```
**********************************************************
  911463 Mon May  5 11:11:15 2014
new/usr/src/uts/common/io/scsi/targets/sd.c
4781 sd shouldn't abuse ddi_get_time(9f)
Reviewed by: Richard Elling <richard.elling@gmail.com>
Reviewed by: Robert Mustacchi <rm@joyent.com>
**********************************************************
```

```
/*
 * 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) 1990, 2010, Oracle and/or its affiliates. All rights reserved.
 */
/*
 * Copyright (c) 2011 Bayard G. Bell.  All rights reserved.
 * Copyright (c) 2012 by Delphix. All rights reserved.
 * Copyright 2014 Nexenta Systems, Inc.  All rights reserved.
 * Copyright 2013 Nexenta Systems, Inc.  All rights reserved.
 * Copyright 2012 DEY Storage Systems, Inc.  All rights reserved.
 */
/*
 * Copyright 2011 cyril.galibern@opensvc.com
 */

/*
 * SCSI disk target driver.
 */
#include <sys/scsi/scsi.h>
#include <sys/dkbad.h>
#include <sys/dklabel.h>
#include <sys/dkio.h>
#include <sys/fdio.h>
#include <sys/cdio.h>
#include <sys/mhd.h>
#include <sys/vtoc.h>
#include <sys/dktp/fdisk.h>
#include <sys/kstat.h>
#include <sys/vtrace.h>
#include <sys/note.h>
#include <sys/thread.h>
#include <sys/proc.h>
#include <sys/efi_partition.h>
#include <sys/var.h>
#include <sys/aio_req.h>

#ifdef __lock_lint
#define _LP64
#define __amd64
```



```
#endif

#if (defined(__fibre))
/* Note: is there a leadville version of the following? */
#include <sys/fc4/fcal_linkapp.h>
#endif
#include <sys/taskq.h>
#include <sys/uuid.h>
#include <sys/byteorder.h>
#include <sys/sdt.h>

#include "sd_xbuf.h"

#include <sys/scsi/targets/sddef.h>
#include <sys/cmlb.h>
#include <sys/sysevent/eventdefs.h>
#include <sys/sysevent/dev.h>

#include <sys/fm/protocol.h>

/*
 * Loadable module info.
 */
#if (defined(__fibre))
#define SD_MODULE_NAME  "SCSI SSA/FCAL Disk Driver"
#else /* !__fibre */
#define SD_MODULE_NAME  "SCSI Disk Driver"
#endif /* !__fibre */

/*
 * Define the interconnect type, to allow the driver to distinguish
 * between parallel SCSI (sd) and fibre channel (ssd) behaviors.
 *
 * This is really for backward compatibility. In the future, the driver
 * should actually check the "interconnect-type" property as reported by
 * the HBA; however at present this property is not defined by all HBAs,
 * so we will use this #define (1) to permit the driver to run in
 * backward-compatibility mode; and (2) to print a notification message
 * if an FC HBA does not support the "interconnect-type" property.  The
 * behavior of the driver will be to assume parallel SCSI behaviors unless
 * the "interconnect-type" property is defined by the HBA **AND** has a
 * value of either INTERCONNECT_FIBRE, INTERCONNECT_SSA, or
 * INTERCONNECT_FABRIC, in which case the driver will assume Fibre
 * Channel behaviors (as per the old ssd).  (Note that the
 * INTERCONNECT_1394 and INTERCONNECT_USB types are not supported and
 * will result in the driver assuming parallel SCSI behaviors.)
 *
 * (see common/sys/scsi/impl/services.h)
 *
 * Note: For ssd semantics, don't use INTERCONNECT_FABRIC as the default
 * since some FC HBAs may already support that, and there is some code in
 * the driver that already looks for it.  Using INTERCONNECT_FABRIC as the
 * default would confuse that code, and besides things should work fine
 * anyways if the FC HBA already reports INTERCONNECT_FABRIC for the
 * "interconnect_type" property.
 *
 */
#if (defined(__fibre))
#define SD_DEFAULT_INTERCONNECT_TYPE    SD_INTERCONNECT_FIBRE
#else
#define SD_DEFAULT_INTERCONNECT_TYPE    SD_INTERCONNECT_PARALLEL
#endif

/*
 * The name of the driver, established from the module name in _init.
 */
```

```
125 static  char *sd_label                 = NULL;

127 /*
128  * Driver name is unfortunately prefixed on some driver.conf properties.
129  */
130 #if (defined(__fibre))
131 #define sd_max_xfer_size              ssd_max_xfer_size
132 #define sd_config_list                ssd_config_list
133 static  char *sd_max_xfer_size        = "ssd_max_xfer_size";
134 static  char *sd_config_list          = "ssd-config-list";
135 #else
136 static  char *sd_max_xfer_size        = "sd_max_xfer_size";
137 static  char *sd_config_list          = "sd-config-list";
138 #endif

140 /*
141  * Driver global variables
142  */

144 #if (defined(__fibre))
145 /*
146  * These #defines are to avoid namespace collisions that occur because this
147  * code is currently used to compile two separate driver modules: sd and ssd.
148  * All global variables need to be treated this way (even if declared static)
149  * in order to allow the debugger to resolve the names properly.
150  * It is anticipated that in the near future the ssd module will be obsoleted,
151  * at which time this namespace issue should go away.
