {
15826             mptsas_log(mpt, CE_WARN, "unable to "
15827                 "create prop for target %d lun %d (LUN_PROP)",
15828                 target, lun);
15829             mdi_rtn = MDI_FAILURE;
15830             goto virt_create_done;
15831         }
15832         lun64 = (int64_t)lun;

```

```

15836         if (mdi_prop_update_int64(*pip, LUN64_PROP,
15837             lun64) != DDI_SUCCESS) {
15838             mptsas_log(mpt, CE_WARN, "unable to "
15839                 "create prop for target %d (LUN64_PROP)",
15840                 target);
15841             mdi_rtn = MDI_FAILURE;
15842             goto virt_create_done;
15843         }
15844         if (mdi_prop_update_string_array(*pip, "compatible",
15845             compatible, ncompatible) !=
15846             DDI_PROP_SUCCESS) {
15847             mptsas_log(mpt, CE_WARN, "unable to "
15848                 "create prop for target %d lun %d (COMPATIBLE)",
15849                 target, lun);
15850             mdi_rtn = MDI_FAILURE;
15851             goto virt_create_done;
15852         }
15853         if (sas_wnn && (mdi_prop_update_string(*pip,
15854             SCSI_ADDR_PROP_TARGET_PORT, wwn_str) != DDI_PROP_SUCCESS)) {
15855             mptsas_log(mpt, CE_WARN, "unable to "
15856                 "create prop for target %d lun %d "
15857                 "(target-port)", target, lun);
15858             mdi_rtn = MDI_FAILURE;
15859             goto virt_create_done;
15860         } else if ((sas_wnn == 0) && (mdi_prop_update_int(*pip,
15861             "sata-phy", phy) != DDI_PROP_SUCCESS)) {
15862             /*
15863              * Direct attached SATA device without DeviceName
15864              */
15865             mptsas_log(mpt, CE_WARN, "unable to "
15866                 "create prop for SAS target %d lun %d "
15867                 "(sata-phy)", target, lun);
15868             mdi_rtn = MDI_FAILURE;
15869             goto virt_create_done;
15870         }
15871         mutex_enter(&mpt->m_mutex);

15873         page_address = (MPI2_SAS_DEVICE_PGAD_FORM_HANDLE &
15874             MPI2_SAS_DEVICE_PGAD_FORM_MASK) |
15875             (uint32_t)ptgt->m_devhdl;
15876         rval = mptsas_get_sas_device_page0(mpt, page_address,
15877             &dev_hdl, &dev_sas_wnn, &dev_info, &physport,
15878             &phy_id, &pdev_hdl, &bay_num, &enclosure, &io_flags);
15879         if (rval != DDI_SUCCESS) {
15880             mutex_exit(&mpt->m_mutex);
15881             mptsas_log(mpt, CE_WARN, "mptsas unable to get "
15882                 "parent device for handle %d", page_address);
15883             mdi_rtn = MDI_FAILURE;
15884             goto virt_create_done;
15885         }

15887         page_address = (MPI2_SAS_DEVICE_PGAD_FORM_HANDLE &
15888             MPI2_SAS_DEVICE_PGAD_FORM_MASK) | (uint32_t)pdev_hdl;
15889         rval = mptsas_get_sas_device_page0(mpt, page_address,
15890             &dev_hdl, &pdev_sas_wnn, &pdev_info, &physport,
15891             &phy_id, &pdev_hdl, &bay_num, &enclosure, &io_flags);
15892         if (rval != DDI_SUCCESS) {
15893             mutex_exit(&mpt->m_mutex);
15894             mptsas_log(mpt, CE_WARN, "mptsas unable to get "
15895                 "device info for handle %d", page_address);
15896             mdi_rtn = MDI_FAILURE;
15897             goto virt_create_done;
15898         }

15900         mutex_exit(&mpt->m_mutex);

```

```

15902         /*
15903          * If this device direct attached to the controller
15904          * set the attached-port to the base wwid
15905          */
15906         if ((ptgt->m_deviceinfo & DEVINFO_DIRECT_ATTACHED)
15907             != DEVINFO_DIRECT_ATTACHED) {
15908             (void) sprintf(pdev_wnn_str, "w%016"PRIx64,
15909                 pdev_sas_wnn);
15910         } else {
15911             /*
15912              * Update the iport's attached-port to guid
15913              */
15914             if (sas_wnn == 0) {
15915                 (void) sprintf(wnn_str, "p%x", phy);
15916             } else {
15917                 (void) sprintf(wnn_str, "w%016"PRIx64, sas_wnn);
15918             }
15919             if (ddi_prop_update_string(DDI_DEV_T_NONE,
15920                 pdip, SCSI_ADDR_PROP_ATTACHED_PORT, wwn_str) !=
15921                 DDI_PROP_SUCCESS) {
15922                 mptsas_log(mpt, CE_WARN,
15923                     "mptsas unable to create "
15924                     "property for iport target-port "
15925                     "%s (sas_wnn)",
15926                     wwn_str);
15927                 mdi_rtn = MDI_FAILURE;
15928                 goto virt_create_done;
15929             }

15931             (void) sprintf(pdev_wnn_str, "w%016"PRIx64,
15932                 mpt->un.m_base_wwid);
15933         }

15935         if (mdi_prop_update_string(*pip,
15936             SCSI_ADDR_PROP_ATTACHED_PORT, pdev_wnn_str) !=
15937             DDI_PROP_SUCCESS) {
15938             mptsas_log(mpt, CE_WARN, "unable to create "
15939                 "property for iport attached-port %s (sas_wnn)",
15940                 pdev_wnn_str);
15941             mdi_rtn = MDI_FAILURE;
15942             goto virt_create_done;
15943         }

15946         if (inq->inq_dtype == 0) {
15947             component = kmem_zalloc(MAXPATHLEN, KM_SLEEP);
15948             /*
15949              * set obp path for pathinfo
15950              */
15951             (void) snprintf(component, MAXPATHLEN,
15952                 "disk%s", lun_addr);

15954             if (mdi_pi_pathname_obp_set(*pip, component) !=
15955                 DDI_SUCCESS) {
15956                 mptsas_log(mpt, CE_WARN,
15957                     "unable to set obp-path for object %s",
15958                     component);
15959                 mdi_rtn = MDI_FAILURE;
15960                 goto virt_create_done;
15961             }
15962         }

15964         *lun_dip = MDI_PI(*pip)->pi_client->ct_dip;
15965         if (devinfo & (MPI2_SAS_DEVICE_INFO_SATA_DEVICE |
15966             MPI2_SAS_DEVICE_INFO_ATAPI_DEVICE)) {
15967             if ((ndi_prop_update_int(DDI_DEV_T_NONE, *lun_dip,

```

```

15968         "pm-capable", 1)) !=
15969         DDI_PROP_SUCCESS) {
15970             mptsas_log(mpt, CE_WARN,
15971                 "failed to create pm-capable "
15972                 "property, target %d", target);
15973             mdi_rtn = MDI_FAILURE;
15974             goto virt_create_done;
15975         }
15976     }
15977     /*
15978     * Create the phy-num property
15979     */
15980     if (mdi_prop_update_int(*pip, "phy-num",
15981         ptgt->m_phynum) != DDI_SUCCESS) {
15982         mptsas_log(mpt, CE_WARN, "unable to "
15983             "create phy-num property for target %d lun %d",
15984             target, lun);
15985         mdi_rtn = MDI_FAILURE;
15986         goto virt_create_done;
15987     }
15988     NDBG20(("new path:%s onlineing,", MDI_PI(*pip)->pi_addr));
15989     mdi_rtn = mdi_pi_online(*pip, 0);
15990     if (mdi_rtn == MDI_SUCCESS) {
15991         mutex_enter(&mpt->m_mutex);
15992         ptgt->m_led_status = 0;
15993         (void) mptsas_flush_led_status(mpt, ptgt);
15994         mutex_exit(&mpt->m_mutex);
15995     }
15996     if (mdi_rtn == MDI_NOT_SUPPORTED) {
15997         mdi_rtn = MDI_FAILURE;
15998     }
15999     virt_create_done:
16000     if (*pip && mdi_rtn != MDI_SUCCESS) {
16001         (void) mdi_pi_free(*pip, 0);
16002         *pip = NULL;
16003         *lun_dip = NULL;
16004     }
16005 }
16007     scsi_hba_nodename_compatible_free(nodename, compatible);
16008     if (lun_addr != NULL) {
16009         kmem_free(lun_addr, SCSI_MAXNAMELEN);
16010     }
16011     if (component != NULL) {
16012         kmem_free(component, MAXPATHLEN);
16013     }
16015     return ((mdi_rtn == MDI_SUCCESS) ? DDI_SUCCESS : DDI_FAILURE);
16016 }
16018 static int
16019 mptsas_create_phys_lun(dev_info_t *pdip, struct scsi_inquiry *inq,
16020     char *guid, dev_info_t **lun_dip, mptsas_target_t *ptgt, int lun)
16021 {
16022     int         target;
16023     int         rval;
16024     int         ndi_rtn = NDI_FAILURE;
16025     uint64_t    be_sas_wwn;
16026     char        *nodename = NULL;
16027     char        **compatible = NULL;
16028     int         ncompatible = 0;
16029     int         instance = 0;
16030     mptsas_t    *mpt = DIP2MPT(pdip);
16031     char        wwn_str[MPTSAS_WWN_STRLEN];
16032     char        component[MAXPATHLEN];
16033     uint8_t     phy = 0xFF;

```

```

16034     uint64_t    sas_wwn;
16035     uint32_t    devinfo;
16036     uint16_t    dev_hdl;
16037     uint16_t    pdev_hdl;
16038     uint64_t    pdev_sas_wwn;
16039     uint64_t    dev_sas_wwn;
16040     uint32_t    pdev_info;
16041     uint8_t     physport;
16042     uint8_t     phy_id;
16043     uint32_t    page_address;
16044     uint16_t    bay_num, enclosure, io_flags;
16045     char        pdev_wwn_str[MPTSAS_WWN_STRLEN];
16046     uint32_t    dev_info;
16047     int64_t     lun64 = 0;
16049     mutex_enter(&mpt->m_mutex);
16050     target = ptgt->m_devhdl;
16051     sas_wwn = ptgt->m_addr.mta_wwn;
16052     devinfo = ptgt->m_deviceinfo;
16053     phy = ptgt->m_phynum;
16054     mutex_exit(&mpt->m_mutex);
16056     /*
16057     * generate compatible property with binding-set "mpt"
16058     */
16059     scsi_hba_nodename_compatible_get(inq, NULL, inq->inq_dtype, NULL,
16060         &nodename, &compatible, &ncompatible);
16062     /*
16063     * if nodename can't be determined then print a message and skip it
16064     */
16065     if (nodename == NULL) {
16066         mptsas_log(mpt, CE_WARN, "mptsas found no compatible driver "
16067             "for target %d lun %d", target, lun);
16068         return (DDI_FAILURE);
16069     }
16071     ndi_rtn = ndi_devi_alloc(pdip, nodename,
16072         DEVI_SID_NODEID, lun_dip);
16074     /*
16075     * if lun alloc success, set props
16076     */
16077     if (ndi_rtn == NDI_SUCCESS) {
16079         if (ndi_prop_update_int(DDI_DEV_T_NONE,
16080             *lun_dip, LUN_PROP, lun) !=
16081             DDI_PROP_SUCCESS) {
16082             mptsas_log(mpt, CE_WARN, "mptsas unable to create "
16083                 "property for target %d lun %d (LUN_PROP)",
16084                 target, lun);
16085             ndi_rtn = NDI_FAILURE;
16086             goto phys_create_done;
16087         }
16089         lun64 = (int64_t)lun;
16090         if (ndi_prop_update_int64(DDI_DEV_T_NONE,
16091             *lun_dip, LUN64_PROP, lun64) !=
16092             DDI_PROP_SUCCESS) {
16093             mptsas_log(mpt, CE_WARN, "mptsas unable to create "
16094                 "property for target %d lun64 %d (LUN64_PROP)",
16095                 target, lun);
16096             ndi_rtn = NDI_FAILURE;
16097             goto phys_create_done;
16098         }
16099         if (ndi_prop_update_string_array(DDI_DEV_T_NONE,

```

```

16100         *lun_dip, "compatible", compatible, ncompatible)
16101         != DDI_PROP_SUCCESS) {
16102             mptsas_log(mpt, CE_WARN, "mptsas unable to create "
16103                 "property for target %d lun %d (COMPATIBLE)",
16104                 target, lun);
16105             ndi_rtn = NDI_FAILURE;
16106             goto phys_create_done;
16107         }
16108     /*
16109     * We need the SAS WWN for non-multipath devices, so
16110     * we'll use the same property as that multipathing
16111     * devices need to present for MPAPI. If we don't have
16112     * a WWN (e.g. parallel SCSI), don't create the prop.
16113     */
16114     (void) sprintf(wnn_str, "w%016"PRIx64, sas_wnn);
16115     if (sas_wnn && ndi_prop_update_string(DDI_DEV_T_NONE,
16116         *lun_dip, SCSI_ADDR_PROP_TARGET_PORT, wnn_str)
16117         != DDI_PROP_SUCCESS) {
16118         mptsas_log(mpt, CE_WARN, "mptsas unable to "
16119             "create property for SAS target %d lun %d "
16120             "(target-port)", target, lun);
16121         ndi_rtn = NDI_FAILURE;
16122         goto phys_create_done;
16123     }
16124
16125     be_sas_wnn = BE_64(sas_wnn);
16126     if (sas_wnn && ndi_prop_update_byte_array(
16127         DDI_DEV_T_NONE, *lun_dip, "port-wnn",
16128         (uchar_t *)&be_sas_wnn, 8) != DDI_PROP_SUCCESS) {
16129         mptsas_log(mpt, CE_WARN, "mptsas unable to "
16130             "create property for SAS target %d lun %d "
16131             "(port-wnn)", target, lun);
16132         ndi_rtn = NDI_FAILURE;
16133         goto phys_create_done;
16134     } else if ((sas_wnn == 0) && (ndi_prop_update_int(
16135         DDI_DEV_T_NONE, *lun_dip, "sata-phy", phy) !=
16136         DDI_PROP_SUCCESS)) {
16137         /*
16138         * Direct attached SATA device without DeviceName
16139         */
16140         mptsas_log(mpt, CE_WARN, "mptsas unable to "
16141             "create property for SAS target %d lun %d "
16142             "(sata-phy)", target, lun);
16143         ndi_rtn = NDI_FAILURE;
16144         goto phys_create_done;
16145     }
16146
16147     if (ndi_prop_create_boolean(DDI_DEV_T_NONE,
16148         *lun_dip, SAS_PROP) != DDI_PROP_SUCCESS) {
16149         mptsas_log(mpt, CE_WARN, "mptsas unable to "
16150             "create property for SAS target %d lun %d "
16151             "(SAS_PROP)", target, lun);
16152         ndi_rtn = NDI_FAILURE;
16153         goto phys_create_done;
16154     }
16155     if (guid && (ndi_prop_update_string(DDI_DEV_T_NONE,
16156         *lun_dip, NDI_GUID, guid) != DDI_SUCCESS)) {
16157         mptsas_log(mpt, CE_WARN, "mptsas unable "
16158             "to create guid property for target %d "
16159             "lun %d", target, lun);
16160         ndi_rtn = NDI_FAILURE;
16161         goto phys_create_done;
16162     }
16163
16164     /*

```

```

16166         * The following code is to set properties for SM-HBA support,
16167         * it doesn't apply to RAID volumes
16168         */
16169         if (ptgt->m_addr.mta_phymask == 0)
16170             goto phys_raid_lun;
16171
16172         mutex_enter(&mpt->m_mutex);
16173
16174         page_address = (MPI2_SAS_DEVICE_PGAD_FORM_HANDLE &
16175             MPI2_SAS_DEVICE_PGAD_FORM_MASK) |
16176             (uint32_t)ptgt->m_devhdl;
16177         rval = mptsas_get_sas_device_page0(mpt, page_address,
16178             &dev_hdl, &dev_sas_wnn, &dev_info,
16179             &physport, &phy_id, &pdev_hdl,
16180             &bay_num, &enclosure, &io_flags);
16181         if (rval != DDI_SUCCESS) {
16182             mutex_exit(&mpt->m_mutex);
16183             mptsas_log(mpt, CE_WARN, "mptsas unable to get "
16184                 "parent device for handle %d.", page_address);
16185             ndi_rtn = NDI_FAILURE;
16186             goto phys_create_done;
16187         }
16188
16189         page_address = (MPI2_SAS_DEVICE_PGAD_FORM_HANDLE &
16190             MPI2_SAS_DEVICE_PGAD_FORM_MASK) | (uint32_t)pdev_hdl;
16191         rval = mptsas_get_sas_device_page0(mpt, page_address,
16192             &dev_hdl, &pdev_sas_wnn, &pdev_info, &physport,
16193             &phy_id, &pdev_hdl, &bay_num, &enclosure, &io_flags);
16194         if (rval != DDI_SUCCESS) {
16195             mutex_exit(&mpt->m_mutex);
16196             mptsas_log(mpt, CE_WARN, "mptsas unable to create "
16197                 "device for handle %d.", page_address);
16198             ndi_rtn = NDI_FAILURE;
16199             goto phys_create_done;
16200         }
16201
16202         mutex_exit(&mpt->m_mutex);
16203
16204         /*
16205         * If this device direct attached to the controller
16206         * set the attached-port to the base wwid
16207         */
16208         if ((ptgt->m_deviceinfo & DEVINFO_DIRECT_ATTACHED)
16209             != DEVINFO_DIRECT_ATTACHED) {
16210             (void) sprintf(pdev_wnn_str, "w%016"PRIx64,
16211                 pdev_sas_wnn);
16212         } else {
16213             /*
16214             * Update the iport's attached-port to guid
16215             */
16216             if (sas_wnn == 0) {
16217                 (void) sprintf(wnn_str, "p%x", phy);
16218             } else {
16219                 (void) sprintf(wnn_str, "w%016"PRIx64, sas_wnn);
16220             }
16221             if (ddi_prop_update_string(DDI_DEV_T_NONE,
16222                 pdip, SCSI_ADDR_PROP_ATTACHED_PORT, wnn_str) !=
16223                 DDI_PROP_SUCCESS) {
16224                 mptsas_log(mpt, CE_WARN,
16225                     "mptsas unable to create "
16226                     "property for iport target-port "
16227                     "%s (sas_wnn)",
16228                     wnn_str);
16229                 ndi_rtn = NDI_FAILURE;
16230                 goto phys_create_done;
16231             }

```

```

16233         (void) sprintf(pdev_wnn_str, "%016"PRIx64,
16234             mpt->un.m_base_wwid);
16235     }

16237     if (ndi_prop_update_string(DDI_DEV_T_NONE,
16238         *lun_dip, SCSI_ADDR_PROP_ATTACHED_PORT, pdev_wnn_str) !=
16239         DDI_PROP_SUCCESS) {
16240         mptsas_log(mpt, CE_WARN,
16241             "mptsas unable to create "
16242             "property for iport attached-port %s (sas_wnn)",
16243             pdev_wnn_str);
16244         ndi_rtn = NDI_FAILURE;
16245         goto phys_create_done;
16246     }

16248     if (IS_SATA_DEVICE(dev_info)) {
16249         if (ndi_prop_update_string(DDI_DEV_T_NONE,
16250             *lun_dip, MPTSAS_VARIANT, "sata") !=
16251             DDI_PROP_SUCCESS) {
16252             mptsas_log(mpt, CE_WARN,
16253                 "mptsas unable to create "
16254                 "property for device variant ");
16255             ndi_rtn = NDI_FAILURE;
16256             goto phys_create_done;
16257         }
16258     }

16260     if (IS_ATAPI_DEVICE(dev_info)) {
16261         if (ndi_prop_update_string(DDI_DEV_T_NONE,
16262             *lun_dip, MPTSAS_VARIANT, "atapi") !=
16263             DDI_PROP_SUCCESS) {
16264             mptsas_log(mpt, CE_WARN,
16265                 "mptsas unable to create "
16266                 "property for device variant ");
16267             ndi_rtn = NDI_FAILURE;
16268             goto phys_create_done;
16269         }
16270     }

16272 phys_raid_lun:
16273     /*
16274     * if this is a SAS controller, and the target is a SATA
16275     * drive, set the 'pm-capable' property for sd and if on
16276     * an OPL platform, also check if this is an ATAPI
16277     * device.
16278     */
16279     instance = ddi_get_instance(mpt->m_dip);
16280     if (devinfo & (MPI2_SAS_DEVICE_INFO_SATA_DEVICE |
16281         MPI2_SAS_DEVICE_INFO_ATAPI_DEVICE)) {
16282         NDBG2(("mptsas3%d: creating pm-capable property, "
16283             "target %d", instance, target));

16285         if ((ndi_prop_update_int(DDI_DEV_T_NONE,
16286             *lun_dip, "pm-capable", 1)) !=
16287             DDI_PROP_SUCCESS) {
16288             mptsas_log(mpt, CE_WARN, "mptsas "
16289                 "failed to create pm-capable "
16290                 "property, target %d", target);
16291             ndi_rtn = NDI_FAILURE;
16292             goto phys_create_done;
16293         }

16295     }

16297     if ((inq->inq_dtype == 0) || (inq->inq_dtype == 5)) {

```

```

16298     /*
16299     * add 'obp-path' properties for devinfo
16300     */
16301     bzero(wnn_str, sizeof(wnn_str));
16302     (void) sprintf(wnn_str, "%016"PRIx64, sas_wnn);
16303     if (guid) {
16304         (void) snprintf(component, MAXPATHLEN,
16305             "disk@%s,%x", wnn_str, lun);
16306     } else {
16307         (void) snprintf(component, MAXPATHLEN,
16308             "disk@%x,%x", phy, lun);
16309     }
16310     if (ddi_pathname_obp_set(*lun_dip, component)
16311         != DDI_SUCCESS) {
16312         mptsas_log(mpt, CE_WARN, "mpt_sas driver "
16313             "unable to set obp-path for SAS "
16314             "object %s", component);
16315         ndi_rtn = NDI_FAILURE;
16316         goto phys_create_done;
16317     }
16318 }
16319 /*
16320 * Create the phy-num property for non-raid disk
16321 */
16322 if (ptgt->m_addr.mta_phymask != 0) {
16323     if (ndi_prop_update_int(DDI_DEV_T_NONE,
16324         *lun_dip, "phy-num", ptgt->m_phynum) !=
16325         DDI_PROP_SUCCESS) {
16326         mptsas_log(mpt, CE_WARN,
16327             "failed to create phy-num property for "
16328             "target %d", target);
16329         ndi_rtn = NDI_FAILURE;
16330         goto phys_create_done;
16331     }
16332 }
16333 phys_create_done:
16334     /*
16335     * If props were setup ok, online the lun
16336     */
16337     if (ndi_rtn == NDI_SUCCESS) {
16338         /*
16339         * Try to online the new node
16340         */
16341         ndi_rtn = ndi_devi_online(*lun_dip, NDI_ONLINE_ATTACH);
16342     }
16343     if (ndi_rtn == NDI_SUCCESS) {
16344         mutex_enter(&mpt->m_mutex);
16345         ptgt->m_led_status = 0;
16346         (void) mptsas_flush_led_status(mpt, ptgt);
16347         mutex_exit(&mpt->m_mutex);
16348     }

16350     /*
16351     * If success set rtn flag, else unwire alloc'd lun
16352     */
16353     if (ndi_rtn != NDI_SUCCESS) {
16354         NDBG12(("unable to online "
16355             "target %d lun %d", target, lun));
16356         ndi_prop_remove_all(*lun_dip);
16357         (void) ndi_devi_free(*lun_dip);
16358         *lun_dip = NULL;
16359     }
16360 }

16362     scsi_hba_nodename_compatible_free(nodename, compatible);

```

```

16364         return ((ndi_rtn == NDI_SUCCESS) ? DDI_SUCCESS : DDI_FAILURE);
16365     }

16367 static int
16368 mptsas_probe_smp(dev_info_t *pdip, uint64_t wwn)
16369 {
16370     mptsas_t      *mpt = DIP2MPT(pdip);
16371     struct smp_device smp_sd;

16373     /* XXX An HBA driver should not be allocating an smp_device. */
16374     bzero(&smp_sd, sizeof (struct smp_device));
16375     smp_sd.smp_sd_address.smp_a_hba_tran = mpt->m_smptran;
16376     bcopy(&wwn, smp_sd.smp_sd_address.smp_a_wwn, SAS_WWN_BYTE_SIZE);

16378     if (smp_probe(&smp_sd) != DDI_PROBE_SUCCESS)
16379         return (NDI_FAILURE);
16380     return (NDI_SUCCESS);
16381 }

16383 static int
16384 mptsas_config_smp(dev_info_t *pdip, uint64_t sas_wwn, dev_info_t **smp_dip)
16385 {
16386     mptsas_t      *mpt = DIP2MPT(pdip);
16387     mptsas_smp_t  *psmp = NULL;
16388     int            rval;
16389     int            phymask;

16391     /*
16392      * Get the physical port associated to the iport
16393      * PHYMASK TODO
16394      */
16395     phymask = ddi_prop_get_int(DDI_DEV_T_ANY, pdip, 0,
16396         "phymask", 0);
16397     /*
16398      * Find the smp node in hash table with specified sas address and
16399      * physical port
16400      */
16401     psmp = mptsas_wwid_to_psmpt(mpt, phymask, sas_wwn);
16402     if (psmp == NULL) {
16403         return (DDI_FAILURE);
16404     }

16406     rval = mptsas_online_smp(pdip, psmp, smp_dip);

16408     return (rval);
16409 }

16411 static int
16412 mptsas_online_smp(dev_info_t *pdip, mptsas_smp_t *smp_node,
16413     dev_info_t **smp_dip)
16414 {
16415     char            wwn_str[MPTSAS_WWN_STRLEN];
16416     char            attached_wwn_str[MPTSAS_WWN_STRLEN];
16417     int             ndi_rtn = NDI_FAILURE;
16418     int             rval = 0;
16419     mptsas_smp_t    dev_info;
16420     uint32_t        page_address;
16421     mptsas_t        *mpt = DIP2MPT(pdip);
16422     uint16_t        dev_hdl;
16423     uint64_t        sas_wwn;
16424     uint64_t        smp_sas_wwn;
16425     uint8_t         physport;
16426     uint8_t         phy_id;
16427     uint16_t        pdev_hdl;
16428     uint8_t         numphys = 0;
16429     uint16_t        i = 0;

```

```

16430     char            phymask[MPTSAS_MAX_PHYS];
16431     char            *iport = NULL;
16432     mptsas_phymask_t phy_mask = 0;
16433     uint16_t        attached_devhdl;
16434     uint16_t        bay_num, enclosure, io_flags;

16436     (void) sprintf(wwn_str, "%PRIx64", smp_node->m_addr.mta_wwn);

16438     /*
16439      * Probe smp device, prevent the node of removed device from being
16440      * configured successfully
16441      */
16442     if (mptsas_probe_smp(pdip, smp_node->m_addr.mta_wwn) != NDI_SUCCESS) {
16443         return (DDI_FAILURE);
16444     }

16446     if ((*smp_dip = mptsas_find_smp_child(pdip, wwn_str)) != NULL) {
16447         return (DDI_SUCCESS);
16448     }

16450     ndi_rtn = ndi_devi_alloc(pdip, "smp", DEVI_SID_NODEID, smp_dip);

16452     /*
16453      * if lun alloc success, set props
16454      */
16455     if (ndi_rtn == NDI_SUCCESS) {
16456         /*
16457          * Set the flavor of the child to be SMP flavored
16458          */
16459         ndi_flavor_set(*smp_dip, SCESA_FLAVOR_SMP);

16461         if (ndi_prop_update_string(DDI_DEV_T_NONE,
16462             *smp_dip, SMP_WWN, wwn_str) !=
16463             DDI_PROP_SUCCESS) {
16464             mptsas_log(mpt, CE_WARN, "mptsas unable to create "
16465                 "property for smp device %s (sas_wwn)",
16466                 wwn_str);
16467             ndi_rtn = NDI_FAILURE;
16468             goto smp_create_done;
16469         }
16470         (void) sprintf(wwn_str, "%PRIx64", smp_node->m_addr.mta_wwn);
16471         if (ndi_prop_update_string(DDI_DEV_T_NONE,
16472             *smp_dip, SCSI_ADDR_PROP_TARGET_PORT, wwn_str) !=
16473             DDI_PROP_SUCCESS) {
16474             mptsas_log(mpt, CE_WARN, "mptsas unable to create "
16475                 "property for iport target-port %s (sas_wwn)",
16476                 wwn_str);
16477             ndi_rtn = NDI_FAILURE;
16478             goto smp_create_done;
16479         }
16481         mutex_enter(&mpt->m_mutex);

16483         page_address = (MPI2_SAS_EXPAND_PGAD_FORM_HNDL &
16484             MPI2_SAS_EXPAND_PGAD_FORM_MASK) | smp_node->m_devhdl;
16485         rval = mptsas_get_sas_expander_page0(mpt, page_address,
16486             &dev_info);
16487         if (rval != DDI_SUCCESS) {
16488             mutex_exit(&mpt->m_mutex);
16489             mptsas_log(mpt, CE_WARN,
16490                 "mptsas unable to get expander "
16491                 "parent device info for %x", page_address);
16492             ndi_rtn = NDI_FAILURE;
16493             goto smp_create_done;
16494         }
16495     }

```

```

16496     smp_node->m_pdevhdl = dev_info.m_pdevhdl;
16497     page_address = (MPI2_SAS_DEVICE_PGAD_FORM_HANDLE &
16498     MPI2_SAS_DEVICE_PGAD_FORM_MASK) |
16499     (uint32_t)dev_info.m_pdevhdl;
16500     rval = mptsas_get_sas_device_page0(mpt, page_address,
16501     &dev_hdl, &sas_wnn, &smp_node->m_pdevinfo, &physport,
16502     &phy_id, &pdev_hdl, &bay_num, &enclosure, &io_flags);
16503     if (rval != DDI_SUCCESS) {
16504         mutex_exit(&mpt->m_mutex);
16505         mptsas_log(mpt, CE_WARN, "mptsas unable to get "
16506         "device info for %x", page_address);
16507         ndi_rtn = NDI_FAILURE;
16508         goto smp_create_done;
16509     }

16511     page_address = (MPI2_SAS_DEVICE_PGAD_FORM_HANDLE &
16512     MPI2_SAS_DEVICE_PGAD_FORM_MASK) |
16513     (uint32_t)dev_info.m_devhdl;
16514     rval = mptsas_get_sas_device_page0(mpt, page_address,
16515     &dev_hdl, &smp_sas_wnn, &smp_node->m_deviceinfo,
16516     &physport, &phy_id, &pdev_hdl, &bay_num, &enclosure,
16517     &io_flags);
16518     if (rval != DDI_SUCCESS) {
16519         mutex_exit(&mpt->m_mutex);
16520         mptsas_log(mpt, CE_WARN, "mptsas unable to get "
16521         "device info for %x", page_address);
16522         ndi_rtn = NDI_FAILURE;
16523         goto smp_create_done;
16524     }
16525     mutex_exit(&mpt->m_mutex);

16527     /*
16528     * If this smp direct attached to the controller
16529     * set the attached-port to the base wwid
16530     */
16531     if ((smp_node->m_deviceinfo & DEVINFO_DIRECT_ATTACHED)
16532     != DEVINFO_DIRECT_ATTACHED) {
16533         (void) sprintf(attached_wnn_str, "w%016"PRIx64,
16534         sas_wnn);
16535     } else {
16536         (void) sprintf(attached_wnn_str, "w%016"PRIx64,
16537         mpt->un.m_base_wwid);
16538     }

16540     if (ndi_prop_update_string(DDI_DEV_T_NONE,
16541     *smp_dip, SCSI_ADDR_PROP_ATTACHED_PORT, attached_wnn_str) !=
16542     DDI_PROP_SUCCESS) {
16543         mptsas_log(mpt, CE_WARN, "mptsas unable to create "
16544         "property for smp attached-port %s (sas_wnn)",
16545         attached_wnn_str);
16546         ndi_rtn = NDI_FAILURE;
16547         goto smp_create_done;
16548     }

16550     if (ndi_prop_create_boolean(DDI_DEV_T_NONE,
16551     *smp_dip, SMP_PROP) != DDI_PROP_SUCCESS) {
16552         mptsas_log(mpt, CE_WARN, "mptsas unable to "
16553         "create property for SMP %s (SMP_PROP) ",
16554         wnn_str);
16555         ndi_rtn = NDI_FAILURE;
16556         goto smp_create_done;
16557     }

16559     /*
16560     * check the smp to see whether it direct
16561     * attached to the controller

```

```

16562     */
16563     if ((smp_node->m_deviceinfo & DEVINFO_DIRECT_ATTACHED)
16564     != DEVINFO_DIRECT_ATTACHED) {
16565         goto smp_create_done;
16566     }
16567     numphys = ddi_prop_get_int(DDI_DEV_T_ANY, pdip,
16568     DDI_PROP_DONTPASS, MPTSAS_NUM_PHYS, -1);
16569     if (numphys > 0) {
16570         goto smp_create_done;
16571     }
16572     /*
16573     * this iport is an old iport, we need to
16574     * reconfig the props for it.
16575     */
16576     if (ddi_prop_update_int(DDI_DEV_T_NONE, pdip,
16577     MPTSAS_VIRTUAL_PORT, 0) !=
16578     DDI_PROP_SUCCESS) {
16579         (void) ddi_prop_remove(DDI_DEV_T_NONE, pdip,
16580         MPTSAS_VIRTUAL_PORT);
16581         mptsas_log(mpt, CE_WARN, "mptsas virtual port "
16582         "prop update failed");
16583         goto smp_create_done;
16584     }

16586     mutex_enter(&mpt->m_mutex);
16587     numphys = 0;
16588     iport = ddi_get_name_addr(pdip);
16589     for (i = 0; i < MPTSAS_MAX_PHYS; i++) {
16590         bzero(phymask, sizeof(phymask));
16591         (void) sprintf(phymask,
16592         "%x", mpt->m_phy_info[i].phy_mask);
16593         if (strcmp(phymask, iport) == 0) {
16594             phy_mask = mpt->m_phy_info[i].phy_mask;
16595             break;
16596         }
16597     }

16599     for (i = 0; i < MPTSAS_MAX_PHYS; i++) {
16600         if ((phy_mask >> i) & 0x01) {
16601             numphys++;
16602         }
16603     }
16604     /*
16605     * Update PHY info for smhba
16606     */
16607     if (mptsas_smhba_phy_init(mpt)) {
16608         mutex_exit(&mpt->m_mutex);
16609         mptsas_log(mpt, CE_WARN, "mptsas phy update "
16610         "failed");
16611         goto smp_create_done;
16612     }
16613     mutex_exit(&mpt->m_mutex);

16615     mptsas_smhba_set_all_phy_props(mpt, pdip, numphys, phy_mask,
16616     &attached_devhdl);

16618     if (ddi_prop_update_int(DDI_DEV_T_NONE, pdip,
16619     MPTSAS_NUM_PHYS, numphys) !=
16620     DDI_PROP_SUCCESS) {
16621         (void) ddi_prop_remove(DDI_DEV_T_NONE, pdip,
16622         MPTSAS_NUM_PHYS);
16623         mptsas_log(mpt, CE_WARN, "mptsas update "
16624         "num phys props failed");
16625         goto smp_create_done;
16626     }
16627     /*

```

```

16628         * Add parent's props for SMHBA support
16629         */
16630         if (ddi_prop_update_string(DDI_DEV_T_NONE, pdip,
16631             SCSI_ADDR_PROP_ATTACHED_PORT, wwn_str) !=
16632             DDI_PROP_SUCCESS) {
16633             (void) ddi_prop_remove(DDI_DEV_T_NONE, pdip,
16634                 SCSI_ADDR_PROP_ATTACHED_PORT);
16635             mptsas_log(mpt, CE_WARN, "mptsas update iport"
16636                 "attached-port failed");
16637             goto smp_create_done;
16638         }
16639
16640 smp_create_done:
16641 /*
16642  * If props were setup ok, online the lun
16643  */
16644         if (ndi_rtn == NDI_SUCCESS) {
16645             /*
16646              * Try to online the new node
16647              */
16648             ndi_rtn = ndi_devi_online(*smp_dip, NDI_ONLINE_ATTACH);
16649         }
16650
16651 /*
16652  * If success set rtn flag, else unwire alloc'd lun
16653  */
16654         if (ndi_rtn != NDI_SUCCESS) {
16655             NDBG12(("mptsas unable to online "
16656                 "SMP target %s", wwn_str));
16657             ndi_prop_remove_all(*smp_dip);
16658             (void) ndi_devi_free(*smp_dip);
16659         }
16660
16661         return ((ndi_rtn == NDI_SUCCESS) ? DDI_SUCCESS : DDI_FAILURE);
16662     }
16663 }
16664
16665 /* smp transport routine */
16666 static int mptsas_smp_start(struct smp_pkt *smp_pkt)
16667 {
16668     uint64_t wwn;
16669     Mpi2SmpPassthroughRequest_t req;
16670     Mpi2SmpPassthroughReply_t rep;
16671     uint8_t direction = 0;
16672     mptsas_t *mpt;
16673     int ret;
16674     uint64_t tmp64;
16675
16676     mpt = (mptsas_t *)smp_pkt->smp_pkt_address->
16677         smp_a_hba_tran->smp_tran_hba_private;
16678
16679     bcopy(smp_pkt->smp_pkt_address->smp_a_wwn, &wwn, SAS_WWN_BYTE_SIZE);
16680     /*
16681      * Need to compose a SMP request message
16682      * and call mptsas_do_passthru() function
16683      */
16684     bzero(&req, sizeof (req));
16685     bzero(&rep, sizeof (rep));
16686     req.PassthroughFlags = 0;
16687     req.PhysicalPort = 0xff;
16688     req.ChainOffset = 0;
16689     req.Function = MPI2_FUNCTION_SMP_PASSTHROUGH;
16690
16691     if ((smp_pkt->smp_pkt_reqsize & 0xffff0000ul) != 0) {
16692         smp_pkt->smp_pkt_reason = ERANGE;
16693         return (DDI_FAILURE);

```

```

16694     }
16695     req.RequestDataLength = LE_16((uint16_t)(smp_pkt->smp_pkt_reqsize - 4));
16696
16697     req.MsgFlags = 0;
16698     tmp64 = LE_64(wwn);
16699     bcopy(&tmp64, &req.SASAddress, SAS_WWN_BYTE_SIZE);
16700     if (smp_pkt->smp_pkt_rspsize > 0) {
16701         direction |= MPTSAS_PASS_THRU_DIRECTION_READ;
16702     }
16703     if (smp_pkt->smp_pkt_reqsize > 0) {
16704         direction |= MPTSAS_PASS_THRU_DIRECTION_WRITE;
16705     }
16706
16707     mutex_enter(&mpt->m_mutex);
16708     ret = mptsas_do_passthru(mpt, (uint8_t *)&req, (uint8_t *)&rep,
16709         (uint8_t *)smp_pkt->smp_pkt_rsp,
16710         offsetof(Mpi2SmpPassthroughRequest_t, SGL), sizeof (rep),
16711         smp_pkt->smp_pkt_rspsize - 4, direction,
16712         (uint8_t *)smp_pkt->smp_pkt_req, smp_pkt->smp_pkt_reqsize - 4,
16713         smp_pkt->smp_pkt_timeout, FKIOCTL);
16714     mutex_exit(&mpt->m_mutex);
16715     if (ret != 0) {
16716         cmn_err(CE_WARN, "smp_start do passthru error %d", ret);
16717         smp_pkt->smp_pkt_reason = (uchar_t)(ret);
16718         return (DDI_FAILURE);
16719     }
16720     /* do passthrough success, check the smp status */
16721     if (LE_16(rep.IOCStatus) != MPI2_IOCSTATUS_SUCCESS) {
16722         switch (LE_16(rep.IOCStatus)) {
16723             case MPI2_IOCSTATUS_SCSI_DEVICE_NOT_THERE:
16724                 smp_pkt->smp_pkt_reason = ENODEV;
16725                 break;
16726             case MPI2_IOCSTATUS_SAS_SMP_DATA_OVERRUN:
16727                 smp_pkt->smp_pkt_reason = EOVERFLOW;
16728                 break;
16729             case MPI2_IOCSTATUS_SAS_SMP_REQUEST_FAILED:
16730                 smp_pkt->smp_pkt_reason = EIO;
16731                 break;
16732             default:
16733                 mptsas_log(mpt, CE_NOTE, "smp_start: get unknown ioc"
16734                     "status:%x", LE_16(rep.IOCStatus));
16735                 smp_pkt->smp_pkt_reason = EIO;
16736                 break;
16737         }
16738         return (DDI_FAILURE);
16739     }
16740     if (rep.SASStatus != MPI2_SASSTATUS_SUCCESS) {
16741         mptsas_log(mpt, CE_NOTE, "smp_start: get error SAS status:%x",
16742             rep.SASStatus);
16743         smp_pkt->smp_pkt_reason = EIO;
16744         return (DDI_FAILURE);
16745     }
16746
16747     return (DDI_SUCCESS);
16748 }
16749
16750 /*
16751  * If we didn't get a match, we need to get sas page0 for each device, and
16752  * untill we get a match. If failed, return NULL
16753  */
16754 static mptsas_target_t *
16755 mptsas_phy_to_tgt(mptsas_t *mpt, mptsas_phymask_t phymask, uint8_t phy)
16756 {
16757     int i, j = 0;
16758     int rval = 0;
16759     uint16_t cur_handle;

```



```

16760     uint32_t      page_address;
16761     mptsas_target_t *ptgt = NULL;

16763     /*
16764     * PHY named device must be direct attached and attaches to
16765     * narrow port, if the iport is not parent of the device which
16766     * we are looking for.
16767     */
16768     for (i = 0; i < MPTSAS_MAX_PHYS; i++) {
16769         if ((1 < i) & phymask)
16770             j++;
16771     }

16773     if (j > 1)
16774         return (NULL);

16776     /*
16777     * Must be a narrow port and single device attached to the narrow port
16778     * So the physical port num of device which is equal to the iport's
16779     * port num is the device what we are looking for.
16780     */

16782     if (mpt->m_phy_info[phy].phy_mask != phymask)
16783         return (NULL);

16785     mutex_enter(&mpt->m_mutex);

16787     ptgt = rehash_linear_search(mpt->m_targets, mptsas_target_eval_nowwn,
16788         &phy);
16789     if (ptgt != NULL) {
16790         mutex_exit(&mpt->m_mutex);
16791         return (ptgt);
16792     }

16794     if (mpt->m_done_traverse_dev) {
16795         mutex_exit(&mpt->m_mutex);
16796         return (NULL);
16797     }

16799     /* If didn't get a match, come here */
16800     cur_handle = mpt->m_dev_handle;
16801     for (; ; ) {
16802         ptgt = NULL;
16803         page_address = (MPI2_SAS_DEVICE_PGAD_FORM_GET_NEXT_HANDLE &
16804             MPI2_SAS_DEVICE_PGAD_FORM_MASK) | (uint32_t)cur_handle;
16805         rval = mptsas_get_target_device_info(mpt, page_address,
16806             &cur_handle, &ptgt);
16807         if ((rval == DEV_INFO_FAIL_PAGE0) ||
16808             (rval == DEV_INFO_FAIL_ALLOC)) {
16809             break;
16810         }
16811         if ((rval == DEV_INFO_WRONG_DEVICE_TYPE) ||
16812             (rval == DEV_INFO_PHYS_DISK)) {
16813             continue;
16814         }
16815         mpt->m_dev_handle = cur_handle;

16817         if ((ptgt->m_addr.mta_wwn == 0) && (ptgt->m_phynum == phy)) {
16818             break;
16819         }
16820     }

16822     mutex_exit(&mpt->m_mutex);
16823     return (ptgt);
16824 }

```

```

16826     /*
16827     * The ptgt->m_addr.mta_wwn contains the wwid for each disk.
16828     * For Raid volumes, we need to check m_raidvol[x].m_raidwwid
16829     * If we didn't get a match, we need to get sas page0 for each device, and
16830     * untill we get a match
16831     * If failed, return NULL
16832     */
16833     static mptsas_target_t *
16834     mptsas_wwid_to_ptgt(mptsas_t *mpt, mptsas_phymask_t phymask, uint64_t wwid)
16835     {
16836         int          rval = 0;
16837         uint16_t      cur_handle;
16838         uint32_t      page_address;
16839         mptsas_target_t *tmp_tgt = NULL;
16840         mptsas_target_addr_t addr;

16842         addr.mta_wwn = wwid;
16843         addr.mta_phymask = phymask;
16844         mutex_enter(&mpt->m_mutex);
16845         tmp_tgt = rehash_lookup(mpt->m_targets, &addr);
16846         if (tmp_tgt != NULL) {
16847             mutex_exit(&mpt->m_mutex);
16848             return (tmp_tgt);
16849         }

16851         if (phymask == 0) {
16852             /*
16853             * It's IR volume
16854             */
16855             rval = mptsas_get_raid_info(mpt);
16856             if (rval) {
16857                 tmp_tgt = rehash_lookup(mpt->m_targets, &addr);
16858             }
16859             mutex_exit(&mpt->m_mutex);
16860             return (tmp_tgt);
16861         }

16863         if (mpt->m_done_traverse_dev) {
16864             mutex_exit(&mpt->m_mutex);
16865             return (NULL);
16866         }

16868         /* If didn't get a match, come here */
16869         cur_handle = mpt->m_dev_handle;
16870         for (; ; ) {
16871             tmp_tgt = NULL;
16872             page_address = (MPI2_SAS_DEVICE_PGAD_FORM_GET_NEXT_HANDLE &
16873                 MPI2_SAS_DEVICE_PGAD_FORM_MASK) | cur_handle;
16874             rval = mptsas_get_target_device_info(mpt, page_address,
16875                 &cur_handle, &tmp_tgt);
16876             if ((rval == DEV_INFO_FAIL_PAGE0) ||
16877                 (rval == DEV_INFO_FAIL_ALLOC)) {
16878                 tmp_tgt = NULL;
16879                 break;
16880             }
16881             if ((rval == DEV_INFO_WRONG_DEVICE_TYPE) ||
16882                 (rval == DEV_INFO_PHYS_DISK)) {
16883                 continue;
16884             }
16885             mpt->m_dev_handle = cur_handle;
16886             if ((tmp_tgt->m_addr.mta_wwn) &&
16887                 (tmp_tgt->m_addr.mta_wwn == wwid) &&
16888                 (tmp_tgt->m_addr.mta_phymask == phymask)) {
16889                 break;
16890             }
16891         }

```

```

16893     mutex_exit(&mpt->m_mutex);
16894     return (tmp_tgt);
16895 }

16897 static mptsas_smp_t *
16898 mptsas_wwid_to_psmpt(mptsas_t *mpt, mptsas_phymask_t phymask, uint64_t wwid)
16899 {
16900     int            rval = 0;
16901     uint16_t       cur_handle;
16902     uint32_t       page_address;
16903     mptsas_smp_t   smp_node, *psmp = NULL;
16904     mptsas_target_addr_t addr;

16906     addr.mta_wwn = wwid;
16907     addr.mta_phymask = phymask;
16908     mutex_enter(&mpt->m_mutex);
16909     psmp = rehash_lookup(mpt->m_smp_targets, &addr);
16910     if (psmp != NULL) {
16911         mutex_exit(&mpt->m_mutex);
16912         return (psmp);
16913     }

16915     if (mpt->m_done_traverse_smp) {
16916         mutex_exit(&mpt->m_mutex);
16917         return (NULL);
16918     }

16920     /* If didn't get a match, come here */
16921     cur_handle = mpt->m_smp_devhdl;
16922     for (;;) {
16923         psmp = NULL;
16924         page_address = (MPI2_SAS_EXPAND_PGAD_FORM_GET_NEXT_HNDL &
16925             MPI2_SAS_EXPAND_PGAD_FORM_MASK) | (uint32_t)cur_handle;
16926         rval = mptsas_get_sas_expander_page0(mpt, page_address,
16927             &smp_node);
16928         if (rval != DDI_SUCCESS) {
16929             break;
16930         }
16931         mpt->m_smp_devhdl = cur_handle = smp_node.m_devhdl;
16932         psmp = mptsas_smp_alloc(mpt, &smp_node);
16933         ASSERT(psmp);
16934         if ((psmp->m_addr.mta_wwn) && (psmp->m_addr.mta_wwn == wwid) &&
16935             (psmp->m_addr.mta_phymask == phymask)) {
16936             break;
16937         }
16938     }

16940     mutex_exit(&mpt->m_mutex);
16941     return (psmp);
16942 }

16944 mptsas_target_t *
16945 mptsas_tgt_alloc(mptsas_t *mpt, uint16_t devhdl, uint64_t wwid,
16946     uint32_t devinfo, mptsas_phymask_t phymask, uint8_t phynum)
16947 {
16948     mptsas_target_t *tmp_tgt = NULL;
16949     mptsas_target_addr_t addr;

16951     addr.mta_wwn = wwid;
16952     addr.mta_phymask = phymask;
16953     tmp_tgt = rehash_lookup(mpt->m_smp_targets, &addr);
16954     if (tmp_tgt != NULL) {
16955         NDBG20(("Hash item already exist"));
16956         tmp_tgt->m_deviceinfo = devinfo;
16957         tmp_tgt->m_devhdl = devhdl; /* XXX - duplicate? */

```

```

16958         return (tmp_tgt);
16959     }
16960     tmp_tgt = kmem_zalloc(sizeof (struct mptsas_target), KM_SLEEP);
16961     if (tmp_tgt == NULL) {
16962         cmn_err(CE_WARN, "Fatal, allocated tgt failed");
16963         return (NULL);
16964     }
16965     tmp_tgt->m_devhdl = devhdl;
16966     tmp_tgt->m_addr.mta_wwn = wwid;
16967     tmp_tgt->m_deviceinfo = devinfo;
16968     tmp_tgt->m_addr.mta_phymask = phymask;
16969     tmp_tgt->m_phynum = phynum;
16970     /* Initialized the tgt structure */
16971     tmp_tgt->m_qfull_retries = QFULL_RETRIES;
16972     tmp_tgt->m_qfull_retry_interval =
16973         drv_usectohz(QFULL_RETRY_INTERVAL * 1000);
16974     tmp_tgt->m_t_throttle = MAX_THROTTLE;
16975     mutex_init(&tmp_tgt->m_t_mutex, NULL, MUTEX_DRIVER, NULL);
16976     TAILQ_INIT(&tmp_tgt->m_active_cmdq);

16978     rehash_insert(mpt->m_smp_targets, tmp_tgt);

16980     return (tmp_tgt);
16981 }

16983 static void
16984 mptsas_smp_target_copy(mptsas_smp_t *src, mptsas_smp_t *dst)
16985 {
16986     dst->m_devhdl = src->m_devhdl;
16987     dst->m_deviceinfo = src->m_deviceinfo;
16988     dst->m_pdevhdl = src->m_pdevhdl;
16989     dst->m_pdevinfo = src->m_pdevinfo;
16990 }

16992 static mptsas_smp_t *
16993 mptsas_smp_alloc(mptsas_t *mpt, mptsas_smp_t *data)
16994 {
16995     mptsas_target_addr_t addr;
16996     mptsas_smp_t *ret_data;

16998     addr.mta_wwn = data->m_addr.mta_wwn;
16999     addr.mta_phymask = data->m_addr.mta_phymask;
17000     ret_data = rehash_lookup(mpt->m_smp_targets, &addr);
17001     /*
17002      * If there's already a matching SMP target, update its fields
17003      * in place. Since the address is not changing, it's safe to do
17004      * this. We cannot just bcopy() here because the structure we've
17005      * been given has invalid hash links.
17006      */
17007     if (ret_data != NULL) {
17008         mptsas_smp_target_copy(data, ret_data);
17009         return (ret_data);
17010     }

17012     ret_data = kmem_alloc(sizeof (mptsas_smp_t), KM_SLEEP);
17013     bcopy(data, ret_data, sizeof (mptsas_smp_t));
17014     rehash_insert(mpt->m_smp_targets, ret_data);
17015     return (ret_data);
17016 }

17018 /*
17019  * Functions for SGPIO LED support
17020  */
17021 static dev_info_t *
17022 mptsas_get_dip_from_dev(dev_t dev, mptsas_phymask_t *phymask)
17023 {

```

```

17024     dev_info_t      *dip;
17025     int              prop;
17026     dip = e_ddi_hold_devi_by_dev(dev, 0);
17027     if (dip == NULL)
17028         return (dip);
17029     prop = ddi_prop_get_int(DDI_DEV_T_ANY, dip, 0,
17030         "phymask", 0);
17031     *phymask = (mptsas_phymask_t)prop;
17032     ddi_release_devi(dip);
17033     return (dip);
17034 }
17035 static mptsas_target_t *
17036 mptsas_addr_to_ptgt(mptsas_t *mpt, char *addr, mptsas_phymask_t phymask)
17037 {
17038     uint8_t          phynum;
17039     uint64_t          wwn;
17040     int              lun;
17041     mptsas_target_t  *ptgt = NULL;

17043     if (mptsas_parse_address(addr, &wwn, &phynum, &lun) != DDI_SUCCESS) {
17044         return (NULL);
17045     }
17046     if (addr[0] == 'w') {
17047         ptgt = mptsas_wwid_to_ptgt(mpt, (int)phymask, wwn);
17048     } else {
17049         ptgt = mptsas_phy_to_tgt(mpt, (int)phymask, phynum);
17050     }
17051     return (ptgt);
17052 }

17054 static int
17055 mptsas_flush_led_status(mptsas_t *mpt, mptsas_target_t *ptgt)
17056 {
17057     uint32_t slotstatus = 0;

17059     /* Build an MPI2 Slot Status based on our view of the world */
17060     if (ptgt->m_led_status & (1 << (MPTSAS_LEDCTL_LED_IDENT - 1)))
17061         slotstatus |= MPI2_SEP_REQ_SLOTSTATUS_IDENTIFY_REQUEST;
17062     if (ptgt->m_led_status & (1 << (MPTSAS_LEDCTL_LED_FAIL - 1)))
17063         slotstatus |= MPI2_SEP_REQ_SLOTSTATUS_PREDICTED_FAULT;
17064     if (ptgt->m_led_status & (1 << (MPTSAS_LEDCTL_LED_OK2RM - 1)))
17065         slotstatus |= MPI2_SEP_REQ_SLOTSTATUS_REQUEST_REMOVE;

17067     /* Write it to the controller */
17068     NDBG14(("mptsas ioctl: set LED status %x for slot %x",
17069         slotstatus, ptgt->m_slot_num));
17070     return (mptsas_send_sep(mpt, ptgt, &slotstatus,
17071         MPI2_SEP_REQ_ACTION_WRITE_STATUS));
17072 }

17074 /*
17075  * send sep request, use enclosure/slot addressing
17076  */
17077 static int
17078 mptsas_send_sep(mptsas_t *mpt, mptsas_target_t *ptgt,
17079     uint32_t *status, uint8_t act)
17080 {
17081     Mpi2SepRequest_t req;
17082     Mpi2SepReply_t   rep;
17083     int              ret;

17085     ASSERT(mutex_owned(&mpt->m_mutex));

17087     /*
17088      * We only support SEP control of directly-attached targets, in which
17089      * case the "SEP" we're talking to is a virtual one contained within

```

```

17090     * the HBA itself. This is necessary because DA targets typically have
17091     * no other mechanism for LED control. Targets for which a separate
17092     * enclosure service processor exists should be controlled via ses(7d)
17093     * or sgen(7d). Furthermore, since such requests can time out, they
17094     * should be made in user context rather than in response to
17095     * asynchronous fabric changes.
17096     *
17097     * In addition, we do not support this operation for RAID volumes,
17098     * since there is no slot associated with them.
17099     */
17100     if (! (ptgt->m_deviceinfo & DEVINFO_DIRECT_ATTACHED) ||
17101         ptgt->m_addr.mta_phymask == 0) {
17102         return (ENOTTY);
17103     }

17105     bzero(&req, sizeof (req));
17106     bzero(&rep, sizeof (rep));

17108     req.Function = MPI2_FUNCTION_SCSI_ENCLOSURE_PROCESSOR;
17109     req.Action = act;
17110     req.Flags = MPI2_SEP_REQ_FLAGS_ENCLOSURE_SLOT_ADDRESS;
17111     req.EnclosureHandle = LE_16(ptgt->m_enclosure);
17112     req.Slot = LE_16(ptgt->m_slot_num);
17113     if (act == MPI2_SEP_REQ_ACTION_WRITE_STATUS) {
17114         req.SlotStatus = LE_32(*status);
17115     }
17116     ret = mptsas_do_passthru(mpt, (uint8_t *)&req, (uint8_t *)&rep, NULL,
17117         sizeof (req), sizeof (rep), NULL, 0, NULL, 0, 60, FKIOCTL);
17118     if (ret != 0) {
17119         mptsas_log(mpt, CE_NOTE, "mptsas_send_sep: passthru SEP "
17120             "Processor Request message error %d", ret);
17121         return (ret);
17122     }

17123     /* do passthrough success, check the ioc status */
17124     if (LE_16(rep.IOCStatus) != MPI2_IOCSTATUS_SUCCESS) {
17125         mptsas_log(mpt, CE_NOTE, "send_sep act %x: ioc "
17126             "status: %x loginfo %x", act, LE_16(rep.IOCStatus),
17127             LE_32(rep.IOCLogInfo));
17128         switch (LE_16(rep.IOCStatus) & MPI2_IOCSTATUS_MASK) {
17129             case MPI2_IOCSTATUS_INVALID_FUNCTION:
17130             case MPI2_IOCSTATUS_INVALID_VPID:
17131             case MPI2_IOCSTATUS_INVALID_FIELD:
17132             case MPI2_IOCSTATUS_INVALID_STATE:
17133             case MPI2_IOCSTATUS_OP_STATE_NOT_SUPPORTED:
17134             case MPI2_IOCSTATUS_CONFIG_INVALID_ACTION:
17135             case MPI2_IOCSTATUS_CONFIG_INVALID_TYPE:
17136             case MPI2_IOCSTATUS_CONFIG_INVALID_PAGE:
17137             case MPI2_IOCSTATUS_CONFIG_INVALID_DATA:
17138             case MPI2_IOCSTATUS_CONFIG_NO_DEFAULTS:
17139                 return (EINVAL);
17140             case MPI2_IOCSTATUS_BUSY:
17141                 return (EBUSY);
17142             case MPI2_IOCSTATUS_INSUFFICIENT_RESOURCES:
17143                 return (EAGAIN);
17144             case MPI2_IOCSTATUS_INVALID_SGL:
17145             case MPI2_IOCSTATUS_INTERNAL_ERROR:
17146             case MPI2_IOCSTATUS_CONFIG_CANT_COMMIT:
17147             default:
17148                 return (EIO);
17149         }
17150     }
17151     if (act != MPI2_SEP_REQ_ACTION_WRITE_STATUS) {
17152         *status = LE_32(rep.SlotStatus);
17153     }

17155     return (0);

```

```
17156 }

17158 int
17159 mptsas_dma_addr_create(mptsas_t *mpt, ddi_dma_attr_t dma_attr,
17160     ddi_dma_handle_t *dma_hdl, ddi_acc_handle_t *acc_hdl, caddr_t *dma_memp,
17161     uint32_t alloc_size, ddi_dma_cookie_t *cookiep)
17162 {
17163     ddi_dma_cookie_t    new_cookie;
17164     size_t              alloc_len;
17165     uint_t              ncookie;

17167     if (cookiep == NULL)
17168         cookiep = &new_cookie;

17170     if (ddi_dma_alloc_handle(mpt->m_dip, &dma_attr, DDI_DMA_SLEEP,
17171         NULL, dma_hdl) != DDI_SUCCESS) {
17172         return (FALSE);
17173     }

17175     if (ddi_dma_mem_alloc(*dma_hdl, alloc_size, &mpt->m_dev_acc_attr,
17176         DDI_DMA_CONSISTENT, DDI_DMA_SLEEP, NULL, dma_memp, &alloc_len,
17177         acc_hdl) != DDI_SUCCESS) {
17178         ddi_dma_free_handle(dma_hdl);
17179         return (FALSE);
17180     }

17182     if (ddi_dma_addr_bind_handle(*dma_hdl, NULL, *dma_memp, alloc_len,
17183         (DDI_DMA_RDWR | DDI_DMA_CONSISTENT), DDI_DMA_SLEEP, NULL,
17184         cookiep, &ncookie) != DDI_DMA_MAPPED) {
17185         (void) ddi_dma_mem_free(acc_hdl);
17186         ddi_dma_free_handle(dma_hdl);
17187         return (FALSE);
17188     }

17190     return (TRUE);
17191 }

17193 void
17194 mptsas_dma_addr_destroy(ddi_dma_handle_t *dma_hdl, ddi_acc_handle_t *acc_hdl)
17195 {
17196     if (*dma_hdl == NULL)
17197         return;

17199     (void) ddi_dma_unbind_handle(*dma_hdl);
17200     (void) ddi_dma_mem_free(acc_hdl);
17201     ddi_dma_free_handle(dma_hdl);
17202 }
17203 #endif /* ! codereview */
```

new/usr/src/uts/common/io/scsi/adapters/mpt_sas3/mptsas3_hash.c

1

```
*****
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
*****

1 /*
2  * This file and its contents are supplied under the terms of the
3  * Common Development and Distribution License ("CDDL"), version 1.0.
4  * You may only use this file in accordance with the terms of version
5  * 1.0 of the CDDL.
6  *
7  * A full copy of the text of the CDDL should have accompanied this
8  * source. A copy of the CDDL is also available via the Internet at
9  * http://www.illumos.org/license/CDDL.
10 */

12 /*
13  * Copyright 2014 Joyent, Inc. All rights reserved.
14  * Copyright (c) 2014, Tegile Systems Inc. All rights reserved.
15  */

17 #include <sys/scsi/adapters/mpt_sas3/mptsas3_hash.h>
18 #include <sys/sysmacros.h>
19 #include <sys/types.h>
20 #include <sys/kmem.h>
21 #include <sys/list.h>
22 #include <sys/ddi.h>

24 #ifdef lint
25 extern rehash_link_t *obj_to_link(rehash_t *, void *);
26 extern void *link_to_obj(rehash_t *, rehash_link_t *);
27 extern void *obj_to_tag(rehash_t *, void *);
28 #else
29 #define obj_to_link(h, _o) \
30 ((rehash_link_t *)(((char *)(_o)) + (_h)->rh_link_off))
31 #define link_to_obj(h, _l) \
32 ((void *)(((char *)(_l)) - (_h)->rh_link_off))
33 #define obj_to_tag(h, _o) \
34 ((void *)(((char *)(_o)) + (_h)->rh_tag_off))
35 #endif

37 rehash_t *
38 rehash_create(uint_t bucket_count, rehash_hash_f hash,
39 rehash_cmp_f cmp, rehash_dtor_f dtor, size_t obj_size, size_t link_off,
40 size_t tag_off, int km_flags)
41 {
42     rehash_t *hp;
43     uint_t i;

45     hp = kmem_alloc(sizeof (rehash_t), km_flags);
46     if (hp == NULL)
47         return (NULL);
48     hp->rh_buckets = kmem_zalloc(bucket_count * sizeof (list_t), km_flags);
49     if (hp->rh_buckets == NULL) {
50         kmem_free(hp, sizeof (rehash_t));
51         return (NULL);
52     }
53     hp->rh_bucket_count = bucket_count;

55     for (i = 0; i < bucket_count; i++) {
56         list_create(&hp->rh_buckets[i], sizeof (rehash_link_t),
57             offsetof(rehash_link_t, rhl_chain_link));
58     }
59     list_create(&hp->rh_objs, sizeof (rehash_link_t),
60         offsetof(rehash_link_t, rhl_global_link));
```

new/usr/src/uts/common/io/scsi/adapters/mpt_sas3/mptsas3_hash.c

2

```
62     hp->rh_obj_size = obj_size;
63     hp->rh_link_off = link_off;
64     hp->rh_tag_off = tag_off;
65     hp->rh_hash = hash;
66     hp->rh_cmp = cmp;
67     hp->rh_dtor = dtor;

69     return (hp);
70 }

72 void
73 rehash_destroy(rehash_t *hp)
74 {
75     ASSERT(list_is_empty(&hp->rh_objs));

77     kmem_free(hp->rh_buckets, hp->rh_bucket_count * sizeof (list_t));
78     kmem_free(hp, sizeof (rehash_t));
79 }

81 void
82 rehash_insert(rehash_t *hp, void *op)
83 {
84     uint_t bucket;
85     rehash_link_t *lp = obj_to_link(hp, op);

87     bucket = hp->rh_hash(obj_to_tag(hp, op)) % hp->rh_bucket_count;
88     list_link_init(&lp->rhl_chain_link);
89     list_link_init(&lp->rhl_global_link);
90     lp->rhl_flags = 0;
91     lp->rhl_refcnt = 0;
92     list_insert_tail(&hp->rh_buckets[bucket], lp);
93     list_insert_tail(&hp->rh_objs, lp);
94 }

96 static void
97 rehash_delete(rehash_t *hp, void *op)
98 {
99     rehash_link_t *lp = obj_to_link(hp, op);
100     uint_t bucket;

102     bucket = hp->rh_hash(obj_to_tag(hp, op)) % hp->rh_bucket_count;
103     list_remove(&hp->rh_buckets[bucket], lp);
104     list_remove(&hp->rh_objs, lp);
105     hp->rh_dtor(op);
106 }

108 void
109 rehash_remove(rehash_t *hp, void *op)
110 {
111     rehash_link_t *lp = obj_to_link(hp, op);

113     if (lp->rhl_refcnt > 0) {
114         lp->rhl_flags |= RHL_F_DEAD;
115     } else {
116         rehash_delete(hp, op);
117     }
118 }

120 void *
121 rehash_lookup(rehash_t *hp, const void *tp)
122 {
123     uint_t bucket;
124     rehash_link_t *lp;
125     void *op;

127     bucket = hp->rh_hash(tp) % hp->rh_bucket_count;
```

```

128     for (lp = list_head(&hp->rh_buckets[bucket]); lp != NULL;
129         lp = list_next(&hp->rh_buckets[bucket], lp)) {
130         op = link_to_obj(hp, lp);
131         if (hp->rh_cmp(obj_to_tag(hp, op), tp) == 0 &&
132             !(lp->rhl_flags & RHL_F_DEAD)) {
133             return (op);
134         }
135     }
137     return (NULL);
138 }

140 void *
141 rehash_linear_search(rehash_t *hp, rehash_eval_f eval, void *arg)
142 {
143     void *op;
144     rehash_link_t *lp;

146     for (lp = list_head(&hp->rh_objs); lp != NULL;
147         lp = list_next(&hp->rh_objs, lp)) {
148         op = link_to_obj(hp, lp);
149         if (eval(op, arg) == 0)
150             return (op);
151     }

153     return (NULL);
154 }

156 void
157 rehash_hold(rehash_t *hp, void *op)
158 {
159     rehash_link_t *lp = obj_to_link(hp, op);

161     ++lp->rhl_refcnt;
162 }

164 void
165 rehash_rele(rehash_t *hp, void *op)
166 {
167     rehash_link_t *lp = obj_to_link(hp, op);

169     ASSERT(lp->rhl_refcnt > 0);

171     if (--lp->rhl_refcnt == 0 && (lp->rhl_flags & RHL_F_DEAD))
172         rehash_remove(hp, op);
173 }

175 void *
176 rehash_first(rehash_t *hp)
177 {
178     rehash_link_t *lp;

180     lp = list_head(&hp->rh_objs);
181     if (lp == NULL)
182         return (NULL);

184     ++lp->rhl_refcnt;

186     return (link_to_obj(hp, lp));
187 }

189 void *
190 rehash_next(rehash_t *hp, void *op)
191 {
192     rehash_link_t *lp;

```

```

194     lp = obj_to_link(hp, op);
195     while ((lp = list_next(&hp->rh_objs, lp)) != NULL) {
196         if (!(lp->rhl_flags & RHL_F_DEAD))
197             break;
198     }

200     rehash_rele(hp, op);
201     if (lp == NULL)
202         return (NULL);

204     ++lp->rhl_refcnt;

206     return (link_to_obj(hp, lp));
207 }

209 boolean_t
210 rehash_obj_valid(rehash_t *hp, const void *op)
211 {
212     /* LINTED - E_ARG_INCOMPATIBLE_WITH_ARG_L */
213     const rehash_link_t *lp = obj_to_link(hp, op);

215     return ((lp->rhl_flags & RHL_F_DEAD) != 0);
216 }
217 #endif /* ! codereview */

```

new/usr/src/uts/common/io/scsi/adapters/mpt_sas3/mptsas3_impl.c

1

```
*****
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
*****

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

22 /*
23  * Copyright (c) 2009, 2010, Oracle and/or its affiliates. All rights reserved.
24  * Copyright 2012 Nexenta Systems, Inc. All rights reserved.
25  * Copyright (c) 2014, Tegile Systems Inc. All rights reserved.
26  * Copyright 2014 OmniTI Computer Consulting, Inc. All rights reserved.
27 */

29 /*
30  * Copyright (c) 2000 to 2010, LSI Corporation.
31  * All rights reserved.
32  *
33  * Redistribution and use in source and binary forms of all code within
34  * this file that is exclusively owned by LSI, with or without
35  * modification, is permitted provided that, in addition to the CDDL 1.0
36  * License requirements, the following conditions are met:
37  *
38  *     Neither the name of the author nor the names of its contributors may be
39  *     used to endorse or promote products derived from this software without
40  *     specific prior written permission.
41  *
42  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
43  * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
44  * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
45  * FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
46  * COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
47  * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
48  * BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
49  * OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
50  * AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
51  * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
52  * OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
53  * DAMAGE.
54 */

56 /*
57  * mptsas_impl - This file contains all the basic functions for communicating
58  * to MPT based hardware.
59 */

61 #if defined(lint) || defined(DEBUG)
```

new/usr/src/uts/common/io/scsi/adapters/mpt_sas3/mptsas3_impl.c

2

```
62 #define MPTSAS_DEBUG
63 #endif

65 /*
66  * standard header files
67 */
68 #include <sys/note.h>
69 #include <sys/scsi/scsi.h>
70 #include <sys/pci.h>

72 #pragma pack(1)
73 #include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_type.h>
74 #include <sys/scsi/adapters/mpt_sas3/mpi/mpi2.h>
75 #include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_cnfg.h>
76 #include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_init.h>
77 #include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_ioc.h>
78 #include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_sas.h>
79 #include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_tool.h>
80 #pragma pack()

82 /*
83  * private header files.
84 */
85 #include <sys/scsi/adapters/mpt_sas3/mptsas3_var.h>
86 #include <sys/scsi/adapters/mpt_sas3/mptsas3_smhba.h>

88 /*
89  * FMA header files.
90 */
91 #include <sys/fm/io/ddi.h>

93 #if defined(MPTSAS_DEBUG)
94 extern uint32_t mptsas_debug_flags;
95 extern uint32_t mptsas_dbglog_imask;
96 #endif

98 /*
99  * prototypes
100 */
101 static void mptsas_ioc_event_cmdq_add(mptsas_t *mpt, m_event_struct_t *cmd);
102 static void mptsas_ioc_event_cmdq_delete(mptsas_t *mpt, m_event_struct_t *cmd);
103 static m_event_struct_t *mptsas_ioc_event_find_by_cmd(mptsas_t *mpt,
104     struct mptsas_cmd *cmd);

106 /*
107  * add ioc evnet cmd into the queue
108 */
109 static void
110 mptsas_ioc_event_cmdq_add(mptsas_t *mpt, m_event_struct_t *cmd)
111 {
112     if ((cmd->m_event_linkp = mpt->m_ioc_event_cmdq) == NULL) {
113         mpt->m_ioc_event_cmdtail = &cmd->m_event_linkp;
114         mpt->m_ioc_event_cmdq = cmd;
115     } else {
116         cmd->m_event_linkp = NULL;
117         *(mpt->m_ioc_event_cmdtail) = cmd;
118         mpt->m_ioc_event_cmdtail = &cmd->m_event_linkp;
119     }
120 }

122 /*
123  * remove specified cmd from the ioc event queue
124 */
125 static void
126 mptsas_ioc_event_cmdq_delete(mptsas_t *mpt, m_event_struct_t *cmd)
127 {
```

```

128     m_event_struct_t      *prev = mpt->m_ioc_event_cmdq;
129     if (prev == cmd) {
130         if ((mpt->m_ioc_event_cmdq = cmd->m_event_linkp) == NULL) {
131             mpt->m_ioc_event_cmdtail = &mpt->m_ioc_event_cmdq;
132         }
133         cmd->m_event_linkp = NULL;
134         return;
135     }
136     while (prev != NULL) {
137         if (prev->m_event_linkp == cmd) {
138             prev->m_event_linkp = cmd->m_event_linkp;
139             if (cmd->m_event_linkp == NULL) {
140                 mpt->m_ioc_event_cmdtail = &prev->m_event_linkp;
141             }
142             cmd->m_event_linkp = NULL;
143             return;
144         }
145         prev = prev->m_event_linkp;
146     }
147 }
148
150 static m_event_struct_t *
151 mptsas_ioc_event_find_by_cmd(mptsas_t *mpt, struct mptsas_cmd *cmd)
152 {
153     m_event_struct_t      *ioc_cmd = NULL;
154
155     ioc_cmd = mpt->m_ioc_event_cmdq;
156     while (ioc_cmd != NULL) {
157         if (&(ioc_cmd->m_event_cmd) == cmd) {
158             return (ioc_cmd);
159         }
160         ioc_cmd = ioc_cmd->m_event_linkp;
161     }
162     ioc_cmd = NULL;
163     return (ioc_cmd);
164 }
165
166 void
167 mptsas_destroy_ioc_event_cmd(mptsas_t *mpt)
168 {
169     m_event_struct_t      *ioc_cmd = NULL;
170     m_event_struct_t      *ioc_cmd_tmp = NULL;
171     ioc_cmd = mpt->m_ioc_event_cmdq;
172
173     /*
174      * because the IOC event queue is resource of per instance for driver,
175      * it's not only ACK event commands used it, but also some others used
176      * it. We need destroy all ACK event commands when IOC reset, but can't
177      * disturb others. So we use filter to clear the ACK event cmd in ioc
178      * event queue, and other requests should be reserved, and they would
179      * be free by its owner.
180      */
181     while (ioc_cmd != NULL) {
182         if (ioc_cmd->m_event_cmd.cmd_flags & CFLAG_CMDACK) {
183             NDBG20(("destroy!! remove Ack Flag ioc_cmd\n"));
184             if ((mpt->m_ioc_event_cmdq =
185                 ioc_cmd->m_event_linkp) == NULL)
186                 mpt->m_ioc_event_cmdtail =
187                     &mpt->m_ioc_event_cmdq;
188             ioc_cmd_tmp = ioc_cmd;
189             ioc_cmd = ioc_cmd->m_event_linkp;
190             kmem_free(ioc_cmd_tmp, M_EVENT_STRUCT_SIZE);
191         } else {
192             /*
193              * it's not ack cmd, so continue to check next one

```

```

194         */
195
196         NDBG20(("destroy!! it's not Ack Flag, continue\n"));
197         ioc_cmd = ioc_cmd->m_event_linkp;
198     }
199
200 }
201
203 void
204 mptsas_start_config_page_access(mptsas_t *mpt, mptsas_cmd_t *cmd)
205 {
206     pMpi2ConfigRequest_t    request;
207     pMpi2SGESimple64_t      sge;
208     struct scsi_pkt          *pkt = cmd->cmd_pkt;
209     mptsas_config_request_t *config = pkt->pkt_ha_private;
210     uint8_t                  direction;
211     uint32_t                  length, flagslength;
212     uint64_t                  request_desc;
213
214     ASSERT(mutex_owned(&mpt->m_mutex));
215
216     /*
217      * Point to the correct message and clear it as well as the global
218      * config page memory.
219      */
220     request = (pMpi2ConfigRequest_t)(mpt->m_req_frame +
221         (mpt->m_req_frame_size * cmd->cmd_slot));
222     bzero(request, mpt->m_req_frame_size);
223
224     /*
225      * Form the request message.
226      */
227     ddi_put8(mpt->m_acc_req_frame_hdl, &request->Function,
228         MPI2_FUNCTION_CONFIG);
229     ddi_put8(mpt->m_acc_req_frame_hdl, &request->Action, config->action);
230     direction = MPI2_SGE_FLAGS_IOC_TO_HOST;
231     length = 0;
232     sge = (pMpi2SGESimple64_t)&request->PageBufferSGE;
233     if (config->action == MPI2_CONFIG_ACTION_PAGE_HEADER) {
234         if (config->page_type > MPI2_CONFIG_PAGETYPE_MASK) {
235             ddi_put8(mpt->m_acc_req_frame_hdl,
236                 &request->Header.PageType,
237                 MPI2_CONFIG_PAGETYPE_EXTENDED);
238             ddi_put8(mpt->m_acc_req_frame_hdl,
239                 &request->ExtPageType, config->page_type);
240         } else {
241             ddi_put8(mpt->m_acc_req_frame_hdl,
242                 &request->Header.PageType, config->page_type);
243         }
244     } else {
245         ddi_put8(mpt->m_acc_req_frame_hdl, &request->ExtPageType,
246             config->ext_page_type);
247         ddi_put16(mpt->m_acc_req_frame_hdl, &request->ExtPageLength,
248             config->ext_page_length);
249         ddi_put8(mpt->m_acc_req_frame_hdl, &request->Header.PageType,
250             config->page_type);
251         ddi_put8(mpt->m_acc_req_frame_hdl, &request->Header.PageLength,
252             config->page_length);
253         ddi_put8(mpt->m_acc_req_frame_hdl,
254             &request->Header.PageVersion, config->page_version);
255         if ((config->page_type & MPI2_CONFIG_PAGETYPE_MASK) ==
256             MPI2_CONFIG_PAGETYPE_EXTENDED) {
257             length = config->ext_page_length * 4;
258         } else {
259             length = config->page_length * 4;

```



```

260     }
261
262     if (config->action == MPI2_CONFIG_ACTION_PAGE_WRITE_NVRAM) {
263         direction = MPI2_SGE_FLAGS_HOST_TO_IOC;
264     }
265     ddi_put32(mpt->m_acc_req_frame_hdl, &sge->Address.Low,
266             (uint32_t)(cmd->cmd_dma_addr&0xffffffff));
267     ddi_put32(mpt->m_acc_req_frame_hdl, &sge->Address.High,
268             (uint32_t)(cmd->cmd_dma_addr >> 32));
269 }
270 ddi_put8(mpt->m_acc_req_frame_hdl, &request->Header.PageNumber,
271         config->page_number);
272 ddi_put32(mpt->m_acc_req_frame_hdl, &request->PageAddress,
273         config->page_address);
274 flagslength = ((uint32_t)(MPI2_SGE_FLAGS_LAST_ELEMENT |
275         MPI2_SGE_FLAGS_END_OF_BUFFER |
276         MPI2_SGE_FLAGS_SIMPLE_ELEMENT |
277         MPI2_SGE_FLAGS_SYSTEM_ADDRESS |
278         MPI2_SGE_FLAGS_64_BIT_ADDRESSING |
279         direction |
280         MPI2_SGE_FLAGS_END_OF_LIST) << MPI2_SGE_FLAGS_SHIFT);
281 flagslength |= length;
282 ddi_put32(mpt->m_acc_req_frame_hdl, &sge->FlagsLength, flagslength);
283
284 (void) ddi_dma_sync(mpt->m_dma_req_frame_hdl, 0, 0,
285         DDI_DMA_SYNC_FORDEV);
286 request_desc = (cmd->cmd_slot << 16) +
287         MPI2_REQ_DESCRIPTOR_FLAGS_DEFAULT_TYPE;
288 cmd->cmd_rfm = NULL;
289 MPTSAS_START_CMD(mpt, request_desc);
290 if ((mptsas_check_dma_handle(mpt->m_dma_req_frame_hdl) !=
291     DDI_SUCCESS) ||
292     (mptsas_check_acc_handle(mpt->m_acc_req_frame_hdl) !=
293     DDI_SUCCESS)) {
294     ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_UNAFFECTED);
295 }
296 }
297
298 int
299 mptsas_access_config_page(mptsas_t *mpt, uint8_t action, uint8_t page_type,
300     uint8_t page_number, uint32_t page_address, int (*callback) (mptsas_t *,
301     caddr_t, ddi_acc_handle_t, uint16_t, uint32_t, va_list), ...)
302 {
303     va_list          ap;
304     ddi_dma_attr_t   attrs;
305     ddi_dma_cookie_t cookie;
306     ddi_acc_handle_t accessp;
307     size_t           len = 0;
308     mptsas_config_request_t config;
309     int              rval = DDI_SUCCESS, config_flags = 0;
310     mptsas_cmd_t     *cmd;
311     struct scsi_pkt  *pkt;
312     pMpi2ConfigReply_t reply;
313     uint16_t         iocstatus = 0;
314     uint32_t         iocloginfo;
315     caddr_t          page_memp;
316     boolean_t         free_dma = B_FALSE;
317
318     va_start(ap, callback);
319     ASSERT(mutex_owned(&mpt->m_mutex));
320
321     /*
322      * Get a command from the pool.
323      */
324     if ((rval = (mptsas_request_from_pool(mpt, &cmd, &pkt))) == -1) {
325         mptsas_log(mpt, CE_NOTE, "command pool is full for config "

```

```

326         "page request");
327         rval = DDI_FAILURE;
328         goto page_done;
329     }
330     config_flags |= MPTSAS_REQUEST_POOL_CMD;
331
332     bzero((caddr_t)cmd, sizeof (*cmd));
333     bzero((caddr_t)pkt, scsi_pkt_size());
334     bzero((caddr_t)&config, sizeof (config));
335
336     /*
337      * Save the data for this request to be used in the call to start the
338      * config header request.
339      */
340     config.action = MPI2_CONFIG_ACTION_PAGE_HEADER;
341     config.page_type = page_type;
342     config.page_number = page_number;
343     config.page_address = page_address;
344
345     /*
346      * Form a blank cmd/pkt to store the acknowledgement message
347      */
348     pkt->pkt_ha_private = (opaque_t)&config;
349     pkt->pkt_flags = FLAG_HEAD;
350     pkt->pkt_time = 60;
351     cmd->cmd_pkt = pkt;
352     cmd->cmd_flags = CFLAG_CMDIOC | CFLAG_CONFIG;
353
354     /*
355      * Save the config header request message in a slot.
356      */
357     if (mptsas_save_cmd(mpt, cmd) == TRUE) {
358         cmd->cmd_flags |= CFLAG_PREPARED;
359         mptsas_start_config_page_access(mpt, cmd);
360     } else {
361         mptsas_waitq_add(mpt, cmd);
362     }
363
364     /*
365      * If this is a request for a RAID info page, or any page called during
366      * the RAID info page request, poll because these config page requests
367      * are nested. Poll to avoid data corruption due to one page's data
368      * overwriting the outer page request's data. This can happen when
369      * the mutex is released in cv_wait.
370      */
371     if ((page_type == MPI2_CONFIG_EXTPAGE_TYPE_RAID_CONFIG) ||
372         (page_type == MPI2_CONFIG_PAGE_TYPE_RAID_VOLUME) ||
373         (page_type == MPI2_CONFIG_PAGE_TYPE_RAID_PHYSDISK)) {
374         (void) mptsas_poll(mpt, cmd, pkt->pkt_time * 1000);
375     } else {
376         while ((cmd->cmd_flags & CFLAG_FINISHED) == 0) {
377             cv_wait(&mpt->m_config_cv, &mpt->m_mutex);
378         }
379     }
380
381     /*
382      * Check if the header request completed without timing out
383      */
384     if (cmd->cmd_flags & CFLAG_TIMEOUT) {
385         mptsas_log(mpt, CE_WARN, "config header request timeout");
386         rval = DDI_FAILURE;
387         goto page_done;
388     }
389
390     /*
391      * cmd_rfm points to the reply message if a reply was given. Check the

```

```

392     * IOCStatus to make sure everything went OK with the header request.
393     */
394     if (cmd->cmd_rfm) {
395         config_flags |= MPTSAS_ADDRESS_REPLY;
396         (void) ddi_dma_sync(mpt->m_dma_reply_frame_hdl, 0, 0,
397             DDI_DMA_SYNC_FORCPU);
398         reply = (pMpi2ConfigReply_t)(mpt->m_reply_frame + (cmd->cmd_rfm
399             - (mpt->m_reply_frame_dma_addr&0xfffffffful)));
400         config.page_type = ddi_get8(mpt->m_acc_reply_frame_hdl,
401             &reply->Header.PageType);
402         config.page_number = ddi_get8(mpt->m_acc_reply_frame_hdl,
403             &reply->Header.PageNumber);
404         config.page_length = ddi_get8(mpt->m_acc_reply_frame_hdl,
405             &reply->Header.PageLength);
406         config.page_version = ddi_get8(mpt->m_acc_reply_frame_hdl,
407             &reply->Header.PageVersion);
408         config.ext_page_type = ddi_get8(mpt->m_acc_reply_frame_hdl,
409             &reply->ExtPageType);
410         config.ext_page_length = ddi_get16(mpt->m_acc_reply_frame_hdl,
411             &reply->ExtPageLength);
412
413         iocstatus = ddi_get16(mpt->m_acc_reply_frame_hdl,
414             &reply->IOCStatus);
415         iocloginfo = ddi_get32(mpt->m_acc_reply_frame_hdl,
416             &reply->IOCLogInfo);
417
418         if (iocstatus) {
419             NDBG13(("mptsas_access_config_page header: "
420                 "IOCStatus=0x%x, IOCLogInfo=0x%x", iocstatus,
421                 iocloginfo));
422             rval = DDI_FAILURE;
423             goto page_done;
424         }
425
426         if ((config.page_type & MPI2_CONFIG_PAGETYPE_MASK) ==
427             MPI2_CONFIG_PAGETYPE_EXTENDED)
428             len = (config.ext_page_length * 4);
429         else
430             len = (config.page_length * 4);
431
432     }
433
434     if (pkt->pkt_reason == CMD_RESET) {
435         mptsas_log(mpt, CE_WARN, "ioc reset abort config header "
436             "request");
437         rval = DDI_FAILURE;
438         goto page_done;
439     }
440
441     /*
442     * Put the reply frame back on the free queue, increment the free
443     * index, and write the new index to the free index register. But only
444     * if this reply is an ADDRESS reply.
445     */
446     if (config_flags & MPTSAS_ADDRESS_REPLY) {
447         ddi_put32(mpt->m_acc_free_queue_hdl,
448             &((uint32_t *) (void *) mpt->m_free_queue)[mpt->m_free_index],
449             cmd->cmd_rfm);
450         (void) ddi_dma_sync(mpt->m_dma_free_queue_hdl, 0, 0,
451             DDI_DMA_SYNC_FORDEV);
452         if (++mpt->m_free_index == mpt->m_free_queue_depth) {
453             mpt->m_free_index = 0;
454         }
455         ddi_put32(mpt->m_datap, &mpt->m_reg->ReplyFreeHostIndex,
456             mpt->m_free_index);
457         config_flags &= (~MPTSAS_ADDRESS_REPLY);

```

```

458     }
459
460     /*
461     * Allocate DMA buffer here. Store the info regarding this buffer in
462     * the cmd struct so that it can be used for this specific command and
463     * de-allocated after the command completes. The size of the reply
464     * will not be larger than the reply frame size.
465     */
466     attrs = mpt->m_msg_dma_attr;
467     attrs.dma_attr_sgllen = 1;
468     attrs.dma_attr_granular = (uint32_t)len;
469
470     if (mptsas_dma_addr_create(mpt, attrs,
471         &cmd->cmd_dmahandle, &accessp, &page_memp,
472         len, &cookie) == FALSE) {
473         mptsas_log(mpt, CE_WARN,
474             "mptsas_dma_addr_create(len=0x%x) failed", (int)len);
475         rval = DDI_FAILURE;
476         goto page_done;
477     }
478     /* NOW we can safely call mptsas_dma_addr_destroy(). */
479     free_dma = B_TRUE;
480
481     cmd->cmd_dma_addr = cookie.dmac_laddress;
482     bzero(page_memp, len);
483
484     /*
485     * Save the data for this request to be used in the call to start the
486     * config page read
487     */
488     config.action = action;
489     config.page_address = page_address;
490
491     /*
492     * Re-use the cmd that was used to get the header. Reset some of the
493     * values.
494     */
495     bzero((caddr_t)pkt, scsi_pkt_size());
496     pkt->pkt_ha_private = (opaque_t)&config;
497     pkt->pkt_flags = FLAG_HEAD;
498     pkt->pkt_time = 60;
499     cmd->cmd_flags = CFLAG_PREPARED | CFLAG_CMDIOC | CFLAG_CONFIG;
500
501     /*
502     * Send the config page request. cmd is re-used from header request.
503     */
504     mptsas_start_config_page_access(mpt, cmd);
505
506     /*
507     * If this is a request for a RAID info page, or any page called during
508     * the RAID info page request, poll because these config page requests
509     * are nested. Poll to avoid data corruption due to one page's data
510     * overwriting the outer page request's data. This can happen when
511     * the mutex is released in cv_wait.
512     */
513     if ((page_type == MPI2_CONFIG_EXT_PAGETYPE_RAID_CONFIG) ||
514         (page_type == MPI2_CONFIG_PAGETYPE_RAID_VOLUME) ||
515         (page_type == MPI2_CONFIG_PAGETYPE_RAID_PHYSDISK)) {
516         (void) mptsas_poll(mpt, cmd, pkt->pkt_time * 1000);
517     } else {
518         while ((cmd->cmd_flags & CFLAG_FINISHED) == 0) {
519             cv_wait(&mpt->m_config_cv, &mpt->m_mutex);
520         }
521     }
522
523     /*

```

```

524     * Check if the request completed without timing out
525     */
526     if (cmd->cmd_flags & CFLAG_TIMEOUT) {
527         mptsas_log(mpt, CE_WARN, "config page request timeout");
528         rval = DDI_FAILURE;
529         goto page_done;
530     }

532     /*
533     * cmd_rfm points to the reply message if a reply was given. The reply
534     * frame and the config page are returned from this function in the
535     * param list.
536     */
537     if (cmd->cmd_rfm) {
538         config_flags |= MPTSAS_ADDRESS_REPLY;
539         (void) ddi_dma_sync(mpt->m_dma_reply_frame_hdl, 0, 0,
540             DDI_DMA_SYNC_FORCPU);
541         (void) ddi_dma_sync(cmd->cmd_dmahandle, 0, 0,
542             DDI_DMA_SYNC_FORCPU);
543         reply = (pMpi2ConfigReply_t)(mpt->m_reply_frame + (cmd->cmd_rfm
544             - (mpt->m_reply_frame_dma_addr & 0xffffffff)));
545         iocstatus = ddi_get16(mpt->m_acc_reply_frame_hdl,
546             &reply->IOCStatus);
547         iocstatus = MPTSAS_IOCSTATUS(iocstatus);
548         iocloginfo = ddi_get32(mpt->m_acc_reply_frame_hdl,
549             &reply->IOCLogInfo);
550     }

552     if (callback(mpt, page_memp, accesssp, iocstatus, iocloginfo, ap)) {
553         rval = DDI_FAILURE;
554         goto page_done;
555     }

557     mptsas_fma_check(mpt, cmd);
558     /*
559     * Check the DMA/ACC handles and then free the DMA buffer.
560     */
561     if ((mptsas_check_dma_handle(cmd->cmd_dmahandle) != DDI_SUCCESS) ||
562         (mptsas_check_acc_handle(accesssp) != DDI_SUCCESS)) {
563         ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_UNAFFECTED);
564         rval = DDI_FAILURE;
565     }

567     if (pkt->pkt_reason == CMD_TRAN_ERR) {
568         mptsas_log(mpt, CE_WARN, "config fma error");
569         rval = DDI_FAILURE;
570         goto page_done;
571     }
572     if (pkt->pkt_reason == CMD_RESET) {
573         mptsas_log(mpt, CE_WARN, "ioc reset abort config request");
574         rval = DDI_FAILURE;
575         goto page_done;
576     }

578     page_done:
579     va_end(ap);
580     /*
581     * Put the reply frame back on the free queue, increment the free
582     * index, and write the new index to the free index register. But only
583     * if this reply is an ADDRESS reply.
584     */
585     if (config_flags & MPTSAS_ADDRESS_REPLY) {
586         ddi_put32(mpt->m_acc_free_queue_hdl,
587             &((uint32_t *) (void *) mpt->m_free_queue)[mpt->m_free_index],
588             cmd->cmd_rfm);
589         (void) ddi_dma_sync(mpt->m_dma_free_queue_hdl, 0, 0,

```

```

590         DDI_DMA_SYNC_FORDEV);
591         if (++mpt->m_free_index == mpt->m_free_queue_depth) {
592             mpt->m_free_index = 0;
593         }
594         ddi_put32(mpt->m_datap, &mpt->m_reg->ReplyFreeHostIndex,
595             mpt->m_free_index);
596     }

598     if (free_dma)
599         mptsas_dma_addr_destroy(&cmd->cmd_dmahandle, &accesssp);

601     if (cmd && (cmd->cmd_flags & CFLAG_PREPARED)) {
602         mptsas_remove_cmd(mpt, cmd);
603         config_flags &= (~MPTSAS_REQUEST_POOL_CMD);
604     }
605     if (config_flags & MPTSAS_REQUEST_POOL_CMD)
606         mptsas_return_to_pool(mpt, cmd);

608     if (config_flags & MPTSAS_CMD_TIMEOUT) {
609         mpt->m_softstate &= ~MPTSAS_SS_MSG_UNIT_RESET;
610         if ((mptsas_restart_ioc(mpt)) == DDI_FAILURE) {
611             mptsas_log(mpt, CE_WARN, "mptsas_restart_ioc failed");
612         }
613     }

615     return (rval);
616 }

618 int
619 mptsas_send_config_request_msg(mptsas_t *mpt, uint8_t action, uint8_t pagetype,
620     uint32_t pageaddress, uint8_t pagenumber, uint8_t pageversion,
621     uint8_t pagelength, uint32_t SGEflagslength, uint64_t SGEaddress)
622 {
623     pMpi2ConfigRequest_t    config;
624     int                      send_numbytes;

626     bzero(mpt->m_hshk_memp, sizeof (MPI2_CONFIG_REQUEST));
627     config = (pMpi2ConfigRequest_t)mpt->m_hshk_memp;
628     ddi_put8(mpt->m_hshk_acc_hdl, &config->Function, MPI2_FUNCTION_CONFIG);
629     ddi_put8(mpt->m_hshk_acc_hdl, &config->Action, action);
630     ddi_put8(mpt->m_hshk_acc_hdl, &config->Header.PageNumber, pagenumber);
631     ddi_put8(mpt->m_hshk_acc_hdl, &config->Header.PageType, pagetype);
632     ddi_put32(mpt->m_hshk_acc_hdl, &config->PageAddress, pageaddress);
633     ddi_put8(mpt->m_hshk_acc_hdl, &config->Header.PageVersion, pageversion);
634     ddi_put8(mpt->m_hshk_acc_hdl, &config->Header.PageLength, pagelength);
635     ddi_put32(mpt->m_hshk_acc_hdl,
636         &config->PageBufferSGE.MpiSimple.FlagsLength, SGEflagslength);
637     ddi_put32(mpt->m_hshk_acc_hdl,
638         &config->PageBufferSGE.MpiSimple.u.Address64.Low,
639         SGEaddress & 0xffffffff);
640     ddi_put32(mpt->m_hshk_acc_hdl,
641         &config->PageBufferSGE.MpiSimple.u.Address64.High,
642         SGEaddress >> 32);
643     send_numbytes = sizeof (MPI2_CONFIG_REQUEST);

645     /*
646     * Post message via handshake
647     */
648     if (mptsas_send_handshake_msg(mpt, (caddr_t)config, send_numbytes,
649         mpt->m_hshk_acc_hdl) {
650         return (-1);
651     }
652     return (0);
653 }

655 int

```

```

656 mptsas_send_extended_config_request_msg(mptsas_t *mpt, uint8_t action,
657      uint8_t extpagetype, uint32_t pageaddress, uint8_t pagenumber,
658      uint8_t pageversion, uint16_t extpagelength,
659      uint32_t SGEflagslength, uint64_t SGEaddress)
660 {
661     pMpi2ConfigRequest_t    config;
662     int                      send_numbytes;

664     bzero(mpt->m_hshk_memp, sizeof (MPI2_CONFIG_REQUEST));
665     config = (pMpi2ConfigRequest_t)mpt->m_hshk_memp;
666     ddi_put8(mpt->m_hshk_acc_hdl, &config->Function, MPI2_FUNCTION_CONFIG);
667     ddi_put8(mpt->m_hshk_acc_hdl, &config->Action, action);
668     ddi_put8(mpt->m_hshk_acc_hdl, &config->Header.PageNumber, pagenumber);
669     ddi_put8(mpt->m_hshk_acc_hdl, &config->Header.PageType,
670         MPI2_CONFIG_PAGETYPE_EXTENDED);
671     ddi_put8(mpt->m_hshk_acc_hdl, &config->ExtPageType, extpagetype);
672     ddi_put32(mpt->m_hshk_acc_hdl, &config->PageAddress, pageaddress);
673     ddi_put8(mpt->m_hshk_acc_hdl, &config->Header.PageVersion, pageversion);
674     ddi_put16(mpt->m_hshk_acc_hdl, &config->ExtPageLength, extpagelength);
675     ddi_put32(mpt->m_hshk_acc_hdl,
676         &config->PageBufferSGE.MpiSimple.FlagsLength, SGEflagslength);
677     ddi_put32(mpt->m_hshk_acc_hdl,
678         &config->PageBufferSGE.MpiSimple.u.Address64.Low,
679         SGEaddress&0xfffffffful);
680     ddi_put32(mpt->m_hshk_acc_hdl,
681         &config->PageBufferSGE.MpiSimple.u.Address64.High,
682         SGEaddress >> 32);
683     send_numbytes = sizeof (MPI2_CONFIG_REQUEST);

685     /*
686      * Post message via handshake
687      */
688     if (mptsas_send_handshake_msg(mpt, (caddr_t)config, send_numbytes,
689         mpt->m_hshk_acc_hdl)) {
690         return (-1);
691     }
692     return (0);
693 }

695 int
696 mptsas_ioc_wait_for_response(mptsas_t *mpt)
697 {
698     int    polls = 0;

700     while ((ddi_get32(mpt->m_datap,
701         &mpt->m_reg->HostInterruptStatus) & MPI2_HIS_IOP_DOORBELL_STATUS)) {
702         drv_usecwait(1000);
703         if (polls++ > 60000) {
704             return (-1);
705         }
706     }
707     return (0);
708 }

710 int
711 mptsas_ioc_wait_for_doorbell(mptsas_t *mpt)
712 {
713     int    polls = 0;

715     while ((ddi_get32(mpt->m_datap,
716         &mpt->m_reg->HostInterruptStatus) & MPI2_HIM_DIM) == 0) {
717         drv_usecwait(1000);
718         if (polls++ > 300000) {
719             return (-1);
720         }
721     }

```

```

722     return (0);
723 }

725 int
726 mptsas_send_handshake_msg(mptsas_t *mpt, caddr_t memp, int numbytes,
727     ddi_acc_handle_t accessp)
728 {
729     int    i;

731     /*
732      * clean pending doorbells
733      */
734     ddi_put32(mpt->m_datap, &mpt->m_reg->HostInterruptStatus, 0);
735     ddi_put32(mpt->m_datap, &mpt->m_reg->Doorbell,
736         ((MPI2_FUNCTION_HANDSHAKE << MPI2_DOORBELL_FUNCTION_SHIFT) |
737         ((numbytes / 4) << MPI2_DOORBELL_ADD_DWORDS_SHIFT)));

739     if (mptsas_ioc_wait_for_doorbell(mpt)) {
740         NDBG19(("mptsas_send_handshake failed. Doorbell not ready\n"));
741         return (-1);
742     }

744     /*
745      * clean pending doorbells again
746      */
747     ddi_put32(mpt->m_datap, &mpt->m_reg->HostInterruptStatus, 0);

749     if (mptsas_ioc_wait_for_response(mpt)) {
750         NDBG19(("mptsas_send_handshake failed. Doorbell not "
751             "cleared\n"));
752         return (-1);
753     }

755     /*
756      * post handshake message
757      */
758     for (i = 0; (i < numbytes / 4); i++, memp += 4) {
759         ddi_put32(mpt->m_datap, &mpt->m_reg->Doorbell,
760             ddi_get32(accessp, (uint32_t *)((void *)memp)));
761         if (mptsas_ioc_wait_for_response(mpt)) {
762             NDBG19(("mptsas_send_handshake failed posting "
763                 "message\n"));
764             return (-1);
765         }
766     }

768     if (mptsas_check_acc_handle(mpt->m_datap) != DDI_SUCCESS) {
769         ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_UNAFFECTED);
770         ddi_fm_acc_err_clear(mpt->m_datap, DDI_FME_VER0);
771         return (-1);
772     }

774     return (0);
775 }

777 int
778 mptsas_get_handshake_msg(mptsas_t *mpt, caddr_t memp, int numbytes,
779     ddi_acc_handle_t accessp)
780 {
781     int    i, totalbytes, bytesleft;
782     uint16_t    val;

784     /*
785      * wait for doorbell
786      */
787     if (mptsas_ioc_wait_for_doorbell(mpt)) {

```

```

788         NDBG19(("mptsas_get_handshake failed. Doorbell not ready\n"));
789         return (-1);
790     }

792     /*
793     * get first 2 bytes of handshake message to determine how much
794     * data we will be getting
795     */
796     for (i = 0; i < 2; i++, memptr += 2) {
797         val = (ddi_get32(mpt->m_datap,
798             &mpt->m_reg->Doorbell) & MPI2_DOORBELL_DATA_MASK);
799         ddi_put32(mpt->m_datap, &mpt->m_reg->HostInterruptStatus, 0);
800         if (mptsas_ioc_wait_for_doorbell(mpt)) {
801             NDBG19(("mptsas_get_handshake failure getting initial"
802                 " data\n"));
803             return (-1);
804         }
805         ddi_put16(accessp, (uint16_t *)((void *) (memptr)), val);
806         if (i == 1) {
807             totalbytes = (val & 0xFF) * 2;
808         }
809     }

811     /*
812     * If we are expecting less bytes than the message wants to send
813     * we simply save as much as we expected and then throw out the rest
814     * later
815     */
816     if (totalbytes > (numbytes / 2)) {
817         bytesleft = ((numbytes / 2) - 2);
818     } else {
819         bytesleft = (totalbytes - 2);
820     }

822     /*
823     * Get the rest of the data
824     */
825     for (i = 0; i < bytesleft; i++, memptr += 2) {
826         val = (ddi_get32(mpt->m_datap,
827             &mpt->m_reg->Doorbell) & MPI2_DOORBELL_DATA_MASK);
828         ddi_put32(mpt->m_datap, &mpt->m_reg->HostInterruptStatus, 0);
829         if (mptsas_ioc_wait_for_doorbell(mpt)) {
830             NDBG19(("mptsas_get_handshake failure getting"
831                 " main data\n"));
832             return (-1);
833         }
834         ddi_put16(accessp, (uint16_t *)((void *) (memptr)), val);
835     }

837     /*
838     * Sometimes the device will send more data than is expected
839     * This data is not used by us but needs to be cleared from
840     * ioc doorbell. So we just read the values and throw
841     * them out.
842     */
843     if (totalbytes > (numbytes / 2)) {
844         for (i = (numbytes / 2); i < totalbytes; i++) {
845             val = (ddi_get32(mpt->m_datap,
846                 &mpt->m_reg->Doorbell) &
847                 MPI2_DOORBELL_DATA_MASK);
848             ddi_put32(mpt->m_datap,
849                 &mpt->m_reg->HostInterruptStatus, 0);
850             if (mptsas_ioc_wait_for_doorbell(mpt)) {
851                 NDBG19(("mptsas_get_handshake failure getting "
852                     "extra garbage data\n"));
853                 return (-1);

```

```

854         }
855     }
856 }

858     ddi_put32(mpt->m_datap, &mpt->m_reg->HostInterruptStatus, 0);

860     if (mptsas_check_acc_handle(mpt->m_datap) != DDI_SUCCESS) {
861         ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_UNAFFECTED);
862         ddi_fm_acc_err_clear(mpt->m_datap, DDI_FME_VER0);
863         return (-1);
864     }

866     return (0);
867 }

869 int
870 mptsas_kick_start(mptsas_t *mpt)
871 {
872     int        polls = 0;
873     uint32_t    diag_reg, ioc_state, saved_HCB_size;

875     /*
876     * Start a hard reset. Write magic number and wait 500 mSeconds.
877     */
878     MPTSAS_ENABLE_DRWE(mpt);
879     drv_usecwait(500000);

881     /*
882     * Read the current Diag Reg and save the Host Controlled Boot size.
883     */
884     diag_reg = ddi_get32(mpt->m_datap, &mpt->m_reg->HostDiagnostic);
885     saved_HCB_size = ddi_get32(mpt->m_datap, &mpt->m_reg->HCBSize);

887     /*
888     * Set Reset Adapter bit and wait 50 mSeconds.
889     */
890     diag_reg |= MPI2_DIAG_RESET_ADAPTER;
891     ddi_put32(mpt->m_datap, &mpt->m_reg->HostDiagnostic, diag_reg);
892     drv_usecwait(50000);

894     /*
895     * Poll, waiting for Reset Adapter bit to clear. 300 Seconds max
896     * (600000 * 500 = 300,000,000 uSeconds, 300 seconds).
897     * If no more adapter (all FF's), just return failure.
898     */
899     for (polls = 0; polls < 600000; polls++) {
900         diag_reg = ddi_get32(mpt->m_datap,
901             &mpt->m_reg->HostDiagnostic);
902         if (diag_reg == 0xFFFFFFFF) {
903             mptsas_fm_ereport(mpt, DDI_FM_DEVICE_NO_RESPONSE);
904             ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_LOST);
905             return (DDI_FAILURE);
906         }
907         if (!(diag_reg & MPI2_DIAG_RESET_ADAPTER)) {
908             break;
909         }
910         drv_usecwait(500);
911     }

912     if (polls == 600000) {
913         mptsas_fm_ereport(mpt, DDI_FM_DEVICE_NO_RESPONSE);
914         ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_LOST);
915         return (DDI_FAILURE);
916     }

918     /*
919     * Check if adapter is in Host Boot Mode. If so, restart adapter

```

