```
**********************************************************
    2378 Thu Nov  1 14:48:22 2012
new/usr/src/pkg/manifests/driver-storage-aac.mf
*** NO COMMENTS ***
**********************************************************
#
# 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) 2010, Oracle and/or its affiliates. All rights reserved.
#

#
# The default for payload-bearing actions in this package is to appear in the
# global zone only.  See the include file for greater detail, as well as
# information about overriding the defaults.
#
<include global_zone_only_component>
set name=pkg.fmri value=pkg:/driver/storage/aac@$(PKGVERS)
set name=pkg.description \
    value="Adaptec AdvanceRaid Controller SCSI HBA Driver"
set name=pkg.summary value="Adaptec AdvanceRaid Controller SCSI HBA Driver"
set name=info.classification value=org.opensolaris.category.2008:Drivers/Ports
set name=variant.arch value=$(ARCH)
dir path=kernel group=sys
dir path=kernel/drv group=sys
dir path=kernel/drv/$(ARCH64) group=sys
dir path=usr/share/man
dir path=usr/share/man/man7d
driver name=aac class=scsi \
    alias=pci1028,3 \
    alias=pci1028,a \
    alias=pci9005,285 \
    alias=pci9005,286 \
    alias=pciex9005,285 \
    alias=pciex9005,286 \
    alias=pciex9005,28b \
    alias=pciex9005,28c \
    alias=pciex9005,28d \
    alias=pciex9005,28f
    alias=pciex9005,286
file path=kernel/drv/$(ARCH64)/aac group=sys
$(i386_ONLY)file path=kernel/drv/aac group=sys
file path=kernel/drv/aac.conf group=sys \
    original_name=SUNWaac:kernel/drv/aac.conf preserve=true
file path=usr/share/man/man7d/aac.7d
legacy pkg=SUNWaac desc="Adaptec AdvanceRaid Controller SCSI HBA Driver" \
    name="Adaptec AdvanceRaid Controller SCSI HBA Driver"
```



```
license cr_Sun license=cr_Sun
license usr/src/uts/common/io/aac/THIRDPARTYLICENSE \
    license=usr/src/uts/common/io/aac/THIRDPARTYLICENSE
```

```
**********************************************************
    3636 Thu Nov  1 14:48:22 2012
new/usr/src/uts/common/io/aac/README
*** NO COMMENTS ***
**********************************************************
#
# 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.
#

# XXX KEBE SAYS UPDATE ME!

aac driver 2.0
================
The Solaris aac driver enhancement updates the Solaris aac 1.6 driver
to a full-fledged one. The old Solaris aac driver is simple and stable,
but with limited functions.

The new Solaris aac driver adds support of the following features:
1. New firmware support:
        New Communication interface, RawIO command, Large FIB, 64-bit LBA
2. New hardware support:
        Rocket chip based cards, such as 2820SA
3. Other features:
        64-bit DMA, Fast IO, firmware version checking, tagged-queuing
4. IOCTL
5. AIF
6. IOP reset

The new Solaris aac 2.0 driver is mainly based on FreeBSD 6.0 aac driver.
IOP reset, AIF handling, 64-bit LBA, and some other minor features are
implemented from scratch by Adaptec engineers. They are not supported in
current FreeBSD aac driver.

Adaptec approves Sun’s intention to open source the aac RAID driver for
commercial Solaris and the Open Solaris community.

aac driver 2.1.14
==================
UART trace support is added in this release:
        The driver can now make output through the firmware UART trace. You
        have to set AAC_DEBUG and the appropriate flags in aac_debug_flags, at
        least AACDB_FLAGS_FW_PRINT to enable the UART trace and any other flags
        to enable the output class you want to see.
        To use the tip utility to connect to the UART daughter card, a configu-
```



```
        ration line as below should be added to /etc/remote:
        aac:\
                :dv=/dev/term/a:br#115200:el=^C^S^Q^U^D:ie=%$:oe=^D:

aac driver 2.2.0
==================
SPARC platform support is added in this release:
        To support SPARC, the driver is modified for DDI compliance. The driver
        now uses DDI compliant functions to access the device’s IO and memory
        spaces for DMA transfers.

aac driver 2.2.3
==================
MSI interrupts supporting is added in this release:
        Instead of supporting fixed interrupt only, the driver added the MSI
        interrupt whenever the HBA card has this feature. In the same time, the
        driver has replaced all legacy interrupt ddi interfaces callings by
        the according ddi_intr_* ones.

aac driver 2.2.5
==================
Two new features are included in this release:
One is Non-DASD support:
        The driver now supports non-DASD(Non Direct Access Storage Device).
        This means you can use cdroms and other non-DASD devices with
        your aac card. Before trying it, make sure your card’s firmware support
        non-DASD access. For some cards, you may need to explicitly enable it
        through aac BIOS. Make sure nondasd-enable property in aac.conf is
        switched on.
The other is FIB dump:
        The driver now supports FIB contents dumping for driver debugging. To
        enable it, you need to set AAC_DEBUG and the appropriate flags in
        aac_debug_fib_flags.
```

```
**********************************************************
    1502 Thu Nov  1 14:48:23 2012
new/usr/src/uts/common/io/aac/THIRDPARTYLICENSE
*** NO COMMENTS ***
**********************************************************
```

```
 1  * Copyright (c) 2010-12 PMC-Sierra, Inc.
 2  * Copyright (c) 2005-10 Adaptec Inc., Achim Leubner
 1  * Copyright 2005-06 Adaptec, Inc.
 2  * Copyright (c) 2005-06 Adaptec Inc., Achim Leubner
 3  * Copyright (c) 2000 Michael Smith
 4  * Copyright (c) 2001 Scott Long
 5  * Copyright (c) 2000 BSDi
 6  * All rights reserved.
 7  *
 8  * Redistribution and use in source and binary forms, with or without
 9  * modification, are permitted provided that the following conditions
10  * are met:
11  * 1. Redistributions of source code must retain the above copyright
12  *    notice, this list of conditions and the following disclaimer.
13  * 2. Redistributions in binary form must reproduce the above copyright
14  *    notice, this list of conditions and the following disclaimer in the
15  *    documentation and/or other materials provided with the distribution.
16  *
17  * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS'' AND
18  * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
19  * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
20  * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
21  * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
22  * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
23  * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
24  * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
25  * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
26  * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
27  * SUCH DAMAGE.
```

```
**********************************************************
  244603 Thu Nov  1 14:48:24 2012
new/usr/src/uts/common/io/aac/aac.c
*** NO COMMENTS ***
**********************************************************
```

```
/*
 * Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 * Copyright (c) 2010, Oracle and/or its affiliates. All rights reserved.
 */

/*
 * Copyright (c) 2010-12 PMC-Sierra, Inc.
 * Copyright (c) 2005-10 Adaptec Inc., Achim Leubner
 * Copyright 2005-08 Adaptec, Inc.
 * Copyright (c) 2005-08 Adaptec Inc., Achim Leubner
 * Copyright (c) 2000 Michael Smith
 * Copyright (c) 2001 Scott Long
 * Copyright (c) 2000 BSDi
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */
#pragma ident   "@(#)aac.c      1.19    08/01/29 SMI"

#include <sys/modctl.h>
#include <sys/conf.h>
#include <sys/cmn_err.h>
#include <sys/ddi.h>
#include <sys/devops.h>
#include <sys/pci.h>
#include <sys/types.h>
#include <sys/ddidmareq.h>
#include <sys/scsi/scsi.h>
#include <sys/ksynch.h>
#include <sys/sunddi.h>
#include <sys/byteorder.h>
#include <sys/utsname.h>
#include "aac_regs.h"
#include "aac.h"

/*
 * FMA header files, macros
 * FMA header files
 */
#include <sys/ddifm.h>
#include <sys/fm/protocol.h>
```



```
#include <sys/fm/util.h>
#include <sys/fm/io/ddi.h>
#ifndef DDI_FM_DEVICE
#define DDI_FM_DEVICE                   "device"
#define DDI_FM_DEVICE_NO_RESPONSE       "no_response"
#define ddi_fm_acc_err_clear(a, b)
#endif
#define aac_fm_service_impact(a, b) { \
        if (aac_fm_valid) \
                ddi_fm_service_impact(a, b); \
        }
#define aac_fm_acc_err_clear(a, b) { \
        if (aac_fm_valid) \
                ddi_fm_acc_err_clear(a, b); \
        }

char _depends_on[] = "misc/scsi";

/*
 * For minor nodes created by the SCSA framework, minor numbers are
 * formed by left-shifting instance by INST_MINOR_SHIFT and OR in a
 * number less than 64.
 *
 * To support cfgadm, need to confirm the SCSA framework by creating
 * devctl/scsi and driver specific minor nodes under SCSA format,
 * and calling scsi_hba_xxx() functions aacordingly.
 */

#define AAC_MINOR               32
#define INST2AAC(x)             (((x) << INST_MINOR_SHIFT) | AAC_MINOR)
#define AAC_SCSA_MINOR(x)       ((x) & TRAN_MINOR_MASK)
#define AAC_IS_SCSA_NODE(x)     ((x) == DEVCTL_MINOR || (x) == SCSI_MINOR)

#define SD2TRAN(sd)             ((sd)->sd_address.a_hba_tran)
#define AAC_TRAN2SOFTS(tran) ((struct aac_softstate *)(tran)->tran_hba_private)
#define AAC_DIP2TRAN(dip)       ((scsi_hba_tran_t *)ddi_get_driver_private(dip))
#define AAC_DIP2SOFTS(dip)      (AAC_TRAN2SOFTS(AAC_DIP2TRAN(dip)))
#define SD2AAC(sd)              (AAC_TRAN2SOFTS(SD2TRAN(sd)))
#define AAC_PD(t)               ((t) - AAC_MAX_LD)
#define AAC_DEV(softs, t)       (((t) < AAC_MAX_LD) ? \
                                &(softs)->containers[(t)].dev : \
                                ((t) < AAC_MAX_DEV(softs)) ? \
                                &(softs)->nondasds[AAC_PD(t)].dev : NULL)
#define AAC_DEVCFG_BEGIN(softs, tgt) \
                                aac_devcfg((softs), (tgt), 1)
#define AAC_DEVCFG_END(softs, tgt) \
                                aac_devcfg((softs), (tgt), 0)
#define PKT2AC(pkt)             ((struct aac_cmd *)(pkt)->pkt_ha_private)
#define AAC_BUSYWAIT(cond, timeout /* in millisecond */) { \
                if (!(cond)) { \
                        int count = (timeout) * 10; \
                        while (count) { \
                                drv_usecwait(100); \
                                if (cond) \
                                        break; \
                                count--; \
                        } \
                        (timeout) = (count + 9) / 10; \
                } \
        }

#define AAC_SENSE_DATA_DESCR_LEN \
        (sizeof (struct scsi_descr_sense_hdr) + \
        sizeof (struct scsi_information_sense_descr))
#define AAC_ARQ64_LENGTH \
        (sizeof (struct scsi_arq_status) + \
```

```
120         AAC_SENSE_DATA_DESCR_LEN - SENSE_LENGTH)

122 /* NOTE: GETG4ADDRTL(cdbp) is int32_t */
123 #define AAC_GETGXADDR(cmdlen, cdbp) \
124         ((cmdlen == 6) ? GETG0ADDR(cdbp) : \
125         (cmdlen == 10) ? (uint32_t)GETG1ADDR(cdbp) : \
126         ((uint64_t)GETG4ADDR(cdbp) << 32) | (uint32_t)GETG4ADDRTL(cdbp))

128 #define AAC_CDB_INQUIRY_CMDDT   0x02
129 #define AAC_CDB_INQUIRY_EVPD    0x01
130 #define AAC_VPD_PAGE_CODE       1
131 #define AAC_VPD_PAGE_LENGTH     3
132 #define AAC_VPD_PAGE_DATA       4
133 #define AAC_VPD_ID_CODESET      0
134 #define AAC_VPD_ID_TYPE         1
135 #define AAC_VPD_ID_LENGTH       3
136 #define AAC_VPD_ID_DATA         4

138 #define AAC_SCSI_RPTLUNS_HEAD_SIZE                      0x08
139 #define AAC_SCSI_RPTLUNS_ADDR_SIZE                      0x08
140 #define AAC_SCSI_RPTLUNS_ADDR_MASK                      0xC0
141 /* 00b - peripheral device addressing method */
142 #define AAC_SCSI_RPTLUNS_ADDR_PERIPHERAL                0x00
143 /* 01b - flat space addressing method */
144 #define AAC_SCSI_RPTLUNS_ADDR_FLAT_SPACE                0x40
145 /* 10b - logical unit addressing method */
146 #define AAC_SCSI_RPTLUNS_ADDR_LOGICAL_UNIT              0x80

148 /* Return the size of FIB with data part type data_type */
149 #define AAC_FIB_SIZEOF(data_type) \
150         (sizeof (struct aac_fib_header) + sizeof (data_type))
151 /* Return the container size defined in mir */
152 #define AAC_MIR_SIZE(softs, acc, mir) \
153         (((softs)->flags & AAC_FLAGS_LBA_64BIT) ? \
154         (uint64_t)ddi_get32((acc), &(mir)->MntObj.Capacity) + \
155         ((uint64_t)ddi_get32((acc), &(mir)->MntObj.CapacityHigh) << 32) : \
156         (uint64_t)ddi_get32((acc), &(mir)->MntObj.Capacity))

158 /* The last entry of aac_cards[] is for unknown cards */
159 #define AAC_UNKNOWN_CARD \
160         (sizeof (aac_cards) / sizeof (struct aac_card_type) - 1)
161 #define CARD_IS_UNKNOWN(i)      (i == AAC_UNKNOWN_CARD)
162 #define BUF_IS_READ(bp)         ((bp)->b_flags & B_READ)
163 #define AAC_IS_Q_EMPTY(q)       ((q)->q_head == NULL)
164 #define AAC_CMDQ(acp)           (!((acp)->flags & AAC_CMD_SYNC))

166 #define PCI_MEM_GET32(softs, i, off) \
167         ddi_get32((softs)->pci_mem_handle[i], \
168             (uint32_t *)((softs)->pci_mem_base_vaddr[i] + (off)))
169 #define PCI_MEM_PUT32(softs, i, off, val) \
170         ddi_put32((softs)->pci_mem_handle[i], \
171             (uint32_t *)((softs)->pci_mem_base_vaddr[i] + (off)), \
151 #define PCI_MEM_GET32(softs, off) \
152         ddi_get32((softs)->pci_mem_handle, \
153             (void *)((softs)->pci_mem_base_vaddr + (off)))
154 #define PCI_MEM_PUT32(softs, off, val) \
155         ddi_put32((softs)->pci_mem_handle, \
156             (void *)((softs)->pci_mem_base_vaddr + (off)), \
172             (uint32_t)(val))
173 #define PCI_MEM_GET16(softs, i, off) \
174         ddi_get16((softs)->pci_mem_handle[i], \
175         (uint16_t *)((softs)->pci_mem_base_vaddr[i] + (off)))
176 #define PCI_MEM_PUT16(softs, i, off, val) \
177         ddi_put16((softs)->pci_mem_handle[i], \
178         (uint16_t *)((softs)->pci_mem_base_vaddr[i] + (off)), (uint16_t)(val))
179 /* Write host data at valp to device mem[i][off] repeatedly count times */
```



```
180 #define PCI_MEM_REP_PUT8(softs, i, off, valp, count) \
181         ddi_rep_put8((softs)->pci_mem_handle[i], (uint8_t *)(valp), \
182             (uint8_t *)((softs)->pci_mem_base_vaddr[i] + (off)), \
158 #define PCI_MEM_GET16(softs, off) \
159         ddi_get16((softs)->pci_mem_handle, \
160         (void *)((softs)->pci_mem_base_vaddr + (off)))
161 #define PCI_MEM_PUT16(softs, off, val) \
162         ddi_put16((softs)->pci_mem_handle, \
163         (void *)((softs)->pci_mem_base_vaddr + (off)), (uint16_t)(val))
164 /* Write host data at valp to device mem[off] repeatedly count times */
165 #define PCI_MEM_REP_PUT8(softs, off, valp, count) \
166         ddi_rep_put8((softs)->pci_mem_handle, (uint8_t *)(valp), \
167             (uint8_t *)((softs)->pci_mem_base_vaddr + (off)), \
183             count, DDI_DEV_AUTOINCR)
184 /* Read device data at mem[i][off] to host addr valp repeatedly count times */
185 #define PCI_MEM_REP_GET8(softs, i, off, valp, count) \
186         ddi_rep_get8((softs)->pci_mem_handle[i], (uint8_t *)(valp), \
187             (uint8_t *)((softs)->pci_mem_base_vaddr[i] + (off)), \
169 /* Read device data at mem[off] to host addr valp repeatedly count times */
170 #define PCI_MEM_REP_GET8(softs, off, valp, count) \
171         ddi_rep_get8((softs)->pci_mem_handle, (uint8_t *)(valp), \
172             (uint8_t *)((softs)->pci_mem_base_vaddr + (off)), \
188             count, DDI_DEV_AUTOINCR)
189 #define AAC_GET_FIELD8(acc, d, s, field) \
190         (d)->field = ddi_get8(acc, (uint8_t *)&(s)->field)
191 #define AAC_GET_FIELD32(acc, d, s, field) \
192         (d)->field = ddi_get32(acc, (uint32_t *)&(s)->field)
193 #define AAC_GET_FIELD64(acc, d, s, field) \
194         (d)->field = ddi_get64(acc, (uint64_t *)&(s)->field)
195 #define AAC_REP_GET_FIELD8(acc, d, s, field, r) \
196         ddi_rep_get8((acc), (uint8_t *)&(d)->field, \
197             (uint8_t *)&(s)->field, (r), DDI_DEV_AUTOINCR)
198 #define AAC_REP_GET_FIELD32(acc, d, s, field, r) \
199         ddi_rep_get32((acc), (uint32_t *)&(d)->field, \
200             (uint32_t *)&(s)->field, (r), DDI_DEV_AUTOINCR)

202 #define AAC_OUTB_GET(softs)             PCI_MEM_GET32(softs, 0, AAC_OQUE)
203 #define AAC_OUTB_SET(softs, val)        PCI_MEM_PUT32(softs, 0, AAC_OQUE, val)
187 #define AAC_ENABLE_INTR(softs) { \
188                 if (softs->flags & AAC_FLAGS_NEW_COMM) \
189                         PCI_MEM_PUT32(softs, AAC_OIMR, ~AAC_DB_INTR_NEW); \
190                 else \
191                         PCI_MEM_PUT32(softs, AAC_OIMR, ~AAC_DB_INTR_BITS); \
192                 softs->state |= AAC_STATE_INTR; \
193         }

205 #define AAC_SET_INTR(softs, enable)     \
206         ((softs)->aac_if.aif_set_intr((softs), (enable)))
207 #define AAC_STATUS_CLR(softs, mask)     \
208         ((softs)->aac_if.aif_status_clr((softs), (mask)))
209 #define AAC_STATUS_GET(softs)   \
210         ((softs)->aac_if.aif_status_get(softs))
211 #define AAC_NOTIFY(softs, val)  \
212         ((softs)->aac_if.aif_notify((softs), (val)))

195 #define AAC_DISABLE_INTR(softs) { \
196                 PCI_MEM_PUT32(softs, AAC_OIMR, ~0); \
197                 softs->state &= ~AAC_STATE_INTR; \
198         }
199 #define AAC_STATUS_CLR(softs, mask)     PCI_MEM_PUT32(softs, AAC_ODBR, mask)
200 #define AAC_STATUS_GET(softs)           PCI_MEM_GET32(softs, AAC_ODBR)
201 #define AAC_NOTIFY(softs, val)          PCI_MEM_PUT32(softs, AAC_IDBR, val)
202 #define AAC_OUTB_GET(softs)             PCI_MEM_GET32(softs, AAC_OQUE)
203 #define AAC_OUTB_SET(softs, val)        PCI_MEM_PUT32(softs, AAC_OQUE, val)
214 #define AAC_FWSTATUS_GET(softs) \
215         ((softs)->aac_if.aif_get_fwstatus(softs))
```

```
#define AAC_MAILBOX_GET(softs, mb) \
        ((softs)->aac_if.aif_get_mailbox((softs), (mb)))
#define AAC_MAILBOX_SET(softs, cmd, arg0, arg1, arg2, arg3) \
        ((softs)->aac_if.aif_set_mailbox((softs), (cmd), \
            (arg0), (arg1), (arg2), (arg3)))
#define AAC_SEND_COMMAND(softs, slotp) \
        ((softs)->aac_if.aif_send_command((softs), (slotp)))

#define AAC_MGT_SLOT_NUM        2
#define AAC_THROTTLE_DRAIN      -1

#define AAC_QUIESCE_TICK        1       /* 1 second */
#define AAC_QUIESCE_TIMEOUT     180     /* 180 seconds */
#define AAC_DEFAULT_TICK        10      /* 10 seconds */
#define AAC_SYNC_TICK           (30*60) /* 30 minutes */

/* Poll time for aac_do_poll_io() */
#define AAC_POLL_TIME           60      /* 60 seconds */

/* IOP reset */
#define AAC_IOP_RESET_SUCCEED           0       /* IOP reset succeed */
#define AAC_IOP_RESET_FAILED            -1      /* IOP reset failed */
#define AAC_IOP_RESET_ABNORMAL          -2      /* Reset operation abnormal */

/*
 * Hardware access functions
 */
static void aac_rx_set_intr(struct aac_softstate *, int);
static void aac_rx_status_clr(struct aac_softstate *, int);
static int aac_rx_status_get(struct aac_softstate *);
static void aac_rx_notify(struct aac_softstate *, int);
static int aac_rx_get_fwstatus(struct aac_softstate *);
static int aac_rx_get_mailbox(struct aac_softstate *, int);
static void aac_rx_set_mailbox(struct aac_softstate *, uint32_t, uint32_t,
    uint32_t, uint32_t, uint32_t);
static int aac_rx_send_command(struct aac_softstate *, struct aac_slot *);
static int aac_rkt_get_fwstatus(struct aac_softstate *);
static int aac_rkt_get_mailbox(struct aac_softstate *, int);
static void aac_rkt_set_mailbox(struct aac_softstate *, uint32_t, uint32_t,
    uint32_t, uint32_t, uint32_t);
static void aac_src_set_intr(struct aac_softstate *, int);
static void aac_src_status_clr(struct aac_softstate *, int);
static int aac_src_status_get(struct aac_softstate *);
static void aac_src_notify(struct aac_softstate *, int);
static int aac_src_get_fwstatus(struct aac_softstate *);
static int aac_src_get_mailbox(struct aac_softstate *, int);
static void aac_src_set_mailbox(struct aac_softstate *, uint32_t, uint32_t,
    uint32_t, uint32_t, uint32_t);
static int aac_src_send_command(struct aac_softstate *, struct aac_slot *);
static int aac_srcv_get_mailbox(struct aac_softstate *, int);
static void aac_srcv_set_mailbox(struct aac_softstate *, uint32_t, uint32_t,
    uint32_t, uint32_t, uint32_t);

/*
 * SCSA function prototypes
 */
static int aac_attach(dev_info_t *, ddi_attach_cmd_t);
static int aac_detach(dev_info_t *, ddi_detach_cmd_t);
static int aac_reset(dev_info_t *, ddi_reset_cmd_t);
static int aac_quiesce(dev_info_t *);
static int aac_getinfo(dev_info_t *, ddi_info_cmd_t, void *, void **);

/*
 * Interrupt handler functions
 */
static int aac_query_intrs(struct aac_softstate *, int);
```



```
static int aac_add_intrs(struct aac_softstate *);
static void aac_remove_intrs(struct aac_softstate *);
static int aac_enable_intrs(struct aac_softstate *);
static int aac_disable_intrs(struct aac_softstate *);
static uint_t aac_intr_old(caddr_t);
static uint_t aac_intr_new(caddr_t);
static uint_t aac_softintr(caddr_t);

/*
 * Internal functions in attach
 */
static int aac_check_card_type(struct aac_softstate *);
static int aac_check_firmware(struct aac_softstate *);
static int aac_common_attach(struct aac_softstate *);
static void aac_common_detach(struct aac_softstate *);
static int aac_probe_containers(struct aac_softstate *);
static int aac_alloc_comm_space(struct aac_softstate *);
static int aac_setup_comm_space(struct aac_softstate *);
static void aac_free_comm_space(struct aac_softstate *);
static int aac_hba_setup(struct aac_softstate *);

/*
 * Sync FIB operation functions
 */
int aac_sync_mbcommand(struct aac_softstate *, uint32_t, uint32_t,
    uint32_t, uint32_t, uint32_t, uint32_t *);
    uint32_t, uint32_t, uint32_t, uint32_t *);
static int aac_sync_fib(struct aac_softstate *, uint16_t, uint16_t);

/*
 * Command queue operation functions
 */
static void aac_cmd_initq(struct aac_cmd_queue *);
static void aac_cmd_enqueue(struct aac_cmd_queue *, struct aac_cmd *);
static struct aac_cmd *aac_cmd_dequeue(struct aac_cmd_queue *);
static void aac_cmd_delete(struct aac_cmd_queue *, struct aac_cmd *);

/*
 * FIB queue operation functions
 */
static int aac_fib_enqueue(struct aac_softstate *, int, uint32_t, uint32_t);
static int aac_fib_dequeue(struct aac_softstate *, int, int *);

/*
 * Slot operation functions
 */
static int aac_create_slots(struct aac_softstate *);
static void aac_destroy_slots(struct aac_softstate *);
static void aac_alloc_fibs(struct aac_softstate *);
static void aac_destroy_fibs(struct aac_softstate *);
static struct aac_slot *aac_get_slot(struct aac_softstate *);
static void aac_release_slot(struct aac_softstate *, struct aac_slot *);
static int aac_alloc_fib(struct aac_softstate *, struct aac_slot *);
static void aac_free_fib(struct aac_slot *);

/*
 * Internal functions
 */
static void aac_cmd_fib_header(struct aac_softstate *, struct aac_slot *,
    uint16_t, uint16_t);
static void aac_cmd_fib_header(struct aac_softstate *, struct aac_cmd *,
    uint16_t);
static void aac_cmd_fib_rawio(struct aac_softstate *, struct aac_cmd *);
static void aac_cmd_fib_rawio2(struct aac_softstate *, struct aac_cmd *);
static uint_t aac_convert_sgraw2(struct aac_softstate *, struct aac_cmd *,
                                                        uint_t, uint_t,
```

```
static void aac_cmd_fib_brw64(struct aac_softstate *, struct aac_cmd *);
static void aac_cmd_fib_brw(struct aac_softstate *, struct aac_cmd *);
static void aac_cmd_fib_sync(struct aac_softstate *, struct aac_cmd *);
static void aac_cmd_aif_request(struct aac_softstate *, struct aac_cmd *);
static void aac_aifreq_complete(struct aac_softstate *, struct aac_cmd *);
static void aac_cmd_fib_scsi32(struct aac_softstate *, struct aac_cmd *);
static void aac_cmd_fib_scsi64(struct aac_softstate *, struct aac_cmd *);
static void aac_cmd_fib_startstop(struct aac_softstate *, struct aac_cmd *);
static void aac_start_waiting_io(struct aac_softstate *);
static void aac_drain_comp_q(struct aac_softstate *);
int aac_do_io(struct aac_softstate *, struct aac_cmd *);
static int aac_sync_fib_slot_bind(struct aac_softstate *, struct aac_cmd *);
static void aac_sync_fib_slot_release(struct aac_softstate *, struct aac_cmd *);
static void aac_start_io(struct aac_softstate *, struct aac_cmd *);
static int aac_do_poll_io(struct aac_softstate *, struct aac_cmd *);
static int aac_do_sync_io(struct aac_softstate *, struct aac_cmd *);
static int aac_send_command(struct aac_softstate *, struct aac_slot *);
static void aac_cmd_timeout(struct aac_softstate *, struct aac_cmd *);
static int aac_dma_sync_ac(struct aac_cmd *);
static int aac_shutdown(struct aac_softstate *);
static int aac_reset_adapter(struct aac_softstate *);
static int aac_do_quiesce(struct aac_softstate *softs);
static int aac_do_unquiesce(struct aac_softstate *softs);
static void aac_unhold_bus(struct aac_softstate *, int);
static void aac_set_throttle(struct aac_softstate *, struct aac_device *,
    int, int);
static int aac_atoi(char **pptr);
static void aac_config_pd(void *);

/*
 * Adapter Initiated FIB handling function
 */
static int aac_handle_aif(struct aac_softstate *, struct aac_fib *);
static void aac_save_aif(struct aac_softstate *, ddi_acc_handle_t,
    struct aac_fib *, int);
static int aac_handle_aif(struct aac_softstate *, struct aac_aif_command *);

/*
 * Timeout handling thread function
 * Event handling related functions
 */
static void aac_daemon(void *);
static void aac_timer(void *);
static void aac_event_thread(struct aac_softstate *);
static void aac_event_disp(struct aac_softstate *, int);

/*
 * IOCTL interface related functions
 */
static int aac_open(dev_t *, int, int, cred_t *);
static int aac_close(dev_t, int, int, cred_t *);
static int aac_ioctl(dev_t, int, intptr_t, int, cred_t *, int *);
extern int aac_do_ioctl(struct aac_softstate *, dev_t, int, intptr_t, int);

/*
 * FMA Prototypes
 */
static void aac_fm_init(struct aac_softstate *);
static void aac_fm_fini(struct aac_softstate *);
static int aac_fm_error_cb(dev_info_t *, ddi_fm_error_t *, const void *);
int aac_check_acc_handle(ddi_acc_handle_t);
int aac_check_dma_handle(ddi_dma_handle_t);
void aac_fm_ereport(struct aac_softstate *, char *, int);
void aac_fm_ereport(struct aac_softstate *, char *);

/*
```



```
 * Auto enumeration functions
 */
static dev_info_t *aac_find_child(struct aac_softstate *, uint16_t, uint8_t);
static int aac_tran_bus_config(dev_info_t *, uint_t, ddi_bus_config_op_t,
    void *, dev_info_t **);
static int aac_dr_event(struct aac_softstate *, int, int, int);
static int aac_handle_dr(struct aac_softstate *, int, int, int);

#ifdef AAC_DEBUG
extern pri_t minclsyspri;

#ifdef DEBUG
/*
 * UART debug output support
 */

#define AAC_PRINT_BUFFER_SIZE           512
#define AAC_PRINT_TIMEOUT               250     /* 1/4 sec. = 250 msec. */

#define AAC_FW_DBG_STRLEN_OFFSET        0x00
#define AAC_FW_DBG_FLAGS_OFFSET         0x04
#define AAC_FW_DBG_BLED_OFFSET          0x08

static int aac_get_fw_debug_buffer(struct aac_softstate *);
static void aac_print_scmd(struct aac_softstate *, struct aac_cmd *);
static void aac_print_aif(struct aac_softstate *, struct aac_aif_command *);

static char aac_prt_buf[AAC_PRINT_BUFFER_SIZE];
static char aac_fmt[] = " %s";
static char aac_fmt_header[] = " %s.%d: %s";
static kmutex_t aac_prt_mutex;

/*
 * Debug flags to be put into the softstate flags field
 * when initialized
 */
uint32_t aac_debug_flags =
/*    AACDB_FLAGS_KERNEL_PRINT | */
    AACDB_FLAGS_FW_PRINT |
    AACDB_FLAGS_MISC |
/*    AACDB_FLAGS_FW_PRINT |    */
/*    AACDB_FLAGS_MISC |        */
/*    AACDB_FLAGS_FUNC1 |       */
/*    AACDB_FLAGS_FUNC2 |       */
/*    AACDB_FLAGS_SCMD |        */
/*    AACDB_FLAGS_AIF |         */
/*    AACDB_FLAGS_FIB |         */
/*    AACDB_FLAGS_IOCTL |       */
0;
#endif /* AAC_DEBUG */
uint32_t aac_debug_fib_flags =
/*    AACDB_FLAGS_FIB_RW |      */
/*    AACDB_FLAGS_FIB_IOCTL |   */
/*    AACDB_FLAGS_FIB_SRB |     */
/*    AACDB_FLAGS_FIB_SYNC |    */
/*    AACDB_FLAGS_FIB_HEADER |  */
/*    AACDB_FLAGS_FIB_TIMEOUT | */
0;

#ifdef AAC_DEBUG_ALL
#ifndef GETG0COUNT
#define GETG0COUNT(cdb)         (cdb)->g0_count0
#endif
#ifndef GETG1COUNT
#define GETG1COUNT(cdb)         (((cdb)->g1_count1 << 8) + ((cdb)->g1_count0))
#endif
```

```
#ifndef GETG4COUNT
#define GETG4COUNT(cdb)         (((cdb)->g4_count3 << 24) + \
        ((cdb)->g4_count2 << 16) + ((cdb)->g4_count1 << 8) + ((cdb)->g4_count0))
#endif
static void aac_print_scmd(struct aac_softstate *, struct aac_cmd *);
static void aac_print_aif(struct aac_softstate *, struct aac_aif_command *);
#endif /* AAC_DEBUG_ALL */
#endif /* DEBUG */

static struct cb_ops aac_cb_ops = {
        aac_open,       /* open */
        aac_close,      /* close */
        nodev,          /* strategy */
        nodev,          /* print */
        nodev,          /* dump */
        nodev,          /* read */
        nodev,          /* write */
        aac_ioctl,      /* ioctl */
        nodev,          /* devmap */
        nodev,          /* mmap */
        nodev,          /* segmap */
        nochpoll,       /* poll */
        ddi_prop_op,    /* cb_prop_op */
        NULL,           /* streamtab */
        D_64BIT | D_NEW | D_MP | D_HOTPLUG,     /* cb_flag */
        CB_REV,         /* cb_rev */
        nodev,          /* async I/O read entry point */
        nodev           /* async I/O write entry point */
};

static struct dev_ops aac_dev_ops = {
        DEVO_REV,
        0,
        nodev,
        aac_getinfo,
        nulldev,
        nulldev,
        aac_attach,
        aac_detach,
        aac_reset,
        &aac_cb_ops,
        NULL,
        NULL
        NULL,
        aac_quiesce,
};
```

_____***unchanged_portion_omitted_***

```
/*
 * Hardware access functions for i960 based cards
 */
static struct aac_interface aac_rx_interface = {
        aac_rx_set_intr,
        aac_rx_status_clr,
        aac_rx_status_get,
        aac_rx_notify,
        aac_rx_get_fwstatus,
        aac_rx_get_mailbox,
        aac_rx_set_mailbox,
        aac_rx_send_command
        aac_rx_set_mailbox
};

/*
 * Hardware access functions for Rocket based cards
 */
```



```
static struct aac_interface aac_rkt_interface = {
        aac_rx_set_intr,
        aac_rx_status_clr,
        aac_rx_status_get,
        aac_rx_notify,
        aac_rx_get_fwstatus,
        aac_rx_get_mailbox,
        aac_rkt_set_mailbox,
        aac_rkt_send_command
        aac_rkt_set_mailbox
};

/*
 * Hardware access functions for PMC SRC based cards
 */
static struct aac_interface aac_src_interface = {
        aac_src_set_intr,
        aac_src_status_clr,
        aac_src_status_get,
        aac_src_notify,
        aac_src_get_fwstatus,
        aac_src_get_mailbox,
        aac_src_set_mailbox,
        aac_src_send_command
};

/*
 * Hardware access functions for PMC SRCv based cards
 */
static struct aac_interface aac_srcv_interface = {
        aac_src_set_intr,
        aac_src_status_clr,
        aac_src_status_get,
        aac_src_notify,
        aac_src_get_fwstatus,
        aac_srcv_get_mailbox,
        aac_srcv_set_mailbox,
        aac_src_send_command
};

ddi_device_acc_attr_t aac_acc_attr = {
        DDI_DEVICE_ATTR_V0,
        DDI_DEVICE_ATTR_V1,
        DDI_STRUCTURE_LE_ACC,
        DDI_STRICTORDER_ACC,
        DDI_DEFAULT_ACC
};
```

_____***unchanged_portion_omitted_***

```
struct aac_drinfo {
        struct aac_softstate *softs;
        int tgt;
        int lun;
        int event;
};

static int aac_tick = AAC_DEFAULT_TICK; /* tick for the internal timer */
static uint32_t aac_timebase = 0;       /* internal timer in seconds */
static uint32_t aac_sync_time = 0;      /* next time to sync. with firmware */
static int aac_fm_valid = 0;            /* FMA support */

/*
 * Warlock directives
 *
 * Different variables with the same types have to be protected by the
 * same mutex; otherwise, warlock will complain with "variables don't
```

```
 733  * seem to be protected consistently". For example,
 734  * aac_softstate::{q_wait, q_comp} are type of aac_cmd_queue, and protected
 735  * by aac_softstate::{io_lock, q_comp_mutex} respectively. We have to
 736  * declare them as protected explictly at aac_cmd_dequeue().
 737  */
 738 _NOTE(SCHEME_PROTECTS_DATA("unique per pkt", scsi_pkt scsi_cdb scsi_status \
 739     scsi_arq_status scsi_descr_sense_hdr scsi_information_sense_descr \
 740     mode_format mode_geometry mode_header aac_cmd))
 741 _NOTE(SCHEME_PROTECTS_DATA("unique per aac_cmd", aac_fib ddi_dma_cookie_t \
 742     aac_sge))
 743 _NOTE(SCHEME_PROTECTS_DATA("unique per aac_fib", aac_blockread aac_blockwrite \
 744     aac_blockread64 aac_raw_io aac_sg_entry aac_sg_entry64 aac_sg_entryraw \
 745     aac_sg_table aac_srb))
 746 _NOTE(SCHEME_PROTECTS_DATA("unique to sync fib and cdb", scsi_inquiry))
 747 _NOTE(SCHEME_PROTECTS_DATA("stable data", scsi_device scsi_address))
 748 _NOTE(SCHEME_PROTECTS_DATA("unique to dr event", aac_drinfo))
 749 _NOTE(SCHEME_PROTECTS_DATA("unique to scsi_transport", buf))

 751 int
 752 _init(void)
 753 {
 754         int rval = 0;

 756 #ifdef AAC_DEBUG
 697 #ifdef DEBUG
 757         mutex_init(&aac_prt_mutex, NULL, MUTEX_DRIVER, NULL);
 758 #endif
 759         DBCALLED(NULL, 1);

 761         if ((rval = ddi_soft_state_init((void *)&aac_softstatep,
 762             sizeof (struct aac_softstate), 0)) != 0)
 763                 goto error;

 765         if ((rval = scsi_hba_init(&aac_modlinkage)) != 0) {
 766                 ddi_soft_state_fini((void *)&aac_softstatep);
 767                 goto error;
 768         }

 770         if ((rval = mod_install(&aac_modlinkage)) != 0) {
 771                 ddi_soft_state_fini((void *)&aac_softstatep);
 772                 scsi_hba_fini(&aac_modlinkage);
 773                 goto error;
 774         }
 775         return (rval);

 777 error:
 778         AACDB_PRINT(NULL, CE_WARN, "Mod init error!");
 779 #ifdef AAC_DEBUG
 720 #ifdef DEBUG
 780         mutex_destroy(&aac_prt_mutex);
 781 #endif
 782         return (rval);
 783 }
______unchanged_portion_omitted_

 792 /*
 793  * An HBA driver cannot be unload unless you reboot,
 794  * so this function will be of no use.
 795  */
 796 int
 797 _fini(void)
 798 {
 799         int rval;

 801         DBCALLED(NULL, 1);
```



```
 803         if ((rval = mod_remove(&aac_modlinkage)) != 0)
 804                 goto error;

 806         scsi_hba_fini(&aac_modlinkage);
 807         ddi_soft_state_fini((void *)&aac_softstatep);
 808 #ifdef AAC_DEBUG
 749 #ifdef DEBUG
 809         mutex_destroy(&aac_prt_mutex);
 810 #endif
 811         return (0);

 813 error:
 814         AACDB_PRINT(NULL, CE_WARN, "AAC is busy, cannot unload!");
 815         return (rval);
 816 }

 818 static int
 819 aac_attach(dev_info_t *dip, ddi_attach_cmd_t cmd)
 820 {
 821         int instance, i, major, minor;
 762         int instance, i;
 822         struct aac_softstate *softs = NULL;
 823         int attach_state = 0;
 824         char *data;

 826         DBCALLED(NULL, 1);

 828         switch (cmd) {
 829         case DDI_ATTACH:
 830                 break;
 831         case DDI_RESUME:
 832                 return (DDI_FAILURE);
 833         default:
 834                 return (DDI_FAILURE);
 835         }

 837         instance = ddi_get_instance(dip);

 839         /* Get soft state */
 840         if (ddi_soft_state_zalloc(aac_softstatep, instance) != DDI_SUCCESS) {
 841                 AACDB_PRINT(softs, CE_WARN, "Cannot alloc soft state");
 842                 goto error;
 843         }
 844         softs = ddi_get_soft_state(aac_softstatep, instance);
 845         attach_state |= AAC_ATTACH_SOFTSTATE_ALLOCED;

 847         softs->instance = instance;
 848         softs->devinfo_p = dip;
 849         softs->buf_dma_attr = softs->addr_dma_attr = aac_dma_attr;
 850         softs->addr_dma_attr.dma_attr_granular = 1;
 792         softs->acc_attr = aac_acc_attr;
 793         softs->reg_attr = aac_acc_attr;
 851         softs->card = AAC_UNKNOWN_CARD;
 852 #ifdef AAC_DEBUG
 795 #ifdef DEBUG
 853         softs->debug_flags = aac_debug_flags;
 797         softs->debug_fib_flags = aac_debug_fib_flags;
 854 #endif

 800         /* Initialize FMA */
 801         aac_fm_init(softs);

 856         /* Check the card type */
 857         if (aac_check_card_type(softs) == AACERR) {
 858                 AACDB_PRINT(softs, CE_WARN, "Card not supported");
 859                 goto error;
```

```
860         }
861         /* We have found the right card and everything is OK */
862         attach_state |= AAC_ATTACH_CARD_DETECTED;

864         /*
865          * Initialize FMA
866          */
867 #ifdef FORCE_FM_SUPPORT
868         aac_fm_valid = 1;
869 #else
870         aac_fm_valid = 0;
871         /* check OS version */
872         data = (char *)utsname.release;
873         major = aac_atoi(&data);
874         if (*data == '.') {
875                 data++;
876                 minor = aac_atoi(&data);
877                 /* enable FMA support in Solaris 11 first */
878                 if (major >= 5 && minor >= 11)
879                         aac_fm_valid = 1;
880         }
881 #endif
882         aac_fm_init(softs);

884         /* Map PCI mem space */
885         if (ddi_regs_map_setup(dip, 1,
886             (caddr_t *)&softs->pci_mem_base_vaddr[0], 0,
887             softs->map_size_min, &aac_acc_attr,
888             &softs->pci_mem_handle[0]) != DDI_SUCCESS)
813             (caddr_t *)&softs->pci_mem_base_vaddr, 0,
814             softs->map_size_min, &softs->reg_attr,
815             &softs->pci_mem_handle) != DDI_SUCCESS)
889                 goto error;
890         if (softs->hwif == AAC_HWIF_SRC) {
891                 /*
892                  * Map whole space because map_size_min is not enough
893                  * for NEMER/ARK family with APRE
894                  */
895                 if (ddi_regs_map_setup(dip, 2,
896                     (caddr_t *)&softs->pci_mem_base_vaddr[1], 0,
897                     softs->hwif == AAC_HWIF_SRC ? AAC_MAP_SIZE_MIN_SRC_BAR1 : AA
898                         &aac_acc_attr,
899                     &softs->pci_mem_handle[1]) != DDI_SUCCESS) {
900                         ddi_regs_map_free(&softs->pci_mem_handle[0]);
901                         goto error;
902                 }
903         } else {
904                 /* Setup BAR1 related values */
905                 softs->pci_mem_handle[1] = softs->pci_mem_handle[0];
906                 softs->pci_mem_base_vaddr[1] = softs->pci_mem_base_vaddr[0];
907         }

909         softs->map_size = softs->map_size_min;
910         attach_state |= AAC_ATTACH_PCI_MEM_MAPPED;

912         AAC_SET_INTR(softs, 0);
821         AAC_DISABLE_INTR(softs);

914         if (ddi_intr_hilevel(dip, 0)) {
915                 AACDB_PRINT(softs, CE_WARN,
916                     "High level interrupt is not supported!");
917                 goto error;
918         }

920         /* Init mutexes */
921         if (ddi_get_iblock_cookie(dip, 0, &softs->iblock_cookie) !=
```



```
922             DDI_SUCCESS) {
923                 AACDB_PRINT(softs, CE_WARN,
924                     "Can not get interrupt block cookie!");
925                 goto error;
926         }
927         mutex_init(&softs->q_comp_mutex, NULL,
928             MUTEX_DRIVER, (void *)softs->iblock_cookie);
929         cv_init(&softs->event, NULL, CV_DRIVER, NULL);
823         /* Init mutexes and condvars */
824         mutex_init(&softs->io_lock, NULL, MUTEX_DRIVER,
825             DDI_INTR_PRI(softs->intr_pri));
826         mutex_init(&softs->q_comp_mutex, NULL, MUTEX_DRIVER,
827             DDI_INTR_PRI(softs->intr_pri));
828         mutex_init(&softs->time_mutex, NULL, MUTEX_DRIVER,
829             DDI_INTR_PRI(softs->intr_pri));
830         mutex_init(&softs->ev_lock, NULL, MUTEX_DRIVER,
831             DDI_INTR_PRI(softs->intr_pri));
930         mutex_init(&softs->aifq_mutex, NULL,
931             MUTEX_DRIVER, (void *)softs->iblock_cookie);
932         cv_init(&softs->aifv, NULL, CV_DRIVER, NULL);
833             MUTEX_DRIVER, DDI_INTR_PRI(softs->intr_pri));
834         cv_init(&softs->event, NULL, CV_DRIVER, NULL);
835         cv_init(&softs->sync_fib_cv, NULL, CV_DRIVER, NULL);
933         cv_init(&softs->drain_cv, NULL, CV_DRIVER, NULL);
934         mutex_init(&softs->io_lock, NULL, MUTEX_DRIVER,
935             (void *)softs->iblock_cookie);
837         cv_init(&softs->event_wait_cv, NULL, CV_DRIVER, NULL);
838         cv_init(&softs->event_disp_cv, NULL, CV_DRIVER, NULL);
839         cv_init(&softs->aifq_cv, NULL, CV_DRIVER, NULL);
936         attach_state |= AAC_ATTACH_KMUTEX_INITED;

842         /* Init the cmd queues */
843         for (i = 0; i < AAC_CMDQ_NUM; i++)
844                 aac_cmd_initq(&softs->q_wait[i]);
845         aac_cmd_initq(&softs->q_busy);
846         aac_cmd_initq(&softs->q_comp);

938         /* Check for legacy device naming support */
849         softs->legacy = 1; /* default to use legacy name */
939         if ((ddi_prop_lookup_string(DDI_DEV_T_ANY, dip, 0,
940             "legacy-name-enable", &data) == DDI_SUCCESS)) {
941                 if (strcmp(data, "yes") == 0) {
942                         AACDB_PRINT(softs, CE_NOTE, "aac-legacy-name enabled");
943                         softs->legacy = 1;
852                 if (strcmp(data, "no") == 0) {
853                         AACDB_PRINT(softs, CE_NOTE, "legacy-name disabled");
854                         softs->legacy = 0;
944                 }
945                 ddi_prop_free(data);
946         }
947         /* Check for sync. transfer mode */
948         if ((ddi_prop_lookup_string(DDI_DEV_T_ANY, dip, 0,
949             "enforce-sync-mode", &data) == DDI_SUCCESS)) {
950                 if (strcmp(data, "yes") == 0) {
951                         AACDB_PRINT(softs, CE_NOTE, "enforce-sync-mode set");
952                         softs->sync_mode = 1;

859         /*
860          * Everything has been set up till now,
861          * we will do some common attach.
862          */
863         mutex_enter(&softs->io_lock);
864         if (aac_common_attach(softs) == AACERR) {
865                 mutex_exit(&softs->io_lock);
866                 goto error;
953                 }
```

```
954                 ddi_prop_free(data);
955         }
956         /* Convert S/G table (NEW_COMM_TYPE2) */
868         mutex_exit(&softs->io_lock);
869         attach_state |= AAC_ATTACH_COMM_SPACE_SETUP;

871         /* Check for buf breakup support */
957         if ((ddi_prop_lookup_string(DDI_DEV_T_ANY, dip, 0,
958             "disable-sgl-conversion", &data) == DDI_SUCCESS)) {
873             "breakup-enable", &data) == DDI_SUCCESS)) {
959                 if (strcmp(data, "yes") == 0) {
960                         AACDB_PRINT(softs, CE_NOTE, "disable-sgl-conversion set"
961                         softs->no_sgl_conv = 1;
875                         AACDB_PRINT(softs, CE_NOTE, "buf breakup enabled");
876                         softs->flags |= AAC_FLAGS_BRKUP;
962                 }
963                 ddi_prop_free(data);
964         }

966         /* Create a taskq for dealing with dr events, suspend taskq */
967         if ((softs->taskq = ddi_taskq_create(dip, "aac_dr_taskq", 1,
968             TASKQ_DEFAULTPRI, 0)) == NULL) {
969                 AACDB_PRINT(softs, CE_WARN, "ddi_taskq_create failed");
970                 goto error;
880         softs->dma_max = softs->buf_dma_attr.dma_attr_maxxfer;
881         if (softs->flags & AAC_FLAGS_BRKUP) {
882                 softs->dma_max = ddi_prop_get_int(DDI_DEV_T_ANY, dip,
883                     DDI_PROP_DONTPASS, "dma-max", softs->dma_max);
971         }
972         ddi_taskq_suspend(softs->taskq);

974         /*
975          * Everything has been set up till now (especially taskq and hba_tran
976          * for auto-enumeration), we will do some common attach.
977          */
978         if (aac_common_attach(softs) == AACERR)
979                 goto error;
980         attach_state |= AAC_ATTACH_COMM_SPACE_SETUP;

982         /* Init the cmd queues */
983         for (i = 0; i < AAC_CMDQ_NUM; i++)
984                 aac_cmd_initq(&softs->q_wait[i]);
985         aac_cmd_initq(&softs->q_busy);
986         aac_cmd_initq(&softs->q_comp);

988         /* allocate and fill the scsi_hba_tran_t structure */
989         if (aac_hba_setup(softs) != AACOK)
990                 goto error;
991         attach_state |= AAC_ATTACH_SCSI_TRAN_SETUP;

993         /* scsi_hba_tran is now initialized, resume taskq */
994         ddi_taskq_resume(softs->taskq);

996         /* Connect interrupt handlers */
997         if (ddi_add_softintr(dip, DDI_SOFTINT_LOW, &softs->softint_id,
998             NULL, NULL, aac_softintr, (caddr_t)softs) != DDI_SUCCESS) {
999                 AACDB_PRINT(softs, CE_WARN,
1000                    "Can not setup soft interrupt handler!");
1001                goto error;
1002        }
1003        attach_state |= AAC_ATTACH_SOFT_INTR_SETUP;

1005        /* Solaris 11 Express: avoid tran_start calls before aac_unhold_bus() */
1006        mutex_enter(&softs->io_lock);

1008        if (ddi_add_intr(dip, 0, &softs->iblock_cookie,
```



```
1009            (ddi_idevice_cookie_t *)0,
1010            (softs->flags & AAC_FLAGS_NEW_COMM) ?
1011            aac_intr_new : aac_intr_old, (caddr_t)softs) != DDI_SUCCESS) {
1012                AACDB_PRINT(softs, CE_WARN, "Can not setup interrupt handler!");
1013                goto error;
1014        }
1015        attach_state |= AAC_ATTACH_HARD_INTR_SETUP;

1017        /* Create devctl/scsi nodes for cfgadm */
1018        if (ddi_create_minor_node(dip, "devctl", S_IFCHR,
1019            INST2DEVCTL(instance), DDI_NT_SCSI_NEXUS, 0) != DDI_SUCCESS) {
1020                AACDB_PRINT(softs, CE_WARN, "failed to create devctl node");
1021                goto error;
1022        }
1023        attach_state |= AAC_ATTACH_CREATE_DEVCTL;

1025        if (ddi_create_minor_node(dip, "scsi", S_IFCHR, INST2SCSI(instance),
1026            DDI_NT_SCSI_ATTACHMENT_POINT, 0) != DDI_SUCCESS) {
1027                AACDB_PRINT(softs, CE_WARN, "failed to create scsi node");
1028                goto error;
1029        }
1030        attach_state |= AAC_ATTACH_CREATE_SCSI;

1032        /* Create aac node for app. to issue ioctls */
1033        if (ddi_create_minor_node(dip, "aac", S_IFCHR, INST2AAC(instance),
1034            DDI_PSEUDO, 0) != DDI_SUCCESS) {
1035                AACDB_PRINT(softs, CE_WARN, "failed to create aac node");
1036                goto error;
1037        }

912         /* Common attach is OK, so we are attached! */
913         softs->state |= AAC_STATE_RUN;

915         /* Create event thread */
916         softs->fibctx_p = &softs->aifctx;
917         if ((softs->event_thread = thread_create(NULL, 0, aac_event_thread,
918             softs, 0, &p0, TS_RUN, minclsyspri)) == NULL) {
919                 AACDB_PRINT(softs, CE_WARN, "aif thread create failed");
920                 softs->state &= ~AAC_STATE_RUN;
921                 goto error;
922         }

1039        aac_unhold_bus(softs, AAC_IOCMD_SYNC | AAC_IOCMD_ASYNC);
1040        softs->state = AAC_STATE_RUN;
1041        mutex_exit(&softs->io_lock);

1043        /* Create a thread for command timeout */
1044        softs->timeout_id = timeout(aac_daemon, (void *)softs,
1045            (60 * drv_usectohz(1000000)));
927         softs->timeout_id = timeout(aac_timer, (void *)softs,
928             (aac_tick * drv_usectohz(1000000)));

1047        /* Common attach is OK, so we are attached! */
1048        AAC_SET_INTR(softs, 1);
1049        ddi_report_dev(dip);
1050        AACDB_PRINT(softs, CE_NOTE, "aac attached ok");
1051        return (DDI_SUCCESS);

1053 error:
1054        if (softs && softs->taskq)
1055                ddi_taskq_destroy(softs->taskq);
1056        if (attach_state & AAC_ATTACH_CREATE_SCSI)
1057                ddi_remove_minor_node(dip, "scsi");
1058        if (attach_state & AAC_ATTACH_CREATE_DEVCTL)
1059                ddi_remove_minor_node(dip, "devctl");
1060        if (attach_state & AAC_ATTACH_COMM_SPACE_SETUP)
```

```
1061                 aac_common_detach(softs);
1062         if (attach_state & AAC_ATTACH_SCSI_TRAN_SETUP) {
1063                 (void) scsi_hba_detach(dip);
1064                 scsi_hba_tran_free(AAC_DIP2TRAN(dip));
1065         }
1066         if (attach_state & AAC_ATTACH_HARD_INTR_SETUP)
1067                 ddi_remove_intr(dip, 0, softs->iblock_cookie);
1068         if (attach_state & AAC_ATTACH_SOFT_INTR_SETUP)
1069                 ddi_remove_softintr(softs->softint_id);
1070         if (attach_state & AAC_ATTACH_KMUTEX_INITED) {
 947                 mutex_destroy(&softs->io_lock);
1071                 mutex_destroy(&softs->q_comp_mutex);
1072                 cv_destroy(&softs->event);
 949                 mutex_destroy(&softs->time_mutex);
 950                 mutex_destroy(&softs->ev_lock);
1073                 mutex_destroy(&softs->aifq_mutex);
1074                 cv_destroy(&softs->aifv);
 952                 cv_destroy(&softs->event);
 953                 cv_destroy(&softs->sync_fib_cv);
1075                 cv_destroy(&softs->drain_cv);
1076                 mutex_destroy(&softs->io_lock);
 955                 cv_destroy(&softs->event_wait_cv);
 956                 cv_destroy(&softs->event_disp_cv);
 957                 cv_destroy(&softs->aifq_cv);
1077         }
1078         if (attach_state & AAC_ATTACH_PCI_MEM_MAPPED) {
1079                 ddi_regs_map_free(&softs->pci_mem_handle[0]);
1080                 if (softs->hwif == AAC_HWIF_SRC)
1081                         ddi_regs_map_free(&softs->pci_mem_handle[1]);
1082         }
 959         if (attach_state & AAC_ATTACH_PCI_MEM_MAPPED)
 960                 ddi_regs_map_free(&softs->pci_mem_handle);
1083         aac_fm_fini(softs);
1084         if (attach_state & AAC_ATTACH_CARD_DETECTED)
1085                 softs->card = AACERR;
1086         if (attach_state & AAC_ATTACH_SOFTSTATE_ALLOCED)
1087                 ddi_soft_state_free(aac_softstatep, instance);
1088         return (DDI_FAILURE);
1089 }

1091 static int
1092 aac_detach(dev_info_t *dip, ddi_detach_cmd_t cmd)
1093 {
1094         scsi_hba_tran_t *tran = AAC_DIP2TRAN(dip);
1095         struct aac_softstate *softs = AAC_TRAN2SOFTS(tran);

1097         DBCALLED(softs, 1);

1099         switch (cmd) {
1100         case DDI_DETACH:
1101                 break;
1102         case DDI_SUSPEND:
1103                 return (DDI_FAILURE);
1104         default:
1105                 return (DDI_FAILURE);
1106         }

1108         mutex_enter(&softs->io_lock);
1109         AAC_SET_INTR(softs, 0);
 987         AAC_DISABLE_INTR(softs);
1110         softs->state = AAC_STATE_STOPPED;

1112         mutex_exit(&softs->io_lock);
1113         (void) untimeout(softs->timeout_id);
1114         mutex_enter(&softs->io_lock);
1115         softs->timeout_id = 0;
```



```
1117         ddi_taskq_destroy(softs->taskq);

1119         ddi_remove_minor_node(dip, "aac");
1120         ddi_remove_minor_node(dip, "scsi");
1121         ddi_remove_minor_node(dip, "devctl");

1123         mutex_exit(&softs->io_lock);
1124         ddi_remove_intr(dip, 0, softs->iblock_cookie);
1125         ddi_remove_softintr(softs->softint_id);

1127         aac_common_detach(softs);

 997         mutex_enter(&softs->io_lock);
1129         (void) scsi_hba_detach(dip);
1130         scsi_hba_tran_free(tran);
1000         mutex_exit(&softs->io_lock);

1132         mutex_destroy(&softs->q_comp_mutex);
1002         /* Stop timer */
1003         mutex_enter(&softs->time_mutex);
1004         if (softs->timeout_id) {
1005                 timeout_id_t tid = softs->timeout_id;
1006                 softs->timeout_id = 0;

1008                 mutex_exit(&softs->time_mutex);
1009                 (void) untimeout(tid);
1010                 mutex_enter(&softs->time_mutex);
1011         }
1012         mutex_exit(&softs->time_mutex);

1014         /* Destroy event thread */
1015         mutex_enter(&softs->ev_lock);
1016         cv_signal(&softs->event_disp_cv);
1017         cv_wait(&softs->event_wait_cv, &softs->ev_lock);
1018         mutex_exit(&softs->ev_lock);

1020         cv_destroy(&softs->aifq_cv);
1021         cv_destroy(&softs->event_disp_cv);
1022         cv_destroy(&softs->event_wait_cv);
1023         cv_destroy(&softs->drain_cv);
1024         cv_destroy(&softs->sync_fib_cv);
1133         cv_destroy(&softs->event);
1134         mutex_destroy(&softs->aifq_mutex);
1135         cv_destroy(&softs->aifv);
1136         cv_destroy(&softs->drain_cv);
1027         mutex_destroy(&softs->ev_lock);
1028         mutex_destroy(&softs->time_mutex);
1029         mutex_destroy(&softs->q_comp_mutex);
1137         mutex_destroy(&softs->io_lock);

1139         ddi_regs_map_free(&softs->pci_mem_handle[0]);
1140         if (softs->hwif == AAC_HWIF_SRC)
1141                 ddi_regs_map_free(&softs->pci_mem_handle[1]);
1032         ddi_regs_map_free(&softs->pci_mem_handle);
1142         aac_fm_fini(softs);
1143         softs->hwif = AAC_HWIF_UNKNOWN;
1144         softs->card = AAC_UNKNOWN_CARD;
1145         ddi_soft_state_free(aac_softstatep, ddi_get_instance(dip));

1147         return (DDI_SUCCESS);
1148 }

1150 /*ARGSUSED*/
1151 static int
1152 aac_reset(dev_info_t *dip, ddi_reset_cmd_t cmd)
```

```
1153 {
1154         struct aac_softstate *softs = AAC_DIP2SOFTS(dip);

1156         DBCALLED(softs, 1);

1158         mutex_enter(&softs->io_lock);
1050         AAC_DISABLE_INTR(softs);
1159         (void) aac_shutdown(softs);
1160         mutex_exit(&softs->io_lock);

1162         return (DDI_SUCCESS);
1163 }

1165 /*
1058  * quiesce(9E) entry point.
1059  *
1060  * This function is called when the system is single-threaded at high
1061  * PIL with preemption disabled. Therefore, this function must not be
1062  * blocked.
1063  *
1064  * This function returns DDI_SUCCESS on success, or DDI_FAILURE on failure.
1065  * DDI_FAILURE indicates an error condition and should almost never happen.
1066  */
1067 static int
1068 aac_quiesce(dev_info_t *dip)
1069 {
1070         struct aac_softstate *softs = AAC_DIP2SOFTS(dip);

1072         if (softs == NULL)
1073                 return (DDI_FAILURE);

1075         _NOTE(ASSUMING_PROTECTED(softs->state))
1076         AAC_DISABLE_INTR(softs);

1078         return (DDI_SUCCESS);
1079 }

1081 /* ARGSUSED */
1082 static int
1083 aac_getinfo(dev_info_t *self, ddi_info_cmd_t infocmd, void *arg,
1084     void **result)
1085 {
1086         int error = DDI_SUCCESS;

1088         switch (infocmd) {
1089         case DDI_INFO_DEVT2INSTANCE:
1090                 *result = (void *)(intptr_t)(MINOR2INST(getminor((dev_t)arg)));
1091                 break;
1092         default:
1093                 error = DDI_FAILURE;
1094         }
1095         return (error);
1096 }

1098 /*
1166  * Bring the controller down to a dormant state and detach all child devices.
1167  * This function is called before detach or system shutdown.
1168  * Note: we can assume that the q_wait on the controller is empty, as we
1169  * won't allow shutdown if any device is open.