152  */
153 #define sd_state                      ssd_state
154 #define sd_io_time                    ssd_io_time
155 #define sd_failfast_enable            ssd_failfast_enable
156 #define sd_ua_retry_count             ssd_ua_retry_count
157 #define sd_report_pfa                 ssd_report_pfa
158 #define sd_max_throttle               ssd_max_throttle
159 #define sd_min_throttle               ssd_min_throttle
160 #define sd_rot_delay                  ssd_rot_delay

162 #define sd_retry_on_reservation_conflict        \
163                                       ssd_retry_on_reservation_conflict
164 #define sd_reinstate_resv_delay       ssd_reinstate_resv_delay
165 #define sd_resv_conflict_name         ssd_resv_conflict_name

167 #define sd_component_mask             ssd_component_mask
168 #define sd_level_mask                 ssd_level_mask
169 #define sd_debug_un                   ssd_debug_un
170 #define sd_error_level                ssd_error_level

172 #define sd_xbuf_active_limit          ssd_xbuf_active_limit
173 #define sd_xbuf_reserve_limit         ssd_xbuf_reserve_limit

175 #define sd_tr                         ssd_tr
176 #define sd_reset_throttle_timeout     ssd_reset_throttle_timeout
177 #define sd_qfull_throttle_timeout     ssd_qfull_throttle_timeout
178 #define sd_qfull_throttle_enable      ssd_qfull_throttle_enable
179 #define sd_check_media_time           ssd_check_media_time
180 #define sd_wait_cmds_complete         ssd_wait_cmds_complete
181 #define sd_label_mutex                ssd_label_mutex
182 #define sd_detach_mutex               ssd_detach_mutex
183 #define sd_log_buf                    ssd_log_buf
184 #define sd_log_mutex                  ssd_log_mutex

186 #define sd_disk_table                 ssd_disk_table
187 #define sd_disk_table_size            ssd_disk_table_size
188 #define sd_sense_mutex                ssd_sense_mutex
189 #define sd_cdbtab                     ssd_cdbtab
```



```
191 #define sd_cb_ops                     ssd_cb_ops
192 #define sd_ops                        ssd_ops
193 #define sd_additional_codes           ssd_additional_codes
194 #define sd_tgops                      ssd_tgops

196 #define sd_minor_data                 ssd_minor_data
197 #define sd_minor_data_efi             ssd_minor_data_efi

199 #define sd_tq                         ssd_tq
200 #define sd_wmr_tq                     ssd_wmr_tq
201 #define sd_taskq_name                 ssd_taskq_name
202 #define sd_wmr_taskq_name             ssd_wmr_taskq_name
203 #define sd_taskq_minalloc             ssd_taskq_minalloc
204 #define sd_taskq_maxalloc             ssd_taskq_maxalloc

206 #define sd_dump_format_string         ssd_dump_format_string

208 #define sd_iostart_chain              ssd_iostart_chain
209 #define sd_iodone_chain               ssd_iodone_chain

211 #define sd_pm_idletime                ssd_pm_idletime

213 #define sd_force_pm_supported         ssd_force_pm_supported

215 #define sd_dtype_optical_bind         ssd_dtype_optical_bind

217 #define sd_ssc_init                   ssd_ssc_init
218 #define sd_ssc_send                   ssd_ssc_send
219 #define sd_ssc_fini                   ssd_ssc_fini
220 #define sd_ssc_assessment             ssd_ssc_assessment
221 #define sd_ssc_post                   ssd_ssc_post
222 #define sd_ssc_print                  ssd_ssc_print
223 #define sd_ssc_ereport_post           ssd_ssc_ereport_post
224 #define sd_ssc_set_info               ssd_ssc_set_info
225 #define sd_ssc_extract_info           ssd_ssc_extract_info

227 #endif

229 #ifdef  SDDEBUG
230 int     sd_force_pm_supported         = 0;
231 #endif  /* SDDEBUG */

233 void *sd_state                        = NULL;
234 int sd_io_time                        = SD_IO_TIME;
235 int sd_failfast_enable                = 1;
236 int sd_ua_retry_count                 = SD_UA_RETRY_COUNT;
237 int sd_report_pfa                     = 1;
238 int sd_max_throttle                   = SD_MAX_THROTTLE;
239 int sd_min_throttle                   = SD_MIN_THROTTLE;
240 int sd_rot_delay                      = 4; /* Default 4ms Rotation delay */
241 int sd_qfull_throttle_enable          = TRUE;

243 int sd_retry_on_reservation_conflict  = 1;
244 int sd_reinstate_resv_delay           = SD_REINSTATE_RESV_DELAY;
245 _NOTE(SCHEME_PROTECTS_DATA("safe sharing", sd_reinstate_resv_delay))

247 static int sd_dtype_optical_bind      = -1;

249 /* Note: the following is not a bug, it really is "sd_" and not "ssd_" */
250 static  char *sd_resv_conflict_name   = "sd_retry_on_reservation_conflict";

252 /*
253  * Global data for debug logging. To enable debug printing, sd_component_mask
254  * and sd_level_mask should be set to the desired bit patterns as outlined in
255  * sddef.h.
