```
**********************************************************
   98729 Thu Jan 19 20:31:23 2012
new/usr/src/uts/common/io/comstar/port/fct/fct.c
968 fct driver sets incorrect fc-ct revision
**********************************************************
```

```
/*
 * 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) 2008, 2010, Oracle and/or its affiliates. All rights reserved.
 * Copyright 2012 Nexenta Systems, Inc. All rights reserved.
 */

#include <sys/conf.h>
#include <sys/file.h>
#include <sys/ddi.h>
#include <sys/sunddi.h>
#include <sys/modctl.h>
#include <sys/scsi/scsi.h>
#include <sys/scsi/impl/scsi_reset_notify.h>
#include <sys/disp.h>
#include <sys/byteorder.h>
#include <sys/varargs.h>
#include <sys/atomic.h>
#include <sys/sdt.h>

#include <sys/stmf.h>
#include <sys/stmf_ioctl.h>
#include <sys/portif.h>
#include <sys/fct.h>
#include <sys/fctio.h>

#include "fct_impl.h"
#include "discovery.h"

static int fct_attach(dev_info_t *dip, ddi_attach_cmd_t cmd);
static int fct_detach(dev_info_t *dip, ddi_detach_cmd_t cmd);
static int fct_getinfo(dev_info_t *dip, ddi_info_cmd_t cmd, void *arg,
    void **result);
static int fct_open(dev_t *devp, int flag, int otype, cred_t *credp);
static int fct_close(dev_t dev, int flag, int otype, cred_t *credp);
static int fct_ioctl(dev_t dev, int cmd, intptr_t data, int mode,
    cred_t *credp, int *rval);
static int fct_fctiocmd(intptr_t data, int mode);
void fct_init_kstats(fct_i_local_port_t *iport);

static dev_info_t *fct_dip;
static struct cb_ops fct_cb_ops = {
        fct_open,                       /* open */
```



```
        fct_close,                      /* close */
        nodev,                          /* strategy */
        nodev,                          /* print */
        nodev,                          /* dump */
        nodev,                          /* read */
        nodev,                          /* write */
        fct_ioctl,                      /* ioctl */
        nodev,                          /* devmap */
        nodev,                          /* mmap */
        nodev,                          /* segmap */
        nochpoll,                       /* chpoll */
        ddi_prop_op,                    /* cb_prop_op */
        0,                              /* streamtab */
        D_NEW | D_MP,                   /* cb_flag */
        CB_REV,                         /* rev */
        nodev,                          /* aread */
        nodev                           /* awrite */
};
```

*unchanged_portion_omitted_*

```
fct_cmd_t *
fct_create_solct(fct_local_port_t *port, fct_remote_port_t *query_rp,
    uint16_t ctop, fct_icmd_cb_t icmdcb)
{
        fct_cmd_t               *cmd     = NULL;
        fct_i_cmd_t             *icmd    = NULL;
        fct_sol_ct_t            *ct      = NULL;
        uint8_t                 *p       = NULL;
        fct_i_remote_port_t     *irp     = NULL;
        fct_i_local_port_t      *iport   = NULL;
        char                    *nname   = NULL;
        int                      namelen = 0;

        /*
         * Allocate space
         */
        cmd = fct_alloc(FCT_STRUCT_CMD_SOL_CT,
            port->port_fca_sol_ct_private_size, 0);
        if (!cmd) {
                return (NULL);
        }

        /*
         * We should have PLOGIed to the name server (0xFFFFFC)
         * Caution: this irp is not query_rp->rp_fct_private.
