```
**********************************************************
  422140 Tue Oct 22 09:57:21 2013
new/usr/src/uts/common/io/scsi/adapters/mpt_sas/mptsas.c
4233 mptsas topo change buffer overflow
**********************************************************
______unchanged_portion_omitted_


 698 static int
 699 mptsas_iport_attach(dev_info_t *dip, ddi_attach_cmd_t cmd)
 700 {
 701         dev_info_t              *pdip;
 702         mptsas_t                *mpt;
 703         scsi_hba_tran_t         *hba_tran;
 704         char                    *iport = NULL;
 705         char                    phymask[MPTSAS_MAX_PHYS];
 706         mptsas_phymask_t        phy_mask = 0;
 707         int                     dynamic_port = 0;
 708         uint32_t                page_address;
 709         char                    initiator_wwnstr[MPTSAS_WWN_STRLEN];
 710         int                     rval = DDI_FAILURE;
 711         int                     i = 0;
 712         uint8_t                 numphys = 0;
 713         uint8_t                 phy_id;
 714         uint8_t                 phy_port = 0;
 715         uint16_t                attached_devhdl = 0;
 716         uint32_t                dev_info;
 717         uint64_t                attached_sas_wwn;
 718         uint16_t                dev_hdl;
 719         uint16_t                pdev_hdl;
 720         uint16_t                bay_num, enclosure;
 721         char                    attached_wwnstr[MPTSAS_WWN_STRLEN];

 723         /* CONSTCOND */
 724         ASSERT(NO_COMPETING_THREADS);

 726         switch (cmd) {
 727         case DDI_ATTACH:
 728                 break;

 730         case DDI_RESUME:
 731                 /*
 732                  * If this a scsi-iport node, nothing to do here.
 733                  */
 734                 return (DDI_SUCCESS);

 736         default:
 737                 return (DDI_FAILURE);
 738         }

 740         pdip = ddi_get_parent(dip);

 742         if ((hba_tran = ndi_flavorv_get(pdip, SCSA_FLAVOR_SCSI_DEVICE)) ==
 743             NULL) {
 744                 cmn_err(CE_WARN, "Failed attach iport because fail to "
 745                     "get tran vector for the HBA node");
 746                 return (DDI_FAILURE);
 747         }

 749         mpt = TRAN2MPT(hba_tran);
 750         ASSERT(mpt != NULL);
 751         if (mpt == NULL)
 752                 return (DDI_FAILURE);

 754         if ((hba_tran = ndi_flavorv_get(dip, SCSA_FLAVOR_SCSI_DEVICE)) ==
 755             NULL) {
```



```
 756                 mptsas_log(mpt, CE_WARN, "Failed attach iport because fail to "
 757                     "get tran vector for the iport node");
 758                 return (DDI_FAILURE);
 759         }

 761         /*
 762          * Overwrite parent's tran_hba_private to iport's tran vector
 763          */
 764         hba_tran->tran_hba_private = mpt;

 766         ddi_report_dev(dip);

 768         /*
 769          * Get SAS address for initiator port according dev_handle
 770          */
 771         iport = ddi_get_name_addr(dip);
 772         if (iport && strncmp(iport, "v0", 2) == 0) {
 773                 if (ddi_prop_update_int(DDI_DEV_T_NONE, dip,
 774                     MPTSAS_VIRTUAL_PORT, 1) !=
 775                     DDI_PROP_SUCCESS) {
 776                         (void) ddi_prop_remove(DDI_DEV_T_NONE, dip,
 777                             MPTSAS_VIRTUAL_PORT);
 778                         mptsas_log(mpt, CE_WARN, "mptsas virtual port "
 779                             "prop update failed");
 780                         return (DDI_FAILURE);
 781                 }
 782                 return (DDI_SUCCESS);
 783         }

 785         mutex_enter(&mpt->m_mutex);
 786         for (i = 0; i < MPTSAS_MAX_PHYS; i++) {
 787                 bzero(phymask, sizeof (phymask));
 788                 (void) sprintf(phymask,
 789                     "%x", mpt->m_phy_info[i].phy_mask);
 790                 if (strcmp(phymask, iport) == 0) {
 791                         break;
 792                 }
 793         }

 795         if (i == MPTSAS_MAX_PHYS) {
 796                 mptsas_log(mpt, CE_WARN, "Failed attach port %s because port"
 797                     "seems not exist", iport);
 798                 mutex_exit(&mpt->m_mutex);
 799                 return (DDI_FAILURE);
 800         }

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

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

 809         /*
 810          * Update PHY info for smhba
 811          */
 812         if (mptsas_smhba_phy_init(mpt)) {
 813                 mutex_exit(&mpt->m_mutex);
 814                 mptsas_log(mpt, CE_WARN, "mptsas phy update "
 815                     "failed");
 816                 return (DDI_FAILURE);
 817         }

 819         mutex_exit(&mpt->m_mutex);

 821         numphys = 0;
```

```
822         for (i = 0; i < MPTSAS_MAX_PHYS; i++) {
823                 if ((phy_mask >> i) & 0x01) {
824                         numphys++;
825                 }
826         }

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

832         if (ddi_prop_update_string(DDI_DEV_T_NONE, dip,
833             SCSI_ADDR_PROP_INITIATOR_PORT, initiator_wwnstr) !=
834             DDI_PROP_SUCCESS) {
835                 (void) ddi_prop_remove(DDI_DEV_T_NONE,
836                     dip, SCSI_ADDR_PROP_INITIATOR_PORT);
837                 mptsas_log(mpt, CE_WARN, "mptsas Initiator port "
838                     "prop update failed");
839                 return (DDI_FAILURE);
840         }
841         if (ddi_prop_update_int(DDI_DEV_T_NONE, dip,
842             MPTSAS_NUM_PHYS, numphys) !=
843             DDI_PROP_SUCCESS) {
844                 (void) ddi_prop_remove(DDI_DEV_T_NONE, dip, MPTSAS_NUM_PHYS);
845                 return (DDI_FAILURE);
846         }

848         if (ddi_prop_update_int(DDI_DEV_T_NONE, dip,
849             "phymask", phy_mask) !=
850             DDI_PROP_SUCCESS) {
851                 (void) ddi_prop_remove(DDI_DEV_T_NONE, dip, "phymask");
852                 mptsas_log(mpt, CE_WARN, "mptsas phy mask "
853                     "prop update failed");
854                 return (DDI_FAILURE);
855         }

857         if (ddi_prop_update_int(DDI_DEV_T_NONE, dip,
858             "dynamic-port", dynamic_port) !=