256  */
```

```
 257 uint_t  sd_component_mask               = 0x0;
 258 uint_t  sd_level_mask                   = 0x0;
 259 struct  sd_lun *sd_debug_un             = NULL;
 260 uint_t  sd_error_level                  = SCSI_ERR_RETRYABLE;

 262 /* Note: these may go away in the future... */
 263 static uint32_t sd_xbuf_active_limit    = 512;
 264 static uint32_t sd_xbuf_reserve_limit   = 16;

 266 static struct sd_resv_reclaim_request   sd_tr = { NULL, NULL, NULL, 0, 0, 0 };

 268 /*
 269  * Timer value used to reset the throttle after it has been reduced
 270  * (typically in response to TRAN_BUSY or STATUS_QFULL)
 271  */
 272 static int sd_reset_throttle_timeout    = SD_RESET_THROTTLE_TIMEOUT;
 273 static int sd_qfull_throttle_timeout    = SD_QFULL_THROTTLE_TIMEOUT;

 275 /*
 276  * Interval value associated with the media change scsi watch.
 277  */
 278 static int sd_check_media_time          = 3000000;

 280 /*
 281  * Wait value used for in progress operations during a DDI_SUSPEND
 282  */
 283 static int sd_wait_cmds_complete        = SD_WAIT_CMDS_COMPLETE;

 285 /*
 286  * sd_label_mutex protects a static buffer used in the disk label
 287  * component of the driver
 288  */
 289 static kmutex_t sd_label_mutex;

 291 /*
 292  * sd_detach_mutex protects un_layer_count, un_detach_count, and
 293  * un_opens_in_progress in the sd_lun structure.
 294  */
 295 static kmutex_t sd_detach_mutex;

 297 _NOTE(MUTEX_PROTECTS_DATA(sd_detach_mutex,
 298         sd_lun::{un_layer_count un_detach_count un_opens_in_progress}))

 300 /*
 301  * Global buffer and mutex for debug logging
 302  */
 303 static char     sd_log_buf[1024];
 304 static kmutex_t sd_log_mutex;

 306 /*
 307  * Structs and globals for recording attached lun information.
 308  * This maintains a chain. Each node in the chain represents a SCSI controller.
 309  * The structure records the number of luns attached to each target connected
 310  * with the controller.
 311  * For parallel scsi device only.
 312  */
 313 struct sd_scsi_hba_tgt_lun {
 314         struct sd_scsi_hba_tgt_lun      *next;
 315         dev_info_t                      *pdip;
 316         int                             nlun[NTARGETS_WIDE];
 317 };
```

*unchanged_portion_omitted_*

```
6523 /*
6524  *    Function: sd_pm_idletimeout_handler
```



```
6525  *
6526  * Description: A timer routine that's active only while a device is busy.
6527  *              The purpose is to extend slightly the pm framework's busy
6528  *              view of the device to prevent busy/idle thrashing for
6529  *              back-to-back commands. Do this by comparing the current time
6530  *              to the time at which the last command completed and when the
6531  *              difference is greater than sd_pm_idletime, call
6532  *              pm_idle_component. In addition to indicating idle to the pm
6533  *              framework, update the chain type to again use the internal pm
6534  *              layers of the driver.
6535  *
6536  *   Arguments: arg - driver soft state (unit) structure
6537  *
6538  *     Context: Executes in a timeout(9F) thread context
6539  */

6541 static void
6542 sd_pm_idletimeout_handler(void *arg)
6543 {
6544         const hrtime_t idletime = sd_pm_idletime * NANOSEC;
6545 #endif /* ! codereview */
6546         struct sd_lun *un = arg;

6544         time_t  now;

6548         mutex_enter(&sd_detach_mutex);
6549         if (un->un_detach_count != 0) {
6550                 /* Abort if the instance is detaching */
6551                 mutex_exit(&sd_detach_mutex);
6552                 return;
6553         }
6554         mutex_exit(&sd_detach_mutex);

6554         now = ddi_get_time();
6556         /*
6557          * Grab both mutexes, in the proper order, since we're accessing
6558          * both PM and softstate variables.
6559          */
6560         mutex_enter(SD_MUTEX(un));
6561         mutex_enter(&un->un_pm_mutex);
6562         if (((gethrtime() - un->un_pm_idle_time) > idletime) &&
6561         if (((now - un->un_pm_idle_time) > sd_pm_idletime) &&
6563             (un->un_ncmds_in_driver == 0) && (un->un_pm_count == 0)) {
6564                 /*
6565                  * Update the chain types.
6566                  * This takes affect on the next new command received.
6567                  */
6568                 if (un->un_f_non_devbsize_supported) {
6569                         un->un_buf_chain_type = SD_CHAIN_INFO_RMMEDIA;
6570                 } else {
6571                         un->un_buf_chain_type = SD_CHAIN_INFO_DISK;
6572                 }
6573                 un->un_uscsi_chain_type = SD_CHAIN_INFO_USCSI_CMD;

6575                 SD_TRACE(SD_LOG_IO_PM, un,
6576                     "sd_pm_idletimeout_handler: idling device\n");
6577                 (void) pm_idle_component(SD_DEVINFO(un), 0);
6578                 un->un_pm_idle_timeid = NULL;
6579         } else {
6580                 un->un_pm_idle_timeid =
6581                     timeout(sd_pm_idletimeout_handler, un,
6582                     (drv_usectohz((clock_t)300000))); /* 300 ms. */
6583         }
6584         mutex_exit(&un->un_pm_mutex);
6585         mutex_exit(SD_MUTEX(un));
6586 }
```

*unchanged_portion_omitted_*

```
12430 /*
12431  *    Function: sd_buf_iodone
12432  *
12433  * Description: Frees the sd_xbuf & returns the buf to its originator.
12434  *
12435  *     Context: May be called from interrupt context.