         */
        irp = fct_portid_to_portptr((fct_i_local_port_t *)
            port->port_fct_private, FS_NAME_SERVER);
        if (irp == NULL) {
                stmf_trace(PORT_TO_IPORT(port)->iport_alias,
                    "fct_create_solct: Must PLOGI name server first");
                fct_free(cmd);
                return (NULL);
        }

        cmd->cmd_port      = port;
        cmd->cmd_rp        = irp->irp_rp;
        cmd->cmd_rp_handle = irp->irp_rp->rp_handle;
        cmd->cmd_rportid   = irp->irp_rp->rp_id;
        cmd->cmd_lportid   = (PORT_TO_IPORT(port))->iport_link_info.portid;
        cmd->cmd_oxid      = PTR2INT(cmd, uint16_t);
        cmd->cmd_rxid      = 0xFFFF;
        cmd->cmd_handle    = 0;
        icmd               = CMD_TO_ICMD(cmd);
        ct                 = ICMD_TO_CT(icmd);
        icmd->icmd_cb      = icmdcb;
```

```
2537         iport              = ICMD_TO_IPORT(icmd);

2539         switch (ctop) {
2540         case NS_GSNN_NN:
2541                 /*
2542                  * Allocate max space for its sybolic name
2543                  */
2544                 ct->ct_resp_alloc_size = ct->ct_resp_size = 272;
2545                 ct->ct_resp_payload = (uint8_t *)kmem_zalloc(ct->ct_resp_size,
2546                     KM_SLEEP);

2548                 ct->ct_req_size = ct->ct_req_alloc_size = 24;
2549                 p = ct->ct_req_payload = (uint8_t *)kmem_zalloc(ct->ct_req_size,
2550                     KM_SLEEP);

2552                 bcopy(query_rp->rp_nwwn, p + 16, 8);
2553                 break;

2555         case NS_RNN_ID:
2556                 ct->ct_resp_alloc_size = ct->ct_resp_size = 16;
2557                 ct->ct_resp_payload = (uint8_t *)kmem_zalloc(ct->ct_resp_size,
2558                     KM_SLEEP);
2559                 ct->ct_req_size = ct->ct_req_alloc_size = 28;
2560                 p = ct->ct_req_payload = (uint8_t *)kmem_zalloc(ct->ct_req_size,
2561                     KM_SLEEP);

2563                 /*
2564                  * Port Identifier
2565                  */
2566                 p[17] = (iport->iport_link_info.portid >> 16) & 0xFF;
2567                 p[18] = (iport->iport_link_info.portid >>  8) & 0xFF;
2568                 p[19] = (iport->iport_link_info.portid >>  0) & 0xFF;

2570                 /*
2571                  * Node Name
2572                  */
2573                 bcopy(port->port_nwwn, p + 20, 8);
2574                 break;

2576         case NS_RCS_ID:
2577                 ct->ct_resp_alloc_size = ct->ct_resp_size = 16;
2578                 ct->ct_resp_payload = (uint8_t *)kmem_zalloc(ct->ct_resp_size,
2579                     KM_SLEEP);
2580                 ct->ct_req_size = ct->ct_req_alloc_size = 24;
2581                 p = ct->ct_req_payload = (uint8_t *)kmem_zalloc(ct->ct_req_size,
2582                     KM_SLEEP);

2584                 /*
2585                  * Port Identifier
2586                  */
2587                 p[17] = (iport->iport_link_info.portid >> 16) & 0xFF;
2588                 p[18] = (iport->iport_link_info.portid >>  8) & 0xFF;
2589                 p[19] = (iport->iport_link_info.portid >>  0) & 0xFF;

2591                 /*
2592                  * Class of Service
2593                  */
2594                 *(p + 23) = FC_NS_CLASS3;
2595                 break;

2597         case NS_RFT_ID:
2598                 ct->ct_resp_alloc_size = ct->ct_resp_size = 16;
2599                 ct->ct_resp_payload = (uint8_t *)kmem_zalloc(ct->ct_resp_size,
2600                     KM_SLEEP);
2601                 ct->ct_req_size = ct->ct_req_alloc_size = 52;
2602                 p = ct->ct_req_payload = (uint8_t *)kmem_zalloc(ct->ct_req_size,
```



```
2603                     KM_SLEEP);

2605                 /*
2606                  * Port Identifier
2607                  */
2608                 p[17] = (iport->iport_link_info.portid >> 16) & 0xFF;
2609                 p[18] = (iport->iport_link_info.portid >>  8) & 0xFF;
2610                 p[19] = (iport->iport_link_info.portid >>  0) & 0xFF;

2612                 /*
2613                  * FC-4 Protocol Types
2614                  */
2615                 *(p + 22) = 0x1;        /* 0x100 */
2616                 break;

2618         case NS_RSPN_ID:
2619                 /*
2620                  * If we get here, port->port_sym_port_name is always not NULL.
