```
**********************************************************
   62239 Sun Dec  1 10:18:18 2013
new/usr/src/uts/common/io/1394/targets/scsa1394/hba.c
4031 scsa1394 violates DDI scsi_pkt(9S) allocation rules
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */


/*
 * 1394 mass storage HBA driver
 */

#include <sys/param.h>
#include <sys/errno.h>
#include <sys/cred.h>
#include <sys/conf.h>
#include <sys/modctl.h>
#include <sys/stat.h>
#include <sys/byteorder.h>
#include <sys/ddi.h>
#include <sys/sunddi.h>

#include <sys/1394/targets/scsa1394/impl.h>
#include <sys/1394/targets/scsa1394/cmd.h>

/* DDI/DKI entry points */
static int      scsa1394_attach(dev_info_t *, ddi_attach_cmd_t);
static int      scsa1394_detach(dev_info_t *, ddi_detach_cmd_t);
static int      scsa1394_power(dev_info_t *, int, int);
static int      scsa1394_cpr_suspend(dev_info_t *);
static void     scsa1394_cpr_resume(dev_info_t *);

/* configuration routines */
static void     scsa1394_cleanup(scsa1394_state_t *, int);
static int      scsa1394_attach_1394(scsa1394_state_t *);
static void     scsa1394_detach_1394(scsa1394_state_t *);
static int      scsa1394_attach_threads(scsa1394_state_t *);
static void     scsa1394_detach_threads(scsa1394_state_t *);
static int      scsa1394_attach_scsa(scsa1394_state_t *);
static void     scsa1394_detach_scsa(scsa1394_state_t *);
static int      scsa1394_create_cmd_cache(scsa1394_state_t *);
static void     scsa1394_destroy_cmd_cache(scsa1394_state_t *);
static int      scsa1394_add_events(scsa1394_state_t *);
```



```
static void     scsa1394_remove_events(scsa1394_state_t *);

/* device configuration */
static int      scsa1394_scsi_bus_config(dev_info_t *, uint_t,
                ddi_bus_config_op_t, void *, dev_info_t **);
static int      scsa1394_scsi_bus_unconfig(dev_info_t *, uint_t,
                ddi_bus_config_op_t, void *);
static void     scsa1394_create_children(scsa1394_state_t *);
static void     scsa1394_bus_reset(dev_info_t *, ddi_eventcookie_t, void *,
                void *);
static void     scsa1394_disconnect(dev_info_t *, ddi_eventcookie_t, void *,
                void *);
static void     scsa1394_reconnect(dev_info_t *, ddi_eventcookie_t, void *,
                void *);

/* SCSA HBA entry points */
static int      scsa1394_scsi_tgt_init(dev_info_t *, dev_info_t *,
                scsi_hba_tran_t *, struct scsi_device *);
static void     scsa1394_scsi_tgt_free(dev_info_t *, dev_info_t *,
                scsi_hba_tran_t *, struct scsi_device *);
static int      scsa1394_scsi_tgt_probe(struct scsi_device *, int (*)());
static int      scsa1394_probe_g0_nodata(struct scsi_device *, int (*)(),
                uchar_t, uint_t, uint_t);
static int      scsa1394_probe_tran(struct scsi_pkt *);
static struct scsi_pkt *scsa1394_scsi_init_pkt(struct scsi_address *,
                struct scsi_pkt *, struct buf *, int, int, int, int,
                int (*)(), caddr_t arg);
static void     scsa1394_scsi_destroy_pkt(struct scsi_address *,
                struct scsi_pkt *);
static int      scsa1394_scsi_start(struct scsi_address *, struct scsi_pkt *);
static int      scsa1394_scsi_abort(struct scsi_address *, struct scsi_pkt *);
static int      scsa1394_scsi_reset(struct scsi_address *, int);
static int      scsa1394_scsi_getcap(struct scsi_address *, char *, int);
static int      scsa1394_scsi_setcap(struct scsi_address *, char *, int, int);
static void     scsa1394_scsi_dmafree(struct scsi_address *, struct scsi_pkt *);
static void     scsa1394_scsi_sync_pkt(struct scsi_address *,
                struct scsi_pkt *);

/* pkt resource allocation routines */
static int      scsa1394_cmd_cache_constructor(void *, void *, int);
static void     scsa1394_cmd_cache_destructor(void *, void *);
static int      scsa1394_cmd_ext_alloc(scsa1394_state_t *, scsa1394_cmd_t *,
                int);
static void     scsa1394_cmd_ext_free(scsa1394_state_t *, scsa1394_cmd_t *);
static int      scsa1394_cmd_cdb_dma_alloc(scsa1394_state_t *, scsa1394_cmd_t *,
                int, int (*)(), caddr_t);
static void     scsa1394_cmd_cdb_dma_free(scsa1394_state_t *, scsa1394_cmd_t *);
static int      scsa1394_cmd_buf_dma_alloc(scsa1394_state_t *, scsa1394_cmd_t *,
                int, int (*)(), caddr_t, struct buf *);
static void     scsa1394_cmd_buf_dma_free(scsa1394_state_t *, scsa1394_cmd_t *);
static int      scsa1394_cmd_dmac2seg(scsa1394_state_t *, scsa1394_cmd_t *,
                ddi_dma_cookie_t *, uint_t, int);
static void     scsa1394_cmd_seg_free(scsa1394_state_t *, scsa1394_cmd_t *);
static int      scsa1394_cmd_pt_dma_alloc(scsa1394_state_t *, scsa1394_cmd_t *,
                int (*)(), caddr_t, int);
static void     scsa1394_cmd_pt_dma_free(scsa1394_state_t *, scsa1394_cmd_t *);
static int      scsa1394_cmd_buf_addr_alloc(scsa1394_state_t *,
                scsa1394_cmd_t *);
static void     scsa1394_cmd_buf_addr_free(scsa1394_state_t *,
                scsa1394_cmd_t *);
static int      scsa1394_cmd_buf_dma_move(scsa1394_state_t *, scsa1394_cmd_t *);


/* pkt and data transfer routines */
static void     scsa1394_prepare_pkt(scsa1394_state_t *, struct scsi_pkt *);
static void     scsa1394_cmd_fill_cdb(scsa1394_lun_t *, scsa1394_cmd_t *);
```

```
121 static void     scsa1394_cmd_fill_cdb_rbc(scsa1394_lun_t *, scsa1394_cmd_t *);
122 static void     scsa1394_cmd_fill_cdb_other(scsa1394_lun_t *, scsa1394_cmd_t *);
123 static void     scsa1394_cmd_fill_cdb_len(scsa1394_cmd_t *, int);
124 static void     scsa1394_cmd_fill_cdb_lba(scsa1394_cmd_t *, int);
125 static void     scsa1394_cmd_fill_12byte_cdb_len(scsa1394_cmd_t *, int);
126 static void     scsa1394_cmd_fill_read_cd_cdb_len(scsa1394_cmd_t *, int);
127 static int      scsa1394_cmd_read_cd_blk_size(uchar_t);
128 static int      scsa1394_cmd_fake_mode_sense(scsa1394_state_t *,
129                 scsa1394_cmd_t *);
130 static int      scsa1394_cmd_fake_inquiry(scsa1394_state_t *, scsa1394_cmd_t *);
131 static int      scsa1394_cmd_fake_comp(scsa1394_state_t *, scsa1394_cmd_t *);
132 static int      scsa1394_cmd_setup_next_xfer(scsa1394_lun_t *,
133                 scsa1394_cmd_t *);
134 static void     scsa1394_cmd_adjust_cdb(scsa1394_lun_t *, scsa1394_cmd_t *);
135 static void     scsa1394_cmd_status_wrka(scsa1394_lun_t *, scsa1394_cmd_t *);

137 /* other routines */
138 static boolean_t scsa1394_is_my_child(dev_info_t *);
139 static void *   scsa1394_kmem_realloc(void *, int, int, size_t, int);

141 static void     *scsa1394_statep;