12436  */
12437 /* ARGSUSED */
12438 static void
12439 sd_buf_iodone(int index, struct sd_lun *un, struct buf *bp)
12440 {
12441         struct sd_xbuf *xp;

12443         ASSERT(un != NULL);
12444         ASSERT(bp != NULL);
12445         ASSERT(!mutex_owned(SD_MUTEX(un)));

12447         SD_TRACE(SD_LOG_IO_CORE, un, "sd_buf_iodone: entry.\n");

12449         xp = SD_GET_XBUF(bp);
12450         ASSERT(xp != NULL);

12452         /* xbuf is gone after this */
12453         if (ddi_xbuf_done(bp, un->un_xbuf_attr)) {
12454                 mutex_enter(SD_MUTEX(un));

12456                 /*
12457                  * Grab time when the cmd completed.
12458                  * This is used for determining if the system has been
12459                  * idle long enough to make it idle to the PM framework.
12460                  * This is for lowering the overhead, and therefore improving
12461                  * performance per I/O operation.
12462                  */
12463                 un->un_pm_idle_time = gethrtime();
12462                 un->un_pm_idle_time = ddi_get_time();

12465                 un->un_ncmds_in_driver--;
12466                 ASSERT(un->un_ncmds_in_driver >= 0);
12467                 SD_INFO(SD_LOG_IO, un,
12468                     "sd_buf_iodone: un_ncmds_in_driver = %ld\n",
12469                     un->un_ncmds_in_driver);

12471                 mutex_exit(SD_MUTEX(un));
12472         }

12474         biodone(bp);                            /* bp is gone after this */

12476         SD_TRACE(SD_LOG_IO_CORE, un, "sd_buf_iodone: exit.\n");
12477 }


12480 /*
12481  *    Function: sd_uscsi_iodone
12482  *
12483  * Description: Frees the sd_xbuf & returns the buf to its originator.
12484  *
12485  *     Context: May be called from interrupt context.
12486  */
12487 /* ARGSUSED */
12488 static void
12489 sd_uscsi_iodone(int index, struct sd_lun *un, struct buf *bp)
12490 {
12491         struct sd_xbuf *xp;

12493         ASSERT(un != NULL);
```



```
12494         ASSERT(bp != NULL);

12496         xp = SD_GET_XBUF(bp);
12497         ASSERT(xp != NULL);
12498         ASSERT(!mutex_owned(SD_MUTEX(un)));

12500         SD_INFO(SD_LOG_IO, un, "sd_uscsi_iodone: entry.\n");

12502         bp->b_private = xp->xb_private;

12504         mutex_enter(SD_MUTEX(un));

12506         /*
12507          * Grab time when the cmd completed.
12508          * This is used for determining if the system has been
12509          * idle long enough to make it idle to the PM framework.
12510          * This is for lowering the overhead, and therefore improving
12511          * performance per I/O operation.
12512          */
12513         un->un_pm_idle_time = gethrtime();
12512         un->un_pm_idle_time = ddi_get_time();

12515         un->un_ncmds_in_driver--;
12516         ASSERT(un->un_ncmds_in_driver >= 0);
12517         SD_INFO(SD_LOG_IO, un, "sd_uscsi_iodone: un_ncmds_in_driver = %ld\n",
12518             un->un_ncmds_in_driver);

12520         mutex_exit(SD_MUTEX(un));

12522         if (((struct uscsi_cmd *)(xp->xb_pktinfo))->uscsi_rqlen >
12523             SENSE_LENGTH) {
12524                 kmem_free(xp, sizeof (struct sd_xbuf) - SENSE_LENGTH +
12525                     MAX_SENSE_LENGTH);
12526         } else {
12527                 kmem_free(xp, sizeof (struct sd_xbuf));
12528         }

12530         biodone(bp);

12532         SD_INFO(SD_LOG_IO, un, "sd_uscsi_iodone: exit.\n");
12533 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   75245 Mon May  5 11:11:16 2014
new/usr/src/uts/common/sys/scsi/targets/sddef.h
4781 sd shouldn't abuse ddi_get_time(9f)
Reviewed by: Richard Elling <richard.elling@gmail.com>
Reviewed by: Robert Mustacchi <rm@joyent.com>
**********************************************************
/*
 * 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) 1990, 2010, Oracle and/or its affiliates. All rights reserved.
 */
/*
 * Copyright 2011 cyril.galibern@opensvc.com
 * Copyright 2014 Nexenta Systems, Inc.  All rights reserved.
#endif /* ! codereview */
 */

#ifndef _SYS_SCSI_TARGETS_SDDEF_H
#define _SYS_SCSI_TARGETS_SDDEF_H

#include <sys/dktp/fdisk.h>
#include <sys/note.h>
#include <sys/mhd.h>
#include <sys/cmlb.h>

#ifdef  __cplusplus
extern "C" {
#endif


#if defined(_KERNEL) || defined(_KMEMUSER)


#define SD_SUCCESS              0
#define SD_FAILURE              (-1)

#if defined(TRUE)
#undef TRUE
#endif

#if defined(FALSE)
#undef FALSE
#endif

#define TRUE                    1
#define FALSE                   0
```



```
#if defined(VERBOSE)
#undef VERBOSE
#endif

#if defined(SILENT)
#undef SILENT
#endif


/*
 *  Fault Injection Flag for Inclusion of Code
 *
 *  This should only be defined when SDDEBUG is defined
 * #if DEBUG || lint
 * #define      SD_FAULT_INJECTION
 * #endif
 */

#if DEBUG || lint
#define SD_FAULT_INJECTION
#endif
#define VERBOSE                 1
#define SILENT                  0

/*
 * Structures for recording whether a device is fully open or closed.