2621                  */
2622                 ASSERT(port->port_sym_port_name);
2623                 namelen = strlen(port->port_sym_port_name);
2624                 ct->ct_resp_alloc_size = ct->ct_resp_size = 16;
2625                 ct->ct_resp_payload = (uint8_t *)kmem_zalloc(ct->ct_resp_size,
2626                     KM_SLEEP);
2627                 ct->ct_req_size = ct->ct_req_alloc_size =
2628                     (21 + namelen + 3) & ~3;
2629                 p = ct->ct_req_payload = (uint8_t *)kmem_zalloc(ct->ct_req_size,
2630                     KM_SLEEP);

2632                 /*
2633                  * Port Identifier
2634                  */
2635                 p[17] = (iport->iport_link_info.portid >> 16) & 0xFF;
2636                 p[18] = (iport->iport_link_info.portid >>  8) & 0xFF;
2637                 p[19] = (iport->iport_link_info.portid >>  0) & 0xFF;

2639                 /*
2640                  * String length
2641                  */
2642                 p[20] = namelen;

2644                 /*
2645                  * Symbolic port name
2646                  */
2647                 bcopy(port->port_sym_port_name, p + 21, ct->ct_req_size - 21);
2648                 break;

2650         case NS_RSNN_NN:
2651                 namelen = port->port_sym_node_name == NULL ?
2652                     strlen(utsname.nodename) :
2653                     strlen(port->port_sym_node_name);
2654                 nname = port->port_sym_node_name == NULL ?
2655                     utsname.nodename : port->port_sym_node_name;

2657                 ct->ct_resp_alloc_size = ct->ct_resp_size = 16;
2658                 ct->ct_resp_payload = (uint8_t *)kmem_zalloc(ct->ct_resp_size,
2659                     KM_SLEEP);
2660                 ct->ct_req_size = ct->ct_req_alloc_size =
2661                     (25 + namelen + 3) & ~3;
2662                 p = ct->ct_req_payload = (uint8_t *)kmem_zalloc(ct->ct_req_size,
2663                     KM_SLEEP);

2665                 /*
2666                  * Node name
2667                  */
2668                 bcopy(port->port_nwwn, p + 16, 8);
```

```
2670                /*
2671                 * String length
2672                 */
2673                p[24] = namelen;

2675                /*
2676                 * Symbolic node name
2677                 */
2678                bcopy(nname, p + 25, ct->ct_req_size - 25);
2679                break;

2681        case NS_GSPN_ID:
2682                ct->ct_resp_alloc_size = ct->ct_resp_size = 272;
2683                ct->ct_resp_payload = (uint8_t *)kmem_zalloc(ct->ct_resp_size,
2684                    KM_SLEEP);
2685                ct->ct_req_size = ct->ct_req_alloc_size = 20;
2686                p = ct->ct_req_payload = (uint8_t *)kmem_zalloc(ct->ct_req_size,
2687                    KM_SLEEP);
2688                /*
2689                 * Port Identifier
2690                 */
2691                p[17] = (query_rp->rp_id >> 16) & 0xFF;
2692                p[18] = (query_rp->rp_id >>  8) & 0xFF;
2693                p[19] = (query_rp->rp_id >>  0) & 0xFF;
2694                break;

2696        case NS_GCS_ID:
2697                ct->ct_resp_alloc_size = ct->ct_resp_size = 20;
2698                ct->ct_resp_payload = (uint8_t *)kmem_zalloc(ct->ct_resp_size,
2699                    KM_SLEEP);
2700                ct->ct_req_size = ct->ct_req_alloc_size = 20;
2701                p = ct->ct_req_payload = (uint8_t *)kmem_zalloc(ct->ct_req_size,
2702                    KM_SLEEP);
2703                /*
2704                 * Port Identifier
2705                 */
2706                p[17] = (query_rp->rp_id >> 16) & 0xFF;
2707                p[18] = (query_rp->rp_id >>  8) & 0xFF;
2708                p[19] = (query_rp->rp_id >>  0) & 0xFF;
2709                break;

2711        case NS_GFT_ID:
2712                ct->ct_resp_alloc_size = ct->ct_resp_size = 48;
2713                ct->ct_resp_payload = (uint8_t *)kmem_zalloc(ct->ct_resp_size,
2714                    KM_SLEEP);
2715                ct->ct_req_size = ct->ct_req_alloc_size = 20;
2716                p = ct->ct_req_payload = (uint8_t *)kmem_zalloc(ct->ct_req_size,
2717                    KM_SLEEP);
2718                /*
2719                 * Port Identifier
2720                 */
2721                p[17] = (query_rp->rp_id >> 16) & 0xFF;
2722                p[18] = (query_rp->rp_id >>  8) & 0xFF;
2723                p[19] = (query_rp->rp_id >>  0) & 0xFF;
2724                break;

2726        case NS_GID_PN:
2727                ct->ct_resp_alloc_size = ct->ct_resp_size = 20;
2728                ct->ct_resp_payload = (uint8_t *)kmem_zalloc(ct->ct_resp_size,
2729                    KM_SLEEP);

2731                ct->ct_req_size = ct->ct_req_alloc_size = 24;
2732                p = ct->ct_req_payload = (uint8_t *)kmem_zalloc(ct->ct_req_size,
2733                    KM_SLEEP);
```



```
2735                bcopy(query_rp->rp_pwwn, p + 16, 8);
2736                break;

2738        default:
2739                /* CONSTCOND */
2740                ASSERT(0);
2741        }

2743        FCT_FILL_CTIU_PREAMBLE(p, ctop);
2742        FCT_FILL_CTIU_PREAMPLE(p, ctop);
2744        return (cmd);
2745 }
______unchanged_portion_omitted_
```

```
**********************************************************
   12667 Thu Jan 19 20:31:24 2012
new/usr/src/uts/common/sys/fct.h
968 fct driver sets incorrect fc-ct revision
**********************************************************
```

```
   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 (c) 2008, 2010, Oracle and/or its affiliates. All rights reserved.
