```
**********************************************************
  413907 Tue Dec  4 16:30:22 2012
new/usr/src/uts/common/io/scsi/adapters/mpt_sas/mptsas.c
re #8346 rb2639 KT disk failures
**********************************************************
```

```
/*
 * 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/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>

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

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

/*
```

```
 128  * Resource initilaization for hardware
 129  */
 130 static void mptsas_setup_cmd_reg(mptsas_t *mpt);
 131 static void mptsas_disable_bus_master(mptsas_t *mpt);
 132 static void mptsas_hba_fini(mptsas_t *mpt);
 133 static void mptsas_cfg_fini(mptsas_t *mptsas_blkp);
 134 static int mptsas_hba_setup(mptsas_t *mpt);
 135 static void mptsas_hba_teardown(mptsas_t *mpt);
 136 static int mptsas_config_space_init(mptsas_t *mpt);
 137 static void mptsas_config_space_fini(mptsas_t *mpt);
 138 static void mptsas_iport_register(mptsas_t *mpt);
 139 static int mptsas_smp_setup(mptsas_t *mpt);
 140 static void mptsas_smp_teardown(mptsas_t *mpt);
 141 static int mptsas_cache_create(mptsas_t *mpt);
 142 static void mptsas_cache_destroy(mptsas_t *mpt);
 143 static int mptsas_alloc_request_frames(mptsas_t *mpt);
 144 static int mptsas_alloc_reply_frames(mptsas_t *mpt);
 145 static int mptsas_alloc_free_queue(mptsas_t *mpt);
 146 static int mptsas_alloc_post_queue(mptsas_t *mpt);
 147 static void mptsas_alloc_reply_args(mptsas_t *mpt);
 148 static int mptsas_alloc_extra_sgl_frame(mptsas_t *mpt, mptsas_cmd_t *cmd);
 149 static void mptsas_free_extra_sgl_frame(mptsas_t *mpt, mptsas_cmd_t *cmd);
 150 static int mptsas_init_chip(mptsas_t *mpt, int first_time);

 152 /*
 153  * SCSA function prototypes
 154  */
 155 static int mptsas_scsi_start(struct scsi_address *ap, struct scsi_pkt *pkt);
 156 static int mptsas_scsi_reset(struct scsi_address *ap, int level);
 157 static int mptsas_scsi_abort(struct scsi_address *ap, struct scsi_pkt *pkt);
 158 static int mptsas_scsi_getcap(struct scsi_address *ap, char *cap, int tgtonly);
 159 static int mptsas_scsi_setcap(struct scsi_address *ap, char *cap, int value,
 160     int tgtonly);
 161 static void mptsas_scsi_dmafree(struct scsi_address *ap, struct scsi_pkt *pkt);
 162 static struct scsi_pkt *mptsas_scsi_init_pkt(struct scsi_address *ap,
 163     struct scsi_pkt *pkt, struct buf *bp, int cmdlen, int statuslen,
 164         int tgtlen, int flags, int (*callback)(), caddr_t arg);
 165 static void mptsas_scsi_sync_pkt(struct scsi_address *ap, struct scsi_pkt *pkt);
 166 static void mptsas_scsi_destroy_pkt(struct scsi_address *ap,
 167     struct scsi_pkt *pkt);
 168 static int mptsas_scsi_tgt_init(dev_info_t *hba_dip, dev_info_t *tgt_dip,
 169     scsi_hba_tran_t *hba_tran, struct scsi_device *sd);
 170 static void mptsas_scsi_tgt_free(dev_info_t *hba_dip, dev_info_t *tgt_dip,
 171     scsi_hba_tran_t *hba_tran, struct scsi_device *sd);
 172 static int mptsas_scsi_reset_notify(struct scsi_address *ap, int flag,
 173     void (*callback)(caddr_t), caddr_t arg);
 174 static int mptsas_get_name(struct scsi_device *sd, char *name, int len);
 175 static int mptsas_get_bus_addr(struct scsi_device *sd, char *name, int len);
 176 static int mptsas_scsi_quiesce(dev_info_t *dip);
 177 static int mptsas_scsi_unquiesce(dev_info_t *dip);
 178 static int mptsas_bus_config(dev_info_t *pdip, uint_t flags,
 179     ddi_bus_config_op_t op, void *arg, dev_info_t **childp);

 181 /*
 182  * SMP functions
 183  */
 184 static int mptsas_smp_start(struct smp_pkt *smp_pkt);

 186 /*
 187  * internal function prototypes.
 188  */
 189 static void mptsas_list_add(mptsas_t *mpt);
 190 static void mptsas_list_del(mptsas_t *mpt);

 192 static int mptsas_quiesce_bus(mptsas_t *mpt);
 193 static int mptsas_unquiesce_bus(mptsas_t *mpt);
```