859             DDI_PROP_SUCCESS) {
860                 (void) ddi_prop_remove(DDI_DEV_T_NONE, dip, "dynamic-port");
861                 mptsas_log(mpt, CE_WARN, "mptsas dynamic port "
862                     "prop update failed");
863                 return (DDI_FAILURE);
864         }
865         if (ddi_prop_update_int(DDI_DEV_T_NONE, dip,
866             MPTSAS_VIRTUAL_PORT, 0) !=
867             DDI_PROP_SUCCESS) {
868                 (void) ddi_prop_remove(DDI_DEV_T_NONE, dip,
869                     MPTSAS_VIRTUAL_PORT);
870                 mptsas_log(mpt, CE_WARN, "mptsas virtual port "
871                     "prop update failed");
872                 return (DDI_FAILURE);
873         }
874         mptsas_smhba_set_all_phy_props(mpt, dip, numphys, phy_mask,
875             &attached_devhdl);
874         mptsas_smhba_set_phy_props(mpt,
875             iport, dip, numphys, &attached_devhdl);

877         mutex_enter(&mpt->m_mutex);
878         page_address = (MPI2_SAS_DEVICE_PGAD_FORM_HANDLE &
879             MPI2_SAS_DEVICE_PGAD_FORM_MASK) | (uint32_t)attached_devhdl;
880         rval = mptsas_get_sas_device_page0(mpt, page_address, &dev_hdl,
881             &attached_sas_wwn, &dev_info, &phy_port, &phy_id,
882             &pdev_hdl, &bay_num, &enclosure);
883         if (rval != DDI_SUCCESS) {
884                 mptsas_log(mpt, CE_WARN,
885                     "Failed to get device page0 for handle:%d",
```



```
886                     attached_devhdl);
887                 mutex_exit(&mpt->m_mutex);
888                 return (DDI_FAILURE);
889         }

891         for (i = 0; i < MPTSAS_MAX_PHYS; i++) {
892                 bzero(phymask, sizeof (phymask));
893                 (void) sprintf(phymask, "%x", mpt->m_phy_info[i].phy_mask);
894                 if (strcmp(phymask, iport) == 0) {
895                         (void) sprintf(&mpt->m_phy_info[i].smhba_info.path[0],
896                             "%x",
897                             mpt->m_phy_info[i].phy_mask);
898                 }
899         }
900         mutex_exit(&mpt->m_mutex);

902         bzero(attached_wwnstr, sizeof (attached_wwnstr));
903         (void) sprintf(attached_wwnstr, "w%016"PRIx64,
904             attached_sas_wwn);
905         if (ddi_prop_update_string(DDI_DEV_T_NONE, dip,
906             SCSI_ADDR_PROP_ATTACHED_PORT, attached_wwnstr) !=
907             DDI_PROP_SUCCESS) {
908                 (void) ddi_prop_remove(DDI_DEV_T_NONE,
909                     dip, SCSI_ADDR_PROP_ATTACHED_PORT);
910                 return (DDI_FAILURE);
911         }

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

915         mptsas_create_phy_stats(mpt, iport, dip);

917         /*
918          * register sas hba iport with mdi (MPxIO/vhci)
919          */
920         if (mdi_phci_register(MDI_HCI_CLASS_SCSI,
921             dip, 0) == MDI_SUCCESS) {
922                 mpt->m_mpxio_enable = TRUE;
923         }
924         return (DDI_SUCCESS);
925 }
_____unchanged_portion_omitted_

5800 static void
5801 mptsas_handle_topo_change(mptsas_topo_change_list_t *topo_node,
5802     dev_info_t *parent)
5803 {
5804         mptsas_target_t *ptgt = NULL;
5805         mptsas_smp_t    *psmp = NULL;
5806         mptsas_t        *mpt = (void *)topo_node->mpt;
5807         uint16_t        devhdl;
5808         uint16_t        attached_devhdl;
5809         uint64_t        sas_wwn = 0;
5810         int             rval = 0;
5811         uint32_t        page_address;
5812         uint8_t         phy, flags;
5813         char            *addr = NULL;
5814         dev_info_t      *lundip;
5815         int             circ = 0, circ1 = 0;
5816         char            attached_wwnstr[MPTSAS_WWN_STRLEN];

5818         NDBG20(("mptsas%d handle_topo_change enter", mpt->m_instance));

5820         ASSERT(mutex_owned(&mpt->m_mutex));

5822         switch (topo_node->event) {
5823         case MPTSAS_DR_EVENT_RECONFIG_TARGET:
```

```
5824        {
5825                char *phy_mask_name;
5826                mptsas_phymask_t phymask = 0;

5828                if (topo_node->flags == MPTSAS_TOPO_FLAG_RAID_ASSOCIATED) {
5829                        /*
5830                         * Get latest RAID info.
5831                         */
5832                        (void) mptsas_get_raid_info(mpt);
5833                        ptgt = mptsas_search_by_devhdl(
5834                            &mpt->m_active->m_tgttbl, topo_node->devhdl);
5835                        if (ptgt == NULL)
5836                                break;
5837                } else {
5838                        ptgt = (void *)topo_node->object;
5839                }

5841                if (ptgt == NULL) {
5842                        /*
5843                         * If a Phys Disk was deleted, RAID info needs to be
5844                         * updated to reflect the new topology.
5845                         */
5846                        (void) mptsas_get_raid_info(mpt);

5848                        /*
5849                         * Get sas device page 0 by DevHandle to make sure if
5850                         * SSP/SATA end device exist.
5851                         */
5852                        page_address = (MPI2_SAS_DEVICE_PGAD_FORM_HANDLE &
5853                            MPI2_SAS_DEVICE_PGAD_FORM_MASK) |
5854                            topo_node->devhdl;

5856                        rval = mptsas_get_target_device_info(mpt, page_address,
5857                            &devhdl, &ptgt);
5858                        if (rval == DEV_INFO_WRONG_DEVICE_TYPE) {
5859                                mptsas_log(mpt, CE_NOTE,
5860                                    "mptsas_handle_topo_change: target %d is "
5861                                    "not a SAS/SATA device. \n",
5862                                    topo_node->devhdl);
5863                        } else if (rval == DEV_INFO_FAIL_ALLOC) {
5864                                mptsas_log(mpt, CE_NOTE,
5865                                    "mptsas_handle_topo_change: could not "
5866                                    "allocate memory. \n");
5867                        }
5868                        /*
5869                         * If rval is DEV_INFO_PHYS_DISK than there is nothing
5870                         * else to do, just leave.
