```
**********************************************************
  443570 Wed May 23 16:20:26 2012
new/usr/src/uts/common/io/scsi/adapters/mpt_sas/mptsas.c
2777 mpt_sas needs to try MUR reset at attach() time.
**********************************************************
```

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

/*
 * Copyright (c) 2009, 2010, Oracle and/or its affiliates. All rights reserved.
 * Copyright 2012 Nexenta Systems, Inc. All rights reserved.
 */

/*
 * Copyright (c) 2000 to 2010, LSI Corporation.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms of all code within
 * this file that is exclusively owned by LSI, with or without
 * modification, is permitted provided that, in addition to the CDDL 1.0
 * License requirements, the following conditions are met:
 *
 *    Neither the name of the author nor the names of its contributors may be
 *    used to endorse or promote products derived from this software without
 *    specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
 * FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
 * COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
 * BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
 * OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
 * AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
 * OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
 * DAMAGE.
 */

/*
 * mptsas - This is a driver based on LSI Logic's MPT2.0 interface.
 *
 */

#if defined(lint) || defined(DEBUG)
#define MPTSAS_DEBUG
#endif
```



```
/*
 * standard header files.
 */
#include <sys/note.h>
#include <sys/scsi/scsi.h>
#include <sys/pci.h>
#include <sys/file.h>
#include <sys/cpuvar.h>
#include <sys/policy.h>
#include <sys/sysevent.h>
#include <sys/sysevent/eventdefs.h>
#include <sys/sysevent/dr.h>
#include <sys/sata/sata_defs.h>
#include <sys/scsi/generic/sas.h>
#include <sys/scsi/impl/scsi_sas.h>

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

/*
 * private header files.
 *
 */
#include <sys/scsi/impl/scsi_reset_notify.h>
#include <sys/scsi/adapters/mpt_sas/mptsas_var.h>
#include <sys/scsi/adapters/mpt_sas/mptsas_ioctl.h>
#include <sys/scsi/adapters/mpt_sas/mptsas_smhba.h>

#include <sys/raidioctl.h>

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

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

/*
 * For anyone who would modify the code in mptsas_driver, it must be awared
 * that from snv_145 where CR6910752(mpt_sas driver performance can be
 * improved) is integrated, the per_instance mutex m_mutex is not hold
 * in the key IO code path, including mptsas_scsi_start(), mptsas_intr()
 * and all of the recursive functions called in them, so don't
 * make it for granted that all operations are sync/exclude correctly. Before
 * doing any modification in key code path, and even other code path such as
 * DR, watchsubr, ioctl, passthrough etc, make sure the elements modified have
 * no releationship to elements shown in the fastpath
 * (function mptsas_handle_io_fastpath()) in ISR and its recursive functions.
 * otherwise, you have to use the new introduced mutex to protect them.
 * As to how to do correctly, refer to the comments in mptsas_intr().
 */

/*
 * autoconfiguration data and routines.
```

```
 128  */
 129 static int mptsas_attach(dev_info_t *dip, ddi_attach_cmd_t cmd);
 130 static int mptsas_detach(dev_info_t *devi, ddi_detach_cmd_t cmd);
 131 static int mptsas_power(dev_info_t *dip, int component, int level);

 133 /*
 134  * cb_ops function
 135  */
 136 static int mptsas_ioctl(dev_t dev, int cmd, intptr_t data, int mode,
 137         cred_t *credp, int *rval);
 138 #ifdef __sparc
 139 static int mptsas_reset(dev_info_t *devi, ddi_reset_cmd_t cmd);
 140 #else  /* __sparc */
 141 static int mptsas_quiesce(dev_info_t *devi);
 142 #endif  /* __sparc */

 144 /*
 145  * Resource initilaization for hardware
 146  */
 147 static void mptsas_setup_cmd_reg(mptsas_t *mpt);
 148 static void mptsas_disable_bus_master(mptsas_t *mpt);
 149 static void mptsas_hba_fini(mptsas_t *mpt);
 150 static void mptsas_cfg_fini(mptsas_t *mptsas_blkp);
 151 static int mptsas_hba_setup(mptsas_t *mpt);
 152 static void mptsas_hba_teardown(mptsas_t *mpt);
 153 static int mptsas_config_space_init(mptsas_t *mpt);
 154 static void mptsas_config_space_fini(mptsas_t *mpt);
 155 static void mptsas_iport_register(mptsas_t *mpt);
 156 static int mptsas_smp_setup(mptsas_t *mpt);
 157 static void mptsas_smp_teardown(mptsas_t *mpt);
 158 static int mptsas_cache_create(mptsas_t *mpt);
 159 static void mptsas_cache_destroy(mptsas_t *mpt);
 160 static int mptsas_alloc_request_frames(mptsas_t *mpt);
 161 static int mptsas_alloc_reply_frames(mptsas_t *mpt);
 162 static int mptsas_alloc_free_queue(mptsas_t *mpt);
 163 static int mptsas_alloc_post_queue(mptsas_t *mpt);
 164 static void mptsas_alloc_reply_args(mptsas_t *mpt);
 165 static int mptsas_alloc_extra_sgl_frame(mptsas_t *mpt, mptsas_cmd_t *cmd);
 166 static void mptsas_free_extra_sgl_frame(mptsas_t *mpt, mptsas_cmd_t *cmd);
 167 static int mptsas_init_chip(mptsas_t *mpt, int first_time);

 169 /*
 170  * SCSA function prototypes
 171  */
 172 static int mptsas_scsi_start(struct scsi_address *ap, struct scsi_pkt *pkt);
 173 static int mptsas_scsi_reset(struct scsi_address *ap, int level);
 174 static int mptsas_scsi_abort(struct scsi_address *ap, struct scsi_pkt *pkt);
 175 static int mptsas_scsi_getcap(struct scsi_address *ap, char *cap, int tgtonly);
 176 static int mptsas_scsi_setcap(struct scsi_address *ap, char *cap, int value,
 177     int tgtonly);
 178 static void mptsas_scsi_dmafree(struct scsi_address *ap, struct scsi_pkt *pkt);
 179 static struct scsi_pkt *mptsas_scsi_init_pkt(struct scsi_address *ap,
 180     struct scsi_pkt *pkt, struct buf *bp, int cmdlen, int statuslen,
 181         int tgtlen, int flags, int (*callback)(), caddr_t arg);
 182 static void mptsas_scsi_sync_pkt(struct scsi_address *ap, struct scsi_pkt *pkt);
 183 static void mptsas_scsi_destroy_pkt(struct scsi_address *ap,
 184     struct scsi_pkt *pkt);
 185 static int mptsas_scsi_tgt_init(dev_info_t *hba_dip, dev_info_t *tgt_dip,
 186     scsi_hba_tran_t *hba_tran, struct scsi_device *sd);
 187 static void mptsas_scsi_tgt_free(dev_info_t *hba_dip, dev_info_t *tgt_dip,
 188     scsi_hba_tran_t *hba_tran, struct scsi_device *sd);
 189 static int mptsas_scsi_reset_notify(struct scsi_address *ap, int flag,
 190     void (*callback)(caddr_t), caddr_t arg);
 191 static int mptsas_get_name(struct scsi_device *sd, char *name, int len);
 192 static int mptsas_get_bus_addr(struct scsi_device *sd, char *name, int len);
 193 static int mptsas_scsi_quiesce(dev_info_t *dip);
```



```
 194 static int mptsas_scsi_unquiesce(dev_info_t *dip);
 195 static int mptsas_bus_config(dev_info_t *pdip, uint_t flags,
 196     ddi_bus_config_op_t op, void *arg, dev_info_t **childp);

 198 /*
 199  * SMP functions
 200  */
 201 static int mptsas_smp_start(struct smp_pkt *smp_pkt);

 203 /*
 204  * internal function prototypes.
 205  */
 206 static void mptsas_list_add(mptsas_t *mpt);
 207 static void mptsas_list_del(mptsas_t *mpt);

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

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

 215 static void mptsas_ncmds_checkdrain(void *arg);

 217 static int mptsas_prepare_pkt(mptsas_cmd_t *cmd);
 218 static int mptsas_accept_pkt(mptsas_t *mpt, mptsas_cmd_t *sp);

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

 226 static void mptsas_handle_qfull(mptsas_t *mpt, mptsas_cmd_t *cmd);
 227 static void mptsas_handle_event(void *args);
 228 static int mptsas_handle_event_sync(void *args);
 229 static void mptsas_handle_dr(void *args);
 230 static void mptsas_handle_topo_change(mptsas_topo_change_list_t *topo_node,
 231     dev_info_t *pdip);

 233 static void mptsas_restart_cmd(void *);

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

 241 static uint_t mptsas_intr(caddr_t arg1, caddr_t arg2);
 242 static void mptsas_process_intr(mptsas_t *mpt,
 243     pMpi2ReplyDescriptorsUnion_t reply_desc_union);
 244 static int mptsas_handle_io_fastpath(mptsas_t *mpt, uint16_t SMID);
 245 static void mptsas_handle_scsi_io_success(mptsas_t *mpt,
 246     pMpi2ReplyDescriptorsUnion_t reply_desc);
 247 static void mptsas_handle_address_reply(mptsas_t *mpt,
 248     pMpi2ReplyDescriptorsUnion_t reply_desc);
 249 static int mptsas_wait_intr(mptsas_t *mpt, int polltime);
 250 static void mptsas_sge_setup(mptsas_t *mpt, mptsas_cmd_t *cmd,
 251     uint32_t *control, pMpi2SCSIIORequest_t frame, ddi_acc_handle_t acc_hdl);