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

 198 static void mptsas_ncmds_checkdrain(void *arg);

 200 static int mptsas_prepare_pkt(mptsas_cmd_t *cmd);
 201 static int mptsas_accept_pkt(mptsas_t *mpt, mptsas_cmd_t *sp);
 202 static int mptsas_accept_txwq_and_pkt(mptsas_t *mpt, mptsas_cmd_t *sp);
 203 static void mptsas_accept_tx_waitq(mptsas_t *mpt);

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

 211 static void mptsas_handle_qfull(mptsas_t *mpt, mptsas_cmd_t *cmd);
 212 static void mptsas_handle_event(void *args);
 213 static int mptsas_handle_event_sync(void *args);
 214 static void mptsas_handle_dr(void *args);
 215 static void mptsas_handle_topo_change(mptsas_topo_change_list_t *topo_node,
 216     dev_info_t *pdip);

 218 static void mptsas_restart_cmd(void *);

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

 226 static uint_t mptsas_intr(caddr_t arg1, caddr_t arg2);
 227 static void mptsas_process_intr(mptsas_t *mpt,
 228     pMpi2ReplyDescriptorsUnion_t reply_desc_union);
 229 static void mptsas_handle_scsi_io_success(mptsas_t *mpt,
 230     pMpi2ReplyDescriptorsUnion_t reply_desc);
 231 static void mptsas_handle_address_reply(mptsas_t *mpt,
 232     pMpi2ReplyDescriptorsUnion_t reply_desc);
 233 static int mptsas_wait_intr(mptsas_t *mpt, int polltime);
 234 static void mptsas_sge_setup(mptsas_t *mpt, mptsas_cmd_t *cmd,
 235     uint32_t *control, pMpi2SCSIIORequest_t frame, ddi_acc_handle_t acc_hdl);

 237 static void mptsas_watch(void *arg);
 238 static void mptsas_watchsubr(mptsas_t *mpt);
 239 static void mptsas_cmd_timeout(mptsas_t *mpt, uint16_t devhdl);
 240 static void mptsas_kill_target(mptsas_t *mpt, mptsas_target_t *ptgt);

 242 static void mptsas_start_passthru(mptsas_t *mpt, mptsas_cmd_t *cmd);
 243 static int mptsas_do_passthru(mptsas_t *mpt, uint8_t *request, uint8_t *reply,
 244     uint8_t *data, uint32_t request_size, uint32_t reply_size,
 245     uint32_t data_size, uint32_t direction, uint8_t *dataout,
 246     uint32_t dataout_size, short timeout, int mode);
 247 static int mptsas_free_devhdl(mptsas_t *mpt, uint16_t devhdl);

 249 static uint8_t mptsas_get_fw_diag_buffer_number(mptsas_t *mpt,
 250     uint32_t unique_id);
 251 static void mptsas_start_diag(mptsas_t *mpt, mptsas_cmd_t *cmd);
 252 static int mptsas_post_fw_diag_buffer(mptsas_t *mpt,
 253     mptsas_fw_diagnostic_buffer_t *pBuffer, uint32_t *return_code);
 254 static int mptsas_release_fw_diag_buffer(mptsas_t *mpt,
 255     mptsas_fw_diagnostic_buffer_t *pBuffer, uint32_t *return_code,
 256     uint32_t diag_type);
 257 static int mptsas_diag_register(mptsas_t *mpt,
 258     mptsas_fw_diag_register_t *diag_register, uint32_t *return_code);
 259 static int mptsas_diag_unregister(mptsas_t *mpt,
```

```
260     mptsas_fw_diag_unregister_t *diag_unregister, uint32_t *return_code);
261 static int mptsas_diag_query(mptsas_t *mpt, mptsas_fw_diag_query_t *diag_query,
262     uint32_t *return_code);
263 static int mptsas_diag_read_buffer(mptsas_t *mpt,
264     mptsas_diag_read_buffer_t *diag_read_buffer, uint8_t *ioctl_buf,
265     uint32_t *return_code, int ioctl_mode);
266 static int mptsas_diag_release(mptsas_t *mpt,
267     mptsas_fw_diag_release_t *diag_release, uint32_t *return_code);
268 static int mptsas_do_diag_action(mptsas_t *mpt, uint32_t action,
269     uint8_t *diag_action, uint32_t length, uint32_t *return_code,
270     int ioctl_mode);
271 static int mptsas_diag_action(mptsas_t *mpt, mptsas_diag_action_t *data,
272     int mode);