 * Assumptions:
 *
 *      + There are only 8 (sparc) or 16 (x86) disk slices possible.
 *      + BLK, MNT, CHR, SWP don't change in some future release!
 */

#if defined(_SUNOS_VTOC_8)

#define SDUNIT_SHIFT    3
#define SDPART_MASK     7
#define NSDMAP          NDKMAP

#elif defined(_SUNOS_VTOC_16)

/*
 * XXX - NSDMAP has multiple definitions, one more in cmlb_impl.h
 * If they are coalesced into one, this definition will follow suit.
 * FDISK partitions - 4 primary and MAX_EXT_PARTS number of Extended
 * Partitions.
 */
#define FDISK_PARTS             (FD_NUMPART + MAX_EXT_PARTS)

#define SDUNIT_SHIFT    6
#define SDPART_MASK     63
#define NSDMAP          (NDKMAP + FDISK_PARTS + 1)

#else
#error "No VTOC format defined."
#endif


#define SDUNIT(dev)     (getminor((dev)) >> SDUNIT_SHIFT)
#define SDPART(dev)     (getminor((dev)) &  SDPART_MASK)

/*
 * maximum number of partitions the driver keeps track of; with
 * EFI this can be larger than the number of partitions accessible
 * through the minor nodes.  It won't be used for keeping track
 * of open counts, partition kstats, etc.
 */
```

```
#define MAXPART         (NSDMAP + 1)

/*
 * Macro to retrieve the DDI instance number from the given buf struct.
 * The instance number is encoded in the minor device number.
 */
#define SD_GET_INSTANCE_FROM_BUF(bp)                                    \
        (getminor((bp)->b_edev) >> SDUNIT_SHIFT)



struct ocinfo {
        /*
         * Types BLK, MNT, CHR, SWP,
         * assumed to be types 0-3.
         */
        uint64_t  lyr_open[NSDMAP];
        uint64_t  reg_open[OTYPCNT - 1];
};

#define OCSIZE  sizeof (struct ocinfo)

union ocmap {
        uchar_t chkd[OCSIZE];
        struct ocinfo rinfo;
};

#define lyropen rinfo.lyr_open
#define regopen rinfo.reg_open


#define SD_CDB_GROUP0           0
#define SD_CDB_GROUP1           1
#define SD_CDB_GROUP5           2
#define SD_CDB_GROUP4           3

struct sd_cdbinfo {
        uchar_t  sc_grpcode;    /* CDB group code */
        uchar_t  sc_grpmask;    /* CDB group code mask (for cmd opcode) */
        uint64_t sc_maxlba;     /* Maximum logical block addr. supported */
        uint32_t sc_maxlen;     /* Maximum transfer length supported */
};



/*
 * The following declaration are for Non-512 byte block support for the
 * removable devices. (ex - DVD RAM, MO).
 * wm_state: This is an enumeration for the different states for
 * manipalating write range list during the read-modify-write-operation.
 */
typedef enum {
        SD_WM_CHK_LIST,         /* Check list for overlapping writes */
        SD_WM_WAIT_MAP,         /* Wait for an overlapping I/O to complete */
        SD_WM_LOCK_RANGE,       /* Lock the range of lba to be written */
        SD_WM_DONE              /* I/O complete */
} wm_state;

/*
 * sd_w_map: Every write I/O will get one w_map allocated for it which will tell
 * the range on the media which is being written for that request.
 */
struct sd_w_map {
        uint_t          wm_start;       /* Write start location */
        uint_t          wm_end;         /* Write end location */
        ushort_t        wm_flags;       /* State of the wmap */
```



```
        ushort_t        wm_wanted_count; /* # of threads waiting for region */
        void            *wm_private;    /* Used to store bp->b_private */
        struct buf      *wm_bufp;       /* to store buf pointer */
        struct sd_w_map *wm_next;       /* Forward pointed to sd_w_map */
        struct sd_w_map *wm_prev;       /* Back pointer to sd_w_map */
        kcondvar_t      wm_avail;       /* Sleep on this, while not available */
};

_NOTE(MUTEX_PROTECTS_DATA(scsi_device::sd_mutex, sd_w_map::wm_flags))


/*
 * This is the struct for the layer-private data area for the
 * mapblocksize layer.
 */

struct sd_mapblocksize_info {
        void            *mbs_oprivate;  /* saved value of xb_private */
        struct buf      *mbs_orig_bp;   /* ptr to original bp */
        struct sd_w_map *mbs_wmp;       /* ptr to write-map struct for RMW */
        ssize_t         mbs_copy_offset;
        int             mbs_layer_index;        /* chain index for RMW */
};

_NOTE(SCHEME_PROTECTS_DATA("unshared data", sd_mapblocksize_info))


/*
 * sd_lun: The main data structure for a scsi logical unit.
 * Stored as the softstate structure for each device.