1170  */
1171 static int
1172 aac_shutdown(struct aac_softstate *softs)
1173 {
1174         ddi_acc_handle_t acc = softs->sync_slot->fib_acc_handle;
1175         struct aac_close_command *cc = (struct aac_close_command *) \
1176             &softs->sync_slot->fibp->data[0];
```



```
1107         ddi_acc_handle_t acc;
1108         struct aac_close_command *cc;
1177         int rval;

1111         (void) aac_sync_fib_slot_bind(softs, &softs->sync_ac);
1112         acc = softs->sync_ac.slotp->fib_acc_handle;

1114         cc = (struct aac_close_command *)&softs->sync_ac.slotp->fibp->data[0];

1179         ddi_put32(acc, &cc->Command, VM_CloseAll);
1180         ddi_put32(acc, &cc->ContainerId, 0xffffffffu);

1182         /* Flush all caches, set FW to write through mode */
1183         rval = aac_sync_fib(softs, ContainerCommand,
1184             AAC_FIB_SIZEOF(struct aac_close_command));
1122         aac_sync_fib_slot_release(softs, &softs->sync_ac);

1186         AACDB_PRINT(softs, CE_NOTE,
1187             "shutting down aac %s", (rval == AACOK) ? "ok" : "fail");
1188         return (rval);
1189 }

1191 static uint_t
1192 aac_softintr(caddr_t arg)
1193 {
1194         struct aac_softstate *softs = (struct aac_softstate *)arg;
1132         struct aac_softstate *softs = (void *)arg;

1196         if (!AAC_IS_Q_EMPTY(&softs->q_comp)) {
1197                 aac_drain_comp_q(softs);
1198                 return (DDI_INTR_CLAIMED);
1199         } else {
1200                 return (DDI_INTR_UNCLAIMED);
1201         }
1137         return (DDI_INTR_CLAIMED);
1202 }

1204 /*
1205  * Setup auto sense data for pkt
1206  */
1207 static struct scsi_arq_status *
1208 aac_set_arq_common(struct scsi_pkt *pkt)
1143 static void
1144 aac_set_arq_data(struct scsi_pkt *pkt, uchar_t key,
1145     uchar_t add_code, uchar_t qual_code, uint64_t info)
1209 {
1210         struct scsi_arq_status *arqstat;
1147         struct scsi_arq_status *arqstat = (void *)(pkt->pkt_scbp);

1212         pkt->pkt_state |= STATE_GOT_STATUS | STATE_ARQ_DONE;
1149         *pkt->pkt_scbp = STATUS_CHECK; /* CHECK CONDITION */
1150         pkt->pkt_state |= STATE_ARQ_DONE;

1214         arqstat = (struct scsi_arq_status *)(pkt->pkt_scbp);
1215         arqstat->sts_status.sts_chk = 1; /* CHECK CONDITION */
1152         *(uint8_t *)&arqstat->sts_rqpkt_status = STATUS_GOOD;
1216         arqstat->sts_rqpkt_reason = CMD_CMPLT;
1217         arqstat->sts_rqpkt_resid = 0;
1218         arqstat->sts_rqpkt_state =
1219             STATE_GOT_BUS |
1220             STATE_GOT_TARGET |
1221             STATE_SENT_CMD |
1222             STATE_XFERRED_DATA;
1223         arqstat->sts_rqpkt_statistics = 0;

1225         return (arqstat);
```

```
}

static void
aac_copy_arq_data(struct scsi_pkt *pkt, uint8_t *sense_data, int32_t sense_len,
    ddi_acc_handle_t acc)
{
        struct scsi_arq_status *arqstat;

        arqstat = aac_set_arq_common(pkt);

        if (sense_len > sizeof (struct scsi_extended_sense))
                sense_len = sizeof (struct scsi_extended_sense);

        ddi_rep_get8(acc, (uint8_t *)&arqstat->sts_sensedata,
            (uint8_t *)sense_data, sense_len, DDI_DEV_AUTOINCR);
}

static void
aac_set_arq_data(struct scsi_pkt *pkt, uchar_t key,
    uchar_t add_code, uchar_t qual_code, uint64_t info)
{
        struct scsi_arq_status *arqstat;

        arqstat = aac_set_arq_common(pkt);

        if (info <= 0xffffffful) {
                arqstat->sts_sensedata.es_valid = 1;
                arqstat->sts_sensedata.es_class = CLASS_EXTENDED_SENSE;
                arqstat->sts_sensedata.es_code = CODE_FMT_FIXED_CURRENT;
                arqstat->sts_sensedata.es_key = key;
                arqstat->sts_sensedata.es_add_code = add_code;
                arqstat->sts_sensedata.es_qual_code = qual_code;

                arqstat->sts_sensedata.es_info_1 = (info >> 24) & 0xFF;
                arqstat->sts_sensedata.es_info_2 = (info >> 16) & 0xFF;
                arqstat->sts_sensedata.es_info_3 = (info >>  8) & 0xFF;
                arqstat->sts_sensedata.es_info_4 = info & 0xFF;
        } else { /* 64-bit LBA */
                struct scsi_descr_sense_hdr *dsp;
                struct scsi_information_sense_descr *isd;

                dsp = (struct scsi_descr_sense_hdr *)&arqstat->sts_sensedata;
                dsp->ds_class = CLASS_EXTENDED_SENSE;
                dsp->ds_code = CODE_FMT_DESCR_CURRENT;
                dsp->ds_key = key;
                dsp->ds_add_code = add_code;
                dsp->ds_qual_code = qual_code;
                dsp->ds_addl_sense_length =
                    sizeof (struct scsi_information_sense_descr);

                isd = (struct scsi_information_sense_descr *)(dsp+1);
                isd->isd_descr_type = DESCR_INFORMATION;
                isd->isd_valid = 1;
                isd->isd_information[0] = (info >> 56) & 0xFF;
                isd->isd_information[1] = (info >> 48) & 0xFF;
                isd->isd_information[2] = (info >> 40) & 0xFF;
                isd->isd_information[3] = (info >> 32) & 0xFF;
                isd->isd_information[4] = (info >> 24) & 0xFF;
                isd->isd_information[5] = (info >> 16) & 0xFF;
                isd->isd_information[6] = (info >>  8) & 0xFF;
                isd->isd_information[7] = (info) & 0xFF;
        }
}

/*
 * Setup auto sense data for HARDWARE ERROR
```



```
 */
static void
aac_set_arq_data_hwerr(struct aac_cmd *acp)
{
        union scsi_cdb *cdbp;
        uint64_t err_blkno;

        cdbp = (union scsi_cdb *)acp->pkt->pkt_cdbp;
        cdbp = (void *)acp->pkt->pkt_cdbp;
        err_blkno = AAC_GETGXADDR(acp->cmdlen, cdbp);
        aac_set_arq_data(acp->pkt, KEY_HARDWARE_ERROR, 0x00, 0x00, err_blkno);
}

/*
 * Setup auto sense data for UNIT ATTENTION
 * Send a command to the adapter in New Comm. interface
 */
/*ARGSUSED*/
static void
aac_set_arq_data_reset(struct aac_softstate *softs, struct aac_cmd *acp)
static int
aac_send_command(struct aac_softstate *softs, struct aac_slot *slotp)
{
        struct aac_container *dvp = (struct aac_container *)acp->dvp;
        uint32_t index, device;

        ASSERT(dvp->dev.type == AAC_DEV_LD);

        if (dvp->reset) {
                dvp->reset = 0;
                aac_set_arq_data(acp->pkt, KEY_UNIT_ATTENTION, 0x29, 0x02, 0);
        index = PCI_MEM_GET32(softs, AAC_IQUE);
        if (index == 0xffffffffUL) {
                index = PCI_MEM_GET32(softs, AAC_IQUE);
                if (index == 0xffffffffUL)
                        return (AACERR);
        }

        device = index;
        PCI_MEM_PUT32(softs, device,
            (uint32_t)(slotp->fib_phyaddr & 0xffffffful));
        device += 4;
        PCI_MEM_PUT32(softs, device, (uint32_t)(slotp->fib_phyaddr >> 32));
        device += 4;
        PCI_MEM_PUT32(softs, device, slotp->acp->fib_size);
        PCI_MEM_PUT32(softs, AAC_IQUE, index);
        return (AACOK);
}

static void
aac_end_io(struct aac_softstate *softs, struct aac_cmd *acp)
{
        struct aac_device *dvp = acp->dvp;
        int q = AAC_CMDQ(acp);

        if (acp->slotp) { /* outstanding cmd */
                if (!(acp->flags & AAC_CMD_IN_SYNC_SLOT)) {
                aac_release_slot(softs, acp->slotp);
                acp->slotp = NULL;
                }
                if (dvp) {
                        dvp->ncmds[q]--;
                        if (dvp->throttle[q] == AAC_THROTTLE_DRAIN &&
                            dvp->ncmds[q] == 0 && q == AAC_CMDQ_ASYNC)
                                aac_set_throttle(softs, dvp, q,
                                    softs->total_slots);
```

```
                        /*
                         * Setup auto sense data for UNIT ATTENTION
                         * Each lun should generate a unit attention
                         * condition when reset.
                         * Phys. drives are treated as logical ones
                         * during error recovery.
                         */
                        if (dvp->type == AAC_DEV_LD) {
                                struct aac_container *ctp =
                                    (struct aac_container *)dvp;
                                if (ctp->reset == 0)
                                        goto noreset;

                                AACDB_PRINT(softs, CE_NOTE,
                                    "Unit attention: reset");
                                ctp->reset = 0;
                                aac_set_arq_data(acp->pkt, KEY_UNIT_ATTENTION,
                                    0x29, 0x02, 0);
                }
                }
noreset:
                softs->bus_ncmds[q]--;
                (void) aac_cmd_delete(&softs->q_busy, acp);
                aac_cmd_delete(&softs->q_busy, acp);
        } else { /* cmd in waiting queue */
                aac_cmd_delete(&softs->q_wait[q], acp);
        }

        acp->cur_segment++;
        if (acp->cur_segment >= acp->segment_cnt) {
                if (!(acp->flags & (AAC_CMD_NO_CB | AAC_CMD_NO_INTR | AAC_CMD_AI
        if (!(acp->flags & (AAC_CMD_NO_CB | AAC_CMD_NO_INTR))) { /* async IO */
                        mutex_enter(&softs->q_comp_mutex);
                        aac_cmd_enqueue(&softs->q_comp, acp);
                        mutex_exit(&softs->q_comp_mutex);
                } else if (acp->flags & AAC_CMD_NO_CB) { /* sync IO */
                        cv_broadcast(&softs->event);
                }
        } else {
                acp->flags &= AAC_CMD_CONSISTENT | AAC_CMD_DMA_PARTIAL | \
                        AAC_CMD_BUF_READ | AAC_CMD_BUF_WRITE | AAC_CMD_DMA_VALID
                acp->timeout = acp->pkt->pkt_time;
                if (acp->pkt->pkt_flags & FLAG_NOINTR)
                        acp->flags |= AAC_CMD_NO_INTR;
                aac_do_io(softs, acp);
        }
}

static void
aac_handle_io(struct aac_softstate *softs, struct aac_slot *slotp, int fast)
aac_handle_io(struct aac_softstate *softs, int index)
{
        struct aac_slot *slotp;
        struct aac_cmd *acp;
        uint32_t fast;

        fast = index & AAC_SENDERADDR_MASK_FAST_RESPONSE;
        index >>= 2;

        /* Make sure firmware reported index is valid */
        ASSERT(index >= 0 && index < softs->total_slots);
        slotp = &softs->io_slot[index];
        ASSERT(slotp->index == index);
        acp = slotp->acp;

        if (acp == NULL || acp->slotp != slotp) {
```



```
                AACDB_PRINT(softs, CE_NOTE, "Command already freed, discarding")
                cmn_err(CE_WARN,
                    "Firmware error: invalid slot index received from FW");
                return;
        }

        acp->flags |= AAC_CMD_CMPLT;
        (void) ddi_dma_sync(slotp->fib_dma_handle, 0, 0, DDI_DMA_SYNC_FORCPU);

        if (aac_check_dma_handle(slotp->fib_dma_handle) == DDI_SUCCESS) {
                /*
                 * For fast response IO, the firmware do not return any FIB
                 * data, so we need to fill in the FIB status and state so that
                 * FIB users can handle it correctly.
                 */
                if (fast) {
                        uint32_t state;

                        state = ddi_get32(slotp->fib_acc_handle,
                            &slotp->fibp->Header.XferState);
                        /*
                         * Update state for CPU not for device, no DMA sync
                         * needed
                         */
                        ddi_put32(slotp->fib_acc_handle,
                            &slotp->fibp->Header.XferState,
                            state | AAC_FIBSTATE_DONEADAP);
                        ddi_put32(slotp->fib_acc_handle,
                            (uint32_t *)&slotp->fibp->data[0], ST_OK);
                        acp->flags |= AAC_CMD_FASTRESP;
                        (void *)&slotp->fibp->data[0], ST_OK);
                }

                /* Handle completed ac */
                if (acp->cur_segment + 1 >= acp->segment_cnt)
                        acp->ac_comp(softs, acp);
        } else {
                aac_fm_service_impact(softs->devinfo_p, DDI_SERVICE_UNAFFECTED);
                ddi_fm_service_impact(softs->devinfo_p, DDI_SERVICE_UNAFFECTED);
                acp->flags |= AAC_CMD_ERR;
                if (acp->pkt) {
                        acp->pkt->pkt_reason = CMD_TRAN_ERR;
                        acp->pkt->pkt_statistics = 0;
                }
                acp->cur_segment = acp->segment_cnt - 1;
        }
        aac_end_io(softs, acp);
}

/*
 * Interrupt handler for New Comm. Type1 interface
 */
static int
aac_process_intr_new_type1(struct aac_softstate *softs)
{
        struct aac_cmd *acp;
        uint32_t bellbits, bellbits_shifted, index, handle;
        int isFastResponse;
        int our_interrupt = 0;

        bellbits = PCI_MEM_GET32(softs, 0, AAC_SRC_ODBR_R);
        if (bellbits & AAC_DB_RESPONSE_SENT_NS) {
                bellbits = AAC_DB_RESPONSE_SENT_NS;
                /* handle async. status */
                our_interrupt = 1;
                index = softs->aac_host_rrq_idx;
```

```
1432                 for (;;) {
1433                         isFastResponse = 0;
1434                         /* remove toggle bit (31) */
1435                         handle = (softs->comm_space->aac_host_rrq[index] & 0x7ff
1436                         /* check fast response bit (30) */
1437                         if (handle & 0x40000000)
1438                                 isFastResponse = 1;
1440                         handle &= 0x0000ffff;
1441                         if (handle == 0)
1442                                 break;
1444                         acp = softs->io_slot[handle-1].acp;
1445                         ASSERT((handle-1) >= 0 && (handle-1) < softs->total_slot
1446                         aac_handle_io(softs, &softs->io_slot[handle-1], isFastRe
1448                         softs->comm_space->aac_host_rrq[index++] = 0;
1449                         if (index == softs->aac_max_fibs)
1450                                 index = 0;
1451                         softs->aac_host_rrq_idx = index;
1453                         if (acp && (acp->flags & AAC_CMD_AIF)) {
1454                                 if (acp->flags & AAC_CMD_AIF_NOMORE) {
1455                                         kmem_free(acp, sizeof(struct aac_cmd));
1456                                 } else {
1457                                         acp->timeout = AAC_AIF_TIMEOUT;
1458                                         acp->aac_cmd_fib = aac_cmd_aif_request;
1459                                         acp->ac_comp = aac_aifreq_complete;
1460                                         aac_do_io(softs, acp);
1461                                 }
1462                         }
1463                 }
1464         } else {
1465                 bellbits_shifted = (bellbits >> AAC_SRC_ODR_SHIFT);
1466                 if (bellbits_shifted & AAC_DB_AIF_PENDING) {
1467                         our_interrupt = 1;
1468                         /* handle AIF */
1469                         if ((acp = kmem_zalloc(sizeof(struct aac_cmd), KM_NOSLEE
1470                                 acp->timeout = AAC_AIF_TIMEOUT;
1471                                 acp->aac_cmd_fib = aac_cmd_aif_request;
1472                                 acp->ac_comp = aac_aifreq_complete;
1473                                 aac_do_io(softs, acp);
1474                         }
1475                 } else if (bellbits_shifted & AAC_DB_SYNC_COMMAND) {
1476                         if (softs->sync_mode_slot) {
1477                                 our_interrupt = 1;
1478                                 aac_handle_io(softs, softs->sync_mode_slot, isFa
1479                                 softs->sync_slot_busy = 0;
1480                         }
1481                 }
1482         }
1483         if (our_interrupt)
1484                 PCI_MEM_PUT32(softs, 0, AAC_SRC_ODBR_C, bellbits);
1486     /*
1487      * Process waiting cmds before start new ones to
1488      * ensure first IOs are serviced first.
1489      */
1490     aac_start_waiting_io(softs);
1491     return (our_interrupt ? AAC_DB_COMMAND_READY:0);
1492 }
1494 /*
1495  * Interrupt handler for New Comm. interface
1496  * New Comm. interface use a different mechanism for interrupt. No explict
1497  * message queues, and driver need only accesses the mapped PCI mem space to
```



```
1498  * find the completed FIB or AIF.
1499  */
1500 static int
1501 aac_process_intr_new(struct aac_softstate *softs)
1502 {
1503         uint32_t index;
1504     int fast;
1506         index = AAC_OUTB_GET(softs);
1507         if (index == 0xffffffffu)
1508                 index = AAC_OUTB_GET(softs);
1509         if (aac_check_acc_handle(softs->pci_mem_handle[0]) != DDI_SUCCESS) {
1510                 aac_fm_service_impact(softs->devinfo_p, DDI_SERVICE_UNAFFECTED);
1511                 return (DDI_INTR_UNCLAIMED);
1368         if (aac_check_acc_handle(softs->pci_mem_handle) != DDI_SUCCESS) {
1369                 ddi_fm_service_impact(softs->devinfo_p, DDI_SERVICE_UNAFFECTED);
1370                 return (0);
1512         }
1513         if (index != 0xffffffffu) {
1514                 do {
1515                         if ((index & AAC_SENDERADDR_MASK_AIF) == 0) {
1516                                 fast= index & AAC_SENDERADDR_MASK_FAST_RESPONSE;
1517                                 index >>= 2;
1518                                 ASSERT(index >= 0 && index < softs->total_slots)
1519                                 aac_handle_io(softs, &softs->io_slot[index], fas
1375                                 aac_handle_io(softs, index);
1520                         } else if (index != 0xfffffffeul) {
1521                                 struct aac_fib *fibp;   /* FIB in AIF queue */
1522                                 uint16_t fib_size, fib_size0;
1378                                 uint16_t fib_size;
1524                                 /*
1525                                  * 0xfffffffe means that the controller wants
1526                                  * more work, ignore it for now. Otherwise,
1527                                  * AIF received.
1528                                  */
1529                                 index &= ~2;
1531                                 mutex_enter(&softs->aifq_mutex);
1532                                 /*
1533                                  * Copy AIF from adapter to the empty AIF slot
1534                                  */
1535                                 fibp = &softs->aifq[softs->aifq_idx].d;
1536                                 fib_size0 = PCI_MEM_GET16(softs, 0, index + \
1387                                 fibp = (struct aac_fib *)(softs-> \
1388                                     pci_mem_base_vaddr + index);
1389                                 fib_size = PCI_MEM_GET16(softs, index + \
1537                                     offsetof(struct aac_fib, Header.Size));
1538                                 fib_size = (fib_size0 > AAC_FIB_SIZE) ?
1539                                     AAC_FIB_SIZE : fib_size0;
1540                                 PCI_MEM_REP_GET8(softs, 0, index, fibp,
1541                                     fib_size);
1543                                 if (aac_check_acc_handle(softs-> \
1544                                     pci_mem_handle[0]) == DDI_SUCCESS)
1545                                         (void) aac_handle_aif(softs, fibp);
1546                                 else
1547                                         aac_fm_service_impact(softs->devinfo_p,
1548                                             DDI_SERVICE_UNAFFECTED);
1549                                 mutex_exit(&softs->aifq_mutex);
1392                                 aac_save_aif(softs, softs->pci_mem_handle,
1393                                     fibp, fib_size);
1551                                 /*
1552                                  * AIF memory is owned by the adapter, so let it
1553                                  * know that we are done with it.
```

```
1554                                 */
1555                                AAC_OUTB_SET(softs, index);
1556                                AAC_STATUS_CLR(softs, AAC_DB_RESPONSE_READY);
1557                        }

1559                        index = AAC_OUTB_GET(softs);
1560                } while (index != 0xffffffffful);

1562                /*
1563                 * Process waiting cmds before start new ones to
1564                 * ensure first IOs are serviced first.
1565                 */
1566                aac_start_waiting_io(softs);
1567                return (AAC_DB_COMMAND_READY);
1568        } else {
1569                return (0);
1570        }
1571 }

1573 static uint_t
1574 aac_intr_new(caddr_t arg)
1575 {
1576         struct aac_softstate *softs = (struct aac_softstate *)arg;
1577         int status;
1420         struct aac_softstate *softs = (void *)arg;
1578         uint_t rval;

1580         mutex_enter(&softs->io_lock);
1581         if ((softs->flags & AAC_FLAGS_NEW_COMM_TYPE1) ||
1582                 (softs->flags & AAC_FLAGS_NEW_COMM_TYPE2) ||
1583                 (softs->flags & AAC_FLAGS_NEW_COMM_TYPE34))
1584                 status = aac_process_intr_new_type1(softs);
1585         else
1586                 status = aac_process_intr_new(softs);
1587         if (status)
1424         if (aac_process_intr_new(softs))
1588                 rval = DDI_INTR_CLAIMED;
1589         else
1590                 rval = DDI_INTR_UNCLAIMED;
1591         mutex_exit(&softs->io_lock);

1593         aac_drain_comp_q(softs);
1594         return (rval);
1595 }

1597 /*
1598  * Interrupt handler for old interface
1599  * Explicit message queues are used to send FIB to and get completed FIB from
1600  * the adapter. Driver and adapter maitain the queues in the producer/consumer
1601  * manner. The driver has to query the queues to find the completed FIB.
1602  */
1603 static int
1604 aac_process_intr_old(struct aac_softstate *softs)
1605 {
1606         uint16_t status;

1608         status = AAC_STATUS_GET(softs);
1609         if (aac_check_acc_handle(softs->pci_mem_handle[0]) != DDI_SUCCESS) {
1610                 aac_fm_service_impact(softs->devinfo_p, DDI_SERVICE_UNAFFECTED);
1446         if (aac_check_acc_handle(softs->pci_mem_handle) != DDI_SUCCESS) {
1447                 ddi_fm_service_impact(softs->devinfo_p, DDI_SERVICE_UNAFFECTED);
1611                 return (DDI_INTR_UNCLAIMED);
1612         }
1613         if (status & AAC_DB_RESPONSE_READY) {
1614                 int slot_idx, fast;
1451                 int slot_idx;
```



```
1616                /* ACK the intr */
1617                AAC_STATUS_CLR(softs, AAC_DB_RESPONSE_READY);
1618                (void) AAC_STATUS_GET(softs);
1619                while (aac_fib_dequeue(softs, AAC_HOST_NORM_RESP_Q,
1620                    &slot_idx) == AACOK) {
1621                        fast = slot_idx & AAC_SENDERADDR_MASK_FAST_RESPONSE;
1622                        slot_idx >>= 2;
1623                        ASSERT(slot_idx >= 0 && slot_idx < softs->total_slots);
1624                        aac_handle_io(softs, &softs->io_slot[slot_idx], fast);
1625                }
1457                    &slot_idx) == AACOK)
1458                        aac_handle_io(softs, slot_idx);

1627                /*
1628                 * Process waiting cmds before start new ones to
1629                 * ensure first IOs are serviced first.
1630                 */
1631                aac_start_waiting_io(softs);
1632                return (AAC_DB_RESPONSE_READY);
1633        } else if (status & AAC_DB_COMMAND_READY) {
1634                int aif_idx;

1636                AAC_STATUS_CLR(softs, AAC_DB_COMMAND_READY);
1637                (void) AAC_STATUS_GET(softs);
1638                if (aac_fib_dequeue(softs, AAC_HOST_NORM_CMD_Q, &aif_idx) ==
1639                    AACOK) {
1640                        ddi_acc_handle_t acc = softs->comm_space_acc_handle;
1641                        struct aac_fib *fibp;   /* FIB in AIF queue */
1642                        struct aac_fib *fibp0;  /* FIB in communication space */
1643                        uint16_t fib_size, fib_size0;
1474                        struct aac_fib *fibp;   /* FIB in communication space */
1475                        uint16_t fib_size;
1644                        uint32_t fib_xfer_state;
1645                        uint32_t addr, size;

1647                        ASSERT((aif_idx >= 0) && (aif_idx < AAC_ADAPTER_FIBS));

1649 #define AAC_SYNC_AIF(softs, aif_idx, type) \
1650         { (void) ddi_dma_sync((softs)->comm_space_dma_handle, \
1651             offsetof(struct aac_comm_space, \
1652             adapter_fibs[(aif_idx)]), AAC_FIB_SIZE, \
1653             (type)); }

1655                        mutex_enter(&softs->aifq_mutex);
1656                        /* Copy AIF from adapter to the empty AIF slot */
1657                        fibp = &softs->aifq[softs->aifq_idx].d;
1658                        AAC_SYNC_AIF(softs, aif_idx, DDI_DMA_SYNC_FORCPU);
1659                        fibp0 = &softs->comm_space->adapter_fibs[aif_idx];
1660                        fib_size0 = ddi_get16(acc, &fibp0->Header.Size);
1661                        fib_size = (fib_size0 > AAC_FIB_SIZE) ?
1662                            AAC_FIB_SIZE : fib_size0;
1663                        ddi_rep_get8(acc, (uint8_t *)fibp, (uint8_t *)fibp0,
1664                            fib_size, DDI_DEV_AUTOINCR);
1489                        fibp = &softs->comm_space->adapter_fibs[aif_idx];
1490                        fib_size = ddi_get16(acc, &fibp->Header.Size);

1666                        (void) aac_handle_aif(softs, fibp);
1667                        mutex_exit(&softs->aifq_mutex);
1492                        aac_save_aif(softs, acc, fibp, fib_size);

1669                        /* Complete AIF back to adapter with good status */
1670                        fib_xfer_state = LE_32(fibp->Header.XferState);
1671                        if (fib_xfer_state & AAC_FIBSTATE_FROMADAP) {
1672                                ddi_put32(acc, &fibp0->Header.XferState,
1497                                ddi_put32(acc, &fibp->Header.XferState,
```

```
                                    fib_xfer_state | AAC_FIBSTATE_DONEHOST);
                                ddi_put32(acc, (uint32_t *)&fibp0->data[0],
                                    ST_OK);
                                if (fib_size0 > AAC_FIB_SIZE)
                                        ddi_put16(acc, &fibp0->Header.Size,
ddi_put32(acc, (void *)&fibp->data[0], ST_OK);
if (fib_size > AAC_FIB_SIZE)
                                        ddi_put16(acc, &fibp->Header.Size,
                                            AAC_FIB_SIZE);
                                AAC_SYNC_AIF(softs, aif_idx,
                                    DDI_DMA_SYNC_FORDEV);
                        }

                        /* Put the AIF response on the response queue */
                        addr = ddi_get32(acc,
                            &softs->comm_space->adapter_fibs[aif_idx]. \
                            Header.SenderFibAddress);
                        size = (uint32_t)ddi_get16(acc,
                            &softs->comm_space->adapter_fibs[aif_idx]. \
                            Header.Size);
                        ddi_put32(acc,
                            &softs->comm_space->adapter_fibs[aif_idx]. \
                            Header.a.ReceiverFibAddress, addr);
                            Header.ReceiverFibAddress, addr);
                        if (aac_fib_enqueue(softs, AAC_ADAP_NORM_RESP_Q,
                            addr, size) == AACERR)
                                cmn_err(CE_NOTE, "!AIF ack failed");
                }
                return (AAC_DB_COMMAND_READY);
        } else if (status & AAC_DB_PRINTF_READY) {
                /* ACK the intr */
                AAC_STATUS_CLR(softs, AAC_DB_PRINTF_READY);
                (void) AAC_STATUS_GET(softs);
                (void) ddi_dma_sync(softs->comm_space_dma_handle,
                    offsetof(struct aac_comm_space, adapter_print_buf),
                    AAC_ADAPTER_PRINT_BUFSIZE, DDI_DMA_SYNC_FORCPU);
                if (aac_check_dma_handle(softs->comm_space_dma_handle) ==
                    DDI_SUCCESS)
                        cmn_err(CE_NOTE, "MSG From Adapter: %s",
                            softs->comm_space->adapter_print_buf);
                else
                        aac_fm_service_impact(softs->devinfo_p,
                        ddi_fm_service_impact(softs->devinfo_p,
                            DDI_SERVICE_UNAFFECTED);
                AAC_NOTIFY(softs, AAC_DB_PRINTF_READY);
                return (AAC_DB_PRINTF_READY);
        } else if (status & AAC_DB_COMMAND_NOT_FULL) {
                /*
                 * Without these two condition statements, the OS could hang
                 * after a while, especially if there are a lot of AIF's to
                 * handle, for instance if a drive is pulled from an array
                 * under heavy load.
                 */
                AAC_STATUS_CLR(softs, AAC_DB_COMMAND_NOT_FULL);
                return (AAC_DB_COMMAND_NOT_FULL);
        } else if (status & AAC_DB_RESPONSE_NOT_FULL) {
                AAC_STATUS_CLR(softs, AAC_DB_COMMAND_NOT_FULL);
                AAC_STATUS_CLR(softs, AAC_DB_RESPONSE_NOT_FULL);
                return (AAC_DB_RESPONSE_NOT_FULL);
        } else {
                return (0);
        }
}

static uint_t
aac_intr_old(caddr_t arg)
```



```
{
        struct aac_softstate *softs = (struct aac_softstate *)arg;
struct aac_softstate *softs = (void *)arg;
        int rval;

        mutex_enter(&softs->io_lock);
        if (aac_process_intr_old(softs))
                rval = DDI_INTR_CLAIMED;
        else
                rval = DDI_INTR_UNCLAIMED;
        mutex_exit(&softs->io_lock);

        aac_drain_comp_q(softs);
        return (rval);
}

/*
 * Query FIXED or MSI interrupts
 */
static int
aac_query_intrs(struct aac_softstate *softs, int intr_type)
{
        dev_info_t *dip = softs->devinfo_p;
        int avail, actual, count;
        int i, flag, ret;

        AACDB_PRINT(softs, CE_NOTE,
            "aac_query_intrs:interrupt type 0x%x", intr_type);

        /* Get number of interrupts */
        ret = ddi_intr_get_nintrs(dip, intr_type, &count);
        if ((ret != DDI_SUCCESS) || (count == 0)) {
                AACDB_PRINT(softs, CE_WARN,
                    "ddi_intr_get_nintrs() failed, ret %d count %d",
                    ret, count);
                return (DDI_FAILURE);
        }

        /* Get number of available interrupts */
        ret = ddi_intr_get_navail(dip, intr_type, &avail);
        if ((ret != DDI_SUCCESS) || (avail == 0)) {
                AACDB_PRINT(softs, CE_WARN,
                    "ddi_intr_get_navail() failed, ret %d avail %d",
                    ret, avail);
                return (DDI_FAILURE);
        }

        AACDB_PRINT(softs, CE_NOTE,
            "ddi_intr_get_nvail returned %d, navail() returned %d",
            count, avail);

        /* Allocate an array of interrupt handles */
        softs->intr_size = count * sizeof (ddi_intr_handle_t);
        softs->htable = kmem_alloc(softs->intr_size, KM_SLEEP);

        if (intr_type == DDI_INTR_TYPE_MSI) {
                count = 1; /* only one vector needed by now */
                flag = DDI_INTR_ALLOC_STRICT;
        } else { /* must be DDI_INTR_TYPE_FIXED */
                flag = DDI_INTR_ALLOC_NORMAL;
        }

        /* Call ddi_intr_alloc() */
        ret = ddi_intr_alloc(dip, softs->htable, intr_type, 0,
            count, &actual, flag);
```

```
1623         if ((ret != DDI_SUCCESS) || (actual == 0)) {
1624                 AACDB_PRINT(softs, CE_WARN,
1625                     "ddi_intr_alloc() failed, ret = %d", ret);
1626                 actual = 0;
1627                 goto error;
1628         }

1630         if (actual < count) {
1631                 AACDB_PRINT(softs, CE_NOTE,
1632                     "Requested: %d, Received: %d", count, actual);
1633                 goto error;
1634         }

1636         softs->intr_cnt = actual;

1638         /* Get priority for first msi, assume remaining are all the same */
1639         if ((ret = ddi_intr_get_pri(softs->htable[0],
1640             &softs->intr_pri)) != DDI_SUCCESS) {
1641                 AACDB_PRINT(softs, CE_WARN,
1642                     "ddi_intr_get_pri() failed, ret = %d", ret);
1643                 goto error;
1644         }

1646         /* Test for high level mutex */
1647         if (softs->intr_pri >= ddi_intr_get_hilevel_pri()) {
1648                 AACDB_PRINT(softs, CE_WARN,
1649                     "aac_query_intrs: Hi level interrupt not supported");
1650                 goto error;
1651         }

1653         return (DDI_SUCCESS);

1655 error:
1656         /* Free already allocated intr */
1657         for (i = 0; i < actual; i++)
1658                 (void) ddi_intr_free(softs->htable[i]);

1660         kmem_free(softs->htable, softs->intr_size);
1661         return (DDI_FAILURE);
1662 }


1665 /*
1666  * Register FIXED or MSI interrupts, and enable them
1667  */
1668 static int
1669 aac_add_intrs(struct aac_softstate *softs)
1670 {
1671         int i, ret;
1672         int actual;
1673         ddi_intr_handler_t *aac_intr;

1675         actual = softs->intr_cnt;
1676         aac_intr = (ddi_intr_handler_t *)((softs->flags & AAC_FLAGS_NEW_COMM) ?
1677             aac_intr_new : aac_intr_old);

1679         /* Call ddi_intr_add_handler() */
1680         for (i = 0; i < actual; i++) {
1681                 if ((ret = ddi_intr_add_handler(softs->htable[i],
1682                     aac_intr, (caddr_t)softs, NULL)) != DDI_SUCCESS) {
1683                         cmn_err(CE_WARN,
1684                             "ddi_intr_add_handler() failed ret = %d", ret);

1686                         /* Free already allocated intr */
1687                         for (i = 0; i < actual; i++)
1688                                 (void) ddi_intr_free(softs->htable[i]);
```



```
1690                         kmem_free(softs->htable, softs->intr_size);
1691                         return (DDI_FAILURE);
1692                 }
1693         }

1695         if ((ret = ddi_intr_get_cap(softs->htable[0], &softs->intr_cap))
1696             != DDI_SUCCESS) {
1697                 cmn_err(CE_WARN, "ddi_intr_get_cap() failed, ret = %d", ret);

1699                 /* Free already allocated intr */
1700                 for (i = 0; i < actual; i++)
1701                         (void) ddi_intr_free(softs->htable[i]);

1703                 kmem_free(softs->htable, softs->intr_size);
1704                 return (DDI_FAILURE);
1705         }

1707         return (DDI_SUCCESS);
1708 }

1710 /*
1711  * Unregister FIXED or MSI interrupts
1712  */
1713 static void
1714 aac_remove_intrs(struct aac_softstate *softs)
1715 {
1716         int i;

1718         /* Disable all interrupts */
1719         (void) aac_disable_intrs(softs);
1720         /* Call ddi_intr_remove_handler() */
1721         for (i = 0; i < softs->intr_cnt; i++) {
1722                 (void) ddi_intr_remove_handler(softs->htable[i]);
1723                 (void) ddi_intr_free(softs->htable[i]);
1724         }

1726         kmem_free(softs->htable, softs->intr_size);
1727 }

1729 static int
1730 aac_enable_intrs(struct aac_softstate *softs)
1731 {
1732         int rval = AACOK;

1734         if (softs->intr_cap & DDI_INTR_FLAG_BLOCK) {
1735                 /* for MSI block enable */
1736                 if (ddi_intr_block_enable(softs->htable, softs->intr_cnt) !=
1737                     DDI_SUCCESS)
1738                         rval = AACERR;
1739         } else {
1740                 int i;

1742                 /* Call ddi_intr_enable() for legacy/MSI non block enable */
1743                 for (i = 0; i < softs->intr_cnt; i++) {
1744                         if (ddi_intr_enable(softs->htable[i]) != DDI_SUCCESS)
1745                                 rval = AACERR;
1746                 }
1747         }
1748         return (rval);
1749 }

1751 static int
1752 aac_disable_intrs(struct aac_softstate *softs)
1753 {
1754         int rval = AACOK;
```

```
1756         if (softs->intr_cap & DDI_INTR_FLAG_BLOCK) {
1757                 /* Call ddi_intr_block_disable() */
1758                 if (ddi_intr_block_disable(softs->htable, softs->intr_cnt) !=
1759                     DDI_SUCCESS)
1760                         rval = AACERR;
1761         } else {
1762                 int i;

1764                 for (i = 0; i < softs->intr_cnt; i++) {
1765                         if (ddi_intr_disable(softs->htable[i]) != DDI_SUCCESS)
1766                                 rval = AACERR;
1767                 }
1768         }
1769         return (rval);
1770 }

1772 /*
1750  * Set pkt_reason and OR in pkt_statistics flag
1751  */
1752 static void
1753 aac_set_pkt_reason(struct aac_softstate *softs, struct aac_cmd *acp,
1754     uchar_t reason, uint_t stat)
1755 {
1756 #ifndef __lock_lint
1757         _NOTE(ARGUNUSED(softs))
1758 #endif
1759         if (acp->pkt->pkt_reason == CMD_CMPLT)
1760                 acp->pkt->pkt_reason = reason;
1761         acp->pkt->pkt_statistics |= stat;
1762 }

1764 /*
1765  * Handle a finished pkt of soft SCMD
1766  */
1767 static void
1768 aac_soft_callback(struct aac_softstate *softs, struct aac_cmd *acp)
1769 {
1770         ASSERT(acp->pkt);

1772         acp->flags |= AAC_CMD_CMPLT;

1774         acp->pkt->pkt_state |= STATE_GOT_BUS | STATE_GOT_TARGET | \
1775             STATE_SENT_CMD;
1798             STATE_SENT_CMD | STATE_GOT_STATUS;
1776         if (acp->pkt->pkt_state & STATE_XFERRED_DATA)
1777                 acp->pkt->pkt_resid = 0;

1779         /* AAC_CMD_NO_INTR means no complete callback */
1780         if (!(acp->flags & AAC_CMD_NO_INTR)) {
1781                 mutex_enter(&softs->q_comp_mutex);
1782                 aac_cmd_enqueue(&softs->q_comp, acp);
1783                 mutex_exit(&softs->q_comp_mutex);
1784                 ddi_trigger_softintr(softs->softint_id);
1785         }
1786 }

1788 /*
1789  * Handlers for completed IOs, common to aac_intr_new() and aac_intr_old()
1790  */

1792 /*
1793  * Handle completed logical device IO command
1794  */
1795 /*ARGSUSED*/
1796 static void
```



```
1797 aac_ld_complete(struct aac_softstate *softs, struct aac_cmd *acp)
1798 {
1799         struct aac_slot *slotp = acp->slotp;
1800         struct aac_blockread_response *resp;
1801         uint32_t status;

1803         ASSERT(!(acp->flags & AAC_CMD_SYNC));
1804         ASSERT(!(acp->flags & AAC_CMD_NO_CB));

1829         acp->pkt->pkt_state |= STATE_GOT_STATUS;

1806         /*
1807          * block_read/write has a similar response header, use blockread
1808          * response for both.
1809          */
1810         resp = (struct aac_blockread_response *)&slotp->fibp->data[0];
1811         status = ddi_get32(slotp->fib_acc_handle, &resp->Status);
1812         if (status == ST_OK) {
1813                 acp->pkt->pkt_resid = 0;
1814                 acp->pkt->pkt_state |= STATE_XFERRED_DATA;
1815         } else if (status == ST_NOT_READY) {
1816                 aac_set_pkt_reason(softs, acp, CMD_TIMEOUT, STAT_TIMEOUT);
1817         } else {
1818                 aac_set_arq_data_hwerr(acp);
1819         }
1820 }

1822 /*
1823  * Handle completed phys. device IO command
1824  */
1825 static void
1826 aac_pd_complete(struct aac_softstate *softs, struct aac_cmd *acp)
1827 {
1828         ddi_acc_handle_t acc = acp->slotp->fib_acc_handle;
1829         struct aac_fib *fibp = acp->slotp->fibp;
1830         struct scsi_pkt *pkt = acp->pkt;
1831         struct aac_srb_reply *resp;
1832         uint32_t resp_status, scsi_status;
1833         uint32_t srb_status, data_xfer_length;
1834         uint32_t sense_data_size;
1855         uint32_t resp_status;

1836         ASSERT(!(acp->flags & AAC_CMD_SYNC));
1837         ASSERT(!(acp->flags & AAC_CMD_NO_CB));

1839         resp = (struct aac_srb_reply *)&fibp->data[0];
1840         if (acp->flags & AAC_CMD_FASTRESP) {
1841                 resp_status = ST_OK;
1842                 sense_data_size = 0;
1843                 srb_status = SRB_STATUS_SUCCESS;
1844                 data_xfer_length = 0;
1845                 scsi_status = 0;
1846         } else {
1847                 resp_status = ddi_get32(acc, &resp->status);
1848                 sense_data_size = ddi_get32(acc, &resp->sense_data_size);
1849                 srb_status = ddi_get32(acc, &resp->srb_status);
1850                 data_xfer_length = ddi_get32(acc, &resp->data_xfer_length);
1851                 scsi_status = ddi_get32(acc, &resp->scsi_status);
1852         }

1854         /* First check FIB status */
1855         if (resp_status == ST_OK) {
1856                 if (data_xfer_length) {
1857                         /* When the cmd failed, data_xfer_length is 0 */
1858                         pkt->pkt_state |= STATE_XFERRED_DATA;
1859                         pkt->pkt_resid = acp->bcount - data_xfer_length;
```

```
1860                }
1865                uint32_t scsi_status;
1866                uint32_t srb_status;
1867                uint32_t data_xfer_length;

1869                scsi_status = ddi_get32(acc, &resp->scsi_status);
1870                srb_status = ddi_get32(acc, &resp->srb_status);
1871                data_xfer_length = ddi_get32(acc, &resp->data_xfer_length);

1873                *pkt->pkt_scbp = (uint8_t)scsi_status;
1874                pkt->pkt_state |= STATE_GOT_STATUS;
1875                if (scsi_status == STATUS_GOOD) {
1876                        uchar_t cmd = ((union scsi_cdb *)(void *)
1877                            (pkt->pkt_cdbp))->scc_cmd;

1862                /* Next check SRB status */
1863                switch (srb_status & 0x3f) {
1881                        case SRB_STATUS_DATA_OVERRUN:
1882                                AACDB_PRINT(softs, CE_NOTE, "DATA_OVERRUN: " \
1883                                    "scmd=%d, xfer=%d, buflen=%d",
1884                                    (uint32_t)cmd, data_xfer_length,
1885                                    acp->bcount);

1887                                switch (cmd) {
1888                                case SCMD_READ:
1889                                case SCMD_WRITE:
1890                                case SCMD_READ_G1:
1891                                case SCMD_WRITE_G1:
1892                                case SCMD_READ_G4:
1893                                case SCMD_WRITE_G4:
1894                                case SCMD_READ_G5:
1895                                case SCMD_WRITE_G5:
1896                                        aac_set_pkt_reason(softs, acp,
1897                                            CMD_DATA_OVR, 0);
1898                                        break;
1899                                }
1900                                /*FALLTHRU*/
1864                case SRB_STATUS_ERROR_RECOVERY:
1865                case SRB_STATUS_PENDING:
1866                case SRB_STATUS_SUCCESS:
1904                                /*
1905                                 * pkt_resid should only be calculated if the
1906                                 * status is ERROR_RECOVERY/PENDING/SUCCESS/
1907                                 * OVERRUN/UNDERRUN
1908                                 */
1909                                if (data_xfer_length) {
1910                                        pkt->pkt_state |= STATE_XFERRED_DATA;
1911                                        pkt->pkt_resid = acp->bcount - \
1912                                            data_xfer_length;
1913                                        ASSERT(pkt->pkt_resid >= 0);
1914                                }
1867                        break;
1868                case SRB_STATUS_ABORTED:
1917                                AACDB_PRINT(softs, CE_NOTE,
1918                                    "SRB_STATUS_ABORTED, xfer=%d, resid=%d",
1919                                    data_xfer_length, pkt->pkt_resid);
1869                        aac_set_pkt_reason(softs, acp, CMD_ABORTED,
1870                            STAT_ABORTED);
1871                        break;
1872                case SRB_STATUS_ABORT_FAILED:
1873                        aac_set_pkt_reason(softs, acp, CMD_ABORT_FAIL, 0);
1924                                AACDB_PRINT(softs, CE_NOTE,
1925                                    "SRB_STATUS_ABORT_FAILED, xfer=%d, " \
1926                                    "resid=%d", data_xfer_length,
1927                                    pkt->pkt_resid);
1928                                aac_set_pkt_reason(softs, acp, CMD_ABORT_FAIL,
```



```
1929                                        0);
1874                        break;
1875                case SRB_STATUS_DATA_OVERRUN:
1876                        switch (((union scsi_cdb *)(pkt->pkt_cdbp))->scc_cmd) {
1877                        case SCMD_READ:
1878                        case SCMD_WRITE:
1879                        case SCMD_READ_G1:
1880                        case SCMD_WRITE_G1:
1881                        case SCMD_READ_G4:
1882                        case SCMD_WRITE_G4:
1883                                aac_set_pkt_reason(softs, acp, CMD_DATA_OVR, 0);
1931                        case SRB_STATUS_PARITY_ERROR:
1932                                AACDB_PRINT(softs, CE_NOTE,
1933                                    "SRB_STATUS_PARITY_ERROR, xfer=%d, " \
1934                                    "resid=%d", data_xfer_length,
1935                                    pkt->pkt_resid);
1936                                aac_set_pkt_reason(softs, acp, CMD_PER_FAIL, 0);
1884                                break;
1938                        case SRB_STATUS_NO_DEVICE:
1939                        case SRB_STATUS_INVALID_PATH_ID:
1940                        case SRB_STATUS_INVALID_TARGET_ID:
1941                        case SRB_STATUS_INVALID_LUN:
1942                        case SRB_STATUS_SELECTION_TIMEOUT:
1943 #ifdef DEBUG
1944                                if (AAC_DEV_IS_VALID(acp->dvp)) {
1945                                        AACDB_PRINT(softs, CE_NOTE,
1946                                            "SRB_STATUS_NO_DEVICE(%d), " \
1947                                            "xfer=%d, resid=%d ",
1948                                            srb_status & 0x3f,
1949                                            data_xfer_length, pkt->pkt_resid);
1885                        }
1951 #endif
1952                                aac_set_pkt_reason(softs, acp, CMD_DEV_GONE, 0);
1886                        break;
1887                case SRB_STATUS_COMMAND_TIMEOUT:
1888                case SRB_STATUS_TIMEOUT:
1956                                AACDB_PRINT(softs, CE_NOTE,
1957                                    "SRB_STATUS_COMMAND_TIMEOUT, xfer=%d, " \
1958                                    "resid=%d", data_xfer_length,
1959                                    pkt->pkt_resid);
1889                        aac_set_pkt_reason(softs, acp, CMD_TIMEOUT,
1890                            STAT_TIMEOUT);
1891                        break;
1892                case SRB_STATUS_BUS_RESET:
1964                                AACDB_PRINT(softs, CE_NOTE,
1965                                    "SRB_STATUS_BUS_RESET, xfer=%d, " \
1966                                    "resid=%d", data_xfer_length,
1967                                    pkt->pkt_resid);
1893                        aac_set_pkt_reason(softs, acp, CMD_RESET,
1894                            STAT_BUS_RESET);
1895                        break;
1896                default:
1972                                AACDB_PRINT(softs, CE_NOTE, "srb_status=%d, " \
1973                                    "xfer=%d, resid=%d", srb_status & 0x3f,
1974                                    data_xfer_length, pkt->pkt_resid);
1897                        aac_set_pkt_reason(softs, acp, CMD_TRAN_ERR, 0);
1898                        break;
1899                }
1900        } else if (resp_status == ST_NOT_READY) {
1901                aac_set_pkt_reason(softs, acp, CMD_TIMEOUT, STAT_TIMEOUT);
1978                } else if (scsi_status == STATUS_CHECK) {
1979                        /* CHECK CONDITION */
1980                        struct scsi_arq_status *arqstat =
1981                            (void *)(pkt->pkt_scbp);
1982                        uint32_t sense_data_size;
```

```
1984                        pkt->pkt_state |= STATE_ARQ_DONE;

1986                        *(uint8_t *)&arqstat->sts_rqpkt_status = STATUS_GOOD;
1987                        arqstat->sts_rqpkt_reason = CMD_CMPLT;
1988                        arqstat->sts_rqpkt_resid = 0;
1989                        arqstat->sts_rqpkt_state =
1990                            STATE_GOT_BUS |
1991                            STATE_GOT_TARGET |
1992                            STATE_SENT_CMD |
1993                            STATE_XFERRED_DATA;
1994                        arqstat->sts_rqpkt_statistics = 0;

1996                        sense_data_size = ddi_get32(acc,
1997                            &resp->sense_data_size);
1998                        ASSERT(sense_data_size <= AAC_SENSE_BUFFERSIZE);
1999                        AACDB_PRINT(softs, CE_NOTE,
2000                            "CHECK CONDITION: sense len=%d, xfer len=%d",
2001                            sense_data_size, data_xfer_length);

2003                        if (sense_data_size > SENSE_LENGTH)
2004                                sense_data_size = SENSE_LENGTH;
2005                        ddi_rep_get8(acc, (uint8_t *)&arqstat->sts_sensedata,
2006                            (uint8_t *)resp->sense_data, sense_data_size,
2007                            DDI_DEV_AUTOINCR);
1902                } else {
2009                        AACDB_PRINT(softs, CE_WARN, "invaild scsi status: " \
2010                            "scsi_status=%d, srb_status=%d",
2011                            scsi_status, srb_status);
2012                        aac_set_pkt_reason(softs, acp, CMD_TRAN_ERR, 0);
2013                }
2014        } else {
1903                AACDB_PRINT(softs, CE_NOTE, "SRB failed: fib status %d",
1904                    resp_status);
2017                aac_set_pkt_reason(softs, acp, CMD_TRAN_ERR, 0);
1905        }

1907        /* Finally SCSI status */
1908        if (scsi_status == 2) {
1909                AACDB_PRINT(softs, CE_NOTE, "CHECK CONDITION: len=%d",
1910                    sense_data_size);
1911                ASSERT(sense_data_size <= AAC_SENSE_BUFFERSIZE);
1912                aac_copy_arq_data(pkt, resp->sense_data, sense_data_size, acc);
1913        }
1914 }
```

_____***unchanged_portion_omitted_***

```
1942 /*
2048  * Handle completed sync fib command
2049  */
2050 /*ARGSUSED*/
2051 void
2052 aac_sync_complete(struct aac_softstate *softs, struct aac_cmd *acp)
2053 {
2054 }

2056 /*
1943  * Handle completed Flush command
1944  */
1945 /*ARGSUSED*/
1946 static void
1947 aac_synccache_complete(struct aac_softstate *softs, struct aac_cmd *acp)
1948 {
1949         struct aac_slot *slotp = acp->slotp;
1950         ddi_acc_handle_t acc = slotp->fib_acc_handle;
1951         struct aac_synchronize_reply *resp;
1952         uint32_t status;
```



```
1954         ASSERT(!(acp->flags & AAC_CMD_SYNC));

2070         acp->pkt->pkt_state |= STATE_GOT_STATUS;

1956         resp = (struct aac_synchronize_reply *)&slotp->fibp->data[0];
1957         status = ddi_get32(acc, &resp->Status);
1958         if (status != CT_OK)
1959                 aac_set_arq_data_hwerr(acp);
1960 }

1962 /*
1963  * Handle completed AIF request command
1964  */
1965 /*ARGSUSED*/
1966 static void
1967 aac_aifreq_complete(struct aac_softstate *softs, struct aac_cmd *acp)
2080 aac_startstop_complete(struct aac_softstate *softs, struct aac_cmd *acp)
1968 {
1969         struct aac_slot *slotp = acp->slotp;
1970         ddi_acc_handle_t acc = slotp->fib_acc_handle;
2084         struct aac_Container_resp *resp;
1971         uint32_t status;
1972         struct aac_fib *fibp;   /* FIB in AIF queue */
1973         uint16_t fib_size, fib_size0;

1975         ASSERT(!(acp->flags & AAC_CMD_SYNC));

1977         acp->flags |= AAC_CMD_AIF;
1978         status = ddi_get32(acc, &slotp->fibp->Header.XferState);
1979         if (status & AAC_FIBSTATE_NOMOREAIF) {
1980                 acp->flags |= AAC_CMD_AIF_NOMORE;
1981         } else {
1982                 mutex_enter(&softs->aifq_mutex);
1983                 /*
1984                  * Copy AIF from adapter to the empty AIF slot
1985                  */
1986                 fibp = &softs->aifq[softs->aifq_idx].d;
1987                 fib_size0 = ddi_get16(acc, &slotp->fibp->Header.Size);
1988                 fib_size = (fib_size0 > AAC_FIB_SIZE) ?
1989                         AAC_FIB_SIZE : fib_size0;
1990                 ddi_rep_get8(acc, (uint8_t *)fibp, (uint8_t *)slotp->fibp,
1991                         fib_size, DDI_DEV_AUTOINCR);
2089         acp->pkt->pkt_state |= STATE_GOT_STATUS;

1993                 if (aac_check_acc_handle(softs-> \
1994                         pci_mem_handle[0]) == DDI_SUCCESS)
1995                         (void) aac_handle_aif(softs, fibp);
1996                 else
1997                         aac_fm_service_impact(softs->devinfo_p,
1998                                 DDI_SERVICE_UNAFFECTED);
1999                 mutex_exit(&softs->aifq_mutex);
2091         resp = (struct aac_Container_resp *)&slotp->fibp->data[0];
2092         status = ddi_get32(acc, &resp->Status);
2093         if (status != 0) {
2094                 AACDB_PRINT(softs, CE_WARN, "Cannot start/stop a unit");
2095                 aac_set_arq_data_hwerr(acp);
2000         }
2001 }

2003 /*
2004  * Access PCI space to see if the driver can support the card
2005  */
2006 static int
2007 aac_check_card_type(struct aac_softstate *softs)
2008 {
```

```
2009         ddi_acc_handle_t pci_config_handle;
2010         int card_index;
2011         uint32_t pci_cmd;

2013         /* Map pci configuration space */
2014         if ((pci_config_setup(softs->devinfo_p, &pci_config_handle)) !=
2015             DDI_SUCCESS) {
2016                 AACDB_PRINT(softs, CE_WARN, "Cannot setup pci config space");
2017                 return (AACERR);
2018         }

2020         softs->vendid = pci_config_get16(pci_config_handle, PCI_CONF_VENID);
2021         softs->devid = pci_config_get16(pci_config_handle, PCI_CONF_DEVID);
2022         softs->subvendid = pci_config_get16(pci_config_handle,
2023             PCI_CONF_SUBVENID);
2024         softs->subsysid = pci_config_get16(pci_config_handle,
2025             PCI_CONF_SUBSYSID);

2027         card_index = 0;
2028         while (!CARD_IS_UNKNOWN(card_index)) {
2029                 if ((aac_cards[card_index].vendor == softs->vendid) &&
2030                     (aac_cards[card_index].device == softs->devid) &&
2031                     (aac_cards[card_index].subvendor == softs->subvendid) &&
2032                     (aac_cards[card_index].subsys == softs->subsysid)) {
2033                         break;
2034                 }
2035                 card_index++;
2036         }

2038         softs->card = card_index;
2039         softs->hwif = aac_cards[card_index].hwif;

2041         /*
2042          * Unknown aac card
2043          * do a generic match based on the VendorID and DeviceID to
2044          * support the new cards in the aac family
2045          */
2046         if (CARD_IS_UNKNOWN(card_index)) {
2047                 if (softs->vendid != 0x9005) {
2048                         AACDB_PRINT(softs, CE_WARN,
2049                             "Unknown vendor 0x%x", softs->vendid);
2050                         goto error;
2051                 }
2052                 switch (softs->devid) {
2053                 case 0x285:
2054                         softs->hwif = AAC_HWIF_I960RX;
2055                         break;
2056                 case 0x286:
2057                         softs->hwif = AAC_HWIF_RKT;
2058                         break;
2059                 case 0x28b:
2060                         softs->hwif = AAC_HWIF_SRC;
2061                         break;
2062                 case 0x28c:
2063                 case 0x28d:
2064                 case 0x28f:
2065                         softs->hwif = AAC_HWIF_SRCV;
2066                         break;
2067                 default:
2068                         AACDB_PRINT(softs, CE_WARN,
2069                             "Unknown device \"pci9005,%x\"", softs->devid);
2070                         goto error;
2071                 }
2072         }

2074         /* Set hardware dependent interface */
```



```
2075         switch (softs->hwif) {
2076         case AAC_HWIF_I960RX:
2077                 softs->aac_if = aac_rx_interface;
2078                 softs->map_size_min = AAC_MAP_SIZE_MIN_RX;
2079                 break;
2080         case AAC_HWIF_RKT:
2081                 softs->aac_if = aac_rkt_interface;
2082                 softs->map_size_min = AAC_MAP_SIZE_MIN_RKT;
2083                 break;
2084         case AAC_HWIF_SRC:
2085                 softs->aac_if = aac_src_interface;
2086                 softs->map_size_min = AAC_MAP_SIZE_MIN_SRC_BAR0;
2087                 break;
2088         case AAC_HWIF_SRCV:
2089                 softs->aac_if = aac_srcv_interface;
2090                 softs->map_size_min = AAC_MAP_SIZE_MIN_SRCV_BAR0;
2091                 break;
2092         default:
2093                 AACDB_PRINT(softs, CE_WARN,
2094                     "Unknown hardware interface %d", softs->hwif);
2095                 goto error;
2096         }

2098         /* Set card names */
2099         (void *)strncpy(softs->vendor_name, aac_cards[card_index].vid,
2100             AAC_VENDOR_LEN);
2101         (void *)strncpy(softs->product_name, aac_cards[card_index].desc,
2102             AAC_PRODUCT_LEN);

2104         /* Set up quirks */
2105         softs->flags = aac_cards[card_index].quirks;

2107         /* Force the busmaster enable bit on */
2108         pci_cmd = pci_config_get16(pci_config_handle, PCI_CONF_COMM);
2109         if ((pci_cmd & PCI_COMM_ME) == 0) {
2110                 pci_cmd |= PCI_COMM_ME;
2111                 pci_config_put16(pci_config_handle, PCI_CONF_COMM, pci_cmd);
2112                 pci_cmd = pci_config_get16(pci_config_handle, PCI_CONF_COMM);
2113                 if ((pci_cmd & PCI_COMM_ME) == 0) {
2114                         cmn_err(CE_CONT, "?Cannot enable busmaster bit");
2115                         goto error;
2116                 }
2117         }

2119         /* Set memory base to map */
2120         softs->pci_mem_base_paddr[0] = 0xfffffff0UL & \
2200         softs->pci_mem_base_paddr = 0xfffffff0UL & \
2121             pci_config_get32(pci_config_handle, PCI_CONF_BASE0);
2122         if (softs->hwif == AAC_HWIF_SRC) {
2123                 softs->pci_mem_base_paddr[1] = 0xfffffff0UL & \
2124                     pci_config_get32(pci_config_handle, PCI_CONF_BASE2);
2125         } else {
2126                 softs->pci_mem_base_paddr[1] = softs->pci_mem_base_paddr[0];
2127         }

2129         pci_config_teardown(&pci_config_handle);

2131         return (AACOK); /* card type detected */
2132 error:
2133         pci_config_teardown(&pci_config_handle);
2134         return (AACERR); /* no matched card found */
2135 }

2137 /*
2212  * Do the usual interrupt handler setup stuff.
2213  */
```

```
2214 static int
2215 aac_register_intrs(struct aac_softstate *softs)
2216 {
2217         dev_info_t *dip;
2218         int intr_types;

2220         ASSERT(softs->devinfo_p);
2221         dip = softs->devinfo_p;

2223         /* Get the type of device intrrupts */
2224         if (ddi_intr_get_supported_types(dip, &intr_types) != DDI_SUCCESS) {
2225                 AACDB_PRINT(softs, CE_WARN,
2226                     "ddi_intr_get_supported_types() failed");
2227                 return (AACERR);
2228         }
2229         AACDB_PRINT(softs, CE_NOTE,
2230             "ddi_intr_get_supported_types() ret: 0x%x", intr_types);

2232         /* Query interrupt, and alloc/init all needed struct */
2233         if (intr_types & DDI_INTR_TYPE_MSI) {
2234                 if (aac_query_intrs(softs, DDI_INTR_TYPE_MSI)
2235                     != DDI_SUCCESS) {
2236                         AACDB_PRINT(softs, CE_WARN,
2237                             "MSI interrupt query failed");
2238                         return (AACERR);
2239                 }
2240                 softs->intr_type = DDI_INTR_TYPE_MSI;
2241         } else if (intr_types & DDI_INTR_TYPE_FIXED) {
2242                 if (aac_query_intrs(softs, DDI_INTR_TYPE_FIXED)
2243                     != DDI_SUCCESS) {
2244                         AACDB_PRINT(softs, CE_WARN,
2245                             "FIXED interrupt query failed");
2246                         return (AACERR);
2247                 }
2248                 softs->intr_type = DDI_INTR_TYPE_FIXED;
2249         } else {
2250                 AACDB_PRINT(softs, CE_WARN,
2251                     "Device cannot suppport both FIXED and MSI interrupts");
2252                 return (AACERR);
2253         }

2255         /* Connect interrupt handlers */
2256         if (aac_add_intrs(softs) != DDI_SUCCESS) {
2257                 AACDB_PRINT(softs, CE_WARN,
2258                     "Interrupt registration failed, intr type: %s",
2259                     softs->intr_type == DDI_INTR_TYPE_MSI ? "MSI" : "FIXED");
2260                 return (AACERR);
2261         }
2262         (void) aac_enable_intrs(softs);

2264         if (ddi_add_softintr(dip, DDI_SOFTINT_LOW, &softs->softint_id,
2265             NULL, NULL, aac_softintr, (caddr_t)softs) != DDI_SUCCESS) {
2266                 AACDB_PRINT(softs, CE_WARN,
2267                     "Can not setup soft interrupt handler!");
2268                 aac_remove_intrs(softs);
2269                 return (AACERR);
2270         }

2272         return (AACOK);
2273 }

2275 static void
2276 aac_unregister_intrs(struct aac_softstate *softs)
2277 {
2278         aac_remove_intrs(softs);
2279         ddi_remove_softintr(softs->softint_id);
```



```
2280 }

2282 /*
2138  * Check the firmware to determine the features to support and the FIB
2139  * parameters to use.
2140  */
2141 static int
2142 aac_check_firmware(struct aac_softstate *softs)
2143 {
2144         uint32_t options;
2145         uint32_t atu_size;
2146         ddi_acc_handle_t pci_handle;
2147         char *pci_mbr;
2148         uint32_t max_fibs, max_aif;
2292         uint8_t *data;
2293         uint32_t max_fibs;
2149         uint32_t max_fib_size;
2150         uint32_t sg_tablesize;
2151         uint32_t max_sectors;
2152         uint32_t status;

2154         /* Get supported options */
2155         softs->sync_slot_busy = 1;
2156         if ((aac_sync_mbcommand(softs, AAC_MONKER_GETINFO, 0, 0, 0, 0,
2157             &status, NULL)) != AACOK) {
2301             &status)) != AACOK) {
2158                 if (status != SRB_STATUS_INVALID_REQUEST) {
2159                         cmn_err(CE_CONT,
2160                             "?Fatal error: request adapter info error");
2161                         return (AACERR);
2162                 }
2163                 options = 0;
2164                 atu_size = 0;
2165         } else {
2166                 options = AAC_MAILBOX_GET(softs, 1);
2167                 atu_size = AAC_MAILBOX_GET(softs, 2);
2168         }

2170         if (softs->state & AAC_STATE_RESET) {
2171                 if ((softs->support_opt == options) &&
2172                     (softs->atu_size == atu_size))
2173                         return (AACOK);

2175                 cmn_err(CE_WARN,
2176                     "?Fatal error: firmware changed, system needs reboot");
2177                 return (AACERR);
2178         }

2180         /*
2181          * The following critical settings are initialized only once during
2182          * driver attachment.
2183          */
2184         softs->support_opt = options;
2185         softs->atu_size = atu_size;

2187         /* Process supported options */
2188         if ((options & AAC_SUPPORTED_4GB_WINDOW) != 0 &&
2189             (softs->flags & AAC_FLAGS_NO4GB) == 0) {
2190                 AACDB_PRINT(softs, CE_NOTE, "!Enable FIB map 4GB window");
2191                 softs->flags |= AAC_FLAGS_4GB_WINDOW;
2192         } else {
2193                 /*
2194                  * Quirk AAC_FLAGS_NO4GB is for FIB address and thus comm space
2195                  * only. IO is handled by the DMA engine which does not suffer
2196                  * from the ATU window programming workarounds necessary for
2197                  * CPU copy operations.