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

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

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

285 static void mptsas_check_scsi_io_error(mptsas_t *mpt, pMpi2SCSIIOReply_t reply,
286     mptsas_cmd_t *cmd);
287 static void mptsas_check_task_mgt(mptsas_t *mpt,
288     pMpi2SCSIManagementReply_t reply, mptsas_cmd_t *cmd);
289 static int mptsas_send_scsi_cmd(mptsas_t *mpt, struct scsi_address *ap,
290     mptsas_target_t *ptgt, uchar_t *cdb, int cdblen, struct buf *data_bp,
291     int *resid);

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

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

300 static void mptsas_deliver_doneq_thread(mptsas_t *mpt);
301 static void mptsas_doneq_add(mptsas_t *mpt, mptsas_cmd_t *cmd);
302 static void mptsas_doneq_mv(mptsas_t *mpt, uint64_t t);

304 static mptsas_cmd_t *mptsas_doneq_thread_rm(mptsas_t *mpt, uint64_t t);
305 static void mptsas_doneq_empty(mptsas_t *mpt);
306 static void mptsas_doneq_thread(mptsas_doneq_thread_arg_t *arg);

308 static mptsas_cmd_t *mptsas_waitq_rm(mptsas_t *mpt);
309 static void mptsas_waitq_delete(mptsas_t *mpt, mptsas_cmd_t *cmd);
310 static mptsas_cmd_t *mptsas_tx_waitq_rm(mptsas_t *mpt);
311 static void mptsas_tx_waitq_delete(mptsas_t *mpt, mptsas_cmd_t *cmd);


314 static void mptsas_start_watch_reset_delay();
315 static void mptsas_setup_bus_reset_delay(mptsas_t *mpt);
316 static void mptsas_watch_reset_delay(void *arg);
317 static int mptsas_watch_reset_delay_subr(mptsas_t *mpt);

319 /*
320  * helper functions
321  */
322 static void mptsas_dump_cmd(mptsas_t *mpt, mptsas_cmd_t *cmd);

324 static dev_info_t *mptsas_find_child(dev_info_t *pdip, char *name);
325 static dev_info_t *mptsas_find_child_phy(dev_info_t *pdip, uint8_t phy);
```



```
326 static dev_info_t *mptsas_find_child_addr(dev_info_t *pdip, uint64_t sasaddr,
327     int lun);
328 static mdi_pathinfo_t *mptsas_find_path_addr(dev_info_t *pdip, uint64_t sasaddr,
329     int lun);
330 static mdi_pathinfo_t *mptsas_find_path_phy(dev_info_t *pdip, uint8_t phy);
331 static dev_info_t *mptsas_find_smp_child(dev_info_t *pdip, char *str_wwn);

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

337 static mptsas_target_t *mptsas_phy_to_tgt(mptsas_t *mpt, int phymask,
338     uint8_t phy);
339 static mptsas_target_t *mptsas_wwid_to_ptgt(mptsas_t *mpt, int phymask,
340     uint64_t wwid);
341 static mptsas_smp_t *mptsas_wwid_to_psmp(mptsas_t *mpt, int phymask,
342     uint64_t wwid);

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

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

351 static dev_info_t *mptsas_get_dip_from_dev(dev_t dev,
352     mptsas_phymask_t *phymask);
353 static mptsas_target_t *mptsas_addr_to_ptgt(mptsas_t *mpt, char *addr,
354     mptsas_phymask_t phymask);


357 /*
358  * Enumeration / DR functions
359  */
360 static void mptsas_config_all(dev_info_t *pdip);
361 static int mptsas_config_one_addr(dev_info_t *pdip, uint64_t sasaddr, int lun,
362     dev_info_t **lundip);
363 static int mptsas_config_one_phy(dev_info_t *pdip, uint8_t phy, int lun,
364     dev_info_t **lundip);

366 static int mptsas_config_target(dev_info_t *pdip, mptsas_target_t *ptgt);
367 static int mptsas_offline_target(dev_info_t *pdip, char *name);

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

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

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

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

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

390 static int mptsas_config_smp(dev_info_t *pdip, uint64_t sas_wwn,
391     dev_info_t **smp_dip);
```