5871                         */
5872                        if (rval != DEV_INFO_SUCCESS) {
5873                                return;
5874                        }
5875                }

5877                ASSERT(ptgt->m_devhdl == topo_node->devhdl);

5879                mutex_exit(&mpt->m_mutex);
5880                flags = topo_node->flags;

5882                if (flags == MPTSAS_TOPO_FLAG_RAID_PHYSDRV_ASSOCIATED) {
5883                        phymask = ptgt->m_phymask;
5884                        phy_mask_name = kmem_zalloc(MPTSAS_MAX_PHYS, KM_SLEEP);
5885                        (void) sprintf(phy_mask_name, "%x", phymask);
5886                        parent = scsi_hba_iport_find(mpt->m_dip,
5887                            phy_mask_name);
5888                        kmem_free(phy_mask_name, MPTSAS_MAX_PHYS);
5889                        if (parent == NULL) {
```



```
5890                                mptsas_log(mpt, CE_WARN, "Failed to find a "
5891                                    "iport for PD, should not happen!");
5892                                mutex_enter(&mpt->m_mutex);
5893                                break;
5894                        }
5895                }

5897                if (flags == MPTSAS_TOPO_FLAG_RAID_ASSOCIATED) {
5898                        ndi_devi_enter(parent, &circ1);
5899                        (void) mptsas_config_raid(parent, topo_node->devhdl,
5900                            &lundip);
5901                        ndi_devi_exit(parent, circ1);
5902                } else {
5903                        /*
5904                         * hold nexus for bus configure
5905                         */
5906                        ndi_devi_enter(scsi_vhci_dip, &circ);
5907                        ndi_devi_enter(parent, &circ1);
5908                        rval = mptsas_config_target(parent, ptgt);
5909                        /*
5910                         * release nexus for bus configure
5911                         */
5912                        ndi_devi_exit(parent, circ1);
5913                        ndi_devi_exit(scsi_vhci_dip, circ);

5915                        /*
5916                         * Add parent's props for SMHBA support
5917                         */
5918                        if (flags == MPTSAS_TOPO_FLAG_DIRECT_ATTACHED_DEVICE) {
5919                                bzero(attached_wwnstr,
5920                                    sizeof (attached_wwnstr));
5921                                (void) sprintf(attached_wwnstr, "w%016"PRIx64,
5922                                    ptgt->m_sas_wwn);
5923                                if (ddi_prop_update_string(DDI_DEV_T_NONE,
5924                                    parent,
5925                                    SCSI_ADDR_PROP_ATTACHED_PORT,
5926                                    attached_wwnstr)
5927                                    != DDI_PROP_SUCCESS) {
5928                                        (void) ddi_prop_remove(DDI_DEV_T_NONE,
5929                                            parent,
5930                                            SCSI_ADDR_PROP_ATTACHED_PORT);
5931                                        mptsas_log(mpt, CE_WARN, "Failed to"
5932                                            "attached-port props");
5933                                        return;
5934                                }
5935                                if (ddi_prop_update_int(DDI_DEV_T_NONE, parent,
5936                                    MPTSAS_NUM_PHYS, 1) !=
5937                                    DDI_PROP_SUCCESS) {
5938                                        (void) ddi_prop_remove(DDI_DEV_T_NONE,
5939                                            parent, MPTSAS_NUM_PHYS);
5940                                        mptsas_log(mpt, CE_WARN, "Failed to"
5941                                            " create num-phys props");
5942                                        return;
5943                                }

5945                                /*
5946                                 * Update PHY info for smhba
5947                                 */
5948                                mutex_enter(&mpt->m_mutex);
5949                                if (mptsas_smhba_phy_init(mpt)) {
5950                                        mutex_exit(&mpt->m_mutex);
5951                                        mptsas_log(mpt, CE_WARN, "mptsas phy"
5952                                            " update failed");
5953                                        return;
5954                                }
5955                                mutex_exit(&mpt->m_mutex);
```

```
                                /*
                                 * topo_node->un.physport is really the PHY#
                                 * for direct attached devices
                                 */
                                mptsas_smhba_set_one_phy_props(mpt, parent,
                                    topo_node->un.physport, &attached_devhdl);

                                mptsas_smhba_set_phy_props(mpt,
                                    ddi_get_name_addr(parent), parent,
                                    1, &attached_devhdl);
                                if (ddi_prop_update_int(DDI_DEV_T_NONE, parent,
                                    MPTSAS_VIRTUAL_PORT, 0) !=
                                    DDI_PROP_SUCCESS) {
                                        (void) ddi_prop_remove(DDI_DEV_T_NONE,
                                            parent, MPTSAS_VIRTUAL_PORT);
                                        mptsas_log(mpt, CE_WARN,
                                            "mptsas virtual-port"
                                            "port prop update failed");
                                        return;
                                }
                        }
                }
                mutex_enter(&mpt->m_mutex);

                NDBG20(("mptsas%d handle_topo_change to online devhdl:%x, "
                    "phymask:%x.", mpt->m_instance, ptgt->m_devhdl,
                    ptgt->m_phymask));
                break;
        }
        case MPTSAS_DR_EVENT_OFFLINE_TARGET:
        {
                mptsas_hash_table_t *tgttbl = &mpt->m_active->m_tgttbl;
                devhdl = topo_node->devhdl;
                ptgt = mptsas_search_by_devhdl(tgttbl, devhdl);
                if (ptgt == NULL)
                        break;

                sas_wwn = ptgt->m_sas_wwn;
                phy = ptgt->m_phynum;

                addr = kmem_zalloc(SCSI_MAXNAMELEN, KM_SLEEP);

                if (sas_wwn) {
                        (void) sprintf(addr, "w%016"PRIx64, sas_wwn);
                } else {
                        (void) sprintf(addr, "p%x", phy);
                }
                ASSERT(ptgt->m_devhdl == devhdl);

                if ((topo_node->flags == MPTSAS_TOPO_FLAG_RAID_ASSOCIATED) ||
                    (topo_node->flags ==
                    MPTSAS_TOPO_FLAG_RAID_PHYSDRV_ASSOCIATED)) {
                        /*
                         * Get latest RAID info if RAID volume status changes
                         * or Phys Disk status changes
                         */
                        (void) mptsas_get_raid_info(mpt);
                }
                /*
                 * Abort all outstanding command on the device
                 */
                rval = mptsas_do_scsi_reset(mpt, devhdl);
                if (rval) {
                        NDBG20(("mptsas%d handle_topo_change to reset target "
                            "before offline devhdl:%x, phymask:%x, rval:%x",
```



```
                            mpt->m_instance, ptgt->m_devhdl, ptgt->m_phymask,
                            rval));
                }

                mutex_exit(&mpt->m_mutex);

                ndi_devi_enter(scsi_vhci_dip, &circ);
                ndi_devi_enter(parent, &circ1);
                rval = mptsas_offline_target(parent, addr);
                ndi_devi_exit(parent, circ1);
                ndi_devi_exit(scsi_vhci_dip, circ);
                NDBG20(("mptsas%d handle_topo_change to offline devhdl:%x, "
                    "phymask:%x, rval:%x", mpt->m_instance,
                    ptgt->m_devhdl, ptgt->m_phymask, rval));

                kmem_free(addr, SCSI_MAXNAMELEN);

                /*
                 * Clear parent's props for SMHBA support
                 */
                flags = topo_node->flags;
                if (flags == MPTSAS_TOPO_FLAG_DIRECT_ATTACHED_DEVICE) {
                        bzero(attached_wwnstr, sizeof (attached_wwnstr));
                        if (ddi_prop_update_string(DDI_DEV_T_NONE, parent,
                            SCSI_ADDR_PROP_ATTACHED_PORT, attached_wwnstr) !=
                            DDI_PROP_SUCCESS) {
                                (void) ddi_prop_remove(DDI_DEV_T_NONE, parent,
                                    SCSI_ADDR_PROP_ATTACHED_PORT);
                                mptsas_log(mpt, CE_WARN, "mptsas attached port "
                                    "prop update failed");
                                break;
                        }
                        if (ddi_prop_update_int(DDI_DEV_T_NONE, parent,
                            MPTSAS_NUM_PHYS, 0) !=
                            DDI_PROP_SUCCESS) {
                                (void) ddi_prop_remove(DDI_DEV_T_NONE, parent,
                                    MPTSAS_NUM_PHYS);
                                mptsas_log(mpt, CE_WARN, "mptsas num phys "
                                    "prop update failed");
                                break;
                        }
                        if (ddi_prop_update_int(DDI_DEV_T_NONE, parent,
                            MPTSAS_VIRTUAL_PORT, 1) !=
                            DDI_PROP_SUCCESS) {
                                (void) ddi_prop_remove(DDI_DEV_T_NONE, parent,
                                    MPTSAS_VIRTUAL_PORT);
                                mptsas_log(mpt, CE_WARN, "mptsas virtual port "
                                    "prop update failed");
                                break;
                        }
                }

                mutex_enter(&mpt->m_mutex);
                ptgt->m_led_status = 0;
                (void) mptsas_flush_led_status(mpt, ptgt);
                if (rval == DDI_SUCCESS) {
                        mptsas_tgt_free(&mpt->m_active->m_tgttbl,
                            ptgt->m_sas_wwn, ptgt->m_phymask);
                        ptgt = NULL;
                } else {
                        /*
                         * clean DR_INTRANSITION flag to allow I/O down to
                         * PHCI driver since failover finished.