 253 static void mptsas_watch(void *arg);
 254 static void mptsas_watchsubr(mptsas_t *mpt);
 255 static void mptsas_cmd_timeout(mptsas_t *mpt, uint16_t devhdl);

 257 static void mptsas_start_passthru(mptsas_t *mpt, mptsas_cmd_t *cmd);
 258 static int mptsas_do_passthru(mptsas_t *mpt, uint8_t *request, uint8_t *reply,
 259     uint8_t *data, uint32_t request_size, uint32_t reply_size,
```

```
260     uint32_t data_size, uint32_t direction, uint8_t *dataout,
261     uint32_t dataout_size, short timeout, int mode);
262 static int mptsas_free_devhdl(mptsas_t *mpt, uint16_t devhdl);

264 static uint8_t mptsas_get_fw_diag_buffer_number(mptsas_t *mpt,
265     uint32_t unique_id);
266 static void mptsas_start_diag(mptsas_t *mpt, mptsas_cmd_t *cmd);
267 static int mptsas_post_fw_diag_buffer(mptsas_t *mpt,
268     mptsas_fw_diagnostic_buffer_t *pBuffer, uint32_t *return_code);
269 static int mptsas_release_fw_diag_buffer(mptsas_t *mpt,
270     mptsas_fw_diagnostic_buffer_t *pBuffer, uint32_t *return_code,
271     uint32_t diag_type);
272 static int mptsas_diag_register(mptsas_t *mpt,
273     mptsas_fw_diag_register_t *diag_register, uint32_t *return_code);
274 static int mptsas_diag_unregister(mptsas_t *mpt,
275     mptsas_fw_diag_unregister_t *diag_unregister, uint32_t *return_code);
276 static int mptsas_diag_query(mptsas_t *mpt, mptsas_fw_diag_query_t *diag_query,
277     uint32_t *return_code);
278 static int mptsas_diag_read_buffer(mptsas_t *mpt,
279     mptsas_diag_read_buffer_t *diag_read_buffer, uint8_t *ioctl_buf,
280     uint32_t *return_code, int ioctl_mode);
281 static int mptsas_diag_release(mptsas_t *mpt,
282     mptsas_fw_diag_release_t *diag_release, uint32_t *return_code);
283 static int mptsas_do_diag_action(mptsas_t *mpt, uint32_t action,
284     uint8_t *diag_action, uint32_t length, uint32_t *return_code,
285     int ioctl_mode);
286 static int mptsas_diag_action(mptsas_t *mpt, mptsas_diag_action_t *data,
287     int mode);

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

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

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

300 static void mptsas_check_scsi_io_error(mptsas_t *mpt, pMpi2SCSIIOReply_t reply,
301     mptsas_cmd_t *cmd);
302 static void mptsas_check_task_mgt(mptsas_t *mpt,
303     pMpi2SCSIManagementReply_t reply, mptsas_cmd_t *cmd);
304 static int mptsas_send_scsi_cmd(mptsas_t *mpt, struct scsi_address *ap,
305     mptsas_target_t *ptgt, uchar_t *cdb, int cdblen, struct buf *data_bp,
306     int *resid);

308 static int mptsas_alloc_active_slots(mptsas_t *mpt, int flag);
309 static void mptsas_free_active_slots(mptsas_t *mpt);
310 static int mptsas_start_cmd(mptsas_t *mpt, mptsas_cmd_t *cmd);
311 static int mptsas_start_cmd0(mptsas_t *mpt, mptsas_cmd_t *cmd);

313 static void mptsas_restart_hba(mptsas_t *mpt);

315 static void mptsas_deliver_doneq_thread(mptsas_t *mpt);
316 static void mptsas_doneq_add(mptsas_t *mpt, mptsas_cmd_t *cmd);
317 static inline void mptsas_doneq_add0(mptsas_t *mpt, mptsas_cmd_t *cmd);
318 static void mptsas_doneq_mv(mptsas_t *mpt, uint64_t t);

320 static mptsas_cmd_t *mptsas_doneq_thread_rm(mptsas_t *mpt, uint64_t t);
321 static void mptsas_doneq_empty(mptsas_t *mpt);
322 static void mptsas_doneq_thread(mptsas_doneq_thread_arg_t *arg);