142 #define SCSA1394_INST2STATE(inst) (ddi_get_soft_state(scsa1394_statep, inst))

144 static struct cb_ops scsa1394_cb_ops = {
145         nodev,                  /* open */
146         nodev,                  /* close */
147         nodev,                  /* strategy */
148         nodev,                  /* print */
149         nodev,                  /* dump */
150         nodev,                  /* read */
151         nodev,                  /* write */
152         NULL,                   /* ioctl */
153         nodev,                  /* devmap */
154         nodev,                  /* mmap */
155         nodev,                  /* segmap */
156         nochpoll,               /* poll */
157         ddi_prop_op,            /* prop_op */
158         NULL,                   /* stream */
159         D_MP,                   /* cb_flag */
160         CB_REV,                 /* rev */
161         nodev,                  /* aread */
162         nodev                   /* awrite */
163 };
```

*unchanged_portion_omitted_*

```
253 static int
254 scsa1394_attach(dev_info_t *dip, ddi_attach_cmd_t cmd)
255 {
256         int             instance = ddi_get_instance(dip);
257         scsa1394_state_t *sp;

259         switch (cmd) {
260         case DDI_ATTACH:
261                 break;
262         case DDI_RESUME:
263                 scsa1394_cpr_resume(dip);
264                 return (DDI_SUCCESS);
265         default:
266                 return (DDI_FAILURE);
267         }

269         if (ddi_soft_state_zalloc(scsa1394_statep, instance) != 0) {
270                 return (DDI_FAILURE);
271         }
272         sp = SCSA1394_INST2STATE(instance);
```



```
274 #ifndef __lock_lint
275         sp->s_dip = dip;
276         sp->s_instance = instance;
277 #endif
278         mutex_init(&sp->s_mutex, NULL, MUTEX_DRIVER,
279             sp->s_attachinfo.iblock_cookie);
280         cv_init(&sp->s_event_cv, NULL, CV_DRIVER, NULL);

282         if (scsa1394_attach_1394(sp) != DDI_SUCCESS) {
283                 scsa1394_cleanup(sp, 1);
284                 return (DDI_FAILURE);
285         }

287         if (scsa1394_sbp2_attach(sp) != DDI_SUCCESS) {
288                 scsa1394_cleanup(sp, 2);
289                 return (DDI_FAILURE);
290         }

292         if (scsa1394_attach_threads(sp) != DDI_SUCCESS) {
293                 scsa1394_cleanup(sp, 3);
294                 return (DDI_FAILURE);
295         }

297         if (scsa1394_attach_scsa(sp) != DDI_SUCCESS) {
298                 scsa1394_cleanup(sp, 4);
299                 return (DDI_FAILURE);
300         }

302         if (scsa1394_add_events(sp) != DDI_SUCCESS) {
309         if (scsa1394_create_cmd_cache(sp) != DDI_SUCCESS) {
303                 scsa1394_cleanup(sp, 5);
304                 return (DDI_FAILURE);
305         }

314         if (scsa1394_add_events(sp) != DDI_SUCCESS) {
315                 scsa1394_cleanup(sp, 6);
316                 return (DDI_FAILURE);
317         }

307         /* prevent async PM changes until we are done */
308         (void) pm_busy_component(dip, 0);

310         /* Set power to full on */
311         (void) pm_raise_power(dip, 0, PM_LEVEL_D0);

313         /* we are done */
314         (void) pm_idle_component(dip, 0);

316 #ifndef __lock_lint
317         sp->s_dev_state = SCSA1394_DEV_ONLINE;
318 #endif

320         ddi_report_dev(dip);

322         return (DDI_SUCCESS);
323 }
```

*unchanged_portion_omitted_*

```
436 /*
437  *
438  * --- configuration routines
439  *
440  */
441 static void
```

```
 442 scsa1394_cleanup(scsa1394_state_t *sp, int level)
 443 {
 444         ASSERT((level > 0) && (level <= SCSA1394_CLEANUP_LEVEL_MAX));

 446         switch (level) {
 447         default:
 448                 scsa1394_remove_events(sp);
 449                 /* FALLTHRU */
 450         case 5:
 462         case 6:
 451                 scsa1394_detach_scsa(sp);
 452                 /* FALLTHRU */
 465         case 5:
 466                 scsa1394_destroy_cmd_cache(sp);
 467                 /* FALLTHRU */
 453         case 4:
 454                 scsa1394_detach_threads(sp);
 455                 /* FALLTHRU */
 456         case 3:
 457                 scsa1394_sbp2_detach(sp);
 458                 /* FALLTHRU */
 459         case 2:
 460                 scsa1394_detach_1394(sp);
 461                 /* FALLTHRU */
 462         case 1:
 463                 cv_destroy(&sp->s_event_cv);
 464                 mutex_destroy(&sp->s_mutex);
 465                 ddi_soft_state_free(scsa1394_statep, sp->s_instance);
 466         }
 467 }
```

unchanged_portion_omitted_

```
 584 static int
 600 scsa1394_create_cmd_cache(scsa1394_state_t *sp)
 601 {
 602         char    name[64];

 604         (void) sprintf(name, "scsa1394%d_cache", sp->s_instance);
 605         sp->s_cmd_cache = kmem_cache_create(name,
 606             SCSA1394_CMD_SIZE, sizeof (void *),
 607             scsa1394_cmd_cache_constructor, scsa1394_cmd_cache_destructor,
 608             NULL, (void *)sp, NULL, 0);

 610         return ((sp->s_cmd_cache == NULL) ? DDI_FAILURE : DDI_SUCCESS);
 611 }

 613 static void
 614 scsa1394_destroy_cmd_cache(scsa1394_state_t *sp)
 615 {
 616         kmem_cache_destroy(sp->s_cmd_cache);
 617 }

 619 static int
 585 scsa1394_add_events(scsa1394_state_t *sp)
 586 {
 587         ddi_eventcookie_t       br_evc, rem_evc, ins_evc;

 589         if (ddi_get_eventcookie(sp->s_dip, DDI_DEVI_BUS_RESET_EVENT,
 590             &br_evc) != DDI_SUCCESS) {
 591                 return (DDI_FAILURE);
 592         }
 593         if (ddi_add_event_handler(sp->s_dip, br_evc, scsa1394_bus_reset,
 594             sp, &sp->s_reset_cb_id) != DDI_SUCCESS) {
 595                 return (DDI_FAILURE);
 596         }
```



```
 598         if (ddi_get_eventcookie(sp->s_dip, DDI_DEVI_REMOVE_EVENT,
 599             &rem_evc) != DDI_SUCCESS) {
 600                 (void) ddi_remove_event_handler(sp->s_reset_cb_id);
 601                 return (DDI_FAILURE);
 602         }
 603         if (ddi_add_event_handler(sp->s_dip, rem_evc, scsa1394_disconnect,
 604             sp, &sp->s_remove_cb_id) != DDI_SUCCESS) {
 605                 (void) ddi_remove_event_handler(sp->s_reset_cb_id);
 606                 return (DDI_FAILURE);
 607         }

 609         if (ddi_get_eventcookie(sp->s_dip, DDI_DEVI_INSERT_EVENT,
 610             &ins_evc) != DDI_SUCCESS) {
 611                 (void) ddi_remove_event_handler(sp->s_remove_cb_id);
 612                 (void) ddi_remove_event_handler(sp->s_reset_cb_id);
 613                 return (DDI_FAILURE);