                         * Invalidate the devhdl
                         */
                        ptgt->m_devhdl = MPTSAS_INVALID_DEVHDL;
```

```
6085                        ptgt->m_tgt_unconfigured = 0;
6086                        mutex_enter(&mpt->m_tx_waitq_mutex);
6087                        ptgt->m_dr_flag = MPTSAS_DR_INACTIVE;
6088                        mutex_exit(&mpt->m_tx_waitq_mutex);
6089                }

6091                /*
6092                 * Send SAS IO Unit Control to free the dev handle
6093                 */
6094                if ((flags == MPTSAS_TOPO_FLAG_DIRECT_ATTACHED_DEVICE) ||
6095                    (flags == MPTSAS_TOPO_FLAG_EXPANDER_ATTACHED_DEVICE)) {
6096                        rval = mptsas_free_devhdl(mpt, devhdl);

6098                        NDBG20(("mptsas%d handle_topo_change to remove "
6099                            "devhdl:%x, rval:%x", mpt->m_instance, devhdl,
6100                            rval));
6101                }

6103                break;
6104        }
6105        case MPTSAS_TOPO_FLAG_REMOVE_HANDLE:
6106        {
6107                devhdl = topo_node->devhdl;
6108                /*
6109                 * If this is the remove handle event, do a reset first.
6110                 */
6111                if (topo_node->event == MPTSAS_TOPO_FLAG_REMOVE_HANDLE) {
6112                        rval = mptsas_do_scsi_reset(mpt, devhdl);
6113                        if (rval) {
6114                                NDBG20(("mpt%d reset target before remove "
6115                                    "devhdl:%x, rval:%x", mpt->m_instance,
6116                                    devhdl, rval));
6117                        }
6118                }

6120                /*
6121                 * Send SAS IO Unit Control to free the dev handle
6122                 */
6123                rval = mptsas_free_devhdl(mpt, devhdl);
6124                NDBG20(("mptsas%d handle_topo_change to remove "
6125                    "devhdl:%x, rval:%x", mpt->m_instance, devhdl,
6126                    rval));
6127                break;
6128        }
6129        case MPTSAS_DR_EVENT_RECONFIG_SMP:
6130        {
6131                mptsas_smp_t smp;
6132                dev_info_t *smpdip;
6133                mptsas_hash_table_t *smptbl = &mpt->m_active->m_smptbl;

6135                devhdl = topo_node->devhdl;

6137                page_address = (MPI2_SAS_EXPAND_PGAD_FORM_HNDL &
6138                    MPI2_SAS_EXPAND_PGAD_FORM_MASK) | (uint32_t)devhdl;
6139                rval = mptsas_get_sas_expander_page0(mpt, page_address, &smp);
6140                if (rval != DDI_SUCCESS) {
6141                        mptsas_log(mpt, CE_WARN, "failed to online smp, "
6142                            "handle %x", devhdl);
6143                        return;
6144                }

6146                psmp = mptsas_smp_alloc(smptbl, &smp);
6147                if (psmp == NULL) {
6148                        return;
6149                }
```



```
6151                mutex_exit(&mpt->m_mutex);
6152                ndi_devi_enter(parent, &circ1);
6153                (void) mptsas_online_smp(parent, psmp, &smpdip);
6154                ndi_devi_exit(parent, circ1);

6156                mutex_enter(&mpt->m_mutex);
6157                break;
6158        }
6159        case MPTSAS_DR_EVENT_OFFLINE_SMP:
6160        {
6161                mptsas_hash_table_t *smptbl = &mpt->m_active->m_smptbl;
6162                devhdl = topo_node->devhdl;
6163                uint32_t dev_info;

6165                psmp = mptsas_search_by_devhdl(smptbl, devhdl);
6166                if (psmp == NULL)
6167                        break;
6168                /*
6169                 * The mptsas_smp_t data is released only if the dip is offlined
6170                 * successfully.
6171                 */
6172                mutex_exit(&mpt->m_mutex);

6174                ndi_devi_enter(parent, &circ1);
6175                rval = mptsas_offline_smp(parent, psmp, NDI_DEVI_REMOVE);
6176                ndi_devi_exit(parent, circ1);

6178                dev_info = psmp->m_deviceinfo;
6179                if ((dev_info & DEVINFO_DIRECT_ATTACHED) ==
6180                    DEVINFO_DIRECT_ATTACHED) {
6181                        if (ddi_prop_update_int(DDI_DEV_T_NONE, parent,
6182                            MPTSAS_VIRTUAL_PORT, 1) !=
6183                            DDI_PROP_SUCCESS) {
6184                                (void) ddi_prop_remove(DDI_DEV_T_NONE, parent,
6185                                    MPTSAS_VIRTUAL_PORT);
6186                                mptsas_log(mpt, CE_WARN, "mptsas virtual port "
6187                                    "prop update failed");
6188                                return;
6189                        }
6190                        /*
6191                         * Check whether the smp connected to the iport,
6192                         */
6193                        if (ddi_prop_update_int(DDI_DEV_T_NONE, parent,
6194                            MPTSAS_NUM_PHYS, 0) !=
6195                            DDI_PROP_SUCCESS) {
6196                                (void) ddi_prop_remove(DDI_DEV_T_NONE, parent,
6197                                    MPTSAS_NUM_PHYS);
6198                                mptsas_log(mpt, CE_WARN, "mptsas num phys"
6199                                    "prop update failed");
6200                                return;
6201                        }
6202                        /*
6203                         * Clear parent's attached-port props
6204                         */
6205                        bzero(attached_wwnstr, sizeof (attached_wwnstr));
6206                        if (ddi_prop_update_string(DDI_DEV_T_NONE, parent,
6207                            SCSI_ADDR_PROP_ATTACHED_PORT, attached_wwnstr) !=
6208                            DDI_PROP_SUCCESS) {
6209                                (void) ddi_prop_remove(DDI_DEV_T_NONE, parent,
6210                                    SCSI_ADDR_PROP_ATTACHED_PORT);
6211                                mptsas_log(mpt, CE_WARN, "mptsas attached port "
6212                                    "prop update failed");
6213                                return;
6214                        }
6215                }
```

```
6217                 mutex_enter(&mpt->m_mutex);
6218                 NDBG20(("mptsas%d handle_topo_change to remove devhdl:%x, "
6219                     "rval:%x", mpt->m_instance, psmp->m_devhdl, rval));
6220                 if (rval == DDI_SUCCESS) {
6221                         mptsas_smp_free(smptbl, psmp->m_sasaddr,
6222                             psmp->m_phymask);
6223                 } else {
6224                         psmp->m_devhdl = MPTSAS_INVALID_DEVHDL;
6225                 }

6227                 bzero(attached_wwnstr, sizeof (attached_wwnstr));

6229                 break;
6230         }
6231         default:
6232                 return;
6233         }
6234 }
```

*unchanged_portion_omitted_*

```
14704 static int
14705 mptsas_online_smp(dev_info_t *pdip, mptsas_smp_t *smp_node,
14706     dev_info_t **smp_dip)
14707 {
14708         char            wwn_str[MPTSAS_WWN_STRLEN];
14709         char            attached_wwn_str[MPTSAS_WWN_STRLEN];
14710         int             ndi_rtn = NDI_FAILURE;
14711         int             rval = 0;
14712         mptsas_smp_t    dev_info;
14713         uint32_t        page_address;
14714         mptsas_t        *mpt = DIP2MPT(pdip);
14715         uint16_t        dev_hdl;