```

920      * assuming the HCB points to good FW.
921      * Set BootDeviceSel to HCDW (Host Code and Data Window).
922      */
923      if (diag_reg & MPI2_DIAG_HCB_MODE) {
924          diag_reg &= ~MPI2_DIAG_BOOT_DEVICE_SELECT_MASK;
925          diag_reg |= MPI2_DIAG_BOOT_DEVICE_SELECT_HCDW;
926          ddi_put32(mpt->m_datap, &mpt->m_reg->HostDiagnostic, diag_reg);

928          /*
929           * Re-enable the HCDW.
930           */
931          ddi_put32(mpt->m_datap, &mpt->m_reg->HCBSize,
932                  (saved_HCB_size | MPI2_HCB_SIZE_HCB_ENABLE));
933      }

935      /*
936       * Restart the adapter.
937       */
938      diag_reg &= ~MPI2_DIAG_HOLD_IOC_RESET;
939      ddi_put32(mpt->m_datap, &mpt->m_reg->HostDiagnostic, diag_reg);

941      /*
942       * Disable writes to the Host Diag register.
943       */
944      ddi_put32(mpt->m_datap, &mpt->m_reg->WriteSequence,
945              MPI2_WRSEQ_FLUSH_KEY_VALUE);

947      /*
948       * Wait 60 seconds max for FW to come to ready state.
949       */
950      for (polls = 0; polls < 60000; polls++) {
951          ioc_state = ddi_get32(mpt->m_datap, &mpt->m_reg->Doorbell);
952          if (ioc_state == 0xFFFFFFFF) {
953              mptsas_fm_ereport(mpt, DDI_FM_DEVICE_NO_RESPONSE);
954              ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_LOST);
955              return (DDI_FAILURE);
956          }
957          if ((ioc_state & MPI2_IOC_STATE_MASK) ==
958              MPI2_IOC_STATE_READY) {
959              break;
960          }
961          drv_usecwait(1000);
962      }
963      if (polls == 60000) {
964          mptsas_fm_ereport(mpt, DDI_FM_DEVICE_NO_RESPONSE);
965          ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_LOST);
966          return (DDI_FAILURE);
967      }

969      /*
970       * Clear the ioc ack events queue.
971       */
972      mptsas_destroy_ioc_event_cmd(mpt);

974      return (DDI_SUCCESS);
975 }

977 int
978 mptsas_ioc_reset(mptsas_t *mpt, int first_time)
979 {
980     int        polls = 0;
981     uint32_t    reset_msg;
982     uint32_t    ioc_state;

984     ioc_state = ddi_get32(mpt->m_datap, &mpt->m_reg->Doorbell);
985     /*

```

```

986     * If chip is already in ready state then there is nothing to do.
987     */
988     if (ioc_state == MPI2_IOC_STATE_READY) {
989         return (MPTSAS_NO_RESET);
990     }
991     /*
992     * If the chip is already operational, we just need to send
993     * it a message unit reset to put it back in the ready state
994     */
995     if (ioc_state & MPI2_IOC_STATE_OPERATIONAL) {
996         /*
997          * If the first time, try MUR anyway, because we haven't even
998          * queried the card for m_event_replay and other capabilities.
999          * Other platforms do it this way, we can still do a hard
1000          * reset if we need to, MUR takes less time than a full
1001          * adapter reset, and there are reports that some HW
1002          * combinations will lock up when receiving a hard reset.
1003          */
1004         if ((first_time || mpt->m_event_replay) &&
1005             (mpt->m_softstate & MPTSAS_SS_MSG_UNIT_RESET)) {
1006             mpt->m_softstate &= ~MPTSAS_SS_MSG_UNIT_RESET;
1007             reset_msg = MPI2_FUNCTION_IOC_MESSAGE_UNIT_RESET;
1008             ddi_put32(mpt->m_datap, &mpt->m_reg->Doorbell,
1009                     (reset_msg << MPI2_DOORBELL_FUNCTION_SHIFT));
1010             if (mptsas_ioc_wait_for_response(mpt)) {
1011                 NDBG19(("mptsas_ioc_reset failure sending "
1012                     "message_unit_reset\n"));
1013                 goto hard_reset;
1014             }
1015         }

1016         /*
1017          * Wait no more than 60 seconds for chip to become
1018          * ready.
1019          */
1020         while ((ddi_get32(mpt->m_datap, &mpt->m_reg->Doorbell) &
1021             MPI2_IOC_STATE_READY) == 0x0) {
1022             drv_usecwait(1000);
1023             if (polls++ > 60000) {
1024                 goto hard_reset;
1025             }
1026         }

1028         /*
1029          * Save the last reset mode done on IOC which will be
1030          * helpful while resuming from suspension.
1031          */
1032         mpt->m_softstate |= MPTSAS_DID_MSG_UNIT_RESET;

1034         /*
1035          * the message unit reset would do reset operations
1036          * clear reply and request queue, so we should clear
1037          * ACK event cmd.
1038          */
1039         mptsas_destroy_ioc_event_cmd(mpt);
1040         return (MPTSAS_SUCCESS_MUR);
1041     }
1042 }
1043 hard_reset:
1044     mpt->m_softstate &= ~MPTSAS_DID_MSG_UNIT_RESET;
1045     if (mptsas_kick_start(mpt) == DDI_FAILURE) {
1046         mptsas_fm_ereport(mpt, DDI_FM_DEVICE_NO_RESPONSE);
1047         ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_LOST);
1048         return (MPTSAS_RESET_FAIL);
1049     }
1050     return (MPTSAS_SUCCESS_HARDRESET);
1051 }

```

```

1054 int
1055 mptsas_request_from_pool(mptsas_t *mpt, mptsas_cmd_t **cmd,
1056     struct scsi_pkt **pkt)
1057 {
1058     m_event_struct_t      *ioc_cmd = NULL;

1060     ioc_cmd = kmem_zalloc(M_EVENT_STRUCT_SIZE, KM_SLEEP);
1061     if (ioc_cmd == NULL) {
1062         return (DDI_FAILURE);
1063     }
1064     ioc_cmd->m_event_linkp = NULL;
1065     mptsas_ioc_event_cmdq_add(mpt, ioc_cmd);
1066     *cmd = &(ioc_cmd->m_event_cmd);
1067     *pkt = &(ioc_cmd->m_event_pkt);

1069     return (DDI_SUCCESS);
1070 }

1072 void
1073 mptsas_return_to_pool(mptsas_t *mpt, mptsas_cmd_t *cmd)
1074 {
1075     m_event_struct_t      *ioc_cmd = NULL;

1077     ioc_cmd = mptsas_ioc_event_find_by_cmd(mpt, cmd);
1078     if (ioc_cmd == NULL) {
1079         return;
1080     }

1082     mptsas_ioc_event_cmdq_delete(mpt, ioc_cmd);
1083     kmem_free(ioc_cmd, M_EVENT_STRUCT_SIZE);
1084     ioc_cmd = NULL;
1085 }

1087 /*
1088  * NOTE: We should be able to queue TM requests in the controller to make this
1089  * a lot faster.  If resetting all targets, for example, we can load the hi
1090  * priority queue with its limit and the controller will reply as they are
1091  * completed.  This way, we don't have to poll for one reply at a time.
1092  * Think about enhancing this later.
1093  */
1094 int
1095 mptsas_ioc_task_management(mptsas_t *mpt, int task_type, uint16_t dev_handle,
1096     int lun, uint8_t *reply, uint32_t reply_size, int mode)
1097 {
1098     /*
1099      * In order to avoid allocating variables on the stack,
1100      * we make use of the pre-existing mptsas_cmd_t and
1101      * scsi_pkt which are included in the mptsas_t which
1102      * is passed to this routine.
1103      */

1105     pMpi2SCSITaskManagementRequest_t    task;
1106     int                                   rval = FALSE;
1107     mptsas_cmd_t                          *cmd;
1108     struct scsi_pkt                       *pkt;
1109     mptsas_slots_t                        *slots = mpt->m_active;
1110     uint32_t                              i;
1111     uint64_t                              request_desc;
1112     pMPI2DefaultReply_t                   reply_msg;

1114     /*
1115      * Can't start another task management routine.
1116      */
1117     if (slots->m_slot[MPTSAS_TM_SLOT(mpt)] != NULL) {

```

```

1118         mptsas_log(mpt, CE_WARN, "Can only start 1 task management"
1119             " command at a time\n");
1120         return (FALSE);
1121     }

1123     cmd = &(mpt->m_event_task_mgmt.m_event_cmd);
1124     pkt = &(mpt->m_event_task_mgmt.m_event_pkt);

1126     bzero((caddr_t)cmd, sizeof (*cmd));
1127     bzero((caddr_t)pkt, scsi_pkt_size());

1129     pkt->pkt_cdbp      = (opaque_t)&cmd->cmd_cdb[0];
1130     pkt->pkt_scbp      = (opaque_t)&cmd->cmd_scb;
1131     pkt->pkt_ha_private = (opaque_t)cmd;
1132     pkt->pkt_flags      = (FLAG_NOINTR | FLAG_HEAD);
1133     pkt->pkt_time       = 60;
1134     pkt->pkt_address.a_target = dev_handle;
1135     pkt->pkt_address.a_lun = (uchar_t)lun;
1136     cmd->cmd_pkt        = pkt;
1137     cmd->cmd_scblen     = 1;
1138     cmd->cmd_flags      = CFLAG_TM_CMD;
1139     cmd->cmd_slot       = MPTSAS_TM_SLOT(mpt);

1141     slots->m_slot[MPTSAS_TM_SLOT(mpt)] = cmd;

1143     /*
1144      * Store the TM message in memory location corresponding to the TM slot
1145      * number.
1146      */
1147     task = (pMpi2SCSITaskManagementRequest_t)(mpt->m_req_frame +
1148         (mpt->m_req_frame_size * cmd->cmd_slot));
1149     bzero(task, mpt->m_req_frame_size);

1151     /*
1152      * form message for requested task
1153      */
1154     mptsas_init_std_hdr(mpt->m_acc_req_frame_hdl, task, dev_handle, lun, 0,
1155         MPI2_FUNCTION_SCSI_TASK_MGMT);

1157     /*
1158      * Set the task type
1159      */
1160     ddi_put8(mpt->m_acc_req_frame_hdl, &task->TaskType, task_type);

1162     /*
1163      * Send TM request using High Priority Queue.
1164      */
1165     (void) ddi_dma_sync(mpt->m_dma_req_frame_hdl, 0, 0,
1166         DDI_DMA_SYNC_FORDEV);
1167     request_desc = (cmd->cmd_slot << 16) +
1168         MPI2_REQ_DESCRIPTOR_FLAGS_HIGH_PRIORITY;
1169     MPTSAS_START_CMD(mpt, request_desc);
1170     rval = mptsas_poll(mpt, cmd, MPTSAS_POLL_TIME);

1172     if (pkt->pkt_reason == CMD_INCOMPLETE)
1173         rval = FALSE;

1175     /*
1176      * If a reply frame was used and there is a reply buffer to copy the
1177      * reply data into, copy it.  If this fails, log a message, but don't
1178      * fail the TM request.
1179      */
1180     if (cmd->cmd_rfm && reply) {
1181         (void) ddi_dma_sync(mpt->m_dma_reply_frame_hdl, 0, 0,
1182             DDI_DMA_SYNC_FORCPU);
1183         reply_msg = (pMPI2DefaultReply_t)

```

```

1184         (mpt->m_reply_frame + (cmd->cmd_rfm -
1185         (mpt->m_reply_frame_dma_addr&0xffffffffful)));
1186     if (reply_size > sizeof (MPI2_SCSI_TASK_MANAGE_REPLY)) {
1187         reply_size = sizeof (MPI2_SCSI_TASK_MANAGE_REPLY);
1188     }
1189     mutex_exit(&mpt->m_mutex);
1190     for (i = 0; i < reply_size; i++) {
1191         if (ddi_copyout((uint8_t *)reply_msg + i, reply + i, 1,
1192             mode)) {
1193             mptsas_log(mpt, CE_WARN, "failed to copy out "
1194                 "reply data for TM request");
1195             break;
1196         }
1197     }
1198     mutex_enter(&mpt->m_mutex);
1199 }
1200
1201 /*
1202  * clear the TM slot before returning
1203  */
1204 slots->m_slot[MPTSAS_TM_SLOT(mpt)] = NULL;
1205
1206 /*
1207  * If we lost our task management command
1208  * we need to reset the ioc
1209  */
1210 if (rval == FALSE) {
1211     mptsas_log(mpt, CE_WARN, "mptsas_ioc_task_management failed "
1212         "try to reset ioc to recovery!");
1213     mpt->m_softstate &= ~MPTSAS_SS_MSG_UNIT_RESET;
1214     if (mptsas_restart_ioc(mpt)) {
1215         mptsas_log(mpt, CE_WARN, "mptsas_restart_ioc failed");
1216         rval = FAILED;
1217     }
1218 }
1219
1220 return (rval);
1221 }
1222
1223 /*
1224  * Complete firmware download frame for v2.0 cards.
1225  */
1226 static void
1227 mptsas_uflash2(pMpi2FWDDownloadRequest fwddownload,
1228     ddi_acc_handle_t acc_hdl, uint32_t size, uint8_t type,
1229     ddi_dma_cookie_t flsh_cookie)
1230 {
1231     pMpi2FWDDownloadTCSGE_t tcsge;
1232     pMpi2SGESimple64_t sge;
1233     uint32_t flagslength;
1234
1235     ddi_put8(acc_hdl, &fwddownload->Function,
1236         MPI2_FUNCTION_FW_DOWNLOAD);
1237     ddi_put8(acc_hdl, &fwddownload->ImageType, type);
1238     ddi_put8(acc_hdl, &fwddownload->MsgFlags,
1239         MPI2_FW_DOWNLOAD_MSGFLGS_LAST_SEGMENT);
1240     ddi_put32(acc_hdl, &fwddownload->TotalImageSize, size);
1241
1242     tcsge = (pMpi2FWDDownloadTCSGE_t)&fwddownload->SGL;
1243     ddi_put8(acc_hdl, &tcsge->ContextSize, 0);
1244     ddi_put8(acc_hdl, &tcsge->DetailsLength, 12);
1245     ddi_put8(acc_hdl, &tcsge->Flags, 0);
1246     ddi_put32(acc_hdl, &tcsge->ImageOffset, 0);
1247     ddi_put32(acc_hdl, &tcsge->ImageSize, size);
1248
1249     sge = (pMpi2SGESimple64_t)(tcsge + 1);

```

```

1250     flagslength = size;
1251     flagslength |= ((uint32_t)(MPI2_SGE_FLAGS_LAST_ELEMENT |
1252         MPI2_SGE_FLAGS_END_OF_BUFFER |
1253         MPI2_SGE_FLAGS_SIMPLE_ELEMENT |
1254         MPI2_SGE_FLAGS_SYSTEM_ADDRESS |
1255         MPI2_SGE_FLAGS_64_BIT_ADDRESSING |
1256         MPI2_SGE_FLAGS_HOST_TO_IOC |
1257         MPI2_SGE_FLAGS_END_OF_LIST) << MPI2_SGE_FLAGS_SHIFT);
1258     ddi_put32(acc_hdl, &sge->FlagsLength, flagslength);
1259     ddi_put32(acc_hdl, &sge->Address.Low,
1260         flsh_cookie.dmac_address);
1261     ddi_put32(acc_hdl, &sge->Address.High,
1262         (uint32_t)(flsh_cookie.dmac_laddress >> 32));
1263 }
1264
1265 /*
1266  * Complete firmware download frame for v2.5 cards.
1267  */
1268 static void
1269 mptsas_uflash25(pMpi25FWDDownloadRequest fwddownload,
1270     ddi_acc_handle_t acc_hdl, uint32_t size, uint8_t type,
1271     ddi_dma_cookie_t flsh_cookie)
1272 {
1273     pMpi2IeeeSgeSimple64_t sge;
1274     uint8_t flags;
1275
1276     ddi_put8(acc_hdl, &fwddownload->Function,
1277         MPI2_FUNCTION_FW_DOWNLOAD);
1278     ddi_put8(acc_hdl, &fwddownload->ImageType, type);
1279     ddi_put8(acc_hdl, &fwddownload->MsgFlags,
1280         MPI2_FW_DOWNLOAD_MSGFLGS_LAST_SEGMENT);
1281     ddi_put32(acc_hdl, &fwddownload->TotalImageSize, size);
1282
1283     ddi_put32(acc_hdl, &fwddownload->ImageOffset, 0);
1284     ddi_put32(acc_hdl, &fwddownload->ImageSize, size);
1285
1286     sge = (pMpi2IeeeSgeSimple64_t)&fwddownload->SGL;
1287     flags = MPI2_IEEE_SGE_FLAGS_SIMPLE_ELEMENT |
1288         MPI2_IEEE_SGE_FLAGS_SYSTEM_ADDR |
1289         MPI25_IEEE_SGE_FLAGS_END_OF_LIST;
1290     ddi_put8(acc_hdl, &sge->Flags, flags);
1291     ddi_put32(acc_hdl, &sge->Length, size);
1292     ddi_put32(acc_hdl, &sge->Address.Low,
1293         flsh_cookie.dmac_address);
1294     ddi_put32(acc_hdl, &sge->Address.High,
1295         (uint32_t)(flsh_cookie.dmac_laddress >> 32));
1296 }
1297
1298 static int mptsas_enable_mpi25_flashupdate = 0;
1299
1300 int
1301 mptsas_update_flash(mptsas_t *mpt, caddr_t ptrbuffer, uint32_t size,
1302     uint8_t type, int mode)
1303 {
1304     /*
1305      * In order to avoid allocating variables on the stack,
1306      * we make use of the pre-existing mptsas_cmd_t and
1307      * scsi_pkt which are included in the mptsas_t which
1308      * is passed to this routine.
1309      */
1310
1311     ddi_dma_attr_t flsh_dma_attr;
1312     ddi_dma_cookie_t flsh_cookie;
1313     ddi_dma_handle_t flsh_dma_handle;
1314     ddi_acc_handle_t flsh_accessp;

```



```

1316     caddr_t      memp, flsh_memp;
1317     mptsas_cmd_t *cmd;
1318     struct scsi_pkt *pkt;
1319     int i;
1320     int rvalue = 0;
1321     uint64_t request_desc;

1323     if (mpt->m_MPI25) {
1324         /*
1325          * The code is there but not tested yet.
1326          * User has to know there are risks here.
1327          */
1328         mptsas_log(mpt, CE_WARN, "mptsas_update_flash(): "
1329             "Updating firmware through MPI 2.5 has not been "
1330             "tested yet!\n");
1331         "To enable set mptsas_enable_mpi25_flashupdate to 1\n";
1332         if (!mptsas_enable_mpi25_flashupdate)
1333             return (-1);
1334     }

1336     if ((rvalue = (mptsas_request_from_pool(mpt, &cmd, &pkt))) == -1) {
1337         mptsas_log(mpt, CE_WARN, "mptsas_update_flash(): allocation "
1338             "failed. event ack command pool is full\n");
1339         return (rvalue);
1340     }

1342     bzero((caddr_t)cmd, sizeof (*cmd));
1343     bzero((caddr_t)pkt, scsi_pkt_size());
1344     cmd->ioc_cmd_slot = (uint32_t)rvalue;

1346     /*
1347      * dynamically create a customized dma attribute structure
1348      * that describes the flash file.
1349      */
1350     flsh_dma_attrs = mpt->m_msg_dma_attr;
1351     flsh_dma_attrs.dma_attr_sgllen = 1;

1353     if (mptsas_dma_addr_create(mpt, flsh_dma_attrs, &flsh_dma_handle,
1354         &flsh_accesssp, &flsh_memp, size, &flsh_cookie) == FALSE) {
1355         mptsas_log(mpt, CE_WARN,
1356             "(unable to allocate dma resource.);");
1357         mptsas_return_to_pool(mpt, cmd);
1358         return (-1);
1359     }

1361     bzero(flsh_memp, size);

1363     for (i = 0; i < size; i++) {
1364         (void) ddi_copyin(ptrbuffer + i, flsh_memp + i, 1, mode);
1365     }
1366     (void) ddi_dma_sync(flsh_dma_handle, 0, 0, DDI_DMA_SYNC_FORDEV);

1368     /*
1369      * form a cmd/pkt to store the fw download message
1370      */
1371     pkt->pkt_cdbp = (opaque_t)&cmd->cmd_cdb[0];
1372     pkt->pkt_scbp = (opaque_t)&cmd->cmd_scb;
1373     pkt->pkt_ha_private = (opaque_t)cmd;
1374     pkt->pkt_flags = FLAG_HEAD;
1375     pkt->pkt_time = 60;
1376     cmd->cmd_pkt = pkt;
1377     cmd->cmd_scbllen = 1;
1378     cmd->cmd_flags = CFLAG_CMDIOC | CFLAG_FW_CMD;

1380     /*
1381      * Save the command in a slot

```

```

1382     */
1383     if (mptsas_save_cmd(mpt, cmd) == FALSE) {
1384         mptsas_dma_addr_destroy(&flsh_dma_handle, &flsh_accesssp);
1385         mptsas_return_to_pool(mpt, cmd);
1386         return (-1);
1387     }

1389     /*
1390      * Fill in fw download message
1391      */
1392     ASSERT(cmd->cmd_slot != 0);
1393     memp = mpt->m_req_frame + (mpt->m_req_frame_size * cmd->cmd_slot);
1394     bzero(memp, mpt->m_req_frame_size);

1396     if (mpt->m_MPI25)
1397         mptsas_uflash25((pMpi25FWDownloadRequest)memp,
1398             mpt->m_acc_req_frame_hdl, size, type, flsh_cookie);
1399     else
1400         mptsas_uflash2((pMpi2FWDownloadRequest)memp,
1401             mpt->m_acc_req_frame_hdl, size, type, flsh_cookie);

1403     /*
1404      * Start command
1405      */
1406     (void) ddi_dma_sync(mpt->m_dma_req_frame_hdl, 0, 0,
1407         DDI_DMA_SYNC_FORDEV);
1408     request_desc = (cmd->cmd_slot << 16) +
1409         MPI2_REQ_DESCRIPTOR_FLAGS_DEFAULT_TYPE;
1410     cmd->cmd_rfm = NULL;
1411     MPTSAS_START_CMD(mpt, request_desc);

1413     rvalue = 0;
1414     (void) cv_reltimedwait(&mpt->m_fw_cv, &mpt->m_mutex,
1415         drv_usectohz(60 * MICROSEC), TR_CLOCK_TICK);
1416     if (!(cmd->cmd_flags & CFLAG_FINISHED)) {
1417         mpt->m_softstate &= ~MPTSAS_SS_MSG_UNIT_RESET;
1418         if ((mptsas_restart_ioc(mpt)) == DDI_FAILURE) {
1419             mptsas_log(mpt, CE_WARN, "mptsas_restart_ioc failed");
1420         }
1421         rvalue = -1;
1422     }
1423     mptsas_remove_cmd(mpt, cmd);
1424     mptsas_dma_addr_destroy(&flsh_dma_handle, &flsh_accesssp);

1426     return (rvalue);
1427 }

1429 static int
1430 mptsas_sasdevpage_0_cb(mptsas_t *mpt, caddr_t page_memp,
1431     ddi_acc_handle_t accesssp, uint16_t iocstatus, uint32_t iocloginfo,
1432     va_list ap)
1433 {
1434     #ifndef __lock_lint
1435         _NOTE(ARGUNUSED(ap))
1436     #endif

1437     pMpi2SasDevicePage0_t sasdevpage;
1438     int rval = DDI_SUCCESS, i;
1439     uint8_t *sas_addr = NULL;
1440     uint8_t tmp_sas_wnn[SAS_WWN_BYTE_SIZE];
1441     uint16_t *devhdl, *bay_num, *enclosure;
1442     uint64_t *sas_wnn;
1443     *dev_info;
1444     *physport, *phynum;
1445     *pdevhdl, *io_flags;
1446     uint32_t page_address;

```

```

1448     if ((iocstatus != MPI2_IOCSTATUS_SUCCESS) &&
1449         (iocstatus != MPI2_IOCSTATUS_CONFIG_INVALID_PAGE)) {
1450         mptsas_log(mpt, CE_WARN, "mptsas_get_sas_device_page0 "
1451             "header: IOCStatus=0x%x, IOCLogInfo=0x%x",
1452             iocstatus, iocloginfo);
1453         rval = DDI_FAILURE;
1454         return (rval);
1455     }
1456     page_address = va_arg(ap, uint32_t);
1457     /*
1458     * The INVALID_PAGE status is normal if using GET_NEXT_HANDLE and there
1459     * are no more pages. If everything is OK up to this point but the
1460     * status is INVALID_PAGE, change rval to FAILURE and quit. Also,
1461     * signal that device traversal is complete.
1462     */
1463     if (iocstatus == MPI2_IOCSTATUS_CONFIG_INVALID_PAGE) {
1464         if ((page_address & MPI2_SAS_DEVICE_PGAD_FORM_MASK) ==
1465             MPI2_SAS_DEVICE_PGAD_FORM_GET_NEXT_HANDLE) {
1466             mpt->m_done_traverse_dev = 1;
1467         }
1468         rval = DDI_FAILURE;
1469         return (rval);
1470     }
1471     devhdl = va_arg(ap, uint16_t *);
1472     sas_wnn = va_arg(ap, uint64_t *);
1473     dev_info = va_arg(ap, uint32_t *);
1474     physport = va_arg(ap, uint8_t *);
1475     phynum = va_arg(ap, uint8_t *);
1476     pdevhdl = va_arg(ap, uint16_t *);
1477     bay_num = va_arg(ap, uint16_t *);
1478     enclosure = va_arg(ap, uint16_t *);
1479     io_flags = va_arg(ap, uint16_t *);
1481     sasdevpage = (pMpi2SasDevicePage0_t)page_memp;
1483     *dev_info = ddi_get32(accessp, &sasdevpage->DeviceInfo);
1484     *devhdl = ddi_get16(accessp, &sasdevpage->DevHandle);
1485     sas_addr = (uint8_t *)(&sasdevpage->SASAddress);
1486     for (i = 0; i < SAS_WWN_BYTE_SIZE; i++) {
1487         tmp_sas_wnn[i] = ddi_get8(accessp, sas_addr + i);
1488     }
1489     bcopy(tmp_sas_wnn, sas_wnn, SAS_WWN_BYTE_SIZE);
1490     *sas_wnn = LE_64(*sas_wnn);
1491     *physport = ddi_get8(accessp, &sasdevpage->PhysicalPort);
1492     *phynum = ddi_get8(accessp, &sasdevpage->PhyNum);
1493     *pdevhdl = ddi_get16(accessp, &sasdevpage->ParentDevHandle);
1494     *bay_num = ddi_get16(accessp, &sasdevpage->Slot);
1495     *enclosure = ddi_get16(accessp, &sasdevpage->EnclosureHandle);
1496     *io_flags = ddi_get16(accessp, &sasdevpage->Flags);
1498     if (*io_flags & MPI25_SAS_DEVICE0_FLAGS_FAST_PATH_CAPABLE) {
1499         /*
1500         * Leave a messages about FP cabability in the log.
1501         */
1502         mptsas_log(mpt, CE_CONT,
1503             "!w%016\"PRIx64\" FastPath Capable%s", *sas_wnn,
1504             (*io_flags &
1505             MPI25_SAS_DEVICE0_FLAGS_ENABLED_FAST_PATH)?
1506             " and Enabled":" but Disabled");
1507     }
1509     return (rval);
1510 }
1512 /*
1513 * Request MPI configuration page SAS device page 0 to get DevHandle, device

```

```

1514 * info and SAS address.
1515 */
1516 int
1517 mptsas_get_sas_device_page0(mptsas_t *mpt, uint32_t page_address,
1518     uint16_t *dev_handle, uint64_t *sas_wnn, uint32_t *dev_info,
1519     uint8_t *physport, uint8_t *phynum, uint16_t *pdev_handle,
1520     uint16_t *bay_num, uint16_t *enclosure, uint16_t *io_flags)
1521 {
1522     int rval = DDI_SUCCESS;
1524     ASSERT(mutex_owned(&mpt->m_mutex));
1526     /*
1527     * Get the header and config page. reply contains the reply frame,
1528     * which holds status info for the request.
1529     */
1530     rval = mptsas_access_config_page(mpt,
1531         MPI2_CONFIG_ACTION_PAGE_READ_CURRENT,
1532         MPI2_CONFIG_EXTPAGETYPE_SAS_DEVICE, 0, page_address,
1533         mptsas_sasdevpage_0_cb, page_address, dev_handle, sas_wnn,
1534         dev_info, physport, phynum, pdev_handle,
1535         bay_num, enclosure, io_flags);
1537     return (rval);
1538 }
1540 static int
1541 mptsas_sasexpdpag0_cb(mptsas_t *mpt, caddr_t page_memp,
1542     ddi_acc_handle_t accessp, uint16_t iocstatus, uint32_t iocloginfo,
1543     va_list ap)
1544 {
1545     #ifndef __lock_lint
1546     _NOTE(ARGUNUSED(ap))
1547     #endif
1548     pMpi2ExpanderPage0_t expddvpage;
1549     int rval = DDI_SUCCESS, i;
1550     uint8_t *sas_addr = NULL;
1551     tmp_sas_wnn[SAS_WWN_BYTE_SIZE];
1552     *devhdl;
1553     *sas_wnn;
1554     physport;
1555     mptsas_phymask_t *phymask;
1556     uint16_t *pdevhdl;
1557     uint32_t page_address;
1559     if ((iocstatus != MPI2_IOCSTATUS_SUCCESS) &&
1560         (iocstatus != MPI2_IOCSTATUS_CONFIG_INVALID_PAGE)) {
1561         mptsas_log(mpt, CE_WARN, "mptsas_get_sas_expander_page0 "
1562             "config: IOCStatus=0x%x, IOCLogInfo=0x%x",
1563             iocstatus, iocloginfo);
1564         rval = DDI_FAILURE;
1565         return (rval);
1566     }
1567     page_address = va_arg(ap, uint32_t);
1568     /*
1569     * The INVALID_PAGE status is normal if using GET_NEXT_HANDLE and there
1570     * are no more pages. If everything is OK up to this point but the
1571     * status is INVALID_PAGE, change rval to FAILURE and quit. Also,
1572     * signal that device traversal is complete.
1573     */
1574     if (iocstatus == MPI2_IOCSTATUS_CONFIG_INVALID_PAGE) {
1575         if ((page_address & MPI2_SAS_EXPAND_PGAD_FORM_MASK) ==
1576             MPI2_SAS_EXPAND_PGAD_FORM_GET_NEXT_HNDL) {
1577             mpt->m_done_traverse_smp = 1;
1578         }
1579         rval = DDI_FAILURE;

```

```

1580         return (rval);
1581     }
1582     devhdl = va_arg(ap, uint16_t *);
1583     sas_wnn = va_arg(ap, uint64_t *);
1584     phymask = va_arg(ap, mptsas_phymask_t *);
1585     pdevhdl = va_arg(ap, uint16_t *);

1587     expddvpage = (pMpi2ExpanderPage0_t)page_memp;

1589     *devhdl = ddi_get16(accessp, &expddvpage->DevHandle);
1590     physport = ddi_get8(accessp, &expddvpage->PhysicalPort);
1591     *phymask = mptsas_physport_to_phymask(mpt, physport);
1592     *pdevhdl = ddi_get16(accessp, &expddvpage->ParentDevHandle);
1593     sas_addr = (uint8_t *)(&expddvpage->SASAddress);
1594     for (i = 0; i < SAS_WWN_BYTE_SIZE; i++) {
1595         tmp_sas_wnn[i] = ddi_get8(accessp, sas_addr + i);
1596     }
1597     bcopy(tmp_sas_wnn, sas_wnn, SAS_WWN_BYTE_SIZE);
1598     *sas_wnn = LE_64(*sas_wnn);

1600     return (rval);
1601 }

1603 /*
1604  * Request MPI configuration page SAS device page 0 to get DevHandle, phymask
1605  * and SAS address.
1606  */
1607 int
1608 mptsas_get_sas_expander_page0(mptsas_t *mpt, uint32_t page_address,
1609     mptsas_smp_t *info)
1610 {
1611     int rval = DDI_SUCCESS;

1613     ASSERT(mutex_owned(&mpt->m_mutex));

1615     /*
1616      * Get the header and config page.  reply contains the reply frame,
1617      * which holds status info for the request.
1618      */
1619     rval = mptsas_access_config_page(mpt,
1620         MPI2_CONFIG_ACTION_PAGE_READ_CURRENT,
1621         MPI2_CONFIG_EXTPAGETYPE_SAS_EXPANDER, 0, page_address,
1622         mptsas_sasexpdpdp0_cb, page_address, &info->m_devhdl,
1623         &info->m_addr.mta_wnn, &info->m_addr.mta_phymask, &info->m_pdevhdl);

1625     return (rval);
1626 }

1628 static int
1629 mptsas_sasportpage_0_cb(mptsas_t *mpt, caddr_t page_memp,
1630     ddi_acc_handle_t accessp, uint16_t iocstatus, uint32_t iocloginfo,
1631     va_list ap)
1632 {
1633     #ifndef __lock_lint
1634         _NOTE(ARGUNUSED(ap))
1635     #endif
1636     int rval = DDI_SUCCESS, i;
1637     uint8_t *sas_addr = NULL;
1638     uint64_t *sas_wnn;
1639     uint8_t tmp_sas_wnn[SAS_WWN_BYTE_SIZE];
1640     uint8_t *portwidth;
1641     pMpi2SasPortPage0_t sasportpage;

1643     if (iocstatus != MPI2_IOCSTATUS_SUCCESS) {
1644         mptsas_log(mpt, CE_WARN, "mptsas_get_sas_port_page0 "
1645             "config: IOCStatus=0x%x, IOCLogInfo=0x%x",

```

```

1646         iocstatus, iocloginfo);
1647         rval = DDI_FAILURE;
1648         return (rval);
1649     }
1650     sas_wnn = va_arg(ap, uint64_t *);
1651     portwidth = va_arg(ap, uint8_t *);

1653     sasportpage = (pMpi2SasPortPage0_t)page_memp;
1654     sas_addr = (uint8_t *)(&sasportpage->SASAddress);
1655     for (i = 0; i < SAS_WWN_BYTE_SIZE; i++) {
1656         tmp_sas_wnn[i] = ddi_get8(accessp, sas_addr + i);
1657     }
1658     bcopy(tmp_sas_wnn, sas_wnn, SAS_WWN_BYTE_SIZE);
1659     *sas_wnn = LE_64(*sas_wnn);
1660     *portwidth = ddi_get8(accessp, &sasportpage->PortWidth);
1661     return (rval);
1662 }

1664 /*
1665  * Request MPI configuration page SAS port page 0 to get initiator SAS address
1666  * and port width.
1667  */
1668 int
1669 mptsas_get_sas_port_page0(mptsas_t *mpt, uint32_t page_address,
1670     uint64_t *sas_wnn, uint8_t *portwidth)
1671 {
1672     int rval = DDI_SUCCESS;

1674     ASSERT(mutex_owned(&mpt->m_mutex));

1676     /*
1677      * Get the header and config page.  reply contains the reply frame,
1678      * which holds status info for the request.
1679      */
1680     rval = mptsas_access_config_page(mpt,
1681         MPI2_CONFIG_ACTION_PAGE_READ_CURRENT,
1682         MPI2_CONFIG_EXTPAGETYPE_SAS_PORT, 0, page_address,
1683         mptsas_sasportpage_0_cb, sas_wnn, portwidth);

1685     return (rval);
1686 }

1688 static int
1689 mptsas_sasiou_page_0_cb(mptsas_t *mpt, caddr_t page_memp,
1690     ddi_acc_handle_t accessp, uint16_t iocstatus, uint32_t iocloginfo,
1691     va_list ap)
1692 {
1693     #ifndef __lock_lint
1694         _NOTE(ARGUNUSED(ap))
1695     #endif
1696     int rval = DDI_SUCCESS;
1697     pMpi2SasIOUnitPage0_t sasioupage0;
1698     int i, num_phys;
1699     uint32_t cpdi[MPTSAS_MAX_PHYS], *retrypage0, *readpagel;
1700     uint8_t port_flags;

1702     if (iocstatus != MPI2_IOCSTATUS_SUCCESS) {
1703         mptsas_log(mpt, CE_WARN, "mptsas_get_sas_io_unit_page0 "
1704             "config: IOCStatus=0x%x, IOCLogInfo=0x%x",
1705             iocstatus, iocloginfo);
1706         rval = DDI_FAILURE;
1707         return (rval);
1708     }
1709     readpagel = va_arg(ap, uint32_t *);
1710     retrypage0 = va_arg(ap, uint32_t *);

```

```

1712     sasioupage0 = (pMpi2SasIOUnitPage0_t)page_memp;
1713
1714     num_phys = ddi_get8(accessp, &sasioupage0->NumPhys);
1715     /*
1716      * ASSERT that the num_phys value in SAS IO Unit Page 0 is the same as
1717      * was initially set. This should never change throughout the life of
1718      * the driver.
1719      */
1720     ASSERT(num_phys == mpt->m_num_phys);
1721     for (i = 0; i < num_phys; i++) {
1722         cpdi[i] = ddi_get32(accessp,
1723             &sasioupage0->PhyData[i].
1724             ControllerPhyDeviceInfo);
1725         port_flags = ddi_get8(accessp,
1726             &sasioupage0->PhyData[i].PortFlags);
1727         mpt->m_phy_info[i].port_num =
1728             ddi_get8(accessp,
1729             &sasioupage0->PhyData[i].Port);
1730         mpt->m_phy_info[i].ctrl_devhdl =
1731             ddi_get16(accessp, &sasioupage0->
1732             PhyData[i].ControllerDevHandle);
1733         mpt->m_phy_info[i].attached_devhdl =
1734             ddi_get16(accessp, &sasioupage0->
1735             PhyData[i].AttachedDevHandle);
1736         mpt->m_phy_info[i].phy_device_type = cpdi[i];
1737         mpt->m_phy_info[i].port_flags = port_flags;
1738
1739         if (port_flags & DISCOVERY_IN_PROGRESS) {
1740             *retrypage0 = *retrypage0 + 1;
1741             break;
1742         } else {
1743             *retrypage0 = 0;
1744         }
1745         if (!(port_flags & AUTO_PORT_CONFIGURATION)) {
1746             /*
1747              * some PHY configuration described in
1748              * SAS IO Unit Page1
1749              */
1750             *readpage1 = 1;
1751         }
1752     }
1753
1754     return (rval);
1755 }
1756
1757 static int
1758 mptsas_sasiou_page_1_cb(mptsas_t *mpt, caddr_t page_memp,
1759     ddi_acc_handle_t accessp, uint16_t iocstatus, uint32_t iocloginfo,
1760     va_list ap)
1761 {
1762     #ifndef __lock_lint
1763     _NOTE(ARGUNUSED(ap))
1764     #endif
1765     int rval = DDI_SUCCESS;
1766     pMpi2SasIOUnitPage1_t sasioupage1;
1767     int i, num_phys;
1768     uint32_t cpdi[MPTSAS_MAX_PHYS];
1769     uint8_t port_flags;
1770
1771     if (iocstatus != MPI2_IOCSTATUS_SUCCESS) {
1772         mptsas_log(mpt, CE_WARN, "mptsas_get_sas_io_unit_page1 "
1773             "config: IOCStatus=0x%x, IOCLogInfo=0x%x",
1774             iocstatus, iocloginfo);
1775         rval = DDI_FAILURE;
1776         return (rval);
1777     }

```

```

1779     sasioupage1 = (pMpi2SasIOUnitPage1_t)page_memp;
1780     num_phys = ddi_get8(accessp, &sasioupage1->NumPhys);
1781     /*
1782      * ASSERT that the num_phys value in SAS IO Unit Page 1 is the same as
1783      * was initially set. This should never change throughout the life of
1784      * the driver.
1785      */
1786     ASSERT(num_phys == mpt->m_num_phys);
1787     for (i = 0; i < num_phys; i++) {
1788         cpdi[i] = ddi_get32(accessp, &sasioupage1->PhyData[i].
1789             ControllerPhyDeviceInfo);
1790         port_flags = ddi_get8(accessp,
1791             &sasioupage1->PhyData[i].PortFlags);
1792         mpt->m_phy_info[i].port_num =
1793             ddi_get8(accessp,
1794             &sasioupage1->PhyData[i].Port);
1795         mpt->m_phy_info[i].port_flags = port_flags;
1796         mpt->m_phy_info[i].phy_device_type = cpdi[i];
1797     }
1798     return (rval);
1799 }
1800
1801 /*
1802  * Read IO unit page 0 to get information for each PHY. If needed, Read IO Unit
1803  * page1 to update the PHY information. This is the message passing method of
1804  * this function which should be called except during initialization.
1805  */
1806 int
1807 mptsas_get_sas_io_unit_page(mptsas_t *mpt)
1808 {
1809     int rval = DDI_SUCCESS, state;
1810     uint32_t readpage1 = 0, retrypage0 = 0;
1811
1812     ASSERT(mutex_owned(&mpt->m_mutex));
1813
1814     /*
1815      * Now we cycle through the state machine. Here's what happens:
1816      * 1. Read IO unit page 0 and set phy information
1817      * 2. See if Read IO unit page1 is needed because of port configuration
1818      * 3. Read IO unit page 1 and update phy information.
1819      */
1820     state = IOUC_READ_PAGE0;
1821     while (state != IOUC_DONE) {
1822         if (state == IOUC_READ_PAGE0) {
1823             rval = mptsas_access_config_page(mpt,
1824                 MPI2_CONFIG_ACTION_PAGE_READ_CURRENT,
1825                 MPI2_CONFIG_EXTPAGE_TYPE_SAS_IO_UNIT, 0, 0,
1826                 mptsas_sasiou_page_0_cb, &readpage1,
1827                 &retrypage0);
1828         } else if (state == IOUC_READ_PAGE1) {
1829             rval = mptsas_access_config_page(mpt,
1830                 MPI2_CONFIG_ACTION_PAGE_READ_CURRENT,
1831                 MPI2_CONFIG_EXTPAGE_TYPE_SAS_IO_UNIT, 1, 0,
1832                 mptsas_sasiou_page_1_cb);
1833         }
1834
1835         if (rval == DDI_SUCCESS) {
1836             switch (state) {
1837                 case IOUC_READ_PAGE0:
1838                     /*
1839                      * retry 30 times if discovery is in process
1840                      */
1841                     if (retrypage0 && (retrypage0 < 30)) {
1842                         drv_usecwait(1000 * 100);
1843                         state = IOUC_READ_PAGE0;

```

```

1844         break;
1845     } else if (retrypage0 == 30) {
1846         mptsas_log(mpt, CE_WARN,
1847             "!Discovery in progress, can't "
1848             "verify IO unit config, then "
1849             "after 30 times retry, give "
1850             "up!");
1851         state = IOUC_DONE;
1852         rval = DDI_FAILURE;
1853         break;
1854     }
1855
1856     if (readpage1 == 0) {
1857         state = IOUC_DONE;
1858         rval = DDI_SUCCESS;
1859         break;
1860     }
1861
1862     state = IOUC_READ_PAGE1;
1863     break;
1864
1865     case IOUC_READ_PAGE1:
1866         state = IOUC_DONE;
1867         rval = DDI_SUCCESS;
1868         break;
1869     }
1870 } else {
1871     return (rval);
1872 }
1873
1874 return (rval);
1875 }
1876
1877 static int
1878 mptsas_biospage_3_cb(mptsas_t *mpt, caddr_t page_memp,
1879     ddi_acc_handle_t accessp, uint16_t iocstatus, uint32_t iocloginfo,
1880     va_list ap)
1881 {
1882     #ifndef __lock_lint
1883     _NOTE(ARGUNUSED(ap))
1884     #endif
1885     #endif
1886     pMpi2BiosPage3_t      sasbiospage;
1887     int                   rval = DDI_SUCCESS;
1888     uint32_t              *bios_version;
1889
1890     if ((iocstatus != MPI2_IOCSTATUS_SUCCESS) &&
1891         (iocstatus != MPI2_IOCSTATUS_CONFIG_INVALID_PAGE)) {
1892         mptsas_log(mpt, CE_WARN, "mptsas_get_bios_page3 header: "
1893             "IOCStatus=0x%x, IOCLogInfo=0x%x", iocstatus, iocloginfo);
1894         rval = DDI_FAILURE;
1895         return (rval);
1896     }
1897     bios_version = va_arg(ap, uint32_t *);
1898     sasbiospage = (pMpi2BiosPage3_t)page_memp;
1899     *bios_version = ddi_get32(accessp, &sasbiospage->BiosVersion);
1900
1901     return (rval);
1902 }
1903
1904 /*
1905  * Request MPI configuration page BIOS page 3 to get BIOS version. Since all
1906  * other information in this page is not needed, just ignore it.
1907  */
1908 int
1909 mptsas_get_bios_page3(mptsas_t *mpt, uint32_t *bios_version)

```

```

1910 {
1911     int rval = DDI_SUCCESS;
1912
1913     ASSERT(mutex_owned(&mpt->m_mutex));
1914
1915     /*
1916      * Get the header and config page. reply contains the reply frame,
1917      * which holds status info for the request.
1918      */
1919     rval = mptsas_access_config_page(mpt,
1920         MPI2_CONFIG_ACTION_PAGE_READ_CURRENT, MPI2_CONFIG_PAGETYPE_BIOS, 3,
1921         0, mptsas_biospage_3_cb, bios_version);
1922
1923     return (rval);
1924 }
1925
1926 /*
1927  * Read IO unit page 0 to get information for each PHY. If needed, Read IO Unit
1928  * page1 to update the PHY information. This is the handshaking version of
1929  * this function, which should be called during initialization only.
1930  */
1931 int
1932 mptsas_get_sas_io_unit_page_hndshk(mptsas_t *mpt)
1933 {
1934     ddi_dma_attr_t      recv_dma_attrs, page_dma_attrs;
1935     ddi_dma_cookie_t     page_cookie;
1936     ddi_dma_handle_t     recv_dma_handle, page_dma_handle;
1937     ddi_acc_handle_t     recv_accessp, page_accessp;
1938     pMpi2ConfigReply_t   configreply;
1939     pMpi2SasIOUnitPage0_t sasioupage0;
1940     pMpi2SasIOUnitPage1_t sasioupage1;
1941     int                  recv_numbytes;
1942     caddr_t              recv_memp, page_memp;
1943     int                  i, num_phys, start_phy = 0;
1944     int                  page0_size =
1945         sizeof (MPI2_CONFIG_PAGE_SASIOUNIT_0) +
1946         (sizeof (MPI2_SAS_IO_UNIT0_PHY_DATA) * (MPTSAS_MAX_PHYS - 1));
1947     int                  page1_size =
1948         sizeof (MPI2_CONFIG_PAGE_SASIOUNIT_1) +
1949         (sizeof (MPI2_SAS_IO_UNIT1_PHY_DATA) * (MPTSAS_MAX_PHYS - 1));
1950     uint32_t             flags_length;
1951     uint32_t             cpdi[MPTSAS_MAX_PHYS];
1952     uint32_t             readpage1 = 0, retrypage0 = 0;
1953     uint16_t             iocstatus;
1954     uint8_t              port_flags, page_number, action;
1955     uint32_t             reply_size = 256; /* Big enough for any page */
1956     uint_t               state;
1957     int                  rval = DDI_FAILURE;
1958     boolean_t            free_recv = B_FALSE, free_page = B_FALSE;
1959
1960     /*
1961      * Initialize our "state machine". This is a bit convoluted,
1962      * but it keeps us from having to do the ddi allocations numerous
1963      * times.
1964      */
1965
1966     NDBG20(("mptsas_get_sas_io_unit_page_hndshk enter"));
1967     ASSERT(mutex_owned(&mpt->m_mutex));
1968     state = IOUC_READ_PAGE0;
1969
1970     /*
1971      * dynamically create a customized dma attribute structure
1972      * that describes mpt's config reply page request structure.
1973      */
1974     recv_dma_attrs = mpt->m_msg_dma_attr;
1975     recv_dma_attrs.dma_attr_sgllen = 1;

```

```

1976     recv_dma_attrs.dma_attr_granular = (sizeof (MPI2_CONFIG_REPLY));
1978     if (mptsas_dma_addr_create(mpt, recv_dma_attrs,
1979         &recv_dma_handle, &recv_accessp, &recv_memp,
1980         (sizeof (MPI2_CONFIG_REPLY)), NULL) == FALSE) {
1981         mptsas_log(mpt, CE_WARN,
1982             "mptsas_get_sas_io_unit_page_hndshk: recv dma failed");
1983         goto cleanup;
1984     }
1985     /* Now safe to call mptsas_dma_addr_destroy(recv_dma_handle). */
1986     free_recv = B_TRUE;

1988     page_dma_attrs = mpt->m_msg_dma_attr;
1989     page_dma_attrs.dma_attr_sgllen = 1;
1990     page_dma_attrs.dma_attr_granular = reply_size;

1992     if (mptsas_dma_addr_create(mpt, page_dma_attrs,
1993         &page_dma_handle, &page_accessp, &page_memp,
1994         reply_size, &page_cookie) == FALSE) {
1995         mptsas_log(mpt, CE_WARN,
1996             "mptsas_get_sas_io_unit_page_hndshk: page dma failed");
1997         goto cleanup;
1998     }
1999     /* Now safe to call mptsas_dma_addr_destroy(page_dma_handle). */
2000     free_page = B_TRUE;

2002     /*
2003      * Now we cycle through the state machine. Here's what happens:
2004      * 1. Read IO unit page 0 and set phy information
2005      * 2. See if Read IO unit page1 is needed because of port configuration
2006      * 3. Read IO unit page 1 and update phy information.
2007      */

2009     sasioupage0 = (pMpi2SasIOUnitPage0_t)page_memp;
2010     sasioupage1 = (pMpi2SasIOUnitPage1_t)page_memp;

2012     while (state != IOUC_DONE) {
2013         switch (state) {
2014             case IOUC_READ_PAGE0:
2015                 page_number = 0;
2016                 action = MPI2_CONFIG_ACTION_PAGE_READ_CURRENT;
2017                 flags_length = (uint32_t)page0_size;
2018                 flags_length |= ((uint32_t)(
2019                     MPI2_SGE_FLAGS_LAST_ELEMENT |
2020                     MPI2_SGE_FLAGS_END_OF_BUFFER |
2021                     MPI2_SGE_FLAGS_SIMPLE_ELEMENT |
2022                     MPI2_SGE_FLAGS_SYSTEM_ADDRESS |
2023                     MPI2_SGE_FLAGS_64_BIT_ADDRESSING |
2024                     MPI2_SGE_FLAGS_IOC_TO_HOST |
2025                     MPI2_SGE_FLAGS_END_OF_LIST) <<
2026                     MPI2_SGE_FLAGS_SHIFT);

2028                 break;

2030             case IOUC_READ_PAGE1:
2031                 page_number = 1;
2032                 action = MPI2_CONFIG_ACTION_PAGE_READ_CURRENT;
2033                 flags_length = (uint32_t)page1_size;
2034                 flags_length |= ((uint32_t)(
2035                     MPI2_SGE_FLAGS_LAST_ELEMENT |
2036                     MPI2_SGE_FLAGS_END_OF_BUFFER |
2037                     MPI2_SGE_FLAGS_SIMPLE_ELEMENT |
2038                     MPI2_SGE_FLAGS_SYSTEM_ADDRESS |
2039                     MPI2_SGE_FLAGS_64_BIT_ADDRESSING |
2040                     MPI2_SGE_FLAGS_IOC_TO_HOST |
2041                     MPI2_SGE_FLAGS_END_OF_LIST) <<

```

```

2042         MPI2_SGE_FLAGS_SHIFT);

2044         break;
2045     default:
2046         break;
2047     }

2049     bzero(recv_memp, sizeof (MPI2_CONFIG_REPLY));
2050     configreply = (pMpi2ConfigReply_t)recv_memp;
2051     recv_numbytes = sizeof (MPI2_CONFIG_REPLY);

2053     if (mptsas_send_extended_config_request_msg(mpt,
2054         MPI2_CONFIG_ACTION_PAGE_HEADER,
2055         MPI2_CONFIG_EXTPAGETYPE_SAS_IO_UNIT,
2056         0, page_number, 0, 0, 0, 0)) {
2057         goto cleanup;
2058     }

2060     if (mptsas_get_handshake_msg(mpt, recv_memp, recv_numbytes,
2061         &recv_accessp)) {
2062         goto cleanup;
2063     }

2065     iocstatus = ddi_get16(recv_accessp, &configreply->IOCStatus);
2066     iocstatus = MPTSAS_IOCSTATUS(iocstatus);

2068     if (iocstatus != MPI2_IOCSTATUS_SUCCESS) {
2069         mptsas_log(mpt, CE_WARN,
2070             "mptsas_get_sas_io_unit_page_hndshk: read page "
2071             "header iocstatus = 0x%x", iocstatus);
2072         goto cleanup;
2073     }

2075     if (action != MPI2_CONFIG_ACTION_PAGE_WRITE_NVRAM) {
2076         bzero(page_memp, reply_size);
2077     }

2079     if (mptsas_send_extended_config_request_msg(mpt, action,
2080         MPI2_CONFIG_EXTPAGETYPE_SAS_IO_UNIT, 0, page_number,
2081         ddi_get8(recv_accessp, &configreply->Header.PageVersion),
2082         ddi_get16(recv_accessp, &configreply->ExtPageLength),
2083         flags_length, page_cookie.dmac_laddress)) {
2084         goto cleanup;
2085     }

2087     if (mptsas_get_handshake_msg(mpt, recv_memp, recv_numbytes,
2088         &recv_accessp)) {
2089         goto cleanup;
2090     }

2092     iocstatus = ddi_get16(recv_accessp, &configreply->IOCStatus);
2093     iocstatus = MPTSAS_IOCSTATUS(iocstatus);

2095     if (iocstatus != MPI2_IOCSTATUS_SUCCESS) {
2096         mptsas_log(mpt, CE_WARN,
2097             "mptsas_get_sas_io_unit_page_hndshk: IO unit "
2098             "config failed for action %d, iocstatus = 0x%x",
2099             action, iocstatus);
2100         goto cleanup;
2101     }

2103     switch (state) {
2104         case IOUC_READ_PAGE0:
2105             if ((ddi_dma_sync(page_dma_handle, 0, 0,
2106                 DDI_DMA_SYNC_FORCPU) != DDI_SUCCESS) {
2107                 goto cleanup;

```

```

2108     }
2110     num_phys = ddi_get8(page_accesssp,
2111         &sasioupage0->NumPhys);
2112     ASSERT(num_phys == mpt->m_num_phys);
2113     if (num_phys > MPTSAS_MAX_PHYS) {
2114         mptsas_log(mpt, CE_WARN, "Number of phys "
2115             "supported by HBA (%d) is more than max "
2116             "supported by driver (%d). Driver will "
2117             "not attach.", num_phys,
2118             MPTSAS_MAX_PHYS);
2119         rval = DDI_FAILURE;
2120         goto cleanup;
2121     }
2122     for (i = start_phy; i < num_phys; i++, start_phy = i) {
2123         cpdi[i] = ddi_get32(page_accesssp,
2124             &sasioupage0->PhyData[i].
2125             ControllerPhyDeviceInfo);
2126         port_flags = ddi_get8(page_accesssp,
2127             &sasioupage0->PhyData[i].PortFlags);
2129         mpt->m_phy_info[i].port_num =
2130             ddi_get8(page_accesssp,
2131                 &sasioupage0->PhyData[i].Port);
2132         mpt->m_phy_info[i].ctrl_devhdl =
2133             ddi_get16(page_accesssp, &sasioupage0->
2134                 PhyData[i].ControllerDevHandle);
2135         mpt->m_phy_info[i].attached_devhdl =
2136             ddi_get16(page_accesssp, &sasioupage0->
2137                 PhyData[i].AttachedDevHandle);
2138         mpt->m_phy_info[i].phy_device_type = cpdi[i];
2139         mpt->m_phy_info[i].port_flags = port_flags;
2141         if (port_flags & DISCOVERY_IN_PROGRESS) {
2142             retrypage0++;
2143             NDBG20(("Discovery in progress, can't "
2144                 "verify IO unit config, then NO.%d"
2145                 " times retry", retrypage0));
2146             break;
2147         } else {
2148             retrypage0 = 0;
2149         }
2150         if (!(port_flags & AUTO_PORT_CONFIGURATION)) {
2151             /*
2152              * some PHY configuration described in
2153              * SAS IO Unit Page1
2154              */
2155             readpage1 = 1;
2156         }
2157     }
2159     /*
2160     * retry 30 times if discovery is in process
2161     */
2162     if (retrypage0 && (retrypage0 < 30)) {
2163         drv_usecwait(1000 * 100);
2164         state = IOUC_READ_PAGE0;
2165         break;
2166     } else if (retrypage0 == 30) {
2167         mptsas_log(mpt, CE_WARN,
2168             "!Discovery in progress, can't "
2169             "verify IO unit config, then after "
2170             " 30 times retry, give up!");
2171         state = IOUC_DONE;
2172         rval = DDI_FAILURE;
2173         break;

```

```

2174     }
2176     if (readpage1 == 0) {
2177         state = IOUC_DONE;
2178         rval = DDI_SUCCESS;
2179         break;
2180     }
2182     state = IOUC_READ_PAGE1;
2183     break;
2185     case IOUC_READ_PAGE1:
2186         if ((ddi_dma_sync(page_dma_handle, 0, 0,
2187             DDI_DMA_SYNC_FORCPU)) != DDI_SUCCESS) {
2188             goto cleanup;
2189         }
2191         num_phys = ddi_get8(page_accesssp,
2192             &sasioupage1->NumPhys);
2193         ASSERT(num_phys == mpt->m_num_phys);
2194         if (num_phys > MPTSAS_MAX_PHYS) {
2195             mptsas_log(mpt, CE_WARN, "Number of phys "
2196                 "supported by HBA (%d) is more than max "
2197                 "supported by driver (%d). Driver will "
2198                 "not attach.", num_phys,
2199                 MPTSAS_MAX_PHYS);
2200             rval = DDI_FAILURE;
2201             goto cleanup;
2202         }
2203         for (i = 0; i < num_phys; i++) {
2204             cpdi[i] = ddi_get32(page_accesssp,
2205                 &sasioupage1->PhyData[i].
2206                 ControllerPhyDeviceInfo);
2207             port_flags = ddi_get8(page_accesssp,
2208                 &sasioupage1->PhyData[i].PortFlags);
2209             mpt->m_phy_info[i].port_num =
2210                 ddi_get8(page_accesssp,
2211                     &sasioupage1->PhyData[i].Port);
2212             mpt->m_phy_info[i].port_flags = port_flags;
2213             mpt->m_phy_info[i].phy_device_type = cpdi[i];
2215         }
2217         state = IOUC_DONE;
2218         rval = DDI_SUCCESS;
2219         break;
2220     }
2221     }
2222     if ((mptsas_check_dma_handle(recv_dma_handle) != DDI_SUCCESS) ||
2223         (mptsas_check_dma_handle(page_dma_handle) != DDI_SUCCESS)) {
2224         ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_UNAFFECTED);
2225         rval = DDI_FAILURE;
2226         goto cleanup;
2227     }
2228     if ((mptsas_check_acc_handle(recv_accesssp) != DDI_SUCCESS) ||
2229         (mptsas_check_acc_handle(page_accesssp) != DDI_SUCCESS)) {
2230         ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_UNAFFECTED);
2231         rval = DDI_FAILURE;
2232         goto cleanup;
2233     }
2235 cleanup:
2236     if (free_recv)
2237         mptsas_dma_addr_destroy(&recv_dma_handle, &recv_accesssp);
2238     if (free_page)
2239         mptsas_dma_addr_destroy(&page_dma_handle, &page_accesssp);

```

```

2240     if (rval != DDI_SUCCESS) {
2241         mptsas_fm_ereport(mpt, DDI_FM_DEVICE_NO_RESPONSE);
2242         ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_LOST);
2243     }
2244     return (rval);
2245 }

2247 /*
2248  * mptsas_get_manufacture_page5
2249  */
2250 * This function will retrieve the base WWID from the adapter. Since this
2251 * function is only called during the initialization process, use handshaking.
2252 */
2253 int
2254 mptsas_get_manufacture_page5(mptsas_t *mpt)
2255 {
2256     ddi_dma_attr_t      recv_dma_attrs, page_dma_attrs;
2257     ddi_dma_cookie_t    page_cookie;
2258     ddi_dma_handle_t    recv_dma_handle, page_dma_handle;
2259     ddi_acc_handle_t    recv_accesssp, page_accesssp;
2260     pMpi2ConfigReply_t  configreply;
2261     caddr_t             recv_memp, page_memp;
2262     int                 recv_numbytes;
2263     pMpi2ManufacturingPage5_t m5;
2264     uint32_t            flagslength;
2265     int                 rval = DDI_SUCCESS;
2266     uint_t              iocstatus;
2267     boolean_t           free_recv = B_FALSE, free_page = B_FALSE;

2269     MPTSAS_DISABLE_INTR(mpt);

2271     if (mptsas_send_config_request_msg(mpt, MPI2_CONFIG_ACTION_PAGE_HEADER,
2272         MPI2_CONFIG_PAGETYPE_MANUFACTURING, 0, 5, 0, 0, 0, 0)) {
2273         rval = DDI_FAILURE;
2274         goto done;
2275     }

2277     /*
2278      * dynamically create a customized dma attribute structure
2279      * that describes the MPT's config reply page request structure.
2280      */
2281     recv_dma_attrs = mpt->m_msg_dma_attr;
2282     recv_dma_attrs.dma_attr_sgllen = 1;
2283     recv_dma_attrs.dma_attr_granular = (sizeof (MPI2_CONFIG_REPLY));

2285     if (mptsas_dma_addr_create(mpt, recv_dma_attrs,
2286         &recv_dma_handle, &recv_accesssp, &recv_memp,
2287         (sizeof (MPI2_CONFIG_REPLY)), NULL) == FALSE) {
2288         rval = DDI_FAILURE;
2289         goto done;
2290     }
2291     /* Now safe to call mptsas_dma_addr_destroy(recv_dma_handle). */
2292     free_recv = B_TRUE;

2294     bzero(recv_memp, sizeof (MPI2_CONFIG_REPLY));
2295     configreply = (pMpi2ConfigReply_t)recv_memp;
2296     recv_numbytes = sizeof (MPI2_CONFIG_REPLY);

2298     /*
2299      * get config reply message
2300      */
2301     if (mptsas_get_handshake_msg(mpt, recv_memp, recv_numbytes,
2302         recv_accesssp)) {
2303         rval = DDI_FAILURE;
2304         goto done;
2305     }

```

```

2307     if (iocstatus = ddi_get16(recv_accesssp, &configreply->IOCStatus)) {
2308         mptsas_log(mpt, CE_WARN, "mptsas_get_manufacture_page5 update: "
2309             "IOCStatus=0x%x, IOCLogInfo=0x%x", iocstatus,
2310             ddi_get32(recv_accesssp, &configreply->IOCLogInfo));
2311         goto done;
2312     }

2314     /*
2315      * dynamically create a customized dma attribute structure
2316      * that describes the MPT's config page structure.
2317      */
2318     page_dma_attrs = mpt->m_msg_dma_attr;
2319     page_dma_attrs.dma_attr_sgllen = 1;
2320     page_dma_attrs.dma_attr_granular = (sizeof (MPI2_CONFIG_PAGE_MAN_5));

2322     if (mptsas_dma_addr_create(mpt, page_dma_attrs, &page_dma_handle,
2323         &page_accesssp, &page_memp, (sizeof (MPI2_CONFIG_PAGE_MAN_5)),
2324         &page_cookie) == FALSE) {
2325         rval = DDI_FAILURE;
2326         goto done;
2327     }
2328     /* Now safe to call mptsas_dma_addr_destroy(page_dma_handle). */
2329     free_page = B_TRUE;

2331     bzero(page_memp, sizeof (MPI2_CONFIG_PAGE_MAN_5));
2332     m5 = (pMpi2ManufacturingPage5_t)page_memp;
2333     NDBG20(("mptsas_get_manufacture_page5: paddr 0x%x%08x",
2334         (uint32_t)(page_cookie.dmac_laddress>>32),
2335         (uint32_t)(page_cookie.dmac_laddress&0xffffffff)));

2337     /*
2338      * Give reply address to IOC to store config page in and send
2339      * config request out.
2340      */

2342     flagslength = sizeof (MPI2_CONFIG_PAGE_MAN_5);
2343     flagslength |= ((uint32_t)(MPI2_SGE_FLAGS_LAST_ELEMENT |
2344         MPI2_SGE_FLAGS_END_OF_BUFFER | MPI2_SGE_FLAGS_SIMPLE_ELEMENT |
2345         MPI2_SGE_FLAGS_SYSTEM_ADDRESS | MPI2_SGE_FLAGS_64_BIT_ADDRESSING |
2346         MPI2_SGE_FLAGS_IOC_TO_HOST |
2347         MPI2_SGE_FLAGS_END_OF_LIST) << MPI2_SGE_FLAGS_SHIFT);

2349     if (mptsas_send_config_request_msg(mpt,
2350         MPI2_CONFIG_ACTION_PAGE_READ_CURRENT,
2351         MPI2_CONFIG_PAGETYPE_MANUFACTURING, 0, 5,
2352         ddi_get8(recv_accesssp, &configreply->Header.PageVersion),
2353         ddi_get8(recv_accesssp, &configreply->Header.PageLength),
2354         flagslength, page_cookie.dmac_laddress)) {
2355         rval = DDI_FAILURE;
2356         goto done;
2357     }

2359     /*
2360      * get reply view handshake
2361      */
2362     if (mptsas_get_handshake_msg(mpt, recv_memp, recv_numbytes,
2363         recv_accesssp)) {
2364         rval = DDI_FAILURE;
2365         goto done;
2366     }

2368     if (iocstatus = ddi_get16(recv_accesssp, &configreply->IOCStatus)) {
2369         mptsas_log(mpt, CE_WARN, "mptsas_get_manufacture_page5 config: "
2370             "IOCStatus=0x%x, IOCLogInfo=0x%x", iocstatus,
2371             ddi_get32(recv_accesssp, &configreply->IOCLogInfo));

```



```

2372         goto done;
2373     }

2375     (void) ddi_dma_sync(page_dma_handle, 0, 0, DDI_DMA_SYNC_FORCPU);

2377     /*
2378      * Fusion-MPT stores fields in little-endian format. This is
2379      * why the low-order 32 bits are stored first.
2380      */
2381     mpt->un.sasaddr.m_base_wwid_lo =
2382         ddi_get32(page_accesssp, (uint32_t *) (void *) &m5->Phy[0].WWID);
2383     mpt->un.sasaddr.m_base_wwid_hi =
2384         ddi_get32(page_accesssp, (uint32_t *) (void *) &m5->Phy[0].WWID + 1);

2386     if (ddi_prop_update_int64(DDI_DEV_T_NONE, mpt->m_dip,
2387         "base-wwid", mpt->un.m_base_wwid) != DDI_PROP_SUCCESS) {
2388         NDBG2(("s%d: failed to create base-wwid property",
2389             ddi_driver_name(mpt->m_dip), ddi_get_instance(mpt->m_dip)));
2390     }

2392     /*
2393      * Set the number of PHYs present.
2394      */
2395     mpt->m_num_phys = ddi_get8(page_accesssp, (uint8_t *) &m5->NumPhys);

2397     if (ddi_prop_update_int(DDI_DEV_T_NONE, mpt->m_dip,
2398         "num-phys", mpt->m_num_phys) != DDI_PROP_SUCCESS) {
2399         NDBG2(("s%d: failed to create num-phys property",
2400             ddi_driver_name(mpt->m_dip), ddi_get_instance(mpt->m_dip)));
2401     }

2403     mptsas_log(mpt, CE_NOTE, "!mpt%d: Initiator WWNs: 0x%016llx-0x%016llx",
2404         mpt->m_instance, (unsigned long long)mpt->un.m_base_wwid,
2405         (unsigned long long)mpt->un.m_base_wwid + mpt->m_num_phys - 1);

2407     if ((mptsas_check_dma_handle(recv_dma_handle) != DDI_SUCCESS) ||
2408         (mptsas_check_dma_handle(page_dma_handle) != DDI_SUCCESS)) {
2409         ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_UNAFFECTED);
2410         rval = DDI_FAILURE;
2411         goto done;
2412     }
2413     if ((mptsas_check_acc_handle(recv_accesssp) != DDI_SUCCESS) ||
2414         (mptsas_check_acc_handle(page_accesssp) != DDI_SUCCESS)) {
2415         ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_UNAFFECTED);
2416         rval = DDI_FAILURE;
2417     }
2418 done:
2419     /*
2420      * free up memory
2421      */
2422     if (free_recv)
2423         mptsas_dma_addr_destroy(&recv_dma_handle, &recv_accesssp);
2424     if (free_page)
2425         mptsas_dma_addr_destroy(&page_dma_handle, &page_accesssp);
2426     MPTSAS_ENABLE_INTR(mpt);

2428     return (rval);
2429 }

2431 static int
2432 mptsas_sasphypage_0_cb(mptsas_t *mpt, caddr_t page_memp,
2433     ddi_acc_handle_t accesssp, uint16_t iocstatus, uint32_t iocloginfo,
2434     va_list ap)
2435 {
2436 #ifndef __lock_lint
2437     _NOTE(ARGUNUSED(ap))

```

```

2438 #endif
2439     pMpi2SasPhyPage0_t    sasphypage;
2440     int                    rval = DDI_SUCCESS;
2441     uint16_t                *owner_devhdl, *attached_devhdl;
2442     uint8_t                 *attached_phy_identify;
2443     uint32_t                *attached_phy_info;
2444     uint8_t                 *programmed_link_rate;
2445     uint8_t                 *hw_link_rate;
2446     uint8_t                 *change_count;
2447     uint32_t                *phy_info;
2448     uint8_t                 *negotiated_link_rate;
2449     uint32_t                page_address;

2451     if ((iocstatus != MPI2_IOCSTATUS_SUCCESS) &&
2452         (iocstatus != MPI2_IOCSTATUS_CONFIG_INVALID_PAGE)) {
2453         mptsas_log(mpt, CE_WARN, "mptsas_get_sas_expander_page0 "
2454             "config: IOCStatus=0x%x, IOCLogInfo=0x%x",
2455             iocstatus, iocloginfo);
2456         rval = DDI_FAILURE;
2457         return (rval);
2458     }
2459     page_address = va_arg(ap, uint32_t);
2460     /*
2461      * The INVALID_PAGE status is normal if using GET_NEXT_HANDLE and there
2462      * are no more pages. If everything is OK up to this point but the
2463      * status is INVALID_PAGE, change rval to FAILURE and quit. Also,
2464      * signal that device traversal is complete.
2465      */
2466     if (iocstatus == MPI2_IOCSTATUS_CONFIG_INVALID_PAGE) {
2467         if ((page_address & MPI2_SAS_EXPAND_PGAD_FORM_MASK) ==
2468             MPI2_SAS_EXPAND_PGAD_FORM_GET_NEXT_HNDL) {
2469             mpt->m_done_traverse_smp = 1;
2470         }
2471         rval = DDI_FAILURE;
2472         return (rval);
2473     }
2474     owner_devhdl = va_arg(ap, uint16_t *);
2475     attached_devhdl = va_arg(ap, uint16_t *);
2476     attached_phy_identify = va_arg(ap, uint8_t *);
2477     attached_phy_info = va_arg(ap, uint32_t *);
2478     programmed_link_rate = va_arg(ap, uint8_t *);
2479     hw_link_rate = va_arg(ap, uint8_t *);
2480     change_count = va_arg(ap, uint8_t *);
2481     phy_info = va_arg(ap, uint32_t *);
2482     negotiated_link_rate = va_arg(ap, uint8_t *);

2484     sasphypage = (pMpi2SasPhyPage0_t)page_memp;

2486     *owner_devhdl =
2487         ddi_get16(accesssp, &sasphypage->OwnerDevHandle);
2488     *attached_devhdl =
2489         ddi_get16(accesssp, &sasphypage->AttachedDevHandle);
2490     *attached_phy_identify =
2491         ddi_get8(accesssp, &sasphypage->AttachedPhyIdentifier);
2492     *attached_phy_info =
2493         ddi_get32(accesssp, &sasphypage->AttachedPhyInfo);
2494     *programmed_link_rate =
2495         ddi_get8(accesssp, &sasphypage->ProgrammedLinkRate);
2496     *hw_link_rate =
2497         ddi_get8(accesssp, &sasphypage->HwLinkRate);
2498     *change_count =
2499         ddi_get8(accesssp, &sasphypage->ChangeCount);
2500     *phy_info =
2501         ddi_get32(accesssp, &sasphypage->PhyInfo);
2502     *negotiated_link_rate =
2503         ddi_get8(accesssp, &sasphypage->NegotiatedLinkRate);

```

```

2505     return (rval);
2506 }

2508 /*
2509  * Request MPI configuration page SAS phy page 0 to get DevHandle, phymask
2510  * and SAS address.
2511  */
2512 int
2513 mptsas_get_sas_phy_page0(mptsas_t *mpt, uint32_t page_address,
2514     smhba_info_t *info)
2515 {
2516     int                rval = DDI_SUCCESS;

2518     ASSERT(mutex_owned(&mpt->m_mutex));

2520     /*
2521     * Get the header and config page.  reply contains the reply frame,
2522     * which holds status info for the request.
2523     */
2524     rval = mptsas_access_config_page(mpt,
2525         MPI2_CONFIG_ACTION_PAGE_READ_CURRENT,
2526         MPI2_CONFIG_EXTPAGETYPE_SAS_PHY, 0, page_address,
2527         mptsas_sasphy_page_0_cb, page_address, &info->owner_devhdl,
2528         &info->attached_devhdl, &info->attached_phy_identify,
2529         &info->attached_phy_info, &info->programmed_link_rate,
2530         &info->hw_link_rate, &info->change_count,
2531         &info->phy_info, &info->negotiated_link_rate);

2533     return (rval);
2534 }

2536 static int
2537 mptsas_sasphy_page_1_cb(mptsas_t *mpt, caddr_t page_memp,
2538     ddi_acc_handle_t accessp, uint16_t iocstatus, uint32_t iocloginfo,
2539     va_list ap)
2540 {
2541     #ifndef __lock_lint
2542     _NOTE(ARGUNUSED(ap))
2543     #endif
2544     pMpi2SasPhyPage1_t    sasphy_page;
2545     int                    rval = DDI_SUCCESS;

2547     uint32_t               *invalid_dword_count;
2548     uint32_t               *running_disparity_error_count;
2549     uint32_t               *loss_of_dword_sync_count;
2550     uint32_t               *phy_reset_problem_count;
2551     uint32_t               page_address;

2553     if ((iocstatus != MPI2_IOCSTATUS_SUCCESS) &&
2554         (iocstatus != MPI2_IOCSTATUS_CONFIG_INVALID_PAGE)) {
2555         mptsas_log(mpt, CE_WARN, "mptsas get sas expander_page1 "
2556             "config: IOCStatus=0x%x, IOCLogInfo=0x%x",
2557             iocstatus, iocloginfo);
2558         rval = DDI_FAILURE;
2559         return (rval);
2560     }
2561     page_address = va_arg(ap, uint32_t);
2562     /*
2563     * The INVALID_PAGE status is normal if using GET_NEXT_HANDLE and there
2564     * are no more pages.  If everything is OK up to this point but the
2565     * status is INVALID_PAGE, change rval to FAILURE and quit.  Also,
2566     * signal that device traversal is complete.
2567     */
2568     if (iocstatus == MPI2_IOCSTATUS_CONFIG_INVALID_PAGE) {
2569         if ((page_address & MPI2_SAS_EXPAND_PGAD_FORM_MASK) ==

```

```

2570         MPI2_SAS_EXPAND_PGAD_FORM_GET_NEXT_HNDL) {
2571             mpt->m_done_traverse_smp = 1;
2572         }
2573         rval = DDI_FAILURE;
2574         return (rval);
2575     }

2577     invalid_dword_count = va_arg(ap, uint32_t *);
2578     running_disparity_error_count = va_arg(ap, uint32_t *);
2579     loss_of_dword_sync_count = va_arg(ap, uint32_t *);
2580     phy_reset_problem_count = va_arg(ap, uint32_t *);

2582     sasphy_page = (pMpi2SasPhyPage1_t)page_memp;

2584     *invalid_dword_count =
2585         ddi_get32(accessp, &sasphy_page->InvalidDwordCount);
2586     *running_disparity_error_count =
2587         ddi_get32(accessp, &sasphy_page->RunningDisparityErrorCount);
2588     *loss_of_dword_sync_count =
2589         ddi_get32(accessp, &sasphy_page->LossDwordSynchCount);
2590     *phy_reset_problem_count =
2591         ddi_get32(accessp, &sasphy_page->PhyResetProblemCount);

2593     return (rval);
2594 }

2596 /*
2597  * Request MPI configuration page SAS phy page 0 to get DevHandle, phymask
2598  * and SAS address.
2599  */
2600 int
2601 mptsas_get_sas_phy_page0(mptsas_t *mpt, uint32_t page_address,
2602     smhba_info_t *info)
2603 {
2604     int                rval = DDI_SUCCESS;

2606     ASSERT(mutex_owned(&mpt->m_mutex));

2608     /*
2609     * Get the header and config page.  reply contains the reply frame,
2610     * which holds status info for the request.
2611     */
2612     rval = mptsas_access_config_page(mpt,
2613         MPI2_CONFIG_ACTION_PAGE_READ_CURRENT,
2614         MPI2_CONFIG_EXTPAGETYPE_SAS_PHY, 1, page_address,
2615         mptsas_sasphy_page_1_cb, page_address,
2616         &info->invalid_dword_count,
2617         &info->running_disparity_error_count,
2618         &info->loss_of_dword_sync_count,
2619         &info->phy_reset_problem_count);

2621     return (rval);
2622 }
2623 /*
2624  * mptsas_get_manufacture_page0
2625  *
2626  * This function will retrieve the base
2627  * Chip name, Board Name, Board Trace number from the adapter.
2628  * Since this function is only called during the
2629  * initialization process, use handshaking.
2630  */
2631 int
2632 mptsas_get_manufacture_page0(mptsas_t *mpt)
2633 {
2634     ddi_dma_attr_t        recvd_dma_attrs, page_dma_attrs;
2635     ddi_dma_cookie_t      page_cookie;

```

```

2636     ddi_dma_handle_t      recv_dma_handle, page_dma_handle;
2637     ddi_acc_handle_t      recv_accessp, page_accessp;
2638     pMpi2ConfigReply_t    configreply;
2639     caddr_t               recv_memp, page_memp;
2640     int                   recv_numbytes;
2641     pMpi2ManufacturingPage0_t m0;
2642     uint32_t              flagslength;
2643     int                   rval = DDI_SUCCESS;
2644     uint_t                iocstatus;
2645     uint8_t               i = 0;
2646     boolean_t             free_recv = B_FALSE, free_page = B_FALSE;

2648     MPTSAS_DISABLE_INTR(mpt);

2650     if (mptsas_send_config_request_msg(mpt, MPI2_CONFIG_ACTION_PAGE_HEADER,
2651         MPI2_CONFIG_PAGETYPE_MANUFACTURING, 0, 0, 0, 0, 0)) {
2652         rval = DDI_FAILURE;
2653         goto done;
2654     }

2656     /*
2657      * dynamically create a customized dma attribute structure
2658      * that describes the MPT's config reply page request structure.
2659      */
2660     recv_dma_attr = mpt->m_msg_dma_attr;
2661     recv_dma_attr.dma_attr_sgllen = 1;
2662     recv_dma_attr.dma_attr_granular = (sizeof (MPI2_CONFIG_REPLY));

2664     if (mptsas_dma_addr_create(mpt, recv_dma_attr, &recv_dma_handle,
2665         &recv_accessp, &recv_memp, (sizeof (MPI2_CONFIG_REPLY)),
2666         NULL) == FALSE) {
2667         rval = DDI_FAILURE;
2668         goto done;
2669     }
2670     /* Now safe to call mptsas_dma_addr_destroy(recv_dma_handle). */
2671     free_recv = B_TRUE;

2673     bzero(recv_memp, sizeof (MPI2_CONFIG_REPLY));
2674     configreply = (pMpi2ConfigReply_t)recv_memp;
2675     recv_numbytes = sizeof (MPI2_CONFIG_REPLY);

2677     /*
2678      * get config reply message
2679      */
2680     if (mptsas_get_handshake_msg(mpt, recv_memp, recv_numbytes,
2681         &recv_accessp)) {
2682         rval = DDI_FAILURE;
2683         goto done;
2684     }

2686     if (iocstatus = ddi_get16(recv_accessp, &configreply->IOCStatus)) {
2687         mptsas_log(mpt, CE_WARN, "mptsas get manufacture page5 update: "
2688             "IOCStatus=0x%x, IOCLogInfo=0x%x", iocstatus,
2689             ddi_get32(recv_accessp, &configreply->IOCLogInfo));
2690         goto done;
2691     }

2693     /*
2694      * dynamically create a customized dma attribute structure
2695      * that describes the MPT's config page structure.
2696      */
2697     page_dma_attr = mpt->m_msg_dma_attr;
2698     page_dma_attr.dma_attr_sgllen = 1;
2699     page_dma_attr.dma_attr_granular = (sizeof (MPI2_CONFIG_PAGE_MAN_0));

2701     if (mptsas_dma_addr_create(mpt, page_dma_attr, &page_dma_handle,

```

```

2702         &page_accessp, &page_memp, (sizeof (MPI2_CONFIG_PAGE_MAN_0)),
2703         &page_cookie) == FALSE) {
2704         rval = DDI_FAILURE;
2705         goto done;
2706     }
2707     /* Now safe to call mptsas_dma_addr_destroy(page_dma_handle). */
2708     free_page = B_TRUE;

2710     bzero(page_memp, sizeof (MPI2_CONFIG_PAGE_MAN_0));
2711     m0 = (pMpi2ManufacturingPage0_t)page_memp;

2713     /*
2714      * Give reply address to IOC to store config page in and send
2715      * config request out.
2716      */

2718     flagslength = sizeof (MPI2_CONFIG_PAGE_MAN_0);
2719     flagslength |= ((uint32_t)(MPI2_SGE_FLAGS_LAST_ELEMENT |
2720         MPI2_SGE_FLAGS_END_OF_BUFFER | MPI2_SGE_FLAGS_SIMPLE_ELEMENT |
2721         MPI2_SGE_FLAGS_SYSTEM_ADDRESS | MPI2_SGE_FLAGS_64_BIT_ADDRESSING |
2722         MPI2_SGE_FLAGS_IOC_TO_HOST |
2723         MPI2_SGE_FLAGS_END_OF_LIST) << MPI2_SGE_FLAGS_SHIFT);

2725     if (mptsas_send_config_request_msg(mpt,
2726         MPI2_CONFIG_ACTION_PAGE_READ_CURRENT,
2727         MPI2_CONFIG_PAGETYPE_MANUFACTURING, 0, 0,
2728         ddi_get8(recv_accessp, &configreply->Header.PageVersion),
2729         ddi_get8(recv_accessp, &configreply->Header.PageLength),
2730         flagslength, page_cookie.dmac_laddress)) {
2731         rval = DDI_FAILURE;
2732         goto done;
2733     }

2735     /*
2736      * get reply view handshake
2737      */
2738     if (mptsas_get_handshake_msg(mpt, recv_memp, recv_numbytes,
2739         &recv_accessp)) {
2740         rval = DDI_FAILURE;
2741         goto done;
2742     }

2744     if (iocstatus = ddi_get16(recv_accessp, &configreply->IOCStatus)) {
2745         mptsas_log(mpt, CE_WARN, "mptsas get manufacture page0 config: "
2746             "IOCStatus=0x%x, IOCLogInfo=0x%x", iocstatus,
2747             ddi_get32(recv_accessp, &configreply->IOCLogInfo));
2748         goto done;
2749     }

2751     (void) ddi_dma_sync(page_dma_handle, 0, 0, DDI_DMA_SYNC_FORCPU);

2753     /*
2754      * Fusion-MPT stores fields in little-endian format. This is
2755      * why the low-order 32 bits are stored first.
2756      */

2758     for (i = 0; i < 16; i++) {
2759         mpt->m_MANU_page0.ChipName[i] =
2760             ddi_get8(page_accessp,
2761                 (uint8_t *) (void *) &m0->ChipName[i]);
2762     }

2764     for (i = 0; i < 8; i++) {
2765         mpt->m_MANU_page0.ChipRevision[i] =
2766             ddi_get8(page_accessp,
2767                 (uint8_t *) (void *) &m0->ChipRevision[i]);

```

```
2768     }
2770     for (i = 0; i < 16; i++) {
2771         mpt->m_MANU_page0.BoardName[i] =
2772             ddi_get8(page_accesssp,
2773                 (uint8_t *) (void *) &m0->BoardName[i]);
2774     }
2776     for (i = 0; i < 16; i++) {
2777         mpt->m_MANU_page0.BoardAssembly[i] =
2778             ddi_get8(page_accesssp,
2779                 (uint8_t *) (void *) &m0->BoardAssembly[i]);
2780     }
2782     for (i = 0; i < 16; i++) {
2783         mpt->m_MANU_page0.BoardTracerNumber[i] =
2784             ddi_get8(page_accesssp,
2785                 (uint8_t *) (void *) &m0->BoardTracerNumber[i]);
2786     }
2788     if ((mptsas_check_dma_handle(recv_dma_handle) != DDI_SUCCESS) ||
2789         (mptsas_check_dma_handle(page_dma_handle) != DDI_SUCCESS)) {
2790         ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_UNAFFECTED);
2791         rval = DDI_FAILURE;
2792         goto done;
2793     }
2794     if ((mptsas_check_acc_handle(recv_accesssp) != DDI_SUCCESS) ||
2795         (mptsas_check_acc_handle(page_accesssp) != DDI_SUCCESS)) {
2796         ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_UNAFFECTED);
2797         rval = DDI_FAILURE;
2798     }
2799 done:
2800     /*
2801      * free up memory
2802      */
2803     if (free_recv)
2804         mptsas_dma_addr_destroy(&recv_dma_handle, &recv_accesssp);
2805     if (free_page)
2806         mptsas_dma_addr_destroy(&page_dma_handle, &page_accesssp);
2807     MPTSAS_ENABLE_INTR(mpt);
2809     return (rval);
2810 }
2811 #endif /* ! codereview */
```

new/usr/src/uts/common/io/scsi/adapters/mpt_sas3/mptsas3_init.c

1

```
*****
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
*****

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

22 /*
23  * Copyright 2009 Sun Microsystems, Inc. All rights reserved.
24  * Use is subject to license terms.
25  * Copyright (c) 2014, Tegile Systems Inc. All rights reserved.
26  */

28 /*
29  * Copyright (c) 2000 to 2009, LSI Corporation.
30  * All rights reserved.
31  *
32  * Redistribution and use in source and binary forms of all code within
33  * this file that is exclusively owned by LSI, with or without
34  * modification, is permitted provided that, in addition to the CDDL 1.0
35  * License requirements, the following conditions are met:
36  *
37  *   Neither the name of the author nor the names of its contributors may be
38  *   used to endorse or promote products derived from this software without
39  *   specific prior written permission.
40  *
41  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
42  * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
43  * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
44  * FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
45  * COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
46  * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
47  * BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
48  * OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
49  * AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
50  * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
51  * OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
52  * DAMAGE.
53 */

55 /*
56  * mptsas_init - This file contains all the functions used to initialize
57  * MPT2.0 based hardware.
58  */

60 #if defined(lint) || defined(DEBUG)
61 #define MPTSAS_DEBUG
```

new/usr/src/uts/common/io/scsi/adapters/mpt_sas3/mptsas3_init.c

2

```
62 #endif

64 /*
65  * standard header files
66  */
67 #include <sys/note.h>
68 #include <sys/scsi/scsi.h>

70 #pragma pack(1)
71 #include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_type.h>
72 #include <sys/scsi/adapters/mpt_sas3/mpi/mpi2.h>
73 #include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_cnfg.h>
74 #include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_init.h>
75 #include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_ioc.h>
76 #include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_tool.h>
77 #pragma pack()
78 /*
79  * private header files.
80  */
81 #include <sys/scsi/adapters/mpt_sas3/mptsas3_var.h>

83 static int mptsas_ioc_do_get_facts(mptsas_t *mpt, caddr_t memp, int var,
84 ddi_acc_handle_t accessp);
85 static int mptsas_ioc_do_get_facts_reply(mptsas_t *mpt, caddr_t memp, int var,
86 ddi_acc_handle_t accessp);
87 static int mptsas_ioc_do_get_port_facts(mptsas_t *mpt, caddr_t memp, int var,
88 ddi_acc_handle_t accessp);
89 static int mptsas_ioc_do_get_port_facts_reply(mptsas_t *mpt, caddr_t memp,
90 int var, ddi_acc_handle_t accessp);
91 static int mptsas_ioc_do_enable_port(mptsas_t *mpt, caddr_t memp, int var,
92 ddi_acc_handle_t accessp);
93 static int mptsas_ioc_do_enable_port_reply(mptsas_t *mpt, caddr_t memp, int var,
94 ddi_acc_handle_t accessp);
95 static int mptsas_ioc_do_enable_event_notification(mptsas_t *mpt, caddr_t memp,
96 int var, ddi_acc_handle_t accessp);
97 static int mptsas_ioc_do_enable_event_notification_reply(mptsas_t *mpt,
98 caddr_t memp, int var, ddi_acc_handle_t accessp);
99 static int mptsas_do_ioc_init(mptsas_t *mpt, caddr_t memp, int var,
100 ddi_acc_handle_t accessp);
101 static int mptsas_do_ioc_init_reply(mptsas_t *mpt, caddr_t memp, int var,
102 ddi_acc_handle_t accessp);

104 static const char *
105 mptsas_devid_type_string(mptsas_t *mpt)
106 {
107     switch (mpt->m_devid) {
108     case MPI2_MFGPAGE_DEVID_SAS2008:
109         return ("SAS2008");
110     case MPI2_MFGPAGE_DEVID_SAS2004:
111         return ("SAS2004");
112     case MPI2_MFGPAGE_DEVID_SAS2108_1:
113     case MPI2_MFGPAGE_DEVID_SAS2108_2:
114     case MPI2_MFGPAGE_DEVID_SAS2108_3:
115         return ("SAS2108");
116     case MPI2_MFGPAGE_DEVID_SAS2116_1:
117     case MPI2_MFGPAGE_DEVID_SAS2116_2:
118         return ("SAS2116");
119     case MPI2_MFGPAGE_DEVID_SAS2208_1:
120     case MPI2_MFGPAGE_DEVID_SAS2208_2:
121     case MPI2_MFGPAGE_DEVID_SAS2208_3:
122     case MPI2_MFGPAGE_DEVID_SAS2208_4:
123     case MPI2_MFGPAGE_DEVID_SAS2208_5:
124     case MPI2_MFGPAGE_DEVID_SAS2208_6:
125     }
126     return (0);
127 }

/* These are the same as the next 2?? */
case MPI2_MFGPAGE_DEVID_SAS2208_7:
```

```

128     case MPI2_MFGPAGE_DEVID_SAS2208_8:
129 #endif
130         return ("SAS2208");
131     case MPI2_MFGPAGE_DEVID_SAS2308_1:
132     case MPI2_MFGPAGE_DEVID_SAS2308_2:
133     case MPI2_MFGPAGE_DEVID_SAS2308_3:
134         return ("SAS2308");
135     case MPI25_MFGPAGE_DEVID_SAS3004:
136         return ("SAS3004");
137     case MPI25_MFGPAGE_DEVID_SAS3008:
138         return ("SAS3008");
139     case MPI25_MFGPAGE_DEVID_SAS3108_1:
140     case MPI25_MFGPAGE_DEVID_SAS3108_2:
141     case MPI25_MFGPAGE_DEVID_SAS3108_3:
142     case MPI25_MFGPAGE_DEVID_SAS3108_4:
143     case MPI25_MFGPAGE_DEVID_SAS3108_5:
144     case MPI25_MFGPAGE_DEVID_SAS3108_6:
145         return ("SAS3108");
146     default:
147         return ("?");
148     }
149 }

151 int
152 mptsas_ioc_get_facts(mptsas_t *mpt)
153 {
154     /*
155      * Send get facts messages
156      */
157     if (mptsas_do_dma(mpt, sizeof (MPI2_IOC_FACTS_REQUEST), NULL,
158         mptsas_ioc_do_get_facts)) {
159         return (DDI_FAILURE);
160     }

162     /*
163      * Get facts reply messages
164      */
165     if (mptsas_do_dma(mpt, sizeof (MPI2_IOC_FACTS_REPLY), NULL,
166         mptsas_ioc_do_get_facts_reply)) {
167         return (DDI_FAILURE);
168     }

170     return (DDI_SUCCESS);
171 }

173 static int
174 mptsas_ioc_do_get_facts(mptsas_t *mpt, caddr_t memp, int var,
175     ddi_acc_handle_t accessp)
176 {
177 #ifndef __lock_lint
178     _NOTE(ARGUNUSED(var))
179 #endif
180     pMpi2IOCFactsRequest_t facts;
181     int numbytes;

183     bzero(memp, sizeof (*facts));
184     facts = (void *)memp;
185     ddi_put8(accessp, &facts->Function, MPI2_FUNCTION_IOC_FACTS);
186     numbytes = sizeof (*facts);

188     /*
189      * Post message via handshake
190      */
191     if (mptsas_send_handshake_msg(mpt, memp, numbytes, accessp)) {
192         return (DDI_FAILURE);
193     }

```

```

195     return (DDI_SUCCESS);
196 }

198 static int
199 mptsas_ioc_do_get_facts_reply(mptsas_t *mpt, caddr_t memp, int var,
200     ddi_acc_handle_t accessp)
201 {
202 #ifndef __lock_lint
203     _NOTE(ARGUNUSED(var))
204 #endif

206     pMpi2IOCFactsReply_t factsreply;
207     int numbytes;
208     uint_t iocstatus;
209     char buf[32];
210     uint16_t numReplyFrames;
211     uint16_t queueSize, queueDiff;
212     int simple_sge_main;
213     int simple_sge_next;
214     uint32_t capabilities;
215     uint16_t msgversion;

217     bzero(memp, sizeof (*factsreply));
218     factsreply = (void *)memp;
219     numbytes = sizeof (*factsreply);

221     /*
222      * get ioc facts reply message
223      */
224     if (mptsas_get_handshake_msg(mpt, memp, numbytes, accessp)) {
225         return (DDI_FAILURE);
226     }

228     if (iocstatus = ddi_get16(accessp, &factsreply->IOCStatus)) {
229         mptsas_log(mpt, CE_WARN, "mptsas_ioc_do_get_facts_reply: "
230             "IOCStatus=0x%x, IOCLogInfo=0x%x", iocstatus,
231             ddi_get32(accessp, &factsreply->IOCLogInfo));
232         return (DDI_FAILURE);
233     }

235     /*
236      * store key values from reply to mpt structure
237      */
238     mpt->m_fwversion = ddi_get32(accessp, &factsreply->FWVersion.Word);
239     mpt->m_productid = ddi_get16(accessp, &factsreply->ProductID);

242     (void) sprintf(buf, "%u.%u.%u.%u",
243         ddi_get8(accessp, &factsreply->FWVersion.Struct.Major),
244         ddi_get8(accessp, &factsreply->FWVersion.Struct.Minor),
245         ddi_get8(accessp, &factsreply->FWVersion.Struct.Unit),
246         ddi_get8(accessp, &factsreply->FWVersion.Struct.Dev));
247     mptsas_log(mpt, CE_NOTE, "?MPT Firmware version v%s (%s)\n",
248         buf, mptsas_devid_type_string(mpt));
249     (void) ddi_prop_update_string(DDI_DEV_T_NONE, mpt->m_dip,
250         "firmware-version", buf);

252     /*
253      * Set up request info.
254      */
255     mpt->m_max_requests = ddi_get16(accessp,
256         &factsreply->RequestCredit) - 1;
257     mpt->m_req_frame_size = ddi_get16(accessp,
258         &factsreply->IOCRequestFrameSize) * 4;

```

```

260 /*
261  * Size of reply free queue should be the number of requests
262  * plus some additional for events (32). Make sure number of
263  * reply frames is not a multiple of 16 so that the queue sizes
264  * are calculated correctly later to be a multiple of 16.
265  */
266 mpt->m_reply_frame_size = ddi_get8(accessp,
267     &factsreply->ReplyFrameSize) * 4;
268 numReplyFrames = mpt->m_max_requests + 32;
269 if (!(numReplyFrames % 16)) {
270     numReplyFrames--;
271 }
272 mpt->m_max_replies = numReplyFrames;
273 queueSize = numReplyFrames;
274 queueSize += 16 - (queueSize % 16);
275 mpt->m_free_queue_depth = queueSize;
276
277 /*
278  * Size of reply descriptor post queue should be the number of
279  * request frames + the number of reply frames + 1 and needs to
280  * be a multiple of 16. This size can be no larger than
281  * MaxReplyDescriptorPostQueueDepth from IOCFacts. If the
282  * calculated queue size is larger than allowed, subtract a
283  * multiple of 16 from m_max_requests, m_max_replies, and
284  * m_reply_free_depth.
285  *
286  * There is no indication in the spec that you can reduce the
287  * queue size if you have many.
288  */
289 queueSize = mpt->m_max_requests + numReplyFrames + 1;
290 if (queueSize % 16) {
291     queueSize += 16 - (queueSize % 16);
292 }
293 mpt->m_post_queue_depth = ddi_get16(accessp,
294     &factsreply->MaxReplyDescriptorPostQueueDepth);
295 if (queueSize > mpt->m_post_queue_depth) {
296     queueDiff = queueSize - mpt->m_post_queue_depth;
297     if (queueDiff % 16) {
298         queueDiff += 16 - (queueDiff % 16);
299     }
300     mpt->m_max_requests -= queueDiff;
301     mpt->m_max_replies -= queueDiff;
302     mpt->m_free_queue_depth -= queueDiff;
303     queueSize -= queueDiff;
304 }
305 mpt->m_post_queue_depth = queueSize;
306
307 /*
308  * Set up max chain depth.
309  */
310 mpt->m_max_chain_depth = ddi_get8(accessp,
311     &factsreply->MaxChainDepth);
312 mpt->m_ioc_capabilities = ddi_get32(accessp,
313     &factsreply->IOCCapabilities);
314 if (mpt->m_ioc_capabilities & MPI2_IOCFacts_CAPABILITY_MSI_X_INDEX) {
315     mpt->m_max_msix_vectors = ddi_get8(accessp,
316         &factsreply->MaxMSIXVectors);
317 }
318
319 /*
320  * Set flag to check for SAS3 support.
321  */
322 msgversion = ddi_get16(accessp, &factsreply->MsgVersion);
323 if (msgversion == MPI2_VERSION_02_05) {
324     mptsas_log(mpt, CE_NOTE, "?mpt_sas3%d SAS 3 Supported\n",
325         mpt->m_instance);

```

```

326     mpt->m_MPI25 = TRUE;
327 } else {
328     mptsas_log(mpt, CE_NOTE, "?mpt_sas3%d MPI Version 0x%x\n",
329         mpt->m_instance, msgversion);
330 }
331
332 /*
333  * Calculate max frames per request based on DMA S/G length.
334  */
335 simple_sge_main = MPTSAS_MAX_FRAME_SGES64(mpt) - 1;
336 simple_sge_next = mpt->m_req_frame_size /
337     (mpt->m_MPI25 ? sizeof (MPI2_IEEE_SGE_SIMPLE64) :
338         sizeof (MPI2_SGE_SIMPLE64)) - 1;
339
340 mpt->m_max_request_frames = (MPTSAS_MAX_DMA_SEGS -
341     simple_sge_main) / simple_sge_next + 1;
342 if (((MPTSAS_MAX_DMA_SEGS - simple_sge_main) %
343     simple_sge_next) > 1) {
344     mpt->m_max_request_frames++;
345 }
346
347 /*
348  * Check if controller supports FW diag buffers and set flag to enable
349  * each type.
350  */
351 capabilities = mpt->m_ioc_capabilities;
352 if (capabilities & MPI2_IOCFacts_CAPABILITY_DIAG_TRACE_BUFFER) {
353     mpt->m_fw_diag_buffer_list[MPI2_DIAG_BUF_TYPE_TRACE].enabled =
354         TRUE;
355 }
356 if (capabilities & MPI2_IOCFacts_CAPABILITY_SNAPSHOT_BUFFER) {
357     mpt->m_fw_diag_buffer_list[MPI2_DIAG_BUF_TYPE_SNAPSHOT].
358         enabled = TRUE;
359 }
360 if (capabilities & MPI2_IOCFacts_CAPABILITY_EXTENDED_BUFFER) {
361     mpt->m_fw_diag_buffer_list[MPI2_DIAG_BUF_TYPE_EXTENDED].
362         enabled = TRUE;
363 }
364
365 /*
366  * Check if controller supports replaying events when issuing Message
367  * Unit Reset and set flag to enable MUR.
368  */
369 if (capabilities & MPI2_IOCFacts_CAPABILITY_EVENT_REPLAY) {
370     mpt->m_event_replay = TRUE;
371 }
372
373 /*
374  * Check if controller supports IR.
375  */
376 if (capabilities & MPI2_IOCFacts_CAPABILITY_INTEGRATED_RAID) {
377     mpt->m_ir_capable = TRUE;
378 }
379
380 return (DDI_SUCCESS);
381 }
382
383 int
384 mptsas_ioc_get_port_facts(mptsas_t *mpt, int port)
385 {
386     /*
387      * Send get port facts message
388      */
389     if (mptsas_do_dma(mpt, sizeof (MPI2_PORT_FACTS_REQUEST), port,
390         mptsas_ioc_do_get_port_facts)) {
391         return (DDI_FAILURE);

```

```

392     }
393
394     /*
395     * Get port facts reply message
396     */
397     if (mptsas_do_dma(mpt, sizeof (MPI2_PORT_FACTS_REPLY), port,
398         mptsas_ioc_do_get_port_facts_reply)) {
399         return (DDI_FAILURE);
400     }
401
402     return (DDI_SUCCESS);
403 }
404
405 static int
406 mptsas_ioc_do_get_port_facts(mptsas_t *mpt, caddr_t memp, int var,
407     ddi_acc_handle_t accessp)
408 {
409     pMpi2PortFactsRequest_t facts;
410     int numbytes;
411
412     bzero(memp, sizeof (*facts));
413     facts = (void *)memp;
414     ddi_put8(accessp, &facts->Function, MPI2_FUNCTION_PORT_FACTS);
415     ddi_put8(accessp, &facts->PortNumber, var);
416     numbytes = sizeof (*facts);
417
418     /*
419     * Send port facts message via handshake
420     */
421     if (mptsas_send_handshake_msg(mpt, memp, numbytes, accessp)) {
422         return (DDI_FAILURE);
423     }
424
425     return (DDI_SUCCESS);
426 }
427
428 static int
429 mptsas_ioc_do_get_port_facts_reply(mptsas_t *mpt, caddr_t memp, int var,
430     ddi_acc_handle_t accessp)
431 {
432     #ifndef __lock_lint
433     _NOTE(ARGUNUSED(var))
434     #endif
435     pMpi2PortFactsReply_t factsreply;
436     int numbytes;
437     uint_t iocstatus;
438
439     bzero(memp, sizeof (*factsreply));
440     factsreply = (void *)memp;
441     numbytes = sizeof (*factsreply);
442
443     /*
444     * Get port facts reply message via handshake
445     */
446     if (mptsas_get_handshake_msg(mpt, memp, numbytes, accessp)) {
447         return (DDI_FAILURE);
448     }
449
450     if (iocstatus = ddi_get16(accessp, &factsreply->IOCStatus)) {
451         mptsas_log(mpt, CE_WARN, "mptsas_ioc_do_get_port_facts_reply: "
452             "IOCStatus=0x%x, IOCLogInfo=0x%x", iocstatus,
453             ddi_get32(accessp, &factsreply->IOCLogInfo));
454         return (DDI_FAILURE);
455     }
456
457     return (DDI_SUCCESS);

```

```

458 }
459
460 int
461 mptsas_ioc_enable_port(mptsas_t *mpt)
462 {
463     /*
464     * Send enable port message
465     */
466     if (mptsas_do_dma(mpt, sizeof (MPI2_PORT_ENABLE_REQUEST), 0,
467         mptsas_ioc_do_enable_port)) {
468         return (DDI_FAILURE);
469     }
470
471     /*
472     * Get enable port reply message
473     */
474     if (mptsas_do_dma(mpt, sizeof (MPI2_PORT_ENABLE_REPLY), 0,
475         mptsas_ioc_do_enable_port_reply)) {
476         return (DDI_FAILURE);
477     }
478
479     return (DDI_SUCCESS);
480 }
481
482 static int
483 mptsas_ioc_do_enable_port(mptsas_t *mpt, caddr_t memp, int var,
484     ddi_acc_handle_t accessp)
485 {
486     #ifndef __lock_lint
487     _NOTE(ARGUNUSED(var))
488     #endif
489     pMpi2PortEnableRequest_t enable;
490     int numbytes;
491
492     bzero(memp, sizeof (*enable));
493     enable = (void *)memp;
494     ddi_put8(accessp, &enable->Function, MPI2_FUNCTION_PORT_ENABLE);
495     numbytes = sizeof (*enable);
496
497     /*
498     * Send message via handshake
499     */
500     if (mptsas_send_handshake_msg(mpt, memp, numbytes, accessp)) {
501         return (DDI_FAILURE);
502     }
503
504     return (DDI_SUCCESS);
505 }
506
507 static int
508 mptsas_ioc_do_enable_port_reply(mptsas_t *mpt, caddr_t memp, int var,
509     ddi_acc_handle_t accessp)
510 {
511     #ifndef __lock_lint
512     _NOTE(ARGUNUSED(var))
513     #endif
514
515     int numbytes;
516     uint_t iocstatus;
517     pMpi2PortEnableReply_t portreply;
518
519     numbytes = sizeof (MPI2_PORT_ENABLE_REPLY);
520     bzero(memp, numbytes);
521     portreply = (void *)memp;
522
523     /*

```



```

524     /* Get message via handshake
525     */
526     if (mptsas_get_handshake_msg(mpt, memp, numbytes, accessp)) {
527         return (DDI_FAILURE);
528     }

530     if (iocstatus = ddi_getl6(accessp, &portreply->IOCStatus)) {
531         mptsas_log(mpt, CE_WARN, "mptsas_ioc_do_enable_port_reply: "
532             "IOCStatus=0x%x, IOCLogInfo=0x%x", iocstatus,
533             ddi_get32(accessp, &portreply->IOCLogInfo));
534         return (DDI_FAILURE);
535     }

537     return (DDI_SUCCESS);
538 }

540 int
541 mptsas_ioc_enable_event_notification(mptsas_t *mpt)
542 {
543     ASSERT(mutex_owned(&mpt->m_mutex));