 614         }
 615         if (ddi_add_event_handler(sp->s_dip, ins_evc, scsa1394_reconnect,
 616             sp, &sp->s_insert_cb_id) != DDI_SUCCESS) {
 617                 (void) ddi_remove_event_handler(sp->s_remove_cb_id);
 618                 (void) ddi_remove_event_handler(sp->s_reset_cb_id);
 619                 return (DDI_FAILURE);
 620         }

 622         return (DDI_SUCCESS);
 623 }
```

unchanged_portion_omitted_

```
1237 /*
1238  *
1239  * --- pkt resource allocation routines
1240  *
1241  */
1242 static struct scsi_pkt *
1243 scsa1394_scsi_init_pkt(struct scsi_address *ap, struct scsi_pkt *pkt,
1244     struct buf *bp, int cmdlen, int statuslen, int tgtlen, int flags,
1245     int (*callback)(), caddr_t arg)
1246 {
1247         scsa1394_state_t *sp = ADDR2STATE(ap);
1248         scsa1394_lun_t  *lp;
1249         scsa1394_cmd_t  *cmd;
1250         boolean_t       is_new; /* new cmd is being allocated */
1251         int             kf = (callback == SLEEP_FUNC) ? KM_SLEEP : KM_NOSLEEP;

1253         if (ap->a_lun >= sp->s_nluns) {
1254                 return (NULL);
1255         }
1256         lp = &sp->s_lun[ap->a_lun];

1258         /*
1259          * allocate cmd space
1260          */
1261         if (pkt == NULL) {
1262                 is_new = B_TRUE;
1263                 pkt = scsi_hba_pkt_alloc(NULL, ap, max(SCSI_CDB_SIZE, cmdlen),
1264                     statuslen, tgtlen, sizeof (scsa1394_cmd_t), callback, arg);
1265                 if (!pkt)
1298                 if ((cmd = kmem_cache_alloc(sp->s_cmd_cache, kf)) == NULL) {
1266                         return (NULL);
1300                 }

1268                 /* initialize cmd */
1269                 cmd = pkt->pkt_ha_private;
1303                 pkt = &cmd->sc_scsi_pkt;
1304                 pkt->pkt_ha_private     = cmd;
1305                 pkt->pkt_address        = *ap;
```

```
1306                pkt->pkt_private        = cmd->sc_priv;
1307                pkt->pkt_scbp           = (uchar_t *)&cmd->sc_scb;
1308                pkt->pkt_cdbp           = (uchar_t *)&cmd->sc_pkt_cdb;
1309                pkt->pkt_resid          = 0;

1270                cmd->sc_lun = lp;
1271                cmd->sc_pkt = pkt;
1272                cmd->sc_orig_cdblen = cmdlen;
1273                cmd->sc_task.ts_drv_priv = cmd;
1313                cmd->sc_cdb_len         = cmdlen;
1314                cmd->sc_scb_len         = statuslen;
1315                cmd->sc_priv_len        = tgtlen;

1317                /* need external space? */
1318                if ((cmdlen > sizeof (cmd->sc_pkt_cdb)) ||
1319                    (statuslen > sizeof (cmd->sc_scb)) ||
1320                    (tgtlen > sizeof (cmd->sc_priv))) {
1321                        if (scsa1394_cmd_ext_alloc(sp, cmd, kf) !=
1322                            DDI_SUCCESS) {
1323                                kmem_cache_free(sp->s_cmd_cache, cmd);
1324                                lp->l_stat.stat_err_pkt_kmem_alloc++;
1325                                return (NULL);
1326                        }
1327                }

1275                /* allocate DMA resources for CDB */
1276                if (scsa1394_cmd_cdb_dma_alloc(sp, cmd, flags, callback, arg) !=
1277                    DDI_SUCCESS) {
1278                        scsa1394_scsi_destroy_pkt(ap, pkt);
1279                        return (NULL);
1280                }
1281        } else {
1282                is_new = B_FALSE;
1283                cmd = PKT2CMD(pkt);
1284        }

1286        cmd->sc_flags &= ~SCSA1394_CMD_RDWR;

1288        /* allocate/move DMA resources for data buffer */
1289        if ((bp != NULL) && (bp->b_bcount > 0)) {
1290                if ((cmd->sc_flags & SCSA1394_CMD_DMA_BUF_VALID) == 0) {
1291                        if (scsa1394_cmd_buf_dma_alloc(sp, cmd, flags, callback,
1292                            arg, bp) != DDI_SUCCESS) {
1293                                if (is_new) {
1294                                        scsa1394_scsi_destroy_pkt(ap, pkt);
1295                                }
1296                                return (NULL);
1297                        }
1298                } else {
1299                        if (scsa1394_cmd_buf_dma_move(sp, cmd) != DDI_SUCCESS) {
1300                                return (NULL);
1301                        }
1302                }

1304                ASSERT(cmd->sc_win_len > 0);
1305                pkt->pkt_resid = bp->b_bcount - cmd->sc_win_len;
1306        }

1308        /*
1309         * kernel virtual address may be required for certain workarounds
1310         * and in case of B_PHYS or B_PAGEIO, bp_mapin() will get it for us
1311         */
1312        if ((bp != NULL) && ((bp->b_flags & (B_PAGEIO | B_PHYS)) != 0) &&
1313            (bp->b_bcount < SCSA1394_MAPIN_SIZE_MAX) &&
1314            ((cmd->sc_flags & SCSA1394_CMD_DMA_BUF_MAPIN) == 0)) {
1315                bp_mapin(bp);
```



```
1316                cmd->sc_flags |= SCSA1394_CMD_DMA_BUF_MAPIN;
1317        }

1319        return (pkt);
1320 }

1322 static void
1323 scsa1394_scsi_destroy_pkt(struct scsi_address *ap, struct scsi_pkt *pkt)
1324 {
1325        scsa1394_state_t *sp = ADDR2STATE(ap);
1326        scsa1394_cmd_t  *cmd = PKT2CMD(pkt);

1328        if (cmd->sc_flags & SCSA1394_CMD_DMA_BUF_VALID) {
1329                scsa1394_cmd_buf_dma_free(sp, cmd);
1330        }
1331        if (cmd->sc_flags & SCSA1394_CMD_DMA_CDB_VALID) {
1332                scsa1394_cmd_cdb_dma_free(sp, cmd);
1333        }
1334        if (cmd->sc_flags & SCSA1394_CMD_DMA_BUF_MAPIN) {
1335                bp_mapout(cmd->sc_bp);
1336                cmd->sc_flags &= ~SCSA1394_CMD_DMA_BUF_MAPIN;
1337        }
1392        if (cmd->sc_flags & SCSA1394_CMD_EXT) {
1393                scsa1394_cmd_ext_free(sp, cmd);
1394        }

1339        scsi_hba_pkt_free(ap, pkt);
1396        kmem_cache_free(sp->s_cmd_cache, cmd);
1340 }
______unchanged_portion_omitted_

1357 /*ARGSUSED*/
1358 static int
1416 scsa1394_cmd_cache_constructor(void *buf, void *cdrarg, int kf)
1417 {
1418        scsa1394_cmd_t  *cmd = buf;

1420        bzero(buf, SCSA1394_CMD_SIZE);
1421        cmd->sc_task.ts_drv_priv = cmd;

1423        return (0);
1424 }

1426 /*ARGSUSED*/
1427 static void
1428 scsa1394_cmd_cache_destructor(void *buf, void *cdrarg)
1429 {
1430 }

1432 /*
1433  * allocate and deallocate external cmd space (ie. not part of scsa1394_cmd_t)
1434  * for non-standard length cdb, pkt_private, status areas
1435  */
1436 static int