```
392 static int mptsas_offline_smp(dev_info_t *pdip, mptsas_smp_t *smp_node,
393     uint_t flags);

395 static int mptsas_event_query(mptsas_t *mpt, mptsas_event_query_t *data,
396     int mode, int *rval);
397 static int mptsas_event_enable(mptsas_t *mpt, mptsas_event_enable_t *data,
398     int mode, int *rval);
399 static int mptsas_event_report(mptsas_t *mpt, mptsas_event_report_t *data,
400     int mode, int *rval);
401 static void mptsas_record_event(void *args);
402 static int mptsas_reg_access(mptsas_t *mpt, mptsas_reg_access_t *data,
403     int mode);

405 static void mptsas_hash_init(mptsas_hash_table_t *hashtab);
406 static void mptsas_hash_uninit(mptsas_hash_table_t *hashtab, size_t datalen);
407 static void mptsas_hash_add(mptsas_hash_table_t *hashtab, void *data);
408 static void * mptsas_hash_rem(mptsas_hash_table_t *hashtab, uint64_t key1,
409     mptsas_phymask_t key2);
410 static void * mptsas_hash_search(mptsas_hash_table_t *hashtab, uint64_t key1,
411     mptsas_phymask_t key2);
412 static void * mptsas_hash_traverse(mptsas_hash_table_t *hashtab, int pos);

414 mptsas_target_t *mptsas_tgt_alloc(mptsas_hash_table_t *, uint16_t, uint64_t,
415     uint32_t, mptsas_phymask_t, uint8_t);
416 static mptsas_smp_t *mptsas_smp_alloc(mptsas_hash_table_t *hashtab,
417     mptsas_smp_t *data);
418 static void mptsas_smp_free(mptsas_hash_table_t *hashtab, uint64_t wwid,
419     mptsas_phymask_t phymask);
420 static void mptsas_tgt_free(mptsas_hash_table_t *, uint64_t, mptsas_phymask_t);
421 static void * mptsas_search_by_devhdl(mptsas_hash_table_t *, uint16_t);
422 static int mptsas_online_smp(dev_info_t *pdip, mptsas_smp_t *smp_node,
423     dev_info_t **smp_dip);

425 /*
426  * Power management functions
427  */
428 static int mptsas_get_pci_cap(mptsas_t *mpt);
429 static int mptsas_init_pm(mptsas_t *mpt);

431 /*
432  * MPT MSI tunable:
433  *
434  * By default MSI is enabled on all supported platforms.
435  */
436 boolean_t mptsas_enable_msi = B_TRUE;
437 boolean_t mptsas_physical_bind_failed_page_83 = B_FALSE;

439 static int mptsas_register_intrs(mptsas_t *);
440 static void mptsas_unregister_intrs(mptsas_t *);
441 static int mptsas_add_intrs(mptsas_t *, int);
442 static void mptsas_rem_intrs(mptsas_t *);

444 /*
445  * FMA Prototypes
446  */
447 static void mptsas_fm_init(mptsas_t *mpt);
448 static void mptsas_fm_fini(mptsas_t *mpt);
449 static int mptsas_fm_error_cb(dev_info_t *, ddi_fm_error_t *, const void *);

451 extern pri_t minclsyspri, maxclsyspri;

453 /*
454  * This device is created by the SCSI pseudo nexus driver (SCSI vHCI).  It is
455  * under this device that the paths to a physical device are created when
456  * MPxIO is used.
457  */
```



```
458 extern dev_info_t       *scsi_vhci_dip;

460 /*
461  * Tunable timeout value for Inquiry VPD page 0x83
462  * By default the value is 30 seconds.
463  */
464 int mptsas_inq83_retry_timeout = 30;
465 /*
466  * Maximum number of command timeouts (0 - 255) considered acceptable.
467  */
468 int mptsas_timeout_threshold = 2;
469 /*
470  * Timeouts exceeding threshold within this period are considered excessive.
471  */
472 int mptsas_timeout_interval = 30;