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



```
327 static void mptsas_start_watch_reset_delay();
328 static void mptsas_setup_bus_reset_delay(mptsas_t *mpt);
329 static void mptsas_watch_reset_delay(void *arg);
330 static int mptsas_watch_reset_delay_subr(mptsas_t *mpt);

332 static int mptsas_outstanding_cmds_n(mptsas_t *mpt);
333 /*
334  * helper functions
335  */
336 static void mptsas_dump_cmd(mptsas_t *mpt, mptsas_cmd_t *cmd);

338 static dev_info_t *mptsas_find_child(dev_info_t *pdip, char *name);
339 static dev_info_t *mptsas_find_child_phy(dev_info_t *pdip, uint8_t phy);
340 static dev_info_t *mptsas_find_child_addr(dev_info_t *pdip, uint64_t sasaddr,
341     int lun);
342 static mdi_pathinfo_t *mptsas_find_path_addr(dev_info_t *pdip, uint64_t sasaddr,
343     int lun);
344 static mdi_pathinfo_t *mptsas_find_path_phy(dev_info_t *pdip, uint8_t phy);
345 static dev_info_t *mptsas_find_smp_child(dev_info_t *pdip, char *str_wwn);

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

351 static mptsas_target_t *mptsas_phy_to_tgt(mptsas_t *mpt, int phymask,
352     uint8_t phy);
353 static mptsas_target_t *mptsas_wwid_to_ptgt(mptsas_t *mpt, int phymask,
354     uint64_t wwid);
355 static mptsas_smp_t *mptsas_wwid_to_psmp(mptsas_t *mpt, int phymask,
356     uint64_t wwid);

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

361 static int mptsas_get_target_device_info(mptsas_t *mpt, uint32_t page_address,
362     uint16_t *handle, mptsas_target_t **pptgt);
363 static void mptsas_update_phymask(mptsas_t *mpt);
364 static inline void mptsas_remove_cmd0(mptsas_t *mpt, mptsas_cmd_t *cmd);

366 static int mptsas_send_sep(mptsas_t *mpt, mptsas_target_t *ptgt,
367     uint32_t *status, uint8_t cmd);
368 static dev_info_t *mptsas_get_dip_from_dev(dev_t dev,
369     mptsas_phymask_t *phymask);
370 static mptsas_target_t *mptsas_addr_to_ptgt(mptsas_t *mpt, char *addr,
371     mptsas_phymask_t phymask);
372 static int mptsas_set_led_status(mptsas_t *mpt, mptsas_target_t *ptgt,
373     uint32_t slotstatus);


376 /*
377  * Enumeration / DR functions
378  */
379 static void mptsas_config_all(dev_info_t *pdip);
380 static int mptsas_config_one_addr(dev_info_t *pdip, uint64_t sasaddr, int lun,
381     dev_info_t **lundip);
382 static int mptsas_config_one_phy(dev_info_t *pdip, uint8_t phy, int lun,
383     dev_info_t **lundip);

385 static int mptsas_config_target(dev_info_t *pdip, mptsas_target_t *ptgt);
386 static int mptsas_offline_target(dev_info_t *pdip, char *name);

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

391 static int mptsas_config_luns(dev_info_t *pdip, mptsas_target_t *ptgt);
```

```
392 static int mptsas_probe_lun(dev_info_t *pdip, int lun,
393     dev_info_t **dip, mptsas_target_t *ptgt);

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

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

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

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

414 static int mptsas_event_query(mptsas_t *mpt, mptsas_event_query_t *data,
415     int mode, int *rval);
416 static int mptsas_event_enable(mptsas_t *mpt, mptsas_event_enable_t *data,
417     int mode, int *rval);
418 static int mptsas_event_report(mptsas_t *mpt, mptsas_event_report_t *data,
419     int mode, int *rval);
420 static void mptsas_record_event(void *args);
421 static int mptsas_reg_access(mptsas_t *mpt, mptsas_reg_access_t *data,
422     int mode);

424 static void mptsas_hash_init(mptsas_hash_table_t *hashtab);
425 static void mptsas_hash_uninit(mptsas_hash_table_t *hashtab, size_t datalen);
426 static void mptsas_hash_add(mptsas_hash_table_t *hashtab, void *data);
427 static void * mptsas_hash_rem(mptsas_hash_table_t *hashtab, uint64_t key1,
428     mptsas_phymask_t key2);
429 static void * mptsas_hash_search(mptsas_hash_table_t *hashtab, uint64_t key1,
430     mptsas_phymask_t key2);
431 static void * mptsas_hash_traverse(mptsas_hash_table_t *hashtab, int pos);

433 mptsas_target_t *mptsas_tgt_alloc(mptsas_hash_table_t *, uint16_t, uint64_t,
434     uint32_t, mptsas_phymask_t, uint8_t, mptsas_t *);
435 static mptsas_smp_t *mptsas_smp_alloc(mptsas_hash_table_t *hashtab,
436     mptsas_smp_t *data);
437 static void mptsas_smp_free(mptsas_hash_table_t *hashtab, uint64_t wwid,
438     mptsas_phymask_t phymask);
439 static void mptsas_tgt_free(mptsas_hash_table_t *, uint64_t, mptsas_phymask_t);
440 static void * mptsas_search_by_devhdl(mptsas_hash_table_t *, uint16_t);
441 static int mptsas_online_smp(dev_info_t *pdip, mptsas_smp_t *smp_node,
442     dev_info_t **smp_dip);

444 /*
445  * Power management functions
446  */
447 static int mptsas_get_pci_cap(mptsas_t *mpt);
448 static int mptsas_init_pm(mptsas_t *mpt);

450 /*
451  * MPT MSI tunable:
452  *
453  * By default MSI is enabled on all supported platforms.
454  */
455 boolean_t mptsas_enable_msi = B_TRUE;
456 boolean_t mptsas_physical_bind_failed_page_83 = B_FALSE;
```



```
458 static int mptsas_register_intrs(mptsas_t *);
459 static void mptsas_unregister_intrs(mptsas_t *);
460 static int mptsas_add_intrs(mptsas_t *, int);
461 static void mptsas_rem_intrs(mptsas_t *);

463 /*
464  * FMA Prototypes
465  */
466 static void mptsas_fm_init(mptsas_t *mpt);
467 static void mptsas_fm_fini(mptsas_t *mpt);
468 static int mptsas_fm_error_cb(dev_info_t *, ddi_fm_error_t *, const void *);

470 extern pri_t minclsyspri, maxclsyspri;

472 /*
473  * This device is created by the SCSI pseudo nexus driver (SCSI vHCI).  It is
474  * under this device that the paths to a physical device are created when
475  * MPxIO is used.
476  */
477 extern dev_info_t       *scsi_vhci_dip;

479 /*
480  * Tunable timeout value for Inquiry VPD page 0x83
481  * By default the value is 30 seconds.
482  */
483 int mptsas_inq83_retry_timeout = 30;

485 /*
486  * This is used to allocate memory for message frame storage, not for
487  * data I/O DMA. All message frames must be stored in the first 4G of
488  * physical memory.
489  */
490 ddi_dma_attr_t mptsas_dma_attrs = {
491         DMA_ATTR_V0,    /* attribute layout version             */
492         0x0ull,         /* address low - should be 0 (longlong) */
493         0xffffffffull,  /* address high - 32-bit max range      */
494         0x00ffffffull,  /* count max - max DMA object size      */
495         4,              /* allocation alignment requirements    */
496         0x78,           /* burstsizes - binary encoded values   */
497         1,              /* minxfer - gran. of DMA engine        */
498         0x00ffffffull,  /* maxxfer - gran. of DMA engine        */
499         0xffffffffull,  /* max segment size (DMA boundary)      */
500         MPTSAS_MAX_DMA_SEGS, /* scatter/gather list length      */
501         512,            /* granularity - device transfer size   */
502         0               /* flags, set to 0                      */
503 };
_____unchanged_portion_omitted_

1691 static int
1692 mptsas_do_detach(dev_info_t *dip)
1693 {
1694         mptsas_t        *mpt;
1695         scsi_hba_tran_t *tran;
1696         int             circ = 0;
1697         int             circ1 = 0;
1698         mdi_pathinfo_t  *pip = NULL;
1699         int             i;
1700         int             doneq_thread_num = 0;

1702         NDBG0(("mptsas_do_detach: dip=0x%p", (void *)dip));

1704         if ((tran = ndi_flavorv_get(dip, SCSA_FLAVOR_SCSI_DEVICE)) == NULL)
1705                 return (DDI_FAILURE);

1707         mpt = TRAN2MPT(tran);
1708         if (!mpt) {
```

```
1709                    return (DDI_FAILURE);
1710            }
1711            /*
1712             * Still have pathinfo child, should not detach mpt driver
1713             */
1714            if (scsi_hba_iport_unit_address(dip)) {
1715                    if (mpt->m_mpxio_enable) {
1716                            /*
1717                             * MPxIO enabled for the iport
1718                             */
1719                            ndi_devi_enter(scsi_vhci_dip, &circ1);
1720                            ndi_devi_enter(dip, &circ);
1721                            while (pip = mdi_get_next_client_path(dip, NULL)) {