1437 scsa1394_cmd_ext_alloc(scsa1394_state_t *sp, scsa1394_cmd_t *cmd, int kf)
1438 {
1439        struct scsi_pkt *pkt = cmd->sc_pkt;
1440        void            *buf;

1442        if (cmd->sc_cdb_len > sizeof (cmd->sc_pkt_cdb)) {
1443                if ((buf = kmem_zalloc(cmd->sc_cdb_len, kf)) == NULL) {
1444                        return (DDI_FAILURE);
1445                }
1446                pkt->pkt_cdbp = buf;
1447                cmd->sc_flags |= SCSA1394_CMD_CDB_EXT;
1448        }
```

```
1450         if (cmd->sc_scb_len > sizeof (cmd->sc_scb)) {
1451                 if ((buf = kmem_zalloc(cmd->sc_scb_len, kf)) == NULL) {
1452                         scsa1394_cmd_ext_free(sp, cmd);
1453                         return (DDI_FAILURE);
1454                 }
1455                 pkt->pkt_scbp = buf;
1456                 cmd->sc_flags |= SCSA1394_CMD_SCB_EXT;
1457         }

1459         if (cmd->sc_priv_len > sizeof (cmd->sc_priv)) {
1460                 if ((buf = kmem_zalloc(cmd->sc_priv_len, kf)) == NULL) {
1461                         scsa1394_cmd_ext_free(sp, cmd);
1462                         return (DDI_FAILURE);
1463                 }
1464                 pkt->pkt_private = buf;
1465                 cmd->sc_flags |= SCSA1394_CMD_PRIV_EXT;
1466         }

1468         return (DDI_SUCCESS);
1469 }

1471 /*ARGSUSED*/
1472 static void
1473 scsa1394_cmd_ext_free(scsa1394_state_t *sp, scsa1394_cmd_t *cmd)
1474 {
1475         struct scsi_pkt *pkt = cmd->sc_pkt;

1477         if (cmd->sc_flags & SCSA1394_CMD_CDB_EXT) {
1478                 kmem_free(pkt->pkt_cdbp, cmd->sc_cdb_len);
1479         }
1480         if (cmd->sc_flags & SCSA1394_CMD_SCB_EXT) {
1481                 kmem_free(pkt->pkt_scbp, cmd->sc_scb_len);
1482         }
1483         if (cmd->sc_flags & SCSA1394_CMD_PRIV_EXT) {
1484                 kmem_free(pkt->pkt_private, cmd->sc_priv_len);
1485         }
1486         cmd->sc_flags &= ~SCSA1394_CMD_EXT;
1487 }

1489 /*ARGSUSED*/
1490 static int
1359 scsa1394_cmd_cdb_dma_alloc(scsa1394_state_t *sp, scsa1394_cmd_t *cmd,
1360     int flags, int (*callback)(), caddr_t arg)
1361 {
1362         if (sbp2_task_orb_alloc(cmd->sc_lun->l_lun, &cmd->sc_task,
1363             sizeof (scsa1394_cmd_orb_t)) != SBP2_SUCCESS) {
1364                 return (DDI_FAILURE);
1365         }

1367         cmd->sc_flags |= SCSA1394_CMD_DMA_CDB_VALID;
1368         return (DDI_SUCCESS);
1369 }
```

_____unchanged_portion_omitted_

```
1909 static void
1910 scsa1394_cmd_fill_cdb(scsa1394_lun_t *lp, scsa1394_cmd_t *cmd)
1911 {
2044         cmd->sc_cdb_actual_len = cmd->sc_cdb_len;

1912         mutex_enter(&lp->l_mutex);

1914         switch (lp->l_dtype_orig) {
1915         case DTYPE_DIRECT:
1916         case DTYPE_RODIRECT:
1917         case DTYPE_OPTICAL:
```



```
1918         case SCSA1394_DTYPE_RBC:
1919                 scsa1394_cmd_fill_cdb_rbc(lp, cmd);
1920                 break;
1921         default:
1922                 scsa1394_cmd_fill_cdb_other(lp, cmd);
1923                 break;
1924         }

1926         mutex_exit(&lp->l_mutex);
1927 }

1929 static void
1930 scsa1394_cmd_fill_cdb_rbc(scsa1394_lun_t *lp, scsa1394_cmd_t *cmd)
1931 {
1932         scsa1394_state_t *sp = lp->l_sp;
1933         struct scsi_pkt *pkt = CMD2PKT(cmd);
1934         int             lba, opcode;
1935         struct buf      *bp = cmd->sc_bp;
1936         size_t          len;
1937         size_t          blk_size;
1938         int             sz;

1940         opcode = pkt->pkt_cdbp[0];
1941         blk_size  = lp->l_lba_size;

1943         switch (opcode) {
1944         case SCMD_READ:
1945                 /* RBC only supports 10-byte read/write */
1946                 lba = SCSA1394_LBA_6BYTE(pkt);
1947                 len = SCSA1394_LEN_6BYTE(pkt);
1948                 opcode = SCMD_READ_G1;
1949                 cmd->sc_orig_cdblen = CDB_GROUP1;
2083                 cmd->sc_cdb_actual_len = CDB_GROUP1;
1950                 break;
1951         case SCMD_WRITE:
1952                 lba = SCSA1394_LBA_6BYTE(pkt);
1953                 len = SCSA1394_LEN_6BYTE(pkt);
1954                 opcode = SCMD_WRITE_G1;
1955                 cmd->sc_orig_cdblen = CDB_GROUP1;
2089                 cmd->sc_cdb_actual_len = CDB_GROUP1;
1956                 break;
1957         case SCMD_READ_G1:
1958         case SCMD_READ_LONG:
1959                 lba = SCSA1394_LBA_10BYTE(pkt);
1960                 len = SCSA1394_LEN_10BYTE(pkt);
1961                 break;
1962         case SCMD_WRITE_G1:
1963         case SCMD_WRITE_LONG:
1964                 lba = SCSA1394_LBA_10BYTE(pkt);
1965                 len = SCSA1394_LEN_10BYTE(pkt);
1966                 if ((lp->l_dtype_orig == DTYPE_RODIRECT) &&
1967                     (bp != NULL) && (len != 0)) {
1968                         sz = SCSA1394_CDRW_BLKSZ(bp->b_bcount, len);
1969                         if (SCSA1394_VALID_CDRW_BLKSZ(sz)) {
1970                                 blk_size = sz;
1971                         }
1972                 }
1973                 break;
1974         case SCMD_READ_CD:
1975                 lba = SCSA1394_LBA_10BYTE(pkt);
1976                 len = SCSA1394_LEN_READ_CD(pkt);
1977                 blk_size = scsa1394_cmd_read_cd_blk_size(pkt->pkt_cdbp[1] >> 2);
1978                 break;
1979         case SCMD_READ_G5:
1980                 lba = SCSA1394_LBA_12BYTE(pkt);
1981                 len = SCSA1394_LEN_12BYTE(pkt);
```

```
1982                 break;
1983         case SCMD_WRITE_G5:
1984                 lba = SCSA1394_LBA_12BYTE(pkt);
1985                 len = SCSA1394_LEN_12BYTE(pkt);
1986                 break;
1987         default:
1988                 /* no special mapping for other commands */
1989                 scsa1394_cmd_fill_cdb_other(lp, cmd);
1990                 return;
1991         }
1992         cmd->sc_blk_size = blk_size;

1994         /* limit xfer length for Symbios workaround */
1995         if (sp->s_symbios && (len * blk_size > scsa1394_symbios_size_max)) {
1996                 cmd->sc_flags |= SCSA1394_CMD_SYMBIOS_BREAKUP;

1998                 cmd->sc_total_blks = cmd->sc_resid_blks = len;

2000                 len = scsa1394_symbios_size_max / blk_size;
2001         }
2002         cmd->sc_xfer_blks = len;
2003         cmd->sc_xfer_bytes = len * blk_size;

2005         /* finalize new CDB */
2006         switch (pkt->pkt_cdbp[0]) {
2007         case SCMD_READ:
2008         case SCMD_WRITE:
2009                 /*
2010                  * We rewrite READ/WRITE G0 commands as READ/WRITE G1.