545     /*
546     * Send enable event notification message
547     */
548     if (mptsas_do_dma(mpt, sizeof (MPI2_EVENT_NOTIFICATION_REQUEST), NULL,
549         mptsas_ioc_do_enable_event_notification)) {
550         return (DDI_FAILURE);
551     }

553     /*
554     * Get enable event reply message
555     */
556     if (mptsas_do_dma(mpt, sizeof (MPI2_EVENT_NOTIFICATION_REPLY), NULL,
557         mptsas_ioc_do_enable_event_notification_reply)) {
558         return (DDI_FAILURE);
559     }

561     return (DDI_SUCCESS);
562 }

564 static int
565 mptsas_ioc_do_enable_event_notification(mptsas_t *mpt, caddr_t memp, int var,
566     ddi_acc_handle_t accessp)
567 {
568     #ifndef __lock_lint
569     _NOTE(ARGUNUSED(var))
570     #endif

572     pMpi2EventNotificationRequest_t event;
573     int numbytes;

575     bzero(memp, sizeof (*event));
576     event = (void *)memp;
577     ddi_put8(accessp, &event->Function, MPI2_FUNCTION_EVENT_NOTIFICATION);
578     numbytes = sizeof (*event);

580     /*
581     * Send message via handshake
582     */
583     if (mptsas_send_handshake_msg(mpt, memp, numbytes, accessp)) {
584         return (DDI_FAILURE);
585     }

587     return (DDI_SUCCESS);
588 }

```

```

590 static int
591 mptsas_ioc_do_enable_event_notification_reply(mptsas_t *mpt, caddr_t memp,
592     int var, ddi_acc_handle_t accessp)
593 {
594     #ifndef __lock_lint
595     _NOTE(ARGUNUSED(var))
596     #endif

597     int numbytes;
598     uint_t iocstatus;
599     pMpi2EventNotificationReply_t eventsreply;

601     numbytes = sizeof (MPI2_EVENT_NOTIFICATION_REPLY);
602     bzero(memp, numbytes);
603     eventsreply = (void *)memp;

605     /*
606     * Get message via handshake
607     */
608     if (mptsas_get_handshake_msg(mpt, memp, numbytes, accessp)) {
609         return (DDI_FAILURE);
610     }

612     if (iocstatus = ddi_getl6(accessp, &eventsreply->IOCStatus)) {
613         mptsas_log(mpt, CE_WARN,
614             "mptsas_ioc_do_enable_event_notification_reply: "
615             "IOCStatus=0x%x, IOCLogInfo=0x%x", iocstatus,
616             ddi_get32(accessp, &eventsreply->IOCLogInfo));
617         return (DDI_FAILURE);
618     }

620     return (DDI_SUCCESS);
621 }

623 int
624 mptsas_ioc_init(mptsas_t *mpt)
625 {
626     /*
627     * Send ioc init message
628     */
629     if (mptsas_do_dma(mpt, sizeof (MPI2_IOC_INIT_REQUEST), NULL,
630         mptsas_do_ioc_init)) {
631         return (DDI_FAILURE);
632     }

634     /*
635     * Get ioc init reply message
636     */
637     if (mptsas_do_dma(mpt, sizeof (MPI2_IOC_INIT_REPLY), NULL,
638         mptsas_do_ioc_init_reply)) {
639         return (DDI_FAILURE);
640     }

642     return (DDI_SUCCESS);
643 }

645 static int
646 mptsas_do_ioc_init(mptsas_t *mpt, caddr_t memp, int var,
647     ddi_acc_handle_t accessp)
648 {
649     #ifndef __lock_lint
650     _NOTE(ARGUNUSED(var))
651     #endif

653     pMpi2IOCInitRequest_t init;
654     int numbytes;
655     timespec_t time;

```

```

656     uint64_t          mSec;

658     bzero(memp, sizeof (*init));
659     init = (void *)memp;
660     ddi_put8(accessp, &init->Function, MPI2_FUNCTION_IOC_INIT);
661     ddi_put8(accessp, &init->WhoInit, MPI2_WHOWINIT_HOST_DRIVER);
662     ddi_put16(accessp, &init->MsgVersion, MPI2_VERSION);
663     ddi_put16(accessp, &init->HeaderVersion, MPI2_HEADER_VERSION);
664     if (mpt->m_intr_type == DDI_INTR_TYPE_MSIX) {
665         ddi_put8(accessp, &init->HostMSixVectors, mpt->m_intr_cnt);
666     }
667     ddi_put16(accessp, &init->SystemRequestFrameSize,
668         mpt->m_req_frame_size / 4);
669     ddi_put16(accessp, &init->ReplyDescriptorPostQueueDepth,
670         mpt->m_post_queue_depth);
671     ddi_put16(accessp, &init->ReplyFreeQueueDepth,
672         mpt->m_free_queue_depth);

674     /*
675      * These addresses are set using the DMA cookie addresses from when the
676      * memory was allocated. Sense buffer hi address should be 0.
677      */
678     ddi_put32(accessp, &init->SenseBufferAddressHigh,
679         (uint32_t)(mpt->m_req_sense_dma_addr >> 32));
680     ddi_put32(accessp, &init->SystemReplyAddressHigh,
681         (uint32_t)(mpt->m_reply_frame_dma_addr >> 32));
682     ddi_put32(accessp, &init->SystemRequestFrameBaseAddress.High,
683         (uint32_t)(mpt->m_req_frame_dma_addr >> 32));
684     ddi_put32(accessp, &init->SystemRequestFrameBaseAddress.Low,
685         (uint32_t)mpt->m_req_frame_dma_addr);
686     ddi_put32(accessp, &init->ReplyDescriptorPostQueueAddress.High,
687         (uint32_t)(mpt->m_post_queue_dma_addr >> 32));
688     ddi_put32(accessp, &init->ReplyDescriptorPostQueueAddress.Low,
689         (uint32_t)mpt->m_post_queue_dma_addr);
690     ddi_put32(accessp, &init->ReplyFreeQueueAddress.High,
691         (uint32_t)(mpt->m_free_queue_dma_addr >> 32));
692     ddi_put32(accessp, &init->ReplyFreeQueueAddress.Low,
693         (uint32_t)mpt->m_free_queue_dma_addr);

695     /*
696      * Fill in the timestamp with the number of milliseconds since midnight
697      * of January 1, 1970 UT (Greenwich Mean Time). Time is returned in
698      * seconds and nanoseconds. Translate both to milliseconds and add
699      * them together to get total milliseconds.
700      */
701     getthrestime(&time);
702     mSec = time.tv_sec * MILLISEC;
703     mSec += (time.tv_nsec / MICROSEC);
704     ddi_put32(accessp, &init->TimeStamp.High, (uint32_t)(mSec >> 32));
705     ddi_put32(accessp, &init->TimeStamp.Low, (uint32_t)mSec);

707     numbytes = sizeof (*init);

709     /*
710      * Post message via handshake
711      */
712     if (mptsas_send_handshake_msg(mpt, memp, numbytes, accessp)) {
713         return (DDI_FAILURE);
714     }

716     return (DDI_SUCCESS);
717 }

719 static int
720 mptsas_do_ioc_init_reply(mptsas_t *mpt, caddr_t memp, int var,
721     ddi_acc_handle_t accessp)

```

```

722 {
723     #ifndef __lock_lint
724         _NOTE(ARGUNUSED(var))
725     #endif

727     pMpi2IOCInitReply_t    initreply;
728     int                    numbytes;
729     uint_t                 iocstatus;

731     numbytes = sizeof (MPI2_IOC_INIT_REPLY);
732     bzero(memp, numbytes);
733     initreply = (void *)memp;

735     /*
736      * Get reply message via handshake
737      */
738     if (mptsas_get_handshake_msg(mpt, memp, numbytes, accessp)) {
739         return (DDI_FAILURE);
740     }

742     if (iocstatus = ddi_get16(accessp, &initreply->IOCStatus)) {
743         mptsas_log(mpt, CE_WARN, "mptsas_do_ioc_init_reply: "
744             "IOCStatus=0x%x, IOCLogInfo=0x%x", iocstatus,
745             ddi_get32(accessp, &initreply->IOCLogInfo));
746         return (DDI_FAILURE);
747     }

749     if ((ddi_get32(mpt->m_datap, &mpt->m_reg->Doorbell) &
750         MPI2_IOC_STATE_OPERATIONAL) {
751         mptsas_log(mpt, CE_NOTE,
752             "?mpt%d: IOC Operational.\n", mpt->m_instance);
753     } else {
754         return (DDI_FAILURE);
755     }

757     return (DDI_SUCCESS);
758 }
759 #endif /* ! codereview */

```

new/usr/src/uts/common/io/scsi/adapters/mpt_sas3/mptsas3_raid.c

1

```
*****
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
*****

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

22 /*
23  * Copyright 2010 Sun Microsystems, Inc. All rights reserved.
24  * Use is subject to license terms.
25  * Copyright (c) 2014, Tegile Systems Inc. All rights reserved.
26  */

28 /*
29  * Copyright (c) 2000 to 2010, LSI Corporation.
30  * All rights reserved.
31  *
32  * Redistribution and use in source and binary forms of all code within
33  * this file that is exclusively owned by LSI, with or without
34  * modification, is permitted provided that, in addition to the CDDL 1.0
35  * License requirements, the following conditions are met:
36  *
37  * Neither the name of the author nor the names of its contributors may be
38  * used to endorse or promote products derived from this software without
39  * specific prior written permission.
40  *
41  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
42  * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
43  * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
44  * FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
45  * COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
46  * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
47  * BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
48  * OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
49  * AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
50  * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
51  * OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
52  * DAMAGE.
53 */

55 /*
56  * mptsas_raid - This file contains all the RAID related functions for the
57  * MPT interface.
58  */

60 #if defined(lint) || defined(DEBUG)
61 #define MPTSAS_DEBUG
```

new/usr/src/uts/common/io/scsi/adapters/mpt_sas3/mptsas3_raid.c

2

```
62 #endif

64 #define MPI_RAID_VOL_PAGE_0_PHYSDISK_MAX      2

66 /*
67  * standard header files
68  */
69 #include <sys/note.h>
70 #include <sys/scsi/scsi.h>
71 #include <sys/byteorder.h>
72 #include <sys/raidioctl.h>

74 #pragma pack(1)

76 #include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_type.h>
77 #include <sys/scsi/adapters/mpt_sas3/mpi/mpi2.h>
78 #include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_cnfg.h>
79 #include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_init.h>
80 #include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_ioc.h>
81 #include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_raid.h>
82 #include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_tool.h>

84 #pragma pack()

86 /*
87  * private header files.
88  */
89 #include <sys/scsi/adapters/mpt_sas3/mptsas3_var.h>

91 static int mptsas_get_raid_wwid(mptsas_t *mpt, mptsas_raidvol_t *raidvol);

93 extern int mptsas_check_dma_handle(ddi_dma_handle_t handle);
94 extern int mptsas_check_acc_handle(ddi_acc_handle_t handle);
95 extern mptsas_target_t *mptsas_tgt_alloc(mptsas_t *, uint16_t,
96     uint64_t, uint32_t, mptsas_phymask_t, uint8_t);

98 static int
99 mptsas_raidconf_page_0_cb(mptsas_t *mpt, caddr_t page_memp,
100     ddi_acc_handle_t accessp, uint16_t iocstatus, uint32_t iocloginfo,
101     va_list ap)
102 {
103     #ifndef __lock_lint
104     _NOTE(ARGUNUSED(ap))
105     #endif
106     pMpi2RaidConfigurationPage0_t raidconf_page0;
107     pMpi2RaidConfig0ConfigElement_t element;
108     uint32_t *confignum;
109     int rval = DDI_SUCCESS, i;
110     uint8_t numelements, vol, disk;
111     uint16_t elementtype, voldevhandle;
112     uint16_t etype_vol, etype_pd, etype_hs;
113     uint16_t etype_oce;
114     m_raidconfig_t *raidconfig;
115     uint64_t raidwn;
116     uint32_t native;
117     mptsas_target_t *ptgt;
118     uint32_t configindex;

120     if (iocstatus == MPI2_IOCSTATUS_CONFIG_INVALID_PAGE) {
121         return (DDI_FAILURE);
122     }

124     if (iocstatus != MPI2_IOCSTATUS_SUCCESS) {
125         mptsas_log(mpt, CE_WARN, "mptsas_get_raid_conf_page0 "
126             "config: IOCStatus=0x%x, IOCLogInfo=0x%x",
127             iocstatus, iocloginfo);
```

```

128         rval = DDI_FAILURE;
129         return (rval);
130     }
131     confignum = va_arg(ap, uint32_t *);
132     configindex = va_arg(ap, uint32_t);
133     raidconfig_page0 = (pMpi2RaidConfigurationPage0_t)page_memp;
134     /*
135      * Get all RAID configurations.
136      */
137     etype_vol = MPI2_RAIDCONFIG0_EFLAGS_VOLUME_ELEMENT;
138     etype_pd = MPI2_RAIDCONFIG0_EFLAGS_VOL_PHYS_DISK_ELEMENT;
139     etype_hs = MPI2_RAIDCONFIG0_EFLAGS_HOT_SPARE_ELEMENT;
140     etype_oce = MPI2_RAIDCONFIG0_EFLAGS_OCE_ELEMENT;
141     /*
142      * Set up page address for next time through.
143      */
144     *confignum = ddi_get8(accessp,
145         &raidconfig_page0->ConfigNum);

147     /*
148      * Point to the right config in the structure.
149      * Increment the number of valid RAID configs.
150      */
151     raidconfig = &mpt->m_raidconfig[configindex];
152     mpt->m_num_raid_configs++;

154     /*
155      * Set the native flag if this is not a foreign
156      * configuration.
157      */
158     native = ddi_get32(accessp, &raidconfig_page0->Flags);
159     if (native & MPI2_RAIDCONFIG0_FLAG_FOREIGN_CONFIG) {
160         native = FALSE;
161     } else {
162         native = TRUE;
163     }
164     raidconfig->m_native = (uint8_t)native;

166     /*
167      * Get volume information for the volumes in the
168      * config.
169      */
170     numelements = ddi_get8(accessp, &raidconfig_page0->NumElements);
171     vol = 0;
172     disk = 0;
173     element = (pMpi2RaidConfig0ConfigElement_t)
174         &raidconfig_page0->ConfigElement;

176     for (i = 0; ((i < numelements) && native); i++, element++) {
177         /*
178          * Get the element type. Could be Volume,
179          * PhysDisk, Hot Spare, or Online Capacity
180          * Expansion PhysDisk.
181          */
182         elementtype = ddi_get16(accessp, &element->ElementFlags);
183         elementtype &= MPI2_RAIDCONFIG0_EFLAGS_MASK_ELEMENT_TYPE;

185         /*
186          * For volumes, get the RAID settings and the
187          * WWID.
188          */
189         if (elementtype == etype_vol) {
190             voldevhandle = ddi_get16(accessp,
191                 &element->VolDevHandle);
192             raidconfig->m_raidvol[vol].m_israid = 1;
193             raidconfig->m_raidvol[vol].

```

```

194             m_raidhandle = voldevhandle;
195             /*
196              * Get the settings for the raid
197              * volume. This includes the
198              * DevHandles for the disks making up
199              * the raid volume.
200              */
201             if (mptsas_get_raid_settings(mpt,
202                 &raidconfig->m_raidvol[vol]))
203                 continue;

205             /*
206              * Get the WWID of the RAID volume for
207              * SAS HBA
208              */
209             if (mptsas_get_raid_wwid(mpt,
210                 &raidconfig->m_raidvol[vol]))
211                 continue;

213             raidwn = raidconfig->m_raidvol[vol].
214                 m_raidwwid;

216             /*
217              * RAID uses phymask of 0.
218              */
219             ptgt = mptsas_tgt_alloc(mpt,
220                 voldevhandle, raidwn, 0, 0, 0);

222             raidconfig->m_raidvol[vol].m_raidtgt =
223                 ptgt;

225             /*
226              * Increment volume index within this
227              * raid config.
228              */
229             vol++;
230         } else if ((elementtype == etype_pd) ||
231             (elementtype == etype_hs) ||
232             (elementtype == etype_oce)) {
233             /*
234              * For all other element types, put
235              * their DevHandles in the phys disk
236              * list of the config. These are all
237              * some variation of a Phys Disk and
238              * this list is used to keep these
239              * disks from going online.
240              */
241             raidconfig->m_physdisk_devhdl[disk] = ddi_get16(accessp,
242                 &element->PhysDiskDevHandle);

244             /*
245              * Increment disk index within this
246              * raid config.
247              */
248             disk++;
249         }
250     }

252     return (rval);
253 }

255 int
256 mptsas_get_raid_info(mptsas_t *mpt)
257 {
258     int rval = DDI_SUCCESS;
259     uint32_t confignum, pageaddress;

```

```

260     uint8_t configindex;

262     ASSERT(mutex_owned(&mpt->m_mutex));

264     /*
265      * Clear all RAID info before starting.
266      */
267     bzero(mpt->m_raidconfig, sizeof (mpt->m_raidconfig));
268     mpt->m_num_raid_configs = 0;

270     configindex = 0;
271     confignum = 0xff;
272     pageaddress = MPI2_RAID_PGAD_FORM_GET_NEXT_CONFIGNUM | confignum;
273     while (rval == DDI_SUCCESS) {
274         /*
275          * Get the header and config page.  reply contains the reply
276          * frame, which holds status info for the request.
277          */
278         rval = mptsas_access_config_page(mpt,
279             MPI2_CONFIG_ACTION_PAGE_READ_CURRENT,
280             MPI2_CONFIG_EXTPAGETYPE_RAID_CONFIG, 0, pageaddress,
281             mptsas_raidconf_page_0_cb, &confignum, configindex);
282         configindex++;
283         pageaddress = MPI2_RAID_PGAD_FORM_GET_NEXT_CONFIGNUM |
284             confignum;
285     }

287     return (rval);
288 }

290 static int
291 mptsas_raidvol_page_0_cb(mptsas_t *mpt, caddr_t page_memp,
292     ddi_acc_handle_t accesssp, uint16_t iocstatus, uint32_t iocloginfo,
293     va_list ap)
294 {
295     #ifndef __lock_lint
296     _NOTE(ARGUNUSED(ap))
297     #endif
298     pMpi2RaidVolPage0_t raidpage;
299     int rval = DDI_SUCCESS, i;
300     mptsas_raidvol_t *raidvol;
301     uint8_t numdisks, volstate, voltype, physdisknum;
302     uint32_t volsetting;
303     uint32_t statusflags, resync_flag;

305     if (iocstatus == MPI2_IOCSTATUS_CONFIG_INVALID_PAGE)
306         return (DDI_FAILURE);

308     if (iocstatus != MPI2_IOCSTATUS_SUCCESS) {
309         mptsas_log(mpt, CE_WARN, "mptsas_raidvol_page0_cb "
310             "config: IOCStatus=0x%x, IOCLogInfo=0x%x",
311             iocstatus, iocloginfo);
312         rval = DDI_FAILURE;
313         return (rval);
314     }

316     raidvol = va_arg(ap, mptsas_raidvol_t *);

318     raidpage = (pMpi2RaidVolPage0_t)page_memp;
319     volstate = ddi_get8(accesssp, &raidpage->VolumeState);
320     volsetting = ddi_get32(accesssp,
321         (uint32_t *)(&raidpage->VolumeSettings));
322     statusflags = ddi_get32(accesssp, &raidpage->VolumeStatusFlags);
323     voltype = ddi_get8(accesssp, &raidpage->VolumeType);

325     raidvol->m_state = volstate;

```

```

326     raidvol->m_statusflags = statusflags;
327     /*
328      * Volume size is not used right now. Set to 0.
329      */
330     raidvol->m_raidsize = 0;
331     raidvol->m_settings = volsetting;
332     raidvol->m_raidlevel = voltype;

334     if (statusflags & MPI2_RAIDVOL0_STATUS_FLAG_QUIESCED) {
335         mptsas_log(mpt, CE_NOTE, "?Volume %d is quiesced\n",
336             raidvol->m_raidhandle);
337     }

339     if (statusflags &
340         MPI2_RAIDVOL0_STATUS_FLAG_RESYNC_IN_PROGRESS) {
341         mptsas_log(mpt, CE_NOTE, "?Volume %d is resyncing\n",
342             raidvol->m_raidhandle);
343     }

345     resync_flag = MPI2_RAIDVOL0_STATUS_FLAG_RESYNC_IN_PROGRESS;
346     switch (volstate) {
347     case MPI2_RAID_VOL_STATE_OPTIMAL:
348         mptsas_log(mpt, CE_NOTE, "?Volume %d is "
349             "optimal\n", raidvol->m_raidhandle);
350         break;
351     case MPI2_RAID_VOL_STATE_DEGRADED:
352         if ((statusflags & resync_flag) == 0) {
353             mptsas_log(mpt, CE_WARN, "Volume %d "
354                 "is degraded\n",
355                 raidvol->m_raidhandle);
356         }
357         break;
358     case MPI2_RAID_VOL_STATE_FAILED:
359         mptsas_log(mpt, CE_WARN, "Volume %d is "
360             "failed\n", raidvol->m_raidhandle);
361         break;
362     case MPI2_RAID_VOL_STATE_MISSING:
363         mptsas_log(mpt, CE_WARN, "Volume %d is "
364             "missing\n", raidvol->m_raidhandle);
365         break;
366     default:
367         break;
368     }
369     numdisks = raidpage->NumPhysDisks;
370     raidvol->m_ndisks = numdisks;
371     for (i = 0; i < numdisks; i++) {
372         physdisknum = raidpage->PhysDisk[i].PhysDiskNum;
373         raidvol->m_disknum[i] = physdisknum;
374         if (mptsas_get_physdisk_settings(mpt, raidvol,
375             physdisknum))
376             break;
377     }
378     return (rval);
379 }

381 int
382 mptsas_get_raid_settings(mptsas_t *mpt, mptsas_raidvol_t *raidvol)
383 {
384     int rval = DDI_SUCCESS;
385     uint32_t page_address;

387     ASSERT(mutex_owned(&mpt->m_mutex));

389     /*
390      * Get the header and config page.  reply contains the reply frame,
391      * which holds status info for the request.

```

```

392     */
393     page_address = (MPI2_RAID_VOLUME_PGAD_FORM_MASK &
394                     MPI2_RAID_VOLUME_PGAD_FORM_HANDLE) | raidvol->m_raidhandle;
395     rval = mptsas_access_config_page(mpt,
396                                     MPI2_CONFIG_ACTION_PAGE_READ_CURRENT,
397                                     MPI2_CONFIG_PAGETYPE_RAID_VOLUME, 0, page_address,
398                                     mptsas_raidvol_page_0_cb, raidvol);
400     return (rval);
401 }

403 static int
404 mptsas_raidvol_page_1_cb(mptsas_t *mpt, caddr_t page_memp,
405                          ddi_acc_handle_t accessp, uint16_t iocstatus, uint32_t iocloginfo,
406                          va_list ap)
407 {
408     #ifndef __lock_lint
409         _NOTE(ARGUNUSED(ap))
410     #endif
411     pMpi2RaidVolPage1_t      raidpage;
412     int                      rval = DDI_SUCCESS, i;
413     uint8_t                  *sas_addr = NULL;
414     uint8_t                  tmp_sas_wnn[SAS_WWN_BYTE_SIZE];
415     uint64_t                  *sas_wnn;

417     if (iocstatus != MPI2_IOCSTATUS_SUCCESS) {
418         mptsas_log(mpt, CE_WARN, "mptsas_raidvol_page_1_cb "
419                    "config: IOCStatus=0x%x, IOCLogInfo=0x%x",
420                    iocstatus, iocloginfo);
421         rval = DDI_FAILURE;
422         return (rval);
423     }
424     sas_wnn = va_arg(ap, uint64_t *);

426     raidpage = (pMpi2RaidVolPage1_t)page_memp;
427     sas_addr = (uint8_t *)(&raidpage->WWID);
428     for (i = 0; i < SAS_WWN_BYTE_SIZE; i++) {
429         tmp_sas_wnn[i] = ddi_get8(accessp, sas_addr + i);
430     }
431     bcopy(tmp_sas_wnn, sas_wnn, SAS_WWN_BYTE_SIZE);
432     *sas_wnn = LE_64(*sas_wnn);
433     return (rval);
434 }

436 static int
437 mptsas_get_raid_wwid(mptsas_t *mpt, mptsas_raidvol_t *raidvol)
438 {
439     int rval = DDI_SUCCESS;
440     uint32_t page_address;
441     uint64_t sas_wnn;

443     ASSERT(mutex_owned(&mpt->m_mutex));

445     /*
446      * Get the header and config page.  reply contains the reply frame,
447      * which holds status info for the request.
448      */
449     page_address = (MPI2_RAID_VOLUME_PGAD_FORM_MASK &
450                     MPI2_RAID_VOLUME_PGAD_FORM_HANDLE) | raidvol->m_raidhandle;
451     rval = mptsas_access_config_page(mpt,
452                                     MPI2_CONFIG_ACTION_PAGE_READ_CURRENT,
453                                     MPI2_CONFIG_PAGETYPE_RAID_VOLUME, 1, page_address,
454                                     mptsas_raidvol_page_1_cb, &sas_wnn);

456     /*
457      * Get the required information from the page.

```

```

458     */
459     if (rval == DDI_SUCCESS) {
461         /*
462          * replace top nibble of WWID of RAID to '3' for OBP
463          */
464         sas_wnn = MPTSAS_RAID_WWID(sas_wnn);
465         raidvol->m_raidwwid = sas_wnn;
466     }

468 done:
469     return (rval);
470 }

472 static int
473 mptsas_raidphysdisk_page_0_cb(mptsas_t *mpt, caddr_t page_memp,
474                                ddi_acc_handle_t accessp, uint16_t iocstatus, uint32_t iocloginfo,
475                                va_list ap)
476 {
477     #ifndef __lock_lint
478         _NOTE(ARGUNUSED(ap))
479     #endif
480     pMpi2RaidPhysDiskPage0_t diskpage;
481     int                      rval = DDI_SUCCESS;
482     uint16_t                  *devhdl;
483     uint8_t                  *state;

485     if (iocstatus == MPI2_IOCSTATUS_CONFIG_INVALID_PAGE)
486         return (DDI_FAILURE);

488     if (iocstatus != MPI2_IOCSTATUS_SUCCESS) {
489         mptsas_log(mpt, CE_WARN, "mptsas_raidphysdisk_page0_cb "
490                    "config: IOCStatus=0x%x, IOCLogInfo=0x%x",
491                    iocstatus, iocloginfo);
492         rval = DDI_FAILURE;
493         return (rval);
494     }
495     devhdl = va_arg(ap, uint16_t *);
496     state = va_arg(ap, uint8_t *);
497     diskpage = (pMpi2RaidPhysDiskPage0_t)page_memp;
498     *devhdl = ddi_get16(accessp, &diskpage->DevHandle);
499     *state = ddi_get8(accessp, &diskpage->PhysDiskState);
500     return (rval);
501 }

503 int
504 mptsas_get_physdisk_settings(mptsas_t *mpt, mptsas_raidvol_t *raidvol,
505                              uint8_t physdisknum)
506 {
507     int                      rval = DDI_SUCCESS, i;
508     uint8_t                  state;
509     uint16_t                  devhdl;
510     uint32_t                  page_address;

512     ASSERT(mutex_owned(&mpt->m_mutex));

514     /*
515      * Get the header and config page.  reply contains the reply frame,
516      * which holds status info for the request.
517      */
518     page_address = (MPI2_PHYSDISK_PGAD_FORM_MASK &
519                     MPI2_PHYSDISK_PGAD_FORM_PHYSDISKNUM) | physdisknum;
520     rval = mptsas_access_config_page(mpt,
521                                     MPI2_CONFIG_ACTION_PAGE_READ_CURRENT,
522                                     MPI2_CONFIG_PAGETYPE_RAID_PHYSDISK, 0, page_address,
523                                     mptsas_raidphysdisk_page_0_cb, &devhdl, &state);

```

```

525     /*
526     * Get the required information from the page.
527     */
528     if (rval == DDI_SUCCESS) {
529         for (i = 0; i < MPTSAS_MAX_DISKS_IN_VOL; i++) {
530             /* find the correct position in the arrays */
531             if (raidvol->m_disknum[i] == physdisknum)
532                 break;
533         }
534         raidvol->m_devhdl[i] = devhdl;

536         switch (state) {
537             case MPI2_RAID_PD_STATE_OFFLINE:
538                 raidvol->m_diskstatus[i] =
539                     RAID_DISKSTATUS_FAILED;
540                 break;

542             case MPI2_RAID_PD_STATE_HOT_SPARE:
543             case MPI2_RAID_PD_STATE_NOT_CONFIGURED:
544             case MPI2_RAID_PD_STATE_NOT_COMPATIBLE:
545                 break;

547             case MPI2_RAID_PD_STATE_DEGRADED:
548             case MPI2_RAID_PD_STATE_OPTIMAL:
549             case MPI2_RAID_PD_STATE_REBUILDING:
550             case MPI2_RAID_PD_STATE_ONLINE:
551             default:
552                 raidvol->m_diskstatus[i] =
553                     RAID_DISKSTATUS_GOOD;
554                 break;
555         }
556     }

558     return (rval);
559 }

561 /*
562 * RAID Action for System Shutdown. This request uses the dedicated TM slot to
563 * avoid a call to mptsas_save_cmd. Since Solaris requires that the mutex is
564 * not held during the mptsas_quiesce function, this RAID action must not use
565 * the normal code path of requests and replies.
566 */
567 void
568 mptsas_raid_action_system_shutdown(mptsas_t *mpt)
569 {
570     mptsas_reply_pqueue_t      *rpqp;
571     pMpi2RaidActionRequest_t    action;
572     uint8_t                    ir_active = FALSE, reply_type;
573     uint8_t                    function, found_reply = FALSE;
574     uint16_t                    SMID, action_type;
575     mptsas_slots_t             *slots = mpt->m_active;
576     int                         config, vol;
577     mptsas_cmd_t               *cmd;
578     uint32_t                    reply_addr;
579     uint64_t                    request_desc;
580     int                         cnt;
581     pMpi2ReplyDescriptorsUnion_t reply_desc_union;
582     pMPI2DefaultReply_t         reply;
583     pMpi2AddressReplyDescriptor_t address_reply;

585     /*
586     * Before doing the system shutdown RAID Action, make sure that the IOC
587     * supports IR and make sure there is a valid volume for the request.
588     */
589     if (mpt->m_ir_capable) {

```

```

590         for (config = 0; (config < mpt->m_num_raid_configs) &&
591             (!ir_active); config++) {
592             for (vol = 0; vol < MPTSAS_MAX_RAIDVOLS; vol++) {
593                 if (mpt->m_raidconfig[config].m_raidvol[vol].
594                     m_raidraid) {
595                     ir_active = TRUE;
596                     break;
597                 }
598             }
599         }
600     }
601     if (!ir_active) {
602         return;
603     }

605     /*
606     * If TM slot is already being used (highly unlikely), show message and
607     * don't issue the RAID action.
608     */
609     if (slots->m_slot[MPTSAS_TM_SLOT(mpt)] != NULL) {
610         mptsas_log(mpt, CE_WARN, "RAID Action slot in use. Cancelling"
611             " System Shutdown RAID Action.\n");
612         return;
613     }

615     /*
616     * Create the cmd and put it in the dedicated TM slot.
617     */
618     cmd = &(mpt->m_event_task_mgmt.m_event_cmd);
619     bzero((caddr_t)cmd, sizeof (*cmd));
620     cmd->cmd_pkt = NULL;
621     cmd->cmd_slot = MPTSAS_TM_SLOT(mpt);
622     slots->m_slot[MPTSAS_TM_SLOT(mpt)] = cmd;

624     /*
625     * Form message for raid action.
626     */
627     action = (pMpi2RaidActionRequest_t)(mpt->m_req_frame +
628         (mpt->m_req_frame_size * cmd->cmd_slot));
629     bzero(action, mpt->m_req_frame_size);
630     action->Function = MPI2_FUNCTION_RAID_ACTION;
631     action->Action = MPI2_RAID_ACTION_SYSTEM_SHUTDOWN_INITIATED;

633     /*
634     * Send RAID Action.
635     * Defaults to MSIxIndex of 0, so check reply q 0 below.
636     */
637     (void) ddi_dma_sync(mpt->m_dma_req_frame_hdl, 0, 0,
638         DDI_DMA_SYNC_FORDEV);
639     request_desc = (cmd->cmd_slot << 16) +
640         MPI2_REQ_DESCRIPTOR_FLAGS_DEFAULT_TYPE;
641     MPTSAS_START_CMD(mpt, request_desc);

643     /*
644     * Even though reply does not matter because the system is shutting
645     * down, wait no more than 5 seconds here to get the reply just because
646     * we don't want to leave it hanging if it's coming. Poll because
647     * interrupts are disabled when this function is called.
648     */
649     rpqp = mpt->m_rep_post_queues;
650     for (cnt = 0; cnt < 5000; cnt++) {
651         /*
652         * Check for a reply.
653         */
654         (void) ddi_dma_sync(mpt->m_dma_post_queue_hdl, 0, 0,
655             DDI_DMA_SYNC_FORCPU);

```

```

657     reply_desc_union = (mPmpi2ReplyDescriptorsUnion_t)
658         MPTSAS_GET_NEXT_REPLY(rpq, rpq->rpq_index);

660     if (ddi_get32(mpt->m_acc_post_queue_hdl,
661         &reply_desc_union->Words.Low) == 0xFFFFFFFF ||
662         ddi_get32(mpt->m_acc_post_queue_hdl,
663         &reply_desc_union->Words.High) == 0xFFFFFFFF) {
664         drv_usecwait(1000);
665         continue;
666     }

668     /*
669     * There is a reply. If it's not an address reply, ignore it.
670     */
671     reply_type = ddi_get8(mpt->m_acc_post_queue_hdl,
672         &reply_desc_union->Default.ReplyFlags);
673     reply_type &= MPI2_RPY_DESCRIPTOR_FLAGS_TYPE_MASK;
674     if (reply_type != MPI2_RPY_DESCRIPTOR_FLAGS_ADDRESS_REPLY) {
675         goto clear_and_continue;
676     }

678     /*
679     * SMID must be the TM slot since that's what we're using for
680     * this RAID action. If not, ignore this reply.
681     */
682     address_reply =
683         (pMpi2AddressReplyDescriptor_t)reply_desc_union;
684     SMID = ddi_get16(mpt->m_acc_post_queue_hdl,
685         &address_reply->SMID);
686     if (SMID != MPTSAS_TM_SLOT(mpt)) {
687         goto clear_and_continue;
688     }

690     /*
691     * If reply frame is not in the proper range ignore it.
692     */
693     reply_addr = ddi_get32(mpt->m_acc_post_queue_hdl,
694         &address_reply->ReplyFrameAddress);
695     if ((reply_addr < mpt->m_reply_frame_dma_addr) ||
696         (reply_addr >= (mpt->m_reply_frame_dma_addr +
697             (mpt->m_reply_frame_size * mpt->m_free_queue_depth))) ||
698         ((reply_addr - mpt->m_reply_frame_dma_addr) %
699             mpt->m_reply_frame_size != 0)) {
700         goto clear_and_continue;
701     }

703     /*
704     * If not a RAID action reply ignore it.
705     */
706     (void) ddi_dma_sync(mpt->m_dma_reply_frame_hdl, 0, 0,
707         DDI_DMA_SYNC_FORCPU);
708     reply = (pMPI2DefaultReply_t)(mpt->m_reply_frame +
709         (reply_addr - mpt->m_reply_frame_dma_addr));
710     function = ddi_get8(mpt->m_acc_reply_frame_hdl,
711         &reply->Function);
712     if (function != MPI2_FUNCTION_RAID_ACTION) {
713         goto clear_and_continue;
714     }

716     /*
717     * Finally, make sure this is the System Shutdown RAID action.
718     * If not, ignore reply.
719     */
720     action_type = ddi_get16(mpt->m_acc_reply_frame_hdl,
721         &reply->FunctionDependent1);

```

```

722     if (action_type !=
723         MPI2_RAID_ACTION_SYSTEM_SHUTDOWN_INITIATED) {
724         goto clear_and_continue;
725     }
726     found_reply = TRUE;

728 clear_and_continue:
729     /*
730     * Clear the reply descriptor for re-use and increment index.
731     */
732     ddi_put64(mpt->m_acc_post_queue_hdl,
733         &((uint64_t *) (void *) rpq->rpq_queue)[rpq->rpq_index],
734         0xFFFFFFFFFFFFFFFF);
735     (void) ddi_dma_sync(mpt->m_dma_post_queue_hdl, 0, 0,
736         DDI_DMA_SYNC_FORDEV);

738     /*
739     * Update the reply index and keep looking for the
740     * reply if not found yet.
741     */
742     if (++rpq->rpq_index == mpt->m_post_queue_depth) {
743         rpq->rpq_index = 0;
744     }

746     ddi_put32(mpt->m_datap, &mpt->m_reg->ReplyPostHostIndex,
747         rpq->rpq_index);
748     if (!found_reply) {
749         continue;
750     }

752     break;
753 }

755     /*
756     * clear the used slot as the last step.
757     */
758     slots->m_slot[MPTSAS_TM_SLOT(mpt)] = NULL;
759 }

761 int
762 mptsas_delete_volume(mptsas_t *mpt, uint16_t volid)
763 {
764     int config, i = 0, vol = (-1);

766     for (config = 0; (config < mpt->m_num_raid_configs) && (vol != i);
767         config++) {
768         for (i = 0; i < MPTSAS_MAX_RAIDVOLS; i++) {
769             if (mpt->m_raidconfig[config].m_raidvol[i].
770                 m_raidhandle == valid) {
771                 vol = i;
772                 break;
773             }
774         }
775     }

777     if (vol < 0) {
778         mptsas_log(mpt, CE_WARN, "raid doesn't exist at specified "
779             "target.");
780         return (-1);
781     }

783     mpt->m_raidconfig[config].m_raidvol[vol].m_israid = 0;
784     mpt->m_raidconfig[config].m_raidvol[vol].m_ndisks = 0;
785     for (i = 0; i < MPTSAS_MAX_DISKS_IN_VOL; i++) {
786         mpt->m_raidconfig[config].m_raidvol[vol].m_disknum[i] = 0;
787         mpt->m_raidconfig[config].m_raidvol[vol].m_devhdl[i] = 0;

```



```
788     }  
790     return (0);  
791 }  
792 #endif /* ! codereview */
```

new/usr/src/uts/common/io/scsi/adapters/mpt_sas3/mptsas3_smhba.c

1

```
*****
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
*****

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

22 /*
23  * Copyright (c) 2009, 2010, Oracle and/or its affiliates. All rights reserved.
24  * Copyright 2013 Nexenta Systems, Inc. All rights reserved.
25  * Copyright (c) 2014, Tegile Systems Inc. All rights reserved.
26 */
27 /*
28  * This file contains SM-HBA support for MPT SAS3 driver
29 */

31 #if defined(lint) || defined(DEBUG)
32 #define MPTSAS_DEBUG
33 #endif

35 /*
36  * standard header files
37 */
38 #include <sys/note.h>
39 #include <sys/scsi/scsi.h>
40 #include <sys/pci.h>
41 #include <sys/scsi/generic/sas.h>
42 #include <sys/scsi/impl/scsi_sas.h>

44 #pragma pack(1)
45 #include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_type.h>
46 #include <sys/scsi/adapters/mpt_sas3/mpi/mpi2.h>
47 #include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_cnfg.h>
48 #include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_init.h>
49 #include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_loc.h>
50 #include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_sas.h>
51 #pragma pack()

53 /*
54  * private header files.
55 */
56 #include <sys/scsi/adapters/mpt_sas3/mptsas3_var.h>
57 #include <sys/scsi/adapters/mpt_sas3/mptsas3_smhba.h>

59 /*
60  * SM - HBA statics
61 */
```

new/usr/src/uts/common/io/scsi/adapters/mpt_sas3/mptsas3_smhba.c

2

```
62 extern char *mptsas_driver_rev;

64 static void mptsas_smhba_create_phy_props(nvlist_t **, smhba_info_t *, uint8_t,
65     uint16_t *);
66 static void mptsas_smhba_update_phy_props(mptsas_t *, dev_info_t *, nvlist_t **,
67     uint8_t);

69 static void
70 mptsas_smhba_add_hba_prop(mptsas_t *mpt, data_type_t dt,
71     char *prop_name, void *prop_val);

73 void
74 mptsas_smhba_show_phy_info(mptsas_t *mpt);

76 static void
77 mptsas_smhba_add_hba_prop(mptsas_t *mpt, data_type_t dt,
78     char *prop_name, void *prop_val)
79 {
80     ASSERT(mpt != NULL);

82     switch (dt) {
83     case DATA_TYPE_INT32:
84         if (ddi_prop_update_int(DDI_DEV_T_NONE, mpt->m_dip,
85             prop_name, *(int *)prop_val)) {
86             mptsas_log(mpt, CE_WARN,
87                 "%s: %s prop update failed", __func__, prop_name);
88         }
89         break;
90     case DATA_TYPE_STRING:
91         if (ddi_prop_update_string(DDI_DEV_T_NONE, mpt->m_dip,
92             prop_name, (char *)prop_val)) {
93             mptsas_log(mpt, CE_WARN,
94                 "%s: %s prop update failed", __func__, prop_name);
95         }
96         break;
97     default:
98         mptsas_log(mpt, CE_WARN, "%s: "
99             "Unhandled datatype(%d) for (%s). Skipping prop update.",
100             __func__, dt, prop_name);
101     }
102 }

104 void
105 mptsas_smhba_show_phy_info(mptsas_t *mpt)
106 {
107     int i;

109     ASSERT(mpt != NULL);

111     for (i = 0; i < MPTSAS_MAX_PHYS; i++) {
112         mptsas_log(mpt, CE_WARN,
113             "phy %d, Owner hdl:0x%x, attached hdl: 0x%x,"
114             "attached phy identifier %d,Program link rate 0x%x,"
115             "hw link rate 0x%x, negotiator link rate 0x%x, path %s",
116             i, mpt->m_phy_info[i].smhba_info.owner_devhdl,
117             mpt->m_phy_info[i].smhba_info.attached_devhdl,
118             mpt->m_phy_info[i].smhba_info.attached_phy_identify,
119             mpt->m_phy_info[i].smhba_info.programmed_link_rate,
120             mpt->m_phy_info[i].smhba_info.hw_link_rate,
121             mpt->m_phy_info[i].smhba_info.negotiated_link_rate,
122             mpt->m_phy_info[i].smhba_info.path);
123     }
124 }

126 static void
127 mptsas_smhba_create_phy_props(nvlist_t **phy_props, smhba_info_t *pSmhba,
```

```

128     uint8_t phy_id, uint16_t *attached_devhdl)
129 {
130     (void) nvlist_alloc(phy_props, NV_UNIQUE_NAME, KM_SLEEP);
131     (void) nvlist_add_uint8(*phy_props, SAS_PHY_ID, phy_id);
132     (void) nvlist_add_uint8(*phy_props, "phyState",
133         (pSmhba->negotiated_link_rate & 0x0f));
134     (void) nvlist_add_int8(*phy_props, SAS_NEG_LINK_RATE,
135         (pSmhba->negotiated_link_rate & 0x0f));
136     (void) nvlist_add_int8(*phy_props, SAS_PROG_MIN_LINK_RATE,
137         (pSmhba->programmed_link_rate & 0x0f));
138     (void) nvlist_add_int8(*phy_props, SAS_HW_MIN_LINK_RATE,
139         (pSmhba->hw_link_rate & 0x0f));
140     (void) nvlist_add_int8(*phy_props, SAS_PROG_MAX_LINK_RATE,
141         ((pSmhba->programmed_link_rate & 0xf0) >> 4));
142     (void) nvlist_add_int8(*phy_props, SAS_HW_MAX_LINK_RATE,
143         ((pSmhba->hw_link_rate & 0xf0) >> 4));
144
145     if (pSmhba->attached_devhdl && (attached_devhdl != NULL))
146         *attached_devhdl = pSmhba->attached_devhdl;
147 }
148
149 static void
150 mptsas_smhba_update_phy_props(mptsas_t *mpt, dev_info_t *dip,
151     nvlist_t **phy_props, uint8_t phy_nums)
152 {
153     int            rval;
154     size_t         packed_size;
155     char           *packed_data = NULL;
156     nvlist_t       *nvl;
157
158     if (nvlist_alloc(&nvl, NV_UNIQUE_NAME, KM_SLEEP) != 0) {
159         mptsas_log(mpt, CE_WARN, "%s: nvlist_alloc() failed", __func__);
160         return;
161     }
162
163     rval = nvlist_add_nvlist_array(nvl, SAS_PHY_INFO_NVL, phy_props,
164         phy_nums);
165     if (rval) {
166         mptsas_log(mpt, CE_WARN,
167             "nv list array add failed, return value %d.", rval);
168         goto exit;
169     }
170     (void) nvlist_size(nvl, &packed_size, NV_ENCODE_NATIVE);
171     packed_data = kmem_zalloc(packed_size, KM_SLEEP);
172     (void) nvlist_pack(nvl, &packed_data, &packed_size,
173         NV_ENCODE_NATIVE, 0);
174
175     (void) ddi_prop_update_byte_array(DDI_DEV_T_NONE, dip,
176         SAS_PHY_INFO, (uchar_t *)packed_data, packed_size);
177
178 exit:
179     nvlist_free(nvl);
180
181     if (packed_data != NULL) {
182         kmem_free(packed_data, packed_size);
183     }
184 }
185
186 void
187 mptsas_smhba_set_one_phy_props(mptsas_t *mpt, dev_info_t *dip, uint8_t phy_id,
188     uint16_t *attached_devhdl)
189 {
190     nvlist_t       *phy_props;
191
192     ASSERT(phy_id < mpt->m_num_phys);

```

```

194     mptsas_smhba_create_phy_props(&phy_props,
195         &mpt->m_phy_info[phy_id].smhba_info, phy_id, attached_devhdl);
196
197     mptsas_smhba_update_phy_props(mpt, dip, &phy_props, 1);
198
199     nvlist_free(phy_props);
200 }
201
202 void
203 mptsas_smhba_set_all_phy_props(mptsas_t *mpt, dev_info_t *dip, uint8_t phy_nums,
204     mptsas_phymask_t phy_mask, uint16_t *attached_devhdl)
205 {
206     int            i, j;
207     nvlist_t       **phy_props;
208
209     if (phy_nums == 0)
210         return;
211
212     phy_props = kmem_zalloc(sizeof (nvlist_t *) * phy_nums, KM_SLEEP);
213
214     for (i = 0, j = 0; i < mpt->m_num_phys && j < phy_nums; i++)
215         if (phy_mask == mpt->m_phy_info[i].phy_mask)
216             mptsas_smhba_create_phy_props(&phy_props[j++],
217                 &mpt->m_phy_info[i].smhba_info, i, attached_devhdl);
218
219     mptsas_smhba_update_phy_props(mpt, dip, phy_props, j);
220
221     for (i = 0; i < j && phy_props[i] != NULL; i++)
222         nvlist_free(phy_props[i]);
223
224     kmem_free(phy_props, sizeof (nvlist_t *) * phy_nums);
225 }
226
227 /*
228  * Called with PHY lock held on phy
229  */
230 void
231 mptsas_smhba_log_sysevent(mptsas_t *mpt, char *subclass, char *etype,
232     smhba_info_t *phyp)
233 {
234     nvlist_t       *attr_list;
235     char           *pname;
236     char           sas_addr[MPTSAS_WWN_STRLEN];
237     uint8_t        phynum = 0;
238     uint8_t        lrate = 0;
239
240     if (mpt->m_dip == NULL)
241         return;
242     if (phyp == NULL)
243         return;
244
245     pname = kmem_zalloc(MAXPATHLEN, KM_NOSLEEP);
246     if (pname == NULL)
247         return;
248
249     if ((strcmp(subclass, ESC_SAS_PHY_EVENT) == 0) ||
250         (strcmp(subclass, ESC_SAS_HBA_PORT_BROADCAST) == 0)) {
251         ASSERT(phyp != NULL);
252         (void) strncpy(pname, phyp->path, strlen(phyp->path));
253         phynum = phyp->phy_id;
254         bzero(sas_addr, sizeof (sas_addr));
255         (void) sprintf(sas_addr, "%016"PRIx64, phyp->sas_addr);
256         if (strcmp(etype, SAS_PHY_ONLINE) == 0) {
257             lrate = phyp->negotiated_link_rate;
258         }
259     }

```

```

260     if (strcmp(subclass, ESC_SAS_HBA_PORT_BROADCAST) == 0) {
261         (void) ddi_pathname(mpt->m_dip, pname);
262     }

264     if (nvlist_alloc(&attr_list, NV_UNIQUE_NAME_TYPE, 0) != 0) {
265         mptsas_log(mpt, CE_WARN,
266             "%s: Failed to post sysevent", __func__);
267         kmem_free(pname, MAXPATHLEN);
268         return;
269     }

271     if (nvlist_add_int32(attr_list, SAS_DRV_INST,
272         ddi_get_instance(mpt->m_dip)) != 0)
273         goto fail;

275     if (nvlist_add_string(attr_list, SAS_PORT_ADDR, sas_addr) != 0)
276         goto fail;

278     if (nvlist_add_string(attr_list, SAS_DEVFS_PATH, pname) != 0)
279         goto fail;

281     if (nvlist_add_uint8(attr_list, SAS_PHY_ID, phynum) != 0)
282         goto fail;

284     if (strcmp(etype, SAS_PHY_ONLINE) == 0) {
285         if (nvlist_add_uint8(attr_list, SAS_LINK_RATE, lrate) != 0)
286             goto fail;
287     }

289     if (nvlist_add_string(attr_list, SAS_EVENT_TYPE, etype) != 0)
290         goto fail;

292     (void) ddi_log_sysevent(mpt->m_dip, DDI_VENDOR_SUNW, EC_HBA, subclass,
293         attr_list, NULL, DDI_NOSLEEP);

295 fail:
296     kmem_free(pname, MAXPATHLEN);
297     nvlist_free(attr_list);
298 }

300 void
301 mptsas_create_phy_stats(mptsas_t *mpt, char *iport, dev_info_t *dip)
302 {
303     sas_phy_stats_t      *ps;
304     smhba_info_t         *phyp;
305     int                   ndata;
306     char                  ks_name[KSTAT_STRLEN];
307     char                  phymask[MPTSAS_MAX_PHYS];
308     int                   i;

310     ASSERT(iport != NULL);
311     ASSERT(mpt != NULL);

313     for (i = 0; i < mpt->m_num_phys; i++) {

315         bzero(phymask, sizeof (phymask));
316         (void) sprintf(phymask, "%x", mpt->m_phy_info[i].phy_mask);
317         if (strcmp(phymask, iport) == 0) {

319             phyp = &mpt->m_phy_info[i].smhba_info;
320             mutex_enter(&phyp->phy_mutex);

322             if (phyp->phy_stats != NULL) {
323                 mutex_exit(&phyp->phy_mutex);
324                 /* We've already created this kstat instance */
325                 continue;

```

```

326     }

328     ndata = (sizeof (sas_phy_stats_t)/
329         sizeof (kstat_named_t));
330     (void) snprintf(ks_name, sizeof (ks_name),
331         "%s.%llx.%d.%d", ddi_driver_name(dip),
332         (longlong_t)mpt->un.m_base_wwid,
333         ddi_get_instance(dip), i);

335     phyp->phy_stats = kstat_create("mptsas3",
336         ddi_get_instance(dip), ks_name, KSTAT_SAS_PHY_CLASS,
337         KSTAT_TYPE_NAMED, ndata, 0);

339     if (phyp->phy_stats == NULL) {
340         mutex_exit(&phyp->phy_mutex);
341         mptsas_log(mpt, CE_WARN,
342             "%s: Failed to create %s kstats", __func__,
343             ks_name);
344         continue;
345     }

347     ps = (sas_phy_stats_t *)phyp->phy_stats->ks_data;

349     kstat_named_init(&ps->seconds_since_last_reset,
350         "SecondsSinceLastReset", KSTAT_DATA_ULONGLONG);
351     kstat_named_init(&ps->tx_frames,
352         "TxFrames", KSTAT_DATA_ULONGLONG);
353     kstat_named_init(&ps->rx_frames,
354         "RxFrames", KSTAT_DATA_ULONGLONG);
355     kstat_named_init(&ps->tx_words,
356         "TxWords", KSTAT_DATA_ULONGLONG);
357     kstat_named_init(&ps->rx_words,
358         "RxWords", KSTAT_DATA_ULONGLONG);
359     kstat_named_init(&ps->invalid_dword_count,
360         "InvalidDwordCount", KSTAT_DATA_ULONGLONG);
361     kstat_named_init(&ps->running_disparity_error_count,
362         "RunningDisparityErrorCount", KSTAT_DATA_ULONGLONG);
363     kstat_named_init(&ps->loss_of_dword_sync_count,
364         "LossOfDwordSyncCount", KSTAT_DATA_ULONGLONG);
365     kstat_named_init(&ps->phy_reset_problem_count,
366         "PhyResetProblemCount", KSTAT_DATA_ULONGLONG);

368     phyp->phy_stats->ks_private = phyp;
369     phyp->phy_stats->ks_update = mptsas_update_phy_stats;
370     kstat_install(phyp->phy_stats);
371     mutex_exit(&phyp->phy_mutex);
372 }
373 }
374 }

376 int
377 mptsas_update_phy_stats(kstat_t *ks, int rw)
378 {
379     int                 ret = DDI_FAILURE;
380     smhba_info_t        *pptr = NULL;
381     sas_phy_stats_t     *ps = ks->ks_data;
382     uint32_t             page_address;
383     mptsas_t             *mpt;

385     _NOTE(ARGUNUSED(rw));

387     pptr = (smhba_info_t *)ks->ks_private;
388     ASSERT((pptr != NULL));
389     mpt = (mptsas_t *)pptr->mpt;
390     ASSERT((mpt != NULL));
391     page_address = (MPI2_SAS_PHY_PGAD_FORM_PHY_NUMBER | pptr->phy_id);

```

```

393  /*
394   * We just want to lock against other invocations of kstat;
395   * we don't need to pmcs_lock_phy() for this.
396   */
397  mutex_enter(&mpt->m_mutex);

399  ret = mptsas_get_sas_phy_page1(pptr->mpt, page_address, pptr);

401  if (ret == DDI_FAILURE)
402      goto fail;

404  ps->invalid_dword_count.value.ull =
405      (unsigned long long)pptr->invalid_dword_count;

407  ps->running_disparity_error_count.value.ull =
408      (unsigned long long)pptr->running_disparity_error_count;

410  ps->loss_of_dword_sync_count.value.ull =
411      (unsigned long long)pptr->loss_of_dword_sync_count;

413  ps->phy_reset_problem_count.value.ull =
414      (unsigned long long)pptr->phy_reset_problem_count;

416  ret = DDI_SUCCESS;
417 fail:
418  mutex_exit(&mpt->m_mutex);

420  return (ret);
421 }

423 void
424 mptsas_destroy_phy_stats(mptsas_t *mpt)
425 {
426     smhba_info_t    *phyp;
427     int              i = 0;

429     ASSERT(mpt != NULL);

431     for (i = 0; i < mpt->m_num_phys; i++) {
432         phyp = &mpt->m_phy_info[i].smhba_info;
433         if (phyp == NULL) {
434             continue;
435         }

437         mutex_enter(&phyp->phy_mutex);
438         if (phyp->phy_stats != NULL) {
439             kstat_delete(phyp->phy_stats);
440             phyp->phy_stats = NULL;
441         }
442         mutex_exit(&phyp->phy_mutex);
443     }
444 }

446 int
447 mptsas_smhba_phy_init(mptsas_t *mpt)
448 {
449     int              i = 0;
450     int              rval = DDI_SUCCESS;
451     uint32_t         page_address;

453     ASSERT(mutex_owned(&mpt->m_mutex));

455     for (i = 0; i < mpt->m_num_phys; i++) {
456         page_address =
457             (MPI2_SAS_PHY_PGAD_FORM_PHY_NUMBER |

```

```

458         (MPI2_SAS_PHY_PGAD_PHY_NUMBER_MASK & i));
459         rval = mptsas_get_sas_phy_page0(mpt,
460             page_address, &mpt->m_phy_info[i].smhba_info);
461         if (rval != DDI_SUCCESS) {
462             mptsas_log(mpt, CE_WARN,
463                 "Failed to get sas phy page 0"
464                 " for each phy");
465             return (DDI_FAILURE);
466         }
467         mpt->m_phy_info[i].smhba_info.phy_id = (uint8_t)i;
468         mpt->m_phy_info[i].smhba_info.sas_addr =
469             mpt->un.m_base_wwid + i;
470         mpt->m_phy_info[i].smhba_info.mpt = mpt;
471     }
472     return (DDI_SUCCESS);
473 }

475 int
476 mptsas_smhba_setup(mptsas_t *mpt)
477 {
478     int              sm_hba = 1;
479     char              chiprev, hw_rev[24];
480     char              serial_number[72];
481     int              protocol = 0;

483     mutex_enter(&mpt->m_mutex);
484     if (mptsas_smhba_phy_init(mpt)) {
485         mutex_exit(&mpt->m_mutex);
486         return (DDI_FAILURE);
487     }
488     mutex_exit(&mpt->m_mutex);

490     /* SM-HBA support */
491     mptsas_smhba_add_hba_prop(mpt, DATA_TYPE_INT32, MPTSAS_SMHBA_SUPPORTED,
492         &sm_hba);

494     /* SM-HBA driver version */
495     mptsas_smhba_add_hba_prop(mpt, DATA_TYPE_STRING, MPTSAS_DRV_VERSION,
496         mptsas_driver_rev);

498     /* SM-HBA hardware version */
499     chiprev = 'A' + mpt->m_revid;
500     (void) snprintf(hw_rev, 2, "%s", &chiprev);
501     mptsas_smhba_add_hba_prop(mpt, DATA_TYPE_STRING, MPTSAS_HWARE_VERSION,
502         hw_rev);

504     /* SM-HBA phy number per HBA */
505     mptsas_smhba_add_hba_prop(mpt, DATA_TYPE_INT32, MPTSAS_NUM_PHYS_HBA,
506         &(mpt->m_num_phys));

508     /* SM-HBA protocol support */
509     protocol = SAS_SSP_SUPPORT | SAS_SATA_SUPPORT | SAS_SMP_SUPPORT;
510     mptsas_smhba_add_hba_prop(mpt, DATA_TYPE_INT32,
511         MPTSAS_SUPPORTED_PROTOCOL, &protocol);

513     mptsas_smhba_add_hba_prop(mpt, DATA_TYPE_STRING, MPTSAS_MANUFACTURER,
514         mpt->m_MANU_page0.ChipName);

516     mptsas_smhba_add_hba_prop(mpt, DATA_TYPE_STRING, MPTSAS_MODEL_NAME,
517         mpt->m_MANU_page0.BoardName);

519     /*
520      * VPD data is not available, we make a serial number for this.
521      */

523     (void) sprintf(serial_number, "%s%s%s%s",

```

```
524         mpt->m_MANU_page0.ChipName,  
525         mpt->m_MANU_page0.ChipRevision,  
526         mpt->m_MANU_page0.BoardName,  
527         mpt->m_MANU_page0.BoardAssembly,  
528         mpt->m_MANU_page0.BoardTracerNumber);  
  
530     mptsas_smhba_add_hba_prop(mpt, DATA_TYPE_STRING, MPTSAS_SERIAL_NUMBER,  
531                             &serial_number[0]);  
  
533     return (DDI_SUCCESS);  
534 }  
535 #endif /* ! codereview */
```

new/usr/src/uts/common/sys/scsi/adapters/mpt_sas3/mpi/mpi2.h

1

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

```
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */
21
22 /*
23  * Copyright (c) 2000-2012 LSI Corporation.
24  *
25  * Redistribution and use in source and binary forms of all code within
26  * this file that is exclusively owned by LSI, with or without
27  * modification, is permitted provided that, in addition to the CDDL 1.0
28  * License requirements, the following conditions are met:
29  *
30  * Neither the name of the author nor the names of its contributors may be
31  * used to endorse or promote products derived from this software without
32  * specific prior written permission.
33  *
34  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
35  * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
36  * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
37  * FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
38  * COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
39  * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
40  * BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
41  * OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
42  * AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
43  * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
44  * OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
45  * DAMAGE.
46 */
47
48 /*
49  * Name: mpi2.h
50  * Title: MPI Message independent structures and definitions
51  * including System Interface Register Set and
52  * scatter/gather formats.
53  * Creation Date: June 21, 2006
54  *
55  * mpi2.h Version: 02.00.xx
56  *
57  * NOTE: Names (typedefs, defines, etc.) beginning with an MPI25 or Mpi25
58  * prefix are for use only on MPI v2.5 products, and must not be used
59  * with MPI v2.0 products. Unless otherwise noted, names beginning with
60  * MPI2 or Mpi2 are for use with both MPI v2.0 and MPI v2.5 products.
61  */
```

new/usr/src/uts/common/sys/scsi/adapters/mpt_sas3/mpi/mpi2.h

2

```
62 * Version History
63 * -----
64 *
65 * Date      Version  Description
66 * -----
67 * 04-30-07  02.00.00  Corresponds to Fusion-MPT MPI Specification Rev A.
68 * 06-04-07  02.00.01  Bumped MPI2_HEADER_VERSION_UNIT.
69 * 06-26-07  02.00.02  Bumped MPI2_HEADER_VERSION_UNIT.
70 * 08-31-07  02.00.03  Bumped MPI2_HEADER_VERSION_UNIT.
71 *
72 * Moved ReplyPostHostIndex register to offset 0x6C of the
73 * MPI2_SYSTEM_INTERFACE_REGS and modified the define for
74 * MPI2_REPLY_POST_HOST_INDEX_OFFSET.
75 *
76 * Added union of request descriptors.
77 * Added union of reply descriptors.
78 *
79 * Bumped MPI2_HEADER_VERSION_UNIT.
80 * 10-31-07  02.00.04  Added define for MPI2_VERSION_02_00.
81 *
82 * Fixed the size of the FunctionDependent5 field in the
83 * MPI2_DEFAULT_REPLY structure.
84 *
85 * Bumped MPI2_HEADER_VERSION_UNIT.
86 * 12-18-07  02.00.05  Removed the MPI-defined Fault Codes and extended the
87 * product specific codes up to 0xEFFF.
88 *
89 * Added a sixth key value for the WriteSequence register
90 * and changed the flush value to 0x0.
91 *
92 * Added message function codes for Diagnostic Buffer Post
93 * and Diagnostic Release.
94 *
95 * New IOCStatus define: MPI2_IOCSTATUS_DIAGNOSTIC_RELEASED
96 * Moved MPI2_VERSION_UNION from mpi2_ioc.h.
97 *
98 * Bumped MPI2_HEADER_VERSION_UNIT.
99 * 02-29-08  02.00.06  Bumped MPI2_HEADER_VERSION_UNIT.
100 * 03-03-08  02.00.07  Bumped MPI2_HEADER_VERSION_UNIT.
101 * 05-21-08  02.00.08  Bumped MPI2_HEADER_VERSION_UNIT.
102 *
103 * Added #defines for marking a reply descriptor as unused.
104 * Bumped MPI2_HEADER_VERSION_UNIT.
105 * 06-27-08  02.00.09  Bumped MPI2_HEADER_VERSION_UNIT.
106 * 10-02-08  02.00.10  Moved LUN field defines from mpi2_init.h.
107 *
108 * Bumped MPI2_HEADER_VERSION_UNIT.
109 * 01-19-09  02.00.11  Bumped MPI2_HEADER_VERSION_UNIT.
110 * 05-06-09  02.00.12  In all request and reply descriptors, replaced VF_ID
111 * field with MSIxIndex field.
112 *
113 * Removed DevHandle field from
114 * MPI2_SCSI_IO_SUCCESS_REPLY_DESCRIPTOR and made those
115 * bytes reserved.
116 *
117 * Added RAID Accelerator functionality.
118 * Bumped MPI2_HEADER_VERSION_UNIT.
119 * 07-30-09  02.00.13
120 * -----
121 */
122
123 #ifndef MPI2_H
124 #define MPI2_H
125
126 /*****
127  *
128  * MPI Version Definitions
129  *
130  *****/
131
132 #define MPI2_VERSION_MAJOR_MASK      (0xFF00)
133 #define MPI2_VERSION_MAJOR_SHIFT    (8)
134 #define MPI2_VERSION_MINOR_MASK     (0x00FF)
135 #define MPI2_VERSION_MINOR_SHIFT    (0)
136
137 /* major version for all MPI v2.x */
138 #define MPI2_VERSION_MAJOR          (0x02)
139
140 /* minor version for MPI v2.0 compatible products */
141 #define MPI2_VERSION_MINOR          (0x00)
```

```

128 #define MPI2_VERSION ((MPI2_VERSION_MAJOR << MPI2_VERSION_MAJOR_SHIFT) | \
129                      MPI2_VERSION_MINOR)
130 #define MPI2_VERSION_02_00 (0x0200)

133 /* minor version for MPI v2.5 compatible products */
134 #define MPI25_VERSION_MINOR (0x05)
135 #define MPI25_VERSION ((MPI2_VERSION_MAJOR << MPI2_VERSION_MAJOR_SHIFT) | \
136                      MPI25_VERSION_MINOR)
137 #define MPI2_VERSION_02_05 (0x0205)

140 /* Unit and Dev versioning for this MPI header set */
141 #define MPI2_HEADER_VERSION_UNIT (0x12)
142 #define MPI2_HEADER_VERSION_DEV (0x00)
143 #define MPI2_HEADER_VERSION_UNIT_MASK (0xFF00)
144 #define MPI2_HEADER_VERSION_UNIT_SHIFT (8)
145 #define MPI2_HEADER_VERSION_DEV_MASK (0x00FF)
146 #define MPI2_HEADER_VERSION_DEV_SHIFT (0)
147 #define MPI2_HEADER_VERSION ((MPI2_HEADER_VERSION_UNIT << 8) | MPI2_HEADER_VERSION_DEV_SHIFT)

150 /*****
151 *
152 *      IOC State Definitions
153 *
154 *****/

156 #define MPI2_IOC_STATE_RESET (0x00000000)
157 #define MPI2_IOC_STATE_READY (0x10000000)
158 #define MPI2_IOC_STATE_OPERATIONAL (0x20000000)
159 #define MPI2_IOC_STATE_FAULT (0x40000000)

161 #define MPI2_IOC_STATE_MASK (0xF0000000)
162 #define MPI2_IOC_STATE_SHIFT (28)

164 /* Fault state range for product specific codes */
165 #define MPI2_FAULT_PRODUCT_SPECIFIC_MIN (0x0000)
166 #define MPI2_FAULT_PRODUCT_SPECIFIC_MAX (0xEFFF)

169 /*****
170 *
171 *      System Interface Register Definitions
172 *
173 *****/

175 typedef volatile struct _MPI2_SYSTEM_INTERFACE_REGS
176 {
177     U32      Doorbell;          /* 0x00 */
178     U32      WriteSequence;     /* 0x04 */
179     U32      HostDiagnostic;    /* 0x08 */
180     U32      Reserved1;        /* 0x0C */
181     U32      DiagRWDData;       /* 0x10 */
182     U32      DiagRWAddressLow;  /* 0x14 */
183     U32      DiagRWAddressHigh; /* 0x18 */
184     U32      Reserved2[5];      /* 0x1C */
185     U32      HostInterruptStatus; /* 0x30 */
186     U32      HostInterruptMask; /* 0x34 */
187     U32      DCRData;          /* 0x38 */
188     U32      DCRAddress;       /* 0x3C */
189     U32      Reserved3[2];      /* 0x40 */
190     U32      ReplyFreeHostIndex; /* 0x48 */
191     U32      Reserved4[8];      /* 0x4C */
192     U32      ReplyPostHostIndex; /* 0x6C */
193     U32      Reserved5;        /* 0x70 */

```

```

194     U32      HCBSize;          /* 0x74 */
195     U32      HCBAddressLow;    /* 0x78 */
196     U32      HCBAddressHigh;  /* 0x7C */
197     U32      Reserved6[16];    /* 0x80 */
198     U32      RequestDescriptorPostLow; /* 0xC0 */
199     U32      RequestDescriptorPostHigh; /* 0xC4 */
200     U32      Reserved7[14];    /* 0xC8 */
201     U32      Reserved8[128];   /* 0x100 */
202     U32      Reserved10[3];    /* 0x300 */
203     U32      SuppReplyPostHostIndex[32]; /* 0x30C */
204 } MPI2_SYSTEM_INTERFACE_REGS, MPI2_POINTER PTR_MPI2_SYSTEM_INTERFACE_REGS,
205   Mpi2SystemInterfaceRegs_t, MPI2_POINTER pMpi2SystemInterfaceRegs_t;

207 /*
208 * Defines for working with the Doorbell register.
209 */
210 #define MPI2_DOORBELL_OFFSET (0x00000000)

212 /* IOC --> System values */
213 #define MPI2_DOORBELL_USED (0x08000000)
214 #define MPI2_DOORBELL_WHO_INIT_MASK (0x07000000)
215 #define MPI2_DOORBELL_WHO_INIT_SHIFT (24)
216 #define MPI2_DOORBELL_FAULT_CODE_MASK (0x0000FFFF)
217 #define MPI2_DOORBELL_DATA_MASK (0x0000FFFF)

219 /* System --> IOC values */
220 #define MPI2_DOORBELL_FUNCTION_MASK (0xFF000000)
221 #define MPI2_DOORBELL_FUNCTION_SHIFT (24)
222 #define MPI2_DOORBELL_ADD_DWORDS_MASK (0x00FF0000)
223 #define MPI2_DOORBELL_ADD_DWORDS_SHIFT (16)

226 /*
227 * Defines for the WriteSequence register
228 */
229 #define MPI2_WRITE_SEQUENCE_OFFSET (0x00000004)
230 #define MPI2_WRSEQ_KEY_VALUE_MASK (0x0000000F)
231 #define MPI2_WRSEQ_FLUSH_KEY_VALUE (0x0)
232 #define MPI2_WRSEQ_1ST_KEY_VALUE (0xF)
233 #define MPI2_WRSEQ_2ND_KEY_VALUE (0x4)
234 #define MPI2_WRSEQ_3RD_KEY_VALUE (0xB)
235 #define MPI2_WRSEQ_4TH_KEY_VALUE (0x2)
236 #define MPI2_WRSEQ_5TH_KEY_VALUE (0x7)
237 #define MPI2_WRSEQ_6TH_KEY_VALUE (0xD)

239 /*
240 * Defines for the HostDiagnostic register
241 */
242 #define MPI2_HOST_DIAGNOSTIC_OFFSET (0x00000008)

244 #define MPI2_DIAG_BOOT_DEVICE_SELECT_MASK (0x00001800)
245 #define MPI2_DIAG_BOOT_DEVICE_SELECT_DEFAULT (0x00000000)
246 #define MPI2_DIAG_BOOT_DEVICE_SELECT_HCDW (0x00000800)

248 #define MPI2_DIAG_CLEAR_FLASH_BAD_SIG (0x00000400)
249 #define MPI2_DIAG_FORCE_HCB_ON_RESET (0x00000200)
250 #define MPI2_DIAG_HCB_MODE (0x00000100)
251 #define MPI2_DIAG_DIAG_WRITE_ENABLE (0x00000080)
252 #define MPI2_DIAG_FLASH_BAD_SIG (0x00000040)
253 #define MPI2_DIAG_RESET_HISTORY (0x00000020)
254 #define MPI2_DIAG_DIAG_RW_ENABLE (0x00000010)
255 #define MPI2_DIAG_RESET_ADAPTER (0x00000004)
256 #define MPI2_DIAG_HOLD_IOC_RESET (0x00000002)

258 /*
259 * Offsets for DiagRWDData and address

```



```

260 */
261 #define MPI2_DIAG_RW_DATA_OFFSET          (0x00000010)
262 #define MPI2_DIAG_RW_ADDRESS_LOW_OFFSET  (0x00000014)
263 #define MPI2_DIAG_RW_ADDRESS_HIGH_OFFSET (0x00000018)

265 /*
266  * Defines for the HostInterruptStatus register
267  */
268 #define MPI2_HOST_INTERRUPT_STATUS_OFFSET (0x00000030)
269 #define MPI2_HIS_SYS2IOC_DB_STATUS        (0x80000000)
270 #define MPI2_HIS_IOP_DOORBELL_STATUS      MPI2_HIS_SYS2IOC_DB_STATUS
271 #define MPI2_HIS_RESET_IRQ_STATUS         (0x40000000)
272 #define MPI2_HIS_REPLY_DESCRIPTOR_INTERRUPT (0x00000008)
273 #define MPI2_HIS_IOC2SYS_DB_STATUS        (0x00000001)
274 #define MPI2_HIS_DOORBELL_INTERRUPT      MPI2_HIS_IOC2SYS_DB_STATUS

276 /*
277  * Defines for the HostInterruptMask register
278  */
279 #define MPI2_HOST_INTERRUPT_MASK_OFFSET    (0x00000034)
280 #define MPI2_HIM_RESET_IRQ_MASK           (0x40000000)
281 #define MPI2_HIM_REPLY_INT_MASK           (0x00000008)
282 #define MPI2_HIM_REPLY_MASK               MPI2_HIM_REPLY_INT_MASK
283 #define MPI2_HIM_IOC2SYS_DB_MASK          (0x00000001)
284 #define MPI2_HIM_DIM                      MPI2_HIM_IOC2SYS_DB_MASK

286 /*
287  * Offsets for DCRData and address
288  */
289 #define MPI2_DCR_DATA_OFFSET              (0x00000038)
290 #define MPI2_DCR_ADDRESS_OFFSET           (0x0000003C)

292 /*
293  * Offset for the Reply Free Queue
294  */
295 #define MPI2_REPLY_FREE_HOST_INDEX_OFFSET (0x00000048)

297 /*
298  * Offset for the Reply Descriptor Post Queue
299  */
300 #define MPI2_REPLY_POST_HOST_INDEX_OFFSET (0x0000006C)
301 #define MPI2_REPLY_POST_HOST_INDEX_MASK  (0x0FFFFFFF)
302 #define MPI2_RPHI_MSIX_INDEX_MASK        (0xFF000000)
303 #define MPI2_RPHI_MSIX_INDEX_SHIFT       (24)

305 /*
306  * Defines for the HCBSIZE and address
307  */
308 #define MPI2_HCB_SIZE_OFFSET              (0x00000074)
309 #define MPI2_HCB_SIZE_SIZE_MASK           (0xFFFFF000)
310 #define MPI2_HCB_SIZE_HCB_ENABLE          (0x00000001)

312 #define MPI2_HCB_ADDRESS_LOW_OFFSET        (0x00000078)
313 #define MPI2_HCB_ADDRESS_HIGH_OFFSET      (0x0000007C)

315 /*
316  * Offsets for the Request Queue
317  */
318 #define MPI2_REQUEST_DESCRIPTOR_POST_LOW_OFFSET (0x000000C0)
319 #define MPI2_REQUEST_DESCRIPTOR_POST_HIGH_OFFSET (0x000000C4)

321 /*
322  * Offset for the Supplementary Host Index Base
323  * For use with more than 8 MSI-X interrupts.
324  */
325 #define MPI2_SUP_REPLY_POST_HOST_INDEX_OFFSET (0x0000030C)

```

```

328 /*****
329  *
330  *      Message Descriptors
331  *
332  *****/

334 /* Request Descriptors */

336 /* Default Request Descriptor */
337 typedef struct _MPI2_DEFAULT_REQUEST_DESCRIPTOR
338 {
339     U8          RequestFlags;          /* 0x00 */
340     U8          MSIXIndex;             /* 0x01 */
341     U16         SMID;                  /* 0x02 */
342     U16         LMID;                  /* 0x04 */
343     U16         DescriptorTypeDependent; /* 0x06 */
344 } MPI2_DEFAULT_REQUEST_DESCRIPTOR,
345 MPI2_POINTER PTR_MPI2_DEFAULT_REQUEST_DESCRIPTOR,
346 Mpi2DefaultRequestDescriptor_t, MPI2_POINTER pMpi2DefaultRequestDescriptor_t;

348 /* defines for the RequestFlags field */
349 #define MPI2_REQ_DESCRIPTOR_FLAGS_TYPE_MASK          (0x0E)
350 #define MPI2_REQ_DESCRIPTOR_FLAGS_SCSI_IO           (0x00)
351 #define MPI2_REQ_DESCRIPTOR_FLAGS_SCSI_TARGET       (0x02)
352 #define MPI2_REQ_DESCRIPTOR_FLAGS_HIGH_PRIORITY    (0x06)
353 #define MPI2_REQ_DESCRIPTOR_FLAGS_DEFAULT_TYPE     (0x08)
354 #define MPI2_REQ_DESCRIPTOR_FLAGS_RAID_ACCELERATOR (0x0A)
355 #define MPI25_REQ_DESCRIPTOR_FLAGS_FAST_PATH_SCSI_IO (0x0C)

357 #define MPI2_REQ_DESCRIPTOR_FLAGS_IOC_FIFO_MARKER (0x01)

360 /* High Priority Request Descriptor */
361 typedef struct _MPI2_HIGH_PRIORITY_REQUEST_DESCRIPTOR
362 {
363     U8          RequestFlags;          /* 0x00 */
364     U8          MSIXIndex;             /* 0x01 */
365     U16         SMID;                  /* 0x02 */
366     U16         LMID;                  /* 0x04 */
367     U16         Reserved1;             /* 0x06 */
368 } MPI2_HIGH_PRIORITY_REQUEST_DESCRIPTOR,
369 MPI2_POINTER PTR_MPI2_HIGH_PRIORITY_REQUEST_DESCRIPTOR,
370 Mpi2HighPriorityRequestDescriptor_t,
371 MPI2_POINTER pMpi2HighPriorityRequestDescriptor_t;

374 /* SCSI IO Request Descriptor */
375 typedef struct _MPI2_SCSI_IO_REQUEST_DESCRIPTOR
376 {
377     U8          RequestFlags;          /* 0x00 */
378     U8          MSIXIndex;             /* 0x01 */
379     U16         SMID;                  /* 0x02 */
380     U16         LMID;                  /* 0x04 */
381     U16         DevHandle;             /* 0x06 */
382 } MPI2_SCSI_IO_REQUEST_DESCRIPTOR,
383 MPI2_POINTER PTR_MPI2_SCSI_IO_REQUEST_DESCRIPTOR,
384 Mpi2SCSIIORequestDescriptor_t, MPI2_POINTER pMpi2SCSIIORequestDescriptor_t;

387 /* SCSI Target Request Descriptor */
388 typedef struct _MPI2_SCSI_TARGET_REQUEST_DESCRIPTOR
389 {
390     U8          RequestFlags;          /* 0x00 */
391     U8          MSIXIndex;             /* 0x01 */

```

```

392     U16          SMID;                /* 0x02 */
393     U16          LMID;                /* 0x04 */
394     U16          IoIndex;             /* 0x06 */
395 } MPI2_SCSI_TARGET_REQUEST_DESCRIPTOR,
396 MPI2_POINTER PTR_MPI2_SCSI_TARGET_REQUEST_DESCRIPTOR,
397 Mpi2SCSITargetRequestDescriptor_t,
398 MPI2_POINTER pMpi2SCSITargetRequestDescriptor_t;

401 /* RAID Accelerator Request Descriptor */
402 typedef struct _MPI2_RAID_ACCEL_REQUEST_DESCRIPTOR
403 {
404     U8          RequestFlags;         /* 0x00 */
405     U8          MSIxIndex;            /* 0x01 */
406     U16         SMID;                /* 0x02 */
407     U16         LMID;                /* 0x04 */
408     U16         Reserved;            /* 0x06 */
409 } MPI2_RAID_ACCEL_REQUEST_DESCRIPTOR,
410 MPI2_POINTER PTR_MPI2_RAID_ACCEL_REQUEST_DESCRIPTOR,
411 Mpi2RAIDAcceleratorRequestDescriptor_t,
412 MPI2_POINTER pMpi2RAIDAcceleratorRequestDescriptor_t;

415 /* Fast Path SCSI IO Request Descriptor */
416 typedef MPI2_SCSI_IO_REQUEST_DESCRIPTOR
417 MPI25_FP_SCSI_IO_REQUEST_DESCRIPTOR,
418 MPI2_POINTER PTR_MPI25_FP_SCSI_IO_REQUEST_DESCRIPTOR,
419 Mpi25FastPathSCSIIORequestDescriptor_t,
420 MPI2_POINTER pMpi25FastPathSCSIIORequestDescriptor_t;

423 /* union of Request Descriptors */
424 typedef union _MPI2_REQUEST_DESCRIPTOR_UNION
425 {
426     MPI2_DEFAULT_REQUEST_DESCRIPTOR      Default;
427     MPI2_HIGH_PRIORITY_REQUEST_DESCRIPTOR HighPriority;
428     MPI2_SCSI_IO_REQUEST_DESCRIPTOR      SCSIIO;
429     MPI2_SCSI_TARGET_REQUEST_DESCRIPTOR  SCSITarget;
430     MPI2_RAID_ACCEL_REQUEST_DESCRIPTOR   RAIDAccelerator;
431     MPI25_FP_SCSI_IO_REQUEST_DESCRIPTOR  FastPathSCSIIO;
432     U64                                  Words;
433 } MPI2_REQUEST_DESCRIPTOR_UNION, MPI2_POINTER PTR_MPI2_REQUEST_DESCRIPTOR_UNION,
434 Mpi2RequestDescriptorUnion_t, MPI2_POINTER pMpi2RequestDescriptorUnion_t;

437 /* Reply Descriptors */

439 /* Default Reply Descriptor */
440 typedef struct _MPI2_DEFAULT_REPLY_DESCRIPTOR
441 {
442     U8          ReplyFlags;           /* 0x00 */
443     U8          MSIxIndex;            /* 0x01 */
444     U16         DescriptorTypeDependent1; /* 0x02 */
445     U32         DescriptorTypeDependent2; /* 0x04 */
446 } MPI2_DEFAULT_REPLY_DESCRIPTOR, MPI2_POINTER PTR_MPI2_DEFAULT_REPLY_DESCRIPTOR,
447 Mpi2DefaultReplyDescriptor_t, MPI2_POINTER pMpi2DefaultReplyDescriptor_t;

449 /* defines for the ReplyFlags field */
450 #define MPI2_RPY_DESCRIPTOR_FLAGS_TYPE_MASK (0x0F)
451 #define MPI2_RPY_DESCRIPTOR_FLAGS_SCSI_IO_SUCCESS (0x00)
452 #define MPI2_RPY_DESCRIPTOR_FLAGS_ADDRESS_REPLY (0x01)
453 #define MPI2_RPY_DESCRIPTOR_FLAGS_TARGETASSIST_SUCCESS (0x02)
454 #define MPI2_RPY_DESCRIPTOR_FLAGS_TARGET_COMMAND_BUFFER (0x03)
455 #define MPI2_RPY_DESCRIPTOR_FLAGS_RAID_ACCELERATOR_SUCCESS (0x05)
456 #define MPI25_RPY_DESCRIPTOR_FLAGS_FAST_PATH_SCSI_IO_SUCCESS (0x06)
457 #define MPI2_RPY_DESCRIPTOR_FLAGS_UNUSED (0x0F)

```

```

459 /* values for marking a reply descriptor as unused */
460 #define MPI2_RPY_DESCRIPTOR_UNUSED_WORD0_MARK (0xFFFFFFFF)
461 #define MPI2_RPY_DESCRIPTOR_UNUSED_WORD1_MARK (0xFFFFFFFF)

463 /* Address Reply Descriptor */
464 typedef struct _MPI2_ADDRESS_REPLY_DESCRIPTOR
465 {
466     U8          ReplyFlags;           /* 0x00 */
467     U8          MSIxIndex;            /* 0x01 */
468     U16         SMID;                /* 0x02 */
469     U32         ReplyFrameAddress;    /* 0x04 */
470 } MPI2_ADDRESS_REPLY_DESCRIPTOR, MPI2_POINTER PTR_MPI2_ADDRESS_REPLY_DESCRIPTOR,
471 Mpi2AddressReplyDescriptor_t, MPI2_POINTER pMpi2AddressReplyDescriptor_t;

473 #define MPI2_ADDRESS_REPLY_SMID_INVALID (0x00)

476 /* SCSI IO Success Reply Descriptor */
477 typedef struct _MPI2_SCSI_IO_SUCCESS_REPLY_DESCRIPTOR
478 {
479     U8          ReplyFlags;           /* 0x00 */
480     U8          MSIxIndex;            /* 0x01 */
481     U16         SMID;                /* 0x02 */
482     U16         TaskTag;              /* 0x04 */
483     U16         Reserved1;            /* 0x06 */
484 } MPI2_SCSI_IO_SUCCESS_REPLY_DESCRIPTOR,
485 MPI2_POINTER PTR_MPI2_SCSI_IO_SUCCESS_REPLY_DESCRIPTOR,
486 Mpi2SCSIIOSuccessReplyDescriptor_t,
487 MPI2_POINTER pMpi2SCSIIOSuccessReplyDescriptor_t;

490 /* TargetAssist Success Reply Descriptor */
491 typedef struct _MPI2_TARGETASSIST_SUCCESS_REPLY_DESCRIPTOR
492 {
493     U8          ReplyFlags;           /* 0x00 */
494     U8          MSIxIndex;            /* 0x01 */
495     U16         SMID;                /* 0x02 */
496     U8          SequenceNumber;       /* 0x04 */
497     U8          Reserved1;            /* 0x05 */
498     U16         IoIndex;              /* 0x06 */
499 } MPI2_TARGETASSIST_SUCCESS_REPLY_DESCRIPTOR,
500 MPI2_POINTER PTR_MPI2_TARGETASSIST_SUCCESS_REPLY_DESCRIPTOR,
501 Mpi2TargetAssistSuccessReplyDescriptor_t,
502 MPI2_POINTER pMpi2TargetAssistSuccessReplyDescriptor_t;

505 /* Target Command Buffer Reply Descriptor */
506 typedef struct _MPI2_TARGET_COMMAND_BUFFER_REPLY_DESCRIPTOR
507 {
508     U8          ReplyFlags;           /* 0x00 */
509     U8          MSIxIndex;            /* 0x01 */
510     U8          VP_ID;                /* 0x02 */
511     U8          Flags;                /* 0x03 */
512     U16         InitiatorDevHandle;   /* 0x04 */
513     U16         IoIndex;              /* 0x06 */
514 } MPI2_TARGET_COMMAND_BUFFER_REPLY_DESCRIPTOR,
515 MPI2_POINTER PTR_MPI2_TARGET_COMMAND_BUFFER_REPLY_DESCRIPTOR,
516 Mpi2TargetCommandBufferReplyDescriptor_t,
517 MPI2_POINTER pMpi2TargetCommandBufferReplyDescriptor_t;

519 /* defines for Flags field */
520 #define MPI2_RPY_DESCRIPTOR_TCB_FLAGS_PHYNUM_MASK (0x3F)

523 /* RAID Accelerator Success Reply Descriptor */

```

```

524 typedef struct _MPI2_RAID_ACCELERATOR_SUCCESS_REPLY_DESCRIPTOR
525 {
526     U8      ReplyFlags;          /* 0x00 */
527     U8      MSixIndex;          /* 0x01 */
528     U16     SMID;               /* 0x02 */
529     U32     Reserved;          /* 0x04 */
530 } MPI2_RAID_ACCELERATOR_SUCCESS_REPLY_DESCRIPTOR,
531 MPI2_POINTER PTR_MPI2_RAID_ACCELERATOR_SUCCESS_REPLY_DESCRIPTOR,
532 MPI2RAIDAcceleratorSuccessReplyDescriptor_t,
533 MPI2_POINTER pMPI2RAIDAcceleratorSuccessReplyDescriptor_t;

536 /* Fast Path SCSI IO Success Reply Descriptor */
537 typedef MPI2_SCSI_IO_SUCCESS_REPLY_DESCRIPTOR
538 MPI25_FP_SCSI_IO_SUCCESS_REPLY_DESCRIPTOR,
539 MPI2_POINTER PTR_MPI25_FP_SCSI_IO_SUCCESS_REPLY_DESCRIPTOR,
540 MPI25FastPathSCSIIOSuccessReplyDescriptor_t,
541 MPI2_POINTER pMPI25FastPathSCSIIOSuccessReplyDescriptor_t;

544 /* union of Reply Descriptors */
545 typedef union _MPI2_REPLY_DESCRIPTOR_UNION
546 {
547     MPI2_DEFAULT_REPLY_DESCRIPTOR      Default;
548     MPI2_ADDRESS_REPLY_DESCRIPTOR      AddressReply;
549     MPI2_SCSI_IO_SUCCESS_REPLY_DESCRIPTOR SCSIIOSuccess;
550     MPI2_TARGETASSIST_SUCCESS_REPLY_DESCRIPTOR TargetAssistSuccess;
551     MPI2_TARGET_COMMAND_BUFFER_REPLY_DESCRIPTOR TargetCommandBuffer;
552     MPI2_RAID_ACCELERATOR_SUCCESS_REPLY_DESCRIPTOR RAIDAcceleratorSuccess;
553     MPI25_FP_SCSI_IO_SUCCESS_REPLY_DESCRIPTOR FastPathSCSIIOSuccess;
554     U64      Words;
555 } MPI2_REPLY_DESCRIPTOR_UNION, MPI2_POINTER PTR_MPI2_REPLY_DESCRIPTOR_UNION,
556 MPI2ReplyDescriptorsUnion_t, MPI2_POINTER pMPI2ReplyDescriptorsUnion_t;

560 /*****
561 *
562 *      Message Functions
563 *      0x80 -> 0x8F reserved for private message use per product
564 *
565 *
566 *****/

568 #define MPI2_FUNCTION_SCSI_IO_REQUEST      (0x00) /* SCSI IO */
569 #define MPI2_FUNCTION_SCSI_TASK_MGMT      (0x01) /* SCSI Task Manageme
570 #define MPI2_FUNCTION_IOC_INIT            (0x02) /* IOC Init */
571 #define MPI2_FUNCTION_IOC_FACTS          (0x03) /* IOC Facts */
572 #define MPI2_FUNCTION_CONFIG              (0x04) /* Configuration */
573 #define MPI2_FUNCTION_PORT_FACTS         (0x05) /* Port Facts */
574 #define MPI2_FUNCTION_PORT_ENABLE         (0x06) /* Port Enable */
575 #define MPI2_FUNCTION_EVENT_NOTIFICATION (0x07) /* Event Notification
576 #define MPI2_FUNCTION_EVENT_ACK           (0x08) /* Event Acknowledge
577 #define MPI2_FUNCTION_FW_DOWNLOAD         (0x09) /* FW Download */
578 #define MPI2_FUNCTION_TARGET_ASSIST       (0x0B) /* Target Assist */
579 #define MPI2_FUNCTION_TARGET_STATUS_SEND  (0x0C) /* Target Status Send
580 #define MPI2_FUNCTION_TARGET_MODE_ABORT   (0x0D) /* Target Mode Abort
581 #define MPI2_FUNCTION_FW_UPLOAD           (0x12) /* FW Upload */
582 #define MPI2_FUNCTION_RAID_ACTION          (0x15) /* RAID Action */
583 #define MPI2_FUNCTION_RAID_SCSI_IO_PASSTHROUGH (0x16) /* SCSI IO RAID Passt
584 #define MPI2_FUNCTION_TOOLBOX             (0x17) /* Toolbox */
585 #define MPI2_FUNCTION_SCSI_ENCLOSURE_PROCESSOR (0x18) /* SCSI Enclosure Pro
586 #define MPI2_FUNCTION_SMP_PASSTHROUGH     (0x1A) /* SMP Passthrough */
587 #define MPI2_FUNCTION_SAS_IO_UNIT_CONTROL (0x1B) /* SAS IO Unit Contro
588 #define MPI2_FUNCTION_SATA_PASSTHROUGH    (0x1C) /* SATA Passthrough */
589 #define MPI2_FUNCTION_DIAG_BUFFER_POST    (0x1D) /* Diagnostic Buffer

```

```

590 #define MPI2_FUNCTION_DIAG_RELEASE        (0x1E) /* Diagnostic Release
591 #define MPI2_FUNCTION_TARGET_CMD_BUF_BASE_POST (0x24) /* Target Command Buf
592 #define MPI2_FUNCTION_TARGET_CMD_BUF_LIST_POST (0x25) /* Target Command Buf
593 #define MPI2_FUNCTION_RAID_ACCELERATOR    (0x2C) /* RAID Accelerator */
594 #define MPI2_FUNCTION_HOST_BASED_DISCOVERY_ACTION (0x2F) /* Host Based Discove
595 #define MPI2_FUNCTION_PWR_MGMT_CONTROL    (0x30) /* Power Management C
596 #define MPI2_FUNCTION_MIN_PRODUCT_SPECIFIC (0xF0) /* beginning of produ
597 #define MPI2_FUNCTION_MAX_PRODUCT_SPECIFIC (0xFF) /* end of product-spe

601 /* Doorbell functions */
602 #define MPI2_FUNCTION_IOC_MESSAGE_UNIT_RESET (0x40)
603 #define MPI2_FUNCTION_HANDSHAKE            (0x42)

606 /*****
607 *
608 *      IOC Status Values
609 *
610 *****/

612 /* mask for IOCStatus status value */
613 #define MPI2_IOCSTATUS_MASK                (0x7FFF)

615 /*****
616 *      Common IOCStatus values for all replies
617 *****/

619 #define MPI2_IOCSTATUS_SUCCESS              (0x0000)
620 #define MPI2_IOCSTATUS_INVALID_FUNCTION    (0x0001)
621 #define MPI2_IOCSTATUS_BUSY                (0x0002)
622 #define MPI2_IOCSTATUS_INVALID_SGL        (0x0003)
623 #define MPI2_IOCSTATUS_INTERNAL_ERROR      (0x0004)
624 #define MPI2_IOCSTATUS_INVALID_VPID       (0x0005)
625 #define MPI2_IOCSTATUS_INSUFFICIENT_RESOURCES (0x0006)
626 #define MPI2_IOCSTATUS_INVALID_FIELD      (0x0007)
627 #define MPI2_IOCSTATUS_INVALID_STATE      (0x0008)
628 #define MPI2_IOCSTATUS_OP_STATE_NOT_SUPPORTED (0x0009)

630 /*****
631 *      Config IOCStatus values
632 *****/

634 #define MPI2_IOCSTATUS_CONFIG_INVALID_ACTION (0x0020)
635 #define MPI2_IOCSTATUS_CONFIG_INVALID_TYPE  (0x0021)
636 #define MPI2_IOCSTATUS_CONFIG_INVALID_PAGE  (0x0022)
637 #define MPI2_IOCSTATUS_CONFIG_INVALID_DATA  (0x0023)
638 #define MPI2_IOCSTATUS_CONFIG_NO_DEFAULTS   (0x0024)
639 #define MPI2_IOCSTATUS_CONFIG_CANT_COMMIT   (0x0025)

641 /*****
642 *      SCSI IO Reply
643 *****/

645 #define MPI2_IOCSTATUS_SCSI_RECOVERED_ERROR (0x0040)
646 #define MPI2_IOCSTATUS_SCSI_INVALID_DEVHANDLE (0x0042)
647 #define MPI2_IOCSTATUS_SCSI_DEVICE_NOT_THERE (0x0043)
648 #define MPI2_IOCSTATUS_SCSI_DATA_OVERRUN    (0x0044)
649 #define MPI2_IOCSTATUS_SCSI_DATA_UNDEERRUN  (0x0045)
650 #define MPI2_IOCSTATUS_SCSI_IO_DATA_ERROR   (0x0046)
651 #define MPI2_IOCSTATUS_SCSI_PROTOCOL_ERROR  (0x0047)
652 #define MPI2_IOCSTATUS_SCSI_TASK_TERMINATED (0x0048)
653 #define MPI2_IOCSTATUS_SCSI_RESIDUAL_MISMATCH (0x0049)
654 #define MPI2_IOCSTATUS_SCSI_TASK_MGMT_FAILED (0x004A)
655 #define MPI2_IOCSTATUS_SCSI_IOC_TERMINATED  (0x004B)

```

```

656 #define MPI2_IOCSTATUS_SCSI_EXT_TERMINATED      (0x004C)

658 /*****
659 * For use by SCSI Initiator and SCSI Target end-to-end data protection
660 *****/

662 #define MPI2_IOCSTATUS_EEDP_GUARD_ERROR          (0x004D)
663 #define MPI2_IOCSTATUS_EEDP_REF_TAG_ERROR        (0x004E)
664 #define MPI2_IOCSTATUS_EEDP_APP_TAG_ERROR        (0x004F)

666 /*****
667 * SCSI Target values
668 *****/

670 #define MPI2_IOCSTATUS_TARGET_INVALID_IO_INDEX   (0x0062)
671 #define MPI2_IOCSTATUS_TARGET_ABORTED            (0x0063)
672 #define MPI2_IOCSTATUS_TARGET_NO_CONN_RETRYABLE (0x0064)
673 #define MPI2_IOCSTATUS_TARGET_NO_CONNECTION      (0x0065)
674 #define MPI2_IOCSTATUS_TARGET_XFER_COUNT_MISMATCH (0x006A)
675 #define MPI2_IOCSTATUS_TARGET_DATA_OFFSET_ERROR (0x006D)
676 #define MPI2_IOCSTATUS_TARGET_TOO_MUCH_WRITE_DATA (0x006E)
677 #define MPI2_IOCSTATUS_TARGET_IU_TOO_SHORT      (0x006F)
678 #define MPI2_IOCSTATUS_TARGET_ACK_NAK_TIMEOUT    (0x0070)
679 #define MPI2_IOCSTATUS_TARGET_NAK_RECEIVED       (0x0071)

681 /*****
682 * Serial Attached SCSI values
683 *****/

685 #define MPI2_IOCSTATUS_SAS_SMP_REQUEST_FAILED    (0x0090)
686 #define MPI2_IOCSTATUS_SAS_SMP_DATA_OVERRUN      (0x0091)

688 /*****
689 * Diagnostic Buffer Post / Diagnostic Release values
690 *****/

692 #define MPI2_IOCSTATUS_DIAGNOSTIC_RELEASED        (0x00A0)

694 /*****
695 * RAID Accelerator values
696 *****/

698 #define MPI2_IOCSTATUS_RAID_ACCEL_ERROR           (0x00B0)

700 /*****
701 * IOCStatus flag to indicate that log info is available
702 *****/

704 #define MPI2_IOCSTATUS_FLAG_LOG_INFO_AVAILABLE   (0x8000)

706 /*****
707 * IOCLogInfo Types
708 *****/

710 #define MPI2_IOCLOGINFO_TYPE_MASK                 (0xF0000000)
711 #define MPI2_IOCLOGINFO_TYPE_SHIFT                (28)
712 #define MPI2_IOCLOGINFO_TYPE_NONE                 (0x0)
713 #define MPI2_IOCLOGINFO_TYPE_SCSI                 (0x1)
714 #define MPI2_IOCLOGINFO_TYPE_FC                   (0x2)
715 #define MPI2_IOCLOGINFO_TYPE_SAS                  (0x3)
716 #define MPI2_IOCLOGINFO_TYPE_ISCSI                (0x4)
717 #define MPI2_IOCLOGINFO_LOG_DATA_MASK             (0x0FFFFFFF)

720 /*****
721 *

```

```

722 *      Standard Message Structures
723 *
724 *****/

726 /*****
727 * Request Message Header for all request messages
728 *****/

730 typedef struct _MPI2_REQUEST_HEADER
731 {
732     U16      FunctionDependent1;      /* 0x00 */
733     U8        ChainOffset;             /* 0x02 */
734     U8        Function;                /* 0x03 */
735     U16      FunctionDependent2;      /* 0x04 */
736     U8        FunctionDependent3;     /* 0x06 */
737     U8        MsgFlags;                /* 0x07 */
738     U8        VP_ID;                   /* 0x08 */
739     U8        VF_ID;                   /* 0x09 */
740     U16      Reserved1;                /* 0x0A */
741 } MPI2_REQUEST_HEADER, MPI2_POINTER PTR_MPI2_REQUEST_HEADER,
742   MPI2RequestHeader_t, MPI2_POINTER pMPI2RequestHeader_t;

745 /*****
746 * Default Reply
747 *****/

749 typedef struct _MPI2_DEFAULT_REPLY
750 {
751     U16      FunctionDependent1;      /* 0x00 */
752     U8        MsgLength;               /* 0x02 */
753     U8        Function;                /* 0x03 */
754     U16      FunctionDependent2;      /* 0x04 */
755     U8        FunctionDependent3;     /* 0x06 */
756     U8        MsgFlags;                /* 0x07 */
757     U8        VP_ID;                   /* 0x08 */
758     U8        VF_ID;                   /* 0x09 */
759     U16      Reserved1;                /* 0x0A */
760     U16      FunctionDependent5;      /* 0x0C */
761     U16      IOCStatus;                /* 0x0E */
762     U32      IOCLogInfo;               /* 0x10 */
763 } MPI2_DEFAULT_REPLY, MPI2_POINTER PTR_MPI2_DEFAULT_REPLY,
764   MPI2DefaultReply_t, MPI2_POINTER pMPI2DefaultReply_t;

767 /* common version structure/union used in messages and configuration pages */

769 typedef struct _MPI2_VERSION_STRUCT
770 {
771     U8        Dev;                     /* 0x00 */
772     U8        Unit;                    /* 0x01 */
773     U8        Minor;                   /* 0x02 */
774     U8        Major;                   /* 0x03 */
775 } MPI2_VERSION_STRUCT;

777 typedef union _MPI2_VERSION_UNION
778 {
779     MPI2_VERSION_STRUCT    Struct;
780     U32                     Word;
781 } MPI2_VERSION_UNION;

784 /* LUN field defines, common to many structures */
785 #define MPI2_LUN_FIRST_LEVEL_ADDRESSING          (0x0000FFFF)
786 #define MPI2_LUN_SECOND_LEVEL_ADDRESSING         (0xFFFF0000)
787 #define MPI2_LUN_THIRD_LEVEL_ADDRESSING          (0x0000FFFF)

```

```

788 #define MPI2_LUN_FOURTH_LEVEL_ADDRESSING (0xFFFF0000)
789 #define MPI2_LUN_LEVEL_1_WORD (0xFF00)
790 #define MPI2_LUN_LEVEL_1_DWORD (0x0000FF00)

793 /*****
794 *
795 *      Fusion-MPT MPI Scatter Gather Elements
796 *
797 *****/

799 /*****
800 *      MPI Simple Element structures
801 *****/

803 typedef struct _MPI2_SGE_SIMPLE32
804 {
805     U32          FlagsLength;
806     U32          Address;
807 } MPI2_SGE_SIMPLE32, MPI2_POINTER PTR_MPI2_SGE_SIMPLE32,
808   Mpi2SGESimple32_t, MPI2_POINTER pMpi2SGESimple32_t;

810 typedef struct _MPI2_SGE_SIMPLE64
811 {
812     U32          FlagsLength;
813     U64          Address;
814 } MPI2_SGE_SIMPLE64, MPI2_POINTER PTR_MPI2_SGE_SIMPLE64,
815   Mpi2SGESimple64_t, MPI2_POINTER pMpi2SGESimple64_t;

817 typedef struct _MPI2_SGE_SIMPLE_UNION
818 {
819     U32          FlagsLength;
820     union
821     {
822         U32          Address32;
823         U64          Address64;
824     } u;
825 } MPI2_SGE_SIMPLE_UNION, MPI2_POINTER PTR_MPI2_SGE_SIMPLE_UNION,
826   Mpi2SGESimpleUnion_t, MPI2_POINTER pMpi2SGESimpleUnion_t;

829 /*****
830 *      MPI Chain Element structures - for MPI v2.0 products only
831 *****/

833 typedef struct _MPI2_SGE_CHAIN32
834 {
835     U16          Length;
836     U8           NextChainOffset;
837     U8           Flags;
838     U32          Address;
839 } MPI2_SGE_CHAIN32, MPI2_POINTER PTR_MPI2_SGE_CHAIN32,
840   Mpi2SGEChain32_t, MPI2_POINTER pMpi2SGEChain32_t;

842 typedef struct _MPI2_SGE_CHAIN64
843 {
844     U16          Length;
845     U8           NextChainOffset;
846     U8           Flags;
847     U64          Address;
848 } MPI2_SGE_CHAIN64, MPI2_POINTER PTR_MPI2_SGE_CHAIN64,
849   Mpi2SGEChain64_t, MPI2_POINTER pMpi2SGEChain64_t;

851 typedef struct _MPI2_SGE_CHAIN_UNION
852 {
853     U16          Length;

```

```

854     U8           NextChainOffset;
855     U8           Flags;
856     union
857     {
858         U32          Address32;
859         U64          Address64;
860     } u;
861 } MPI2_SGE_CHAIN_UNION, MPI2_POINTER PTR_MPI2_SGE_CHAIN_UNION,
862   Mpi2SGEChainUnion_t, MPI2_POINTER pMpi2SGEChainUnion_t;

865 /*****
866 *      MPI Transaction Context Element structures - for MPI v2.0 products only
867 *****/

869 typedef struct _MPI2_SGE_TRANSACTION32
870 {
871     U8           Reserved;
872     U8           ContextSize;
873     U8           DetailsLength;
874     U8           Flags;
875     U32          TransactionContext[1];
876     U32          TransactionDetails[1];
877 } MPI2_SGE_TRANSACTION32, MPI2_POINTER PTR_MPI2_SGE_TRANSACTION32,
878   Mpi2SGETransaction32_t, MPI2_POINTER pMpi2SGETransaction32_t;

880 typedef struct _MPI2_SGE_TRANSACTION64
881 {
882     U8           Reserved;
883     U8           ContextSize;
884     U8           DetailsLength;
885     U8           Flags;
886     U32          TransactionContext[2];
887     U32          TransactionDetails[1];
888 } MPI2_SGE_TRANSACTION64, MPI2_POINTER PTR_MPI2_SGE_TRANSACTION64,
889   Mpi2SGETransaction64_t, MPI2_POINTER pMpi2SGETransaction64_t;

891 typedef struct _MPI2_SGE_TRANSACTION96
892 {
893     U8           Reserved;
894     U8           ContextSize;
895     U8           DetailsLength;
896     U8           Flags;
897     U32          TransactionContext[3];
898     U32          TransactionDetails[1];
899 } MPI2_SGE_TRANSACTION96, MPI2_POINTER PTR_MPI2_SGE_TRANSACTION96,
900   Mpi2SGETransaction96_t, MPI2_POINTER pMpi2SGETransaction96_t;

902 typedef struct _MPI2_SGE_TRANSACTION128
903 {
904     U8           Reserved;
905     U8           ContextSize;
906     U8           DetailsLength;
907     U8           Flags;
908     U32          TransactionContext[4];
909     U32          TransactionDetails[1];
910 } MPI2_SGE_TRANSACTION128, MPI2_POINTER PTR_MPI2_SGE_TRANSACTION128,
911   Mpi2SGETransaction_t128, MPI2_POINTER pMpi2SGETransaction_t128;