```

```
2198                 */
2199                softs->addr_dma_attr.dma_attr_addr_lo = 0x2000ull;
2200                softs->addr_dma_attr.dma_attr_addr_hi = 0x7fffffffull;
2201        }

2203        if ((options & AAC_SUPPORTED_SGMAP_HOST64) != 0) {
2204                AACDB_PRINT(softs, CE_NOTE, "!Enable SG map 64-bit address");
2205                softs->buf_dma_attr.dma_attr_addr_hi = 0xffffffffffffffffull;
2206                softs->buf_dma_attr.dma_attr_seg = 0xffffffffffffffffull;
2207                softs->flags |= AAC_FLAGS_SG_64BIT;
2208        }

2210        if (options & AAC_SUPPORTED_64BIT_ARRAYSIZE) {
2211                softs->flags |= AAC_FLAGS_ARRAY_64BIT;
2212                AACDB_PRINT(softs, CE_NOTE, "!Enable 64-bit array size");
2213        }

2215        if (options & AAC_SUPPORTED_NONDASD) {
2360                if ((ddi_prop_lookup_string(DDI_DEV_T_ANY, softs->devinfo_p, 0,
2361                    "nondasd-enable", (char **)&data) == DDI_SUCCESS)) {
2362                        if (strcmp((char *)data, "yes") == 0) {
2363                                AACDB_PRINT(softs, CE_NOTE,
2364                                    "!Enable Non-DASD access");
2216                softs->flags |= AAC_FLAGS_NONDASD;
2217                AACDB_PRINT(softs, CE_NOTE, "!Enable Non-DASD access");
2218        }

2220        if ((options & AAC_SUPPORTED_NEW_COMM_TYPE3) ||
2221                (options & AAC_SUPPORTED_NEW_COMM_TYPE4)) {
2222                softs->flags |= AAC_FLAGS_NEW_COMM | AAC_FLAGS_NEW_COMM_TYPE34;
2223        } else if (options & AAC_SUPPORTED_NEW_COMM_TYPE2) {
2224                softs->flags |= AAC_FLAGS_NEW_COMM | AAC_FLAGS_NEW_COMM_TYPE2;
2225                AACDB_PRINT(softs, CE_NOTE,     "!Enable New Comm. Type2 interfa
2226        } else if (options & AAC_SUPPORTED_NEW_COMM_TYPE1) {
2227                softs->flags |= AAC_FLAGS_NEW_COMM | AAC_FLAGS_NEW_COMM_TYPE1;
2228                AACDB_PRINT(softs, CE_NOTE,     "!Enable New Comm. Type1 interfa
2229        } else if (options & AAC_SUPPORTED_NEW_COMM) {
2230                softs->flags |= AAC_FLAGS_NEW_COMM;
2231                AACDB_PRINT(softs, CE_NOTE,     "!Enable New Comm. interface");
2367                        ddi_prop_free(data);
2232        }

2234        if (softs->sync_mode &&
2235                ((softs->flags & AAC_FLAGS_NEW_COMM_TYPE1) ||
2236                 (softs->flags & AAC_FLAGS_NEW_COMM_TYPE2) ||
2237                 (softs->flags & AAC_FLAGS_NEW_COMM_TYPE34))) {
2238                cmn_err(CE_NOTE, "aac: Sync. mode enforced by driver parameter.
2239                softs->flags |= AAC_FLAGS_SYNC_MODE;
2240        } else if (softs->flags & AAC_FLAGS_NEW_COMM_TYPE34) {
2241                cmn_err(CE_NOTE, "aac: Async. mode not supported by current driv
2242                cmn_err(CE_NOTE, "aac: Please update driver to get full performa
2243                softs->flags |= AAC_FLAGS_SYNC_MODE;
2244        }

2246        /* Read preferred settings */
2247        max_fib_size = 0;
2248        softs->sync_slot_busy = 1;
2249        if ((aac_sync_mbcommand(softs, AAC_MONKER_GETCOMMPREF,
2250            0, 0, 0, 0, NULL, NULL)) == AACOK) {
2374            0, 0, 0, 0, NULL)) == AACOK) {
2251                options = AAC_MAILBOX_GET(softs, 1);
2252                max_fib_size = (options & 0xffff);
2253                max_sectors = (options >> 16) << 1;
2254                options = AAC_MAILBOX_GET(softs, 2);
2255                sg_tablesize = (options >> 16);
2256                options = AAC_MAILBOX_GET(softs, 3);
```



```
2257                max_fibs = (options & 0xffff);
2258                options = AAC_MAILBOX_GET(softs, 4);
2259                max_aif = (options & 0xffff);
2260        }

2262        /* Enable new comm. and rawio at the same time */
2263        if ((softs->flags & AAC_FLAGS_NEW_COMM) &&
2385            ((softs->support_opt & AAC_SUPPORTED_NEW_COMM) &&
2264             (max_fib_size != 0)) {
2387                /* read out and save PCI MBR */
2265        if ((atu_size > softs->map_size) &&
2266                        (ddi_regs_map_setup(softs->devinfo_p, 1,
2267                                (caddr_t *)&pci_mbr, 0, atu_size, &aac_acc_attr,
2390                            (caddr_t *)&data, 0, atu_size, &softs->reg_attr,
2268                                &pci_handle) == DDI_SUCCESS)) {
2269                        ddi_regs_map_free(&softs->pci_mem_handle[0]);
2270                        softs->pci_mem_handle[0] = pci_handle;
2271                        softs->pci_mem_base_vaddr[0] = pci_mbr;
2392                        ddi_regs_map_free(&softs->pci_mem_handle);
2393                        softs->pci_mem_handle = pci_handle;
2394                        softs->pci_mem_base_vaddr = data;
2272                        softs->map_size = atu_size;
2273                        if (softs->hwif != AAC_HWIF_SRC) {
2274                                softs->pci_mem_handle[1] =
2275                                        softs->pci_mem_handle[0];
2276                                softs->pci_mem_base_vaddr[1] =
2277                                        softs->pci_mem_base_vaddr[0];
2278                        }
2397                if (atu_size == softs->map_size) {
2398                        softs->flags |= AAC_FLAGS_NEW_COMM;
2399                        AACDB_PRINT(softs, CE_NOTE,
2400                            "!Enable New Comm. interface");
2279                }
2280        }

2282        /* Set FIB parameters */
2283        if (softs->flags & AAC_FLAGS_NEW_COMM) {
2284                softs->aac_max_fibs = max_fibs;
2285                softs->aac_max_fib_size = max_fib_size;
2286                softs->aac_max_sectors = max_sectors;
2287                softs->aac_sg_tablesize = sg_tablesize;
2288                softs->aac_max_aif = max_aif;

2290                softs->flags |= AAC_FLAGS_RAW_IO;
2291                AACDB_PRINT(softs, CE_NOTE, "!Enable RawIO");
2292        } else {
2293                softs->aac_max_fibs =
2294                    (softs->flags & AAC_FLAGS_256FIBS) ? 256 : 512;
2295                softs->aac_max_fib_size = AAC_FIB_SIZE;
2296                softs->aac_max_sectors = 128;   /* 64K */
2297                if (softs->flags & AAC_FLAGS_17SG)
2298                        softs->aac_sg_tablesize = 17;
2299                else if (softs->flags & AAC_FLAGS_34SG)
2300                        softs->aac_sg_tablesize = 34;
2301                else if (softs->flags & AAC_FLAGS_SG_64BIT)
2302                        softs->aac_sg_tablesize = (AAC_FIB_DATASIZE -
2303                            sizeof (struct aac_blockwrite64) +
2304                            sizeof (struct aac_sg_entry64)) /
2305                            sizeof (struct aac_sg_entry64);
2306                else
2307                        softs->aac_sg_tablesize = (AAC_FIB_DATASIZE -
2308                            sizeof (struct aac_blockwrite) +
2309                            sizeof (struct aac_sg_entry)) /
2310                            sizeof (struct aac_sg_entry);
2311        }
```

```
2313         if ((softs->flags & AAC_FLAGS_RAW_IO) &&
2314             (softs->flags & AAC_FLAGS_ARRAY_64BIT)) {
2315                 softs->flags |= AAC_FLAGS_LBA_64BIT;
2316                 AACDB_PRINT(softs, CE_NOTE, "!Enable 64-bit array");
2317         }
2318         softs->buf_dma_attr.dma_attr_sgllen = softs->aac_sg_tablesize;
2319         softs->buf_dma_attr.dma_attr_maxxfer = softs->aac_max_sectors << 9;
2320         /*
2321          * 64K maximum segment size in scatter gather list is controlled by
2322          * the NEW_COMM bit in the adapter information. If not set, the card
2323          * can only accept a maximum of 64K. It is not recommended to permit
2324          * more than 128KB of total transfer size to the adapters because
2325          * performance is negatively impacted.
2326          *
2327          * For new comm, segment size equals max xfer size. For old comm,
2328          * we use 64K for both.
2329          */
2330         softs->buf_dma_attr.dma_attr_count_max =
2331             softs->buf_dma_attr.dma_attr_maxxfer - 1;

2333         /* Setup FIB operations */
2334         if (softs->flags & AAC_FLAGS_NEW_COMM_TYPE2)
2335                 softs->aac_cmd_fib = aac_cmd_fib_rawio2;
2336         else if (softs->flags & AAC_FLAGS_RAW_IO)
2455         if (softs->flags & AAC_FLAGS_RAW_IO)
2337                 softs->aac_cmd_fib = aac_cmd_fib_rawio;
2338         else if (softs->flags & AAC_FLAGS_SG_64BIT)
2339                 softs->aac_cmd_fib = aac_cmd_fib_brw64;
2340         else
2341                 softs->aac_cmd_fib = aac_cmd_fib_brw;
2342         softs->aac_cmd_fib_scsi = (softs->flags & AAC_FLAGS_SG_64BIT) ? \
2343             aac_cmd_fib_scsi64 : aac_cmd_fib_scsi32;

2345         /* 64-bit LBA needs descriptor format sense data */
2346         softs->slen = sizeof (struct scsi_arq_status);
2347         if ((softs->flags & AAC_FLAGS_LBA_64BIT) &&
2348             softs->slen < AAC_ARQ64_LENGTH)
2349                 softs->slen = AAC_ARQ64_LENGTH;

2351         AACDB_PRINT(softs, CE_NOTE,
2352             "!max_fibs %d max_fibsize 0x%x max_sectors %d max_sg %d",
2353             softs->aac_max_fibs, softs->aac_max_fib_size,
2354             softs->aac_max_sectors, softs->aac_sg_tablesize);

2356         return (AACOK);
2357 }

2359 static void
2360 aac_fsa_rev(struct aac_softstate *softs, struct FsaRev *fsarev0,
2361     struct FsaRev *fsarev1)
2362 {
2363         ddi_acc_handle_t acc = softs->comm_space_acc_handle;
2482         ddi_acc_handle_t acc = softs->sync_ac.slotp->fib_acc_handle;

2365         AAC_GET_FIELD8(acc, fsarev1, fsarev0, external.comp.dash);
2366         AAC_GET_FIELD8(acc, fsarev1, fsarev0, external.comp.type);
2367         AAC_GET_FIELD8(acc, fsarev1, fsarev0, external.comp.minor);
2368         AAC_GET_FIELD8(acc, fsarev1, fsarev0, external.comp.major);
2369         AAC_GET_FIELD32(acc, fsarev1, fsarev0, buildNumber);
2370 }

2372 /*
2373  * The following function comes from Adaptec:
2374  *
2375  * Query adapter information and supplement adapter information
2376  */
```



```
2377 static int
2378 aac_get_adapter_info(struct aac_softstate *softs,
2379     struct aac_adapter_info *ainfr, struct aac_supplement_adapter_info *sinfr)
2380 {
2381         ddi_acc_handle_t acc = softs->sync_slot->fib_acc_handle;
2382         struct aac_fib *fibp = softs->sync_slot->fibp;
2500         struct aac_cmd *acp = &softs->sync_ac;
2501         ddi_acc_handle_t acc;
2502         struct aac_fib *fibp;
2383         struct aac_adapter_info *ainfp;
2384         struct aac_supplement_adapter_info *sinfp;
2505         int rval;

2507         (void) aac_sync_fib_slot_bind(softs, acp);
2508         acc = acp->slotp->fib_acc_handle;
2509         fibp = acp->slotp->fibp;

2386         ddi_put8(acc, &fibp->data[0], 0);
2387         if (aac_sync_fib(softs, RequestAdapterInfo,
2388             AAC_FIB_SIZEOF(struct aac_adapter_info)) != AACOK) {
2389                 AACDB_PRINT(softs, CE_WARN, "RequestAdapterInfo failed");
2390                 return (AACERR);
2515                 rval = AACERR;
2516                 goto finish;
2391         }
2392         ainfp = (struct aac_adapter_info *)fibp->data;
2393         if (ainfr) {
2394                 AAC_GET_FIELD32(acc, ainfr, ainfp, SupportedOptions);
2395                 AAC_GET_FIELD32(acc, ainfr, ainfp, PlatformBase);
2396                 AAC_GET_FIELD32(acc, ainfr, ainfp, CpuArchitecture);
2397                 AAC_GET_FIELD32(acc, ainfr, ainfp, CpuVariant);
2398                 AAC_GET_FIELD32(acc, ainfr, ainfp, ClockSpeed);
2399                 AAC_GET_FIELD32(acc, ainfr, ainfp, ExecutionMem);
2400                 AAC_GET_FIELD32(acc, ainfr, ainfp, BufferMem);
2401                 AAC_GET_FIELD32(acc, ainfr, ainfp, TotalMem);
2402                 aac_fsa_rev(softs, &ainfp->KernelRevision,
2403                     &ainfr->KernelRevision);
2404                 aac_fsa_rev(softs, &ainfp->MonitorRevision,
2405                     &ainfr->MonitorRevision);
2406                 aac_fsa_rev(softs, &ainfp->HardwareRevision,
2407                     &ainfr->HardwareRevision);
2408                 aac_fsa_rev(softs, &ainfp->BIOSRevision,
2409                     &ainfr->BIOSRevision);
2410                 AAC_GET_FIELD32(acc, ainfr, ainfp, ClusteringEnabled);
2411                 AAC_GET_FIELD32(acc, ainfr, ainfp, ClusterChannelMask);
2412                 AAC_GET_FIELD64(acc, ainfr, ainfp, SerialNumber);
2413                 AAC_GET_FIELD32(acc, ainfr, ainfp, batteryPlatform);
2414                 AAC_GET_FIELD32(acc, ainfr, ainfp, SupportedOptions);
2415                 AAC_GET_FIELD32(acc, ainfr, ainfp, OemVariant);
2416         }
2417         if (sinfr) {
2418                 if (!(softs->support_opt &
2419                     AAC_SUPPORTED_SUPPLEMENT_ADAPTER_INFO)) {
2420                         AACDB_PRINT(softs, CE_WARN,
2421                             "SupplementAdapterInfo not supported");
2422                         return (AACERR);
2548                         rval = AACERR;
2549                         goto finish;
2423                 }
2424                 ddi_put8(acc, &fibp->data[0], 0);
2425                 if (aac_sync_fib(softs, RequestSupplementAdapterInfo,
2426                     AAC_FIB_SIZEOF(struct aac_supplement_adapter_info)) != AACOK
2553                     AAC_FIB_SIZEOF(struct aac_supplement_adapter_info))
2554                     != AACOK) {
2427                         AACDB_PRINT(softs, CE_WARN,
2428                             "RequestSupplementAdapterInfo failed");
```

```
2429                        return (AACERR);
2557                        rval = AACERR;
2558                        goto finish;
2430                }
2431                sinfp = (struct aac_supplement_adapter_info *)fibp->data;
2432                AAC_REP_GET_FIELD8(acc, sinfr, sinfp, AdapterTypeText[0], 17+1);
2433                AAC_REP_GET_FIELD8(acc, sinfr, sinfp, Pad[0], 2);
2434                AAC_GET_FIELD32(acc, sinfr, sinfp, FlashMemoryByteSize);
2435                AAC_GET_FIELD32(acc, sinfr, sinfp, FlashImageId);
2436                AAC_GET_FIELD32(acc, sinfr, sinfp, MaxNumberPorts);
2437                AAC_GET_FIELD32(acc, sinfr, sinfp, Version);
2438                AAC_GET_FIELD32(acc, sinfr, sinfp, FeatureBits);
2439                AAC_GET_FIELD8(acc, sinfr, sinfp, SlotNumber);
2440                AAC_REP_GET_FIELD8(acc, sinfr, sinfp, ReservedPad0[0], 3);
2441                AAC_REP_GET_FIELD8(acc, sinfr, sinfp, BuildDate[0], 12);
2442                AAC_GET_FIELD32(acc, sinfr, sinfp, CurrentNumberPorts);
2443                AAC_REP_GET_FIELD8(acc, sinfr, sinfp, VpdInfo,
2444                    sizeof (struct vpd_info));
2445                aac_fsa_rev(softs, &sinfp->FlashFirmwareRevision,
2446                    &sinfr->FlashFirmwareRevision);
2447                AAC_GET_FIELD32(acc, sinfr, sinfp, RaidTypeMorphOptions);
2448                aac_fsa_rev(softs, &sinfp->FlashFirmwareBootRevision,
2449                    &sinfr->FlashFirmwareBootRevision);
2450                AAC_REP_GET_FIELD8(acc, sinfr, sinfp, MfgPcbaSerialNo,
2451                    MFG_PCBA_SERIAL_NUMBER_WIDTH);
2452                AAC_REP_GET_FIELD8(acc, sinfr, sinfp, MfgWWNName[0],
2453                    MFG_WWN_WIDTH);
2454                AAC_GET_FIELD32(acc, sinfr, sinfp, SupportedOptions2);
2455                AAC_GET_FIELD32(acc, sinfr, sinfp, ExpansionFlag);
2456                if (sinfr->ExpansionFlag == 1) {
2457                        AAC_GET_FIELD32(acc, sinfr, sinfp, FeatureBits3);
2458                        AAC_GET_FIELD32(acc, sinfr, sinfp, SupportedPerformanceM
2459                        AAC_REP_GET_FIELD32(acc, sinfr, sinfp, ReservedGrowth[0]
2587                        AAC_GET_FIELD32(acc, sinfr, sinfp,
2588                            SupportedPerformanceMode);
2589                        AAC_REP_GET_FIELD32(acc, sinfr, sinfp,
2590                            ReservedGrowth[0], 80);
2460                }
2461        }
2462        return (AACOK);
2593        rval = AACOK;
2594 finish:
2595        aac_sync_fib_slot_release(softs, acp);
2596        return (rval);
2463 }

2465 static int
2466 aac_get_bus_info(struct aac_softstate *softs, uint32_t *bus_max,
2467     uint32_t *tgt_max)
2468 {
2469        ddi_acc_handle_t acc = softs->sync_slot->fib_acc_handle;
2470        struct aac_fib *fibp = softs->sync_slot->fibp;
2603        struct aac_cmd *acp = &softs->sync_ac;
2604        ddi_acc_handle_t acc;
2605        struct aac_fib *fibp;
2471        struct aac_ctcfg *c_cmd;
2472        struct aac_ctcfg_resp *c_resp;
2473        uint32_t scsi_method_id;
2474        struct aac_bus_info *cmd;
2475        struct aac_bus_info_response *resp;
2476        int rval;

2613        (void) aac_sync_fib_slot_bind(softs, acp);
2614        acc = acp->slotp->fib_acc_handle;
2615        fibp = acp->slotp->fibp;
```



```
2478        /* Detect MethodId */
2479        c_cmd = (struct aac_ctcfg *)&fibp->data[0];
2480        ddi_put32(acc, &c_cmd->Command, VM_ContainerConfig);
2481        ddi_put32(acc, &c_cmd->cmd, CT_GET_SCSI_METHOD);
2482        ddi_put32(acc, &c_cmd->param, 0);
2483        rval = aac_sync_fib(softs, ContainerCommand,
2484            AAC_FIB_SIZEOF(struct aac_ctcfg));
2485        c_resp = (struct aac_ctcfg_resp *)&fibp->data[0];
2486        if (rval != AACOK || ddi_get32(acc, &c_resp->Status) != 0) {
2487                AACDB_PRINT(softs, CE_WARN,
2488                    "VM_ContainerConfig command fail");
2489                return (AACERR);
2628                rval = AACERR;
2629                goto finish;
2490        }
2491        scsi_method_id = ddi_get32(acc, &c_resp->param);

2493        /* Detect phys. bus count and max. target id first */
2494        cmd = (struct aac_bus_info *)&fibp->data[0];
2495        ddi_put32(acc, &cmd->Command, VM_Ioctl);
2496        ddi_put32(acc, &cmd->ObjType, FT_DRIVE); /* physical drive */
2497        ddi_put32(acc, &cmd->MethodId, scsi_method_id);
2498        ddi_put32(acc, &cmd->ObjectId, 0);
2499        ddi_put32(acc, &cmd->CtlCmd, GetBusInfo);
2500        /*
2501         * For VM_Ioctl, the firmware uses the Header.Size filled from the
2502         * driver as the size to be returned. Therefore the driver has to use
2503         * sizeof (struct aac_bus_info_response) because it is greater than
2504         * sizeof (struct aac_bus_info).
2505         */
2506        rval = aac_sync_fib(softs, ContainerCommand,
2507            AAC_FIB_SIZEOF(struct aac_bus_info_response));
2508        resp = (struct aac_bus_info_response *)cmd;

2510        /* Scan all coordinates with INQUIRY */
2511        if ((rval != AACOK) || (ddi_get32(acc, &resp->Status) != 0)) {
2512                AACDB_PRINT(softs, CE_WARN, "GetBusInfo command fail");
2513                return (AACERR);
2653                rval = AACERR;
2654                goto finish;
2514        }
2515        *bus_max = ddi_get32(acc, &resp->BusCount);
2516        *tgt_max = ddi_get32(acc, &resp->TargetsPerBus);

2659 finish:
2660        aac_sync_fib_slot_release(softs, acp);
2517        return (AACOK);
2518 }

2520 /*
2521  * The following function comes from Adaptec:
2522  *
2523  * Routine to be called during initialization of communications with
2524  * the adapter to handle possible adapter configuration issues. When
2525  * the adapter first boots up, it examines attached drives, etc, and
2526  * potentially comes up with a new or revised configuration (relative to
2527  * what's stored in it's NVRAM). Additionally it may discover problems
2528  * that make the current physical configuration unworkable (currently
2529  * applicable only to cluster configuration issues).
2530  *
2531  * If there are no configuration issues or the issues are considered
2532  * trival by the adapter, it will set it's configuration status to
2533  * "FSACT_CONTINUE" and execute the "commit confiuguration" action
2534  * automatically on it's own.
2535  *
2536  * However, if there are non-trivial issues, the adapter will set it's
```

```
 * internal configuration status to "FSACT_PAUSE" or "FASCT_ABORT"
 * and wait for some agent on the host to issue the "\ContainerCommand
 * \VM_ContainerConfig\CT_COMMIT_CONFIG" FIB command to cause the
 * adapter to commit the new/updated configuration and enable
 * un-inhibited operation.  The host agent should first issue the
 * "\ContainerCommand\VM_ContainerConfig\CT_GET_CONFIG_STATUS" FIB
 * command to obtain information about config issues detected by
 * the adapter.
 *
 * Normally the adapter's PC BIOS will execute on the host following
 * adapter poweron and reset and will be responsible for querring the
 * adapter with CT_GET_CONFIG_STATUS and issuing the CT_COMMIT_CONFIG
 * command if appropriate.
 *
 * However, with the introduction of IOP reset support, the adapter may
 * boot up without the benefit of the adapter's PC BIOS host agent.
 * This routine is intended to take care of these issues in situations
 * where BIOS doesn't execute following adapter poweron or reset.  The
 * CT_COMMIT_CONFIG command is a no-op if it's already been issued, so
 * there is no harm in doing this when it's already been done.
 */
static int
aac_handle_adapter_config_issues(struct aac_softstate *softs)
{
        ddi_acc_handle_t acc = softs->sync_slot->fib_acc_handle;
        struct aac_fib *fibp = softs->sync_slot->fibp;
        struct aac_cmd *acp = &softs->sync_ac;
        ddi_acc_handle_t acc;
        struct aac_fib *fibp;
        struct aac_Container *cmd;
        struct aac_Container_resp *resp;
        struct aac_cf_status_header *cfg_sts_hdr;
        uint32_t resp_status;
        uint32_t ct_status;
        uint32_t cfg_stat_action;
        int rval;

        (void) aac_sync_fib_slot_bind(softs, acp);
        acc = acp->slotp->fib_acc_handle;
        fibp = acp->slotp->fibp;

        /* Get adapter config status */
        cmd = (struct aac_Container *)&fibp->data[0];

        bzero(cmd, sizeof (*cmd) - CT_PACKET_SIZE);
        ddi_put32(acc, &cmd->Command, VM_ContainerConfig);
        ddi_put32(acc, &cmd->CTCommand.command, CT_GET_CONFIG_STATUS);
        ddi_put32(acc, &cmd->CTCommand.param[CNT_SIZE],
            sizeof (struct aac_cf_status_header));
        rval = aac_sync_fib(softs, ContainerCommand,
            AAC_FIB_SIZEOF(struct aac_Container));
        resp = (struct aac_Container_resp *)cmd;
        cfg_sts_hdr = (struct aac_cf_status_header *)resp->CTResponse.data;

        resp_status = ddi_get32(acc, &resp->Status);
        ct_status = ddi_get32(acc, &resp->CTResponse.param[0]);
        if ((rval == AACOK) && (resp_status == 0) && (ct_status == CT_OK)) {
                cfg_stat_action = ddi_get32(acc, &cfg_sts_hdr->action);

                /* Commit configuration if it's reasonable to do so. */
                if (cfg_stat_action <= CFACT_PAUSE) {
                        bzero(cmd, sizeof (*cmd) - CT_PACKET_SIZE);
                        ddi_put32(acc, &cmd->Command, VM_ContainerConfig);
                        ddi_put32(acc, &cmd->CTCommand.command,
                            CT_COMMIT_CONFIG);
                        rval = aac_sync_fib(softs, ContainerCommand,
```



```
                            AAC_FIB_SIZEOF(struct aac_Container));

                        resp_status = ddi_get32(acc, &resp->Status);
                        ct_status = ddi_get32(acc, &resp->CTResponse.param[0]);
                        if ((rval == AACOK) && (resp_status == 0) &&
                            (ct_status == CT_OK))
                                /* Successful completion */
                                rval = AACMPE_OK;
                        else
                                /* Auto-commit aborted due to error(s). */
                                rval = AACMPE_COMMIT_CONFIG;
                } else {
                        /*
                         * Auto-commit aborted due to adapter indicating
                         * configuration issue(s) too dangerous to auto-commit.
                         */
                        rval = AACMPE_CONFIG_STATUS;
                }
        } else {
                cmn_err(CE_WARN, "!Configuration issue, auto-commit aborted");
                rval = AACMPE_CONFIG_STATUS;
        }

        aac_sync_fib_slot_release(softs, acp);
        return (rval);
}

/*
 * Hardware initialization and resource allocation
 */
static int
aac_common_attach(struct aac_softstate *softs)
{
        uint32_t status;
        int i;
        char version_string[160];
        struct aac_supplement_adapter_info sinf;

        DBCALLED(softs, 1);

        /*
         * Do a little check here to make sure there aren't any outstanding
         * FIBs in the message queue. At this point there should not be and
         * if there are they are probably left over from another instance of
         * the driver like when the system crashes and the crash dump driver
         * gets loaded.
         */
        if (softs->hwif != AAC_HWIF_SRC && softs->hwif != AAC_HWIF_SRCV) {
                while (AAC_OUTB_GET(softs) != 0xffffffffful)
                        ;
        }

        /*
         * Wait the card to complete booting up before do anything that
         * attempts to communicate with it.
         */
        status = AAC_FWSTATUS_GET(softs);
        if (status == AAC_SELF_TEST_FAILED || status == AAC_KERNEL_PANIC)
                goto error;
        i = AAC_FWUP_TIMEOUT * 1000; /* set timeout */
        AAC_BUSYWAIT(AAC_FWSTATUS_GET(softs) & AAC_KERNEL_UP_AND_RUNNING, i);
        if (i == 0) {
                cmn_err(CE_CONT, "?Fatal error: controller not ready");
                aac_fm_ereport(softs, DDI_FM_DEVICE_NO_RESPONSE, DDI_SERVICE_LOS
                aac_fm_ereport(softs, DDI_FM_DEVICE_NO_RESPONSE);
                ddi_fm_service_impact(softs->devinfo_p, DDI_SERVICE_LOST);
```

```
2658                    goto error;
2659            }

2661            /* Read and set card supported options and settings */
2662            if (aac_check_firmware(softs) == AACERR) {
2663                    aac_fm_ereport(softs, DDI_FM_DEVICE_NO_RESPONSE, DDI_SERVICE_LOS
2812                    aac_fm_ereport(softs, DDI_FM_DEVICE_NO_RESPONSE);
2813                    ddi_fm_service_impact(softs->devinfo_p, DDI_SERVICE_LOST);
2664                    goto error;
2665            }

2667            /* Clear out all interrupts */
2668            AAC_STATUS_CLR(softs, ~0);
2817            /* Add interrupt handlers */
2818            if (aac_register_intrs(softs) == AACERR) {
2819                    cmn_err(CE_CONT,
2820                        "?Fatal error: interrupts register failed");
2821                    goto error;
2822            }

2670            /* Allocate communication space with the card */
2824            /* Setup communication space with the card */
2671            if (softs->comm_space_dma_handle == NULL) {
2672                    if (aac_alloc_comm_space(softs) != AACOK)
2673                            goto error;
2674            }
2829            if (aac_setup_comm_space(softs) != AACOK) {
2830                    cmn_err(CE_CONT, "?Setup communication space failed");
2831                    aac_fm_ereport(softs, DDI_FM_DEVICE_NO_RESPONSE);
2832                    ddi_fm_service_impact(softs->devinfo_p, DDI_SERVICE_LOST);
2833                    goto error;
2834            }

2836 #ifdef DEBUG
2837            if (aac_get_fw_debug_buffer(softs) != AACOK)
2838                    cmn_err(CE_CONT, "?firmware UART trace not supported");
2839 #endif

2676            /* Allocate slots */
2677            if ((softs->total_slots == 0) && (aac_create_slots(softs) != AACOK)) {
2678                    cmn_err(CE_CONT, "?Fatal error: slots allocate failed");
2679                    goto error;
2680            }
2681            AACDB_PRINT(softs, CE_NOTE, "%d slots allocated", softs->total_slots);

2683            /* Allocate FIBs */
2684            if (softs->total_fibs < softs->total_slots) {
2685                    aac_alloc_fibs(softs);
2686                    if (softs->total_fibs == 0)
2687                            goto error;
2688                    AACDB_PRINT(softs, CE_NOTE, "%d fibs allocated",
2689                        softs->total_fibs);
2690            }

2692            if (aac_get_adapter_info(softs, NULL, &sinf) != AACOK) {
2693                    cmn_err(CE_CONT, "?Query adapter information failed");
2694            } else {
2695                    softs->aac_feature_bits = sinf.FeatureBits;
2696                    softs->aac_support_opt2 = sinf.SupportedOptions2;
2857            AAC_STATUS_CLR(softs, ~0); /* Clear out all interrupts */
2858            AAC_ENABLE_INTR(softs); /* Enable the interrupts we can handle */

2860            if (aac_get_adapter_info(softs, NULL, &sinf) == AACOK) {
2861                    softs->feature_bits = sinf.FeatureBits;
2862                    softs->support_opt2 = sinf.SupportedOptions2;
```



```
2698                /* Get adapter names */
2699                if (CARD_IS_UNKNOWN(softs->card)) {
2700                        char *p, *p0, *p1;

2702                        /*
2703                         * Now find the controller name in supp_adapter_info->
2704                         * AdapterTypeText. Use the first word as the vendor
2705                         * and the other words as the product name.
2706                         */
2707                        AACDB_PRINT(softs, CE_NOTE, "sinf.AdapterTypeText = "
2708                            "\"%s\"", sinf.AdapterTypeText);
2709                        p = sinf.AdapterTypeText;
2710                        p0 = p1 = NULL;
2711                        /* Skip heading spaces */
2712                        while (*p && (*p == ' ' || *p == '\t'))
2713                                p++;
2714                        p0 = p;
2715                        while (*p && (*p != ' ' && *p != '\t'))
2716                                p++;
2717                        /* Remove middle spaces */
2718                        while (*p && (*p == ' ' || *p == '\t'))
2719                                *p++ = 0;
2720                        p1 = p;
2721                        /* Remove trailing spaces */
2722                        p = p1 + strlen(p1) - 1;
2723                        while (p > p1 && (*p == ' ' || *p == '\t'))
2724                                *p-- = 0;
2725                        if (*p0 && *p1) {
2726                                (void *)strncpy(softs->vendor_name, p0,
2727                                    AAC_VENDOR_LEN);
2728                                (void *)strncpy(softs->product_name, p1,
2729                                    AAC_PRODUCT_LEN);
2730                        } else {
2731                                cmn_err(CE_WARN,
2732                                    "?adapter name mis-formatted\n");
2733                                if (*p0)
2734                                        (void *)strncpy(softs->product_name,
2735                                            p0, AAC_PRODUCT_LEN);
2736                        }
2737                }
2904        } else {
2905                cmn_err(CE_CONT, "?Query adapter information failed");
2738        }

2740        if (!(softs->flags & AAC_FLAGS_SYNC_MODE)) {
2741                /* Setup communication space with the card */
2742                if (aac_setup_comm_space(softs) != AACOK) {
2743                        cmn_err(CE_CONT, "?Setup communication space failed");
2744                        aac_fm_ereport(softs, DDI_FM_DEVICE_NO_RESPONSE, DDI_SER
2745                        goto error;
2746                }
2747        }

2749 #ifdef AAC_DEBUG
2750        if (aac_get_fw_debug_buffer(softs) != AACOK)
2751                cmn_err(CE_CONT, "?firmware UART trace not supported");
2752 #endif

2754        sprintf(version_string,
2909        cmn_err(CE_NOTE,
2755            "!aac driver %d.%02d.%02d-%d, found card: " \
2756            "%s %s(pci0x%x.%x.%x.%x) at 0x%x",
2757            AAC_DRIVER_MAJOR_VERSION,
2758            AAC_DRIVER_MINOR_VERSION,
2759            AAC_DRIVER_BUGFIX_LEVEL,
2760            AAC_DRIVER_BUILD,
```

```
2761            softs->vendor_name, softs->product_name,
2762            softs->vendid, softs->devid, softs->subvendid, softs->subsysid,
2763            softs->pci_mem_base_paddr[0]);
2764        cmn_err(CE_NOTE, version_string);
2765        AACDB_PRINT(softs, CE_NOTE, version_string);
2918            softs->pci_mem_base_paddr);

2767        /* Perform acceptance of adapter-detected config changes if possible */
2768        if (aac_handle_adapter_config_issues(softs) != AACMPE_OK) {
2769                cmn_err(CE_CONT, "?Handle adapter config issues failed");
2770                aac_fm_ereport(softs, DDI_FM_DEVICE_NO_RESPONSE, DDI_SERVICE_LOS
2923                aac_fm_ereport(softs, DDI_FM_DEVICE_NO_RESPONSE);
2924                ddi_fm_service_impact(softs->devinfo_p, DDI_SERVICE_LOST);
2771                goto error;
2772        }

2774        /* Setup containers (logical devices) */
2775        if (aac_probe_containers(softs) != AACOK) {
2776                cmn_err(CE_CONT, "?Fatal error: get container info error");
2777                goto error;
2778        }

2934        /* Check for JBOD support. Default disable */
2935        char *data;
2936        if (softs->feature_bits & AAC_FEATURE_SUPPORTED_JBOD) {
2937                if ((ddi_prop_lookup_string(DDI_DEV_T_ANY, softs->devinfo_p,
2938                    0, "jbod-enable", &data) == DDI_SUCCESS)) {
2939                        if (strcmp(data, "yes") == 0) {
2940                                AACDB_PRINT(softs, CE_NOTE,
2941                                    "Enable JBOD access");
2942                                softs->flags |= AAC_FLAGS_JBOD;
2943                        }
2944                        ddi_prop_free(data);
2945                }
2946        }

2780        /* Setup phys. devices */
2781        if (softs->flags & AAC_FLAGS_NONDASD) {
2949        if (softs->flags & (AAC_FLAGS_NONDASD | AAC_FLAGS_JBOD)) {
2782                uint32_t bus_max, tgt_max;
2783                uint32_t bus, tgt;
2784                int index;

2786                if (aac_get_bus_info(softs, &bus_max, &tgt_max) != AACOK) {
2787                        cmn_err(CE_CONT, "?Fatal error: get bus info error");
2788                        goto error;
2789                }
2790                AACDB_PRINT(softs, CE_NOTE, "bus_max=%d, tgt_max=%d",
2791                    bus_max, tgt_max);
2792                if (bus_max != softs->bus_max || tgt_max != softs->tgt_max) {
2793                        if (softs->state & AAC_STATE_RESET) {
2794                                cmn_err(CE_WARN,
2795                                    "?Fatal error: bus map changed");
2796                                goto error;
2797                        }
2798                        softs->bus_max = bus_max;
2799                        softs->tgt_max = tgt_max;
2800                        if (softs->nondasds) {
2801                                kmem_free(softs->nondasds, AAC_MAX_PD(softs) * \
2802                                    sizeof (struct aac_nondasd));
2803                        }
2804                        softs->nondasds = kmem_zalloc(AAC_MAX_PD(softs) * \
2805                            sizeof (struct aac_nondasd), KM_SLEEP);

2807                        index = 0;
2808                        for (bus = 0; bus < softs->bus_max; bus++) {
```



```
2809                            for (tgt = 0; tgt < softs->tgt_max; tgt++) {
2810                                    struct aac_nondasd *dvp =
2811                                        &softs->nondasds[index++];
2812                                    dvp->dev.type = AAC_DEV_PD;
2813                                    dvp->bus = bus;
2814                                    dvp->tid = tgt;
2815                            }
2816                    }
2817            }
2818    }

2820    /* Check dma & acc handles allocated in attach */
2821    if (aac_check_dma_handle(softs->comm_space_dma_handle) != DDI_SUCCESS) {
2822            aac_fm_service_impact(softs->devinfo_p, DDI_SERVICE_LOST);
2990            ddi_fm_service_impact(softs->devinfo_p, DDI_SERVICE_LOST);
2823            goto error;
2824    }

2826    if (aac_check_acc_handle(softs->pci_mem_handle[0]) != DDI_SUCCESS) {
2827            aac_fm_service_impact(softs->devinfo_p, DDI_SERVICE_LOST);
2994    if (aac_check_acc_handle(softs->pci_mem_handle) != DDI_SUCCESS) {
2995            ddi_fm_service_impact(softs->devinfo_p, DDI_SERVICE_LOST);
2828            goto error;
2829    }

2831    for (i = 0; i < softs->total_slots; i++) {
2832            if (aac_check_dma_handle(softs->io_slot[i].fib_dma_handle) !=
2833                DDI_SUCCESS) {
2834                    aac_fm_service_impact(softs->devinfo_p,
3002                    ddi_fm_service_impact(softs->devinfo_p,
2835                        DDI_SERVICE_LOST);
2836                    goto error;
2837            }
2838    }

2840    return (AACOK);
2841 error:
2842    if (softs->state & AAC_STATE_RESET)
2843            return (AACERR);
2844    if (softs->nondasds) {
2845            kmem_free(softs->nondasds, AAC_MAX_PD(softs) * \
2846                sizeof (struct aac_nondasd));
2847            softs->nondasds = NULL;
2848    }
2849    if (softs->total_fibs > 0)
2850            aac_destroy_fibs(softs);
2851    if (softs->total_slots > 0)
2852            aac_destroy_slots(softs);
2853    if (softs->comm_space_dma_handle)
2854            aac_free_comm_space(softs);
2855    return (AACERR);
2856 }

2858 /*
2859  * Hardware shutdown and resource release
2860  */
2861 static void
2862 aac_common_detach(struct aac_softstate *softs)
2863 {
2864    DBCALLED(softs, 1);

3034    aac_unregister_intrs(softs);

3036    mutex_enter(&softs->io_lock);
2866    (void) aac_shutdown(softs);
```

```
2868         if (softs->nondasds) {
2869                 kmem_free(softs->nondasds, AAC_MAX_PD(softs) * \
2870                     sizeof (struct aac_nondasd));
2871                 softs->nondasds = NULL;
2872         }
2873         aac_destroy_fibs(softs);
2874         aac_destroy_slots(softs);
2875         aac_free_comm_space(softs);
3047         mutex_exit(&softs->io_lock);
2876 }

2878 /*
2879  * Send a synchronous command to the controller and wait for a result.
2880  * Indicate if the controller completed the command with an error status.
2881  */
2882 int
2883 aac_sync_mbcommand(struct aac_softstate *softs, uint32_t cmd,
2884     uint32_t arg0, uint32_t arg1, uint32_t arg2, uint32_t arg3,
2885     uint32_t *statusp, uint32_t *r1)
3057     uint32_t *statusp)
2886 {
2887         int timeout;
2888         uint32_t status;

3062         if (statusp != NULL)
3063                 *statusp = SRB_STATUS_SUCCESS;

2890         /* Fill in mailbox */
2891         AAC_MAILBOX_SET(softs, cmd, arg0, arg1, arg2, arg3);

2893         /* Ensure the sync command doorbell flag is cleared */
2894         AAC_STATUS_CLR(softs, AAC_DB_SYNC_COMMAND);

2896         /* Then set it to signal the adapter */
2897         AAC_NOTIFY(softs, AAC_DB_SYNC_COMMAND);

2899         if ((cmd != AAC_MONKER_SYNCFIB) || (statusp == NULL) || (*statusp != 0))
2900                 if (statusp != NULL)
2901                         *statusp = SRB_STATUS_SUCCESS;
2902                 /* Spin waiting for the command to complete */
2903                 timeout = AAC_IMMEDIATE_TIMEOUT * 1000;
2904                 AAC_BUSYWAIT(AAC_STATUS_GET(softs) & AAC_DB_SYNC_COMMAND, timeou
2905                 if (!timeout) {
2906                         AACDB_PRINT(softs, CE_WARN,
2907                                 "Sync command %d timed out after %d seconds (0x%
2908                                 cmd, AAC_IMMEDIATE_TIMEOUT, AAC_FWSTATUS_GET(sof
2909                         softs->sync_slot_busy = 0;
3079                     "Sync command timed out after %d seconds (0x%x)!",
3080                     AAC_IMMEDIATE_TIMEOUT, AAC_FWSTATUS_GET(softs));
2910                         return (AACERR);
2911                 }

2913                 /* Clear the completion flag */
2914                 AAC_STATUS_CLR(softs, AAC_DB_SYNC_COMMAND);

2916                 /* Get the command status */
2917                 status = AAC_MAILBOX_GET(softs, 0);
2918                 if (statusp != NULL)
2919                         *statusp = status;
2920                 if (status != SRB_STATUS_SUCCESS) {
2921                         AACDB_PRINT(softs, CE_WARN,
2922                                 "Sync command %d failed: status = 0x%x", cmd, st
2923                         softs->sync_slot_busy = 0;
3093                     "Sync command fail: status = 0x%x", status);
2924                         return (AACERR);
2925                 }
```



```
2926                 if (r1 != NULL)
2927                         *r1 = AAC_MAILBOX_GET(softs, 1);
2928                 softs->sync_slot_busy = 0;
2929         }

2931         return (AACOK);
2932 }

2934 /*
2935  * Send a synchronous FIB to the adapter and wait for its completion
2936  */
2937 static int
2938 aac_sync_fib(struct aac_softstate *softs, uint16_t cmd, uint16_t fibsize)
2939 {
2940         struct aac_slot *slotp = softs->sync_slot;
2941         ddi_dma_handle_t dma = slotp->fib_dma_handle;
2942         uint32_t status = 1;
2943         int rval;
3106         struct aac_cmd *acp = &softs->sync_ac;

2945         /* Sync fib only supports 512 bytes */
2946         if (fibsize > AAC_FIB_SIZE)
2947                 return (AACERR);
3108         acp->flags = AAC_CMD_SYNC | AAC_CMD_IN_SYNC_SLOT;
3109         if (softs->state & AAC_STATE_INTR)
3110                 acp->flags |= AAC_CMD_NO_CB;
3111         else
3112                 acp->flags |= AAC_CMD_NO_INTR;

3114         acp->ac_comp = aac_sync_complete;
3115         acp->timeout = AAC_SYNC_TIMEOUT;
3116         acp->fib_size = fibsize;

2949         /*
2950          * Setup sync fib
2951          * Need not reinitialize FIB header if it's already been filled
2952          * by others like aac_cmd_fib_scsi as aac_cmd.
3119          * Only need to setup sync fib header, caller should have init
3120          * fib data
2953          */
2954         if (slotp->acp == NULL)
2955                 aac_cmd_fib_header(softs, slotp, cmd, fibsize);
3122         aac_cmd_fib_header(softs, acp, cmd);

2957         (void) ddi_dma_sync(dma, 0, fibsize, DDI_DMA_SYNC_FORDEV);
3124         (void) ddi_dma_sync(acp->slotp->fib_dma_handle, 0, fibsize,
3125             DDI_DMA_SYNC_FORDEV);

2959         /* Give the FIB to the controller, wait for a response. */
2960         softs->sync_slot_busy = 1;
2961         rval = aac_sync_mbcommand(softs, AAC_MONKER_SYNCFIB,
2962             slotp->fib_phyaddr, 0, 0, 0, &status, NULL);
2963         if (rval == AACERR) {
2964                 AACDB_PRINT(softs, CE_WARN,
2965                     "Send sync fib to controller failed");
2966                 return (AACERR);
2967         }
3127         aac_start_io(softs, acp);

2969         (void) ddi_dma_sync(dma, 0, AAC_FIB_SIZE, DDI_DMA_SYNC_FORCPU);

2971         if ((aac_check_acc_handle(softs->pci_mem_handle[0]) != DDI_SUCCESS) ||
2972             (aac_check_dma_handle(dma) != DDI_SUCCESS)) {
2973                 aac_fm_service_impact(softs->devinfo_p, DDI_SERVICE_UNAFFECTED);
2974                 return (AACERR);
2975         }
```

```
2977         return (AACOK);
3129         if (softs->state & AAC_STATE_INTR)
3130                 return (aac_do_sync_io(softs, acp));
3131         else
3132                 return (aac_do_poll_io(softs, acp));
2978 }
_____unchanged_portion_omitted_

3037 /*
3038  * Atomically insert an entry into the nominated queue, returns 0 on success or
3039  * AACERR if the queue is full.
3040  *
3041  * Note: it would be more efficient to defer notifying the controller in
3042  *       the case where we may be inserting several entries in rapid succession,
3043  *       but implementing this usefully may be difficult (it would involve a
3044  *       separate queue/notify interface).
3045  */
3046 static int
3047 aac_fib_enqueue(struct aac_softstate *softs, int queue, uint32_t fib_addr,
3048     uint32_t fib_size)
3049 {
3050         ddi_dma_handle_t dma = softs->comm_space_dma_handle;
3051         ddi_acc_handle_t acc = softs->comm_space_acc_handle;
3052         uint32_t pi, ci;

3054         DBCALLED(softs, 2);

3056         ASSERT(queue == AAC_ADAP_NORM_CMD_Q || queue == AAC_ADAP_NORM_RESP_Q);

3058         /* Get the producer/consumer indices */
3059         (void) ddi_dma_sync(dma, (uint8_t *)softs->qtablep->qt_qindex[queue] - \
3060             (uint8_t *)softs->comm_space, sizeof (uint32_t) * 2,
3214         (void) ddi_dma_sync(dma, (uintptr_t)softs->qtablep->qt_qindex[queue] - \
3215             (uintptr_t)softs->comm_space, sizeof (uint32_t) * 2,
3061             DDI_DMA_SYNC_FORCPU);
3062         if (aac_check_dma_handle(dma) != DDI_SUCCESS) {
3063                 aac_fm_service_impact(softs->devinfo_p, DDI_SERVICE_UNAFFECTED);
3218                 ddi_fm_service_impact(softs->devinfo_p, DDI_SERVICE_UNAFFECTED);
3064                 return (AACERR);
3065         }

3067         pi = ddi_get32(acc,
3068             &softs->qtablep->qt_qindex[queue][AAC_PRODUCER_INDEX]);
3069         ci = ddi_get32(acc,
3070             &softs->qtablep->qt_qindex[queue][AAC_CONSUMER_INDEX]);

3072         /*
3073          * Wrap the queue first before we check the queue to see
3074          * if it is full
3075          */
3076         if (pi >= aac_qinfo[queue].size)
3077                 pi = 0;

3079         /* XXX queue full */
3080         if ((pi + 1) == ci)
3081                 return (AACERR);

3083         /* Fill in queue entry */
3084         ddi_put32(acc, &((softs->qentries[queue] + pi)->aq_fib_size), fib_size);
3085         ddi_put32(acc, &((softs->qentries[queue] + pi)->aq_fib_addr), fib_addr);
3086         (void) ddi_dma_sync(dma, (uint8_t *)(softs->qentries[queue] + pi) - \
3087             (uint8_t *)softs->comm_space, sizeof (struct aac_queue_entry),
3241         (void) ddi_dma_sync(dma, (uintptr_t)(softs->qentries[queue] + pi) - \
3242             (uintptr_t)softs->comm_space, sizeof (struct aac_queue_entry),
3088             DDI_DMA_SYNC_FORDEV);
```



```
3090         /* Update producer index */
3091         ddi_put32(acc, &softs->qtablep->qt_qindex[queue][AAC_PRODUCER_INDEX],
3092             pi + 1);
3093         (void) ddi_dma_sync(dma,
3094             (uint8_t *)&softs->qtablep->qt_qindex[queue][AAC_PRODUCER_INDEX] - \
3095             (uint8_t *)softs->comm_space, sizeof (uint32_t),
3249             (uintptr_t)&softs->qtablep->qt_qindex[queue][AAC_PRODUCER_INDEX] - \
3250             (uintptr_t)softs->comm_space, sizeof (uint32_t),
3096             DDI_DMA_SYNC_FORDEV);

3098         if (aac_qinfo[queue].notify != 0)
3099                 AAC_NOTIFY(softs, aac_qinfo[queue].notify);
3100         return (AACOK);
3101 }

3103 /*
3104  * Atomically remove one entry from the nominated queue, returns 0 on
3105  * success or AACERR if the queue is empty.
3106  */
3107 static int
3108 aac_fib_dequeue(struct aac_softstate *softs, int queue, int *idxp)
3109 {
3110         ddi_acc_handle_t acc = softs->comm_space_acc_handle;
3111         ddi_dma_handle_t dma = softs->comm_space_dma_handle;
3112         uint32_t pi, ci;
3113         int unfull = 0;

3115         DBCALLED(softs, 2);

3117         ASSERT(idxp);

3119         /* Get the producer/consumer indices */
3120         (void) ddi_dma_sync(dma, (uint8_t *)softs->qtablep->qt_qindex[queue] - \
3121             (uint8_t *)softs->comm_space, sizeof (uint32_t) * 2,
3275         (void) ddi_dma_sync(dma, (uintptr_t)softs->qtablep->qt_qindex[queue] - \
3276             (uintptr_t)softs->comm_space, sizeof (uint32_t) * 2,
3122             DDI_DMA_SYNC_FORCPU);
3123         pi = ddi_get32(acc,
3124             &softs->qtablep->qt_qindex[queue][AAC_PRODUCER_INDEX]);
3125         ci = ddi_get32(acc,
3126             &softs->qtablep->qt_qindex[queue][AAC_CONSUMER_INDEX]);

3128         /* Check for queue empty */
3129         if (ci == pi)
3130                 return (AACERR);

3132         if (pi >= aac_qinfo[queue].size)
3133                 pi = 0;

3135         /* Check for queue full */
3136         if (ci == pi + 1)
3137                 unfull = 1;

3139         /*
3140          * The controller does not wrap the queue,
3141          * so we have to do it by ourselves
3142          */
3143         if (ci >= aac_qinfo[queue].size)
3144                 ci = 0;

3146         /* Fetch the entry */
3147         (void) ddi_dma_sync(dma, (uint8_t *)(softs->qentries[queue] + pi) - \
3148             (uint8_t *)softs->comm_space, sizeof (struct aac_queue_entry),
3302         (void) ddi_dma_sync(dma, (uintptr_t)(softs->qentries[queue] + pi) - \
3303             (uintptr_t)softs->comm_space, sizeof (struct aac_queue_entry),
```

```
3149            DDI_DMA_SYNC_FORCPU);
3150        if (aac_check_dma_handle(dma) != DDI_SUCCESS) {
3151                aac_fm_service_impact(softs->devinfo_p, DDI_SERVICE_UNAFFECTED);
3306                ddi_fm_service_impact(softs->devinfo_p, DDI_SERVICE_UNAFFECTED);
3152                return (AACERR);
3153        }

3155        switch (queue) {
3156        case AAC_HOST_NORM_RESP_Q:
3157        case AAC_HOST_HIGH_RESP_Q:
3158                *idxp = ddi_get32(acc,
3159                    &(softs->qentries[queue] + ci)->aq_fib_addr);
3160                break;

3162        case AAC_HOST_NORM_CMD_Q:
3163        case AAC_HOST_HIGH_CMD_Q:
3164                *idxp = ddi_get32(acc,
3165                    &(softs->qentries[queue] + ci)->aq_fib_addr) / AAC_FIB_SIZE;
3166                break;

3168        default:
3169                cmn_err(CE_NOTE, "!Invalid queue in aac_fib_dequeue()");
3170                return (AACERR);
3171        }

3173        /* Update consumer index */
3174        ddi_put32(acc, &softs->qtablep->qt_qindex[queue][AAC_CONSUMER_INDEX],
3175            ci + 1);
3176        (void) ddi_dma_sync(dma,
3177            (uint8_t *)&softs->qtablep->qt_qindex[queue][AAC_CONSUMER_INDEX] - \
3178            (uint8_t *)softs->comm_space, sizeof (uint32_t),
3332            (uintptr_t)&softs->qtablep->qt_qindex[queue][AAC_CONSUMER_INDEX] - \
3333            (uintptr_t)softs->comm_space, sizeof (uint32_t),
3179            DDI_DMA_SYNC_FORDEV);

3181        if (unfull && aac_qinfo[queue].notify != 0)
3182                AAC_NOTIFY(softs, aac_qinfo[queue].notify);
3183        return (AACOK);
3184 }

3186 /*
3187  * Request information of the container cid
3188  */
3189 static struct aac_mntinforesp *
3190 aac_get_container_info(struct aac_softstate *softs, int cid)
3342 aac_get_mntinfo(struct aac_softstate *softs, int cid)
3191 {
3192        ddi_acc_handle_t acc = softs->sync_slot->fib_acc_handle;
3193        struct aac_fib *fibp = softs->sync_slot->fibp;
3344        ddi_acc_handle_t acc = softs->sync_ac.slotp->fib_acc_handle;
3345        struct aac_fib *fibp = softs->sync_ac.slotp->fibp;
3194        struct aac_mntinfo *mi = (struct aac_mntinfo *)&fibp->data[0];
3195        struct aac_mntinforesp *mir;

3197        ddi_put32(acc, &mi->Command, /* Use 64-bit LBA if enabled */
3198            (softs->flags & AAC_FLAGS_LBA_64BIT) ?
3199            VM_NameServe64 : VM_NameServe);
3200        ddi_put32(acc, &mi->MntType, FT_FILESYS);
3201        ddi_put32(acc, &mi->MntCount, cid);

3203        if (aac_sync_fib(softs, ContainerCommand,
3204            AAC_FIB_SIZEOF(struct aac_mntinfo)) == AACERR) {
3205                AACDB_PRINT(softs, CE_WARN, "Error probe container %d", cid);
3206                return (NULL);
3207        }
```



```
3209        mir = (struct aac_mntinforesp *)&fibp->data[0];
3210        if (ddi_get32(acc, &mir->Status) == ST_OK)
3211                return (mir);
3212        return (NULL);
3213 }

3215 static int
3216 aac_get_container_count(struct aac_softstate *softs, int *count)
3217 {
3218        ddi_acc_handle_t acc = softs->sync_slot->fib_acc_handle;
3370        ddi_acc_handle_t acc;
3219        struct aac_mntinforesp *mir;
3372        int rval;

3221        if ((mir = aac_get_container_info(softs, 0)) == NULL)
3222                return (AACERR);
3374        (void) aac_sync_fib_slot_bind(softs, &softs->sync_ac);
3375        acc = softs->sync_ac.slotp->fib_acc_handle;

3377        if ((mir = aac_get_mntinfo(softs, 0)) == NULL) {
3378                rval = AACERR;
3379                goto finish;
3380        }
3223        *count = ddi_get32(acc, &mir->MntRespCount);
3224        if (*count > AAC_MAX_LD) {
3225                AACDB_PRINT(softs, CE_CONT,
3226                    "container count(%d) > AAC_MAX_LD", *count);
3227                return (AACERR);
3385                rval = AACERR;
3386                goto finish;
3228        }
3229        return (AACOK);
3388        rval = AACOK;

3390 finish:
3391        aac_sync_fib_slot_release(softs, &softs->sync_ac);
3392        return (rval);
3230 }

3232 static int
3233 aac_get_container_uid(struct aac_softstate *softs, uint32_t cid, uint32_t *uid)
3234 {
3235        ddi_acc_handle_t acc = softs->sync_slot->fib_acc_handle;
3398        ddi_acc_handle_t acc = softs->sync_ac.slotp->fib_acc_handle;
3236        struct aac_Container *ct = (struct aac_Container *) \
3237            &softs->sync_slot->fibp->data[0];
3400            &softs->sync_ac.slotp->fibp->data[0];

3239        bzero(ct, sizeof (*ct) - CT_PACKET_SIZE);
3240        ddi_put32(acc, &ct->Command, VM_ContainerConfig);
3241        ddi_put32(acc, &ct->CTCommand.command, CT_CID_TO_32BITS_UID);
3242        ddi_put32(acc, &ct->CTCommand.param[0], cid);

3244        if (aac_sync_fib(softs, ContainerCommand,
3245            AAC_FIB_SIZEOF(struct aac_Container)) == AACERR)
3246                return (AACERR);
3247        if (ddi_get32(acc, &ct->CTCommand.param[0]) != CT_OK)
3248                return (AACERR);

3250        *uid = ddi_get32(acc, &ct->CTCommand.param[1]);
3251        return (AACOK);
3252 }

3254 static int
3417 /*
3418  * Request information of the container cid
```

```
 */
static struct aac_mntinforesp *
aac_get_container_info(struct aac_softstate *softs, int cid)
{
        ddi_acc_handle_t acc = softs->sync_ac.slotp->fib_acc_handle;
        struct aac_mntinforesp *mir;
        int rval_uid;
        uint32_t uid;

        /* Get container UID first so that it will not overwrite mntinfo */
        rval_uid = aac_get_container_uid(softs, cid, &uid);

        /* Get container basic info */
        if ((mir = aac_get_mntinfo(softs, cid)) == NULL) {
                AACDB_PRINT(softs, CE_CONT,
                    "query container %d info failed", cid);
                return (NULL);
        }
        if (ddi_get32(acc, &mir->MntObj.VolType) == CT_NONE)
                return (mir);
        if (rval_uid != AACOK) {
                AACDB_PRINT(softs, CE_CONT,
                    "query container %d uid failed", cid);
                return (NULL);
        }

        ddi_put32(acc, &mir->Status, uid);
        return (mir);
}

static enum aac_cfg_event
aac_probe_container(struct aac_softstate *softs, uint32_t cid)
{
        enum aac_cfg_event event = AAC_CFG_NULL_NOEXIST;
        struct aac_container *dvp = &softs->containers[cid];
        ddi_acc_handle_t acc = softs->sync_slot->fib_acc_handle;
        struct aac_mntinforesp *mir;
        uint64_t size;
        uint32_t uid;
        ddi_acc_handle_t acc;

        (void) aac_sync_fib_slot_bind(softs, &softs->sync_ac);
        acc = softs->sync_ac.slotp->fib_acc_handle;

        /* Get container basic info */
        if ((mir = aac_get_container_info(softs, cid)) == NULL)
                return (AACERR);
        if ((mir = aac_get_container_info(softs, cid)) == NULL) {
                /* AAC_CFG_NULL_NOEXIST */
                goto finish;
        }

        if (ddi_get32(acc, &mir->MntObj.VolType) == CT_NONE) {
                if (dvp->dev.valid) {
                if (AAC_DEV_IS_VALID(&dvp->dev)) {
                        AACDB_PRINT(softs, CE_NOTE,
                            ">>> Container %d deleted", cid);
                        dvp->dev.valid = 0;
                        (void) aac_dr_event(softs, dvp->cid, -1,
                            AAC_DEV_OFFLINE);
                        dvp->dev.flags &= ~AAC_DFLAG_VALID;
                        event = AAC_CFG_DELETE;
                }
                /* AAC_CFG_NULL_NOEXIST */
        } else {
                size = AAC_MIR_SIZE(softs, acc, mir);
```



```
                uint64_t size;
                uint32_t uid;

                /* Get container UID */
                if (aac_get_container_uid(softs, cid, &uid) == AACERR) {
                        AACDB_PRINT(softs, CE_CONT,
                            "query container %d uid failed", cid);
                        return (AACERR);
                }
                AACDB_PRINT(softs, CE_CONT, "uid=0x%08x", uid);
                event = AAC_CFG_NULL_EXIST;

                if (dvp->dev.valid) {
                size = AAC_MIR_SIZE(softs, acc, mir);
                uid = ddi_get32(acc, &mir->Status);
                if (AAC_DEV_IS_VALID(&dvp->dev)) {
                        if (dvp->uid != uid) {
                                AACDB_PRINT(softs, CE_WARN,
                                    ">>> Container %u uid changed to %d",
                                    cid, uid);
                                dvp->uid = uid;
                                event = AAC_CFG_CHANGE;
                        }
                        if (dvp->size != size) {
                                AACDB_PRINT(softs, CE_NOTE,
                                    ">>> Container %u size changed to %"PRIu64,
                                    cid, size);
                                dvp->size = size;
                                event = AAC_CFG_CHANGE;
                        }
                } else { /* Init new container */
                        AACDB_PRINT(softs, CE_NOTE,
                            ">>> Container %d added: " \
                            "size=0x%x.%08x, type=%d, name=%s",
                            cid,
                            ddi_get32(acc, &mir->MntObj.CapacityHigh),
                            ddi_get32(acc, &mir->MntObj.Capacity),
                            ddi_get32(acc, &mir->MntObj.VolType),
                            mir->MntObj.FileSystemName);
                        dvp->dev.valid = 1;
                        dvp->dev.flags |= AAC_DFLAG_VALID;
                        dvp->dev.type = AAC_DEV_LD;

                        dvp->cid = cid;
                        dvp->uid = uid;
                        dvp->size = size;
                        dvp->locked = 0;
                        dvp->deleted = 0;
                        (void) aac_dr_event(softs, dvp->cid, -1,
                            AAC_DEV_ONLINE);

                        event = AAC_CFG_ADD;
                }
        }
        return (AACOK);

finish:
        aac_sync_fib_slot_release(softs, &softs->sync_ac);
        return (event);
}

/*
 * Do a rescan of all the possible containers and update the container list
 * with newly online/offline containers, and prepare for autoconfiguration.