14716         uint64_t        sas_wwn;
14717         uint64_t        smp_sas_wwn;
14718         uint8_t         physport;
14719         uint8_t         phy_id;
14720         uint16_t        pdev_hdl;
14721         uint8_t         numphys = 0;
14722         uint16_t        i = 0;
14723         char            phymask[MPTSAS_MAX_PHYS];
14724         char            *iport = NULL;
14725         mptsas_phymask_t        phy_mask = 0;
14726         uint16_t        attached_devhdl;
14727         uint16_t        bay_num, enclosure;

14729         (void) sprintf(wwn_str, "%"PRIx64, smp_node->m_sasaddr);

14731         /*
14732          * Probe smp device, prevent the node of removed device from being
14733          * configured succesfully
14734          */
14735         if (mptsas_probe_smp(pdip, smp_node->m_sasaddr) != NDI_SUCCESS) {
14736                 return (DDI_FAILURE);
14737         }

14739         if ((*smp_dip = mptsas_find_smp_child(pdip, wwn_str)) != NULL) {
14740                 return (DDI_SUCCESS);
14741         }

14743         ndi_rtn = ndi_devi_alloc(pdip, "smp", DEVI_SID_NODEID, smp_dip);

14745         /*
14746          * if lun alloc success, set props
14747          */
14748         if (ndi_rtn == NDI_SUCCESS) {
14749                 /*
```



```
14750                  * Set the flavor of the child to be SMP flavored
14751                  */
14752                 ndi_flavor_set(*smp_dip, SCSA_FLAVOR_SMP);

14754                 if (ndi_prop_update_string(DDI_DEV_T_NONE,
14755                     *smp_dip, SMP_WWN, wwn_str) !=
14756                     DDI_PROP_SUCCESS) {
14757                         mptsas_log(mpt, CE_WARN, "mptsas unable to create "
14758                             "property for smp device %s (sas_wwn)",
14759                             wwn_str);
14760                         ndi_rtn = NDI_FAILURE;
14761                         goto smp_create_done;
14762                 }
14763                 (void) sprintf(wwn_str, "w%"PRIx64, smp_node->m_sasaddr);
14764                 if (ndi_prop_update_string(DDI_DEV_T_NONE,
14765                     *smp_dip, SCSI_ADDR_PROP_TARGET_PORT, wwn_str) !=
14766                     DDI_PROP_SUCCESS) {
14767                         mptsas_log(mpt, CE_WARN, "mptsas unable to create "
14768                             "property for iport target-port %s (sas_wwn)",
14769                             wwn_str);
14770                         ndi_rtn = NDI_FAILURE;
14771                         goto smp_create_done;
14772                 }

14774                 mutex_enter(&mpt->m_mutex);

14776                 page_address = (MPI2_SAS_EXPAND_PGAD_FORM_HNDL &
14777                     MPI2_SAS_EXPAND_PGAD_FORM_MASK) | smp_node->m_devhdl;
14778                 rval = mptsas_get_sas_expander_page0(mpt, page_address,
14779                     &dev_info);
14780                 if (rval != DDI_SUCCESS) {
14781                         mutex_exit(&mpt->m_mutex);
14782                         mptsas_log(mpt, CE_WARN,
14783                             "mptsas unable to get expander "
14784                             "parent device info for %x", page_address);
14785                         ndi_rtn = NDI_FAILURE;
14786                         goto smp_create_done;
14787                 }

14789                 smp_node->m_pdevhdl = dev_info.m_pdevhdl;
14790                 page_address = (MPI2_SAS_DEVICE_PGAD_FORM_HANDLE &
14791                     MPI2_SAS_DEVICE_PGAD_FORM_MASK) |
14792                     (uint32_t)dev_info.m_pdevhdl;
14793                 rval = mptsas_get_sas_device_page0(mpt, page_address,
14794                     &dev_hdl, &sas_wwn, &smp_node->m_pdevinfo,
14795                     &physport, &phy_id, &pdev_hdl, &bay_num, &enclosure);
14796                 if (rval != DDI_SUCCESS) {
14797                         mutex_exit(&mpt->m_mutex);
14798                         mptsas_log(mpt, CE_WARN, "mptsas unable to get "
14799                             "device info for %x", page_address);
14800                         ndi_rtn = NDI_FAILURE;
14801                         goto smp_create_done;
14802                 }

14804                 page_address = (MPI2_SAS_DEVICE_PGAD_FORM_HANDLE &
14805                     MPI2_SAS_DEVICE_PGAD_FORM_MASK) |
14806                     (uint32_t)dev_info.m_devhdl;
14807                 rval = mptsas_get_sas_device_page0(mpt, page_address,
14808                     &dev_hdl, &smp_sas_wwn, &smp_node->m_deviceinfo,
14809                     &physport, &phy_id, &pdev_hdl, &bay_num, &enclosure);
14810                 if (rval != DDI_SUCCESS) {
14811                         mutex_exit(&mpt->m_mutex);
14812                         mptsas_log(mpt, CE_WARN, "mptsas unable to get "
14813                             "device info for %x", page_address);
14814                         ndi_rtn = NDI_FAILURE;
14815                         goto smp_create_done;
```

```
14816                   }
14817                   mutex_exit(&mpt->m_mutex);

14819                   /*
14820                    * If this smp direct attached to the controller
14821                    * set the attached-port to the base wwid
14822                    */
14823                   if ((smp_node->m_deviceinfo & DEVINFO_DIRECT_ATTACHED)
14824                       != DEVINFO_DIRECT_ATTACHED) {
14825                           (void) sprintf(attached_wwn_str, "w%016"PRIx64,
14826                               sas_wwn);
14827                   } else {
14828                           (void) sprintf(attached_wwn_str, "w%016"PRIx64,
14829                               mpt->un.m_base_wwid);
14830                   }

14832                   if (ndi_prop_update_string(DDI_DEV_T_NONE,
14833                       *smp_dip, SCSI_ADDR_PROP_ATTACHED_PORT, attached_wwn_str) !=
14834                       DDI_PROP_SUCCESS) {
14835                           mptsas_log(mpt, CE_WARN, "mptsas unable to create "
14836                               "property for smp attached-port %s (sas_wwn)",
14837                               attached_wwn_str);
14838                           ndi_rtn = NDI_FAILURE;
14839                           goto smp_create_done;
14840                   }

14842                   if (ndi_prop_create_boolean(DDI_DEV_T_NONE,
14843                       *smp_dip, SMP_PROP) != DDI_PROP_SUCCESS) {
14844                           mptsas_log(mpt, CE_WARN, "mptsas unable to "
14845                               "create property for SMP %s (SMP_PROP) ",
14846                               wwn_str);
14847                           ndi_rtn = NDI_FAILURE;
14848                           goto smp_create_done;
14849                   }

14851                   /*
14852                    * check the smp to see whether it direct
14853                    * attached to the controller
14854                    */
14855                   if ((smp_node->m_deviceinfo & DEVINFO_DIRECT_ATTACHED)
14856                       != DEVINFO_DIRECT_ATTACHED) {
14857                           goto smp_create_done;
14858                   }
14859                   numphys = ddi_prop_get_int(DDI_DEV_T_ANY, pdip,
14860                       DDI_PROP_DONTPASS, MPTSAS_NUM_PHYS, -1);
14861                   if (numphys > 0) {
14862                           goto smp_create_done;
14863                   }
14864                   /*
14865                    * this iport is an old iport, we need to
14866                    * reconfig the props for it.