1722                                    if (mdi_pi_free(pip, 0) == MDI_SUCCESS) {
1723                                            continue;
1724                                    }
1725                                    ndi_devi_exit(dip, circ);
1726                                    ndi_devi_exit(scsi_vhci_dip, circ1);
1727                                    NDBG12(("detach failed because of "
1728                                        "outstanding path info"));
1729                                    return (DDI_FAILURE);
1730                            }
1731                            ndi_devi_exit(dip, circ);
1732                            ndi_devi_exit(scsi_vhci_dip, circ1);
1733                            (void) mdi_phci_unregister(dip, 0);
1734                    }

1736                    ddi_prop_remove_all(dip);

1738                    return (DDI_SUCCESS);
1739            }

1741            /* Make sure power level is D0 before accessing registers */
1742            if (mpt->m_options & MPTSAS_OPT_PM) {
1743                    (void) pm_busy_component(dip, 0);
1744                    if (mpt->m_power_level != PM_LEVEL_D0) {
1745                            if (pm_raise_power(dip, 0, PM_LEVEL_D0) !=
1746                                DDI_SUCCESS) {
1747                                mptsas_log(mpt, CE_WARN,
1748                                    "mptsas%d: Raise power request failed.",
1749                                    mpt->m_instance);
1750                                (void) pm_idle_component(dip, 0);
1751                                return (DDI_FAILURE);
1752                            }
1753                    }
1754            }

1756            /*
1757             * Send RAID action system shutdown to sync IR.  After action, send a
1758             * Message Unit Reset. Since after that DMA resource will be freed,
1759             * set ioc to READY state will avoid HBA initiated DMA operation.
1760             */
1761            mutex_enter(&mpt->m_mutex);
1762            MPTSAS_DISABLE_INTR(mpt);
1763            mptsas_raid_action_system_shutdown(mpt);
1764            mpt->m_softstate |= MPTSAS_SS_MSG_UNIT_RESET;
1765            (void) mptsas_ioc_reset(mpt, FALSE);
1764            (void) mptsas_ioc_reset(mpt);
1766            mutex_exit(&mpt->m_mutex);
1767            mptsas_rem_intrs(mpt);
1768            ddi_taskq_destroy(mpt->m_event_taskq);
1769            ddi_taskq_destroy(mpt->m_dr_taskq);

1771            if (mpt->m_doneq_thread_n) {
1772                    mutex_enter(&mpt->m_doneq_mutex);
1773                    doneq_thread_num = mpt->m_doneq_thread_n;
```



```
1774                    for (i = 0; i < mpt->m_doneq_thread_n; i++) {
1775                            mutex_enter(&mpt->m_doneq_thread_id[i].mutex);
1776                            mpt->m_doneq_thread_id[i].flag &=
1777                                (~MPTSAS_DONEQ_THREAD_ACTIVE);
1778                            cv_signal(&mpt->m_doneq_thread_id[i].cv);
1779                            mutex_exit(&mpt->m_doneq_thread_id[i].mutex);
1780                    }
1781                    while (mpt->m_doneq_thread_n) {
1782                            cv_wait(&mpt->m_doneq_thread_cv,
1783                                &mpt->m_doneq_mutex);
1784                    }
1785                    for (i = 0;  i < doneq_thread_num; i++) {
1786                            cv_destroy(&mpt->m_doneq_thread_id[i].cv);
1787                            mutex_destroy(&mpt->m_doneq_thread_id[i].mutex);
1788                    }
1789                    kmem_free(mpt->m_doneq_thread_id,
1790                        sizeof (mptsas_doneq_thread_list_t)
1791                        * doneq_thread_num);
1792                    mutex_exit(&mpt->m_doneq_mutex);
1793                    cv_destroy(&mpt->m_doneq_thread_cv);
1794                    mutex_destroy(&mpt->m_doneq_mutex);
1795            }

1797            scsi_hba_reset_notify_tear_down(mpt->m_reset_notify_listf);

1799            mptsas_list_del(mpt);

1801            /*
1802             * Cancel timeout threads for this mpt
1803             */
1804            mutex_enter(&mpt->m_mutex);
1805            if (mpt->m_quiesce_timeid) {
1806                    timeout_id_t tid = mpt->m_quiesce_timeid;
1807                    mpt->m_quiesce_timeid = 0;
1808                    mutex_exit(&mpt->m_mutex);
1809                    (void) untimeout(tid);
1810                    mutex_enter(&mpt->m_mutex);
1811            }

1813            if (mpt->m_restart_cmd_timeid) {
1814                    timeout_id_t tid = mpt->m_restart_cmd_timeid;
1815                    mpt->m_restart_cmd_timeid = 0;
1816                    mutex_exit(&mpt->m_mutex);
1817                    (void) untimeout(tid);
1818                    mutex_enter(&mpt->m_mutex);
1819            }

1821            mutex_exit(&mpt->m_mutex);

1823            /*
1824             * last mpt? ... if active, CANCEL watch threads.
1825             */
1826            mutex_enter(&mptsas_global_mutex);
1827            if (mptsas_head == NULL) {
1828                    timeout_id_t tid;
1829                    /*
1830                     * Clear mptsas_timeouts_enable so that the watch thread
1831                     * gets restarted on DDI_ATTACH
1832                     */
1833                    mptsas_timeouts_enabled = 0;
1834                    if (mptsas_timeout_id) {
1835                            tid = mptsas_timeout_id;
1836                            mptsas_timeout_id = 0;
1837                            mutex_exit(&mptsas_global_mutex);
1838                            (void) untimeout(tid);
1839                            mutex_enter(&mptsas_global_mutex);
```

```
1840                }
1841                if (mptsas_reset_watch) {
1842                        tid = mptsas_reset_watch;
1843                        mptsas_reset_watch = 0;
1844                        mutex_exit(&mptsas_global_mutex);
1845                        (void) untimeout(tid);
1846                        mutex_enter(&mptsas_global_mutex);
1847                }
1848        }
1849        mutex_exit(&mptsas_global_mutex);

1851        /*
1852         * Delete Phy stats
1853         */
1854        mptsas_destroy_phy_stats(mpt);

1856        /*
1857         * Delete nt_active.
1858         */
1859        mutex_enter(&mpt->m_mutex);
1860        mptsas_hash_uninit(&mpt->m_active->m_tgttbl, sizeof (mptsas_target_t));
1861        mptsas_hash_uninit(&mpt->m_active->m_smptbl, sizeof (mptsas_smp_t));
1862        mptsas_free_active_slots(mpt);
1863        mutex_exit(&mpt->m_mutex);

1865        /* deallocate everything that was allocated in mptsas_attach */
1866        mptsas_cache_destroy(mpt);

1868        mptsas_hba_fini(mpt);
1869        mptsas_cfg_fini(mpt);

1871        /* Lower the power informing PM Framework */
1872        if (mpt->m_options & MPTSAS_OPT_PM) {
1873                if (pm_lower_power(dip, 0, PM_LEVEL_D3) != DDI_SUCCESS)
1874                        mptsas_log(mpt, CE_WARN,
1875                            "!mptsas%d: Lower power request failed "
1876                            "during detach, ignoring.",
1877                            mpt->m_instance);
1878        }

1880        mutex_destroy(&mpt->m_intr_mutex);
1881        mutex_destroy(&mpt->m_passthru_mutex);
1882        mutex_destroy(&mpt->m_mutex);
1883        for (i = 0; i < MPTSAS_MAX_PHYS; i++) {
1884                mutex_destroy(&mpt->m_phy_info[i].smhba_info.phy_mutex);
1885        }
1886        cv_destroy(&mpt->m_cv);
1887        cv_destroy(&mpt->m_passthru_cv);
1888        cv_destroy(&mpt->m_fw_cv);
1889        cv_destroy(&mpt->m_config_cv);
1890        cv_destroy(&mpt->m_fw_diag_cv);


1893        mptsas_smp_teardown(mpt);
1894        mptsas_hba_teardown(mpt);

1896        mptsas_config_space_fini(mpt);

1898        mptsas_free_handshake_msg(mpt);

1900        mptsas_fm_fini(mpt);
1901        ddi_soft_state_free(mptsas_state, ddi_get_instance(dip));
1902        ddi_prop_remove_all(dip);

1904        return (DDI_SUCCESS);
1905 }
```