 */
static int
```

```
3328 aac_probe_containers(struct aac_softstate *softs)
3329 {
3330         int i, count, total;

3332         /* Loop over possible containers */
3333         count = softs->container_count;
3334         if (aac_get_container_count(softs, &count) == AACERR)
3335                 return (AACERR);

3336         for (i = total = 0; i < count; i++) {
3337                 if (aac_probe_container(softs, i) == AACOK)
3539                 enum aac_cfg_event event = aac_probe_container(softs, i);
3540                 if ((event != AAC_CFG_NULL_NOEXIST) &&
3541                     (event != AAC_CFG_NULL_EXIST)) {
3542                         (void) aac_handle_dr(softs, i, -1, event);
3338                         total++;
3339         }
3545         }

3340         if (count < softs->container_count) {
3341                 struct aac_container *dvp;

3343                 for (dvp = &softs->containers[count];
3344                     dvp < &softs->containers[softs->container_count]; dvp++) {
3345                         if (dvp->dev.valid == 0)
3552                         if (!AAC_DEV_IS_VALID(&dvp->dev))
3346                                 continue;
3347                         AACDB_PRINT(softs, CE_NOTE, ">>> Container %d deleted",
3348                             dvp->cid);
3349                         dvp->dev.valid = 0;
3350                         (void) aac_dr_event(softs, dvp->cid, -1,
3351                             AAC_DEV_OFFLINE);
3556                         dvp->dev.flags &= ~AAC_DFLAG_VALID;
3557                         (void) aac_handle_dr(softs, dvp->cid, -1,
3558                             AAC_CFG_DELETE);
3352                 }
3353         }

3354         softs->container_count = count;
3355         AACDB_PRINT(softs, CE_CONT, "Total %d container(s) found", total);
3563         AACDB_PRINT(softs, CE_CONT, "?Total %d container(s) found", total);
3356         return (AACOK);
3357 }

3359 static int
3568 aac_probe_jbod(struct aac_softstate *softs, int tgt, int event)
3569 {
3570         ASSERT(AAC_MAX_LD <= tgt);
3571         ASSERT(tgt < AAC_MAX_DEV(softs));
3572         struct aac_device *dvp;
3573         dvp = AAC_DEV(softs, tgt);

3575         switch (event) {
3576         case AAC_CFG_ADD:
3577                 AACDB_PRINT(softs, CE_NOTE,
3578                     ">>> Jbod %d added", tgt - AAC_MAX_LD);
3579                 dvp->flags |= AAC_DFLAG_VALID;
3580                 dvp->type = AAC_DEV_PD;
3581                 break;
3582         case AAC_CFG_DELETE:
3583                 AACDB_PRINT(softs, CE_NOTE,
3584                     ">>> Jbod %d deleted", tgt - AAC_MAX_LD);
3585                 dvp->flags &= ~AAC_DFLAG_VALID;
3586                 break;
3587         default:
3588                 return (AACERR);
```



```
3589         }
3590         (void) aac_handle_dr(softs, tgt, 0, event);
3591         return (AACOK);
3592 }

3594 static int
3360 aac_alloc_comm_space(struct aac_softstate *softs)
3361 {
3362         size_t rlen, maxsize;
3597         size_t rlen;
3363         ddi_dma_cookie_t cookie;
3364         uint_t cookien;

3366         /* Allocate DMA for comm. space */
3367         if (ddi_dma_alloc_handle(
3368             softs->devinfo_p,
3369             &softs->addr_dma_attr,
3370             DDI_DMA_SLEEP,
3371             NULL,
3372             &softs->comm_space_dma_handle) != DDI_SUCCESS) {
3373                 AACDB_PRINT(softs, CE_WARN,
3374                     "Cannot alloc dma handle for communication area");
3375                 goto error;
3376         }

3378         maxsize = sizeof(struct aac_comm_space);
3379         if ((softs->flags & AAC_FLAGS_NEW_COMM_TYPE1) ||
3380                 (softs->flags & AAC_FLAGS_NEW_COMM_TYPE2))
3381                 maxsize += (softs->aac_max_fibs-1) * sizeof(uint32_t);
3382         if (ddi_dma_mem_alloc(
3383             softs->comm_space_dma_handle,
3384             maxsize,
3385             &aac_acc_attr,
3614             sizeof (struct aac_comm_space),
3615             &softs->acc_attr,
3386             DDI_DMA_RDWR | DDI_DMA_CONSISTENT,
3387             DDI_DMA_SLEEP,
3388             NULL,
3389             (caddr_t *)&softs->comm_space,
3390             &rlen,
3391             &softs->comm_space_acc_handle) != DDI_SUCCESS) {
3392                 AACDB_PRINT(softs, CE_WARN,
3393                     "Cannot alloc mem for communication area");
3394                 goto error;
3395         }
3396         if (ddi_dma_addr_bind_handle(
3397             softs->comm_space_dma_handle,
3398             NULL,
3399             (caddr_t)softs->comm_space,
3400             maxsize,
3630             sizeof (struct aac_comm_space),
3401             DDI_DMA_RDWR | DDI_DMA_CONSISTENT,
3402             DDI_DMA_SLEEP,
3403             NULL,
3404             &cookie,
3405             &cookien) != DDI_DMA_MAPPED) {
3406                 AACDB_PRINT(softs, CE_WARN,
3407                     "DMA bind failed for communication area");
3408                 goto error;
3409         }

3411         bzero(softs->comm_space, maxsize);
3412         softs->comm_space_phyaddr = cookie.dmac_address;

3414         return (AACOK);
3415 error:
```

```
3416         if (softs->comm_space_acc_handle) {
3417                 ddi_dma_mem_free(&softs->comm_space_acc_handle);
3418                 softs->comm_space_acc_handle = NULL;
3419         }
3420         if (softs->comm_space_dma_handle) {
3421                 ddi_dma_free_handle(&softs->comm_space_dma_handle);
3422                 softs->comm_space_dma_handle = NULL;
3423         }
3424         return (AACERR);
3425 }

3427 static void
3428 aac_free_comm_space(struct aac_softstate *softs)
3429 {

3430         (void) ddi_dma_unbind_handle(softs->comm_space_dma_handle);
3431         ddi_dma_mem_free(&softs->comm_space_acc_handle);
3432         softs->comm_space_acc_handle = NULL;
3433         ddi_dma_free_handle(&softs->comm_space_dma_handle);
3434         softs->comm_space_dma_handle = NULL;
3435         softs->comm_space_phyaddr = NULL;
3436 }

3438 /*
3439  * Initialize the data structures that are required for the communication
3440  * interface to operate
3441  */
3442 static int
3443 aac_setup_comm_space(struct aac_softstate *softs)
3444 {
3445         ddi_dma_handle_t dma = softs->comm_space_dma_handle;
3446         ddi_acc_handle_t acc = softs->comm_space_acc_handle;
3447         uint32_t comm_space_phyaddr;
3448         struct aac_adapter_init *initp;
3449         int qoffset;

3451         comm_space_phyaddr = softs->comm_space_phyaddr;

3453         /* reset rrq index */
3454         softs->aac_host_rrq_idx = 0;

3456         /* Setup adapter init struct */
3457         initp = &softs->comm_space->init_data;
3684         bzero(initp, sizeof (struct aac_adapter_init));

3459         ddi_put32(acc, &initp->InitStructRevision, AAC_INIT_STRUCT_REVISION);
3460         ddi_put32(acc, &initp->HostElapsedSeconds, ddi_get_time());
3461         ddi_put32(acc, &initp->MiniPortRevision,AAC_INIT_STRUCT_MINIPORT_REVISIO

3463         /* Setup new/old comm. specific data */
3464         if (softs->flags & AAC_FLAGS_NEW_COMM) {
3690         if (softs->flags & AAC_FLAGS_RAW_IO) {
3465                 uint32_t init_flags = 0;

3693                 if (softs->flags & AAC_FLAGS_NEW_COMM)
3467                 init_flags |= AAC_INIT_FLAGS_NEW_COMM_SUPPORTED;
3468                 if (softs->flags & AAC_FLAGS_NEW_COMM_TYPE1) {
3469                         init_flags |= (AAC_INITFLAGS_NEW_COMM_TYPE1_SUPPORTED |
3470                                 AAC_INITFLAGS_DRIVER_SUPPORTS_FAST_JBOD);
3471                         ddi_put32(acc, &initp->InitStructRevision,
3472                                 AAC_INIT_STRUCT_REVISION_6);
3473                 } else if (softs->flags & AAC_FLAGS_NEW_COMM_TYPE2) {
3474                         init_flags |= (AAC_INITFLAGS_NEW_COMM_TYPE2_SUPPORTED |
3475                                 AAC_INITFLAGS_DRIVER_SUPPORTS_FAST_JBOD);
3476                         ddi_put32(acc, &initp->InitStructRevision,
3477                                 AAC_INIT_STRUCT_REVISION_7);
```



```
3478                         ddi_put32(acc, &initp->MiniPortRevision, 0L);
3479                 } else {
3480                         ddi_put32(acc, &initp->InitStructRevision,
3481                                 AAC_INIT_STRUCT_REVISION_4);
3482                 }
3483                 if (softs->aac_support_opt2 & AAC_SUPPORTED_POWER_MANAGEMENT) {
3695                 /* AAC_SUPPORTED_POWER_MANAGEMENT */
3484                         init_flags |= AAC_INIT_FLAGS_DRIVER_SUPPORTS_PM;
3485                         init_flags |= AAC_INIT_FLAGS_DRIVER_USES_UTC_TIME;
3486                 }

3699                 ddi_put32(acc, &initp->InitStructRevision,
3700                     AAC_INIT_STRUCT_REVISION_4);
3488                 ddi_put32(acc, &initp->InitFlags, init_flags);
3489                 /* Setup the preferred settings */
3490                 ddi_put32(acc, &initp->MaxIoCommands, softs->aac_max_fibs);
3491                 ddi_put32(acc, &initp->MaxIoSize,
3492                     (softs->aac_max_sectors << 9));
3493                 ddi_put32(acc, &initp->MaxFibSize, softs->aac_max_fib_size);
3494                 ddi_put32(acc, &initp->MaxNumAif, softs->aac_max_aif);
3495                 ddi_put32(acc, &initp->HostRRQ_AddrLow,
3496                         comm_space_phyaddr + offsetof(struct aac_comm_space,aac_
3497         } else {
3498                 /*
3499                  * Tells the adapter about the physical location of various
3500                  * important shared data structures
3501                  */
3502                 ddi_put32(acc, &initp->AdapterFibsPhysicalAddress,
3503                     comm_space_phyaddr + \
3504                     offsetof(struct aac_comm_space, adapter_fibs));
3505                 ddi_put32(acc, &initp->AdapterFibsVirtualAddress, 0);
3506                 ddi_put32(acc, &initp->AdapterFibAlign, AAC_FIB_SIZE);
3507                 ddi_put32(acc, &initp->AdapterFibsSize,
3508                     AAC_ADAPTER_FIBS * AAC_FIB_SIZE);
3509                 ddi_put32(acc, &initp->PrintfBufferAddress,
3510                     comm_space_phyaddr + \
3511                     offsetof(struct aac_comm_space, adapter_print_buf));
3512                 ddi_put32(acc, &initp->PrintfBufferSize,
3513                     AAC_ADAPTER_PRINT_BUFSIZE);
3724                 ddi_put32(acc, &initp->MiniPortRevision,
3725                     AAC_INIT_STRUCT_MINIPORT_REVISION);
3514                 ddi_put32(acc, &initp->HostPhysMemPages, AAC_MAX_PFN);

3516                 qoffset = (comm_space_phyaddr + \
3517                     offsetof(struct aac_comm_space, qtable)) % \
3518                     AAC_QUEUE_ALIGN;
3519                 if (qoffset)
3520                         qoffset = AAC_QUEUE_ALIGN - qoffset;
3521                 softs->qtablep = (struct aac_queue_table *) \
3522                     ((char *)&softs->comm_space->qtable + qoffset);
3523                 ddi_put32(acc, &initp->CommHeaderAddress, comm_space_phyaddr + \
3524                     offsetof(struct aac_comm_space, qtable) + qoffset);

3526                 /* Init queue table */
3527                 ddi_put32(acc, &softs->qtablep-> \
3528                     qt_qindex[AAC_HOST_NORM_CMD_Q][AAC_PRODUCER_INDEX],
3529                     AAC_HOST_NORM_CMD_ENTRIES);
3530                 ddi_put32(acc, &softs->qtablep-> \
3531                     qt_qindex[AAC_HOST_NORM_CMD_Q][AAC_CONSUMER_INDEX],
3532                     AAC_HOST_NORM_CMD_ENTRIES);
3533                 ddi_put32(acc, &softs->qtablep-> \
3534                     qt_qindex[AAC_HOST_HIGH_CMD_Q][AAC_PRODUCER_INDEX],
3535                     AAC_HOST_HIGH_CMD_ENTRIES);
3536                 ddi_put32(acc, &softs->qtablep-> \
3537                     qt_qindex[AAC_HOST_HIGH_CMD_Q][AAC_CONSUMER_INDEX],
3538                     AAC_HOST_HIGH_CMD_ENTRIES);
```

```
3539                ddi_put32(acc, &softs->qtablep-> \
3540                    qt_qindex[AAC_ADAP_NORM_CMD_Q][AAC_PRODUCER_INDEX],
3541                    AAC_ADAP_NORM_CMD_ENTRIES);
3542                ddi_put32(acc, &softs->qtablep-> \
3543                    qt_qindex[AAC_ADAP_NORM_CMD_Q][AAC_CONSUMER_INDEX],
3544                    AAC_ADAP_NORM_CMD_ENTRIES);
3545                ddi_put32(acc, &softs->qtablep-> \
3546                    qt_qindex[AAC_ADAP_HIGH_CMD_Q][AAC_PRODUCER_INDEX],
3547                    AAC_ADAP_HIGH_CMD_ENTRIES);
3548                ddi_put32(acc, &softs->qtablep-> \
3549                    qt_qindex[AAC_ADAP_HIGH_CMD_Q][AAC_CONSUMER_INDEX],
3550                    AAC_ADAP_HIGH_CMD_ENTRIES);
3551                ddi_put32(acc, &softs->qtablep-> \
3552                    qt_qindex[AAC_HOST_NORM_RESP_Q][AAC_PRODUCER_INDEX],
3553                    AAC_HOST_NORM_RESP_ENTRIES);
3554                ddi_put32(acc, &softs->qtablep-> \
3555                    qt_qindex[AAC_HOST_NORM_RESP_Q][AAC_CONSUMER_INDEX],
3556                    AAC_HOST_NORM_RESP_ENTRIES);
3557                ddi_put32(acc, &softs->qtablep-> \
3558                    qt_qindex[AAC_HOST_HIGH_RESP_Q][AAC_PRODUCER_INDEX],
3559                    AAC_HOST_HIGH_RESP_ENTRIES);
3560                ddi_put32(acc, &softs->qtablep-> \
3561                    qt_qindex[AAC_HOST_HIGH_RESP_Q][AAC_CONSUMER_INDEX],
3562                    AAC_HOST_HIGH_RESP_ENTRIES);
3563                ddi_put32(acc, &softs->qtablep-> \
3564                    qt_qindex[AAC_ADAP_NORM_RESP_Q][AAC_PRODUCER_INDEX],
3565                    AAC_ADAP_NORM_RESP_ENTRIES);
3566                ddi_put32(acc, &softs->qtablep-> \
3567                    qt_qindex[AAC_ADAP_NORM_RESP_Q][AAC_CONSUMER_INDEX],
3568                    AAC_ADAP_NORM_RESP_ENTRIES);
3569                ddi_put32(acc, &softs->qtablep-> \
3570                    qt_qindex[AAC_ADAP_HIGH_RESP_Q][AAC_PRODUCER_INDEX],
3571                    AAC_ADAP_HIGH_RESP_ENTRIES);
3572                ddi_put32(acc, &softs->qtablep-> \
3573                    qt_qindex[AAC_ADAP_HIGH_RESP_Q][AAC_CONSUMER_INDEX],
3574                    AAC_ADAP_HIGH_RESP_ENTRIES);

3576                /* Init queue entries */
3577                softs->qentries[AAC_HOST_NORM_CMD_Q] =
3578                    &softs->qtablep->qt_HostNormCmdQueue[0];
3579                softs->qentries[AAC_HOST_HIGH_CMD_Q] =
3580                    &softs->qtablep->qt_HostHighCmdQueue[0];
3581                softs->qentries[AAC_ADAP_NORM_CMD_Q] =
3582                    &softs->qtablep->qt_AdapNormCmdQueue[0];
3583                softs->qentries[AAC_ADAP_HIGH_CMD_Q] =
3584                    &softs->qtablep->qt_AdapHighCmdQueue[0];
3585                softs->qentries[AAC_HOST_NORM_RESP_Q] =
3586                    &softs->qtablep->qt_HostNormRespQueue[0];
3587                softs->qentries[AAC_HOST_HIGH_RESP_Q] =
3588                    &softs->qtablep->qt_HostHighRespQueue[0];
3589                softs->qentries[AAC_ADAP_NORM_RESP_Q] =
3590                    &softs->qtablep->qt_AdapNormRespQueue[0];
3591                softs->qentries[AAC_ADAP_HIGH_RESP_Q] =
3592                    &softs->qtablep->qt_AdapHighRespQueue[0];
3593        }
3594        (void) ddi_dma_sync(dma, 0, 0, DDI_DMA_SYNC_FORDEV);

3596        /* Send init structure to the card */
3597        softs->sync_slot_busy = 1;
3598        if (aac_sync_mbcommand(softs, AAC_MONKER_INITSTRUCT,
3599            comm_space_phyaddr + \
3600            offsetof(struct aac_comm_space, init_data),
3601            0, 0, 0, NULL, NULL) == AACERR) {
3812            0, 0, 0, NULL) == AACERR) {
3602                AACDB_PRINT(softs, CE_WARN,
3603                    "Cannot send init structure to adapter");
```



```
3604                return (AACERR);
3605        }

3607        return (AACOK);
3608 }
_____unchanged_portion_omitted_

3646 /*
3647  * SPC-3 7.5 INQUIRY command implementation
3648  */
3649 static void
3650 aac_inquiry(struct aac_softstate *softs, struct scsi_pkt *pkt,
3651     union scsi_cdb *cdbp, struct buf *bp)
3652 {
3653        int tgt = pkt->pkt_address.a_target;
3654        char *b_addr = NULL;
3655        uchar_t page = cdbp->cdb_opaque[2];

3657        if (cdbp->cdb_opaque[1] & AAC_CDB_INQUIRY_CMDDT) {
3658                /* Command Support Data is not supported */
3659                aac_set_arq_data(pkt, KEY_ILLEGAL_REQUEST, 0x24, 0x00, 0);
3660                return;
3661        }

3663        if (bp && bp->b_un.b_addr && bp->b_bcount) {
3664                if (bp->b_flags & (B_PHYS | B_PAGEIO))
3665                        bp_mapin(bp);
3666                b_addr = bp->b_un.b_addr;
3667        }

3669        if (cdbp->cdb_opaque[1] & AAC_CDB_INQUIRY_EVPD) {
3670                uchar_t *vpdp = (uchar_t *)b_addr;
3671                uchar_t *idp, *sp;

3673                /* SPC-3 8.4 Vital product data parameters */
3674                switch (page) {
3675                case 0x00:
3676                        /* Supported VPD pages */
3677                        if (vpdp == NULL ||
3678                                bp->b_bcount < (AAC_VPD_PAGE_DATA + 3))
3679                                return;
3680                        bzero(vpdp, AAC_VPD_PAGE_LENGTH);
3681                        vpdp[AAC_VPD_PAGE_CODE] = 0x00;
3682                        vpdp[AAC_VPD_PAGE_LENGTH] = 3;

3684                        vpdp[AAC_VPD_PAGE_DATA] = 0x00;
3685                        vpdp[AAC_VPD_PAGE_DATA + 1] = 0x80;
3686                        vpdp[AAC_VPD_PAGE_DATA + 2] = 0x83;

3688                        pkt->pkt_state |= STATE_XFERRED_DATA;
3689                        break;

3691                case 0x80:
3692                        /* Unit serial number page */
3693                        if (vpdp == NULL ||
3694                                bp->b_bcount < (AAC_VPD_PAGE_DATA + 8))
3695                                return;
3696                        bzero(vpdp, AAC_VPD_PAGE_LENGTH);
3697                        vpdp[AAC_VPD_PAGE_CODE] = 0x80;
3698                        vpdp[AAC_VPD_PAGE_LENGTH] = 8;

3700                        sp = &vpdp[AAC_VPD_PAGE_DATA];
3701                        (void) aac_lun_serialno(softs, tgt, sp);

3703                        pkt->pkt_state |= STATE_XFERRED_DATA;
3704                        break;
```

```
3706                case 0x83:
3707                        /* Device identification page */
3708                        if (vpdp == NULL ||
3709                            bp->b_bcount < (AAC_VPD_ID_DATA + 32))
3920                            bp->b_bcount < (AAC_VPD_PAGE_DATA + 32))
3710                                return;
3711                        bzero(vpdp, AAC_VPD_PAGE_LENGTH);
3712                        vpdp[AAC_VPD_PAGE_CODE] = 0x83;

3714                        idp = &vpdp[AAC_VPD_PAGE_DATA];
3715                        bzero(idp, AAC_VPD_ID_LENGTH);
3716                        idp[AAC_VPD_ID_CODESET] = 0x02;
3717                        idp[AAC_VPD_ID_TYPE] = 0x01;

3719                        /*
3720                         * SPC-3 Table 111 - Identifier type
3721                         * One recommanded method of constructing the remainder
3722                         * of identifier field is to concatenate the product
3723                         * identification field from the standard INQUIRY data
3724                         * field and the product serial number field from the
3725                         * unit serial number page.
3726                         */
3727                        sp = &idp[AAC_VPD_ID_DATA];
3728                        sp = aac_vendor_id(softs, sp);
3729                        sp = aac_product_id(softs, sp);
3730                        sp = aac_lun_serialno(softs, tgt, sp);
3731                        idp[AAC_VPD_ID_LENGTH] = sp - &idp[AAC_VPD_ID_DATA];
3942                        idp[AAC_VPD_ID_LENGTH] = (uintptr_t)sp - \
3943                            (uintptr_t)&idp[AAC_VPD_ID_DATA];

3733                        vpdp[AAC_VPD_PAGE_LENGTH] =
3734                            sp - &vpdp[AAC_VPD_PAGE_DATA];
3945                        vpdp[AAC_VPD_PAGE_LENGTH] = (uintptr_t)sp - \
3946                            (uintptr_t)&vpdp[AAC_VPD_PAGE_DATA];
3735                        pkt->pkt_state |= STATE_XFERRED_DATA;
3736                        break;

3738                default:
3739                        aac_set_arq_data(pkt, KEY_ILLEGAL_REQUEST,
3740                            0x24, 0x00, 0);
3741                        break;
3742                }
3743        } else {
3744                struct scsi_inquiry *inqp = (struct scsi_inquiry *)b_addr;
3745                size_t len = sizeof (struct scsi_inquiry);

3747                if (page != 0) {
3748                        aac_set_arq_data(pkt, KEY_ILLEGAL_REQUEST,
3749                            0x24, 0x00, 0);
3750                        return;
3751                }
3752                if (inqp == NULL || bp->b_bcount < len)
3753                        return;

3755                bzero(inqp, len);
3756                inqp->inq_len = AAC_ADDITIONAL_LEN;
3757                inqp->inq_ansi = AAC_ANSI_VER;
3758                inqp->inq_rdf = AAC_RESP_DATA_FORMAT;
3759                (void) aac_vendor_id(softs, (uchar_t *)inqp->inq_vid);
3760                (void) aac_product_id(softs, (uchar_t *)inqp->inq_pid);
3761                bcopy("V1.0", inqp->inq_revision, 4);
3762                inqp->inq_cmdque = 1; /* enable tagged-queuing */
3763                /*
3764                 * For "sd-max-xfer-size" property which may impact performance
3765                 * when IO threads increase.
```



```
3766                 */
3767                inqp->inq_wbus32 = 1;

3769                pkt->pkt_state |= STATE_XFERRED_DATA;
3770        }
3771 }

3773 /*
3774  * SPC-3 7.10 MODE SENSE command implementation
3775  */
3776 static void
3777 aac_mode_sense(struct aac_softstate *softs, struct scsi_pkt *pkt,
3778     union scsi_cdb *cdbp, struct buf *bp, int capacity)
3779 {
3780         uchar_t pagecode;
3781         struct mode_format *page3p;
3782         struct mode_geometry *page4p;
3783         struct mode_header *headerp;
3994         struct mode_header_g1 *g1_headerp;
3784         unsigned int ncyl;
3785         union {
3786                 struct mode_header header;
3787                 struct {
3788                         uchar_t header[MODE_HEADER_LENGTH + MODE_BLK_DESC_LENGTH
3789                         struct mode_format page;
3790                 } page3;
3791                 struct {
3792                         uchar_t header[MODE_HEADER_LENGTH + MODE_BLK_DESC_LENGTH
3793                         struct mode_geometry page;
3794                 } page4;
3795                 struct {
3796                         uchar_t header[MODE_HEADER_LENGTH + MODE_BLK_DESC_LENGTH
3797                         struct mode_control_scsi3 page;
3798                 } pageA;
3799         } mode;
3996         caddr_t sense_data;
3997         caddr_t next_page;
3998         size_t sdata_size;
3999         size_t pages_size;
4000         int unsupport_page = 0;

4002         ASSERT(cdbp->scc_cmd == SCMD_MODE_SENSE ||
4003             cdbp->scc_cmd == SCMD_MODE_SENSE_G1);

3801         if (!(bp && bp->b_un.b_addr && bp->b_bcount))
3802                 return;

3804         if (bp->b_flags & (B_PHYS | B_PAGEIO))
3805                 bp_mapin(bp);
3806         pkt->pkt_state |= STATE_XFERRED_DATA;
3807         bzero(&mode, sizeof(mode));
3808         bzero(bp->b_un.b_addr, bp->b_bcount);
3809         pagecode = cdbp->cdb_un.sg.scsi[0];
3810         headerp = &mode.header;
4011         pagecode = cdbp->cdb_un.sg.scsi[0] & 0x3F;

4013         /* calculate the size of needed buffer */
4014         if (cdbp->scc_cmd == SCMD_MODE_SENSE)
4015                 sdata_size = MODE_HEADER_LENGTH;
4016         else /* must be SCMD_MODE_SENSE_G1 */
4017                 sdata_size = MODE_HEADER_LENGTH_G1;

4019         pages_size = 0;
3812         switch (pagecode) {
3813         /* SBC-3 7.1.3.3 Format device page */
3814         case SD_MODE_SENSE_PAGE3_CODE:
```

```
3815                    headerp->bdesc_length = MODE_BLK_DESC_LENGTH;
3816                    page3p = &mode.page3.page;
4022                    pages_size += sizeof (struct mode_format);
4023                    break;

4025            case SD_MODE_SENSE_PAGE4_CODE:
4026                    pages_size += sizeof (struct mode_geometry);
4027                    break;

4029            case MODEPAGE_CTRL_MODE:
4030                    if (softs->flags & AAC_FLAGS_LBA_64BIT) {
4031                            pages_size += sizeof (struct mode_control_scsi3);
4032                    } else {
4033                            unsupport_page = 1;
4034                    }
4035                    break;

4037            case MODEPAGE_ALLPAGES:
4038                    if (softs->flags & AAC_FLAGS_LBA_64BIT) {
4039                            pages_size += sizeof (struct mode_format) +
4040                                sizeof (struct mode_geometry) +
4041                                sizeof (struct mode_control_scsi3);
4042                    } else {
4043                            pages_size += sizeof (struct mode_format) +
4044                                sizeof (struct mode_geometry);
4045                    }
4046                    break;

4048            default:
4049                    /* unsupported pages */
4050                    unsupport_page = 1;
4051            }

4053            /* allocate buffer to fill the send data */
4054            sdata_size += pages_size;
4055            sense_data = kmem_zalloc(sdata_size, KM_SLEEP);

4057            if (cdbp->scc_cmd == SCMD_MODE_SENSE) {
4058                    headerp = (struct mode_header *)sense_data;
4059                    headerp->length = MODE_HEADER_LENGTH + pages_size -
4060                        sizeof (headerp->length);
4061                    headerp->bdesc_length = 0;
4062                    next_page = sense_data + sizeof (struct mode_header);
4063            } else {
4064                    g1_headerp = (void *)sense_data;
4065                    g1_headerp->length = BE_16(MODE_HEADER_LENGTH_G1 + pages_size -
4066                        sizeof (g1_headerp->length));
4067                    g1_headerp->bdesc_length = 0;
4068                    next_page = sense_data + sizeof (struct mode_header_g1);
4069            }

4071            if (unsupport_page)
4072                    goto finish;

4074            if (pagecode == SD_MODE_SENSE_PAGE3_CODE ||
4075                pagecode == MODEPAGE_ALLPAGES) {
4076                    /* SBC-3 7.1.3.3 Format device page */
4077                    struct mode_format *page3p;

4079                    page3p = (void *)next_page;
3817                    page3p->mode_page.code = SD_MODE_SENSE_PAGE3_CODE;
3818                    page3p->mode_page.length = sizeof (struct mode_format);
3819                    page3p->data_bytes_sect = BE_16(AAC_SECTOR_SIZE);
3820                    page3p->sect_track = BE_16(AAC_SECTORS_PER_TRACK);
3821                    break;
```



```
4085                    next_page += sizeof (struct mode_format);
4086            }

4088            if (pagecode == SD_MODE_SENSE_PAGE4_CODE ||
4089                pagecode == MODEPAGE_ALLPAGES) {
3823            /* SBC-3 7.1.3.8 Rigid disk device geometry page */
3824            case SD_MODE_SENSE_PAGE4_CODE:
3825                    headerp->bdesc_length = MODE_BLK_DESC_LENGTH;
3826                    page4p = &mode.page4.page;
4091                    struct mode_geometry *page4p;

4093                    page4p = (void *)next_page;
3827                    page4p->mode_page.code = SD_MODE_SENSE_PAGE4_CODE;
3828                    page4p->mode_page.length = sizeof (struct mode_geometry);
3829                    page4p->heads = AAC_NUMBER_OF_HEADS;
3830                    page4p->rpm = BE_16(AAC_ROTATION_SPEED);
3831                    ncyl = capacity / (AAC_NUMBER_OF_HEADS * AAC_SECTORS_PER_TRACK);
3832                    page4p->cyl_lb = ncyl & 0xff;
3833                    page4p->cyl_mb = (ncyl >> 8) & 0xff;
3834                    page4p->cyl_ub = (ncyl >> 16) & 0xff;
3835                    break;

3837            case MODEPAGE_CTRL_MODE: /* 64-bit LBA need large sense data */
3838                    if (softs->flags & AAC_FLAGS_LBA_64BIT) {
4103                    next_page += sizeof (struct mode_geometry);
4104            }

4106            if ((pagecode == MODEPAGE_CTRL_MODE || pagecode == MODEPAGE_ALLPAGES) &&
4107                softs->flags & AAC_FLAGS_LBA_64BIT) {
4108                    /* 64-bit LBA need large sense data */
3839                            struct mode_control_scsi3 *mctl;

3841                            headerp->bdesc_length = MODE_BLK_DESC_LENGTH;
3842                            mctl = &mode.pageA.page;
4111                    mctl = (void *)next_page;
3843                            mctl->mode_page.code = MODEPAGE_CTRL_MODE;
3844                            mctl->mode_page.length =
3845                                sizeof (struct mode_control_scsi3) -
3846                                sizeof (struct mode_page);
3847                            mctl->d_sense = 1;
3848                    }
3849                    break;
3850            }
3851            bcopy(&mode, bp->b_un.b_addr,
3852                    (bp->b_bcount < sizeof(mode)) ? bp->b_bcount : sizeof(mode));
3853    }

3855    /*
3856     * Start/Stop unit (Power Management)
3857     */
3858    static void
3859    aac_startstop(struct aac_softstate *softs, union scsi_cdb *cdbp, uint32_t cid)
3860    {
3861            ddi_acc_handle_t acc = softs->sync_slot->fib_acc_handle;
3862            struct aac_fib *fibp = softs->sync_slot->fibp;
3863            struct aac_Container *cmd;
3864            struct aac_Container_resp *resp;
3865            uint32_t resp_status;
3866            int rval;

3868            /* Get adapter config status */
3869            cmd = (struct aac_Container *)&fibp->data[0];

3871            bzero(cmd, sizeof (*cmd) - CT_PACKET_SIZE);
3872            ddi_put32(acc, &cmd->Command, VM_ContainerConfig);
3873            ddi_put32(acc, &cmd->CTCommand.command, CT_PM_DRIVER_SUPPORT);
```

```
3874         ddi_put32(acc, &cmd->CTCommand.param[0], cdbp->cdb_opaque[4] & 1 ?
3875                 AAC_PM_DRIVERSUP_START_UNIT : AAC_PM_DRIVERSUP_STOP_UNIT);
3876         ddi_put32(acc, &cmd->CTCommand.param[1], cid);
3877         ddi_put32(acc, &cmd->CTCommand.param[2], cdbp->cdb_opaque[1] & 1);

3879         rval = aac_sync_fib(softs, ContainerCommand,
3880                 AAC_FIB_SIZEOF(struct aac_Container));

3882         resp = (struct aac_Container_resp *)cmd;
3883         resp_status = ddi_get32(acc, &resp->Status);
3884         if (rval != AACOK || resp_status != 0)
3885                 AACDB_PRINT(softs, CE_WARN, "Cannot start/stop a unit");
4119 finish:
4120         /* copyout the valid data. */
4121         bcopy(sense_data, bp->b_un.b_addr, min(sdata_size, bp->b_bcount));
4122         kmem_free(sense_data, sdata_size);
3886 }
_____unchanged_portion_omitted_

3906 /*ARGSUSED*/
3907 static int
3908 aac_tran_tgt_init(dev_info_t *hba_dip, dev_info_t *tgt_dip,
3909     scsi_hba_tran_t *tran, struct scsi_device *sd)
3910 {
3911         struct aac_softstate *softs = AAC_TRAN2SOFTS(tran);
3912 #if defined(AAC_DEBUG) || defined(__lock_lint)
4149 #if defined(DEBUG) || defined(__lock_lint)
3913         int ctl = ddi_get_instance(softs->devinfo_p);
3914 #endif
3915         int tgt = sd->sd_address.a_target;
3916         int lun = sd->sd_address.a_lun;
4152         uint16_t tgt = sd->sd_address.a_target;
4153         uint8_t lun = sd->sd_address.a_lun;
3917         struct aac_device *dvp;

3919         DBCALLED(softs, 2);

3921         if (ndi_dev_is_persistent_node(tgt_dip) == 0) {
3922                 (void) ndi_merge_node(tgt_dip, aac_name_node);
3923                 ddi_set_name_addr(tgt_dip, NULL);
4159                 /*
4160                  * If no persistent node exist, we don't allow .conf node
4161                  * to be created.
4162                  */
4163                 if (aac_find_child(softs, tgt, lun) != NULL) {
4164                         if (ndi_merge_node(tgt_dip, aac_name_node) !=
4165                             DDI_SUCCESS)
4166                                 /* Create this .conf node */
4167                                 return (DDI_SUCCESS);
4168                 }
3924                 return (DDI_FAILURE);
3925         }

3927         /*
3928          * Only support container/phys. device that has been
3929          * detected and valid
3930          */
3931         mutex_enter(&softs->io_lock);
3932         if (tgt < 0 || tgt >= AAC_MAX_DEV(softs)) {
4177         if (tgt >= AAC_MAX_DEV(softs)) {
3933                 AACDB_PRINT_TRAN(softs,
3934                     "aac_tran_tgt_init: c%dt%dL%d out", ctl, tgt, lun);
3935                 mutex_exit(&softs->io_lock);
3936                 return (DDI_FAILURE);
3937         }
```



```
3939         dvp = AAC_DEV(softs, tgt);
3940         if (tgt < AAC_MAX_LD) {
3941                 if (dvp == NULL || !(dvp->valid) || lun != 0) {
4185                 dvp = (struct aac_device *)&softs->containers[tgt];
4186                 if (lun != 0 || !AAC_DEV_IS_VALID(dvp)) {
3942                         AACDB_PRINT_TRAN(softs, "aac_tran_tgt_init: c%dt%dL%d",
3943                             ctl, tgt, lun);
3944                         mutex_exit(&softs->io_lock);
3945                         return (DDI_FAILURE);
3946                 }
3947                 /*
3948                  * Save the tgt_dip for the given target if one doesn't exist
3949                  * already. Dip's for non-existance tgt's will be cleared in
3950                  * tgt_free.
3951                  */
3952                 if (softs->containers[tgt].dev.dip == NULL &&
3953                     strcmp(ddi_driver_name(sd->sd_dev), "sd") == 0)
3954                         softs->containers[tgt].dev.dip = tgt_dip;
4200         } else {
4201                 dvp = (struct aac_device *)&softs->nondasds[AAC_PD(tgt)];
4202                 /*
4203                  * Save the tgt_dip for the given target if one doesn't exist
4204                  * already. Dip's for non-existance tgt's will be cleared in
4205                  * tgt_free.
4206                  */

4208                 if (softs->nondasds[AAC_PD(tgt)].dev.dip  == NULL &&
4209                     strcmp(ddi_driver_name(sd->sd_dev), "sd") == 0)
4210                         softs->nondasds[AAC_PD(tgt)].dev.dip  = tgt_dip;
3955         }

4213         if (softs->flags & AAC_FLAGS_BRKUP) {
4214                 if (ndi_prop_update_int(DDI_DEV_T_NONE, tgt_dip,
4215                     "buf_break", 1) != DDI_PROP_SUCCESS) {
4216                         cmn_err(CE_CONT, "unable to create "
4217                             "property for t%dL%d (buf_break)", tgt, lun);
4218                 }
4219         }

3957         AACDB_PRINT(softs, CE_NOTE,
3958             "aac_tran_tgt_init: c%dt%dL%d ok (%s)", ctl, tgt, lun,
3959             (dvp->type == AAC_DEV_PD) ? "pd" : "ld");
3960         mutex_exit(&softs->io_lock);
3961         return (DDI_SUCCESS);
3962 }

3964 static void
3965 aac_tran_tgt_free(dev_info_t *hba_dip, dev_info_t *tgt_dip,
3966     scsi_hba_tran_t *hba_tran, struct scsi_device *sd)
3967 {
3968 #ifndef __lock_lint
3969         _NOTE(ARGUNUSED(hba_dip, tgt_dip, hba_tran))
3970 #endif

3972         struct aac_softstate *softs = SD2AAC(sd);
3973         int tgt = sd->sd_address.a_target;

3975         mutex_enter(&softs->io_lock);
3976         if (tgt < AAC_MAX_LD)
4240         if (tgt < AAC_MAX_LD) {
4241                 if (softs->containers[tgt].dev.dip == tgt_dip)
3977                 softs->containers[tgt].dev.dip = NULL;
3978         else
3979                 softs->nondasds[AAC_PD(tgt)].dev.valid = 0;
4243         } else {
4244                 if (softs->nondasds[AAC_PD(tgt)].dev.dip == tgt_dip)
```

```
4245                        softs->nondasds[AAC_PD(tgt)].dev.dip = NULL;
4246                softs->nondasds[AAC_PD(tgt)].dev.flags &= ~AAC_DFLAG_VALID;
4247        }
3980        mutex_exit(&softs->io_lock);
3981 }

3983 /*
3984  * Check if the firmware is Up And Running. If it is in the Kernel Panic
3985  * state, (BlinkLED code + 1) is returned.
3986  *    0 -- firmware up and running
3987  *   -1 -- firmware dead
3988  *   >0 -- firmware kernel panic
3989  */
3990 static int
3991 aac_check_adapter_health(struct aac_softstate *softs)
3992 {
3993        int rval;

3995        rval = AAC_FWSTATUS_GET(softs);
4263        rval = PCI_MEM_GET32(softs, AAC_OMR0);

3997        if (rval & AAC_KERNEL_UP_AND_RUNNING) {
3998                rval = 0;
3999        } else if (rval & AAC_KERNEL_PANIC) {
4268                cmn_err(CE_WARN, "firmware panic");
4000                rval = ((rval >> 16) & 0xff) + 1; /* avoid 0 as return value */
4001                cmn_err(CE_WARN, "!Adapter firmware crashed with BLED %d", rval
4002        } else {
4271                cmn_err(CE_WARN, "firmware dead");
4003                rval = -1;
4004                cmn_err(CE_WARN, "!Adapter firmware dead");
4005        }
4006        return (rval);
4007 }

4009 static void
4010 aac_abort_iocmd(struct aac_softstate *softs, struct aac_cmd *acp,
4011     uchar_t reason)
4012 {
4013        acp->flags |= AAC_CMD_ABORT;

4015        if (acp->pkt) {
4016                /*
4017                 * Each lun should generate a unit attention
4018                 * condition when reset.
4019                 * Phys. drives are treated as logical ones
4020                 * during error recovery.
4021                 */
4022                if (softs->flags & AAC_STATE_RESET)
4023                        aac_set_arq_data_reset(softs, acp);
4284                if (acp->slotp) { /* outstanding cmd */
4285                        acp->pkt->pkt_state |= STATE_GOT_STATUS;
4286                }

4025                switch (reason) {
4026                case CMD_TIMEOUT:
4290                        AACDB_PRINT(softs, CE_NOTE, "CMD_TIMEOUT: acp=0x%p",
4291                            acp);
4027                        aac_set_pkt_reason(softs, acp, CMD_TIMEOUT,
4028                            STAT_TIMEOUT | STAT_BUS_RESET);
4029                        break;
4030                case CMD_RESET:
4031                        /* aac support only RESET_ALL */
4297                        AACDB_PRINT(softs, CE_NOTE, "CMD_RESET: acp=0x%p", acp);
4032                        aac_set_pkt_reason(softs, acp, CMD_RESET,
4033                            STAT_BUS_RESET);
```



```
4034                        break;
4035                case CMD_ABORTED:
4302                        AACDB_PRINT(softs, CE_NOTE, "CMD_ABORTED: acp=0x%p",
4303                            acp);
4036                        aac_set_pkt_reason(softs, acp, CMD_ABORTED,
4037                            STAT_ABORTED);
4038                        break;
4039                }
4040        }
4041        aac_end_io(softs, acp);
4042 }

4044 /*
4045  * Abort all the pending commands of type iocmd or just the command pkt
4046  * corresponding to pkt
4047  */
4048 static void
4049 aac_abort_iocmds(struct aac_softstate *softs, int iocmd, struct scsi_pkt *pkt,
4050     int reason)
4051 {
4052        struct aac_cmd *ac_arg, *acp;
4053        int i;

4055        if (pkt == NULL) {
4056                ac_arg = NULL;
4057        } else {
4058                ac_arg = PKT2AC(pkt);
4059                iocmd = (ac_arg->flags & AAC_CMD_SYNC) ?
4060                    AAC_IOCMD_SYNC : AAC_IOCMD_ASYNC;
4061        }

4063        /*
4064         * a) outstanding commands on the controller
4065         * Note: should abort outstanding commands only after one
4066         * IOP reset has been done.
4067         */
4068        if (iocmd & AAC_IOCMD_OUTSTANDING) {
4069                struct aac_cmd *acp;

4071                for (i = 0; i < AAC_MAX_LD; i++) {
4072                        if (softs->containers[i].dev.valid)
4340                        if (AAC_DEV_IS_VALID(&softs->containers[i].dev))
4073                                softs->containers[i].reset = 1;
4074                }
4075                while ((acp = softs->q_busy.q_head) != NULL)
4076                        aac_abort_iocmd(softs, acp, reason);
4077        }

4079        /* b) commands in the waiting queues */
4080        for (i = 0; i < AAC_CMDQ_NUM; i++) {
4081                if (iocmd & (1 << i)) {
4082                        if (ac_arg) {
4083                                aac_abort_iocmd(softs, ac_arg, reason);
4084                        } else {
4085                                while ((acp = softs->q_wait[i].q_head) != NULL)
4086                                        aac_abort_iocmd(softs, acp, reason);
4087                        }
4088                }
4089        }
4090 }

4092 /*
4093  * The draining thread is shared among quiesce threads. It terminates
4094  * when the adapter is quiesced or stopped by aac_stop_drain().
4095  */
4096 static void
```

```
4097 aac_check_drain(void *arg)
4098 {
4099         struct aac_softstate *softs = arg;

4101         mutex_enter(&softs->io_lock);
4102         if (softs->ndrains) {
4371                 softs->drain_timeid = 0;
4103                 /*
4104                  * If both ASYNC and SYNC bus throttle are held,
4105                  * wake up threads only when both are drained out.
4106                  */
4107                 if ((softs->bus_throttle[AAC_CMDQ_ASYNC] > 0 ||
4108                     softs->bus_ncmds[AAC_CMDQ_ASYNC] == 0) &&
4109                     (softs->bus_throttle[AAC_CMDQ_SYNC] > 0 ||
4110                     softs->bus_ncmds[AAC_CMDQ_SYNC] == 0))
4111                         cv_broadcast(&softs->drain_cv);
4112                 else
4113                         softs->drain_timeid = timeout(aac_check_drain, softs,
4114                             AAC_QUIESCE_TICK * drv_usectohz(1000000));
4115         }
4116         mutex_exit(&softs->io_lock);
4117 }

4119 /*
4120  * If not draining the outstanding cmds, drain them. Otherwise,
4121  * only update ndrains.
4122  */
4123 static void
4124 aac_start_drain(struct aac_softstate *softs)
4125 {
4126         if (softs->ndrains == 0) {
4396                 ASSERT(softs->drain_timeid == 0);
4127                 softs->drain_timeid = timeout(aac_check_drain, softs,
4128                     AAC_QUIESCE_TICK * drv_usectohz(1000000));
4129         }
4130         softs->ndrains++;
4131 }
_____unchanged_portion_omitted_

4153 /*
4154  * The following function comes from Adaptec:
4155  *
4156  * Once do an IOP reset, basically the driver have to re-initialize the card
4157  * as if up from a cold boot, and the driver is responsible for any IO that
4158  * is outstanding to the adapter at the time of the IOP RESET. And prepare
4159  * for IOP RESET by making the init code modular with the ability to call it
4160  * from multiple places.
4161  */
4162 static int
4163 aac_reset_adapter(struct aac_softstate *softs)
4164 {
4165         ddi_acc_handle_t acc = softs->comm_space_acc_handle;
4166         int health;
4167         uint32_t status, wait_count, reset_mask;
4168         struct aac_fib *fibp;
4169         struct aac_pause_command *pc;
4170         int rval = AACERR;
4436         uint32_t status;
4437         int rval = AAC_IOP_RESET_FAILED;

4172         DBCALLED(softs, 1);

4173         ASSERT(softs->state & AAC_STATE_RESET);

4175         aac_fm_acc_err_clear(softs->pci_mem_handle[0], DDI_FME_VER0);
4443         ddi_fm_acc_err_clear(softs->pci_mem_handle, DDI_FME_VER0);
```



```
4176         /* Disable interrupt */
4177         AAC_SET_INTR(softs, 0);
4445         AAC_DISABLE_INTR(softs);

4179         health = aac_check_adapter_health(softs);
4180         AACDB_PRINT(softs, CE_NOTE,
4181                 "aac_reset_adapter() called, health %d", health);
4182         if (health == -1) {
4183                 aac_fm_service_impact(softs->devinfo_p, DDI_SERVICE_LOST);
4449                 ddi_fm_service_impact(softs->devinfo_p, DDI_SERVICE_LOST);
4184                 goto finish;
4185         }
4186         if (health == 0) /* flush drives if possible */
4187                 (void) aac_shutdown(softs);

4189         /* Execute IOP reset */
4190         if (softs->aac_support_opt2 & AAC_SUPPORTED_MU_RESET) {
4456         if ((aac_sync_mbcommand(softs, AAC_IOP_RESET, 0, 0, 0, 0,
4457             &status)) != AACOK) {
4458                 ddi_acc_handle_t acc;
4459                 struct aac_fib *fibp;
4460                 struct aac_pause_command *pc;

4462                 if ((status & 0xf) == 0xf) {
4463                         uint32_t wait_count;

4191                 /*
4192                  * Sunrise Lake has dual cores and we must drag the
4193                  * other core with us to reset simultaneously. There
4194                  * are 2 bits in the Inbound Reset Control and Status
4195                  * Register (offset 0x38) of the Sunrise Lake to reset
4196                  * the chip without clearing out the PCI configuration
4197                  * info (COMMAND & BARS).
4198                  */
4199                 PCI_MEM_PUT32(softs, 0, AAC_IRCSR, AAC_IRCSR_CORES_RST);
4473                         PCI_MEM_PUT32(softs, AAC_IRCSR, AAC_IRCSR_CORES_RST);

4201                 /*
4202                  * We need to wait for 5 seconds before accessing the MU
4203                  * again 10000 * 100us = 1000,000us = 1000ms = 1s
4204                  */
4205                 wait_count = 5 * 10000;
4206                 while (wait_count) {
4207                         drv_usecwait(100); /* delay 100 microseconds */
4208                         wait_count--;
4209                 }
4210         } else {
4211                 softs->sync_slot_busy = 1;
4212                 if ((aac_sync_mbcommand(softs, AAC_IOP_RESET_ALWAYS, 0, 0, 0, 0,
4213                         &status, &reset_mask)) != AACOK) {
4214                         if (status == SRB_STATUS_INVALID_REQUEST) {
4485                         if (status == SRB_STATUS_INVALID_REQUEST)
4215                                 cmn_err(CE_WARN, "!IOP_RESET not supported");
4216                                 AACDB_PRINT(softs, CE_NOTE,
4217                                         "IOP_RESET not supported");
4218                         } else {                /* probably timeout */
4487                         else /* probably timeout */
4219                                 cmn_err(CE_WARN, "!IOP_RESET failed");
4220                                 AACDB_PRINT(softs, CE_NOTE,
4221                                         "IOP_RESET failed");
4222                         }

4223                         /* Unwind aac_shutdown() */
4224                         fibp = softs->sync_slot->fibp;
4491                         (void) aac_sync_fib_slot_bind(softs, &softs->sync_ac);
4492                         acc = softs->sync_ac.slotp->fib_acc_handle;
```

```
4494                        fibp = softs->sync_ac.slotp->fibp;
4225                        pc = (struct aac_pause_command *)&fibp->data[0];

4227                        bzero(pc, sizeof (*pc));
4228                        ddi_put32(acc, &pc->Command, VM_ContainerConfig);
4229                        ddi_put32(acc, &pc->Type, CT_PAUSE_IO);
4230                        ddi_put32(acc, &pc->Timeout, 1);
4231                        ddi_put32(acc, &pc->Min, 1);
4232                        ddi_put32(acc, &pc->NoRescan, 1);

4234                        (void) aac_sync_fib(softs, ContainerCommand,
4235                                AAC_FIB_SIZEOF(struct aac_pause_command));
4506                        aac_sync_fib_slot_release(softs, &softs->sync_ac);

4237                        aac_fm_service_impact(softs->devinfo_p,
4508                        if (aac_check_adapter_health(softs) != 0)
4509                                ddi_fm_service_impact(softs->devinfo_p,
4238                                DDI_SERVICE_LOST);
4239                        if (health == 0)
4240                                rval = AACOK2;
4241                        goto finish;
4242                } else if (softs->aac_support_opt2 & AAC_SUPPORTED_DOORBELL_RESE
4243                        PCI_MEM_PUT32(softs, 0, AAC_SRC_IDBR, reset_mask);
4511                        else
4244                        /*
4245                         * We need to wait for 5 seconds before accessing the do
4246                         * again 10000 * 100us = 1000,000us = 1000ms = 1s
4513                                * IOP reset not supported or IOP not reseted
4247                         */
4248                        wait_count = 5 * 10000;
4249                        while (wait_count) {
4250                                drv_usecwait(100); /* delay 100 microseconds */
4251                                wait_count--;
4515                                rval = AAC_IOP_RESET_ABNORMAL;
4516                        goto finish;
4252                        }
4253                }
4254        }
4255        cmn_err(CE_NOTE, "!IOP_RESET finished successfully");
4256        AACDB_PRINT(softs, CE_NOTE, "IOP_RESET finished successfully");

4258        /*
4259         * Re-read and renegotiate the FIB parameters, as one of the actions
4260         * that can result from an IOP reset is the running of a new firmware
4261         * image.
4262         */
4263        if (aac_common_attach(softs) != AACOK)
4264                goto finish;

4266        rval = AACOK;
4528        rval = AAC_IOP_RESET_SUCCEED;

4268 finish:
4269        AAC_SET_INTR(softs, 1);
4531        AAC_ENABLE_INTR(softs);
4270        return (rval);
4271 }
______unchanged_portion_omitted_

4305 static void
4306 aac_unhold_bus(struct aac_softstate *softs, int iocmds)
4307 {
4308        int i, q;
4570        int i, q, max_throttle;
```



```
4310        for (q = 0; q < AAC_CMDQ_NUM; q++) {
4311                if (iocmds & (1 << q)) {
4312                        /*
4313                         * Should not unhold AAC_IOCMD_ASYNC bus, if it has been
4314                         * quiesced or being drained by possibly some quiesce
4315                         * threads.
4316                         */
4317                        if (q == AAC_CMDQ_ASYNC && ((softs->state &
4318                            AAC_STATE_QUIESCED) || softs->ndrains))
4319                                continue;
4320                        softs->bus_throttle[q] = softs->total_slots;
4582                        if (q == AAC_CMDQ_ASYNC)
4583                                max_throttle = softs->total_slots -
4584                                    AAC_MGT_SLOT_NUM;
4585                        else
4586                                max_throttle = softs->total_slots - 1;
4587                        softs->bus_throttle[q] = max_throttle;
4321                        for (i = 0; i < AAC_MAX_LD; i++)
4322                                aac_set_throttle(softs,
4323                                    &softs->containers[i].dev,
4324                                    q, softs->total_slots);
4591                                    q, max_throttle);
4325                        for (i = 0; i < AAC_MAX_PD(softs); i++)
4326                                aac_set_throttle(softs, &softs->nondasds[i].dev,
4327                                    q, softs->total_slots);
4594                                    q, max_throttle);
4328                }
4329        }
4330 }

4332 static int
4333 aac_do_reset(struct aac_softstate *softs)
4334 {
4335        int health, rval;
4336        int sync_cmds, async_cmds;
4602        int health;
4603        int rval;

4338        softs->state |= AAC_STATE_RESET;
4339        health = aac_check_adapter_health(softs);
4340        AACDB_PRINT(softs, CE_NOTE, "aac_do_reset() called, health %d", health);

4342        /*
4343         * Hold off new io commands and wait all outstanding io
4344         * commands to complete.
4345         */
4346        sync_cmds = softs->bus_ncmds[AAC_CMDQ_SYNC];
4347        async_cmds = softs->bus_ncmds[AAC_CMDQ_ASYNC];
4348        if (health == 0 && (sync_cmds || async_cmds)) {
4612        if (health == 0) {
4613                int sync_cmds = softs->bus_ncmds[AAC_CMDQ_SYNC];
4614                int async_cmds = softs->bus_ncmds[AAC_CMDQ_ASYNC];

4616                if (sync_cmds == 0 && async_cmds == 0) {
4617                        rval = AAC_IOP_RESET_SUCCEED;
4618                        goto finish;
4619                }
4349                /*
4350                 * Give the adapter up to AAC_QUIESCE_TIMEOUT more seconds
4351                 * to complete the outstanding io commands
4352                 */
4353                int timeout = AAC_QUIESCE_TIMEOUT * 1000 * 10;
4354                int (*intr_handler)(struct aac_softstate *);

4356                aac_hold_bus(softs, AAC_IOCMD_SYNC | AAC_IOCMD_ASYNC);
4357                /*
```

```
4358                  * Poll the adapter by ourselves in case interrupt is disabled
4359                  * and to avoid releasing the io_lock.
4360                  */
4361                 if ((softs->flags & AAC_FLAGS_NEW_COMM_TYPE1) ||
4362                         (softs->flags & AAC_FLAGS_NEW_COMM_TYPE2) ||
4363                         (softs->flags & AAC_FLAGS_NEW_COMM_TYPE34))
4364                         intr_handler = aac_process_intr_new_type1;
4365                 else if (softs->flags & AAC_FLAGS_NEW_COMM)
4366                         intr_handler = aac_process_intr_new;
4367                 else
4368                         intr_handler = aac_process_intr_old;
4632                 intr_handler = (softs->flags & AAC_FLAGS_NEW_COMM) ?
4633                     aac_process_intr_new : aac_process_intr_old;
4369                 while ((softs->bus_ncmds[AAC_CMDQ_SYNC] ||
4370                     softs->bus_ncmds[AAC_CMDQ_ASYNC]) && timeout) {
4371                         drv_usecwait(100);
4372                         (void) intr_handler(softs);
4373                         timeout--;
4374                 }
4375                 aac_unhold_bus(softs, AAC_IOCMD_SYNC | AAC_IOCMD_ASYNC);

4642                 if (softs->bus_ncmds[AAC_CMDQ_SYNC] == 0 &&
4643                     softs->bus_ncmds[AAC_CMDQ_ASYNC] == 0) {
4644                         /* Cmds drained out */
4645                         rval = AAC_IOP_RESET_SUCCEED;
4646                         goto finish;
4647                 } else if (softs->bus_ncmds[AAC_CMDQ_SYNC] < sync_cmds ||
4648                     softs->bus_ncmds[AAC_CMDQ_ASYNC] < async_cmds) {
4649                         /* Cmds not drained out, adapter overloaded */
4650                         rval = AAC_IOP_RESET_ABNORMAL;
4651                         goto finish;
4376         }
4653         }

4378         /*
4379          * If a longer waiting time still can't drain all outstanding io
4656          * If a longer waiting time still can't drain any outstanding io
4380          * commands, do IOP reset.
4381          */
4382         if ((sync_cmds || async_cmds) &&
4383                 (softs->bus_ncmds[AAC_CMDQ_SYNC] == sync_cmds) &&
4384                 (softs->bus_ncmds[AAC_CMDQ_ASYNC] == async_cmds)) {
4385                 if ((rval = aac_reset_adapter(softs)) == AACERR)
4659         if ((rval = aac_reset_adapter(softs)) == AAC_IOP_RESET_FAILED)
4386                         softs->state |= AAC_STATE_DEAD;
4387         } else {
4388                 rval = AACOK2;
4389         }

4662 finish:
4391         softs->state &= ~AAC_STATE_RESET;
4392         return (rval);
4393 }

4395 static int
4396 aac_tran_reset(struct scsi_address *ap, int level)
4397 {
4398         struct aac_softstate *softs = AAC_TRAN2SOFTS(ap->a_hba_tran);
4399         int rval, rval2;
4671         int rval;

4401         DBCALLED(softs, 1);
4402         AACDB_PRINT(softs, CE_NOTE, "aac_tran_reset() called, level %d", level);

4404         if (level != RESET_ALL) {
4405                 cmn_err(CE_NOTE, "!reset target/lun not supported");
```



```
4406                 return (0);
4407         }

4409         mutex_enter(&softs->io_lock);
4410         rval2 = aac_do_reset(softs);
4411         rval = (rval2 != AACERR) ? 1 : 0;
4412         if (rval == 1 && !ddi_in_panic()) {
4413                 if (rval2 == AACOK)
4681         switch (rval = aac_do_reset(softs)) {
4682         case AAC_IOP_RESET_SUCCEED:
4414                         aac_abort_iocmds(softs, AAC_IOCMD_OUTSTANDING | AAC_IOCM
4415                             NULL, CMD_RESET);
4416                 aac_start_waiting_io(softs);
4417         } else {
4418                 /* Abort IOCTL cmds when system panic or adapter dead */
4686                 break;
4687         case AAC_IOP_RESET_FAILED:
4688                 /* Abort IOCTL cmds when adapter is dead */
4419                 aac_abort_iocmds(softs, AAC_IOCMD_ALL, NULL, CMD_RESET);
4690                 break;
4691         case AAC_IOP_RESET_ABNORMAL:
4692                 aac_start_waiting_io(softs);
4420         }
4421         mutex_exit(&softs->io_lock);

4423         aac_drain_comp_q(softs);
4424         return (rval);
4697         return (rval == 0);
4425 }

4427 static int
4428 aac_tran_abort(struct scsi_address *ap, struct scsi_pkt *pkt)
4429 {
4430         struct aac_softstate *softs = AAC_TRAN2SOFTS(ap->a_hba_tran);

4432         DBCALLED(softs, 1);
4433         AACDB_PRINT(softs, CE_NOTE, "aac_tran_abort() called, pkt %p", pkt);

4435         mutex_enter(&softs->io_lock);
4436         aac_abort_iocmds(softs, 0, pkt, CMD_ABORTED);
4437         mutex_exit(&softs->io_lock);

4439         aac_drain_comp_q(softs);
4440         return (1);
4441 }

4443 void
4444 aac_free_dmamap(struct aac_cmd *acp)
4445 {
4446         uint_t i;
4447         uint32_t offset;

4449         /* Free dma mapping */
4450         if (acp->flags & AAC_CMD_DMA_VALID) {
4451                 ASSERT(acp->segments[0].buf_dma_handle);
4452                 for (i = 0; i < acp->segment_cnt; ++i)
4453                         (void) ddi_dma_unbind_handle(acp->segments[i].buf_dma_ha
4720                 ASSERT(acp->buf_dma_handle);
4721                 (void) ddi_dma_unbind_handle(acp->buf_dma_handle);
4454                 acp->flags &= ~AAC_CMD_DMA_VALID;
4455         }

4457         if (acp->segments[0].abp != NULL) { /* free non-aligned buf DMA */
4458                 ASSERT(acp->segments[0].buf_dma_handle);
4459                 for (i = 0, offset = 0; i < acp->segment_cnt; ++i) {
4725         if (acp->abp != NULL) { /* free non-aligned buf DMA */
```

```
4726                    ASSERT(acp->buf_dma_handle);
4460                    if ((acp->flags & AAC_CMD_BUF_WRITE) == 0 && acp->bp)
4461                            ddi_rep_get8(acp->segments[i].abh,
4462                                    (uint8_t *)acp->bp->b_un.b_addr + offset
4463                                    (uint8_t *)acp->segments[i].abp,
4464                                    acp->segments[i].abp_size,
4728                    ddi_rep_get8(acp->abh, (uint8_t *)acp->bp->b_un.b_addr,
4729                        (uint8_t *)acp->abp, acp->bp->b_bcount,
4465                                    DDI_DEV_AUTOINCR);
4466                    offset += acp->segments[i].abp_size;
4467                    ddi_dma_mem_free(&acp->segments[i].abh);
4468                    acp->segments[i].abp = NULL;
4731                ddi_dma_mem_free(&acp->abh);
4732                acp->abp = NULL;
4469                }
4470        }

4472        for (i = 0; i < acp->segment_cnt; ++i) {
4473                if (acp->segments[i].buf_dma_handle) {
4474                        ddi_dma_free_handle(&acp->segments[i].buf_dma_handle);
4475                        acp->segments[i].buf_dma_handle = NULL;
4735        if (acp->buf_dma_handle) {
4736                ddi_dma_free_handle(&acp->buf_dma_handle);
4737                acp->buf_dma_handle = NULL;
4476                }
4477        }
4478 }

4480 static void
4481 aac_unknown_scmd(struct aac_softstate *softs, struct aac_cmd *acp)
4482 {
4483         AACDB_PRINT(softs, CE_CONT, "SCMD 0x%x not supported",
4484             ((union scsi_cdb *)acp->pkt->pkt_cdbp)->scc_cmd);
4745             ((union scsi_cdb *)(void *)acp->pkt->pkt_cdbp)->scc_cmd);
4485         aac_free_dmamap(acp);
4486         aac_set_arq_data(acp->pkt, KEY_ILLEGAL_REQUEST, 0x20, 0x00, 0);
4487         aac_soft_callback(softs, acp);
4488 }

4490 /*
4491  * Handle command to logical device
4492  */
4493 static int
4494 aac_tran_start_ld(struct aac_softstate *softs, struct aac_cmd *acp)
4495 {
4496         struct aac_container *dvp;
4497         struct scsi_pkt *pkt;
4498         union scsi_cdb *cdbp;
4499         struct buf *bp;
4500         int rval;

4502         dvp = (struct aac_container *)acp->dvp;
4503         pkt = acp->pkt;
4504         cdbp = (union scsi_cdb *)pkt->pkt_cdbp;
4765         cdbp = (void *)pkt->pkt_cdbp;
4505         bp = acp->bp;

4507         switch (cdbp->scc_cmd) {
4508         case SCMD_INQUIRY: /* inquiry */
4509                 aac_free_dmamap(acp);
4510                 aac_inquiry(softs, pkt, cdbp, bp);
4511                 aac_soft_callback(softs, acp);
4512                 rval = TRAN_ACCEPT;
4513                 break;

4515         case SCMD_READ_CAPACITY: /* read capacity */
```



```
4516                 if (bp && bp->b_un.b_addr && bp->b_bcount) {
4517                         struct scsi_capacity cap;
4518                         uint64_t last_lba;

4520                         /* check 64-bit LBA */
4521                         last_lba = dvp->size - 1;
4522                         if (last_lba > 0xffffffffull) {
4523                                 cap.capacity = 0xffffffff;
4524                         } else {
4525                                 cap.capacity = BE_32(last_lba);
4526                         }
4527                         cap.lbasize = BE_32(AAC_SECTOR_SIZE);

4529                         aac_free_dmamap(acp);
4530                         if (bp->b_flags & (B_PHYS|B_PAGEIO))
4531                                 bp_mapin(bp);
4532                         bcopy(&cap, bp->b_un.b_addr,
4533                                 bp->b_bcount < 8 ? bp->b_bcount : 8);
4793                         bcopy(&cap, bp->b_un.b_addr, min(bp->b_bcount, 8));
4534                         pkt->pkt_state |= STATE_XFERRED_DATA;
4535                 }
4536                 aac_soft_callback(softs, acp);
4537                 rval = TRAN_ACCEPT;
4538                 break;

4540         case SCMD_SVC_ACTION_IN_G4: /* read capacity 16 */
4541                 /* Check if containers need 64-bit LBA support */
4542                 if (cdbp->cdb_opaque[1] == SSVC_ACTION_READ_CAPACITY_G4) {
4543                         if (bp && bp->b_un.b_addr && bp->b_bcount) {
4544                                 struct scsi_capacity_16 cap16;
4545                                 int cap_len = sizeof (struct scsi_capacity_16);

4547                                 bzero(&cap16, cap_len);
4548                                 cap16.sc_capacity = BE_64(dvp->size);
4808                                 cap16.sc_capacity = BE_64(dvp->size - 1);
4549                                 cap16.sc_lbasize = BE_32(AAC_SECTOR_SIZE);

4551                                 aac_free_dmamap(acp);
4552                                 if (bp->b_flags & (B_PHYS | B_PAGEIO))
4553                                         bp_mapin(bp);
4554                                 bcopy(&cap16, bp->b_un.b_addr,
4555                                     bp->b_bcount < cap_len ? bp->b_bcount :
4815                                         min(bp->b_bcount, cap_len));
4556                                 pkt->pkt_state |= STATE_XFERRED_DATA;
4557                         }
4558                         aac_soft_callback(softs, acp);
4559                 } else {
4560                         aac_unknown_scmd(softs, acp);
4561                 }
4562                 rval = TRAN_ACCEPT;
4563                 break;

4565         case SCMD_READ_G4: /* read_16 */
4566         case SCMD_WRITE_G4: /* write_16 */
4567                 if (softs->flags & AAC_FLAGS_RAW_IO) {
4568                         /* NOTE: GETG4ADDRTL(cdbp) is int32_t */
4569                         acp->blkno = ((uint64_t) \
4570                             GETG4ADDR(cdbp) << 32) | \
4571                             (uint32_t)GETG4ADDRTL(cdbp);
4572                         goto do_io;
4573                 }
4574                 AACDB_PRINT(softs, CE_WARN, "64-bit LBA not supported");
4575                 aac_unknown_scmd(softs, acp);
4576                 rval = TRAN_ACCEPT;
4577                 break;
```

```
4579         case SCMD_READ: /* read_6 */
4580         case SCMD_WRITE: /* write_6 */
4581                 acp->blkno = GETG0ADDR(cdbp);
4582                 goto do_io;

4844         case SCMD_READ_G5: /* read_12 */
4845         case SCMD_WRITE_G5: /* write_12 */
4846                 acp->blkno = GETG5ADDR(cdbp);
4847                 goto do_io;

4584         case SCMD_READ_G1: /* read_10 */
4585         case SCMD_WRITE_G1: /* write_10 */
4586                 acp->blkno = (uint32_t)GETG1ADDR(cdbp);
4587 do_io:
4588                 if (acp->flags & AAC_CMD_DMA_VALID) {
4589                         uint64_t cnt_size = dvp->size;

4591                         /*
4592                          * If LBA > array size AND rawio, the
4593                          * adapter may hang. So check it before
4594                          * sending.