2011                  * Build new cdb from scatch.
2012                  * The lba and length fields is updated below.
2013                  */
2014                 bzero(pkt->pkt_cdbp, cmd->sc_orig_cdblen);
2148                 bzero(cmd->sc_cdb, cmd->sc_cdb_actual_len);
2015                 break;
2016         default:
2151                 /*
2152                  * Copy the non lba/len fields.
2153                  * The lba and length fields is updated below.
2154                  */
2155                 bcopy(pkt->pkt_cdbp, cmd->sc_cdb, cmd->sc_cdb_actual_len);
2017                 break;
2018         }

2020         pkt->pkt_cdbp[0] = (uchar_t)opcode;
2159         cmd->sc_cdb[0] = (uchar_t)opcode;
2021         scsa1394_cmd_fill_cdb_lba(cmd, lba);
2022         switch (opcode) {
2023         case SCMD_READ_CD:
2024                 scsa1394_cmd_fill_read_cd_cdb_len(cmd, len);
2025                 break;
2026         case SCMD_WRITE_G5:
2027         case SCMD_READ_G5:
2028                 scsa1394_cmd_fill_12byte_cdb_len(cmd, len);
2029                 break;
2030         default:
2031                 scsa1394_cmd_fill_cdb_len(cmd, len);
2032                 break;
2033         }
2034 }

2036 /*ARGSUSED*/
2037 static void
2038 scsa1394_cmd_fill_cdb_other(scsa1394_lun_t *lp, scsa1394_cmd_t *cmd)
2039 {
2179         struct scsi_pkt *pkt = CMD2PKT(cmd);
```



```
2040         cmd->sc_xfer_bytes = cmd->sc_win_len;
2041         cmd->sc_xfer_blks = cmd->sc_xfer_bytes / lp->l_lba_size;
2042         cmd->sc_total_blks = cmd->sc_xfer_blks;
2043         cmd->sc_lba = 0;

2186         bcopy(pkt->pkt_cdbp, cmd->sc_cdb, cmd->sc_cdb_len);
2044 }

2046 /*
2047  * fill up parts of CDB
2048  */
2049 static void
2050 scsa1394_cmd_fill_cdb_len(scsa1394_cmd_t *cmd, int len)
2051 {
2052         struct scsi_pkt *pkt = CMD2PKT(cmd);

2054         pkt->pkt_cdbp[7] = len >> 8;
2055         pkt->pkt_cdbp[8] = (uchar_t)len;
2195         cmd->sc_cdb[7] = len >> 8;
2196         cmd->sc_cdb[8] = (uchar_t)len;
2056 }

2058 static void
2059 scsa1394_cmd_fill_cdb_lba(scsa1394_cmd_t *cmd, int lba)
2060 {
2061         struct scsi_pkt *pkt = CMD2PKT(cmd);

2063         pkt->pkt_cdbp[2] = lba >> 24;
2064         pkt->pkt_cdbp[3] = lba >> 16;
2065         pkt->pkt_cdbp[4] = lba >> 8;
2066         pkt->pkt_cdbp[5] = (uchar_t)lba;
2202         cmd->sc_cdb[2] = lba >> 24;
2203         cmd->sc_cdb[3] = lba >> 16;
2204         cmd->sc_cdb[4] = lba >> 8;
2205         cmd->sc_cdb[5] = (uchar_t)lba;
2067         cmd->sc_lba = lba;
2068 }

2070 static void
2071 scsa1394_cmd_fill_12byte_cdb_len(scsa1394_cmd_t *cmd, int len)
2072 {
2073         struct scsi_pkt *pkt = CMD2PKT(cmd);

2075         pkt->pkt_cdbp[6] = len >> 24;
2076         pkt->pkt_cdbp[7] = len >> 16;
2077         pkt->pkt_cdbp[8] = len >> 8;
2078         pkt->pkt_cdbp[9] = (uchar_t)len;
2212         cmd->sc_cdb[6] = len >> 24;
2213         cmd->sc_cdb[7] = len >> 16;
2214         cmd->sc_cdb[8] = len >> 8;
2215         cmd->sc_cdb[9] = (uchar_t)len;
2079 }

2081 static void
2082 scsa1394_cmd_fill_read_cd_cdb_len(scsa1394_cmd_t *cmd, int len)
2083 {
2084         struct scsi_pkt *pkt = CMD2PKT(cmd);

2086         pkt->pkt_cdbp[6] = len >> 16;
2087         pkt->pkt_cdbp[7] = len >> 8;
2088         pkt->pkt_cdbp[8] = (uchar_t)len;
2221         cmd->sc_cdb[6] = len >> 16;
2222         cmd->sc_cdb[7] = len >> 8;
2223         cmd->sc_cdb[8] = (uchar_t)len;
2089 }
```

*_____unchanged_portion_omitted_*

```
2297 /*
2298  * new lba = current lba + previous xfer len
2299  */
2300 /*ARGSUSED*/
2301 static void
2302 scsa1394_cmd_adjust_cdb(scsa1394_lun_t *lp, scsa1394_cmd_t *cmd)
2303 {
2304         int             len;

2306         ASSERT(cmd->sc_flags & SCSA1394_CMD_SYMBIOS_BREAKUP);

2308         cmd->sc_lba += cmd->sc_xfer_blks;
2309         len = cmd->sc_resid_blks;

2311         /* limit xfer length for Symbios workaround */
2312         if (len * cmd->sc_blk_size > scsa1394_symbios_size_max) {
2313                 len = scsa1394_symbios_size_max / cmd->sc_blk_size;
2314         }

2316         switch (cmd->sc_pkt->pkt_cdbp[0]) {
2451         switch (cmd->sc_cdb[0]) {
2317         case SCMD_READ_CD:
2318                 scsa1394_cmd_fill_read_cd_cdb_len(cmd, len);
2319                 break;
2320         case SCMD_WRITE_G5:
2321         case SCMD_READ_G5:
2322                 scsa1394_cmd_fill_12byte_cdb_len(cmd, len);
2323                 break;
2324         case SCMD_WRITE_G1:
2325         case SCMD_WRITE_LONG:
2326         default:
2327                 scsa1394_cmd_fill_cdb_len(cmd, len);
2328         }

2330         scsa1394_cmd_fill_cdb_lba(cmd, cmd->sc_lba);

2332         cmd->sc_xfer_blks = len;
2333         cmd->sc_xfer_bytes = len * cmd->sc_blk_size;
2334 }
```

*unchanged_portion_omitted_*

```
**********************************************************
   25314 Sun Dec  1 10:18:22 2013