 */

struct sd_lun {

        /* Back ptr to the SCSA scsi_device struct for this LUN */
        struct scsi_device      *un_sd;

        /*
         * Support for Auto-Request sense capability
         */
        struct buf      *un_rqs_bp;     /* ptr to request sense bp */
        struct scsi_pkt *un_rqs_pktp;   /* ptr to request sense scsi_pkt */
        int     un_sense_isbusy;        /* Busy flag for RQS buf */

        /*
         * These specify the layering chains to use with this instance. These
         * are initialized according to the values in the sd_chain_index_map[]
         * array. See the description of sd_chain_index_map[] for details.
         */
        int un_buf_chain_type;
        int un_uscsi_chain_type;
        int un_direct_chain_type;
        int un_priority_chain_type;

        /* Head & tail ptrs to the queue of bufs awaiting transport */
        struct buf      *un_waitq_headp;
        struct buf      *un_waitq_tailp;

        /* Ptr to the buf currently being retried (NULL if none) */
        struct buf      *un_retry_bp;

        /* This tracks the last kstat update for the un_retry_bp buf */
        void            (*un_retry_statp)(kstat_io_t *);

        void            *un_xbuf_attr;  /* xbuf attribute struct */
```

```
259         /* System logical block size, in bytes. (defaults to DEV_BSIZE.) */
260         uint32_t        un_sys_blocksize;
261
262         /* The size of a logical block on the target, in bytes. */
263         uint32_t        un_tgt_blocksize;
264
265         /* The size of a physical block on the target, in bytes. */
266         uint32_t        un_phy_blocksize;
267
268         /*
269          * The number of logical blocks on the target. This is adjusted
270          * to be in terms of the block size specified by un_sys_blocksize
271          * (ie, the system block size).
272          */
273         uint64_t        un_blockcount;
274
275         /*
276          * Various configuration data
277          */
278         uchar_t un_ctype;               /* Controller type */
279         char    *un_node_type;          /* minor node type */
280         uchar_t un_interconnect_type;   /* Interconnect for underlying HBA */
281
282         uint_t  un_notready_retry_count; /* Per disk notready retry count */
283         uint_t  un_busy_retry_count;    /* Per disk BUSY retry count */
284
285         uint_t  un_retry_count;         /* Per disk retry count */
286         uint_t  un_victim_retry_count;  /* Per disk victim retry count */
287
288         /* (4356701, 4367306) */
289         uint_t  un_reset_retry_count; /* max io retries before issuing reset */
290         ushort_t un_reserve_release_time; /* reservation release timeout */
291
292         uchar_t un_reservation_type;    /* SCSI-3 or SCSI-2 */
293         uint_t  un_max_xfer_size;       /* Maximum DMA transfer size */
294         int     un_partial_dma_supported;
295         int     un_buf_breakup_supported;
296
297         int     un_mincdb;              /* Smallest CDB to use */
298         int     un_maxcdb;              /* Largest CDB to use */
299         int     un_max_hba_cdb;         /* Largest CDB supported by HBA */
300         int     un_status_len;
301         int     un_pkt_flags;
302
303         /*
304          * Note: un_uscsi_timeout is a "mirror" of un_cmd_timeout, adjusted
305          * for ISCD().  Any updates to un_cmd_timeout MUST be reflected
306          * in un_uscsi_timeout as well!
307          */
308         ushort_t un_cmd_timeout;        /* Timeout for completion */
309         ushort_t un_uscsi_timeout;      /* Timeout for USCSI completion */
310         ushort_t un_busy_timeout;       /* Timeout for busy retry */
311
312         /*
313          * Info on current states, statuses, etc. (Updated frequently)
314          */
315         uchar_t un_state;               /* current state */
316         uchar_t un_last_state;          /* last state */
317         uchar_t un_last_pkt_reason;     /* used to suppress multiple msgs */
318         int     un_tagflags;            /* Pkt Flags for Tagged Queueing  */
319         short   un_resvd_status;        /* Reservation Status */
320         ulong_t un_detach_count;        /* !0 if executing detach routine */
321         ulong_t un_layer_count;         /* Current total # of layered opens */
322         ulong_t un_opens_in_progress;   /* Current # of threads in sdopen */
```



```
324         ksema_t un_semoclose;           /* serialize opens/closes */
325
326         /*
327          * Control & status info for command throttling
328          */
329         long    un_ncmds_in_driver;     /* number of cmds in driver */
330         short   un_ncmds_in_transport;  /* number of cmds in transport */
331         short   un_throttle;            /* max #cmds allowed in transport */
332         short   un_saved_throttle;      /* saved value of un_throttle */
333         short   un_busy_throttle;       /* saved un_throttle for BUSY */
334         short   un_min_throttle;        /* min value of un_throttle */
335         timeout_id_t    un_reset_throttle_timeid; /* timeout(9F) handle */
336
337         /*
338          * Multi-host (clustering) support
339          */
340         opaque_t        un_mhd_token;           /* scsi watch request */
341         timeout_id_t    un_resvd_timeid;        /* for resvd recover */
342
343         /* Event callback resources (photon) */
344         ddi_eventcookie_t un_insert_event;      /* insert event */
345         ddi_callback_id_t un_insert_cb_id;      /* insert callback */
346         ddi_eventcookie_t un_remove_event;      /* remove event */
347         ddi_callback_id_t un_remove_cb_id;      /* remove callback */
348
349         uint_t          un_start_stop_cycle_page;       /* Saves start/stop */
350                                                         /* cycle page */
351         timeout_id_t    un_dcvb_timeid;         /* dlyd cv broadcast */
352
353         /*
354          * Data structures for open counts, partition info, VTOC,
355          * stats, and other such bookkeeping info.