474 /*
475  * This is used to allocate memory for message frame storage, not for
476  * data I/O DMA. All message frames must be stored in the first 4G of
477  * physical memory.
478  */
479 ddi_dma_attr_t mptsas_dma_attrs = {
480         DMA_ATTR_V0,    /* attribute layout version            */
481         0x0ull,         /* address low - should be 0 (longlong) */
482         0xffffffffull,  /* address high - 32-bit max range      */
483         0x00ffffffull,  /* count max - max DMA object size      */
484         4,              /* allocation alignment requirements    */
485         0x78,           /* burstsizes - binary encoded values   */
486         1,              /* minxfer - gran. of DMA engine        */
487         0x00ffffffull,  /* maxxfer - gran. of DMA engine        */
488         0xffffffffull,  /* max segment size (DMA boundary)      */
489         MPTSAS_MAX_DMA_SEGS, /* scatter/gather list length      */
490         512,            /* granularity - device transfer size   */
491         0               /* flags, set to 0                      */
492 };
_____unchanged_portion_omitted_

8490 /*
8491  * Clean up from a device reset.
8492  * For the case of target reset, this function clears the waitq of all
8493  * commands for a particular target.   For the case of abort task set, this
8494  * function clears the waitq of all commonds for a particular target/lun.
8495  */
8496 static void
8497 mptsas_flush_target(mptsas_t *mpt, ushort_t target, int lun, uint8_t tasktype)
8498 {
8499         mptsas_slots_t  *slots = mpt->m_active;
8500         mptsas_cmd_t    *cmd, *next_cmd;
8501         int             slot;
8502         uchar_t         reason;
8503         uint_t          stat;

8505         NDBG25(("mptsas_flush_target: target=%d lun=%d", target, lun));

8507         /*
8508          * Make sure the I/O Controller has flushed all cmds
8509          * that are associated with this target for a target reset
8510          * and target/lun for abort task set.
8511          * Account for TM requests, which use the last SMID.
8512          */
8513         for (slot = 0; slot <= mpt->m_active->m_n_slots; slot++) {
8514                 if ((cmd = slots->m_slot[slot]) == NULL)
8515                         continue;
8516                 reason = CMD_RESET;
8517                 stat = STAT_DEV_RESET;
8518                 switch (tasktype) {
```

```
8519                case MPI2_SCSITASKMGMT_TASKTYPE_TARGET_RESET:
8520                        if (Tgt(cmd) == target) {
8521                                if (cmd->cmd_tgt_addr->m_timeout < 0) {
8522                                        /*
8523                                         * When timeout requested, propagate
8524                                         * proper reason and statistics to
8525                                         * target drivers.
8526                                         */
8527                                        reason = CMD_TIMEOUT;
8528                                        stat |= STAT_TIMEOUT;
8529                                }
8530
8531                                NDBG25(("mptsas_flush_target discovered non-"
8532                                    "NULL cmd in slot %d, tasktype 0x%x", slot,
8533                                    tasktype));
8534                                mptsas_dump_cmd(mpt, cmd);
8535                                mptsas_remove_cmd(mpt, cmd);
8536                                mptsas_set_pkt_reason(mpt, cmd, reason, stat);
8537                                mptsas_doneq_add(mpt, cmd);
8538                        }
8539                        break;
8540                case MPI2_SCSITASKMGMT_TASKTYPE_ABRT_TASK_SET:
8541                        reason = CMD_ABORTED;
8542                        stat = STAT_ABORTED;
8543                        /*FALLTHROUGH*/
8544                case MPI2_SCSITASKMGMT_TASKTYPE_LOGICAL_UNIT_RESET:
8545                        if ((Tgt(cmd) == target) && (Lun(cmd) == lun)) {

8547                                NDBG25(("mptsas_flush_target discovered non-"
8548                                    "NULL cmd in slot %d, tasktype 0x%x", slot,
8549                                    tasktype));
8550                                mptsas_dump_cmd(mpt, cmd);
8551                                mptsas_remove_cmd(mpt, cmd);
8552                                mptsas_set_pkt_reason(mpt, cmd, reason,
8553                                    stat);
8554                                mptsas_doneq_add(mpt, cmd);
8555                        }
8556                        break;
8557                default:
8558                        break;
8559                }
8560        }

8562        /*
8563         * Flush the waitq and tx_waitq of this target's cmds
8564         */
8565        cmd = mpt->m_waitq;

8567        reason = CMD_RESET;
8568        stat = STAT_DEV_RESET;

8570        switch (tasktype) {
8571        case MPI2_SCSITASKMGMT_TASKTYPE_TARGET_RESET:
8572                while (cmd != NULL) {
8573                        next_cmd = cmd->cmd_linkp;
8574                        if (Tgt(cmd) == target) {
8575                                mptsas_waitq_delete(mpt, cmd);
8576                                mptsas_set_pkt_reason(mpt, cmd,
8577                                    reason, stat);
8578                                mptsas_doneq_add(mpt, cmd);
8579                        }
8580                        cmd = next_cmd;
8581                }
8582                mutex_enter(&mpt->m_tx_waitq_mutex);
8583                cmd = mpt->m_tx_waitq;
8584                while (cmd != NULL) {
```