new/usr/src/uts/common/io/1394/targets/scsa1394/sbp2_driver.c
4031 scsa1394 violates DDI scsi_pkt(9S) allocation rules
**********************************************************
______unchanged_portion_omitted_

 544 /*
 545  * convert command into DMA-mapped SBP-2 ORB
 546  */
 547 void
 548 scsa1394_sbp2_cmd2orb(scsa1394_lun_t *lp, scsa1394_cmd_t *cmd)
 549 {
 550         scsa1394_state_t *sp = lp->l_sp;
 551         scsa1394_cmd_orb_t *orb = sbp2_task_orb_kaddr(&cmd->sc_task);

 553         mutex_enter(&lp->l_mutex);

 555         lp->l_stat.stat_cmd_cnt++;

 557         bzero(orb->co_cdb, sizeof (orb->co_cdb));

 559         /* CDB */
 560         bcopy(cmd->sc_pkt->pkt_cdbp, orb->co_cdb, cmd->sc_orig_cdblen);
 560         bcopy(cmd->sc_cdb, orb->co_cdb, cmd->sc_cdb_actual_len);

 562         /*
 563          * ORB parameters
 564          *
 565          * use max speed and max payload for this speed.
 566          * max async data transfer for a given speed is 512<<speed
 567          * SBP-2 defines (see 5.1.2) max data transfer as 2^(max_payload+2),
 568          * hence max_payload = 7 + speed
 569          */
 570         orb->co_params = SBP2_ORB_NOTIFY | SBP2_ORB_RQ_FMT_SBP2 |
 571             (sp->s_targetinfo.current_max_speed << SBP2_ORB_CMD_SPD_SHIFT) |
 572             ((7 + sp->s_targetinfo.current_max_speed -
 573             scsa1394_sbp2_max_payload_sub) << SBP2_ORB_CMD_MAX_PAYLOAD_SHIFT);

 575         /* direction: initiator's read is target's write (and vice versa) */
 576         if (cmd->sc_flags & SCSA1394_CMD_READ) {
 577                 orb->co_params |= SBP2_ORB_CMD_DIR;
 578         }

 580         /*
 581          * data_size and data_descriptor
 582          */
 583         if (cmd->sc_buf_nsegs == 0) {
 584                 /* no data */
 585                 orb->co_data_size = 0;
 586                 SCSA1394_ADDR_SET(sp, orb->co_data_descr, 0);
 587         } else if (cmd->sc_buf_nsegs == 1) {
 588                 /* contiguous buffer - use direct addressing */
 589                 ASSERT(cmd->sc_buf_seg[0].ss_len != 0);
 590                 orb->co_data_size = SBP2_SWAP16(cmd->sc_buf_seg[0].ss_len);
 591                 SCSA1394_ADDR_SET(sp, orb->co_data_descr,
 592                     cmd->sc_buf_seg[0].ss_baddr);
 593         } else {
 594                 /* non-contiguous s/g list - page table */
 595                 ASSERT(cmd->sc_pt_cmd_size > 0);
 596                 orb->co_params |= SBP2_ORB_CMD_PT;
 597                 orb->co_data_size = SBP2_SWAP16(cmd->sc_pt_cmd_size);
 598                 SCSA1394_ADDR_SET(sp, orb->co_data_descr, cmd->sc_pt_baddr);
 599         }

 601         SBP2_SWAP16_1(orb->co_params);
```



```
 603         SBP2_ORBP_SET(orb->co_next_orb, SBP2_ORBP_NULL);

 605         mutex_exit(&lp->l_mutex);

 607         sbp2_task_orb_sync(lp->l_lun, &cmd->sc_task, DDI_DMA_SYNC_FORDEV);
 608 }
______unchanged_portion_omitted_

 745 static void
 746 scsa1394_sbp2_status_proc(scsa1394_lun_t *lp, scsa1394_cmd_t *cmd,
 747     scsa1394_status_t *st)
 748 {
 749         sbp2_task_t             *task = CMD2TASK(cmd);
 750         struct scsi_pkt         *pkt = CMD2PKT(cmd);
 751         uint64_t                *p;

 753         if (cmd->sc_flags & SCSA1394_CMD_READ) {
 754                 (void) ddi_dma_sync(cmd->sc_buf_dma_hdl, 0, 0,
 755                     DDI_DMA_SYNC_FORKERNEL);
 756         }

 758         if (task->ts_error != SBP2_TASK_ERR_NONE) {
 759                 pkt->pkt_state |= STATE_GOT_BUS;
 760                 switch (task->ts_error) {
 761                 case SBP2_TASK_ERR_ABORT:
 762                         pkt->pkt_state |= STATE_GOT_TARGET;
 763                         pkt->pkt_reason = CMD_ABORTED;
 764                         break;
 765                 case SBP2_TASK_ERR_LUN_RESET:
 766                         pkt->pkt_state |= STATE_GOT_TARGET;
 767                         pkt->pkt_reason = CMD_RESET;
 768                         pkt->pkt_statistics |= STAT_DEV_RESET;
 769                         break;
 770                 case SBP2_TASK_ERR_TGT_RESET:
 771                         pkt->pkt_state |= STATE_GOT_TARGET;
 772                         pkt->pkt_reason = CMD_RESET;
 773                         pkt->pkt_statistics |= STAT_DEV_RESET;
 774                         break;
 775                 case SBP2_TASK_ERR_TIMEOUT:
 776                         (void) scsa1394_sbp2_reset(lp, RESET_TARGET, cmd);
 777                         return;
 778                 case SBP2_TASK_ERR_DEAD:
 779                 case SBP2_TASK_ERR_BUS:
 780                 default:
 781                         pkt->pkt_reason = CMD_TRAN_ERR;
 782                         break;
 783                 }
 784         } else if ((st->st_param & SBP2_ST_RESP) == SBP2_ST_RESP_COMPLETE) {
 785                 /*
 786                  * SBP-2 status block has been received, now look at sbp_status.