356          */
357         union   ocmap   un_ocmap;               /* open partition map */
358         struct  kstat   *un_pstats[NSDMAP];     /* partition statistics */
359         struct  kstat   *un_stats;              /* disk statistics */
360         kstat_t         *un_errstats;           /* for error statistics */
361         uint64_t        un_exclopen;            /* exclusive open bitmask */
362         ddi_devid_t     un_devid;               /* device id */
363         uint_t          un_vpd_page_mask;       /* Supported VPD pages */
364
365         /*
366          * Bit fields for various configuration/state/status info.
367          * Comments indicate the condition if the value of the
368          * variable is TRUE (nonzero).
369          */
370         uint32_t
371             un_f_arq_enabled            :1,     /* Auto request sense is */
372                                                 /* currently enabled */
373             un_f_blockcount_is_valid    :1,     /* The un_blockcount */
374                                                 /* value is currently valid */
375             un_f_tgt_blocksize_is_valid :1,     /* The un_tgt_blocksize */
376                                                 /* value is currently valid */
377             un_f_allow_bus_device_reset :1,     /* Driver may issue a BDR as */
378                                                 /* a part of error recovery. */
379             un_f_is_fibre               :1,     /* The device supports fibre */
380                                                 /* channel */
381             un_f_sync_cache_supported   :1,     /* sync cache cmd supported */
382                                                 /* supported */
383             un_f_format_in_progress     :1,     /* The device is currently */
384                                                 /* executing a FORMAT cmd. */
385             un_f_opt_queueing           :1,     /* Enable Command Queuing to */
386                                                 /* Host Adapter */
387             un_f_opt_fab_devid          :1,     /* Disk has no valid/unique */
388                                                 /* serial number.  */
389             un_f_opt_disable_cache      :1,     /* Read/Write disk cache is */
```

```
390                                             /* disabled.  */
391             un_f_cfg_is_atapi           :1,     /* This is an ATAPI device.  */
392             un_f_write_cache_enabled    :1,     /* device return success on */
393                                             /* writes before transfer to */
394                                             /* physical media complete */
395             un_f_cfg_playmsf_bcd        :1,     /* Play Audio, BCD params. */
396             un_f_cfg_readsub_bcd        :1,     /* READ SUBCHANNEL BCD resp. */
397             un_f_cfg_read_toc_trk_bcd   :1,     /* track # is BCD */
398             un_f_cfg_read_toc_addr_bcd  :1,     /* address is BCD */
399             un_f_cfg_no_read_header     :1,     /* READ HEADER not supported */
400             un_f_cfg_read_cd_xd4        :1,     /* READ CD opcode is 0xd4 */
401             un_f_mmc_cap                :1,     /* Device is MMC compliant */
402             un_f_mmc_writable_media     :1,     /* writable media in device */
403             un_f_dvdram_writable_device :1,     /* DVDRAM device is writable */
404             un_f_cfg_cdda               :1,     /* READ CDDA supported */
405             un_f_cfg_tur_check          :1,     /* verify un_ncmds before tur */

407             un_f_use_adaptive_throttle  :1,     /* enable/disable adaptive */
408                                             /* throttling */
409             un_f_pm_is_enabled          :1,     /* PM is enabled on this */
410                                             /* instance */
411             un_f_watcht_stopped         :1,     /* media watch thread flag */
412             un_f_pkstats_enabled        :1,     /* Flag to determine if */
413                                             /* partition kstats are */
414                                             /* enabled. */
415             un_f_disksort_disabled      :1,     /* Flag to disable disksort */
416             un_f_lun_reset_enabled      :1,     /* Set if target supports */
417                                             /* SCSI Logical Unit Reset */
418             un_f_doorlock_supported     :1,     /* Device supports Doorlock */
419             un_f_start_stop_supported   :1,     /* device has motor */
420             un_f_reserved1              :1;

422         uint32_t
423             un_f_mboot_supported        :1,     /* mboot supported */
424             un_f_is_hotpluggable        :1,     /* hotpluggable */
425             un_f_has_removable_media    :1,     /* has removable media */
426             un_f_non_devbsize_supported :1,     /* non-512 blocksize */
427             un_f_devid_supported        :1,     /* device ID supported */
428             un_f_eject_media_supported  :1,     /* media can be ejected */
429             un_f_chk_wp_open            :1,     /* check if write-protected */
430                                             /* when being opened */
431             un_f_descr_format_supported :1,     /* support descriptor format */
432                                             /* for sense data */
433             un_f_check_start_stop       :1,     /* needs to check if */
434                                             /* START-STOP command is */
435                                             /* supported by hardware */
436                                             /* before issuing it */
437             un_f_monitor_media_state    :1,     /* need a watch thread to */
438                                             /* monitor device state */
439             un_f_attach_spinup          :1,     /* spin up once the */
440                                             /* device is attached */
441             un_f_log_sense_supported    :1,     /* support log sense */
442             un_f_pm_supported           :1,     /* support power-management */
443             un_f_cfg_is_lsi             :1,     /* Is LSI device, */
444                                             /* default to NO */
445             un_f_wcc_inprog             :1,     /* write cache change in */
446                                             /* progress */
447             un_f_ejecting               :1,     /* media is ejecting */
448             un_f_suppress_cache_flush   :1,     /* supress flush on */
449                                             /* write cache */
450             un_f_sync_nv_supported      :1,     /* SYNC_NV */
451                                             /* bit is supported */
452             un_f_sync_cache_required    :1,     /* flag to check if */
453                                             /* SYNC CACHE needs to be */
454                                             /* sent in sdclose */
455             un_f_devid_transport_defined :1,    /* devid defined by transport */
```



```
456             un_f_rmw_type                :2,    /* RMW type */
457             un_f_power_condition_disabled :1,   /* power condition disabled */
458                                                 /* through sd configuration */
459             un_f_power_condition_supported :1,  /* support power condition */
460                                                 /* field by hardware */
461             un_f_pm_log_sense_smart     :1,     /* log sense support SMART */
462                                                 /* feature attribute */
463             un_f_is_solid_state         :1,     /* has solid state media */
464             un_f_mmc_gesn_polling       :1,     /* use GET EVENT STATUS */
465                                                 /* NOTIFICATION for polling */
466             un_f_enable_rmw             :1,     /* Force RMW in sd driver */
467             un_f_expnevent              :1,
468             un_f_reserved               :3;

470         /* Ptr to table of strings for ASC/ASCQ error message printing */
471         struct scsi_asq_key_strings     *un_additional_codes;

473         /*
474          * Power Management support.