4595                          * NOTE: (blkno + blkcnt) may overflow
4596                          */
4597                         if ((acp->blkno < cnt_size) &&
4598                             ((acp->blkno + acp->bcount /
4599                             AAC_BLK_SIZE) <= cnt_size)) {
4600                                 rval = aac_do_io(softs, acp);
4601                         } else {
4602                         /*
4603                          * Request exceeds the capacity of disk,
4604                          * set error block number to last LBA
4605                          * + 1.
4606                          */
4607                                 aac_set_arq_data(pkt,
4608                                     KEY_ILLEGAL_REQUEST, 0x21,
4609                                     0x00, cnt_size);
4610                                 aac_soft_callback(softs, acp);
4611                                 rval = TRAN_ACCEPT;
4612                         }
4613                 } else if (acp->bcount == 0) {
4614                         /* For 0 length IO, just return ok */
4615                         aac_soft_callback(softs, acp);
4616                         rval = TRAN_ACCEPT;
4617                 } else {
4618                         rval = TRAN_BADPKT;
4619                 }
4620                 break;

4622         case SCMD_MODE_SENSE: /* mode_sense_6 */
4623         case SCMD_MODE_SENSE_G1: { /* mode_sense_10 */
4624                 int capacity;

4626                 aac_free_dmamap(acp);
4627                 if (dvp->size > 0xfffffffful)
4628                         capacity = 0xffffffffu; /* 64-bit LBA */
4629                 else
4630                         capacity = dvp->size;
4631                 aac_mode_sense(softs, pkt, cdbp, bp, capacity);
4632                 aac_soft_callback(softs, acp);
4633                 rval = TRAN_ACCEPT;
4634                 break;
4635         }

4902         case SCMD_START_STOP:
4903                 if (softs->support_opt2 & AAC_SUPPORTED_POWER_MANAGEMENT) {
4904                         acp->aac_cmd_fib = aac_cmd_fib_startstop;
```



```
4905                         acp->ac_comp = aac_startstop_complete;
4906                         rval = aac_do_io(softs, acp);
4907                         break;
4908                 }
4909         /* FALLTHRU */
4637         case SCMD_TEST_UNIT_READY:
4638         case SCMD_REQUEST_SENSE:
4639         case SCMD_FORMAT:
4640                 aac_free_dmamap(acp);
4641                 if (bp && bp->b_un.b_addr && bp->b_bcount) {
4642                         if (acp->flags & AAC_CMD_BUF_READ) {
4643                                 if (bp->b_flags & (B_PHYS|B_PAGEIO))
4644                                         bp_mapin(bp);
4645                                 bzero(bp->b_un.b_addr, bp->b_bcount);
4646                         }
4647                         pkt->pkt_state |= STATE_XFERRED_DATA;
4648                 }
4649                 aac_soft_callback(softs, acp);
4650                 rval = TRAN_ACCEPT;
4651                 break;

4653         case SCMD_SYNCHRONIZE_CACHE:
4654                 acp->flags |= AAC_CMD_NTAG;
4655                 acp->aac_cmd_fib = aac_cmd_fib_sync;
4656                 acp->ac_comp = aac_synccache_complete;
4657                 rval = aac_do_io(softs, acp);
4658                 break;

4660         case SCMD_DOORLOCK:
4661                 aac_free_dmamap(acp);
4662                 dvp->locked = (pkt->pkt_cdbp[4] & 0x01) ? 1 : 0;
4663                 aac_soft_callback(softs, acp);
4664                 rval = TRAN_ACCEPT;
4665                 break;

4667         case SCMD_START_STOP:
4668                 aac_free_dmamap(acp);
4669                 if (softs->aac_support_opt2 & AAC_SUPPORTED_POWER_MANAGEMENT)
4670                         aac_startstop(softs, cdbp, dvp->cid);
4671                 aac_soft_callback(softs, acp);
4672                 rval = TRAN_ACCEPT;
4673                 break;

4675         default: /* unknown command */
4676                 aac_unknown_scmd(softs, acp);
4677                 rval = TRAN_ACCEPT;
4678                 break;
4679         }

4681         return (rval);
4682 }

4684 static int
4685 aac_tran_start(struct scsi_address *ap, struct scsi_pkt *pkt)
4686 {
4687         struct aac_softstate *softs = AAC_TRAN2SOFTS(ap->a_hba_tran);
4688         struct aac_cmd *acp = PKT2AC(pkt);
4689         struct aac_device *dvp = acp->dvp;
4690         int rval;

4692         DBCALLED(softs, 2);

4694         /*
4695          * Reinitialize some fields of ac and pkt; the packet may
4696          * have been resubmitted
4697          */
```

```
4698         acp->flags &= AAC_CMD_CONSISTENT | AAC_CMD_DMA_PARTIAL | \
4699             AAC_CMD_BUF_READ | AAC_CMD_BUF_WRITE | AAC_CMD_DMA_VALID;
4700         acp->timeout = acp->pkt->pkt_time;
4701         if (pkt->pkt_flags & FLAG_NOINTR)
4702                 acp->flags |= AAC_CMD_NO_INTR;
4703         acp->cur_segment = 0;
4968 #ifdef DEBUG
4969         acp->fib_flags = AACDB_FLAGS_FIB_SCMD;
4970 #endif
4704         pkt->pkt_reason = CMD_CMPLT;
4705         pkt->pkt_state = 0;
4706         pkt->pkt_statistics = 0;
4707         *pkt->pkt_scbp = 0; /* clear arq scsi_status */
4974         *pkt->pkt_scbp = STATUS_GOOD; /* clear arq scsi_status */

4709         if (acp->flags & AAC_CMD_DMA_VALID) {
4710                 pkt->pkt_resid = acp->bcount;
4711                 /* Consistent packets need to be sync'ed first */
4712                 if ((acp->flags & AAC_CMD_CONSISTENT) &&
4713                     (acp->flags & AAC_CMD_BUF_WRITE))
4714                         if (aac_dma_sync_ac(acp) != AACOK) {
4715                                 aac_fm_service_impact(softs->devinfo_p,
4982                                 ddi_fm_service_impact(softs->devinfo_p,
4716                                     DDI_SERVICE_UNAFFECTED);
4717                                 return (TRAN_BADPKT);
4718                         }
4719         } else {
4720                 pkt->pkt_resid = 0;
4721         }

4723         mutex_enter(&softs->io_lock);
4724         AACDB_PRINT_SCMD(softs, acp);
4725         if (dvp->valid && ap->a_lun == 0 && !(softs->state & AAC_STATE_DEAD)) {
4726                 if (dvp->type == AAC_DEV_LD)
4992         if ((dvp->flags & (AAC_DFLAG_VALID | AAC_DFLAG_CONFIGURING)) &&
4993             !(softs->state & AAC_STATE_DEAD)) {
4994                 if (dvp->type == AAC_DEV_LD) {
4995                         if (ap->a_lun == 0)
4727                         rval = aac_tran_start_ld(softs, acp);
4728                 else if (acp->segment_cnt > 1)
4729                         rval = TRAN_BADPKT;
4730                 else
4998                                 goto error;
4999                 } else {
4731                         rval = aac_do_io(softs, acp);
5001                 }
4732         } else {
5003 error:
5004 #ifdef DEBUG
5005                 if (!(softs->state & AAC_STATE_DEAD)) {
5006                         AACDB_PRINT_TRAN(softs,
5007                             "Cannot send cmd to target t%dL%d: %s",
5008                             ap->a_target, ap->a_lun,
5009                             "target invalid");
5010                 } else {
4733                 AACDB_PRINT(softs, CE_WARN,
4734                     "Cannot send cmd to target t%dL%d: %s",
4735                     ap->a_target, ap->a_lun,
4736                     (softs->state & AAC_STATE_DEAD) ?
4737                     "adapter dead" : "target invalid");
5014                             "adapter dead");
5015                 }
5016 #endif
4738                 rval = TRAN_FATAL_ERROR;
4739         }
4740         mutex_exit(&softs->io_lock);
```



```
4741         return (rval);
4742 }

4744 static int
4745 aac_tran_getcap(struct scsi_address *ap, char *cap, int whom)
4746 {
4747         struct aac_softstate *softs = AAC_TRAN2SOFTS(ap->a_hba_tran);
4748         struct aac_device *dvp;
4749         int rval;

4751         DBCALLED(softs, 2);

4753         /* We don't allow inquiring about capabilities for other targets */
4754         if (cap == NULL || whom == 0) {
4755                 AACDB_PRINT(softs, CE_WARN,
4756                     "GetCap> %s not supported: whom=%d", cap, whom);
4757                 return (-1);
4758         }

4760         mutex_enter(&softs->io_lock);
4761         dvp = AAC_DEV(softs, ap->a_target);
4762         if (dvp == NULL || !dvp->valid) {
5041         if (dvp == NULL || !AAC_DEV_IS_VALID(dvp)) {
4763                 mutex_exit(&softs->io_lock);
4764                 AACDB_PRINT(softs, CE_WARN, "Bad target t%dL%d to getcap",
5043                 AACDB_PRINT_TRAN(softs, "Bad target t%dL%d to getcap",
4765                     ap->a_target, ap->a_lun);
4766                 return (-1);
4767         }

4769         switch (scsi_hba_lookup_capstr(cap)) {
4770         case SCSI_CAP_ARQ: /* auto request sense */
4771                 rval = 1;
4772                 break;
4773         case SCSI_CAP_UNTAGGED_QING:
4774         case SCSI_CAP_TAGGED_QING:
4775                 rval = 1;
4776                 break;
4777         case SCSI_CAP_DMA_MAX:
4778                 rval = softs->buf_dma_attr.dma_attr_maxxfer;
5057                 rval = softs->dma_max;
4779                 break;
4780         default:
4781                 rval = -1;
4782                 break;
4783         }
4784         mutex_exit(&softs->io_lock);

4786         AACDB_PRINT_TRAN(softs, "GetCap> %s t%dL%d: rval=%d",
4787             cap, ap->a_target, ap->a_lun, rval);
4788         return (rval);
4789 }

4791 /*ARGSUSED*/
4792 static int
4793 aac_tran_setcap(struct scsi_address *ap, char *cap, int value, int whom)
4794 {
4795         struct aac_softstate *softs = AAC_TRAN2SOFTS(ap->a_hba_tran);
4796         struct aac_device *dvp;
4797         int rval;

4799         DBCALLED(softs, 2);

4801         /* We don't allow inquiring about capabilities for other targets */
4802         if (cap == NULL || whom == 0) {
4803                 AACDB_PRINT(softs, CE_WARN,
```

```
4804                    "SetCap> %s not supported: whom=%d", cap, whom);
4805                return (-1);
4806        }

4808        mutex_enter(&softs->io_lock);
4809        dvp = AAC_DEV(softs, ap->a_target);
4810        if (dvp == NULL || !dvp->valid) {
5089        if (dvp == NULL || !AAC_DEV_IS_VALID(dvp)) {
4811                mutex_exit(&softs->io_lock);
4812                AACDB_PRINT(softs, CE_WARN, "Bad target t%dL%d to setcap",
5091                AACDB_PRINT_TRAN(softs, "Bad target t%dL%d to setcap",
4813                    ap->a_target, ap->a_lun);
4814                return (-1);
4815        }

4817        switch (scsi_hba_lookup_capstr(cap)) {
4818        case SCSI_CAP_ARQ:
4819                /* Force auto request sense */
4820                rval = (value == 1) ? 1 : 0;
4821                break;
4822        case SCSI_CAP_UNTAGGED_QING:
4823        case SCSI_CAP_TAGGED_QING:
4824                rval = (value == 1) ? 1 : 0;
4825                break;
4826        default:
4827                rval = -1;
4828                break;
4829        }
4830        mutex_exit(&softs->io_lock);

4832        AACDB_PRINT_TRAN(softs, "SetCap> %s t%dL%d val=%d: rval=%d",
4833            cap, ap->a_target, ap->a_lun, value, rval);
4834        return (rval);
4835 }
______unchanged_portion_omitted_

4853 int
4854 aac_cmd_dma_alloc(struct aac_softstate *softs, struct aac_cmd *acp,
4855     struct buf *bp, int flags, int (*cb)(), caddr_t arg)
4856 {
4857        int kf = (cb == SLEEP_FUNC) ? KM_SLEEP : KM_NOSLEEP;
4858        uint_t oldcookiec;
4859        int bioerr;
4860        int rval;

4862        oldcookiec = acp->left_cookien;

4864        /* Move window to build s/g map */
4865        if (acp->total_nwin > 0) {
4866                if (++acp->cur_win < acp->total_nwin) {
4867                        off_t off;
4868                        size_t len;

4870                        rval = ddi_dma_getwin(acp->segments[0].buf_dma_handle,
4871                                acp->cur_win, &off, &len,
4872                                &acp->segments[0].cookie, &acp->segments[0].left
4873                        acp->segment_cnt = 1;
4874                        acp->left_cookien = acp->segments[0].left_cookien;
5149                        rval = ddi_dma_getwin(acp->buf_dma_handle, acp->cur_win,
5150                            &off, &len, &acp->cookie, &acp->left_cookien);
4875                        if (rval == DDI_SUCCESS)
4876                                goto get_dma_cookies;
4877                        AACDB_PRINT(softs, CE_WARN,
4878                            "ddi_dma_getwin() fail %d", rval);
4879                        return (AACERR);
4880                } else {
```



```
5156                }
4881                        AACDB_PRINT(softs, CE_WARN, "Nothing to transfer");
4882                        return (AACERR);
4883                }
4884        }

4886        /* We need to transfer data, so we alloc DMA resources for this pkt */
4887        if (bp && bp->b_bcount != 0 && !(acp->flags & AAC_CMD_DMA_VALID)) {
4888                uint_t dma_flags = 0;
4889                struct aac_sge *sge;
4890                int repeat = 0;
4891                uint_t i, j;
4892                uint32_t offset;

4894                /*
4895                 * We will still use this point to fake some
4896                 * infomation in tran_start
4897                 */
4898                acp->bp = bp;

4900                /* Set dma flags */
4901                if (BUF_IS_READ(bp)) {
4902                        dma_flags |= DDI_DMA_READ;
4903                        acp->flags |= AAC_CMD_BUF_READ;
4904                } else {
4905                        dma_flags |= DDI_DMA_WRITE;
4906                        acp->flags |= AAC_CMD_BUF_WRITE;
4907                }
4908                if (flags & PKT_CONSISTENT)
4909                        dma_flags |= DDI_DMA_CONSISTENT;
4910                if (flags & PKT_DMA_PARTIAL)
4911                        dma_flags |= DDI_DMA_PARTIAL;

4913                do {
4914                        /* Bind buf */
4915                        if (((uintptr_t)bp->b_un.b_addr & AAC_DMA_ALIGN_MASK) ==
4916                                && !repeat) {
4917                                /* Alloc buf dma handle */
4918                                if (!acp->segments[0].buf_dma_handle) {
5186                if (!acp->buf_dma_handle) {
4919                                        rval = ddi_dma_alloc_handle(softs->devin
4920                                                &softs->buf_dma_attr, cb, arg,
4921                                                &acp->segments[0].buf_dma_handle
5189                                            &acp->buf_dma_handle);
4922                                        if (rval != DDI_SUCCESS) {
4923                                                AACDB_PRINT(softs, CE_WARN,
4924                                                        "Can't allocate DMA hand
4925                                                        rval);
4926                                                goto error_out;
4927                                        }
4928                                }
4929                                rval = ddi_dma_buf_bind_handle(acp->segments[0].
4930                                        bp, dma_flags, cb, arg, &acp->segments[0
4931                                        &acp->segments[0].left_cookien);
4932                                acp->segment_cnt = 1;
4933                                acp->left_cookien = acp->segments[0].left_cookie

5198                /* Bind buf */
5199                if (((uintptr_t)bp->b_un.b_addr & AAC_DMA_ALIGN_MASK) == 0) {
5200                        rval = ddi_dma_buf_bind_handle(acp->buf_dma_handle,
5201                            bp, dma_flags, cb, arg, &acp->cookie,
5202                            &acp->left_cookien);
4934                        } else {
5204                        size_t bufsz;

4935                                AACDB_PRINT_TRAN(softs,
```

```
4936                                    "non-aligned buffer: addr=0x%p, cnt=%lu"
4937                                    (void *)bp->b_un.b_addr, bp->b_bcount);
4938                            if (bp->b_flags & (B_PAGEIO|B_PHYS))
4939                                    bp_mapin(bp);

4941                            if (repeat) {
4942                                    repeat = 0;
4943                                    acp->segment_cnt = (bp->b_bcount - 1) /
4944                                        softs->buf_dma_attr.dma_attr_max
4945                                    if (acp->segment_cnt > AAC_MAXSEGMENTS)
4946                                            AACDB_PRINT(softs, CE_WARN,
4947                                                    "After DDI_DMA_TOOBIG: T
4948                                            acp->segment_cnt = AAC_MAXSEGMEN
4949                                            goto error_out;
4950                                    }

4952                                    for (i = 0; i < acp->segment_cnt; ++i) {
4953                                            rval = ddi_dma_alloc_handle(soft
4954                                                    &softs->buf_dma_attr, cb
4955                                                    &acp->segments[i].buf_dm
4956                                            if (rval != DDI_SUCCESS) {
4957                                                    AACDB_PRINT(softs, CE_WA
4958                                                            "After DDI_DMA_T
4959                                                            rval);
4960                                                    goto error_out;
4961                                            }
4962                                            if ((i == acp->segment_cnt - 1)
4963                                                    (bp->b_bcount % softs->b
4964                                                    acp->segments[i].abp_siz
4965                                                            softs->buf_dma_a
4966                                            else
4967                                                    acp->segments[i].abp_siz
4968                                                            softs->buf_dma_a
4969                                            rval = ddi_dma_mem_alloc(acp->se
4970                                                    acp->segments[i].abp_siz
4971                                                    &aac_acc_attr, DDI_DMA_S
4972                                                    cb, arg, &acp->segments[
4973                                                    &acp->segments[i].abp_re
4974                                                    &acp->segments[i].abh);

4976                                            if (rval != DDI_SUCCESS) {
4977                                                    AACDB_PRINT(softs, CE_NO
4978                                                            "Cannot alloc DM
4979                                                    bioerr = 0;
4980                                                    goto error_out;
4981                                            }
4982                                    }
4983                            } else {
4984                                    /* Alloc buf dma handle */
4985                                    if (!acp->segments[0].buf_dma_handle) {
4986                                            rval = ddi_dma_alloc_handle(soft
4987                                                    &softs->buf_dma_attr, cb
4988                                                    &acp->segments[0].buf_dm
4989                                            if (rval != DDI_SUCCESS) {
4990                                                    AACDB_PRINT(softs, CE_WA
4991                                                            "Can't allocate
4992                                                            rval);
4993                                                    goto error_out;
4994                                            }
4995                                    }
4996                                    rval = ddi_dma_mem_alloc(acp->segments[0
5212                        rval = ddi_dma_mem_alloc(acp->buf_dma_handle,
4997                                            AAC_ROUNDUP(bp->b_bcount, AAC_DM
4998                                            &aac_acc_attr, DDI_DMA_STREAMING
4999                                            cb, arg, &acp->segments[0].abp,
5000                                            &acp->segments[0].abp_real_size,
```



```
5001                                            &acp->segments[0].abh);
5214                            &softs->acc_attr, DDI_DMA_STREAMING,
5215                            cb, arg, &acp->abp, &bufsz, &acp->abh);

5003                                    if (rval != DDI_SUCCESS) {
5004                                            AACDB_PRINT(softs, CE_NOTE,
5005                                                    "Cannot alloc DMA to non
5006                                            bioerr = 0;
5007                                            goto error_out;
5008                                    }
5009                                    acp->segment_cnt = 1;
5010                                    acp->segments[0].abp_size = bp->b_bcount
5011                            }

5013                            acp->left_cookien = 0;
5014                            for (i = 0, offset = 0; i < acp->segment_cnt; ++
5015                                    if (acp->flags & AAC_CMD_BUF_WRITE) {
5016                                            ddi_rep_put8(acp->segments[i].ab
5017                                                    (uint8_t *)bp->b_un.b_ad
5018                                                    (uint8_t *)acp->segments
5019                                                    acp->segments[i].abp_siz
5224                    if (acp->flags & AAC_CMD_BUF_WRITE)
5225                            ddi_rep_put8(acp->abh,
5226                                (uint8_t *)bp->b_un.b_addr,
5227                                (uint8_t *)acp->abp, bp->b_bcount,
5020                                                    DDI_DEV_AUTOINCR);
5021                                            offset += acp->segments[i].abp_s
5022                                    }

5024                                    rval = ddi_dma_addr_bind_handle(acp->seg
5025                                            NULL, acp->segments[i].abp,
5026                                            acp->segments[i].abp_real_size,
5027                                            &acp->segments[i].cookie, &acp->
5028                                    if (rval != DDI_DMA_MAPPED)
5029                                            break;

5031                                    acp->left_cookien += acp->segments[i].le
5230                    rval = ddi_dma_addr_bind_handle(acp->buf_dma_handle,
5231                        NULL, acp->abp, bufsz, dma_flags, cb, arg,
5232                        &acp->cookie, &acp->left_cookien);
5032                            }
5033                    }

5035                    switch (rval) {
5036                    case DDI_DMA_PARTIAL_MAP:
5037                            if (ddi_dma_numwin(acp->segments[0].buf_dma_hand
5237                    if (ddi_dma_numwin(acp->buf_dma_handle,
5038                                    &acp->total_nwin) == DDI_FAILURE) {
5039                                    AACDB_PRINT(softs, CE_WARN,
5040                                            "Cannot get number of DMA window
5041                                    bioerr = 0;
5042                                    goto error_out;
5043                            }
5044                            AACDB_PRINT_TRAN(softs, "buf bind, %d seg(s)",
5045                                    acp->left_cookien);
5046                            acp->cur_win = 0;
5047                            break;

5049                    case DDI_DMA_MAPPED:
5050                            AACDB_PRINT_TRAN(softs, "buf bind, %d seg(s)",
5051                                    acp->left_cookien);
5052                            acp->cur_win = 0;
5053                            acp->total_nwin = 1;
5054                            break;

5056                    case DDI_DMA_NORESOURCES:
```

```
5057                                    bioerr = 0;
5058                                    AACDB_PRINT(softs, CE_WARN,
5059                                            "Cannot bind buf for DMA: DDI_DMA_NORESO
5060                                    goto error_out;
5061                            case DDI_DMA_BADATTR:
5062                            case DDI_DMA_NOMAPPING:
5063                                    bioerr = EFAULT;
5064                                    AACDB_PRINT(softs, CE_WARN,
5065                                            "Cannot bind buf for DMA: DDI_DMA_NOMAPP
5066                                    goto error_out;
5067                            case DDI_DMA_TOOBIG:
5068                                    if (bp->b_bcount > softs->buf_dma_attr.dma_attr_
5069                                            acp->segment_cnt == 1) {
5070                                            repeat = 1;
5071                                            break;
5072                                    } else {
5073                                            bioerr = EINVAL;
5074                                            AACDB_PRINT(softs, CE_WARN,
5075                                                    "Cannot bind buf for DMA: DDI_DM
5270                                        "Cannot bind buf for DMA: DDI_DMA_TOOBIG(%d)",
5076                                                    bp->b_bcount);
5077                                            goto error_out;
5078                                    }
5079                            default:
5080                                    bioerr = EINVAL;
5081                                    AACDB_PRINT(softs, CE_WARN,
5082                                            "Cannot bind buf for DMA: %d", rval);
5083                                    goto error_out;
5084                            }
5085                    } while (repeat);
5086                    acp->flags |= AAC_CMD_DMA_VALID;

5088 get_dma_cookies:
5089                    ASSERT(acp->left_cookien > 0);
5283                    if (acp->left_cookien > softs->aac_sg_tablesize) {
5284                            AACDB_PRINT(softs, CE_NOTE, "large cookiec received %d",
5285                                acp->left_cookien);
5286                            bioerr = EINVAL;
5287                            goto error_out;
5288                    }
5090                    if (oldcookiec != acp->left_cookien && acp->sgt != NULL) {
5091                            kmem_free(acp->sgt, sizeof (struct aac_sge) * \
5092                                oldcookiec);
5093                            acp->sgt = NULL;
5094                    }
5095                    if (acp->sgt == NULL) {
5096                            acp->sgt = kmem_alloc(sizeof (struct aac_sge) * \
5097                                acp->left_cookien, kf);
5098                            if (acp->sgt == NULL) {
5099                                    AACDB_PRINT(softs, CE_WARN,
5100                                        "sgt kmem_alloc fail");
5101                                    bioerr = ENOMEM;
5102                                    goto error_out;
5103                            }
5104                    }

5106                    sge = &acp->sgt[0];
5107                    acp->bcount = 0;
5108                    for (i = 0; i < acp->segment_cnt; ++i) {
5109                            acp->segments[i].sgt = sge;
5110                            if (acp->segments[i].left_cookien > softs->aac_sg_tables
5111                                    AACDB_PRINT(softs, CE_NOTE, "large cookiec recei
5112                                            acp->segments[i].left_cookien);
5113                                    bioerr = EINVAL;
5114                                    goto error_out;
5306                    sge->bcount = acp->cookie.dmac_size;
```



```
5307                    sge->addr.ad64.lo = AAC_LS32(acp->cookie.dmac_laddress);
5308                    sge->addr.ad64.hi = AAC_MS32(acp->cookie.dmac_laddress);
5309                    acp->bcount = acp->cookie.dmac_size;
5310                    for (sge++; sge < &acp->sgt[acp->left_cookien]; sge++) {
5311                            ddi_dma_nextcookie(acp->buf_dma_handle, &acp->cookie);
5312                            sge->bcount = acp->cookie.dmac_size;
5313                            sge->addr.ad64.lo = AAC_LS32(acp->cookie.dmac_laddress);
5314                            sge->addr.ad64.hi = AAC_MS32(acp->cookie.dmac_laddress);
5315                            acp->bcount += acp->cookie.dmac_size;
5115                            }
5116                            for (j = 0; j < acp->segments[i].left_cookien; ++j) {
5117                                    if (j > 0)
5118                                            ddi_dma_nextcookie(acp->segments[i].buf_
5119                                                    &acp->segments[i].cookie);
5120                                    sge->bcount = acp->segments[i].cookie.dmac_size;
5121                                    sge->addr.ad64.lo = AAC_LS32(acp->segments[i].co
5122                                    sge->addr.ad64.hi = AAC_MS32(acp->segments[i].co
5123                                    acp->bcount += acp->segments[i].cookie.dmac_size
5124                                    sge++;
5125                            }
5126                    }

5128                    /*
5129                     * Note: The old DMA engine do not correctly handle
5130                     * dma_attr_maxxfer attribute. So we have to ensure
5131                     * it by ourself.
5132                     */
5133                    if (acp->segment_cnt == 1 &&
5134                            acp->bcount > softs->buf_dma_attr.dma_attr_maxxfer) {
5323                    if (acp->bcount > softs->buf_dma_attr.dma_attr_maxxfer) {
5135                            AACDB_PRINT(softs, CE_NOTE,
5136                                "large xfer size received %d\n", acp->bcount);
5137                            bioerr = EINVAL;
5138                            goto error_out;
5139                    }

5141                    acp->total_xfer += acp->bcount;

5142                    if (acp->pkt) {
5143                            /* Return remaining byte count */
5144                            acp->pkt->pkt_resid = bp->b_bcount - acp->total_xfer;

5334                            if (acp->total_xfer <= bp->b_bcount) {
5335                                    acp->pkt->pkt_resid = bp->b_bcount - \
5336                                        acp->total_xfer;
5337                            } else {
5338                                    /*
5339                                     * Allocated DMA size is greater than the buf
5340                                     * size of bp. This is caused by devices like
5341                                     * tape. we have extra bytes allocated, but
5342                                     * the packet residual has to stay correct.
5343                                     */
5344                                    acp->pkt->pkt_resid = 0;
5345                            }
5146                            AACDB_PRINT_TRAN(softs,
5147                                "bp=0x%p, xfered=%d/%d, resid=%d",
5148                                (void *)bp->b_un.b_addr, (int)acp->total_xfer,
5149                                (int)bp->b_bcount, (int)acp->pkt->pkt_resid);

5151                            ASSERT(acp->pkt->pkt_resid >= 0);
5152                    }
5153            }
5154            return (AACOK);

5156 error_out:
5157            bioerror(bp, bioerr);
```

```
5158         return (AACERR);
5159 }
_____unchanged_portion_omitted_

5211 /*
5212  * tran_sync_pkt(9E) - explicit DMA synchronization
5213  */
5214 /*ARGSUSED*/
5215 static void
5216 aac_tran_sync_pkt(struct scsi_address *ap, struct scsi_pkt *pkt)
5217 {
5218         struct aac_cmd *acp = PKT2AC(pkt);

5220         DBCALLED(NULL, 2);

5222         if (aac_dma_sync_ac(acp) != AACOK)
5223                 aac_fm_service_impact(
5421                 ddi_fm_service_impact(
5224                     (AAC_TRAN2SOFTS(ap->a_hba_tran))->devinfo_p,
5225                     DDI_SERVICE_UNAFFECTED);
5226 }
_____unchanged_portion_omitted_

5309 static int
5310 aac_hba_setup(struct aac_softstate *softs)
5311 {
5312         scsi_hba_tran_t *hba_tran;
5313         int rval;

5315         hba_tran = scsi_hba_tran_alloc(softs->devinfo_p, SCSI_HBA_CANSLEEP);
5316         if (hba_tran == NULL)
5317                 return (AACERR);
5318         hba_tran->tran_hba_private = softs;
5319         hba_tran->tran_tgt_init = aac_tran_tgt_init;
5320         hba_tran->tran_tgt_free = aac_tran_tgt_free;
5321         hba_tran->tran_tgt_probe = scsi_hba_probe;
5322         hba_tran->tran_start = aac_tran_start;
5323         hba_tran->tran_getcap = aac_tran_getcap;
5324         hba_tran->tran_setcap = aac_tran_setcap;
5325         hba_tran->tran_init_pkt = aac_tran_init_pkt;
5326         hba_tran->tran_destroy_pkt = aac_tran_destroy_pkt;
5327         hba_tran->tran_reset = aac_tran_reset;
5328         hba_tran->tran_abort = aac_tran_abort;
5329         hba_tran->tran_sync_pkt = aac_tran_sync_pkt;
5330         hba_tran->tran_dmafree = aac_tran_dmafree;
5331         hba_tran->tran_quiesce = aac_tran_quiesce;
5332         hba_tran->tran_unquiesce = aac_tran_unquiesce;
5333         hba_tran->tran_bus_config = aac_tran_bus_config;
5334 #if 0
5335         hba_tran->tran_interconnect_type = INTERCONNECT_PARALLEL;
5336 #endif
5337         rval = scsi_hba_attach_setup(softs->devinfo_p, &softs->buf_dma_attr,
5338             hba_tran, 0);
5339         if (rval != DDI_SUCCESS) {
5340                 scsi_hba_tran_free(hba_tran);
5341                 AACDB_PRINT(softs, CE_WARN, "aac_hba_setup failed");
5342                 return (AACERR);
5343         }

5345         softs->hba_tran = hba_tran;
5346         return (AACOK);
5347 }

5349 /*
5350  * FIB setup operations
5351  */
```



```
5353 /*
5354  * Init FIB header
5355  */
5356 static void
5357 aac_cmd_fib_header(struct aac_softstate *softs, struct aac_slot *slotp,
5358     uint16_t cmd, uint16_t fib_size)
5552 aac_cmd_fib_header(struct aac_softstate *softs, struct aac_cmd *acp,
5553     uint16_t cmd)
5359 {
5555         struct aac_slot *slotp = acp->slotp;
5360         ddi_acc_handle_t acc = slotp->fib_acc_handle;
5361         struct aac_fib *fibp = slotp->fibp;
5362         uint32_t xfer_state;

5364         xfer_state =
5365             AAC_FIBSTATE_HOSTOWNED |
5366             AAC_FIBSTATE_INITIALISED |
5367             AAC_FIBSTATE_EMPTY |
5564             AAC_FIBSTATE_FAST_RESPONSE | /* enable fast io */
5368             AAC_FIBSTATE_FROMHOST |
5369             AAC_FIBSTATE_REXPECTED |
5370             AAC_FIBSTATE_NORM;
5371         if (slotp->acp && !(slotp->acp->flags & AAC_CMD_SYNC)) {
5372                 xfer_state |=
5373                     AAC_FIBSTATE_ASYNC |
5374                     AAC_FIBSTATE_FAST_RESPONSE /* enable fast io */;
5375                 ddi_put16(acc, &fibp->Header.SenderSize,
5376                     softs->aac_max_fib_size);
5377         } else {
5378                 ddi_put16(acc, &fibp->Header.SenderSize, AAC_FIB_SIZE);
5379         }

5569         if (!(acp->flags & AAC_CMD_SYNC))
5570                 xfer_state |= AAC_FIBSTATE_ASYNC;

5381         ddi_put32(acc, &fibp->Header.XferState, xfer_state);
5382         ddi_put16(acc, &fibp->Header.Command, cmd);
5383         ddi_put8(acc, &fibp->Header.StructType, AAC_FIBTYPE_TFIB);
5384         ddi_put8(acc, &fibp->Header.Unused, 0); /* don't care */
5385         ddi_put16(acc, &fibp->Header.Size, fib_size);
5575         ddi_put8(acc, &fibp->Header.Flags, 0); /* don't care */
5576         ddi_put16(acc, &fibp->Header.Size, acp->fib_size);
5577         ddi_put16(acc, &fibp->Header.SenderSize, softs->aac_max_fib_size);
5386         ddi_put32(acc, &fibp->Header.SenderFibAddress, (slotp->index << 2));
5387         ddi_put32(acc, &fibp->Header.a.ReceiverFibAddress, slotp->fib_phyaddr);
5388         ddi_put32(acc, &fibp->Header.Handle, slotp->index + 1);
5579         ddi_put32(acc, &fibp->Header.ReceiverFibAddress, slotp->fib_phyaddr);
5580         ddi_put32(acc, &fibp->Header.SenderData, 0); /* don't care */
5389 }

5391 /*
5392  * Init FIB for raw IO2 command
5393  */
5394 static void
5395 aac_cmd_fib_rawio2(struct aac_softstate *softs, struct aac_cmd *acp)
5396 {
5397         ddi_acc_handle_t acc = acp->slotp->fib_acc_handle;
5398         struct aac_raw_io2 *io = (struct aac_raw_io2 *)&acp->slotp->fibp->data[0
5399         struct aac_sge_ieee1212 *sgp;
5400         struct aac_sge *sge = &acp->sgt[0];
5401         uint_t i, left_cookien, conformable;
5402         uint32_t min_size, cur_size, nom_size;
5403         uint16_t flags;

5405         if (acp->segment_cnt > 1) {
```

```
5406                uint_t idx = acp->cur_segment;
5407                left_cookien = acp->segments[idx].left_cookien;
5408                acp->bcount = acp->segments[idx].abp_size;
5409                sge = acp->segments[idx].sgt;
5410                if (idx > 0)
5411                        acp->blkno += acp->segments[idx-1].abp_size / AAC_BLK_SI
5412        } else
5413                left_cookien = acp->left_cookien;

5415        ddi_put32(acc, &io->strtBlkLow, AAC_LS32(acp->blkno));
5416        ddi_put32(acc, &io->strtBlkHigh, AAC_MS32(acp->blkno));
5417        ddi_put32(acc, &io->byteCnt, acp->bcount);
5418        ddi_put16(acc, &io->ldNum,
5419            ((struct aac_container *)acp->dvp)->cid);
5420        flags = (acp->flags & AAC_CMD_BUF_READ) ?
5421                RIO2_IO_TYPE_READ : RIO2_IO_TYPE_WRITE;
5422        flags |= RIO2_SG_FORMAT_IEEE1212;

5424        /* Fill SG table */
5425        for (i = 0, sgp = &io->sge[0]; i < left_cookien; i++, sge++, sgp++) {
5426                ddi_put32(acc, &sgp->addrLow, sge->addr.ad64.lo);
5427                ddi_put32(acc, &sgp->addrHigh, sge->addr.ad64.hi);
5428                ddi_put32(acc, &sgp->length, sge->bcount);
5429                ddi_put32(acc, &sgp->flags, 0L);
5430                cur_size = sge->bcount;
5431                if (i == 0) {
5432                        conformable = 1;
5433                        ddi_put32(acc, &io->sgeFirstSize, cur_size);
5434                } else if (i == 1) {
5435                        nom_size = cur_size;
5436                        ddi_put32(acc, &io->sgeNominalSize, nom_size);
5437                        min_size = cur_size;
5438                } else if ((i+1) < left_cookien && cur_size != nom_size) {
5439                        conformable = 0;
5440                        if (cur_size < min_size)
5441                                min_size = cur_size;
5442                }
5443        }

5445        /* not conformable: evaluate required sg elements */
5446        if (!conformable) {
5447                uint_t j, err_found, left_cookien_new = left_cookien;
5448                for (i = min_size / AAC_PAGE_SIZE; i >= 1; --i) {
5449                        err_found = 0;
5450                        left_cookien_new = 2;
5451                        for (j = 1; j < left_cookien - 1; ++j) {
5452                                if (ddi_get32(acc, &io->sge[j].length) % (i*AAC_
5453                                        err_found = 1;
5454                                        break;
5455                                }
5456                                left_cookien_new += (ddi_get32(acc, &io->sge[j].
5457                        }
5458                        if (!err_found)
5459                                break;
5460                }
5461                if (i > 0 && left_cookien_new <= softs->aac_sg_tablesize &&
5462                        !softs->no_sgl_conv)
5463                        left_cookien = aac_convert_sgraw2(softs,
5464                                acp, i, left_cookien, left_cookien_new, &flags);
5465        } else {
5466                flags |= RIO2_SGL_CONFORMANT;
5467        }
5468        ddi_put16(acc, &io->flags, flags);
5469        ddi_put8(acc, &io->sgeCnt, left_cookien);

5471        /* Calculate FIB size */
```



```
5472        acp->fib_size = sizeof (struct aac_fib_header) + \
5473            sizeof (struct aac_raw_io2) + (left_cookien - 1) * \
5474            sizeof (struct aac_sge_ieee1212);

5476        aac_cmd_fib_header(softs, acp->slotp, RawIo2, acp->fib_size);
5477 }


5480 static uint_t
5481 aac_convert_sgraw2(struct aac_softstate *softs, struct aac_cmd *acp,
5482                                    uint_t pages, uint_t left_cookien, uint_t lef
5483                                    uint16_t *flags)
5484 {
5485        ddi_acc_handle_t acc = acp->slotp->fib_acc_handle;
5486        struct aac_raw_io2 *io = (struct aac_raw_io2 *)&acp->slotp->fibp->data[0
5487        struct aac_sge_ieee1212 *sge;
5488        uint_t i, j, pos;
5489        uint32_t addr_low_new, addr_low;

5491        sge = kmem_alloc(left_cookien_new * sizeof(struct aac_sge_ieee1212),
5492                                        KM_NOSLEEP);
5493        if (sge == NULL)
5494                return left_cookien;

5496        for (i = 1, pos = 1; i < left_cookien-1; ++i) {
5497                for (j = 0; j < ddi_get32(acc, &io->sge[i].length) / (pages * AA
5498                        addr_low = ddi_get32(acc, &io->sge[i].addrLow);
5499                        addr_low_new = addr_low + j * pages * AAC_PAGE_SIZE;
5500                        sge[pos].addrLow = addr_low_new;
5501                        sge[pos].addrHigh = ddi_get32(acc, &io->sge[i].addrHigh)
5502                        if (addr_low_new < addr_low)
5503                                sge[pos].addrHigh++;
5504                        sge[pos].length = pages * AAC_PAGE_SIZE;
5505                        sge[pos].flags = 0;
5506                        pos++;
5507                }
5508        }
5509        sge[pos].addrLow = ddi_get32(acc, &io->sge[left_cookien-1].addrLow);
5510        sge[pos].addrHigh = ddi_get32(acc, &io->sge[left_cookien-1].addrHigh);
5511        sge[pos].length = ddi_get32(acc, &io->sge[left_cookien-1].length);
5512        sge[pos].flags = 0;
5513        ddi_rep_put8(acc, (uint8_t *)&sge[1],
5514                (uint8_t *)&io->sge[1], (left_cookien_new-1) *
5515                sizeof(struct aac_sge_ieee1212), DDI_DEV_AUTOINCR);

5517        kmem_free(sge, left_cookien_new * sizeof(struct aac_sge_ieee1212));
5518        *flags |= RIO2_SGL_CONFORMANT;
5519        ddi_put32(acc, &io->sgeNominalSize, pages * AAC_PAGE_SIZE);
5520        return left_cookien_new;
5521 }

5523 /*
5524  * Init FIB for raw IO command
5525  */
5526 static void
5527 aac_cmd_fib_rawio(struct aac_softstate *softs, struct aac_cmd *acp)
5528 {
5529        ddi_acc_handle_t acc = acp->slotp->fib_acc_handle;
5530        struct aac_raw_io *io = (struct aac_raw_io *)&acp->slotp->fibp->data[0];
5531        struct aac_sg_entryraw *sgp;
5532        struct aac_sge *sge = &acp->sgt[0];
5533        uint_t i, left_cookien;
5592        struct aac_sge *sge;

5535        if (acp->segment_cnt > 1) {
5536                uint_t idx = acp->cur_segment;
```

```
5537                left_cookien = acp->segments[idx].left_cookien;
5538                acp->bcount = acp->segments[idx].abp_size;
5539                sge = acp->segments[idx].sgt;
5540                if (idx > 0)
5541                        acp->blkno += acp->segments[idx-1].abp_size / AAC_BLK_SI
5542        } else
5543                left_cookien = acp->left_cookien;

5545        /* Calculate FIB size */
5546        acp->fib_size = sizeof (struct aac_fib_header) + \
5547            sizeof (struct aac_raw_io) + (left_cookien - 1) * \
5596            sizeof (struct aac_raw_io) + (acp->left_cookien - 1) * \
5548            sizeof (struct aac_sg_entryraw);

5550        aac_cmd_fib_header(softs, acp->slotp, RawIo, acp->fib_size);
5599        aac_cmd_fib_header(softs, acp, RawIo);

5552        ddi_put16(acc, &io->Flags, (acp->flags & AAC_CMD_BUF_READ) ? 1 : 0);
5553        ddi_put16(acc, &io->BpTotal, 0);
5554        ddi_put16(acc, &io->BpComplete, 0);

5556        ddi_put32(acc, AAC_LO32(&io->BlockNumber), AAC_LS32(acp->blkno));
5557        ddi_put32(acc, AAC_HI32(&io->BlockNumber), AAC_MS32(acp->blkno));
5558        ddi_put16(acc, &io->ContainerId,
5559            ((struct aac_container *)acp->dvp)->cid);

5561        /* Fill SG table */
5562        ddi_put32(acc, &io->SgMapRaw.SgCount, left_cookien);
5611        ddi_put32(acc, &io->SgMapRaw.SgCount, acp->left_cookien);
5563        ddi_put32(acc, &io->ByteCount, acp->bcount);

5565        for (i = 0, sgp = &io->SgMapRaw.SgEntryRaw[0];
5566                i < left_cookien; i++, sge++, sgp++) {
5614        for (sge = &acp->sgt[0], sgp = &io->SgMapRaw.SgEntryRaw[0];
5615            sge < &acp->sgt[acp->left_cookien]; sge++, sgp++) {
5567                ddi_put32(acc, AAC_LO32(&sgp->SgAddress), sge->addr.ad64.lo);
5568                ddi_put32(acc, AAC_HI32(&sgp->SgAddress), sge->addr.ad64.hi);
5569                ddi_put32(acc, &sgp->SgByteCount, sge->bcount);
5570                sgp->Next = 0;
5571                sgp->Prev = 0;
5572                sgp->Flags = 0;
5573        }
5574 }

5576 /* Init FIB for 64-bit block IO command */
5577 static void
5578 aac_cmd_fib_brw64(struct aac_softstate *softs, struct aac_cmd *acp)
5579 {
5580        ddi_acc_handle_t acc = acp->slotp->fib_acc_handle;
5581        struct aac_blockread64 *br = (struct aac_blockread64 *) \
5582            &acp->slotp->fibp->data[0];
5583        struct aac_sg_entry64 *sgp;
5584        struct aac_sge *sge = &acp->sgt[0];
5585        uint_t i, left_cookien;
5633        struct aac_sge *sge;

5587        if (acp->segment_cnt > 1) {
5588                uint_t idx = acp->cur_segment;
5589                left_cookien = acp->segments[idx].left_cookien;
5590                acp->bcount = acp->segments[idx].abp_size;
5591                sge = acp->segments[idx].sgt;
5592                if (idx > 0)
5593                        acp->blkno += acp->segments[idx-1].abp_size / AAC_BLK_SI
5594        } else
5595                left_cookien = acp->left_cookien;
```



```
5597        acp->fib_size = sizeof (struct aac_fib_header) + \
5598            sizeof (struct aac_blockread64) + (left_cookien - 1) * \
5636            sizeof (struct aac_blockread64) + (acp->left_cookien - 1) * \
5599            sizeof (struct aac_sg_entry64);

5601        aac_cmd_fib_header(softs, acp->slotp, ContainerCommand64,
5602            acp->fib_size);
5639        aac_cmd_fib_header(softs, acp, ContainerCommand64);

5604        /*
5605         * The definitions for aac_blockread64 and aac_blockwrite64
5606         * are the same.
5607         */
5608        ddi_put32(acc, &br->BlockNumber, (uint32_t)acp->blkno);
5609        ddi_put16(acc, &br->ContainerId,
5610            ((struct aac_container *)acp->dvp)->cid);
5611        ddi_put32(acc, &br->Command, (acp->flags & AAC_CMD_BUF_READ) ?
5612            VM_CtHostRead64 : VM_CtHostWrite64);
5613        ddi_put16(acc, &br->Pad, 0);
5614        ddi_put16(acc, &br->Flags, 0);

5616        /* Fill SG table */
5617        ddi_put32(acc, &br->SgMap64.SgCount, left_cookien);
5654        ddi_put32(acc, &br->SgMap64.SgCount, acp->left_cookien);
5618        ddi_put16(acc, &br->SectorCount, acp->bcount / AAC_BLK_SIZE);

5620        for (i = 0, sgp = &br->SgMap64.SgEntry64[0];
5621            i < left_cookien; i++, sge++, sgp++) {
5657        for (sge = &acp->sgt[0], sgp = &br->SgMap64.SgEntry64[0];
5658            sge < &acp->sgt[acp->left_cookien]; sge++, sgp++) {
5622                ddi_put32(acc, AAC_LO32(&sgp->SgAddress), sge->addr.ad64.lo);
5623                ddi_put32(acc, AAC_HI32(&sgp->SgAddress), sge->addr.ad64.hi);
5624                ddi_put32(acc, &sgp->SgByteCount, sge->bcount);
5625        }
5626 }

5628 /* Init FIB for block IO command */
5629 static void
5630 aac_cmd_fib_brw(struct aac_softstate *softs, struct aac_cmd *acp)
5631 {
5632        ddi_acc_handle_t acc = acp->slotp->fib_acc_handle;
5633        struct aac_blockread *br = (struct aac_blockread *) \
5634            &acp->slotp->fibp->data[0];
5635        struct aac_sg_entry *sgp;
5636        struct aac_sge *sge = &acp->sgt[0];
5637        uint_t i, left_cookien;

5639        if (acp->segment_cnt > 1) {
5640                uint_t idx = acp->cur_segment;
5641                left_cookien = acp->segments[idx].left_cookien;
5642                acp->bcount = acp->segments[idx].abp_size;
5643                sge = acp->segments[idx].sgt;
5644                if (idx > 0)
5645                        acp->blkno += acp->segments[idx-1].abp_size / AAC_BLK_SI
5646        } else
5647                left_cookien = acp->left_cookien;

5649        if (acp->flags & AAC_CMD_BUF_READ) {
5650                acp->fib_size = sizeof (struct aac_fib_header) + \
5651                    sizeof (struct aac_blockread) + (left_cookien - 1) * \
5677                    sizeof (struct aac_blockread) + (acp->left_cookien - 1) * \
5652                    sizeof (struct aac_sg_entry);

5654                ddi_put32(acc, &br->Command, VM_CtBlockRead);
5655                ddi_put32(acc, &br->SgMap.SgCount, left_cookien);
5681                ddi_put32(acc, &br->SgMap.SgCount, acp->left_cookien);
```

```
5656                 sgp = &br->SgMap.SgEntry[0];
5657         } else {
5658                 struct aac_blockwrite *bw = (struct aac_blockwrite *)br;

5660                 acp->fib_size = sizeof (struct aac_fib_header) + \
5661                     sizeof (struct aac_blockwrite) + (left_cookien - 1) * \
5687                     sizeof (struct aac_blockwrite) + (acp->left_cookien - 1) * \
5662                     sizeof (struct aac_sg_entry);

5664                 ddi_put32(acc, &bw->Command, VM_CtBlockWrite);
5665                 ddi_put32(acc, &bw->Stable, CUNSTABLE);
5666                 ddi_put32(acc, &bw->SgMap.SgCount, left_cookien);
5692                 ddi_put32(acc, &bw->SgMap.SgCount, acp->left_cookien);
5667                 sgp = &bw->SgMap.SgEntry[0];
5668         }
5669         aac_cmd_fib_header(softs, acp->slotp, ContainerCommand, acp->fib_size);
5695         aac_cmd_fib_header(softs, acp, ContainerCommand);

5671         /*
5672          * aac_blockread and aac_blockwrite have the similar
5673          * structure head, so use br for bw here
5674          */
5675         ddi_put32(acc, &br->BlockNumber, (uint32_t)acp->blkno);
5676         ddi_put32(acc, &br->ContainerId,
5677             ((struct aac_container *)acp->dvp)->cid);
5678         ddi_put32(acc, &br->ByteCount, acp->bcount);

5680         /* Fill SG table */
5681         for (i = 0; i < left_cookien; i++, sge++, sgp++) {
5707         for (sge = &acp->sgt[0];
5708             sge < &acp->sgt[acp->left_cookien]; sge++, sgp++) {
5682                 ddi_put32(acc, &sgp->SgAddress, sge->addr.ad32);
5683                 ddi_put32(acc, &sgp->SgByteCount, sge->bcount);
5684         }
5685 }

5687 /*ARGSUSED*/
5688 void
5689 aac_cmd_fib_copy(struct aac_softstate *softs, struct aac_cmd *acp)
5690 {
5691         struct aac_slot *slotp = acp->slotp;
5692         struct aac_fib *fibp = slotp->fibp;
5693         ddi_acc_handle_t acc = slotp->fib_acc_handle;

5695         ddi_rep_put8(acc, (uint8_t *)acp->fibp, (uint8_t *)fibp,
5696             acp->fib_size,   /* only copy data of needed length */
5697             DDI_DEV_AUTOINCR);
5698         ddi_put32(acc, &fibp->Header.a.ReceiverFibAddress,slotp->fib_phyaddr);
5725         ddi_put32(acc, &fibp->Header.ReceiverFibAddress, slotp->fib_phyaddr);
5699         ddi_put32(acc, &fibp->Header.SenderFibAddress, slotp->index << 2);
5700         ddi_put32(acc, &fibp->Header.Handle, slotp->index + 1);
5701 }

5703 static void
5704 aac_cmd_fib_sync(struct aac_softstate *softs, struct aac_cmd *acp)
5705 {
5706         struct aac_slot *slotp = acp->slotp;
5707         ddi_acc_handle_t acc = slotp->fib_acc_handle;
5732         ddi_acc_handle_t acc = acp->slotp->fib_acc_handle;
5708         struct aac_synchronize_command *sync =
5709             (struct aac_synchronize_command *)&slotp->fibp->data[0];
5734             (struct aac_synchronize_command *)&acp->slotp->fibp->data[0];

5711         acp->fib_size = sizeof (struct aac_fib_header) + \
5712             sizeof (struct aac_synchronize_command);
5736         acp->fib_size = AAC_FIB_SIZEOF(struct aac_synchronize_command);
```



```
5714         aac_cmd_fib_header(softs, slotp, ContainerCommand, acp->fib_size);
5738         aac_cmd_fib_header(softs, acp, ContainerCommand);
5715         ddi_put32(acc, &sync->Command, VM_ContainerConfig);
5716         ddi_put32(acc, &sync->Type, (uint32_t)CT_FLUSH_CACHE);
5717         ddi_put32(acc, &sync->Cid, ((struct aac_container *)acp->dvp)->cid);
5718         ddi_put32(acc, &sync->Count,
5719             sizeof (((struct aac_synchronize_reply *)0)->Data));
5720 }

5746 /*
5747  * Start/Stop unit (Power Management)
5748  */
5722 static void
5723 aac_cmd_aif_request(struct aac_softstate *softs, struct aac_cmd *acp)
5750 aac_cmd_fib_startstop(struct aac_softstate *softs, struct aac_cmd *acp)
5724 {
5725         struct aac_slot *slotp = acp->slotp;
5726         ddi_acc_handle_t acc = slotp->fib_acc_handle;
5727         struct aac_aif_command *aif =
5728             (struct aac_aif_command *)&slotp->fibp->data[0];
5752         ddi_acc_handle_t acc = acp->slotp->fib_acc_handle;
5753         struct aac_Container *cmd =
5754             (struct aac_Container *)&acp->slotp->fibp->data[0];
5755         union scsi_cdb *cdbp = (void *)acp->pkt->pkt_cdbp;

5730         acp->fib_size = sizeof(struct aac_fib);
5757         acp->fib_size = AAC_FIB_SIZEOF(struct aac_Container);

5732         aac_cmd_fib_header(softs, slotp, AifRequest, acp->fib_size);
5733         ddi_put32(acc, (uint32_t *)&aif->command, AifReqEvent);
5759         aac_cmd_fib_header(softs, acp, ContainerCommand);
5760         bzero(cmd, sizeof (*cmd) - CT_PACKET_SIZE);
5761         ddi_put32(acc, &cmd->Command, VM_ContainerConfig);
5762         ddi_put32(acc, &cmd->CTCommand.command, CT_PM_DRIVER_SUPPORT);
5763         ddi_put32(acc, &cmd->CTCommand.param[0], cdbp->cdb_opaque[4] & 1 ? \
5764             AAC_PM_DRIVERSUP_START_UNIT : AAC_PM_DRIVERSUP_STOP_UNIT);
5765         ddi_put32(acc, &cmd->CTCommand.param[1],
5766             ((struct aac_container *)acp->dvp)->cid);
5767         ddi_put32(acc, &cmd->CTCommand.param[2], cdbp->cdb_opaque[1] & 1);
5734 }

5736 /*
5737  * Init FIB for pass-through SCMD
5738  */
5739 static void
5740 aac_cmd_fib_srb(struct aac_cmd *acp)
5741 {
5742         struct aac_slot *slotp = acp->slotp;
5743         ddi_acc_handle_t acc = slotp->fib_acc_handle;
5744         struct aac_srb *srb = (struct aac_srb *)&slotp->fibp->data[0];
5776         ddi_acc_handle_t acc = acp->slotp->fib_acc_handle;
5777         struct aac_srb *srb = (struct aac_srb *)&acp->slotp->fibp->data[0];
5745         uint8_t *cdb;

5747         ddi_put32(acc, &srb->function, SRBF_ExecuteScsi);
5748         ddi_put32(acc, &srb->retry_limit, 0);
5749         ddi_put32(acc, &srb->cdb_size, acp->cmdlen);
5750         ddi_put32(acc, &srb->timeout, 0); /* use driver timeout */
5751         if (acp->fibp == NULL) {
5752                 if (acp->flags & AAC_CMD_BUF_READ)
5753                         ddi_put32(acc, &srb->flags, SRB_DataIn);
5754                 else if (acp->flags & AAC_CMD_BUF_WRITE)
5755                         ddi_put32(acc, &srb->flags, SRB_DataOut);
5756                 ddi_put32(acc, &srb->channel,
5757                     ((struct aac_nondasd *)acp->dvp)->bus);
```

```
5758                 ddi_put32(acc, &srb->id, ((struct aac_nondasd *)acp->dvp)->tid);
5759                 ddi_put32(acc, &srb->lun, 0);
5760                 cdb = acp->pkt->pkt_cdbp;
5761         } else {
5762                 struct aac_srb *srb0 = (struct aac_srb *)&acp->fibp->data[0];

5764                 ddi_put32(acc, &srb->flags, srb0->flags);
5765                 ddi_put32(acc, &srb->channel, srb0->channel);
5766                 ddi_put32(acc, &srb->id, srb0->id);
5767                 ddi_put32(acc, &srb->lun, srb0->lun);
5768                 cdb = srb0->cdb;
5769         }
5770         ddi_rep_put8(acc, cdb, srb->cdb, acp->cmdlen, DDI_DEV_AUTOINCR);
5771 }

5773 static void
5774 aac_cmd_fib_scsi32(struct aac_softstate *softs, struct aac_cmd *acp)
5775 {
5776         ddi_acc_handle_t acc = acp->slotp->fib_acc_handle;
5777         struct aac_srb *srb = (struct aac_srb *)&acp->slotp->fibp->data[0];
5778         struct aac_sg_entry *sgp;
5779         struct aac_sge *sge;

5781         acp->fib_size = sizeof (struct aac_fib_header) + \
5782             sizeof (struct aac_srb) - sizeof (struct aac_sg_entry) + \
5783             acp->left_cookien * sizeof (struct aac_sg_entry);

5785         /* Fill FIB and SRB headers, and copy cdb */
5786         aac_cmd_fib_header(softs, acp->slotp, ScsiPortCommand, acp->fib_size);
5819         aac_cmd_fib_header(softs, acp, ScsiPortCommand);
5787         aac_cmd_fib_srb(acp);

5789         /* Fill SG table */
5790         ddi_put32(acc, &srb->sg.SgCount, acp->left_cookien);
5791         ddi_put32(acc, &srb->count, acp->bcount);

5793         for (sge = &acp->sgt[0], sgp = &srb->sg.SgEntry[0];
5794             sge < &acp->sgt[acp->left_cookien]; sge++, sgp++) {
5795                 ddi_put32(acc, &sgp->SgAddress, sge->addr.ad32);
5796                 ddi_put32(acc, &sgp->SgByteCount, sge->bcount);
5797         }
5798 }

5800 static void
5801 aac_cmd_fib_scsi64(struct aac_softstate *softs, struct aac_cmd *acp)
5802 {
5803         ddi_acc_handle_t acc = acp->slotp->fib_acc_handle;
5804         struct aac_srb *srb = (struct aac_srb *)&acp->slotp->fibp->data[0];
5805         struct aac_sg_entry64 *sgp;
5806         struct aac_sge *sge;

5808         acp->fib_size = sizeof (struct aac_fib_header) + \
5809             sizeof (struct aac_srb) - sizeof (struct aac_sg_entry) + \
5810             acp->left_cookien * sizeof (struct aac_sg_entry64);

5812         /* Fill FIB and SRB headers, and copy cdb */
5813         aac_cmd_fib_header(softs, acp->slotp, ScsiPortCommandU64,
5814             acp->fib_size);
5846         aac_cmd_fib_header(softs, acp, ScsiPortCommandU64);
5815         aac_cmd_fib_srb(acp);

5817         /* Fill SG table */
5818         ddi_put32(acc, &srb->sg.SgCount, acp->left_cookien);
5819         ddi_put32(acc, &srb->count, acp->bcount);

5821         for (sge = &acp->sgt[0],
```



```
5822             sgp = &((struct aac_sg_table64 *)&srb->sg)->SgEntry64[0];
5823             sge < &acp->sgt[acp->left_cookien]; sge++, sgp++) {
5824                 ddi_put32(acc, AAC_LO32(&sgp->SgAddress), sge->addr.ad64.lo);
5825                 ddi_put32(acc, AAC_HI32(&sgp->SgAddress), sge->addr.ad64.hi);
5826                 ddi_put32(acc, &sgp->SgByteCount, sge->bcount);
5827         }
5828 }

5830 static int
5831 aac_cmd_slot_bind(struct aac_softstate *softs, struct aac_cmd *acp)
5832 {
5833         struct aac_slot *slotp;

5835         if ((softs->flags & AAC_FLAGS_SYNC_MODE) &&
5836                 softs->sync_slot_busy)
5837                 return (AACERR);

5839         if (slotp = aac_get_slot(softs)) {
5840                 acp->slotp = slotp;
5841                 slotp->acp = acp;
5842                 acp->aac_cmd_fib(softs, acp);
5843                 (void) ddi_dma_sync(slotp->fib_dma_handle, 0, 0,
5844                         DDI_DMA_SYNC_FORDEV);
5845                 if (softs->flags & AAC_FLAGS_SYNC_MODE) {
5846                         softs->sync_slot_busy = 1;
5847                         softs->sync_mode_slot = slotp;
5848                 }
5849                 return (AACOK);
5850         }
5851         return (AACERR);
5852 }

5854 static int
5855 aac_bind_io(struct aac_softstate *softs, struct aac_cmd *acp)
5856 {
5857         struct aac_device *dvp = acp->dvp;
5858         int q = AAC_CMDQ(acp);

5884         if (softs->bus_ncmds[q] < softs->bus_throttle[q]) {
5860         if (dvp) {
5861                 if (dvp->ncmds[q] < dvp->throttle[q]) {
5862                         if (!(acp->flags & AAC_CMD_NTAG) ||
5863                             dvp->ncmds[q] == 0) {
5864 do_bind:
5865                                 return (aac_cmd_slot_bind(softs, acp));
5866                         }
5867                         ASSERT(q == AAC_CMDQ_ASYNC);
5868                         aac_set_throttle(softs, dvp, AAC_CMDQ_ASYNC,
5869                             AAC_THROTTLE_DRAIN);
5870                 }
5871         } else {
5872                 if (softs->bus_ncmds[q] < softs->bus_throttle[q])
5873                         goto do_bind;
5896                         return (aac_cmd_slot_bind(softs, acp));
5874         }
5898         }
5875         return (AACERR);
5876 }

5902 static int
5903 aac_sync_fib_slot_bind(struct aac_softstate *softs, struct aac_cmd *acp)
5904 {
5905         struct aac_slot *slotp;

5907         while (softs->sync_ac.slotp)
5908                 cv_wait(&softs->sync_fib_cv, &softs->io_lock);
```

```
5910         if (slotp = aac_get_slot(softs)) {
5911                 ASSERT(acp->slotp == NULL);

5913                 acp->slotp = slotp;
5914                 slotp->acp = acp;
5915                 return (AACOK);
5916         }
5917         return (AACERR);
5918 }

5878 static void
5921 aac_sync_fib_slot_release(struct aac_softstate *softs, struct aac_cmd *acp)
5922 {
5923         ASSERT(acp->slotp);

5925         aac_release_slot(softs, acp->slotp);
5926         acp->slotp->acp = NULL;
5927         acp->slotp = NULL;

5929         cv_signal(&softs->sync_fib_cv);
5930 }

5932 static void
5879 aac_start_io(struct aac_softstate *softs, struct aac_cmd *acp)
5880 {
5881         struct aac_slot *slotp = acp->slotp;
5882         int q = AAC_CMDQ(acp);
5883         int rval;

5885         /* Set ac and pkt */
5886         if (acp->pkt) { /* ac from ioctl has no pkt */
5887                 acp->pkt->pkt_state |=
5888                     STATE_GOT_BUS | STATE_GOT_TARGET | STATE_SENT_CMD;
5889         }
5890         if (acp->timeout) /* 0 indicates no timeout */
5891                 acp->timeout += aac_timebase + aac_tick;

5893         if (acp->dvp)
5894                 acp->dvp->ncmds[q]++;
5895         softs->bus_ncmds[q]++;
5896         aac_cmd_enqueue(&softs->q_busy, acp);

5952         AACDB_PRINT_FIB(softs, slotp);

5899         if (softs->flags & AAC_FLAGS_SYNC_MODE) {
5900                 uint32_t wait = 0;
5901                 rval = aac_sync_mbcommand(softs, AAC_MONKER_SYNCFIB,
5902                     slotp->fib_phyaddr, 0, 0, 0, &wait, NULL);
5903         } else if (softs->flags & AAC_FLAGS_NEW_COMM) {
5904                 rval = AAC_SEND_COMMAND(softs, slotp);
5954         if (softs->flags & AAC_FLAGS_NEW_COMM) {
5955                 rval = aac_send_command(softs, slotp);
5905         } else {
5906                 /*
5907                  * If fib can not be enqueued, the adapter is in an abnormal
5908                  * state, there will be no interrupt to us.