913 typedef struct _MPI2_SGE_TRANSACTION_UNION
914 {
915     U8           Reserved;
916     U8           ContextSize;
917     U8           DetailsLength;
918     U8           Flags;
919     union

```

```

920 {
921     U32          TransactionContext32[1];
922     U32          TransactionContext64[2];
923     U32          TransactionContext96[3];
924     U32          TransactionContext128[4];
925 } u;
926 U32          TransactionDetails[1];
927 } MPI2_SGE_TRANSACTION_UNION, MPI2_POINTER PTR_MPI2_SGE_TRANSACTION_UNION,
928 Mpi2SGETransactionUnion_t, MPI2_POINTER pMpi2SGETransactionUnion_t;

931 /*****
932 * MPI SGE union for IO SGL's - for MPI v2.0 products only
933 *****/

935 typedef struct _MPI2_MPI_SGE_IO_UNION
936 {
937     union
938     {
939         MPI2_SGE_SIMPLE_UNION    Simple;
940         MPI2_SGE_CHAIN_UNION     Chain;
941     } u;
942 } MPI2_MPI_SGE_IO_UNION, MPI2_POINTER PTR_MPI2_MPI_SGE_IO_UNION,
943 Mpi2MpiSGEIOUnion_t, MPI2_POINTER pMpi2MpiSGEIOUnion_t;

946 /*****
947 * MPI SGE union for SGL's with Simple and Transaction elements - for MPI v2.0 p
948 *****/

950 typedef struct _MPI2_SGE_TRANS_SIMPLE_UNION
951 {
952     union
953     {
954         MPI2_SGE_SIMPLE_UNION    Simple;
955         MPI2_SGE_TRANSACTION_UNION Transaction;
956     } u;
957 } MPI2_SGE_TRANS_SIMPLE_UNION, MPI2_POINTER PTR_MPI2_SGE_TRANS_SIMPLE_UNION,
958 Mpi2SGETransSimpleUnion_t, MPI2_POINTER pMpi2SGETransSimpleUnion_t;

961 /*****
962 * All MPI SGE types union
963 *****/

965 typedef struct _MPI2_MPI_SGE_UNION
966 {
967     union
968     {
969         MPI2_SGE_SIMPLE_UNION    Simple;
970         MPI2_SGE_CHAIN_UNION     Chain;
971         MPI2_SGE_TRANSACTION_UNION Transaction;
972     } u;
973 } MPI2_MPI_SGE_UNION, MPI2_POINTER PTR_MPI2_MPI_SGE_UNION,
974 Mpi2MpiSgeUnion_t, MPI2_POINTER pMpi2MpiSgeUnion_t;

977 /*****
978 * MPI SGE field definition and masks
979 *****/

981 /* Flags field bit definitions */

983 #define MPI2_SGE_FLAGS_LAST_ELEMENT          (0x80)
984 #define MPI2_SGE_FLAGS_END_OF_BUFFER        (0x40)
985 #define MPI2_SGE_FLAGS_ELEMENT_TYPE_MASK    (0x30)

```

```

986 #define MPI2_SGE_FLAGS_LOCAL_ADDRESS          (0x08)
987 #define MPI2_SGE_FLAGS_DIRECTION            (0x04)
988 #define MPI2_SGE_FLAGS_ADDRESS_SIZE          (0x02)
989 #define MPI2_SGE_FLAGS_END_OF_LIST          (0x01)

991 #define MPI2_SGE_FLAGS_SHIFT                (24)

993 #define MPI2_SGE_LENGTH_MASK                 (0x00FFFFFF)
994 #define MPI2_SGE_CHAIN_LENGTH_MASK          (0x0000FFFF)

996 /* Element Type */

998 #define MPI2_SGE_FLAGS_TRANSACTION_ELEMENT    (0x00) /* for MPI v2.0 products
999 #define MPI2_SGE_FLAGS_SIMPLE_ELEMENT        (0x10)
1000 #define MPI2_SGE_FLAGS_CHAIN_ELEMENT        (0x30) /* for MPI v2.0 products
1001 #define MPI2_SGE_FLAGS_ELEMENT_MASK         (0x30)

1003 /* Address location */

1005 #define MPI2_SGE_FLAGS_SYSTEM_ADDRESS        (0x00)

1007 /* Direction */

1009 #define MPI2_SGE_FLAGS_IOC_TO_HOST           (0x00)
1010 #define MPI2_SGE_FLAGS_HOST_TO_IOC          (0x04)

1012 #define MPI2_SGE_FLAGS_DEST                  (MPI2_SGE_FLAGS_IOC_TO_HOST)
1013 #define MPI2_SGE_FLAGS_SOURCE                (MPI2_SGE_FLAGS_HOST_TO_IOC)

1015 /* Address Size */

1017 #define MPI2_SGE_FLAGS_32_BIT_ADDRESSING      (0x00)
1018 #define MPI2_SGE_FLAGS_64_BIT_ADDRESSING      (0x02)

1020 /* Context Size */

1022 #define MPI2_SGE_FLAGS_32_BIT_CONTEXT         (0x00)
1023 #define MPI2_SGE_FLAGS_64_BIT_CONTEXT         (0x02)
1024 #define MPI2_SGE_FLAGS_96_BIT_CONTEXT         (0x04)
1025 #define MPI2_SGE_FLAGS_128_BIT_CONTEXT        (0x06)

1027 #define MPI2_SGE_CHAIN_OFFSET_MASK            (0x00FF0000)
1028 #define MPI2_SGE_CHAIN_OFFSET_SHIFT          (16)

1030 /*****
1031 * MPI SGE operation Macros
1032 *****/

1034 /* SIMPLE FlagsLength manipulations... */
1035 #define MPI2_SGE_SET_FLAGS(f)                (((U32)(f) << MPI2_SGE_FLAGS_SHIFT)
1036 #define MPI2_SGE_GET_FLAGS(f)                (((f) & ~MPI2_SGE_LENGTH_MASK) >> MPI2_SG
1037 #define MPI2_SGE_LENGTH(f)                   ((f) & MPI2_SGE_LENGTH_MASK)
1038 #define MPI2_SGE_CHAIN_LENGTH(f)             ((f) & MPI2_SGE_CHAIN_LENGTH_MASK)

1040 #define MPI2_SGE_SET_FLAGS_LENGTH(f,l) (MPI2_SGE_SET_FLAGS(f) | MPI2_SGE_LENGTH(
1042 #define MPI2_pSGE_GET_FLAGS(psg)             MPI2_SGE_GET_FLAGS((psg)->FlagsLengt
1043 #define MPI2_pSGE_GET_LENGTH(psg)            MPI2_SGE_LENGTH((psg)->FlagsLength)
1044 #define MPI2_pSGE_SET_FLAGS_LENGTH(psg,f,l) (psg)->FlagsLength = MPI2_SGE_SET_FL

1046 /* CAUTION - The following are READ-MODIFY-WRITE! */
1047 #define MPI2_pSGE_SET_FLAGS(psg,f)           (psg)->FlagsLength |= MPI2_SGE_SET_FLAGS
1048 #define MPI2_pSGE_SET_LENGTH(psg,l)          (psg)->FlagsLength |= MPI2_SGE_LENGTH(l)

1050 #define MPI2_GET_CHAIN_OFFSET(x)              ((x & MPI2_SGE_CHAIN_OFFSET_MASK) >> MPI2_SG

```

```

1053 /*****
1054 *
1055 *      Fusion-MPT IEEE Scatter Gather Elements
1056 *
1057 *****/

1059 /*****
1060 * IEEE Simple Element structures
1061 *****/

1063 /* MPI2_IEEE_SGE_SIMPLE32 is for MPI v2.0 products only */
1064 typedef struct _MPI2_IEEE_SGE_SIMPLE32
1065 {
1066     U32          Address;
1067     U32          FlagsLength;
1068 } MPI2_IEEE_SGE_SIMPLE32, MPI2_POINTER PTR_MPI2_IEEE_SGE_SIMPLE32,
1069   Mpi2IeeeSgeSimple32_t, MPI2_POINTER pMpi2IeeeSgeSimple32_t;

1071 typedef struct _MPI2_IEEE_SGE_SIMPLE64
1072 {
1073     U64          Address;
1074     U32          Length;
1075     U16          Reserved1;
1076     U8           Reserved2;
1077     U8           Flags;
1078 } MPI2_IEEE_SGE_SIMPLE64, MPI2_POINTER PTR_MPI2_IEEE_SGE_SIMPLE64,
1079   Mpi2IeeeSgeSimple64_t, MPI2_POINTER pMpi2IeeeSgeSimple64_t;

1081 typedef union _MPI2_IEEE_SGE_SIMPLE_UNION
1082 {
1083     MPI2_IEEE_SGE_SIMPLE32  Simple32;
1084     MPI2_IEEE_SGE_SIMPLE64  Simple64;
1085 } MPI2_IEEE_SGE_SIMPLE_UNION, MPI2_POINTER PTR_MPI2_IEEE_SGE_SIMPLE_UNION,
1086   Mpi2IeeeSgeSimpleUnion_t, MPI2_POINTER pMpi2IeeeSgeSimpleUnion_t;

1089 /*****
1090 * IEEE Chain Element structures
1091 *****/

1093 /* MPI2_IEEE_SGE_CHAIN32 is for MPI v2.0 products only */
1094 typedef MPI2_IEEE_SGE_SIMPLE32  MPI2_IEEE_SGE_CHAIN32;

1096 /* MPI2_IEEE_SGE_CHAIN64 is for MPI v2.0 products only */
1097 typedef MPI2_IEEE_SGE_SIMPLE64  MPI2_IEEE_SGE_CHAIN64;

1099 typedef union _MPI2_IEEE_SGE_CHAIN_UNION
1100 {
1101     MPI2_IEEE_SGE_CHAIN32  Chain32;
1102     MPI2_IEEE_SGE_CHAIN64  Chain64;
1103 } MPI2_IEEE_SGE_CHAIN_UNION, MPI2_POINTER PTR_MPI2_IEEE_SGE_CHAIN_UNION,
1104   Mpi2IeeeSgeChainUnion_t, MPI2_POINTER pMpi2IeeeSgeChainUnion_t;

1106 /* MPI25_IEEE_SGE_CHAIN64 is for MPI v2.5 products only */
1107 typedef struct _MPI25_IEEE_SGE_CHAIN64
1108 {
1109     U64          Address;
1110     U32          Length;
1111     U16          Reserved1;
1112     U8           NextChainOffset;
1113     U8           Flags;
1114 } MPI25_IEEE_SGE_CHAIN64, MPI2_POINTER PTR_MPI25_IEEE_SGE_CHAIN64,
1115   Mpi25IeeeSgeChain64_t, MPI2_POINTER pMpi25IeeeSgeChain64_t;

```

```

1118 /*****
1119 * All IEEE SGE types union
1120 *****/

1122 /* MPI2_IEEE_SGE_UNION is for MPI v2.0 products only */
1123 typedef struct _MPI2_IEEE_SGE_UNION
1124 {
1125     union
1126     {
1127         MPI2_IEEE_SGE_SIMPLE_UNION  Simple;
1128         MPI2_IEEE_SGE_CHAIN_UNION   Chain;
1129     } u;
1130 } MPI2_IEEE_SGE_UNION, MPI2_POINTER PTR_MPI2_IEEE_SGE_UNION,
1131   Mpi2IeeeSgeUnion_t, MPI2_POINTER pMpi2IeeeSgeUnion_t;

1134 /*****
1135 * IEEE SGE union for IO SGL's
1136 *****/

1138 typedef union _MPI25_SGE_IO_UNION
1139 {
1140     MPI2_IEEE_SGE_SIMPLE64  IeeeSimple;
1141     MPI25_IEEE_SGE_CHAIN64  IeeeChain;
1142 } MPI25_SGE_IO_UNION, MPI2_POINTER PTR_MPI25_SGE_IO_UNION,
1143   Mpi25SgeIOUnion_t, MPI2_POINTER pMpi25SgeIOUnion_t;

1146 /*****
1147 * IEEE SGE field definitions and masks
1148 *****/

1150 /* Flags field bit definitions */

1152 #define MPI2_IEEE_SGE_FLAGS_ELEMENT_TYPE_MASK    (0x80)
1153 #define MPI25_IEEE_SGE_FLAGS_END_OF_LIST         (0x40)

1155 #define MPI2_IEEE32_SGE_FLAGS_SHIFT              (24)

1157 #define MPI2_IEEE32_SGE_LENGTH_MASK              (0x00FFFFFF)

1159 /* Element Type */

1161 #define MPI2_IEEE_SGE_FLAGS_SIMPLE_ELEMENT       (0x00)
1162 #define MPI2_IEEE_SGE_FLAGS_CHAIN_ELEMENT       (0x80)

1164 /* Data Location Address Space */

1166 #define MPI2_IEEE_SGE_FLAGS_ADDR_MASK           (0x03)
1167 #define MPI2_IEEE_SGE_FLAGS_SYSTEM_ADDR        (0x00) /* IEEE Simple Element on
1168 #define MPI2_IEEE_SGE_FLAGS_IOCDDR_ADDR        (0x01) /* IEEE Simple Element on
1169 #define MPI2_IEEE_SGE_FLAGS_IOCPLB_ADDR        (0x02)
1170 #define MPI2_IEEE_SGE_FLAGS_IOCPLBNTA_ADDR     (0x03) /* IEEE Simple Element on
1171 #define MPI2_IEEE_SGE_FLAGS_SYSTEMPLBCPI_ADDR  (0x03) /* IEEE Chain Element onl

1174 /*****
1175 * IEEE SGE operation Macros
1176 *****/

1178 /* SIMPLE FlagsLength manipulations... */
1179 #define MPI2_IEEE32_SGE_SET_FLAGS(f)            ((U32)(f) << MPI2_IEEE32_SGE_FLAGS_SHIF
1180 #define MPI2_IEEE32_SGE_GET_FLAGS(f)            (((f) & ~MPI2_IEEE32_SGE_LENGTH_MASK) >
1181 #define MPI2_IEEE32_SGE_LENGTH(f)              ((f) & MPI2_IEEE32_SGE_LENGTH_MASK)

1183 #define MPI2_IEEE32_SGE_SET_FLAGS_LENGTH(f, 1)  (MPI2_IEEE32_SGE_SET_FLAGS(f

```

```
1185 #define MPI2_IEEE32_pSGE_GET_FLAGS(psg)          MPI2_IEEE32_SGE_GET_FLAGS((p
1186 #define MPI2_IEEE32_pSGE_GET_LENGTH(psg)          MPI2_IEEE32_SGE_LENGTH((psg)
1187 #define MPI2_IEEE32_pSGE_SET_FLAGS(psg,f,l)      (psg)->FlagsLength = MPI2_IE

1189 /* CAUTION - The following are READ-MODIFY-WRITE! */
1190 #define MPI2_IEEE32_pSGE_SET_FLAGS(psg,f)         (psg)->FlagsLength |= MPI2_IEEE32_S
1191 #define MPI2_IEEE32_pSGE_SET_LENGTH(psg,l)        (psg)->FlagsLength |= MPI2_IEEE32_S

1196 /*****
1197 *
1198 *      Fusion-MPT MPI/IEEE Scatter Gather Unions
1199 *
1200 *****/

1202 typedef union _MPI2_SIMPLE_SGE_UNION
1203 {
1204     MPI2_SGE_SIMPLE_UNION      MpiSimple;
1205     MPI2_IEEE_SGE_SIMPLE_UNION IeeeSimple;
1206 } MPI2_SIMPLE_SGE_UNION, MPI2_POINTER PTR_MPI2_SIMPLE_SGE_UNION,
1207   Mpi2SimpleSgeUnion_t, MPI2_POINTER pMpi2SimpleSgeUnion_t;

1210 typedef union _MPI2_SGE_IO_UNION
1211 {
1212     MPI2_SGE_SIMPLE_UNION      MpiSimple;
1213     MPI2_SGE_CHAIN_UNION       MpiChain;
1214     MPI2_IEEE_SGE_SIMPLE_UNION IeeeSimple;
1215     MPI2_IEEE_SGE_CHAIN_UNION  IeeeChain;
1216 } MPI2_SGE_IO_UNION, MPI2_POINTER PTR_MPI2_SGE_IO_UNION,
1217   Mpi2SGEIOUnion_t, MPI2_POINTER pMpi2SGEIOUnion_t;

1220 /*****
1221 *
1222 *      Values for SGLFlags field, used in many request messages with an SGL
1223 *
1224 *****/

1226 /* values for MPI SGL Data Location Address Space subfield */
1227 #define MPI2_SGLFLAGS_ADDRESS_SPACE_MASK      (0x0C)
1228 #define MPI2_SGLFLAGS_SYSTEM_ADDRESS_SPACE    (0x00)
1229 #define MPI2_SGLFLAGS_IOCDDR_ADDRESS_SPACE     (0x04)
1230 #define MPI2_SGLFLAGS_IOCPLB_ADDRESS_SPACE     (0x08)
1231 #define MPI2_SGLFLAGS_IOCPLBNTA_ADDRESS_SPACE (0x0C)
1232 /* values for SGL Type subfield */
1233 #define MPI2_SGLFLAGS_SGL_TYPE_MASK           (0x03)
1234 #define MPI2_SGLFLAGS_SGL_TYPE_MPI            (0x00)
1235 #define MPI2_SGLFLAGS_SGL_TYPE_IEEE32        (0x01)
1236 #define MPI2_SGLFLAGS_SGL_TYPE_IEEE64        (0x02)

1239 #endif

1241 #endif /* ! codereview */
```


new/usr/src/uts/common/sys/scsi/adapters/mpt_sas3/mpi/mpi2_cnfg.h

1

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

```
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */
21
22 /*
23  * Copyright (c) 2000-2012 LSI Corporation.
24  *
25  * Redistribution and use in source and binary forms of all code within
26  * this file that is exclusively owned by LSI, with or without
27  * modification, is permitted provided that, in addition to the CDDL 1.0
28  * License requirements, the following conditions are met:
29  *
30  * Neither the name of the author nor the names of its contributors may be
31  * used to endorse or promote products derived from this software without
32  * specific prior written permission.
33  *
34  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
35  * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
36  * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
37  * FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
38  * COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
39  * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
40  * BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
41  * OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
42  * AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
43  * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
44  * OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
45  * DAMAGE.
46 */
47
48 /*
49  * Name: mpi2_cnfg.h
50  * Title: MPI Configuration messages and pages
51  * Creation Date: November 10, 2006
52  *
53  * mpi2_cnfg.h Version: 02.00.xx
54  *
55  * NOTE: Names (typedefs, defines, etc.) beginning with an MPI25 or Mpi25
56  * prefix are for use only on MPI v2.5 products, and must not be used
57  * with MPI v2.0 products. Unless otherwise noted, names beginning with
58  * MPI2 or Mpi2 are for use with both MPI v2.0 and MPI v2.5 products.
59  *
60  * Version History
61  * -----
```

new/usr/src/uts/common/sys/scsi/adapters/mpt_sas3/mpi/mpi2_cnfg.h

2

```
62 *
63 * Date Version Description
64 * -----
65 * 04-30-07 02.00.00 Corresponds to Fusion-MPT MPI Specification Rev A.
66 * 06-04-07 02.00.01 Added defines for SAS IO Unit Page 2 PhyFlags.
67 * Added Manufacturing Page 11.
68 * Added MPI2_SAS_EXPANDER0_FLAGS_CONNECTOR_END_DEVICE
69 * define.
70 * 06-26-07 02.00.02 Adding generic structure for product-specific
71 * Manufacturing pages: MPI2_CONFIG_PAGE_MANUFACTURING_PS.
72 * Rework of BIOS Page 2 configuration page.
73 * Fixed MPI2_BIOSPAGE2_BOOT_DEVICE to be a union of the
74 * forms.
75 * Added configuration pages IOC Page 8 and Driver
76 * Persistent Mapping Page 0.
77 * 08-31-07 02.00.03 Modified configuration pages dealing with Integrated
78 * RAID (Manufacturing Page 4, RAID Volume Pages 0 and 1,
79 * RAID Physical Disk Pages 0 and 1, RAID Configuration
80 * Page 0).
81 * Added new value for AccessStatus field of SAS Device
82 * Page 0 (_SATA_NEEDS_INITIALIZATION).
83 * 10-31-07 02.00.04 Added missing SEPDevHandle field to
84 * MPI2_CONFIG_PAGE_SAS_ENCLOSURE_0.
85 * 12-18-07 02.00.05 Modified IO Unit Page 0 to use 32-bit version fields for
86 * NVDATA.
87 * Modified IOC Page 7 to use masks and added field for
88 * SASBroadcastPrimitiveMasks.
89 * Added MPI2_CONFIG_PAGE_BIOS_4.
90 * Added MPI2_CONFIG_PAGE_LOG_0.
91 * 02-29-08 02.00.06 Modified various names to make them 32-character unique.
92 * Added SAS Device IDs.
93 * Updated Integrated RAID configuration pages including
94 * Manufacturing Page 4, IOC Page 6, and RAID Configuration
95 * Page 0.
96 * 05-21-08 02.00.07 Added define MPI2_MANPAGE4_MIX_SSD_SAS_SATA.
97 * Added define MPI2_MANPAGE4_PHYSDISK_128MB_COERCION.
98 * Fixed define MPI2_IOCPAGE8_FLAGS_ENCLOSURE_SLOT_MAPPING.
99 * Added missing MaxNumRoutedSasAddresses field to
100 * MPI2_CONFIG_PAGE_EXPANDER_0.
101 * Added SAS Port Page 0.
102 * Modified structure layout for
103 * MPI2_CONFIG_PAGE_DRIVER_MAPPING_0.
104 * 06-27-08 02.00.08 Changed MPI2_CONFIG_PAGE_RD_PDISK_1 to use
105 * MPI2_RAID_PHYS_DISK1_PATH_MAX to size the array.
106 * 10-02-08 02.00.09 Changed MPI2_RAID_PGAD_CONFIGNUM_MASK from 0x0000FFFF
107 * to 0x000000FF.
108 * Added two new values for the Physical Disk Coercion Size
109 * bits in the Flags field of Manufacturing Page 4.
110 * Added product-specific Manufacturing pages 16 to 31.
111 * Modified Flags bits for controlling write cache on SATA
112 * drives in IO Unit Page 1.
113 * Added new bit to AdditionalControlFlags of SAS IO Unit
114 * Page 1 to control Invalid Topology Correction.
115 * Added additional defines for RAID Volume Page 0
116 * VolumeStatusFlags field.
117 * Modified meaning of RAID Volume Page 0 VolumeSettings
118 * define for auto-configure of hot-swap drives.
119 * Added SupportedPhysDisks field to RAID Volume Page 1 and
120 * added related defines.
121 * Added PhysDiskAttributes field (and related defines) to
122 * RAID Physical Disk Page 0.
123 * Added MPI2_SAS_PHYINFO_PHY_VACANT define.
124 * Added three new DiscoveryStatus bits for SAS IO Unit
125 * Page 0 and SAS Expander Page 0.
126 * Removed multiplexing information from SAS IO Unit pages.
127 * Added BootDeviceWaitTime field to SAS IO Unit Page 4.
```

```

128 *      Removed Zone Address Resolved bit from PhyInfo and from
129 *      Expander Page 0 Flags field.
130 *      Added two new AccessStatus values to SAS Device Page 0
131 *      for indicating routing problems. Added 3 reserved words
132 *      to this page.
133 *      01-19-09  02.00.10  Fixed defines for GPIOVal field of IO Unit Page 3.
134 *      Inserted missing reserved field into structure for IOC
135 *      Page 6.
136 *      Added more pending task bits to RAID Volume Page 0
137 *      VolumeStatusFlags defines.
138 *      Added MPI2_PHYSDISK0_STATUS_FLAG_NOT_CERTIFIED define.
139 *      Added a new DiscoveryStatus bit for SAS IO Unit Page 0
140 *      and SAS Expander Page 0 to flag a downstream initiator
141 *      when in simplified routing mode.
142 *      Removed SATA Init Failure defines for DiscoveryStatus
143 *      fields of SAS IO Unit Page 0 and SAS Expander Page 0.
144 *      Added MPI2_SAS_DEVICE0_ASTATUS_DEVICE_BLOCKED define.
145 *      Added PortGroups, DmaGroup, and ControlGroup fields to
146 *      SAS Device Page 0.
147 *      05-06-09  02.00.11  Added structures and defines for IO Unit Page 5 and IO
148 *      Unit Page 6.
149 *      Added expander reduced functionality data to SAS
150 *      Expander Page 0.
151 *      Added SAS PHY Page 2 and SAS PHY Page 3.
152 *      07-30-09  02.00.12  Added IO Unit Page 7.
153 *      Added new device ids.
154 *      Added SAS IO Unit Page 5.
155 *      Added partial and slumber power management capable flags
156 *      to SAS Device Page 0 Flags field.
157 *      Added PhyInfo defines for power condition.
158 *      Added Ethernet configuration pages.
159 *      10-28-09  02.00.13  Added MPI2_IOUNITPAGE1_ENABLE_HOST_BASED_DISCOVERY.
160 *      Added SAS PHY Page 4 structure and defines.
161 *      02-10-10  02.00.14  Modified the comments for the configuration page
162 *      structures that contain an array of data. The host
163 *      should use the "count" field in the page data (e.g. the
164 *      NumPhys field) to determine the number of valid elements
165 *      in the array.
166 *      Added/modified some MPI2_MFGPAGE_DEVID_SAS defines.
167 *      Added PowerManagementCapabilities to IO Unit Page 7.
168 *      Added PortWidthModGroup field to
169 *      MPI2_SAS_IO_UNITS_PHY_PM_SETTINGS.
170 *      Added MPI2_CONFIG_PAGE_SASIOUNIT_6 and related defines.
171 *      Added MPI2_CONFIG_PAGE_SASIOUNIT_7 and related defines.
172 *      Added MPI2_CONFIG_PAGE_SASIOUNIT_8 and related defines.
173 *      05-12-10  02.00.15  Added MPI2_RAIDVOL0_STATUS_FLAG_VOL_NOT_CONSISTENT
174 *      define.
175 *      Added MPI2_PHYSDISK0_INCOMPATIBLE_MEDIA_TYPE define.
176 *      Added MPI2_SAS_NEG_LINK_RATE_UNSUPPORTED_PHY define.
177 *      08-11-10  02.00.16  Removed IO Unit Page 1 device path (multi-pathing)
178 *      defines.
179 *      11-10-10  02.00.17  Added ReceptacleID field (replacing Reserved1) to
180 *      MPI2_MANPAGE7_CONNECTOR_INFO and reworked defines for
181 *      the Pinout field.
182 *      Added BoardTemperature and BoardTemperatureUnits fields
183 *      to MPI2_CONFIG_PAGE_IO_UNIT_7.
184 *      Added MPI2_CONFIG_EXTPAGETYPE_EXT_MANUFACTURING define
185 *      and MPI2_CONFIG_PAGE_EXT_MAN_PS structure.
186 *      -----
187 */

189 #ifndef MPI2_CNFG_H
190 #define MPI2_CNFG_H

192 /*****
193 *      Configuration Page Header and defines

```

```

194 *****/

196 /* Config Page Header */
197 typedef struct _MPI2_CONFIG_PAGE_HEADER
198 {
199     U8      PageVersion;          /* 0x00 */
200     U8      PageLength;          /* 0x01 */
201     U8      PageNumber;          /* 0x02 */
202     U8      PageType;            /* 0x03 */
203 } MPI2_CONFIG_PAGE_HEADER, MPI2_POINTER PTR_MPI2_CONFIG_PAGE_HEADER,
204   Mpi2ConfigPageHeader_t, MPI2_POINTER pMpi2ConfigPageHeader_t;

206 typedef union _MPI2_CONFIG_PAGE_HEADER_UNION
207 {
208     MPI2_CONFIG_PAGE_HEADER  Struct;
209     U8      Bytes[4];
210     U16     Word16[2];
211     U32     Word32;
212 } MPI2_CONFIG_PAGE_HEADER_UNION, MPI2_POINTER PTR_MPI2_CONFIG_PAGE_HEADER_UNION,
213   Mpi2ConfigPageHeaderUnion, MPI2_POINTER pMpi2ConfigPageHeaderUnion;

215 /* Extended Config Page Header */
216 typedef struct _MPI2_CONFIG_EXTENDED_PAGE_HEADER
217 {
218     U8      PageVersion;          /* 0x00 */
219     U8      Reserved1;           /* 0x01 */
220     U8      PageNumber;          /* 0x02 */
221     U8      PageType;            /* 0x03 */
222     U16     ExtPageLength;       /* 0x04 */
223     U8      ExtPageType;         /* 0x06 */
224     U8      Reserved2;           /* 0x07 */
225 } MPI2_CONFIG_EXTENDED_PAGE_HEADER,
226   MPI2_POINTER PTR_MPI2_CONFIG_EXTENDED_PAGE_HEADER,
227   Mpi2ConfigExtendedPageHeader_t, MPI2_POINTER pMpi2ConfigExtendedPageHeader_t;

229 typedef union _MPI2_CONFIG_EXT_PAGE_HEADER_UNION
230 {
231     MPI2_CONFIG_PAGE_HEADER  Struct;
232     MPI2_CONFIG_EXTENDED_PAGE_HEADER  Ext;
233     U8      Bytes[8];
234     U16     Word16[4];
235     U32     Word32[2];
236 } MPI2_CONFIG_EXT_PAGE_HEADER_UNION, MPI2_POINTER PTR_MPI2_CONFIG_EXT_PAGE_HEADE
237   Mpi2ConfigPageExtendedHeaderUnion, MPI2_POINTER pMpi2ConfigPageExtendedHeaderU

240 /* PageType field values */
241 #define MPI2_CONFIG_PAGEATTR_READ_ONLY          (0x00)
242 #define MPI2_CONFIG_PAGEATTR_CHANGEABLE        (0x10)
243 #define MPI2_CONFIG_PAGEATTR_PERSISTENT        (0x20)
244 #define MPI2_CONFIG_PAGEATTR_MASK              (0xF0)

246 #define MPI2_CONFIG_PAGETYPE_IO_UNIT            (0x00)
247 #define MPI2_CONFIG_PAGETYPE_IOC                (0x01)
248 #define MPI2_CONFIG_PAGETYPE_BIOS               (0x02)
249 #define MPI2_CONFIG_PAGETYPE_RAID_VOLUME        (0x08)
250 #define MPI2_CONFIG_PAGETYPE_MANUFACTURING      (0x09)
251 #define MPI2_CONFIG_PAGETYPE_RAID_PHYSDISK      (0x0A)
252 #define MPI2_CONFIG_PAGETYPE_EXTENDED           (0x0F)
253 #define MPI2_CONFIG_PAGETYPE_MASK               (0x0F)

255 #define MPI2_CONFIG_TYPENUM_MASK                (0x0FFF)

258 /* ExtPageType field values */
259 #define MPI2_CONFIG_EXTPAGETYPE_SAS_IO_UNIT      (0x10)

```

```

260 #define MPI2_CONFIG_EXTPAGETYPE_SAS_EXPANDER      (0x11)
261 #define MPI2_CONFIG_EXTPAGETYPE_SAS_DEVICE        (0x12)
262 #define MPI2_CONFIG_EXTPAGETYPE_SAS_PHY           (0x13)
263 #define MPI2_CONFIG_EXTPAGETYPE_LOG               (0x14)
264 #define MPI2_CONFIG_EXTPAGETYPE_ENCLOSURE         (0x15)
265 #define MPI2_CONFIG_EXTPAGETYPE_RAID_CONFIG       (0x16)
266 #define MPI2_CONFIG_EXTPAGETYPE_DRIVER_MAPPING    (0x17)
267 #define MPI2_CONFIG_EXTPAGETYPE_SAS_PORT          (0x18)
268 #define MPI2_CONFIG_EXTPAGETYPE_ETHERNET          (0x19)
269 #define MPI2_CONFIG_EXTPAGETYPE_EXT_MANUFACTURING (0x1A)

272 /*****
273 *   PageAddress defines
274 *****/

276 /* RAID Volume PageAddress format */
277 #define MPI2_RAID_VOLUME_PGAD_FORM_MASK            (0xF0000000)
278 #define MPI2_RAID_VOLUME_PGAD_FORM_GET_NEXT_HANDLE (0x00000000)
279 #define MPI2_RAID_VOLUME_PGAD_FORM_HANDLE          (0x10000000)

281 #define MPI2_RAID_VOLUME_PGAD_HANDLE_MASK          (0x0000FFFF)

284 /* RAID Physical Disk PageAddress format */
285 #define MPI2_PHYSDISK_PGAD_FORM_MASK                (0xF0000000)
286 #define MPI2_PHYSDISK_PGAD_FORM_GET_NEXT_PHYSDISKNUM (0x00000000)
287 #define MPI2_PHYSDISK_PGAD_FORM_PHYSDISKNUM         (0x10000000)
288 #define MPI2_PHYSDISK_PGAD_FORM_DEVHANDLE           (0x20000000)

290 #define MPI2_PHYSDISK_PGAD_PHYSDISKNUM_MASK         (0x000000FF)
291 #define MPI2_PHYSDISK_PGAD_DEVHANDLE_MASK           (0x0000FFFF)

294 /* SAS Expander PageAddress format */
295 #define MPI2_SAS_EXPAND_PGAD_FORM_MASK              (0xF0000000)
296 #define MPI2_SAS_EXPAND_PGAD_FORM_GET_NEXT_HNDL     (0x00000000)
297 #define MPI2_SAS_EXPAND_PGAD_FORM_HNDL_PHY_NUM      (0x10000000)
298 #define MPI2_SAS_EXPAND_PGAD_FORM_HNDL              (0x20000000)

300 #define MPI2_SAS_EXPAND_PGAD_HANDLE_MASK            (0x0000FFFF)
301 #define MPI2_SAS_EXPAND_PGAD_PHYNUM_MASK            (0x00FF0000)
302 #define MPI2_SAS_EXPAND_PGAD_PHYNUM_SHIFT           (16)

305 /* SAS Device PageAddress format */
306 #define MPI2_SAS_DEVICE_PGAD_FORM_MASK              (0xF0000000)
307 #define MPI2_SAS_DEVICE_PGAD_FORM_GET_NEXT_HANDLE   (0x00000000)
308 #define MPI2_SAS_DEVICE_PGAD_FORM_HANDLE            (0x20000000)

310 #define MPI2_SAS_DEVICE_PGAD_HANDLE_MASK            (0x0000FFFF)

313 /* SAS PHY PageAddress format */
314 #define MPI2_SAS_PHY_PGAD_FORM_MASK                 (0xF0000000)
315 #define MPI2_SAS_PHY_PGAD_FORM_PHY_NUMBER           (0x00000000)
316 #define MPI2_SAS_PHY_PGAD_FORM_PHY_TBL_INDEX        (0x10000000)

318 #define MPI2_SAS_PHY_PGAD_PHY_NUMBER_MASK           (0x000000FF)
319 #define MPI2_SAS_PHY_PGAD_PHY_TBL_INDEX_MASK        (0x0000FFFF)

322 /* SAS Port PageAddress format */
323 #define MPI2_SASPORT_PGAD_FORM_MASK                 (0xF0000000)
324 #define MPI2_SASPORT_PGAD_FORM_GET_NEXT_PORT        (0x00000000)
325 #define MPI2_SASPORT_PGAD_FORM_PORT_NUM             (0x10000000)

```

```

327 #define MPI2_SASPORT_PGAD_PORTNUMBER_MASK          (0x00000FFF)

330 /* SAS Enclosure PageAddress format */
331 #define MPI2_SAS_ENCLOS_PGAD_FORM_MASK              (0xF0000000)
332 #define MPI2_SAS_ENCLOS_PGAD_FORM_GET_NEXT_HANDLE  (0x00000000)
333 #define MPI2_SAS_ENCLOS_PGAD_FORM_HANDLE            (0x10000000)

335 #define MPI2_SAS_ENCLOS_PGAD_HANDLE_MASK            (0x0000FFFF)

338 /* RAID Configuration PageAddress format */
339 #define MPI2_RAID_PGAD_FORM_MASK                    (0xF0000000)
340 #define MPI2_RAID_PGAD_FORM_GET_NEXT_CONFIGNUM      (0x00000000)
341 #define MPI2_RAID_PGAD_FORM_CONFIGNUM               (0x10000000)
342 #define MPI2_RAID_PGAD_FORM_ACTIVE_CONFIG           (0x20000000)

344 #define MPI2_RAID_PGAD_CONFIGNUM_MASK               (0x000000FF)

347 /* Driver Persistent Mapping PageAddress format */
348 #define MPI2_DPM_PGAD_FORM_MASK                     (0xF0000000)
349 #define MPI2_DPM_PGAD_FORM_ENTRY_RANGE              (0x00000000)

351 #define MPI2_DPM_PGAD_ENTRY_COUNT_MASK              (0x0FFF0000)
352 #define MPI2_DPM_PGAD_ENTRY_COUNT_SHIFT             (16)
353 #define MPI2_DPM_PGAD_START_ENTRY_MASK              (0x0000FFFF)

356 /* Ethernet PageAddress format */
357 #define MPI2_ETHERNET_PGAD_FORM_MASK                 (0xF0000000)
358 #define MPI2_ETHERNET_PGAD_FORM_IF_NUM              (0x00000000)

360 #define MPI2_ETHERNET_PGAD_IF_NUMBER_MASK            (0x000000FF)

364 /*****
365 *   Configuration messages
366 *****/

368 /* Configuration Request Message */
369 typedef struct _MPI2_CONFIG_REQUEST
370 {
371     U8      Action; /* 0x00 */
372     U8      SGLFlags; /* 0x01 */
373     U8      ChainOffset; /* 0x02 */
374     U8      Function; /* 0x03 */
375     U16     ExtPageLength; /* 0x04 */
376     U8      ExtPageType; /* 0x06 */
377     U8      MsgFlags; /* 0x07 */
378     U8      VP_ID; /* 0x08 */
379     U8      VF_ID; /* 0x09 */
380     U16     Reserved1; /* 0x0A */
381     U32     Reserved2; /* 0x0C */
382     U32     Reserved3; /* 0x10 */
383     MPI2_CONFIG_PAGE_HEADER Header; /* 0x14 */
384     U32     PageAddress; /* 0x18 */
385     MPI2_SGE_IO_UNION PageBufferSGE; /* 0x1C */
386 } MPI2_CONFIG_REQUEST, MPI2_POINTER PTR_MPI2_CONFIG_REQUEST,
387   Mpi2ConfigRequest_t, MPI2_POINTER pMpi2ConfigRequest_t;

389 /* values for the Action field */
390 #define MPI2_CONFIG_ACTION_PAGE_HEADER            (0x00)
391 #define MPI2_CONFIG_ACTION_PAGE_READ_CURRENT      (0x01)

```

```

392 #define MPI2_CONFIG_ACTION_PAGE_WRITE_CURRENT      (0x02)
393 #define MPI2_CONFIG_ACTION_PAGE_DEFAULT            (0x03)
394 #define MPI2_CONFIG_ACTION_PAGE_WRITE_NVRAM        (0x04)
395 #define MPI2_CONFIG_ACTION_PAGE_READ_DEFAULT       (0x05)
396 #define MPI2_CONFIG_ACTION_PAGE_READ_NVRAM        (0x06)
397 #define MPI2_CONFIG_ACTION_PAGE_GET_CHANGEABLE     (0x07)

399 /* values for SGLFlags field are in the SGL section of mpi2.h */

402 /* Config Reply Message */
403 typedef struct _MPI2_CONFIG_REPLY
404 {
405     U8          Action;                /* 0x00 */
406     U8          SGLFlags;              /* 0x01 */
407     U8          MsgLength;             /* 0x02 */
408     U8          Function;              /* 0x03 */
409     U16         ExtPageLength;         /* 0x04 */
410     U8          ExtPageType;          /* 0x06 */
411     U8          MsgFlags;              /* 0x07 */
412     U8          VP_ID;                 /* 0x08 */
413     U8          VF_ID;                 /* 0x09 */
414     U16         Reserved1;             /* 0x0A */
415     U16         Reserved2;             /* 0x0C */
416     U16         IOCStatus;             /* 0x0E */
417     U32         IOCLogInfo;           /* 0x10 */
418     MPI2_CONFIG_PAGE_HEADER Header;   /* 0x14 */
419 } MPI2_CONFIG_REPLY, MPI2_POINTER PTR_MPI2_CONFIG_REPLY,
420   Mpi2ConfigReply_t, MPI2_POINTER pMpi2ConfigReply_t;

424 /*****
425 *
426 *           C o n f i g u r a t i o n   P a g e s
427 *
428 *****/

430 /*****
431 *   Manufacturing Config pages
432 *****/

434 #define MPI2_MFGPAGE_VENDORID_LSI                (0x1000)

436 /* MPI v2.0 SAS products */
437 #define MPI2_MFGPAGE_DEVID_SAS2004                (0x0070)
438 #define MPI2_MFGPAGE_DEVID_SAS2008                (0x0072)
439 #define MPI2_MFGPAGE_DEVID_SAS2108_1              (0x0074)
440 #define MPI2_MFGPAGE_DEVID_SAS2108_2              (0x0076)
441 #define MPI2_MFGPAGE_DEVID_SAS2108_3              (0x0077)
442 #define MPI2_MFGPAGE_DEVID_SAS2116_1              (0x0064)
443 #define MPI2_MFGPAGE_DEVID_SAS2116_2              (0x0065)

445 #define MPI2_MFGPAGE_DEVID_SSS6200                (0x007E)

447 #define MPI2_MFGPAGE_DEVID_SAS2208_1              (0x0080)
448 #define MPI2_MFGPAGE_DEVID_SAS2208_2              (0x0081)
449 #define MPI2_MFGPAGE_DEVID_SAS2208_3              (0x0082)
450 #define MPI2_MFGPAGE_DEVID_SAS2208_4              (0x0083)
451 #define MPI2_MFGPAGE_DEVID_SAS2208_5              (0x0084)
452 #define MPI2_MFGPAGE_DEVID_SAS2208_6              (0x0085)
453 #define MPI2_MFGPAGE_DEVID_SAS2208_7              (0x0086)
454 #define MPI2_MFGPAGE_DEVID_SAS2208_8              (0x0087)
455 #define MPI2_MFGPAGE_DEVID_SAS2308_1              (0x0086)
456 #define MPI2_MFGPAGE_DEVID_SAS2308_2              (0x0087)
457 #define MPI2_MFGPAGE_DEVID_SAS2308_3              (0x006E)

```

```

459 /* MPI v2.5 SAS products */
460 #define MPI25_MFGPAGE_DEVID_SAS3004                (0x0096)
461 #define MPI25_MFGPAGE_DEVID_SAS3008                (0x0097)
462 #define MPI25_MFGPAGE_DEVID_SAS3108_1              (0x0090)
463 #define MPI25_MFGPAGE_DEVID_SAS3108_2              (0x0091)
464 #define MPI25_MFGPAGE_DEVID_SAS3108_3              (0x0092)
465 #define MPI25_MFGPAGE_DEVID_SAS3108_4              (0x0093)
466 #define MPI25_MFGPAGE_DEVID_SAS3108_5              (0x0094)
467 #define MPI25_MFGPAGE_DEVID_SAS3108_6              (0x0095)

469 /* Manufacturing Page 0 */

471 typedef struct _MPI2_CONFIG_PAGE_MAN_0
472 {
473     MPI2_CONFIG_PAGE_HEADER Header;          /* 0x00 */
474     U8          ChipName[16];                /* 0x04 */
475     U8          ChipRevision[8];             /* 0x14 */
476     U8          BoardName[16];               /* 0x1C */
477     U8          BoardAssembly[16];           /* 0x2C */
478     U8          BoardTracerNumber[16];       /* 0x3C */
479 } MPI2_CONFIG_PAGE_MAN_0,
480   MPI2_POINTER PTR_MPI2_CONFIG_PAGE_MAN_0,
481   Mpi2ManufacturingPage0_t, MPI2_POINTER pMpi2ManufacturingPage0_t;

483 #define MPI2_MANUFACTURING0_PAGEVERSION            (0x00)

486 /* Manufacturing Page 1 */

488 typedef struct _MPI2_CONFIG_PAGE_MAN_1
489 {
490     MPI2_CONFIG_PAGE_HEADER Header;          /* 0x00 */
491     U8          VPD[256];                   /* 0x04 */
492 } MPI2_CONFIG_PAGE_MAN_1,
493   MPI2_POINTER PTR_MPI2_CONFIG_PAGE_MAN_1,
494   Mpi2ManufacturingPage1_t, MPI2_POINTER pMpi2ManufacturingPage1_t;

496 #define MPI2_MANUFACTURING1_PAGEVERSION            (0x00)

499 typedef struct _MPI2_CHIP_REVISION_ID
500 {
501     U16 DeviceID;                           /* 0x00 */
502     U8  PCIRevisionID;                      /* 0x02 */
503     U8  Reserved;                           /* 0x03 */
504 } MPI2_CHIP_REVISION_ID, MPI2_POINTER PTR_MPI2_CHIP_REVISION_ID,
505   Mpi2ChipRevisionId_t, MPI2_POINTER pMpi2ChipRevisionId_t;

508 /* Manufacturing Page 2 */

510 /*
511 * Host code (drivers, BIOS, utilities, etc.) should leave this define set to
512 * one and check Header.PageLength at runtime.
513 */
514 #ifndef MPI2_MAN_PAGE_2_HW_SETTINGS_WORDS
515 #define MPI2_MAN_PAGE_2_HW_SETTINGS_WORDS    (1)
516 #endif

518 typedef struct _MPI2_CONFIG_PAGE_MAN_2
519 {
520     MPI2_CONFIG_PAGE_HEADER Header;          /* 0x00 */
521     MPI2_CHIP_REVISION_ID  ChipId;           /* 0x04 */
522     U32                    HwSettings[MPI2_MAN_PAGE_2_HW_SETTINGS_WORDS]; /* 0x0
523 } MPI2_CONFIG_PAGE_MAN_2,

```

```

524 MPI2_POINTER PTR_MPI2_CONFIG_PAGE_MAN_2,
525 Mpi2ManufacturingPage2_t, MPI2_POINTER pMpi2ManufacturingPage2_t;

527 #define MPI2_MANUFACTURING2_PAGEVERSION          (0x00)

530 /* Manufacturing Page 3 */

532 /*
533  * Host code (drivers, BIOS, utilities, etc.) should leave this define set to
534  * one and check Header.PageLength at runtime.
535  */
536 #ifndef MPI2_MAN_PAGE_3_INFO_WORDS
537 #define MPI2_MAN_PAGE_3_INFO_WORDS                (1)
538 #endif

540 typedef struct _MPI2_CONFIG_PAGE_MAN_3
541 {
542     MPI2_CONFIG_PAGE_HEADER      Header;          /* 0x00 */
543     MPI2_CHIP_REVISION_ID        ChipId;          /* 0x04 */
544     U32                          Info[MPI2_MAN_PAGE_3_INFO_WORDS]; /* 0x08 */
545 } MPI2_CONFIG_PAGE_MAN_3,
546 MPI2_POINTER PTR_MPI2_CONFIG_PAGE_MAN_3,
547 Mpi2ManufacturingPage3_t, MPI2_POINTER pMpi2ManufacturingPage3_t;

549 #define MPI2_MANUFACTURING3_PAGEVERSION          (0x00)

552 /* Manufacturing Page 4 */

554 typedef struct _MPI2_MANPAGE4_PWR_SAVE_SETTINGS
555 {
556     U8                          PowerSaveFlags;    /* 0x00 */
557     U8                          InternalOperationsSleepTime; /* 0x01 */
558     U8                          InternalOperationsRunTime;  /* 0x02 */
559     U8                          HostIdleTime;         /* 0x03 */
560 } MPI2_MANPAGE4_PWR_SAVE_SETTINGS,
561 MPI2_POINTER PTR_MPI2_MANPAGE4_PWR_SAVE_SETTINGS,
562 Mpi2ManPage4PwrSaveSettings_t, MPI2_POINTER pMpi2ManPage4PwrSaveSettings_t;

564 /* defines for the PowerSaveFlags field */
565 #define MPI2_MANPAGE4_MASK_POWERSAVE_MODE          (0x03)
566 #define MPI2_MANPAGE4_POWERSAVE_MODE_DISABLED     (0x00)
567 #define MPI2_MANPAGE4_CUSTOM_POWERSAVE_MODE        (0x01)
568 #define MPI2_MANPAGE4_FULL_POWERSAVE_MODE          (0x02)

570 typedef struct _MPI2_CONFIG_PAGE_MAN_4
571 {
572     MPI2_CONFIG_PAGE_HEADER      Header;          /* 0x00 */
573     U32                          Reserved1;         /* 0x04 */
574     U32                          Flags;             /* 0x08 */
575     U8                          InquirySize;        /* 0x0C */
576     U8                          Reserved2;          /* 0x0D */
577     U16                         Reserved3;          /* 0x0E */
578     U8                          InquiryData[56];    /* 0x10 */
579     U32                         RAID0VolumeSettings; /* 0x48 */
580     U32                         RAID1VolumeSettings; /* 0x4C */
581     U32                         RAID1VolumeSettings; /* 0x50 */
582     U32                         RAID10VolumeSettings; /* 0x54 */
583     U32                         Reserved4;          /* 0x58 */
584     U32                         Reserved5;          /* 0x5C */
585     MPI2_MANPAGE4_PWR_SAVE_SETTINGS PowerSaveSettings; /* 0x60 */
586     U8                          MaxOCEDisks;       /* 0x64 */
587     U8                          ResyncRate;        /* 0x65 */
588     U16                         DataScrubDuration;  /* 0x66 */
589     U8                          MaxHotSpares;      /* 0x68 */

```

```

590     U8                          MaxPhysDisksPerVol; /* 0x69 */
591     U8                          MaxPhysDisks;      /* 0x6A */
592     U8                          MaxVolumes;         /* 0x6B */
593 } MPI2_CONFIG_PAGE_MAN_4,
594 MPI2_POINTER PTR_MPI2_CONFIG_PAGE_MAN_4,
595 Mpi2ManufacturingPage4_t, MPI2_POINTER pMpi2ManufacturingPage4_t;

597 #define MPI2_MANUFACTURING4_PAGEVERSION          (0x0A)

599 /* Manufacturing Page 4 Flags field */
600 #define MPI2_MANPAGE4_METADATA_SIZE_MASK        (0x00030000)
601 #define MPI2_MANPAGE4_METADATA_512MB           (0x00000000)

603 #define MPI2_MANPAGE4_MIX_SSD_SAS_SATA          (0x00008000)
604 #define MPI2_MANPAGE4_MIX_SSD_AND_NON_SSD       (0x00004000)
605 #define MPI2_MANPAGE4_HIDE_PHYSDISK_NON_IR      (0x00002000)

607 #define MPI2_MANPAGE4_MASK_PHYSDISK_COERCION    (0x00001C00)
608 #define MPI2_MANPAGE4_PHYSDISK_COERCION_1GB     (0x00000000)
609 #define MPI2_MANPAGE4_PHYSDISK_128MB_COERCION   (0x00000040)
610 #define MPI2_MANPAGE4_PHYSDISK_ADAPTIVE_COERCION (0x00000800)
611 #define MPI2_MANPAGE4_PHYSDISK_ZERO_COERCION    (0x00000C00)

613 #define MPI2_MANPAGE4_MASK_BAD_BLOCK_MARKING    (0x00000300)
614 #define MPI2_MANPAGE4_DEFAULT_BAD_BLOCK_MARKING (0x00000000)
615 #define MPI2_MANPAGE4_TABLE_BAD_BLOCK_MARKING   (0x00000100)
616 #define MPI2_MANPAGE4_WRITE_LONG_BAD_BLOCK_MARKING (0x00000200)

618 #define MPI2_MANPAGE4_FORCE_OFFLINE_FAILOVER    (0x00000080)
619 #define MPI2_MANPAGE4_RAID10_DISABLE            (0x00000040)
620 #define MPI2_MANPAGE4_RAID1_DISABLE            (0x00000020)
621 #define MPI2_MANPAGE4_RAID1_DISABLE            (0x00000010)
622 #define MPI2_MANPAGE4_RAID0_DISABLE            (0x00000008)
623 #define MPI2_MANPAGE4_IR_MODEPAGE8_DISABLE     (0x00000004)
624 #define MPI2_MANPAGE4_IM_RESYNC_CACHE_ENABLE   (0x00000002)
625 #define MPI2_MANPAGE4_IR_NO_MIX_SAS_SATA       (0x00000001)

628 /* Manufacturing Page 5 */

630 /*
631  * Host code (drivers, BIOS, utilities, etc.) should leave this define set to
632  * one and check Header.PageLength or NumPhys at runtime.
633  */
634 #ifndef MPI2_MAN_PAGE_5_PHY_ENTRIES
635 #define MPI2_MAN_PAGE_5_PHY_ENTRIES              (1)
636 #endif

638 typedef struct _MPI2_MANUFACTURING5_ENTRY
639 {
640     U64                          WWID;              /* 0x00 */
641     U64                          DeviceName;        /* 0x08 */
642 } MPI2_MANUFACTURING5_ENTRY, MPI2_POINTER PTR_MPI2_MANUFACTURING5_ENTRY,
643 Mpi2Manufacturing5Entry_t, MPI2_POINTER pMpi2Manufacturing5Entry_t;

645 typedef struct _MPI2_CONFIG_PAGE_MAN_5
646 {
647     MPI2_CONFIG_PAGE_HEADER      Header;          /* 0x00 */
648     U8                          NumPhys;           /* 0x04 */
649     U8                          Reserved1;          /* 0x05 */
650     U16                         Reserved2;          /* 0x06 */
651     U32                         Reserved3;          /* 0x08 */
652     U32                         Reserved4;          /* 0x0C */
653     MPI2_MANUFACTURING5_ENTRY    Phy[MPI2_MAN_PAGE_5_PHY_ENTRIES]; /* 0x08 */
654 } MPI2_CONFIG_PAGE_MAN_5,
655 MPI2_POINTER PTR_MPI2_CONFIG_PAGE_MAN_5,

```

```

656  MPI2ManufacturingPage5_t, MPI2_POINTER pMpi2ManufacturingPage5_t;

658 #define MPI2_MANUFACTURING5_PAGEVERSION                (0x03)

661 /* Manufacturing Page 6 */

663 typedef struct _MPI2_CONFIG_PAGE_MAN_6
664 {
665     MPI2_CONFIG_PAGE_HEADER    Header;          /* 0x00 */
666     U32                        ProductSpecificInfo; /* 0x04 */
667 } MPI2_CONFIG_PAGE_MAN_6,
668 MPI2_POINTER PTR_MPI2_CONFIG_PAGE_MAN_6,
669 Mpi2ManufacturingPage6_t, MPI2_POINTER pMpi2ManufacturingPage6_t;

671 #define MPI2_MANUFACTURING6_PAGEVERSION                (0x00)

674 /* Manufacturing Page 7 */

676 typedef struct _MPI2_MANPAGE7_CONNECTOR_INFO
677 {
678     U32                        Pinout;            /* 0x00 */
679     U8                        Connector[16];      /* 0x04 */
680     U8                        Location;           /* 0x14 */
681     U8                        ReceptacleID;       /* 0x15 */
682     U16                       Slot;              /* 0x16 */
683     U32                        Reserved2;         /* 0x18 */
684 } MPI2_MANPAGE7_CONNECTOR_INFO, MPI2_POINTER PTR_MPI2_MANPAGE7_CONNECTOR_INFO,
685 Mpi2ManPage7ConnectorInfo_t, MPI2_POINTER pMpi2ManPage7ConnectorInfo_t;

687 /* defines for the Pinout field */
688 #define MPI2_MANPAGE7_PINOUT_SFF_8484_L4                (0x00080000)
689 #define MPI2_MANPAGE7_PINOUT_SFF_8484_L3                (0x00040000)
690 #define MPI2_MANPAGE7_PINOUT_SFF_8484_L2                (0x00020000)
691 #define MPI2_MANPAGE7_PINOUT_SFF_8484_L1                (0x00010000)
692 #define MPI2_MANPAGE7_PINOUT_SFF_8470_L4                (0x00000800)
693 #define MPI2_MANPAGE7_PINOUT_SFF_8470_L3                (0x00000400)
694 #define MPI2_MANPAGE7_PINOUT_SFF_8470_L2                (0x00000200)
695 #define MPI2_MANPAGE7_PINOUT_SFF_8470_L1                (0x00000100)
696 #define MPI2_MANPAGE7_PINOUT_SFF_8482                  (0x00000002)
697 #define MPI2_MANPAGE7_PINOUT_CONNECTION_UNKNOWN        (0x00000001)

699 /* defines for the Location field */
700 #define MPI2_MANPAGE7_LOCATION_UNKNOWN                  (0x01)
701 #define MPI2_MANPAGE7_LOCATION_INTERNAL                 (0x02)
702 #define MPI2_MANPAGE7_LOCATION_EXTERNAL                (0x04)
703 #define MPI2_MANPAGE7_LOCATION_SWITCHABLE              (0x08)
704 #define MPI2_MANPAGE7_LOCATION_AUTO                    (0x10)
705 #define MPI2_MANPAGE7_LOCATION_NOT_PRESENT             (0x20)
706 #define MPI2_MANPAGE7_LOCATION_NOT_CONNECTED           (0x80)

708 /*
709  * Host code (drivers, BIOS, utilities, etc.) should leave this define set to
710  * one and check NumPhys at runtime.
711  */
712 #ifndef MPI2_MANPAGE7_CONNECTOR_INFO_MAX
713 #define MPI2_MANPAGE7_CONNECTOR_INFO_MAX (1)
714 #endif

716 typedef struct _MPI2_CONFIG_PAGE_MAN_7
717 {
718     MPI2_CONFIG_PAGE_HEADER    Header;          /* 0x00 */
719     U32                        Reserved1;        /* 0x04 */
720     U32                        Reserved2;        /* 0x08 */
721     U32                        Flags;            /* 0x0C */

```

```

722     U8                        EnclosureName[16]; /* 0x10 */
723     U8                        NumPhys;          /* 0x20 */
724     U8                        Reserved3;        /* 0x21 */
725     U16                       Reserved4;        /* 0x22 */
726     MPI2_MANPAGE7_CONNECTOR_INFO ConnectorInfo[MPI2_MANPAGE7_CONNECTOR_INFO_M
727 } MPI2_CONFIG_PAGE_MAN_7,
728 MPI2_POINTER PTR_MPI2_CONFIG_PAGE_MAN_7,
729 Mpi2ManufacturingPage7_t, MPI2_POINTER pMpi2ManufacturingPage7_t;

731 #define MPI2_MANUFACTURING7_PAGEVERSION                (0x01)

733 /* defines for the Flags field */
734 #define MPI2_MANPAGE7_FLAG_USE_SLOT_INFO                (0x00000001)

737 /*
738  * Generic structure to use for product-specific manufacturing pages
739  * (currently Manufacturing Page 8 through Manufacturing Page 31).
740  */

742 typedef struct _MPI2_CONFIG_PAGE_MAN_PS
743 {
744     MPI2_CONFIG_PAGE_HEADER    Header;          /* 0x00 */
745     U32                        ProductSpecificInfo; /* 0x04 */
746 } MPI2_CONFIG_PAGE_MAN_PS,
747 MPI2_POINTER PTR_MPI2_CONFIG_PAGE_MAN_PS,
748 Mpi2ManufacturingPagePS_t, MPI2_POINTER pMpi2ManufacturingPagePS_t;

750 #define MPI2_MANUFACTURING8_PAGEVERSION                (0x00)
751 #define MPI2_MANUFACTURING9_PAGEVERSION                (0x00)
752 #define MPI2_MANUFACTURING10_PAGEVERSION               (0x00)
753 #define MPI2_MANUFACTURING11_PAGEVERSION               (0x00)
754 #define MPI2_MANUFACTURING12_PAGEVERSION               (0x00)
755 #define MPI2_MANUFACTURING13_PAGEVERSION               (0x00)
756 #define MPI2_MANUFACTURING14_PAGEVERSION               (0x00)
757 #define MPI2_MANUFACTURING15_PAGEVERSION               (0x00)
758 #define MPI2_MANUFACTURING16_PAGEVERSION               (0x00)
759 #define MPI2_MANUFACTURING17_PAGEVERSION               (0x00)
760 #define MPI2_MANUFACTURING18_PAGEVERSION               (0x00)
761 #define MPI2_MANUFACTURING19_PAGEVERSION               (0x00)
762 #define MPI2_MANUFACTURING20_PAGEVERSION               (0x00)
763 #define MPI2_MANUFACTURING21_PAGEVERSION               (0x00)
764 #define MPI2_MANUFACTURING22_PAGEVERSION               (0x00)
765 #define MPI2_MANUFACTURING23_PAGEVERSION               (0x00)
766 #define MPI2_MANUFACTURING24_PAGEVERSION               (0x00)
767 #define MPI2_MANUFACTURING25_PAGEVERSION               (0x00)
768 #define MPI2_MANUFACTURING26_PAGEVERSION               (0x00)
769 #define MPI2_MANUFACTURING27_PAGEVERSION               (0x00)
770 #define MPI2_MANUFACTURING28_PAGEVERSION               (0x00)
771 #define MPI2_MANUFACTURING29_PAGEVERSION               (0x00)
772 #define MPI2_MANUFACTURING30_PAGEVERSION               (0x00)
773 #define MPI2_MANUFACTURING31_PAGEVERSION               (0x00)

776 /*****
777  * IO Unit Config Pages
778  *****/

780 /* IO Unit Page 0 */

782 typedef struct _MPI2_CONFIG_PAGE_IO_UNIT_0
783 {
784     MPI2_CONFIG_PAGE_HEADER    Header;          /* 0x00 */
785     U64                        UniqueValue;      /* 0x04 */
786     MPI2_VERSION_UNION         NvdataVersionDefault; /* 0x08 */
787     MPI2_VERSION_UNION         NvdataVersionPersistent; /* 0x0A */

```

```

788 } MPI2_CONFIG_PAGE_IO_UNIT_0, MPI2_POINTER PTR_MPI2_CONFIG_PAGE_IO_UNIT_0,
789   Mpi2IOUnitPage0_t, MPI2_POINTER pMpi2IOUnitPage0_t;

791 #define MPI2_IUNITPAGE0_PAGEVERSION          (0x02)

794 /* IO Unit Page 1 */

796 typedef struct _MPI2_CONFIG_PAGE_IO_UNIT_1
797 {
798     MPI2_CONFIG_PAGE_HEADER Header;          /* 0x00 */
799     U32 Flags;                               /* 0x04 */
800 } MPI2_CONFIG_PAGE_IO_UNIT_1, MPI2_POINTER PTR_MPI2_CONFIG_PAGE_IO_UNIT_1,
801   Mpi2IOUnitPage1_t, MPI2_POINTER pMpi2IOUnitPage1_t;

803 #define MPI2_IUNITPAGE1_PAGEVERSION          (0x04)

805 /* IO Unit Page 1 Flags defines */
806 #define MPI25_IUNITPAGE1_NEW_DEVICE_FAST_PATH_DISABLE (0x00002000)
807 #define MPI25_IUNITPAGE1_DISABLE_FAST_PATH          (0x00001000)
808 #define MPI2_IUNITPAGE1_ENABLE_HOST_BASED_DISCOVERY (0x00000800)
809 #define MPI2_IUNITPAGE1_MASK_SATA_WRITE_CACHE       (0x00000600)
810 #define MPI2_IUNITPAGE1_SATA_WRITE_CACHE_SHIFT      (9)
811 #define MPI2_IUNITPAGE1_ENABLE_SATA_WRITE_CACHE     (0x00000000)
812 #define MPI2_IUNITPAGE1_DISABLE_SATA_WRITE_CACHE    (0x00000200)
813 #define MPI2_IUNITPAGE1_UNCHANGED_SATA_WRITE_CACHE  (0x00000400)
814 #define MPI2_IUNITPAGE1_NATIVE_COMMAND_Q_DISABLE   (0x00000100)
815 #define MPI2_IUNITPAGE1_DISABLE_IR                  (0x00000040)
816 #define MPI2_IUNITPAGE1_DISABLE_TASK_SET_FULL_HANDLING (0x00000020)
817 #define MPI2_IUNITPAGE1_IR_USE_STATIC_VOLUME_ID     (0x00000004)
818 #define MPI2_IUNITPAGE1_MULTI_PATHING               (0x00000002)
819 #define MPI2_IUNITPAGE1_SINGLE_PATHING              (0x00000000)

822 /* IO Unit Page 3 */

824 /*
825  * Host code (drivers, BIOS, utilities, etc.) should leave this define set to
826  * one and check Header.PageLength at runtime.
827  */
828 #ifndef MPI2_IO_UNIT_PAGE_3_GPIO_VAL_MAX
829 #define MPI2_IO_UNIT_PAGE_3_GPIO_VAL_MAX      (1)
830 #endif

832 typedef struct _MPI2_CONFIG_PAGE_IO_UNIT_3
833 {
834     MPI2_CONFIG_PAGE_HEADER Header;          /* 0x00 */
835     U8 GPIOCount;                            /* 0x04 */
836     U8 Reserved1;                            /* 0x05 */
837     U16 Reserved2;                           /* 0x06 */
838     U16 GPIOVal[MPI2_IO_UNIT_PAGE_3_GPIO_VAL_MAX]; /* 0x08 */
839 } MPI2_CONFIG_PAGE_IO_UNIT_3, MPI2_POINTER PTR_MPI2_CONFIG_PAGE_IO_UNIT_3,
840   Mpi2IOUnitPage3_t, MPI2_POINTER pMpi2IOUnitPage3_t;

842 #define MPI2_IUNITPAGE3_PAGEVERSION          (0x01)

844 /* defines for IO Unit Page 3 GPIOVal field */
845 #define MPI2_IUNITPAGE3_GPIO_FUNCTION_MASK    (0xFFFFC)
846 #define MPI2_IUNITPAGE3_GPIO_FUNCTION_SHIFT   (2)
847 #define MPI2_IUNITPAGE3_GPIO_SETTING_OFF      (0x0000)
848 #define MPI2_IUNITPAGE3_GPIO_SETTING_ON       (0x0001)

851 /* IO Unit Page 5 */
853 /*

```

```

854 * Upper layer code (drivers, utilities, etc.) should leave this define set to
855 * one and check Header.PageLength or NumDmaEngines at runtime.
856 */
857 #ifndef MPI2_IUNITPAGE5_DMAENGINE_ENTRIES
858 #define MPI2_IUNITPAGE5_DMAENGINE_ENTRIES      (1)
859 #endif

861 typedef struct _MPI2_CONFIG_PAGE_IO_UNIT_5
862 {
863     MPI2_CONFIG_PAGE_HEADER Header;          /* 0x00 */
864     U64 RaidAcceleratorBufferBaseAddress;    /* 0x04 */
865     U64 RaidAcceleratorBufferSize;           /* 0x0C */
866     U64 RaidAcceleratorControlBaseAddress;    /* 0x14 */
867     U8 RAControlSize;                         /* 0x1C */
868     U8 NumDmaEngines;                         /* 0x1D */
869     U8 RAMinControlSize;                      /* 0x1E */
870     U8 RAMaxControlSize;                      /* 0x1F */
871     U32 Reserved1;                            /* 0x20 */
872     U32 Reserved2;                            /* 0x24 */
873     U32 Reserved3;                            /* 0x28 */
874     U32 DmaEngineCapabilities[MPI2_IUNITPAGE5_DMAENGINE_ENTRIES];
875 } MPI2_CONFIG_PAGE_IO_UNIT_5, MPI2_POINTER PTR_MPI2_CONFIG_PAGE_IO_UNIT_5,
876   Mpi2IOUnitPage5_t, MPI2_POINTER pMpi2IOUnitPage5_t;

878 #define MPI2_IUNITPAGE5_PAGEVERSION          (0x00)

880 /* defines for IO Unit Page 5 DmaEngineCapabilities field */
881 #define MPI2_IUNITPAGE5_DMA_CAP_MASK_MAX_REQUESTS (0xFF00)
882 #define MPI2_IUNITPAGE5_DMA_CAP_SHIFT_MAX_REQUESTS (16)

884 #define MPI2_IUNITPAGE5_DMA_CAP_EEDP            (0x0008)
885 #define MPI2_IUNITPAGE5_DMA_CAP_PARITY_GENERATION (0x0004)
886 #define MPI2_IUNITPAGE5_DMA_CAP_HASHING         (0x0002)
887 #define MPI2_IUNITPAGE5_DMA_CAP_ENCRYPTION       (0x0001)

890 /* IO Unit Page 6 */

892 typedef struct _MPI2_CONFIG_PAGE_IO_UNIT_6
893 {
894     MPI2_CONFIG_PAGE_HEADER Header;          /* 0x00 */
895     U16 Flags;                               /* 0x04 */
896     U8 RAHostControlSize;                    /* 0x06 */
897     U8 Reserved0;                             /* 0x07 */
898     U64 RaidAcceleratorHostControlBaseAddress; /* 0x08 */
899     U32 Reserved1;                            /* 0x10 */
900     U32 Reserved2;                            /* 0x14 */
901     U32 Reserved3;                            /* 0x18 */
902 } MPI2_CONFIG_PAGE_IO_UNIT_6, MPI2_POINTER PTR_MPI2_CONFIG_PAGE_IO_UNIT_6,
903   Mpi2IOUnitPage6_t, MPI2_POINTER pMpi2IOUnitPage6_t;

905 #define MPI2_IUNITPAGE6_PAGEVERSION          (0x00)

907 /* defines for IO Unit Page 6 Flags field */
908 #define MPI2_IUNITPAGE6_FLAGS_ENABLE_RAID_ACCELERATOR (0x0001)

911 /* IO Unit Page 7 */

913 typedef struct _MPI2_CONFIG_PAGE_IO_UNIT_7
914 {
915     MPI2_CONFIG_PAGE_HEADER Header;          /* 0x00 */
916     U16 Reserved1;                           /* 0x04 */
917     U8 PCIEWidth;                            /* 0x06 */
918     U8 PCIESpeed;                            /* 0x07 */
919     U32 ProcessorState;                       /* 0x08 */

```

```

920     U32                PowerManagementCapabilities;    /* 0x0C */
921     U16                IOCTemperature;                /* 0x10 */
922     U8                 IOCTemperatureUnits;           /* 0x12 */
923     U8                 IOCSpeed;                      /* 0x13 */
924     U16                BoardTemperature;              /* 0x14 */
925     U8                 BoardTemperatureUnits;         /* 0x16 */
926     U8                 Reserved3;                     /* 0x17 */
927 } MPI2_CONFIG_PAGE_IO_UNIT_7, MPI2_POINTER PTR_MPI2_CONFIG_PAGE_IO_UNIT_7,
928   Mpi2IOUnitPage7_t, MPI2_POINTER pMpi2IOUnitPage7_t;

930 #define MPI2_IOUNITPAGE7_PAGEVERSION                (0x02)

932 /* defines for IO Unit Page 7 PCIEWidth field */
933 #define MPI2_IOUNITPAGE7_PCIE_WIDTH_X1              (0x01)
934 #define MPI2_IOUNITPAGE7_PCIE_WIDTH_X2              (0x02)
935 #define MPI2_IOUNITPAGE7_PCIE_WIDTH_X4              (0x04)
936 #define MPI2_IOUNITPAGE7_PCIE_WIDTH_X8              (0x08)

938 /* defines for IO Unit Page 7 PCIESpeed field */
939 #define MPI2_IOUNITPAGE7_PCIE_SPEED_2_5_GBPS        (0x00)
940 #define MPI2_IOUNITPAGE7_PCIE_SPEED_5_0_GBPS        (0x01)
941 #define MPI2_IOUNITPAGE7_PCIE_SPEED_8_0_GBPS        (0x02)

943 /* defines for IO Unit Page 7 ProcessorState field */
944 #define MPI2_IOUNITPAGE7_PSTATE_MASK_SECOND         (0x0000000F)
945 #define MPI2_IOUNITPAGE7_PSTATE_SHIFT_SECOND        (0)

947 #define MPI2_IOUNITPAGE7_PSTATE_NOT_PRESENT         (0x00)
948 #define MPI2_IOUNITPAGE7_PSTATE_DISABLED            (0x01)
949 #define MPI2_IOUNITPAGE7_PSTATE_ENABLED             (0x02)

951 /* defines for IO Unit Page 7 PowerManagementCapabilities field */
952 #define MPI2_IOUNITPAGE7_PMCAP_12_5_PCT_IOCSPD      (0x00000400)
953 #define MPI2_IOUNITPAGE7_PMCAP_25_0_PCT_IOCSPD      (0x00000200)
954 #define MPI2_IOUNITPAGE7_PMCAP_50_0_PCT_IOCSPD      (0x00000100)
955 #define MPI2_IOUNITPAGE7_PMCAP_PCIE_WIDTH_CHANGE    (0x00000008)
956 #define MPI2_IOUNITPAGE7_PMCAP_PCIE_SPEED_CHANGE    (0x00000004)

958 /* defines for IO Unit Page 7 IOCTemperatureUnits field */
959 #define MPI2_IOUNITPAGE7_IOC_TEMP_NOT_PRESENT        (0x00)
960 #define MPI2_IOUNITPAGE7_IOC_TEMP_FAHRENHEIT         (0x01)
961 #define MPI2_IOUNITPAGE7_IOC_TEMP_CELSIUS            (0x02)

963 /* defines for IO Unit Page 7 IOCSpeed field */
964 #define MPI2_IOUNITPAGE7_IOC_SPEED_FULL              (0x01)
965 #define MPI2_IOUNITPAGE7_IOC_SPEED_HALF              (0x02)
966 #define MPI2_IOUNITPAGE7_IOC_SPEED_QUARTER           (0x04)
967 #define MPI2_IOUNITPAGE7_IOC_SPEED_EIGHTH            (0x08)

969 /* defines for IO Unit Page 7 BoardTemperatureUnits field */
970 #define MPI2_IOUNITPAGE7_BOARD_TEMP_NOT_PRESENT      (0x00)
971 #define MPI2_IOUNITPAGE7_BOARD_TEMP_FAHRENHEIT       (0x01)
972 #define MPI2_IOUNITPAGE7_BOARD_TEMP_CELSIUS          (0x02)

975 /*****
976  *   IOC Config Pages
977  *****/

979 /* IOC Page 0 */

981 typedef struct _MPI2_CONFIG_PAGE_IOC_0
982 {
983     MPI2_CONFIG_PAGE_HEADER Header;          /* 0x00 */
984     U32                Reserved1;            /* 0x04 */
985     U32                Reserved2;            /* 0x08 */

```

```

986     U16                VendorID;              /* 0x0C */
987     U16                DeviceID;             /* 0x0E */
988     U8                 RevisionID;           /* 0x10 */
989     U8                 Reserved3;            /* 0x11 */
990     U16                Reserved4;            /* 0x12 */
991     U32                ClassCode;            /* 0x14 */
992     U16                SubsystemVendorID;     /* 0x18 */
993     U16                SubsystemID;          /* 0x1A */
994 } MPI2_CONFIG_PAGE_IOC_0, MPI2_POINTER PTR_MPI2_CONFIG_PAGE_IOC_0,
995   Mpi2IOCPage0_t, MPI2_POINTER pMpi2IOCPage0_t;

997 #define MPI2_IOCPAGE0_PAGEVERSION                (0x02)

1000 /* IOC Page 1 */

1002 typedef struct _MPI2_CONFIG_PAGE_IOC_1
1003 {
1004     MPI2_CONFIG_PAGE_HEADER Header;          /* 0x00 */
1005     U32                Flags;                /* 0x04 */
1006     U32                CoalescingTimeout;     /* 0x08 */
1007     U8                 CoalescingDepth;       /* 0x0C */
1008     U8                 PCISlotNum;           /* 0x0D */
1009     U8                 PCIBusNum;            /* 0x0E */
1010     U8                 PCIDomainSegment;     /* 0x0F */
1011     U32                Reserved1;            /* 0x10 */
1012     U32                Reserved2;            /* 0x14 */
1013 } MPI2_CONFIG_PAGE_IOC_1, MPI2_POINTER PTR_MPI2_CONFIG_PAGE_IOC_1,
1014   Mpi2IOCPage1_t, MPI2_POINTER pMpi2IOCPage1_t;

1016 #define MPI2_IOCPAGE1_PAGEVERSION                (0x05)

1018 /* defines for IOC Page 1 Flags field */
1019 #define MPI2_IOCPAGE1_REPLY_COALESCING              (0x00000001)

1021 #define MPI2_IOCPAGE1_PCISLOTNUM_UNKNOWN            (0xFF)
1022 #define MPI2_IOCPAGE1_PCIBUSNUM_UNKNOWN            (0xFF)
1023 #define MPI2_IOCPAGE1_PCIDOMAIN_UNKNOWN            (0xFF)

1025 /* IOC Page 6 */

1027 typedef struct _MPI2_CONFIG_PAGE_IOC_6
1028 {
1029     MPI2_CONFIG_PAGE_HEADER Header;          /* 0x00 */
1030     U32                CapabilitiesFlags;     /* 0x04 */
1031     U8                 MaxDrivesRAID0;        /* 0x08 */
1032     U8                 MaxDrivesRAID1;        /* 0x09 */
1033     U8                 MaxDrivesRAID1E;       /* 0x0A */
1034     U8                 MaxDrivesRAID10;       /* 0x0B */
1035     U8                 MinDrivesRAID0;        /* 0x0C */
1036     U8                 MinDrivesRAID1;        /* 0x0D */
1037     U8                 MinDrivesRAID1E;       /* 0x0E */
1038     U8                 MinDrivesRAID10;       /* 0x0F */
1039     U32                Reserved1;            /* 0x10 */
1040     U8                 MaxGlobalHotSpares;    /* 0x14 */
1041     U8                 MaxPhysDisks;          /* 0x15 */
1042     U8                 MaxVolumes;            /* 0x16 */
1043     U8                 MaxConfigs;            /* 0x17 */
1044     U8                 MaxOCEDisks;           /* 0x18 */
1045     U8                 Reserved2;            /* 0x19 */
1046     U16                Reserved3;            /* 0x1A */
1047     U32                SupportedStripesSizeMapRAID0; /* 0x1C */
1048     U32                SupportedStripesSizeMapRAID1E; /* 0x20 */
1049     U32                SupportedStripesSizeMapRAID10; /* 0x24 */
1050     U32                Reserved4;            /* 0x28 */
1051     U32                Reserved5;            /* 0x2C */

```



```

1052     U16          DefaultMetadataSize;          /* 0x30 */
1053     U16          Reserved6;                     /* 0x32 */
1054     U16          MaxBadBlockTableEntries;       /* 0x34 */
1055     U16          Reserved7;                     /* 0x36 */
1056     U32          IRNvsramVersion;               /* 0x38 */
1057 } MPI2_CONFIG_PAGE_IOC_6, MPI2_POINTER PTR_MPI2_CONFIG_PAGE_IOC_6,
1058   Mpi2IOCPage6_t, MPI2_POINTER pMpi2IOCPage6_t;

1060 #define MPI2_IOC_PAGE6_PAGEVERSION              (0x04)

1062 /* defines for IOC Page 6 CapabilitiesFlags */
1063 #define MPI2_IOC_PAGE6_CAP_FLAGS_RAID10_SUPPORT (0x00000010)
1064 #define MPI2_IOC_PAGE6_CAP_FLAGS_RAID1_SUPPORT (0x00000008)
1065 #define MPI2_IOC_PAGE6_CAP_FLAGS_RAID1E_SUPPORT (0x00000004)
1066 #define MPI2_IOC_PAGE6_CAP_FLAGS_RAID0_SUPPORT (0x00000002)
1067 #define MPI2_IOC_PAGE6_CAP_FLAGS_GLOBAL_HOT_SPARE (0x00000001)

1070 /* IOC Page 7 */

1072 #define MPI2_IOC_PAGE7_EVENTMASK_WORDS          (4)

1074 typedef struct _MPI2_CONFIG_PAGE_IOC_7
1075 {
1076     MPI2_CONFIG_PAGE_HEADER Header;              /* 0x00 */
1077     U32          Reserved1;                       /* 0x04 */
1078     U32          EventMasks[MPI2_IOC_PAGE7_EVENTMASK_WORDS]; /* 0x08 */
1079     U16          SASBroadcastPrimitiveMasks;      /* 0x18 */
1080     U16          Reserved2;                       /* 0x1A */
1081     U32          Reserved3;                       /* 0x1C */
1082 } MPI2_CONFIG_PAGE_IOC_7, MPI2_POINTER PTR_MPI2_CONFIG_PAGE_IOC_7,
1083   Mpi2IOCPage7_t, MPI2_POINTER pMpi2IOCPage7_t;

1085 #define MPI2_IOC_PAGE7_PAGEVERSION              (0x01)

1088 /* IOC Page 8 */

1090 typedef struct _MPI2_CONFIG_PAGE_IOC_8
1091 {
1092     MPI2_CONFIG_PAGE_HEADER Header;              /* 0x00 */
1093     U8          NumDevsPerEnclosure;              /* 0x04 */
1094     U8          Reserved1;                       /* 0x05 */
1095     U16          Reserved2;                       /* 0x06 */
1096     U16          MaxPersistentEntries;            /* 0x08 */
1097     U16          MaxNumPhysicalMappedIDs;         /* 0x0A */
1098     U16          Flags;                          /* 0x0C */
1099     U16          Reserved3;                       /* 0x0E */
1100     U16          IRVolumeMappingFlags;           /* 0x10 */
1101     U16          Reserved4;                       /* 0x12 */
1102     U32          Reserved5;                       /* 0x14 */
1103 } MPI2_CONFIG_PAGE_IOC_8, MPI2_POINTER PTR_MPI2_CONFIG_PAGE_IOC_8,
1104   Mpi2IOCPage8_t, MPI2_POINTER pMpi2IOCPage8_t;

1106 #define MPI2_IOC_PAGE8_PAGEVERSION              (0x00)

1108 /* defines for IOC Page 8 Flags field */
1109 #define MPI2_IOC_PAGE8_FLAGS_DA_START_SLOT_1    (0x00000020)
1110 #define MPI2_IOC_PAGE8_FLAGS_RESERVED_TARGETID_0 (0x00000010)

1112 #define MPI2_IOC_PAGE8_FLAGS_MASK_MAPPING_MODE (0x0000000E)
1113 #define MPI2_IOC_PAGE8_FLAGS_DEVICE_PERSISTENCE_MAPPING (0x00000000)
1114 #define MPI2_IOC_PAGE8_FLAGS_ENCLOSURE_SLOT_MAPPING (0x00000002)

1116 #define MPI2_IOC_PAGE8_FLAGS_DISABLE_PERSISTENT_MAPPING (0x00000001)
1117 #define MPI2_IOC_PAGE8_FLAGS_ENABLE_PERSISTENT_MAPPING (0x00000000)

```

```

1119 /* defines for IOC Page 8 IRVolumeMappingFlags */
1120 #define MPI2_IOC_PAGE8_IRFLAGS_MASK_VOLUME_MAPPING_MODE (0x00000003)
1121 #define MPI2_IOC_PAGE8_IRFLAGS_LOW_VOLUME_MAPPING (0x00000000)
1122 #define MPI2_IOC_PAGE8_IRFLAGS_HIGH_VOLUME_MAPPING (0x00000001)

1125 /*****
1126  *   BIOS Config Pages
1127  *****/

1129 /* BIOS Page 1 */

1131 typedef struct _MPI2_CONFIG_PAGE_BIOS_1
1132 {
1133     MPI2_CONFIG_PAGE_HEADER Header;              /* 0x00 */
1134     U32          BiosOptions;                    /* 0x04 */
1135     U32          IOCSettings;                    /* 0x08 */
1136     U32          Reserved1;                      /* 0x0C */
1137     U32          DeviceSettings;                 /* 0x10 */
1138     U16          NumberOfDevices;                /* 0x14 */
1139     U16          Reserved2;                      /* 0x16 */
1140     U16          IOTimeoutBlockDevicesNonRM;     /* 0x18 */
1141     U16          IOTimeoutSequential;            /* 0x1A */
1142     U16          IOTimeoutOther;                 /* 0x1C */
1143     U16          IOTimeoutBlockDevicesRM;        /* 0x1E */
1144 } MPI2_CONFIG_PAGE_BIOS_1, MPI2_POINTER PTR_MPI2_CONFIG_PAGE_BIOS_1,
1145   Mpi2BiosPage1_t, MPI2_POINTER pMpi2BiosPage1_t;

1147 #define MPI2_BIOS_PAGE1_PAGEVERSION              (0x04)

1149 /* values for BIOS Page 1 BiosOptions field */
1150 #define MPI2_BIOS_PAGE1_OPTIONS_DISABLE_BIOS    (0x00000001)

1152 /* values for BIOS Page 1 IOCSettings field */
1153 #define MPI2_BIOS_PAGE1_IOCSET_MASK_BOOT_PREFERENCE (0x00030000)
1154 #define MPI2_BIOS_PAGE1_IOCSET_ENCLOSURE_SLOT_BOOT (0x00000000)
1155 #define MPI2_BIOS_PAGE1_IOCSET_SAS_ADDRESS_BOOT (0x00010000)

1157 #define MPI2_BIOS_PAGE1_IOCSET_MASK_RM_SETTING (0x000000C0)
1158 #define MPI2_BIOS_PAGE1_IOCSET_NONE_RM_SETTING (0x00000000)
1159 #define MPI2_BIOS_PAGE1_IOCSET_BOOT_RM_SETTING (0x00000040)
1160 #define MPI2_BIOS_PAGE1_IOCSET_MEDIA_RM_SETTING (0x00000080)

1162 #define MPI2_BIOS_PAGE1_IOCSET_MASK_ADAPTER_SUPPORT (0x00000030)
1163 #define MPI2_BIOS_PAGE1_IOCSET_NO_SUPPORT (0x00000000)
1164 #define MPI2_BIOS_PAGE1_IOCSET_BIOS_SUPPORT (0x00000010)
1165 #define MPI2_BIOS_PAGE1_IOCSET_OS_SUPPORT (0x00000020)
1166 #define MPI2_BIOS_PAGE1_IOCSET_ALL_SUPPORT (0x00000030)

1168 #define MPI2_BIOS_PAGE1_IOCSET_ALTERNATE_CHS    (0x00000008)

1170 /* values for BIOS Page 1 DeviceSettings field */
1171 #define MPI2_BIOS_PAGE1_DEVSET_DISABLE_SMART_POLLING (0x00000010)
1172 #define MPI2_BIOS_PAGE1_DEVSET_DISABLE_SEQ_LUN (0x00000008)
1173 #define MPI2_BIOS_PAGE1_DEVSET_DISABLE_RM_LUN (0x00000004)
1174 #define MPI2_BIOS_PAGE1_DEVSET_DISABLE_NON_RM_LUN (0x00000002)
1175 #define MPI2_BIOS_PAGE1_DEVSET_DISABLE_OTHER_LUN (0x00000001)

1178 /* BIOS Page 2 */

1180 typedef struct _MPI2_BOOT_DEVICE_ADAPTER_ORDER
1181 {
1182     U32          Reserved1;                      /* 0x00 */
1183     U32          Reserved2;                      /* 0x04 */

```

```

1184     U32         Reserved3;                /* 0x08 */
1185     U32         Reserved4;                /* 0x0C */
1186     U32         Reserved5;                /* 0x10 */
1187     U32         Reserved6;                /* 0x14 */
1188 } MPI2_BOOT_DEVICE_ADAPTER_ORDER,
1189 MPI2_POINTER PTR_MPI2_BOOT_DEVICE_ADAPTER_ORDER,
1190 Mpi2BootDeviceAdapterOrder_t, MPI2_POINTER pMpi2BootDeviceAdapterOrder_t;

1192 typedef struct _MPI2_BOOT_DEVICE_SAS_WWID
1193 {
1194     U64         SASAddress;                /* 0x00 */
1195     U8          LUN[8];                    /* 0x08 */
1196     U32         Reserved1;                /* 0x10 */
1197     U32         Reserved2;                /* 0x14 */
1198 } MPI2_BOOT_DEVICE_SAS_WWID, MPI2_POINTER PTR_MPI2_BOOT_DEVICE_SAS_WWID,
1199 Mpi2BootDeviceSasWwid_t, MPI2_POINTER pMpi2BootDeviceSasWwid_t;

1201 typedef struct _MPI2_BOOT_DEVICE_ENCLOSURE_SLOT
1202 {
1203     U64         EnclosureLogicalID;        /* 0x00 */
1204     U32         Reserved1;                /* 0x08 */
1205     U32         Reserved2;                /* 0x0C */
1206     U16         SlotNumber;                /* 0x10 */
1207     U16         Reserved3;                /* 0x12 */
1208     U32         Reserved4;                /* 0x14 */
1209 } MPI2_BOOT_DEVICE_ENCLOSURE_SLOT,
1210 MPI2_POINTER PTR_MPI2_BOOT_DEVICE_ENCLOSURE_SLOT,
1211 Mpi2BootDeviceEnclosuresSlot_t, MPI2_POINTER pMpi2BootDeviceEnclosuresSlot_t;

1213 typedef struct _MPI2_BOOT_DEVICE_DEVICE_NAME
1214 {
1215     U64         DeviceName;                /* 0x00 */
1216     U8          LUN[8];                    /* 0x08 */
1217     U32         Reserved1;                /* 0x10 */
1218     U32         Reserved2;                /* 0x14 */
1219 } MPI2_BOOT_DEVICE_DEVICE_NAME, MPI2_POINTER PTR_MPI2_BOOT_DEVICE_DEVICE_NAME,
1220 Mpi2BootDeviceDeviceName_t, MPI2_POINTER pMpi2BootDeviceDeviceName_t;

1222 typedef union _MPI2_MPI2_BIOSPAGE2_BOOT_DEVICE
1223 {
1224     MPI2_BOOT_DEVICE_ADAPTER_ORDER  AdapterOrder;
1225     MPI2_BOOT_DEVICE_SAS_WWID        SasWwid;
1226     MPI2_BOOT_DEVICE_ENCLOSURE_SLOT  EnclosureSlot;
1227     MPI2_BOOT_DEVICE_DEVICE_NAME     DeviceName;
1228 } MPI2_BIOSPAGE2_BOOT_DEVICE, MPI2_POINTER PTR_MPI2_BIOSPAGE2_BOOT_DEVICE,
1229 Mpi2BiosPage2BootDevice_t, MPI2_POINTER pMpi2BiosPage2BootDevice_t;

1231 typedef struct _MPI2_CONFIG_PAGE_BIOS_2
1232 {
1233     MPI2_CONFIG_PAGE_HEADER  Header;        /* 0x00 */
1234     U32                     Reserved1;      /* 0x04 */
1235     U32                     Reserved2;      /* 0x08 */
1236     U32                     Reserved3;      /* 0x0C */
1237     U32                     Reserved4;      /* 0x10 */
1238     U32                     Reserved5;      /* 0x14 */
1239     U32                     Reserved6;      /* 0x18 */
1240     U8                      ReqBootDeviceForm; /* 0x1C */
1241     U8                      Reserved7;      /* 0x1D */
1242     U16                     Reserved8;      /* 0x1E */
1243     MPI2_BIOSPAGE2_BOOT_DEVICE RequestedBootDevice; /* 0x20 */
1244     U8                      ReqAltBootDeviceForm; /* 0x38 */
1245     U8                      Reserved9;      /* 0x39 */
1246     U16                     Reserved10;     /* 0x3A */
1247     MPI2_BIOSPAGE2_BOOT_DEVICE RequestedAltBootDevice; /* 0x3C */
1248     U8                      CurrentBootDeviceForm; /* 0x58 */
1249     U8                      Reserved11;     /* 0x59 */

```

```

1250     U16         Reserved12;                /* 0x5A */
1251     MPI2_BIOSPAGE2_BOOT_DEVICE CurrentBootDevice; /* 0x58 */
1252 } MPI2_CONFIG_PAGE_BIOS_2, MPI2_POINTER PTR_MPI2_CONFIG_PAGE_BIOS_2,
1253 Mpi2BiosPage2_t, MPI2_POINTER pMpi2BiosPage2_t;

1255 #define MPI2_BIOSPAGE2_PAGEVERSION          (0x04)

1257 /* values for BIOS Page 2 BootDeviceForm fields */
1258 #define MPI2_BIOSPAGE2_FORM_MASK            (0x0F)
1259 #define MPI2_BIOSPAGE2_FORM_NO_DEVICE_SPECIFIED (0x00)
1260 #define MPI2_BIOSPAGE2_FORM_SAS_WWID        (0x05)
1261 #define MPI2_BIOSPAGE2_FORM_ENCLOSURE_SLOT  (0x06)
1262 #define MPI2_BIOSPAGE2_FORM_DEVICE_NAME     (0x07)

1265 /* BIOS Page 3 */

1267 typedef struct _MPI2_ADAPTER_INFO
1268 {
1269     U8          PciBusNumber;                /* 0x00 */
1270     U8          PciDeviceAndFunctionNumber; /* 0x01 */
1271     U16         AdapterFlags;                /* 0x02 */
1272 } MPI2_ADAPTER_INFO, MPI2_POINTER PTR_MPI2_ADAPTER_INFO,
1273 Mpi2AdapterInfo_t, MPI2_POINTER pMpi2AdapterInfo_t;

1275 #define MPI2_ADAPTER_INFO_FLAGS_EMBEDDED      (0x0001)
1276 #define MPI2_ADAPTER_INFO_FLAGS_INIT_STATUS  (0x0002)

1278 typedef struct _MPI2_CONFIG_PAGE_BIOS_3
1279 {
1280     MPI2_CONFIG_PAGE_HEADER  Header;        /* 0x00 */
1281     U32                     GlobalFlags;     /* 0x04 */
1282     U32                     BiosVersion;     /* 0x08 */
1283     MPI2_ADAPTER_INFO        AdapterOrder[4]; /* 0x0C */
1284     U32                     Reserved1;       /* 0x1C */
1285 } MPI2_CONFIG_PAGE_BIOS_3, MPI2_POINTER PTR_MPI2_CONFIG_PAGE_BIOS_3,
1286 Mpi2BiosPage3_t, MPI2_POINTER pMpi2BiosPage3_t;

1288 #define MPI2_BIOSPAGE3_PAGEVERSION          (0x00)

1290 /* values for BIOS Page 3 GlobalFlags */
1291 #define MPI2_BIOSPAGE3_FLAGS_PAUSE_ON_ERROR (0x00000002)
1292 #define MPI2_BIOSPAGE3_FLAGS_VERBOSE_ENABLE (0x00000004)
1293 #define MPI2_BIOSPAGE3_FLAGS_HOOK_INT_40_DISABLE (0x00000010)

1295 #define MPI2_BIOSPAGE3_FLAGS_DEV_LIST_DISPLAY_MASK (0x000000E0)
1296 #define MPI2_BIOSPAGE3_FLAGS_INSTALLED_DEV_DISPLAY (0x00000000)
1297 #define MPI2_BIOSPAGE3_FLAGS_ADAPTER_DISPLAY (0x00000020)
1298 #define MPI2_BIOSPAGE3_FLAGS_ADAPTER_DEV_DISPLAY (0x00000040)

1301 /* BIOS Page 4 */

1303 /*
1304  * Host code (drivers, BIOS, utilities, etc.) should leave this define set to
1305  * one and check Header.PageLength or NumPhys at runtime.
1306  */
1307 #ifndef MPI2_BIOS_PAGE_4_PHY_ENTRIES
1308 #define MPI2_BIOS_PAGE_4_PHY_ENTRIES      (1)
1309 #endif

1311 typedef struct _MPI2_BIOS4_ENTRY
1312 {
1313     U64         ReassignmentWWID;          /* 0x00 */
1314     U64         ReassignmentDeviceName;    /* 0x08 */
1315 } MPI2_BIOS4_ENTRY, MPI2_POINTER PTR_MPI2_BIOS4_ENTRY,

```

```

1316  Mpi2MBios4Entry_t, MPI2_POINTER pMpi2Bios4Entry_t;

1318 typedef struct _MPI2_CONFIG_PAGE_BIOS_4
1319 {
1320     MPI2_CONFIG_PAGE_HEADER Header;          /* 0x00 */
1321     U8 NumPhys;                             /* 0x04 */
1322     U8 Reserved1;                           /* 0x05 */
1323     U16 Reserved2;                          /* 0x06 */
1324     MPI2_BIOS4_ENTRY Phy[MPI2_BIOS_PAGE_4_PHY_ENTRIES]; /* 0x08 */
1325 } MPI2_CONFIG_PAGE_BIOS_4, MPI2_POINTER PTR_MPI2_CONFIG_PAGE_BIOS_4,
1326   Mpi2BiosPage4_t, MPI2_POINTER pMpi2BiosPage4_t;

1328 #define MPI2_BIOSPAGE4_PAGEVERSION          (0x01)

1331 /*****
1332  * RAID Volume Config Pages
1333  *****/

1335 /* RAID Volume Page 0 */

1337 typedef struct _MPI2_RAIDVOL0_PHYS_DISK
1338 {
1339     U8 RAIDSetNum;                          /* 0x00 */
1340     U8 PhysDiskMap;                         /* 0x01 */
1341     U8 PhysDiskNum;                        /* 0x02 */
1342     U8 Reserved;                           /* 0x03 */
1343 } MPI2_RAIDVOL0_PHYS_DISK, MPI2_POINTER PTR_MPI2_RAIDVOL0_PHYS_DISK,
1344   Mpi2RaidVol0PhysDisk_t, MPI2_POINTER pMpi2RaidVol0PhysDisk_t;

1346 /* defines for the PhysDiskMap field */
1347 #define MPI2_RAIDVOL0_PHYSDISK_PRIMARY      (0x01)
1348 #define MPI2_RAIDVOL0_PHYSDISK_SECONDARY    (0x02)

1350 typedef struct _MPI2_RAIDVOL0_SETTINGS
1351 {
1352     U16 Settings;                          /* 0x00 */
1353     U8 HotSparePool;                       /* 0x01 */
1354     U8 Reserved;                           /* 0x02 */
1355 } MPI2_RAIDVOL0_SETTINGS, MPI2_POINTER PTR_MPI2_RAIDVOL0_SETTINGS,
1356   Mpi2RaidVol0Settings_t, MPI2_POINTER pMpi2RaidVol0Settings_t;

1358 /* RAID Volume Page 0 HotSparePool defines, also used in RAID Physical Disk */
1359 #define MPI2_RAID_HOT_SPARE_POOL_0          (0x01)
1360 #define MPI2_RAID_HOT_SPARE_POOL_1          (0x02)
1361 #define MPI2_RAID_HOT_SPARE_POOL_2          (0x04)
1362 #define MPI2_RAID_HOT_SPARE_POOL_3          (0x08)
1363 #define MPI2_RAID_HOT_SPARE_POOL_4          (0x10)
1364 #define MPI2_RAID_HOT_SPARE_POOL_5          (0x20)
1365 #define MPI2_RAID_HOT_SPARE_POOL_6          (0x40)
1366 #define MPI2_RAID_HOT_SPARE_POOL_7          (0x80)

1368 /* RAID Volume Page 0 VolumeSettings defines */
1369 #define MPI2_RAIDVOL0_SETTING_USE_PRODUCT_ID_SUFFIX (0x0008)
1370 #define MPI2_RAIDVOL0_SETTING_AUTO_CONFIG_HSWAP_DISABLE (0x0004)

1372 #define MPI2_RAIDVOL0_SETTING_MASK_WRITE_CACHING (0x0003)
1373 #define MPI2_RAIDVOL0_SETTING_UNCHANGED          (0x0000)
1374 #define MPI2_RAIDVOL0_SETTING_DISABLE_WRITE_CACHING (0x0001)
1375 #define MPI2_RAIDVOL0_SETTING_ENABLE_WRITE_CACHING (0x0002)

1377 /*
1378  * Host code (drivers, BIOS, utilities, etc.) should leave this define set to
1379  * one and check Header.PageLength at runtime.
1380  */
1381 #ifndef MPI2_RAID_VOL_PAGE_0_PHYSDISK_MAX

```

```

1382 #define MPI2_RAID_VOL_PAGE_0_PHYSDISK_MAX      (1)
1383 #endif

1385 typedef struct _MPI2_CONFIG_PAGE_RAID_VOL_0
1386 {
1387     MPI2_CONFIG_PAGE_HEADER Header;          /* 0x00 */
1388     U16 DevHandle;                           /* 0x04 */
1389     U8 VolumeState;                          /* 0x06 */
1390     U8 VolumeType;                           /* 0x07 */
1391     U32 VolumeStatusFlags;                   /* 0x08 */
1392     MPI2_RAIDVOL0_SETTINGS VolumeSettings;  /* 0x0C */
1393     U64 MaxLBA;                              /* 0x10 */
1394     U32 StripeSize;                          /* 0x18 */
1395     U16 BlockSize;                           /* 0x1C */
1396     U16 Reserved1;                           /* 0x1E */
1397     U8 SupportedPhysDisks;                   /* 0x20 */
1398     U8 ResyncRate;                           /* 0x21 */
1399     U16 DataScrubDuration;                    /* 0x22 */
1400     U8 NumPhysDisks;                         /* 0x24 */
1401     U8 Reserved2;                            /* 0x25 */
1402     U8 Reserved3;                            /* 0x26 */
1403     U8 InactiveStatus;                       /* 0x27 */
1404     MPI2_RAIDVOL0_PHYS_DISK PhysDisk[MPI2_RAID_VOL_PAGE_0_PHYSDISK_MAX]; /* 0x28 */
1405 } MPI2_CONFIG_PAGE_RAID_VOL_0, MPI2_POINTER PTR_MPI2_CONFIG_PAGE_RAID_VOL_0,
1406   Mpi2RaidVolPage0_t, MPI2_POINTER pMpi2RaidVolPage0_t;

1408 #define MPI2_RAIDVOLPAGE0_PAGEVERSION          (0x0A)

1410 /* values for RAID VolumeState */
1411 #define MPI2_RAID_VOL_STATE_MISSING          (0x00)
1412 #define MPI2_RAID_VOL_STATE_FAILED           (0x01)
1413 #define MPI2_RAID_VOL_STATE_INITIALIZING     (0x02)
1414 #define MPI2_RAID_VOL_STATE_ONLINE           (0x03)
1415 #define MPI2_RAID_VOL_STATE_DEGRADED         (0x04)
1416 #define MPI2_RAID_VOL_STATE_OPTIMAL          (0x05)

1418 /* values for RAID VolumeType */
1419 #define MPI2_RAID_VOL_TYPE_RAID0             (0x00)
1420 #define MPI2_RAID_VOL_TYPE_RAID1E           (0x01)
1421 #define MPI2_RAID_VOL_TYPE_RAID1            (0x02)
1422 #define MPI2_RAID_VOL_TYPE_RAID10           (0x05)
1423 #define MPI2_RAID_VOL_TYPE_UNKNOWN           (0xFF)

1425 /* values for RAID Volume Page 0 VolumeStatusFlags field */
1426 #define MPI2_RAIDVOL0_STATUS_FLAG_PENDING_RESYNC (0x02000000)
1427 #define MPI2_RAIDVOL0_STATUS_FLAG_BACKG_INIT_PENDING (0x01000000)
1428 #define MPI2_RAIDVOL0_STATUS_FLAG_MDC_PENDING (0x00800000)
1429 #define MPI2_RAIDVOL0_STATUS_FLAG_USER_CONSIST_PENDING (0x00400000)
1430 #define MPI2_RAIDVOL0_STATUS_FLAG_MAKE_DATA_CONSISTENT (0x00200000)
1431 #define MPI2_RAIDVOL0_STATUS_FLAG_DATA_SCRUB (0x00100000)
1432 #define MPI2_RAIDVOL0_STATUS_FLAG_CONSISTENCY_CHECK (0x00080000)
1433 #define MPI2_RAIDVOL0_STATUS_FLAG_CAPACITY_EXPANSION (0x00040000)
1434 #define MPI2_RAIDVOL0_STATUS_FLAG_BACKGROUND_INIT (0x00020000)
1435 #define MPI2_RAIDVOL0_STATUS_FLAG_RESYNC_IN_PROGRESS (0x00010000)
1436 #define MPI2_RAIDVOL0_STATUS_FLAG_VOL_NOT_CONSISTENT (0x00000080)
1437 #define MPI2_RAIDVOL0_STATUS_FLAG_OCE_ALLOWED (0x00000040)
1438 #define MPI2_RAIDVOL0_STATUS_FLAG_BGI_COMPLETE (0x00000020)
1439 #define MPI2_RAIDVOL0_STATUS_FLAG_1E_OFFSET_MIRROR (0x00000010)
1440 #define MPI2_RAIDVOL0_STATUS_FLAG_1E_ADJACENT_MIRROR (0x00000010)
1441 #define MPI2_RAIDVOL0_STATUS_FLAG_BAD_BLOCK_TABLE_FULL (0x00000008)
1442 #define MPI2_RAIDVOL0_STATUS_FLAG_VOLUME_INACTIVE (0x00000004)
1443 #define MPI2_RAIDVOL0_STATUS_FLAG_QUIESCED (0x00000002)
1444 #define MPI2_RAIDVOL0_STATUS_FLAG_ENABLED (0x00000001)

1446 /* values for RAID Volume Page 0 SupportedPhysDisks field */
1447 #define MPI2_RAIDVOL0_SUPPORT_SOLID_STATE_DISKS (0x08)

```

```

1448 #define MPI2_RAIDVOL0_SUPPORT_HARD_DISKS (0x04)
1449 #define MPI2_RAIDVOL0_SUPPORT_SAS_PROTOCOL (0x02)
1450 #define MPI2_RAIDVOL0_SUPPORT_SATA_PROTOCOL (0x01)

1452 /* values for RAID Volume Page 0 InactiveStatus field */
1453 #define MPI2_RAIDVOLPAGE0_UNKNOWN_INACTIVE (0x00)
1454 #define MPI2_RAIDVOLPAGE0_STALE_METADATA_INACTIVE (0x01)
1455 #define MPI2_RAIDVOLPAGE0_FOREIGN_VOLUME_INACTIVE (0x02)
1456 #define MPI2_RAIDVOLPAGE0_INSUFFICIENT_RESOURCE_INACTIVE (0x03)
1457 #define MPI2_RAIDVOLPAGE0_CLONE_VOLUME_INACTIVE (0x04)
1458 #define MPI2_RAIDVOLPAGE0_INSUFFICIENT_METADATA_INACTIVE (0x05)
1459 #define MPI2_RAIDVOLPAGE0_PREVIOUSLY_DELETED (0x06)