  23  * Copyright 2012 Nexenta Systems, Inc. All rights reserved.
  24  */
  25 #ifndef _FCT_H
  26 #define _FCT_H

  28 /*
  29  * Definitions for common FC Target.
  30  */
  31 #include <sys/note.h>
  32 #include <sys/stmf_defines.h>
  33 #include <sys/fct_defines.h>
  34 #include <sys/portif.h>

  36 #ifdef  __cplusplus
  37 extern "C" {
  38 #endif

  40 typedef enum fct_struct_id {
  41         FCT_STRUCT_LOCAL_PORT = 1,
  42         FCT_STRUCT_REMOTE_PORT,
  43         FCT_STRUCT_CMD_RCVD_ELS,
  44         FCT_STRUCT_CMD_SOL_ELS,
  45         FCT_STRUCT_CMD_SOL_CT,
  46         FCT_STRUCT_CMD_RCVD_ABTS,
  47         FCT_STRUCT_CMD_FCP_XCHG,
  48         FCT_STRUCT_DBUF_STORE,

  50         FCT_MAX_STRUCT_IDS
  51 } fct_struct_id_t;
______unchanged_portion_omitted_

 317 /*
 318  * port topology
 319  */
 320 #define PORT_TOPOLOGY_UNKNOWN           0
 321 #define PORT_TOPOLOGY_PT_TO_PT          1
 322 #define PORT_TOPOLOGY_PRIVATE_LOOP      2
 323 #define PORT_TOPOLOGY_PUBLIC_LOOP       6
 324 #define PORT_TOPOLOGY_FABRIC_PT_TO_PT   5
```



```
 325 #define PORT_TOPOLOGY_FABRIC_BIT        4

 327 #define PORT_FLOGI_DONE(li)     (((li)->port_fca_flogi_done) || \
 328                                         ((li)->port_fct_flogi_done))

 330 /*
 331  * port speed
 332  */
 333 #define PORT_SPEED_UNKNOWN              0
 334 #define PORT_SPEED_1G                   1
 335 #define PORT_SPEED_2G                   2
 336 #define PORT_SPEED_4G                   4
 337 #define PORT_SPEED_8G                   8
 338 #define PORT_SPEED_10G                  16

 340 /*
 341  * Abort commands
 342  */
 343 #define FCT_TERMINATE_CMD               1

 345 /*
 346  * FCT port states.
 347  */
 348 #define FCT_STATE_OFFLINE       0
 349 #define FCT_STATE_ONLINING      1
 350 #define FCT_STATE_ONLINE        2
 351 #define FCT_STATE_OFFLINING     3

 353 /*
 354  * fct ctl commands. These should not conflict with stmf ctl commands
 355  */
 356 #define FCT_CMD_PORT_ONLINE             (STMF_LPORT_CTL_CMDS | 0x01)
 357 #define FCT_CMD_PORT_ONLINE_COMPLETE    (STMF_LPORT_CTL_CMDS | 0x02)
 358 #define FCT_CMD_PORT_OFFLINE            (STMF_LPORT_CTL_CMDS | 0x03)
 359 #define FCT_CMD_PORT_OFFLINE_COMPLETE   (STMF_LPORT_CTL_CMDS | 0x04)
 360 #define FCT_ACK_PORT_ONLINE_COMPLETE    (STMF_LPORT_CTL_CMDS | 0x05)
 361 #define FCT_ACK_PORT_OFFLINE_COMPLETE   (STMF_LPORT_CTL_CMDS | 0x06)
 362 #define FCT_CMD_FORCE_LIP               (STMF_LPORT_CTL_CMDS | 0x07)

 364 /*
 365  * IO flags for cmd flow.