5909                  */
5910                 rval = aac_fib_enqueue(softs, AAC_ADAP_NORM_CMD_Q,
5911                     slotp->fib_phyaddr, acp->fib_size);
5912         }

5914         if (aac_check_dma_handle(slotp->fib_dma_handle) != DDI_SUCCESS)
5915                 aac_fm_service_impact(softs->devinfo_p, DDI_SERVICE_UNAFFECTED);
5966                 ddi_fm_service_impact(softs->devinfo_p, DDI_SERVICE_UNAFFECTED);
```



```
5917         /*
5918          * NOTE: We send command only when slots availabe, so should never
5919          * reach here.
5920          */
5921         if (rval != AACOK) {
5922                 AACDB_PRINT(softs, CE_NOTE, "SCMD send failed");
5923                 if (acp->pkt) {
5924                         acp->pkt->pkt_state &= ~STATE_SENT_CMD;
5925                         aac_set_pkt_reason(softs, acp, CMD_INCOMPLETE, 0);
5926                 }
5927                 aac_end_io(softs, acp);
5928                 if (!(acp->flags & (AAC_CMD_NO_INTR | AAC_CMD_NO_CB)))
5929                         ddi_trigger_softintr(softs->softint_id);
5930         }
5931 }
```

_____***unchanged_portion_omitted_***

```
5963 static void
5964 aac_drain_comp_q(struct aac_softstate *softs)
5965 {
5966         struct aac_cmd *acp;
5967         struct scsi_pkt *pkt;

5969         /*CONSTCOND*/
5970         while (1) {
5971                 mutex_enter(&softs->q_comp_mutex);
5972                 acp = aac_cmd_dequeue(&softs->q_comp);
5973                 mutex_exit(&softs->q_comp_mutex);
5974                 if (acp != NULL) {
5975                         ASSERT(acp->pkt != NULL);
5976                         pkt = acp->pkt;

5978                         if (pkt->pkt_reason == CMD_CMPLT) {
5979                                 /*
5980                                  * Consistent packets need to be sync'ed first
5981                                  */
5982                                 if ((acp->flags & AAC_CMD_CONSISTENT) &&
5983                                     (acp->flags & AAC_CMD_BUF_READ)) {
5984                                         if (aac_dma_sync_ac(acp) != AACOK) {
5985                                                 aac_fm_service_impact(
6036                                                 ddi_fm_service_impact(
5986                                                     softs->devinfo_p,
5987                                                     DDI_SERVICE_UNAFFECTED);
5988                                                 pkt->pkt_reason = CMD_TRAN_ERR;
5989                                                 pkt->pkt_statistics = 0;
5990                                         }
5991                                 }
5992                                 if ((aac_check_acc_handle(softs-> \
5993                                     comm_space_acc_handle) != DDI_SUCCESS) ||
5994                                     (aac_check_acc_handle(softs-> \
5995                                     pci_mem_handle[0]) != DDI_SUCCESS)) {
5996                                         aac_fm_service_impact(softs->devinfo_p,
6046                                         pci_mem_handle) != DDI_SUCCESS)) {
6047                                         ddi_fm_service_impact(softs->devinfo_p,
5997                                             DDI_SERVICE_UNAFFECTED);
5998                                         aac_fm_acc_err_clear(softs-> \
5999                                             pci_mem_handle[0], DDI_FME_VER0);
6049                                         ddi_fm_acc_err_clear(softs-> \
6050                                             pci_mem_handle, DDI_FME_VER0);
6000                                         pkt->pkt_reason = CMD_TRAN_ERR;
6001                                         pkt->pkt_statistics = 0;
6002                                 }
6003                                 if (aac_check_dma_handle(softs-> \
6004                                     comm_space_dma_handle) != DDI_SUCCESS) {
6005                                         aac_fm_service_impact(softs->devinfo_p,
6056                                         ddi_fm_service_impact(softs->devinfo_p,
```

```
                                        DDI_SERVICE_UNAFFECTED);
                                    pkt->pkt_reason = CMD_TRAN_ERR;
                                    pkt->pkt_statistics = 0;
                                }
                        }
                        (*pkt->pkt_comp)(pkt);
                        scsi_hba_pkt_comp(pkt);
                } else {
                        break;
                }
        }
}

static int
aac_alloc_fib(struct aac_softstate *softs, struct aac_slot *slotp)
{
        size_t rlen, maxsize;
        size_t rlen;
        ddi_dma_cookie_t cookie;
        uint_t cookien;

        /* Allocate FIB dma resource */
        if (ddi_dma_alloc_handle(
            softs->devinfo_p,
            &softs->addr_dma_attr,
            DDI_DMA_SLEEP,
            NULL,
            &slotp->fib_dma_handle) != DDI_SUCCESS) {
                AACDB_PRINT(softs, CE_WARN,
                    "Cannot alloc dma handle for slot fib area");
                goto error;
        }

        if (softs->flags & AAC_FLAGS_NEW_COMM_TYPE1)
                maxsize = softs->aac_max_fib_size + sizeof(struct aac_fib_xporth
        else if (softs->flags & AAC_FLAGS_NEW_COMM_TYPE2)
                maxsize = softs->aac_max_fib_size + 31;
        else
                maxsize = softs->aac_max_fib_size;
        if (ddi_dma_mem_alloc(
            slotp->fib_dma_handle,
            maxsize,
            &aac_acc_attr,
            softs->aac_max_fib_size,
            &softs->acc_attr,
            DDI_DMA_RDWR | DDI_DMA_CONSISTENT,
            DDI_DMA_SLEEP,
            NULL,
            (caddr_t *)&slotp->fibp,
            &rlen,
            &slotp->fib_acc_handle) != DDI_SUCCESS) {
                AACDB_PRINT(softs, CE_WARN,
                    "Cannot alloc mem for slot fib area");
                goto error;
        }
        if (ddi_dma_addr_bind_handle(
            slotp->fib_dma_handle,
            NULL,
            (caddr_t)slotp->fibp,
            maxsize,
            softs->aac_max_fib_size,
            DDI_DMA_RDWR | DDI_DMA_CONSISTENT,
            DDI_DMA_SLEEP,
            NULL,
            &cookie,
            &cookien) != DDI_DMA_MAPPED) {
```



```
                AACDB_PRINT(softs, CE_WARN,
                    "dma bind failed for slot fib area");
                goto error;
        }

        /* Check dma handles allocated in fib attach */
        if (aac_check_dma_handle(slotp->fib_dma_handle) != DDI_SUCCESS) {
                aac_fm_service_impact(softs->devinfo_p, DDI_SERVICE_LOST);
                ddi_fm_service_impact(softs->devinfo_p, DDI_SERVICE_LOST);
                goto error;
        }

        /* Check acc handles allocated in fib attach */
        if (aac_check_acc_handle(slotp->fib_acc_handle) != DDI_SUCCESS) {
                aac_fm_service_impact(softs->devinfo_p, DDI_SERVICE_LOST);
                ddi_fm_service_impact(softs->devinfo_p, DDI_SERVICE_LOST);
                goto error;
        }

        slotp->fib_phyaddr = cookie.dmac_laddress;
        if (softs->flags & AAC_FLAGS_NEW_COMM_TYPE1) {
                uint64_t fibphys_aligned;
                fibphys_aligned =
                        (slotp->fib_phyaddr + sizeof(struct aac_fib_xporthdr) +
                slotp->fibp = (struct aac_fib *)
                        ((uint8_t *)slotp->fibp + (fibphys_aligned - slotp->fib_
                slotp->fib_phyaddr = fibphys_aligned;
        } else if (softs->flags & AAC_FLAGS_NEW_COMM_TYPE2) {
                uint64_t fibphys_aligned;
                fibphys_aligned =
                        (slotp->fib_phyaddr + 31) & ~31;
                slotp->fibp = (struct aac_fib *)
                        ((uint8_t *)slotp->fibp + (fibphys_aligned - slotp->fib_
                slotp->fib_phyaddr = fibphys_aligned;
        }
        return (AACOK);

error:
        if (slotp->fib_acc_handle) {
                ddi_dma_mem_free(&slotp->fib_acc_handle);
                slotp->fib_acc_handle = NULL;
        }
        if (slotp->fib_dma_handle) {
                ddi_dma_free_handle(&slotp->fib_dma_handle);
                slotp->fib_dma_handle = NULL;
        }
        return (AACERR);
}
______unchanged_portion_omitted_

static void
aac_alloc_fibs(struct aac_softstate *softs)
{
        int i;
        struct aac_slot *slotp;

        for (i = 0; i < softs->total_slots &&
            softs->total_fibs < softs->total_slots; i++) {
                slotp = &(softs->io_slot[i]);
                if (slotp->fib_phyaddr)
                        continue;
                if (aac_alloc_fib(softs, slotp) != AACOK)
                        break;

                /* Insert the slot to the free slot list */
                aac_release_slot(softs, slotp);
```

```
6141                    softs->total_fibs++;
6142            }
6143            aac_alloc_fib(softs, softs->sync_slot);
6144    }

6146    static void
6147    aac_destroy_fibs(struct aac_softstate *softs)
6148    {
6149            struct aac_slot *slotp;

6151            while ((slotp = softs->free_io_slot_head) != NULL) {
6152                    ASSERT(slotp->fib_phyaddr);
6153                    softs->free_io_slot_head = slotp->next;
6154                    aac_free_fib(slotp);
6155                    ASSERT(slotp->index == (slotp - softs->io_slot));
6156                    softs->total_fibs--;
6157            }
6158            ASSERT(softs->total_fibs == 0);
6159            aac_free_fib(softs->sync_slot);
6160    }

6162    static int
6163    aac_create_slots(struct aac_softstate *softs)
6164    {
6165            int i;

6167            softs->total_slots = softs->aac_max_fibs;
6168            /* FIXME: patch max. outstanding commands */
6169            if (softs->total_slots > 1)
6170                    softs->total_slots -= 1;

6172            softs->io_slot = kmem_zalloc(sizeof (struct aac_slot) * \
6173                (softs->total_slots + 1), KM_SLEEP);
6196                softs->total_slots, KM_SLEEP);
6174            if (softs->io_slot == NULL) {
6175                    AACDB_PRINT(softs, CE_WARN, "Cannot allocate slot");
6176                    return (AACERR);
6177            }
6178            for (i = 0; i < softs->total_slots; i++)
6179                    softs->io_slot[i].index = i;
6180            softs->free_io_slot_head = NULL;
6181            softs->total_fibs = 0;
6182            softs->sync_slot = &softs->io_slot[softs->total_slots];
6183            return (AACOK);
6184    }

6186    static void
6187    aac_destroy_slots(struct aac_softstate *softs)
6188    {
6189            ASSERT(softs->free_io_slot_head == NULL);

6191            kmem_free(softs->io_slot, sizeof (struct aac_slot) * \
6192                (softs->total_slots + 1));
6214                softs->total_slots);
6193            softs->io_slot = NULL;
6194            softs->total_slots = 0;
6195            softs->sync_slot = NULL;
6196    }
______unchanged_portion_omitted_

6245    static int
6246    aac_do_poll_io(struct aac_softstate *softs, struct aac_cmd *acp)
6247    {
6248            int (*intr_handler)(struct aac_softstate *);

6250            /*
```



```
6251             * Interrupt is disabled, we have to poll the adapter by ourselves.
6252             */
6253            if ((softs->flags & AAC_FLAGS_NEW_COMM_TYPE1) ||
6254                    (softs->flags & AAC_FLAGS_NEW_COMM_TYPE2) ||
6255                    (softs->flags & AAC_FLAGS_NEW_COMM_TYPE34))
6256                    intr_handler = aac_process_intr_new_type1;
6257            else if (softs->flags & AAC_FLAGS_NEW_COMM)
6258                    intr_handler = aac_process_intr_new;
6259            else
6260                    intr_handler = aac_process_intr_old;
6274            intr_handler = (softs->flags & AAC_FLAGS_NEW_COMM) ?
6275                aac_process_intr_new : aac_process_intr_old;
6261            while (!(acp->flags & (AAC_CMD_CMPLT | AAC_CMD_ABORT))) {
6262                    int i = AAC_POLL_TIME * 1000;

6264                    AAC_BUSYWAIT((intr_handler(softs) != AAC_DB_RESPONSE_READY), i);
6265                    if (i == 0)
6266                            aac_cmd_timeout(softs, acp);
6267            }

6269            ddi_trigger_softintr(softs->softint_id);

6271            if ((acp->flags & AAC_CMD_CMPLT) && !(acp->flags & AAC_CMD_ERR))
6272                    return (AACOK);
6273            return (AACERR);
6274    }
______unchanged_portion_omitted_

6289    static int
6290    aac_dma_sync_ac(struct aac_cmd *acp)
6291    {
6292            uint_t i;
6293            uint32_t offset;
6294            int rval = AACOK;

6296            for (i = 0, offset = 0; i < acp->segment_cnt; ++i) {
6297                    if (!acp->segments[i].buf_dma_handle) {
6298                            offset += acp->segments[i].abp_size;
6299                            continue;
6300                    }
6307            if (acp->buf_dma_handle) {
6301                    if (acp->flags & AAC_CMD_BUF_WRITE) {
6302                            if (acp->segments[i].abp != NULL) {
6303                                    ddi_rep_put8(acp->segments[i].abh,
6304                                            (uint8_t *)acp->bp->b_un.b_addr + offset
6305                                            (uint8_t *)acp->segments[i].abp,
6306                                            acp->segments[i].abp_size,
6309                            if (acp->abp != NULL)
6310                                    ddi_rep_put8(acp->abh,
6311                                        (uint8_t *)acp->bp->b_un.b_addr,
6312                                        (uint8_t *)acp->abp, acp->bp->b_bcount,
6307                                            DDI_DEV_AUTOINCR);
6308                            }
6309                            (void) ddi_dma_sync(acp->segments[i].buf_dma_handle,
6310                                    0, 0, DDI_DMA_SYNC_FORDEV);
6314                            (void) ddi_dma_sync(acp->buf_dma_handle, 0, 0,
6315                                DDI_DMA_SYNC_FORDEV);
6311                    } else {
6312                            (void) ddi_dma_sync(acp->segments[i].buf_dma_handle,
6313                                    0, 0, DDI_DMA_SYNC_FORCPU);
6314                            if (aac_check_dma_handle(acp->segments[i].buf_dma_handle
6315                                DDI_SUCCESS) {
6316                                    rval = AACERR;
6317                            } else if (acp->segments[i].abp != NULL) {
6318                                    ddi_rep_get8(acp->segments[i].abh,
6319                                            (uint8_t *)acp->bp->b_un.b_addr + offset
```

```
6320                                        (uint8_t *)acp->segments[i].abp,
6321                                        acp->segments[i].abp_size,
6317                        (void) ddi_dma_sync(acp->buf_dma_handle, 0, 0,
6318                            DDI_DMA_SYNC_FORCPU);
6319                        if (aac_check_dma_handle(acp->buf_dma_handle) !=
6320                            DDI_SUCCESS)
6321                                return (AACERR);
6322                        if (acp->abp != NULL)
6323                                ddi_rep_get8(acp->abh,
6324                                    (uint8_t *)acp->bp->b_un.b_addr,
6325                                    (uint8_t *)acp->abp, acp->bp->b_bcount,
6322                                    DDI_DEV_AUTOINCR);
6323                        }
6324                }
6325                offset += acp->segments[i].abp_size;
6329        return (AACOK);
6330 }

6332 /*
6333  * Copy AIF from adapter to the empty AIF slot and inform AIF threads
6334  */
6335 static void
6336 aac_save_aif(struct aac_softstate *softs, ddi_acc_handle_t acc,
6337     struct aac_fib *fibp0, int fib_size0)
6338 {
6339        struct aac_fib *fibp;   /* FIB in AIF queue */
6340        int fib_size;
6341        uint16_t fib_command;
6342        int current, next;

6344        /* Ignore non AIF messages */
6345        fib_command = ddi_get16(acc, &fibp0->Header.Command);
6346        if (fib_command != AifRequest) {
6347                cmn_err(CE_WARN, "!Unknown command from controller");
6348                return;
6326        }

6351        mutex_enter(&softs->aifq_mutex);

6353        /* Save AIF */
6354        fibp = &softs->aifq[softs->aifq_idx].d;
6355        fib_size = (fib_size0 > AAC_FIB_SIZE) ? AAC_FIB_SIZE : fib_size0;
6356        ddi_rep_get8(acc, (uint8_t *)fibp, (uint8_t *)fibp0, fib_size,
6357            DDI_DEV_AUTOINCR);

6359        if (aac_check_acc_handle(softs->pci_mem_handle) != DDI_SUCCESS) {
6360                ddi_fm_service_impact(softs->devinfo_p,
6361                    DDI_SERVICE_UNAFFECTED);
6362                mutex_exit(&softs->aifq_mutex);
6363                return;
6364        }

6366        AACDB_PRINT_AIF(softs, (struct aac_aif_command *)&fibp->data[0]);

6368        /* Modify AIF contexts */
6369        current = softs->aifq_idx;
6370        next = (current + 1) % AAC_AIFQ_LENGTH;
6371        if (next == 0) {
6372                struct aac_fib_context *ctx_p;

6374                softs->aifq_wrap = 1;
6375                for (ctx_p = softs->fibctx_p; ctx_p; ctx_p = ctx_p->next) {
6376                        if (next == ctx_p->ctx_idx) {
6377                                ctx_p->ctx_flags |= AAC_CTXFLAG_FILLED;
6378                        } else if (current == ctx_p->ctx_idx &&
6379                            (ctx_p->ctx_flags & AAC_CTXFLAG_FILLED)) {
```



```
6380                                ctx_p->ctx_idx = next;
6381                                ctx_p->ctx_overrun++;
6382                        }
6383                }
6384        }
6385        softs->aifq_idx = next;

6387        /* Wakeup AIF threads */
6388        cv_broadcast(&softs->aifq_cv);
6389        mutex_exit(&softs->aifq_mutex);

6391        /* Wakeup event thread to handle aif */
6392        aac_event_disp(softs, AAC_EVENT_AIF);
6393 }

6395 static int
6396 aac_return_aif_common(struct aac_softstate *softs, struct aac_fib_context *ctx,
6397     struct aac_fib **fibpp)
6398 {
6399        int current;

6401        current = ctx->ctx_idx;
6402        if (current == softs->aifq_idx &&
6403            !(ctx->ctx_flags & AAC_CTXFLAG_FILLED))
6404                return (EAGAIN); /* Empty */

6406        *fibpp = &softs->aifq[current].d;

6408        ctx->ctx_flags &= ~AAC_CTXFLAG_FILLED;
6409        ctx->ctx_idx = (current + 1) % AAC_AIFQ_LENGTH;
6410        return (0);
6411 }

6413 int
6414 aac_return_aif(struct aac_softstate *softs, struct aac_fib_context *ctx,
6415     struct aac_fib **fibpp)
6416 {
6417        int rval;

6419        mutex_enter(&softs->aifq_mutex);
6420        rval = aac_return_aif_common(softs, ctx, fibpp);
6421        mutex_exit(&softs->aifq_mutex);
6327        return (rval);
6328 }

6425 int
6426 aac_return_aif_wait(struct aac_softstate *softs, struct aac_fib_context *ctx,
6427     struct aac_fib **fibpp)
6428 {
6429        int rval;

6431        mutex_enter(&softs->aifq_mutex);
6432        rval = aac_return_aif_common(softs, ctx, fibpp);
6433        if (rval == EAGAIN) {
6434                AACDB_PRINT(softs, CE_NOTE, "Waiting for AIF");
6435                rval = cv_wait_sig(&softs->aifq_cv, &softs->aifq_mutex);
6436        }
6437        mutex_exit(&softs->aifq_mutex);
6438        return ((rval > 0) ? 0 : EINTR);
6439 }

6330 /*
6331  * The following function comes from Adaptec:
6332  *
6333  * When driver sees a particular event that means containers are changed, it
6334  * will rescan containers. However a change may not be complete until some
```

```
6335  * other event is received. For example, creating or deleting an array will
6336  * incur as many as six AifEnConfigChange events which would generate six
6337  * container rescans. To diminish rescans, driver set a flag to wait for
6338  * another particular event. When sees that events come in, it will do rescan.
6339  */
6340 static int
6341 aac_handle_aif(struct aac_softstate *softs, struct aac_fib *fibp)
6452 aac_handle_aif(struct aac_softstate *softs, struct aac_aif_command *aif)
6342 {
6343         ddi_acc_handle_t acc = softs->comm_space_acc_handle;
6344         uint16_t fib_command;
6345         struct aac_aif_command *aif;
6346         int en_type;
6347         int devcfg_needed;
6348         int current, next;
6457         int cid;
6458         uint32_t bus_id, tgt_id;
6459         enum aac_cfg_event event = AAC_CFG_NULL_EXIST;

6350         fib_command = LE_16(fibp->Header.Command);
6351         if (fib_command != AifRequest) {
6352                 cmn_err(CE_NOTE, "!Unknown command from controller: 0x%x",
6353                     fib_command);
6354                 return (AACERR);
6355         }

6357         /* Update internal container state */
6358         aif = (struct aac_aif_command *)&fibp->data[0];

6360         AACDB_PRINT_AIF(softs, aif);
6361         devcfg_needed = 0;
6362         en_type = LE_32((uint32_t)aif->data.EN.type);

6364         switch (LE_32((uint32_t)aif->command)) {
6365         case AifCmdDriverNotify: {
6366                 int cid = LE_32(aif->data.EN.data.ECC.container[0]);
6466                 cid = LE_32(aif->data.EN.data.ECC.container[0]);

6368                 switch (en_type) {
6369                 case AifDenMorphComplete:
6370                 case AifDenVolumeExtendComplete:
6371                         if (cid < AAC_MAX_LD && softs->containers[cid].dev.valid
6471                         if (AAC_DEV_IS_VALID(&softs->containers[cid].dev))
6372                                 softs->devcfg_wait_on = AifEnConfigChange;
6373                         break;
6374                 }
6375                 if (softs->devcfg_wait_on == en_type)
6376                         devcfg_needed = 1;
6377                 break;
6378         }

6380         case AifCmdEventNotify:
6481                 cid = LE_32(aif->data.EN.data.ECC.container[0]);
6381                 switch (en_type) {
6382                 case AifEnAddContainer:
6383                 case AifEnDeleteContainer:
6384                         softs->devcfg_wait_on = AifEnConfigChange;
6385                         break;
6386                 case AifEnContainerChange:
6387                         if (!softs->devcfg_wait_on)
6388                                 softs->devcfg_wait_on = AifEnConfigChange;
6389                         break;
6390                 case AifEnContainerEvent:
6391                         if (ddi_get32(acc, &aif-> \
6392                             data.EN.data.ECE.eventType) == CT_PUP_MISSING_DRIVE)
6393                                 devcfg_needed = 1;
```



```
6394                         break;
6395                 case AifEnAddJBOD:
6497                         if (!(softs->flags & AAC_FLAGS_JBOD))
6498                                 return (AACERR);
6499                         event = AAC_CFG_ADD;
6500                         bus_id = (cid >> 24) & 0xf;
6501                         tgt_id = cid & 0xffff;
6502                         break;
6396                 case AifEnDeleteJBOD:
6397                 case AifRawDeviceRemove:
6398                         timeout(aac_config_pd, softs, drv_usectohz(1000));
6504                         if (!(softs->flags & AAC_FLAGS_JBOD))
6505                                 return (AACERR);
6506                         event = AAC_CFG_DELETE;
6507                         bus_id = (cid >> 24) & 0xf;
6508                         tgt_id = cid & 0xffff;
6399                         break;
6400                 }
6401                 if (softs->devcfg_wait_on == en_type)
6402                         devcfg_needed = 1;
6403                 break;

6405         case AifCmdJobProgress:
6406                 if (LE_32((uint32_t)aif->data.PR[0].jd.type) == AifJobCtrZero) {
6407                         int pr_status;
6408                         uint32_t pr_ftick, pr_ctick;

6410                         pr_status = LE_32((uint32_t)aif->data.PR[0].status);
6411                         pr_ctick = LE_32(aif->data.PR[0].currentTick);
6412                         pr_ftick = LE_32(aif->data.PR[0].finalTick);

6414                         if ((pr_ctick == pr_ftick) ||
6415                             (pr_status == AifJobStsSuccess))
6416                                 softs->devcfg_wait_on = AifEnContainerChange;
6417                         else if ((pr_ctick == 0) &&
6418                             (pr_status == AifJobStsRunning))
6419                                 softs->devcfg_wait_on = AifEnContainerChange;
6420                 }
6421                 break;
6422         }

6424         if (devcfg_needed)
6534         if (devcfg_needed) {
6535                 softs->devcfg_wait_on = 0;
6425                 (void) aac_probe_containers(softs);

6427         /* Modify AIF contexts */
6428         current = softs->aifq_idx;
6429         next = (current + 1) % AAC_AIFQ_LENGTH;
6430         if (next == 0) {
6431                 struct aac_fib_context *ctx;

6433                 softs->aifq_wrap = 1;
6434                 for (ctx = softs->fibctx; ctx; ctx = ctx->next) {
6435                         if (next == ctx->ctx_idx) {
6436                                 ctx->ctx_filled = 1;
6437                         } else if (current == ctx->ctx_idx && ctx->ctx_filled) {
6438                                 ctx->ctx_idx = next;
6439                         }
6440                 }
6441         }
6442         softs->aifq_idx = next;

6444         /* Wakeup applications */
6445         cv_broadcast(&softs->aifv);
6539         if (event != AAC_CFG_NULL_EXIST) {
```

```
6540                 ASSERT(en_type == AifEnAddJBOD || en_type == AifEnDeleteJBOD);
6541                 (void) aac_probe_jbod(softs,
6542                     AAC_P2VTGT(softs, bus_id, tgt_id), event);
6543         }
6446         return (AACOK);
6447 }


6449 /*
6450  * Timeout recovery
6549  * Check and handle AIF events
6451  */
6452 static void
6453 aac_cmd_timeout(struct aac_softstate *softs, struct aac_cmd *acp)
6552 aac_aif_event(struct aac_softstate *softs)
6454 {
6455         int rval;
6554         struct aac_fib *fibp;

6457         /* print timed out command */
6458 #ifdef AAC_DEBUG
6459         {
6460                 struct aac_slot *slotp = acp->slotp;
6461                 ddi_acc_handle_t acc = slotp->fib_acc_handle;
6462                 uint16_t i, cmd, size;
6463                 uint8_t *dataptr;
6464                 uint8_t data[16];
6465                 uint32_t timeout;
6556         /*CONSTCOND*/
6557         while (1) {
6558                 if (aac_return_aif(softs, &softs->aifctx, &fibp) != 0)
6559                         break; /* No more AIFs to handle, end loop */

6467                 timeout = acp->timeout - aac_timebase - aac_tick;
6468                 if (acp->ac_comp == aac_ld_complete) {
6469                         AACDB_PRINT(softs, CE_NOTE,
6470                                 "Container command timed out, index %d timeout %
6471                                 slotp->index, timeout, acp->flags,
6472                                 ((struct aac_container *)acp->dvp)->cid);
6473                 } else if (acp->ac_comp == aac_pd_complete) {
6474                         AACDB_PRINT(softs, CE_NOTE,
6475                                 "Nondasd command timed out, index %d timeout %d
6476                                 slotp->index, timeout, acp->flags,
6477                                 ((struct aac_nondasd *)acp->dvp)->bus,
6478                                 ((struct aac_nondasd *)acp->dvp)->tid);
6479                 } else if (acp->ac_comp == aac_ioctl_complete) {
6480                         AACDB_PRINT(softs, CE_NOTE,
6481                                 "IOCTL command timed out, index %d timeout %d fl
6482                                 slotp->index, timeout, acp->flags);
6483                 } else if (acp->ac_comp == aac_synccache_complete) {
6484                         AACDB_PRINT(softs, CE_NOTE,
6485                                 "Flush command timed out, index %d timeout %d fl
6486                                 slotp->index, timeout, acp->flags);
6487                 } else if (acp->ac_comp == aac_aifreq_complete) {
6488                         AACDB_PRINT(softs, CE_NOTE,
6489                                 "AIF request command timed out, index %d timeout
6490                                 slotp->index, timeout, acp->flags);
6561                 /* AIF overrun, array create/delete may missed. */
6562                 if (softs->aifctx.ctx_overrun) {
6563                         softs->aifctx.ctx_overrun = 0;
6491                 }
6492                 cmd = ddi_get16(acc, &slotp->fibp->Header.Command);
6493                 size = acp->fib_size - sizeof(struct aac_fib_header);
6494                 AACDB_PRINT(softs, CE_NOTE, "Fib command 0x%x size %d data:",
6495                         cmd, size);
```



```
6497                 if (size > AAC_FIB_DATASIZE) size = AAC_FIB_DATASIZE;
6498                 dataptr = &slotp->fibp->data[0];
6499                 while (size) {
6500                         for (i = 0; i < 16; ++i) {
6501                                 if (size) {
6502                                         data[i] = ddi_get8(acc, dataptr);
6503                                         ++dataptr;
6504                                         --size;
6505                                 } else {
6506                                         data[i] = 0;
6566                 /* AIF received, handle it */
6567                 struct aac_aif_command *aifp =
6568                     (struct aac_aif_command *)&fibp->data[0];
6569                 uint32_t aif_command = LE_32((uint32_t)aifp->command);

6571                 if (aif_command == AifCmdDriverNotify ||
6572                     aif_command == AifCmdEventNotify ||
6573                     aif_command == AifCmdJobProgress)
6574                         (void) aac_handle_aif(softs, aifp);
6507                         }
6508                         }
6509                         AACDB_PRINT(softs, CE_NOTE, "0x%02x 0x%02x 0x%02x 0x%02x
6510                                 data[0], data[1], data[2], data[3],
6511                                 data[4], data[5], data[6], data[7],
6512                                 data[8], data[9], data[10], data[11],
6513                                 data[12], data[13], data[14], data[15]);
6514                 }
6515         }
6576 }

6578 /*
6579  * Timeout recovery
6580  */
6581 /*ARGSUSED*/
6582 static void
6583 aac_cmd_timeout(struct aac_softstate *softs, struct aac_cmd *acp)
6584 {
6585 #ifdef DEBUG
6586         acp->fib_flags |= AACDB_FLAGS_FIB_TIMEOUT;
6587         AACDB_PRINT(softs, CE_WARN, "acp %p timed out", acp);
6588         AACDB_PRINT_FIB(softs, acp->slotp);
6516 #endif


6519         /*
6520          * Besides the firmware in unhealthy state, an overloaded
6521          * adapter may also incur pkt timeout.
6522          * There is a chance for an adapter with a slower IOP to take
6523          * longer than 60 seconds to process the commands, such as when
6524          * to perform IOs. So the adapter is doing a build on a RAID-5
6525          * while being required longer completion times should be
6526          * tolerated.
6527          */
6528         rval = aac_do_reset(softs);
6529         if (rval == AACOK) {
6530                 aac_abort_iocmds(softs, AAC_IOCMD_OUTSTANDING, NULL,
6531                     CMD_RESET);
6600         switch (aac_do_reset(softs)) {
6601         case AAC_IOP_RESET_SUCCEED:
6602                 aac_abort_iocmds(softs, AAC_IOCMD_OUTSTANDING, NULL, CMD_RESET);
6532                 aac_start_waiting_io(softs);
6533         } else if (rval == AACOK2) {
6534                 aac_start_waiting_io(softs);
6535         } else {
6604                 break;
6605         case AAC_IOP_RESET_FAILED:
```

```
6536                 /* Abort all waiting cmds when adapter is dead */
6537                 aac_abort_iocmds(softs, AAC_IOCMD_ALL, NULL,
6538                     CMD_TIMEOUT);
6607                 aac_abort_iocmds(softs, AAC_IOCMD_ALL, NULL, CMD_TIMEOUT);
6608                 break;
6609         case AAC_IOP_RESET_ABNORMAL:
6610                 aac_start_waiting_io(softs);
6539         }
6540 }

6542 /*
6543  * The following function comes from Adaptec:
6544  *
6545  * Time sync. command added to synchronize time with firmware every 30
6546  * minutes (required for correct AIF timestamps etc.)
6547  */
6548 static int
6620 static void
6549 aac_sync_tick(struct aac_softstate *softs)
6550 {
6551         ddi_acc_handle_t acc = softs->sync_slot->fib_acc_handle;
6552         struct aac_fib *fibp = softs->sync_slot->fibp;
6623         ddi_acc_handle_t acc;
6624         int rval;

6554         ddi_put32(acc, (uint32_t *)&fibp->data[0], ddi_get_time());
6555         return (aac_sync_fib(softs, SendHostTime, AAC_FIB_SIZEOF(uint32_t)));
6556 }
6626         mutex_enter(&softs->time_mutex);
6627         ASSERT(softs->time_sync <= softs->timebase);
6628         softs->time_sync = 0;
6629         mutex_exit(&softs->time_mutex);

6558 static void
6559 aac_daemon(void *arg)
6560 {
6561         struct aac_softstate *softs = (struct aac_softstate *)arg;
6562         struct aac_cmd *acp;
6631         /* Time sync. with firmware every AAC_SYNC_TICK */
6632         (void) aac_sync_fib_slot_bind(softs, &softs->sync_ac);
6633         acc = softs->sync_ac.slotp->fib_acc_handle;

6564         DBCALLED(softs, 2);
6635         ddi_put32(acc, (void *)&softs->sync_ac.slotp->fibp->data[0],
6636             ddi_get_time());
6637         rval = aac_sync_fib(softs, SendHostTime, AAC_FIB_SIZEOF(uint32_t));
6638         aac_sync_fib_slot_release(softs, &softs->sync_ac);

6566         mutex_enter(&softs->io_lock);
6567         /* Check slot for timeout pkts */
6568         aac_timebase += aac_tick;
6569         for (acp = softs->q_busy.q_head; acp; acp = acp->next) {
6570                 if (acp->timeout) {
6571                         if (acp->timeout <= aac_timebase) {
6572                                 aac_cmd_timeout(softs, acp);
6573                                 ddi_trigger_softintr(softs->softint_id);
6574                         }
6575                         break;
6576                 }
6577         }

6579         /* Time sync. with firmware every AAC_SYNC_TICK */
6580         if (aac_sync_time <= aac_timebase) {
6581                 aac_sync_time = aac_timebase;
6582                 if (aac_sync_tick(softs) != AACOK)
6583                         aac_sync_time += aac_tick << 1; /* retry shortly */
```



```
6640         mutex_enter(&softs->time_mutex);
6641         softs->time_sync = softs->timebase;
6642         if (rval != AACOK)
6643                 /* retry shortly */
6644                 softs->time_sync += aac_tick << 1;
6584                 else
6585                         aac_sync_time += AAC_SYNC_TICK;
6586         }

6588         if ((softs->state & AAC_STATE_RUN) && (softs->timeout_id != 0))
6589                 softs->timeout_id = timeout(aac_daemon, (void *)softs,
6590                     (aac_tick * drv_usectohz(1000000)));
6591         mutex_exit(&softs->io_lock);
6646         softs->time_sync += AAC_SYNC_TICK;
6647         mutex_exit(&softs->time_mutex);
6592 }

6594 /*
6595  * Architecture dependent functions
6651  * Timeout checking and handling
6596  */
6597 static void
6598 aac_rx_set_intr(struct aac_softstate *softs, int enable)
6654 aac_daemon(struct aac_softstate *softs)
6599 {
6600         if (enable) {
6601         if (softs->flags & AAC_FLAGS_NEW_COMM)
6602                         PCI_MEM_PUT32(softs, 0, AAC_OIMR, ~AAC_DB_INTR_NEW);
6603                 else
6604                         PCI_MEM_PUT32(softs, 0, AAC_OIMR, ~AAC_DB_INTR_BITS);
6605         } else {
6606                 PCI_MEM_PUT32(softs, 0, AAC_OIMR, ~0);
6607         }
6608 }
6656         int time_out; /* set if timeout happened */
6657         int time_adjust;
6658         uint32_t softs_timebase;

6610 static void
6611 aac_rx_status_clr(struct aac_softstate *softs, int mask)
6612 {
6613         PCI_MEM_PUT32(softs, 0, AAC_ODBR, mask);
6614 }
6660         mutex_enter(&softs->time_mutex);
6661         ASSERT(softs->time_out <= softs->timebase);
6662         softs->time_out = 0;
6663         softs_timebase = softs->timebase;
6664         mutex_exit(&softs->time_mutex);

6616 static int
6617 aac_rx_status_get(struct aac_softstate *softs)
6618 {
6619         return (PCI_MEM_GET32(softs, 0, AAC_ODBR));
6620 }
6666         /* Check slots for timeout pkts */
6667         time_adjust = 0;
6668         do {
6669                 struct aac_cmd *acp;

6622 static void
6623 aac_rx_notify(struct aac_softstate *softs, int val)
6624 {
6625         PCI_MEM_PUT32(softs, 0, AAC_IDBR, val);
6626 }
6671                 time_out = 0;
6672                 for (acp = softs->q_busy.q_head; acp; acp = acp->next) {
```

```
6673                        if (acp->timeout == 0)
6674                                continue;
```

```
6628 static int
6629 aac_rx_get_fwstatus(struct aac_softstate *softs)
6630 {
6631         return (PCI_MEM_GET32(softs, 0, AAC_OMR0));
6632 }
```

```
6676                        /*
6677                         * If timeout happened, update outstanding cmds
6678                         * to be checked later again.
6679                         */
6680                        if (time_adjust) {
6681                                acp->timeout += time_adjust;
6682                                continue;
6683                        }
```

```
6634 static int
6635 aac_rx_get_mailbox(struct aac_softstate *softs, int mb)
6636 {
6637         return (PCI_MEM_GET32(softs, 0, AAC_RX_MAILBOX + mb * 4));
```

```
6685                        if (acp->timeout <= softs_timebase) {
6686                                aac_cmd_timeout(softs, acp);
6687                                time_out = 1;
6688                                time_adjust = aac_tick * drv_usectohz(1000000);
6689                                break; /* timeout happened */
6690                        } else {
6691                                break; /* no timeout */
6692                        }
6693                }
6694         } while (time_out);
```

```
6696         mutex_enter(&softs->time_mutex);
6697         softs->time_out = softs->timebase + aac_tick;
6698         mutex_exit(&softs->time_mutex);
6638 }
```

```
6701 /*
6702  * The event thread handles various tasks serially for the other parts of
6703  * the driver, so that they can run fast.
6704  */
6640 static void
6641 aac_rx_set_mailbox(struct aac_softstate *softs, uint32_t cmd,
6642     uint32_t arg0, uint32_t arg1, uint32_t arg2, uint32_t arg3)
6706 aac_event_thread(struct aac_softstate *softs)
6643 {
6644         PCI_MEM_PUT32(softs, 0, AAC_RX_MAILBOX, cmd);
6645         PCI_MEM_PUT32(softs, 0, AAC_RX_MAILBOX + 4, arg0);
6646         PCI_MEM_PUT32(softs, 0, AAC_RX_MAILBOX + 8, arg1);
6647         PCI_MEM_PUT32(softs, 0, AAC_RX_MAILBOX + 12, arg2);
6648         PCI_MEM_PUT32(softs, 0, AAC_RX_MAILBOX + 16, arg3);
6649 }
6708         int run = 1;
```

```
6651 static int
6652 aac_rx_send_command(struct aac_softstate *softs, struct aac_slot *slotp)
6653 {
6654         uint32_t index, device;
6710         DBCALLED(softs, 1);
```

```
6656         index = PCI_MEM_GET32(softs, 0, AAC_IQUE);
6657         if (index == 0xffffffffUL) {
6658                 index = PCI_MEM_GET32(softs, 0, AAC_IQUE);
6659                 if (index == 0xffffffffUL)
6660                         return (AACERR);
6712         mutex_enter(&softs->ev_lock);
```



```
6713         while (run) {
6714                 int events;
```

```
6716                 if ((events = softs->events) == 0) {
6717                         cv_wait(&softs->event_disp_cv, &softs->ev_lock);
6718                         events = softs->events;
6661         }
6720                 softs->events = 0;
6721                 mutex_exit(&softs->ev_lock);
```

```
6663         device = index;
6664         PCI_MEM_PUT32(softs, 0, device,
6665             (uint32_t)(slotp->fib_phyaddr & 0xffffffffUL));
6666         device += 4;
6667         PCI_MEM_PUT32(softs, 0, device, (uint32_t)(slotp->fib_phyaddr >> 32));
6668         device += 4;
6669         PCI_MEM_PUT32(softs, 0, device, slotp->acp->fib_size);
6670         PCI_MEM_PUT32(softs, 0, AAC_IQUE, index);
6671         return (AACOK);
6672 }
```

```
6723                 mutex_enter(&softs->io_lock);
6724                 if ((softs->state & AAC_STATE_RUN) &&
6725                     (softs->state & AAC_STATE_DEAD) == 0) {
6726                         if (events & AAC_EVENT_TIMEOUT)
6727                                 aac_daemon(softs);
6728                         if (events & AAC_EVENT_SYNCTICK)
6729                                 aac_sync_tick(softs);
6730                         if (events & AAC_EVENT_AIF)
6731                                 aac_aif_event(softs);
6732                 } else {
6733                         run = 0;
6734                 }
6735                 mutex_exit(&softs->io_lock);
```

```
6674 static int
6675 aac_rkt_get_fwstatus(struct aac_softstate *softs)
6676 {
6677         return (PCI_MEM_GET32(softs, 0, AAC_OMR0));
6678 }
6737         mutex_enter(&softs->ev_lock);
6738         }
```

```
6680 static int
6681 aac_rkt_get_mailbox(struct aac_softstate *softs, int mb)
6682 {
6683         return (PCI_MEM_GET32(softs, 0, AAC_RKT_MAILBOX + mb *4));
6740         cv_signal(&softs->event_wait_cv);
6741         mutex_exit(&softs->ev_lock);
6684 }
```

```
6744 /*
6745  * Internal timer. It is only responsbile for time counting and report time
6746  * related events. Events handling is done by aac_event_thread(), so that
6747  * the timer itself could be as precise as possible.
6748  */
6686 static void
6687 aac_rkt_set_mailbox(struct aac_softstate *softs, uint32_t cmd,
6688     uint32_t arg0, uint32_t arg1, uint32_t arg2, uint32_t arg3)
6750 aac_timer(void *arg)
6689 {
6690         PCI_MEM_PUT32(softs, 0, AAC_RKT_MAILBOX, cmd);
6691         PCI_MEM_PUT32(softs, 0, AAC_RKT_MAILBOX + 4, arg0);
6692         PCI_MEM_PUT32(softs, 0, AAC_RKT_MAILBOX + 8, arg1);
6693         PCI_MEM_PUT32(softs, 0, AAC_RKT_MAILBOX + 12, arg2);
6694         PCI_MEM_PUT32(softs, 0, AAC_RKT_MAILBOX + 16, arg3);
6695 }
```

```
6752         struct aac_softstate *softs = arg;
6753         int events = 0;

6697 static void
6698 aac_src_set_intr(struct aac_softstate *softs, int enable)
6699 {
6700         if (enable) {
6701                 PCI_MEM_PUT32(softs, 0, AAC_SRC_OIMR, ~AAC_DB_INTR_NEW_TYPE1);
6755         mutex_enter(&softs->time_mutex);

6757         /* If timer is being stopped, exit */
6758         if (softs->timeout_id) {
6759                 softs->timeout_id = timeout(aac_timer, (void *)softs,
6760                     (aac_tick * drv_usectohz(1000000)));
6702         } else {
6703                 PCI_MEM_PUT32(softs, 0, AAC_SRC_OIMR, ~0);
6762                 mutex_exit(&softs->time_mutex);
6763                 return;
6704         }
6705 }

6707 static void
6708 aac_src_status_clr(struct aac_softstate *softs, int mask)
6709 {
6710         PCI_MEM_PUT32(softs, 0, AAC_SRC_ODBR_C, mask << AAC_SRC_ODR_SHIFT);
6711 }
6766         /* Time counting */
6767         softs->timebase += aac_tick;

6713 static int
6714 aac_src_status_get(struct aac_softstate *softs)
6715 {
6716         return (PCI_MEM_GET32(softs, 0, AAC_SRC_ODBR_R) >> AAC_SRC_ODR_SHIFT);
6769         /* Check time related events */
6770         if (softs->time_out && softs->time_out <= softs->timebase)
6771                 events |= AAC_EVENT_TIMEOUT;
6772         if (softs->time_sync && softs->time_sync <= softs->timebase)
6773                 events |= AAC_EVENT_SYNCTICK;

6775         mutex_exit(&softs->time_mutex);

6777         if (events)
6778                 aac_event_disp(softs, events);
6717 }

6781 /*
6782  * Dispatch events to daemon thread for handling
6783  */
6719 static void
6720 aac_src_notify(struct aac_softstate *softs, int val)
6785 aac_event_disp(struct aac_softstate *softs, int events)
6721 {
6722         PCI_MEM_PUT32(softs, 0, AAC_SRC_IDBR, val << AAC_SRC_IDR_SHIFT);
6787         mutex_enter(&softs->ev_lock);
6788         softs->events |= events;
6789         cv_broadcast(&softs->event_disp_cv);
6790         mutex_exit(&softs->ev_lock);
6723 }

6793 /*
6794  * Architecture dependent functions
6795  */
6725 static int
6726 aac_src_get_fwstatus(struct aac_softstate *softs)
6797 aac_rx_get_fwstatus(struct aac_softstate *softs)
6727 {
```



```
6728         return (PCI_MEM_GET32(softs, 0, AAC_SRC_OMR));
6799         return (PCI_MEM_GET32(softs, AAC_OMR0));
6729 }

6731 static int
6732 aac_src_get_mailbox(struct aac_softstate *softs, int mb)
6803 aac_rx_get_mailbox(struct aac_softstate *softs, int mb)
6733 {
6734         return (PCI_MEM_GET32(softs, 0, AAC_SRC_MAILBOX + mb *4));
6805         return (PCI_MEM_GET32(softs, AAC_RX_MAILBOX + mb * 4));
6735 }

6737 static void
6738 aac_src_set_mailbox(struct aac_softstate *softs, uint32_t cmd,
6809 aac_rx_set_mailbox(struct aac_softstate *softs, uint32_t cmd,
6739     uint32_t arg0, uint32_t arg1, uint32_t arg2, uint32_t arg3)
6740 {
6741         PCI_MEM_PUT32(softs, 0, AAC_SRC_MAILBOX, cmd);
6742         PCI_MEM_PUT32(softs, 0, AAC_SRC_MAILBOX + 4, arg0);
6743         PCI_MEM_PUT32(softs, 0, AAC_SRC_MAILBOX + 8, arg1);
6744         PCI_MEM_PUT32(softs, 0, AAC_SRC_MAILBOX + 12, arg2);
6745         PCI_MEM_PUT32(softs, 0, AAC_SRC_MAILBOX + 16, arg3);
6812         PCI_MEM_PUT32(softs, AAC_RX_MAILBOX, cmd);
6813         PCI_MEM_PUT32(softs, AAC_RX_MAILBOX + 4, arg0);
6814         PCI_MEM_PUT32(softs, AAC_RX_MAILBOX + 8, arg1);
6815         PCI_MEM_PUT32(softs, AAC_RX_MAILBOX + 12, arg2);
6816         PCI_MEM_PUT32(softs, AAC_RX_MAILBOX + 16, arg3);
6746 }

6748 static int
6749 aac_src_send_command(struct aac_softstate *softs, struct aac_slot *slotp)
6820 aac_rkt_get_fwstatus(struct aac_softstate *softs)
6750 {
6751         ddi_acc_handle_t acc = slotp->fib_acc_handle;
6752         uint32_t fibsize, hdr_size, high_addr;
6753         uint64_t address;

6755         hdr_size = (uint32_t)ddi_get16(acc, &slotp->fibp->Header.Size);
6756         if (softs->flags & AAC_FLAGS_NEW_COMM_TYPE2) {
6757                 struct aac_fib *fibp = slotp->fibp;

6759                 /* Calculate the amount to the fibsize bits */
6760                 fibsize = (hdr_size + 127) / 128 - 1;
6761                 /* New FIB header */
6762                 address = slotp->fib_phyaddr;
6763                 high_addr = (uint32_t)(address >> 32);
6764                 if (high_addr == 0L) {
6765                         ddi_put8(acc, &fibp->Header.StructType, AAC_FIBTYPE_TFIB
6766                         ddi_put32(acc, &fibp->Header.a.TimeStamp, 0L);
6767                 } else {
6768                         ddi_put8(acc, &fibp->Header.StructType, AAC_FIBTYPE_TFIB
6769                         ddi_put32(acc, &fibp->Header.a.SenderFibAddressHigh, hig
6770                 }
6771                 ddi_put32(acc, &fibp->Header.SenderFibAddress, (uint32_t)address
6772         } else {
6773                 struct aac_fib_xporthdr *pFibX;

6775                 /* Calculate the amount to the fibsize bits */
6776                 fibsize = (sizeof(struct aac_fib_xporthdr) + hdr_size + 127) / 1
6777                 /* Fill XPORT header */
6778                 address = slotp->fib_phyaddr - sizeof(struct aac_fib_xporthdr);
6779                 high_addr = (uint32_t)(address >> 32);
6780                 pFibX = (struct aac_fib_xporthdr *)
6781                         ((unsigned char *)slotp->fibp - sizeof(struct aac_fib_xp
6782                 ddi_put32(acc, &pFibX->Handle, slotp->index + 1);
6783                 ddi_put64(acc, &pFibX->HostAddress, slotp->fib_phyaddr);
```

```
6784                    ddi_put32(acc, &pFibX->Size, hdr_size);
6785            }

6787            if (fibsize > 31)
6788                    fibsize = 31;
6789            if (high_addr) {
6790                    PCI_MEM_PUT32(softs, 0, AAC_SRC_IQUE64_H, high_addr);
6791                    PCI_MEM_PUT32(softs, 0, AAC_SRC_IQUE64_L, (uint32_t)address + fi
6792            } else {
6793                    PCI_MEM_PUT32(softs, 0, AAC_SRC_IQUE32, (uint32_t)address + fibs
6794            }
6795            return (AACOK);
6822            return (PCI_MEM_GET32(softs, AAC_OMR0));
6796 }

6798 static int
6799 aac_srcv_get_mailbox(struct aac_softstate *softs, int mb)
6826 aac_rkt_get_mailbox(struct aac_softstate *softs, int mb)
6800 {
6801            return (PCI_MEM_GET32(softs, 0, AAC_SRCV_MAILBOX + mb *4));
6828            return (PCI_MEM_GET32(softs, AAC_RKT_MAILBOX + mb *4));
6802 }

6804 static void
6805 aac_srcv_set_mailbox(struct aac_softstate *softs, uint32_t cmd,
6832 aac_rkt_set_mailbox(struct aac_softstate *softs, uint32_t cmd,
6806     uint32_t arg0, uint32_t arg1, uint32_t arg2, uint32_t arg3)
6807 {
6808            PCI_MEM_PUT32(softs, 0, AAC_SRCV_MAILBOX, cmd);
6809            PCI_MEM_PUT32(softs, 0, AAC_SRCV_MAILBOX + 4, arg0);
6810            PCI_MEM_PUT32(softs, 0, AAC_SRCV_MAILBOX + 8, arg1);
6811            PCI_MEM_PUT32(softs, 0, AAC_SRCV_MAILBOX + 12, arg2);
6812            PCI_MEM_PUT32(softs, 0, AAC_SRCV_MAILBOX + 16, arg3);
6835            PCI_MEM_PUT32(softs, AAC_RKT_MAILBOX, cmd);
6836            PCI_MEM_PUT32(softs, AAC_RKT_MAILBOX + 4, arg0);
6837            PCI_MEM_PUT32(softs, AAC_RKT_MAILBOX + 8, arg1);
6838            PCI_MEM_PUT32(softs, AAC_RKT_MAILBOX + 12, arg2);
6839            PCI_MEM_PUT32(softs, AAC_RKT_MAILBOX + 16, arg3);
6813 }
______unchanged_portion_omitted_

6899 static int
6900 aac_atoi(char **pptr)
6901 {
6902            char *ptr = *pptr;
6903            int digit, rval = 0;

6905            while (*ptr >= '0' && *ptr <= '9') {
6906                    digit = (int)(*ptr - '0');
6907                    rval = rval*10 + digit;
6908                    ptr++;
6909            }
6910            *pptr = ptr;
6911            return (rval);
6912 }

6914 /*
6915  * The IO fault service error handling callback function
6916  */
6917 /*ARGSUSED*/
6918 static int
6919 aac_fm_error_cb(dev_info_t *dip, ddi_fm_error_t *err, const void *impl_data)
6920 {
6921            /*
6922             * as the driver can always deal with an error in any dma or
6923             * access handle, we can just return the fme_status value.
```



```
6924             */
6925            pci_ereport_post(dip, err, NULL);
6926            return (err->fme_status);
6927 }

6929 /*
6930  * aac_fm_init - initialize fma capabilities and register with IO
6931  *               fault services.
6932  */
6933 static void
6934 aac_fm_init(struct aac_softstate *softs)
6935 {
6936            /*
6937             * Need to change iblock to priority for new MSI intr
6938             */
6939            ddi_iblock_cookie_t fm_ibc;

6941            if (!aac_fm_valid)
6942                    return;

6944            /*
6945             * Initialize FMA
6946             */
6947            softs->fm_capabilities = ddi_getprop(DDI_DEV_T_ANY, softs->devinfo_p,
6948                    DDI_PROP_CANSLEEP | DDI_PROP_DONTPASS, "fm-capable",
6949                    DDI_FM_EREPORT_CAPABLE | DDI_FM_ACCCHK_CAPABLE |
6950                    DDI_FM_DMACHK_CAPABLE | DDI_FM_ERRCB_CAPABLE);

6952            /* Only register with IO Fault Services if we have some capability */
6953            if (softs->fm_capabilities) {
6954                    /* Adjust access and dma attributes for FMA */
6955                    aac_acc_attr.devacc_attr_access = DDI_FLAGERR_ACC;
6956                    softs->buf_dma_attr.dma_attr_flags = DDI_DMA_FLAGERR;
6957                    softs->addr_dma_attr.dma_attr_flags = DDI_DMA_FLAGERR;
6961                    softs->reg_attr.devacc_attr_access = DDI_FLAGERR_ACC;
6962                    softs->addr_dma_attr.dma_attr_flags |= DDI_DMA_FLAGERR;
6963                    softs->buf_dma_attr.dma_attr_flags |= DDI_DMA_FLAGERR;

6959                    /*
6960                     * Register capabilities with IO Fault Services.
6961                     * fm_capabilities will be updated to indicate
6962                     * capabilities actually supported (not requested.)
6963                     */
6964                    ddi_fm_init(softs->devinfo_p, &softs->fm_capabilities, &fm_ibc);

6966                    /*
6967                     * Initialize pci ereport capabilities if ereport
6968                     * capable (should always be.)
6969                     */
6970                    if (DDI_FM_EREPORT_CAP(softs->fm_capabilities) ||
6971                        DDI_FM_ERRCB_CAP(softs->fm_capabilities)) {
6972                            pci_ereport_setup(softs->devinfo_p);
6973                    }

6975                    /*
6976                     * Register error callback if error callback capable.
6977                     */
6978                    if (DDI_FM_ERRCB_CAP(softs->fm_capabilities)) {
6979                            ddi_fm_handler_register(softs->devinfo_p,
6980                                aac_fm_error_cb, (void *) softs);
6981                    }
6982            } else {
6983                    /* Clear FMA if no capabilities */
6984                    aac_acc_attr.devacc_attr_access = DDI_DEFAULT_ACC;
6985                    softs->buf_dma_attr.dma_attr_flags = 0;
6986                    softs->addr_dma_attr.dma_attr_flags = 0;
```

```
6987         }
6988 }

6990 /*
6991  * aac_fm_fini - Releases fma capabilities and un-registers with IO
6992  *               fault services.
6993  */
6994 static void
6995 aac_fm_fini(struct aac_softstate *softs)
6996 {
6997         if (!aac_fm_valid)
6998                 return;

7000         /* Only unregister FMA capabilities if registered */
7001         if (softs->fm_capabilities) {
7002                 /*
7003                  * Un-register error callback if error callback capable.