1462 /* RAID Volume Page 1 */

1464 typedef struct _MPI2_CONFIG_PAGE_RAID_VOL_1
1465 {
1466     MPI2_CONFIG_PAGE_HEADER Header; /* 0x00 */
1467     U16 DevHandle; /* 0x04 */
1468     U16 Reserved0; /* 0x06 */
1469     U8 GUID[24]; /* 0x08 */
1470     U8 Name[16]; /* 0x20 */
1471     U64 WWID; /* 0x30 */
1472     U32 Reserved1; /* 0x38 */
1473     U32 Reserved2; /* 0x3C */
1474 } MPI2_CONFIG_PAGE_RAID_VOL_1, MPI2_POINTER PTR_MPI2_CONFIG_PAGE_RAID_VOL_1,
1475     Mpi2RaidVolPage1_t, MPI2_POINTER pMpi2RaidVolPage1_t;

1477 #define MPI2_RAIDVOLPAGE1_PAGEVERSION (0x03)

1480 /*****
1481 * RAID Physical Disk Config Pages
1482 *****/

1484 /* RAID Physical Disk Page 0 */

1486 typedef struct _MPI2_RAIDPHYSDISK0_SETTINGS
1487 {
1488     U16 Reserved1; /* 0x00 */
1489     U8 HotSparePool; /* 0x02 */
1490     U8 Reserved2; /* 0x03 */
1491 } MPI2_RAIDPHYSDISK0_SETTINGS, MPI2_POINTER PTR_MPI2_RAIDPHYSDISK0_SETTINGS,
1492     Mpi2RaidPhysDisk0Settings_t, MPI2_POINTER pMpi2RaidPhysDisk0Settings_t;

1494 /* use MPI2_RAID_HOT_SPARE_POOL defines for the HotSparePool field */

1496 typedef struct _MPI2_RAIDPHYSDISK0_INQUIRY_DATA
1497 {
1498     U8 VendorID[8]; /* 0x00 */
1499     U8 ProductID[16]; /* 0x08 */
1500     U8 ProductRevLevel[4]; /* 0x18 */
1501     U8 SerialNum[32]; /* 0x1C */
1502 } MPI2_RAIDPHYSDISK0_INQUIRY_DATA,
1503     MPI2_POINTER PTR_MPI2_RAIDPHYSDISK0_INQUIRY_DATA,
1504     Mpi2RaidPhysDisk0InquiryData_t, MPI2_POINTER pMpi2RaidPhysDisk0InquiryData_t;

1506 typedef struct _MPI2_CONFIG_PAGE_RD_PDISK_0
1507 {
1508     MPI2_CONFIG_PAGE_HEADER Header; /* 0x00 */
1509     U16 DevHandle; /* 0x04 */
1510     U8 Reserved1; /* 0x06 */
1511     U8 PhysDiskNum; /* 0x07 */
1512     MPI2_RAIDPHYSDISK0_SETTINGS PhysDiskSettings; /* 0x08 */
1513     U32 Reserved2; /* 0x0C */

```

```

1514 MPI2_RAIDPHYSDISK0_INQUIRY_DATA InquiryData; /* 0x10 */
1515 U32 Reserved3; /* 0x4C */
1516 U8 PhysDiskState; /* 0x50 */
1517 U8 OfflineReason; /* 0x51 */
1518 U8 IncompatibleReason; /* 0x52 */
1519 U8 PhysDiskAttributes; /* 0x53 */
1520 U32 PhysDiskStatusFlags; /* 0x54 */
1521 U64 DeviceMaxLBA; /* 0x58 */
1522 U64 HostMaxLBA; /* 0x60 */
1523 U64 CoercedMaxLBA; /* 0x68 */
1524 U16 BlockSize; /* 0x70 */
1525 U16 Reserved5; /* 0x72 */
1526 U32 Reserved6; /* 0x74 */
1527 } MPI2_CONFIG_PAGE_RD_PDISK_0,
1528     MPI2_POINTER PTR_MPI2_CONFIG_PAGE_RD_PDISK_0,
1529     Mpi2RaidPhysDiskPage0_t, MPI2_POINTER pMpi2RaidPhysDiskPage0_t;

1531 #define MPI2_RAIDPHYSDISKPAGE0_PAGEVERSION (0x05)

1533 /* PhysDiskState defines */
1534 #define MPI2_RAID_PD_STATE_NOT_CONFIGURED (0x00)
1535 #define MPI2_RAID_PD_STATE_NOT_COMPATIBLE (0x01)
1536 #define MPI2_RAID_PD_STATE_OFFLINE (0x02)
1537 #define MPI2_RAID_PD_STATE_ONLINE (0x03)
1538 #define MPI2_RAID_PD_STATE_HOT_SPARE (0x04)
1539 #define MPI2_RAID_PD_STATE_DEGRADED (0x05)
1540 #define MPI2_RAID_PD_STATE_REBUILDING (0x06)
1541 #define MPI2_RAID_PD_STATE_OPTIMAL (0x07)

1543 /* OfflineReason defines */
1544 #define MPI2_PHYSDISK0_ONLINE (0x00)
1545 #define MPI2_PHYSDISK0_OFFLINE_MISSING (0x01)
1546 #define MPI2_PHYSDISK0_OFFLINE_FAILED (0x03)
1547 #define MPI2_PHYSDISK0_OFFLINE_INITIALIZING (0x04)
1548 #define MPI2_PHYSDISK0_OFFLINE_REQUESTED (0x05)
1549 #define MPI2_PHYSDISK0_OFFLINE_FAILED_REQUESTED (0x06)
1550 #define MPI2_PHYSDISK0_OFFLINE_OTHER (0xFF)

1552 /* IncompatibleReason defines */
1553 #define MPI2_PHYSDISK0_COMPATIBLE (0x00)
1554 #define MPI2_PHYSDISK0_INCOMPATIBLE_PROTOCOL (0x01)
1555 #define MPI2_PHYSDISK0_INCOMPATIBLE_BLOCKSIZE (0x02)
1556 #define MPI2_PHYSDISK0_INCOMPATIBLE_MAX_LBA (0x03)
1557 #define MPI2_PHYSDISK0_INCOMPATIBLE_SATA_EXTENDED_CMD (0x04)
1558 #define MPI2_PHYSDISK0_INCOMPATIBLE_REMOVABLE_MEDIA (0x05)
1559 #define MPI2_PHYSDISK0_INCOMPATIBLE_MEDIA_TYPE (0x06)
1560 #define MPI2_PHYSDISK0_INCOMPATIBLE_UNKNOWN (0xFF)

1562 /* PhysDiskAttributes defines */
1563 #define MPI2_PHYSDISK0_ATTRIB_SOLID_STATE_DRIVE (0x08)
1564 #define MPI2_PHYSDISK0_ATTRIB_HARD_DISK_DRIVE (0x04)

1566 #define MPI2_PHYSDISK0_ATTRIB_PROTOCOL_MASK (0x03)
1567 #define MPI2_PHYSDISK0_ATTRIB_SAS_PROTOCOL (0x02)
1568 #define MPI2_PHYSDISK0_ATTRIB_SATA_PROTOCOL (0x01)

1570 /* PhysDiskStatusFlags defines */
1571 #define MPI2_PHYSDISK0_STATUS_FLAG_NOT_CERTIFIED (0x00000040)
1572 #define MPI2_PHYSDISK0_STATUS_FLAG_OCE_TARGET (0x00000020)
1573 #define MPI2_PHYSDISK0_STATUS_FLAG_WRITE_CACHE_ENABLED (0x00000010)
1574 #define MPI2_PHYSDISK0_STATUS_FLAG_OPTIMAL_PREVIOUS (0x00000000)
1575 #define MPI2_PHYSDISK0_STATUS_FLAG_NOT_OPTIMAL_PREVIOUS (0x00000008)
1576 #define MPI2_PHYSDISK0_STATUS_FLAG_INACTIVE_VOLUME (0x00000004)
1577 #define MPI2_PHYSDISK0_STATUS_FLAG_QUIESCED (0x00000002)
1578 #define MPI2_PHYSDISK0_STATUS_FLAG_OUT_OF_SYNC (0x00000001)

```

```

1581 /* RAID Physical Disk Page 1 */
1583 /*
1584 * Host code (drivers, BIOS, utilities, etc.) should leave this define set to
1585 * one and check Header.PageLength or NumPhysDiskPaths at runtime.
1586 */
1587 #ifndef MPI2_RAID_PHYS_DISK1_PATH_MAX
1588 #define MPI2_RAID_PHYS_DISK1_PATH_MAX (1)
1589 #endif

1591 typedef struct _MPI2_RAIDPHYSDISK1_PATH
1592 {
1593     U16          DevHandle;          /* 0x00 */
1594     U16          Reserved1;          /* 0x02 */
1595     U64          WWID;               /* 0x04 */
1596     U64          OwnerWWID;          /* 0x0C */
1597     U8           OwnerIdentifier;     /* 0x14 */
1598     U8           Reserved2;          /* 0x15 */
1599     U16          Flags;              /* 0x16 */
1600 } MPI2_RAIDPHYSDISK1_PATH, MPI2_POINTER PTR_MPI2_RAIDPHYSDISK1_PATH,
1601   Mpi2RaidPhysDisk1Path_t, MPI2_POINTER pMpi2RaidPhysDisk1Path_t;

1603 /* RAID Physical Disk Page 1 Physical Disk Path Flags field defines */
1604 #define MPI2_RAID_PHYSDISK1_FLAG_PRIMARY (0x0004)
1605 #define MPI2_RAID_PHYSDISK1_FLAG_BROKEN (0x0002)
1606 #define MPI2_RAID_PHYSDISK1_FLAG_INVALID (0x0001)

1608 typedef struct _MPI2_CONFIG_PAGE_RD_PDISK_1
1609 {
1610     MPI2_CONFIG_PAGE_HEADER    Header;          /* 0x00 */
1611     U8                         NumPhysDiskPaths; /* 0x04 */
1612     U8                         PhysDiskNum;      /* 0x05 */
1613     U16                        Reserved1;         /* 0x06 */
1614     U32                        Reserved2;         /* 0x08 */
1615     MPI2_RAIDPHYSDISK1_PATH    PhysicalDiskPath[MPI2_RAID_PHYS_DISK1_PATH_M
1616 } MPI2_CONFIG_PAGE_RD_PDISK_1,
1617   MPI2_POINTER PTR_MPI2_CONFIG_PAGE_RD_PDISK_1,
1618   Mpi2RaidPhysDiskPage1_t, MPI2_POINTER pMpi2RaidPhysDiskPage1_t;

1620 #define MPI2_RAIDPHYSDISKPAGE1_PAGEVERSION (0x02)

1623 /*****
1624 * values for fields used by several types of SAS Config Pages
1625 *****/

1627 /* values for NegotiatedLinkRates fields */
1628 #define MPI2_SAS_NEG_LINK_RATE_MASK_LOGICAL (0xF0)
1629 #define MPI2_SAS_NEG_LINK_RATE_SHIFT_LOGICAL (4)
1630 #define MPI2_SAS_NEG_LINK_RATE_MASK_PHYSICAL (0x0F)
1631 /* link rates used for Negotiated Physical and Logical Link Rate */
1632 #define MPI2_SAS_NEG_LINK_RATE_UNKNOWN_LINK_RATE (0x00)
1633 #define MPI2_SAS_NEG_LINK_RATE_PHY_DISABLED (0x01)
1634 #define MPI2_SAS_NEG_LINK_RATE_NEGOTIATION_FAILED (0x02)
1635 #define MPI2_SAS_NEG_LINK_RATE_SATA_OOB_COMPLETE (0x03)
1636 #define MPI2_SAS_NEG_LINK_RATE_PORT_SELECTOR (0x04)
1637 #define MPI2_SAS_NEG_LINK_RATE_SMP_RESET_IN_PROGRESS (0x05)
1638 #define MPI2_SAS_NEG_LINK_RATE_UNSUPPORTED_PHY (0x06)
1639 #define MPI2_SAS_NEG_LINK_RATE_1_5 (0x08)
1640 #define MPI2_SAS_NEG_LINK_RATE_3_0 (0x09)
1641 #define MPI2_SAS_NEG_LINK_RATE_6_0 (0x0A)
1642 #define MPI2_SAS_NEG_LINK_RATE_12_0 (0x0B)

1645 /* values for AttachedPhyInfo fields */

```

```

1646 #define MPI2_SAS_APHYINFO_INSIDE_ZPSDS_PERSISTENT (0x00000040)
1647 #define MPI2_SAS_APHYINFO_REQUESTED_INSIDE_ZPSDS (0x00000020)
1648 #define MPI2_SAS_APHYINFO_BREAK_REPLY_CAPABLE (0x00000010)

1650 #define MPI2_SAS_APHYINFO_REASON_MASK (0x0000000F)
1651 #define MPI2_SAS_APHYINFO_REASON_UNKNOWN (0x00000000)
1652 #define MPI2_SAS_APHYINFO_REASON_POWER_ON (0x00000001)
1653 #define MPI2_SAS_APHYINFO_REASON_HARD_RESET (0x00000002)
1654 #define MPI2_SAS_APHYINFO_REASON_SMP_PHY_CONTROL (0x00000003)
1655 #define MPI2_SAS_APHYINFO_REASON_LOSS_OF_SYNC (0x00000004)
1656 #define MPI2_SAS_APHYINFO_REASON_MULTIPLEXING_SEQ (0x00000005)
1657 #define MPI2_SAS_APHYINFO_REASON_IT_NEXUS_LOSS_TIMER (0x00000006)
1658 #define MPI2_SAS_APHYINFO_REASON_BREAK_TIMEOUT (0x00000007)
1659 #define MPI2_SAS_APHYINFO_REASON_PHY_TEST_STOPPED (0x00000008)

1662 /* values for PhyInfo fields */
1663 #define MPI2_SAS_PHYINFO_PHY_VACANT (0x80000000)

1665 #define MPI2_SAS_PHYINFO_PHY_POWER_CONDITION_MASK (0x18000000)
1666 #define MPI2_SAS_PHYINFO_SHIFT_PHY_POWER_CONDITION (27)
1667 #define MPI2_SAS_PHYINFO_PHY_POWER_ACTIVE (0x00000000)
1668 #define MPI2_SAS_PHYINFO_PHY_POWER_PARTIAL (0x08000000)
1669 #define MPI2_SAS_PHYINFO_PHY_POWER_SLUMBER (0x10000000)

1671 #define MPI2_SAS_PHYINFO_CHANGED_REQ_INSIDE_ZPSDS (0x04000000)
1672 #define MPI2_SAS_PHYINFO_INSIDE_ZPSDS_PERSISTENT (0x02000000)
1673 #define MPI2_SAS_PHYINFO_REQ_INSIDE_ZPSDS (0x01000000)
1674 #define MPI2_SAS_PHYINFO_ZONE_GROUP_PERSISTENT (0x00400000)
1675 #define MPI2_SAS_PHYINFO_INSIDE_ZPSDS (0x00200000)
1676 #define MPI2_SAS_PHYINFO_ZONING_ENABLED (0x00100000)

1678 #define MPI2_SAS_PHYINFO_REASON_MASK (0x000F0000)
1679 #define MPI2_SAS_PHYINFO_REASON_UNKNOWN (0x00000000)
1680 #define MPI2_SAS_PHYINFO_REASON_POWER_ON (0x00010000)
1681 #define MPI2_SAS_PHYINFO_REASON_HARD_RESET (0x00020000)
1682 #define MPI2_SAS_PHYINFO_REASON_SMP_PHY_CONTROL (0x00030000)
1683 #define MPI2_SAS_PHYINFO_REASON_LOSS_OF_SYNC (0x00040000)
1684 #define MPI2_SAS_PHYINFO_REASON_MULTIPLEXING_SEQ (0x00050000)
1685 #define MPI2_SAS_PHYINFO_REASON_IT_NEXUS_LOSS_TIMER (0x00060000)
1686 #define MPI2_SAS_PHYINFO_REASON_BREAK_TIMEOUT (0x00070000)
1687 #define MPI2_SAS_PHYINFO_REASON_PHY_TEST_STOPPED (0x00080000)

1689 #define MPI2_SAS_PHYINFO_MULTIPLEXING_SUPPORTED (0x00008000)
1690 #define MPI2_SAS_PHYINFO_SATA_PORT_ACTIVE (0x00004000)
1691 #define MPI2_SAS_PHYINFO_SATA_PORT_SELECTOR_PRESENT (0x00002000)
1692 #define MPI2_SAS_PHYINFO_VIRTUAL_PHY (0x00001000)

1694 #define MPI2_SAS_PHYINFO_MASK_PARTIAL_PATHWAY_TIME (0x00000F00)
1695 #define MPI2_SAS_PHYINFO_SHIFT_PARTIAL_PATHWAY_TIME (8)

1697 #define MPI2_SAS_PHYINFO_MASK_ROUTING_ATTRIBUTE (0x000000F0)
1698 #define MPI2_SAS_PHYINFO_DIRECT_ROUTING (0x00000000)
1699 #define MPI2_SAS_PHYINFO_SUBTRACTIVE_ROUTING (0x00000010)
1700 #define MPI2_SAS_PHYINFO_TABLE_ROUTING (0x00000020)

1703 /* values for SAS ProgrammedLinkRate fields */
1704 #define MPI2_SAS_PRATE_MAX_RATE_MASK (0xF0)
1705 #define MPI2_SAS_PRATE_MAX_RATE_NOT_PROGRAMMABLE (0x00)
1706 #define MPI2_SAS_PRATE_MAX_RATE_1_5 (0x80)
1707 #define MPI2_SAS_PRATE_MAX_RATE_3_0 (0x90)
1708 #define MPI2_SAS_PRATE_MAX_RATE_6_0 (0xA0)
1709 #define MPI2_SAS_PRATE_MIN_RATE_MASK (0x0F)
1710 #define MPI2_SAS_PRATE_MIN_RATE_NOT_PROGRAMMABLE (0x00)
1711 #define MPI2_SAS_PRATE_MIN_RATE_1_5 (0x08)

```

```

1712 #define MPI2_SAS_PRATE_MIN_RATE_3_0          (0x09)
1713 #define MPI2_SAS_PRATE_MIN_RATE_6_0          (0x0A)
1714 #define MPI25_SAS_PRATE_MIN_RATE_12_0         (0x0B)

1717 /* values for SAS HwLinkRate fields */
1718 #define MPI2_SAS_HWRATE_MAX_RATE_MASK        (0xF0)
1719 #define MPI2_SAS_HWRATE_MAX_RATE_1_5         (0x80)
1720 #define MPI2_SAS_HWRATE_MAX_RATE_3_0         (0x90)
1721 #define MPI2_SAS_HWRATE_MAX_RATE_6_0         (0xA0)
1722 #define MPI2_SAS_HWRATE_MIN_RATE_MASK        (0x0F)
1723 #define MPI2_SAS_HWRATE_MIN_RATE_1_5         (0x08)
1724 #define MPI2_SAS_HWRATE_MIN_RATE_3_0         (0x09)
1725 #define MPI2_SAS_HWRATE_MIN_RATE_6_0         (0x0A)
1726 #define MPI25_SAS_HWRATE_MIN_RATE_12_0       (0x0B)

1730 /*****
1731  * SAS IO Unit Config Pages
1732  *****/
1733
1734 /* SAS IO Unit Page 0 */
1735
1736 typedef struct _MPI2_SAS_IO_UNIT0_PHY_DATA
1737 {
1738     U8      Port;          /* 0x00 */
1739     U8      PortFlags;     /* 0x01 */
1740     U8      PhyFlags;      /* 0x02 */
1741     U8      NegotiatedLinkRate; /* 0x03 */
1742     U32     ControllerPhyDeviceInfo; /* 0x04 */
1743     U16     AttachedDevHandle; /* 0x08 */
1744     U16     ControllerDevHandle; /* 0x0A */
1745     U32     DiscoveryStatus; /* 0x0C */
1746     U32     Reserved;      /* 0x10 */
1747 } MPI2_SAS_IO_UNIT0_PHY_DATA, MPI2_POINTER PTR_MPI2_SAS_IO_UNIT0_PHY_DATA,
1748   Mpi2SasIOUnit0PhyData_t, MPI2_POINTER pMpi2SasIOUnit0PhyData_t;

1750 /*
1751  * Host code (drivers, BIOS, utilities, etc.) should leave this define set to
1752  * one and check Header.ExtPageLength or NumPhys at runtime.
1753  */
1754 #ifndef MPI2_SAS_IOUNIT0_PHY_MAX
1755 #define MPI2_SAS_IOUNIT0_PHY_MAX          (1)
1756 #endif

1758 typedef struct _MPI2_CONFIG_PAGE_SASIOUNIT_0
1759 {
1760     MPI2_CONFIG_EXTENDED_PAGE_HEADER Header; /* 0
1761     U32 Reserved1; /* 0
1762     U8 NumPhys; /* 0
1763     U8 Reserved2; /* 0
1764     U16 Reserved3; /* 0
1765     MPI2_SAS_IO_UNIT0_PHY_DATA PhyData[MPI2_SAS_IOUNIT0_PHY_MAX]; /* 0
1766 } MPI2_CONFIG_PAGE_SASIOUNIT_0,
1767 MPI2_POINTER PTR_MPI2_CONFIG_PAGE_SASIOUNIT_0,
1768   Mpi2SasIOUnitPage0_t, MPI2_POINTER pMpi2SasIOUnitPage0_t;

1770 #define MPI2_SASIOUNITPAGE0_PAGEVERSION      (0x05)

1772 /* values for SAS IO Unit Page 0 PortFlags */
1773 #define MPI2_SASIOUNIT0_PORTFLAGS_DISCOVERY_IN_PROGRESS (0x08)
1774 #define MPI2_SASIOUNIT0_PORTFLAGS_AUTO_PORT_CONFIG      (0x01)

1776 /* values for SAS IO Unit Page 0 PhyFlags */
1777 #define MPI2_SASIOUNIT0_PHYFLAGS_ZONING_ENABLED          (0x10)

```

```

1778 #define MPI2_SASIOUNIT0_PHYFLAGS_PHY_DISABLED          (0x08)

1780 /* use MPI2_SAS_NEG_LINK_RATE defines for the NegotiatedLinkRate field */

1782 /* see mpi2_sas.h for values for SAS IO Unit Page 0 ControllerPhyDeviceInfo value */

1784 /* values for SAS IO Unit Page 0 DiscoveryStatus */
1785 #define MPI2_SASIOUNIT0_DS_MAX_ENCLOSURES_EXCEED      (0x80000000)
1786 #define MPI2_SASIOUNIT0_DS_MAX_EXPANDERS_EXCEED      (0x40000000)
1787 #define MPI2_SASIOUNIT0_DS_MAX_DEVICES_EXCEED        (0x20000000)
1788 #define MPI2_SASIOUNIT0_DS_MAX_TOPO_PHYS_EXCEED      (0x10000000)
1789 #define MPI2_SASIOUNIT0_DS_DOWNSTREAM_INITIATOR      (0x08000000)
1790 #define MPI2_SASIOUNIT0_DS_MULTI_SUBTRACTIVE_SUBTRACTIVE (0x00008000)
1791 #define MPI2_SASIOUNIT0_DS_EXP_MULTI_SUBTRACTIVE     (0x00004000)
1792 #define MPI2_SASIOUNIT0_DS_MULTI_PORT_DOMAIN         (0x00002000)
1793 #define MPI2_SASIOUNIT0_DS_TABLE_TO_SUBTRACTIVE_LINK (0x00001000)
1794 #define MPI2_SASIOUNIT0_DS_UNSUPPORTED_DEVICE        (0x00000800)
1795 #define MPI2_SASIOUNIT0_DS_TABLE_LINK                (0x00000400)
1796 #define MPI2_SASIOUNIT0_DS_SUBTRACTIVE_LINK          (0x00000200)
1797 #define MPI2_SASIOUNIT0_DS_SMP_CRC_ERROR             (0x00000100)
1798 #define MPI2_SASIOUNIT0_DS_SMP_FUNCTION_FAILED       (0x00000080)
1799 #define MPI2_SASIOUNIT0_DS_INDEX_NOT_EXIST           (0x00000040)
1800 #define MPI2_SASIOUNIT0_DS_OUT_ROUTE_ENTRIES         (0x00000020)
1801 #define MPI2_SASIOUNIT0_DS_SMP_TIMEOUT               (0x00000010)
1802 #define MPI2_SASIOUNIT0_DS_MULTIPLE_PORTS            (0x00000004)
1803 #define MPI2_SASIOUNIT0_DS_UNADDRESSABLE_DEVICE     (0x00000002)
1804 #define MPI2_SASIOUNIT0_DS_LOOP_DETECTED             (0x00000001)

1807 /* SAS IO Unit Page 1 */

1809 typedef struct _MPI2_SAS_IO_UNIT1_PHY_DATA
1810 {
1811     U8      Port;          /* 0x00 */
1812     U8      PortFlags;     /* 0x01 */
1813     U8      PhyFlags;      /* 0x02 */
1814     U8      MaxMinLinkRate; /* 0x03 */
1815     U32     ControllerPhyDeviceInfo; /* 0x04 */
1816     U16     MaxTargetPortConnectTime; /* 0x08 */
1817     U16     Reserved1;     /* 0x0A */
1818 } MPI2_SAS_IO_UNIT1_PHY_DATA, MPI2_POINTER PTR_MPI2_SAS_IO_UNIT1_PHY_DATA,
1819   Mpi2SasIOUnit1PhyData_t, MPI2_POINTER pMpi2SasIOUnit1PhyData_t;

1821 /*
1822  * Host code (drivers, BIOS, utilities, etc.) should leave this define set to
1823  * one and check Header.ExtPageLength or NumPhys at runtime.
1824  */
1825 #ifndef MPI2_SAS_IOUNIT1_PHY_MAX
1826 #define MPI2_SAS_IOUNIT1_PHY_MAX          (1)
1827 #endif

1829 typedef struct _MPI2_CONFIG_PAGE_SASIOUNIT_1
1830 {
1831     MPI2_CONFIG_EXTENDED_PAGE_HEADER Header; /* 0
1832     U16 ControlFlags; /* 0
1833     U16 SASNarrowMaxQueueDepth; /* 0
1834     U16 AdditionalControlFlags; /* 0
1835     U16 SASWideMaxQueueDepth; /* 0
1836     U8 NumPhys; /* 0
1837     U8 SATAMaxQDepth; /* 0
1838     U8 ReportDeviceMissingDelay; /* 0
1839     U8 IODeviceMissingDelay; /* 0
1840     MPI2_SAS_IO_UNIT1_PHY_DATA PhyData[MPI2_SAS_IOUNIT1_PHY_MAX]; /* 0
1841 } MPI2_CONFIG_PAGE_SASIOUNIT_1,
1842 MPI2_POINTER PTR_MPI2_CONFIG_PAGE_SASIOUNIT_1,
1843   Mpi2SasIOUnitPage1_t, MPI2_POINTER pMpi2SasIOUnitPage1_t;

```

```

1845 #define MPI2_SASIOUNITPAGE1_PAGEVERSION      (0x09)

1847 /* values for SAS IO Unit Page 1 ControlFlags */
1848 #define MPI2_SASIOUNIT1_CONTROL_DEVICE_SELF_TEST      (0x8000)
1849 #define MPI2_SASIOUNIT1_CONTROL_SATA_3_0_MAX          (0x4000)
1850 #define MPI2_SASIOUNIT1_CONTROL_SATA_1_5_MAX          (0x2000)
1851 #define MPI2_SASIOUNIT1_CONTROL_SATA_SW_PRESERVE      (0x1000)

1853 #define MPI2_SASIOUNIT1_CONTROL_MASK_DEV_SUPPORT      (0x0600)
1854 #define MPI2_SASIOUNIT1_CONTROL_SHIFT_DEV_SUPPORT      (9)
1855 #define MPI2_SASIOUNIT1_CONTROL_DEV_SUPPORT_BOTH      (0x0)
1856 #define MPI2_SASIOUNIT1_CONTROL_DEV_SAS_SUPPORT      (0x1)
1857 #define MPI2_SASIOUNIT1_CONTROL_DEV_SATA_SUPPORT      (0x2)

1859 #define MPI2_SASIOUNIT1_CONTROL_SATA_48BIT_LBA_REQUIRED      (0x0080)
1860 #define MPI2_SASIOUNIT1_CONTROL_SATA_SMART_REQUIRED      (0x0040)
1861 #define MPI2_SASIOUNIT1_CONTROL_SATA_NCQ_REQUIRED      (0x0020)
1862 #define MPI2_SASIOUNIT1_CONTROL_SATA_FUA_REQUIRED      (0x0010)
1863 #define MPI2_SASIOUNIT1_CONTROL_TABLE_SUBTRACTIVE_ILLEGAL      (0x0008)
1864 #define MPI2_SASIOUNIT1_CONTROL_SUBTRACTIVE_ILLEGAL      (0x0004)
1865 #define MPI2_SASIOUNIT1_CONTROL_FIRST_LVL_DISC_ONLY      (0x0002)
1866 #define MPI2_SASIOUNIT1_CONTROL_CLEAR_AFFILIATION      (0x0001)

1868 /* values for SAS IO Unit Page 1 AdditionalControlFlags */
1869 #define MPI2_SASIOUNIT1_ACONTROL_MULTI_PORT_DOMAIN_ILLEGAL      (0x0080)
1870 #define MPI2_SASIOUNIT1_ACONTROL_SATA_ASYNCHRONOUS_NOTIFICATION      (0x0040)
1871 #define MPI2_SASIOUNIT1_ACONTROL_INVALID_TOPOLOGY_CORRECTION      (0x0020)
1872 #define MPI2_SASIOUNIT1_ACONTROL_PORT_ENABLE_ONLY_SATA_LINK_RESET      (0x0010)
1873 #define MPI2_SASIOUNIT1_ACONTROL_OTHER_AFFILIATION_SATA_LINK_RESET      (0x0008)
1874 #define MPI2_SASIOUNIT1_ACONTROL_SELF_AFFILIATION_SATA_LINK_RESET      (0x0004)
1875 #define MPI2_SASIOUNIT1_ACONTROL_NO_AFFILIATION_SATA_LINK_RESET      (0x0002)
1876 #define MPI2_SASIOUNIT1_ACONTROL_ALLOW_TABLE_TO_TABLE      (0x0001)

1878 /* defines for SAS IO Unit Page 1 ReportDeviceMissingDelay */
1879 #define MPI2_SASIOUNIT1_REPORT_MISSING_TIMEOUT_MASK      (0x7F)
1880 #define MPI2_SASIOUNIT1_REPORT_MISSING_UNIT_16      (0x80)

1882 /* values for SAS IO Unit Page 1 PortFlags */
1883 #define MPI2_SASIOUNIT1_PORT_FLAGS_AUTO_PORT_CONFIG      (0x01)

1885 /* values for SAS IO Unit Page 1 PhyFlags */
1886 #define MPI2_SASIOUNIT1_PHYFLAGS_ZONING_ENABLE      (0x10)
1887 #define MPI2_SASIOUNIT1_PHYFLAGS_PHY_DISABLE      (0x08)

1889 /* values for SAS IO Unit Page 1 MaxMinLinkRate */
1890 #define MPI2_SASIOUNIT1_MAX_RATE_MASK      (0xF0)
1891 #define MPI2_SASIOUNIT1_MAX_RATE_1_5      (0x80)
1892 #define MPI2_SASIOUNIT1_MAX_RATE_3_0      (0x90)
1893 #define MPI2_SASIOUNIT1_MAX_RATE_6_0      (0xA0)
1894 #define MPI2_SASIOUNIT1_MAX_RATE_12_0      (0xB0)
1895 #define MPI2_SASIOUNIT1_MIN_RATE_MASK      (0x0F)
1896 #define MPI2_SASIOUNIT1_MIN_RATE_1_5      (0x08)
1897 #define MPI2_SASIOUNIT1_MIN_RATE_3_0      (0x09)
1898 #define MPI2_SASIOUNIT1_MIN_RATE_6_0      (0x0A)
1899 #define MPI2_SASIOUNIT1_MIN_RATE_12_0      (0x0B)

1901 /* see mpi2_sas.h for values for SAS IO Unit Page 1 ControllerPhyDeviceInfo valu

1904 /* SAS IO Unit Page 4 */

1906 typedef struct _MPI2_SAS_IOUNIT4_SPINUP_GROUP
1907 {
1908     U8          MaxTargetSpinup;          /* 0x00 */
1909     U8          SpinupDelay;              /* 0x01 */

```

```

1910     U16          Reserved1;              /* 0x02 */
1911 } MPI2_SAS_IOUNIT4_SPINUP_GROUP, MPI2_POINTER PTR_MPI2_SAS_IOUNIT4_SPINUP_GROUP,
1912   Mpi2SasIOUnit4SpinupGroup_t, MPI2_POINTER pMpi2SasIOUnit4SpinupGroup_t;

1914 /*
1915  * Host code (drivers, BIOS, utilities, etc.) should leave this define set to
1916  * four and check Header.ExtPageLength or NumPhys at runtime.
1917  */
1918 #ifndef MPI2_SAS_IOUNIT4_PHY_MAX
1919 #define MPI2_SAS_IOUNIT4_PHY_MAX      (4)
1920 #endif

1922 typedef struct _MPI2_CONFIG_PAGE_SASIOUNIT_4
1923 {
1924     MPI2_CONFIG_EXTENDED_PAGE_HEADER      Header;          /* 0x00
1925     MPI2_SAS_IOUNIT4_SPINUP_GROUP          SpinupGroupParameters[4];      /* 0x08
1926     U32          Reserved1;                /* 0x18
1927     U32          Reserved2;                /* 0x1C
1928     U32          Reserved3;                /* 0x20
1929     U8           BootDeviceWaitTime;       /* 0x24
1930     U8           Reserved4;                /* 0x25
1931     U16          Reserved5;                /* 0x26
1932     U8           NumPhys;                  /* 0x28
1933     U8           PEInitialSpinupDelay;     /* 0x29
1934     U8           PEReplyDelay;              /* 0x2A
1935     U8           Flags;                    /* 0x2B
1936     U8           PHY[MPI2_SAS_IOUNIT4_PHY_MAX]; /* 0x2C
1937 } MPI2_CONFIG_PAGE_SASIOUNIT_4,
1938   MPI2_POINTER PTR_MPI2_CONFIG_PAGE_SASIOUNIT_4,
1939   Mpi2SasIOUnitPage4_t, MPI2_POINTER pMpi2SasIOUnitPage4_t;

1941 #define MPI2_SASIOUNITPAGE4_PAGEVERSION      (0x02)

1943 /* defines for Flags field */
1944 #define MPI2_SASIOUNIT4_FLAGS_AUTO_PORTENABLE      (0x01)

1946 /* defines for PHY field */
1947 #define MPI2_SASIOUNIT4_PHY_SPINUP_GROUP_MASK      (0x03)

1950 /* SAS IO Unit Page 5 */

1952 typedef struct _MPI2_SAS_IO_UNIT5_PHY_PM_SETTINGS
1953 {
1954     U8           ControlFlags;              /* 0x00 */
1955     U8           PortWidthModGroup;         /* 0x01 */
1956     U16          InactivityTimerExponent;   /* 0x02 */
1957     U8           SATAPartialTimeout;        /* 0x04 */
1958     U8           Reserved2;                 /* 0x05 */
1959     U8           SASLumberTimeout;          /* 0x06 */
1960     U8           Reserved3;                 /* 0x07 */
1961     U8           SASPartialTimeout;         /* 0x08 */
1962     U8           Reserved4;                 /* 0x09 */
1963     U8           SASSlumberTimeout;         /* 0x0A */
1964     U8           Reserved5;                 /* 0x0B */
1965 } MPI2_SAS_IO_UNIT5_PHY_PM_SETTINGS,
1966   MPI2_POINTER PTR_MPI2_SAS_IO_UNIT5_PHY_PM_SETTINGS,
1967   Mpi2SasIOUnit5PhyPmSettings_t, MPI2_POINTER pMpi2SasIOUnit5PhyPmSettings_t;

1969 /* defines for ControlFlags field */
1970 #define MPI2_SASIOUNIT5_CONTROL_SAS_SLUMBER_ENABLE      (0x08)
1971 #define MPI2_SASIOUNIT5_CONTROL_SAS_PARTIAL_ENABLE      (0x04)
1972 #define MPI2_SASIOUNIT5_CONTROL_SATA_SLUMBER_ENABLE      (0x02)
1973 #define MPI2_SASIOUNIT5_CONTROL_SATA_PARTIAL_ENABLE      (0x01)

1975 /* defines for PortWidthModeGroup field */

```

```

1976 #define MPI2_SASIOUNIT5_PWMG_DISABLE (0xFF)

1978 /* defines for InactivityTimerExponent field */
1979 #define MPI2_SASIOUNIT5_ITE_MASK_SAS_SLUMBER (0x7000)
1980 #define MPI2_SASIOUNIT5_ITE_SHIFT_SAS_SLUMBER (12)
1981 #define MPI2_SASIOUNIT5_ITE_MASK_SAS_PARTIAL (0x0700)
1982 #define MPI2_SASIOUNIT5_ITE_SHIFT_SAS_PARTIAL (8)
1983 #define MPI2_SASIOUNIT5_ITE_MASK_SATA_SLUMBER (0x0070)
1984 #define MPI2_SASIOUNIT5_ITE_SHIFT_SATA_SLUMBER (4)
1985 #define MPI2_SASIOUNIT5_ITE_MASK_SATA_PARTIAL (0x0007)
1986 #define MPI2_SASIOUNIT5_ITE_SHIFT_SATA_PARTIAL (0)

1988 #define MPI2_SASIOUNIT5_ITE_TEN_SECONDS (7)
1989 #define MPI2_SASIOUNIT5_ITE_ONE_SECOND (6)
1990 #define MPI2_SASIOUNIT5_ITE_HUNDRED_MILLISECONDS (5)
1991 #define MPI2_SASIOUNIT5_ITE_TEN_MILLISECONDS (4)
1992 #define MPI2_SASIOUNIT5_ITE_ONE_MILLISECOND (3)
1993 #define MPI2_SASIOUNIT5_ITE_HUNDRED_MICROSECONDS (2)
1994 #define MPI2_SASIOUNIT5_ITE_TEN_MICROSECONDS (1)
1995 #define MPI2_SASIOUNIT5_ITE_ONE_MICROSECOND (0)

1997 /*
1998  * Host code (drivers, BIOS, utilities, etc.) should leave this define set to
1999  * one and check Header.ExtPageLength or NumPhys at runtime.
2000  */
2001 #ifndef MPI2_SAS_IOUNIT5_PHY_MAX
2002 #define MPI2_SAS_IOUNIT5_PHY_MAX (1)
2003 #endif

2005 typedef struct _MPI2_CONFIG_PAGE_SASIOUNIT_5
2006 {
2007     MPI2_CONFIG_EXTENDED_PAGE_HEADER Header; /* 0
2008     U8 NumPhys; /* 0
2009     U8 Reserved1; /* 0
2010     U16 Reserved2; /* 0
2011     U32 Reserved3; /* 0
2012     MPI2_SAS_IO_UNIT5_PHY_PM_SETTINGS SASPhyPowerManagementSettings[MPI2_SAS_I
2013 } MPI2_CONFIG_PAGE_SASIOUNIT_5,
2014 MPI2_POINTER PTR_MPI2_CONFIG_PAGE_SASIOUNIT_5,
2015 Mpi2SasIOUnitPage5_t, MPI2_POINTER pMpi2SasIOUnitPage5_t;

2017 #define MPI2_SASIOUNITPAGE5_PAGEVERSION (0x01)

2020 /* SAS IO Unit Page 6 */

2022 typedef struct _MPI2_SAS_IO_UNIT6_PORT_WIDTH_MOD_GROUP_STATUS
2023 {
2024     U8 CurrentStatus; /* 0x00 */
2025     U8 CurrentModulation; /* 0x01 */
2026     U8 CurrentUtilization; /* 0x02 */
2027     U8 Reserved1; /* 0x03 */
2028     U32 Reserved2; /* 0x04 */
2029 } MPI2_SAS_IO_UNIT6_PORT_WIDTH_MOD_GROUP_STATUS,
2030 MPI2_POINTER PTR_MPI2_SAS_IO_UNIT6_PORT_WIDTH_MOD_GROUP_STATUS,
2031 Mpi2SasIOUnit6PortWidthModGroupStatus_t,
2032 MPI2_POINTER pMpi2SasIOUnit6PortWidthModGroupStatus_t;

2034 /* defines for CurrentStatus field */
2035 #define MPI2_SASIOUNIT6_STATUS_UNAVAILABLE (0x00)
2036 #define MPI2_SASIOUNIT6_STATUS_UNCONFIGURED (0x01)
2037 #define MPI2_SASIOUNIT6_STATUS_INVALID_CONFIG (0x02)
2038 #define MPI2_SASIOUNIT6_STATUS_LINK_DOWN (0x03)
2039 #define MPI2_SASIOUNIT6_STATUS_OBSERVATION_ONLY (0x04)
2040 #define MPI2_SASIOUNIT6_STATUS_INACTIVE (0x05)
2041 #define MPI2_SASIOUNIT6_STATUS_ACTIVE_IOUNIT (0x06)

```

```

2042 #define MPI2_SASIOUNIT6_STATUS_ACTIVE_HOST (0x07)

2044 /* defines for CurrentModulation field */
2045 #define MPI2_SASIOUNIT6_MODULATION_25_PERCENT (0x00)
2046 #define MPI2_SASIOUNIT6_MODULATION_50_PERCENT (0x01)
2047 #define MPI2_SASIOUNIT6_MODULATION_75_PERCENT (0x02)
2048 #define MPI2_SASIOUNIT6_MODULATION_100_PERCENT (0x03)

2050 /*
2051  * Host code (drivers, BIOS, utilities, etc.) should leave this define set to
2052  * one and check the value returned for NumGroups at runtime.
2053  */
2054 #ifndef MPI2_SAS_IOUNIT6_GROUP_MAX
2055 #define MPI2_SAS_IOUNIT6_GROUP_MAX (1)
2056 #endif

2058 typedef struct _MPI2_CONFIG_PAGE_SASIOUNIT_6
2059 {
2060     MPI2_CONFIG_EXTENDED_PAGE_HEADER Header; /* 0x00 */
2061     U32 Reserved1; /* 0x08 */
2062     U32 Reserved2; /* 0x0C */
2063     U8 NumGroups; /* 0x10 */
2064     U8 Reserved3; /* 0x11 */
2065     U16 Reserved4; /* 0x12 */
2066     MPI2_SAS_IO_UNIT6_PORT_WIDTH_MOD_GROUP_STATUS
2067     PortWidthModulationGroupStatus[MPI2_SAS_IOUNIT6_GROUP_MAX]; /* 0x14 */
2068 } MPI2_CONFIG_PAGE_SASIOUNIT_6,
2069 MPI2_POINTER PTR_MPI2_CONFIG_PAGE_SASIOUNIT_6,
2070 Mpi2SasIOUnitPage6_t, MPI2_POINTER pMpi2SasIOUnitPage6_t;

2072 #define MPI2_SASIOUNITPAGE6_PAGEVERSION (0x00)

2075 /* SAS IO Unit Page 7 */

2077 typedef struct _MPI2_SAS_IO_UNIT7_PORT_WIDTH_MOD_GROUP_SETTINGS
2078 {
2079     U8 Flags; /* 0x00 */
2080     U8 Reserved1; /* 0x01 */
2081     U16 Reserved2; /* 0x02 */
2082     U8 Threshold75Pct; /* 0x04 */
2083     U8 Threshold50Pct; /* 0x05 */
2084     U8 Threshold25Pct; /* 0x06 */
2085     U8 Reserved3; /* 0x07 */
2086 } MPI2_SAS_IO_UNIT7_PORT_WIDTH_MOD_GROUP_SETTINGS,
2087 MPI2_POINTER PTR_MPI2_SAS_IO_UNIT7_PORT_WIDTH_MOD_GROUP_SETTINGS,
2088 Mpi2SasIOUnit7PortWidthModGroupSettings_t,
2089 MPI2_POINTER pMpi2SasIOUnit7PortWidthModGroupSettings_t;

2091 /* defines for Flags field */
2092 #define MPI2_SASIOUNIT7_FLAGS_ENABLE_PORT_WIDTH_MODULATION (0x01)

2095 /*
2096  * Host code (drivers, BIOS, utilities, etc.) should leave this define set to
2097  * one and check the value returned for NumGroups at runtime.
2098  */
2099 #ifndef MPI2_SAS_IOUNIT7_GROUP_MAX
2100 #define MPI2_SAS_IOUNIT7_GROUP_MAX (1)
2101 #endif

2103 typedef struct _MPI2_CONFIG_PAGE_SASIOUNIT_7
2104 {
2105     MPI2_CONFIG_EXTENDED_PAGE_HEADER Header; /* 0x00 */
2106     U8 SamplingInterval; /* 0x08 */
2107     U8 WindowLength; /* 0x09 */

```



```

2240     U32      AttachedDeviceInfo;          /* 0x18 */
2241     U16      ExpanderDevHandle;           /* 0x1C */
2242     U8       ChangeCount;                  /* 0x1E */
2243     U8       NegotiatedLinkRate;          /* 0x1F */
2244     U8       PhyIdentifier;                /* 0x20 */
2245     U8       AttachedPhyIdentifier;        /* 0x21 */
2246     U8       Reserved3;                    /* 0x22 */
2247     U8       DiscoveryInfo;                /* 0x23 */
2248     U32      AttachedPhyInfo;              /* 0x24 */
2249     U8       ZoneGroup;                    /* 0x28 */
2250     U8       SelfConfigStatus;             /* 0x29 */
2251     U16      Reserved4;                    /* 0x2A */
2252 } MPI2_CONFIG_PAGE_EXPANDER_1, MPI2_POINTER PTR_MPI2_CONFIG_PAGE_EXPANDER_1,
2253   Mpi2ExpanderPage1_t, MPI2_POINTER pMpi2ExpanderPage1_t;

2255 #define MPI2_SASEXPANDER1_PAGEVERSION      (0x02)

2257 /* use MPI2_SAS_PRATE_ defines for the ProgrammedLinkRate field */

2259 /* use MPI2_SAS_HWRATE_ defines for the HwLinkRate field */

2261 /* use MPI2_SAS_PHYINFO_ for the PhyInfo field */

2263 /* see mpi2_sas.h for the MPI2_SAS_DEVICE_INFO_ defines used for the AttachedDev

2265 /* use MPI2_SAS_NEG_LINK_RATE_ defines for the NegotiatedLinkRate field */

2267 /* use MPI2_SAS_APHYINFO_ defines for AttachedPhyInfo field */

2269 /* values for SAS Expander Page 1 DiscoveryInfo field */
2270 #define MPI2_SAS_EXPANDER1_DISCINFO_BAD_PHY_DISABLED (0x04)
2271 #define MPI2_SAS_EXPANDER1_DISCINFO_LINK_STATUS_CHANGE (0x02)
2272 #define MPI2_SAS_EXPANDER1_DISCINFO_NO_ROUTING_ENTRIES (0x01)

2275 /*****
2276 * SAS Device Config Pages
2277 *****/

2279 /* SAS Device Page 0 */

2281 typedef struct _MPI2_CONFIG_PAGE_SAS_DEV_0
2282 {
2283     MPI2_CONFIG_EXTENDED_PAGE_HEADER Header;          /* 0x00 */
2284     U16 Slot;                                          /* 0x08 */
2285     U16 EnclosureHandle;                              /* 0x0A */
2286     U64 SASAddress;                                   /* 0x0C */
2287     U16 ParentDevHandle;                              /* 0x14 */
2288     U8 PhyNum;                                        /* 0x16 */
2289     U8 AccessStatus;                                  /* 0x17 */
2290     U16 DevHandle;                                    /* 0x18 */
2291     U8 AttachedPhyIdentifier;                          /* 0x1A */
2292     U8 ZoneGroup;                                     /* 0x1B */
2293     U32 DeviceInfo;                                   /* 0x1C */
2294     U16 Flags;                                        /* 0x20 */
2295     U8 PhysicalPort;                                  /* 0x22 */
2296     U8 MaxPortConnections;                            /* 0x23 */
2297     U64 DeviceName;                                   /* 0x24 */
2298     U8 PortGroups;                                    /* 0x2C */
2299     U8 DmaGroup;                                      /* 0x2D */
2300     U8 ControlGroup;                                  /* 0x2E */
2301     U8 Reserved1;                                     /* 0x2F */
2302     U32 Reserved2;                                    /* 0x30 */
2303     U32 Reserved3;                                    /* 0x34 */
2304 } MPI2_CONFIG_PAGE_SAS_DEV_0, MPI2_POINTER PTR_MPI2_CONFIG_PAGE_SAS_DEV_0,
2305   Mpi2SasDevicePage0_t, MPI2_POINTER pMpi2SasDevicePage0_t;

```

```

2307 #define MPI2_SASDEVICE0_PAGEVERSION      (0x08)

2309 /* values for SAS Device Page 0 AccessStatus field */
2310 #define MPI2_SAS_DEVICE0_ASTATUS_NO_ERRORS (0x00)
2311 #define MPI2_SAS_DEVICE0_ASTATUS_SATA_INIT_FAILED (0x01)
2312 #define MPI2_SAS_DEVICE0_ASTATUS_SATA_CAPABILITY_FAILED (0x02)
2313 #define MPI2_SAS_DEVICE0_ASTATUS_SATA_AFFILIATION_CONFLICT (0x03)
2314 #define MPI2_SAS_DEVICE0_ASTATUS_SATA_NEEDS_INITIALIZATION (0x04)
2315 #define MPI2_SAS_DEVICE0_ASTATUS_ROUTE_NOT_ADDRESSABLE (0x05)
2316 #define MPI2_SAS_DEVICE0_ASTATUS_SMP_ERROR_NOT_ADDRESSABLE (0x06)
2317 #define MPI2_SAS_DEVICE0_ASTATUS_DEVICE_BLOCKED (0x07)
2318 /* specific values for SATA Init failures */
2319 #define MPI2_SAS_DEVICE0_ASTATUS_SIF_UNKNOWN (0x10)
2320 #define MPI2_SAS_DEVICE0_ASTATUS_SIF_AFFILIATION_CONFLICT (0x11)
2321 #define MPI2_SAS_DEVICE0_ASTATUS_SIF_DIAG (0x12)
2322 #define MPI2_SAS_DEVICE0_ASTATUS_SIF_IDENTIFICATION (0x13)
2323 #define MPI2_SAS_DEVICE0_ASTATUS_SIF_CHECK_POWER (0x14)
2324 #define MPI2_SAS_DEVICE0_ASTATUS_SIF_PIO_SN (0x15)
2325 #define MPI2_SAS_DEVICE0_ASTATUS_SIF_MDMA_SN (0x16)
2326 #define MPI2_SAS_DEVICE0_ASTATUS_SIF_UDMA_SN (0x17)
2327 #define MPI2_SAS_DEVICE0_ASTATUS_SIF_ZONING_VIOLATION (0x18)
2328 #define MPI2_SAS_DEVICE0_ASTATUS_SIF_NOT_ADDRESSABLE (0x19)
2329 #define MPI2_SAS_DEVICE0_ASTATUS_SIF_MAX (0x1F)

2331 /* see mpi2_sas.h for values for SAS Device Page 0 DeviceInfo values */

2333 /* values for SAS Device Page 0 Flags field */
2334 #define MPI25_SAS_DEVICE0_FLAGS_ENABLED_FAST_PATH (0x4000)
2335 #define MPI25_SAS_DEVICE0_FLAGS_FAST_PATH_CAPABLE (0x2000)
2336 #define MPI2_SAS_DEVICE0_FLAGS_SLUMBER_PM_CAPABLE (0x1000)
2337 #define MPI2_SAS_DEVICE0_FLAGS_PARTIAL_PM_CAPABLE (0x0800)
2338 #define MPI2_SAS_DEVICE0_FLAGS_SATA_ASYNCRONOUS_NOTIFY (0x0400)
2339 #define MPI2_SAS_DEVICE0_FLAGS_SATA_SW_PRESERVE (0x0200)
2340 #define MPI2_SAS_DEVICE0_FLAGS_UNSUPPORTED_DEVICE (0x0100)
2341 #define MPI2_SAS_DEVICE0_FLAGS_SATA_48BIT_LBA_SUPPORTED (0x0080)
2342 #define MPI2_SAS_DEVICE0_FLAGS_SATA_SMART_SUPPORTED (0x0040)
2343 #define MPI2_SAS_DEVICE0_FLAGS_SATA_NCQ_SUPPORTED (0x0020)
2344 #define MPI2_SAS_DEVICE0_FLAGS_SATA_FUA_SUPPORTED (0x0010)
2345 #define MPI2_SAS_DEVICE0_FLAGS_PORT_SELECTOR_ATTACH (0x0008)
2346 #define MPI2_SAS_DEVICE0_FLAGS_DEVICE_PRESENT (0x0001)

2349 /* SAS Device Page 1 */

2351 typedef struct _MPI2_CONFIG_PAGE_SAS_DEV_1
2352 {
2353     MPI2_CONFIG_EXTENDED_PAGE_HEADER Header;          /* 0x00 */
2354     U32 U32;                                          /* 0x08 */
2355     U64 SASAddress;                                   /* 0x0C */
2356     U32 U32;                                          /* 0x14 */
2357     U16 DevHandle;                                    /* 0x18 */
2358     U16 U16;                                          /* 0x1A */
2359     U8 U8;                                           /* 0x1C */
2360 } MPI2_CONFIG_PAGE_SAS_DEV_1, MPI2_POINTER PTR_MPI2_CONFIG_PAGE_SAS_DEV_1,
2361   Mpi2SasDevicePage1_t, MPI2_POINTER pMpi2SasDevicePage1_t;

2363 #define MPI2_SASDEVICE1_PAGEVERSION      (0x01)

2366 /*****
2367 * SAS PHY Config Pages
2368 *****/

2370 /* SAS PHY Page 0 */

```

```

2372 typedef struct _MPI2_CONFIG_PAGE_SAS_PHY_0
2373 {
2374     MPI2_CONFIG_EXTENDED_PAGE_HEADER Header;          /* 0x00 */
2375     U16 OwnerDevHandle;                          /* 0x08 */
2376     U16 Reserved1;                                /* 0x0A */
2377     U16 AttachedDevHandle;                        /* 0x0C */
2378     U8 AttachedPhyIdentifier;                     /* 0x0E */
2379     U8 Reserved2;                                /* 0x0F */
2380     U32 AttachedPhyInfo;                          /* 0x10 */
2381     U8 ProgrammedLinkRate;                       /* 0x14 */
2382     U8 HwLinkRate;                               /* 0x15 */
2383     U8 ChangeCount;                              /* 0x16 */
2384     U8 Flags;                                    /* 0x17 */
2385     U32 PhyInfo;                                 /* 0x18 */
2386     U8 NegotiatedLinkRate;                      /* 0x1C */
2387     U8 Reserved3;                               /* 0x1D */
2388     U16 Reserved4;                              /* 0x1E */
2389 } MPI2_CONFIG_PAGE_SAS_PHY_0, MPI2_POINTER PTR_MPI2_CONFIG_PAGE_SAS_PHY_0,
2390     Mpi2SasPhyPage0_t, MPI2_POINTER pMpi2SasPhyPage0_t;

2392 #define MPI2_SASPHY0_PAGEVERSION                (0x03)

2394 /* use MPI2_SAS_PRATE_ defines for the ProgrammedLinkRate field */

2396 /* use MPI2_SAS_HWRATE_ defines for the HwLinkRate field */

2398 /* values for SAS PHY Page 0 Flags field */
2399 #define MPI2_SAS_PHY0_FLAGS_SGPIO_DIRECT_ATTACH_ENC    (0x01)

2401 /* use MPI2_SAS_APHYINFO_ defines for AttachedPhyInfo field */

2403 /* use MPI2_SAS_NEG_LINK_RATE_ defines for the NegotiatedLinkRate field */

2405 /* use MPI2_SAS_PHYINFO_ for the PhyInfo field */

2408 /* SAS PHY Page 1 */

2410 typedef struct _MPI2_CONFIG_PAGE_SAS_PHY_1
2411 {
2412     MPI2_CONFIG_EXTENDED_PAGE_HEADER Header;          /* 0x00 */
2413     U32 Reserved1;                                /* 0x08 */
2414     U32 InvalidDwordCount;                        /* 0x0C */
2415     U32 RunningDisparityErrorCount;               /* 0x10 */
2416     U32 LossDwordSynchCount;                     /* 0x14 */
2417     U32 PhyResetProblemCount;                    /* 0x18 */
2418 } MPI2_CONFIG_PAGE_SAS_PHY_1, MPI2_POINTER PTR_MPI2_CONFIG_PAGE_SAS_PHY_1,
2419     Mpi2SasPhyPage1_t, MPI2_POINTER pMpi2SasPhyPage1_t;

2421 #define MPI2_SASPHY1_PAGEVERSION                (0x01)

2424 /* SAS PHY Page 2 */

2426 typedef struct _MPI2_SASPHY2_PHY_EVENT
2427 {
2428     U8 PhyEventCode;                             /* 0x00 */
2429     U8 Reserved1;                                /* 0x01 */
2430     U16 Reserved2;                                /* 0x02 */
2431     U32 PhyEventInfo;                            /* 0x04 */
2432 } MPI2_SASPHY2_PHY_EVENT, MPI2_POINTER PTR_MPI2_SASPHY2_PHY_EVENT,
2433     Mpi2SasPhy2PhyEvent_t, MPI2_POINTER pMpi2SasPhy2PhyEvent_t;

2435 /* use MPI2_SASPHY3_EVENT_CODE_ for the PhyEventCode field */

```

```

2438 /*
2439  * Host code (drivers, BIOS, utilities, etc.) should leave this define set to
2440  * one and check Header.ExtPageLength or NumPhyEvents at runtime.
2441  */
2442 #ifndef MPI2_SASPHY2_PHY_EVENT_MAX
2443 #define MPI2_SASPHY2_PHY_EVENT_MAX                (1)
2444 #endif

2446 typedef struct _MPI2_CONFIG_PAGE_SAS_PHY_2
2447 {
2448     MPI2_CONFIG_EXTENDED_PAGE_HEADER Header;          /* 0x00 */
2449     U32 Reserved1;                                /* 0x08 */
2450     U8 NumPhyEvents;                              /* 0x0C */
2451     U8 Reserved2;                                /* 0x0D */
2452     U16 Reserved3;                              /* 0x0E */
2453     MPI2_SASPHY2_PHY_EVENT PhyEvent[MPI2_SASPHY2_PHY_EVENT_MAX]; /*
2454 } MPI2_CONFIG_PAGE_SAS_PHY_2, MPI2_POINTER PTR_MPI2_CONFIG_PAGE_SAS_PHY_2,
2455     Mpi2SasPhyPage2_t, MPI2_POINTER pMpi2SasPhyPage2_t;

2457 #define MPI2_SASPHY2_PAGEVERSION                (0x00)

2460 /* SAS PHY Page 3 */

2462 typedef struct _MPI2_SASPHY3_PHY_EVENT_CONFIG
2463 {
2464     U8 PhyEventCode;                             /* 0x00 */
2465     U8 Reserved1;                                /* 0x01 */
2466     U16 Reserved2;                                /* 0x02 */
2467     U8 CounterType;                              /* 0x04 */
2468     U8 ThresholdWindow;                         /* 0x05 */
2469     U8 TimeUnits;                               /* 0x06 */
2470     U8 Reserved3;                               /* 0x07 */
2471     U32 EventThreshold;                         /* 0x08 */
2472     U16 ThresholdFlags;                        /* 0x0C */
2473     U16 Reserved4;                              /* 0x0E */
2474 } MPI2_SASPHY3_PHY_EVENT_CONFIG, MPI2_POINTER PTR_MPI2_SASPHY3_PHY_EVENT_CONFIG,
2475     Mpi2SasPhy3PhyEventConfig_t, MPI2_POINTER pMpi2SasPhy3PhyEventConfig_t;

2477 /* values for PhyEventCode field */
2478 #define MPI2_SASPHY3_EVENT_CODE_NO_EVENT                (0x00)
2479 #define MPI2_SASPHY3_EVENT_CODE_INVALID_DWORD          (0x01)
2480 #define MPI2_SASPHY3_EVENT_CODE_RUNNING_DISPARITY_ERROR (0x02)
2481 #define MPI2_SASPHY3_EVENT_CODE_LOSS_DWORD_SYNC        (0x03)
2482 #define MPI2_SASPHY3_EVENT_CODE_PHY_RESET_PROBLEM      (0x04)
2483 #define MPI2_SASPHY3_EVENT_CODE_ELASTICITY_BUF_OVERFLOW (0x05)
2484 #define MPI2_SASPHY3_EVENT_CODE_RX_ERROR                (0x06)
2485 #define MPI2_SASPHY3_EVENT_CODE_RX_ADDR_FRAME_ERROR    (0x20)
2486 #define MPI2_SASPHY3_EVENT_CODE_TX_AC_OPEN_REJECT      (0x21)
2487 #define MPI2_SASPHY3_EVENT_CODE_RX_AC_OPEN_REJECT      (0x22)
2488 #define MPI2_SASPHY3_EVENT_CODE_TX_RC_OPEN_REJECT      (0x23)
2489 #define MPI2_SASPHY3_EVENT_CODE_RX_RC_OPEN_REJECT      (0x24)
2490 #define MPI2_SASPHY3_EVENT_CODE_RX_AIP_PARTIAL_WAITING_ON (0x25)
2491 #define MPI2_SASPHY3_EVENT_CODE_RX_AIP_CONNECT_WAITING_ON (0x26)
2492 #define MPI2_SASPHY3_EVENT_CODE_TX_BREAK               (0x27)
2493 #define MPI2_SASPHY3_EVENT_CODE_RX_BREAK               (0x28)
2494 #define MPI2_SASPHY3_EVENT_CODE_BREAK_TIMEOUT          (0x29)
2495 #define MPI2_SASPHY3_EVENT_CODE_CONNECTION             (0x2A)
2496 #define MPI2_SASPHY3_EVENT_CODE_PEAKTX_PATHWAY_BLOCKED (0x2B)
2497 #define MPI2_SASPHY3_EVENT_CODE_PEAKTX_ARB_WAIT_TIME   (0x2C)
2498 #define MPI2_SASPHY3_EVENT_CODE_PEAK_ARB_WAIT_TIME     (0x2D)
2499 #define MPI2_SASPHY3_EVENT_CODE_PEAK_CONNECT_TIME      (0x2E)
2500 #define MPI2_SASPHY3_EVENT_CODE_TX_SSP_FRAMES          (0x40)
2501 #define MPI2_SASPHY3_EVENT_CODE_RX_SSP_FRAMES          (0x41)
2502 #define MPI2_SASPHY3_EVENT_CODE_TX_SSP_ERROR_FRAMES    (0x42)
2503 #define MPI2_SASPHY3_EVENT_CODE_RX_SSP_ERROR_FRAMES    (0x43)

```

```

2504 #define MPI2_SASPHY3_EVENT_CODE_TX_CREDIT_BLOCKED (0x44)
2505 #define MPI2_SASPHY3_EVENT_CODE_RX_CREDIT_BLOCKED (0x45)
2506 #define MPI2_SASPHY3_EVENT_CODE_TX_SATA_FRAMES (0x50)
2507 #define MPI2_SASPHY3_EVENT_CODE_RX_SATA_FRAMES (0x51)
2508 #define MPI2_SASPHY3_EVENT_CODE_SATA_OVERFLOW (0x52)
2509 #define MPI2_SASPHY3_EVENT_CODE_TX_SMP_FRAMES (0x60)
2510 #define MPI2_SASPHY3_EVENT_CODE_RX_SMP_FRAMES (0x61)
2511 #define MPI2_SASPHY3_EVENT_CODE_RX_SMP_ERROR_FRAMES (0x63)
2512 #define MPI2_SASPHY3_EVENT_CODE_HOTPLUG_TIMEOUT (0xD0)
2513 #define MPI2_SASPHY3_EVENT_CODE_MISALIGNED_MUX_PRIMITIVE (0xD1)
2514 #define MPI2_SASPHY3_EVENT_CODE_RX_AIP (0xD2)

2516 /* values for the CounterType field */
2517 #define MPI2_SASPHY3_COUNTER_TYPE_WRAPPING (0x00)
2518 #define MPI2_SASPHY3_COUNTER_TYPE_SATURATING (0x01)
2519 #define MPI2_SASPHY3_COUNTER_TYPE_PEAK_VALUE (0x02)

2521 /* values for the TimeUnits field */
2522 #define MPI2_SASPHY3_TIME_UNITS_10_MICROSECONDS (0x00)
2523 #define MPI2_SASPHY3_TIME_UNITS_100_MICROSECONDS (0x01)
2524 #define MPI2_SASPHY3_TIME_UNITS_1_MILLISECOND (0x02)
2525 #define MPI2_SASPHY3_TIME_UNITS_10_MILLISECONDS (0x03)

2527 /* values for the ThresholdFlags field */
2528 #define MPI2_SASPHY3_TFLAGS_PHY_RESET (0x0002)
2529 #define MPI2_SASPHY3_TFLAGS_EVENT_NOTIFY (0x0001)

2531 /*
2532 * Host code (drivers, BIOS, utilities, etc.) should leave this define set to
2533 * one and check Header.ExtPageLength or NumPhyEvents at runtime.
2534 */
2535 #ifndef MPI2_SASPHY3_PHY_EVENT_MAX
2536 #define MPI2_SASPHY3_PHY_EVENT_MAX (1)
2537 #endif

2539 typedef struct _MPI2_CONFIG_PAGE_SAS_PHY_3
2540 {
2541     MPI2_CONFIG_EXTENDED_PAGE_HEADER Header; /* 0x00 */
2542     U32 Reserved1; /* 0x08 */
2543     U8 NumPhyEvents; /* 0x0C */
2544     U8 Reserved2; /* 0x0D */
2545     U16 Reserved3; /* 0x0E */
2546     MPI2_SASPHY3_PHY_EVENT_CONFIG PhyEventConfig[MPI2_SASPHY3_PHY_EVENT_MAX];
2547 } MPI2_CONFIG_PAGE_SAS_PHY_3, MPI2_POINTER PTR_MPI2_CONFIG_PAGE_SAS_PHY_3,
2548     Mpi2SasPhyPage3_t, MPI2_POINTER pMpi2SasPhyPage3_t;

2550 #define MPI2_SASPHY3_PAGEVERSION (0x00)

2553 /* SAS PHY Page 4 */

2555 typedef struct _MPI2_CONFIG_PAGE_SAS_PHY_4
2556 {
2557     MPI2_CONFIG_EXTENDED_PAGE_HEADER Header; /* 0x00 */
2558     U16 Reserved1; /* 0x08 */
2559     U8 Reserved2; /* 0x0A */
2560     U8 Flags; /* 0x0B */
2561     U8 InitialFrame[28]; /* 0x0C */
2562 } MPI2_CONFIG_PAGE_SAS_PHY_4, MPI2_POINTER PTR_MPI2_CONFIG_PAGE_SAS_PHY_4,
2563     Mpi2SasPhyPage4_t, MPI2_POINTER pMpi2SasPhyPage4_t;

2565 #define MPI2_SASPHY4_PAGEVERSION (0x00)

2567 /* values for the Flags field */
2568 #define MPI2_SASPHY4_FLAGS_FRAME_VALID (0x02)
2569 #define MPI2_SASPHY4_FLAGS_SATA_FRAME (0x01)

```

```

2574 /*****
2575 * SAS Port Config Pages
2576 *****/

2578 /* SAS Port Page 0 */

2580 typedef struct _MPI2_CONFIG_PAGE_SAS_PORT_0
2581 {
2582     MPI2_CONFIG_EXTENDED_PAGE_HEADER Header; /* 0x00 */
2583     U8 PortNumber; /* 0x08 */
2584     U8 PhysicalPort; /* 0x09 */
2585     U8 PortWidth; /* 0x0A */
2586     U8 PhysicalPortWidth; /* 0x0B */
2587     U8 ZoneGroup; /* 0x0C */
2588     U8 Reserved1; /* 0x0D */
2589     U16 Reserved2; /* 0x0E */
2590     U64 SASAddress; /* 0x10 */
2591     U32 DeviceInfo; /* 0x18 */
2592     U32 Reserved3; /* 0x1C */
2593     U32 Reserved4; /* 0x20 */
2594 } MPI2_CONFIG_PAGE_SAS_PORT_0, MPI2_POINTER PTR_MPI2_CONFIG_PAGE_SAS_PORT_0,
2595     Mpi2SasPortPage0_t, MPI2_POINTER pMpi2SasPortPage0_t;

2597 #define MPI2_SASPORT0_PAGEVERSION (0x00)

2599 /* see mpi2_sas.h for values for SAS Port Page 0 DeviceInfo values */

2602 /*****
2603 * SAS Enclosure Config Pages
2604 *****/

2606 /* SAS Enclosure Page 0 */

2608 typedef struct _MPI2_CONFIG_PAGE_SAS_ENCLOSURE_0
2609 {
2610     MPI2_CONFIG_EXTENDED_PAGE_HEADER Header; /* 0x00 */
2611     U32 Reserved1; /* 0x08 */
2612     U64 EnclosureLogicalID; /* 0x0C */
2613     U16 Flags; /* 0x14 */
2614     U16 EnclosureHandle; /* 0x16 */
2615     U16 NumSlots; /* 0x18 */
2616     U16 StartSlot; /* 0x1A */
2617     U16 Reserved2; /* 0x1C */
2618     U16 SEPDevHandle; /* 0x1E */
2619     U32 Reserved3; /* 0x20 */
2620     U32 Reserved4; /* 0x24 */
2621 } MPI2_CONFIG_PAGE_SAS_ENCLOSURE_0,
2622     MPI2_POINTER PTR_MPI2_CONFIG_PAGE_SAS_ENCLOSURE_0,
2623     Mpi2SasEnclosurePage0_t, MPI2_POINTER pMpi2SasEnclosurePage0_t;

2625 #define MPI2_SASENCLOSURE0_PAGEVERSION (0x03)

2627 /* values for SAS Enclosure Page 0 Flags field */
2628 #define MPI2_SAS_ENCLS0_FLAGS_MNG_MASK (0x000F)
2629 #define MPI2_SAS_ENCLS0_FLAGS_MNG_UNKNOWN (0x0000)
2630 #define MPI2_SAS_ENCLS0_FLAGS_MNG_IOC_SES (0x0001)
2631 #define MPI2_SAS_ENCLS0_FLAGS_MNG_IOC_SGPIO (0x0002)
2632 #define MPI2_SAS_ENCLS0_FLAGS_MNG_EXP_SGPIO (0x0003)
2633 #define MPI2_SAS_ENCLS0_FLAGS_MNG_SES_ENCLOSURE (0x0004)
2634 #define MPI2_SAS_ENCLS0_FLAGS_MNG_IOC_GPIO (0x0005)

```

```

2637 /*****
2638 *   Log Config Page
2639 *****/
2641 /* Log Page 0 */

2643 /*
2644 * Host code (drivers, BIOS, utilities, etc.) should leave this define set to
2645 * one and check Header.ExtPageLength or NumPhys at runtime.
2646 */
2647 #ifndef MPI2_LOG_0_NUM_LOG_ENTRIES
2648 #define MPI2_LOG_0_NUM_LOG_ENTRIES      (1)
2649 #endif

2651 #define MPI2_LOG_0_LOG_DATA_LENGTH      (0x1C)

2653 typedef struct _MPI2_LOG_0_ENTRY
2654 {
2655     U64      TimeStamp;                /* 0x00 */
2656     U32      Reserved1;                /* 0x08 */
2657     U16      LogSequence;              /* 0x0C */
2658     U16      LogEntryQualifier;        /* 0x0E */
2659     U8       VP_ID;                    /* 0x10 */
2660     U8       VF_ID;                    /* 0x11 */
2661     U16      Reserved2;                /* 0x12 */
2662     U8       LogData[MPI2_LOG_0_LOG_DATA_LENGTH]; /* 0x14 */
2663 } MPI2_LOG_0_ENTRY, MPI2_POINTER PTR_MPI2_LOG_0_ENTRY,
2664   Mpi2Log0Entry_t, MPI2_POINTER pMpi2Log0Entry_t;

2666 /* values for Log Page 0 LogEntry LogEntryQualifier field */
2667 #define MPI2_LOG_0_ENTRY_QUAL_ENTRY_UNUSED      (0x0000)
2668 #define MPI2_LOG_0_ENTRY_QUAL_POWER_ON_RESET   (0x0001)
2669 #define MPI2_LOG_0_ENTRY_QUAL_TIMESTAMP_UPDATE (0x0002)
2670 #define MPI2_LOG_0_ENTRY_QUAL_MIN_IMPLEMENT_SPEC (0x8000)
2671 #define MPI2_LOG_0_ENTRY_QUAL_MAX_IMPLEMENT_SPEC (0xFFFF)

2673 typedef struct _MPI2_CONFIG_PAGE_LOG_0
2674 {
2675     MPI2_CONFIG_EXTENDED_PAGE_HEADER Header;        /* 0x00 */
2676     U32      Reserved1;                /* 0x08 */
2677     U32      Reserved2;                /* 0x0C */
2678     U16      NumLogEntries;            /* 0x10 */
2679     U16      Reserved3;                /* 0x12 */
2680     MPI2_LOG_0_ENTRY LogEntry[MPI2_LOG_0_NUM_LOG_ENTRIES]; /*
2681 } MPI2_CONFIG_PAGE_LOG_0, MPI2_POINTER PTR_MPI2_CONFIG_PAGE_LOG_0,
2682   Mpi2LogPage0_t, MPI2_POINTER pMpi2LogPage0_t;

2684 #define MPI2_LOG_0_PAGEVERSION      (0x02)

2687 /*****
2688 *   RAID Config Page
2689 *****/
2691 /* RAID Page 0 */

2693 /*
2694 * Host code (drivers, BIOS, utilities, etc.) should leave this define set to
2695 * one and check Header.ExtPageLength or NumPhys at runtime.
2696 */
2697 #ifndef MPI2_RAIDCONFIG0_MAX_ELEMENTS
2698 #define MPI2_RAIDCONFIG0_MAX_ELEMENTS      (1)
2699 #endif

2701 typedef struct _MPI2_RAIDCONFIG0_CONFIG_ELEMENT

```

```

2702 {
2703     U16      ElementFlags;              /* 0x00 */
2704     U16      VolDevHandle;              /* 0x02 */
2705     U8       HotSparePool;              /* 0x04 */
2706     U8       PhysDiskNum;              /* 0x05 */
2707     U16      PhysDiskDevHandle;         /* 0x06 */
2708 } MPI2_RAIDCONFIG0_CONFIG_ELEMENT,
2709 MPI2_POINTER PTR_MPI2_RAIDCONFIG0_CONFIG_ELEMENT,
2710   Mpi2RaidConfig0ConfigElement_t, MPI2_POINTER pMpi2RaidConfig0ConfigElement_t;

2712 /* values for the ElementFlags field */
2713 #define MPI2_RAIDCONFIG0_EFLAGS_MASK_ELEMENT_TYPE      (0x000F)
2714 #define MPI2_RAIDCONFIG0_EFLAGS_VOLUME_ELEMENT         (0x0000)
2715 #define MPI2_RAIDCONFIG0_EFLAGS_VOL_PHYS_DISK_ELEMENT (0x0001)
2716 #define MPI2_RAIDCONFIG0_EFLAGS_HOT_SPARE_ELEMENT     (0x0002)
2717 #define MPI2_RAIDCONFIG0_EFLAGS_OCE_ELEMENT           (0x0003)

2720 typedef struct _MPI2_CONFIG_PAGE_RAID_CONFIGURATION_0
2721 {
2722     MPI2_CONFIG_EXTENDED_PAGE_HEADER Header;        /* 0x00 */
2723     U8       NumHotSpares;                /* 0x08 */
2724     U8       NumPhysDisks;                /* 0x09 */
2725     U8       NumVolumes;                  /* 0x0A */
2726     U8       ConfigNum;                   /* 0x0B */
2727     U32      Flags;                        /* 0x0C */
2728     U8       ConfigGUID[24];              /* 0x10 */
2729     U32      Reserved1;                   /* 0x28 */
2730     U8       NumElements;                 /* 0x2C */
2731     U8       Reserved2;                   /* 0x2D */
2732     U16      Reserved3;                   /* 0x2E */
2733     MPI2_RAIDCONFIG0_CONFIG_ELEMENT ConfigElement[MPI2_RAIDCONFIG0_MAX_ELEMENTS];
2734 } MPI2_CONFIG_PAGE_RAID_CONFIGURATION_0,
2735 MPI2_POINTER PTR_MPI2_CONFIG_PAGE_RAID_CONFIGURATION_0,
2736   Mpi2RaidConfigurationPage0_t, MPI2_POINTER pMpi2RaidConfigurationPage0_t;

2738 #define MPI2_RAIDCONFIG0_PAGEVERSION      (0x00)

2740 /* values for RAID Configuration Page 0 Flags field */
2741 #define MPI2_RAIDCONFIG0_FLAG_FOREIGN_CONFIG      (0x00000001)

2744 /*****
2745 *   Driver Persistent Mapping Config Pages
2746 *****/

2748 /* Driver Persistent Mapping Page 0 */

2750 typedef struct _MPI2_CONFIG_PAGE_DRIVER_MAP0_ENTRY
2751 {
2752     U64      PhysicalIdentifier;          /* 0x00 */
2753     U16      MappingInformation;          /* 0x08 */
2754     U16      DeviceIndex;                 /* 0x0A */
2755     U32      PhysicalBitsMapping;         /* 0x0C */
2756     U32      Reserved1;                  /* 0x10 */
2757 } MPI2_CONFIG_PAGE_DRIVER_MAP0_ENTRY,
2758 MPI2_POINTER PTR_MPI2_CONFIG_PAGE_DRIVER_MAP0_ENTRY,
2759   Mpi2DriverMap0Entry_t, MPI2_POINTER pMpi2DriverMap0Entry_t;

2761 typedef struct _MPI2_CONFIG_PAGE_DRIVER_MAPPING_0
2762 {
2763     MPI2_CONFIG_EXTENDED_PAGE_HEADER Header;        /* 0x00 */
2764     MPI2_CONFIG_PAGE_DRIVER_MAP0_ENTRY Entry;        /* 0x08 */
2765 } MPI2_CONFIG_PAGE_DRIVER_MAPPING_0,
2766 MPI2_POINTER PTR_MPI2_CONFIG_PAGE_DRIVER_MAPPING_0,
2767   Mpi2DriverMappingPage0_t, MPI2_POINTER pMpi2DriverMappingPage0_t;

```

```

2769 #define MPI2_DRIVERMAPPING0_PAGEVERSION      (0x00)

2771 /* values for Driver Persistent Mapping Page 0 MappingInformation field */
2772 #define MPI2_DRVMAP0_MAPINFO_SLOT_MASK        (0x07F0)
2773 #define MPI2_DRVMAP0_MAPINFO_SLOT_SHIFT       (4)
2774 #define MPI2_DRVMAP0_MAPINFO_MISSING_MASK     (0x000F)

2777 /*****
2778 *   Ethernet Config Pages
2779 *****/

2781 /* Ethernet Page 0 */

2783 /* IP address (union of IPv4 and IPv6) */
2784 typedef union _MPI2_ETHERNET_IP_ADDR
2785 {
2786     U32      IPv4Addr;
2787     U32      IPv6Addr[4];
2788 } MPI2_ETHERNET_IP_ADDR, MPI2_POINTER PTR_MPI2_ETHERNET_IP_ADDR,
2789   Mpi2EthernetIpAddr_t, MPI2_POINTER pMpi2EthernetIpAddr_t;

2791 #define MPI2_ETHERNET_HOST_NAME_LENGTH         (32)

2793 typedef struct _MPI2_CONFIG_PAGE_ETHERNET_0
2794 {
2795     MPI2_CONFIG_EXTENDED_PAGE_HEADER  Header;          /* 0x00 */
2796     U8                                NumInterfaces;    /* 0x08 */
2797     U8                                Reserved0;         /* 0x09 */
2798     U16                               Reserved1;        /* 0x0A */
2799     U32                               Status;           /* 0x0C */
2800     U8                                MediaState;       /* 0x10 */
2801     U8                                Reserved2;        /* 0x11 */
2802     U16                               Reserved3;       /* 0x12 */
2803     U8                                MacAddress[6];    /* 0x14 */
2804     U8                                Reserved4;        /* 0x1A */
2805     U8                                Reserved5;        /* 0x1B */
2806     MPI2_ETHERNET_IP_ADDR             IpAddress;       /* 0x1C */
2807     MPI2_ETHERNET_IP_ADDR             SubnetMask;      /* 0x2C */
2808     MPI2_ETHERNET_IP_ADDR             GatewayIpAddress; /* 0x3C */
2809     MPI2_ETHERNET_IP_ADDR             DNS1IpAddress;  /* 0x4C */
2810     MPI2_ETHERNET_IP_ADDR             DNS2IpAddress;  /* 0x5C */
2811     MPI2_ETHERNET_IP_ADDR             DhcpIpAddress;  /* 0x6C */
2812     U8                                HostName[MPI2_ETHERNET_HOST_NAME_LENGTH]
2813 } MPI2_CONFIG_PAGE_ETHERNET_0, MPI2_POINTER PTR_MPI2_CONFIG_PAGE_ETHERNET_0,
2814   Mpi2EthernetPage0_t, MPI2_POINTER pMpi2EthernetPage0_t;

2816 #define MPI2_ETHERNETPAGE0_PAGEVERSION        (0x00)

2818 /* values for Ethernet Page 0 Status field */
2819 #define MPI2_ETHPG0_STATUS_IPV6_CAPABLE       (0x80000000)
2820 #define MPI2_ETHPG0_STATUS_IPV4_CAPABLE       (0x40000000)
2821 #define MPI2_ETHPG0_STATUS_CONSOLE_CONNECTED (0x20000000)
2822 #define MPI2_ETHPG0_STATUS_DEFAULT_IF        (0x00000100)
2823 #define MPI2_ETHPG0_STATUS_FW_DWNLD_ENABLED  (0x00000080)
2824 #define MPI2_ETHPG0_STATUS_TELNET_ENABLED     (0x00000040)
2825 #define MPI2_ETHPG0_STATUS_SSH2_ENABLED       (0x00000020)
2826 #define MPI2_ETHPG0_STATUS_DHCP_CLIENT_ENABLED (0x00000010)
2827 #define MPI2_ETHPG0_STATUS_IPV6_ENABLED       (0x00000008)
2828 #define MPI2_ETHPG0_STATUS_IPV4_ENABLED       (0x00000004)
2829 #define MPI2_ETHPG0_STATUS_IPV6_ADDRESSES     (0x00000002)
2830 #define MPI2_ETHPG0_STATUS_ETH_IF_ENABLED     (0x00000001)

2832 /* values for Ethernet Page 0 MediaState field */
2833 #define MPI2_ETHPG0_MS_DUPLEX_MASK            (0x80)

```

```

2834 #define MPI2_ETHPG0_MS_HALF_DUPLEX           (0x00)
2835 #define MPI2_ETHPG0_MS_FULL_DUPLEX           (0x80)

2837 #define MPI2_ETHPG0_MS_CONNECT_SPEED_MASK    (0x07)
2838 #define MPI2_ETHPG0_MS_NOT_CONNECTED         (0x00)
2839 #define MPI2_ETHPG0_MS_10MBIT                (0x01)
2840 #define MPI2_ETHPG0_MS_100MBIT              (0x02)
2841 #define MPI2_ETHPG0_MS_1GBIT                 (0x03)

2844 /* Ethernet Page 1 */

2846 typedef struct _MPI2_CONFIG_PAGE_ETHERNET_1
2847 {
2848     MPI2_CONFIG_EXTENDED_PAGE_HEADER  Header;          /* 0x00 */
2849     U32                                Reserved0;        /* 0x08 */
2850     U32                                Flags;            /* 0x0C */
2851     U8                                MediaState;       /* 0x10 */
2852     U8                                Reserved1;        /* 0x11 */
2853     U16                               Reserved2;        /* 0x12 */
2854     U8                                MacAddress[6];    /* 0x14 */
2855     U8                                Reserved3;        /* 0x1A */
2856     U8                                Reserved4;        /* 0x1B */
2857     MPI2_ETHERNET_IP_ADDR             StaticIpAddress;  /* 0x1C */
2858     MPI2_ETHERNET_IP_ADDR             StaticSubnetMask; /* 0x2C */
2859     MPI2_ETHERNET_IP_ADDR             StaticGatewayIpAddress; /* 0x3C */
2860     MPI2_ETHERNET_IP_ADDR             StaticDNS1IpAddress; /* 0x4C */
2861     MPI2_ETHERNET_IP_ADDR             StaticDNS2IpAddress; /* 0x5C */
2862     U32                                Reserved5;        /* 0x6C */
2863     U32                                Reserved6;        /* 0x70 */
2864     U32                                Reserved7;        /* 0x74 */
2865     U32                                Reserved8;        /* 0x78 */
2866     U8                                HostName[MPI2_ETHERNET_HOST_NAME_LENGTH]
2867 } MPI2_CONFIG_PAGE_ETHERNET_1, MPI2_POINTER PTR_MPI2_CONFIG_PAGE_ETHERNET_1,
2868   Mpi2EthernetPage1_t, MPI2_POINTER pMpi2EthernetPage1_t;

2870 #define MPI2_ETHERNETPAGE1_PAGEVERSION        (0x00)

2872 /* values for Ethernet Page 1 Flags field */
2873 #define MPI2_ETHPG1_FLAG_SET_DEFAULT_IF       (0x00000100)
2874 #define MPI2_ETHPG1_FLAG_ENABLE_FW_DOWNLOAD  (0x00000080)
2875 #define MPI2_ETHPG1_FLAG_ENABLE_TELNET       (0x00000040)
2876 #define MPI2_ETHPG1_FLAG_ENABLE_SSH2         (0x00000020)
2877 #define MPI2_ETHPG1_FLAG_ENABLE_DHCP_CLIENT  (0x00000010)
2878 #define MPI2_ETHPG1_FLAG_ENABLE_IPV6         (0x00000008)
2879 #define MPI2_ETHPG1_FLAG_ENABLE_IPV4         (0x00000004)
2880 #define MPI2_ETHPG1_FLAG_USE_IPV6_ADDRESSES  (0x00000002)
2881 #define MPI2_ETHPG1_FLAG_ENABLE_ETH_IF       (0x00000001)

2883 /* values for Ethernet Page 1 MediaState field */
2884 #define MPI2_ETHPG1_MS_DUPLEX_MASK            (0x80)
2885 #define MPI2_ETHPG1_MS_HALF_DUPLEX           (0x00)
2886 #define MPI2_ETHPG1_MS_FULL_DUPLEX           (0x80)

2888 #define MPI2_ETHPG1_MS_DATA_RATE_MASK        (0x07)
2889 #define MPI2_ETHPG1_MS_DATA_RATE_AUTO        (0x00)
2890 #define MPI2_ETHPG1_MS_DATA_RATE_10MBIT      (0x01)
2891 #define MPI2_ETHPG1_MS_DATA_RATE_100MBIT     (0x02)
2892 #define MPI2_ETHPG1_MS_DATA_RATE_1GBIT       (0x03)

2895 /*****
2896 *   Extended Manufacturing Config Pages
2897 *****/

2899 /*

```

```
2900 * Generic structure to use for product-specific extended manufacturing pages
2901 * (currently Extended Manufacturing Page 40 through Extended Manufacturing
2902 * Page 60).
2903 */

2905 typedef struct _MPI2_CONFIG_PAGE_EXT_MAN_PS
2906 {
2907     MPI2_CONFIG_EXTENDED_PAGE_HEADER    Header;          /* 0x00 */
2908     U32                                 ProductSpecificInfo; /* 0x08 */
2909 } MPI2_CONFIG_PAGE_EXT_MAN_PS,
2910 MPI2_POINTER PTR_MPI2_CONFIG_PAGE_EXT_MAN_PS,
2911 Mpi2ExtManufacturingPagePS_t, MPI2_POINTER pMpi2ExtManufacturingPagePS_t;

2913 /* PageVersion should be provided by product-specific code */

2915 #endif

2917 #endif /* ! codereview */
```

new/usr/src/uts/common/sys/scsi/adapters/mpt_sas3/mpi/mpi2_init.h

1

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

```
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */
21
22 /*
23  * Copyright (c) 2000-2012 LSI Corporation.
24  *
25  * Redistribution and use in source and binary forms of all code within
26  * this file that is exclusively owned by LSI, with or without
27  * modification, is permitted provided that, in addition to the CDDL 1.0
28  * License requirements, the following conditions are met:
29  *
30  * Neither the name of the author nor the names of its contributors may be
31  * used to endorse or promote products derived from this software without
32  * specific prior written permission.
33  *
34  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
35  * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
36  * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
37  * FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
38  * COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
39  * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
40  * BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
41  * OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
42  * AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
43  * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
44  * OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
45  * DAMAGE.
46 */
47
48 /*
49  * Name: mpi2_init.h
50  * Title: MPI SCSI initiator mode messages and structures
51  * Creation Date: June 23, 2006
52  *
53  * mpi2_init.h Version: 02.00.xx
54  *
55  * NOTE: Names (typedefs, defines, etc.) beginning with an MPI25 or Mpi25
56  * prefix are for use only on MPI v2.5 products, and must not be used
57  * with MPI v2.0 products. Unless otherwise noted, names beginning with
58  * MPI2 or Mpi2 are for use with both MPI v2.0 and MPI v2.5 products.
59  *
60  * Version History
61  * -----
```

new/usr/src/uts/common/sys/scsi/adapters/mpt_sas3/mpi/mpi2_init.h

2

```
62 *
63 * Date Version Description
64 * -----
65 * 04-30-07 02.00.00 Corresponds to Fusion-MPT MPI Specification Rev A.
66 * 10-31-07 02.00.01 Fixed name for pMpi2SCSITaskManagementRequest_t.
67 * 12-18-07 02.00.02 Modified Task Management Target Reset Method defines.
68 * 02-29-08 02.00.03 Added Query Task Set and Query Unit Attention.
69 * 03-03-08 02.00.04 Fixed name of struct _MPI2_SCSI_TASK_MANAGE_REPLY.
70 * 05-21-08 02.00.05 Fixed typo in name of Mpi2SepRequest_t.
71 * 10-02-08 02.00.06 Removed Untagged and No Disconnect values from SCSI IO
72 * Control field Task Attribute flags.
73 *
74 * Moved LUN field defines to mpi2.h because they are
75 * common to many structures.
76 *
77 * 05-06-09 02.00.07 Changed task management type of Query Unit Attention to
78 * Query Asynchronous Event.
79 *
80 * Defined two new bits in the SlotStatus field of the SCSI
81 * Enclosure Processor Request and Reply.
82 *
83 * 10-28-09 02.00.08 Added defines for decoding the ResponseInfo bytes for
84 * both SCSI IO Error Reply and SCSI Task Management Reply.
85 *
86 * Added ResponseInfo field to MPI2_SCSI_TASK_MANAGE_REPLY.
87 *
88 * Added MPI2_SCSITASKMGMT_RSP_TM_OVERLAPPED_TAG define.
89 *
90 * 02-10-10 02.00.09 Removed unused structure that had "#if 0" around it.
91 *
92 * 05-12-10 02.00.10 Added optional vendor-unique region to SCSI IO Request.
93 *
94 * 11-10-10 02.00.11 Added MPI2_SCSIIO_NUM_SGLOFFSETS define.
95 *
96 * -----
97 */
98
99 #ifndef MPI2_INIT_H
100 #define MPI2_INIT_H
101
102 /*****
103  *
104  * SCSI Initiator Messages
105  *
106  *****/
107
108 /*****
109  * SCSI IO messages and associated structures
110  *****/
111
112 typedef struct
113 {
114     U8 CDB[20]; /* 0x00 */
115     U32 PrimaryReferenceTag; /* 0x14 */
116     U16 PrimaryApplicationTag; /* 0x18 */
117     U16 PrimaryApplicationTagMask; /* 0x1A */
118     U32 TransferLength; /* 0x1C */
119 } MPI2_SCSI_IO_CDB_EEDP32, MPI2_POINTER PTR_MPI2_SCSI_IO_CDB_EEDP32,
120 Mpi2ScsiIoCdbEedp32_t, MPI2_POINTER pMpi2ScsiIoCdbEedp32_t;
121
122 typedef union
123 {
124     U8 CDB32[32];
125     MPI2_SCSI_IO_CDB_EEDP32 EEDP32;
126     MPI2_SGE_SIMPLE_UNION SGE;
127 } MPI2_SCSI_IO_CDB_UNION, MPI2_POINTER PTR_MPI2_SCSI_IO_CDB_UNION,
128 Mpi2ScsiIoCdb_t, MPI2_POINTER pMpi2ScsiIoCdb_t;
129
130 /* MPI v2.0 SCSI IO Request Message */
131 typedef struct _MPI2_SCSI_IO_REQUEST
132 {
133     U16 DevHandle; /* 0x00 */
134     U8 ChainOffset; /* 0x02 */
135     U8 Function; /* 0x03 */
136     U16 Reserved1; /* 0x04 */
137     U8 Reserved2; /* 0x06 */
138 }
```



```

128     U8             MsgFlags;           /* 0x07 */
129     U8             VP_ID;              /* 0x08 */
130     U8             VF_ID;              /* 0x09 */
131     U16            Reserved3;          /* 0x0A */
132     U32            SenseBufferLowAddress; /* 0x0C */
133     U16            SGLFlags;           /* 0x10 */
134     U8             SenseBufferLength;  /* 0x12 */
135     U8             Reserved4;          /* 0x13 */
136     U8             SGLOffset0;         /* 0x14 */
137     U8             SGLOffset1;         /* 0x15 */
138     U8             SGLOffset2;         /* 0x16 */
139     U8             SGLOffset3;         /* 0x17 */
140     U32            SkipCount;          /* 0x18 */
141     U32            DataLength;         /* 0x1C */
142     U32            BidirectionalDataLength; /* 0x20 */
143     U16            IoFlags;            /* 0x24 */
144     U16            EEDPFlags;          /* 0x26 */
145     U32            EEDPBlockSize;      /* 0x28 */
146     U32            SecondaryReferenceTag; /* 0x2C */
147     U16            SecondaryApplicationTag; /* 0x30 */
148     U16            ApplicationTagTranslationMask; /* 0x32 */
149     U8             LUN[8];             /* 0x34 */
150     U32            Control;            /* 0x3C */
151     MPI2_SCSI_IO_CDB_UNION CDB;        /* 0x40 */
152     MPI2_SGE_IO_UNION SGL;            /* 0x60 */
153 } MPI2_SCSI_IO_REQUEST, MPI2_POINTER PTR_MPI2_SCSI_IO_REQUEST,
154   Mpi2SCSIIORequest_t, MPI2_POINTER pMpi2SCSIIORequest_t;

156 /* SCSI IO MsgFlags bits */

158 /* MsgFlags for SenseBufferAddressSpace */
159 #define MPI2_SCSIIO_MSGFLAGS_MASK_SENSE_ADDR (0x0C)
160 #define MPI2_SCSIIO_MSGFLAGS_SYSTEM_SENSE_ADDR (0x00)
161 #define MPI2_SCSIIO_MSGFLAGS_IOCDDR_SENSE_ADDR (0x04)
162 #define MPI2_SCSIIO_MSGFLAGS_IOCPLB_SENSE_ADDR (0x08)
163 #define MPI2_SCSIIO_MSGFLAGS_IOCPLBNTA_SENSE_ADDR (0x0C)

165 /* SCSI IO SGLFlags bits */

167 /* base values for Data Location Address Space */
168 #define MPI2_SCSIIO_SGLFLAGS_ADDR_MASK (0x0C)
169 #define MPI2_SCSIIO_SGLFLAGS_SYSTEM_ADDR (0x00)
170 #define MPI2_SCSIIO_SGLFLAGS_IOCDDR_ADDR (0x04)
171 #define MPI2_SCSIIO_SGLFLAGS_IOCPLB_ADDR (0x08)
172 #define MPI2_SCSIIO_SGLFLAGS_IOCPLBNTA_ADDR (0x0C)

174 /* base values for Type */
175 #define MPI2_SCSIIO_SGLFLAGS_TYPE_MASK (0x03)
176 #define MPI2_SCSIIO_SGLFLAGS_TYPE_MPI (0x00)
177 #define MPI2_SCSIIO_SGLFLAGS_TYPE_IEEE32 (0x01)
178 #define MPI2_SCSIIO_SGLFLAGS_TYPE_IEEE64 (0x02)

180 /* shift values for each sub-field */
181 #define MPI2_SCSIIO_SGLFLAGS_SGL3_SHIFT (12)
182 #define MPI2_SCSIIO_SGLFLAGS_SGL2_SHIFT (8)
183 #define MPI2_SCSIIO_SGLFLAGS_SGL1_SHIFT (4)
184 #define MPI2_SCSIIO_SGLFLAGS_SGL0_SHIFT (0)

186 /* number of SGLOffset fields */
187 #define MPI2_SCSIIO_NUM_SGLOFFSETS (4)

189 /* SCSI IO IoFlags bits */

191 /* Large CDB Address Space */
192 #define MPI2_SCSIIO_CDB_ADDR_MASK (0x6000)
193 #define MPI2_SCSIIO_CDB_ADDR_SYSTEM (0x0000)

```

```

194 #define MPI2_SCSIIO_CDB_ADDR_IOCDDR (0x2000)
195 #define MPI2_SCSIIO_CDB_ADDR_IOCPLB (0x4000)
196 #define MPI2_SCSIIO_CDB_ADDR_IOCPLBNTA (0x6000)

198 #define MPI2_SCSIIO_IOFLAGS_LARGE_CDB (0x1000)
199 #define MPI2_SCSIIO_IOFLAGS_BIDIRECTIONAL (0x0800)
200 #define MPI2_SCSIIO_IOFLAGS_MULTICAST (0x0400)
201 #define MPI2_SCSIIO_IOFLAGS_CMD_DETERMINES_DATA_DIR (0x0200)
202 #define MPI2_SCSIIO_IOFLAGS_CDBLENGTH_MASK (0x01FF)

204 /* SCSI IO EEDPFlags bits */

206 #define MPI2_SCSIIO_EEDPFLAGS_INC_PRI_REFTAG (0x8000)
207 #define MPI2_SCSIIO_EEDPFLAGS_INC_SEC_REFTAG (0x4000)
208 #define MPI2_SCSIIO_EEDPFLAGS_INC_PRI_APPTAG (0x2000)
209 #define MPI2_SCSIIO_EEDPFLAGS_INC_SEC_APPTAG (0x1000)

211 #define MPI2_SCSIIO_EEDPFLAGS_CHECK_REFTAG (0x0400)
212 #define MPI2_SCSIIO_EEDPFLAGS_CHECK_APPTAG (0x0200)
213 #define MPI2_SCSIIO_EEDPFLAGS_CHECK_GUARD (0x0100)

215 #define MPI2_SCSIIO_EEDPFLAGS_PASSTHRU_REFTAG (0x0008)

217 #define MPI2_SCSIIO_EEDPFLAGS_MASK_OP (0x0007)
218 #define MPI2_SCSIIO_EEDPFLAGS_NOOP_OP (0x0000)
219 #define MPI2_SCSIIO_EEDPFLAGS_CHECK_OP (0x0001)
220 #define MPI2_SCSIIO_EEDPFLAGS_STRIP_OP (0x0002)
221 #define MPI2_SCSIIO_EEDPFLAGS_CHECK_REMOVE_OP (0x0003)
222 #define MPI2_SCSIIO_EEDPFLAGS_INSERT_OP (0x0004)
223 #define MPI2_SCSIIO_EEDPFLAGS_REPLACE_OP (0x0006)
224 #define MPI2_SCSIIO_EEDPFLAGS_CHECK_REGEN_OP (0x0007)

226 /* SCSI IO LUN fields: use MPI2_LUN from mpi2.h */

228 /* SCSI IO Control bits */
229 #define MPI2_SCSIIO_CONTROL_ADDCDBLEN_MASK (0xFC000000)
230 #define MPI2_SCSIIO_CONTROL_ADDCDBLEN_SHIFT (26)

232 #define MPI2_SCSIIO_CONTROL_DATADIRECTION_MASK (0x03000000)
233 #define MPI2_SCSIIO_CONTROL_NODATATRANSFER (0x00000000)
234 #define MPI2_SCSIIO_CONTROL_WRITE (0x01000000)
235 #define MPI2_SCSIIO_CONTROL_READ (0x02000000)
236 #define MPI2_SCSIIO_CONTROL_BIDIRECTIONAL (0x03000000)

238 #define MPI2_SCSIIO_CONTROL_TASKPRI_MASK (0x00007800)
239 #define MPI2_SCSIIO_CONTROL_TASKPRI_SHIFT (11)

241 #define MPI2_SCSIIO_CONTROL_TASKATTRIBUTE_MASK (0x00000700)
242 #define MPI2_SCSIIO_CONTROL_SIMPLEQ (0x00000000)
243 #define MPI2_SCSIIO_CONTROL_HEADOFQ (0x00000100)
244 #define MPI2_SCSIIO_CONTROL_ORDEREDQ (0x00000200)
245 #define MPI2_SCSIIO_CONTROL_ACAQ (0x00000400)

247 #define MPI2_SCSIIO_CONTROL_TLR_MASK (0x000000C0)
248 #define MPI2_SCSIIO_CONTROL_NO_TLR (0x00000000)
249 #define MPI2_SCSIIO_CONTROL_TLR_ON (0x00000040)
250 #define MPI2_SCSIIO_CONTROL_TLR_OFF (0x00000080)

253 /* MPI v2.5 CDB field */
254 typedef union _MPI25_SCSI_IO_CDB_UNION
255 {
256     U8 CDB32[32];
257     MPI2_SCSI_IO_CDB_EEDP32 EEDP32;
258     MPI2_IEEE_SGE_SIMPLE64 SGE;
259 } MPI25_SCSI_IO_CDB_UNION, MPI2_POINTER PTR_MPI25_SCSI_IO_CDB_UNION,

```

```

260  Mpi25ScsiIoCdb_t, MPI2_POINTER pMpi25ScsiIoCdb_t;

262 /* MPI v2.5 SCSI IO Request Message */
263 typedef struct _MPI25_SCSI_IO_REQUEST
264 {
265     U16      DevHandle;          /* 0x00 */
266     U8       ChainOffset;        /* 0x02 */
267     U8       Function;           /* 0x03 */
268     U16      Reserved1;          /* 0x04 */
269     U8       Reserved2;          /* 0x06 */
270     U8       MsgFlags;           /* 0x07 */
271     U8       VP_ID;              /* 0x08 */
272     U8       VF_ID;              /* 0x09 */
273     U16      Reserved3;          /* 0x0A */
274     U32      SenseBufferLowAddress; /* 0x0C */
275     U8       DMAFlags;           /* 0x10 */
276     U8       Reserved5;          /* 0x11 */
277     U8       SenseBufferLength;   /* 0x12 */
278     U8       Reserved4;          /* 0x13 */
279     U8       SGLOffset0;         /* 0x14 */
280     U8       SGLOffset1;         /* 0x15 */
281     U8       SGLOffset2;         /* 0x16 */
282     U8       SGLOffset3;         /* 0x17 */
283     U32      SkipCount;          /* 0x18 */
284     U32      DataLength;         /* 0x1C */
285     U32      BidirectionalDataLength; /* 0x20 */
286     U16      IoFlags;            /* 0x24 */
287     U16      EEDPFlags;          /* 0x26 */
288     U16      EEDPBlockSize;      /* 0x28 */
289     U16      Reserved6;          /* 0x2A */
290     U32      SecondaryReferenceTag; /* 0x2C */
291     U16      SecondaryApplicationTag; /* 0x30 */
292     U16      ApplicationTagTranslationMask; /* 0x32 */
293     U8       LUN[8];             /* 0x34 */
294     U32      Control;            /* 0x3C */
295     MPI25_SCSI_IO_CDB_UNION CDB; /* 0x40 */

297 #ifdef MPI25_SCSI_IO_VENDOR_UNIQUE_REGION /* typically this is left undefined */
298     MPI25_SCSI_IO_VENDOR_UNIQUE VendorRegion;
299 #endif

301     MPI25_SGE_IO_UNION      SGL; /* 0x60 */

303 } MPI25_SCSI_IO_REQUEST, MPI2_POINTER PTR_MPI25_SCSI_IO_REQUEST,
304   Mpi25SCSIIORequest_t, MPI2_POINTER pMpi25SCSIIORequest_t;

306 /* use MPI2_SCSIIO_MSGFLAGS_ defines for the MsgFlags field */

308 /* Defines for the DMAFlags field
309  * Each setting affects 4 SGLs, from SGL0 to SGL3.
310  *   D = Data
311  *   C = Cache DIF
312  *   I = Interleaved
313  *   H = Host DIF
314  */
315 #define MPI25_SCSIIO_DMAFLAGS_OP_MASK (0x0F)
316 #define MPI25_SCSIIO_DMAFLAGS_OP_D_D_D (0x00)
317 #define MPI25_SCSIIO_DMAFLAGS_OP_D_D_D_C (0x01)
318 #define MPI25_SCSIIO_DMAFLAGS_OP_D_D_D_I (0x02)
319 #define MPI25_SCSIIO_DMAFLAGS_OP_D_D_C_C (0x03)
320 #define MPI25_SCSIIO_DMAFLAGS_OP_D_D_C_I (0x04)
321 #define MPI25_SCSIIO_DMAFLAGS_OP_D_D_I_I (0x05)
322 #define MPI25_SCSIIO_DMAFLAGS_OP_D_C_C_C (0x06)
323 #define MPI25_SCSIIO_DMAFLAGS_OP_D_C_C_I (0x07)
324 #define MPI25_SCSIIO_DMAFLAGS_OP_D_C_I_I (0x08)
325 #define MPI25_SCSIIO_DMAFLAGS_OP_D_I_I_I (0x09)

```

```

326 #define MPI25_SCSIIO_DMAFLAGS_OP_D_H_D_D (0x0A)
327 #define MPI25_SCSIIO_DMAFLAGS_OP_D_H_D_C (0x0B)
328 #define MPI25_SCSIIO_DMAFLAGS_OP_D_H_D_I (0x0C)
329 #define MPI25_SCSIIO_DMAFLAGS_OP_D_H_C_C (0x0D)
330 #define MPI25_SCSIIO_DMAFLAGS_OP_D_H_C_I (0x0E)
331 #define MPI25_SCSIIO_DMAFLAGS_OP_D_H_I_I (0x0F)

333 /* number of SGLOffset fields */
334 #define MPI25_SCSIIO_NUM_SGLOFFSETS (4)

336 /* defines for the IoFlags field */
337 #define MPI25_SCSIIO_IOFLAGS_IO_PATH_MASK (0x0000)
338 #define MPI25_SCSIIO_IOFLAGS_NORMAL_PATH (0x0000)
339 #define MPI25_SCSIIO_IOFLAGS_FAST_PATH (0x4000)