14867                    */
14868                   if (ddi_prop_update_int(DDI_DEV_T_NONE, pdip,
14869                       MPTSAS_VIRTUAL_PORT, 0) !=
14870                       DDI_PROP_SUCCESS) {
14871                           (void) ddi_prop_remove(DDI_DEV_T_NONE, pdip,
14872                               MPTSAS_VIRTUAL_PORT);
14873                           mptsas_log(mpt, CE_WARN, "mptsas virtual port "
14874                               "prop update failed");
14875                           goto smp_create_done;
14876                   }

14878                   mutex_enter(&mpt->m_mutex);
14879                   numphys = 0;
14880                   iport = ddi_get_name_addr(pdip);
14881                   for (i = 0; i < MPTSAS_MAX_PHYS; i++) {
```



```
14882                           bzero(phymask, sizeof (phymask));
14883                           (void) sprintf(phymask,
14884                               "%x", mpt->m_phy_info[i].phy_mask);
14885                           if (strcmp(phymask, iport) == 0) {
14886                                   phy_mask = mpt->m_phy_info[i].phy_mask;
14887                                   break;
14888                           }
14889                   }

14891                   for (i = 0; i < MPTSAS_MAX_PHYS; i++) {
14892                           if ((phy_mask >> i) & 0x01) {
14893                                   numphys++;
14894                           }
14895                   }
14896                   /*
14897                    * Update PHY info for smhba
14898                    */
14899                   if (mptsas_smhba_phy_init(mpt)) {
14900                           mutex_exit(&mpt->m_mutex);
14901                           mptsas_log(mpt, CE_WARN, "mptsas phy update "
14902                               "failed");
14903                           goto smp_create_done;
14904                   }
14905                   mutex_exit(&mpt->m_mutex);

14907                   mptsas_smhba_set_all_phy_props(mpt, pdip, numphys, phy_mask,
14908                       &attached_devhdl);
14902                   mptsas_smhba_set_phy_props(mpt, iport, pdip,
14903                       numphys, &attached_devhdl);

14910                   if (ddi_prop_update_int(DDI_DEV_T_NONE, pdip,
14911                       MPTSAS_NUM_PHYS, numphys) !=
14912                       DDI_PROP_SUCCESS) {
14913                           (void) ddi_prop_remove(DDI_DEV_T_NONE, pdip,
14914                               MPTSAS_NUM_PHYS);
14915                           mptsas_log(mpt, CE_WARN, "mptsas update "
14916                               "num phys props failed");
14917                           goto smp_create_done;
14918                   }
14919                   /*
14920                    * Add parent's props for SMHBA support
14921                    */
14922                   if (ddi_prop_update_string(DDI_DEV_T_NONE, pdip,
14923                       SCSI_ADDR_PROP_ATTACHED_PORT, wwn_str) !=
14924                       DDI_PROP_SUCCESS) {
14925                           (void) ddi_prop_remove(DDI_DEV_T_NONE, pdip,
14926                               SCSI_ADDR_PROP_ATTACHED_PORT);
14927                           mptsas_log(mpt, CE_WARN, "mptsas update iport"
14928                               "attached-port failed");
14929                           goto smp_create_done;
14930                   }

14932 smp_create_done:
14933                   /*
14934                    * If props were setup ok, online the lun
14935                    */
14936                   if (ndi_rtn == NDI_SUCCESS) {
14937                           /*
14938                            * Try to online the new node
14939                            */
14940                           ndi_rtn = ndi_devi_online(*smp_dip, NDI_ONLINE_ATTACH);
14941                   }

14943                   /*
14944                    * If success set rtn flag, else unwire alloc'd lun
14945                    */
```

```
14946                 if (ndi_rtn != NDI_SUCCESS) {
14947                         NDBG12(("mptsas unable to online "
14948                             "SMP target %s", wwn_str));
14949                         ndi_prop_remove_all(*smp_dip);
14950                         (void) ndi_devi_free(*smp_dip);
14951                 }
14952         }

14954         return ((ndi_rtn == NDI_SUCCESS) ? DDI_SUCCESS : DDI_FAILURE);
14955 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   14573 Tue Oct 22 09:57:26 2013
new/usr/src/uts/common/io/scsi/adapters/mpt_sas/mptsas_smhba.c
4233 mptsas topo change buffer overflow
**********************************************************
```

```
   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  */
  26 /*
  27  * This file contains SM-HBA support for MPT SAS driver
  28  */

  30 #if defined(lint) || defined(DEBUG)
  31 #define MPTSAS_DEBUG
  32 #endif

  34 /*
  35  * standard header files
  36  */
  37 #include <sys/note.h>
  38 #include <sys/scsi/scsi.h>
  39 #include <sys/pci.h>
  40 #include <sys/scsi/generic/sas.h>
  41 #include <sys/scsi/impl/scsi_sas.h>

  43 #pragma pack(1)
  44 #include <sys/scsi/adapters/mpt_sas/mpi/mpi2_type.h>
  45 #include <sys/scsi/adapters/mpt_sas/mpi/mpi2.h>
  46 #include <sys/scsi/adapters/mpt_sas/mpi/mpi2_cnfg.h>
  47 #include <sys/scsi/adapters/mpt_sas/mpi/mpi2_init.h>
  48 #include <sys/scsi/adapters/mpt_sas/mpi/mpi2_ioc.h>
  49 #include <sys/scsi/adapters/mpt_sas/mpi/mpi2_sas.h>
  50 #pragma pack()

  52 /*
  53  * private header files.