```
8585                        next_cmd = cmd->cmd_linkp;
8586                        if (Tgt(cmd) == target) {
8587                                mptsas_tx_waitq_delete(mpt, cmd);
8588                                mutex_exit(&mpt->m_tx_waitq_mutex);
8589                                mptsas_set_pkt_reason(mpt, cmd,
8590                                    reason, stat);
8591                                mptsas_doneq_add(mpt, cmd);
8592                                mutex_enter(&mpt->m_tx_waitq_mutex);
8593                        }
8594                        cmd = next_cmd;
8595                }
8596                mutex_exit(&mpt->m_tx_waitq_mutex);
8597                break;
8598        case MPI2_SCSITASKMGMT_TASKTYPE_ABRT_TASK_SET:
8599                reason = CMD_ABORTED;
8600                stat =  STAT_ABORTED;
8601                /*FALLTHROUGH*/
8602        case MPI2_SCSITASKMGMT_TASKTYPE_LOGICAL_UNIT_RESET:
8603                while (cmd != NULL) {
8604                        next_cmd = cmd->cmd_linkp;
8605                        if ((Tgt(cmd) == target) && (Lun(cmd) == lun)) {
8606                                mptsas_waitq_delete(mpt, cmd);
8607                                mptsas_set_pkt_reason(mpt, cmd,
8608                                    reason, stat);
8609                                mptsas_doneq_add(mpt, cmd);
8610                        }
8611                        cmd = next_cmd;
8612                }
8613                mutex_enter(&mpt->m_tx_waitq_mutex);
8614                cmd = mpt->m_tx_waitq;
8615                while (cmd != NULL) {
8616                        next_cmd = cmd->cmd_linkp;
8617                        if ((Tgt(cmd) == target) && (Lun(cmd) == lun)) {
8618                                mptsas_tx_waitq_delete(mpt, cmd);
8619                                mutex_exit(&mpt->m_tx_waitq_mutex);
8620                                mptsas_set_pkt_reason(mpt, cmd,
8621                                    reason, stat);
8622                                mptsas_doneq_add(mpt, cmd);
8623                                mutex_enter(&mpt->m_tx_waitq_mutex);
8624                        }
8625                        cmd = next_cmd;
8626                }
8627                mutex_exit(&mpt->m_tx_waitq_mutex);
8628                break;
8629        default:
8630                mptsas_log(mpt, CE_WARN, "Unknown task management type %d.",
8631                    tasktype);
8632                break;
8633        }
8634 }
```

_____unchanged_portion_omitted_

```
9317 static void
9318 mptsas_watchsubr(mptsas_t *mpt)
9319 {
9320        int             i;
9321        mptsas_cmd_t    *cmd;
9322        mptsas_target_t *ptgt = NULL;

9324        NDBG30(("mptsas_watchsubr: mpt=0x%p", (void *)mpt));

9326 #ifdef MPTSAS_TEST
9327        if (mptsas_enable_untagged) {
9328                mptsas_test_untagged++;
9329        }
9330 #endif
```

```
9332         /*
9333          * Check for commands stuck in active slot
9334          * Account for TM requests, which use the last SMID.
9335          */
9336         for (i = 0; i <= mpt->m_active->m_n_slots; i++) {
9337                 if ((cmd = mpt->m_active->m_slot[i]) != NULL) {
9338                         if ((cmd->cmd_flags & CFLAG_CMDIOC) == 0) {
9339                                 cmd->cmd_active_timeout -=
9340                                     mptsas_scsi_watchdog_tick;
9341                                 if (cmd->cmd_active_timeout <= 0) {
9342                                         /*
9343                                          * There seems to be a command stuck
9344                                          * in the active slot.  Drain throttle.
9345                                          */
9346                                         mptsas_set_throttle(mpt,
9347                                             cmd->cmd_tgt_addr,
9348                                             DRAIN_THROTTLE);
9349                                 }
9350                         }
9351                         if ((cmd->cmd_flags & CFLAG_PASSTHRU) ||
9352                             (cmd->cmd_flags & CFLAG_CONFIG) ||
9353                             (cmd->cmd_flags & CFLAG_FW_DIAG)) {
9354                                 cmd->cmd_active_timeout -=
9355                                     mptsas_scsi_watchdog_tick;
9356                                 if (cmd->cmd_active_timeout <= 0) {
9357                                         /*
9358                                          * passthrough command timeout
9359                                          */
9360                                         cmd->cmd_flags |= (CFLAG_FINISHED |
9361                                             CFLAG_TIMEOUT);
9362                                         cv_broadcast(&mpt->m_passthru_cv);
9363                                         cv_broadcast(&mpt->m_config_cv);
9364                                         cv_broadcast(&mpt->m_fw_diag_cv);
9365                                 }
9366                         }
9367                 }
9368         }