341 #define MPI25_SCSIIO_IOFLAGS_LARGE_CDB (0x1000)
342 #define MPI25_SCSIIO_IOFLAGS_BIDIRECTIONAL (0x0800)
343 #define MPI25_SCSIIO_IOFLAGS_CMD_DETERMINES_DATA_DIR (0x0200)
344 #define MPI25_SCSIIO_IOFLAGS_CDBLENGTH_MASK (0x01FF)

346 /* MPI v2.5 defines for the EEDPFlags bits */
347 /* use MPI2_SCSIIO_EEDPFLAGS_ defines for the other EEDPFlags bits */
348 #define MPI25_SCSIIO_EEDPFLAGS_ESCAPE_MODE_MASK (0x00C0)
349 #define MPI25_SCSIIO_EEDPFLAGS_COMPATIBLE_MODE (0x0000)
350 #define MPI25_SCSIIO_EEDPFLAGS_DO_NOT_DISABLE_MODE (0x0040)
351 #define MPI25_SCSIIO_EEDPFLAGS_APPTAG_DISABLE_MODE (0x0080)
352 #define MPI25_SCSIIO_EEDPFLAGS_APPTAG_REFTAG_DISABLE_MODE (0x00C0)

354 #define MPI25_SCSIIO_EEDPFLAGS_HOST_GUARD_METHOD_MASK (0x0030)
355 #define MPI25_SCSIIO_EEDPFLAGS_T10_CRC_HOST_GUARD (0x0000)
356 #define MPI25_SCSIIO_EEDPFLAGS_IP_CHKSUM_HOST_GUARD (0x0010)

358 /* use MPI2_LUN_ defines from mpi2.h for the LUN field */

360 /* use MPI2_SCSIIO_CONTROL_ defines for the Control field */

363 /* NOTE: The SCSI IO Reply is the same for MPI 2.0 and MPI 2.5, so
364  * MPI2_SCSI_IO_REPLY is used for both.
365  */

367 /* SCSI IO Error Reply Message */
368 typedef struct _MPI2_SCSI_IO_REPLY
369 {
370     U16      DevHandle;          /* 0x00 */
371     U8       MsgLength;          /* 0x02 */
372     U8       Function;           /* 0x03 */
373     U16      Reserved1;          /* 0x04 */
374     U8       Reserved2;          /* 0x06 */
375     U8       MsgFlags;           /* 0x07 */
376     U8       VP_ID;              /* 0x08 */
377     U8       VF_ID;              /* 0x09 */
378     U16      Reserved3;          /* 0x0A */
379     U8       SCSIStatus;         /* 0x0C */
380     U8       SCSIState;          /* 0x0D */
381     U16      IOCStatus;          /* 0x0E */
382     U32      IOCLogInfo;         /* 0x10 */
383     U32      TransferCount;      /* 0x14 */
384     U32      SenseCount;         /* 0x18 */
385     U32      ResponseInfo;       /* 0x1C */
386     U16      TaskTag;            /* 0x20 */
387     U16      Reserved4;          /* 0x22 */
388     U32      BidirectionalTransferCount; /* 0x24 */
389     U32      Reserved5;          /* 0x28 */
390     U32      Reserved6;          /* 0x2C */
391 } MPI2_SCSI_IO_REPLY, MPI2_POINTER PTR_MPI2_SCSI_IO_REPLY,

```

```

392  Mpi2SCSIIOReply_t, MPI2_POINTER pMpi2SCSIIOReply_t;

394 /* SCSI IO Reply SCSIStatus values (SAM-4 status codes) */

396 #define MPI2_SCSI_STATUS_GOOD                (0x00)
397 #define MPI2_SCSI_STATUS_CHECK_CONDITION      (0x02)
398 #define MPI2_SCSI_STATUS_CONDITION_MET        (0x04)
399 #define MPI2_SCSI_STATUS_BUSY                 (0x08)
400 #define MPI2_SCSI_STATUS_INTERMEDIATE         (0x10)
401 #define MPI2_SCSI_STATUS_INTERMEDIATE_CONDMET (0x14)
402 #define MPI2_SCSI_STATUS_RESERVATION_CONFLICT (0x18)
403 #define MPI2_SCSI_STATUS_COMMAND_TERMINATED   (0x22) /* obsolete */
404 #define MPI2_SCSI_STATUS_TASK_SET_FULL        (0x28)
405 #define MPI2_SCSI_STATUS_ACA_ACTIVE            (0x30)
406 #define MPI2_SCSI_STATUS_TASK_ABORTED         (0x40)

408 /* SCSI IO Reply SCSIState flags */

410 #define MPI2_SCSI_STATE_RESPONSE_INFO_VALID   (0x10)
411 #define MPI2_SCSI_STATE_TERMINATED            (0x08)
412 #define MPI2_SCSI_STATE_NO_SCSI_STATUS        (0x04)
413 #define MPI2_SCSI_STATE_AUTONSENSE_FAILED     (0x02)
414 #define MPI2_SCSI_STATE_AUTONSENSE_VALID      (0x01)

416 /* masks and shifts for the ResponseInfo field */

418 #define MPI2_SCSI_RI_MASK_REASONCODE          (0x000000FF)
419 #define MPI2_SCSI_RI_SHIFT_REASONCODE         (0)

421 #define MPI2_SCSI_TASKTAG_UNKNOWN              (0xFFFF)

424 /*****
425  * SCSI Task Management messages
426  *****/

428 /* SCSI Task Management Request Message */
429 typedef struct _MPI2_SCSI_TASK_MANAGE_REQUEST
430 {
431     U16          DevHandle;                /* 0x00 */
432     U8           ChainOffset;               /* 0x02 */
433     U8           Function;                  /* 0x03 */
434     U8           Reserved1;                 /* 0x04 */
435     U8           TaskType;                  /* 0x05 */
436     U8           Reserved2;                 /* 0x06 */
437     U8           MsgFlags;                  /* 0x07 */
438     U8           VP_ID;                     /* 0x08 */
439     U8           VF_ID;                     /* 0x09 */
440     U16          Reserved3;                 /* 0x0A */
441     U8           LUN[8];                    /* 0x0C */
442     U32          Reserved4[7];              /* 0x14 */
443     U16          TaskMID;                   /* 0x30 */
444     U16          Reserved5;                 /* 0x32 */
445 } MPI2_SCSI_TASK_MANAGE_REQUEST,
446 MPI2_POINTER PTR_MPI2_SCSI_TASK_MANAGE_REQUEST,
447 Mpi2SCSITaskManagementRequest_t,
448 MPI2_POINTER pMpi2SCSITaskManagementRequest_t;

450 /* TaskType values */

452 #define MPI2_SCSITASKMGMT_TASKTYPE_ABORT_TASK (0x01)
453 #define MPI2_SCSITASKMGMT_TASKTYPE_ABORT_TASK_SET (0x02)
454 #define MPI2_SCSITASKMGMT_TASKTYPE_TARGET_RESET (0x03)
455 #define MPI2_SCSITASKMGMT_TASKTYPE_LOGICAL_UNIT_RESET (0x05)
456 #define MPI2_SCSITASKMGMT_TASKTYPE_CLEAR_TASK_SET (0x06)
457 #define MPI2_SCSITASKMGMT_TASKTYPE_QUERY_TASK (0x07)

```

```

458 #define MPI2_SCSITASKMGMT_TASKTYPE_CLR_ACA      (0x08)
459 #define MPI2_SCSITASKMGMT_TASKTYPE_QRY_TASK_SET (0x09)
460 #define MPI2_SCSITASKMGMT_TASKTYPE_QRY_ASYNC_EVENT (0x0A)

462 /* obsolete TaskType name */
463 #define MPI2_SCSITASKMGMT_TASKTYPE_QRY_UNIT_ATTENTION (MPI2_SCSITASKMGMT_TASKTYPE_QRY_ASYNC_EVENT)

465 /* MsgFlags bits */

467 #define MPI2_SCSITASKMGMT_MSGFLAGS_MASK_TARGET_RESET (0x18)
468 #define MPI2_SCSITASKMGMT_MSGFLAGS_LINK_RESET        (0x00)
469 #define MPI2_SCSITASKMGMT_MSGFLAGS_NEXUS_RESET_SRST  (0x08)
470 #define MPI2_SCSITASKMGMT_MSGFLAGS_SAS_HARD_LINK_RESET (0x10)

472 #define MPI2_SCSITASKMGMT_MSGFLAGS_DO_NOT_SEND_TASK_IU (0x01)

476 /* SCSI Task Management Reply Message */
477 typedef struct _MPI2_SCSI_TASK_MANAGE_REPLY
478 {
479     U16          DevHandle;                /* 0x00 */
480     U8           MsgLength;                 /* 0x02 */
481     U8           Function;                  /* 0x03 */
482     U8           ResponseCode;              /* 0x04 */
483     U8           TaskType;                  /* 0x05 */
484     U8           Reserved1;                 /* 0x06 */
485     U8           MsgFlags;                  /* 0x07 */
486     U8           VP_ID;                     /* 0x08 */
487     U8           VF_ID;                     /* 0x09 */
488     U16          Reserved2;                 /* 0x0A */
489     U16          Reserved3;                 /* 0x0C */
490     U16          IOCStatus;                 /* 0x0E */
491     U32          IOCLogInfo;                /* 0x10 */
492     U32          TerminationCount;          /* 0x14 */
493     U32          ResponseInfo;              /* 0x18 */
494 } MPI2_SCSI_TASK_MANAGE_REPLY,
495 MPI2_POINTER PTR_MPI2_SCSI_TASK_MANAGE_REPLY,
496 Mpi2SCSITaskManagementReply_t, MPI2_POINTER pMpi2SCSITaskManagementReply_t;

498 /* ResponseCode values */

500 #define MPI2_SCSITASKMGMT_RSP_TM_COMPLETE        (0x00)
501 #define MPI2_SCSITASKMGMT_RSP_INVALID_FRAME      (0x02)
502 #define MPI2_SCSITASKMGMT_RSP_TM_NOT_SUPPORTED   (0x04)
503 #define MPI2_SCSITASKMGMT_RSP_TM_FAILED          (0x05)
504 #define MPI2_SCSITASKMGMT_RSP_TM_SUCCEEDED      (0x08)
505 #define MPI2_SCSITASKMGMT_RSP_TM_INVALID_LUN     (0x09)
506 #define MPI2_SCSITASKMGMT_RSP_TM_OVERLAPPED_TAG (0x0A)
507 #define MPI2_SCSITASKMGMT_RSP_IO_QUEUED_ON_IOC   (0x80)

509 /* masks and shifts for the ResponseInfo field */

511 #define MPI2_SCSITASKMGMT_RI_MASK_REASONCODE      (0x000000FF)
512 #define MPI2_SCSITASKMGMT_RI_SHIFT_REASONCODE     (0)
513 #define MPI2_SCSITASKMGMT_RI_MASK_ARI2           (0x0000FF00)
514 #define MPI2_SCSITASKMGMT_RI_SHIFT_ARI2          (8)
515 #define MPI2_SCSITASKMGMT_RI_MASK_ARI1           (0x00FF0000)
516 #define MPI2_SCSITASKMGMT_RI_SHIFT_ARI1          (16)
517 #define MPI2_SCSITASKMGMT_RI_MASK_ARI0           (0xFF000000)
518 #define MPI2_SCSITASKMGMT_RI_SHIFT_ARI0          (24)

521 /*****
522  * SCSI Enclosure Processor messages
523  *****/

```

```

525 /* SCSI Enclosure Processor Request Message */
526 typedef struct _MPI2_SEP_REQUEST
527 {
528     U16      DevHandle;          /* 0x00 */
529     U8       ChainOffset;        /* 0x02 */
530     U8       Function;           /* 0x03 */
531     U8       Action;             /* 0x04 */
532     U8       Flags;              /* 0x05 */
533     U8       Reserved1;          /* 0x06 */
534     U8       MsgFlags;           /* 0x07 */
535     U8       VP_ID;              /* 0x08 */
536     U8       VF_ID;              /* 0x09 */
537     U16      Reserved2;          /* 0x0A */
538     U32      SlotStatus;         /* 0x0C */
539     U32      Reserved3;          /* 0x10 */
540     U32      Reserved4;          /* 0x14 */
541     U32      Reserved5;          /* 0x18 */
542     U16      Slot;               /* 0x1C */
543     U16      EnclosureHandle;    /* 0x1E */
544 } MPI2_SEP_REQUEST, MPI2_POINTER PTR_MPI2_SEP_REQUEST,
545   Mpi2SepRequest_t, MPI2_POINTER pMpi2SepRequest_t;

547 /* Action defines */
548 #define MPI2_SEP_REQ_ACTION_WRITE_STATUS      (0x00)
549 #define MPI2_SEP_REQ_ACTION_READ_STATUS       (0x01)

551 /* Flags defines */
552 #define MPI2_SEP_REQ_FLAGS_DEVHANDLE_ADDRESS  (0x00)
553 #define MPI2_SEP_REQ_FLAGS_ENCLOSURE_SLOT_ADDRESS (0x01)

555 /* SlotStatus defines */
556 #define MPI2_SEP_REQ_SLOTSTATUS_REQUEST_REMOVE      (0x00040000)
557 #define MPI2_SEP_REQ_SLOTSTATUS_IDENTIFY_REQUEST    (0x00020000)
558 #define MPI2_SEP_REQ_SLOTSTATUS_REBUILD_STOPPED     (0x00000200)
559 #define MPI2_SEP_REQ_SLOTSTATUS_HOT_SPARE            (0x00000100)
560 #define MPI2_SEP_REQ_SLOTSTATUS_UNCONFIGURED         (0x00000080)
561 #define MPI2_SEP_REQ_SLOTSTATUS_PREDICTED_FAULT      (0x00000040)
562 #define MPI2_SEP_REQ_SLOTSTATUS_IN_CRITICAL_ARRAY    (0x00000010)
563 #define MPI2_SEP_REQ_SLOTSTATUS_IN_FAILED_ARRAY      (0x00000008)
564 #define MPI2_SEP_REQ_SLOTSTATUS_DEV_REBUILDING       (0x00000004)
565 #define MPI2_SEP_REQ_SLOTSTATUS_DEV_FAULTY           (0x00000002)
566 #define MPI2_SEP_REQ_SLOTSTATUS_NO_ERROR             (0x00000001)

569 /* SCSI Enclosure Processor Reply Message */
570 typedef struct _MPI2_SEP_REPLY
571 {
572     U16      DevHandle;          /* 0x00 */
573     U8       MsgLength;          /* 0x02 */
574     U8       Function;           /* 0x03 */
575     U8       Action;             /* 0x04 */
576     U8       Flags;              /* 0x05 */
577     U8       Reserved1;          /* 0x06 */
578     U8       MsgFlags;           /* 0x07 */
579     U8       VP_ID;              /* 0x08 */
580     U8       VF_ID;              /* 0x09 */
581     U16      Reserved2;          /* 0x0A */
582     U16      Reserved3;          /* 0x0C */
583     U16      IOCStatus;          /* 0x0E */
584     U32      IOCLogInfo;         /* 0x10 */
585     U32      SlotStatus;         /* 0x14 */
586     U32      Reserved4;          /* 0x18 */
587     U16      Slot;               /* 0x1C */
588     U16      EnclosureHandle;    /* 0x1E */
589 } MPI2_SEP_REPLY, MPI2_POINTER PTR_MPI2_SEP_REPLY,

```

```

590   Mpi2SepReply_t, MPI2_POINTER pMpi2SepReply_t;

592 /* SlotStatus defines */
593 #define MPI2_SEP_REPLY_SLOTSTATUS_REMOVE_READY      (0x00040000)
594 #define MPI2_SEP_REPLY_SLOTSTATUS_IDENTIFY_REQUEST  (0x00020000)
595 #define MPI2_SEP_REPLY_SLOTSTATUS_REBUILD_STOPPED   (0x00000200)
596 #define MPI2_SEP_REPLY_SLOTSTATUS_HOT_SPARE         (0x00000100)
597 #define MPI2_SEP_REPLY_SLOTSTATUS_UNCONFIGURED      (0x00000080)
598 #define MPI2_SEP_REPLY_SLOTSTATUS_PREDICTED_FAULT   (0x00000040)
599 #define MPI2_SEP_REPLY_SLOTSTATUS_IN_CRITICAL_ARRAY  (0x00000010)
600 #define MPI2_SEP_REPLY_SLOTSTATUS_IN_FAILED_ARRAY    (0x00000008)
601 #define MPI2_SEP_REPLY_SLOTSTATUS_DEV_REBUILDING     (0x00000004)
602 #define MPI2_SEP_REPLY_SLOTSTATUS_DEV_FAULTY         (0x00000002)
603 #define MPI2_SEP_REPLY_SLOTSTATUS_NO_ERROR           (0x00000001)

606 #endif

609 #endif /* ! codereview */

```

new/usr/src/uts/common/sys/scsi/adapters/mpt_sas3/mpi/mpi2_ioc.h

1

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

```
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */
21
22 /*
23  * Copyright (c) 2000-2012 LSI Corporation.
24  *
25  * Redistribution and use in source and binary forms of all code within
26  * this file that is exclusively owned by LSI, with or without
27  * modification, is permitted provided that, in addition to the CDDL 1.0
28  * License requirements, the following conditions are met:
29  *
30  * Neither the name of the author nor the names of its contributors may be
31  * used to endorse or promote products derived from this software without
32  * specific prior written permission.
33  *
34  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
35  * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
36  * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
37  * FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
38  * COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
39  * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
40  * BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
41  * OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
42  * AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
43  * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
44  * OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
45  * DAMAGE.
46 */
47
48 /*
49  * Name: mpi2_ioc.h
50  * Title: MPI IOC, Port, Event, FW Download, and FW Upload messages
51  * Creation Date: October 11, 2006
52  *
53  * mpi2_ioc.h Version: 02.00.xx
54  *
55  * NOTE: Names (typedefs, defines, etc.) beginning with an MPI25 or Mpi25
56  * prefix are for use only on MPI v2.5 products, and must not be used
57  * with MPI v2.0 products. Unless otherwise noted, names beginning with
58  * MPI2 or Mpi2 are for use with both MPI v2.0 and MPI v2.5 products.
59  *
60  * Version History
61  * -----
```

new/usr/src/uts/common/sys/scsi/adapters/mpt_sas3/mpi/mpi2_ioc.h

2

```
62 *
63 * Date Version Description
64 * -----
65 * 04-30-07 02.00.00 Corresponds to Fusion-MPT MPI Specification Rev A.
66 * 06-04-07 02.00.01 In IOCFacts Reply structure, renamed MaxDevices to
67 * MaxTargets.
68 * Added TotalImageSize field to FWDnload Request.
69 * Added reserved words to FWUpld Request.
70 * 06-26-07 02.00.02 Added IR Configuration Change List Event.
71 * 08-31-07 02.00.03 Removed SystemReplyQueueDepth field from the IOCInit
72 * request and replaced it with
73 * ReplyDescriptorPostQueueDepth and ReplyFreeQueueDepth.
74 * Replaced the MinReplyQueueDepth field of the IOCFacts
75 * reply with MaxReplyDescriptorPostQueueDepth.
76 * Added MPI2_RDPQ_DEPTH_MIN define to specify the minimum
77 * depth for the Reply Descriptor Post Queue.
78 * Added SASAddress field to Initiator Device Table
79 * Overflow Event data.
80 * 10-31-07 02.00.04 Added ReasonCode MPI2_EVENT_SAS_INIT_RC_NOT_RESPONDING
81 * for SAS Initiator Device Status Change Event data.
82 * Modified Reason Code defines for SAS Topology Change
83 * List Event data, including adding a bit for PHY Vacant
84 * status, and adding a mask for the Reason Code.
85 * Added define for
86 * MPI2_EVENT_SAS_TOPO_ES_DELAY_NOT_RESPONDING.
87 * Added define for MPI2_EXT_IMAGE_TYPE_MEGARAID.
88 * 12-18-07 02.00.05 Added Boot Status defines for the IOCExceptions field of
89 * the IOCFacts Reply.
90 * Removed MPI2_IOCFACTS_CAPABILITY_EXTENDED_BUFFER define.
91 * Moved MPI2_VERSION_UNION to mpi2.h.
92 * Changed MPI2_EVENT_NOTIFICATION_REQUEST to use masks
93 * instead of enables, and added SASBroadcastPrimitiveMasks
94 * field.
95 * Added Log Entry Added Event and related structure.
96 * 02-29-08 02.00.06 Added define MPI2_IOCFACTS_CAPABILITY_INTEGRATED_RAID.
97 * Removed define MPI2_IOCFACTS_PROTOCOL_SMP_TARGET.
98 * Added MaxVolumes and MaxPersistentEntries fields to
99 * IOCFacts reply.
100 * Added ProtocolFlags and IOCCapabilities fields to
101 * MPI2_FW_IMAGE_HEADER.
102 * Removed MPI2_PORTENABLE_FLAGS_ENABLE_SINGLE_PORT.
103 * 03-03-08 02.00.07 Fixed MPI2_FW_IMAGE_HEADER by changing Reserved26 to
104 * a U16 (from a U32).
105 * Removed extra 's' from EventMasks name.
106 * 06-27-08 02.00.08 Fixed an offset in a comment.
107 * 10-02-08 02.00.09 Removed SystemReplyFrameSize from MPI2_IOC_INIT_REQUEST.
108 * Removed CurReplyFrameSize from MPI2_IOC_FACTS_REPLY and
109 * renamed MinReplyFrameSize to ReplyFrameSize.
110 * Added MPI2_IOCFACTS_EXCEPT_IR_FOREIGN_CONFIG_MAX.
111 * Added two new RAIDOperation values for Integrated RAID
112 * Operations Status Event data.
113 * Added four new IR Configuration Change List Event data
114 * ReasonCode values.
115 * Added two new ReasonCode defines for SAS Device Status
116 * Change Event data.
117 * Added three new DiscoveryStatus bits for the SAS
118 * Discovery event data.
119 * Added Multiplexing Status Change bit to the PhyStatus
120 * field of the SAS Topology Change List event data.
121 * Removed define for MPI2_INIT_IMAGE_BOOTFLAGS_XMEMCOPY.
122 * BootFlags are now product-specific.
123 * Added defines for the individual signature bytes
124 * for MPI2_INIT_IMAGE_FOOTER.
125 * 01-19-09 02.00.10 Added MPI2_IOCFACTS_CAPABILITY_EVENT_REPLAY define.
126 * Added MPI2_EVENT_SAS_DISC_DS_DOWNSTREAM_INITIATOR
127 * define.
```

```

128 *      Added MPI2_EVENT_SAS_DEV_STAT_RC_SATA_INIT_FAILURE
129 *      define.
130 *      Removed MPI2_EVENT_SAS_DISC_DS_SATA_INIT_FAILURE define.
131 * 05-06-09 02.00.11 Added MPI2_IOCFACTS_CAPABILITY_RAID_ACCELERATOR define.
132 *      Added MPI2_IOCFACTS_CAPABILITY_MSI_X_INDEX define.
133 *      Added two new reason codes for SAS Device Status Change
134 *      Event.
135 *      Added new event: SAS PHY Counter.
136 * 07-30-09 02.00.12 Added GPIO Interrupt event define and structure.
137 *      Added MPI2_IOCFACTS_CAPABILITY_EXTENDED_BUFFER define.
138 *      Added new product id family for 2208.
139 * 10-28-09 02.00.13 Added HostMSIxVectors field to MPI2_IOC_INIT_REQUEST.
140 *      Added MaxMSIxVectors field to MPI2_IOC_FACTS_REPLY.
141 *      Added MinDevHandle field to MPI2_IOC_FACTS_REPLY.
142 *      Added MPI2_IOCFACTS_CAPABILITY_HOST_BASED_DISCOVERY.
143 *      Added MPI2_EVENT_HOST_BASED_DISCOVERY_PHY define.
144 *      Added MPI2_EVENT_SAS_TOPO_ES_NO_EXPANDER define.
145 *      Added Host Based Discovery Phy Event data.
146 *      Added defines for ProductID Product field
147 *      (MPI2_FW_HEADER_PID_).
148 *      Modified values for SAS ProductID Family
149 *      (MPI2_FW_HEADER_PID_FAMILY_).
150 * 02-10-10 02.00.14 Added SAS Quiesce Event structure and defines.
151 *      Added PowerManagementControl Request structures and
152 *      defines.
153 * 05-12-10 02.00.15 Marked Task Set Full Event as obsolete.
154 *      Added MPI2_EVENT_SAS_TOPO_LR_UNSUPPORTED_PHY define.
155 * 11-10-10 02.00.16 Added MPI2_FW_DOWNLOAD_ITYPE_MIN_PRODUCT_SPECIFIC.
156 * -----
157 */

159 #ifndef MPI2_IOC_H
160 #define MPI2_IOC_H

162 /*****
163 *
164 *      IOC Messages
165 *
166 *****/

168 /*****
169 *      IOCInit message
170 *****/

172 /* IOCInit Request message */
173 typedef struct _MPI2_IOC_INIT_REQUEST
174 {
175     U8      WhoInit;          /* 0x00 */
176     U8      Reserved1;        /* 0x01 */
177     U8      ChainOffset;      /* 0x02 */
178     U8      Function;         /* 0x03 */
179     U16     Reserved2;        /* 0x04 */
180     U8      Reserved3;        /* 0x06 */
181     U8      MsgFlags;         /* 0x07 */
182     U8      VP_ID;            /* 0x08 */
183     U8      VF_ID;            /* 0x09 */
184     U16     Reserved4;        /* 0x0A */
185     U16     MsgVersion;       /* 0x0C */
186     U16     HeaderVersion;    /* 0x0E */
187     U32     Reserved5;        /* 0x10 */
188     U16     Reserved6;        /* 0x14 */
189     U8      Reserved7;        /* 0x16 */
190     U8      HostMSIxVectors;  /* 0x17 */
191     U16     Reserved8;        /* 0x18 */
192     U16     SystemRequestFrameSize; /* 0x1A */
193     U16     ReplyDescriptorPostQueueDepth; /* 0x1C */

```

```

194     U16     ReplyFreeQueueDepth; /* 0x1E */
195     U32     SenseBufferAddressHigh; /* 0x20 */
196     U32     SystemReplyAddressHigh; /* 0x24 */
197     U64     SystemRequestFrameBaseAddress; /* 0x28 */
198     U64     ReplyDescriptorPostQueueAddress; /* 0x30 */
199     U64     ReplyFreeQueueAddress; /* 0x38 */
200     U64     TimeStamp; /* 0x40 */
201 } MPI2_IOC_INIT_REQUEST, MPI2_POINTER PTR_MPI2_IOC_INIT_REQUEST,
202   Mpi2IOCInitRequest_t, MPI2_POINTER pMpi2IOCInitRequest_t;

204 /* WhoInit values */
205 #define MPI2_WHOINIT_NOT_INITIALIZED (0x00)
206 #define MPI2_WHOINIT_SYSTEM_BIOS (0x01)
207 #define MPI2_WHOINIT_ROM_BIOS (0x02)
208 #define MPI2_WHOINIT_PCI_PEER (0x03)
209 #define MPI2_WHOINIT_HOST_DRIVER (0x04)
210 #define MPI2_WHOINIT_MANUFACTURER (0x05)

212 /* MsgVersion */
213 #define MPI2_IOCINIT_MSGVERSION_MAJOR_MASK (0xFF00)
214 #define MPI2_IOCINIT_MSGVERSION_MAJOR_SHIFT (8)
215 #define MPI2_IOCINIT_MSGVERSION_MINOR_MASK (0x00FF)
216 #define MPI2_IOCINIT_MSGVERSION_MINOR_SHIFT (0)

218 /* HeaderVersion */
219 #define MPI2_IOCINIT_HDRVERSION_UNIT_MASK (0xFF00)
220 #define MPI2_IOCINIT_HDRVERSION_UNIT_SHIFT (8)
221 #define MPI2_IOCINIT_HDRVERSION_DEV_MASK (0x00FF)
222 #define MPI2_IOCINIT_HDRVERSION_DEV_SHIFT (0)

224 /* minimum depth for the Reply Descriptor Post Queue */
225 #define MPI2_RDPQ_DEPTH_MIN (16)

228 /* IOCInit Reply message */
229 typedef struct _MPI2_IOC_INIT_REPLY
230 {
231     U8      WhoInit;          /* 0x00 */
232     U8      Reserved1;        /* 0x01 */
233     U8      MsgLength;        /* 0x02 */
234     U8      Function;         /* 0x03 */
235     U16     Reserved2;        /* 0x04 */
236     U8      Reserved3;        /* 0x06 */
237     U8      MsgFlags;         /* 0x07 */
238     U8      VP_ID;            /* 0x08 */
239     U8      VF_ID;            /* 0x09 */
240     U16     Reserved4;        /* 0x0A */
241     U16     Reserved5;        /* 0x0C */
242     U16     IOCStatus;        /* 0x0E */
243     U32     IOCLogInfo;       /* 0x10 */
244 } MPI2_IOC_INIT_REPLY, MPI2_POINTER PTR_MPI2_IOC_INIT_REPLY,
245   Mpi2IOCInitReply_t, MPI2_POINTER pMpi2IOCInitReply_t;

248 /*****
249 *      IOCFacts message
250 *****/

252 /* IOCFacts Request message */
253 typedef struct _MPI2_IOC_FACTS_REQUEST
254 {
255     U16     Reserved1;        /* 0x00 */
256     U8      ChainOffset;      /* 0x02 */
257     U8      Function;         /* 0x03 */
258     U16     Reserved2;        /* 0x04 */
259     U8      Reserved3;        /* 0x06 */

```

```

260     U8             MsgFlags;                /* 0x07 */
261     U8             VP_ID;                   /* 0x08 */
262     U8             VF_ID;                   /* 0x09 */
263     U16            Reserved4;               /* 0x0A */
264 } MPI2_IOC_FACTS_REQUEST, MPI2_POINTER PTR_MPI2_IOC_FACTS_REQUEST,
265   Mpi2IOCFactsRequest_t, MPI2_POINTER pMpi2IOCFactsRequest_t;

268 /* IOCFacts Reply message */
269 typedef struct _MPI2_IOC_FACTS_REPLY
270 {
271     U16            MsgVersion;              /* 0x00 */
272     U8             MsgLength;               /* 0x02 */
273     U8             Function;                /* 0x03 */
274     U16            HeaderVersion;           /* 0x04 */
275     U8             IOCTNumber;             /* 0x06 */
276     U8             MsgFlags;               /* 0x07 */
277     U8             VP_ID;                  /* 0x08 */
278     U8             VF_ID;                  /* 0x09 */
279     U16            Reserved1;               /* 0x0A */
280     U16            IOCExceptions;          /* 0x0C */
281     U16            IOCStatus;              /* 0x0E */
282     U32            IOCLogInfo;             /* 0x10 */
283     U8             MaxChainDepth;          /* 0x14 */
284     U8             WhoInit;                /* 0x15 */
285     U8             NumberOfPorts;          /* 0x16 */
286     U8             MaxMSIXVectors;         /* 0x17 */
287     U16            RequestCredit;          /* 0x18 */
288     U16            ProductID;              /* 0x1A */
289     U32            IOCCapabilities;        /* 0x1C */
290     MPI2_VERSION_UNION FWVersion;          /* 0x20 */
291     U16            IOCRequestFrameSize;    /* 0x24 */
292     U16            Reserved3;              /* 0x26 */
293     U16            MaxInitiators;          /* 0x28 */
294     U16            MaxTargets;             /* 0x2A */
295     U16            MaxSasExpanders;        /* 0x2C */
296     U16            MaxEnclosures;          /* 0x2E */
297     U16            ProtocolFlags;          /* 0x30 */
298     U16            HighPriorityCredit;      /* 0x32 */
299     U16            MaxReplyDescriptorPostQueueDepth; /* 0x34 */
300     U8             ReplyFrameSize;         /* 0x36 */
301     U8             MaxVolumes;             /* 0x37 */
302     U16            MaxDevHandle;           /* 0x38 */
303     U16            MaxPersistentEntries;   /* 0x3A */
304     U16            MinDevHandle;           /* 0x3C */
305     U16            Reserved4;              /* 0x3E */
306 } MPI2_IOC_FACTS_REPLY, MPI2_POINTER PTR_MPI2_IOC_FACTS_REPLY,
307   Mpi2IOCFactsReply_t, MPI2_POINTER pMpi2IOCFactsReply_t;

309 /* MsgVersion */
310 #define MPI2_IOCFACTS_MSGVERSION_MAJOR_MASK (0xFF00)
311 #define MPI2_IOCFACTS_MSGVERSION_MAJOR_SHIFT (8)
312 #define MPI2_IOCFACTS_MSGVERSION_MINOR_MASK (0x00FF)
313 #define MPI2_IOCFACTS_MSGVERSION_MINOR_SHIFT (0)

315 /* HeaderVersion */
316 #define MPI2_IOCFACTS_HDRVERSION_UNIT_MASK (0xFF00)
317 #define MPI2_IOCFACTS_HDRVERSION_UNIT_SHIFT (8)
318 #define MPI2_IOCFACTS_HDRVERSION_DEV_MASK (0x00FF)
319 #define MPI2_IOCFACTS_HDRVERSION_DEV_SHIFT (0)

321 /* IOCExceptions */
322 #define MPI2_IOCFACTS_EXCEPT_IR_FOREIGN_CONFIG_MAX (0x0100)

324 #define MPI2_IOCFACTS_EXCEPT_BOOTSTAT_MASK (0x00E0)
325 #define MPI2_IOCFACTS_EXCEPT_BOOTSTAT_GOOD (0x0000)

```

```

326 #define MPI2_IOCFACTS_EXCEPT_BOOTSTAT_BACKUP (0x0020)
327 #define MPI2_IOCFACTS_EXCEPT_BOOTSTAT_RESTORED (0x0040)
328 #define MPI2_IOCFACTS_EXCEPT_BOOTSTAT_CORRUPT_BACKUP (0x0060)

330 #define MPI2_IOCFACTS_EXCEPT_METADATA_UNSUPPORTED (0x0010)
331 #define MPI2_IOCFACTS_EXCEPT_MANUFACT_CHECKSUM_FAIL (0x0008)
332 #define MPI2_IOCFACTS_EXCEPT_FW_CHECKSUM_FAIL (0x0004)
333 #define MPI2_IOCFACTS_EXCEPT_RAID_CONFIG_INVALID (0x0002)
334 #define MPI2_IOCFACTS_EXCEPT_CONFIG_CHECKSUM_FAIL (0x0001)

336 /* defines for WhoInit field are after the IOCInit Request */

338 /* ProductID field uses MPI2_FW_HEADER_PID */

340 /* IOCCapabilities */
341 #define MPI25_IOCFACTS_CAPABILITY_FAST_PATH_CAPABLE (0x00020000)
342 #define MPI2_IOCFACTS_CAPABILITY_HOST_BASED_DISCOVERY (0x00010000)
343 #define MPI2_IOCFACTS_CAPABILITY_MSI_X_INDEX (0x00008000)
344 #define MPI2_IOCFACTS_CAPABILITY_RAID_ACCELERATOR (0x00004000)
345 #define MPI2_IOCFACTS_CAPABILITY_EVENT_REPLAY (0x00002000)
346 #define MPI2_IOCFACTS_CAPABILITY_INTEGRATED_RAID (0x00001000)
347 #define MPI2_IOCFACTS_CAPABILITY_TLR (0x00000800)
348 #define MPI2_IOCFACTS_CAPABILITY_MULTICAST (0x00000100)
349 #define MPI2_IOCFACTS_CAPABILITY_BIDIRECTIONAL_TARGET (0x00000080)
350 #define MPI2_IOCFACTS_CAPABILITY_EEDP (0x00000040)
351 #define MPI2_IOCFACTS_CAPABILITY_EXTENDED_BUFFER (0x00000020)
352 #define MPI2_IOCFACTS_CAPABILITY_SNAPSHOT_BUFFER (0x00000010)
353 #define MPI2_IOCFACTS_CAPABILITY_DIAG_TRACE_BUFFER (0x00000008)
354 #define MPI2_IOCFACTS_CAPABILITY_TASK_SET_FULL_HANDLING (0x00000004)

356 /* ProtocolFlags */
357 #define MPI2_IOCFACTS_PROTOCOL_SCSI_TARGET (0x0001)
358 #define MPI2_IOCFACTS_PROTOCOL_SCSI_INITIATOR (0x0002)

361 /*****
362  * PortFacts message
363  *****/

365 /* PortFacts Request message */
366 typedef struct _MPI2_PORT_FACTS_REQUEST
367 {
368     U16            Reserved1;              /* 0x00 */
369     U8             ChainOffset;            /* 0x02 */
370     U8             Function;               /* 0x03 */
371     U16            Reserved2;             /* 0x04 */
372     U8             PortNumber;             /* 0x06 */
373     U8             MsgFlags;              /* 0x07 */
374     U8             VP_ID;                  /* 0x08 */
375     U8             VF_ID;                  /* 0x09 */
376     U16            Reserved3;             /* 0x0A */
377 } MPI2_PORT_FACTS_REQUEST, MPI2_POINTER PTR_MPI2_PORT_FACTS_REQUEST,
378   Mpi2PortFactsRequest_t, MPI2_POINTER pMpi2PortFactsRequest_t;

380 /* PortFacts Reply message */
381 typedef struct _MPI2_PORT_FACTS_REPLY
382 {
383     U16            Reserved1;              /* 0x00 */
384     U8             MsgLength;              /* 0x02 */
385     U8             Function;               /* 0x03 */
386     U16            Reserved2;             /* 0x04 */
387     U8             PortNumber;             /* 0x06 */
388     U8             MsgFlags;              /* 0x07 */
389     U8             VP_ID;                  /* 0x08 */
390     U8             VF_ID;                  /* 0x09 */
391     U16            Reserved3;             /* 0x0A */

```

```

392     U16             Reserved4;           /* 0x0C */
393     U16             IOCStatus;           /* 0x0E */
394     U32             IOCLogInfo;          /* 0x10 */
395     U8              Reserved5;           /* 0x14 */
396     U8              PortType;            /* 0x15 */
397     U16             Reserved6;           /* 0x16 */
398     U16             MaxPostedCmdBuffers; /* 0x18 */
399     U16             Reserved7;           /* 0x1A */
400 } MPI2_PORT_FACTS_REPLY, MPI2_POINTER PTR_MPI2_PORT_FACTS_REPLY,
401   Mpi2PortFactsReply_t, MPI2_POINTER pMpi2PortFactsReply_t;

403 /* PortType values */
404 #define MPI2_PORTFACTS_PORTTYPE_INACTIVE (0x00)
405 #define MPI2_PORTFACTS_PORTTYPE_FC      (0x10)
406 #define MPI2_PORTFACTS_PORTTYPE_ISCSI   (0x20)
407 #define MPI2_PORTFACTS_PORTTYPE_SAS_PHYSICAL (0x30)
408 #define MPI2_PORTFACTS_PORTTYPE_SAS_VIRTUAL (0x31)

411 /*****
412  * PortEnable message
413  *****/

415 /* PortEnable Request message */
416 typedef struct _MPI2_PORT_ENABLE_REQUEST
417 {
418     U16             Reserved1;           /* 0x00 */
419     U8              ChainOffset;         /* 0x02 */
420     U8              Function;            /* 0x03 */
421     U8              Reserved2;           /* 0x04 */
422     U8              PortFlags;           /* 0x05 */
423     U8              Reserved3;           /* 0x06 */
424     U8              MsgFlags;            /* 0x07 */
425     U8              VP_ID;               /* 0x08 */
426     U8              VF_ID;               /* 0x09 */
427     U16             Reserved4;           /* 0x0A */
428 } MPI2_PORT_ENABLE_REQUEST, MPI2_POINTER PTR_MPI2_PORT_ENABLE_REQUEST,
429   Mpi2PortEnableRequest_t, MPI2_POINTER pMpi2PortEnableRequest_t;

432 /* PortEnable Reply message */
433 typedef struct _MPI2_PORT_ENABLE_REPLY
434 {
435     U16             Reserved1;           /* 0x00 */
436     U8              MsgLength;           /* 0x02 */
437     U8              Function;            /* 0x03 */
438     U8              Reserved2;           /* 0x04 */
439     U8              PortFlags;           /* 0x05 */
440     U8              Reserved3;           /* 0x06 */
441     U8              MsgFlags;            /* 0x07 */
442     U8              VP_ID;               /* 0x08 */
443     U8              VF_ID;               /* 0x09 */
444     U16             Reserved4;           /* 0x0A */
445     U16             Reserved5;           /* 0x0C */
446     U16             IOCStatus;           /* 0x0E */
447     U32             IOCLogInfo;          /* 0x10 */
448 } MPI2_PORT_ENABLE_REPLY, MPI2_POINTER PTR_MPI2_PORT_ENABLE_REPLY,
449   Mpi2PortEnableReply_t, MPI2_POINTER pMpi2PortEnableReply_t;

452 /*****
453  * EventNotification message
454  *****/

456 /* EventNotification Request message */
457 #define MPI2_EVENT_NOTIFY_EVENTMASK_WORDS (4)

```

```

459 typedef struct _MPI2_EVENT_NOTIFICATION_REQUEST
460 {
461     U16             Reserved1;           /* 0x00 */
462     U8              ChainOffset;         /* 0x02 */
463     U8              Function;            /* 0x03 */
464     U16             Reserved2;           /* 0x04 */
465     U8              Reserved3;           /* 0x06 */
466     U8              MsgFlags;            /* 0x07 */
467     U8              VP_ID;               /* 0x08 */
468     U8              VF_ID;               /* 0x09 */
469     U16             Reserved4;           /* 0x0A */
470     U32             Reserved5;           /* 0x0C */
471     U32             Reserved6;           /* 0x10 */
472     U32             EventMasks[MPI2_EVENT_NOTIFY_EVENTMASK_WORDS]; /* 0x1
473     U16             SASBroadcastPrimitiveMasks; /* 0x24 */
474     U16             Reserved7;           /* 0x26 */
475     U32             Reserved8;           /* 0x28 */
476 } MPI2_EVENT_NOTIFICATION_REQUEST,
477   MPI2_POINTER PTR_MPI2_EVENT_NOTIFICATION_REQUEST,
478   Mpi2EventNotificationRequest_t, MPI2_POINTER pMpi2EventNotificationRequest_t;

481 /* EventNotification Reply message */
482 typedef struct _MPI2_EVENT_NOTIFICATION_REPLY
483 {
484     U16             EventDataLength;      /* 0x00 */
485     U8              MsgLength;           /* 0x02 */
486     U8              Function;            /* 0x03 */
487     U16             Reserved1;           /* 0x04 */
488     U8              AckRequired;         /* 0x06 */
489     U8              MsgFlags;            /* 0x07 */
490     U8              VP_ID;               /* 0x08 */
491     U8              VF_ID;               /* 0x09 */
492     U16             Reserved2;           /* 0x0A */
493     U16             Reserved3;           /* 0x0C */
494     U16             IOCStatus;           /* 0x0E */
495     U32             IOCLogInfo;          /* 0x10 */
496     U16             Event;               /* 0x14 */
497     U16             Reserved4;           /* 0x16 */
498     U32             EventContext;         /* 0x18 */
499     U32             EventData[1];        /* 0x1C */
500 } MPI2_EVENT_NOTIFICATION_REPLY, MPI2_POINTER PTR_MPI2_EVENT_NOTIFICATION_REPLY,
501   Mpi2EventNotificationReply_t, MPI2_POINTER pMpi2EventNotificationReply_t;

503 /* AckRequired */
504 #define MPI2_EVENT_NOTIFICATION_ACK_NOT_REQUIRED (0x00)
505 #define MPI2_EVENT_NOTIFICATION_ACK_REQUIRED (0x01)

507 /* Event */
508 #define MPI2_EVENT_LOG_DATA (0x0001)
509 #define MPI2_EVENT_SAS_INIT_DEVICE_STATUS_CHANGE (0x0002)
510 #define MPI2_EVENT_HARD_RESET_RECEIVED (0x0005)
511 #define MPI2_EVENT_EVENT_CHANGE (0x000A)
512 #define MPI2_EVENT_TASK_SET_FULL (0x000E) /* obsolete */
513 #define MPI2_EVENT_SAS_DEVICE_STATUS_CHANGE (0x000F)
514 #define MPI2_EVENT_IR_OPERATION_STATUS (0x0014)
515 #define MPI2_EVENT_SAS_DISCOVERY (0x0016)
516 #define MPI2_EVENT_SAS_BROADCAST_PRIMITIVE (0x0017)
517 #define MPI2_EVENT_SAS_INIT_DEVICE_STATUS_CHANGE (0x0018)
518 #define MPI2_EVENT_SAS_INIT_TABLE_OVERFLOW (0x0019)
519 #define MPI2_EVENT_SAS_TOPOLOGY_CHANGE_LIST (0x001C)
520 #define MPI2_EVENT_SAS_ENCL_DEVICE_STATUS_CHANGE (0x001D)
521 #define MPI2_EVENT_IR_VOLUME (0x001E)
522 #define MPI2_EVENT_IR_PHYSICAL_DISK (0x001F)
523 #define MPI2_EVENT_IR_CONFIGURATION_CHANGE_LIST (0x0020)

```



```

524 #define MPI2_EVENT_LOG_ENTRY_ADDED          (0x0021)
525 #define MPI2_EVENT_SAS_PHY_COUNTER          (0x0022)
526 #define MPI2_EVENT_GPIO_INTERRUPT           (0x0023)
527 #define MPI2_EVENT_HOST_BASED_DISCOVERY_PHY (0x0024)
528 #define MPI2_EVENT_SAS_QUIESCE              (0x0025)

531 /* Log Entry Added Event data */

533 /* the following structure matches MPI2_LOG_0_ENTRY in mpi2_cnfg.h */
534 #define MPI2_EVENT_DATA_LOG_DATA_LENGTH      (0x1C)

536 typedef struct _MPI2_EVENT_DATA_LOG_ENTRY_ADDED
537 {
538     U64      TimeStamp;          /* 0x00 */
539     U32      Reserved1;          /* 0x08 */
540     U16      LogSequence;        /* 0x0C */
541     U16      LogEntryQualifier;   /* 0x0E */
542     U8       VP_ID;              /* 0x10 */
543     U8       VF_ID;              /* 0x11 */
544     U16      Reserved2;          /* 0x12 */
545     U8       LogData[MPI2_EVENT_DATA_LOG_DATA_LENGTH]; /* 0x14 */
546 } MPI2_EVENT_DATA_LOG_ENTRY_ADDED,
547 MPI2_POINTER PTR_MPI2_EVENT_DATA_LOG_ENTRY_ADDED,
548 Mpi2EventDataLogEntryAdded_t, MPI2_POINTER pMpi2EventDataLogEntryAdded_t;

550 /* GPIO Interrupt Event data */

552 typedef struct _MPI2_EVENT_DATA_GPIO_INTERRUPT
553 {
554     U8      GPIONum;             /* 0x00 */
555     U8      Reserved1;           /* 0x01 */
556     U16     Reserved2;           /* 0x02 */
557 } MPI2_EVENT_DATA_GPIO_INTERRUPT,
558 MPI2_POINTER PTR_MPI2_EVENT_DATA_GPIO_INTERRUPT,
559 Mpi2EventDataGpioInterrupt_t, MPI2_POINTER pMpi2EventDataGpioInterrupt_t;

561 /* Hard Reset Received Event data */

563 typedef struct _MPI2_EVENT_DATA_HARD_RESET_RECEIVED
564 {
565     U8      Reserved1;           /* 0x00 */
566     U8      Port;                /* 0x01 */
567     U16     Reserved2;           /* 0x02 */
568 } MPI2_EVENT_DATA_HARD_RESET_RECEIVED,
569 MPI2_POINTER PTR_MPI2_EVENT_DATA_HARD_RESET_RECEIVED,
570 Mpi2EventDataHardResetReceived_t,
571 MPI2_POINTER pMpi2EventDataHardResetReceived_t;

573 /* Task Set Full Event data */
574 /* this event is obsolete */

576 typedef struct _MPI2_EVENT_DATA_TASK_SET_FULL
577 {
578     U16     DevHandle;           /* 0x00 */
579     U16     CurrentDepth;        /* 0x02 */
580 } MPI2_EVENT_DATA_TASK_SET_FULL, MPI2_POINTER PTR_MPI2_EVENT_DATA_TASK_SET_FULL,
581 Mpi2EventDataTaskSetFull_t, MPI2_POINTER pMpi2EventDataTaskSetFull_t;

584 /* SAS Device Status Change Event data */

586 typedef struct _MPI2_EVENT_DATA_SAS_DEVICE_STATUS_CHANGE
587 {
588     U16     TaskTag;             /* 0x00 */
589     U8      ReasonCode;          /* 0x02 */

```

```

590     U8      Reserved1;          /* 0x03 */
591     U8      ASC;                 /* 0x04 */
592     U8      ASCQ;                /* 0x05 */
593     U16     DevHandle;           /* 0x06 */
594     U32     Reserved2;           /* 0x08 */
595     U64     SASAddress;          /* 0x0C */
596     U8      LUN[8];             /* 0x14 */
597 } MPI2_EVENT_DATA_SAS_DEVICE_STATUS_CHANGE,
598 MPI2_POINTER PTR_MPI2_EVENT_DATA_SAS_DEVICE_STATUS_CHANGE,
599 Mpi2EventDataSasDeviceStatusChange_t,
600 MPI2_POINTER pMpi2EventDataSasDeviceStatusChange_t;

602 /* SAS Device Status Change Event data ReasonCode values */
603 #define MPI2_EVENT_SAS_DEV_STAT_RC_SMART_DATA          (0x05)
604 #define MPI2_EVENT_SAS_DEV_STAT_RC_UNSUPPORTED        (0x07)
605 #define MPI2_EVENT_SAS_DEV_STAT_RC_INTERNAL_DEVICE_RESET (0x08)
606 #define MPI2_EVENT_SAS_DEV_STAT_RC_TASK_ABORT_INTERNAL (0x09)
607 #define MPI2_EVENT_SAS_DEV_STAT_RC_ABORT_TASK_SET_INTERNAL (0x0A)
608 #define MPI2_EVENT_SAS_DEV_STAT_RC_CLEAR_TASK_SET_INTERNAL (0x0B)
609 #define MPI2_EVENT_SAS_DEV_STAT_RC_QUERY_TASK_INTERNAL (0x0C)
610 #define MPI2_EVENT_SAS_DEV_STAT_RC_ASYNC_NOTIFICATION (0x0D)
611 #define MPI2_EVENT_SAS_DEV_STAT_RC_CMP_INTERNAL_DEV_RESET (0x0E)
612 #define MPI2_EVENT_SAS_DEV_STAT_RC_CMP_TASK_ABORT_INTERNAL (0x0F)
613 #define MPI2_EVENT_SAS_DEV_STAT_RC_SATA_INIT_FAILURE (0x10)
614 #define MPI2_EVENT_SAS_DEV_STAT_RC_EXPANDER_REDUCED_FUNCTIONALITY (0x11)
615 #define MPI2_EVENT_SAS_DEV_STAT_RC_CMP_EXPANDER_REDUCED_FUNCTIONALITY (0x12)

618 /* Integrated RAID Operation Status Event data */

620 typedef struct _MPI2_EVENT_DATA_IR_OPERATION_STATUS
621 {
622     U16     VolDevHandle;        /* 0x00 */
623     U16     Reserved1;           /* 0x02 */
624     U8      RAIDOperation;       /* 0x04 */
625     U8      PercentComplete;     /* 0x05 */
626     U16     Reserved2;           /* 0x06 */
627     U32     Resereved3;          /* 0x08 */
628 } MPI2_EVENT_DATA_IR_OPERATION_STATUS,
629 MPI2_POINTER PTR_MPI2_EVENT_DATA_IR_OPERATION_STATUS,
630 Mpi2EventDataIrOperationStatus_t,
631 MPI2_POINTER pMpi2EventDataIrOperationStatus_t;

633 /* Integrated RAID Operation Status Event data RAIDOperation values */
634 #define MPI2_EVENT_IR_RAIDOP_RESYNC          (0x00)
635 #define MPI2_EVENT_IR_RAIDOP_ONLINE_CAP_EXPANSION (0x01)
636 #define MPI2_EVENT_IR_RAIDOP_CONSISTENCY_CHECK (0x02)
637 #define MPI2_EVENT_IR_RAIDOP_BACKGROUND_INIT (0x03)
638 #define MPI2_EVENT_IR_RAIDOP_MAKE_DATA_CONSISTENT (0x04)

641 /* Integrated RAID Volume Event data */

643 typedef struct _MPI2_EVENT_DATA_IR_VOLUME
644 {
645     U16     VolDevHandle;        /* 0x00 */
646     U8      ReasonCode;          /* 0x02 */
647     U8      Reserved1;           /* 0x03 */
648     U32     NewValue;            /* 0x04 */
649     U32     PreviousValue;       /* 0x08 */
650 } MPI2_EVENT_DATA_IR_VOLUME, MPI2_POINTER PTR_MPI2_EVENT_DATA_IR_VOLUME,
651 Mpi2EventDataIrVolume_t, MPI2_POINTER pMpi2EventDataIrVolume_t;

653 /* Integrated RAID Volume Event data ReasonCode values */
654 #define MPI2_EVENT_IR_VOLUME_RC_SETTINGS_CHANGED (0x01)
655 #define MPI2_EVENT_IR_VOLUME_RC_STATUS_FLAGS_CHANGED (0x02)

```

```

656 #define MPI2_EVENT_IR_VOLUME_RC_STATE_CHANGED      (0x03)

659 /* Integrated RAID Physical Disk Event data */

661 typedef struct _MPI2_EVENT_DATA_IR_PHYSICAL_DISK
662 {
663     U16      Reserved1;          /* 0x00 */
664     U8       ReasonCode;         /* 0x02 */
665     U8       PhysDiskNum;        /* 0x03 */
666     U16      PhysDiskDevHandle;  /* 0x04 */
667     U16      Reserved2;         /* 0x06 */
668     U16      Slot;              /* 0x08 */
669     U16      EnclosureHandle;    /* 0x0A */
670     U32      NewValue;          /* 0x0C */
671     U32      PreviousValue;     /* 0x10 */
672 } MPI2_EVENT_DATA_IR_PHYSICAL_DISK,
673 MPI2_POINTER PTR_MPI2_EVENT_DATA_IR_PHYSICAL_DISK,
674 MPI2EventDataIrPhysicalDisk_t, MPI2_POINTER pMpi2EventDataIrPhysicalDisk_t;

676 /* Integrated RAID Physical Disk Event data ReasonCode values */
677 #define MPI2_EVENT_IR_PHYSDISK_RC_SETTINGS_CHANGED (0x01)
678 #define MPI2_EVENT_IR_PHYSDISK_RC_STATUS_FLAGS_CHANGED (0x02)
679 #define MPI2_EVENT_IR_PHYSDISK_RC_STATE_CHANGED (0x03)

682 /* Integrated RAID Configuration Change List Event data */

684 /*
685  * Host code (drivers, BIOS, utilities, etc.) should leave this define set to
686  * one and check NumElements at runtime.
687  */
688 #ifndef MPI2_EVENT_IR_CONFIG_ELEMENT_COUNT
689 #define MPI2_EVENT_IR_CONFIG_ELEMENT_COUNT      (1)
690 #endif

692 typedef struct _MPI2_EVENT_IR_CONFIG_ELEMENT
693 {
694     U16      ElementFlags;       /* 0x00 */
695     U16      VolDevHandle;       /* 0x02 */
696     U8       ReasonCode;         /* 0x04 */
697     U8       PhysDiskNum;        /* 0x05 */
698     U16      PhysDiskDevHandle;  /* 0x06 */
699 } MPI2_EVENT_IR_CONFIG_ELEMENT, MPI2_POINTER PTR_MPI2_EVENT_IR_CONFIG_ELEMENT,
700 MPI2EventIrConfigElement_t, MPI2_POINTER pMpi2EventIrConfigElement_t;

702 /* IR Configuration Change List Event data ElementFlags values */
703 #define MPI2_EVENT_IR_CHANGE_EFLAGS_ELEMENT_TYPE_MASK (0x000F)
704 #define MPI2_EVENT_IR_CHANGE_EFLAGS_VOLUME_ELEMENT (0x0000)
705 #define MPI2_EVENT_IR_CHANGE_EFLAGS_VOLPHYSDISK_ELEMENT (0x0001)
706 #define MPI2_EVENT_IR_CHANGE_EFLAGS_HOTSPARE_ELEMENT (0x0002)

708 /* IR Configuration Change List Event data ReasonCode values */
709 #define MPI2_EVENT_IR_CHANGE_RC_ADDED (0x01)
710 #define MPI2_EVENT_IR_CHANGE_RC_REMOVED (0x02)
711 #define MPI2_EVENT_IR_CHANGE_RC_NO_CHANGE (0x03)
712 #define MPI2_EVENT_IR_CHANGE_RC_HIDE (0x04)
713 #define MPI2_EVENT_IR_CHANGE_RC_UNHIDE (0x05)
714 #define MPI2_EVENT_IR_CHANGE_RC_VOLUME_CREATED (0x06)
715 #define MPI2_EVENT_IR_CHANGE_RC_VOLUME_DELETED (0x07)
716 #define MPI2_EVENT_IR_CHANGE_RC_PD_CREATED (0x08)
717 #define MPI2_EVENT_IR_CHANGE_RC_PD_DELETED (0x09)

719 typedef struct _MPI2_EVENT_DATA_IR_CONFIG_CHANGE_LIST
720 {
721     U8      NumElements;        /* 0x00 */

```

```

722     U8      Reserved1;          /* 0x01 */
723     U8      Reserved2;          /* 0x02 */
724     U8      ConfigNum;          /* 0x03 */
725     U32      Flags;             /* 0x04 */
726     MPI2_EVENT_IR_CONFIG_ELEMENT ConfigElement[MPI2_EVENT_IR_CONFIG_ELEMENT_C
727 } MPI2_EVENT_DATA_IR_CONFIG_CHANGE_LIST,
728 MPI2_POINTER PTR_MPI2_EVENT_DATA_IR_CONFIG_CHANGE_LIST,
729 MPI2EventDataIrConfigChangeList_t,
730 MPI2_POINTER pMpi2EventDataIrConfigChangeList_t;

732 /* IR Configuration Change List Event data Flags values */
733 #define MPI2_EVENT_IR_CHANGE_FLAGS_FOREIGN_CONFIG (0x00000001)

736 /* SAS Discovery Event data */

738 typedef struct _MPI2_EVENT_DATA_SAS_DISCOVERY
739 {
740     U8      Flags;              /* 0x00 */
741     U8      ReasonCode;         /* 0x01 */
742     U8      PhysicalPort;       /* 0x02 */
743     U8      Reserved1;          /* 0x03 */
744     U32      DiscoveryStatus;    /* 0x04 */
745 } MPI2_EVENT_DATA_SAS_DISCOVERY,
746 MPI2_POINTER PTR_MPI2_EVENT_DATA_SAS_DISCOVERY,
747 MPI2EventDataSasDiscovery_t, MPI2_POINTER pMpi2EventDataSasDiscovery_t;

749 /* SAS Discovery Event data Flags values */
750 #define MPI2_EVENT_SAS_DISC_DEVICE_CHANGE (0x02)
751 #define MPI2_EVENT_SAS_DISC_IN_PROGRESS (0x01)

753 /* SAS Discovery Event data ReasonCode values */
754 #define MPI2_EVENT_SAS_DISC_RC_STARTED (0x01)
755 #define MPI2_EVENT_SAS_DISC_RC_COMPLETED (0x02)

757 /* SAS Discovery Event data DiscoveryStatus values */
758 #define MPI2_EVENT_SAS_DISC_DS_MAX_ENCLOSURES_EXCEED (0x80000000)
759 #define MPI2_EVENT_SAS_DISC_DS_MAX_EXPANDERS_EXCEED (0x40000000)
760 #define MPI2_EVENT_SAS_DISC_DS_MAX_DEVICES_EXCEED (0x20000000)
761 #define MPI2_EVENT_SAS_DISC_DS_MAX_TOPO_PHYS_EXCEED (0x10000000)
762 #define MPI2_EVENT_SAS_DISC_DS_DOWNSTREAM_INITIATOR (0x08000000)
763 #define MPI2_EVENT_SAS_DISC_DS_MULTI_SUBTRACTIVE_SUBTRACTIVE (0x00008000)
764 #define MPI2_EVENT_SAS_DISC_DS_EXP_MULTI_SUBTRACTIVE (0x00004000)
765 #define MPI2_EVENT_SAS_DISC_DS_MULTI_PORT_DOMAIN (0x00002000)
766 #define MPI2_EVENT_SAS_DISC_DS_TABLE_TO_SUBTRACTIVE_LINK (0x00001000)
767 #define MPI2_EVENT_SAS_DISC_DS_UNSUPPORTED_DEVICE (0x00000800)
768 #define MPI2_EVENT_SAS_DISC_DS_TABLE_LINK (0x00000400)
769 #define MPI2_EVENT_SAS_DISC_DS_SUBTRACTIVE_LINK (0x00000200)
770 #define MPI2_EVENT_SAS_DISC_DS_SMP_CRC_ERROR (0x00000100)
771 #define MPI2_EVENT_SAS_DISC_DS_SMP_FUNCTION_FAILED (0x00000080)
772 #define MPI2_EVENT_SAS_DISC_DS_INDEX_NOT_EXIST (0x00000040)
773 #define MPI2_EVENT_SAS_DISC_DS_OUT_ROUTE_ENTRIES (0x00000020)
774 #define MPI2_EVENT_SAS_DISC_DS_SMP_TIMEOUT (0x00000010)
775 #define MPI2_EVENT_SAS_DISC_DS_MULTIPLE_PORTS (0x00000004)
776 #define MPI2_EVENT_SAS_DISC_DS_UNADDRESSABLE_DEVICE (0x00000002)
777 #define MPI2_EVENT_SAS_DISC_DS_LOOP_DETECTED (0x00000001)

780 /* SAS Broadcast Primitive Event data */

782 typedef struct _MPI2_EVENT_DATA_SAS_BROADCAST_PRIMITIVE
783 {
784     U8      PhyNum;             /* 0x00 */
785     U8      Port;               /* 0x01 */
786     U8      PortWidth;          /* 0x02 */
787     U8      Primitive;          /* 0x03 */

```

```

788 } MPI2_EVENT_DATA_SAS_BROADCAST_PRIMITIVE,
789 MPI2_POINTER PTR_MPI2_EVENT_DATA_SAS_BROADCAST_PRIMITIVE,
790 Mpi2EventDataSasBroadcastPrimitive_t,
791 MPI2_POINTER pMpi2EventDataSasBroadcastPrimitive_t;

793 /* defines for the Primitive field */
794 #define MPI2_EVENT_PRIMITIVE_CHANGE (0x01)
795 #define MPI2_EVENT_PRIMITIVE_SES (0x02)
796 #define MPI2_EVENT_PRIMITIVE_EXPANDER (0x03)
797 #define MPI2_EVENT_PRIMITIVE_ASYNCHRONOUS_EVENT (0x04)
798 #define MPI2_EVENT_PRIMITIVE_RESERVED3 (0x05)
799 #define MPI2_EVENT_PRIMITIVE_RESERVED4 (0x06)
800 #define MPI2_EVENT_PRIMITIVE_CHANGE0_RESERVED (0x07)
801 #define MPI2_EVENT_PRIMITIVE_CHANGE1_RESERVED (0x08)

804 /* SAS Initiator Device Status Change Event data */

806 typedef struct _MPI2_EVENT_DATA_SAS_INIT_DEV_STATUS_CHANGE
807 {
808     U8 ReasonCode; /* 0x00 */
809     U8 PhysicalPort; /* 0x01 */
810     U16 DevHandle; /* 0x02 */
811     U64 SASAddress; /* 0x04 */
812 } MPI2_EVENT_DATA_SAS_INIT_DEV_STATUS_CHANGE,
813 MPI2_POINTER PTR_MPI2_EVENT_DATA_SAS_INIT_DEV_STATUS_CHANGE,
814 Mpi2EventDataSasInitDevStatusChange_t,
815 MPI2_POINTER pMpi2EventDataSasInitDevStatusChange_t;

817 /* SAS Initiator Device Status Change event ReasonCode values */
818 #define MPI2_EVENT_SAS_INIT_RC_ADDED (0x01)
819 #define MPI2_EVENT_SAS_INIT_RC_NOT_RESPONDING (0x02)

822 /* SAS Initiator Device Table Overflow Event data */

824 typedef struct _MPI2_EVENT_DATA_SAS_INIT_TABLE_OVERFLOW
825 {
826     U16 MaxInit; /* 0x00 */
827     U16 CurrentInit; /* 0x02 */
828     U64 SASAddress; /* 0x04 */
829 } MPI2_EVENT_DATA_SAS_INIT_TABLE_OVERFLOW,
830 MPI2_POINTER PTR_MPI2_EVENT_DATA_SAS_INIT_TABLE_OVERFLOW,
831 Mpi2EventDataSasInitTableOverflow_t,
832 MPI2_POINTER pMpi2EventDataSasInitTableOverflow_t;

835 /* SAS Topology Change List Event data */

837 /*
838 * Host code (drivers, BIOS, utilities, etc.) should leave this define set to
839 * one and check NumEntries at runtime.
840 */
841 #ifndef MPI2_EVENT_SAS_TOPO_PHY_COUNT
842 #define MPI2_EVENT_SAS_TOPO_PHY_COUNT (1)
843 #endif

845 typedef struct _MPI2_EVENT_SAS_TOPO_PHY_ENTRY
846 {
847     U16 AttachedDevHandle; /* 0x00 */
848     U8 LinkRate; /* 0x02 */
849     U8 PhyStatus; /* 0x03 */
850 } MPI2_EVENT_SAS_TOPO_PHY_ENTRY, MPI2_POINTER PTR_MPI2_EVENT_SAS_TOPO_PHY_ENTRY,
851 Mpi2EventSasTopoPhyEntry_t, MPI2_POINTER pMpi2EventSasTopoPhyEntry_t;

853 typedef struct _MPI2_EVENT_DATA_SAS_TOPOLOGY_CHANGE_LIST

```

```

854 {
855     U16 EnclosureHandle; /* 0x00 */
856     U16 ExpanderDevHandle; /* 0x02 */
857     U8 NumPhys; /* 0x04 */
858     U8 Reserved1; /* 0x05 */
859     U16 Reserved2; /* 0x06 */
860     U8 NumEntries; /* 0x08 */
861     U8 StartPhyNum; /* 0x09 */
862     U8 ExpStatus; /* 0x0A */
863     U8 PhysicalPort; /* 0x0B */
864     MPI2_EVENT_SAS_TOPO_PHY_ENTRY PHY[MPI2_EVENT_SAS_TOPO_PHY_COUNT]; /* 0x0C */
865 } MPI2_EVENT_DATA_SAS_TOPOLOGY_CHANGE_LIST,
866 MPI2_POINTER PTR_MPI2_EVENT_DATA_SAS_TOPOLOGY_CHANGE_LIST,
867 Mpi2EventDataSasTopologyChangeList_t,
868 MPI2_POINTER pMpi2EventDataSasTopologyChangeList_t;

870 /* values for the ExpStatus field */
871 #define MPI2_EVENT_SAS_TOPO_ES_NO_EXPANDER (0x00)
872 #define MPI2_EVENT_SAS_TOPO_ES_ADDED (0x01)
873 #define MPI2_EVENT_SAS_TOPO_ES_NOT_RESPONDING (0x02)
874 #define MPI2_EVENT_SAS_TOPO_ES_RESPONDING (0x03)
875 #define MPI2_EVENT_SAS_TOPO_ES_DELAY_NOT_RESPONDING (0x04)

877 /* defines for the LinkRate field */
878 #define MPI2_EVENT_SAS_TOPO_LR_CURRENT_MASK (0xF0)
879 #define MPI2_EVENT_SAS_TOPO_LR_CURRENT_SHIFT (4)
880 #define MPI2_EVENT_SAS_TOPO_LR_PREV_MASK (0x0F)
881 #define MPI2_EVENT_SAS_TOPO_LR_PREV_SHIFT (0)

883 #define MPI2_EVENT_SAS_TOPO_LR_UNKNOWN_LINK_RATE (0x00)
884 #define MPI2_EVENT_SAS_TOPO_LR_PHY_DISABLED (0x01)
885 #define MPI2_EVENT_SAS_TOPO_LR_NEGOTIATION_FAILED (0x02)
886 #define MPI2_EVENT_SAS_TOPO_LR_SATA_OOB_COMPLETE (0x03)
887 #define MPI2_EVENT_SAS_TOPO_LR_PORT_SELECTOR (0x04)
888 #define MPI2_EVENT_SAS_TOPO_LR_SMP_RESET_IN_PROGRESS (0x05)
889 #define MPI2_EVENT_SAS_TOPO_LR_UNSUPPORTED_PHY (0x06)
890 #define MPI2_EVENT_SAS_TOPO_LR_RATE_1_5 (0x08)
891 #define MPI2_EVENT_SAS_TOPO_LR_RATE_3_0 (0x09)
892 #define MPI2_EVENT_SAS_TOPO_LR_RATE_6_0 (0x0A)
893 #define MPI2_EVENT_SAS_TOPO_LR_RATE_12_0 (0x0B)

895 /* values for the PhyStatus field */
896 #define MPI2_EVENT_SAS_TOPO_PHYSTATUS_VACANT (0x80)
897 #define MPI2_EVENT_SAS_TOPO_PS_MULTIPLEX_CHANGE (0x10)
898 /* values for the PhyStatus ReasonCode sub-field */
899 #define MPI2_EVENT_SAS_TOPO_RC_MASK (0x0F)
900 #define MPI2_EVENT_SAS_TOPO_RC_TARG_ADDED (0x01)
901 #define MPI2_EVENT_SAS_TOPO_RC_TARG_NOT_RESPONDING (0x02)
902 #define MPI2_EVENT_SAS_TOPO_RC_PHY_CHANGED (0x03)
903 #define MPI2_EVENT_SAS_TOPO_RC_NO_CHANGE (0x04)
904 #define MPI2_EVENT_SAS_TOPO_RC_DELAY_NOT_RESPONDING (0x05)

907 /* SAS Enclosure Device Status Change Event data */

909 typedef struct _MPI2_EVENT_DATA_SAS_ENCL_DEV_STATUS_CHANGE
910 {
911     U16 EnclosureHandle; /* 0x00 */
912     U8 ReasonCode; /* 0x02 */
913     U8 PhysicalPort; /* 0x03 */
914     U64 EnclosureLogicalID; /* 0x04 */
915     U16 NumSlots; /* 0x0C */
916     U16 StartSlot; /* 0x0E */
917     U32 PhyBits; /* 0x10 */
918 } MPI2_EVENT_DATA_SAS_ENCL_DEV_STATUS_CHANGE,
919 MPI2_POINTER PTR_MPI2_EVENT_DATA_SAS_ENCL_DEV_STATUS_CHANGE,

```

```

920  Mpi2EventDataSasEnclDevStatusChange_t,
921  MPI2_POINTER pMpi2EventDataSasEnclDevStatusChange_t;

923 /* SAS Enclosure Device Status Change event ReasonCode values */
924 #define MPI2_EVENT_SAS_ENCL_RC_ADDED          (0x01)
925 #define MPI2_EVENT_SAS_ENCL_RC_NOT_RESPONDING (0x02)

928 /* SAS PHY Counter Event data */

930 typedef struct _MPI2_EVENT_DATA_SAS_PHY_COUNTER
931 {
932     U64      TimeStamp;          /* 0x00 */
933     U32      Reserved1;          /* 0x08 */
934     U8       PhyEventCode;       /* 0x0C */
935     U8       PhyNum;             /* 0x0D */
936     U16      Reserved2;          /* 0x0E */
937     U32      PhyEventInfo;       /* 0x10 */
938     U8       CounterType;        /* 0x14 */
939     U8       ThresholdWindow;    /* 0x15 */
940     U8       TimeUnits;          /* 0x16 */
941     U8       Reserved3;          /* 0x17 */
942     U32      EventThreshold;     /* 0x18 */
943     U16      ThresholdFlags;     /* 0x1C */
944     U16      Reserved4;          /* 0x1E */
945 } MPI2_EVENT_DATA_SAS_PHY_COUNTER,
946 MPI2_POINTER PTR_MPI2_EVENT_DATA_SAS_PHY_COUNTER,
947 Mpi2EventDataSasPhyCounter_t, MPI2_POINTER pMpi2EventDataSasPhyCounter_t;

949 /* use MPI2_SASPHY3_EVENT_CODE values from mpi2_cnfg.h for the PhyEventCode fie

951 /* use MPI2_SASPHY3_COUNTER_TYPE values from mpi2_cnfg.h for the CounterType fi

953 /* use MPI2_SASPHY3_TIME_UNITS values from mpi2_cnfg.h for the TimeUnits field

955 /* use MPI2_SASPHY3_TFLAGS values from mpi2_cnfg.h for the ThresholdFlags field

958 /* SAS Quiesce Event data */

960 typedef struct _MPI2_EVENT_DATA_SAS_QUIESCE
961 {
962     U8       ReasonCode;          /* 0x00 */
963     U8       Reserved1;           /* 0x01 */
964     U16      Reserved2;           /* 0x02 */
965     U32      Reserved3;           /* 0x04 */
966 } MPI2_EVENT_DATA_SAS_QUIESCE,
967 MPI2_POINTER PTR_MPI2_EVENT_DATA_SAS_QUIESCE,
968 Mpi2EventDataSasQuiesce_t, MPI2_POINTER pMpi2EventDataSasQuiesce_t;

970 /* SAS Quiesce Event data ReasonCode values */
971 #define MPI2_EVENT_SAS_QUIESCE_RC_STARTED      (0x01)
972 #define MPI2_EVENT_SAS_QUIESCE_RC_COMPLETED    (0x02)

975 /* Host Based Discovery Phy Event data */

977 typedef struct _MPI2_EVENT_HBD_PHY_SAS
978 {
979     U8       Flags;               /* 0x00 */
980     U8       NegotiatedLinkRate;  /* 0x01 */
981     U8       PhyNum;              /* 0x02 */
982     U8       PhysicalPort;        /* 0x03 */
983     U32      Reserved1;           /* 0x04 */
984     U8       InitialFrame[28];    /* 0x08 */
985 } MPI2_EVENT_HBD_PHY_SAS, MPI2_POINTER PTR_MPI2_EVENT_HBD_PHY_SAS,

```

```

986  Mpi2EventHbdPhySas_t, MPI2_POINTER pMpi2EventHbdPhySas_t;

988 /* values for the Flags field */
989 #define MPI2_EVENT_HBD_SAS_FLAGS_FRAME_VALID    (0x02)
990 #define MPI2_EVENT_HBD_SAS_FLAGS_SATA_FRAME     (0x01)

992 /* use MPI2_SAS_NEG_LINK_RATE defines from mpi2_cnfg.h for the NegotiatedLinkRa

994 typedef union _MPI2_EVENT_HBD_DESCRIPTOR
995 {
996     MPI2_EVENT_HBD_PHY_SAS      Sas;
997 } MPI2_EVENT_HBD_DESCRIPTOR, MPI2_POINTER PTR_MPI2_EVENT_HBD_DESCRIPTOR,
998 Mpi2EventHbdDescriptor_t, MPI2_POINTER pMpi2EventHbdDescriptor_t;

1000 typedef struct _MPI2_EVENT_DATA_HBD_PHY
1001 {
1002     U8       DescriptorType;      /* 0x00 */
1003     U8       Reserved1;           /* 0x01 */
1004     U16      Reserved2;           /* 0x02 */
1005     U32      Reserved3;           /* 0x04 */
1006     MPI2_EVENT_HBD_DESCRIPTOR    Descriptor; /* 0x08 */
1007 } MPI2_EVENT_DATA_HBD_PHY, MPI2_POINTER PTR_MPI2_EVENT_DATA_HBD_PHY,
1008 Mpi2EventDataHbdPhy_t, MPI2_POINTER pMpi2EventDataMpi2EventDataHbdPhy_t;

1010 /* values for the DescriptorType field */
1011 #define MPI2_EVENT_HBD_DT_SAS      (0x01)

1015 /*****
1016  * EventAck message
1017  *****/

1019 /* EventAck Request message */
1020 typedef struct _MPI2_EVENT_ACK_REQUEST
1021 {
1022     U16      Reserved1;           /* 0x00 */
1023     U8       ChainOffset;         /* 0x02 */
1024     U8       Function;            /* 0x03 */
1025     U16      Reserved2;           /* 0x04 */
1026     U8       Reserved3;           /* 0x06 */
1027     U8       MsgFlags;            /* 0x07 */
1028     U8       VP_ID;               /* 0x08 */
1029     U8       VF_ID;               /* 0x09 */
1030     U16      Reserved4;           /* 0x0A */
1031     U16      Event;               /* 0x0C */
1032     U16      Reserved5;           /* 0x0E */
1033     U32      EventContext;        /* 0x10 */
1034 } MPI2_EVENT_ACK_REQUEST, MPI2_POINTER PTR_MPI2_EVENT_ACK_REQUEST,
1035 Mpi2EventAckRequest_t, MPI2_POINTER pMpi2EventAckRequest_t;

1038 /* EventAck Reply message */
1039 typedef struct _MPI2_EVENT_ACK_REPLY
1040 {
1041     U16      Reserved1;           /* 0x00 */
1042     U8       MsgLength;           /* 0x02 */
1043     U8       Function;            /* 0x03 */
1044     U16      Reserved2;           /* 0x04 */
1045     U8       Reserved3;           /* 0x06 */
1046     U8       MsgFlags;            /* 0x07 */
1047     U8       VP_ID;               /* 0x08 */
1048     U8       VF_ID;               /* 0x09 */
1049     U16      Reserved4;           /* 0x0A */
1050     U16      Reserved5;           /* 0x0C */
1051     U16      IOCStatus;           /* 0x0E */

```

```

1052     U32                IOCLogInfo;                /* 0x10 */
1053 } MPI2_EVENT_ACK_REPLY, MPI2_POINTER PTR_MPI2_EVENT_ACK_REPLY,
1054   Mpi2EventAckReply_t, MPI2_POINTER pMpi2EventAckReply_t;

1057 /*****
1058  * FWDdownload message
1059 *****/

1061 /* MPI v2.0 FWDdownload Request message */
1062 typedef struct _MPI2_FW_DOWNLOAD_REQUEST
1063 {
1064     U8                ImageType;                /* 0x00 */
1065     U8                Reserved1;                /* 0x01 */
1066     U8                ChainOffset;             /* 0x02 */
1067     U8                Function;                /* 0x03 */
1068     U16               Reserved2;                /* 0x04 */
1069     U8                Reserved3;                /* 0x06 */
1070     U8                MsgFlags;                /* 0x07 */
1071     U8                VP_ID;                   /* 0x08 */
1072     U8                VF_ID;                   /* 0x09 */
1073     U16               Reserved4;                /* 0x0A */
1074     U32               TotalImageSize;          /* 0x0C */
1075     U32               Reserved5;                /* 0x10 */
1076     MPI2_MPI_SGE_UNION SGL;                    /* 0x14 */
1077 } MPI2_FW_DOWNLOAD_REQUEST, MPI2_POINTER PTR_MPI2_FW_DOWNLOAD_REQUEST,
1078   Mpi2FWDownloadRequest, MPI2_POINTER pMpi2FWDownloadRequest;

1080 #define MPI2_FW_DOWNLOAD_MSGFLGS_LAST_SEGMENT    (0x01)

1082 #define MPI2_FW_DOWNLOAD_ITYPE_FW                (0x01)
1083 #define MPI2_FW_DOWNLOAD_ITYPE_BIOS              (0x02)
1084 #define MPI2_FW_DOWNLOAD_ITYPE_MANUFACTURING     (0x06)
1085 #define MPI2_FW_DOWNLOAD_ITYPE_CONFIG_1          (0x07)
1086 #define MPI2_FW_DOWNLOAD_ITYPE_CONFIG_2          (0x08)
1087 #define MPI2_FW_DOWNLOAD_ITYPE_MEGARAID          (0x09)
1088 #define MPI2_FW_DOWNLOAD_ITYPE_COMPLETE          (0x0A)
1089 #define MPI2_FW_DOWNLOAD_ITYPE_COMMON_BOOT_BLOCK (0x0B)
1090 #define MPI2_FW_DOWNLOAD_ITYPE_MIN_PRODUCT_SPECIFIC (0x0F)

1092 /* MPI v2.0 FWDdownload TransactionContext Element */
1093 typedef struct _MPI2_FW_DOWNLOAD_TCSGE
1094 {
1095     U8                Reserved1;                /* 0x00 */
1096     U8                ContextSize;              /* 0x01 */
1097     U8                DetailsLength;            /* 0x02 */
1098     U8                Flags;                    /* 0x03 */
1099     U32               Reserved2;                /* 0x04 */
1100     U32               ImageOffset;              /* 0x08 */
1101     U32               ImageSize;                /* 0x0C */
1102 } MPI2_FW_DOWNLOAD_TCSGE, MPI2_POINTER PTR_MPI2_FW_DOWNLOAD_TCSGE,
1103   Mpi2FWDownloadTCSGE_t, MPI2_POINTER pMpi2FWDownloadTCSGE_t;

1106 /* MPI v2.5 FWDdownload Request message */
1107 typedef struct _MPI25_FW_DOWNLOAD_REQUEST
1108 {
1109     U8                ImageType;                /* 0x00 */
1110     U8                Reserved1;                /* 0x01 */
1111     U8                ChainOffset;             /* 0x02 */
1112     U8                Function;                /* 0x03 */
1113     U16               Reserved2;                /* 0x04 */
1114     U8                Reserved3;                /* 0x06 */
1115     U8                MsgFlags;                /* 0x07 */
1116     U8                VP_ID;                   /* 0x08 */
1117     U8                VF_ID;                   /* 0x09 */

```

```

1118     U16               Reserved4;                /* 0x0A */
1119     U32               TotalImageSize;          /* 0x0C */
1120     U32               Reserved5;                /* 0x10 */
1121     U32               Reserved6;                /* 0x14 */
1122     U32               ImageOffset;             /* 0x18 */
1123     U32               ImageSize;                /* 0x1C */
1124     MPI25_SGE_IO_UNION SGL;                    /* 0x20 */
1125 } MPI25_FW_DOWNLOAD_REQUEST, MPI2_POINTER PTR_MPI25_FW_DOWNLOAD_REQUEST,
1126   Mpi25FWDownloadRequest, MPI2_POINTER pMpi25FWDownloadRequest;

1129 /* FWDdownload Reply message */
1130 typedef struct _MPI2_FW_DOWNLOAD_REPLY
1131 {
1132     U8                ImageType;                /* 0x00 */
1133     U8                Reserved1;                /* 0x01 */
1134     U8                MsgLength;                /* 0x02 */
1135     U8                Function;                /* 0x03 */
1136     U16               Reserved2;                /* 0x04 */
1137     U8                Reserved3;                /* 0x06 */
1138     U8                MsgFlags;                /* 0x07 */
1139     U8                VP_ID;                   /* 0x08 */
1140     U8                VF_ID;                   /* 0x09 */
1141     U16               Reserved4;                /* 0x0A */
1142     U16               Reserved5;                /* 0x0C */
1143     U16               IOCStatus;                /* 0x0E */
1144     U32               IOCLogInfo;              /* 0x10 */
1145 } MPI2_FW_DOWNLOAD_REPLY, MPI2_POINTER PTR_MPI2_FW_DOWNLOAD_REPLY,
1146   Mpi2FWDownloadReply_t, MPI2_POINTER pMpi2FWDownloadReply_t;

1149 /*****
1150  * FWUpload message
1151 *****/

1153 /* MPI v2.0 FWUpload Request message */
1154 typedef struct _MPI2_FW_UPLOAD_REQUEST
1155 {
1156     U8                ImageType;                /* 0x00 */
1157     U8                Reserved1;                /* 0x01 */
1158     U8                ChainOffset;             /* 0x02 */
1159     U8                Function;                /* 0x03 */
1160     U16               Reserved2;                /* 0x04 */
1161     U8                Reserved3;                /* 0x06 */
1162     U8                MsgFlags;                /* 0x07 */
1163     U8                VP_ID;                   /* 0x08 */
1164     U8                VF_ID;                   /* 0x09 */
1165     U16               Reserved4;                /* 0x0A */
1166     U32               Reserved5;                /* 0x0C */
1167     U32               Reserved6;                /* 0x10 */
1168     MPI2_MPI_SGE_UNION SGL;                    /* 0x14 */
1169 } MPI2_FW_UPLOAD_REQUEST, MPI2_POINTER PTR_MPI2_FW_UPLOAD_REQUEST,
1170   Mpi2FWUploadRequest_t, MPI2_POINTER pMpi2FWUploadRequest_t;

1172 #define MPI2_FW_UPLOAD_ITYPE_FW_CURRENT          (0x00)
1173 #define MPI2_FW_UPLOAD_ITYPE_FW_FLASH            (0x01)
1174 #define MPI2_FW_UPLOAD_ITYPE_BIOS_FLASH          (0x02)
1175 #define MPI2_FW_UPLOAD_ITYPE_FW_BACKUP           (0x05)
1176 #define MPI2_FW_UPLOAD_ITYPE_MANUFACTURING       (0x06)
1177 #define MPI2_FW_UPLOAD_ITYPE_CONFIG_1            (0x07)
1178 #define MPI2_FW_UPLOAD_ITYPE_CONFIG_2            (0x08)
1179 #define MPI2_FW_UPLOAD_ITYPE_MEGARAID            (0x09)
1180 #define MPI2_FW_UPLOAD_ITYPE_COMPLETE            (0x0A)
1181 #define MPI2_FW_UPLOAD_ITYPE_COMMON_BOOT_BLOCK   (0x0B)

1183 /* MPI v2.0 FWUpload TransactionContext Element */

```

```

1184 typedef struct _MPI2_FW_UPLOAD_TCSGE
1185 {
1186     U8      Reserved1;          /* 0x00 */
1187     U8      ContextSize;        /* 0x01 */
1188     U8      DetailsLength;      /* 0x02 */
1189     U8      Flags;              /* 0x03 */
1190     U32     Reserved2;          /* 0x04 */
1191     U32     ImageOffset;        /* 0x08 */
1192     U32     ImageSize;          /* 0x0C */
1193 } MPI2_FW_UPLOAD_TCSGE, MPI2_POINTER PTR_MPI2_FW_UPLOAD_TCSGE,
1194   Mpi2FWUploadTCSGE_t, MPI2_POINTER pMpi2FWUploadTCSGE_t;

1197 /* MPI v2.5 FWUpload Request message */
1198 typedef struct _MPI25_FW_UPLOAD_REQUEST
1199 {
1200     U8      ImageType;          /* 0x00 */
1201     U8      Reserved1;          /* 0x01 */
1202     U8      ChainOffset;        /* 0x02 */
1203     U8      Function;           /* 0x03 */
1204     U16     Reserved2;          /* 0x04 */
1205     U8      Reserved3;          /* 0x06 */
1206     U8      MsgFlags;           /* 0x07 */
1207     U8      VP_ID;              /* 0x08 */
1208     U8      VF_ID;              /* 0x09 */
1209     U16     Reserved4;          /* 0x0A */
1210     U32     Reserved5;          /* 0x0C */
1211     U32     Reserved6;          /* 0x10 */
1212     U32     Reserved7;          /* 0x14 */
1213     U32     ImageOffset;        /* 0x18 */
1214     U32     ImageSize;          /* 0x1C */
1215     MPI25_SGE_IO_UNION          /* 0x20 */
1216 } MPI25_FW_UPLOAD_REQUEST, MPI2_POINTER PTR_MPI25_FW_UPLOAD_REQUEST,
1217   Mpi25FWUploadRequest_t, MPI2_POINTER pMpi25FWUploadRequest_t;

1220 /* FWUpload Reply message */
1221 typedef struct _MPI2_FW_UPLOAD_REPLY
1222 {
1223     U8      ImageType;          /* 0x00 */
1224     U8      Reserved1;          /* 0x01 */
1225     U8      MsgLength;          /* 0x02 */
1226     U8      Function;           /* 0x03 */
1227     U16     Reserved2;          /* 0x04 */
1228     U8      Reserved3;          /* 0x06 */
1229     U8      MsgFlags;           /* 0x07 */
1230     U8      VP_ID;              /* 0x08 */
1231     U8      VF_ID;              /* 0x09 */
1232     U16     Reserved4;          /* 0x0A */
1233     U16     Reserved5;          /* 0x0C */
1234     U16     IOCStatus;          /* 0x0E */
1235     U32     IOCLogInfo;         /* 0x10 */
1236     U32     ActualImageSize;    /* 0x14 */
1237 } MPI2_FW_UPLOAD_REPLY, MPI2_POINTER PTR_MPI2_FW_UPLOAD_REPLY,
1238   Mpi2FWUploadReply_t, MPI2_POINTER pMpi2FWUploadReply_t;

1241 /* FW Image Header */
1242 typedef struct _MPI2_FW_IMAGE_HEADER
1243 {
1244     U32     Signature;          /* 0x00 */
1245     U32     Signature0;         /* 0x04 */
1246     U32     Signature1;         /* 0x08 */
1247     U32     Signature2;         /* 0x0C */
1248     MPI2_VERSION_UNION          /* 0x10 */
1249     MPI2_VERSION_UNION          /* 0x14 */

```

```

1250     MPI2_VERSION_UNION          /* 0x18 */
1251     MPI2_VERSION_UNION          /* 0x1C */
1252     U16     VendorID;           /* 0x20 */
1253     U16     ProductID;          /* 0x22 */
1254     U16     ProtocolFlags;      /* 0x24 */
1255     U16     Reserved26;         /* 0x26 */
1256     U32     IOCCapabilities;    /* 0x28 */
1257     U32     ImageSize;          /* 0x2C */
1258     U32     NextImageHeaderOffset; /* 0x30 */
1259     U32     Checksum;           /* 0x34 */
1260     U32     Reserved38;         /* 0x38 */
1261     U32     Reserved3C;         /* 0x3C */
1262     U32     Reserved40;         /* 0x40 */
1263     U32     Reserved44;         /* 0x44 */
1264     U32     Reserved48;         /* 0x48 */
1265     U32     Reserved4C;         /* 0x4C */
1266     U32     Reserved50;         /* 0x50 */
1267     U32     Reserved54;         /* 0x54 */
1268     U32     Reserved58;         /* 0x58 */
1269     U32     Reserved5C;         /* 0x5C */
1270     U32     Reserved60;         /* 0x60 */
1271     U32     FirmwareVersionNameWhat; /* 0x64 */
1272     U8      FirmwareVersionName[32]; /* 0x68 */
1273     U32     VendorNameWhat;      /* 0x88 */
1274     U8      VendorName[32];      /* 0x8C */
1275     U32     PackageNameWhat;     /* 0x88 */
1276     U8      PackageName[32];     /* 0x8C */
1277     U32     ReservedD0;          /* 0xD0 */
1278     U32     ReservedD4;          /* 0xD4 */
1279     U32     ReservedD8;          /* 0xD8 */
1280     U32     ReservedDC;          /* 0xDC */
1281     U32     ReservedE0;          /* 0xE0 */
1282     U32     ReservedE4;          /* 0xE4 */
1283     U32     ReservedE8;          /* 0xE8 */
1284     U32     ReservedEC;          /* 0xEC */
1285     U32     ReservedF0;          /* 0xF0 */
1286     U32     ReservedF4;          /* 0xF4 */
1287     U32     ReservedF8;          /* 0xF8 */
1288     U32     ReservedFC;          /* 0xFC */
1289 } MPI2_FW_IMAGE_HEADER, MPI2_POINTER PTR_MPI2_FW_IMAGE_HEADER,
1290   Mpi2FWImageHeader_t, MPI2_POINTER pMpi2FWImageHeader_t;

1292 /* Signature field */
1293 #define MPI2_FW_HEADER_SIGNATURE_OFFSET      (0x00)
1294 #define MPI2_FW_HEADER_SIGNATURE_MASK       (0xFF000000)
1295 #define MPI2_FW_HEADER_SIGNATURE           (0xEA000000)

1297 /* Signature0 field */
1298 #define MPI2_FW_HEADER_SIGNATURE0_OFFSET     (0x04)
1299 #define MPI2_FW_HEADER_SIGNATURE0           (0x5AFAA55A)

1301 /* Signature1 field */
1302 #define MPI2_FW_HEADER_SIGNATURE1_OFFSET     (0x08)
1303 #define MPI2_FW_HEADER_SIGNATURE1           (0xA55AFAA5)

1305 /* Signature2 field */
1306 #define MPI2_FW_HEADER_SIGNATURE2_OFFSET     (0x0C)
1307 #define MPI2_FW_HEADER_SIGNATURE2           (0x5AA55AFA)

1310 /* defines for using the ProductID field */
1311 #define MPI2_FW_HEADER_PID_TYPE_MASK         (0xF000)
1312 #define MPI2_FW_HEADER_PID_TYPE_SAS         (0x2000)

1314 #define MPI2_FW_HEADER_PID_PROD_MASK        (0xF00)
1315 #define MPI2_FW_HEADER_PID_PROD_A          (0x000)

```

```

1316 #define MPI2_FW_HEADER_PID_PROD_TARGET_INITIATOR_SCSI (0x0200)
1317 #define MPI2_FW_HEADER_PID_PROD_IR_SCSI (0x0700)

1319 #define MPI2_FW_HEADER_PID_FAMILY_MASK (0x00FF)
1320 /* SAS ProductID Family bits */
1321 #define MPI2_FW_HEADER_PID_FAMILY_2108_SAS (0x0013)
1322 #define MPI2_FW_HEADER_PID_FAMILY_2208_SAS (0x0014)
1323 #define MPI25_FW_HEADER_PID_FAMILY_3108_SAS (0x0021)

1325 /* use MPI2_IOCFACTS_PROTOCOL_ defines for ProtocolFlags field */

1327 /* use MPI2_IOCFACTS_CAPABILITY_ defines for IOCCapabilities field */

1330 #define MPI2_FW_HEADER_IMAGESIZE_OFFSET (0x2C)
1331 #define MPI2_FW_HEADER_NEXTIMAGE_OFFSET (0x30)
1332 #define MPI2_FW_HEADER_VERNMHWAT_OFFSET (0x64)

1334 #define MPI2_FW_HEADER_WHAT_SIGNATURE (0x29232840)

1336 #define MPI2_FW_HEADER_SIZE (0x100)

1339 /* Extended Image Header */
1340 typedef struct _MPI2_EXT_IMAGE_HEADER

1342 {
1343     U8 ImageType; /* 0x00 */
1344     U8 Reserved1; /* 0x01 */
1345     U16 Reserved2; /* 0x02 */
1346     U32 Checksum; /* 0x04 */
1347     U32 ImageSize; /* 0x08 */
1348     U32 NextImageHeaderOffset; /* 0x0C */
1349     U32 PackageVersion; /* 0x10 */
1350     U32 Reserved3; /* 0x14 */
1351     U32 Reserved4; /* 0x18 */
1352     U32 Reserved5; /* 0x1C */
1353     U8 IdentifyString[32]; /* 0x20 */
1354 } MPI2_EXT_IMAGE_HEADER, MPI2_POINTER PTR_MPI2_EXT_IMAGE_HEADER,
1355 Mpi2ExtImageHeader_t, MPI2_POINTER pMpi2ExtImageHeader_t;

1357 /* useful offsets */
1358 #define MPI2_EXT_IMAGE_IMAGETYPE_OFFSET (0x00)
1359 #define MPI2_EXT_IMAGE_IMAGESIZE_OFFSET (0x08)
1360 #define MPI2_EXT_IMAGE_NEXTIMAGE_OFFSET (0x0C)

1362 #define MPI2_EXT_IMAGE_HEADER_SIZE (0x40)

1364 /* defines for the ImageType field */
1365 #define MPI2_EXT_IMAGE_TYPE_UNSPECIFIED (0x00)
1366 #define MPI2_EXT_IMAGE_TYPE_FW (0x01)
1367 #define MPI2_EXT_IMAGE_TYPE_NVDATA (0x03)
1368 #define MPI2_EXT_IMAGE_TYPE_BOOTLOADER (0x04)
1369 #define MPI2_EXT_IMAGE_TYPE_INITIALIZATION (0x05)
1370 #define MPI2_EXT_IMAGE_TYPE_FLASH_LAYOUT (0x06)
1371 #define MPI2_EXT_IMAGE_TYPE_SUPPORTED_DEVICES (0x07)
1372 #define MPI2_EXT_IMAGE_TYPE_MEGARAID (0x08)

1374 #define MPI2_EXT_IMAGE_TYPE_MAX (MPI2_EXT_IMAGE_TYPE_MEGARAID)

1378 /* FLASH Layout Extended Image Data */

1380 /*
1381  * Host code (drivers, BIOS, utilities, etc.) should leave this define set to

```

```

1382 * one and check RegionsPerLayout at runtime.
1383 */
1384 #ifndef MPI2_FLASH_NUMBER_OF_REGIONS
1385 #define MPI2_FLASH_NUMBER_OF_REGIONS (1)
1386 #endif

1388 /*
1389  * Host code (drivers, BIOS, utilities, etc.) should leave this define set to
1390  * one and check NumberOfLayouts at runtime.
1391 */
1392 #ifndef MPI2_FLASH_NUMBER_OF_LAYOUTS
1393 #define MPI2_FLASH_NUMBER_OF_LAYOUTS (1)
1394 #endif

1396 typedef struct _MPI2_FLASH_REGION
1397 {
1398     U8 RegionType; /* 0x00 */
1399     U8 Reserved1; /* 0x01 */
1400     U16 Reserved2; /* 0x02 */
1401     U32 RegionOffset; /* 0x04 */
1402     U32 RegionSize; /* 0x08 */
1403     U32 Reserved3; /* 0x0C */
1404 } MPI2_FLASH_REGION, MPI2_POINTER PTR_MPI2_FLASH_REGION,
1405 Mpi2FlashRegion_t, MPI2_POINTER pMpi2FlashRegion_t;

1407 typedef struct _MPI2_FLASH_LAYOUT
1408 {
1409     U32 FlashSize; /* 0x00 */
1410     U32 Reserved1; /* 0x04 */
1411     U32 Reserved2; /* 0x08 */
1412     U32 Reserved3; /* 0x0C */
1413     MPI2_FLASH_REGION Region[MPI2_FLASH_NUMBER_OF_REGIONS]; /* 0x10 */
1414 } MPI2_FLASH_LAYOUT, MPI2_POINTER PTR_MPI2_FLASH_LAYOUT,
1415 Mpi2FlashLayout_t, MPI2_POINTER pMpi2FlashLayout_t;

1417 typedef struct _MPI2_FLASH_LAYOUT_DATA
1418 {
1419     U8 ImageRevision; /* 0x00 */
1420     U8 Reserved1; /* 0x01 */
1421     U8 SizeOfRegion; /* 0x02 */
1422     U8 Reserved2; /* 0x03 */
1423     U16 NumberOfLayouts; /* 0x04 */
1424     U16 RegionsPerLayout; /* 0x06 */
1425     U16 MinimumSectorAlignment; /* 0x08 */
1426     U16 Reserved3; /* 0x0A */
1427     U32 Reserved4; /* 0x0C */
1428     MPI2_FLASH_LAYOUT Layout[MPI2_FLASH_NUMBER_OF_LAYOUTS]; /* 0x10 */
1429 } MPI2_FLASH_LAYOUT_DATA, MPI2_POINTER PTR_MPI2_FLASH_LAYOUT_DATA,
1430 Mpi2FlashLayoutData_t, MPI2_POINTER pMpi2FlashLayoutData_t;

1432 /* defines for the RegionType field */
1433 #define MPI2_FLASH_REGION_UNUSED (0x00)
1434 #define MPI2_FLASH_REGION_FIRMWARE (0x01)
1435 #define MPI2_FLASH_REGION_BIOS (0x02)
1436 #define MPI2_FLASH_REGION_NVDATA (0x03)
1437 #define MPI2_FLASH_REGION_FIRMWARE_BACKUP (0x05)
1438 #define MPI2_FLASH_REGION_MFG_INFORMATION (0x06)
1439 #define MPI2_FLASH_REGION_CONFIG_1 (0x07)
1440 #define MPI2_FLASH_REGION_CONFIG_2 (0x08)
1441 #define MPI2_FLASH_REGION_MEGARAID (0x09)
1442 #define MPI2_FLASH_REGION_INIT (0x0A)

1444 /* ImageRevision */
1445 #define MPI2_FLASH_LAYOUT_IMAGE_REVISION (0x00)

```

```

1449 /* Supported Devices Extended Image Data */
1451 /*
1452  * Host code (drivers, BIOS, utilities, etc.) should leave this define set to
1453  * one and check NumberOfDevices at runtime.
1454  */
1455 #ifndef MPI2_SUPPORTED_DEVICES_IMAGE_NUM_DEVICES
1456 #define MPI2_SUPPORTED_DEVICES_IMAGE_NUM_DEVICES (1)
1457 #endif
1459 typedef struct _MPI2_SUPPORTED_DEVICE
1460 {
1461     U16 DeviceID; /* 0x00 */
1462     U16 VendorID; /* 0x02 */
1463     U16 DeviceIDMask; /* 0x04 */
1464     U16 Reserved1; /* 0x06 */
1465     U8 LowPCIRev; /* 0x08 */
1466     U8 HighPCIRev; /* 0x09 */
1467     U16 Reserved2; /* 0x0A */
1468     U32 Reserved3; /* 0x0C */
1469 } MPI2_SUPPORTED_DEVICE, MPI2_POINTER PTR_MPI2_SUPPORTED_DEVICE,
1470 Mpi2SupportedDevice_t, MPI2_POINTER pMpi2SupportedDevice_t;
1472 typedef struct _MPI2_SUPPORTED_DEVICES_DATA
1473 {
1474     U8 ImageRevision; /* 0x00 */
1475     U8 Reserved1; /* 0x01 */
1476     U8 NumberOfDevices; /* 0x02 */
1477     U8 Reserved2; /* 0x03 */
1478     U32 Reserved3; /* 0x04 */
1479     MPI2_SUPPORTED_DEVICE SupportedDevice[MPI2_SUPPORTED_DEVICES_IMAGE_NUM_DEV
1480 } MPI2_SUPPORTED_DEVICES_DATA, MPI2_POINTER PTR_MPI2_SUPPORTED_DEVICES_DATA,
1481 Mpi2SupportedDevicesData_t, MPI2_POINTER pMpi2SupportedDevicesData_t;
1483 /* ImageRevision */
1484 #define MPI2_SUPPORTED_DEVICES_IMAGE_REVISION (0x00)
1487 /* Init Extended Image Data */
1489 typedef struct _MPI2_INIT_IMAGE_FOOTER
1491 {
1492     U32 BootFlags; /* 0x00 */
1493     U32 ImageSize; /* 0x04 */
1494     U32 Signature0; /* 0x08 */
1495     U32 Signature1; /* 0x0C */
1496     U32 Signature2; /* 0x10 */
1497     U32 ResetVector; /* 0x14 */
1498 } MPI2_INIT_IMAGE_FOOTER, MPI2_POINTER PTR_MPI2_INIT_IMAGE_FOOTER,
1499 Mpi2InitImageFooter_t, MPI2_POINTER pMpi2InitImageFooter_t;
1501 /* defines for the BootFlags field */
1502 #define MPI2_INIT_IMAGE_BOOTFLAGS_OFFSET (0x00)
1504 /* defines for the ImageSize field */
1505 #define MPI2_INIT_IMAGE_IMAGESIZE_OFFSET (0x04)
1507 /* defines for the Signature0 field */
1508 #define MPI2_INIT_IMAGE_SIGNATURE0_OFFSET (0x08)
1509 #define MPI2_INIT_IMAGE_SIGNATURE0 (0x5AA5AEA)
1511 /* defines for the Signature1 field */
1512 #define MPI2_INIT_IMAGE_SIGNATURE1_OFFSET (0x0C)
1513 #define MPI2_INIT_IMAGE_SIGNATURE1 (0xA5A5AEA5)

```

```

1515 /* defines for the Signature2 field */
1516 #define MPI2_INIT_IMAGE_SIGNATURE2_OFFSET (0x10)
1517 #define MPI2_INIT_IMAGE_SIGNATURE2 (0x5AA5A5A)
1519 /* Signature fields as individual bytes */
1520 #define MPI2_INIT_IMAGE_SIGNATURE_BYTE_0 (0xEA)
1521 #define MPI2_INIT_IMAGE_SIGNATURE_BYTE_1 (0x5A)
1522 #define MPI2_INIT_IMAGE_SIGNATURE_BYTE_2 (0xA5)
1523 #define MPI2_INIT_IMAGE_SIGNATURE_BYTE_3 (0x5A)
1525 #define MPI2_INIT_IMAGE_SIGNATURE_BYTE_4 (0xA5)
1526 #define MPI2_INIT_IMAGE_SIGNATURE_BYTE_5 (0xEA)
1527 #define MPI2_INIT_IMAGE_SIGNATURE_BYTE_6 (0x5A)
1528 #define MPI2_INIT_IMAGE_SIGNATURE_BYTE_7 (0xA5)
1530 #define MPI2_INIT_IMAGE_SIGNATURE_BYTE_8 (0x5A)
1531 #define MPI2_INIT_IMAGE_SIGNATURE_BYTE_9 (0xA5)
1532 #define MPI2_INIT_IMAGE_SIGNATURE_BYTE_A (0xEA)
1533 #define MPI2_INIT_IMAGE_SIGNATURE_BYTE_B (0x5A)
1535 /* defines for the ResetVector field */
1536 #define MPI2_INIT_IMAGE_RESETVECTOR_OFFSET (0x14)
1539 /*****
1540  * PowerManagementControl message
1541  *****/
1543 /* PowerManagementControl Request message */
1544 typedef struct _MPI2_PWR_MGMT_CONTROL_REQUEST
1545 {
1546     U8 Feature; /* 0x00 */
1547     U8 Reserved1; /* 0x01 */
1548     U8 ChainOffset; /* 0x02 */
1549     U8 Function; /* 0x03 */
1550     U16 Reserved2; /* 0x04 */
1551     U8 Reserved3; /* 0x06 */
1552     U8 MsgFlags; /* 0x07 */
1553     U8 VP_ID; /* 0x08 */
1554     U8 VF_ID; /* 0x09 */
1555     U16 Reserved4; /* 0x0A */
1556     U8 Parameter1; /* 0x0C */
1557     U8 Parameter2; /* 0x0D */
1558     U8 Parameter3; /* 0x0E */
1559     U8 Parameter4; /* 0x0F */
1560     U32 Reserved5; /* 0x10 */
1561     U32 Reserved6; /* 0x14 */
1562 } MPI2_PWR_MGMT_CONTROL_REQUEST, MPI2_POINTER PTR_MPI2_PWR_MGMT_CONTROL_REQUEST,
1563 Mpi2PwrMgmtControlRequest_t, MPI2_POINTER pMpi2PwrMgmtControlRequest_t;
1565 /* defines for the Feature field */
1566 #define MPI2_PM_CONTROL_FEATURE_DA_PHY_POWER_COND (0x01)
1567 #define MPI2_PM_CONTROL_FEATURE_PORT_WIDTH_MODULATION (0x02)
1568 #define MPI2_PM_CONTROL_FEATURE_PCIE_LINK (0x03)
1569 #define MPI2_PM_CONTROL_FEATURE_IOC_SPEED (0x04)
1570 #define MPI2_PM_CONTROL_FEATURE_MIN_PRODUCT_SPECIFIC (0x80)
1571 #define MPI2_PM_CONTROL_FEATURE_MAX_PRODUCT_SPECIFIC (0xFF)
1573 /* parameter usage for the MPI2_PM_CONTROL_FEATURE_DA_PHY_POWER_COND Feature */
1574 /* Parameter1 contains a PHY number */
1575 /* Parameter2 indicates power condition action using these defines */
1576 #define MPI2_PM_CONTROL_PARAM2_PARTIAL (0x01)
1577 #define MPI2_PM_CONTROL_PARAM2_SLUMBER (0x02)
1578 #define MPI2_PM_CONTROL_PARAM2_EXIT_PWR_MGMT (0x03)
1579 /* Parameter3 and Parameter4 are reserved */

```