  54  */
  55 #include <sys/scsi/adapters/mpt_sas/mptsas_var.h>
  56 #include <sys/scsi/adapters/mpt_sas/mptsas_smhba.h>

  58 /*
  59  * SM - HBA statics
  60  */
  61 extern char *mptsas_driver_rev;
```



```
  63 static void mptsas_smhba_create_phy_props(nvlist_t **, smhba_info_t *, uint8_t,
  64     uint16_t *);
  65 static void mptsas_smhba_update_phy_props(mptsas_t *, dev_info_t *, nvlist_t **,
  66     uint8_t);

  68 static void
  69 mptsas_smhba_add_hba_prop(mptsas_t *mpt, data_type_t dt,
  70     char *prop_name, void *prop_val);

  72 void
  73 mptsas_smhba_show_phy_info(mptsas_t *mpt);

  75 static void
  76 mptsas_smhba_add_hba_prop(mptsas_t *mpt, data_type_t dt,
  77     char *prop_name, void *prop_val)
  78 {
  79         ASSERT(mpt != NULL);

  81         switch (dt) {
  82         case DATA_TYPE_INT32:
  83                 if (ddi_prop_update_int(DDI_DEV_T_NONE, mpt->m_dip,
  84                     prop_name, *(int *)prop_val)) {
  85                         mptsas_log(mpt, CE_WARN,
  86                             "%s: %s prop update failed", __func__, prop_name);
  87                 }
  88                 break;
  89         case DATA_TYPE_STRING:
  90                 if (ddi_prop_update_string(DDI_DEV_T_NONE, mpt->m_dip,
  91                     prop_name, (char *)prop_val)) {
  92                         mptsas_log(mpt, CE_WARN,
  93                             "%s: %s prop update failed", __func__, prop_name);
  94                 }
  95                 break;
  96         default:
  97                 mptsas_log(mpt, CE_WARN, "%s: "
  98                     "Unhandled datatype(%d) for (%s). Skipping prop update.",
  99                     __func__, dt, prop_name);
 100         }
 101 }
_____unchanged_portion_omitted_

 125 static void
 126 mptsas_smhba_create_phy_props(nvlist_t **phy_props, smhba_info_t *pSmhba,
 127     uint8_t phy_id, uint16_t *attached_devhdl)
 119 void
 120 mptsas_smhba_set_phy_props(mptsas_t *mpt, char *iport, dev_info_t *dip,
 121     uint8_t phy_nums, uint16_t *attached_devhdl)
 128 {
 129         (void) nvlist_alloc(phy_props, NV_UNIQUE_NAME, KM_SLEEP);
 130         (void) nvlist_add_uint8(*phy_props, SAS_PHY_ID, phy_id);
 131         (void) nvlist_add_uint8(*phy_props, "phyState",
 132             (pSmhba->negotiated_link_rate & 0x0f));
 133         (void) nvlist_add_int8(*phy_props, SAS_NEG_LINK_RATE,
 134             (pSmhba->negotiated_link_rate & 0x0f));
 135         (void) nvlist_add_int8(*phy_props, SAS_PROG_MIN_LINK_RATE,
 136             (pSmhba->programmed_link_rate & 0x0f));
 137         (void) nvlist_add_int8(*phy_props, SAS_HW_MIN_LINK_RATE,
 138             (pSmhba->hw_link_rate & 0x0f));
 139         (void) nvlist_add_int8(*phy_props, SAS_PROG_MAX_LINK_RATE,
 140             ((pSmhba->programmed_link_rate & 0xf0) >> 4));
 141         (void) nvlist_add_int8(*phy_props, SAS_HW_MAX_LINK_RATE,
 142             ((pSmhba->hw_link_rate & 0xf0) >> 4));

 144         if (pSmhba->attached_devhdl && (attached_devhdl != NULL))
 145                 *attached_devhdl = pSmhba->attached_devhdl;
```

```
146 }

148 static void
149 mptsas_smhba_update_phy_props(mptsas_t *mpt, dev_info_t *dip,
150     nvlist_t **phy_props, uint8_t phy_nums)
151 {
123         int             i;
124         int             j = 0;
152         int             rval;
153         size_t          packed_size;
154         char            *packed_data = NULL;
128         char            phymask[MPTSAS_MAX_PHYS];
129         nvlist_t        **phy_props;
155         nvlist_t        *nvl;
131         smhba_info_t    *pSmhba = NULL;

157         if (nvlist_alloc(&nvl, NV_UNIQUE_NAME, KM_SLEEP) != 0) {
158                 mptsas_log(mpt, CE_WARN, "%s: nvlist_alloc() failed", __func__);
133         if (phy_nums == 0) {
159                 return;
160         }
136         if (nvlist_alloc(&nvl, NV_UNIQUE_NAME, 0) != 0) {
137                 mptsas_log(mpt, CE_WARN, "%s: nvlist_alloc() failed", __func__);
138         }

140         phy_props = kmem_zalloc(sizeof (nvlist_t *) * phy_nums,
141             KM_SLEEP);

143         for (i = 0; i < mpt->m_num_phys; i++) {

145                 bzero(phymask, sizeof (phymask));
146                 (void) sprintf(phymask, "%x", mpt->m_phy_info[i].phy_mask);
147                 if (strcmp(phymask, iport) == 0) {
148                         pSmhba = &mpt->m_phy_info[i].smhba_info;
149                         (void) nvlist_alloc(&phy_props[j], NV_UNIQUE_NAME, 0);
150                         (void) nvlist_add_uint8(phy_props[j], SAS_PHY_ID, i);
151                         (void) nvlist_add_uint8(phy_props[j],
152                             "phyState",
153                             (pSmhba->negotiated_link_rate
154                             & 0x0f));
155                         (void) nvlist_add_int8(phy_props[j],
156                             SAS_NEG_LINK_RATE,
157                             (pSmhba->negotiated_link_rate
158                             & 0x0f));
159                         (void) nvlist_add_int8(phy_props[j],
160                             SAS_PROG_MIN_LINK_RATE,
161                             (pSmhba->programmed_link_rate
162                             & 0x0f));
163                         (void) nvlist_add_int8(phy_props[j],
164                             SAS_HW_MIN_LINK_RATE,
165                             (pSmhba->hw_link_rate
166                             & 0x0f));
167                         (void) nvlist_add_int8(phy_props[j],
168                             SAS_PROG_MAX_LINK_RATE,
169                             ((pSmhba->programmed_link_rate
170                             & 0xf0) >> 4));
171                         (void) nvlist_add_int8(phy_props[j],
172                             SAS_HW_MAX_LINK_RATE,
173                             ((pSmhba->hw_link_rate
174                             & 0xf0) >> 4));

176                         j++;

178                         if (pSmhba->attached_devhdl &&
179                             (attached_devhdl != NULL)) {
180                                 *attached_devhdl =
```



```
181                                     pSmhba->attached_devhdl;
182                         }
183                 }
184         }

162         rval = nvlist_add_nvlist_array(nvl, SAS_PHY_INFO_NVL, phy_props,
163             phy_nums);
164         if (rval) {
165                 mptsas_log(mpt, CE_WARN,
166                     " nv list array add failed, return value %d.", rval);
190                     " nv list array add failed, return value %d.",
191                     rval);
167                 goto exit;
168         }
169         (void) nvlist_size(nvl, &packed_size, NV_ENCODE_NATIVE);
170         packed_data = kmem_zalloc(packed_size, KM_SLEEP);
171         (void) nvlist_pack(nvl, &packed_data, &packed_size,
172             NV_ENCODE_NATIVE, 0);

174         (void) ddi_prop_update_byte_array(DDI_DEV_T_NONE, dip,