9370         ptgt = (mptsas_target_t *)mptsas_hash_traverse(&mpt->m_active->m_tgttbl,
9371             MPTSAS_HASH_FIRST);
9372         while (ptgt != NULL) {
9373                 /*
9374                  * If we were draining due to a qfull condition,
9375                  * go back to full throttle.
9376                  */
9377                 if ((ptgt->m_t_throttle < MAX_THROTTLE) &&
9378                     (ptgt->m_t_throttle > HOLD_THROTTLE) &&
9379                     (ptgt->m_t_ncmds < ptgt->m_t_throttle)) {
9380                         mptsas_set_throttle(mpt, ptgt, MAX_THROTTLE);
9381                         mptsas_restart_hba(mpt);
9382                 }

9384                 if ((ptgt->m_t_ncmds > 0) &&
9385                     (ptgt->m_timebase)) {

9387                         if (ptgt->m_timebase <=
9388                             mptsas_scsi_watchdog_tick) {
9389                                 ptgt->m_timebase +=
9390                                     mptsas_scsi_watchdog_tick;
9391                                 ptgt = (mptsas_target_t *)mptsas_hash_traverse(
9392                                     &mpt->m_active->m_tgttbl, MPTSAS_HASH_NEXT);
9393                                 continue;
9394                         }

9396                         ptgt->m_timeout -= mptsas_scsi_watchdog_tick;
```



```
9398                         if (ptgt->m_timeout_count > 0) {
9399                                 ptgt->m_timeout_interval +=
9400                                     mptsas_scsi_watchdog_tick;
9401                         }
9402                         if (ptgt->m_timeout_interval > mptsas_timeout_interval)
9403                                 ptgt->m_timeout_interval = 0;
9404                                 ptgt->m_timeout_count = 0;
9405                         }

9407                         if (ptgt->m_timeout < 0) {
9408                                 ptgt->m_timeout_count++;
9409                                 if (ptgt->m_timeout_count >
9410                                     mptsas_timeout_threshold) {
9411                                         ptgt->m_timeout_count = 0;
9412                                         mptsas_kill_target(mpt, ptgt);
9413                                 } else {
9414                                         mptsas_cmd_timeout(mpt, ptgt->m_devhdl);
9415                                 }
9416                                 ptgt = (mptsas_target_t *)mptsas_hash_traverse(
9417                                     &mpt->m_active->m_tgttbl, MPTSAS_HASH_NEXT);
9418                                 continue;
9419                         }

9421                         if ((ptgt->m_timeout) <=
9422                             mptsas_scsi_watchdog_tick) {
9423                                 NDBG23(("pending timeout"));
9424                                 mptsas_set_throttle(mpt, ptgt,
9425                                     DRAIN_THROTTLE);
9426                         }
9427                 }

9429                 ptgt = (mptsas_target_t *)mptsas_hash_traverse(
9430                     &mpt->m_active->m_tgttbl, MPTSAS_HASH_NEXT);
9431         }
9432 }
```

*_____unchanged_portion_omitted_*

```
9456 /*
9457  * target causing too many timeouts
9458  */
9459 static void
9460 mptsas_kill_target(mptsas_t *mpt, mptsas_target_t *ptgt)
9461 {
9462         mptsas_topo_change_list_t       *topo_node = NULL;

9464         NDBG29(("mptsas_tgt_kill: target=%d", ptgt->m_devhdl));
9465         mptsas_log(mpt, CE_WARN, "timeout threshold exceeded for "
9466             "Target %d", ptgt->m_devhdl);