```
1581 /* parameter usage for the MPI2_PM_CONTROL_FEATURE_PORT_WIDTH_MODULATION Feature
1582 /* Parameter1 contains SAS port width modulation group number */
1583 /* Parameter2 indicates IOC action using these defines */
1584 #define MPI2_PM_CONTROL_PARAM2_REQUEST_OWNERSHIP (0x01)
1585 #define MPI2_PM_CONTROL_PARAM2_CHANGE_MODULATION (0x02)
1586 #define MPI2_PM_CONTROL_PARAM2_RELINQUISH_OWNERSHIP (0x03)
1587 /* Parameter3 indicates desired modulation level using these defines */
1588 #define MPI2_PM_CONTROL_PARAM3_25_PERCENT (0x00)
1589 #define MPI2_PM_CONTROL_PARAM3_50_PERCENT (0x01)
1590 #define MPI2_PM_CONTROL_PARAM3_75_PERCENT (0x02)
1591 #define MPI2_PM_CONTROL_PARAM3_100_PERCENT (0x03)
1592 /* Parameter4 is reserved */

1594 /* parameter usage for the MPI2_PM_CONTROL_FEATURE_PCIE_LINK Feature */
1595 /* Parameter1 indicates desired PCIe link speed using these defines */
1596 #define MPI2_PM_CONTROL_PARAM1_PCIE_2_5_GBPS (0x00)
1597 #define MPI2_PM_CONTROL_PARAM1_PCIE_5_0_GBPS (0x01)
1598 #define MPI2_PM_CONTROL_PARAM1_PCIE_8_0_GBPS (0x02)
1599 /* Parameter2 indicates desired PCIe link width using these defines */
1600 #define MPI2_PM_CONTROL_PARAM2_WIDTH_X1 (0x01)
1601 #define MPI2_PM_CONTROL_PARAM2_WIDTH_X2 (0x02)
1602 #define MPI2_PM_CONTROL_PARAM2_WIDTH_X4 (0x04)
1603 #define MPI2_PM_CONTROL_PARAM2_WIDTH_X8 (0x08)
1604 /* Parameter3 and Parameter4 are reserved */

1606 /* parameter usage for the MPI2_PM_CONTROL_FEATURE_IOC_SPEED Feature */
1607 /* Parameter1 indicates desired IOC hardware clock speed using these defines */
1608 #define MPI2_PM_CONTROL_PARAM1_FULL_IOC_SPEED (0x01)
1609 #define MPI2_PM_CONTROL_PARAM1_HALF_IOC_SPEED (0x02)
1610 #define MPI2_PM_CONTROL_PARAM1_QUARTER_IOC_SPEED (0x04)
1611 #define MPI2_PM_CONTROL_PARAM1_EIGHTH_IOC_SPEED (0x08)
1612 /* Parameter2, Parameter3, and Parameter4 are reserved */

1615 /* PowerManagementControl Reply message */
1616 typedef struct _MPI2_PWR_MGMT_CONTROL_REPLY
1617 {
1618     U8 Feature; /* 0x00 */
1619     U8 Reserved1; /* 0x01 */
1620     U8 MsgLength; /* 0x02 */
1621     U8 Function; /* 0x03 */
1622     U16 Reserved2; /* 0x04 */
1623     U8 Reserved3; /* 0x06 */
1624     U8 MsgFlags; /* 0x07 */
1625     U8 VP_ID; /* 0x08 */
1626     U8 VF_ID; /* 0x09 */
1627     U16 Reserved4; /* 0x0A */
1628     U16 Reserved5; /* 0x0C */
1629     U16 IOCStatus; /* 0x0E */
1630     U32 IOCLogInfo; /* 0x10 */
1631 } MPI2_PWR_MGMT_CONTROL_REPLY, MPI2_POINTER PTR_MPI2_PWR_MGMT_CONTROL_REPLY,
1632 Mpi2PwrMgmtControlReply_t, MPI2_POINTER pMpi2PwrMgmtControlReply_t;

1635 #endif

1637 #endif /* ! codereview */
```

new/usr/src/uts/common/sys/scsi/adapters/mpt_sas3/mpi/mpi2_raid.h

1

```
*****
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
*****

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

22 /*
23  * Copyright (c) 2000-2012 LSI Corporation.
24  *
25  * Redistribution and use in source and binary forms of all code within
26  * this file that is exclusively owned by LSI, with or without
27  * modification, is permitted provided that, in addition to the CDDL 1.0
28  * License requirements, the following conditions are met:
29  *
30  * Neither the name of the author nor the names of its contributors may be
31  * used to endorse or promote products derived from this software without
32  * specific prior written permission.
33  *
34  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
35  * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
36  * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
37  * FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
38  * COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
39  * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
40  * BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
41  * OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
42  * AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
43  * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
44  * OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
45  * DAMAGE.
46 */

48 /*
49  *      Name:  mpi2_raid.h
50  *      Title: MPI Integrated RAID messages and structures
51  *      Creation Date: April 26, 2007
52  *
53  *      mpi2_raid.h Version: 02.00.05
54  *
55  *      Version History
56  *      -----
57  *
58  *      Date      Version      Description
59  *      -----
60  *      04-30-07  02.00.00  Corresponds to Fusion-MPT MPI Specification Rev A.
61  *      08-31-07  02.00.01  Modifications to RAID Action request and reply,
```

new/usr/src/uts/common/sys/scsi/adapters/mpt_sas3/mpi/mpi2_raid.h

2

```
62 *      including the Actions and ActionData.
63 *      02-29-08  02.00.02  Added MPI2_RAID_ACTION_ADATA_DISABL_FULL_REBUILD.
64 *      05-21-08  02.00.03  Added MPI2_RAID_VOL_CREATION_NUM_PHYSDISKS so that
65 *                          the PhysDisk array in MPI2_RAID_VOLUME_CREATION_STRUCT
66 *                          can be sized by the build environment.
67 *      07-30-09  02.00.04  Added proper define for the Use Default Settings bit of
68 *                          VolumeCreationFlags and marked the old one as obsolete.
69 *      05-12-10  02.00.05  Added MPI2_RAID_VOL_FLAGS_OP_MDC define.
70 *      -----
71 */

73 #ifndef MPI2_RAID_H
74 #define MPI2_RAID_H

76 /*****
77  *
78  *      Integrated RAID Messages
79  *
80  *****/

82 /*****
83  *      RAID Action messages
84  *****/

86 /* ActionDataWord defines for use with MPI2_RAID_ACTION_DELETE_VOLUME action */
87 #define MPI2_RAID_ACTION_ADATA_KEEP_LBA0      (0x00000000)
88 #define MPI2_RAID_ACTION_ADATA_ZERO_LBA0      (0x00000001)

90 /* use MPI2_RAIDVOL0_SETTING_ defines from mpi2_cnfg.h for MPI2_RAID_ACTION_CHAN

92 /* ActionDataWord defines for use with MPI2_RAID_ACTION_DISABLE_ALL_VOLUMES acti
93 #define MPI2_RAID_ACTION_ADATA_DISABL_FULL_REBUILD (0x00000001)

95 /* ActionDataWord for MPI2_RAID_ACTION_SET_RAID_FUNCTION_RATE Action */
96 typedef struct _MPI2_RAID_ACTION_RATE_DATA
97 {
98     U8      RateToChange;          /* 0x00 */
99     U8      RateOrMode;            /* 0x01 */
100    U16     DataScrubDuration;      /* 0x02 */
101 } MPI2_RAID_ACTION_RATE_DATA, MPI2_POINTER PTR_MPI2_RAID_ACTION_RATE_DATA,
102   Mpi2RaidActionRateData_t, MPI2_POINTER pMpi2RaidActionRateData_t;

104 #define MPI2_RAID_ACTION_SET_RATE_RESYNC      (0x00)
105 #define MPI2_RAID_ACTION_SET_RATE_DATA_SCRUB (0x01)
106 #define MPI2_RAID_ACTION_SET_RATE_POWERSAVE_MODE (0x02)

108 /* ActionDataWord for MPI2_RAID_ACTION_START_RAID_FUNCTION Action */
109 typedef struct _MPI2_RAID_ACTION_START_RAID_FUNCTION
110 {
111     U8      RAIDFunction;          /* 0x00 */
112     U8      Flags;                 /* 0x01 */
113     U16     Reserved1;             /* 0x02 */
114 } MPI2_RAID_ACTION_START_RAID_FUNCTION,
115   MPI2_POINTER PTR_MPI2_RAID_ACTION_START_RAID_FUNCTION,
116   Mpi2RaidActionStartRaidFunction_t,
117   MPI2_POINTER pMpi2RaidActionStartRaidFunction_t;

119 /* defines for the RAIDFunction field */
120 #define MPI2_RAID_ACTION_START_BACKGROUND_INIT (0x00)
121 #define MPI2_RAID_ACTION_START_ONLINE_CAP_EXPANSION (0x01)
122 #define MPI2_RAID_ACTION_START_CONSISTENCY_CHECK (0x02)

124 /* defines for the Flags field */
125 #define MPI2_RAID_ACTION_START_NEW (0x00)
126 #define MPI2_RAID_ACTION_START_RESUME (0x01)
```

```

128 /* ActionDataWord for MPI2_RAID_ACTION_STOP_RAID_FUNCTION Action */
129 typedef struct _MPI2_RAID_ACTION_STOP_RAID_FUNCTION
130 {
131     U8          RAIDFunction;          /* 0x00 */
132     U8          Flags;                 /* 0x01 */
133     U16         Reserved1;             /* 0x02 */
134 } MPI2_RAID_ACTION_STOP_RAID_FUNCTION,
135 MPI2_POINTER PTR_MPI2_RAID_ACTION_STOP_RAID_FUNCTION,
136 Mpi2RaidActionStopRaidFunction_t,
137 MPI2_POINTER pMpi2RaidActionStopRaidFunction_t;

139 /* defines for the RAIDFunction field */
140 #define MPI2_RAID_ACTION_STOP_BACKGROUND_INIT      (0x00)
141 #define MPI2_RAID_ACTION_STOP_ONLINE_CAP_EXPANSION (0x01)
142 #define MPI2_RAID_ACTION_STOP_CONSISTENCY_CHECK    (0x02)

144 /* defines for the Flags field */
145 #define MPI2_RAID_ACTION_STOP_ABORT                (0x00)
146 #define MPI2_RAID_ACTION_STOP_PAUSE                (0x01)

148 /* ActionDataWord for MPI2_RAID_ACTION_CREATE_HOT_SPARE Action */
149 typedef struct _MPI2_RAID_ACTION_HOT_SPARE
150 {
151     U8          HotSparePool;          /* 0x00 */
152     U8          Reserved1;              /* 0x01 */
153     U16         DevHandle;             /* 0x02 */
154 } MPI2_RAID_ACTION_HOT_SPARE, MPI2_POINTER PTR_MPI2_RAID_ACTION_HOT_SPARE,
155 Mpi2RaidActionHotSpare_t, MPI2_POINTER pMpi2RaidActionHotSpare_t;

157 /* ActionDataWord for MPI2_RAID_ACTION_DEVICE_FW_UPDATE_MODE Action */
158 typedef struct _MPI2_RAID_ACTION_FW_UPDATE_MODE
159 {
160     U8          Flags;                 /* 0x00 */
161     U8          DeviceFirmwareUpdateModeTimeout; /* 0x01 */
162     U16         Reserved1;             /* 0x02 */
163 } MPI2_RAID_ACTION_FW_UPDATE_MODE,
164 MPI2_POINTER PTR_MPI2_RAID_ACTION_FW_UPDATE_MODE,
165 Mpi2RaidActionFwUpdateMode_t, MPI2_POINTER pMpi2RaidActionFwUpdateMode_t;

167 /* ActionDataWord defines for use with MPI2_RAID_ACTION_DEVICE_FW_UPDATE_MODE ac
168 #define MPI2_RAID_ACTION_ADATA_DISABLE_FW_UPDATE    (0x00)
169 #define MPI2_RAID_ACTION_ADATA_ENABLE_FW_UPDATE     (0x01)

171 typedef union _MPI2_RAID_ACTION_DATA
172 {
173     U32          Word;
174     MPI2_RAID_ACTION_RATE_DATA      Rates;
175     MPI2_RAID_ACTION_START_RAID_FUNCTION StartRaidFunction;
176     MPI2_RAID_ACTION_STOP_RAID_FUNCTION StopRaidFunction;
177     MPI2_RAID_ACTION_HOT_SPARE      HotSpare;
178     MPI2_RAID_ACTION_FW_UPDATE_MODE FwUpdateMode;
179 } MPI2_RAID_ACTION_DATA, MPI2_POINTER PTR_MPI2_RAID_ACTION_DATA,
180 Mpi2RaidActionData_t, MPI2_POINTER pMpi2RaidActionData_t;

183 /* RAID Action Request Message */
184 typedef struct _MPI2_RAID_ACTION_REQUEST
185 {
186     U8          Action;                /* 0x00 */
187     U8          Reserved1;             /* 0x01 */
188     U8          ChainOffset;           /* 0x02 */
189     U8          Function;               /* 0x03 */
190     U16         VolDevHandle;           /* 0x04 */
191     U8          PhysDiskNum;           /* 0x06 */
192     U8          MsgFlags;               /* 0x07 */
193     U8          VP_ID;                 /* 0x08 */

```

```

194     U8          VF_ID;                 /* 0x09 */
195     U16         Reserved2;             /* 0x0A */
196     U32         Reserved3;             /* 0x0C */
197     MPI2_RAID_ACTION_DATA      ActionDataWord; /* 0x10 */
198     MPI2_SGE_SIMPLE_UNION      ActionDataSGE;   /* 0x14 */
199 } MPI2_RAID_ACTION_REQUEST, MPI2_POINTER PTR_MPI2_RAID_ACTION_REQUEST,
200 Mpi2RaidActionRequest_t, MPI2_POINTER pMpi2RaidActionRequest_t;

202 /* RAID Action request Action values */

204 #define MPI2_RAID_ACTION_INDICATOR_STRUCT          (0x01)
205 #define MPI2_RAID_ACTION_CREATE_VOLUME             (0x02)
206 #define MPI2_RAID_ACTION_DELETE_VOLUME            (0x03)
207 #define MPI2_RAID_ACTION_DISABLE_ALL_VOLUMES      (0x04)
208 #define MPI2_RAID_ACTION_ENABLE_ALL_VOLUMES       (0x05)
209 #define MPI2_RAID_ACTION_PHYSDISK_OFFLINE         (0x0A)
210 #define MPI2_RAID_ACTION_PHYSDISK_ONLINE          (0x0B)
211 #define MPI2_RAID_ACTION_FAIL_PHYSDISK            (0x0F)
212 #define MPI2_RAID_ACTION_ACTIVATE_VOLUME          (0x11)
213 #define MPI2_RAID_ACTION_DEVICE_FW_UPDATE_MODE    (0x15)
214 #define MPI2_RAID_ACTION_CHANGE_VOL_WRITE_CACHE   (0x17)
215 #define MPI2_RAID_ACTION_SET_VOLUME_NAME           (0x18)
216 #define MPI2_RAID_ACTION_SET_RAID_FUNCTION_RATE    (0x19)
217 #define MPI2_RAID_ACTION_ENABLE_FAILED_VOLUME     (0x1C)
218 #define MPI2_RAID_ACTION_CREATE_HOT_SPARE         (0x1D)
219 #define MPI2_RAID_ACTION_DELETE_HOT_SPARE         (0x1E)
220 #define MPI2_RAID_ACTION_SYSTEM_SHUTDOWN_INITIATED (0x20)
221 #define MPI2_RAID_ACTION_START_RAID_FUNCTION      (0x21)
222 #define MPI2_RAID_ACTION_STOP_RAID_FUNCTION        (0x22)
223 #define MPI2_RAID_ACTION_FAST_PATH_PERMITTED      (0x24)

225 /* RAID Volume Creation Structure */

227 /*
228 * The following define can be customized for the targeted product.
229 */
230 #ifndef MPI2_RAID_VOL_CREATION_NUM_PHYSDISKS
231 #define MPI2_RAID_VOL_CREATION_NUM_PHYSDISKS      (1)
232 #endif

234 typedef struct _MPI2_RAID_VOLUME_PHYSDISK
235 {
236     U8          RAIDSetNum;            /* 0x00 */
237     U8          PhysDiskMap;           /* 0x01 */
238     U16         PhysDiskDevHandle;     /* 0x02 */
239 } MPI2_RAID_VOLUME_PHYSDISK, MPI2_POINTER PTR_MPI2_RAID_VOLUME_PHYSDISK,
240 Mpi2RaidVolumePhysDisk_t, MPI2_POINTER pMpi2RaidVolumePhysDisk_t;

242 /* defines for the PhysDiskMap field */
243 #define MPI2_RAIDACTION_PHYSDISK_PRIMARY          (0x01)
244 #define MPI2_RAIDACTION_PHYSDISK_SECONDARY        (0x02)

246 typedef struct _MPI2_RAID_VOLUME_CREATION_STRUCT
247 {
248     U8          NumPhysDisks;          /* 0x00 */
249     U8          VolumeType;            /* 0x01 */
250     U16         Reserved1;             /* 0x02 */
251     U32         VolumeCreationFlags;   /* 0x04 */
252     U32         VolumeSettings;        /* 0x08 */
253     U8          Reserved2;             /* 0x0C */
254     U8          ResyncRate;            /* 0x0D */
255     U16         DataScrubDuration;     /* 0x0E */
256     U64         VolumeMaxLBA;         /* 0x10 */
257     U32         StripeSize;           /* 0x18 */
258     U8          Name[16];              /* 0x1C */
259     MPI2_RAID_VOLUME_PHYSDISK      PhysDisk[MPI2_RAID_VOL_CREATION_NUM_PHYSDISKS];

```

```

260 } MPI2_RAID_VOLUME_CREATION_STRUCT,
261 MPI2_POINTER PTR_MPI2_RAID_VOLUME_CREATION_STRUCT,
262 Mpi2RaidVolumeCreationStruct_t, MPI2_POINTER pMpi2RaidVolumeCreationStruct_t;

264 /* use MPI2_RAID_VOL_TYPE_ defines from mpi2_cnfg.h for VolumeType */

266 /* defines for the VolumeCreationFlags field */
267 #define MPI2_RAID_VOL_CREATION_DEFAULT_SETTINGS (0x80000000)
268 #define MPI2_RAID_VOL_CREATION_BACKGROUND_INIT (0x00000004)
269 #define MPI2_RAID_VOL_CREATION_LOW_LEVEL_INIT (0x00000002)
270 #define MPI2_RAID_VOL_CREATION_MIGRATE_DATA (0x00000001)
271 /* The following is an obsolete define.
272 * It must be shifted left 24 bits in order to set the proper bit.
273 */
274 #define MPI2_RAID_VOL_CREATION_USE_DEFAULT_SETTINGS (0x80)

277 /* RAID Online Capacity Expansion Structure */

279 typedef struct _MPI2_RAID_ONLINE_CAPACITY_EXPANSION
280 {
281     U32          Flags;                      /* 0x00 */
282     U16          DevHandle0;                 /* 0x04 */
283     U16          Reserved1;                  /* 0x06 */
284     U16          DevHandle1;                 /* 0x08 */
285     U16          Reserved2;                  /* 0x0A */
286 } MPI2_RAID_ONLINE_CAPACITY_EXPANSION,
287 MPI2_POINTER PTR_MPI2_RAID_ONLINE_CAPACITY_EXPANSION,
288 Mpi2RaidOnlineCapacityExpansion_t,
289 MPI2_POINTER pMpi2RaidOnlineCapacityExpansion_t;

292 /* RAID Volume Indicator Structure */

294 typedef struct _MPI2_RAID_VOL_INDICATOR
295 {
296     U64          TotalBlocks;                 /* 0x00 */
297     U64          BlocksRemaining;             /* 0x08 */
298     U32          Flags;                       /* 0x10 */
299 } MPI2_RAID_VOL_INDICATOR, MPI2_POINTER PTR_MPI2_RAID_VOL_INDICATOR,
300 Mpi2RaidVolIndicator_t, MPI2_POINTER pMpi2RaidVolIndicator_t;

302 /* defines for RAID Volume Indicator Flags field */
303 #define MPI2_RAID_VOL_FLAGS_OP_MASK (0x0000000F)
304 #define MPI2_RAID_VOL_FLAGS_OP_BACKGROUND_INIT (0x00000000)
305 #define MPI2_RAID_VOL_FLAGS_OP_ONLINE_CAP_EXPANSION (0x00000001)
306 #define MPI2_RAID_VOL_FLAGS_OP_CONSISTENCY_CHECK (0x00000002)
307 #define MPI2_RAID_VOL_FLAGS_OP_RESYNC (0x00000003)
308 #define MPI2_RAID_VOL_FLAGS_OP_MDC (0x00000004)

311 /* RAID Action Reply ActionData union */
312 typedef union _MPI2_RAID_ACTION_REPLY_DATA
313 {
314     U32          Word[5];
315     MPI2_RAID_VOL_INDICATOR RaidVolumeIndicator;
316     U16          VolDevHandle;
317     U8           VolumeState;
318     U8           PhysDiskNum;
319 } MPI2_RAID_ACTION_REPLY_DATA, MPI2_POINTER PTR_MPI2_RAID_ACTION_REPLY_DATA,
320 Mpi2RaidActionReplyData_t, MPI2_POINTER pMpi2RaidActionReplyData_t;

322 /* use MPI2_RAIDVOL0_SETTING_ defines from mpi2_cnfg.h for MPI2_RAID_ACTION_CHAN

325 /* RAID Action Reply Message */

```

```

326 typedef struct _MPI2_RAID_ACTION_REPLY
327 {
328     U8          Action;                      /* 0x00 */
329     U8          Reserved1;                   /* 0x01 */
330     U8          MsgLength;                   /* 0x02 */
331     U8          Function;                    /* 0x03 */
332     U16         VolDevHandle;                /* 0x04 */
333     U8          PhysDiskNum;                 /* 0x06 */
334     U8          MsgFlags;                    /* 0x07 */
335     U8          VP_ID;                       /* 0x08 */
336     U8          VF_ID;                       /* 0x09 */
337     U16         Reserved2;                   /* 0x0A */
338     U16         Reserved3;                   /* 0x0C */
339     U16         IOCStatus;                   /* 0x0E */
340     U32         IOCLogInfo;                  /* 0x10 */
341     MPI2_RAID_ACTION_REPLY_DATA ActionData; /* 0x14 */
342 } MPI2_RAID_ACTION_REPLY, MPI2_POINTER PTR_MPI2_RAID_ACTION_REPLY,
343 Mpi2RaidActionReply_t, MPI2_POINTER pMpi2RaidActionReply_t;

346 #endif

348 #endif /* ! codereview */

```

new/usr/src/uts/common/sys/scsi/adapters/mpt_sas3/mpi/mpi2_sas.h

1

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

```
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */
21
22 /*
23  * Copyright (c) 2000-2012 LSI Corporation.
24  *
25  * Redistribution and use in source and binary forms of all code within
26  * this file that is exclusively owned by LSI, with or without
27  * modification, is permitted provided that, in addition to the CDDL 1.0
28  * License requirements, the following conditions are met:
29  *
30  * Neither the name of the author nor the names of its contributors may be
31  * used to endorse or promote products derived from this software without
32  * specific prior written permission.
33  *
34  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
35  * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
36  * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
37  * FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
38  * COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
39  * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
40  * BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
41  * OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
42  * AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
43  * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
44  * OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
45  * DAMAGE.
46 */
47
48 /*
49  * Name: mpi2_sas.h
50  * Title: MPI Serial Attached SCSI structures and definitions
51  * Creation Date: February 9, 2007
52  *
53  * mpi2_sas.h Version: 02.00.xx
54  *
55  * NOTE: Names (typedefs, defines, etc.) beginning with an MPI25 or Mpi25
56  * prefix are for use only on MPI v2.5 products, and must not be used
57  * with MPI v2.0 products. Unless otherwise noted, names beginning with
58  * MPI2 or Mpi2 are for use with both MPI v2.0 and MPI v2.5 products.
59  *
60  * Version History
61  * -----
```

new/usr/src/uts/common/sys/scsi/adapters/mpt_sas3/mpi/mpi2_sas.h

2

```
62 *
63 * Date Version Description
64 * -----
65 * 04-30-07 02.00.00 Corresponds to Fusion-MPT MPI Specification Rev A.
66 * 06-26-07 02.00.01 Added Clear All Persistent Operation to SAS IO Unit
67 * Control Request.
68 * 10-02-08 02.00.02 Added Set IOC Parameter Operation to SAS IO Unit Control
69 * Request.
70 * 10-28-09 02.00.03 Changed the type of SGL in MPI2_SATA_PASSTHROUGH_REQUEST
71 * to MPI2_SGE_IO_UNION since it supports chained SGLs.
72 * 05-12-10 02.00.04 Modified some comments.
73 * 08-11-10 02.00.05 Added NCQ operations to SAS IO Unit Control.
74 * -----
75 */
76
77 #ifndef MPI2_SAS_H
78 #define MPI2_SAS_H
79
80 /*
81  * Values for SASStatus.
82  */
83 #define MPI2_SASSTATUS_SUCCESS (0x00)
84 #define MPI2_SASSTATUS_UNKNOWN_ERROR (0x01)
85 #define MPI2_SASSTATUS_INVALID_FRAME (0x02)
86 #define MPI2_SASSTATUS_UTC_BAD_DEST (0x03)
87 #define MPI2_SASSTATUS_UTC_BREAK_RECEIVED (0x04)
88 #define MPI2_SASSTATUS_UTC_CONNECT_RATE_NOT_SUPPORTED (0x05)
89 #define MPI2_SASSTATUS_UTC_PORT_LAYER_REQUEST (0x06)
90 #define MPI2_SASSTATUS_UTC_PROTOCOL_NOT_SUPPORTED (0x07)
91 #define MPI2_SASSTATUS_UTC_STP_RESOURCES_BUSY (0x08)
92 #define MPI2_SASSTATUS_UTC_WRONG_DESTINATION (0x09)
93 #define MPI2_SASSTATUS_SHORT_INFORMATION_UNIT (0x0A)
94 #define MPI2_SASSTATUS_LONG_INFORMATION_UNIT (0x0B)
95 #define MPI2_SASSTATUS_XFER_RDY_INCORRECT_WRITE_DATA (0x0C)
96 #define MPI2_SASSTATUS_XFER_RDY_REQUEST_OFFSET_ERROR (0x0D)
97 #define MPI2_SASSTATUS_XFER_RDY_NOT_EXPECTED (0x0E)
98 #define MPI2_SASSTATUS_DATA_INCORRECT_DATA_LENGTH (0x0F)
99 #define MPI2_SASSTATUS_DATA_TOO_MUCH_READ_DATA (0x10)
100 #define MPI2_SASSTATUS_DATA_OFFSET_ERROR (0x11)
101 #define MPI2_SASSTATUS_SDSF_NAK_RECEIVED (0x12)
102 #define MPI2_SASSTATUS_SDSF_CONNECTION_FAILED (0x13)
103 #define MPI2_SASSTATUS_INITIATOR_RESPONSE_TIMEOUT (0x14)
104
105
106 /*
107  * Values for the SAS DeviceInfo field used in SAS Device Status Change Event
108  * data and SAS Configuration pages.
109  */
110 #define MPI2_SAS_DEVICE_INFO_SEP (0x00004000)
111 #define MPI2_SAS_DEVICE_INFO_ATAPI_DEVICE (0x00002000)
112 #define MPI2_SAS_DEVICE_INFO_LSI_DEVICE (0x00001000)
113 #define MPI2_SAS_DEVICE_INFO_DIRECT_ATTACH (0x00000800)
114 #define MPI2_SAS_DEVICE_INFO_SSP_TARGET (0x00000400)
115 #define MPI2_SAS_DEVICE_INFO_STP_TARGET (0x00000200)
116 #define MPI2_SAS_DEVICE_INFO_SMP_TARGET (0x00000100)
117 #define MPI2_SAS_DEVICE_INFO_SATA_DEVICE (0x00000080)
118 #define MPI2_SAS_DEVICE_INFO_SSP_INITIATOR (0x00000040)
119 #define MPI2_SAS_DEVICE_INFO_STP_INITIATOR (0x00000020)
120 #define MPI2_SAS_DEVICE_INFO_SMP_INITIATOR (0x00000010)
121 #define MPI2_SAS_DEVICE_INFO_SATA_HOST (0x00000008)
122
123 #define MPI2_SAS_DEVICE_INFO_MASK_DEVICE_TYPE (0x00000007)
124 #define MPI2_SAS_DEVICE_INFO_NO_DEVICE (0x00000000)
125 #define MPI2_SAS_DEVICE_INFO_END_DEVICE (0x00000001)
126 #define MPI2_SAS_DEVICE_INFO_EDGE_EXPANDER (0x00000002)
127 #define MPI2_SAS_DEVICE_INFO_FANOUT_EXPANDER (0x00000003)
```

```

130 /*****
131 *
132 *      SAS Messages
133 *
134 *****/

136 /*****
137 * SMP Passthrough messages
138 *****/

140 /* SMP Passthrough Request Message */
141 typedef struct _MPI2_SMP_PASSTHROUGH_REQUEST
142 {
143     U8          PassthroughFlags; /* 0x00 */
144     U8          PhysicalPort;     /* 0x01 */
145     U8          ChainOffset;      /* 0x02 */
146     U8          Function;         /* 0x03 */
147     U16         RequestDataLength; /* 0x04 */
148     U8          SGLFlags;         /* 0x06 */ /* MPI v2.0 only. Res
149     U8          MsgFlags;         /* 0x07 */
150     U8          VP_ID;            /* 0x08 */
151     U8          VF_ID;            /* 0x09 */
152     U16         Reserved1;        /* 0x0A */
153     U32         Reserved2;        /* 0x0C */
154     U64         SASAddress;       /* 0x10 */
155     U32         Reserved3;        /* 0x18 */
156     U32         Reserved4;        /* 0x1C */
157     MPI2_SIMPLE_SGE_UNION SGL;    /* 0x20 */ /* MPI v2.5: IEEE Sim
158 } MPI2_SMP_PASSTHROUGH_REQUEST, MPI2_POINTER PTR_MPI2_SMP_PASSTHROUGH_REQUEST,
159   Mpi2SmpPassthroughRequest_t, MPI2_POINTER pMpi2SmpPassthroughRequest_t;

161 /* values for PassthroughFlags field */
162 #define MPI2_SMP_PT_REQ_PT_FLAGS_IMMEDIATE (0x80)

164 /* MPI v2.0: use MPI2_SGLFLAGS_ defines from mpi2.h for the SGLFlags field */

167 /* SMP Passthrough Reply Message */
168 typedef struct _MPI2_SMP_PASSTHROUGH_REPLY
169 {
170     U8          PassthroughFlags; /* 0x00 */
171     U8          PhysicalPort;     /* 0x01 */
172     U8          MsgLength;        /* 0x02 */
173     U8          Function;         /* 0x03 */
174     U16         ResponseDataLength; /* 0x04 */
175     U8          SGLFlags;         /* 0x06 */
176     U8          MsgFlags;         /* 0x07 */
177     U8          VP_ID;            /* 0x08 */
178     U8          VF_ID;            /* 0x09 */
179     U16         Reserved1;        /* 0x0A */
180     U8          Reserved2;        /* 0x0C */
181     U8          SASStatus;        /* 0x0D */
182     U16         IOCStatus;        /* 0x0E */
183     U32         IOCLogInfo;       /* 0x10 */
184     U32         Reserved3;        /* 0x14 */
185     U8          ResponseData[4];  /* 0x18 */
186 } MPI2_SMP_PASSTHROUGH_REPLY, MPI2_POINTER PTR_MPI2_SMP_PASSTHROUGH_REPLY,
187   Mpi2SmpPassthroughReply_t, MPI2_POINTER pMpi2SmpPassthroughReply_t;

189 /* values for PassthroughFlags field */
190 #define MPI2_SMP_PT_REPLY_PT_FLAGS_IMMEDIATE (0x80)

192 /* values for SASStatus field are at the top of this file */

```

```

195 /*****
196 * SATA Passthrough messages
197 *****/

199 /* SATA Passthrough Request Message */
200 typedef struct _MPI2_SATA_PASSTHROUGH_REQUEST
201 {
202     U16         DevHandle;        /* 0x00 */
203     U8          ChainOffset;      /* 0x02 */
204     U8          Function;         /* 0x03 */
205     U16         PassthroughFlags; /* 0x04 */
206     U8          SGLFlags;         /* 0x06 */ /* MPI v2.0 only. Res
207     U8          MsgFlags;         /* 0x07 */
208     U8          VP_ID;            /* 0x08 */
209     U8          VF_ID;            /* 0x09 */
210     U16         Reserved1;        /* 0x0A */
211     U32         Reserved2;        /* 0x0C */
212     U32         Reserved3;        /* 0x10 */
213     U32         Reserved4;        /* 0x14 */
214     U32         DataLength;       /* 0x18 */
215     U8          CommandFIS[20];   /* 0x1C */
216     MPI2_SGE_IO_UNION SGL;        /* 0x30 */ /* MPI v2.5: IEEE 64
217 } MPI2_SATA_PASSTHROUGH_REQUEST, MPI2_POINTER PTR_MPI2_SATA_PASSTHROUGH_REQUEST,
218   Mpi2SataPassthroughRequest_t, MPI2_POINTER pMpi2SataPassthroughRequest_t;

220 /* values for PassthroughFlags field */
221 #define MPI2_SATA_PT_REQ_PT_FLAGS_EXECUTE_DIAG (0x0100)
222 #define MPI2_SATA_PT_REQ_PT_FLAGS_DMA (0x0020)
223 #define MPI2_SATA_PT_REQ_PT_FLAGS_PIO (0x0010)
224 #define MPI2_SATA_PT_REQ_PT_FLAGS_UNSPECIFIED_VU (0x0004)
225 #define MPI2_SATA_PT_REQ_PT_FLAGS_WRITE (0x0002)
226 #define MPI2_SATA_PT_REQ_PT_FLAGS_READ (0x0001)

228 /* MPI v2.0: use MPI2_SGLFLAGS_ defines from mpi2.h for the SGLFlags field */

231 /* SATA Passthrough Reply Message */
232 typedef struct _MPI2_SATA_PASSTHROUGH_REPLY
233 {
234     U16         DevHandle;        /* 0x00 */
235     U8          MsgLength;        /* 0x02 */
236     U8          Function;         /* 0x03 */
237     U16         PassthroughFlags; /* 0x04 */
238     U8          SGLFlags;         /* 0x06 */
239     U8          MsgFlags;         /* 0x07 */
240     U8          VP_ID;            /* 0x08 */
241     U8          VF_ID;            /* 0x09 */
242     U16         Reserved1;        /* 0x0A */
243     U8          Reserved2;        /* 0x0C */
244     U8          SASStatus;        /* 0x0D */
245     U16         IOCStatus;        /* 0x0E */
246     U32         IOCLogInfo;       /* 0x10 */
247     U8          StatusFIS[20];    /* 0x14 */
248     U32         StatusControlRegisters; /* 0x28 */
249     U32         TransferCount;    /* 0x2C */
250 } MPI2_SATA_PASSTHROUGH_REPLY, MPI2_POINTER PTR_MPI2_SATA_PASSTHROUGH_REPLY,
251   Mpi2SataPassthroughReply_t, MPI2_POINTER pMpi2SataPassthroughReply_t;

253 /* values for SASStatus field are at the top of this file */

256 /*****
257 * SAS IO Unit Control messages
258 *****/

```

```

260 /* SAS IO Unit Control Request Message */
261 typedef struct _MPI2_SAS_IUNIT_CONTROL_REQUEST
262 {
263     U8          Operation;          /* 0x00 */
264     U8          Reserved1;          /* 0x01 */
265     U8          ChainOffset;        /* 0x02 */
266     U8          Function;           /* 0x03 */
267     U16         DevHandle;          /* 0x04 */
268     U8          IOCParameter;       /* 0x06 */
269     U8          MsgFlags;           /* 0x07 */
270     U8          VP_ID;              /* 0x08 */
271     U8          VF_ID;              /* 0x09 */
272     U16         Reserved3;          /* 0x0A */
273     U16         Reserved4;          /* 0x0C */
274     U8          PhyNum;             /* 0x0E */
275     U8          PrimFlags;          /* 0x0F */
276     U32         Primitive;          /* 0x10 */
277     U8          LookupMethod;       /* 0x14 */
278     U8          Reserved5;          /* 0x15 */
279     U16         SlotNumber;         /* 0x16 */
280     U64         LookupAddress;      /* 0x18 */
281     U32         IOCParameterValue; /* 0x20 */
282     U32         Reserved7;          /* 0x24 */
283     U32         Reserved8;          /* 0x28 */
284 } MPI2_SAS_IUNIT_CONTROL_REQUEST,
285 MPI2_POINTER PTR_MPI2_SAS_IUNIT_CONTROL_REQUEST,
286 Mpi2SasIoUnitControlRequest_t, MPI2_POINTER pMpi2SasIoUnitControlRequest_t;

288 /* values for the Operation field */
289 #define MPI2_SAS_OP_CLEAR_ALL_PERSISTENT (0x02)
290 #define MPI2_SAS_OP_PHY_LINK_RESET (0x06)
291 #define MPI2_SAS_OP_PHY_HARD_RESET (0x07)
292 #define MPI2_SAS_OP_PHY_CLEAR_ERROR_LOG (0x08)
293 #define MPI2_SAS_OP_SEND_PRIMITIVE (0x0A)
294 #define MPI2_SAS_OP_FORCE_FULL_DISCOVERY (0x0B)
295 #define MPI2_SAS_OP_TRANSMIT_PORT_SELECT_SIGNAL (0x0C)
296 #define MPI2_SAS_OP_REMOVE_DEVICE (0x0D)
297 #define MPI2_SAS_OP_LOOKUP_MAPPING (0x0E)
298 #define MPI2_SAS_OP_SET_IOC_PARAMETER (0x0F)
299 #define MPI25_SAS_OP_ENABLE_FP_DEVICE (0x10)
300 #define MPI25_SAS_OP_DISABLE_FP_DEVICE (0x11)
301 #define MPI25_SAS_OP_ENABLE_FP_ALL (0x12)
302 #define MPI25_SAS_OP_DISABLE_FP_ALL (0x13)
303 #define MPI2_SAS_OP_DEV_ENABLE_NCQ (0x14)
304 #define MPI2_SAS_OP_DEV_DISABLE_NCQ (0x15)
305 #define MPI2_SAS_OP_PRODUCT_SPECIFIC_MIN (0x80)

307 /* values for the PrimFlags field */
308 #define MPI2_SAS_PRIMFLAGS_SINGLE (0x08)
309 #define MPI2_SAS_PRIMFLAGS_TRIPLE (0x02)
310 #define MPI2_SAS_PRIMFLAGS_REDUNDANT (0x01)

312 /* values for the LookupMethod field */
313 #define MPI2_SAS_LOOKUP_METHOD_SAS_ADDRESS (0x01)
314 #define MPI2_SAS_LOOKUP_METHOD_SAS_ENCLOSURE_SLOT (0x02)
315 #define MPI2_SAS_LOOKUP_METHOD_SAS_DEVICE_NAME (0x03)

```

```

318 /* SAS IO Unit Control Reply Message */
319 typedef struct _MPI2_SAS_IUNIT_CONTROL_REPLY
320 {
321     U8          Operation;          /* 0x00 */
322     U8          Reserved1;          /* 0x01 */
323     U8          MsgLength;          /* 0x02 */
324     U8          Function;           /* 0x03 */
325     U16         DevHandle;          /* 0x04 */

```

```

326     U8          IOCParameter;       /* 0x06 */
327     U8          MsgFlags;           /* 0x07 */
328     U8          VP_ID;              /* 0x08 */
329     U8          VF_ID;              /* 0x09 */
330     U16         Reserved3;          /* 0x0A */
331     U16         Reserved4;          /* 0x0C */
332     U16         IOCStatus;          /* 0x0E */
333     U32         IOCLogInfo;         /* 0x10 */
334 } MPI2_SAS_IUNIT_CONTROL_REPLY,
335 MPI2_POINTER PTR_MPI2_SAS_IUNIT_CONTROL_REPLY,
336 Mpi2SasIoUnitControlReply_t, MPI2_POINTER pMpi2SasIoUnitControlReply_t;

339 #endif

342 #endif /* ! codereview */

```

new/usr/src/uts/common/sys/scsi/adapters/mpt_sas3/mpi/mpi2_tool.h

1

```
*****
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
*****

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

22 /*
23  * Copyright (c) 2000-2012 LSI Corporation.
24  *
25  * Redistribution and use in source and binary forms of all code within
26  * this file that is exclusively owned by LSI, with or without
27  * modification, is permitted provided that, in addition to the CDDL 1.0
28  * License requirements, the following conditions are met:
29  *
30  * Neither the name of the author nor the names of its contributors may be
31  * used to endorse or promote products derived from this software without
32  * specific prior written permission.
33  *
34  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
35  * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
36  * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
37  * FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
38  * COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
39  * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
40  * BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
41  * OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
42  * AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
43  * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
44  * OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
45  * DAMAGE.
46 */

48 /*
49  *      Name:  mpi2_tool.h
50  *      Title: MPI diagnostic tool structures and definitions
51  *      Creation Date:  March 26, 2007
52  *
53  *      mpi2_tool.h Version:  02.00.06
54  *
55  *      Version History
56  *      -----
57  *
58  *      Date      Version  Description
59  *      -----
60  *      04-30-07  02.00.00  Corresponds to Fusion-MPT MPI Specification Rev A.
61  *      12-18-07  02.00.01  Added Diagnostic Buffer Post and Diagnostic Release
```

new/usr/src/uts/common/sys/scsi/adapters/mpt_sas3/mpi/mpi2_tool.h

2

```
62 *      structures and defines.
63 *      02-29-08  02.00.02  Modified various names to make them 32-character unique.
64 *      05-06-09  02.00.03  Added ISTWI Read Write Tool and Diagnostic CLI Tool.
65 *      07-30-09  02.00.04  Added ExtendedType field to DiagnosticBufferPost request
66 *                          and reply messages.
67 *                          Added MPI2_DIAG_BUF_TYPE_EXTENDED.
68 *                          Incremented MPI2_DIAG_BUF_TYPE_COUNT.
69 *      05-12-10  02.00.05  Added Diagnostic Data Upload tool.
70 *      08-11-10  02.00.06  Added defines that were missing for Diagnostic Buffer
71 *                          Post Request.
72 *      -----
73 */

75 #ifndef MPI2_TOOL_H
76 #define MPI2_TOOL_H

78 /*****
79  *
80  *      Toolbox Messages
81  *
82  *****/

84 /* defines for the Tools */
85 #define MPI2_TOOLBOX_CLEAN_TOOL                (0x00)
86 #define MPI2_TOOLBOX_MEMORY_MOVE_TOOL          (0x01)
87 #define MPI2_TOOLBOX_DIAG_DATA_UPLOAD_TOOL     (0x02)
88 #define MPI2_TOOLBOX_ISTWI_READ_WRITE_TOOL     (0x03)
89 #define MPI2_TOOLBOX_BEACON_TOOL               (0x05)
90 #define MPI2_TOOLBOX_DIAGNOSTIC_CLI_TOOL       (0x06)

93 /*****
94  *      Toolbox reply
95  *****/

97 typedef struct _MPI2_TOOLBOX_REPLY
98 {
99     U8                Tool;                /* 0x00 */
100    U8                Reserved1;            /* 0x01 */
101    U8                MsgLength;            /* 0x02 */
102    U8                Function;             /* 0x03 */
103    U16               Reserved2;            /* 0x04 */
104    U8                Reserved3;            /* 0x06 */
105    U8                MsgFlags;             /* 0x07 */
106    U8                VP_ID;                /* 0x08 */
107    U8                VF_ID;                /* 0x09 */
108    U16               Reserved4;            /* 0x0A */
109    U16               Reserved5;            /* 0x0C */
110    U16               IOCStatus;            /* 0x0E */
111    U32               IOCLogInfo;          /* 0x10 */
112 } MPI2_TOOLBOX_REPLY, MPI2_POINTER PTR_MPI2_TOOLBOX_REPLY,
113   Mpi2ToolboxReply_t, MPI2_POINTER pMpi2ToolboxReply_t;

116 /*****
117  *      Toolbox Clean Tool request
118  *****/

120 typedef struct _MPI2_TOOLBOX_CLEAN_REQUEST
121 {
122     U8                Tool;                /* 0x00 */
123     U8                Reserved1;            /* 0x01 */
124     U8                ChainOffset;          /* 0x02 */
125     U8                Function;             /* 0x03 */
126     U16               Reserved2;            /* 0x04 */
127     U8                Reserved3;            /* 0x06 */
128 }
```



```

128     U8           MsgFlags;           /* 0x07 */
129     U8           VP_ID;              /* 0x08 */
130     U8           VF_ID;              /* 0x09 */
131     U16          Reserved4;          /* 0x0A */
132     U32          Flags;              /* 0x0C */
133 } MPI2_TOOLBOX_CLEAN_REQUEST, MPI2_POINTER PTR_MPI2_TOOLBOX_CLEAN_REQUEST,
134   MPI2ToolboxCleanRequest_t, MPI2_POINTER pMPI2ToolboxCleanRequest_t;

136 /* values for the Flags field */
137 #define MPI2_TOOLBOX_CLEAN_BOOT_SERVICES (0x80000000)
138 #define MPI2_TOOLBOX_CLEAN_PERSIST_MANUFACT_PAGES (0x40000000)
139 #define MPI2_TOOLBOX_CLEAN_OTHER_PERSIST_PAGES (0x20000000)
140 #define MPI2_TOOLBOX_CLEAN_FW_CURRENT (0x10000000)
141 #define MPI2_TOOLBOX_CLEAN_FW_BACKUP (0x08000000)
142 #define MPI2_TOOLBOX_CLEAN_MEGARAID (0x02000000)
143 #define MPI2_TOOLBOX_CLEAN_INITIALIZATION (0x01000000)
144 #define MPI2_TOOLBOX_CLEAN_FLASH (0x00000004)
145 #define MPI2_TOOLBOX_CLEAN_SEEPROM (0x00000002)
146 #define MPI2_TOOLBOX_CLEAN_NVSRAM (0x00000001)

149 /*****
150 * Toolbox Memory Move request
151 *****/

153 typedef struct _MPI2_TOOLBOX_MEM_MOVE_REQUEST
154 {
155     U8           Tool;               /* 0x00 */
156     U8           Reserved1;          /* 0x01 */
157     U8           ChainOffset;        /* 0x02 */
158     U8           Function;           /* 0x03 */
159     U16          Reserved2;          /* 0x04 */
160     U8           Reserved3;          /* 0x06 */
161     U8           MsgFlags;           /* 0x07 */
162     U8           VP_ID;              /* 0x08 */
163     U8           VF_ID;              /* 0x09 */
164     U16          Reserved4;          /* 0x0A */
165     MPI2_SGE_SIMPLE_UNION SGL;       /* 0x0C */
166 } MPI2_TOOLBOX_MEM_MOVE_REQUEST, MPI2_POINTER PTR_MPI2_TOOLBOX_MEM_MOVE_REQUEST,
167   MPI2ToolboxMemMoveRequest_t, MPI2_POINTER pMPI2ToolboxMemMoveRequest_t;

170 /*****
171 * Toolbox Diagnostic Data Upload request
172 *****/

174 typedef struct _MPI2_TOOLBOX_DIAG_DATA_UPLOAD_REQUEST
175 {
176     U8           Tool;               /* 0x00 */
177     U8           Reserved1;          /* 0x01 */
178     U8           ChainOffset;        /* 0x02 */
179     U8           Function;           /* 0x03 */
180     U16          Reserved2;          /* 0x04 */
181     U8           Reserved3;          /* 0x06 */
182     U8           MsgFlags;           /* 0x07 */
183     U8           VP_ID;              /* 0x08 */
184     U8           VF_ID;              /* 0x09 */
185     U16          Reserved4;          /* 0x0A */
186     U8           SGLFlags;           /* 0x0C */
187     U8           Reserved5;          /* 0x0D */
188     U16          Reserved6;          /* 0x0E */
189     U32          Flags;              /* 0x10 */
190     U32          DataLength;         /* 0x14 */
191     MPI2_SGE_SIMPLE_UNION SGL;       /* 0x18 */
192 } MPI2_TOOLBOX_DIAG_DATA_UPLOAD_REQUEST,
193   MPI2_POINTER PTR_MPI2_TOOLBOX_DIAG_DATA_UPLOAD_REQUEST,

```

```

194   MPI2ToolboxDiagDataUploadRequest_t,
195   MPI2_POINTER pMPI2ToolboxDiagDataUploadRequest_t;

197 /* use MPI2_SGLFLAGS_ defines from mpi2.h for the SGLFlags field */

200 typedef struct _MPI2_DIAG_DATA_UPLOAD_HEADER
201 {
202     U32          DiagDataLength;     /* 00h */
203     U8           FormatCode;          /* 04h */
204     U8           Reserved1;          /* 05h */
205     U16          Reserved2;          /* 06h */
206 } MPI2_DIAG_DATA_UPLOAD_HEADER, MPI2_POINTER PTR_MPI2_DIAG_DATA_UPLOAD_HEADER,
207   MPI2DiagDataUploadHeader_t, MPI2_POINTER pMPI2DiagDataUploadHeader_t;

210 /*****
211 * Toolbox ISTWI Read Write Tool
212 *****/

214 /* Toolbox ISTWI Read Write Tool request message */
215 typedef struct _MPI2_TOOLBOX_ISTWI_READ_WRITE_REQUEST
216 {
217     U8           Tool;               /* 0x00 */
218     U8           Reserved1;          /* 0x01 */
219     U8           ChainOffset;        /* 0x02 */
220     U8           Function;           /* 0x03 */
221     U16          Reserved2;          /* 0x04 */
222     U8           Reserved3;          /* 0x06 */
223     U8           MsgFlags;           /* 0x07 */
224     U8           VP_ID;              /* 0x08 */
225     U8           VF_ID;              /* 0x09 */
226     U16          Reserved4;          /* 0x0A */
227     U32          Reserved5;          /* 0x0C */
228     U32          Reserved6;          /* 0x10 */
229     U8           DevIndex;           /* 0x14 */
230     U8           Action;              /* 0x15 */
231     U8           SGLFlags;           /* 0x16 */
232     U8           Reserved7;          /* 0x17 */
233     U16          TxDataLength;        /* 0x18 */
234     U16          RxDataLength;        /* 0x1A */
235     U32          Reserved8;          /* 0x1C */
236     U32          Reserved9;          /* 0x20 */
237     U32          Reserved10;         /* 0x24 */
238     U32          Reserved11;         /* 0x28 */
239     U32          Reserved12;         /* 0x2C */
240     MPI2_SGE_SIMPLE_UNION SGL;       /* 0x30 */
241 } MPI2_TOOLBOX_ISTWI_READ_WRITE_REQUEST,
242   MPI2_POINTER PTR_MPI2_TOOLBOX_ISTWI_READ_WRITE_REQUEST,
243   MPI2ToolboxIstwiReadWriteRequest_t,
244   MPI2_POINTER pMPI2ToolboxIstwiReadWriteRequest_t;

246 /* values for the Action field */
247 #define MPI2_TOOL_ISTWI_ACTION_READ_DATA (0x01)
248 #define MPI2_TOOL_ISTWI_ACTION_WRITE_DATA (0x02)
249 #define MPI2_TOOL_ISTWI_ACTION_SEQUENCE (0x03)
250 #define MPI2_TOOL_ISTWI_ACTION_RESERVE_BUS (0x10)
251 #define MPI2_TOOL_ISTWI_ACTION_RELEASE_BUS (0x11)
252 #define MPI2_TOOL_ISTWI_ACTION_RESET (0x12)

254 /* values for SGLFlags field are in the SGL section of mpi2.h */

257 /* Toolbox ISTWI Read Write Tool reply message */
258 typedef struct _MPI2_TOOLBOX_ISTWI_REPLY
259 {

```

```

260 U8 Tool; /* 0x00 */
261 U8 Reserved1; /* 0x01 */
262 U8 MsgLength; /* 0x02 */
263 U8 Function; /* 0x03 */
264 U16 Reserved2; /* 0x04 */
265 U8 Reserved3; /* 0x06 */
266 U8 MsgFlags; /* 0x07 */
267 U8 VP_ID; /* 0x08 */
268 U8 VF_ID; /* 0x09 */
269 U16 Reserved4; /* 0x0A */
270 U16 Reserved5; /* 0x0C */
271 U16 IOCStatus; /* 0x0E */
272 U32 IOCLogInfo; /* 0x10 */
273 U8 DevIndex; /* 0x14 */
274 U8 Action; /* 0x15 */
275 U8 IstwiStatus; /* 0x16 */
276 U8 Reserved6; /* 0x17 */
277 U16 TxDataCount; /* 0x18 */
278 U16 RxDataCount; /* 0x1A */
279 } MPI2_TOOLBOX_ISTWI_REPLY, MPI2_POINTER PTR_MPI2_TOOLBOX_ISTWI_REPLY,
280 Mpi2ToolboxIstwiReply_t, MPI2_POINTER pMpi2ToolboxIstwiReply_t;

283 /*****
284 * Toolbox Beacon Tool request
285 *****/

287 typedef struct _MPI2_TOOLBOX_BEACON_REQUEST
288 {
289 U8 Tool; /* 0x00 */
290 U8 Reserved1; /* 0x01 */
291 U8 ChainOffset; /* 0x02 */
292 U8 Function; /* 0x03 */
293 U16 Reserved2; /* 0x04 */
294 U8 Reserved3; /* 0x06 */
295 U8 MsgFlags; /* 0x07 */
296 U8 VP_ID; /* 0x08 */
297 U8 VF_ID; /* 0x09 */
298 U16 Reserved4; /* 0x0A */
299 U8 Reserved5; /* 0x0C */
300 U8 PhysicalPort; /* 0x0D */
301 U8 Reserved6; /* 0x0E */
302 U8 Flags; /* 0x0F */
303 } MPI2_TOOLBOX_BEACON_REQUEST, MPI2_POINTER PTR_MPI2_TOOLBOX_BEACON_REQUEST,
304 Mpi2ToolboxBeaconRequest_t, MPI2_POINTER pMpi2ToolboxBeaconRequest_t;

306 /* values for the Flags field */
307 #define MPI2_TOOLBOX_FLAGS_BEACONMODE_OFF (0x00)
308 #define MPI2_TOOLBOX_FLAGS_BEACONMODE_ON (0x01)

311 /*****
312 * Toolbox Diagnostic CLI Tool
313 *****/

315 #define MPI2_TOOLBOX_DIAG_CLI_CMD_LENGTH (0x5C)

317 /* Toolbox Diagnostic CLI Tool request message */
318 typedef struct _MPI2_TOOLBOX_DIAGNOSTIC_CLI_REQUEST
319 {
320 U8 Tool; /* 0x00 */
321 U8 Reserved1; /* 0x01 */
322 U8 ChainOffset; /* 0x02 */
323 U8 Function; /* 0x03 */
324 U16 Reserved2; /* 0x04 */
325 U8 Reserved3; /* 0x06 */

```

```

326 U8 MsgFlags; /* 0x07 */
327 U8 VP_ID; /* 0x08 */
328 U8 VF_ID; /* 0x09 */
329 U16 Reserved4; /* 0x0A */
330 U8 SGLFlags; /* 0x0C */
331 U8 Reserved5; /* 0x0D */
332 U16 Reserved6; /* 0x0E */
333 U32 DataLength; /* 0x10 */
334 U8 DiagnosticCliCommand[MPI2_TOOLBOX_DIAG_CLI_CMD_LENGTH]; /* 0x70 */
335 MPI2_SGE_SIMPLE UNION SGL;
336 } MPI2_TOOLBOX_DIAGNOSTIC_CLI_REQUEST,
337 MPI2_POINTER PTR_MPI2_TOOLBOX_DIAGNOSTIC_CLI_REQUEST,
338 Mpi2ToolboxDiagnosticCliRequest_t,
339 MPI2_POINTER pMpi2ToolboxDiagnosticCliRequest_t;

341 /* use MPI2_SGLFLAGS_ defines from mpi2.h for the SGLFlags field */

344 /* Toolbox Diagnostic CLI Tool reply message */
345 typedef struct _MPI2_TOOLBOX_DIAGNOSTIC_CLI_REPLY
346 {
347 U8 Tool; /* 0x00 */
348 U8 Reserved1; /* 0x01 */
349 U8 MsgLength; /* 0x02 */
350 U8 Function; /* 0x03 */
351 U16 Reserved2; /* 0x04 */
352 U8 Reserved3; /* 0x06 */
353 U8 MsgFlags; /* 0x07 */
354 U8 VP_ID; /* 0x08 */
355 U8 VF_ID; /* 0x09 */
356 U16 Reserved4; /* 0x0A */
357 U16 Reserved5; /* 0x0C */
358 U16 IOCStatus; /* 0x0E */
359 U32 IOCLogInfo; /* 0x10 */
360 U32 ReturnedDataLength; /* 0x14 */
361 } MPI2_TOOLBOX_DIAGNOSTIC_CLI_REPLY,
362 MPI2_POINTER PTR_MPI2_TOOLBOX_DIAG_CLI_REPLY,
363 Mpi2ToolboxDiagnosticCliReply_t,
364 MPI2_POINTER pMpi2ToolboxDiagnosticCliReply_t;

367 /*****
368 *
369 * Diagnostic Buffer Messages
370 *
371 *****/

374 /*****
375 * Diagnostic Buffer Post request
376 *****/

378 typedef struct _MPI2_DIAG_BUFFER_POST_REQUEST
379 {
380 U8 ExtendedType; /* 0x00 */
381 U8 BufferType; /* 0x01 */
382 U8 ChainOffset; /* 0x02 */
383 U8 Function; /* 0x03 */
384 U16 Reserved2; /* 0x04 */
385 U8 Reserved3; /* 0x06 */
386 U8 MsgFlags; /* 0x07 */
387 U8 VP_ID; /* 0x08 */
388 U8 VF_ID; /* 0x09 */
389 U16 Reserved4; /* 0x0A */
390 U64 BufferAddress; /* 0x0C */
391 U32 BufferLength; /* 0x14 */

```

```

392     U32             Reserved5;                /* 0x18 */
393     U32             Reserved6;                /* 0x1C */
394     U32             Flags;                    /* 0x20 */
395     U32             ProductSpecific[23];      /* 0x24 */
396 } MPI2_DIAG_BUFFER_POST_REQUEST, MPI2_POINTER PTR_MPI2_DIAG_BUFFER_POST_REQUEST,
397   Mpi2DiagBufferPostRequest_t, MPI2_POINTER pMpi2DiagBufferPostRequest_t;

399 /* values for the ExtendedType field */
400 #define MPI2_DIAG_EXTENDED_TYPE_UTILIZATION (0x02)

402 /* values for the BufferType field */
403 #define MPI2_DIAG_BUF_TYPE_TRACE (0x00)
404 #define MPI2_DIAG_BUF_TYPE_SNAPSHOT (0x01)
405 #define MPI2_DIAG_BUF_TYPE_EXTENDED (0x02)
406 /* count of the number of buffer types */
407 #define MPI2_DIAG_BUF_TYPE_COUNT (0x03)

409 /* values for the Flags field */
410 #define MPI2_DIAG_BUF_FLAG_RELEASE_ON_FULL (0x00000002) /* for MPI v2.0
411 #define MPI2_DIAG_BUF_FLAG_IMMEDIATE_RELEASE (0x00000001)

413 /*****
414  * Diagnostic Buffer Post reply
415  *****/

417 typedef struct _MPI2_DIAG_BUFFER_POST_REPLY
418 {
419     U8             ExtendedType;                /* 0x00 */
420     U8             BufferType;                  /* 0x01 */
421     U8             MsgLength;                  /* 0x02 */
422     U8             Function;                   /* 0x03 */
423     U16            Reserved2;                  /* 0x04 */
424     U8             Reserved3;                  /* 0x06 */
425     U8             MsgFlags;                   /* 0x07 */
426     U8             VP_ID;                      /* 0x08 */
427     U8             VF_ID;                      /* 0x09 */
428     U16            Reserved4;                  /* 0x0A */
429     U16            Reserved5;                  /* 0x0C */
430     U16            IOCStatus;                  /* 0x0E */
431     U32            IOCLogInfo;                 /* 0x10 */
432     U32            TransferLength;             /* 0x14 */
433 } MPI2_DIAG_BUFFER_POST_REPLY, MPI2_POINTER PTR_MPI2_DIAG_BUFFER_POST_REPLY,
434   Mpi2DiagBufferPostReply_t, MPI2_POINTER pMpi2DiagBufferPostReply_t;

437 /*****
438  * Diagnostic Release request
439  *****/

441 typedef struct _MPI2_DIAG_RELEASE_REQUEST
442 {
443     U8             Reserved1;                /* 0x00 */
444     U8             BufferType;                /* 0x01 */
445     U8             ChainOffset;              /* 0x02 */
446     U8             Function;                 /* 0x03 */
447     U16            Reserved2;                /* 0x04 */
448     U8             Reserved3;                /* 0x06 */
449     U8             MsgFlags;                 /* 0x07 */
450     U8             VP_ID;                    /* 0x08 */
451     U8             VF_ID;                    /* 0x09 */
452     U16            Reserved4;                /* 0x0A */
453 } MPI2_DIAG_RELEASE_REQUEST, MPI2_POINTER PTR_MPI2_DIAG_RELEASE_REQUEST,
454   Mpi2DiagReleaseRequest_t, MPI2_POINTER pMpi2DiagReleaseRequest_t;

457 /*****

```

```

458  * Diagnostic Buffer Post reply
459  *****/

461 typedef struct _MPI2_DIAG_RELEASE_REPLY
462 {
463     U8             Reserved1;                /* 0x00 */
464     U8             BufferType;                /* 0x01 */
465     U8             MsgLength;                /* 0x02 */
466     U8             Function;                 /* 0x03 */
467     U16            Reserved2;                /* 0x04 */
468     U8             Reserved3;                /* 0x06 */
469     U8             MsgFlags;                 /* 0x07 */
470     U8             VP_ID;                    /* 0x08 */
471     U8             VF_ID;                    /* 0x09 */
472     U16            Reserved4;                /* 0x0A */
473     U16            Reserved5;                /* 0x0C */
474     U16            IOCStatus;                /* 0x0E */
475     U32            IOCLogInfo;               /* 0x10 */
476 } MPI2_DIAG_RELEASE_REPLY, MPI2_POINTER PTR_MPI2_DIAG_RELEASE_REPLY,
477   Mpi2DiagReleaseReply_t, MPI2_POINTER pMpi2DiagReleaseReply_t;

480 #endif

482 #endif /* ! codereview */

```

new/usr/src/uts/common/sys/scsi/adapters/mpt_sas3/mpi/mpi2_type.h

1

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

```
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */
21
22 /*
23  * Copyright (c) 2000-2012 LSI Corporation.
24  *
25  * Redistribution and use in source and binary forms of all code within
26  * this file that is exclusively owned by LSI, with or without
27  * modification, is permitted provided that, in addition to the CDDL 1.0
28  * License requirements, the following conditions are met:
29  *
30  * Neither the name of the author nor the names of its contributors may be
31  * used to endorse or promote products derived from this software without
32  * specific prior written permission.
33  *
34  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
35  * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
36  * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
37  * FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
38  * COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
39  * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
40  * BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
41  * OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
42  * AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
43  * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
44  * OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
45  * DAMAGE.
46 */
47
48 /*
49  *      Name:  mpi2_type.h
50  *      Title: MPI basic type definitions
51  *      Creation Date: August 16, 2006
52  *
53  *      mpi2_type.h Version: 02.00.00
54  *
55  *      Version History
56  *      -----
57  *
58  *      Date      Version      Description
59  *      -----
60  *      04-30-07  02.00.00  Corresponds to Fusion-MPT MPI Specification Rev A.
61  *      -----
```

new/usr/src/uts/common/sys/scsi/adapters/mpt_sas3/mpi/mpi2_type.h

2

62 */

64 #ifndef MPI2_TYPE_H

65 #define MPI2_TYPE_H

```
68 /*****
69  * Define MPI2_POINTER if it hasn't already been defined. By default
70  * MPI2_POINTER is defined to be a near pointer. MPI2_POINTER can be defined as
71  * a far pointer by defining MPI2_POINTER as "far *" before this header file is
72  * included.
73  */
```

74 #ifndef MPI2_POINTER

75 #define MPI2_POINTER *

76 #endif

78 /* the basic types may have already been included by mpi_type.h */

79 #ifndef MPI_TYPE_H

80 /*****

81 *
82 * Basic Types

83 *

84 *****/

86 typedef signed char S8;

87 typedef unsigned char U8;

88 typedef signed short S16;

89 typedef unsigned short U16;

92 #if defined(unix) || defined(__arm) || defined(ALPHA) || defined(__PPC__) || def

94 typedef signed int S32;

95 typedef unsigned int U32;

97 #else

99 typedef signed long S32;

100 typedef unsigned long U32;

102 #endif

105 typedef struct _S64

106 {

107 U32 Low;

108 S32 High;

109 } S64;

111 typedef struct _U64

112 {

113 U32 Low;

114 U32 High;

115 } U64;

118 /*****

119 *

120 * Pointer Types

121 *

122 *****/

124 typedef S8 *PS8;

125 typedef U8 *PU8;

126 typedef S16 *PS16;

127 typedef U16 *PU16;

new/usr/src/uts/common/sys/scsi/adapters/mpt_sas3/mpi/mpi2_type.h

3

```
128 typedef S32      *PS32;  
129 typedef U32      *PU32;  
130 typedef S64      *PS64;  
131 typedef U64      *PU64;  
  
133 #endif  
  
135 #endif  
  
137 #endif /* ! codereview */
```

new/usr/src/uts/common/sys/scsi/adapters/mpt_sas3/mptsas3_hash.h

1

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

```
1  /*
2  * This file and its contents are supplied under the terms of the
3  * Common Development and Distribution License ("CDDL"), version 1.0.
4  * You may only use this file in accordance with the terms of version
5  * 1.0 of the CDDL.
6  *
7  * A full copy of the text of the CDDL should have accompanied this
8  * source. A copy of the CDDL is also available via the Internet at
9  * http://www.illumos.org/license/CDDL.
10 */

12 /*
13  * Copyright 2014 Joyent, Inc. All rights reserved.
14  * Copyright (c) 2014, Tegile Systems Inc. All rights reserved.
15  */

17 #ifndef _SYS_SCSI_ADAPTERS_MPTSAS3_HASH_H
18 #define _SYS_SCSI_ADAPTERS_MPTSAS3_HASH_H

20 #include <sys/types.h>
21 #include <sys/list.h>

23 #define RHL_F_DEAD      0x01

25 typedef struct rehash_link {
26     list_node_t rhl_chain_link;
27     list_node_t rhl_global_link;
28     uint_t rhl_flags;
29     uint_t rhl_refcnt;
30 } rehash_link_t;

32 typedef uint64_t (*rehash_hash_f)(const void *);
33 typedef int (*rehash_cmp_f)(const void *, const void *);
34 typedef void (*rehash_dtor_f)(void *);
35 typedef int (*rehash_eval_f)(const void *, void *);

37 typedef struct rehash {
38     list_t *rh_buckets;
39     uint_t rh_bucket_count;
40     list_t rh_objs;
41     size_t rh_obj_size;      /* used by mdb */
42     size_t rh_link_off;
43     size_t rh_tag_off;
44     rehash_hash_f rh_hash;
45     rehash_cmp_f rh_cmp;
46     rehash_dtor_f rh_dtor;
47 } rehash_t;

49 extern rehash_t *rehash_create(uint_t, rehash_hash_f, rehash_cmp_f,
50     rehash_dtor_f, size_t, size_t, int);
51 extern void rehash_destroy(rehash_t *);
52 extern void rehash_insert(rehash_t *, void *);
53 extern void rehash_remove(rehash_t *, void *);
54 extern void *rehash_lookup(rehash_t *, const void *);
55 extern void *rehash_linear_search(rehash_t *, rehash_eval_f, void *);
56 extern void rehash_hold(rehash_t *, void *);
57 extern void rehash_rele(rehash_t *, void *);
58 extern void *rehash_first(rehash_t *);
59 extern void *rehash_next(rehash_t *, void *);
60 extern boolean_t rehash_obj_valid(rehash_t *hp, const void *);
```

new/usr/src/uts/common/sys/scsi/adapters/mpt_sas3/mptsas3_hash.h

2

```
62 #endif /* _SYS_SCSI_ADAPTERS_MPTSAS3_HASH_H */
63 #endif /* ! codereview */
```

new/usr/src/uts/common/sys/scsi/adapters/mpt_sas3/mptsas3_ioctl.h

1

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

```
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */
21
22 /*
23 * Copyright 2010 Sun Microsystems, Inc. All rights reserved.
24 * Use is subject to license terms.
25 */
26 /*
27 * Copyright (c) 2013, Joyent, Inc. All rights reserved.
28 * Copyright (c) 2014, Tegile Systems Inc. All rights reserved.
29 */
30
31 /*
32 * Copyright (c) 2000 to 2010, LSI Corporation.
33 * All rights reserved.
34 *
35 * Redistribution and use in source and binary forms of all code within
36 * this file that is exclusively owned by LSI, with or without
37 * modification, is permitted provided that, in addition to the CDDL 1.0
38 * License requirements, the following conditions are met:
39 *
40 * Neither the name of the author nor the names of its contributors may be
41 * used to endorse or promote products derived from this software without
42 * specific prior written permission.
43 *
44 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
45 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
46 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
47 * FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
48 * COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
49 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
50 * BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
51 * OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
52 * AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
53 * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
54 * OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
55 * DAMAGE.
56 */
57
58 #ifndef _MPTSAS3_IOCTL_H
59 #define _MPTSAS3_IOCTL_H
60
61 #ifdef __cplusplus
```

new/usr/src/uts/common/sys/scsi/adapters/mpt_sas3/mptsas3_ioctl.h

2

```
62 extern "C" {
63 #endif
```

```
65 #include <sys/types.h>
```

```
67 #define MPTIOCTL (('I' << 8)
68 #define MPTIOCTL_GET_ADAPTER_DATA (MPTIOCTL | 1)
69 #define MPTIOCTL_UPDATE_FLASH (MPTIOCTL | 2)
70 #define MPTIOCTL_RESET_ADAPTER (MPTIOCTL | 3)
71 #define MPTIOCTL_PASS_THRU (MPTIOCTL | 4)
72 #define MPTIOCTL_EVENT_QUERY (MPTIOCTL | 5)
73 #define MPTIOCTL_EVENT_ENABLE (MPTIOCTL | 6)
74 #define MPTIOCTL_EVENT_REPORT (MPTIOCTL | 7)
75 #define MPTIOCTL_GET_PCI_INFO (MPTIOCTL | 8)
76 #define MPTIOCTL_DIAG_ACTION (MPTIOCTL | 9)
77 #define MPTIOCTL_REG_ACCESS (MPTIOCTL | 10)
78 #define MPTIOCTL_GET_DISK_INFO (MPTIOCTL | 11)
79 #define MPTIOCTL_LED_CONTROL (MPTIOCTL | 12)
```

```
81 /*
82 * The following are our ioctl() return status values. If everything went
83 * well, we return good status. If the buffer length sent to us is too short
84 * we return a status to tell the user.
85 */
86 #define MPTIOCTL_STATUS_GOOD 0
87 #define MPTIOCTL_STATUS_LEN_TOO_SHORT 1
```

```
89 typedef struct mptsas_pci_bits
```

```
90 {
91     union {
92         struct {
93             uint32_t DeviceNumber :5;
94             uint32_t FunctionNumber :3;
95             uint32_t BusNumber :24;
96         } bits;
97         uint32_t AsDWORD;
98     } u;
99     uint32_t PciSegmentId;
100 } mptsas_pci_bits_t;
```

```
101 /*
102 * The following is the MPTIOCTL_GET_ADAPTER_DATA data structure. This data
103 * structure is setup so that we hopefully are properly aligned for both
104 * 32-bit and 64-bit mode applications.
105 */
```

```
106 * Adapter Type - Value = 6 = SCSI Protocol through SAS-3 adapter
107 *
108 * MPI Port Number - The PCI Function number for this device
109 *
110 * PCI Device HW Id - The PCI device number for this device
111 *
```

```
112 */
113 #define MPTIOCTL_ADAPTER_TYPE_SAS3 6
114 typedef struct mptsas_adapter_data
```

```
115 {
116     uint32_t StructureLength;
117     uint32_t AdapterType;
118     uint32_t MpiPortNumber;
119     uint32_t PciDeviceHwId;
120     uint32_t PciDeviceHwRev;
121     uint32_t SubSystemId;
122     uint32_t SubsystemVendorId;
123     uint32_t Reserved1;
124     uint32_t MpiFirmwareVersion;
125     uint32_t BiosVersion;
126     uint8_t DriverVersion[32];
127     uint8_t Reserved2;
```

```

128     uint8_t          ScsiId;
129     uint16_t         Reserved3;
130     mptsas_pci_bits_t PciInformation;
131 } mptsas_adapter_data_t;

134 typedef struct mptsas_update_flash
135 {
136     uint64_t          PtrBuffer;
137     uint32_t          ImageChecksum;
138     uint32_t          ImageOffset;
139     uint32_t          ImageSize;
140     uint32_t          ImageType;
141 } mptsas_update_flash_t;

144 #define MPTSAS_PASS_THRU_DIRECTION_NONE    0
145 #define MPTSAS_PASS_THRU_DIRECTION_READ    1
146 #define MPTSAS_PASS_THRU_DIRECTION_WRITE   2
147 #define MPTSAS_PASS_THRU_DIRECTION_BOTH    3

149 typedef struct mptsas_pass_thru
150 {
151     uint64_t          PtrRequest;
152     uint64_t          PtrReply;
153     uint64_t          PtrData;
154     uint32_t          RequestSize;
155     uint32_t          ReplySize;
156     uint32_t          DataSize;
157     uint32_t          DataDirection;
158     uint64_t          PtrDataOut;
159     uint32_t          DataOutSize;
160     uint32_t          Timeout;
161 } mptsas_pass_thru_t;

164 /*
165  * Event queue defines
166  */
167 #define MPTSAS_EVENT_QUEUE_SIZE             (50) /* Max Events stored in driver */
168 #define MPTSAS_MAX_EVENT_DATA_LENGTH        (48) /* Size of each event in Dwords */

170 typedef struct mptsas_event_query
171 {
172     uint16_t          Entries;
173     uint16_t          Reserved;
174     uint32_t          Types[4];
175 } mptsas_event_query_t;

177 typedef struct mptsas_event_enable
178 {
179     uint32_t          Types[4];
180 } mptsas_event_enable_t;

182 /*
183  * Event record entry for ioctl.
184  */
185 typedef struct mptsas_event_entry
186 {
187     uint32_t          Type;
188     uint32_t          Number;
189     uint32_t          Data[MPTSAS_MAX_EVENT_DATA_LENGTH];
190 } mptsas_event_entry_t;

192 typedef struct mptsas_event_report
193 {

```

```

194     uint32_t          Size;
195     mptsas_event_entry_t Events[1];
196 } mptsas_event_report_t;

199 typedef struct mptsas_pci_info
200 {
201     uint32_t          BusNumber;
202     uint8_t           DeviceNumber;
203     uint8_t           FunctionNumber;
204     uint16_t          InterruptVector;
205     uint8_t           PciHeader[256];
206 } mptsas_pci_info_t;

209 typedef struct mptsas_diag_action
210 {
211     uint32_t          Action;
212     uint32_t          Length;
213     uint64_t          PtrDiagAction;
214     uint32_t          ReturnCode;
215 } mptsas_diag_action_t;

217 #define MPTSAS_FW_DIAGNOSTIC_UID_NOT_FOUND    (0xFF)

219 #define MPTSAS_FW_DIAG_NEW                    (0x806E6577)

221 #define MPTSAS_FW_DIAG_TYPE_REGISTER          (0x00000001)
222 #define MPTSAS_FW_DIAG_TYPE_UNREGISTER        (0x00000002)
223 #define MPTSAS_FW_DIAG_TYPE_QUERY             (0x00000003)
224 #define MPTSAS_FW_DIAG_TYPE_READ_BUFFER       (0x00000004)
225 #define MPTSAS_FW_DIAG_TYPE_RELEASE           (0x00000005)

227 #define MPTSAS_FW_DIAG_INVALID_UID            (0x00000000)

229 #define MPTSAS_FW_DIAG_ERROR_SUCCESS           (0x00000000)
230 #define MPTSAS_FW_DIAG_ERROR_FAILURE          (0x00000001)
231 #define MPTSAS_FW_DIAG_ERROR_INVALID_PARAMETER (0x00000002)
232 #define MPTSAS_FW_DIAG_ERROR_POST_FAILED      (0x00000010)
233 #define MPTSAS_FW_DIAG_ERROR_INVALID_UID      (0x00000011)
234 #define MPTSAS_FW_DIAG_ERROR_RELEASE_FAILED   (0x00000012)
235 #define MPTSAS_FW_DIAG_ERROR_NO_BUFFER        (0x00000013)
236 #define MPTSAS_FW_DIAG_ERROR_ALREADY_RELEASED (0x00000014)

239 typedef struct mptsas_fw_diag_register
240 {
241     uint8_t           ExtendedType;
242     uint8_t           BufferType;
243     uint16_t          ApplicationFlags;
244     uint32_t          DiagnosticFlags;
245     uint32_t          ProductSpecific[23];
246     uint32_t          RequestedBufferSize;
247     uint32_t          UniqueId;
248 } mptsas_fw_diag_register_t;

250 typedef struct mptsas_fw_diag_unregister
251 {
252     uint32_t          UniqueId;
253 } mptsas_fw_diag_unregister_t;

255 #define MPTSAS_FW_DIAG_FLAG_APP_OWNED         (0x0001)
256 #define MPTSAS_FW_DIAG_FLAG_BUFFER_VALID      (0x0002)
257 #define MPTSAS_FW_DIAG_FLAG_FW_BUFFER_ACCESS (0x0004)

259 typedef struct mptsas_fw_diag_query

```



```

260 {
261     uint8_t      ExtendedType;
262     uint8_t      BufferType;
263     uint16_t     ApplicationFlags;
264     uint32_t     DiagnosticFlags;
265     uint32_t     ProductSpecific[23];
266     uint32_t     TotalBufferSize;
267     uint32_t     DriverAddedBufferSize;
268     uint32_t     UniqueId;
269 } mptsas_fw_diag_query_t;

271 typedef struct mptsas_fw_diag_release
272 {
273     uint32_t      UniqueId;
274 } mptsas_fw_diag_release_t;

276 #define MPTSAS_FW_DIAG_FLAG_REREGISTER    (0x0001)
277 #define MPTSAS_FW_DIAG_FLAG_FORCE_RELEASE (0x0002)

279 typedef struct mptsas_diag_read_buffer
280 {
281     uint8_t      Status;
282     uint8_t      Reserved;
283     uint16_t     Flags;
284     uint32_t     StartingOffset;
285     uint32_t     BytesToRead;
286     uint32_t     UniqueId;
287     uint32_t     DataBuffer[1];
288 } mptsas_diag_read_buffer_t;

290 /*
291  * Register Access
292  */
293 #define REG_IO_READ      1
294 #define REG_IO_WRITE     2
295 #define REG_MEM_READ     3
296 #define REG_MEM_WRITE    4

298 typedef struct mptsas_reg_access
299 {
300     uint32_t      Command;
301     uint32_t      RegOffset;
302     uint32_t      RegData;
303 } mptsas_reg_access_t;

305 /*
306  * Disk Toplogy Information
307  */
308 typedef struct mptsas_disk_info
309 {
310     uint64_t      SasAddress;
311     uint16_t      Instance;
312     uint16_t      Enclosure;
313     uint16_t      Slot;
314 } mptsas_disk_info_t;

316 typedef struct mptsas_get_disk_info
317 {
318     uint16_t      DiskCount;
319     mptsas_disk_info_t *PtrDiskInfoArray;
320     uint64_t      DiskInfoArraySize;
321 } mptsas_get_disk_info_t;

323 #ifndef _KERNEL

325 typedef struct mptsas_get_disk_info32

```

```

326 {
327     uint16_t      DiskCount;
328     caddr32_t     PtrDiskInfoArray;
329     uint64_t      DiskInfoArraySize;
330 } mptsas_get_disk_info32_t;

332 #endif /* _KERNEL */

334 /*
335  * LED Control
336  */

338 typedef struct mptsas_led_control
339 {
340     uint8_t      Command;
341     uint16_t     Enclosure;
342     uint16_t     Slot;
343     uint8_t      Led;
344     uint8_t      LedStatus;
345 } mptsas_led_control_t;

347 #define MPTSAS_LEDCTL_FLAG_SET      1
348 #define MPTSAS_LEDCTL_FLAG_GET      2

350 #define MPTSAS_LEDCTL_LED_IDENT      1
351 #define MPTSAS_LEDCTL_LED_FAIL      2
352 #define MPTSAS_LEDCTL_LED_OK2RM      3

354 #define MPTSAS_LEDCTL_LED_MIN      MPTSAS_LEDCTL_LED_IDENT
355 #define MPTSAS_LEDCTL_LED_MAX      MPTSAS_LEDCTL_LED_OK2RM

357 #ifdef __cplusplus
358 }
359 #endif

361 #endif /* _MPTSAS3_IOCTL_H */
362 #endif /* ! codereview */

```

new/usr/src/uts/common/sys/scsi/adapters/mpt_sas3/mptsas3_smhba.h

1

```
*****
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
*****

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

22 /*
23  * Copyright (c) 2009, 2010, Oracle and/or its affiliates. All rights reserved.
24  * Copyright 2013 Nexenta Systems, Inc. All rights reserved.
25  * Copyright (c) 2014, Tegile Systems Inc. All rights reserved.
26 */

28 /*
29  * SM-HBA interfaces/definitions for MPT SAS driver.
30 */

32 #ifndef _MPTSAS3_SMHBA_H
33 #define _MPTSAS3_SMHBA_H
34 #ifdef __cplusplus
35 extern "C" {
36 #endif

38 /* Leverage definition of data_type_t in nvpair.h */
39 #include <sys/nvpair.h>
40 #include <sys/scsi/adapters/mpt_sas3/mptsas3_var.h>

42 #define MPTSAS_NUM_PHYS "num-phys"
43 #define MPTSAS_NUM_PHYS_HBA "num-phys-hba"
44 #define MPTSAS_SMHBA_SUPPORTED "sm-hba-supported"
45 #define MPTSAS_DRV_VERSION "driver-version"
46 #define MPTSAS_HWARE_VERSION "hardware-version"
47 #define MPTSAS_FWARE_VERSION "firmware-version"
48 #define MPTSAS_SUPPORTED_PROTOCOL "supported-protocol"
49 #define MPTSAS_VIRTUAL_PORT "virtual-port"

51 #define MPTSAS_MANUFACTURER "Manufacturer"
52 #define MPTSAS_SERIAL_NUMBER "SerialNumber"
53 #define MPTSAS_MODEL_NAME "ModelName"
54 #define MPTSAS_VARIANT "variant"

56 #define IS_ATAPI_DEVICE(x) ((x) & 0x2000)
57 #define IS_SATA_DEVICE(x) ((x) & 0x80)
58 #define DEVINFO_DIRECT_ATTACHED 0x0800

60 /*
61  * Interfaces to add properties required for SM-HBA
```

new/usr/src/uts/common/sys/scsi/adapters/mpt_sas3/mptsas3_smhba.h

2

```
62 *
63  * _add_xxx_prop() interfaces add only 1 prop that is specified in the args.
64  * _set_xxx_props() interfaces add more than 1 prop for a set of phys/devices.
65 */
66 int mptsas_smhba_setup(mptsas_t *);
67 void mptsas_smhba_show_phy_info(mptsas_t *);
68 void mptsas_smhba_set_all_phy_props(mptsas_t *mpt, dev_info_t *dip,
69     uint8_t phy_nums, mptsas_phymask_t phy_mask, uint16_t *attached_devhdl);
70 void mptsas_smhba_set_one_phy_props(mptsas_t *mpt, dev_info_t *dip,
71     uint8_t phy_id, uint16_t *attached_devhdl);
72 void mptsas_smhba_log_sysevent(mptsas_t *mpt, char *subclass, char *etype,
73     smhba_info_t *phyp);
74 void
75 mptsas_create_phy_stats(mptsas_t *mpt, char *iport, dev_info_t *dip);
76 int mptsas_update_phy_stats(kstat_t *ks, int rw);
77 void mptsas_destroy_phy_stats(mptsas_t *mpt);
78 int mptsas_smhba_phy_init(mptsas_t *mpt);
79 int mptsas_smhba_phy_state_update(mptsas_t *mpt, uint8_t phy);
80 #ifdef __cplusplus
81 }
82 #endif
83 #endif /* _MPTSAS3_SMHBA_H */
84 #endif /* !codereview */
```

new/usr/src/uts/common/sys/scsi/adapters/mpt_sas3/mptsas3_var.h

1

```
*****
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
*****

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

22 /*
23 * Copyright (c) 2009, 2010, Oracle and/or its affiliates. All rights reserved.
24 * Copyright 2014 Nexenta Systems, Inc. All rights reserved.
25 * Copyright (c) 2013, Joyent, Inc. All rights reserved.
26 * Copyright (c) 2014, Tegile Systems Inc. All rights reserved.
27 */

29 /*
30 * Copyright (c) 2000 to 2010, LSI Corporation.
31 * All rights reserved.
32 *
33 * Redistribution and use in source and binary forms of all code within
34 * this file that is exclusively owned by LSI, with or without
35 * modification, is permitted provided that, in addition to the CDDL 1.0
36 * License requirements, the following conditions are met:
37 *
38 *   Neither the name of the author nor the names of its contributors may be
39 *   used to endorse or promote products derived from this software without
40 *   specific prior written permission.
41 *
42 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
43 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
44 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
45 * FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
46 * COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
47 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
48 * BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
49 * OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
50 * AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
51 * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
52 * OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
53 * DAMAGE.
54 */

56 #ifndef _SYS_SCSI_ADAPTERS_MPTSAS3_VAR_H
57 #define _SYS_SCSI_ADAPTERS_MPTSAS3_VAR_H

59 #include <sys/byteorder.h>
60 #include <sys/queue.h>
61 #include <sys/isa_defs.h>
```

new/usr/src/uts/common/sys/scsi/adapters/mpt_sas3/mptsas3_var.h

2

```
62 #include <sys/sunmdi.h>
63 #include <sys/mdi_impldefs.h>
64 #include <sys/scsi/adapters/mpt_sas3/mptsas3_hash.h>
65 #include <sys/scsi/adapters/mpt_sas3/mptsas3_ioctl.h>
66 #include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_tool.h>
67 #include <sys/scsi/adapters/mpt_sas3/mpi/mpi2_cnfg.h>

69 #ifdef __cplusplus
70 extern "C" {
71 #endif

73 /*
74  * Compile options
75  */
76 #ifdef DEBUG
77 #define MPTSAS_DEBUG /* turn on debugging code */
78 #endif /* DEBUG */

80 #define MPTSAS_INITIAL_SOFT_SPACE 4

82 #define MAX_MPI_PORTS 16

84 /*
85  * Note below macro definition and data type definition
86  * are used for phy mask handling, it should be changed
87  * simultaneously.
88  */
89 #define MPTSAS_MAX_PHYS 16
90 typedef uint16_t mptsas_phymask_t;

92 #define MPTSAS_INVALID_DEVHDL 0xffff
93 #define MPTSAS_SATA_GUID "sata-guid"

95 /*
96  * Hash table sizes for SMP targets (i.e., expanders) and ordinary SSP/STP
97  * targets. There's no need to go overboard here, as the ordinary paths for
98  * I/O do not normally require hashed target lookups. These should be good
99  * enough and then some for any fabric within the hardware's capabilities.
100 */
101 #define MPTSAS_SMP_BUCKET_COUNT 23
102 #define MPTSAS_TARGET_BUCKET_COUNT 97

104 /*
105  * MPT HW defines
106  */
107 #define MPTSAS_MAX_DISKS_IN_CONFIG 14
108 #define MPTSAS_MAX_DISKS_IN_VOL 10
109 #define MPTSAS_MAX_HOTSPARES 2
110 #define MPTSAS_MAX_RAIDVOLS 2
111 #define MPTSAS_MAX_RAIDCONFIGS 5

113 /*
114  * 64-bit SAS WWN is displayed as 16 characters as HEX characters,
115  * plus two means the prefix 'w' and end of the string '\0'.
116  */
117 #define MPTSAS_WWN_STRLEN (16 + 2)
118 #define MPTSAS_MAX_GUID_LEN 64

120 /*
121  * DMA routine flags
122  */
123 #define MPTSAS_DMA_HANDLE_ALLOCD 0x2
124 #define MPTSAS_DMA_MEMORY_ALLOCD 0x4
125 #define MPTSAS_DMA_HANDLE_BOUND 0x8

127 /*
```

```

128 * If the HBA supports DMA or bus-mastering, you may have your own
129 * scatter-gather list for physically non-contiguous memory in one
130 * I/O operation; if so, there's probably a size for that list.
131 * It must be placed in the ddi_dma_lim_t structure, so that the system
132 * DMA-support routines can use it to break up the I/O request, so we
133 * define it here.
134 */
135 #if defined(__sparc)
136 #define MPTSAS_MAX_DMA_SEGS      1
137 #define MPTSAS_MAX_CMD_SEGS      1
138 #else
139 #define MPTSAS_MAX_DMA_SEGS      256
140 #define MPTSAS_MAX_CMD_SEGS      257
141 #endif
142 #define MPTSAS_MAX_FRAME_SGES(mpt) \
143     (((mpt->m_req_frame_size - (sizeof (MPI2_SCSI_IO_REQUEST))) / 8) + 1)

145 /*
146 * Calculating how many 64-bit DMA simple elements can be stored in the first
147 * frame. Note that msg_scsi_io_request contains 2 double-words (8 bytes) for
148 * element storage. And 64-bit dma element is 3 double-words (12 bytes) in
149 * size. IEEE 64-bit dma element used for SAS3 controllers is 4 double-words
150 * (16 bytes).
151 */
152 #define MPTSAS_MAX_FRAME_SGES64(mpt) \
153     ((mpt->m_req_frame_size - \
154      sizeof (MPI2_SCSI_IO_REQUEST) + sizeof (MPI2_SGE_IO_UNION)) / \
155      (mpt->m_MPI25 ? sizeof (MPI2_IEEE_SGE_SIMPLE64) : \
156       sizeof (MPI2_SGE_SIMPLE64)))

158 /*
159 * Scatter-gather list structure defined by HBA hardware
160 */
161 typedef struct NcrTableIndirect { /* Table Indirect entries */
162     uint32_t count; /* 24 bit count */
163     union {
164         uint32_t address32; /* 32 bit address */
165         struct {
166             uint32_t Low;
167             uint32_t High;
168         } address64; /* 64 bit address */
169     } addr;
170 } mptti_t;

172 /*
173 * preferred pkt_private length in 64-bit quantities
174 */
175 #ifdef __LP64
176 #define PKT_PRIV_SIZE      2
177 #define PKT_PRIV_LEN      16 /* in bytes */
178 #else /* __ILP32 */
179 #define PKT_PRIV_SIZE      1
180 #define PKT_PRIV_LEN      8 /* in bytes */
181 #endif

183 #define PKT2CMD(pkt)      ((struct mptsas_cmd *)((pkt)->pkt_ha_private))
184 #define CMD2PKT(cmdp)    ((struct scsi_pkt *)((cmdp)->cmd_pkt))
185 #define EXTCMDSTATUS_SIZE (sizeof (struct scsi_arq_status))

187 /*
188 * get offset of item in structure
189 */
190 #define MPTSAS_GET_ITEM_OFF(type, member) ((size_t)(&((type *)0)->member))

192 /*
193 * WWID provided by LSI firmware is generated by firmware but the WWID is not

```

```

194 * IEEE NAA standard format, OBP has no chance to distinguish format of unit
195 * address. According LSI's confirmation, the top nibble of RAID WWID is
196 * meaningless, so the consensus between Solaris and OBP is to replace top nibble
197 * of WWID provided by LSI to "3" always to hint OBP that this is a RAID WWID
198 * format unit address.
199 */
200 #define MPTSAS_RAID_WWID(wwid) \
201     (((wwid & 0x0FFFFFFFFFFFFFFF) | 0x3000000000000000))

203 typedef struct mptsas_target_addr {
204     uint64_t mta_wwn;
205     mptsas_phymask_t mta_phymask;
206 } mptsas_target_addr_t;

208 TAILQ_HEAD(mptsas_active_cmdq, mptsas_cmd);
209 typedef struct mptsas_active_cmdq mptsas_active_cmdq_t;

211 typedef struct mptsas_target {
212     kmutex_t m_t_mutex;
213     mptsas_target_addr_t m_addr;
214     rehash_link_t m_link;
215     uint16_t m_devhdl;
216     uint32_t m_deviceinfo;
217     uint32_t m_dups;
218     uint8_t m_phynum;
219     mptsas_active_cmdq_t m_active_cmdq;
220     int32_t m_t_throttle;
221     int32_t m_t_ncmds;
222     int32_t m_reset_delay;
223     int32_t m_t_nwait;
224     uint16_t m_io_flags;
225     uint16_t m_enclosure;
226     uint16_t m_slot_num;
227     uint16_t m_qfull_retry_interval;
228     uint8_t m_qfull_retries;
229     uint8_t m_tgt_unconfigured;
230     uint8_t m_led_status;
231     uint8_t m_dr_flag;
232 } mptsas_target_t;

234 /*
235 * If you change this structure, be sure that mptsas_smp_target_copy()
236 * does the right thing.
237 */
238 typedef struct mptsas_smp {
239     mptsas_target_addr_t m_addr;
240     rehash_link_t m_link;
241     uint16_t m_devhdl;
242     uint32_t m_deviceinfo;
243     uint16_t m_pdevhdl;
244     uint32_t m_pdevinfo;
245 } mptsas_smp_t;

247 typedef struct mptsas_cache_frames {
248     ddi_dma_handle_t m_dma_hdl;
249     ddi_acc_handle_t m_acc_hdl;
250     caddr_t m_frames_addr;
251     uint64_t m_phys_addr;
252 } mptsas_cache_frames_t;

254 typedef struct mptsas_cmd {
255     uint_t cmd_flags; /* flags from scsi_init_pkt */
256     ddi_dma_handle_t cmd_dmahandle; /* dma handle */
257     ddi_dma_cookie_t cmd_cookie;
258     uint_t cmd_cookiec;
259     uint_t cmd_winindex;

```

```

260     uint_t          cmd_nwin;
261     uint_t          cmd_cur_cookie;
262     off_t           cmd_dma_offset;
263     size_t          cmd_dma_len;
264     uint32_t        cmd_totaldmacount;
265     caddr_t         cmd_arq_buf;

267     int             cmd_pkt_flags;

269     /* pending expiration time for command in active slot */
270     hrtime_t        cmd_active_expiration;
271     TAILQ_ENTRY(mptsas_cmd) cmd_active_link;

273     struct scsi_pkt *cmd_pkt;
274     struct scsi_arq_status cmd_scb;
275     uchar_t         cmd_cdblen; /* length of cdb */
276     uchar_t         cmd_rqslens; /* len of requested rqsense */
277     uchar_t         cmd_privlen;
278     uint16_t        cmd_extrqslens; /* len of extended rqsense */
279     uint16_t        cmd_extrqschunks; /* len in map chunks */
280     uint16_t        cmd_extrqsidx; /* Index into map */
281     uint_t          cmd_scblen;
282     uint32_t        cmd_dmacount;
283     uint64_t        cmd_dma_addr;
284     uchar_t         cmd_age;
285     ushort_t        cmd_qfull_retries;
286     uchar_t         cmd_queued; /* true if queued */
287     struct mptsas_cmd *cmd_linkp;
288     mptti_t         *cmd_sg; /* Scatter/Gather structure */
289     uchar_t         cmd_cdb[SCSI_CDB_SIZE];
290     uint64_t        cmd_pkt_private[PKT_PRIV_LEN];
291     uint32_t        cmd_slot;
292     uint32_t        ioc_cmd_slot;
293     uint8_t         cmd_rpqid;

295     mptsas_cache_frames_t *cmd_extra_frames;

297     uint32_t        cmd_rfm;
298     mptsas_target_t *cmd_tgt_addr;
299 } mptsas_cmd_t;

301 /*
302  * These are the defined cmd_flags for this structure.
303  */
304 #define CFLAG_CMDDISC 0x000001 /* cmd currently disconnected */
305 #define CFLAG_WATCH 0x000002 /* watchdog time for this command */
306 #define CFLAG_FINISHED 0x000004 /* command completed */
307 #define CFLAG_CHKSEG 0x000008 /* check cmd data within seg */
308 #define CFLAG_COMPLETED 0x000010 /* completion routine called */
309 #define CFLAG_PREPARED 0x000020 /* pkt has been init'ed */
310 #define CFLAG_IN_TRANSPORT 0x000040 /* in use by host adapter driver */
311 #define CFLAG_RESTORE_PTRS 0x000080 /* implicit restore ptr on reconnect */
312 #define CFLAG_ARQ_IN_PROGRESS 0x000100 /* auto request sense in progress */
313 #define CFLAG_TRANFLAG 0x0001ff /* covers transport part of flags */
314 #define CFLAG_TM_CMD 0x000200 /* cmd is a task management command */
315 #define CFLAG_CMDARQ 0x000400 /* cmd is a 'rqsense' command */
316 #define CFLAG_DMAVALID 0x000800 /* dma mapping valid */
317 #define CFLAG_DMASEND 0x001000 /* data is going 'out' */
318 #define CFLAG_CMDIOPB 0x002000 /* this is an 'iopb' packet */
319 #define CFLAG_CDBEXTERN 0x004000 /* cdb kmem_alloc'd */
320 #define CFLAG_SCBEXTERN 0x008000 /* scb kmem_alloc'd */
321 #define CFLAG_FREE 0x010000 /* packet is on free list */
322 #define CFLAG_PRIVEXTERN 0x020000 /* target private kmem_alloc'd */
323 #define CFLAG_DMA_PARTIAL 0x040000 /* partial xfer OK */
324 #define CFLAG_QFULL_STATUS 0x080000 /* pkt got qfull status */
325 #define CFLAG_TIMEOUT 0x100000 /* passthru/config command timeout */

```

```

326 #define CFLAG_PMM_RECEIVED 0x200000 /* use cmd_pmm* for saving pointers */
327 #define CFLAG_RETRY 0x400000 /* cmd has been retried */
328 #define CFLAG_CMDIOC 0x800000 /* cmd is just for for ioc, no io */
329 #define CFLAG_PASSTHRU 0x2000000 /* cmd is a passthrough command */
330 #define CFLAG_XARQ 0x4000000 /* cmd requests for extra sense */
331 #define CFLAG_CMDACK 0x8000000 /* cmd for event ack */
332 #define CFLAG_TXQ 0x10000000 /* cmd queued in the tx_waitq */
333 #define CFLAG_FW_CMD 0x20000000 /* cmd is a fw up/down command */
334 #define CFLAG_CONFIG 0x40000000 /* cmd is for config header/page */
335 #define CFLAG_FW_DIAG 0x80000000 /* cmd is for FW diag buffers */

337 #ifdef MPTSAS_DEBUG
338 /* Could be used with cmn_err %b */
339 #define CFLAGS_DEBUG_BITS "\020\0CmdDisc\1Watch\2Finished\3ChkSeg" \
340     "\4CmpltD\5Prepd\6InTran\7RestPtrs\8ARQIP\9TM\10Carq" \
341     "\11DMAVal\12DMASnd\13CIopb\14CDBExt\15SCBExt\16Free" \
342     "\17PrivExt\18DMAPrtl\19QFull\20Tout\21PMMRcv\22Retry" \
343     "\23CIoc\25PThru\26XArq\27CAck\28TXq\29FWCmd\30Config\31FWDiag"
344 #endif

346 #define MPTSAS_SCSI_REPORTLUNS_ADDRESS_SIZE 8
347 #define MPTSAS_SCSI_REPORTLUNS_ADDRESS_MASK 0xC0
348 #define MPTSAS_SCSI_REPORTLUNS_ADDRESS_PERIPHERAL 0x00
349 #define MPTSAS_SCSI_REPORTLUNS_ADDRESS_FLAT_SPACE 0x40
350 #define MPTSAS_SCSI_REPORTLUNS_ADDRESS_LOGICAL_UNIT 0x80
351 #define MPTSAS_SCSI_REPORTLUNS_ADDRESS_EXTENDED_UNIT 0xC0
352 #define MPTSAS_SCSI_REPORTLUNS_ADDRESS_LOGICAL_UNIT_2B 0x00
353 #define MPTSAS_SCSI_REPORTLUNS_ADDRESS_LOGICAL_UNIT_4B 0x01
354 #define MPTSAS_SCSI_REPORTLUNS_ADDRESS_LOGICAL_UNIT_6B 0x10
355 #define MPTSAS_SCSI_REPORTLUNS_ADDRESS_LOGICAL_UNIT_8B 0x20
356 #define MPTSAS_SCSI_REPORTLUNS_ADDRESS_LOGICAL_UNIT_SIZE 0x30

358 #define MPTSAS_HASH_ARRAY_SIZE 16
359 /*
360  * hash table definition
361  */

363 #define MPTSAS_HASH_FIRST 0xffff
364 #define MPTSAS_HASH_NEXT 0x0000

366 typedef struct mptsas_dma_alloc_state
367 {
368     ddi_dma_handle_t handle;
369     caddr_t memmp;
370     size_t size;
371     ddi_acc_handle_t accessp;
372     ddi_dma_cookie_t cookie;
373 } mptsas_dma_alloc_state_t;

375 /*
376  * passthrough request structure
377  */
378 typedef struct mptsas_pt_request {
379     uint8_t *request;
380     uint32_t request_size;
381     uint32_t data_size;
382     uint32_t dataout_size;
383     uint8_t direction;
384     uint8_t simple;
385     uint16_t sgl_offset;
386     ddi_dma_cookie_t data_cookie;
387     ddi_dma_cookie_t dataout_cookie;
388 } mptsas_pt_request_t;

390 /*
391  * config page request structure

```

```

392 */
393 typedef struct mptsas_config_request {
394     uint32_t      page_address;
395     uint8_t       action;
396     uint8_t       page_type;
397     uint8_t       page_number;
398     uint8_t       page_length;
399     uint8_t       page_version;
400     uint8_t       ext_page_type;
401     uint16_t      ext_page_length;
402 } mptsas_config_request_t;

403 typedef struct mptsas_fw_diagnostic_buffer {
404     mptsas_dma_alloc_state_t  buffer_data;
405     uint8_t                   extended_type;
406     uint8_t                   buffer_type;
407     uint8_t                   force_release;
408     uint32_t                  product_specific[23];
409     uint8_t                   immediate;
410     uint8_t                   enabled;
411     uint8_t                   valid_data;
412     uint8_t                   owned_by_firmware;
413     uint32_t                  unique_id;
414 } mptsas_fw_diagnostic_buffer_t;

415 /*
416  * FW diag request structure
417  */
418 typedef struct mptsas_diag_request {
419     mptsas_fw_diagnostic_buffer_t *pBuffer;
420     uint8_t                       function;
421 } mptsas_diag_request_t;

422 typedef struct mptsas_hash_node {
423     void *data;
424     struct mptsas_hash_node *next;
425 } mptsas_hash_node_t;

426 typedef struct mptsas_hash_table {
427     struct mptsas_hash_node *head[MPTSAS_HASH_ARRAY_SIZE];
428     /*
429      * last position in traverse
430      */
431     struct mptsas_hash_node *cur;
432     uint16_t line;
433 } mptsas_hash_table_t;

434 /*
435  * RAID volume information
436  */
437 typedef struct mptsas_raidvol {
438     ushort_t      m_israid;
439     uint16_t      m_raidhandle;
440     uint64_t      m_raidwid;
441     uint8_t       m_state;
442     uint32_t      m_statusflags;
443     uint32_t      m_settings;
444     uint16_t      m_devhdl[MPTSAS_MAX_DISKS_IN_VOL];
445     uint8_t       m_disknum[MPTSAS_MAX_DISKS_IN_VOL];
446     ushort_t      m_diskstatus[MPTSAS_MAX_DISKS_IN_VOL];
447     uint64_t      m_raidsize;
448     int           m_raidlevel;
449     int           m_ndisks;
450     mptsas_target_t *m_raidtgt;
451 } mptsas_raidvol_t;

```

```

459 /*
460  * RAID configurations
461  */
462 typedef struct mptsas_raidconfig {
463     mptsas_raidvol_t      m_raidvol[MPTSAS_MAX_RAIDVOLS];
464     uint16_t              m_physdisk_devhdl[
465         MPTSAS_MAX_DISKS_IN_CONFIG];
466     uint8_t               m_native;
467 } m_raidconfig_t;

468 /*
469  * Track outstanding commands. The index into the m_slot array is the SMID
470  * (system message ID) of the outstanding command. SMID 0 is reserved by the
471  * software/firmware protocol and is never used for any command we generate;
472  * as such, the assertion m_slot[0] == NULL is universally true. The last
473  * entry in the array is slot number MPTSAS_TM_SLOT(mpt) and is used ONLY for
474  * task management commands. No normal SCSI or ATA command will ever occupy
475  * that slot. Finally, the relationship m_slot[X]->cmd_slot == X holds at any
476  * time that a consistent view of the target array is obtainable.
477  *
478  * As such, m_n_normal is the maximum number of slots available to ordinary
479  * commands, and the relationship:
480  * mpt->m_active->m_n_normal == mpt->m_max_requests - 2
481  * always holds after initialisation.
482  */
483 typedef struct mptsas_slots {
484     size_t      m_size;           /* size of struct, bytes */
485     uint_t      m_n_normal;       /* see above */
486     uint_t      m_rotor;         /* next slot idx to consider */
487     mptsas_cmd_t m_slot[1];
488 } mptsas_slots_t;