175             SAS_PHY_INFO, (uchar_t *)packed_data, packed_size);

177 exit:
203         for (i = 0; i < phy_nums && phy_props[i] != NULL; i++) {
204                 nvlist_free(phy_props[i]);
205         }
178         nvlist_free(nvl);
207         kmem_free(phy_props, sizeof (nvlist_t *) * phy_nums);

180         if (packed_data != NULL) {
181                 kmem_free(packed_data, packed_size);
182         }
183 }

185 void
186 mptsas_smhba_set_one_phy_props(mptsas_t *mpt, dev_info_t *dip, uint8_t phy_id,
187     uint16_t *attached_devhdl)
188 {
189         nvlist_t        *phy_props;

191         ASSERT(phy_id < mpt->m_num_phys);

193         mptsas_smhba_create_phy_props(&phy_props,
194             &mpt->m_phy_info[phy_id].smhba_info, phy_id, attached_devhdl);

196         mptsas_smhba_update_phy_props(mpt, dip, &phy_props, 1);

198         nvlist_free(phy_props);
199 }

201 void
202 mptsas_smhba_set_all_phy_props(mptsas_t *mpt, dev_info_t *dip, uint8_t phy_nums,
203     mptsas_phymask_t phy_mask, uint16_t *attached_devhdl)
204 {
205         int             i, j;
206         nvlist_t        **phy_props;

208         if (phy_nums == 0)
209                 return;

211         phy_props = kmem_zalloc(sizeof (nvlist_t *) * phy_nums, KM_SLEEP);

213         for (i = 0, j = 0; i < mpt->m_num_phys && j < phy_nums; i++)
214                 if (phy_mask == mpt->m_phy_info[i].phy_mask)
215                         mptsas_smhba_create_phy_props(&phy_props[j++],
216                             &mpt->m_phy_info[i].smhba_info, i, attached_devhdl);
```

```
218         mptsas_smhba_update_phy_props(mpt, dip, phy_props, j);

220         for (i = 0; i < j && phy_props[i] != NULL; i++)
221                 nvlist_free(phy_props[i]);

223         kmem_free(phy_props, sizeof (nvlist_t *) * phy_nums);
224 }

226 /*
227  * Called with PHY lock held on phyp
228  */
229 void
230 mptsas_smhba_log_sysevent(mptsas_t *mpt, char *subclass, char *etype,
231     smhba_info_t *phyp)
232 {
233         nvlist_t        *attr_list;
234         char            *pname;
235         char            sas_addr[MPTSAS_WWN_STRLEN];
236         uint8_t         phynum = 0;
237         uint8_t         lrate = 0;

239         if (mpt->m_dip == NULL)
240                 return;
241         if (phyp == NULL)
242                 return;

244         pname = kmem_zalloc(MAXPATHLEN, KM_NOSLEEP);
245         if (pname == NULL)
246                 return;

248         if ((strcmp(subclass, ESC_SAS_PHY_EVENT) == 0) ||
249             (strcmp(subclass, ESC_SAS_HBA_PORT_BROADCAST) == 0)) {
250                 ASSERT(phyp != NULL);
251                 (void) strncpy(pname, phyp->path, strlen(phyp->path));
252                 phynum = phyp->phy_id;
253                 bzero(sas_addr, sizeof (sas_addr));
254                 (void) sprintf(sas_addr, "w%016"PRIx64, phyp->sas_addr);
255                 if (strcmp(etype, SAS_PHY_ONLINE) == 0) {
256                         lrate = phyp->negotiated_link_rate;
257                 }
258         }
259         if (strcmp(subclass, ESC_SAS_HBA_PORT_BROADCAST) == 0) {
260                 (void) ddi_pathname(mpt->m_dip, pname);
261         }

263         if (nvlist_alloc(&attr_list, NV_UNIQUE_NAME_TYPE, 0) != 0) {
264                 mptsas_log(mpt, CE_WARN,
265                     "%s: Failed to post sysevent", __func__);
266                 kmem_free(pname, MAXPATHLEN);
267                 return;
268         }

270         if (nvlist_add_int32(attr_list, SAS_DRV_INST,
271             ddi_get_instance(mpt->m_dip)) != 0)
272                 goto fail;

274         if (nvlist_add_string(attr_list, SAS_PORT_ADDR, sas_addr) != 0)
275                 goto fail;

277         if (nvlist_add_string(attr_list, SAS_DEVFS_PATH, pname) != 0)
278                 goto fail;

280         if (nvlist_add_uint8(attr_list, SAS_PHY_ID, phynum) != 0)
281                 goto fail;
```



```
283         if (strcmp(etype, SAS_PHY_ONLINE) == 0) {
284                 if (nvlist_add_uint8(attr_list, SAS_LINK_RATE, lrate) != 0)
285                         goto fail;
286         }

288         if (nvlist_add_string(attr_list, SAS_EVENT_TYPE, etype) != 0)
289                 goto fail;

291         (void) ddi_log_sysevent(mpt->m_dip, DDI_VENDOR_SUNW, EC_HBA, subclass,
292             attr_list, NULL, DDI_NOSLEEP);

294 fail:
295         kmem_free(pname, MAXPATHLEN);
296         nvlist_free(attr_list);
297 }
```

*unchanged_portion_omitted_*

```
**********************************************************
    2889 Tue Oct 22 09:57:27 2013
new/usr/src/uts/common/sys/scsi/adapters/mpt_sas/mptsas_smhba.h
4233 mptsas topo change buffer overflow
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright (c) 2009, 2010, Oracle and/or its affiliates. All rights reserved.
 * Copyright 2013 Nexenta Systems, Inc. All rights reserved.
 */

/*
 * SM-HBA interfaces/definitions for MPT SAS driver.
 */

#ifndef _MPTSAS_SMHBA_H
#define _MPTSAS_SMHBA_H
#ifdef  __cplusplus
extern "C" {
#endif

/* Leverage definition of data_type_t in nvpair.h */
#include <sys/nvpair.h>
#include <sys/scsi/adapters/mpt_sas/mptsas_var.h>

#define MPTSAS_NUM_PHYS         "num-phys"
#define MPTSAS_NUM_PHYS_HBA     "num-phys-hba"
#define MPTSAS_SMHBA_SUPPORTED  "sm-hba-supported"
#define MPTSAS_DRV_VERSION      "driver-version"
#define MPTSAS_HWARE_VERSION    "hardware-version"
#define MPTSAS_FWARE_VERSION    "firmware-version"
#define MPTSAS_SUPPORTED_PROTOCOL       "supported-protocol"
#define MPTSAS_VIRTUAL_PORT     "virtual-port"

#define MPTSAS_MANUFACTURER     "Manufacturer"
#define MPTSAS_SERIAL_NUMBER    "SerialNumber"
#define MPTSAS_MODEL_NAME       "ModelName"
#define MPTSAS_VARIANT          "variant"

#define IS_ATAPI_DEVICE(x)      ((x) & 0x2000)
#define IS_SATA_DEVICE(x)       ((x) & 0x80)
#define DEVINFO_DIRECT_ATTACHED 0x0800

/*
 * Interfaces to add properties required for SM-HBA
 *
```



```
 * _add_xxx_prop() interfaces add only 1 prop that is specified in the args.
 * _set_xxx_props() interfaces add more than 1 prop for a set of phys/devices.
 */
int mptsas_smhba_setup(mptsas_t *);
void mptsas_smhba_show_phy_info(mptsas_t *);
void mptsas_smhba_set_all_phy_props(mptsas_t *mpt, dev_info_t *dip,
    uint8_t phy_nums, mptsas_phymask_t phy_mask, uint16_t *attached_devhdl);
void mptsas_smhba_set_one_phy_props(mptsas_t *mpt, dev_info_t *dip,
    uint8_t phy_id, uint16_t *attached_devhdl);
void mptsas_smhba_set_phy_props(mptsas_t *mpt, char *iport, dev_info_t *dip,
    uint8_t phy_nums, uint16_t *attached_devhdl);
void mptsas_smhba_log_sysevent(mptsas_t *mpt, char *subclass, char *etype,
    smhba_info_t *phyp);
void
mptsas_create_phy_stats(mptsas_t *mpt, char *iport, dev_info_t *dip);
int mptsas_update_phy_stats(kstat_t *ks, int rw);
void mptsas_destroy_phy_stats(mptsas_t *mpt);
int mptsas_smhba_phy_init(mptsas_t *mpt);
int mptsas_smhba_phy_state_update(mptsas_t *mpt, uint8_t phy);
#ifdef  __cplusplus
}
_____unchanged_portion_omitted_
```