_____unchanged_portion_omitted_


```
12323 static int
12324 mptsas_init_chip(mptsas_t *mpt, int first_time)
12325 {
12326        ddi_dma_cookie_t        cookie;
12327        uint32_t                i;
12328        int                     rval;

12330        /*
12331         * Check to see if the firmware image is valid
12332         */
12333        if (ddi_get32(mpt->m_datap, &mpt->m_reg->HostDiagnostic) &
12334            MPI2_DIAG_FLASH_BAD_SIG) {
12335                mptsas_log(mpt, CE_WARN, "mptsas bad flash signature!");
12336                goto fail;
12337        }

12339        /*
12340         * Reset the chip
12341         */
12342        rval = mptsas_ioc_reset(mpt, first_time);
12341        rval = mptsas_ioc_reset(mpt);
12343        if (rval == MPTSAS_RESET_FAIL) {
12344                mptsas_log(mpt, CE_WARN, "hard reset failed!");
12345                goto fail;
12346        }

12348        if ((rval == MPTSAS_SUCCESS_MUR) && (!first_time)) {
12349                goto mur;
12350        }
12351        /*
12352         * Setup configuration space
12353         */
12354        if (mptsas_config_space_init(mpt) == FALSE) {
12355                mptsas_log(mpt, CE_WARN, "mptsas_config_space_init "
12356                    "failed!");
12357                goto fail;
12358        }

12360        /*
12361         * IOC facts can change after a diag reset so all buffers that are
12362         * based on these numbers must be de-allocated and re-allocated.  Get
12363         * new IOC facts each time chip is initialized.
12364         */
12365        if (mptsas_ioc_get_facts(mpt) == DDI_FAILURE) {
12366                mptsas_log(mpt, CE_WARN, "mptsas_ioc_get_facts failed");
12367                goto fail;
12368        }

12370        if (mptsas_alloc_active_slots(mpt, KM_SLEEP)) {
12371                goto fail;
12372        }
12373        /*
12374         * Allocate request message frames, reply free queue, reply descriptor
12375         * post queue, and reply message frames using latest IOC facts.
12376         */
12377        if (mptsas_alloc_request_frames(mpt) == DDI_FAILURE) {
12378                mptsas_log(mpt, CE_WARN, "mptsas_alloc_request_frames failed");
12379                goto fail;
12380        }
12381        if (mptsas_alloc_free_queue(mpt) == DDI_FAILURE) {
12382                mptsas_log(mpt, CE_WARN, "mptsas_alloc_free_queue failed!");
12383                goto fail;
12384        }
12385        if (mptsas_alloc_post_queue(mpt) == DDI_FAILURE) {
12386                mptsas_log(mpt, CE_WARN, "mptsas_alloc_post_queue failed!");
```

```
12387                   goto fail;
12388           }
12389           if (mptsas_alloc_reply_frames(mpt) == DDI_FAILURE) {
12390                   mptsas_log(mpt, CE_WARN, "mptsas_alloc_reply_frames failed!");
12391                   goto fail;
12392           }

12394 mur:
12395           /*
12396            * Re-Initialize ioc to operational state
12397            */
12398           if (mptsas_ioc_init(mpt) == DDI_FAILURE) {
12399                   mptsas_log(mpt, CE_WARN, "mptsas_ioc_init failed");
12400                   goto fail;
12401           }

12403           mptsas_alloc_reply_args(mpt);

12405           /*
12406            * Initialize reply post index.  Reply free index is initialized after
12407            * the next loop.
12408            */
12409           mpt->m_post_index = 0;

12411           /*
12412            * Initialize the Reply Free Queue with the physical addresses of our
12413            * reply frames.
12414            */
12415           cookie.dmac_address = mpt->m_reply_frame_dma_addr;
12416           for (i = 0; i < mpt->m_max_replies; i++) {
12417                   ddi_put32(mpt->m_acc_free_queue_hdl,
12418                       &((uint32_t *)(void *)mpt->m_free_queue)[i],
12419                       cookie.dmac_address);
12420                   cookie.dmac_address += mpt->m_reply_frame_size;
12421           }
12422           (void) ddi_dma_sync(mpt->m_dma_free_queue_hdl, 0, 0,
12423               DDI_DMA_SYNC_FORDEV);

12425           /*
12426            * Initialize the reply free index to one past the last frame on the
12427            * queue.  This will signify that the queue is empty to start with.
12428            */
12429           mpt->m_free_index = i;
12430           ddi_put32(mpt->m_datap, &mpt->m_reg->ReplyFreeHostIndex, i);

12432           /*
12433            * Initialize the reply post queue to 0xFFFFFFFF,0xFFFFFFFF's.
12434            */
12435           for (i = 0; i < mpt->m_post_queue_depth; i++) {
12436                   ddi_put64(mpt->m_acc_post_queue_hdl,
12437                       &((uint64_t *)(void *)mpt->m_post_queue)[i],
12438                       0xFFFFFFFFFFFFFFFF);
12439           }
12440           (void) ddi_dma_sync(mpt->m_dma_post_queue_hdl, 0, 0,
12441               DDI_DMA_SYNC_FORDEV);

12443           /*
12444            * Enable ports
12445            */
12446           if (mptsas_ioc_enable_port(mpt) == DDI_FAILURE) {
12447                   mptsas_log(mpt, CE_WARN, "mptsas_ioc_enable_port failed");
12448                   goto fail;
12449           }

12451           /*
12452            * enable events
```



```
12453            */
12454           if (mptsas_ioc_enable_event_notification(mpt)) {
12455                   goto fail;
12456           }

12458           /*
12459            * We need checks in attach and these.
12460            * chip_init is called in mult. places
12461            */

12463           if ((mptsas_check_dma_handle(mpt->m_dma_req_frame_hdl) !=
12464               DDI_SUCCESS) ||
12465               (mptsas_check_dma_handle(mpt->m_dma_reply_frame_hdl) !=
12466               DDI_SUCCESS) ||
12467               (mptsas_check_dma_handle(mpt->m_dma_free_queue_hdl) !=
12468               DDI_SUCCESS) ||
12469               (mptsas_check_dma_handle(mpt->m_dma_post_queue_hdl) !=
12470               DDI_SUCCESS) ||
12471               (mptsas_check_dma_handle(mpt->m_hshk_dma_hdl) !=
12472               DDI_SUCCESS)) {
12473                   ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_UNAFFECTED);
12474                   goto fail;
12475           }

12477           /* Check all acc handles */
12478           if ((mptsas_check_acc_handle(mpt->m_datap) != DDI_SUCCESS) ||
12479               (mptsas_check_acc_handle(mpt->m_acc_req_frame_hdl) !=
12480               DDI_SUCCESS) ||
12481               (mptsas_check_acc_handle(mpt->m_acc_reply_frame_hdl) !=
12482               DDI_SUCCESS) ||
12483               (mptsas_check_acc_handle(mpt->m_acc_free_queue_hdl) !=
12484               DDI_SUCCESS) ||
12485               (mptsas_check_acc_handle(mpt->m_acc_post_queue_hdl) !=
12486               DDI_SUCCESS) ||
12487               (mptsas_check_acc_handle(mpt->m_hshk_acc_hdl) !=
12488               DDI_SUCCESS) ||
12489               (mptsas_check_acc_handle(mpt->m_config_handle) !=
12490               DDI_SUCCESS)) {
12491                   ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_UNAFFECTED);
12492                   goto fail;
12493           }

12495           return (DDI_SUCCESS);

12497 fail:
12498           return (DDI_FAILURE);
12499 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   79853 Wed May 23 16:20:28 2012
new/usr/src/uts/common/io/scsi/adapters/mpt_sas/mptsas_impl.c
2777 mpt_sas needs to try MUR reset at attach() time.
**********************************************************
```

```
   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  */

  27 /*
  28  * Copyright (c) 2000 to 2010, LSI Corporation.
  29  * All rights reserved.
  30  *
  31  * Redistribution and use in source and binary forms of all code within
  32  * this file that is exclusively owned by LSI, with or without
  33  * modification, is permitted provided that, in addition to the CDDL 1.0
  34  * License requirements, the following conditions are met:
  35  *
  36  *    Neither the name of the author nor the names of its contributors may be
  37  *    used to endorse or promote products derived from this software without
  38  *    specific prior written permission.
  39  *
  40  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
  41  * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
  42  * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
  43  * FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
  44  * COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
  45  * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
  46  * BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
  47  * OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
  48  * AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
  49  * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
  50  * OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
  51  * DAMAGE.
  52  */

  54 /*
  55  * mptsas_impl - This file contains all the basic functions for communicating
  56  * to MPT based hardware.
  57  */

  59 #if defined(lint) || defined(DEBUG)
  60 #define MPTSAS_DEBUG
  61 #endif
```



```
  63 /*
  64  * standard header files
  65  */
  66 #include <sys/note.h>
  67 #include <sys/scsi/scsi.h>
  68 #include <sys/pci.h>

  70 #pragma pack(1)
  71 #include <sys/scsi/adapters/mpt_sas/mpi/mpi2_type.h>
  72 #include <sys/scsi/adapters/mpt_sas/mpi/mpi2.h>
  73 #include <sys/scsi/adapters/mpt_sas/mpi/mpi2_cnfg.h>
  74 #include <sys/scsi/adapters/mpt_sas/mpi/mpi2_init.h>
  75 #include <sys/scsi/adapters/mpt_sas/mpi/mpi2_ioc.h>
  76 #include <sys/scsi/adapters/mpt_sas/mpi/mpi2_sas.h>
  77 #include <sys/scsi/adapters/mpt_sas/mpi/mpi2_tool.h>
  78 #pragma pack()

  80 /*
  81  * private header files.
  82  */
  83 #include <sys/scsi/adapters/mpt_sas/mptsas_var.h>
  84 #include <sys/scsi/adapters/mpt_sas/mptsas_smhba.h>

  86 /*
  87  * FMA header files.
  88  */
  89 #include <sys/fm/io/ddi.h>

  91 #if defined(MPTSAS_DEBUG)
  92 extern uint32_t mptsas_debug_flags;
  93 #endif

  95 /*
  96  *  prototypes
  97  */
  98 static void mptsas_ioc_event_cmdq_add(mptsas_t *mpt, m_event_struct_t *cmd);
  99 static void mptsas_ioc_event_cmdq_delete(mptsas_t *mpt, m_event_struct_t *cmd);
 100 static m_event_struct_t *mptsas_ioc_event_find_by_cmd(mptsas_t *mpt,
 101     struct mptsas_cmd *cmd);

 103 /*
 104  * add ioc evnet cmd into the queue
 105  */
 106 static void
 107 mptsas_ioc_event_cmdq_add(mptsas_t *mpt, m_event_struct_t *cmd)
 108 {
 109         if ((cmd->m_event_linkp = mpt->m_ioc_event_cmdq) == NULL) {
 110                 mpt->m_ioc_event_cmdtail = &cmd->m_event_linkp;
 111                 mpt->m_ioc_event_cmdq = cmd;
 112         } else {
 113                 cmd->m_event_linkp = NULL;
 114                 *(mpt->m_ioc_event_cmdtail) = cmd;
 115                 mpt->m_ioc_event_cmdtail = &cmd->m_event_linkp;
 116         }
 117 }
______unchanged_portion_omitted_

 958 int
 959 mptsas_ioc_reset(mptsas_t *mpt, int first_time)
 958 mptsas_ioc_reset(mptsas_t *mpt)
 960 {
 961         int             polls = 0;
 962         uint32_t        reset_msg;
 963         uint32_t        ioc_state;
```

```
        ioc_state = ddi_get32(mpt->m_datap, &mpt->m_reg->Doorbell);
        /*
         * If chip is already in ready state then there is nothing to do.
         */
        if (ioc_state == MPI2_IOC_STATE_READY) {
                return (MPTSAS_NO_RESET);
        }
        /*
         * If the chip is already operational, we just need to send
         * it a message unit reset to put it back in the ready state
         */
        if (ioc_state & MPI2_IOC_STATE_OPERATIONAL) {
                /*
                 * If the first time, try MUR anyway, because we haven't even
                 * queried the card for m_event_replay and other capabilities.
                 * Other platforms do it this way, we can still do a hard
                 * reset if we need to, MUR takes less time than a full
                 * adapter reset, and there are reports that some HW
                 * combinations will lock up when receiving a hard reset.
                 */
                if ((first_time || mpt->m_event_replay) &&
                    (mpt->m_softstate & MPTSAS_SS_MSG_UNIT_RESET)) {
                if (mpt->m_event_replay && (mpt->m_softstate &
                    MPTSAS_SS_MSG_UNIT_RESET)) {
                        mpt->m_softstate &= ~MPTSAS_SS_MSG_UNIT_RESET;
                        reset_msg = MPI2_FUNCTION_IOC_MESSAGE_UNIT_RESET;
                        ddi_put32(mpt->m_datap, &mpt->m_reg->Doorbell,
                            (reset_msg << MPI2_DOORBELL_FUNCTION_SHIFT));
                        if (mptsas_ioc_wait_for_response(mpt)) {
                                NDBG19(("mptsas_ioc_reset failure sending "
                                    "message_unit_reset\n"));
                                goto hard_reset;
                        }

                        /*
                         * Wait no more than 60 seconds for chip to become
                         * ready.
                         */
                        while ((ddi_get32(mpt->m_datap, &mpt->m_reg->Doorbell) &
                            MPI2_IOC_STATE_READY) == 0x0) {
                                drv_usecwait(1000);
                                if (polls++ > 60000) {
                                        goto hard_reset;
                                }
                        }

                        /*
                         * Save the last reset mode done on IOC which will be
                         * helpful while resuming from suspension.
                         */
                        mpt->m_softstate |= MPTSAS_DID_MSG_UNIT_RESET;

                        /*
                         * the message unit reset would do reset operations
                         * clear reply and request queue, so we should clear
                         * ACK event cmd.
                         */
                        mptsas_destroy_ioc_event_cmd(mpt);
                        return (MPTSAS_SUCCESS_MUR);
                }
        }
hard_reset:
        mpt->m_softstate &= ~MPTSAS_DID_MSG_UNIT_RESET;
        if (mptsas_kick_start(mpt) == DDI_FAILURE) {
                mptsas_fm_ereport(mpt, DDI_FM_DEVICE_NO_RESPONSE);
                ddi_fm_service_impact(mpt->m_dip, DDI_SERVICE_LOST);
```



```
                return (MPTSAS_RESET_FAIL);
        }
        return (MPTSAS_SUCCESS_HARDRESET);
}
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   44532 Wed May 23 16:20:29 2012
new/usr/src/uts/common/sys/scsi/adapters/mpt_sas/mptsas_var.h
2777 mpt_sas needs to try MUR reset at attach() time.
**********************************************************
```