7004                  */
7005                 if (DDI_FM_ERRCB_CAP(softs->fm_capabilities)) {
7006                         ddi_fm_handler_unregister(softs->devinfo_p);
7007                 }

7009                 /*
7010                  * Release any resources allocated by pci_ereport_setup()
7011                  */
7012                 if (DDI_FM_EREPORT_CAP(softs->fm_capabilities) ||
7013                     DDI_FM_ERRCB_CAP(softs->fm_capabilities)) {
7014                         pci_ereport_teardown(softs->devinfo_p);
7015                 }

7017                 /* Unregister from IO Fault Services */
7018                 ddi_fm_fini(softs->devinfo_p);

7018                 /* Adjust access and dma attributes for FMA */
7019                 softs->reg_attr.devacc_attr_access = DDI_DEFAULT_ACC;
7020                 softs->addr_dma_attr.dma_attr_flags &= ~DDI_DMA_FLAGERR;
7021                 softs->buf_dma_attr.dma_attr_flags &= ~DDI_DMA_FLAGERR;
7019         }
7020 }

7022 int
7023 aac_check_acc_handle(ddi_acc_handle_t handle)
7024 {
7025         ddi_fm_error_t de;

7027         if (!aac_fm_valid)
7028                 return (DDI_SUCCESS);

7030         ddi_fm_acc_err_get(handle, &de, DDI_FME_VERSION);
7031         return (de.fme_status);
7032 }

7034 int
7035 aac_check_dma_handle(ddi_dma_handle_t handle)
7036 {
7037         ddi_fm_error_t de;

7039         if (!aac_fm_valid)
7040                 return (DDI_SUCCESS);

7042         ddi_fm_dma_err_get(handle, &de, DDI_FME_VERSION);
7043         return (de.fme_status);
7044 }

7046 void
7047 aac_fm_ereport(struct aac_softstate *softs, char *detail, int impact)
```



```
7044 aac_fm_ereport(struct aac_softstate *softs, char *detail)
7048 {
7049         uint64_t ena;
7050         char buf[FM_MAX_CLASS];

7052         if (!aac_fm_valid)
7053                 return;

7055         (void) snprintf(buf, FM_MAX_CLASS, "%s.%s", DDI_FM_DEVICE, detail);
7056         ena = fm_ena_generate(0, FM_ENA_FMT1);
7057         if (DDI_FM_EREPORT_CAP(softs->fm_capabilities)) {
7058                 ddi_fm_ereport_post(softs->devinfo_p, buf, ena, DDI_NOSLEEP,
7059                     FM_VERSION, DATA_TYPE_UINT8, FM_EREPORT_VERSION, NULL);
7060         }
7061         ddi_fm_service_impact(softs->devinfo_p, impact);
7062 }
______unchanged_portion_omitted_

7109 static dev_info_t *
7110 aac_find_child(struct aac_softstate *softs, int tgt, int lun)
7103 aac_find_child(struct aac_softstate *softs, uint16_t tgt, uint8_t lun)
7111 {
7105         dev_info_t *child = NULL;
7112         char addr[SCSI_MAXNAMELEN];
7113         dev_info_t *child;
7114         char *tmp;
7107         char tmp[MAXNAMELEN];

7116         if (tgt < AAC_MAX_LD) {
7110                 if (lun == 0) {
7117                 struct aac_device *dvp = &softs->containers[tgt].dev;

7119                 if (lun == 0 && dvp->valid)
7120                         return (dvp->dip);
7113                         child = dvp->dip;
7114                 }
7121         } else {
7122                 (void) sprintf(addr, "%x,%x", tgt, lun);
7123                 for (child = ddi_get_child(softs->devinfo_p);
7124                     child; child = ddi_get_next_sibling(child)) {
7125                         if ((tmp = ddi_get_name_addr(child)) != NULL &&
7126                             strcmp(addr, tmp) == 0)
7127                                 return (child);
7119                         /* We don't care about non-persistent node */
7120                         if (ndi_dev_is_persistent_node(child) == 0)
7121                                 continue;

7123                         if (aac_name_node(child, tmp, MAXNAMELEN) !=
7124                             DDI_SUCCESS)
7125                                 continue;
7126                         if (strcmp(addr, tmp) == 0)
7127                                 break;
7128                 }
7129         }
7130         return (NULL);
7130         return (child);
7131 }

7133 static int
7134 aac_config_child(struct aac_softstate *softs, struct scsi_device *sd,
7135     dev_info_t **dipp)
7136 {
7137         char *nodename = NULL;
7138         char **compatible = NULL;
7139         int ncompatible = 0;
7140         char *childname;
```

```
7140         dev_info_t *ldip = NULL;
7141         int tgt = sd->sd_address.a_target;
7142         int lun = sd->sd_address.a_lun;
7143         int dtype = sd->sd_inq->inq_dtype & DTYPE_MASK;
7144         int rval;

7146         DBCALLED(softs, 2);

7148         scsi_hba_nodename_compatible_get(sd->sd_inq, NULL, dtype,
7149             NULL, &nodename, &compatible, &ncompatible);
7150         if (nodename == NULL) {
7151                 AACDB_PRINT(softs, CE_WARN,
7152                     "found no compatible driver for t%dL%d", tgt, lun);
7153                     "found no comptible driver for t%dL%d", tgt, lun);
7153                 rval = NDI_FAILURE;
7154                 goto finish;
7155         }
7156         if (softs->legacy && dtype == DTYPE_DIRECT)
7157                 nodename = "sd";
7157         childname = (softs->legacy && dtype == DTYPE_DIRECT) ? "sd" : nodename;

7159         /* Create dev node */
7160         rval = ndi_devi_alloc(softs->devinfo_p, nodename, DEVI_SID_NODEID,
7160         rval = ndi_devi_alloc(softs->devinfo_p, childname, DEVI_SID_NODEID,
7161             &ldip);
7162         if (rval == NDI_SUCCESS) {
7163                 if (ndi_prop_update_int(DDI_DEV_T_NONE, ldip, "target", tgt)
7164                     != DDI_PROP_SUCCESS) {
7165                         AACDB_PRINT(softs, CE_WARN, "unable to create "
7166                             "property for t%dL%d (target)", tgt, lun);
7167                         rval = NDI_FAILURE;
7168                         goto finish;
7169                 }
7170                 if (ndi_prop_update_int(DDI_DEV_T_NONE, ldip, "lun", lun)
7171                     != DDI_PROP_SUCCESS) {
7172                         AACDB_PRINT(softs, CE_WARN, "unable to create "
7173                             "property for t%dL%d (lun)", tgt, lun);
7174                         rval = NDI_FAILURE;
7175                         goto finish;
7176                 }
7177                 if (ndi_prop_update_string_array(DDI_DEV_T_NONE, ldip,
7178                     "compatible", compatible, ncompatible)
7179                     != DDI_PROP_SUCCESS) {
7180                         AACDB_PRINT(softs, CE_WARN, "unable to create "
7181                             "property for t%dL%d (compatible)", tgt, lun);
7182                         rval = NDI_FAILURE;
7183                         goto finish;
7184                 }

7186                 rval = ndi_devi_online(ldip, NDI_ONLINE_ATTACH);
7187                 if (rval != NDI_SUCCESS) {
7188                         AACDB_PRINT(softs, CE_WARN, "unable to online t%dL%d",
7189                             tgt, lun);
7190                         ndi_prop_remove_all(ldip);
7191                         (void) ndi_devi_free(ldip);
7192                 }
7193         }
7194 finish:
7195         if (dipp)
7196                 *dipp = ldip;

7198         scsi_hba_nodename_compatible_free(nodename, compatible);
7197         return (rval);
7198 }

7200 /*ARGSUSED*/
```



```
7201 static int
7202 aac_probe_lun(struct aac_softstate *softs, struct scsi_device *sd)
7203 {
7204         int tgt = sd->sd_address.a_target;
7205         int lun = sd->sd_address.a_lun;

7207         DBCALLED(softs, 2);

7209         if (tgt < AAC_MAX_LD && lun != 0)
7211         if (tgt < AAC_MAX_LD) {
7212                 enum aac_cfg_event event;

7214                 if (lun == 0) {
7215                         mutex_enter(&softs->io_lock);
7216                         event = aac_probe_container(softs, tgt);
7217                         mutex_exit(&softs->io_lock);
7218                         if ((event != AAC_CFG_NULL_NOEXIST) &&
7219                             (event != AAC_CFG_DELETE)) {
7220                                 if (scsi_hba_probe(sd, NULL) ==
7221                                     SCSIPROBE_EXISTS)
7222                                         return (NDI_SUCCESS);
7223                         }
7224                 }
7210                 return (NDI_FAILURE);
7226         } else {
7227                 int dtype;
7228                 int qual; /* device qualifier */

7212         if (scsi_hba_probe(sd, NULL) != SCSIPROBE_EXISTS)
7213                 return (NDI_FAILURE);
7214         if (tgt >= AAC_MAX_LD) {
7215                 int dtype = sd->sd_inq->inq_dtype & DTYPE_MASK;

7233                 dtype = sd->sd_inq->inq_dtype & DTYPE_MASK;
7234                 qual = dtype >> 5;

7217                 AACDB_PRINT(softs, CE_NOTE,
7218                     "Phys. device found: tgt %d dtype %d: %s",
7219                     tgt, dtype, sd->sd_inq->inq_vid);

7221                 /* Only non-DASD and JBOD devices exposed */
7222                 if (dtype != DTYPE_RODIRECT /* CDROM */ &&
7223                     dtype != DTYPE_SEQUENTIAL /* TYPE */ &&
7224                     dtype != DTYPE_ESI /* SES */ &&
7225                         !(dtype == DTYPE_DIRECT &&
7226                          (softs->aac_feature_bits & AAC_SUPPL_SUPPORTED_JBOD)))
7240                 /* Only non-DASD and JBOD mode DASD are allowed exposed */
7241                 if (dtype == DTYPE_RODIRECT /* CDROM */ ||
7242                     dtype == DTYPE_SEQUENTIAL /* TAPE */ ||
7243                     dtype == DTYPE_ESI /* SES */) {
7244                         if (!(softs->flags & AAC_FLAGS_NONDASD))
7227                         return (NDI_FAILURE);
7228                 AACDB_PRINT(softs, CE_NOTE, "non-DASD/JBOD %d found", tgt);
7246                         AACDB_PRINT(softs, CE_NOTE, "non-DASD %d found", tgt);

7248                 } else if (dtype == DTYPE_DIRECT) {
7249                         if (!(softs->flags & AAC_FLAGS_JBOD) || qual != 0)
7250                                 return (NDI_FAILURE);
7251                         AACDB_PRINT(softs, CE_NOTE, "JBOD DASD %d found", tgt);
7229         }

7254                 mutex_enter(&softs->io_lock);
7255                 softs->nondasds[AAC_PD(tgt)].dev.flags |= AAC_DFLAG_VALID;
7256                 mutex_exit(&softs->io_lock);
7230         return (NDI_SUCCESS);
7258         }
```

```
}

static int
aac_config_lun(struct aac_softstate *softs, int tgt, int lun, dev_info_t **ldip)
aac_config_lun(struct aac_softstate *softs, uint16_t tgt, uint8_t lun,
    dev_info_t **ldip)
{
        struct scsi_device sd;
        dev_info_t *child;
        int rval;

        DBCALLED(softs, 2);

        if ((child = aac_find_child(softs, tgt, lun)) != NULL) {
                if (ldip)
                        *ldip = child;
                return (NDI_SUCCESS);
        }

        bzero(&sd, sizeof (struct scsi_device));
        sd.sd_address.a_hba_tran = softs->hba_tran;
        sd.sd_address.a_target = (uint16_t)tgt;
        sd.sd_address.a_lun = (uint8_t)lun;
        if ((rval = aac_probe_lun(softs, &sd)) == NDI_SUCCESS)
                rval = aac_config_child(softs, &sd, ldip);
        scsi_unprobe(&sd);
        /* scsi_unprobe is blank now. Free buffer manually */
        if (sd.sd_inq) {
                kmem_free(sd.sd_inq, SUN_INQSIZE);
                sd.sd_inq = (struct scsi_inquiry *)NULL;
        }
        return (rval);
}

static int
aac_config_tgt(struct aac_softstate *softs, int tgt)
{
        struct scsi_address ap;
        struct buf *bp = NULL;
        int list_len = 8;
        int buf_len = AAC_SCSI_RPTLUNS_HEAD_SIZE + AAC_SCSI_RPTLUNS_ADDR_SIZE;
        int list_len = 0;
        int lun_total = 0;
        dev_info_t *ldip;
        int i, cmd_ok = 1;
        int i;

        ap.a_hba_tran = softs->hba_tran;
        ap.a_target = (uint16_t)tgt;
        ap.a_lun = 0;

        for (i = 0; i < 2; i++) {
                struct scsi_pkt *pkt;
                uchar_t *cdb;
                uchar_t *p;
                int buf_len;
                uint32_t data;

                if (bp == NULL) {
                        buf_len = AAC_SCSI_RPTLUNS_HEAD_SIZE + list_len;
                        if ((bp = scsi_alloc_consistent_buf(&ap, NULL,
                            buf_len, B_READ, NULL_FUNC, NULL)) == NULL)
                        return (AACERR);
                }
                if ((pkt = scsi_init_pkt(&ap, NULL, bp, CDB_GROUP5,
                    sizeof (struct scsi_arq_status), 0, PKT_CONSISTENT,
```



```
                    NULL, NULL)) == NULL) {
                        scsi_free_consistent_buf(bp);
                        return (AACERR);
                }
                cdb = pkt->pkt_cdbp;
                bzero(cdb, CDB_GROUP5);
                cdb[0] = SCMD_REPORT_LUNS;

                /* Convert buffer len from local to LE_32 */
                data = buf_len;
                for (p = &cdb[9]; p > &cdb[5]; p--) {
                        *p = data & 0xff;
                        data >>= 8;
                }

                if (scsi_poll(pkt) < 0 ||
                    ((struct scsi_status *)pkt->pkt_scbp)->sts_chk) {
                        scsi_destroy_pkt(pkt);
                        cmd_ok = 0;
                        break;
                }
                scsi_destroy_pkt(pkt);

                /* Convert list_len from LE_32 to local */
                for (p = (uchar_t *)bp->b_un.b_addr;
                    p < (uchar_t *)bp->b_un.b_addr + 4; p++) {
                        data <<= 8;
                        data |= *p;
                }

                if (data <= list_len)
                        break;
                if (i == 0) {
                        list_len = data;
                if (buf_len < list_len + AAC_SCSI_RPTLUNS_HEAD_SIZE) {
                        scsi_free_consistent_buf(bp);
                        bp = NULL;
                        buf_len = list_len + AAC_SCSI_RPTLUNS_HEAD_SIZE;
                }
                scsi_destroy_pkt(pkt);
        }
        if (i >= 2) {
                uint8_t *buf = (uint8_t *)(bp->b_un.b_addr +
                    AAC_SCSI_RPTLUNS_HEAD_SIZE);

        if (cmd_ok) {
                char *buf = bp->b_un.b_addr + 8;

                for (i = 0; i < (list_len / AAC_SCSI_RPTLUNS_ADDR_SIZE); i++) {
                        int lun;
                        uint16_t lun;

                        /* Determine report luns addressing type */
                        switch (buf[i] & AAC_SCSI_RPTLUNS_ADDR_MASK) {
                        switch (buf[0] & AAC_SCSI_RPTLUNS_ADDR_MASK) {
                        /*
                         * Vendors in the field have been found to be
                         * concatenating bus/target/lun to equal the
                         * complete lun value instead of switching to
                         * flat space addressing
                         */
                        case AAC_SCSI_RPTLUNS_ADDR_PERIPHERAL:
                        case AAC_SCSI_RPTLUNS_ADDR_LOGICAL_UNIT:
                        case AAC_SCSI_RPTLUNS_ADDR_FLAT_SPACE:
                                lun = (buf[i] & 0x3f) << 8;
                                lun |= buf[i + 1];
```

```
7372                                lun = ((buf[0] & 0x3f) << 8) | buf[1];
7373                                if (lun > UINT8_MAX) {
7374                                        AACDB_PRINT(softs, CE_WARN,
7375                                            "abnormal lun number: %d", lun);
7376                                        break;
7377                                }
7345                                if (aac_config_lun(softs, tgt, lun, &ldip) ==
7346                                    NDI_SUCCESS)
7347                                        lun_total++;
7348                                break;
7349                        }

7384                        buf += AAC_SCSI_RPTLUNS_ADDR_SIZE;
7350                }
7351        } else {
7352                /* The target may do not support SCMD_REPORT_LUNS. */
7353                if (aac_config_lun(softs, tgt, 0, &ldip) == NDI_SUCCESS)
7354                        lun_total++;
7355        }
7356        scsi_free_consistent_buf(bp);
7357        return (lun_total);
7358 }

7360 static void
7361 aac_enable_pd(struct aac_softstate *softs, int tgt, int en)
7396 aac_devcfg(struct aac_softstate *softs, int tgt, int en)
7362 {
7363        if (tgt >= AAC_MAX_LD) {
7398        struct aac_device *dvp;

7364                mutex_enter(&softs->io_lock);
7365                softs->nondasds[AAC_PD(tgt)].dev.valid = (uint8_t)en;
7401        dvp = AAC_DEV(softs, tgt);
7402        if (en)
7403                dvp->flags |= AAC_DFLAG_CONFIGURING;
7404        else
7405                dvp->flags &= ~AAC_DFLAG_CONFIGURING;
7366                mutex_exit(&softs->io_lock);
7367        }
7368 }

7370 static void
7371 aac_config_pd(void *arg)
7372 {
7373        struct aac_softstate *softs = (struct aac_softstate *)arg;
7374        uint32_t bus, tgt;
7375        int index, total;

7377        total = 0;
7378        index = AAC_MAX_LD;
7379        for (bus = 0; bus < softs->bus_max; bus++) {
7380                AACDB_PRINT(softs, CE_NOTE, "bus %d:", bus);
7381                for (tgt = 0; tgt < softs->tgt_max; tgt++, index++) {
7382                        aac_enable_pd(softs, index, 1);
7383                        if (aac_config_tgt(softs, index) == 0)
7384                                aac_enable_pd(softs, index, 0);
7385                        else
7386                                total++;
7387                }
7388        }
7389        AACDB_PRINT(softs, CE_CONT,
7390                "Total %d phys. device(s) found", total);
7391 }

7393 static int
7394 aac_tran_bus_config(dev_info_t *parent, uint_t flags, ddi_bus_config_op_t op,
```



```
7395     void *arg, dev_info_t **childp)
7396 {
7397        struct aac_softstate *softs;
7398        int circ = 0;
7399        int rval;

7401        if ((softs = ddi_get_soft_state(aac_softstatep,
7402            ddi_get_instance(parent))) == NULL)
7403                return (NDI_FAILURE);

7421        /* Commands for bus config should be blocked as the bus is quiesced */
7422        mutex_enter(&softs->io_lock);
7423        if (softs->state & AAC_STATE_QUIESCED) {
7424                AACDB_PRINT(softs, CE_NOTE,
7425                    "bus_config abroted because bus is quiesced");
7426                mutex_exit(&softs->io_lock);
7427                return (NDI_FAILURE);
7428        }
7429        mutex_exit(&softs->io_lock);

7405        DBCALLED(softs, 1);

7407        /* Hold the nexus across the bus_config */
7408        ndi_devi_enter(parent, &circ);
7409        switch (op) {
7410        case BUS_CONFIG_ONE: {
7411                int tgt, lun;
7412                struct scsi_device sd;

7414                if (aac_parse_devname(arg, &tgt, &lun) != AACOK) {
7415                        rval = NDI_FAILURE;
7416                        break;
7417                }
7418                if (*childp = aac_find_child(softs, tgt, lun)) {
7419                        rval = NDI_SUCCESS;
7443                if (tgt >= AAC_MAX_LD) {
7444                        if (tgt >= AAC_MAX_DEV(softs)) {
7445                                rval = NDI_FAILURE;
7420                        break;
7421                }
7448                }

7423                aac_enable_pd(softs, tgt, 1);
7424                bzero(&sd, sizeof (struct scsi_device));
7425                sd.sd_address.a_hba_tran = softs->hba_tran;
7426                sd.sd_address.a_target = (uint16_t)tgt;
7427                sd.sd_address.a_lun = (uint8_t)lun;
7428                if ((rval = aac_probe_lun(softs, &sd)) == NDI_SUCCESS)
7429                        rval = aac_config_child(softs, &sd, childp);
7430                else
7431                        aac_enable_pd(softs, tgt, 0);
7432                scsi_unprobe(&sd);
7450                AAC_DEVCFG_BEGIN(softs, tgt);
7451                rval = aac_config_lun(softs, tgt, lun, childp);
7452                AAC_DEVCFG_END(softs, tgt);
7433                break;
7434        }

7436        case BUS_CONFIG_DRIVER:
7437        case BUS_CONFIG_ALL: {
7438                uint32_t tgt;
7458                uint32_t bus, tgt;
7459                int index, total;

7440                for (tgt = 0; tgt < AAC_MAX_LD; tgt++)
7461                for (tgt = 0; tgt < AAC_MAX_LD; tgt++) {
```

```
7462                        AAC_DEVCFG_BEGIN(softs, tgt);
7441                        (void) aac_config_lun(softs, tgt, 0, NULL);
7464                        AAC_DEVCFG_END(softs, tgt);
7465                }

7443                /* Config the non-DASD devices connected to the card */
7444                aac_config_pd((void *)softs);
7468                total = 0;
7469                index = AAC_MAX_LD;
7470                for (bus = 0; bus < softs->bus_max; bus++) {
7471                        AACDB_PRINT(softs, CE_NOTE, "bus %d:", bus);
7472                        for (tgt = 0; tgt < softs->tgt_max; tgt++, index++) {
7473                                AAC_DEVCFG_BEGIN(softs, index);
7474                                if (aac_config_tgt(softs, index))
7475                                        total++;
7476                                AAC_DEVCFG_END(softs, index);
7477                        }
7478                }
7479                AACDB_PRINT(softs, CE_CONT,
7480                    "?Total %d phys. device(s) found", total);
7445                rval = NDI_SUCCESS;
7446                break;
7447        }
7448        }

7450        if (rval == NDI_SUCCESS)
7451                rval = ndi_busop_bus_config(parent, flags, op, arg, childp, 0);
7452        ndi_devi_exit(parent, circ);
7453        return (rval);
7454 }

7456 static void
7457 aac_handle_dr(struct aac_drinfo *drp)
7492 /*ARGSUSED*/
7493 static int
7494 aac_handle_dr(struct aac_softstate *softs, int tgt, int lun, int event)
7458 {
7459        struct aac_softstate *softs = drp->softs;
7460        struct aac_device *dvp;
7461        dev_info_t *dip;
7462        int valid;
7463        int circ1 = 0;

7465        DBCALLED(softs, 1);

7467        /* Hold the nexus across the bus_config */
7468        mutex_enter(&softs->io_lock);
7469        dvp = AAC_DEV(softs, drp->tgt);
7504        dvp = AAC_DEV(softs, tgt);
7505        valid = AAC_DEV_IS_VALID(dvp);
7470        dip = dvp->dip;
7471        valid = dvp->valid;
7507        if (!(softs->state & AAC_STATE_RUN))
7508                return (AACERR);
7472        mutex_exit(&softs->io_lock);

7474        switch (drp->event) {
7475        case AAC_DEV_ONLINE:
7476        case AAC_DEV_OFFLINE:
7511        switch (event) {
7512        case AAC_CFG_ADD:
7513        case AAC_CFG_DELETE:
7477                /* Device onlined */
7478                if (dip == NULL && valid) {
7479                        ndi_devi_enter(softs->devinfo_p, &circ1);
7480                        (void) aac_config_lun(softs, drp->tgt, 0, NULL);
```



```
7517                        (void) aac_config_lun(softs, tgt, 0, NULL);
7481                        AACDB_PRINT(softs, CE_NOTE, "c%dt%dL%d onlined",
7482                            softs->instance, drp->tgt, drp->lun);
7519                            softs->instance, tgt, lun);
7483                        ndi_devi_exit(softs->devinfo_p, circ1);
7484                }
7485                /* Device offlined */
7486                if (dip && valid == 0) {
7523                if (dip && !valid) {
7487                        mutex_enter(&softs->io_lock);
7488                        (void) aac_do_reset(softs);
7489                        mutex_exit(&softs->io_lock);

7491                        (void) ndi_devi_offline(dip, NDI_DEVI_REMOVE);
7492                        AACDB_PRINT(softs, CE_NOTE, "c%dt%dL%d offlined",
7493                            softs->instance, drp->tgt, drp->lun);
7530                            softs->instance, tgt, lun);
7494                }
7495                break;
7496        }
7497        kmem_free(drp, sizeof (struct aac_drinfo));
7498 }

7500 static int
7501 aac_dr_event(struct aac_softstate *softs, int tgt, int lun, int event)
7502 {
7503        struct aac_drinfo *drp;

7505        DBCALLED(softs, 1);

7507        if (softs->taskq == NULL ||
7508            (drp = kmem_zalloc(sizeof (struct aac_drinfo), KM_NOSLEEP)) == NULL)
7509                return (AACERR);

7511        drp->softs = softs;
7512        drp->tgt = tgt;
7513        drp->lun = lun;
7514        drp->event = event;
7515        if ((ddi_taskq_dispatch(softs->taskq, (void (*)(void *))aac_handle_dr,
7516            drp, DDI_NOSLEEP)) != DDI_SUCCESS) {
7517                AACDB_PRINT(softs, CE_WARN, "DR task start failed");
7518                kmem_free(drp, sizeof (struct aac_drinfo));
7519                return (AACERR);
7520        }
7535        mutex_enter(&softs->io_lock);
7521        return (AACOK);
7522 }

7524 #ifdef AAC_DEBUG_ALL
7539 #ifdef DEBUG

7526 /* -------------------------debug aid functions-------------------------- */

7528 #define AAC_FIB_CMD_KEY_STRINGS \
7529        TestCommandResponse, "TestCommandResponse", \
7530        TestAdapterCommand, "TestAdapterCommand", \
7531        LastTestCommand, "LastTestCommand", \
7532        ReinitHostNormCommandQueue, "ReinitHostNormCommandQueue", \
7533        ReinitHostHighCommandQueue, "ReinitHostHighCommandQueue", \
7534        ReinitHostHighRespQueue, "ReinitHostHighRespQueue", \
7535        ReinitHostNormRespQueue, "ReinitHostNormRespQueue", \
7536        ReinitAdapNormCommandQueue, "ReinitAdapNormCommandQueue", \
7537        ReinitAdapHighCommandQueue, "ReinitAdapHighCommandQueue", \
7538        ReinitAdapHighRespQueue, "ReinitAdapHighRespQueue", \
7539        ReinitAdapNormRespQueue, "ReinitAdapNormRespQueue", \
7540        InterfaceShutdown, "InterfaceShutdown", \
```

```
7541         DmaCommandFib, "DmaCommandFib", \
7542         StartProfile, "StartProfile", \
7543         TermProfile, "TermProfile", \
7544         SpeedTest, "SpeedTest", \
7545         TakeABreakPt, "TakeABreakPt", \
7546         RequestPerfData, "RequestPerfData", \
7547         SetInterruptDefTimer, "SetInterruptDefTimer", \
7548         SetInterruptDefCount, "SetInterruptDefCount", \
7549         GetInterruptDefStatus, "GetInterruptDefStatus", \
7550         LastCommCommand, "LastCommCommand", \
7551         NuFileSystem, "NuFileSystem", \
7552         UFS, "UFS", \
7553         HostFileSystem, "HostFileSystem", \
7554         LastFileSystemCommand, "LastFileSystemCommand", \
7555         ContainerCommand, "ContainerCommand", \
7556         ContainerCommand64, "ContainerCommand64", \
7557         ClusterCommand, "ClusterCommand", \
7558         ScsiPortCommand, "ScsiPortCommand", \
7559         ScsiPortCommandU64, "ScsiPortCommandU64", \
7560         AifRequest, "AifRequest", \
7561         CheckRevision, "CheckRevision", \
7562         FsaHostShutdown, "FsaHostShutdown", \
7563         RequestAdapterInfo, "RequestAdapterInfo", \
7564         IsAdapterPaused, "IsAdapterPaused", \
7565         SendHostTime, "SendHostTime", \
7566         LastMiscCommand, "LastMiscCommand"

7568 #define AAC_CTVM_SUBCMD_KEY_STRINGS \
7569         VM_Null, "VM_Null", \
7570         VM_NameServe, "VM_NameServe", \
7571         VM_ContainerConfig, "VM_ContainerConfig", \
7572         VM_Ioctl, "VM_Ioctl", \
7573         VM_FilesystemIoctl, "VM_FilesystemIoctl", \
7574         VM_CloseAll, "VM_CloseAll", \
7575         VM_CtBlockRead, "VM_CtBlockRead", \
7576         VM_CtBlockWrite, "VM_CtBlockWrite", \
7577         VM_SliceBlockRead, "VM_SliceBlockRead", \
7578         VM_SliceBlockWrite, "VM_SliceBlockWrite", \
7579         VM_DriveBlockRead, "VM_DriveBlockRead", \
7580         VM_DriveBlockWrite, "VM_DriveBlockWrite", \
7581         VM_EnclosureMgt, "VM_EnclosureMgt", \
7582         VM_Unused, "VM_Unused", \
7583         VM_CtBlockVerify, "VM_CtBlockVerify", \
7584         VM_CtPerf, "VM_CtPerf", \
7585         VM_CtBlockRead64, "VM_CtBlockRead64", \
7586         VM_CtBlockWrite64, "VM_CtBlockWrite64", \
7587         VM_CtBlockVerify64, "VM_CtBlockVerify64", \
7588         VM_CtHostRead64, "VM_CtHostRead64", \
7589         VM_CtHostWrite64, "VM_CtHostWrite64", \
7590         VM_NameServe64, "VM_NameServe64"

7592 #define AAC_CT_SUBCMD_KEY_STRINGS \
7593         CT_Null, "CT_Null", \
7594         CT_GET_SLICE_COUNT, "CT_GET_SLICE_COUNT", \
7595         CT_GET_PARTITION_COUNT, "CT_GET_PARTITION_COUNT", \
7596         CT_GET_PARTITION_INFO, "CT_GET_PARTITION_INFO", \
7597         CT_GET_CONTAINER_COUNT, "CT_GET_CONTAINER_COUNT", \
7598         CT_GET_CONTAINER_INFO_OLD, "CT_GET_CONTAINER_INFO_OLD", \
7599         CT_WRITE_MBR, "CT_WRITE_MBR", \
7600         CT_WRITE_PARTITION, "CT_WRITE_PARTITION", \
7601         CT_UPDATE_PARTITION, "CT_UPDATE_PARTITION", \
7602         CT_UNLOAD_CONTAINER, "CT_UNLOAD_CONTAINER", \
7603         CT_CONFIG_SINGLE_PRIMARY, "CT_CONFIG_SINGLE_PRIMARY", \
7604         CT_READ_CONFIG_AGE, "CT_READ_CONFIG_AGE", \
7605         CT_WRITE_CONFIG_AGE, "CT_WRITE_CONFIG_AGE", \
7606         CT_READ_SERIAL_NUMBER, "CT_READ_SERIAL_NUMBER", \
```



```
7607         CT_ZERO_PAR_ENTRY, "CT_ZERO_PAR_ENTRY", \
7608         CT_READ_MBR, "CT_READ_MBR", \
7609         CT_READ_PARTITION, "CT_READ_PARTITION", \
7610         CT_DESTROY_CONTAINER, "CT_DESTROY_CONTAINER", \
7611         CT_DESTROY2_CONTAINER, "CT_DESTROY2_CONTAINER", \
7612         CT_SLICE_SIZE, "CT_SLICE_SIZE", \
7613         CT_CHECK_CONFLICTS, "CT_CHECK_CONFLICTS", \
7614         CT_MOVE_CONTAINER, "CT_MOVE_CONTAINER", \
7615         CT_READ_LAST_DRIVE, "CT_READ_LAST_DRIVE", \
7616         CT_WRITE_LAST_DRIVE, "CT_WRITE_LAST_DRIVE", \
7617         CT_UNMIRROR, "CT_UNMIRROR", \
7618         CT_MIRROR_DELAY, "CT_MIRROR_DELAY", \
7619         CT_GEN_MIRROR, "CT_GEN_MIRROR", \
7620         CT_GEN_MIRROR2, "CT_GEN_MIRROR2", \
7621         CT_TEST_CONTAINER, "CT_TEST_CONTAINER", \
7622         CT_MOVE2, "CT_MOVE2", \
7623         CT_SPLIT, "CT_SPLIT", \
7624         CT_SPLIT2, "CT_SPLIT2", \
7625         CT_SPLIT_BROKEN, "CT_SPLIT_BROKEN", \
7626         CT_SPLIT_BROKEN2, "CT_SPLIT_BROKEN2", \
7627         CT_RECONFIG, "CT_RECONFIG", \
7628         CT_BREAK2, "CT_BREAK2", \
7629         CT_BREAK, "CT_BREAK", \
7630         CT_MERGE2, "CT_MERGE2", \
7631         CT_MERGE, "CT_MERGE", \
7632         CT_FORCE_ERROR, "CT_FORCE_ERROR", \
7633         CT_CLEAR_ERROR, "CT_CLEAR_ERROR", \
7634         CT_ASSIGN_FAILOVER, "CT_ASSIGN_FAILOVER", \
7635         CT_CLEAR_FAILOVER, "CT_CLEAR_FAILOVER", \
7636         CT_GET_FAILOVER_DATA, "CT_GET_FAILOVER_DATA", \
7637         CT_VOLUME_ADD, "CT_VOLUME_ADD", \
7638         CT_VOLUME_ADD2, "CT_VOLUME_ADD2", \
7639         CT_MIRROR_STATUS, "CT_MIRROR_STATUS", \
7640         CT_COPY_STATUS, "CT_COPY_STATUS", \
7641         CT_COPY, "CT_COPY", \
7642         CT_UNLOCK_CONTAINER, "CT_UNLOCK_CONTAINER", \
7643         CT_LOCK_CONTAINER, "CT_LOCK_CONTAINER", \
7644         CT_MAKE_READ_ONLY, "CT_MAKE_READ_ONLY", \
7645         CT_MAKE_READ_WRITE, "CT_MAKE_READ_WRITE", \
7646         CT_CLEAN_DEAD, "CT_CLEAN_DEAD", \
7647         CT_ABORT_MIRROR_COMMAND, "CT_ABORT_MIRROR_COMMAND", \
7648         CT_SET, "CT_SET", \
7649         CT_GET, "CT_GET", \
7650         CT_GET_NVLOG_ENTRY, "CT_GET_NVLOG_ENTRY", \
7651         CT_GET_DELAY, "CT_GET_DELAY", \
7652         CT_ZERO_CONTAINER_SPACE, "CT_ZERO_CONTAINER_SPACE", \
7653         CT_GET_ZERO_STATUS, "CT_GET_ZERO_STATUS", \
7654         CT_SCRUB, "CT_SCRUB", \
7655         CT_GET_SCRUB_STATUS, "CT_GET_SCRUB_STATUS", \
7656         CT_GET_SLICE_INFO, "CT_GET_SLICE_INFO", \
7657         CT_GET_SCSI_METHOD, "CT_GET_SCSI_METHOD", \
7658         CT_PAUSE_IO, "CT_PAUSE_IO", \
7659         CT_RELEASE_IO, "CT_RELEASE_IO", \
7660         CT_SCRUB2, "CT_SCRUB2", \
7661         CT_MCHECK, "CT_MCHECK", \
7662         CT_CORRUPT, "CT_CORRUPT", \
7663         CT_GET_TASK_COUNT, "CT_GET_TASK_COUNT", \
7664         CT_PROMOTE, "CT_PROMOTE", \
7665         CT_SET_DEAD, "CT_SET_DEAD", \
7666         CT_CONTAINER_OPTIONS, "CT_CONTAINER_OPTIONS", \
7667         CT_GET_NV_PARAM, "CT_GET_NV_PARAM", \
7668         CT_GET_PARAM, "CT_GET_PARAM", \
7669         CT_NV_PARAM_SIZE, "CT_NV_PARAM_SIZE", \
7670         CT_COMMON_PARAM_SIZE, "CT_COMMON_PARAM_SIZE", \
7671         CT_PLATFORM_PARAM_SIZE, "CT_PLATFORM_PARAM_SIZE", \
7672         CT_SET_NV_PARAM, "CT_SET_NV_PARAM", \
```

```
7673         CT_ABORT_SCRUB, "CT_ABORT_SCRUB", \
7674         CT_GET_SCRUB_ERROR, "CT_GET_SCRUB_ERROR", \
7675         CT_LABEL_CONTAINER, "CT_LABEL_CONTAINER", \
7676         CT_CONTINUE_DATA, "CT_CONTINUE_DATA", \
7677         CT_STOP_DATA, "CT_STOP_DATA", \
7678         CT_GET_PARTITION_TABLE, "CT_GET_PARTITION_TABLE", \
7679         CT_GET_DISK_PARTITIONS, "CT_GET_DISK_PARTITIONS", \
7680         CT_GET_MISC_STATUS, "CT_GET_MISC_STATUS", \
7681         CT_GET_CONTAINER_PERF_INFO, "CT_GET_CONTAINER_PERF_INFO", \
7682         CT_GET_TIME, "CT_GET_TIME", \
7683         CT_READ_DATA, "CT_READ_DATA", \
7684         CT_CTR, "CT_CTR", \
7685         CT_CTL, "CT_CTL", \
7686         CT_DRAINIO, "CT_DRAINIO", \
7687         CT_RELEASEIO, "CT_RELEASEIO", \
7688         CT_GET_NVRAM, "CT_GET_NVRAM", \
7689         CT_GET_MEMORY, "CT_GET_MEMORY", \
7690         CT_PRINT_CT_LOG, "CT_PRINT_CT_LOG", \
7691         CT_ADD_LEVEL, "CT_ADD_LEVEL", \
7692         CT_NV_ZERO, "CT_NV_ZERO", \
7693         CT_READ_SIGNATURE, "CT_READ_SIGNATURE", \
7694         CT_THROTTLE_ON, "CT_THROTTLE_ON", \
7695         CT_THROTTLE_OFF, "CT_THROTTLE_OFF", \
7696         CT_GET_THROTTLE_STATS, "CT_GET_THROTTLE_STATS", \
7697         CT_MAKE_SNAPSHOT, "CT_MAKE_SNAPSHOT", \
7698         CT_REMOVE_SNAPSHOT, "CT_REMOVE_SNAPSHOT", \
7699         CT_WRITE_USER_FLAGS, "CT_WRITE_USER_FLAGS", \
7700         CT_READ_USER_FLAGS, "CT_READ_USER_FLAGS", \
7701         CT_MONITOR, "CT_MONITOR", \
7702         CT_GEN_MORPH, "CT_GEN_MORPH", \
7703         CT_GET_SNAPSHOT_INFO, "CT_GET_SNAPSHOT_INFO", \
7704         CT_CACHE_SET, "CT_CACHE_SET", \
7705         CT_CACHE_STAT, "CT_CACHE_STAT", \
7706         CT_TRACE_START, "CT_TRACE_START", \
7707         CT_TRACE_STOP, "CT_TRACE_STOP", \
7708         CT_TRACE_ENABLE, "CT_TRACE_ENABLE", \
7709         CT_TRACE_DISABLE, "CT_TRACE_DISABLE", \
7710         CT_FORCE_CORE_DUMP, "CT_FORCE_CORE_DUMP", \
7711         CT_SET_SERIAL_NUMBER, "CT_SET_SERIAL_NUMBER", \
7712         CT_RESET_SERIAL_NUMBER, "CT_RESET_SERIAL_NUMBER", \
7713         CT_ENABLE_RAID5, "CT_ENABLE_RAID5", \
7714         CT_CLEAR_VALID_DUMP_FLAG, "CT_CLEAR_VALID_DUMP_FLAG", \
7715         CT_GET_MEM_STATS, "CT_GET_MEM_STATS", \
7716         CT_GET_CORE_SIZE, "CT_GET_CORE_SIZE", \
7717         CT_CREATE_CONTAINER_OLD, "CT_CREATE_CONTAINER_OLD", \
7718         CT_STOP_DUMPS, "CT_STOP_DUMPS", \
7719         CT_PANIC_ON_TAKE_A_BREAK, "CT_PANIC_ON_TAKE_A_BREAK", \
7720         CT_GET_CACHE_STATS, "CT_GET_CACHE_STATS", \
7721         CT_MOVE_PARTITION, "CT_MOVE_PARTITION", \
7722         CT_FLUSH_CACHE, "CT_FLUSH_CACHE", \
7723         CT_READ_NAME, "CT_READ_NAME", \
7724         CT_WRITE_NAME, "CT_WRITE_NAME", \
7725         CT_TOSS_CACHE, "CT_TOSS_CACHE", \
7726         CT_LOCK_DRAINIO, "CT_LOCK_DRAINIO", \
7727         CT_CONTAINER_OFFLINE, "CT_CONTAINER_OFFLINE", \
7728         CT_SET_CACHE_SIZE, "CT_SET_CACHE_SIZE", \
7729         CT_CLEAN_SHUTDOWN_STATUS, "CT_CLEAN_SHUTDOWN_STATUS", \
7730         CT_CLEAR_DISKLOG_ON_DISK, "CT_CLEAR_DISKLOG_ON_DISK", \
7731         CT_CLEAR_ALL_DISKLOG, "CT_CLEAR_ALL_DISKLOG", \
7732         CT_CACHE_FAVOR, "CT_CACHE_FAVOR", \
7733         CT_READ_PASSTHRU_MBR, "CT_READ_PASSTHRU_MBR", \
7734         CT_SCRUB_NOFIX, "CT_SCRUB_NOFIX", \
7735         CT_SCRUB2_NOFIX, "CT_SCRUB2_NOFIX", \
7736         CT_FLUSH, "CT_FLUSH", \
7737         CT_REBUILD, "CT_REBUILD", \
7738         CT_FLUSH_CONTAINER, "CT_FLUSH_CONTAINER", \
```



```
7739         CT_RESTART, "CT_RESTART", \
7740         CT_GET_CONFIG_STATUS, "CT_GET_CONFIG_STATUS", \
7741         CT_TRACE_FLAG, "CT_TRACE_FLAG", \
7742         CT_RESTART_MORPH, "CT_RESTART_MORPH", \
7743         CT_GET_TRACE_INFO, "CT_GET_TRACE_INFO", \
7744         CT_GET_TRACE_ITEM, "CT_GET_TRACE_ITEM", \
7745         CT_COMMIT_CONFIG, "CT_COMMIT_CONFIG", \
7746         CT_CONTAINER_EXISTS, "CT_CONTAINER_EXISTS", \
7747         CT_GET_SLICE_FROM_DEVT, "CT_GET_SLICE_FROM_DEVT", \
7748         CT_OPEN_READ_WRITE, "CT_OPEN_READ_WRITE", \
7749         CT_WRITE_MEMORY_BLOCK, "CT_WRITE_MEMORY_BLOCK", \
7750         CT_GET_CACHE_PARAMS, "CT_GET_CACHE_PARAMS", \
7751         CT_CRAZY_CACHE, "CT_CRAZY_CACHE", \
7752         CT_GET_PROFILE_STRUCT, "CT_GET_PROFILE_STRUCT", \
7753         CT_SET_IO_TRACE_FLAG, "CT_SET_IO_TRACE_FLAG", \
7754         CT_GET_IO_TRACE_STRUCT, "CT_GET_IO_TRACE_STRUCT", \
7755         CT_CID_TO_64BITS_UID, "CT_CID_TO_64BITS_UID", \
7756         CT_64BITS_UID_TO_CID, "CT_64BITS_UID_TO_CID", \
7757         CT_PAR_TO_64BITS_UID, "CT_PAR_TO_64BITS_UID", \
7758         CT_CID_TO_32BITS_UID, "CT_CID_TO_32BITS_UID", \
7759         CT_32BITS_UID_TO_CID, "CT_32BITS_UID_TO_CID", \
7760         CT_PAR_TO_32BITS_UID, "CT_PAR_TO_32BITS_UID", \
7761         CT_SET_FAILOVER_OPTION, "CT_SET_FAILOVER_OPTION", \
7762         CT_GET_FAILOVER_OPTION, "CT_GET_FAILOVER_OPTION", \
7763         CT_STRIPE_ADD2, "CT_STRIPE_ADD2", \
7764         CT_CREATE_VOLUME_SET, "CT_CREATE_VOLUME_SET", \
7765         CT_CREATE_STRIPE_SET, "CT_CREATE_STRIPE_SET", \
7766         CT_VERIFY_CONTAINER, "CT_VERIFY_CONTAINER", \
7767         CT_IS_CONTAINER_DEAD, "CT_IS_CONTAINER_DEAD", \
7768         CT_GET_CONTAINER_OPTION, "CT_GET_CONTAINER_OPTION", \
7769         CT_GET_SNAPSHOT_UNUSED_STRUCT, "CT_GET_SNAPSHOT_UNUSED_STRUCT", \
7770         CT_CLEAR_SNAPSHOT_UNUSED_STRUCT, "CT_CLEAR_SNAPSHOT_UNUSED_STRUCT", \
7771         CT_GET_CONTAINER_INFO, "CT_GET_CONTAINER_INFO", \
7772         CT_CREATE_CONTAINER, "CT_CREATE_CONTAINER", \
7773         CT_CHANGE_CREATIONINFO, "CT_CHANGE_CREATIONINFO", \
7774         CT_CHECK_CONFLICT_UID, "CT_CHECK_CONFLICT_UID", \
7775         CT_CONTAINER_UID_CHECK, "CT_CONTAINER_UID_CHECK", \
7776         CT_IS_CONTAINER_MEATADATA_STANDARD, \
7777             "CT_IS_CONTAINER_MEATADATA_STANDARD", \
7778         CT_IS_SLICE_METADATA_STANDARD, "CT_IS_SLICE_METADATA_STANDARD", \
7779         CT_GET_IMPORT_COUNT, "CT_GET_IMPORT_COUNT", \
7780         CT_CANCEL_ALL_IMPORTS, "CT_CANCEL_ALL_IMPORTS", \
7781         CT_GET_IMPORT_INFO, "CT_GET_IMPORT_INFO", \
7782         CT_IMPORT_ARRAY, "CT_IMPORT_ARRAY", \
7783         CT_GET_LOG_SIZE, "CT_GET_LOG_SIZE", \
7784         CT_ALARM_GET_STATE, "CT_ALARM_GET_STATE", \
7785         CT_ALARM_SET_STATE, "CT_ALARM_SET_STATE", \
7786         CT_ALARM_ON_OFF, "CT_ALARM_ON_OFF", \
7787         CT_GET_EE_OEM_ID, "CT_GET_EE_OEM_ID", \
7788         CT_GET_PPI_HEADERS, "CT_GET_PPI_HEADERS", \
7789         CT_GET_PPI_DATA, "CT_GET_PPI_DATA", \
7790         CT_GET_PPI_ENTRIES, "CT_GET_PPI_ENTRIES", \
7791         CT_DELETE_PPI_BUNDLE, "CT_DELETE_PPI_BUNDLE", \
7792         CT_GET_PARTITION_TABLE_2, "CT_GET_PARTITION_TABLE_2", \
7793         CT_GET_PARTITION_INFO_2, "CT_GET_PARTITION_INFO_2", \
7794         CT_GET_DISK_PARTITIONS_2, "CT_GET_DISK_PARTITIONS_2", \
7795         CT_QUIESCE_ADAPTER, "CT_QUIESCE_ADAPTER", \
7796         CT_CLEAR_PPI_TABLE, "CT_CLEAR_PPI_TABLE"

7798 #define AAC_CL_SUBCMD_KEY_STRINGS \
7799         CL_NULL, "CL_NULL", \
7800         DS_INIT, "DS_INIT", \
7801         DS_RESCAN, "DS_RESCAN", \
7802         DS_CREATE, "DS_CREATE", \
7803         DS_DELETE, "DS_DELETE", \
7804         DS_ADD_DISK, "DS_ADD_DISK", \
```

```
7805         DS_REMOVE_DISK, "DS_REMOVE_DISK", \
7806         DS_MOVE_DISK, "DS_MOVE_DISK", \
7807         DS_TAKE_OWNERSHIP, "DS_TAKE_OWNERSHIP", \
7808         DS_RELEASE_OWNERSHIP, "DS_RELEASE_OWNERSHIP", \
7809         DS_FORCE_OWNERSHIP, "DS_FORCE_OWNERSHIP", \
7810         DS_GET_DISK_SET_PARAM, "DS_GET_DISK_SET_PARAM", \
7811         DS_GET_DRIVE_PARAM, "DS_GET_DRIVE_PARAM", \
7812         DS_GET_SLICE_PARAM, "DS_GET_SLICE_PARAM", \
7813         DS_GET_DISK_SETS, "DS_GET_DISK_SETS", \
7814         DS_GET_DRIVES, "DS_GET_DRIVES", \
7815         DS_SET_DISK_SET_PARAM, "DS_SET_DISK_SET_PARAM", \
7816         DS_ONLINE, "DS_ONLINE", \
7817         DS_OFFLINE, "DS_OFFLINE", \
7818         DS_ONLINE_CONTAINERS, "DS_ONLINE_CONTAINERS", \
7819         DS_FSAPRINT, "DS_FSAPRINT", \
7820         CL_CFG_SET_HOST_IDS, "CL_CFG_SET_HOST_IDS", \
7821         CL_CFG_SET_PARTNER_HOST_IDS, "CL_CFG_SET_PARTNER_HOST_IDS", \
7822         CL_CFG_GET_CLUSTER_CONFIG, "CL_CFG_GET_CLUSTER_CONFIG", \
7823         CC_CLI_CLEAR_MESSAGE_BUFFER, "CC_CLI_CLEAR_MESSAGE_BUFFER", \
7824         CC_SRV_CLEAR_MESSAGE_BUFFER, "CC_SRV_CLEAR_MESSAGE_BUFFER", \
7825         CC_CLI_SHOW_MESSAGE_BUFFER, "CC_CLI_SHOW_MESSAGE_BUFFER", \
7826         CC_SRV_SHOW_MESSAGE_BUFFER, "CC_SRV_SHOW_MESSAGE_BUFFER", \
7827         CC_CLI_SEND_MESSAGE, "CC_CLI_SEND_MESSAGE", \
7828         CC_SRV_SEND_MESSAGE, "CC_SRV_SEND_MESSAGE", \
7829         CC_CLI_GET_MESSAGE, "CC_CLI_GET_MESSAGE", \
7830         CC_SRV_GET_MESSAGE, "CC_SRV_GET_MESSAGE", \
7831         CC_SEND_TEST_MESSAGE, "CC_SEND_TEST_MESSAGE", \
7832         CC_GET_BUSINFO, "CC_GET_BUSINFO", \
7833         CC_GET_PORTINFO, "CC_GET_PORTINFO", \
7834         CC_GET_NAMEINFO, "CC_GET_NAMEINFO", \
7835         CC_GET_CONFIGINFO, "CC_GET_CONFIGINFO", \
7836         CQ_QUORUM_OP, "CQ_QUORUM_OP"

7838 #define AAC_AIF_SUBCMD_KEY_STRINGS \
7839         AifCmdEventNotify, "AifCmdEventNotify", \
7840         AifCmdJobProgress, "AifCmdJobProgress", \
7841         AifCmdAPIReport, "AifCmdAPIReport", \
7842         AifCmdDriverNotify, "AifCmdDriverNotify", \
7843         AifReqJobList, "AifReqJobList", \
7844         AifReqJobsForCtr, "AifReqJobsForCtr", \
7845         AifReqJobsForScsi, "AifReqJobsForScsi", \
7846         AifReqJobReport, "AifReqJobReport", \
7847         AifReqTerminateJob, "AifReqTerminateJob", \
7848         AifReqSuspendJob, "AifReqSuspendJob", \
7849         AifReqResumeJob, "AifReqResumeJob", \
7850         AifReqSendAPIReport, "AifReqSendAPIReport", \
7851         AifReqAPIJobStart, "AifReqAPIJobStart", \
7852         AifReqAPIJobUpdate, "AifReqAPIJobUpdate", \
7853         AifReqAPIJobFinish, "AifReqAPIJobFinish"

7855 #define AAC_IOCTL_SUBCMD_KEY_STRINGS \
7856         Reserved_IOCTL, "Reserved_IOCTL", \
7857         GetDeviceHandle, "GetDeviceHandle", \
7858         BusTargetLun_to_DeviceHandle, "BusTargetLun_to_DeviceHandle", \
7859         DeviceHandle_to_BusTargetLun, "DeviceHandle_to_BusTargetLun", \
7860         RescanBus, "RescanBus", \
7861         GetDeviceProbeInfo, "GetDeviceProbeInfo", \
7862         GetDeviceCapacity, "GetDeviceCapacity", \
7863         GetContainerProbeInfo, "GetContainerProbeInfo", \
7864         GetRequestedMemorySize, "GetRequestedMemorySize", \
7865         GetBusInfo, "GetBusInfo", \
7866         GetVendorSpecific, "GetVendorSpecific", \
7867         EnhancedGetDeviceProbeInfo, "EnhancedGetDeviceProbeInfo", \
7868         EnhancedGetBusInfo, "EnhancedGetBusInfo", \
7869         SetupExtendedCounters, "SetupExtendedCounters", \
7870         GetPerformanceCounters, "GetPerformanceCounters", \
```



```
7871         ResetPerformanceCounters, "ResetPerformanceCounters", \
7872         ReadModePage, "ReadModePage", \
7873         WriteModePage, "WriteModePage", \
7874         ReadDriveParameter, "ReadDriveParameter", \
7875         WriteDriveParameter, "WriteDriveParameter", \
7876         ResetAdapter, "ResetAdapter", \
7877         ResetBus, "ResetBus", \
7878         ResetBusDevice, "ResetBusDevice", \
7879         ExecuteSrb, "ExecuteSrb", \
7880         Create_IO_Task, "Create_IO_Task", \
7881         Delete_IO_Task, "Delete_IO_Task", \
7882         Get_IO_Task_Info, "Get_IO_Task_Info", \
7883         Check_Task_Progress, "Check_Task_Progress", \
7884         InjectError, "InjectError", \
7885         GetDeviceDefectCounts, "GetDeviceDefectCounts", \
7886         GetDeviceDefectInfo, "GetDeviceDefectInfo", \
7887         GetDeviceStatus, "GetDeviceStatus", \
7888         ClearDeviceStatus, "ClearDeviceStatus", \
7889         DiskSpinControl, "DiskSpinControl", \
7890         DiskSmartControl, "DiskSmartControl", \
7891         WriteSame, "WriteSame", \
7892         ReadWriteLong, "ReadWriteLong", \
7893         FormatUnit, "FormatUnit", \
7894         TargetDeviceControl, "TargetDeviceControl", \
7895         TargetChannelControl, "TargetChannelControl", \
7896         FlashNewCode, "FlashNewCode", \
7897         DiskCheck, "DiskCheck", \
7898         RequestSense, "RequestSense", \
7899         DiskPERControl, "DiskPERControl", \
7900         Read10, "Read10", \
7901         Write10, "Write10"

7903 #define AAC_AIFEN_KEY_STRINGS \
7904         AifEnGeneric, "Generic", \
7905         AifEnTaskComplete, "TaskComplete", \
7906         AifEnConfigChange, "Config change", \
7907         AifEnContainerChange, "Container change", \
7908         AifEnDeviceFailure, "device failed", \
7909         AifEnMirrorFailover, "Mirror failover", \
7910         AifEnContainerEvent, "container event", \
7911         AifEnFileSystemChange, "File system changed", \
7912         AifEnConfigPause, "Container pause event", \
7913         AifEnConfigResume, "Container resume event", \
7914         AifEnFailoverChange, "Failover space assignment changed", \
7915         AifEnRAID5RebuildDone, "RAID5 rebuild finished", \
7916         AifEnEnclosureManagement, "Enclosure management event", \
7917         AifEnBatteryEvent, "battery event", \
7918         AifEnAddContainer, "Add container", \
7919         AifEnDeleteContainer, "Delete container", \
7920         AifEnSMARTEvent, "SMART Event", \
7921         AifEnBatteryNeedsRecond, "battery needs reconditioning", \
7922         AifEnClusterEvent, "cluster event", \
7923         AifEnDiskSetEvent, "disk set event occured", \
7924         AifDenMorphComplete, "morph operation completed", \
7925         AifDenVolumeExtendComplete, "VolumeExtendComplete"

7927 struct aac_key_strings {
7928         int key;
7929         char *message;
7930 };
______unchanged_portion_omitted_
7968 #endif /* AAC_DEBUG_ALL */

7970 #ifdef AAC_DEBUG
7971 /*
7972  * The following function comes from Adaptec:
```

```
 *
 * Get the firmware print buffer parameters from the firmware,
 * if the command was successful map in the address.
 */
static int
aac_get_fw_debug_buffer(struct aac_softstate *softs)
{
        softs->sync_slot_busy = 1;
        if (aac_sync_mbcommand(softs, AAC_MONKER_GETDRVPROP,
            0, 0, 0, 0, NULL, NULL) == AACOK) {
            0, 0, 0, 0, NULL) == AACOK) {
                uint32_t mondrv_buf_paddrl = AAC_MAILBOX_GET(softs, 1);
                uint32_t mondrv_buf_paddrh = AAC_MAILBOX_GET(softs, 2);
                uint32_t mondrv_buf_size = AAC_MAILBOX_GET(softs, 3);
                uint32_t mondrv_hdr_size = AAC_MAILBOX_GET(softs, 4);

                if (mondrv_buf_size) {
                        uint32_t offset = mondrv_buf_paddrl - \
                            softs->pci_mem_base_paddr[1];
                            softs->pci_mem_base_paddr;

                        /*
                         * See if the address is already mapped in, and
                         * if so set it up from the base address
                         */
                        if ((mondrv_buf_paddrh == 0) &&
                            (offset + mondrv_buf_size < softs->map_size)) {
                                mutex_enter(&aac_prt_mutex);
                                softs->debug_buf_offset = offset;
                                softs->debug_header_size = mondrv_hdr_size;
                                softs->debug_buf_size = mondrv_buf_size;
                                softs->debug_fw_flags = 0;
                                softs->debug_flags &= ~AACDB_FLAGS_FW_PRINT;
                                mutex_exit(&aac_prt_mutex);

                                return (AACOK);
                        }
                }
        }
        return (AACERR);
}
```

*unchanged_portion_omitted_*

```
/*
 * The following function comes from Adaptec:
 *
 * Format and print out the data passed in to UART or console
 * as specified by debug flags.
 */
void
aac_printf(struct aac_softstate *softs, uint_t lev, const char *fmt, ...)
{
        va_list args;
        char sl; /* system log character */

        mutex_enter(&aac_prt_mutex);
        /* Set up parameters and call sprintf function to format the data */
        if (strchr("^!?", fmt[0]) == NULL) {
                sl = 0;
        } else {
                sl = fmt[0];
                fmt++;
        }
        va_start(args, fmt);
        (void) vsprintf(aac_prt_buf, fmt, args);
        va_end(args);
```



```
        /* Make sure the softs structure has been passed in for this section */
        if (softs) {
                if ((softs->debug_flags & AACDB_FLAGS_FW_PRINT) &&
                    /* If we are set up for a Firmware print */
                    (softs->debug_buf_size)) {
                        uint32_t count, i;

                        /* Make sure the string size is within boundaries */
                        count = strlen(aac_prt_buf);
                        if (count > softs->debug_buf_size)
                                count = (uint16_t)softs->debug_buf_size;

                        /*
                         * Wait for no more than AAC_PRINT_TIMEOUT for the
                         * previous message length to clear (the handshake).
                         */
                        for (i = 0; i < AAC_PRINT_TIMEOUT; i++) {
                                if (!PCI_MEM_GET32(softs, 1,
                                if (!PCI_MEM_GET32(softs,
                                    softs->debug_buf_offset + \
                                    AAC_FW_DBG_STRLEN_OFFSET))
                                        break;

                                drv_usecwait(1000);
                        }

                        /*
                         * If the length is clear, copy over the message, the
                         * flags, and the length. Make sure the length is the
                         * last because that is the signal for the Firmware to
                         * pick it up.
                         */
                        if (!PCI_MEM_GET32(softs, 1, softs->debug_buf_offset + \
                        if (!PCI_MEM_GET32(softs, softs->debug_buf_offset + \
                            AAC_FW_DBG_STRLEN_OFFSET)) {
                                PCI_MEM_REP_PUT8(softs, 1,
                                PCI_MEM_REP_PUT8(softs,
                                    softs->debug_buf_offset + \
                                    softs->debug_header_size,
                                    aac_prt_buf, count);
                                PCI_MEM_PUT32(softs, 1,
                                PCI_MEM_PUT32(softs,
                                    softs->debug_buf_offset + \
                                    AAC_FW_DBG_FLAGS_OFFSET,
                                    softs->debug_fw_flags);
                                PCI_MEM_PUT32(softs, 1,
                                PCI_MEM_PUT32(softs,
                                    softs->debug_buf_offset + \
                                    AAC_FW_DBG_STRLEN_OFFSET, count);
                        } else {
                                cmn_err(CE_WARN, "UART output fail");
                                softs->debug_flags &= ~AACDB_FLAGS_FW_PRINT;
                        }
                }

                /*
                 * If the Kernel Debug Print flag is set, send it off
                 * to the Kernel Debugger
                 */
                if (softs->debug_flags & AACDB_FLAGS_KERNEL_PRINT)
                        aac_cmn_err(softs, lev, sl,
                            (softs->debug_flags & AACDB_FLAGS_NO_HEADERS));
        } else {
                /* Driver not initialized yet, no firmware or header output */
                if (aac_debug_flags & AACDB_FLAGS_KERNEL_PRINT)
```

```
8129                        aac_cmn_err(softs, lev, sl, 1);
8130        }
8131        mutex_exit(&aac_prt_mutex);
8132 }
8133 #endif /* AAC_DEBUG */

8135 #ifdef AAC_DEBUG_ALL
8136 /*
8137  * Translate command number to description string
8138  */
8139 static char *
8140 aac_cmd_name(int cmd, struct aac_key_strings *cmdlist)
8141 {
8142        int i;

8144        for (i = 0; cmdlist[i].key != -1; i++) {
8145                if (cmd == cmdlist[i].key)
8146                        return (cmdlist[i].message);
8147        }
8148        return (NULL);
8149 }

8151 static void
8152 aac_print_scmd(struct aac_softstate *softs, struct aac_cmd *acp)
8153 {
8154        struct scsi_pkt *pkt = acp->pkt;
8155        struct scsi_address *ap = &pkt->pkt_address;
8156        int is_pd = 0;
8157        int ctl = ddi_get_instance(softs->devinfo_p);
8158        int tgt = ap->a_target;
8159        int lun = ap->a_lun;
8160        union scsi_cdb *cdbp = (union scsi_cdb *)pkt->pkt_cdbp;
8171        union scsi_cdb *cdbp = (void *)pkt->pkt_cdbp;
8161        uchar_t cmd = cdbp->scc_cmd;
8162        char *desc;

8164        if (tgt >= AAC_MAX_LD) {
8165                is_pd = 1;
8166                ctl = ((struct aac_nondasd *)acp->dvp)->bus;
8167                tgt = ((struct aac_nondasd *)acp->dvp)->tid;
8168                lun = 0;
8169        }

8171        if ((desc = aac_cmd_name(cmd,
8172            (struct aac_key_strings *)scsi_cmds)) == NULL) {
8173                aac_printf(softs, CE_NOTE,
8174                    "SCMD> Unknown(0x%2x) --> c%dt%dL%d %s",
8175                    cmd, ctl, tgt, lun, is_pd ? "(pd)" : "");
8176                return;
8177        }

8179        switch (cmd) {
8180        case SCMD_READ:
8181        case SCMD_WRITE:
8182                aac_printf(softs, CE_NOTE,
8183                    "SCMD> %s 0x%x[%d] %s --> c%dt%dL%d %s",
8184                    desc, GETG0ADDR(cdbp), GETG0COUNT(cdbp),
8185                    (acp->flags & AAC_CMD_NO_INTR) ? "poll" : "intr",
8186                    ctl, tgt, lun, is_pd ? "(pd)" : "");
8187                break;
8188        case SCMD_READ_G1:
8189        case SCMD_WRITE_G1:
8190                aac_printf(softs, CE_NOTE,
8191                    "SCMD> %s 0x%x[%d] %s --> c%dt%dL%d %s",
8192                    desc, GETG1ADDR(cdbp), GETG1COUNT(cdbp),
8193                    (acp->flags & AAC_CMD_NO_INTR) ? "poll" : "intr",
```



```
8194                    ctl, tgt, lun, is_pd ? "(pd)" : "");
8195                break;
8196        case SCMD_READ_G4:
8197        case SCMD_WRITE_G4:
8198                aac_printf(softs, CE_NOTE,
8199                    "SCMD> %s 0x%x.%08x[%d] %s --> c%dt%dL%d %s",
8200                    desc, GETG4ADDR(cdbp), GETG4ADDRTL(cdbp),
8201                    GETG4COUNT(cdbp),
8202                    (acp->flags & AAC_CMD_NO_INTR) ? "poll" : "intr",
8203                    ctl, tgt, lun, is_pd ? "(pd)" : "");
8204                break;
8216        case SCMD_READ_G5:
8217        case SCMD_WRITE_G5:
8218                aac_printf(softs, CE_NOTE,
8219                    "SCMD> %s 0x%x[%d] %s --> c%dt%dL%d %s",
8220                    desc, GETG5ADDR(cdbp), GETG5COUNT(cdbp),
8221                    (acp->flags & AAC_CMD_NO_INTR) ? "poll" : "intr",
8222                    ctl, tgt, lun, is_pd ? "(pd)" : "");
8223                break;
8205        default:
8206                aac_printf(softs, CE_NOTE, "SCMD> %s --> c%dt%dL%d %s",
8207                    desc, ctl, tgt, lun, is_pd ? "(pd)" : "");
8208        }
8209 }

8211 void
8212 aac_print_fib(struct aac_softstate *softs, struct aac_fib *fibp)
8231 aac_print_fib(struct aac_softstate *softs, struct aac_slot *slotp)
8213 {
8233        struct aac_cmd *acp = slotp->acp;
8234        struct aac_fib *fibp = slotp->fibp;
8235        ddi_acc_handle_t acc = slotp->fib_acc_handle;
8214        uint16_t fib_size;
8215        int32_t fib_cmd, sub_cmd;
8237        uint32_t fib_cmd, sub_cmd;
8216        char *cmdstr, *subcmdstr;
8217        struct aac_Container *pContainer;
8239        char *caller;
8240        int i;

8219        fib_cmd = LE_16(fibp->Header.Command);
8242        if (acp) {
8243                if (!(softs->debug_fib_flags & acp->fib_flags))
8244                        return;
8245                if (acp->fib_flags & AACDB_FLAGS_FIB_SCMD)
8246                        caller = "SCMD";
8247                else if (acp->fib_flags & AACDB_FLAGS_FIB_IOCTL)
8248                        caller = "IOCTL";
8249                else if (acp->fib_flags & AACDB_FLAGS_FIB_SRB)
8250                        caller = "SRB";
8251                else
8252                        return;
8253        } else {
8254                if (!(softs->debug_fib_flags & AACDB_FLAGS_FIB_SYNC))
8255                        return;
8256                caller = "SYNC";
8257        }

8259        fib_cmd = ddi_get16(acc, &fibp->Header.Command);
8220        cmdstr = aac_cmd_name(fib_cmd, aac_fib_cmds);
8221        sub_cmd = -1;
8261        sub_cmd = (uint32_t)-1;
8222        subcmdstr = NULL;

8264        /* Print FIB header */
8265        if (softs->debug_fib_flags & AACDB_FLAGS_FIB_HEADER) {
```

```
8266                aac_printf(softs, CE_NOTE, "FIB> from %s", caller);
8267                aac_printf(softs, CE_NOTE, "     XferState  %d",
8268                    ddi_get32(acc, &fibp->Header.XferState));
8269                aac_printf(softs, CE_NOTE, "     Command    %d",
8270                    ddi_get16(acc, &fibp->Header.Command));
8271                aac_printf(softs, CE_NOTE, "     StructType %d",
8272                    ddi_get8(acc, &fibp->Header.StructType));
8273                aac_printf(softs, CE_NOTE, "     Flags      0x%x",
8274                    ddi_get8(acc, &fibp->Header.Flags));
8275                aac_printf(softs, CE_NOTE, "     Size       %d",
8276                    ddi_get16(acc, &fibp->Header.Size));
8277                aac_printf(softs, CE_NOTE, "     SenderSize %d",
8278                    ddi_get16(acc, &fibp->Header.SenderSize));
8279                aac_printf(softs, CE_NOTE, "     SenderAddr 0x%x",
8280                    ddi_get32(acc, &fibp->Header.SenderFibAddress));
8281                aac_printf(softs, CE_NOTE, "     RcvrAddr   0x%x",
8282                    ddi_get32(acc, &fibp->Header.ReceiverFibAddress));
8283                aac_printf(softs, CE_NOTE, "     SenderData 0x%x",
8284                    ddi_get32(acc, &fibp->Header.SenderData));
8285        }

8287        /* Print FIB data */
8224        switch (fib_cmd) {
8225        case ContainerCommand:
8226                pContainer = (struct aac_Container *)fibp->data;
8227                sub_cmd = LE_32(pContainer->Command);
8290                sub_cmd = ddi_get32(acc,
8291                    (void *)&(((uint32_t *)(void *)&fibp->data[0])[0]));
8228                subcmdstr = aac_cmd_name(sub_cmd, aac_ctvm_subcmds);
8229                if (subcmdstr == NULL)
8230                        break;

8296                switch (sub_cmd) {
8297                case VM_ContainerConfig: {
8298                        struct aac_Container *pContainer =
8299                            (struct aac_Container *)fibp->data;

8231                fib_cmd = sub_cmd;
8232                cmdstr = subcmdstr;
8233                sub_cmd = -1;
8303                        sub_cmd = (uint32_t)-1;
8234                subcmdstr = NULL;

8236                switch (pContainer->Command) {
8237                case VM_ContainerConfig:
8238                        sub_cmd = LE_32(pContainer->CTCommand.command);
8306                        sub_cmd = ddi_get32(acc,
8307                            &pContainer->CTCommand.command);
8239                        subcmdstr = aac_cmd_name(sub_cmd, aac_ct_subcmds);
8240                        if (subcmdstr == NULL)
8241                                break;
8242                        aac_printf(softs, CE_NOTE, "FIB> %s (0x%x, 0x%x, 0x%x)",
8243                            subcmdstr,
8244                            LE_32(pContainer->CTCommand.param[0]),
8245                            LE_32(pContainer->CTCommand.param[1]),
8246                            LE_32(pContainer->CTCommand.param[2]));
8313                            ddi_get32(acc, &pContainer->CTCommand.param[0]),
8314                            ddi_get32(acc, &pContainer->CTCommand.param[1]),
8315                            ddi_get32(acc, &pContainer->CTCommand.param[2]));
8247                        return;
8317                }

8248                case VM_Ioctl:
8249                        sub_cmd = LE_32(((int32_t *)pContainer)[4]);
8320                        fib_cmd = sub_cmd;
8321                        cmdstr = subcmdstr;
```



```
8322                        sub_cmd = (uint32_t)-1;
8323                        subcmdstr = NULL;

8325                        sub_cmd = ddi_get32(acc,
8326                            (void *)&(((uint32_t *)(void *)&fibp->data[0])[4]));
8250                        subcmdstr = aac_cmd_name(sub_cmd, aac_ioctl_subcmds);
8251                        break;

8330                case VM_CtBlockRead:
8331                case VM_CtBlockWrite: {
8332                        struct aac_blockread *br =
8333                            (struct aac_blockread *)fibp->data;
8334                        struct aac_sg_table *sg = &br->SgMap;
8335                        uint32_t sgcount = ddi_get32(acc, &sg->SgCount);

8337                        aac_printf(softs, CE_NOTE,
8338                            "FIB> %s Container %d  0x%x/%d", subcmdstr,
8339                            ddi_get32(acc, &br->ContainerId),
8340                            ddi_get32(acc, &br->BlockNumber),
8341                            ddi_get32(acc, &br->ByteCount));
8342                        for (i = 0; i < sgcount; i++)
8343                                aac_printf(softs, CE_NOTE,
8344                                    "     %d: 0x%08x/%d", i,
8345                                    ddi_get32(acc, &sg->SgEntry[i].SgAddress),
8346                                    ddi_get32(acc, &sg->SgEntry[i]. \
8347                                    SgByteCount));
8348                        return;
8252                }
8350                }
8253                break;

8353        case ContainerCommand64: {
8354                struct aac_blockread64 *br =
8355                    (struct aac_blockread64 *)fibp->data;
8356                struct aac_sg_table64 *sg = &br->SgMap64;
8357                uint32_t sgcount = ddi_get32(acc, &sg->SgCount);
8358                uint64_t sgaddr;

8360                sub_cmd = br->Command;
8361                subcmdstr = NULL;
8362                if (sub_cmd == VM_CtHostRead64)
8363                        subcmdstr = "VM_CtHostRead64";
8364                else if (sub_cmd == VM_CtHostWrite64)
8365                        subcmdstr = "VM_CtHostWrite64";
8366                else
8367                        break;

8369                aac_printf(softs, CE_NOTE,
8370                    "FIB> %s Container %d  0x%x/%d", subcmdstr,
8371                    ddi_get16(acc, &br->ContainerId),
8372                    ddi_get32(acc, &br->BlockNumber),
8373                    ddi_get16(acc, &br->SectorCount));
8374                for (i = 0; i < sgcount; i++) {
8375                        sgaddr = ddi_get64(acc,
8376                            &sg->SgEntry64[i].SgAddress);
8377                        aac_printf(softs, CE_NOTE,
8378                            "     %d: 0x%08x.%08x/%d", i,
8379                            AAC_MS32(sgaddr), AAC_LS32(sgaddr),
8380                            ddi_get32(acc, &sg->SgEntry64[i]. \
8381                            SgByteCount));
8382                }
8383                return;
8384        }

8386        case RawIo: {
8387                struct aac_raw_io *io = (struct aac_raw_io *)fibp->data;
```

```
8388                *struct aac_sg_tableraw *sg = &io->SgMapRaw;*
8389                *uint32_t sgcount = ddi_get32(acc, &sg->SgCount);*
8390                *uint64_t sgaddr;*
8391
8392                *aac_printf(softs, CE_NOTE,*
8393                    *"FIB> RawIo Container %d  0x%llx/%d 0x%x",*
8394                    *ddi_get16(acc, &io->ContainerId),*
8395                    *ddi_get64(acc, &io->BlockNumber),*
8396                    *ddi_get32(acc, &io->ByteCount),*
8397                    *ddi_get16(acc, &io->Flags));*
8398                *for (i = 0; i < sgcount; i++) {*
8399                        *sgaddr = ddi_get64(acc, &sg->SgEntryRaw[i].SgAddress);*
8400                        *aac_printf(softs, CE_NOTE, "     %d: 0x%08x.%08x/%d", i,*
8401                            *AAC_MS32(sgaddr), AAC_LS32(sgaddr),*
8402                            *ddi_get32(acc, &sg->SgEntryRaw[i].SgByteCount));*
8403                *}*
8404                *return;*
8405        *}*

8255        case ClusterCommand:
8256                **sub_cmd = LE_32(((int32_t *)fibp->data)[0]);**
8408                *sub_cmd = ddi_get32(acc,*
8409                    *(void *)&(((uint32_t *)(void *)fibp->data)[0]));*
8257                subcmdstr = aac_cmd_name(sub_cmd, aac_cl_subcmds);
8258                break;

8260        case AifRequest:
8261                **sub_cmd = LE_32(((int32_t *)fibp->data)[0]);**
8414                *sub_cmd = ddi_get32(acc,*
8415                    *(void *)&(((uint32_t *)(void *)fibp->data)[0]));*
8262                subcmdstr = aac_cmd_name(sub_cmd, aac_aif_subcmds);
8263                break;

8265        default:
8266                break;
8267        }

8269        **fib_size = LE_16(fibp->Header.Size);**
8423        *fib_size = ddi_get16(acc, &(fibp->Header.Size));*
8270        if (subcmdstr)
8271                aac_printf(softs, CE_NOTE, "FIB> %s, sz=%d",
8272                    subcmdstr, fib_size);
8273        **else if (cmdstr && sub_cmd == -1)**
8427        *else if (cmdstr && sub_cmd == (uint32_t)-1)*
8274                aac_printf(softs, CE_NOTE, "FIB> %s, sz=%d",
8275                    cmdstr, fib_size);
8276        else if (cmdstr)
8277                aac_printf(softs, CE_NOTE, "FIB> %s: Unknown(0x%x), sz=%d",
8278                    cmdstr, sub_cmd, fib_size);
8279        else
8280                aac_printf(softs, CE_NOTE, "FIB> Unknown(0x%x), sz=%d",
8281                    fib_cmd, fib_size);
8282 }
_____*unchanged_portion_omitted_*
8346 **#endif /* AAC_DEBUG_ALL */**

8501 *#endif /* DEBUG */*
```

```
**********************************************************
   17538 Thu Nov  1 14:48:26 2012
new/usr/src/uts/common/io/aac/aac.h
*** NO COMMENTS ***
**********************************************************
/*
 * Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 * Copyright (c) 2010, Oracle and/or its affiliates. All rights reserved.