 787                  *
 788                  * Note: ANSI NCITS 325-1998 B.2 requires that when status is
 789                  * GOOD, length must be one, but some devices do not comply
 790                  */
 791                 if (st->st_sbp_status == SBP2_ST_SBP_DUMMY_ORB) {
 792                         pkt->pkt_state |= (STATE_GOT_BUS | STATE_GOT_TARGET);
 793                         pkt->pkt_reason = CMD_ABORTED;
 794                         pkt->pkt_statistics |= STAT_DEV_RESET;
 795                 } else if ((st->st_status & SCSA1394_ST_STATUS) ==
 796                     STATUS_GOOD) {
 797                         /* request complete */
 798                         *(pkt->pkt_scbp) = STATUS_GOOD;
 799                         pkt->pkt_state |= (STATE_GOT_BUS | STATE_GOT_TARGET |
 800                             STATE_SENT_CMD | STATE_XFERRED_DATA |
 801                             STATE_GOT_STATUS);
```

```
802                         pkt->pkt_reason = CMD_CMPLT;
803                 } else if (scsa1394_sbp2_conv_status(cmd, st) == DDI_SUCCESS) {
804                         pkt->pkt_state |= (STATE_GOT_BUS | STATE_GOT_TARGET |
805                             STATE_SENT_CMD | STATE_XFERRED_DATA |
806                             STATE_GOT_STATUS | STATE_ARQ_DONE);
807                         pkt->pkt_reason = CMD_TRAN_ERR;
808                 } else {
809                         pkt->pkt_state |= (STATE_GOT_BUS | STATE_GOT_TARGET |
810                             STATE_SENT_CMD | STATE_XFERRED_DATA |
811                             STATE_GOT_STATUS);
812                         pkt->pkt_reason = CMD_TRAN_ERR;
813                         lp->l_stat.stat_err_status_conv++;
814                 }
815         } else {
816                 /* transport or serial bus failure */
817                 pkt->pkt_state |= (STATE_GOT_BUS | STATE_GOT_TARGET);
818                 pkt->pkt_reason = CMD_TRAN_ERR;
819                 lp->l_stat.stat_err_status_resp++;
820         }

822         if (pkt->pkt_reason == CMD_TRAN_ERR) {
823                 lp->l_stat.stat_err_status_tran_err++;

825                 /* save the command */
826                 p = &lp->l_stat.stat_cmd_last_fail[
827                     lp->l_stat.stat_cmd_last_fail_idx][0];
828                 bcopy(&pkt->pkt_cdbp[0], p, min(cmd->sc_pkt->pkt_cdblen, 16));
828                 bcopy(&pkt->pkt_cdbp[0], p, min(cmd->sc_cdb_len, 16));
829                 *(clock_t *)&p[2] = ddi_get_lbolt();
830                 lp->l_stat.stat_cmd_last_fail_idx =
831                     (lp->l_stat.stat_cmd_last_fail_idx + 1) %
832                     SCSA1394_STAT_NCMD_LAST;
833         }

835         /* generic HBA status processing */
836         scsa1394_cmd_status_proc(lp, cmd);
837 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    4109 Sun Dec  1 10:18:25 2013
new/usr/src/uts/common/sys/1394/targets/scsa1394/cmd.h
4031 scsa1394 violates DDI scsi_pkt(9S) allocation rules
**********************************************************
/*
 * 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 2008 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#ifndef _SYS_1394_TARGETS_SCSA1394_CMD_H
#define _SYS_1394_TARGETS_SCSA1394_CMD_H

#pragma ident   "%Z%%M% %I%     %E% SMI"

/*
 * scsa1394 command
 */

#include <sys/scsi/scsi_types.h>
#include <sys/1394/targets/scsa1394/sbp2.h>
#include <sys/note.h>

#ifdef  __cplusplus
extern "C" {
#endif

/* preferred pkt_private length in 64-bit quantities */
#ifdef  _LP64
#define SCSA1394_CMD_PRIV_SIZE  2
#else /* _ILP32 */
#define SCSA1394_CMD_PRIV_SIZE  1
#endif
#define SCSA1394_CMD_PRIV_LEN   (SCSA1394_CMD_PRIV_SIZE * sizeof (uint64_t))

/* entry describing a page table segment */
typedef struct scsa1394_cmd_seg {
        size_t                  ss_len;
        uint64_t                ss_daddr;
        uint64_t                ss_baddr;
        t1394_addr_handle_t     ss_addr_hdl;
} scsa1394_cmd_seg_t;

/* command packet structure */
typedef struct scsa1394_cmd {
        sbp2_task_t             sc_task;        /* corresponding SBP-2 task */
```



```
        struct scsa1394_lun     *sc_lun;        /* lun it belongs to */
        int                     sc_state;       /* command state */
        int                     sc_flags;       /* command flags */
        struct buf              *sc_bp;         /* data buffer */
        struct scsi_pkt         *sc_pkt;        /* corresponding scsi pkt */
        size_t                  sc_orig_cdblen;
        size_t                  sc_cdb_len;
        size_t                  sc_cdb_actual_len;
        size_t                  sc_scb_len;
        size_t                  sc_priv_len;
        uchar_t                 sc_cdb[SCSI_CDB_SIZE];
        uchar_t                 sc_pkt_cdb[SCSI_CDB_SIZE];
        struct scsi_arq_status  sc_scb;
        uint64_t                sc_priv[SCSA1394_CMD_PRIV_SIZE];
        clock_t                 sc_start_time;
        int                     sc_timeout;

        /* DMA: command ORB */
        ddi_dma_handle_t        sc_orb_dma_hdl;
        ddi_acc_handle_t        sc_orb_acc_hdl;
        ddi_dma_cookie_t        sc_orb_dmac;
        t1394_addr_handle_t     sc_orb_addr_hdl;

        /* DMA: data buffer */
        ddi_dma_handle_t        sc_buf_dma_hdl;
        uint_t                  sc_buf_nsegs;   /* # of segments/cookies */
        uint_t                  sc_buf_nsegs_alloc; /* # of entries allocated */
        scsa1394_cmd_seg_t      *sc_buf_seg;    /* segment array */
        scsa1394_cmd_seg_t      sc_buf_seg_mem; /* backstore for one segment */
        uint_t                  sc_nwin;        /* # windows */
        uint_t                  sc_curwin;      /* current window */
        off_t                   sc_win_offset;  /* current window offset */
        size_t                  sc_win_len;     /* current window length */
        size_t                  sc_xfer_bytes;  /* current xfer byte count */
        size_t                  sc_xfer_blks;   /* current xfer blk count */

        /* DMA: page table */
        ddi_dma_handle_t        sc_pt_dma_hdl;
        ddi_acc_handle_t        sc_pt_acc_hdl;
        ddi_dma_cookie_t        sc_pt_dmac;
        caddr_t                 sc_pt_kaddr;
        uint64_t                sc_pt_baddr;
        t1394_addr_handle_t     sc_pt_addr_hdl;
        size_t                  sc_pt_ent_alloc; /* # allocated entries */
        int                     sc_pt_cmd_size;

        /* for symbios mode only */
        int                     sc_lba;         /* start LBA */
        int                     sc_blk_size;    /* xfer block size */