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

/*
 * Copyright (c) 2009, 2010, Oracle and/or its affiliates. All rights reserved.
 * Copyright 2012 Nexenta Systems, Inc. All rights reserved.
 */

/*
 * Copyright (c) 2000 to 2010, LSI Corporation.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms of all code within
 * this file that is exclusively owned by LSI, with or without
 * modification, is permitted provided that, in addition to the CDDL 1.0
 * License requirements, the following conditions are met:
 *
 *    Neither the name of the author nor the names of its contributors may be
 *    used to endorse or promote products derived from this software without
 *    specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
 * FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
 * COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
 * BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
 * OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
 * AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
 * OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
 * DAMAGE.
 */

#ifndef _SYS_SCSI_ADAPTERS_MPTVAR_H
#define _SYS_SCSI_ADAPTERS_MPTVAR_H

#include <sys/byteorder.h>
#include <sys/isa_defs.h>
#include <sys/sunmdi.h>
#include <sys/mdi_impldefs.h>
#include <sys/scsi/adapters/mpt_sas/mptsas_ioctl.h>
```



```
#include <sys/scsi/adapters/mpt_sas/mpi/mpi2_tool.h>
#include <sys/scsi/adapters/mpt_sas/mpi/mpi2_cnfg.h>

#ifdef  __cplusplus
extern "C" {
#endif

/*
 * Compile options
 */
#ifdef DEBUG
#define MPTSAS_DEBUG            /* turn on debugging code */
#endif  /* DEBUG */

#define MPTSAS_INITIAL_SOFT_SPACE       4

#define MAX_MPI_PORTS           16

/*
 * Note below macro definition and data type definition
 * are used for phy mask handling, it should be changed
 * simultaneously.
 */
#define MPTSAS_MAX_PHYS         16
typedef uint16_t                mptsas_phymask_t;

#define MPTSAS_INVALID_DEVHDL   0xffff
#define MPTSAS_SATA_GUID        "sata-guid"

/*
 * MPT HW defines
 */
#define MPTSAS_MAX_DISKS_IN_CONFIG      14
#define MPTSAS_MAX_DISKS_IN_VOL         10
#define MPTSAS_MAX_HOTSPARES            2
#define MPTSAS_MAX_RAIDVOLS             2
#define MPTSAS_MAX_RAIDCONFIGS          5

/*
 * 64-bit SAS WWN is displayed as 16 characters as HEX characters,
 * plus two means the prefix 'w' and end of the string '\0'.
 */
#define MPTSAS_WWN_STRLEN       (16 + 2)
#define MPTSAS_MAX_GUID_LEN     64

/*
 * DMA routine flags
 */
#define MPTSAS_DMA_HANDLE_ALLOCD        0x2
#define MPTSAS_DMA_MEMORY_ALLOCD        0x4
#define MPTSAS_DMA_HANDLE_BOUND 0x8

/*
 * If the HBA supports DMA or bus-mastering, you may have your own
 * scatter-gather list for physically non-contiguous memory in one
 * I/O operation; if so, there's probably a size for that list.
 * It must be placed in the ddi_dma_lim_t structure, so that the system
 * DMA-support routines can use it to break up the I/O request, so we
 * define it here.
 */
#if defined(__sparc)
#define MPTSAS_MAX_DMA_SEGS     1
#define MPTSAS_MAX_CMD_SEGS     1
#else
#define MPTSAS_MAX_DMA_SEGS     256
#define MPTSAS_MAX_CMD_SEGS     257
```

```
 128 #endif
 129 #define MPTSAS_MAX_FRAME_SGES(mpt) \
 130         (((mpt->m_req_frame_size - (sizeof (MPI2_SCSI_IO_REQUEST))) / 8) + 1)

 132 /*
 133  * Caculating how many 64-bit DMA simple elements can be stored in the first
 134  * frame. Note that msg_scsi_io_request contains 2 double-words (8 bytes) for
 135  * element storage.  And 64-bit dma element is 3 double-words (12 bytes) in
 136  * size.
 137  */
 138 #define MPTSAS_MAX_FRAME_SGES64(mpt) \
 139         ((mpt->m_req_frame_size - \
 140         (sizeof (MPI2_SCSI_IO_REQUEST)) + sizeof (MPI2_SGE_IO_UNION)) / 12)

 142 /*
 143  * Scatter-gather list structure defined by HBA hardware
 144  */
 145 typedef struct NcrTableIndirect {       /* Table Indirect entries */
 146         uint32_t count;         /* 24 bit count */
 147         union {
 148                 uint32_t address32;     /* 32 bit address */
 149                 struct {
 150                         uint32_t Low;
 151                         uint32_t High;
 152                 } address64;            /* 64 bit address */
 153         } addr;
 154 } mptti_t;
_____unchanged_portion_omitted_

1062 /*
1063  * inq_dtype:
1064  * Bits 5 through 7 are the Peripheral Device Qualifier
1065  * 001b: device not connected to the LUN
1066  * Bits 0 through 4 are the Peripheral Device Type
1067  * 1fh: Unknown or no device type
1068  *
1069  * Although the inquiry may return success, the following value
1070  * means no valid LUN connected.
1071  */
1072 #define MPTSAS_VALID_LUN(sd_inq) \
1073         (((sd_inq->inq_dtype & 0xe0) != 0x20) && \
1074         ((sd_inq->inq_dtype & 0x1f) != 0x1f))

1076 /*
1077  * Default is to have 10 retries on receiving QFULL status and
1078  * each retry to be after 100 ms.
1079  */
1080 #define QFULL_RETRIES           10
1081 #define QFULL_RETRY_INTERVAL    100

1083 /*
1084  * Handy macros
1085  */
1086 #define Tgt(sp) ((sp)->cmd_pkt->pkt_address.a_target)
1087 #define Lun(sp) ((sp)->cmd_pkt->pkt_address.a_lun)

1089 #define IS_HEX_DIGIT(n) (((n) >= '0' && (n) <= '9') || \
1090         ((n) >= 'a' && (n) <= 'f') || ((n) >= 'A' && (n) <= 'F'))

1092 /*
1093  * poll time for mptsas_pollret() and mptsas_wait_intr()
1094  */
1095 #define MPTSAS_POLL_TIME        30000   /* 30 seconds */

1097 /*
1098  * default time for mptsas_do_passthru
```



```
1099  */
1100 #define MPTSAS_PASS_THRU_TIME_DEFAULT   60      /* 60 seconds */

1102 /*
1103  * macro to return the effective address of a given per-target field
1104  */
1105 #define EFF_ADDR(start, offset)         ((start) + (offset))

1107 #define SDEV2ADDR(devp)         (&((devp)->sd_address))
1108 #define SDEV2TRAN(devp)         ((devp)->sd_address.a_hba_tran)
1109 #define PKT2TRAN(pkt)           ((pkt)->pkt_address.a_hba_tran)
1110 #define ADDR2TRAN(ap)           ((ap)->a_hba_tran)
1111 #define DIP2TRAN(dip)           (ddi_get_driver_private(dip))


1114 #define TRAN2MPT(hba)           ((mptsas_t *)(hba)->tran_hba_private)
1115 #define DIP2MPT(dip)            (TRAN2MPT((scsi_hba_tran_t *)DIP2TRAN(dip)))
1116 #define SDEV2MPT(sd)            (TRAN2MPT(SDEV2TRAN(sd)))
1117 #define PKT2MPT(pkt)            (TRAN2MPT(PKT2TRAN(pkt)))

1119 #define ADDR2MPT(ap)            (TRAN2MPT(ADDR2TRAN(ap)))

1121 #define POLL_TIMEOUT            (2 * SCSI_POLL_TIMEOUT * 1000000)
1122 #define SHORT_POLL_TIMEOUT      (1000000)       /* in usec, about 1 secs */
1123 #define MPTSAS_QUIESCE_TIMEOUT  1               /* 1 sec */
1124 #define MPTSAS_PM_IDLE_TIMEOUT  60              /* 60 seconds */

1126 #define MPTSAS_GET_ISTAT(mpt)  (ddi_get32((mpt)->m_datap, \
1127                         &(mpt)->m_reg->HostInterruptStatus))

1129 #define MPTSAS_SET_SIGP(P) \
1130                 ClrSetBits(mpt->m_devaddr + NREG_ISTAT, 0, NB_ISTAT_SIGP)

1132 #define MPTSAS_RESET_SIGP(P) (void) ddi_get8(mpt->m_datap, \
1133                         (uint8_t *)(mpt->m_devaddr + NREG_CTEST2))

1135 #define MPTSAS_GET_INTCODE(P) (ddi_get32(mpt->m_datap, \
1136                         (uint32_t *)(mpt->m_devaddr + NREG_DSPS)))


1139 #define MPTSAS_START_CMD(mpt, req_desc_lo, req_desc_hi) \
1140         ddi_put32(mpt->m_datap, &mpt->m_reg->RequestDescriptorPostLow,\
1141             req_desc_lo);\
1142         ddi_put32(mpt->m_datap, &mpt->m_reg->RequestDescriptorPostHigh,\
1143             req_desc_hi);