475          *
476          * un_pm_mutex protects, un_pm_count, un_pm_timeid, un_pm_busy,
477          * un_pm_busy_cv, and un_pm_idle_timeid.
478          * It's not required that SD_MUTEX be acquired before acquiring
479          * un_pm_mutex, however if they must both be held
480          * then acquire SD_MUTEX first.
481          *
482          * un_pm_count is used to indicate PM state as follows:
483          *      less than 0 the device is powered down,
484          *      transition from 0 ==> 1, mark the device as busy via DDI
485          *      transition from 1 ==> 0, mark the device as idle via DDI
486          */
487         kmutex_t        un_pm_mutex;
488         int             un_pm_count;            /* indicates pm state */
489         timeout_id_t    un_pm_timeid;           /* timeout id for pm */
490         uint_t          un_pm_busy;
491         kcondvar_t      un_pm_busy_cv;
492         short           un_power_level;         /* Power Level */
493         uchar_t         un_save_state;
494         kcondvar_t      un_suspend_cv;          /* power management */
495         kcondvar_t      un_disk_busy_cv;        /* wait for IO completion */

497         /* Resources used for media change callback support */
498         kcondvar_t      un_state_cv;            /* Cond Var on mediastate */
499         enum dkio_state un_mediastate;          /* current media state */
500         enum dkio_state un_specified_mediastate; /* expected state */
501         opaque_t        un_swr_token;           /* scsi_watch request token */

503         /* Non-512 byte block support */
504         struct kmem_cache *un_wm_cache; /* fast alloc in non-512 write case */
505         uint_t          un_rmw_count;   /* count of read-modify-writes */
506         struct sd_w_map *un_wm;         /* head of sd_w_map chain */
507         uint64_t        un_rmw_incre_count;     /* count I/O */
508         timeout_id_t    un_rmw_msg_timeid;      /* for RMW message control */

510         /* For timeout callback to issue a START STOP UNIT command */
511         timeout_id_t    un_startstop_timeid;

513         /* Timeout callback handle for SD_PATH_DIRECT_PRIORITY cmd restarts */
514         timeout_id_t    un_direct_priority_timeid;

516         /* TRAN_FATAL_ERROR count. Cleared by TRAN_ACCEPT from scsi_transport */
517         ulong_t         un_tran_fatal_count;

519         timeout_id_t    un_retry_timeid;

521         hrtime_t        un_pm_idle_time;
```

```
 26          time_t          un_pm_idle_time;
522          timeout_id_t    un_pm_idle_timeid;

524          /*
525           * Count to determine if a Sonoma controller is in the process of
526           * failing over, and how many I/O's are failed with the 05/94/01
527           * sense code.
528           */
529          uint_t          un_sonoma_failure_count;

531          /*
532           * Support for failfast operation.
533           */
534          struct buf      *un_failfast_bp;
535          struct buf      *un_failfast_headp;
536          struct buf      *un_failfast_tailp;
537          uint32_t        un_failfast_state;
538          /* Callback routine active counter */
539          short           un_in_callback;

541          kcondvar_t      un_wcc_cv;      /* synchronize changes to */
542                                          /* un_f_write_cache_enabled */

544 #ifdef SD_FAULT_INJECTION
545          /* SD Fault Injection */
546 #define SD_FI_MAX_BUF 65536
547 #define SD_FI_MAX_ERROR 1024
548          kmutex_t                        un_fi_mutex;
549          uint_t                          sd_fi_buf_len;
550          char                            sd_fi_log[SD_FI_MAX_BUF];
551          struct sd_fi_pkt        *sd_fi_fifo_pkt[SD_FI_MAX_ERROR];
552          struct sd_fi_xb         *sd_fi_fifo_xb[SD_FI_MAX_ERROR];
553          struct sd_fi_un         *sd_fi_fifo_un[SD_FI_MAX_ERROR];
554          struct sd_fi_arq        *sd_fi_fifo_arq[SD_FI_MAX_ERROR];
555          uint_t                          sd_fi_fifo_start;
556          uint_t                          sd_fi_fifo_end;
557          uint_t                          sd_injection_mask;

559 #endif

561          cmlb_handle_t   un_cmlbhandle;

563          /*
564           * Pointer to internal struct sd_fm_internal in which
565           * will pass necessary information for FMA ereport posting.
566           */
567          void            *un_fm_private;
568 };
```

_____***unchanged_portion_omitted_***