 366  */
 367 #define FCT_IOF_FCA_DONE                0x10000
 368 #define FCT_IOF_FORCE_FCA_DONE          0x20000

 370 /*
 371  * Fill CTIU preamble
 370  * Fill CTIU preample
 372  */
 373 #ifdef  lint
 374 #define FCT_FILL_CTIU_PREAMBLE(x_payload, x_ctop)       _NOTE(EMPTY)
 373 #define FCT_FILL_CTIU_PREAMPLE(x_payload, x_ctop)       _NOTE(EMPTY)
 375 #else
 376 #define FCT_FILL_CTIU_PREAMBLE(x_payload, x_ctop)       \
 375 #define FCT_FILL_CTIU_PREAMPLE(x_payload, x_ctop)       \
 377         do {                                            \
 378                 x_payload[0] = 0x01;                    \
 377                 x_payload[0] = 0x02;                    \
 379                 x_payload[4] = 0xFC;                    \
 380                 x_payload[5] = 0x02;                    \
 381                 x_payload[8] = 0xFF & (x_ctop >> 8);    \
 382                 x_payload[9] = 0xFF & (x_ctop);         \
 383         } while (0)
 384 #endif

 386 uint64_t fct_netbuf_to_value(uint8_t *buf, uint8_t nbytes);
```

```
387 void fct_value_to_netbuf(uint64_t value, uint8_t *buf, uint8_t nbytes);
388 void *fct_alloc(fct_struct_id_t struct_id, int additional_size, int flags);
389 void fct_free(void *ptr);
390 fct_cmd_t *fct_scsi_task_alloc(struct fct_local_port *port,
391     uint16_t rp_handle, uint32_t rportid, uint8_t *lun,
392     uint16_t cdb_length, uint16_t task_ext);
393 fct_status_t fct_register_local_port(fct_local_port_t *port);
394 fct_status_t fct_deregister_local_port(fct_local_port_t *port);
395 void fct_handle_event(fct_local_port_t *port, int event_id,
396     uint32_t event_flags, caddr_t arg);
397 void fct_post_rcvd_cmd(fct_cmd_t *cmd, stmf_data_buf_t *dbuf);
398 void fct_queue_cmd_for_termination(fct_cmd_t *cmd, fct_status_t s);
399 void fct_queue_scsi_task_for_termination(fct_cmd_t *cmd, fct_status_t s);
400 fct_cmd_t *fct_handle_to_cmd(fct_local_port_t *port, uint32_t fct_handle);
401 void fct_ctl(struct stmf_local_port *lport, int cmd, void *arg);
402 void fct_cmd_fca_aborted(fct_cmd_t *cmd, fct_status_t s, uint32_t ioflags);
403 uint16_t fct_get_rp_handle(fct_local_port_t *port, uint32_t rportid);
404 void fct_send_response_done(fct_cmd_t *cmd, fct_status_t s, uint32_t ioflags);
405 void fct_send_cmd_done(fct_cmd_t *cmd, fct_status_t s, uint32_t ioflags);
406 void fct_scsi_data_xfer_done(fct_cmd_t *cmd, stmf_data_buf_t *dbuf,
407     uint32_t ioflags);
408 fct_status_t fct_port_initialize(fct_local_port_t *port, uint32_t rflags,
409     char *additional_info);
410 fct_status_t fct_port_shutdown(fct_local_port_t *port, uint32_t rflags,
411     char *additional_info);
412 fct_status_t fct_handle_rcvd_flogi(fct_local_port_t *port,
413     fct_flogi_xchg_t *fx);
414 void fct_log_local_port_event(fct_local_port_t *port, char *subclass);
415 void fct_log_remote_port_event(fct_local_port_t *port, char *subclass,
416     uint8_t *rp_pwwn, uint32_t rp_id);
417 void fct_wwn_to_str(char *to_ptr, const uint8_t *from_ptr);

419 #ifdef  __cplusplus
420 }
```

*unchanged_portion_omitted_*