1145 #define INTPENDING(mpt) \
1146         (MPTSAS_GET_ISTAT(mpt) & MPI2_HIS_REPLY_DESCRIPTOR_INTERRUPT)

1148 /*
1149  * Mask all interrupts to disable
1150  */
1151 #define MPTSAS_DISABLE_INTR(mpt)        \
1152         ddi_put32((mpt)->m_datap, &(mpt)->m_reg->HostInterruptMask, \
1153             (MPI2_HIM_RIM | MPI2_HIM_DIM | MPI2_HIM_RESET_IRQ_MASK))

1155 /*
1156  * Mask Doorbell and Reset interrupts to enable reply desc int.
1157  */
1158 #define MPTSAS_ENABLE_INTR(mpt) \
1159         ddi_put32(mpt->m_datap, &mpt->m_reg->HostInterruptMask, \
1160         (MPI2_HIM_DIM | MPI2_HIM_RESET_IRQ_MASK))

1162 #define MPTSAS_GET_NEXT_REPLY(mpt, index)  \
1163         &((uint64_t *)(void *)mpt->m_post_queue)[index]
```

```
#define MPTSAS_GET_NEXT_FRAME(mpt, SMID) \
        (mpt->m_req_frame + (mpt->m_req_frame_size * SMID))

#define ClrSetBits32(hdl, reg, clr, set) \
        ddi_put32(hdl, (reg), \
            ((ddi_get32(mpt->m_datap, (reg)) & ~(clr)) | (set)))

#define ClrSetBits(reg, clr, set) \
        ddi_put8(mpt->m_datap, (uint8_t *)(reg), \
                ((ddi_get8(mpt->m_datap, (uint8_t *)(reg)) & ~(clr)) | (set)))

#define MPTSAS_WAITQ_RM(mpt, cmdp)      \
        if ((cmdp = mpt->m_waitq) != NULL) { \
                /* If the queue is now empty fix the tail pointer */    \
                if ((mpt->m_waitq = cmdp->cmd_linkp) == NULL) \
                        mpt->m_waitqtail = &mpt->m_waitq; \
                cmdp->cmd_linkp = NULL; \
                cmdp->cmd_queued = FALSE; \
        }

#define MPTSAS_TX_WAITQ_RM(mpt, cmdp)   \
        if ((cmdp = mpt->m_tx_waitq) != NULL) { \
                /* If the queue is now empty fix the tail pointer */    \
                if ((mpt->m_tx_waitq = cmdp->cmd_linkp) == NULL) \
                        mpt->m_tx_waitqtail = &mpt->m_tx_waitq; \
                cmdp->cmd_linkp = NULL; \
                cmdp->cmd_queued = FALSE; \
        }

/*
 * defaults for the global properties
 */
#define DEFAULT_SCSI_OPTIONS    SCSI_OPTIONS_DR
#define DEFAULT_TAG_AGE_LIMIT   2
#define DEFAULT_WD_TICK         10

/*
 * invalid hostid.
 */
#define MPTSAS_INVALID_HOSTID  -1

/*
 * Get/Set hostid from SCSI port configuration page
 */
#define MPTSAS_GET_HOST_ID(configuration) (configuration & 0xFF)
#define MPTSAS_SET_HOST_ID(hostid) (hostid | ((1 << hostid) << 16))

/*
 * Config space.
 */
#define MPTSAS_LATENCY_TIMER    0x40

/*
 * Offset to firmware version
 */
#define MPTSAS_FW_VERSION_OFFSET        9

/*
 * Offset and masks to get at the ProductId field
 */
#define MPTSAS_FW_PRODUCTID_OFFSET      8
#define MPTSAS_FW_PRODUCTID_MASK        0xFFFF0000
#define MPTSAS_FW_PRODUCTID_SHIFT       16

/*
 * Subsystem ID for HBAs.
```



```
 */
#define MPTSAS_HBA_SUBSYSTEM_ID    0x10C0
#define MPTSAS_RHEA_SUBSYSTEM_ID        0x10B0

/*
 * reset delay tick
 */
#define MPTSAS_WATCH_RESET_DELAY_TICK 50        /* specified in milli seconds */

/*
 * Ioc reset return values
 */
#define MPTSAS_RESET_FAIL       -1
#define MPTSAS_NO_RESET         0
#define MPTSAS_SUCCESS_HARDRESET        1
#define MPTSAS_SUCCESS_MUR      2

/*
 * throttle support.
 */
#define MAX_THROTTLE    32
#define HOLD_THROTTLE   0
#define DRAIN_THROTTLE  -1
#define QFULL_THROTTLE  -2

/*
 * Passthrough/config request flags
 */
#define MPTSAS_DATA_ALLOCATED           0x0001
#define MPTSAS_DATAOUT_ALLOCATED        0x0002
#define MPTSAS_REQUEST_POOL_CMD         0x0004
#define MPTSAS_ADDRESS_REPLY            0x0008
#define MPTSAS_CMD_TIMEOUT              0x0010

/*
 * response code tlr flag
 */
#define MPTSAS_SCSI_RESPONSE_CODE_TLR_OFF       0x02

/*
 * System Events
 */
#ifndef DDI_VENDOR_LSI
#define DDI_VENDOR_LSI  "LSI"
#endif  /* DDI_VENDOR_LSI */

/*
 * Shared functions
 */
int mptsas_save_cmd(struct mptsas *mpt, struct mptsas_cmd *cmd);
void mptsas_remove_cmd(mptsas_t *mpt, mptsas_cmd_t *cmd);
void mptsas_waitq_add(mptsas_t *mpt, mptsas_cmd_t *cmd);
void mptsas_log(struct mptsas *mpt, int level, char *fmt, ...);
int mptsas_poll(mptsas_t *mpt, mptsas_cmd_t *poll_cmd, int polltime);
int mptsas_do_dma(mptsas_t *mpt, uint32_t size, int var, int (*callback)());
int mptsas_send_config_request_msg(mptsas_t *mpt, uint8_t action,
        uint8_t pagetype, uint32_t pageaddress, uint8_t pagenumber,
        uint8_t pageversion, uint8_t pagelength, uint32_t
        SGEflagslength, uint32_t SGEaddress32);
int mptsas_send_extended_config_request_msg(mptsas_t *mpt, uint8_t action,
        uint8_t extpagetype, uint32_t pageaddress, uint8_t pagenumber,
        uint8_t pageversion, uint16_t extpagelength,
        uint32_t SGEflagslength, uint32_t SGEaddress32);
int mptsas_update_flash(mptsas_t *mpt, caddr_t ptrbuffer, uint32_t size,
        uint8_t type, int mode);
int mptsas_check_flash(mptsas_t *mpt, caddr_t origfile, uint32_t size,
```

```
1297         uint8_t type, int mode);
1298 int mptsas_download_firmware();
1299 int mptsas_can_download_firmware();
1300 int mptsas_dma_alloc(mptsas_t *mpt, mptsas_dma_alloc_state_t *dma_statep);
1301 void mptsas_dma_free(mptsas_dma_alloc_state_t *dma_statep);
1302 mptsas_phymask_t mptsas_physport_to_phymask(mptsas_t *mpt, uint8_t physport);
1303 void mptsas_fma_check(mptsas_t *mpt, mptsas_cmd_t *cmd);
1304 int mptsas_check_acc_handle(ddi_acc_handle_t handle);
1305 int mptsas_check_dma_handle(ddi_dma_handle_t handle);
1306 void mptsas_fm_ereport(mptsas_t *mpt, char *detail);
1307 int mptsas_dma_addr_create(mptsas_t *mpt, ddi_dma_attr_t dma_attr,
1308     ddi_dma_handle_t *dma_hdp, ddi_acc_handle_t *acc_hdp, caddr_t *dma_memp,
1309     uint32_t alloc_size, ddi_dma_cookie_t *cookiep);
1310 void mptsas_dma_addr_destroy(ddi_dma_handle_t *, ddi_acc_handle_t *);

1312 /*
1313  * impl functions
1314  */
1315 int mptsas_ioc_wait_for_response(mptsas_t *mpt);
1316 int mptsas_ioc_wait_for_doorbell(mptsas_t *mpt);
1317 int mptsas_ioc_reset(mptsas_t *mpt, int);
1316 int mptsas_ioc_reset(mptsas_t *mpt);
1318 int mptsas_send_handshake_msg(mptsas_t *mpt, caddr_t memp, int numbytes,
1319     ddi_acc_handle_t accessp);
1320 int mptsas_get_handshake_msg(mptsas_t *mpt, caddr_t memp, int numbytes,
1321     ddi_acc_handle_t accessp);
1322 int mptsas_send_config_request_msg(mptsas_t *mpt, uint8_t action,
1323     uint8_t pagetype, uint32_t pageaddress, uint8_t pagenumber,
1324     uint8_t pageversion, uint8_t pagelength, uint32_t SGEflagslength,
1325     uint32_t SGEaddress32);
1326 int mptsas_send_extended_config_request_msg(mptsas_t *mpt, uint8_t action,
1327     uint8_t extpagetype, uint32_t pageaddress, uint8_t pagenumber,
1328     uint8_t pageversion, uint16_t extpagelength,
1329     uint32_t SGEflagslength, uint32_t SGEaddress32);