9468         topo_node = kmem_zalloc(sizeof (mptsas_topo_change_list_t), KM_SLEEP);
9469         topo_node->mpt = mpt;
9470         topo_node->un.phymask = ptgt->m_phymask;
9471         topo_node->event = MPTSAS_DR_EVENT_OFFLINE_TARGET;
9472         topo_node->devhdl = ptgt->m_devhdl;
9473         if (ptgt->m_deviceinfo & DEVINFO_DIRECT_ATTACHED)
9474                 topo_node->flags = MPTSAS_TOPO_FLAG_DIRECT_ATTACHED_DEVICE;
9475         else
9476                 topo_node->flags = MPTSAS_TOPO_FLAG_EXPANDER_ATTACHED_DEVICE;
9477         topo_node->object = NULL;

9479         /*
9480          * Launch DR taskq to fake topology change
9481          */
9482         if ((ddi_taskq_dispatch(mpt->m_dr_taskq,
9483             mptsas_handle_dr, (void *)topo_node,
```

```
9484             DDI_NOSLEEP)) != DDI_SUCCESS) {
9485                 mptsas_log(mpt, CE_NOTE, "mptsas start taskq "
9486                     "for fake offline event failed. \n");
9487         }
9488 }

9490 /*
9491  * Device / Hotplug control
9492  */
9493 static int
9494 mptsas_scsi_quiesce(dev_info_t *dip)
9495 {
9496         mptsas_t        *mpt;
9497         scsi_hba_tran_t *tran;

9499         tran = ddi_get_driver_private(dip);
9500         if (tran == NULL || (mpt = TRAN2MPT(tran)) == NULL)
9501                 return (-1);

9503         return (mptsas_quiesce_bus(mpt));
9504 }
```

*unchanged_portion_omitted_*

****
43458 Tue Dec 4 16:30:24 2012
new/usr/src/uts/common/sys/scsi/adapters/mpt_sas/mptsas_var.h
re #8346 rb2639 KT disk failures
****
_____***unchanged_portion_omitted_***

```
156 /*
157  * preferred pkt_private length in 64-bit quantities
158  */
159 #ifdef  _LP64
160 #define PKT_PRIV_SIZE   2
161 #define PKT_PRIV_LEN    16      /* in bytes */
162 #else /* _ILP32 */
163 #define PKT_PRIV_SIZE   1
164 #define PKT_PRIV_LEN    8       /* in bytes */
165 #endif

167 #define PKT2CMD(pkt)    ((struct mptsas_cmd *)((pkt)->pkt_ha_private))
168 #define CMD2PKT(cmdp)   ((struct scsi_pkt *)((cmdp)->cmd_pkt))
169 #define EXTCMDS_STATUS_SIZE (sizeof (struct scsi_arq_status))

171 /*
172  * get offset of item in structure
173  */
174 #define MPTSAS_GET_ITEM_OFF(type, member) ((size_t)(&((type *)0)->member))

176 /*
177  * WWID provided by LSI firmware is generated by firmware but the WWID is not
178  * IEEE NAA standard format, OBP has no chance to distinguish format of unit
179  * address. According LSI's confirmation, the top nibble of RAID WWID is
180  * meanless, so the consensus between Solaris and OBP is to replace top nibble
181  * of WWID provided by LSI to "3" always to hint OBP that this is a RAID WWID
182  * format unit address.
183  */
184 #define MPTSAS_RAID_WWID(wwid) \
185         ((wwid & 0x0FFFFFFFFFFFFFFF) | 0x3000000000000000)

187 typedef struct mptsas_target {
188                 uint64_t                m_sas_wwn;      /* hash key1 */
189                 mptsas_phymask_t        m_phymask;      /* hash key2 */
190                 /*
191                  * m_dr_flag is a flag for DR, make sure the member
192                  * take the place of dr_flag of mptsas_hash_data.
193                  */
194                 uint8_t                 m_dr_flag;      /* dr_flag */
195                 uint16_t                m_devhdl;
196                 uint32_t                m_deviceinfo;
197                 uint8_t                 m_phynum;
198                 uint32_t                m_dups;
199                 int32_t                 m_timeout;
200                 int32_t                 m_timebase;
201                 int32_t                 m_t_throttle;
202                 int32_t                 m_t_ncmds;
203                 int32_t                 m_reset_delay;
204                 int32_t                 m_t_nwait;

206                 uint16_t                m_qfull_retry_interval;
207                 uint8_t                 m_qfull_retries;
208                 uint16_t                m_enclosure;
209                 uint16_t                m_slot_num;
210                 uint32_t                m_tgt_unconfigured;
211                 uint32_t                m_timeout_interval;
212                 uint8_t                 m_timeout_count;

214 } mptsas_target_t;
```

_____***unchanged_portion_omitted_***