489 /*
490  * Structure to hold command and packets for event ack
491  * and task management commands.
492  */
493 typedef struct m_event_struct {
494     struct mptsas_cmd      m_event_cmd;
495     struct m_event_struct *m_event_linkp;
496     /*
497      * event member record the failure event and eventcntx
498      * event member would be used in send ack pending process
499      */
500     uint32_t      m_event;
501     uint32_t      m_eventcntx;
502     uint_t        in_use;
503     struct scsi_pkt m_event_pkt; /* must be last */
504     /* ... scsi_pkt_size() */
505 } m_event_struct_t;
506 #define M_EVENT_STRUCT_SIZE (sizeof(m_event_struct_t) - \
507     sizeof(struct scsi_pkt) + scsi_pkt_size())

508 #define MAX_IOC_COMMANDS 8

509 /*
510  * A pool of MAX_IOC_COMMANDS is maintained for event ack commands.
511  * A new event ack command requests mptsas_cmd and scsi_pkt structures
512  * from this pool, and returns it back when done.
513  */

514 typedef struct m_replyh_arg {
515     void *mpt;
516     uint32_t rfm;
517 } m_replyh_arg_t;
518 _NOTE(DATA_READABLE_WITHOUT_LOCK(m_replyh_arg_t:mpt))

```

```

524 _NOTE(DATA_READABLE_WITHOUT_LOCK(m_replyh_arg_t::rfm))

526 /*
527  * Flags for DR handler topology change
528  */
529 #define MPTSAS_TOPO_FLAG_DIRECT_ATTACHED_DEVICE      0x0
530 #define MPTSAS_TOPO_FLAG_EXPANDER_ASSOCIATED         0x1
531 #define MPTSAS_TOPO_FLAG_LUN_ASSOCIATED              0x2
532 #define MPTSAS_TOPO_FLAG_RAID_ASSOCIATED             0x4
533 #define MPTSAS_TOPO_FLAG_RAID_PHYSDRV_ASSOCIATED     0x8
534 #define MPTSAS_TOPO_FLAG_EXPANDER_ATTACHED_DEVICE    0x10

536 typedef struct mptsas_topo_change_list {
537     void *mpt;
538     uint_t event;
539     union {
540         uint8_t physport;
541         mptsas_phymask_t phymask;
542     } un;
543     uint16_t devhdl;
544     void *object;
545     uint8_t flags;
546     struct mptsas_topo_change_list *next;
547 } mptsas_topo_change_list_t;

550 _NOTE(DATA_READABLE_WITHOUT_LOCK(mptsas_topo_change_list_t::mpt))
551 _NOTE(DATA_READABLE_WITHOUT_LOCK(mptsas_topo_change_list_t::event))
552 _NOTE(DATA_READABLE_WITHOUT_LOCK(mptsas_topo_change_list_t::physport))
553 _NOTE(DATA_READABLE_WITHOUT_LOCK(mptsas_topo_change_list_t::devhdl))
554 _NOTE(DATA_READABLE_WITHOUT_LOCK(mptsas_topo_change_list_t::object))
555 _NOTE(DATA_READABLE_WITHOUT_LOCK(mptsas_topo_change_list_t::flags))

557 /*
558  * Status types when calling mptsas_get_target_device_info
559  */
560 #define DEV_INFO_SUCCESS      0x0
561 #define DEV_INFO_FAIL_PAGE0   0x1
562 #define DEV_INFO_WRONG_DEVICE_TYPE 0x2
563 #define DEV_INFO_PHYS_DISK    0x3
564 #define DEV_INFO_FAIL_ALLOC   0x4

566 /*
567  * mpt hotplug event defines
568  */
569 #define MPTSAS_DR_EVENT_RECONFIG_TARGET 0x01
570 #define MPTSAS_DR_EVENT_OFFLINE_TARGET 0x02
571 #define MPTSAS_DR_EVENT_REMOVE_HANDLE 0x04

573 /*
574  * SMP target hotplug events
575  */
576 #define MPTSAS_DR_EVENT_RECONFIG_SMP 0x10
577 #define MPTSAS_DR_EVENT_OFFLINE_SMP 0x20
578 #define MPTSAS_DR_EVENT_MASK 0x3F

580 /*
581  * mpt hotplug status definition for m_dr_flag
582  */

584 /*
585  * MPTSAS_DR_INACTIVE
586  */
587 * The target is in a normal operating state.
588 * No dynamic reconfiguration operation is in progress.
589 */

```

```

590 #define MPTSAS_DR_INACTIVE      0x0
591 /*
592  * MPTSAS_DR_INTRANSITION
593  *
594  * The target is in a transition mode since
595  * hotplug event happens and offline procedure has not
596  * been finished
597  */
598 #define MPTSAS_DR_INTRANSITION 0x1

600 typedef struct mptsas_tgt_private {
601     int t_lun;
602     struct mptsas_target *t_private;
603 } mptsas_tgt_private_t;

605 /*
606  * The following defines are used in mptsas_set_init_mode to track the current
607  * state as we progress through reprogramming the HBA from target mode into
608  * initiator mode.
609  */

611 #define IOUC_READ_PAGE0      0x00000100
612 #define IOUC_READ_PAGE1      0x00000200
613 #define IOUC_WRITE_PAGE1     0x00000400
614 #define IOUC_DONE            0x00000800
615 #define DISCOVERY_IN_PROGRESS MPI2_SASIOUNIT0_PORTFLAGS_DISCOVERY_IN_PROGRESS
616 #define AUTO_PORT_CONFIGURATION MPI2_SASIOUNIT0_PORTFLAGS_AUTO_PORT_CONFIG

618 /*
619  * Last allocated slot is used for TM requests. Since only m_max_requests
620  * frames are allocated, the last SMID will be m_max_requests - 1.
621  */
622 #define MPTSAS_SLOTS_SIZE(mpt) \
623     (sizeof (struct mptsas_slots) + (sizeof (struct mptsas_cmd *) * \
624     mpt->m_max_requests))
625 #define MPTSAS_TM_SLOT(mpt) (mpt->m_max_requests - 1)

627 /*
628  * Macro for phy_flags
629  */

631 typedef struct smhba_info {
632     kmutex_t phy_mutex;
633     uint8_t phy_id;
634     uint64_t sas_addr;
635     char path[8];
636     uint16_t owner_devhdl;
637     uint16_t attached_devhdl;
638     uint8_t attached_phy_identify;
639     uint32_t attached_phy_info;
640     uint8_t programmed_link_rate;
641     uint8_t hw_link_rate;
642     uint8_t change_count;
643     uint32_t phy_info;
644     uint8_t negotiated_link_rate;
645     uint8_t port_num;
646     kstat_t *phy_stats;
647     uint32_t invalid_dword_count;
648     uint32_t running_disparity_error_count;
649     uint32_t loss_of_dword_sync_count;
650     uint32_t phy_reset_problem_count;
651     void *mpt;
652 } smhba_info_t;

654 typedef struct mptsas_phy_info {
655     uint8_t port_num;

```

```

656     uint8_t          port_flags;
657     uint16_t         ctrl_devhdl;
658     uint32_t         phy_device_type;
659     uint16_t         attached_devhdl;
660     mptsas_phymask_t  phy_mask;
661     smhba_info_t     smhba_info;
662 } mptsas_phy_info_t;

665 typedef struct mptsas_thread_arg {
666     void            *mpt;
667     uint32_t        t;
668 } mptsas_thread_arg_t;

670 typedef struct mptsas_done_list {
671     mptsas_cmd_t     *dl_q;
672     mptsas_cmd_t     **dl_tail;
673     uint32_t         dl_len;
674 } mptsas_done_list_t;

676 #define MPTSAS_DONEQ_THREAD_ACTIVE    0x1
677 typedef struct mptsas_doneq_thread_list {
678     mptsas_done_list_t    dlist;
679     kthread_t             *threadp;
680     kcondvar_t            cv;
681     ushort_t              reserv1;
682     uint32_t              reserv2;
683     kmutex_t              mutex;
684     uint32_t              flag;
685     mptsas_thread_arg_t   arg;
686 } mptsas_doneq_thread_list_t;

688 typedef struct mptsas_reply_pqueue {
689     kmutex_t              rpq_mutex;
690     uint8_t              rpq_num;
691     caddr_t              rpq_queue; /* Pointer to this queue base */
692     uint32_t             rpq_index; /* Index of next replyq entry */
693     uint32_t             rpq_ncmds; /* Number of outstanding commands */
694     mptsas_done_list_t   rpq_dlist;
695     uint32_t             rpq_intr_count;
696     uint32_t             rpq_intr_unclaimed;
697     uint32_t             rpq_intr_mutexbusy;
698 } mptsas_reply_pqueue_t;

701 typedef struct mptsas_tx_waitqueue {
702     kmutex_t             txwq_mutex;
703     kcondvar_t           txwq_cv;
704     kcondvar_t           txwq_drain_cv;
705     kthread_t            *txwq_threadp;
706     mptsas_cmd_t         *txwq_cmdq; /* TX cmd queue for active request */
707     mptsas_cmd_t         **txwq_qtail; /* tx_wait queue tail ptr */
708     uint32_t             txwq_len; /* TX queue length */
709     mptsas_thread_arg_t   arg;
710     uint8_t             txwq_active; /* Thread active flag */
711     uint8_t             txwq_draining; /* TX queue draining flag */
712     uint8_t             txwq_wdrain;
713 } mptsas_tx_waitqueue_t;

715 #define NUM_TX_WAITQ    2

717 typedef struct mptsas {
718     int            m_instance;

720     struct mptsas *m_next;

```

```

722     scsi_hba_tran_t    *m_tran;
723     smp_hba_tran_t     *m_smptran;
724     kmutex_t           m_mutex;
725     m_passthru_mutex;
726     kcondvar_t         m_cv;
727     kcondvar_t         m_passthru_cv;
728     kcondvar_t         m_fw_cv;
729     kcondvar_t         m_config_cv;
730     kcondvar_t         m_fw_diag_cv;
731     dev_info_t         *m_dip;

733     /*
734      * soft state flags
735      */
736     uint_t             m_softstate;

738     reflash_t          *m_targets;
739     reflash_t          *m_smp_targets;

741     m_raidconfig_t     m_raidconfig[MPTSAS_MAX_RAIDCONFIGS];
742     uint8_t            m_num_raid_configs;
743     uint8_t            m_pref_tx_waitq;

745     struct mptsas_slots *m_active; /* outstanding cmds */

747     mptsas_cmd_t        *m_waitq; /* cmd queue for active request */
748     mptsas_cmd_t        **m_waitqtail; /* wait queue tail ptr */
749     mptsas_tx_waitqueue_t m_tx_waitq[NUM_TX_WAITQ];
750     uint16_t            m_txwq_thread_threshold;
751     uint16_t            m_txwq_thread_n;
752     uint8_t            m_txwq_enabled;
753     uint8_t            m_txwq_allow_q_jumping;
754     mptsas_done_list_t  m_dlist; /* List of completed commands */

756     /*
757      * variables for helper threads (fan-out interrupts)
758      */
759     mptsas_doneq_thread_list_t *m_doneq_thread_id;
760     uint16_t            m_doneq_thread_n;
761     mptsas_doneq_thread_list_t *m_doneq_next_thread;
762     uint32_t            m_doneq_thread_threshold;
763     uint32_t            m_doneq_length_threshold;
764     kcondvar_t          m_qthread_cv;
765     kmutex_t            m_qthread_mutex;

767     uint32_t            m_ncmds; /* number of outstanding commands */
768     uint32_t            m_ncstarted; /* ncmds started per interval */
769     uint32_t            m_lncstarted; /* record of last value */
770     m_event_struct_t    *m_ioc_event_cmdq; /* cmd queue for ioc event */
771     m_event_struct_t    **m_ioc_event_cmdtail; /* ioc cmd queue tail */

773     ddi_acc_handle_t m_datap; /* operating regs data access handle */

775     struct _MPI2_SYSTEM_INTERFACE_REGS *m_reg;

777     ushort_t           m_devid; /* device id of chip. */
778     uchar_t            m_revid; /* revision of chip. */
779     uint16_t           m_svid; /* subsystem Vendor ID of chip */
780     uint16_t           m_ssid; /* subsystem Device ID of chip */

782     uchar_t            m_sync_offset; /* default offset for this chip. */

784     timeout_id_t        m_quiesce_timeid;

786     ddi_dma_handle_t m_dma_req_frame_hdl;
787     ddi_acc_handle_t m_acc_req_frame_hdl;

```



```

788 ddi_dma_handle_t m_dma_req_sense_hdl;
789 ddi_acc_handle_t m_acc_req_sense_hdl;
790 ddi_dma_handle_t m_dma_reply_frame_hdl;
791 ddi_acc_handle_t m_acc_reply_frame_hdl;
792 ddi_dma_handle_t m_dma_free_queue_hdl;
793 ddi_acc_handle_t m_acc_free_queue_hdl;
794 ddi_dma_handle_t m_dma_post_queue_hdl;
795 ddi_acc_handle_t m_acc_post_queue_hdl;
796 uint8_t          m_dma_flags;

798 /*
799  * list of reset notification requests
800  */
801 struct scsi_reset_notify_entry *m_reset_notify_listf;

803 /*
804  * qfull handling
805  */
806 timeout_id_t      m_restart_cmd_timeid;

808 /*
809  * scsi reset delay per bus
810  */
811 uint_t            m_scsi_reset_delay;

813 int               m_pm_idle_delay;

815 uchar_t           m_polled_intr; /* intr was polled. */
816 uchar_t           m_suspended; /* true if driver is suspended */

818 struct kmem_cache *m_kmem_cache;
819 struct kmem_cache *m_cache_frames;

821 /*
822  * hba options.
823  */
824 uint_t            m_options;

826 int               m_power_level; /* current power level */

828 int               m_busy; /* power management busy state */

830 off_t             m_pmcsr_offset; /* PMCSR offset */

832 ddi_acc_handle_t m_config_handle;

834 ddi_dma_attr_t     m_io_dma_attr; /* Used for data I/O */
835 ddi_dma_attr_t     m_msg_dma_attr; /* Used for message frames */
836 ddi_device_acc_attr_t m_dev_acc_attr;
837 ddi_device_acc_attr_t m_reg_acc_attr;

839 /*
840  * request/reply variables
841  */
842 caddr_t            m_req_frame;
843 uint64_t           m_req_frame_dma_addr;
844 caddr_t            m_req_sense;
845 caddr_t            m_extreq_sense;
846 uint64_t           m_req_sense_dma_addr;
847 caddr_t            m_reply_frame;
848 uint64_t           m_reply_frame_dma_addr;
849 caddr_t            m_free_queue;
850 uint64_t           m_free_queue_dma_addr;
851 caddr_t            m_post_queue;
852 uint64_t           m_post_queue_dma_addr;
853 struct map          *m_ergsense_map;

```

```

854 mptsas_reply_pqueue_t *m_rep_post_queues;

856 m_replyh_arg_t *m_replyh_args;

858 uint16_t          m_max_requests;
859 uint16_t          m_req_frame_size;
860 uint16_t          m_req_sense_size;

862 /*
863  * Max frames per request reported in IOC Facts
864  */
865 uint8_t           m_max_chain_depth;
866 /*
867  * Max frames per request which is used in reality. It's adjusted
868  * according DMA SG length attribute, and shall not exceed the
869  * m_max_chain_depth.
870  */
871 uint8_t           m_max_request_frames;
872 uint8_t           m_max_msix_vectors;
873 uint8_t           m_reply_frame_size;
874 uint8_t           m_post_reply_qcount;

876 uint16_t          m_free_queue_depth;
877 uint16_t          m_post_queue_depth;
878 uint16_t          m_max_replies;
879 uint32_t          m_free_index;
880 uint32_t          m_ioc_capabilities;

882 /*
883  * Housekeeping.
884  */
885 uint64_t          m_interrupt_count;
886 uint32_t          m_unclaimed_pm_interrupt_count;
887 uint32_t          m_unclaimed_polled_interrupt_count;
888 uint32_t          m_unclaimed_no_interrupt_count;
889 uint32_t          m_unclaimed_nocmd_interrupt_count;

891 /*
892  * indicates if the firmware was upload by the driver
893  * at boot time
894  */
895 ushort_t         m_fwupload;

897 uint16_t          m_productid;

899 /*
900  * per instance data structures for dma memory resources for
901  * MPI handshake protocol. only one handshake cmd can run at a time.
902  */
903 ddi_dma_handle_t  m_hshk_dma_hdl;
904 ddi_acc_handle_t  m_hshk_acc_hdl;
905 caddr_t           m_hshk_memp;
906 size_t            m_hshk_dma_size;

908 /* Firmware version on the card at boot time */
909 uint32_t          m_fwversion;

911 /* MSI specific fields */
912 ddi_intr_handle_t *m_htable; /* For array of interrupts */
913 int               m_intr_type; /* What type of interrupt */
914 int               m_intr_cnt; /* # of intrs count returned */
915 size_t            m_intr_size; /* Size of intr array */
916 uint_t            m_intr_pri; /* Interrupt priority */
917 int               m_intr_cap; /* Interrupt capabilities */
918 ddi_taskq_t       *m_event_taskq;

```

```

920      /* SAS specific information */
921
922      union {
923          uint64_t      m_base_wwid;    /* Base WWID */
924          struct {
925              #ifdef _BIG_ENDIAN
926                  uint32_t      m_base_wwid_hi;
927                  uint32_t      m_base_wwid_lo;
928              #else
929                  uint32_t      m_base_wwid_lo;
930                  uint32_t      m_base_wwid_hi;
931              #endif
932          } sasaddr;
933      } un;
934
935      uint8_t      m_num_phys;          /* # of PHYs */
936      mptsas_phy_info_t m_phy_info[MPTSAS_MAX_PHYS];
937      uint8_t      m_port_chng;        /* initiator port changes */
938      MPI2_CONFIG_PAGE_MAN_0 m_MANU_page0; /* Manufactor page 0 info */
939      MPI2_CONFIG_PAGE_MAN_1 m_MANU_page1; /* Manufactor page 1 info */
940
941      /* FMA Capabilities */
942      int          m_fm_capabilities;
943      ddi_taskq_t  *m_dr_taskq;
944      int          m_mpxio_enable;
945      uint8_t      m_done_traverse_dev;
946      uint8_t      m_done_traverse_smp;
947      int          m_diag_action_in_progress;
948      uint16_t     m_dev_handle;
949      uint16_t     m_smp_devhdl;
950
951      /*
952       * Event recording
953       */
954      uint8_t      m_event_index;
955      uint32_t      m_event_number;
956      uint32_t      m_event_mask[4];
957      mptsas_event_entry_t m_events[MPTSAS_EVENT_QUEUE_SIZE];
958
959      /*
960       * FW diag Buffer List
961       */
962      mptsas_fw_diagnostic_buffer_t
963      m_fw_diag_buffer_list[MPI2_DIAG_BUF_TYPE_COUNT];
964
965      /* GEN3 support */
966      uint8_t      m_MPI25;
967
968      /*
969       * Event Replay flag (MUR support)
970       */
971      uint8_t      m_event_replay;
972
973      /*
974       * IR Capable flag
975       */
976      uint8_t      m_ir_capable;
977
978      /*
979       * Is HBA processing a diag reset?
980       */
981      uint8_t      m_in_reset;
982
983      /*
984       * per instance cmd data structures for task management cmds
985       */

```

```

986      m_event_struct_t      m_event_task_mgmt;    /* must be last */
987
988      } mptsas_t;
989      #define MPTSAS_SIZE      (sizeof (struct mptsas) - \
990                               sizeof (struct scsi_pkt) + scsi_pkt_size())
991      /*
992       * Only one of below two conditions is satisfied, we
993       * think the target is associated to the iport and
994       * allow call into mptsas_probe_lun().
995       * 1. physicalsport == physport
996       * 2. (phymask & (1 << physport)) == 0
997       * The condition #2 is because LSI uses lowest PHY
998       * number as the value of physical port when auto port
999       * configuration.
1000      */
1001      #define IS_SAME_PORT(physicalport, physport, phymask, dynamicport) \
1002      ((physicalport == physport) || (dynamicport && (phymask & \
1003      (1 << physport))))
1004
1005      _NOTE(MUTEX_PROTECTS_DATA(mptsas::m_mutex, mptsas))
1006      _NOTE(SCHEME_PROTECTS_DATA("safe sharing", mptsas::m_next))
1007      _NOTE(SCHEME_PROTECTS_DATA("stable data", mptsas::m_dip mptsas::m_tran))
1008      _NOTE(SCHEME_PROTECTS_DATA("stable data", mptsas::m_kmem_cache))
1009      _NOTE(DATA_READABLE_WITHOUT_LOCK(mptsas::m_io_dma_attr.dma_attr_sgllen))
1010      _NOTE(DATA_READABLE_WITHOUT_LOCK(mptsas::m_devid))
1011      _NOTE(DATA_READABLE_WITHOUT_LOCK(mptsas::m_productid))
1012      _NOTE(DATA_READABLE_WITHOUT_LOCK(mptsas::m_mpxio_enable))
1013      _NOTE(DATA_READABLE_WITHOUT_LOCK(mptsas::m_instance))
1014
1015      /*
1016       * These should eventually migrate into the mpt header files
1017       * that may become the /kernel/misc/mpt module...
1018       */
1019      #define mptsas_init_std_hdr(hdl, mp, DevHandle, Lun, ChainOffset, Function) \
1020      mptsas_put_msg_DevHandle(hdl, mp, DevHandle); \
1021      mptsas_put_msg_ChainOffset(hdl, mp, ChainOffset); \
1022      mptsas_put_msg_Function(hdl, mp, Function); \
1023      mptsas_put_msg_Lun(hdl, mp, Lun)
1024
1025      #define mptsas_put_msg_DevHandle(hdl, mp, val) \
1026      ddi_put16(hdl, &(mp)->DevHandle, (val))
1027      #define mptsas_put_msg_ChainOffset(hdl, mp, val) \
1028      ddi_put8(hdl, &(mp)->ChainOffset, (val))
1029      #define mptsas_put_msg_Function(hdl, mp, val) \
1030      ddi_put8(hdl, &(mp)->Function, (val))
1031      #define mptsas_put_msg_Lun(hdl, mp, val) \
1032      ddi_put8(hdl, &(mp)->LUN[1], (val))
1033
1034      #define mptsas_get_msg_Function(hdl, mp) \
1035      ddi_get8(hdl, &(mp)->Function)
1036
1037      #define mptsas_get_msg_MsgFlags(hdl, mp) \
1038      ddi_get8(hdl, &(mp)->MsgFlags)
1039
1040      #define MPTSAS_ENABLE_DRWE(hdl) \
1041      ddi_put32(hdl->m_datap, &hdl->m_reg->WriteSequence, \
1042      MPI2_WRSEQ_FLUSH_KEY_VALUE); \
1043      ddi_put32(hdl->m_datap, &hdl->m_reg->WriteSequence, \
1044      MPI2_WRSEQ_1ST_KEY_VALUE); \
1045      ddi_put32(hdl->m_datap, &hdl->m_reg->WriteSequence, \
1046      MPI2_WRSEQ_2ND_KEY_VALUE); \
1047      ddi_put32(hdl->m_datap, &hdl->m_reg->WriteSequence, \
1048      MPI2_WRSEQ_3RD_KEY_VALUE); \
1049      ddi_put32(hdl->m_datap, &hdl->m_reg->WriteSequence, \
1050      MPI2_WRSEQ_4TH_KEY_VALUE); \
1051      ddi_put32(hdl->m_datap, &hdl->m_reg->WriteSequence, \

```

```

1052     MPI2_WRSEQ_5TH_KEY_VALUE); \
1053     ddi_put32(hdl->m_datap, &hdl->m_reg->WriteSequence, \
1054     MPI2_WRSEQ_6TH_KEY_VALUE);

1056 /*
1057  * m_options flags
1058  */
1059 #define MPTSAS_OPT_PM          0x01    /* Power Management */
1060 #define MPTSAS_OPT_MSI         0x02    /* PCI MSI Interrupts */
1061 #define MPTSAS_OPT_MSI_X       0x04    /* PCI MSI-X Interrupts */

1063 /*
1064  * m_softstate flags
1065  */
1066 #define MPTSAS_SS_DRAINING      0x02
1067 #define MPTSAS_SS_QUIESCED      0x04
1068 #define MPTSAS_SS_MSG_UNIT_RESET 0x08
1069 #define MPTSAS_DID_MSG_UNIT_RESET 0x10

1071 /*
1072  * m_dma_flags (allocated).
1073  */
1074 #define MPTSAS_REQ_FRAME        0x01
1075 #define MPTSAS_REQ_SENSE        0x02
1076 #define MPTSAS_REPLY_FRAME      0x04
1077 #define MPTSAS_FREE_QUEUE       0x08
1078 #define MPTSAS_POST_QUEUE       0x10

1080 /*
1081  * regspec defines.
1082  */
1083 #define CONFIG_SPACE            0        /* regset[0] - configuration space */
1084 #define IO_SPACE                 1        /* regset[1] - used for i/o mapped device */
1085 #define MEM_SPACE                2        /* regset[2] - used for memory mapped device */
1086 #define BASE_REG2                3        /* regset[3] - used for 875 scripts ram */

1088 /*
1089  * Handy constants
1090  */
1091 #define FALSE                    0
1092 #define TRUE                     1
1093 #define BLOCKED                  2
1094 #define UNDEFINED                -1
1095 #define FAILED                   -2

1097 /*
1098  * power management.
1099  */
1100 #define MPTSAS_POWER_ON(mpt) { \
1101     pci_config_put16(mpt->m_config_handle, mpt->m_pmcsr_offset, \
1102     PCI_PMCSR_D0); \
1103     delay(drv_usec_to_hz(10000)); \
1104     (void) pci_restore_config_regs(mpt->m_dip); \
1105     mptsas_setup_cmd_reg(mpt); \
1106 }

1108 #define MPTSAS_POWER_OFF(mpt) { \
1109     (void) pci_save_config_regs(mpt->m_dip); \
1110     pci_config_put16(mpt->m_config_handle, mpt->m_pmcsr_offset, \
1111     PCI_PMCSR_D3HOT); \
1112     mpt->m_power_level = PM_LEVEL_D3; \
1113 }

1115 /*
1116  * inq_dtype:
1117  * Bits 5 through 7 are the Peripheral Device Qualifier

```

```

1118  * 001b: device not connected to the LUN
1119  * Bits 0 through 4 are the Peripheral Device Type
1120  * 1fh: Unknown or no device type
1121  *
1122  * Although the inquiry may return success, the following value
1123  * means no valid LUN connected.
1124  */
1125 #define MPTSAS_VALID_LUN(sd_inq) \
1126     (((sd_inq->inq_dtype & 0xe0) != 0x20) && \
1127     ((sd_inq->inq_dtype & 0x1f) != 0x1f))

1129 /*
1130  * Default is to have 10 retries on receiving QFULL status and
1131  * each retry to be after 100 ms.
1132  */
1133 #define QFULL_RETRIES          10
1134 #define QFULL_RETRY_INTERVAL    100

1136 /*
1137  * Handy macros
1138  */
1139 #define Tgt(sp) ((sp)->cmd_pkt->pkt_address.a_target)
1140 #define Lun(sp) ((sp)->cmd_pkt->pkt_address.a_lun)

1142 #define IS_HEX_DIGIT(n) (((n) >= '0' && (n) <= '9') || \
1143     ((n) >= 'a' && (n) <= 'f') || ((n) >= 'A' && (n) <= 'F'))

1145 /*
1146  * poll time for mptsas_pollret() and mptsas_wait_intr()
1147  */
1148 #define MPTSAS_POLL_TIME        30000    /* 30 seconds */

1150 /*
1151  * default time for mptsas_do_passthru
1152  */
1153 #define MPTSAS_PASS_THRU_TIME_DEFAULT 60    /* 60 seconds */

1155 /*
1156  * macro to return the effective address of a given per-target field
1157  */
1158 #define EFF_ADDR(start, offset)    ((start) + (offset))

1160 #define SDEV2ADDR(devp)             (&((devp)->sd_address))
1161 #define SDEV2TRAN(devp)             ((devp)->sd_address.a_hba_tran)
1162 #define PKT2TRAN(pkt)              ((pkt)->pkt_address.a_hba_tran)
1163 #define ADDR2TRAN(ap)               ((ap)->a_hba_tran)
1164 #define DIP2TRAN(dip)               (ddi_get_driver_private(dip))

1167 #define TRAN2MPT(hba)               ((mptsas_t *) (hba)->tran_hba_private)
1168 #define DIP2MPT(dip)                (TRAN2MPT((scsi_hba_tran_t *) DIP2TRAN(dip)))
1169 #define SDEV2MPT(sd)                (TRAN2MPT(SDEV2TRAN(sd)))
1170 #define PKT2MPT(pkt)                (TRAN2MPT(PKT2TRAN(pkt)))

1172 #define ADDR2MPT(ap)                (TRAN2MPT(ADDR2TRAN(ap)))

1174 #define POLL_TIMEOUT                (2 * SCSI_POLL_TIMEOUT * 1000000)
1175 #define SHORT_POLL_TIMEOUT          (1000000)    /* in usec, about 1 secs */
1176 #define MPTSAS_QUIESCE_TIMEOUT      1    /* 1 sec */
1177 #define MPTSAS_PM_IDLE_TIMEOUT      60    /* 60 seconds */

1179 #define MPTSAS_GET_ISTAT(mpt)        (ddi_get32((mpt)->m_datap, \
1180     &(mpt)->m_reg->HostInterruptStatus))

1182 #define MPTSAS_SET_SIGP(P) \
1183     ClrSetBits(mpt->m_devaddr + NREG_ISTAT, 0, NB_ISTAT_SIGP)

```

```

1185 #define MPTSAS_RESET_SIGP(P) (void) ddi_get8(mpt->m_datap, \
1186         (uint8_t *) (mpt->m_devaddr + NREG_CTEST2))

1188 #define MPTSAS_GET_INTCODE(P) (ddi_get32(mpt->m_datap, \
1189         (uint32_t *) (mpt->m_devaddr + NREG_DSPS)))

1192 #define MPTSAS_START_CMD(mpt, req_desc) \
1193     ddi_put64(mpt->m_datap, \
1194         (uint64_t *) (void *) &mpt->m_reg->RequestDescriptorPostLow, \
1195         req_desc); \
1196     atomic_inc_32(&mpt->m_ncstarted)

1199 #define INTPENDING(mpt) \
1200     (MPTSAS_GET_ISTAT(mpt) & MPI2_HIS_REPLY_DESCRIPTOR_INTERRUPT)

1202 /*
1203  * Mask all interrupts to disable
1204  */
1205 #define MPTSAS_DISABLE_INTR(mpt) \
1206     ddi_put32((mpt->m_datap, &(mpt->m_reg->HostInterruptMask, \
1207         (MPI2_HIM_RIM | MPI2_HIM_DIM | MPI2_HIM_RESET_IRQ_MASK))

1209 /*
1210  * Mask Doorbell and Reset interrupts to enable reply desc int.
1211  */
1212 #define MPTSAS_ENABLE_INTR(mpt) \
1213     ddi_put32(mpt->m_datap, &mpt->m_reg->HostInterruptMask, \
1214         (MPI2_HIM_DIM | MPI2_HIM_RESET_IRQ_MASK))

1216 #define MPTSAS_GET_NEXT_REPLY(rpqp, index) \
1217     &((uint64_t *) (void *) rpqp->rpq_queue)[index]

1219 #define MPTSAS_GET_NEXT_FRAME(mpt, SMID) \
1220     (mpt->m_req_frame + (mpt->m_req_frame_size * SMID))

1222 #define ClrSetBits32(hdl, reg, clr, set) \
1223     ddi_put32(hdl, (reg), \
1224         ((ddi_get32(mpt->m_datap, (reg)) & ~(clr)) | (set)))

1226 #define ClrSetBits(reg, clr, set) \
1227     ddi_put8(mpt->m_datap, (uint8_t *) (reg), \
1228         ((ddi_get8(mpt->m_datap, (uint8_t *) (reg)) & ~(clr)) | (set)))

1230 #define MPTSAS_WAITQ_RM(mpt, cmdp) \
1231     if ((cmdp = mpt->m_waitq) != NULL) { \
1232         /* If the queue is now empty fix the tail pointer */ \
1233         if ((mpt->m_waitq = cmdp->cmd_linkp) == NULL) \
1234             mpt->m_waitqtail = &mpt->m_waitq; \
1235         cmdp->cmd_linkp = NULL; \
1236         cmdp->cmd_queued = FALSE; \
1237     }

1239 #define MPTSAS_TX_WAITQ_RM(mpt, cmdp) \
1240     if ((cmdp = mpt->m_tx_waitq) != NULL) { \
1241         /* If the queue is now empty fix the tail pointer */ \
1242         if ((mpt->m_tx_waitq = cmdp->cmd_linkp) == NULL) \
1243             mpt->m_tx_waitqtail = &mpt->m_tx_waitq; \
1244         cmdp->cmd_linkp = NULL; \
1245         cmdp->cmd_queued = FALSE; \
1246     }

1248 /*
1249  * defaults for the global properties

```

```

1250 */
1251 #define DEFAULT_SCSI_OPTIONS        SCSI_OPTIONS_DR
1252 #define DEFAULT_TAG_AGE_LIMIT      2
1253 #define DEFAULT_WD_TICK             1

1255 /*
1256  * invalid hostid.
1257  */
1258 #define MPTSAS_INVALID_HOSTID      -1

1260 /*
1261  * Get/Set hostid from SCSI port configuration page
1262  */
1263 #define MPTSAS_GET_HOST_ID(configuration) (configuration & 0xFF)
1264 #define MPTSAS_SET_HOST_ID(hostid) (hostid | ((1 << hostid) << 16))

1266 /*
1267  * Config space.
1268  */
1269 #define MPTSAS_LATENCY_TIMER        0x40

1271 /*
1272  * Offset to firmware version
1273  */
1274 #define MPTSAS_FW_VERSION_OFFSET    9

1276 /*
1277  * Offset and masks to get at the ProductId field
1278  */
1279 #define MPTSAS_FW_PRODUCTID_OFFSET  8
1280 #define MPTSAS_FW_PRODUCTID_MASK    0xFFFF0000
1281 #define MPTSAS_FW_PRODUCTID_SHIFT   16

1283 /*
1284  * Subsystem ID for HBAs.
1285  */
1286 #define MPTSAS_HBA_SUBSYSTEM_ID      0x10C0
1287 #define MPTSAS_RHEA_SUBSYSTEM_ID     0x10B0

1289 /*
1290  * reset delay tick
1291  */
1292 #define MPTSAS_WATCH_RESET_DELAY_TICK 50      /* specified in milli seconds */

1294 /*
1295  * Ioc reset return values
1296  */
1297 #define MPTSAS_RESET_FAIL            -1
1298 #define MPTSAS_NO_RESET              0
1299 #define MPTSAS_SUCCESS_HARDRESET     1
1300 #define MPTSAS_SUCCESS_MUR           2

1302 /*
1303  * throttle support.
1304  */
1305 #define MAX_THROTTLE                 32
1306 #define HOLD_THROTTLE                0
1307 #define DRAIN_THROTTLE               -1
1308 #define QFULL_THROTTLE               -2

1310 /*
1311  * Passthrough/config request flags
1312  */
1313 #define MPTSAS_DATA_ALLOCATED         0x0001
1314 #define MPTSAS_DATAOUT_ALLOCATED     0x0002
1315 #define MPTSAS_REQUEST_POOL_CMD      0x0004

```

```

1316 #define MPTSAS_ADDRESS_REPLY          0x0008
1317 #define MPTSAS_CMD_TIMEOUT             0x0010

1319 /*
1320  * response code tlr flag
1321  */
1322 #define MPTSAS_SCSI_RESPONSE_CODE_TLR_OFF 0x02

1324 /*
1325  * System Events
1326  */
1327 #ifndef DDI_VENDOR_LSI
1328 #define DDI_VENDOR_LSI "LSI"
1329 #endif /* DDI_VENDOR_LSI */

1331 /*
1332  * Shared functions
1333  */
1334 int mptsas_save_cmd(struct mptsas *mpt, struct mptsas_cmd *cmd);
1335 void mptsas_remove_cmd(mptsas_t *mpt, mptsas_cmd_t *cmd);
1336 void mptsas_waitq_add(mptsas_t *mpt, mptsas_cmd_t *cmd);
1337 void mptsas_log(struct mptsas *mpt, int level, char *fmt, ...);
1338 int mptsas_poll(mptsas_t *mpt, mptsas_cmd_t *poll_cmd, int polltime);
1339 int mptsas_do_dma(mptsas_t *mpt, uint32_t size, int var, int (*callback)());
1340 int mptsas_update_flash(mptsas_t *mpt, caddr_t ptrbuffer, uint32_t size,
1341     uint8_t type, int mode);
1342 int mptsas_check_flash(mptsas_t *mpt, caddr_t origfile, uint32_t size,
1343     uint8_t type, int mode);
1344 int mptsas_download_firmware();
1345 int mptsas_can_download_firmware();
1346 int mptsas_dma_alloc(mptsas_t *mpt, mptsas_dma_alloc_state_t *dma_statep);
1347 void mptsas_dma_free(mptsas_dma_alloc_state_t *dma_statep);
1348 mptsas_phymask_t mptsas_physport_to_phymask(mptsas_t *mpt, uint8_t physport);
1349 void mptsas_fma_check(mptsas_t *mpt, mptsas_cmd_t *cmd);
1350 int mptsas_check_acc_handle(ddi_acc_handle_t handle);
1351 int mptsas_check_dma_handle(ddi_dma_handle_t handle);
1352 void mptsas_fm_ereport(mptsas_t *mpt, char *detail);
1353 int mptsas_dma_addr_create(mptsas_t *mpt, ddi_dma_attr_t dma_attr,
1354     ddi_dma_handle_t *dma_hdl, ddi_acc_handle_t *acc_hdl, caddr_t *dma_memp,
1355     uint32_t alloc_size, ddi_dma_cookie_t *cookiep);
1356 void mptsas_dma_addr_destroy(ddi_dma_handle_t *, ddi_acc_handle_t *);

1358 /*
1359  * impl functions
1360  */
1361 int mptsas_ioc_wait_for_response(mptsas_t *mpt);
1362 int mptsas_ioc_wait_for_doorbell(mptsas_t *mpt);
1363 int mptsas_ioc_reset(mptsas_t *mpt, int);
1364 int mptsas_send_handshake_msg(mptsas_t *mpt, caddr_t memp, int numbytes,
1365     ddi_acc_handle_t accessp);
1366 int mptsas_get_handshake_msg(mptsas_t *mpt, caddr_t memp, int numbytes,
1367     ddi_acc_handle_t accessp);
1368 int mptsas_send_config_request_msg(mptsas_t *mpt, uint8_t action,
1369     uint8_t pagetype, uint32_t pageaddress, uint8_t pagenumber,
1370     uint8_t pageversion, uint8_t pagelength, uint32_t SGEflagslength,
1371     uint64_t SGEaddress);
1372 int mptsas_send_extended_config_request_msg(mptsas_t *mpt, uint8_t action,
1373     uint8_t extpagetype, uint32_t pageaddress, uint8_t pagenumber,
1374     uint8_t pageversion, uint16_t extpagelength,
1375     uint32_t SGEflagslength, uint64_t SGEaddress);

1377 int mptsas_request_from_pool(mptsas_t *mpt, mptsas_cmd_t **cmd,
1378     struct scsi_pkt **pkt);
1379 void mptsas_return_to_pool(mptsas_t *mpt, mptsas_cmd_t *cmd);
1380 void mptsas_destroy_ioc_event_cmd(mptsas_t *mpt);
1381 void mptsas_start_config_page_access(mptsas_t *mpt, mptsas_cmd_t *cmd);

```

```

1382 int mptsas_access_config_page(mptsas_t *mpt, uint8_t action, uint8_t page_type,
1383     uint8_t page_number, uint32_t page_address, int (*callback) (mptsas_t *,
1384         caddr_t, ddi_acc_handle_t, uint16_t, uint32_t, va_list), ...);

1386 int mptsas_ioc_task_management(mptsas_t *mpt, int task_type,
1387     uint16_t dev_handle, int lun, uint8_t *reply, uint32_t reply_size,
1388     int mode);
1389 int mptsas_send_event_ack(mptsas_t *mpt, uint32_t event, uint32_t eventcntx);
1390 void mptsas_send_pending_event_ack(mptsas_t *mpt);
1391 int mptsas_restart_ioc(mptsas_t *mpt);
1392 void mptsas_update_driver_data(struct mptsas *mpt);
1393 uint64_t mptsas_get_sata_guid(mptsas_t *mpt, mptsas_target_t *tgt, int lun);

1395 /*
1396  * init functions
1397  */
1398 int mptsas_ioc_get_facts(mptsas_t *mpt);
1399 int mptsas_ioc_get_port_facts(mptsas_t *mpt, int port);
1400 void mptsas_ioc_enable_port(mptsas_t *mpt);
1401 int mptsas_ioc_enable_event_notification(mptsas_t *mpt);
1402 int mptsas_ioc_init(mptsas_t *mpt);

1404 /*
1405  * configuration pages operation
1406  */
1407 int mptsas_get_sas_device_page0(mptsas_t *mpt, uint32_t page_address,
1408     uint16_t *dev_handle, uint64_t *sas_wnn, uint32_t *dev_info,
1409     uint8_t *physport, uint8_t *phynum, uint16_t *pdevhandle,
1410     uint16_t *slot_num, uint16_t *enclosure, uint16_t *io_flags);
1411 int mptsas_get_sas_io_unit_page(mptsas_t *mpt);
1412 int mptsas_get_sas_io_unit_page_hndshk(mptsas_t *mpt);
1413 int mptsas_get_sas_expander_page0(mptsas_t *mpt, uint32_t page_address,
1414     mptsas_smp_t *info);
1415 int mptsas_set_ioc_params(mptsas_t *mpt);
1416 int mptsas_get_manufacture_page5(mptsas_t *mpt);
1417 int mptsas_get_sas_port_page0(mptsas_t *mpt, uint32_t page_address,
1418     uint64_t *sas_wnn, uint8_t *portwidth);
1419 int mptsas_get_bios_page3(mptsas_t *mpt, uint32_t *bios_version);
1420 int
1421 mptsas_get_sas_phy_page0(mptsas_t *mpt, uint32_t page_address,
1422     smhba_info_t *info);
1423 int
1424 mptsas_get_sas_phy_page1(mptsas_t *mpt, uint32_t page_address,
1425     smhba_info_t *info);
1426 int
1427 mptsas_get_manufacture_page0(mptsas_t *mpt);
1428 void
1429 mptsas_create_phy_stats(mptsas_t *mpt, char *iport, dev_info_t *dip);
1430 void mptsas_destroy_phy_stats(mptsas_t *mpt);
1431 int mptsas_smhba_phy_init(mptsas_t *mpt);
1432 /*
1433  * RAID functions
1434  */
1435 int mptsas_get_raid_settings(mptsas_t *mpt, mptsas_raidvol_t *raidvol);
1436 int mptsas_get_raid_info(mptsas_t *mpt);
1437 int mptsas_get_physdisk_settings(mptsas_t *mpt, mptsas_raidvol_t *raidvol,
1438     uint8_t physdisknum);
1439 int mptsas_delete_volume(mptsas_t *mpt, uint16_t volid);
1440 void mptsas_raid_action_system_shutdown(mptsas_t *mpt);

1442 #define MPTSAS_IOCSTATUS(status) (status & MPI2_IOCSTATUS_MASK)
1443 /*
1444  * debugging.
1445  */
1446 #if defined(MPTSAS_DEBUG)

```

```

1448 void mptsas_printf(char *fmt, ...);
1449 void mptsas_debug_log(char *fmt, ...);

1451 #define MPTSAS_DBGPR(m, args) \
1452     if (mptsas_debug_flags & (m)) \
1453         mptsas_printf args; \
1454     if (~mptsas_dbglog_imask & (m)) \
1455         mptsas_debug_log args
1456 #else /* ! defined(MPTSAS_DEBUG) */
1457 #define MPTSAS_DBGPR(m, args)
1458 #endif /* defined(MPTSAS_DEBUG) */

1460 #define NDBG0(args)      MPTSAS_DBGPR(0x01, args)    /* init */
1461 #define NDBG1(args)      MPTSAS_DBGPR(0x02, args)    /* normal running */
1462 #define NDBG2(args)      MPTSAS_DBGPR(0x04, args)    /* property handling */
1463 #define NDBG3(args)      MPTSAS_DBGPR(0x08, args)    /* pkt handling */

1465 #define NDBG4(args)      MPTSAS_DBGPR(0x10, args)    /* kmem alloc/free */
1466 #define NDBG5(args)      MPTSAS_DBGPR(0x20, args)    /* polled cmds */
1467 #define NDBG6(args)      MPTSAS_DBGPR(0x40, args)    /* interrupt setup */
1468 #define NDBG7(args)      MPTSAS_DBGPR(0x80, args)    /* queue handling */

1470 #define NDBG8(args)      MPTSAS_DBGPR(0x0100, args)  /* arq */
1471 #define NDBG9(args)      MPTSAS_DBGPR(0x0200, args)  /* Tagged Q'ing */
1472 #define NDBG10(args)     MPTSAS_DBGPR(0x0400, args)  /* halting chip */
1473 #define NDBG11(args)     MPTSAS_DBGPR(0x0800, args)  /* power management */

1475 #define NDBG12(args)     MPTSAS_DBGPR(0x1000, args)  /* enumeration */
1476 #define NDBG13(args)     MPTSAS_DBGPR(0x2000, args)  /* configuration page */
1477 #define NDBG14(args)     MPTSAS_DBGPR(0x4000, args)  /* LED control */
1478 #define NDBG15(args)     MPTSAS_DBGPR(0x8000, args)  /* Passthrough */

1480 #define NDBG16(args)     MPTSAS_DBGPR(0x010000, args) /* SAS Broadcasts */
1481 #define NDBG17(args)     MPTSAS_DBGPR(0x020000, args) /* scatter/gather */
1482 #define NDBG18(args)     MPTSAS_DBGPR(0x040000, args) /* Interrupts */
1483 #define NDBG19(args)     MPTSAS_DBGPR(0x080000, args) /* handshaking */

1485 #define NDBG20(args)     MPTSAS_DBGPR(0x100000, args) /* events */
1486 #define NDBG21(args)     MPTSAS_DBGPR(0x200000, args) /* dma */
1487 #define NDBG22(args)     MPTSAS_DBGPR(0x400000, args) /* reset */
1488 #define NDBG23(args)     MPTSAS_DBGPR(0x800000, args) /* abort */

1490 #define NDBG24(args)     MPTSAS_DBGPR(0x1000000, args) /* capabilities */
1491 #define NDBG25(args)     MPTSAS_DBGPR(0x2000000, args) /* flushing */
1492 #define NDBG26(args)     MPTSAS_DBGPR(0x4000000, args)
1493 #define NDBG27(args)     MPTSAS_DBGPR(0x8000000, args)

1495 #define NDBG28(args)     MPTSAS_DBGPR(0x10000000, args) /* hotplug */
1496 #define NDBG29(args)     MPTSAS_DBGPR(0x20000000, args) /* timeouts */
1497 #define NDBG30(args)     MPTSAS_DBGPR(0x40000000, args) /* mptsas_watch */
1498 #define NDBG31(args)     MPTSAS_DBGPR(0x80000000, args) /* negotiations */

1500 /*
1501  * auto request sense
1502  */
1503 #define RQ_MAKECOM_COMMON(pkt, flag, cmd) \
1504     (pkt)->pkt_flags = (flag), \
1505     ((union scsi_cdb *) (pkt)->pkt_cdbp)->scc_cmd = (cmd), \
1506     ((union scsi_cdb *) (pkt)->pkt_cdbp)->scc_lun = \
1507         (pkt)->pkt_address.a_lun

1509 #define RQ_MAKECOM_G0(pkt, flag, cmd, addr, cnt) \
1510     RQ_MAKECOM_COMMON((pkt), (flag), (cmd)), \
1511     FORMG0ADDR(((union scsi_cdb *) (pkt)->pkt_cdbp), (addr)), \
1512     FORMG0COUNT(((union scsi_cdb *) (pkt)->pkt_cdbp), (cnt))

```

```

1515 #ifdef __cplusplus
1516 }
1517 #endif

1519 #endif /* !_SYS SCSI_ADAPTERS_MPTSAS3_VAR_H */
1520 #endif /* ! codereview */

```

new/usr/src/uts/intel/Makefile.intel

1

```
*****
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
*****

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

21 # Copyright (c) 2005, 2010, Oracle and/or its affiliates. All rights reserved.
22 # Copyright (c) 2012 Nexenta Systems, Inc. All rights reserved.
23 # Copyright (c) 2013 Andrew Stormont. All rights reserved.

25 #
26 # This makefile contains the common definitions for all intel
27 # implementation architecture independent modules.
28 #

30 #
31 # Machine type (implementation architecture):
32 #
33 PLATFORM = i86pc

35 #
36 # Everybody needs to know how to build modstubs.o and to locate unix.o.
37 # Note that unix.o must currently be selected from among the possible
38 # "implementation architectures". Note further, that unix.o is only
39 # used as an optional error check for undefines so (theoretically)
40 # any "implementation architectures" could be used. We choose i86pc
41 # because it is the reference port.
42 #
43 UNIX_DIR = $(UTSBASE)/i86pc/unix
44 GENLIB_DIR = $(UTSBASE)/intel/genunix
45 IPDRV_DIR = $(UTSBASE)/intel/ip
46 MODSTUBS_DIR = $(UNIX_DIR)
47 DSF_DIR = $(UTSBASE)/$(PLATFORM)/genassym
48 LINTS_DIR = $(OBJS_DIR)
49 LINT_LIB_DIR = $(UTSBASE)/intel/lint-libs/$(OBJS_DIR)

51 UNIX_O = $(UNIX_DIR)/$(OBJS_DIR)/unix.o
52 GENLIB = $(GENLIB_DIR)/$(OBJS_DIR)/libgenunix.so
53 MODSTUBS_O = $(MODSTUBS_DIR)/$(OBJS_DIR)/modstubs.o
54 LINT_LIB = $(UTSBASE)/i86pc/lint-libs/$(OBJS_DIR)/llib-lunix.ln
55 GEN_LINT_LIB = $(UTSBASE)/intel/lint-libs/$(OBJS_DIR)/llib-lgenunix.ln

57 #
58 # Include the makefiles which define build rule templates, the
59 # collection of files per module, and a few specific flags. Note
60 # that order is significant, just as with an include path. The
61 # first build rule template which matches the files name will be
```

new/usr/src/uts/intel/Makefile.intel

2

```
62 # used. By including these in order from most machine dependent
63 # to most machine independent, we allow a machine dependent file
64 # to be used in preference over a machine independent version
65 # (Such as a machine specific optimization, which preserves the
66 # interfaces.)
67 #
68 include $(UTSBASE)/intel/Makefile.files
69 include $(UTSBASE)/common/Makefile.files

71 #
72 # ----- TRANSITIONAL SECTION -----
73 #

75 #
76 # Not everything which *should* be a module is a module yet. The
77 # following is a list of such objects which are currently part of
78 # genunix but which might someday become kmods. This must be
79 # defined before we include Makefile.uts, or else genunix's build
80 # won't be as parallel as we might like.
81 #
82 NOT_YET_KMODS = $(OLDPTY_OBJS) $(PTY_OBJS) $(VCONS_CONF_OBJS) $(MOD_OBJS)

84 #
85 # ----- END OF TRANSITIONAL SECTION -----
86 #
87 # Include machine independent rules. Note that this does not imply
88 # that the resulting module from rules in Makefile.uts is machine
89 # independent. Only that the build rules are machine independent.
90 #
91 include $(UTSBASE)/Makefile.uts

93 #
94 # The following must be defined for all implementations:
95 #
96 MODSTUBS = $(UTSBASE)/intel/ia32/ml/modstubs.s

98 #
99 # Define supported builds
100 #
101 DEF_BUILDS = $(DEF_BUILDS64) $(DEF_BUILDS32)
102 ALL_BUILDS = $(ALL_BUILDS64) $(ALL_BUILDS32)

104 #
105 # x86 or amd64 inline templates
106 #
107 INLINES_32 = $(UTSBASE)/intel/ia32/ml/ia32.il
108 INLINES_64 = $(UTSBASE)/intel/amd64/ml/amd64.il
109 INLINES += $(INLINES_$(CLASS))

111 #
112 # kernel-specific optimizations; override default in Makefile.master
113 #

115 CFLAGS_XARCH_32 = $(i386_CFLAGS)
116 CFLAGS_XARCH_64 = $(amd64_CFLAGS)
117 CFLAGS_XARCH = $(CFLAGS_XARCH_$(CLASS))

119 COPTFLAG_32 = $(COPTFLAG)
120 COPTFLAG_64 = $(COPTFLAG64)
121 COPTIMIZE = $(COPTFLAG_$(CLASS))

123 CFLAGS = $(CFLAGS_XARCH)
124 CFLAGS += $(COPTIMIZE)
125 CFLAGS += $(INLINES) -D_ASM_INLINES
126 CFLAGS += $(CCMODE)
127 CFLAGS += $(SPACEFLAG)
```

```

128 CFLAGS          += $(CCUNBOUND)
129 CFLAGS          += $(CFLAGS_uts)
130 CFLAGS          += -xstrconst

132 ASFLAGS_XARCH_32 = $(i386_ASFLAGS)
133 ASFLAGS_XARCH_64 = $(amd64_ASFLAGS)
134 ASFLAGS_XARCH     = $(ASFLAGS_XARCH_$(CLASS))

136 ASFLAGS          += $(ASFLAGS_XARCH)

138 #
139 #       Define the base directory for installation.
140 #
141 BASE_INS_DIR      = $(ROOT)

143 #
144 #       Debugging level
145 #
146 #       Special knowledge of which special debugging options affect which
147 #       file is used to optimize the build if these flags are changed.
148 #
149 DEBUG_DEFS_OBJ32  =
150 DEBUG_DEFS_DBG32  = -DDEBUG
151 DEBUG_DEFS_OBJ64  =
152 DEBUG_DEFS_DBG64  = -DDEBUG
153 DEBUG_DEFS        = $(DEBUG_DEFS_$(BUILD_TYPE))

155 DEBUG_COND_OBJ32  = $(POUND_SIGN)
156 DEBUG_COND_DBG32  =
157 DEBUG_COND_OBJ64  = $(POUND_SIGN)
158 DEBUG_COND_DBG64  =
159 IF_DEBUG_OBJ       = $(DEBUG_COND_$(BUILD_TYPE))$(OBS_DIR)/

161 $(IF_DEBUG_OBJ)syscall.o      :=      DEBUG_DEFS      += -DSYSCALLTRACE
162 $(IF_DEBUG_OBJ)clock.o        :=      DEBUG_DEFS      += -DKSLICE=1

164 #
165 #       Collect the preprocessor definitions to be associated with *all*
166 #       files.
167 #
168 ALL_DEFS           = $(DEBUG_DEFS) $(OPTION_DEFS)

170 #
171 #       The kernels modules which are "implementation architecture"
172 #       specific for this machine are enumerated below. Note that most
173 #       of these modules must exist (in one form or another) for each
174 #       architecture.
175 #
176 #       Common Drivers (usually pseudo drivers) (/kernel/drv)
177 #       DRV_KMODS are built both 32-bit and 64-bit
178 #       DRV_KMODS_32 are built only 32-bit
179 #       DRV_KMODS_64 are built only 64-bit
180 #
181 DRV_KMODS          += aac
182 DRV_KMODS          += aggr
183 DRV_KMODS          += ahci
184 DRV_KMODS          += amd64_gart
185 DRV_KMODS          += amr
186 DRV_KMODS          += agpgart
187 DRV_KMODS          += srn
188 DRV_KMODS          += agptarget
189 DRV_KMODS          += arn
190 DRV_KMODS          += arp
191 DRV_KMODS          += asy
192 DRV_KMODS          += ata
193 DRV_KMODS          += ath

```

```

194 DRV_KMODS          += atu
195 DRV_KMODS          += audio
196 DRV_KMODS          += audio1575
197 DRV_KMODS          += audio810
198 DRV_KMODS          += audiocmi
199 DRV_KMODS          += audiocmihd
200 DRV_KMODS          += audioemul0k
201 DRV_KMODS          += audioens
202 DRV_KMODS          += audiohd
203 DRV_KMODS          += audioixp
204 DRV_KMODS          += audiols
205 DRV_KMODS          += audiopl6x
206 DRV_KMODS          += audiopci
207 DRV_KMODS          += audiosolo
208 DRV_KMODS          += audiotst
209 DRV_KMODS          += audiovia823x
210 DRV_KMODS_32       += audiovia97
211 DRV_KMODS          += bl
212 DRV_KMODS          += blkdev
213 DRV_KMODS          += bge
214 DRV_KMODS          += bofi
215 DRV_KMODS          += bpf
216 DRV_KMODS          += bridge
217 DRV_KMODS          += bscbus
218 DRV_KMODS          += bscv
219 DRV_KMODS          += chxge
220 DRV_KMODS          += cxgbe
221 DRV_KMODS          += ntn
222 DRV_KMODS          += myril0ge
223 DRV_KMODS          += clone
224 DRV_KMODS          += cmdk
225 DRV_KMODS          += cn
226 DRV_KMODS          += conskbd
227 DRV_KMODS          += consms
228 DRV_KMODS          += cpgary3
229 DRV_KMODS          += cpuid
230 DRV_KMODS          += cpunex
231 DRV_KMODS          += crypto
232 DRV_KMODS          += cryptoadm
233 DRV_KMODS          += dca
234 DRV_KMODS          += devinfo
235 DRV_KMODS          += dld
236 DRV_KMODS          += dlpistub
237 DRV_KMODS_32       += dnet
238 DRV_KMODS          += dump
239 DRV_KMODS          += ecpp
240 DRV_KMODS          += emlxs
241 DRV_KMODS          += fd
242 DRV_KMODS          += fdc
243 DRV_KMODS          += fm
244 DRV_KMODS          += fssnap
245 DRV_KMODS          += hxge
246 DRV_KMODS          += i8042
247 DRV_KMODS          += i915
248 DRV_KMODS          += icmp
249 DRV_KMODS          += icmp6
250 DRV_KMODS          += intel_nb5000
251 DRV_KMODS          += intel_nhm
252 DRV_KMODS          += ip
253 DRV_KMODS          += ip6
254 DRV_KMODS          += ipd
255 DRV_KMODS          += ipf
256 DRV_KMODS          += ipnet
257 DRV_KMODS          += ippctl
258 DRV_KMODS          += ipsecah
259 DRV_KMODS          += ipsecesp

```



```

260 DRV_KMODS += ipw
261 DRV_KMODS += iwh
262 DRV_KMODS += iwi
263 DRV_KMODS += iwk
264 DRV_KMODS += iwp
265 DRV_KMODS += iwscn
266 DRV_KMODS += kb8042
267 DRV_KMODS += keysock
268 DRV_KMODS += kssl
269 DRV_KMODS += kstat
270 DRV_KMODS += ksyms
271 DRV_KMODS += kmdb
272 DRV_KMODS += llcl
273 DRV_KMODS += lofi
274 DRV_KMODS += log
275 DRV_KMODS += logindmux
276 DRV_KMODS += mega_sas
277 DRV_KMODS += mc-amd
278 DRV_KMODS += mm
279 DRV_KMODS += mouse8042
280 DRV_KMODS += mpt_sas3
281 #endif /* ! codereview */
282 DRV_KMODS += mpt_sas
283 DRV_KMODS += mr_sas
284 DRV_KMODS += mwl
285 DRV_KMODS += nca
286 DRV_KMODS += nsmb
287 DRV_KMODS += nulldriver
288 DRV_KMODS += nv_sata
289 DRV_KMODS += nxge
290 DRV_KMODS += oce
291 DRV_KMODS += openeep
292 DRV_KMODS += pci_pci
293 DRV_KMODS += pcic
294 DRV_KMODS += pcieb
295 DRV_KMODS += physmem
296 DRV_KMODS += pit_beeep
297 DRV_KMODS += pm
298 DRV_KMODS += poll
299 DRV_KMODS += pool
300 DRV_KMODS += power
301 DRV_KMODS += pseudo
302 DRV_KMODS += ptc
303 DRV_KMODS += ptm
304 DRV_KMODS += pts
305 DRV_KMODS += ptsl
306 DRV_KMODS += qlge
307 DRV_KMODS += radeon
308 DRV_KMODS += ral
309 DRV_KMODS += ramdisk
310 DRV_KMODS += random
311 DRV_KMODS += rds
312 DRV_KMODS += rds3
313 DRV_KMODS += rpcib
314 DRV_KMODS += rsm
315 DRV_KMODS += rts
316 DRV_KMODS += rtw
317 DRV_KMODS += rum
318 DRV_KMODS += rwd
319 DRV_KMODS += rwn
320 DRV_KMODS += sad
321 DRV_KMODS += sd
322 DRV_KMODS += sdhost
323 DRV_KMODS += sgen
324 DRV_KMODS += si3124
325 DRV_KMODS += smbios

```

```

326 DRV_KMODS += softmac
327 DRV_KMODS += spdssock
328 DRV_KMODS += smbsrv
329 DRV_KMODS += smp
330 DRV_KMODS += spps
331 DRV_KMODS += sppptun
332 DRV_KMODS += srpt
333 DRV_KMODS += st
334 DRV_KMODS += sy
335 DRV_KMODS += sysevent
336 DRV_KMODS += sysmsg
337 DRV_KMODS += tcp
338 DRV_KMODS += tcp6
339 DRV_KMODS += tl
340 DRV_KMODS += tn timer
341 DRV_KMODS += tpm
342 DRV_KMODS += trill
343 DRV_KMODS += udp
344 DRV_KMODS += udp6
345 DRV_KMODS += ucode
346 DRV_KMODS += ural
347 DRV_KMODS += uath
348 DRV_KMODS += urtw
349 DRV_KMODS += vgate
350 DRV_KMODS += heci
351 DRV_KMODS += vnic
352 DRV_KMODS += vscan
353 DRV_KMODS += wc
354 DRV_KMODS += winlock
355 DRV_KMODS += wpi
356 DRV_KMODS += xge
357 DRV_KMODS += yge
358 DRV_KMODS += zcons
359 DRV_KMODS += zyd
360 DRV_KMODS += simnet
361 DRV_KMODS += stmf
362 DRV_KMODS += stmf_sbd
363 DRV_KMODS += fct
364 DRV_KMODS += fcoe
365 DRV_KMODS += fcoet
366 DRV_KMODS += fcoei
367 DRV_KMODS += qlt
368 DRV_KMODS += iscsit
369 DRV_KMODS += pppt
370 DRV_KMODS += ncall nsctl sdbc nskern sv
371 DRV_KMODS += ii rdc rdcsrv rdcstub
372 DRV_KMODS += iptun

374 #
375 # Common code drivers
376 #

378 DRV_KMODS += afe
379 DRV_KMODS += atge
380 DRV_KMODS += bfe
381 DRV_KMODS += dmfe
382 DRV_KMODS += e1000g
383 DRV_KMODS += efe
384 DRV_KMODS += elxl
385 DRV_KMODS += hme
386 DRV_KMODS += mxfe
387 DRV_KMODS += nge
388 DRV_KMODS += pcn
389 DRV_KMODS += rge
390 DRV_KMODS += rtls
391 DRV_KMODS += sfe

```

```

392 DRV_KMODS      += amd8111s
393 DRV_KMODS      += igb
394 DRV_KMODS      += ipmi
395 DRV_KMODS      += iprb
396 DRV_KMODS      += ixgbe
397 DRV_KMODS      += vr

399 #
400 # Virtio drivers
401 #

403 # Virtio core
404 DRV_KMODS      += virtio

406 # Virtio block driver
407 DRV_KMODS      += vioblk

409 #
410 #      DTrace and DTrace Providers
411 #
412 DRV_KMODS      += dtrace
413 DRV_KMODS      += fbt
414 DRV_KMODS      += lockstat
415 DRV_KMODS      += profile
416 DRV_KMODS      += sdt
417 DRV_KMODS      += systtrace
418 DRV_KMODS      += fasttrap
419 DRV_KMODS      += dcpc

421 #
422 #      I/O framework test drivers
423 #
424 DRV_KMODS      += pshot
425 DRV_KMODS      += gen_drv
426 DRV_KMODS      += tvhCi tphci tclient
427 DRV_KMODS      += emul64

429 #
430 #      Machine Specific Driver Modules (/kernel/drv):
431 #
432 DRV_KMODS      += options
433 DRV_KMODS      += scsi_vhci
434 DRV_KMODS      += pmcs
435 DRV_KMODS      += pmcs8001fw
436 DRV_KMODS      += arcmsr
437 DRV_KMODS      += fcp
438 DRV_KMODS      += fcip
439 DRV_KMODS      += fcsm
440 DRV_KMODS      += fp
441 DRV_KMODS      += qlc
442 DRV_KMODS      += iscsi

444 #
445 #      PCMCIA specific module(s)
446 #
447 DRV_KMODS      += pcs
448 MISC_KMODS     += cardbus

450 #
451 #      SCSI Enclosure Services driver
452 #
453 DRV_KMODS      += ses

455 #
456 #      USB specific modules
457 #

```

```

458 DRV_KMODS      += hid
459 DRV_KMODS      += hwarc hwhac
460 DRV_KMODS      += hubd
461 DRV_KMODS      += uhci
462 DRV_KMODS      += ehci
463 DRV_KMODS      += ohci
464 DRV_KMODS      += usb_mid
465 DRV_KMODS      += usb_ia
466 DRV_KMODS      += scsa2usb
467 DRV_KMODS      += usbprn
468 DRV_KMODS      += ugen
469 DRV_KMODS      += usbser
470 DRV_KMODS      += usbsacm
471 DRV_KMODS      += usbsksp
472 DRV_KMODS      += usbsprl
473 DRV_KMODS      += usb_ac
474 DRV_KMODS      += usb_as
475 DRV_KMODS      += usbskel
476 DRV_KMODS      += usbvc
477 DRV_KMODS      += usbftdi
478 DRV_KMODS      += wusb_df
479 DRV_KMODS      += wusb_ca
480 DRV_KMODS      += usbecm

482 #
483 #      1394 modules
484 #
485 MISC_KMODS      += s1394 sbp2
486 DRV_KMODS      += hci1394 scsa1394
487 DRV_KMODS      += av1394
488 DRV_KMODS      += dcaml394

490 #
491 #      InfiniBand pseudo drivers
492 #
493 DRV_KMODS      += ib ibp eibnx eoib rdsib sdp iser daplt hermon tavor sol_ucma
494 DRV_KMODS      += sol_umad

496 #
497 #      LVM modules
498 #
499 DRV_KMODS      += md
500 MISC_KMODS      += md_stripe md_hotspares md_mirror md_raid md_trans md_notify
501 MISC_KMODS      += md_sp

503 #
504 #      Brand modules
505 #
506 BRAND_KMODS     += snl_brand s10_brand

508 #
509 #      Exec Class Modules (/kernel/exec):
510 #
511 EXEC_KMODS      += elfexec intpexec shbinexec javaexec

513 #
514 #      Scheduling Class Modules (/kernel/sched):
515 #
516 SCHED_KMODS     += IA RT TS RT_DPTBL TS_DPTBL FSS FX FX_DPTBL SDC

518 #
519 #      File System Modules (/kernel/fs):
520 #
521 FS_KMODS        += autofs cacheefs ctfs dcfs dev devfs fdfs fifofs hsfs lofs
522 FS_KMODS        += mntfs namefs nfs objfs zfs zut
523 FS_KMODS        += pcfs procfs sockfs specfs tmpfs udfs ufs sharefs

```

```

524 FS_KMODS      += smbfs

526 #
527 #      Streams Modules (/kernel/strmod):
528 #
529 STRMOD_KMODS    += bufmod connld dedump ldterm pckt pfmod pipemod
530 STRMOD_KMODS    += ptem redirmod rpcmod rlmod telmod timod
531 STRMOD_KMODS    += sppedsyn spppcomp
532 STRMOD_KMODS    += tirdwr ttcompat
533 STRMOD_KMODS    += usbkbm
534 STRMOD_KMODS    += usbms
535 STRMOD_KMODS    += usbwcm
536 STRMOD_KMODS    += usb_ah
537 STRMOD_KMODS    += drcompat
538 STRMOD_KMODS    += cryptmod
539 STRMOD_KMODS    += vuid2ps2
540 STRMOD_KMODS    += vuid3ps2
541 STRMOD_KMODS    += vuidm3p
542 STRMOD_KMODS    += vuidm4p
543 STRMOD_KMODS    += vuidm5p

545 #
546 #      'System' Modules (/kernel/sys):
547 #
548 SYS_KMODS       += c2audit
549 SYS_KMODS       += doorfs
550 SYS_KMODS       += exacctsys
551 SYS_KMODS       += inst_sync
552 SYS_KMODS       += kaio
553 SYS_KMODS       += msgsys
554 SYS_KMODS       += pipe
555 SYS_KMODS       += portfs
556 SYS_KMODS       += pset
557 SYS_KMODS       += semsys
558 SYS_KMODS       += shmsys
559 SYS_KMODS       += sysacct
560 SYS_KMODS       += acctctl

562 #
563 #      'Misc' Modules (/kernel/misc)
564 #      MISC_KMODS are built both 32-bit and 64-bit
565 #      MISC_KMODS_32 are built only 32-bit
566 #      MISC_KMODS_64 are built only 64-bit
567 #
568 MISC_KMODS      += ac97
569 MISC_KMODS      += acpica
570 MISC_KMODS      += agpmaster
571 MISC_KMODS      += bignum
572 MISC_KMODS      += bootdev
573 MISC_KMODS      += busra
574 MISC_KMODS      += cmlb
575 MISC_KMODS      += consconfig
576 MISC_KMODS      += ctf
577 MISC_KMODS      += dadk
578 MISC_KMODS      += dcopy
579 MISC_KMODS      += dls
580 MISC_KMODS      += drm
581 MISC_KMODS      += fssnap_if
582 MISC_KMODS      += gda
583 MISC_KMODS      += gld
584 MISC_KMODS      += hidparser
585 MISC_KMODS      += hook
586 MISC_KMODS      += hpcsvc
587 MISC_KMODS      += ibcm
588 MISC_KMODS      += ibdm
589 MISC_KMODS      += ibdma

```

```

590 MISC_KMODS      += ibmf
591 MISC_KMODS      += ibtl
592 MISC_KMODS      += idm
593 MISC_KMODS      += idmap
594 MISC_KMODS      += iomulib
595 MISC_KMODS      += ipc
596 MISC_KMODS      += kbtrans
597 MISC_KMODS      += kcf
598 MISC_KMODS      += kgssapi
599 MISC_KMODS      += kmecch_dummy
600 MISC_KMODS      += kmecch_krb5
601 MISC_KMODS      += ksocket
602 MISC_KMODS      += mac
603 MISC_KMODS      += mii
604 MISC_KMODS      += mwlfw
605 MISC_KMODS      += net80211
606 MISC_KMODS      += nfs_dlboot
607 MISC_KMODS      += nfssrv
608 MISC_KMODS      += neti
609 MISC_KMODS      += pci_autoconfig
610 MISC_KMODS      += pcicfg
611 MISC_KMODS      += pcihp
612 MISC_KMODS      += pcmcia
613 MISC_KMODS      += rpcsec
614 MISC_KMODS      += rpcsec_gss
615 MISC_KMODS      += rsmops
616 MISC_KMODS      += sata
617 MISC_KMODS      += scsi
618 MISC_KMODS      += sda
619 MISC_KMODS      += sol_ofs
620 MISC_KMODS      += spuni
621 MISC_KMODS      += strategy
622 MISC_KMODS      += strplumb
623 MISC_KMODS      += tem
624 MISC_KMODS      += tlimod
625 MISC_KMODS      += usba usba10 usbs49_fw
626 MISC_KMODS      += scsi_vhci_f_sym_hds
627 MISC_KMODS      += scsi_vhci_f_sym
628 MISC_KMODS      += scsi_vhci_f_tpgs
629 MISC_KMODS      += scsi_vhci_f_asym_sun
630 MISC_KMODS      += scsi_vhci_f_tape
631 MISC_KMODS      += scsi_vhci_f_tpgs_tape
632 MISC_KMODS      += fctl
633 MISC_KMODS      += emlxs_fw
634 MISC_KMODS      += qlc_fw_2200
635 MISC_KMODS      += qlc_fw_2300
636 MISC_KMODS      += qlc_fw_2400
637 MISC_KMODS      += qlc_fw_2500
638 MISC_KMODS      += qlc_fw_6322
639 MISC_KMODS      += qlc_fw_8100
640 MISC_KMODS      += hwa1480_fw
641 MISC_KMODS      += uathfw
642 MISC_KMODS      += uwba

644 MISC_KMODS      += klmmod klmops

646 #
647 #      Software Cryptographic Providers (/kernel/crypto):
648 #
649 CRYPTO_KMODS    += aes
650 CRYPTO_KMODS    += arcfour
651 CRYPTO_KMODS    += blowfish
652 CRYPTO_KMODS    += des
653 CRYPTO_KMODS    += ecc
654 CRYPTO_KMODS    += md4
655 CRYPTO_KMODS    += md5

```

```

656 CRYPTO_KMODS      += rsa
657 CRYPTO_KMODS      += sha1
658 CRYPTO_KMODS      += sha2
659 CRYPTO_KMODS      += swrand

661 #
662 #       IP Policy Modules (/kernel/ipp)
663 #
664 IPP_KMODS           += dlcsmk
665 IPP_KMODS           += flowacct
666 IPP_KMODS           += ipgpc
667 IPP_KMODS           += dscpmk
668 IPP_KMODS           += tokenmt
669 IPP_KMODS           += tswtclmt

671 #
672 #       generic-unix module (/kernel/genunix):
673 #
674 GENUNIX_KMODS      += genunix

676 #
677 #       Modules eXcluded from the product:
678 #

680 #
681 #       'Dacf' Modules (/kernel/dacf):
682 #

684 #
685 #       Performance Counter BackEnd modules (/usr/kernel/pcbe)
686 #
687 PCBE_KMODS          += p123_pcbe p4_pcbe opteron_pcbe core_pcbe

689 #
690 #       MAC-Type Plugin Modules (/kernel/mac)
691 #
692 MAC_KMODS           += mac_6to4
693 MAC_KMODS           += mac_ether
694 MAC_KMODS           += mac_ipv4
695 MAC_KMODS           += mac_ipv6
696 MAC_KMODS           += mac_wifi
697 MAC_KMODS           += mac_ib

699 #
700 #       socketmod (kernel/socketmod)
701 #
702 SOCKET_KMODS        += sockpfp
703 SOCKET_KMODS        += socksctp
704 SOCKET_KMODS        += socksdp
705 SOCKET_KMODS        += sockrds
706 SOCKET_KMODS        += ksslf

708 #
709 #       kiconv modules (/kernel/kiconv):
710 #
711 KICONV_KMODS        += kiconv_emea kiconv_ja kiconv_ko kiconv_sc kiconv_tc

713 #
714 #       'Dacf' Modules (/kernel/dacf):
715 #
716 DACF_KMODS          += net_dacf

718 #
719 # Ensure that the variable member of the cpu_t (cpu_m) is defined
720 # for the lint builds so as not to cause lint errors during the
721 # global cross check.

```

```

722 #
723 LINTFLAGS            += -D_MACHDEP -I$(UTSBASE)/i86pc

```

new/usr/src/uts/intel/mpt_sas3/Makefile

1

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

```
1 #
2 # CDDL HEADER START
3 #
4 # The contents of this file are subject to the terms of the
5 # Common Development and Distribution License (the "License").
6 # You may not use this file except in compliance with the License.
7 #
8 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9 # or http://www.opensolaris.org/os/licensing.
10 # See the License for the specific language governing permissions
11 # and limitations under the License.
12 #
13 # When distributing Covered Code, include this CDDL HEADER in each
14 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 # If applicable, add the following below this CDDL HEADER, with the
16 # fields enclosed by brackets "[]" replaced with your own identifying
17 # information: Portions Copyright [yyyy] [name of copyright owner]
18 #
19 # CDDL HEADER END
20 #
21 # Copyright 2009 Sun Microsystems, Inc. All rights reserved.
22 # Use is subject to license terms.
23 #
24 # This makefile drives the production of the mpt_sas3 driver
25 # kernel module.
26 #
27 # intel architecture dependent
28 #
29 #
30 #
31 # Paths to the base of the uts directory trees
32 #
33 UTSBASE = ../../../../src/uts
34 #
35 #
36 # Define the module and object file sets.
37 #
38 MODULE = mpt_sas3
39 OBJECTS = $(MPTSAS3_OBJS:%=$(OBJDIR)/%)
40 LINTS = $(MPTSAS3_OBJS:%.o=$(LINTS_DIR)/%.ln)
41 ROOTMODULE = $(ROOT_DRV_DIR)/$(MODULE)
42 CONF_SRCDIR = $(UTSBASE)/common/io/scsi/adapters/mpt_sas3/
43 WARLOCK_OUT = $(MPTSAS3_OBJS:%.o=%%.ll)
44 WARLOCK_OK = $(MODULE).ok
45 WLCMD_DIR = $(UTSBASE)/common/io/warlock
46 #
47 #
48 # Kernel Module Dependencies
49 #
50 LDLFLAGS += -dy -Nmisc/scsi -Ndrv/scsi_vhci
51 #
52 #
53 # Define targets
54 #
55 ALL_TARGET = $(BINARY) $(CONFMOD)
56 LINT_TARGET = $(MODULE).lint
57 INSTALL_TARGET = $(BINARY) $(ROOTMODULE) $(ROOT_CONFFILE)
58 #
59 #
60 # Include common rules.
61 #
```

new/usr/src/uts/intel/mpt_sas3/Makefile

2

```
62 include $(UTSBASE)/intel/Makefile.intel
63 #
64 CERRWARN += -_gcc=-Wno-parentheses
65 CERRWARN += -_gcc=-Wno-uninitialized
66 CERRWARN += -_gcc=-Wno-unused-label
67 CERRWARN += -_gcc=-Wno-switch
68 #
69 #
70 # Default build targets.
71 #
72 .KEEP_STATE:
73 #
74 all: $(ALL_DEPS)
75 #
76 def: $(DEF_DEPS)
77 #
78 clean: $(CLEAN_DEPS)
79 $(RM) $(WARLOCK_OUT) $(WARLOCK_OK)
80 #
81 clobber: $(CLOBBER_DEPS)
82 $(RM) $(WARLOCK_OUT) $(WARLOCK_OK)
83 #
84 lint: $(LINT_DEPS)
85 #
86 modlintlib: $(MODLINTLIB_DEPS)
87 #
88 clean.lint: $(CLEAN_LINT_DEPS)
89 #
90 install: $(INSTALL_DEPS)
91 #
92 #
93 # Include common targets.
94 #
95 include $(UTSBASE)/intel/Makefile.targ
96 #
97 #
98 # Defines for local commands.
99 #
100 WARLOCK = warlock
101 WLCC = wlcc
102 TOUCH = touch
103 TEST = test
104 #
105 #
106 # lock_lint rules
107 #
108 SCSI_FILES = $(SCSI_OBJS:%.o=-l $(UTSBASE)/intel/scsi/%.ll)
109 #
110 warlock: $(WARLOCK_OK)
111 #
112 $(WARLOCK_OK): $(WARLOCK_OUT) warlock_ddi.files scsi.files \
113 $(WLCMD_DIR)/mptsas.wlcmd
114 $(WARLOCK) -c $(WLCMD_DIR)/mptsas.wlcmd $(WARLOCK_OUT) \
115 $(SCSI_FILES) \
116 $(UTSBASE)/intel/warlock/scsi.ll \
117 -l $(UTSBASE)/intel/warlock/ddi_dki_impl.ll
118 $(TOUCH) $@
119 #
120 %.ll: $(UTSBASE)/common/io/scsi/adapters/mpt_sas3/%.c
121 $(WLCC) $(CPPFLAGS) -DDEBUG -o $@ $<
122 #
123 warlock_ddi.files:
124 @cd $(UTSBASE)/intel/warlock; pwd; $(MAKE) warlock
125 #
126 scsi.files:
127 @cd $(UTSBASE)/intel/scsi; pwd; $(MAKE) warlock
```

new/usr/src/uts/intel/mpt_sas3/Makefile

3

129 #endif /* ! codereview */

new/usr/src/uts/sparc/Makefile.sparc

1

```
*****
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
*****

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

22 # Copyright (c) 2005, 2010, Oracle and/or its affiliates. All rights reserved.
23 # Copyright (c) 2013 Andrew Stormont. All rights reserved.

26 #
27 # This makefile contains the common definitions for all sparc
28 # implementation architecture independent modules.
29 #

31 #
32 # Define supported builds
33 #
34 DEF_BUILDS = $(DEF_BUILDS64)
35 ALL_BUILDS = $(ALL_BUILDS64)

37 #
38 # Everybody needs to know how to build modstubs.o and to locate unix.o.
39 # Note that unix.o must currently be selected from among the possible
40 # "implementation architectures". Note further, that unix.o is only
41 # used as an optional error check for undefines so (theoretically)
42 # any "implementation architectures" could be used. We choose sun4u
43 # because it is the reference port.
44 #
45 UNIX_DIR = $(UTSBASE)/sun4u/unix
46 GENLIB_DIR = $(UTSBASE)/sun4u/genunix
47 IPDRV_DIR = $(UTSBASE)/sparc/ip
48 MODSTUBS_DIR = $(UNIX_DIR)
49 DSF_DIR = $(UNIX_DIR)
50 LINTS_DIR = $(OBJSDIR)
51 LINT_LIB_DIR = $(UTSBASE)/sparc/lint-libs/$(OBJSDIR)

53 UNIX_O = $(UNIX_DIR)/$(OBJSDIR)/unix.o
54 MODSTUBS_O = $(MODSTUBS_DIR)/$(OBJSDIR)/modstubs.o
55 GENLIB = $(UTSBASE)/sun4u/lint-libs/$(OBJSDIR)/libgenunix.so

57 LINT_LIB_32 = $(UTSBASE)/sun4u/lint-libs/$(OBJSDIR)/llib-lunix.ln
58 GEN_LINT_LIB_32 = $(UTSBASE)/sun4u/lint-libs/$(OBJSDIR)/llib-lgenunix.ln

60 LINT_LIB_64 = $(UTSBASE)/sun4u/lint-libs/$(OBJSDIR)/llib-lunix.ln
61 GEN_LINT_LIB_64 = $(UTSBASE)/sun4u/lint-libs/$(OBJSDIR)/llib-lgenunix.ln
```

new/usr/src/uts/sparc/Makefile.sparc

2

```
63 LINT_LIB = $(LINT_LIB_$(CLASS))
64 GEN_LINT_LIB = $(GEN_LINT_LIB_$(CLASS))

66 LINT32_DIRS = $(LINT32_BUILDS:=$(UTSBASE)/sparc/lint-libs/%)
67 LINT32_FILES = $(LINT32_DIRS:=$(MODULE).ln)

69 LINT64_DIRS = $(LINT64_BUILDS:=$(UTSBASE)/sparc/lint-libs/%)
70 LINT64_FILES = $(LINT64_DIRS:=$(MODULE).ln)

72 #
73 # Include the makefiles which define build rule templates, the
74 # collection of files per module, and a few specific flags. Note
75 # that order is significant, just as with an include path. The
76 # first build rule template which matches the files name will be
77 # used. By including these in order from most machine dependent
78 # to most machine independent, we allow a machine dependent file
79 # to be used in preference over a machine independent version
80 # (Such as a machine specific optimization, which preserves the
81 # interfaces.)
82 #
83 include $(UTSBASE)/sparc/Makefile.files
84 include $(UTSBASE)/sparc/v9/Makefile.files
85 include $(UTSBASE)/sun/Makefile.files
86 include $(UTSBASE)/common/Makefile.files

88 #
89 # ----- TRANSITIONAL SECTION -----
90 #

92 #
93 # Not everything which *should* be a module is a module yet. The
94 # following is a list of such objects which are currently part of
95 # genunix but which might someday become kmods. This must be
96 # defined before we include Makefile.uts, or else genunix's build
97 # won't be as parallel as we might like.
98 #
99 NOT_YET_KMODS = $(OLDPTY_OBJS) $(PTY_OBJS) $(VCONS_CONF_OBJS) $(MOD_OBJS)

101 #
102 # ----- END OF TRANSITIONAL SECTION -----
103 #
104 # Include machine independent rules. Note that this does not imply
105 # that the resulting module from rules in Makefile.uts is machine
106 # independent. Only that the build rules are machine independent.
107 #
108 include $(UTSBASE)/Makefile.uts

110 #
111 # machine specific optimization, override default in Makefile.master
112 #
113 XARCH_32 = -xarch=v8
114 XARCH_64 = -m64
115 XARCH = $(XARCH_$(CLASS))

117 COPTIMIZE_32 = -xO3
118 COPTIMIZE_64 = -xO3
119 COPTIMIZE = $(COPTIMIZE_$(CLASS))

121 CCMODE = -Xa

123 CFLAGS_32 = -xcg92
124 CFLAGS_64 = -xchip=ultra $(CCABS32) $(CCREGSYM)
125 CFLAGS = $(CFLAGS_$(CLASS))

127 CFLAGS += $(XARCH)
```

```

128 CFLAGS      += $(COPTIMIZE)
129 CFLAGS      += $(EXTRA_CFLAGS)
130 CFLAGS      += $(XAOPT)
131 CFLAGS      += $(INLINES) -D_ASM_INLINES
132 CFLAGS      += $(CCMODE)
133 CFLAGS      += $(SPACEFLAG)
134 CFLAGS      += $(CERRWARN)
135 CFLAGS      += $(CTF_FLAGS_$(CLASS))
136 CFLAGS      += $(C99MODE)
137 CFLAGS      += $(CCUNBOUND)
138 CFLAGS      += $(CCSTATICSYM)
139 CFLAGS      += $(CC32BITCALLERS)
140 CFLAGS      += $(CCNOAUTOINLINE)
141 CFLAGS      += $(IROPTFLAG)
142 CFLAGS      += $(CGLOBALSTATIC)
143 CFLAGS      += -xregs=no%float
144 CFLAGS      += -xstrconst
145 CFLAGS      += $(CSOURCEDEBUGFLAGS)
146 CFLAGS      += $(CUSERFLAGS)

148 ASFLAGS      += $(XARCH)

150 LINT_DEFS_32  =
151 LINT_DEFS_64  = -m64
152 LINT_DEFS     += $(LINT_DEFS_$(CLASS))

154 #
155 #       The following must be defined for all implementations:
156 #
157 #       MODSTUBS:       Module stubs source file.
158 #
159 MODSTUBS      = $(UTSBASE)/sparc/ml/modstubs.s

161 #
162 #       Define the actual specific platforms - obviously none.
163 #
164 MACHINE_DEFS   =

166 #
167 #       Debugging level
168 #
169 #       Special knowledge of which special debugging options effect which
170 #       file is used to optimize the build if these flags are changed.
171 #
172 #       XXX: The above could possibly be done for more flags and files, but
173 #       is left as an experiment to the interested reader. Be forewarned,
174 #       that excessive use could lead to maintenance difficulties.
175 #
176 DEBUG_DEFS_OBJ32 =
177 DEBUG_DEFS_DBG32  = -DDEBUG
178 DEBUG_DEFS_OBJ64  =
179 DEBUG_DEFS_DBG64  = -DDEBUG
180 DEBUG_DEFS        = $(DEBUG_DEFS_$(BUILD_TYPE))

182 DEBUG_COND_OBJ32 = $(POUND_SIGN)
183 DEBUG_COND_DBG32  =
184 DEBUG_COND_OBJ64  = $(POUND_SIGN)
185 DEBUG_COND_DBG64  =
186 IF_DEBUG_OBJ      = $(DEBUG_COND_$(BUILD_TYPE))$(OBJS_DIR)/

188 $(IF_DEBUG_OBJ)syscall.o      :=      DEBUG_DEFS      += -DSYSCALLTRACE
189 $(IF_DEBUG_OBJ)clock.o       :=      DEBUG_DEFS      += -DKSLICE=1

191 # Comment these out if you don't want dispatcher lock statistics.

193 # $(IF_DEBUG_OBJ)disp_lock.o   := DEBUG_DEFS      += -DDISP_LOCK_STATS

```

```

195 #
196 #       Collect the preprocessor definitions to be associated with *all*
197 #       files.
198 #
199 ALL_DEFS      = $(MACHINE_DEFS) $(DEBUG_DEFS) $(OPTION_DEFS)
200 #
201 #
202 #       The kernels modules which are "implementation architecture"
203 #       specific for this machine are enumerated below. Note that most
204 #       of these modules must exist (in one form or another) for each
205 #       architecture.
206 #
207 #       Common Drivers (usually pseudo drivers) (/kernel/drv):
208 #
209 DRV_KMODS      += aggr arp audio bl blkdev bofi clone cn conskbd consms cpuid
210 DRV_KMODS      += crypto cryptoadm devinfo dump
211 DRV_KMODS      += dtrace fasttrap fbt lockstat profile sdt systrace dcpc
212 DRV_KMODS      += fssnap icmp icmp6 ip ip6 ipnet ipsecah
213 DRV_KMODS      += ipsecesp iptun iwscn keysock kmdb kstat ksyms llc1
214 DRV_KMODS      += lofi
215 DRV_KMODS      += log logindmux kssl mm nca physmem pm poll pool
216 DRV_KMODS      += pseudo ptc ptm pts ptsl ramdisk random rsm rts sad
217 DRV_KMODS      += simnet softmac sPPP sppptun sy sysevent sysmsg
218 DRV_KMODS      += spdsock
219 DRV_KMODS      += tcp tcp6 tl tn timer ttymux udp udp6 wc winlock zcons
220 DRV_KMODS      += ippctl
221 DRV_KMODS      += dld
222 DRV_KMODS      += ipd
223 DRV_KMODS      += ipf
224 DRV_KMODS      += rpcib
225 DRV_KMODS      += dlpistub
226 DRV_KMODS      += vnic
227 DRV_KMODS      += xge
228 DRV_KMODS      += rds
229 DRV_KMODS      += rdsrv3
230 DRV_KMODS      += chxge
231 DRV_KMODS      += smbsrv
232 DRV_KMODS      += vscan
233 DRV_KMODS      += nsmb
234 DRV_KMODS      += fm
235 DRV_KMODS      += nulldriver
236 DRV_KMODS      += bridge trill
237 DRV_KMODS      += bpf
238 DRV_KMODS      += dca

240 #
241 #       Hardware Drivers in common space
242 #
243 #
244 DRV_KMODS      += afe
245 DRV_KMODS      += audio1575
246 DRV_KMODS      += audioens
247 DRV_KMODS      += audiols
248 DRV_KMODS      += audiopl6x
249 DRV_KMODS      += audiopci
250 DRV_KMODS      += audiotls
251 DRV_KMODS      += el000g
252 DRV_KMODS      += efe
253 DRV_KMODS      += hxge
254 DRV_KMODS      += mxfe
255 DRV_KMODS      += rge
256 DRV_KMODS      += rtls
257 DRV_KMODS      += sfe
258 DRV_KMODS      += aac
259 DRV_KMODS      += igb

```



```

260 DRV_KMODS      += ixgbe
261 DRV_KMODS      += vr
262 DRV_KMODS      += mr_sas
263 DRV_KMODS      += yge

265 #
266 #      Machine Specific Driver Modules (/kernel/drv):
267 #
268 DRV_KMODS      += audiocs
269 DRV_KMODS      += bge dmfe eri fas hme qfe
270 DRV_KMODS      += openeepr options sd ses st
271 DRV_KMODS      += ssd
272 DRV_KMODS      += ecpp
273 DRV_KMODS      += hid hubd ehci ohci uhci usb_mid usb_ia scsa2usb usbprn ugen
274 DRV_KMODS      += usbser usbsacm usbsksp usbsprl
275 DRV_KMODS      += usb_as usb_ac
276 DRV_KMODS      += usbskel
277 DRV_KMODS      += usbvc
278 DRV_KMODS      += usbftdi
279 DRV_KMODS      += wusb_df hwahc hwarc wusb_ca
280 DRV_KMODS      += usbecm
281 DRV_KMODS      += hcil394 avl394 scsa1394 dcaml394
282 DRV_KMODS      += sbp2
283 DRV_KMODS      += ib ibp eibnx eoib rdsib sdp iser daplt hermon tavor sol_ucma
284 DRV_KMODS      += sol_umad
285 DRV_KMODS      += pci_pci pcieb pcieb_bcm
286 DRV_KMODS      += i8042 kb8042 mouse8042
287 DRV_KMODS      += fcode
288 DRV_KMODS      += mpt_gas3
289 #endif /* ! codereview */
290 DRV_KMODS      += mpt_gas
291 DRV_KMODS      += socal
292 DRV_KMODS      += sgen
293 DRV_KMODS      += myril0ge
294 DRV_KMODS      += smp
295 DRV_KMODS      += dad
296 DRV_KMODS      += scsi_vhci
297 DRV_KMODS      += fcp
298 DRV_KMODS      += fcip
299 DRV_KMODS      += fcsn
300 DRV_KMODS      += fp
301 DRV_KMODS      += qlc
302 DRV_KMODS      += qlge
303 DRV_KMODS      += stmf
304 DRV_KMODS      += stmf_sbd
305 DRV_KMODS      += fct
306 DRV_KMODS      += fcoe
307 DRV_KMODS      += fcoet
308 DRV_KMODS      += fcoei
309 DRV_KMODS      += qlt
310 DRV_KMODS      += iscsit
311 DRV_KMODS      += pppt
312 DRV_KMODS      += ncall nsctl sdbc nsfern sv
313 DRV_KMODS      += ii rdc rdcsrv rdcstub
314 DRV_KMODS      += iscsi
315 DRV_KMODS      += emlxs
316 DRV_KMODS      += oce
317 DRV_KMODS      += srpt
318 DRV_KMODS      += pmcs
319 DRV_KMODS      += pmcs8001fw

321 #
322 #      I/O framework test drivers
323 #
324 DRV_KMODS      += pshot
325 DRV_KMODS      += gen_drv

```

```

326 DRV_KMODS      += tvhci tphci tclient
327 DRV_KMODS      += emul64

329 #
330 #      PCMCIA specific module(s)
331 #
332 DRV_KMODS      += pcs
333 MISC_KMODS     += busra cardbus dada pcmcia
334 DRV_KMODS      += pcic

336 # Add lvm
337 #
338 DRV_KMODS      += md
339 MISC_KMODS     += md_mirror md_stripe md_hotspares md_raid md_trans md_notify
340 MISC_KMODS     += md_sp

342 #
343 #      Exec Class Modules (/kernel/exec):
344 #
345 EXEC_KMODS     += aoutexec elfexec intpexec shbinexec javaexec

347 #
348 #      Scheduling Class Modules (/kernel/sched):
349 #
350 SCHED_KMODS    += RT TS RT_DPTBL TS_DPTBL IA FSS FX FX_DPTBL SDC

352 #
353 #      File System Modules (/kernel/fs):
354 #
355 FS_KMODS       += dev devfs fdfs fifofs hfsfs lofs namefs nfs pcfs tmpfs zfs
356 FS_KMODS       += zut specfs udfs ufs autofs cacheofs procfs sockfs mntfs
357 FS_KMODS       += ctfs objfs sharefs dcfs smbfs

359 #
360 #      Streams Modules (/kernel/strmod):
361 #
362 STRMOD_KMODS   += bufmod connld dedump ldterm ms pckt pfmod
363 STRMOD_KMODS   += pipemod ptem redirmod rpcmod rlmod telmod timod
364 STRMOD_KMODS   += sppsasyn sppscomp
365 STRMOD_KMODS   += tirdwr ttcompat
366 STRMOD_KMODS   += usbkbm usbms usbwcm usb_ah
367 STRMOD_KMODS   += drcompat
368 STRMOD_KMODS   += cryptmod
369 STRMOD_KMODS   += vuid3ps2

371 #
372 #      'System' Modules (/kernel/sys):
373 #
374 SYS_KMODS      += c2audit
375 SYS_KMODS      += exacetsys
376 SYS_KMODS      += inst_sync kaio msgsys semsys shmsys sysacct pipe
377 SYS_KMODS      += doorfs pset acctctl portfs

379 #
380 #      'User' Modules (/kernel/misc):
381 #
382 MISC_KMODS     += ac97
383 MISC_KMODS     += bignum
384 MISC_KMODS     += consconfig gld ipc nfs_dlboot nfssrv scsi
385 MISC_KMODS     += strplumb swapgeneric tlimod
386 MISC_KMODS     += rpcsec rpcsec_gss kgssapi kmecch_dummy
387 MISC_KMODS     += kmecch_krb5
388 MISC_KMODS     += fssnap_if
389 MISC_KMODS     += hidparser kbtrans usba usba10 usbs49_fw
390 MISC_KMODS     += sl394
391 MISC_KMODS     += hpcsvc pcihp

```

```

392 MISC_KMODS      += rsmops
393 MISC_KMODS      += kcf
394 MISC_KMODS      += ksocket
395 MISC_KMODS      += ibcm
396 MISC_KMODS      += ibdm
397 MISC_KMODS      += ibdma
398 MISC_KMODS      += ibmf
399 MISC_KMODS      += ibtl
400 MISC_KMODS      += sol_ofs
401 MISC_KMODS      += idm
402 MISC_KMODS      += idmap
403 MISC_KMODS      += hook
404 MISC_KMODS      += neti
405 MISC_KMODS      += ctf
406 MISC_KMODS      += mac_dls
407 MISC_KMODS      += cmlb
408 MISC_KMODS      += tem
409 MISC_KMODS      += pcicfg_fcodem_fcpci
410 MISC_KMODS      += scsi_vhci_f_sym scsi_vhci_f_tpgs scsi_vhci_f_asym_sun
411 MISC_KMODS      += scsi_vhci_f_sym_hds
412 MISC_KMODS      += scsi_vhci_f_tape scsi_vhci_f_tpgs_tape
413 MISC_KMODS      += fctl
414 MISC_KMODS      += emlxs_fw
415 MISC_KMODS      += qlc_fw_2200
416 MISC_KMODS      += qlc_fw_2300
417 MISC_KMODS      += qlc_fw_2400
418 MISC_KMODS      += qlc_fw_2500
419 MISC_KMODS      += qlc_fw_6322
420 MISC_KMODS      += qlc_fw_8100
421 MISC_KMODS      += spuni
422 MISC_KMODS      += hwa1480_fw_uwba
423 MISC_KMODS      += mii

425 MISC_KMODS      += klmmmod klmmops

427 #
428 #       Software Cryptographic Providers (/kernel/crypto):
429 #
430 CRYPTO_KMODS     += aes
431 CRYPTO_KMODS     += arcfour
432 CRYPTO_KMODS     += blowfish
433 CRYPTO_KMODS     += des
434 CRYPTO_KMODS     += md4
435 CRYPTO_KMODS     += md5
436 CRYPTO_KMODS     += ecc
437 CRYPTO_KMODS     += rsa
438 CRYPTO_KMODS     += sha1
439 CRYPTO_KMODS     += sha2
440 CRYPTO_KMODS     += swrand

442 #
443 # IP Policy Modules (/kernel/ipp):
444 #
445 IPP_KMODS        += dlcosmk
446 IPP_KMODS        += flowacct
447 IPP_KMODS        += ipgpc
448 IPP_KMODS        += dscpmk
449 IPP_KMODS        += tokenmt
450 IPP_KMODS        += tswtclmt

452 #
453 # 'Dacf' modules (/kernel/dacf)
454 DACF_KMODS       += consconfig_dacf

456 #
457 #       SVVS Testing Modules (/kernel/strmod):

```

```

458 #
459 #       These are streams and driver modules which are not to be
460 #       delivered with a released system. However, during development
461 #       it is convenient to build and install the SVVS kernel modules.
462 #
463 SVVS_KMODS       += lmodb lmode lmodr lmodt svvslo tidg tivc tmux

465 #
466 #       Modules eXcluded from the product:
467 #
468 XMODS            +=

470 #
471 #       'Dacf' Modules (/kernel/dacf):
472 #
473 DACF_KMODS       += net_dacf

475 #
476 #       MAC-Type Plugin Modules (/kernel/mac)
477 #
478 MAC_KMODS        += mac_6to4
479 MAC_KMODS        += mac_ether
480 MAC_KMODS        += mac_ipv4
481 MAC_KMODS        += mac_ipv6
482 MAC_KMODS        += mac_wifi
483 MAC_KMODS        += mac_ib

485 #
486 # socketmod (kernel/socketmod)
487 #
488 SOCKET_KMODS     += sockpfp
489 SOCKET_KMODS     += socksctp
490 SOCKET_KMODS     += socksdp
491 SOCKET_KMODS     += sockrds
492 SOCKET_KMODS     += ksslfs

494 #
495 #       kiconv modules (/kernel/kiconv):
496 #
497 KICONV_KMODS     += kiconv_emea kiconv_ja kiconv_ko kiconv_sc kiconv_tc

499 #
500 # Ensure that the variable member of the cpu_t (cpu_m) is defined
501 # for the lint builds so as not to cause lint errors during the
502 # global cross check.
503 #
504 $(LINTFLAGSSUPPRESS)LINTFLAGS += -D_MACHDEF -I$(UTSBASE)/sun4 \
505                                -I$(UTSBASE)/sun4u -I$(UTSBASE)/sfmmu

```

new/usr/src/uts/sparc/mpt_sas3/Makefile

1

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

```
1 #
2 # CDDL HEADER START
3 #
4 # The contents of this file are subject to the terms of the
5 # Common Development and Distribution License (the "License").
6 # You may not use this file except in compliance with the License.
7 #
8 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9 # or http://www.opensolaris.org/os/licensing.
10 # See the License for the specific language governing permissions
11 # and limitations under the License.
12 #
13 # When distributing Covered Code, include this CDDL HEADER in each
14 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 # If applicable, add the following below this CDDL HEADER, with the
16 # fields enclosed by brackets "[]" replaced with your own identifying
17 # information: Portions Copyright [yyyy] [name of copyright owner]
18 #
19 # CDDL HEADER END
20 #
21 # Copyright 2009 Sun Microsystems, Inc. All rights reserved.
22 # Use is subject to license terms.
23 #
24 # This makefile drives the production of the mpt_sas3 driver
25 # kernel module.
26 #
27 # Sparc architecture dependent
28 #
29 #
30 #
31 # Path to the base of the uts directory tree (usually /usr/src/uts).
32 #
33 UTSBASE = ../../../../src/uts
34 #
35 #
36 # Define the module and object file sets.
37 #
38 MODULE = mpt_sas3
39 OBJECTS = $(MPTSAS3_OBJS:%=$(OBJDIR)/%)
40 LINTS = $(MPTSAS3_OBJS:%.o=$(LINTSDIR)/%.ln)
41 ROOTMODULE = $(ROOT_DRV_DIR)/$(MODULE)
42 CONF_SRCDIR = $(UTSBASE)/common/io/scsi/adapters/mpt_sas3
43 WARLOCK_OUT = $(MPTSAS3_OBJS:%.o=%%.11)
44 WARLOCK_OK = $(MODULE).ok
45 WLCMD_DIR = $(UTSBASE)/common/io/warlock
46 #
47 #
48 # Kernel Module Dependencies
49 #
50 LDFLAGS += -dy -Nmisc/scsi -Ndrv/scsi_vhci
51 #
52 #
53 # Define targets
54 #
55 ALL_TARGET = $(BINARY) $(CONFMOD)
56 LINT_TARGET = $(MODULE).lint
57 INSTALL_TARGET = $(BINARY) $(ROOTMODULE) $(ROOT_CONFFILE)
58 #
59 #
60 # Include common rules.
61 #
```

new/usr/src/uts/sparc/mpt_sas3/Makefile

2

62 include \$(UTSBASE)/sparc/Makefile.sparc

```
64 #
65 # lint pass one enforcement
66 #
67 CFLAGS += $(CCVERBOSE)
68 #
69 CERRWARN += -_gcc=-Wno-parentheses
70 CERRWARN += -_gcc=-Wno-uninitialized
71 CERRWARN += -_gcc=-Wno-unused-label
72 CERRWARN += -_gcc=-Wno-switch
```

```
74 #
75 # Default build targets.
76 #
77 .KEEP_STATE:
```

79 all: \$(ALL_DEPS)

81 def: \$(DEF_DEPS)

```
83 clean: $(CLEAN_DEPS)
84 $(RM) $(WARLOCK_OUT) $(WARLOCK_OK)
```

```
86 clobber: $(CLOBBER_DEPS)
87 $(RM) $(WARLOCK_OUT) $(WARLOCK_OK)
```

89 lint: \$(LINT_DEPS)

91 modlintlib: \$(MODLINTLIB_DEPS)

93 clean.lint: \$(CLEAN_LINT_DEPS)

95 install: \$(INSTALL_DEPS)

```
97 #
98 # Include common targets.
99 #
```

100 include \$(UTSBASE)/sparc/Makefile.targ

```
102 #
103 # Defines for local commands.
104 #
```

```
105 WARLOCK = warlock
106 WLCC = wlcc
107 TOUCH = touch
108 TEST = test
```

```
110 #
111 # lock_lint rules
112 #
```

113 SCSI_FILES = \$(SCSI_OBJS:%.o=-1 \$(UTSBASE)/sparc/scsi/%.11)

115 warlock: \$(WARLOCK_OK)

```
117 $(WARLOCK_OK): $(WARLOCK_OUT) warlock_ddi.files scsi.files \
118 $(WLCMD_DIR)/mptsas3.wlcmd
119 $(WARLOCK) -c $(WLCMD_DIR)/mptsas3.wlcmd $(WARLOCK_OUT) \
120 $(UTSBASE)/sparc/warlock/scsi.11 \
121 $(SCSI_FILES) \
122 -l $(UTSBASE)/sparc/warlock/ddi_dki_impl.11
123 $(TOUCH) $@
```

```
125 %.11: $(UTSBASE)/common/io/scsi/adapters/mpt_sas3/%.c
126 $(WLCC) $(CPPFLAGS) -DDEBUG -o $@ $<
```

new/usr/src/uts/sparc/mpt_sas3/Makefile

3

```
128 warlock_ddi.files:
129     @cd $(UTSBASE)/sparc/warlock; pwd; $(MAKE) warlock

131 scsi.files:
132     @cd $(UTSBASE)/sparc/scsi; pwd; $(MAKE) warlock

134 #endif /* ! codereview */
```