1331 int mptsas_request_from_pool(mptsas_t *mpt, mptsas_cmd_t **cmd,
1332     struct scsi_pkt **pkt);
1333 void mptsas_return_to_pool(mptsas_t *mpt, mptsas_cmd_t *cmd);
1334 void mptsas_destroy_ioc_event_cmd(mptsas_t *mpt);
1335 void mptsas_start_config_page_access(mptsas_t *mpt, mptsas_cmd_t *cmd);
1336 int mptsas_access_config_page(mptsas_t *mpt, uint8_t action, uint8_t page_type,
1337     uint8_t page_number, uint32_t page_address, int (*callback) (mptsas_t *,
1338     caddr_t, ddi_acc_handle_t, uint16_t, uint32_t, va_list), ...);

1340 int mptsas_ioc_task_management(mptsas_t *mpt, int task_type,
1341     uint16_t dev_handle, int lun, uint8_t *reply, uint32_t reply_size,
1342     int mode);
1343 int mptsas_send_event_ack(mptsas_t *mpt, uint32_t event, uint32_t eventcntx);
1344 void mptsas_send_pending_event_ack(mptsas_t *mpt);
1345 void mptsas_set_throttle(struct mptsas *mpt, mptsas_target_t *ptgt, int what);
1346 int mptsas_restart_ioc(mptsas_t *mpt);
1347 void mptsas_update_driver_data(struct mptsas *mpt);
1348 uint64_t mptsas_get_sata_guid(mptsas_t *mpt, mptsas_target_t *ptgt, int lun);

1350 /*
1351  * init functions
1352  */
1353 int mptsas_ioc_get_facts(mptsas_t *mpt);
1354 int mptsas_ioc_get_port_facts(mptsas_t *mpt, int port);
1355 int mptsas_ioc_enable_port(mptsas_t *mpt);
1356 int mptsas_ioc_enable_event_notification(mptsas_t *mpt);
1357 int mptsas_ioc_init(mptsas_t *mpt);

1359 /*
1360  * configuration pages operation
1361  */
```



```
1362 int mptsas_get_sas_device_page0(mptsas_t *mpt, uint32_t page_address,
1363     uint16_t *dev_handle, uint64_t *sas_wwn, uint32_t *dev_info,
1364     uint8_t *physport, uint8_t *phynum, uint16_t *pdevhandle,
1365     uint16_t *slot_num, uint16_t *enclosure);
1366 int mptsas_get_sas_io_unit_page(mptsas_t *mpt);
1367 int mptsas_get_sas_io_unit_page_hndshk(mptsas_t *mpt);
1368 int mptsas_get_sas_expander_page0(mptsas_t *mpt, uint32_t page_address,
1369     mptsas_smp_t *info);
1370 int mptsas_set_ioc_params(mptsas_t *mpt);
1371 int mptsas_get_manufacture_page5(mptsas_t *mpt);
1372 int mptsas_get_sas_port_page0(mptsas_t *mpt, uint32_t page_address,
1373     uint64_t *sas_wwn, uint8_t *portwidth);
1374 int mptsas_get_bios_page3(mptsas_t *mpt,  uint32_t *bios_version);
1375 int
1376 mptsas_get_sas_phy_page0(mptsas_t *mpt, uint32_t page_address,
1377     smhba_info_t *info);
1378 int
1379 mptsas_get_sas_phy_page1(mptsas_t *mpt, uint32_t page_address,
1380     smhba_info_t *info);
1381 int
1382 mptsas_get_manufacture_page0(mptsas_t *mpt);
1383 void
1384 mptsas_create_phy_stats(mptsas_t *mpt, char *iport, dev_info_t *dip);
1385 void mptsas_destroy_phy_stats(mptsas_t *mpt);
1386 int mptsas_smhba_phy_init(mptsas_t *mpt);
1387 /*
1388  * RAID functions
1389  */
1390 int mptsas_get_raid_settings(mptsas_t *mpt, mptsas_raidvol_t *raidvol);
1391 int mptsas_get_raid_info(mptsas_t *mpt);
1392 int mptsas_get_physdisk_settings(mptsas_t *mpt, mptsas_raidvol_t *raidvol,
1393     uint8_t physdisknum);
1394 int mptsas_delete_volume(mptsas_t *mpt, uint16_t volid);
1395 void mptsas_raid_action_system_shutdown(mptsas_t *mpt);

1397 #define MPTSAS_IOCSTATUS(status) (status & MPI2_IOCSTATUS_MASK)
1398 /*
1399  * debugging.
1400  */
1401 #if defined(MPTSAS_DEBUG)

1403 void mptsas_printf(char *fmt, ...);

1405 #define MPTSAS_DBGPR(m, args)   \
1406         if (mptsas_debug_flags & (m)) \
1407                 mptsas_printf args
1408 #else   /* ! defined(MPTSAS_DEBUG) */
1409 #define MPTSAS_DBGPR(m, args)
1410 #endif  /* defined(MPTSAS_DEBUG) */

1412 #define NDBG0(args)     MPTSAS_DBGPR(0x01, args)        /* init */
1413 #define NDBG1(args)     MPTSAS_DBGPR(0x02, args)        /* normal running */
1414 #define NDBG2(args)     MPTSAS_DBGPR(0x04, args)        /* property handling */
1415 #define NDBG3(args)     MPTSAS_DBGPR(0x08, args)        /* pkt handling */

1417 #define NDBG4(args)     MPTSAS_DBGPR(0x10, args)        /* kmem alloc/free */
1418 #define NDBG5(args)     MPTSAS_DBGPR(0x20, args)        /* polled cmds */
1419 #define NDBG6(args)     MPTSAS_DBGPR(0x40, args)        /* interrupts */
1420 #define NDBG7(args)     MPTSAS_DBGPR(0x80, args)        /* queue handling */

1422 #define NDBG8(args)     MPTSAS_DBGPR(0x0100, args)      /* arq */
1423 #define NDBG9(args)     MPTSAS_DBGPR(0x0200, args)      /* Tagged Q'ing */
1424 #define NDBG10(args)    MPTSAS_DBGPR(0x0400, args)      /* halting chip */
1425 #define NDBG11(args)    MPTSAS_DBGPR(0x0800, args)      /* power management */

1427 #define NDBG12(args)    MPTSAS_DBGPR(0x1000, args)      /* enumeration */
```

```
#define NDBG13(args)    MPTSAS_DBGPR(0x2000, args)      /* configuration page */
#define NDBG14(args)    MPTSAS_DBGPR(0x4000, args)      /* LED control */
#define NDBG15(args)    MPTSAS_DBGPR(0x8000, args)

#define NDBG16(args)    MPTSAS_DBGPR(0x010000, args)
#define NDBG17(args)    MPTSAS_DBGPR(0x020000, args)    /* scatter/gather */
#define NDBG18(args)    MPTSAS_DBGPR(0x040000, args)
#define NDBG19(args)    MPTSAS_DBGPR(0x080000, args)    /* handshaking */

#define NDBG20(args)    MPTSAS_DBGPR(0x100000, args)    /* events */
#define NDBG21(args)    MPTSAS_DBGPR(0x200000, args)    /* dma */
#define NDBG22(args)    MPTSAS_DBGPR(0x400000, args)    /* reset */
#define NDBG23(args)    MPTSAS_DBGPR(0x800000, args)    /* abort */

#define NDBG24(args)    MPTSAS_DBGPR(0x1000000, args)   /* capabilities */
#define NDBG25(args)    MPTSAS_DBGPR(0x2000000, args)   /* flushing */
#define NDBG26(args)    MPTSAS_DBGPR(0x4000000, args)
#define NDBG27(args)    MPTSAS_DBGPR(0x8000000, args)

#define NDBG28(args)    MPTSAS_DBGPR(0x10000000, args)  /* hotplug */
#define NDBG29(args)    MPTSAS_DBGPR(0x20000000, args)  /* timeouts */
#define NDBG30(args)    MPTSAS_DBGPR(0x40000000, args)  /* mptsas_watch */
#define NDBG31(args)    MPTSAS_DBGPR(0x80000000, args)  /* negotations */

/*
 * auto request sense
 */
#define RQ_MAKECOM_COMMON(pkt, flag, cmd) \
        (pkt)->pkt_flags = (flag), \
        ((union scsi_cdb *)(pkt)->pkt_cdbp)->scc_cmd = (cmd), \
        ((union scsi_cdb *)(pkt)->pkt_cdbp)->scc_lun = \
            (pkt)->pkt_address.a_lun

#define RQ_MAKECOM_G0(pkt, flag, cmd, addr, cnt) \
        RQ_MAKECOM_COMMON((pkt), (flag), (cmd)), \
        FORMG0ADDR(((union scsi_cdb *)(pkt)->pkt_cdbp), (addr)), \
        FORMG0COUNT(((union scsi_cdb *)(pkt)->pkt_cdbp), (cnt))


#ifdef  __cplusplus
}
```

_____***unchanged_portion_omitted_***