        size_t                  sc_total_blks;  /* total xfer blocks */
        size_t                  sc_resid_blks;  /* blocks left */

        struct scsi_pkt         *sc_scsi_pkt;   /* must be last */
                                                /* embedded SCSI packet */
                                                /* ... scsi_pkt_size() */
} scsa1394_cmd_t;
#define SCSA1394_CMD_SIZE       (sizeof (struct scsa1394_cmd) - \
                                sizeof (struct scsi_pkt) + scsi_pkt_size())

_NOTE(SCHEME_PROTECTS_DATA("unique per task", { scsa1394_cmd scsa1394_cmd_seg
    scsi_pkt scsi_inquiry scsi_extended_sense scsi_cdb scsi_arq_status }))

#define PKT2CMD(pktp)   ((scsa1394_cmd_t *)((pktp)->pkt_ha_private))
#define CMD2PKT(cmdp)   ((cmdp)->sc_pkt)
#define CMD2PKT(cmdp)   ((struct scsi_pkt *)((cmdp)->sc_pkt))
#define TASK2CMD(task)  ((scsa1394_cmd_t *)(task)->ts_drv_priv)
```

```
103 #define CMD2TASK(cmdp)  ((sbp2_task_t *)&(cmdp)->sc_task)

105 /* state */
106 enum {
107         SCSA1394_CMD_INIT,
108         SCSA1394_CMD_START,
109         SCSA1394_CMD_STATUS
110 };

112 /* flags */
113 enum {
137         SCSA1394_CMD_CDB_EXT            = 0x0001,
138         SCSA1394_CMD_PRIV_EXT           = 0x0002,
139         SCSA1394_CMD_SCB_EXT            = 0x0004,
140         SCSA1394_CMD_EXT                = (SCSA1394_CMD_CDB_EXT |
141                                             SCSA1394_CMD_PRIV_EXT |
142                                             SCSA1394_CMD_SCB_EXT),

114         SCSA1394_CMD_DMA_CDB_VALID      = 0x0008,
115         SCSA1394_CMD_DMA_BUF_BIND_VALID = 0x0010,
116         SCSA1394_CMD_DMA_BUF_PT_VALID   = 0x0020,
117         SCSA1394_CMD_DMA_BUF_ADDR_VALID = 0x0040,
118         SCSA1394_CMD_DMA_BUF_VALID      = (SCSA1394_CMD_DMA_BUF_BIND_VALID |
119                                             SCSA1394_CMD_DMA_BUF_ADDR_VALID |
120                                             SCSA1394_CMD_DMA_BUF_PT_VALID),
121         SCSA1394_CMD_DMA_BUF_MAPIN      = 0x0080,

123         SCSA1394_CMD_READ               = 0x0100,
124         SCSA1394_CMD_WRITE              = 0x0200,
125         SCSA1394_CMD_RDWR               = (SCSA1394_CMD_READ |
126                                             SCSA1394_CMD_WRITE),

128         SCSA1394_CMD_SYMBIOS_BREAKUP    = 0x400
129 };
```

*unchanged_portion_omitted_*

```
**********************************************************
   11205 Sun Dec  1 10:18:29 2013
new/usr/src/uts/common/sys/1394/targets/scsa1394/impl.h
4031 scsa1394 violates DDI scsi_pkt(9S) allocation rules
**********************************************************
```

```
   1 /*
   2  * CDDL HEADER START
   3  *
   4  * The contents of this file are subject to the terms of the
   5  * Common Development and Distribution License (the "License").
   6  * You may not use this file except in compliance with the License.
   7  *
   8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9  * or http://www.opensolaris.org/os/licensing.
  10  * See the License for the specific language governing permissions
  11  * and limitations under the License.
  12  *
  13  * When distributing Covered Code, include this CDDL HEADER in each
  14  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15  * If applicable, add the following below this CDDL HEADER, with the
  16  * fields enclosed by brackets "[]" replaced with your own identifying
  17  * information: Portions Copyright [yyyy] [name of copyright owner]
  18  *
  19  * CDDL HEADER END
  20  */
  21 /*
  22  * Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
  23  * Use is subject to license terms.
  24  */

  26 #ifndef _SYS_1394_TARGETS_SCSA1394_IMPL_H
  27 #define _SYS_1394_TARGETS_SCSA1394_IMPL_H

  29 #pragma ident   "%Z%%M% %I%     %E% SMI"

  29 /*
  30  * scsa1394 definitions
  31  */

  33 #include <sys/1394/t1394.h>
  34 #include <sys/sbp2/driver.h>
  35 #include <sys/scsi/scsi.h>
  36 #include <sys/cdio.h>
  37 #include <sys/1394/targets/scsa1394/cmd.h>

  39 #ifdef  __cplusplus
  40 extern "C" {
  41 #endif

  43 /*
  44  * each lun uses a worker thread for various deferred processing
  45  */
  46 typedef enum {
  47         SCSA1394_THR_INIT,                      /* initial state */
  48         SCSA1394_THR_RUN,                       /* thread is running */
  49         SCSA1394_THR_EXIT                       /* thread exited */
  50 } scsa1394_thr_state_t;
```

_____unchanged_portion_omitted_

```
 167 /* per-instance soft state structure */
 168 typedef struct scsa1394_state {
 169         kmutex_t                s_mutex;        /* structure mutex */
 170         dev_info_t              *s_dip;         /* device information */
 171         int                     s_instance;     /* instance number */
 172         scsa1394_dev_state_t    s_dev_state;    /* device state */
 173         t1394_handle_t          s_t1394_hdl;    /* 1394 handle */
```



```
 174         t1394_attachinfo_t      s_attachinfo;   /* 1394 attach info */
 175         t1394_targetinfo_t      s_targetinfo;   /* 1394 target info */
 176         ddi_callback_id_t       s_reset_cb_id;  /* reset event cb id */
 177         ddi_callback_id_t       s_remove_cb_id; /* remove event cb id */
 178         ddi_callback_id_t       s_insert_cb_id; /* insert event cb id */
 179         boolean_t               s_event_entered; /* event serialization */
 180         kcondvar_t              s_event_cv;     /* event serialization cv */
 181         ddi_dma_attr_t          s_buf_dma_attr; /* data buffer DMA attrs */
 182         ddi_dma_attr_t          s_pt_dma_attr;  /* page table DMA attrs */
 183         scsi_hba_tran_t         *s_tran;        /* SCSA HBA tran structure */
 184         sbp2_tgt_t              *s_tgt;         /* SBP-2 target */
 185         sbp2_cfgrom_t           *s_cfgrom;      /* Config ROM */
 186         int                     s_nluns;        /* # of logical units */
 187         scsa1394_lun_t          *s_lun;         /* logical units */
 190         kmem_cache_t            *s_cmd_cache;   /* command kmem cache */
 188         ddi_taskq_t             *s_taskq;       /* common taskq for all luns */
 189         boolean_t               s_symbios;      /* need Symbios workaround? */
 190         boolean_t               s_disconnect_warned; /* disconnect warning */
 191         size_t                  s_totalsec;     /* total sectors */
 192         size_t                  s_secsz;        /* sector size */
 193         scsa1394_inst_stat_t    s_stat;         /* statistics */
 194 } scsa1394_state_t;
```

_____unchanged_portion_omitted_