 */

/*
 * Copyright (c) 2010-12 PMC-Sierra, Inc.
 * Copyright (c) 2005-10 Adaptec Inc., Achim Leubner
 * Copyright 2005-06 Adaptec, Inc.
 * Copyright (c) 2005-06 Adaptec Inc., Achim Leubner
 * Copyright (c) 2000 Michael Smith
 * Copyright (c) 2001 Scott Long
 * Copyright (c) 2000 BSDi
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 *
 *    $FreeBSD: src/sys/dev/aac/aacvar.h,v 1.47 2005/10/08 15:55:09 scottl Exp $
 */

#ifndef _AAC_H_
#define _AAC_H_

#pragma ident   "@(#)aac.h      1.16    08/01/29 SMI"

#ifdef  __cplusplus
extern "C" {
#endif

#ifdef AAC_DEBUG_ALL
#define AAC_DEBUG
#define DEBUG                   /* activate assertions */
#endif

#define AAC_ROUNDUP(x, y)               (((x) + (y) - 1) / (y) * (y))

#define AAC_TYPE_DEVO                   1
#define AAC_TYPE_ALPHA                  2
#define AAC_TYPE_BETA                   3
#define AAC_TYPE_RELEASE                4
```



```
#ifndef AAC_DRIVER_BUILD
#define AAC_DRIVER_BUILD                1
#endif

#define AAC_DRIVER_MAJOR_VERSION        2
#define AAC_DRIVER_MINOR_VERSION        7
#define AAC_DRIVER_BUGFIX_LEVEL         1
#define AAC_DRIVER_MINOR_VERSION        2
#define AAC_DRIVER_BUGFIX_LEVEL         11
#define AAC_DRIVER_TYPE                 AAC_TYPE_RELEASE

#define STR(s)                          # s
#define AAC_VERSION(a, b, c)            STR(a.b.c)
#define AAC_DRIVER_VERSION              AAC_VERSION(AAC_DRIVER_MAJOR_VERSION, \
                                        AAC_DRIVER_MINOR_VERSION, \
                                        AAC_DRIVER_BUGFIX_LEVEL)

#define AACOK                           0
#define AACOK2                          1
#define AACERR                          -1

#define AAC_MAX_ADAPTERS                64

/* Definitions for mode sense */
#ifndef SD_MODE_SENSE_PAGE3_CODE
#define SD_MODE_SENSE_PAGE3_CODE        0x03
#endif

#ifndef SD_MODE_SENSE_PAGE4_CODE
#define SD_MODE_SENSE_PAGE4_CODE        0x04
#endif

#ifndef SCMD_SYNCHRONIZE_CACHE
#define SCMD_SYNCHRONIZE_CACHE          0x35
#endif

/*
 * The controller reports status events in AIFs. We hang on to a number of
 * these in order to pass them out to user-space management tools.
 */
#define AAC_AIFQ_LENGTH                 64

#ifdef __x86
#define AAC_IMMEDIATE_TIMEOUT           30      /* seconds */
#else
#define AAC_IMMEDIATE_TIMEOUT           60      /* seconds */
#endif
#define AAC_FWUP_TIMEOUT                180     /* wait up to 3 minutes */
#define AAC_IOCTL_TIMEOUT               900     /* wait up to 15 minutes */
#define AAC_AIF_TIMEOUT         180 /* up to 3 minutes */
#define AAC_SYNC_TIMEOUT                900     /* wait up to 15 minutes */

/* Adapter hardware interface types */
#define AAC_HWIF_UNKNOWN                0
#define AAC_HWIF_I960RX                 1
#define AAC_HWIF_RKT                    2
#define AAC_HWIF_NARK                   3
#define AAC_HWIF_SRC                    4
#define AAC_HWIF_SRCV                   5

#define AAC_TYPE_UNKNOWN                0
#define AAC_TYPE_SCSI                   1
#define AAC_TYPE_SATA                   2
#define AAC_TYPE_SAS                    3

#define AAC_LS32(d)                     ((uint32_t)((d) & 0xffffffffull))
```

```
120 #define AAC_MS32(d)                     ((uint32_t)((d) >> 32))
121 #define AAC_LO32(p64)                   ((uint32_t *)(p64))
122 #define AAC_HI32(p64)                   ((uint32_t *)(p64) + 1)

124 /*
115  * Internal events that will be handled serially by aac_event_thread()
116  */
117 #define AAC_EVENT_AIF                   (1 << 0)
118 #define AAC_EVENT_TIMEOUT               (1 << 1)
119 #define AAC_EVENT_SYNCTICK              (1 << 2)

121 /*
125  * AAC_CMDQ_SYNC should be 0 and AAC_CMDQ_ASYNC be 1 for Sync FIB io
126  * to be served before async FIB io, see aac_start_waiting_io().
127  * So that io requests sent by interactive userland commands get
128  * responded asap.
129  */
130 enum aac_cmdq {
131         AAC_CMDQ_SYNC,  /* sync FIB queue */
132         AAC_CMDQ_ASYNC, /* async FIB queue */
133         AAC_CMDQ_NUM
134 };
______unchanged_portion_omitted_

159 /* Device types */
162 #define AAC_DEV_LD      0       /* logical device */
163 #define AAC_DEV_PD      1       /* physical device */

165 #define AAC_DEV_NONE    0
166 #define AAC_DEV_ONLINE  1
167 #define AAC_DEV_OFFLINE 2
163 /* Device flags */
164 #define AAC_DFLAG_VALID         (1 << 0)
165 #define AAC_DFLAG_CONFIGURING   (1 << 1)

167 #define AAC_DEV_IS_VALID(dvp)   ((dvp)->flags & AAC_DFLAG_VALID)
168 #define AAC_P2VTGT(softs, bus, tgt) \
169                 ((softs)->tgt_max * (bus) + (tgt) + AAC_MAX_LD)

171 /*
172  * Device config change events
173  */
174 enum aac_cfg_event {
175         AAC_CFG_NULL_NOEXIST = 0,       /* No change with no device */
176         AAC_CFG_NULL_EXIST,             /* No change but have device */
177         AAC_CFG_ADD,                    /* Device added */
178         AAC_CFG_DELETE,                 /* Device deleted */
179         AAC_CFG_CHANGE                  /* Device changed */
180 };

168 struct aac_device {
169         uint8_t valid;
183         int flags;

170         uint8_t type;
171         dev_info_t *dip;
172         int ncmds[AAC_CMDQ_NUM];        /* outstanding cmds of the device */
173         int throttle[AAC_CMDQ_NUM];     /* hold IO cmds for the device */
174 };
______unchanged_portion_omitted_

188 /* Non-DASD phys. device descrption, eg. CDROM or tape */
189 struct aac_nondasd {
190         struct aac_device dev;

192         uint32_t bus;
```



```
193         uint32_t tid;
194         uint8_t dtype;          /* SCSI device type */
195 };
______unchanged_portion_omitted_

226 /*
227  * Scatter-gather list structure defined by HBA hardware
228  */
229 struct aac_sge {
230         uint32_t bcount;        /* byte count */
231         union {
232                 uint32_t ad32;  /* 32 bit address */
233                 struct {
234                         uint32_t lo;
235                         uint32_t hi;
236                 } ad64;         /* 64 bit address */
237         } addr;
238 };

240 /* aac_cmd flags */
241 #define AAC_CMD_CONSISTENT              (1 << 0)
242 #define AAC_CMD_DMA_PARTIAL             (1 << 1)
243 #define AAC_CMD_DMA_VALID               (1 << 2)
244 #define AAC_CMD_BUF_READ                (1 << 3)
245 #define AAC_CMD_BUF_WRITE               (1 << 4)
246 #define AAC_CMD_SYNC                    (1 << 5) /* use sync FIB */
247 #define AAC_CMD_NO_INTR                 (1 << 6) /* poll IO, no intr */
248 #define AAC_CMD_NO_CB                   (1 << 7) /* sync IO, no callback */
249 #define AAC_CMD_NTAG                    (1 << 8)
250 #define AAC_CMD_CMPLT                   (1 << 9) /* cmd exec'ed by driver/fw */
251 #define AAC_CMD_ABORT                   (1 << 10)
252 #define AAC_CMD_TIMEOUT                 (1 << 11)
253 #define AAC_CMD_ERR                     (1 << 12)
254 #define AAC_CMD_IN_SYNC_SLOT            (1 << 13)

256 struct aac_softstate;
257 typedef void (*aac_cmd_fib_t)(struct aac_softstate *, struct aac_cmd *);

259 struct aac_cmd {
260         /*
261          * Note: should be the first member for aac_cmd_queue to work
262          * correctly.
263          */
264         struct aac_cmd *next;
265         struct aac_cmd *prev;

267         struct scsi_pkt *pkt;
268         int cmdlen;
269         int flags;
270         uint32_t timeout; /* time when the cmd should have completed */
271         struct buf *bp;
272         ddi_dma_handle_t buf_dma_handle;

274         /* For non-aligned buffer and SRB */
275         caddr_t abp;
276         ddi_acc_handle_t abh;

278         /* Data transfer state */
279         ddi_dma_cookie_t cookie;
280         uint_t left_cookien;
281         uint_t cur_win;
282         uint_t total_nwin;
283         size_t total_xfer;
284         uint64_t blkno;
285         uint32_t bcount;        /* buffer size in byte */
286         struct aac_sge *sgt;    /* sg table */
```

```
288         /* FIB construct function */
289         aac_cmd_fib_t aac_cmd_fib;
290         /* Call back function for completed command */
291         void (*ac_comp)(struct aac_softstate *, struct aac_cmd *);

293         struct aac_slot *slotp; /* slot used by this command */
294         struct aac_device *dvp; /* target device */

296         /* FIB for this IO command */
297         int fib_size; /* size of the FIB xferred to/from the card */
298         struct aac_fib *fibp;

300 #ifdef DEBUG
301         uint32_t fib_flags;
302 #endif
303 };

212 /* Flags for attach tracking */
213 #define AAC_ATTACH_SOFTSTATE_ALLOCED    (1 << 0)
214 #define AAC_ATTACH_CARD_DETECTED        (1 << 1)
215 #define AAC_ATTACH_PCI_MEM_MAPPED       (1 << 2)
216 #define AAC_ATTACH_KMUTEX_INITED        (1 << 3)
217 #define AAC_ATTACH_HARD_INTR_SETUP      (1 << 4)
218 #define AAC_ATTACH_SOFT_INTR_SETUP      (1 << 5)
219 #define AAC_ATTACH_SCSI_TRAN_SETUP      (1 << 6)
220 #define AAC_ATTACH_COMM_SPACE_SETUP     (1 << 7)
221 #define AAC_ATTACH_CREATE_DEVCTL        (1 << 8)
222 #define AAC_ATTACH_CREATE_SCSI          (1 << 9)
310 #define AAC_ATTACH_SCSI_TRAN_SETUP      (1 << 4)
311 #define AAC_ATTACH_COMM_SPACE_SETUP     (1 << 5)
312 #define AAC_ATTACH_CREATE_DEVCTL        (1 << 6)
313 #define AAC_ATTACH_CREATE_SCSI          (1 << 7)

224 /* Driver running states */
225 #define AAC_STATE_STOPPED       0
226 #define AAC_STATE_RUN           (1 << 0)
227 #define AAC_STATE_RESET         (1 << 1)
228 #define AAC_STATE_QUIESCED      (1 << 2)
229 #define AAC_STATE_DEAD          (1 << 3)
321 #define AAC_STATE_INTR          (1 << 4)

231 /*
232  * Flags for aac firmware
233  * Note: Quirks are only valid for the older cards. These cards only supported
234  * old comm. Thus they are not valid for any cards that support new comm.
235  */
236 #define AAC_FLAGS_SG_64BIT      (1 << 0) /* Use 64-bit S/G addresses */
237 #define AAC_FLAGS_4GB_WINDOW    (1 << 1) /* Can access host mem 2-4GB range */
238 #define AAC_FLAGS_NO4GB (1 << 2)        /* quirk: FIB addresses must reside */
239                                         /*       between 0x2000 & 0x7FFFFFFF */
240 #define AAC_FLAGS_256FIBS       (1 << 3) /* quirk: Can only do 256 commands */
241 #define AAC_FLAGS_NEW_COMM      (1 << 4) /* New comm. interface supported */
242 #define AAC_FLAGS_RAW_IO        (1 << 5) /* Raw I/O interface */
243 #define AAC_FLAGS_ARRAY_64BIT   (1 << 6) /* 64-bit array size */
244 #define AAC_FLAGS_LBA_64BIT     (1 << 7) /* 64-bit LBA supported */
245 #define AAC_FLAGS_17SG          (1 << 8) /* quirk: 17 scatter gather maximum */
246 #define AAC_FLAGS_34SG          (1 << 9) /* quirk: 34 scatter gather maximum */
247 #define AAC_FLAGS_NONDASD       (1 << 10) /* non-DASD device supported */
248 #define AAC_FLAGS_NEW_COMM_TYPE1        (1 << 11)       /* New comm. type1 suppo
249 #define AAC_FLAGS_NEW_COMM_TYPE2        (1 << 12)       /* New comm. type2 suppo
250 #define AAC_FLAGS_NEW_COMM_TYPE34       (1 << 13)       /* New comm. type3-4 */
251 #define AAC_FLAGS_SYNC_MODE     (1 << 14)       /* Sync. transfer mode */
340 #define AAC_FLAGS_BRKUP         (1 << 11) /* pkt breakup support */
341 #define AAC_FLAGS_JBOD          (1 << 12) /* JBOD mode support */
```



```
253 struct aac_softstate;
254 struct aac_interface {
255         void (*aif_set_intr)(struct aac_softstate *, int enable);
256         void (*aif_status_clr)(struct aac_softstate *, int mask);
257         int (*aif_status_get)(struct aac_softstate *);
258         void (*aif_notify)(struct aac_softstate *, int val);
259         int (*aif_get_fwstatus)(struct aac_softstate *);
260         int (*aif_get_mailbox)(struct aac_softstate *, int);
261         void (*aif_set_mailbox)(struct aac_softstate *, uint32_t,
262             uint32_t, uint32_t, uint32_t, uint32_t);
263         int (*aif_send_command)(struct aac_softstate *, struct aac_slot *);
264 };

351 #define AAC_CTXFLAG_FILLED      0x01    /* aifq's full for this ctx */
352 #define AAC_CTXFLAG_RESETED     0x02

266 struct aac_fib_context {
267         uint32_t unique;
268         int ctx_idx;
269         int ctx_filled;         /* aifq is full for this fib context */
358         int ctx_flags;
359         int ctx_overrun;
270         struct aac_fib_context *next, *prev;
271 };

273 typedef void (*aac_cmd_fib_t)(struct aac_softstate *, struct aac_cmd *);

275 #define AAC_VENDOR_LEN          8
276 #define AAC_PRODUCT_LEN         16

278 struct aac_softstate {
279         int card;               /* index to aac_cards */
280         uint16_t hwif;          /* card chip type: i960 or Rocket */
281         uint16_t vendid;        /* vendor id */
282         uint16_t subvendid;     /* sub vendor id */
283         uint16_t devid;         /* device id */
284         uint16_t subsysid;      /* sub system id */
285         char vendor_name[AAC_VENDOR_LEN + 1];
286         char product_name[AAC_PRODUCT_LEN + 1];
287         uint32_t support_opt;   /* firmware features */
376         uint32_t support_opt2;
377         uint32_t feature_bits;
288         uint32_t atu_size;      /* actual size of PCI mem space */
289         uint32_t map_size;      /* mapped PCI mem space size */
290         uint32_t map_size_min;  /* minimum size of PCI mem that must be */
291                                 /* mapped to address the card */
292         int flags;              /* firmware features enabled */
293         int instance;
294         dev_info_t *devinfo_p;
295         scsi_hba_tran_t *hba_tran;
296         int slen;
297         int legacy;
298         int sync_mode;
299         int no_sgl_conv;
387         int legacy;             /* legacy device naming */
388         uint32_t dma_max;       /* for buf breakup */

301         /* DMA attributes */
302         ddi_dma_attr_t buf_dma_attr;
303         ddi_dma_attr_t addr_dma_attr;

305         /* PCI spaces */
306         ddi_acc_handle_t pci_mem_handle[AAC_MAX_MEM_SPACE];
307         char *pci_mem_base_vaddr[AAC_MAX_MEM_SPACE];
308         uint32_t pci_mem_base_paddr[AAC_MAX_MEM_SPACE];
395         ddi_device_acc_attr_t acc_attr;
```

```
396         ddi_device_acc_attr_t reg_attr;
397         ddi_acc_handle_t pci_mem_handle;
398         uint8_t *pci_mem_base_vaddr;
399         uint32_t pci_mem_base_paddr;

310         struct aac_interface aac_if;    /* adapter hardware interface */

312         struct aac_slot *sync_slot;      /* sync FIB */
313         int sync_slot_busy;
314         struct aac_slot *sync_mode_slot;
403         struct aac_cmd sync_ac;         /* sync FIB */
315
316         /* Communication space */
317         struct aac_comm_space *comm_space;
318         ddi_acc_handle_t comm_space_acc_handle;
319         ddi_dma_handle_t comm_space_dma_handle;
320         uint32_t comm_space_phyaddr;

322         /* New Comm. type1: response buffer index */
323         uint32_t aac_host_rrq_idx;

325     /* Old Comm. interface: message queues */
326         struct aac_queue_table *qtablep;
327         struct aac_queue_entry *qentries[AAC_QUEUE_COUNT];

329         /* New Comm. interface */
330         uint32_t aac_max_fibs;          /* max. FIB count */
331         uint32_t aac_max_fib_size;      /* max. FIB size */
332         uint32_t aac_sg_tablesize;      /* max. sg count from host */
333         uint32_t aac_max_sectors;       /* max. I/O size from host (blocks) */
334         uint32_t aac_max_aif;           /* max. AIF count */

336         aac_cmd_fib_t aac_cmd_fib;      /* IO cmd FIB construct function */
337         aac_cmd_fib_t aac_cmd_fib_scsi; /* SRB construct function */

339         ddi_iblock_cookie_t iblock_cookie;
340         ddi_softintr_t softint_id;      /* soft intr */

342         kmutex_t io_lock;
343         int state;                      /* driver state */

345         struct aac_container containers[AAC_MAX_LD];
346         int container_count;            /* max container id + 1 */
347         struct aac_nondasd *nondasds;
348         uint32_t bus_max;               /* max FW buses exposed */
349         uint32_t tgt_max;               /* max FW target per bus */
350         uint32_t aac_feature_bits;
351         uint32_t aac_support_opt2;

353         /*
354          * Command queues
355          * Each aac command flows through wait(or wait_sync) queue,
356          * busy queue, and complete queue sequentially.
357          */
358         struct aac_cmd_queue q_wait[AAC_CMDQ_NUM];
359         struct aac_cmd_queue q_busy;    /* outstanding cmd queue */
360         kmutex_t q_comp_mutex;
361         struct aac_cmd_queue q_comp;    /* completed io requests */

363         /* I/O slots and FIBs */
364         int total_slots;                /* total slots allocated */
365         int total_fibs;                 /* total FIBs allocated */
366         struct aac_slot *io_slot;       /* static list for allocated slots */
367         struct aac_slot *free_io_slot_head;

369         timeout_id_t timeout_id;        /* for timeout daemon */
```



```

371         kcondvar_t event;               /* for ioctl_send_fib() and sync IO */
452         kcondvar_t sync_fib_cv;         /* for sync_fib_slot_bind/release */

373         int bus_ncmds[AAC_CMDQ_NUM];    /* total outstanding async cmds */
374         int bus_throttle[AAC_CMDQ_NUM]; /* hold IO cmds for the bus */
375         int ndrains;                    /* number of draining threads */
376         timeout_id_t drain_timeid;      /* for outstanding cmd drain */
377         kcondvar_t drain_cv;            /* for quiesce drain */

460         /* Internal timer */
461         kmutex_t time_mutex;
462         timeout_id_t timeout_id;        /* for timeout daemon */
463         uint32_t timebase;              /* internal timer in seconds */
464         uint32_t time_sync;             /* next time to sync with firmware */
465         uint32_t time_out;              /* next time to check timeout */
466         uint32_t time_throttle;         /* next time to restore throttle */

468         /* Internal events handling */
469         kmutex_t ev_lock;
470         int events;
471         kthread_t *event_thread;        /* for AIF & timeout */
472         kcondvar_t event_wait_cv;
473         kcondvar_t event_disp_cv;

379         /* AIF */
380         kmutex_t aifq_mutex;            /* for AIF queue aifq */
381         kcondvar_t aifv;
477         kcondvar_t aifq_cv;
382         union aac_fib_align aifq[AAC_AIFQ_LENGTH];
383         int aifq_idx;                   /* slot for next new AIF */
384         int aifq_wrap;                  /* AIF queue has ever been wrapped */
385         struct aac_fib_context *fibctx;
481         struct aac_fib_context aifctx;  /* sys aif ctx */
482         struct aac_fib_context *fibctx_p;
386         int devcfg_wait_on;             /* AIF event waited for rescan */

388         int fm_capabilities;
389         ddi_taskq_t *taskq;

391 #ifdef AAC_DEBUG
487         /* MSI specific fields */
488         ddi_intr_handle_t *htable;      /* For array of interrupts */
489         int intr_type;                  /* What type of interrupt */
490         int intr_cnt;                   /* # of intrs count returned */
491         int intr_size;
492         uint_t intr_pri;                /* Interrupt priority   */
493         int intr_cap;                   /* Interrupt capabilities */

495 #ifdef DEBUG
392         /* UART trace printf variables */
393         uint32_t debug_flags;           /* debug print flags bitmap */
498         uint32_t debug_fib_flags;       /* debug FIB print flags bitmap */
394         uint32_t debug_fw_flags;        /* FW debug flags */
395         uint32_t debug_buf_offset;      /* offset from DPMEM start */
396         uint32_t debug_buf_size;        /* FW debug buffer size in bytes */
397         uint32_t debug_header_size;     /* size of debug header */
398 #endif
399 };

401 /*
402  * The following data are kept stable because they are only written at driver
403  * initialization, and we do not allow them changed otherwise even at driver
404  * re-initialization.
405  */
406 _NOTE(SCHEME_PROTECTS_DATA("stable data", aac_softstate::{flags slen \
```

```
407     buf_dma_attr pci_mem_handle pci_mem_base_vaddr \
408     comm_space_acc_handle comm_space_dma_handle aac_max_fib_size \
409     aac_sg_tablesize aac_cmd_fib aac_cmd_fib_scsi debug_flags bus_max tgt_max \
410         aac_feature_bits}))
514     aac_sg_tablesize aac_cmd_fib aac_cmd_fib_scsi debug_flags bus_max tgt_max}))

412 /*
413  * Scatter-gather list structure defined by HBA hardware
414  */
415 struct aac_sge {
416         uint32_t bcount;        /* byte count */
417         union {
418                 uint32_t ad32;  /* 32 bit address */
419                 struct {
420                         uint32_t lo;
421                         uint32_t hi;
422                 } ad64;         /* 64 bit address */
423         } addr;
424 };
516 #ifdef DEBUG

426 /* aac_cmd flags */
427 #define AAC_CMD_CONSISTENT              (1 << 0)
428 #define AAC_CMD_DMA_PARTIAL             (1 << 1)
429 #define AAC_CMD_DMA_VALID               (1 << 2)
430 #define AAC_CMD_BUF_READ                (1 << 3)
431 #define AAC_CMD_BUF_WRITE               (1 << 4)
432 #define AAC_CMD_SYNC                    (1 << 5) /* use sync FIB */
433 #define AAC_CMD_NO_INTR                 (1 << 6) /* poll IO, no intr */
434 #define AAC_CMD_NO_CB                   (1 << 7) /* sync IO, no callback */
435 #define AAC_CMD_NTAG                    (1 << 8)
436 #define AAC_CMD_CMPLT                   (1 << 9) /* cmd exec'ed by driver/fw */
437 #define AAC_CMD_ABORT                   (1 << 10)
438 #define AAC_CMD_TIMEOUT                 (1 << 11)
439 #define AAC_CMD_ERR                     (1 << 12)
440 #define AAC_CMD_AIF                     (1 << 13)
441 #define AAC_CMD_AIF_NOMORE              (1 << 14)
442 #define AAC_CMD_FASTRESP                (1 << 15)

444 #define AAC_MAXSEGMENTS         16

446 struct aac_cmd {
447         /*
448          * Note: should be the first member for aac_cmd_queue to work
449          * correctly.
450          */
451         struct aac_cmd *next;
452         struct aac_cmd *prev;

454         struct scsi_pkt *pkt;
455         int cmdlen;
456         int flags;
457         uint32_t timeout; /* time when the cmd should have completed */
458         struct buf *bp;

460         uint_t segment_cnt;
461         uint_t left_cookien;
462         struct {
463                 ddi_dma_handle_t buf_dma_handle;
464                 /* For non-aligned buffer and SRB */
465                 caddr_t abp;
466                 ddi_acc_handle_t abh;
467                 uint32_t abp_size;
468                 size_t abp_real_size;

470                 /* Data transfer state */
```



```
471                 ddi_dma_cookie_t cookie;
472                 uint_t left_cookien;
473                 struct aac_sge *sgt;
474         } segments[AAC_MAXSEGMENTS];
475         uint_t cur_segment;
476         uint_t cur_win;
477         uint_t total_nwin;
478         size_t total_xfer;
479         uint64_t blkno;
480         uint32_t bcount;        /* buffer size in byte */
481         struct aac_sge *sgt;    /* sg table */

483         /* FIB construct function */
484         aac_cmd_fib_t aac_cmd_fib;
485         /* Call back function for completed command */
486         void (*ac_comp)(struct aac_softstate *, struct aac_cmd *);

488         struct aac_slot *slotp; /* slot used by this command */
489         struct aac_device *dvp; /* target device */

491         /* FIB for this IO command */
492         int fib_size; /* size of the FIB xferred to/from the card */
493         struct aac_fib *fibp;
494 };

496 #ifdef AAC_DEBUG
497 #define AACDB_FLAGS_MASK                0x0000ffff
498 #define AACDB_FLAGS_KERNEL_PRINT        0x00000001
499 #define AACDB_FLAGS_FW_PRINT            0x00000002
500 #define AACDB_FLAGS_NO_HEADERS          0x00000004

502 #define AACDB_FLAGS_MISC                0x00000010
503 #define AACDB_FLAGS_FUNC1               0x00000020
504 #define AACDB_FLAGS_FUNC2               0x00000040
505 #define AACDB_FLAGS_SCMD                0x00000080
506 #define AACDB_FLAGS_AIF                 0x00000100
507 #define AACDB_FLAGS_FIB                 0x00000200
508 #define AACDB_FLAGS_IOCTL               0x00000400

531 /*
532  * Flags for FIB print
533  */
534 /* FIB sources */
535 #define AACDB_FLAGS_FIB_SCMD            0x00000001
536 #define AACDB_FLAGS_FIB_IOCTL           0x00000002
537 #define AACDB_FLAGS_FIB_SRB             0x00000004
538 #define AACDB_FLAGS_FIB_SYNC            0x00000008
539 /* FIB components */
540 #define AACDB_FLAGS_FIB_HEADER          0x00000010
541 /* FIB states */
542 #define AACDB_FLAGS_FIB_TIMEOUT         0x00000100

510 extern uint32_t aac_debug_flags;
511 extern int aac_dbflag_on(struct aac_softstate *, int);
512 extern void aac_printf(struct aac_softstate *, uint_t, const char *, ...);
547 extern void aac_print_fib(struct aac_softstate *, struct aac_slot *);

514 #define AACDB_PRINT(s, lev, ...) { \
515         if (aac_dbflag_on((s), AACDB_FLAGS_MISC)) \
516                 aac_printf((s), (lev), __VA_ARGS__); }

518 #define AACDB_PRINT_IOCTL(s, ...) { \
519         if (aac_dbflag_on((s), AACDB_FLAGS_IOCTL)) \
520                 aac_printf((s), CE_NOTE, __VA_ARGS__); }

522 #define AACDB_PRINT_TRAN(s, ...) { \
```

```
523         if (aac_dbflag_on((s), AACDB_FLAGS_SCMD)) \
524                 aac_printf((s), CE_NOTE, __VA_ARGS__); }

526 #define DBCALLED(s, n) { \
527         if (aac_dbflag_on((s), AACDB_FLAGS_FUNC ## n)) \
528                 aac_printf((s), CE_NOTE, "--- %s() called ---", __func__); }
529 #else
530 #define AACDB_PRINT(s, lev, ...)
531 #define AACDB_PRINT_IOCTL(s, ...)
532 #define AACDB_PRINT_TRAN(s, ...)
533 #define DBCALLED(s, n)
534 #endif /* AAC_DEBUG */

536 #ifdef AAC_DEBUG_ALL
537 extern void aac_print_fib(struct aac_softstate *, struct aac_fib *);

539 #define AACDB_PRINT_SCMD(s, x) { \
540         if (aac_dbflag_on((s), AACDB_FLAGS_SCMD)) aac_print_scmd((s), (x)); }

542 #define AACDB_PRINT_AIF(s, x) { \
543         if (aac_dbflag_on((s), AACDB_FLAGS_AIF)) aac_print_aif((s), (x)); }

545 #define AACDB_PRINT_FIB(s, x) { \
546         if (aac_dbflag_on((s), AACDB_FLAGS_FIB)) aac_print_fib((s), (x)); }
547 #else

574 #else /* DEBUG */

576 #define AACDB_PRINT(s, lev, ...)
577 #define AACDB_PRINT_IOCTL(s, ...)
578 #define AACDB_PRINT_TRAN(s, ...)
548 #define AACDB_PRINT_FIB(s, x)
549 #define AACDB_PRINT_SCMD(s, x)
550 #define AACDB_PRINT_AIF(s, x)
551 #endif /* AAC_DEBUG_ALL */
582 #define DBCALLED(s, n)

584 #endif /* DEBUG */

553 #ifdef  __cplusplus
554 }
```

*unchanged_portion_omitted_*

```
**********************************************************
   21215 Thu Nov  1 14:48:27 2012
new/usr/src/uts/common/io/aac/aac_ioctl.c
*** NO COMMENTS ***
**********************************************************
```

```
/*
 * Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 * Copyright (c) 2010, Oracle and/or its affiliates. All rights reserved.
 */

/*
 * Copyright (c) 2010-12 PMC-Sierra, Inc.
 * Copyright (c) 2005-10 Adaptec Inc., Achim Leubner
 * Copyright 2005-06 Adaptec, Inc.
 * Copyright (c) 2005-06 Adaptec Inc., Achim Leubner
 * Copyright (c) 2000 Michael Smith
 * Copyright (c) 2001 Scott Long
 * Copyright (c) 2000 BSDi
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */
#pragma ident   "@(#)aac_ioctl.c        1.9     07/10/30 SMI"

#include <sys/modctl.h>
#include <sys/conf.h>
#include <sys/cmn_err.h>
#include <sys/ddi.h>
#include <sys/devops.h>
#include <sys/pci.h>
#include <sys/types.h>
#include <sys/ddidmareq.h>
#include <sys/scsi/scsi.h>
#include <sys/ksynch.h>
#include <sys/sunddi.h>
#include <sys/byteorder.h>
#include <sys/kmem.h>
#include "aac_regs.h"
#include "aac.h"
#include "aac_ioctl.h"

struct aac_umem_sge {
        uint32_t bcount;
        caddr_t addr;
        struct aac_cmd acp;
};
```



```
/*
 * External functions
 */
extern int aac_sync_mbcommand(struct aac_softstate *, uint32_t, uint32_t,
    uint32_t, uint32_t, uint32_t, uint32_t *, uint32_t *);
    uint32_t, uint32_t, uint32_t, uint32_t *);
extern int aac_cmd_dma_alloc(struct aac_softstate *, struct aac_cmd *,
    struct buf *, int, int (*)(), caddr_t);
extern void aac_free_dmamap(struct aac_cmd *);
extern int aac_do_io(struct aac_softstate *, struct aac_cmd *);
extern void aac_cmd_fib_copy(struct aac_softstate *, struct aac_cmd *);
extern void aac_ioctl_complete(struct aac_softstate *, struct aac_cmd *);
extern int aac_return_aif_wait(struct aac_softstate *, struct aac_fib_context *,
    struct aac_fib **);
extern int aac_return_aif(struct aac_softstate *, struct aac_fib_context *,
    struct aac_fib **);

extern ddi_device_acc_attr_t aac_acc_attr;
extern int aac_check_dma_handle(ddi_dma_handle_t);

/*
 * IOCTL command handling functions
 */
static int aac_check_revision(struct aac_softstate *, intptr_t, int);
static int aac_ioctl_send_fib(struct aac_softstate *, intptr_t, int);
static int aac_open_getadapter_fib(struct aac_softstate *, intptr_t, int);
static int aac_next_getadapter_fib(struct aac_softstate *, intptr_t, int);
static int aac_close_getadapter_fib(struct aac_softstate *, intptr_t);
static int aac_send_raw_srb(struct aac_softstate *, dev_t, intptr_t, int);
static int aac_get_pci_info(struct aac_softstate *, intptr_t, int);
static int aac_query_disk(struct aac_softstate *, intptr_t, int);
static int aac_delete_disk(struct aac_softstate *, intptr_t, int);
static int aac_supported_features(struct aac_softstate *, intptr_t, int);

/*
 * Warlock directives
 */
_NOTE(SCHEME_PROTECTS_DATA("unique to each handling function", aac_features
    aac_pci_info aac_query_disk aac_revision aac_umem_sge))

int
aac_do_ioctl(struct aac_softstate *softs, dev_t dev, int cmd, intptr_t arg,
    int mode)
{
        int status;

        switch (cmd) {
        case FSACTL_MINIPORT_REV_CHECK:
                AACDB_PRINT_IOCTL(softs, "FSACTL_MINIPORT_REV_CHECK");
                status = aac_check_revision(softs, arg, mode);
                break;
        case FSACTL_SENDFIB:
                AACDB_PRINT_IOCTL(softs, "FSACTL_SEND_LARGE_FIB");
                goto send_fib;
        case FSACTL_SEND_LARGE_FIB:
                AACDB_PRINT_IOCTL(softs, "FSACTL_SEND_LARGE_FIB");
send_fib:
                status = aac_ioctl_send_fib(softs, arg, mode);
                break;
        case FSACTL_OPEN_GET_ADAPTER_FIB:
                AACDB_PRINT_IOCTL(softs, "FSACTL_OPEN_GET_ADAPTER_FIB");
                status = aac_open_getadapter_fib(softs, arg, mode);
                break;
        case FSACTL_GET_NEXT_ADAPTER_FIB:
                AACDB_PRINT_IOCTL(softs, "FSACTL_GET_NEXT_ADAPTER_FIB");
```

```
120                 status = aac_next_getadapter_fib(softs, arg, mode);
121                 break;
122         case FSACTL_CLOSE_GET_ADAPTER_FIB:
123                 AACDB_PRINT_IOCTL(softs, "FSACTL_CLOSE_GET_ADAPTER_FIB");
124                 status = aac_close_getadapter_fib(softs, arg);
125                 break;
126         case FSACTL_SEND_RAW_SRB:
127                 AACDB_PRINT_IOCTL(softs, "FSACTL_SEND_RAW_SRB");
128                 status = aac_send_raw_srb(softs, dev, arg, mode);
129                 break;
130         case FSACTL_GET_PCI_INFO:
131                 AACDB_PRINT_IOCTL(softs, "FSACTL_GET_PCI_INFO");
132                 status = aac_get_pci_info(softs, arg, mode);
133                 break;
134         case FSACTL_QUERY_DISK:
135                 AACDB_PRINT_IOCTL(softs, "FSACTL_QUERY_DISK");
136                 status = aac_query_disk(softs, arg, mode);
137                 break;
138         case FSACTL_DELETE_DISK:
139                 AACDB_PRINT_IOCTL(softs, "FSACTL_DELETE_DISK");
140                 status = aac_delete_disk(softs, arg, mode);
141                 break;
142         case FSACTL_GET_FEATURES:
143                 AACDB_PRINT_IOCTL(softs, "FSACTL_GET_FEATURES");
144                 status = aac_supported_features(softs, arg, mode);
145                 break;
146         default:
147                 status = ENOTTY;
148                 AACDB_PRINT(softs, CE_WARN,
149                     "!IOCTL cmd 0x%x not supported", cmd);
150                 break;
151         }

153         return (status);
154 }
```

*unchanged_portion_omitted_*

```
215 static int
216 aac_ioctl_send_fib(struct aac_softstate *softs, intptr_t arg, int mode)
217 {
218         int hbalen;
219         struct aac_cmd *acp;
220         struct aac_fib *fibp;
221         uint16_t fib_command;
222         uint32_t fib_xfer_state;
223         uint16_t fib_data_size, fib_size;
224         uint16_t fib_sender_size;
225         int rval;

227         DBCALLED(softs, 2);

229         /* Copy in FIB header */
230         hbalen = sizeof (struct aac_cmd) + softs->aac_max_fib_size;
231         if ((acp = kmem_zalloc(hbalen, KM_NOSLEEP)) == NULL)
232                 return (ENOMEM);

234         fibp = (struct aac_fib *)(acp + 1);
235         acp->fibp = fibp;
236         if (ddi_copyin((void *)arg, fibp,
237             sizeof (struct aac_fib_header), mode) != 0) {
238                 rval = EFAULT;
239                 goto finish;
240         }

242         fib_xfer_state = LE_32(fibp->Header.XferState);
243         fib_command = LE_16(fibp->Header.Command);
```



```
244         fib_data_size = LE_16(fibp->Header.Size);
245         fib_sender_size = LE_16(fibp->Header.SenderSize);

247         fib_size = fib_data_size + sizeof (struct aac_fib_header);
248         if (fib_size < fib_sender_size)
249                 fib_size = fib_sender_size;
250         if (fib_size > softs->aac_max_fib_size) {
251                 rval = EFAULT;
252                 goto finish;
253         }

255         /* Copy in FIB data */
256         if (ddi_copyin(((struct aac_fib *)arg)->data, fibp->data,
257             fib_data_size, mode) != 0) {
258                 rval = EFAULT;
259                 goto finish;
260         }
261         acp->fib_size = fib_size;
262         fibp->Header.Size = LE_16(fib_size);

264         AACDB_PRINT_FIB(softs, fibp);

266         /* Process FIB */
267         if (fib_command == TakeABreakPt) {
268                 softs->sync_slot_busy = 1;
267 #ifdef DEBUG
268                 if (aac_dbflag_on(softs, AACDB_FLAGS_FIB) &&
269                     (softs->debug_fib_flags & AACDB_FLAGS_FIB_IOCTL))
270                         aac_printf(softs, CE_NOTE, "FIB> TakeABreakPt, sz=%d",
271                             fib_size);
272 #endif
269                 (void) aac_sync_mbcommand(softs, AAC_BREAKPOINT_REQ,
270                     0, 0, 0, 0, NULL, NULL);
274                     0, 0, 0, 0, NULL);
271                 fibp->Header.XferState = LE_32(0);
272         } else {
273                 ASSERT(!(fib_xfer_state & AAC_FIBSTATE_ASYNC));
274                 fibp->Header.XferState = LE_32(fib_xfer_state | \
275                     (AAC_FIBSTATE_FROMHOST | AAC_FIBSTATE_REXPECTED));

277                 acp->timeout = AAC_IOCTL_TIMEOUT;
278                 acp->aac_cmd_fib = aac_cmd_fib_copy;
283 #ifdef DEBUG
284                 acp->fib_flags = AACDB_FLAGS_FIB_IOCTL;
285 #endif
279                 if ((rval = aac_send_fib(softs, acp)) != 0)
280                         goto finish;
281         }

283         if (acp->flags & AAC_CMD_ERR) {
284                 AACDB_PRINT(softs, CE_CONT, "FIB data corrupt");
285                 rval = EIO;
286                 goto finish;
287         }

289         if (ddi_copyout(fibp, (void *)arg, acp->fib_size, mode) != 0) {
290                 AACDB_PRINT(softs, CE_CONT, "FIB copyout failed");
291                 rval = EFAULT;
292                 goto finish;
293         }

295         rval = 0;
296 finish:
297         kmem_free(acp, hbalen);
298         return (rval);
299 }
```

```
static int
aac_open_getadapter_fib(struct aac_softstate *softs, intptr_t arg, int mode)
{
        struct aac_fib_context *fibctx, *ctx;
        struct aac_fib_context *fibctx_p, *ctx_p;

        DBCALLED(softs, 2);

        fibctx = kmem_zalloc(sizeof (struct aac_fib_context), KM_NOSLEEP);
        if (fibctx == NULL)
        fibctx_p = kmem_zalloc(sizeof (struct aac_fib_context), KM_NOSLEEP);
        if (fibctx_p == NULL)
                return (ENOMEM);

        mutex_enter(&softs->aifq_mutex);
        /* All elements are already 0, add to queue */
        if (softs->fibctx == NULL) {
                softs->fibctx = fibctx;
        if (softs->fibctx_p == NULL) {
                softs->fibctx_p = fibctx_p;
        } else {
                for (ctx = softs->fibctx; ctx->next; ctx = ctx->next)
                for (ctx_p = softs->fibctx_p; ctx_p->next; ctx_p = ctx_p->next)
                        ;
                ctx->next = fibctx;
                fibctx->prev = ctx;
                ctx_p->next = fibctx_p;
                fibctx_p->prev = ctx_p;
        }

        /* Evaluate unique value */
        fibctx->unique = (unsigned long)fibctx & 0xffffffffful;
        ctx = softs->fibctx;
        while (ctx != fibctx) {
                if (ctx->unique == fibctx->unique) {
                        fibctx->unique++;
                        ctx = softs->fibctx;
        fibctx_p->unique = (unsigned long)fibctx_p & 0xffffffffful;
        ctx_p = softs->fibctx_p;
        while (ctx_p != fibctx_p) {
                if (ctx_p->unique == fibctx_p->unique) {
                        fibctx_p->unique++;
                        ctx_p = softs->fibctx_p;
                } else {
                        ctx = ctx->next;
                        ctx_p = ctx_p->next;
                }
        }

        /* Set ctx_idx to the oldest AIF */
        if (softs->aifq_wrap) {
                fibctx->ctx_idx = softs->aifq_idx;
                fibctx->ctx_filled = 1;
                fibctx_p->ctx_idx = softs->aifq_idx;
                fibctx_p->ctx_filled = 1;
        }
        mutex_exit(&softs->aifq_mutex);

        if (ddi_copyout(&fibctx->unique, (void *)arg,
        if (ddi_copyout(&fibctx_p->unique, (void *)arg,
            sizeof (uint32_t), mode) != 0)
                return (EFAULT);

        return (0);
}
```



```
static int
aac_return_aif(struct aac_softstate *softs,
    struct aac_fib_context *ctx, caddr_t uptr, int mode)
{
        int current;

        current = ctx->ctx_idx;
        if (current == softs->aifq_idx && !ctx->ctx_filled)
                return (EAGAIN); /* Empty */
        if (ddi_copyout(&softs->aifq[current].d, (void *)uptr,
            sizeof (struct aac_fib), mode) != 0)
                return (EFAULT);

        ctx->ctx_filled = 0;
        ctx->ctx_idx = (current + 1) % AAC_AIFQ_LENGTH;

        return (0);
}

static int
aac_next_getadapter_fib(struct aac_softstate *softs, intptr_t arg, int mode)
{
        union aac_get_adapter_fib_align un;
        struct aac_get_adapter_fib *af = &un.d;
        struct aac_fib_context *ctx;
        struct aac_fib_context *ctx_p;
        struct aac_fib *fibp;
        int rval;

        DBCALLED(softs, 2);

        if (ddi_copyin((void *)arg, af, sizeof (*af), mode) != 0)
                return (EFAULT);

        mutex_enter(&softs->aifq_mutex);
        for (ctx = softs->fibctx; ctx; ctx = ctx->next) {
                if (af->context == ctx->unique)
        for (ctx_p = softs->fibctx_p; ctx_p; ctx_p = ctx_p->next) {
                if (af->context == ctx_p->unique)
                        break;
        }
        if (ctx) {
        mutex_exit(&softs->aifq_mutex);

        if (ctx_p) {
                if (af->wait)
                        rval = aac_return_aif_wait(softs, ctx_p, &fibp);
                else
                        rval = aac_return_aif(softs, ctx_p, &fibp);
        }
        else
                rval = EFAULT;

finish:
        if (rval == 0) {
                if (ddi_copyout(fibp,
#ifdef  _LP64
                rval = aac_return_aif(softs, ctx,
                    (caddr_t)(uint64_t)af->aif_fib, mode);
                    (void *)(uint64_t)af->aif_fib,
#else
                rval = aac_return_aif(softs, ctx,
                    (caddr_t)af->aif_fib, mode);
                    (void *)af->aif_fib,
#endif
```

```
394                 if (rval == EAGAIN && af->wait) {
395                         AACDB_PRINT(softs, CE_NOTE,
396                             "aac_next_getadapter_fib(): waiting for AIF");
397                         rval = cv_wait_sig(&softs->aifv, &softs->aifq_mutex);
398                         if (rval > 0) {
399 #ifdef  _LP64
400                                 rval = aac_return_aif(softs, ctx,
401                                     (caddr_t)(uint64_t)af->aif_fib, mode);
402 #else
403                                 rval = aac_return_aif(softs, ctx,
404                                     (caddr_t)af->aif_fib, mode);
405 #endif
406                         } else {
407                                 rval = EINTR;
408                         }
409                 }
410         } else {
394                     sizeof (struct aac_fib), mode) != 0)
411                 rval = EFAULT;
412         }
413         mutex_exit(&softs->aifq_mutex);

415         return (rval);
416 }

418 static int
419 aac_close_getadapter_fib(struct aac_softstate *softs, intptr_t arg)
420 {
421         struct aac_fib_context *ctx;
403         struct aac_fib_context *ctx_p;

423         DBCALLED(softs, 2);

425         mutex_enter(&softs->aifq_mutex);
426         for (ctx = softs->fibctx; ctx; ctx = ctx->next) {
427                 if (ctx->unique != (uint32_t)arg)
408         for (ctx_p = softs->fibctx_p; ctx_p; ctx_p = ctx_p->next) {
409                 if (ctx_p->unique != (uint32_t)arg)
428                         continue;

430                 if (ctx == softs->fibctx)
431                         softs->fibctx = ctx->next;
412                 if (ctx_p == softs->fibctx_p)
413                         softs->fibctx_p = ctx_p->next;
432                 else
433                         ctx->prev->next = ctx->next;
434                 if (ctx->next)
435                         ctx->next->prev = ctx->prev;
415                         ctx_p->prev->next = ctx_p->next;
416                 if (ctx_p->next)
417                         ctx_p->next->prev = ctx_p->prev;
436                 break;
437         }
438         mutex_exit(&softs->aifq_mutex);
439         if (ctx)
440                 kmem_free(ctx, sizeof (struct aac_fib_context));
421         if (ctx_p)
422                 kmem_free(ctx_p, sizeof (struct aac_fib_context));

442         return (0);
443 }

445 /*
446  * The following function comes from Adaptec:
447  *
448  * SRB is required for the new management tools
```



```
449  * Note: SRB passed down from IOCTL is always in CPU endianness.
450  */
451 static int
452 aac_send_raw_srb(struct aac_softstate *softs, dev_t dev, intptr_t arg, int mode)
453 {
454         struct aac_cmd *acp;
455         struct aac_fib *fibp;
456         struct aac_srb *srb;
457         uint32_t usr_fib_size;
458         uint32_t srb_sgcount;
459         struct aac_umem_sge *usgt = NULL;
460         struct aac_umem_sge *usge;
461         ddi_umem_cookie_t cookie;
462         int umem_flags = 0;
463         int direct = 0;
464         int locked = 0;
465         caddr_t addrlo = (caddr_t)-1;
466         caddr_t addrhi = 0;
467         struct aac_sge *sge, *sge0;
468         int sg64;
469         int rval;

471         DBCALLED(softs, 2);

473         /* Read srb size */
474         if (ddi_copyin(&((struct aac_srb *)arg)->count, &usr_fib_size,
475             sizeof (uint32_t), mode) != 0)
476                 return (EFAULT);
477         if (usr_fib_size > (softs->aac_max_fib_size - \
478             sizeof (struct aac_fib_header)))
479                 return (EINVAL);

481         if ((acp = kmem_zalloc(sizeof (struct aac_cmd) + usr_fib_size + \
482             sizeof (struct aac_fib_header), KM_NOSLEEP)) == NULL)
483                 return (ENOMEM);

485         acp->fibp = (struct aac_fib *)(acp + 1);
486         fibp = acp->fibp;
487         srb = (struct aac_srb *)fibp->data;

489         /* Copy in srb */
490         if (ddi_copyin((void *)arg, srb, usr_fib_size, mode) != 0) {
491                 rval = EFAULT;
492                 goto finish;
493         }

495         srb_sgcount = srb->sg.SgCount; /* No endianness conversion needed */
496         if (srb_sgcount == 0)
497                 goto send_fib;

499         /* Check FIB size */
500         if (usr_fib_size == (sizeof (struct aac_srb) + \
501             srb_sgcount * sizeof (struct aac_sg_entry64) - \
502             sizeof (struct aac_sg_entry))) {
503                 sg64 = 1;
504         } else if (usr_fib_size == (sizeof (struct aac_srb) + \
505             (srb_sgcount - 1) * sizeof (struct aac_sg_entry))) {
506                 sg64 = 0;
507         } else {
508                 rval = EINVAL;
509                 goto finish;
510         }

512         /* Read user SG table */
513         if ((usgt = kmem_zalloc(sizeof (struct aac_umem_sge) * srb_sgcount,
514             KM_NOSLEEP)) == NULL) {
```

```
515                 rval = ENOMEM;
516                 goto finish;
517         }
518         for (usge = usgt; usge < &usgt[srb_sgcount]; usge++) {
519                 if (sg64) {
520                         usge->bcount = ((struct aac_sg_entry64 *)srb-> \
521                             sg.SgEntry)->SgByteCount;
502                         struct aac_sg_entry64 *sg64p =
503                             (struct aac_sg_entry64 *)srb->sg.SgEntry;

505                         usge->bcount = sg64p->SgByteCount;
522                         usge->addr = (caddr_t)
523 #ifndef _LP64
524                             (uint32_t)((struct aac_sg_entry64 *) \
525                             srb->sg.SgEntry)->SgAddress;
526 #else
527                             ((struct aac_sg_entry64 *) \
528                             srb->sg.SgEntry)->SgAddress;
508                             (uint32_t)
529 #endif
510                             sg64p->SgAddress;
530                 } else {
531                         usge->bcount = srb->sg.SgEntry->SgByteCount;
512                         struct aac_sg_entry *sgp = srb->sg.SgEntry;

514                         usge->bcount = sgp->SgByteCount;
532                         usge->addr = (caddr_t)
533 #ifdef _LP64
534                             (uint64_t)
535 #endif
536                             srb->sg.SgEntry->SgAddress;
519                             sgp->SgAddress;
537                 }
538                 acp->bcount += usge->bcount;
539                 if (usge->addr < addrlo)
540                         addrlo = usge->addr;
541                 if ((usge->addr + usge->bcount) > addrhi)
542                         addrhi = usge->addr + usge->bcount;
543         }
544         if (acp->bcount > softs->buf_dma_attr.dma_attr_maxxfer) {
545                 AACDB_PRINT(softs, CE_NOTE,
546                     "large srb xfer size received %d\n", acp->bcount);
547                 rval = EINVAL;
548                 goto finish;
549         }

551         /* Lock user buffers */
552         if (srb->flags & SRB_DataIn) {
553                 umem_flags |= DDI_UMEMLOCK_READ;
554                 direct |= B_READ;
555         }
556         if (srb->flags & SRB_DataOut) {
557                 umem_flags |= DDI_UMEMLOCK_WRITE;
558                 direct |= B_WRITE;
559         }
560         addrlo = (caddr_t)((uintptr_t)addrlo & (uintptr_t)PAGEMASK);
561         rval = ddi_umem_lock(addrlo, (((size_t)addrhi + PAGEOFFSET) & \
562             PAGEMASK) - (size_t)addrlo, umem_flags, &cookie);
563         if (rval != 0) {
564                 AACDB_PRINT(softs, CE_NOTE, "ddi_umem_lock failed: %d",
565                     rval);
566                 goto finish;
567         }
568         locked = 1;

570         /* Allocate DMA for user buffers */
```



```
571         for (usge = usgt; usge < &usgt[srb_sgcount]; usge++) {
572                 struct buf *bp;

574                 bp = ddi_umem_iosetup(cookie, usge->addr - addrlo,
575                     usge->bcount, direct, dev, 0, NULL, DDI_UMEM_SLEEP);
557                 bp = ddi_umem_iosetup(cookie, (uintptr_t)usge->addr - \
558                     (uintptr_t)addrlo, usge->bcount, direct, dev, 0, NULL,
559                     DDI_UMEM_NOSLEEP);
576                 if (bp == NULL) {
577                         AACDB_PRINT(softs, CE_NOTE, "ddi_umem_iosetup failed");
578                         rval = EFAULT;
562                         rval = ENOMEM;
579                         goto finish;
580                 }
581                 if (aac_cmd_dma_alloc(softs, &usge->acp, bp, 0, NULL_FUNC,
582                     0) != AACOK) {
583                         rval = EFAULT;
584                         goto finish;
585                 }
586                 acp->left_cookien += usge->acp.left_cookien;
587                 if (acp->left_cookien > softs->aac_sg_tablesize) {
588                         AACDB_PRINT(softs, CE_NOTE, "large cookiec received %d",
589                             acp->left_cookien);
590                         rval = EINVAL;
591                         goto finish;
592                 }
593         }

595         /* Construct aac cmd SG table */
596         if ((sge = kmem_zalloc(sizeof (struct aac_sge) * acp->left_cookien,
597             KM_NOSLEEP)) == NULL) {
598                 rval = ENOMEM;
599                 goto finish;
600         }
601         acp->sgt = sge;
602         for (usge = usgt; usge < &usgt[srb_sgcount]; usge++) {
603                 for (sge0 = usge->acp.sgt;
604                     sge0 < &usge->acp.sgt[usge->acp.left_cookien];
605                     sge0++, sge++)
606                         *sge = *sge0;
607         }

609 send_fib:
610         acp->cmdlen = srb->cdb_size;
611         acp->timeout = srb->timeout;

613         /* Send FIB command */
614         AACDB_PRINT_FIB(softs, fibp);
615         acp->aac_cmd_fib = softs->aac_cmd_fib_scsi;
599 #ifdef DEBUG
600         acp->fib_flags = AACDB_FLAGS_FIB_SRB;
601 #endif
616         if ((rval = aac_send_fib(softs, acp)) != 0)
617                 goto finish;

619         /* Status struct */
620         if (ddi_copyout((struct aac_srb_reply *)fibp->data,
621             ((uint8_t *)arg + usr_fib_size),
622             sizeof (struct aac_srb_reply), mode) != 0) {
623                 rval = EFAULT;
624                 goto finish;
625         }

627         rval = 0;
628 finish:
629         if (acp->sgt)
```

```
630                 kmem_free(acp->sgt, sizeof (struct aac_sge) * \
631                     acp->left_cookien);
632         if (usgt) {
633                 for (usge = usgt; usge < &usgt[srb_sgcount]; usge++) {
634                         if (usge->acp.sgt)
635                                 kmem_free(usge->acp.sgt,
636                                     sizeof (struct aac_sge) * \
637                                     usge->acp.left_cookien);
638                         aac_free_dmamap(&usge->acp);
639                         if (usge->acp.bp)
640                                 freerbuf(usge->acp.bp);
641                 }
642                 kmem_free(usgt, sizeof (struct aac_umem_sge) * srb_sgcount);
643         }
644         if (locked)
645                 ddi_umem_unlock(cookie);
646         kmem_free(acp, sizeof (struct aac_cmd) + usr_fib_size + \
647             sizeof (struct aac_fib_header));
648         return (rval);
649 }

651 /*ARGSUSED*/
652 static int
653 aac_get_pci_info(struct aac_softstate *softs, intptr_t arg, int mode)
654 {
655         union aac_pci_info_align un;
656         struct aac_pci_info *resp = &un.d;
657         int val;
658         int *props;
659         unsigned int numProps;
643         pci_regspec_t *pci_rp;
644         uint_t num;

661         DBCALLED(softs, 2);

663         val = ddi_prop_lookup_int_array(DDI_DEV_T_ANY,
664                 softs->devinfo_p, 0, "reg", &props, &numProps);
665         if (val != DDI_PROP_SUCCESS || numProps == 0)
666                 return (EFAULT);
648         if (ddi_prop_lookup_int_array(DDI_DEV_T_ANY, softs->devinfo_p,
649             DDI_PROP_DONTPASS, "reg", (int **)&pci_rp, &num) !=
650             DDI_PROP_SUCCESS)
651                 return (EINVAL);
652         if (num < (sizeof (pci_regspec_t) / sizeof (int))) {
653                 ddi_prop_free(pci_rp);
654                 return (EINVAL);
655         }

668         resp->bus = PCI_REG_BUS_G(props[0]);
669         resp->slot = PCI_REG_DEV_G(props[0]);
670         ddi_prop_free(props);
657         resp->bus = PCI_REG_BUS_G(pci_rp->pci_phys_hi);
658         resp->slot = PCI_REG_DEV_G(pci_rp->pci_phys_hi);
659         ddi_prop_free(pci_rp);

672         if (ddi_copyout(resp, (void *)arg,
673             sizeof (struct aac_pci_info), mode) != 0)
674                 return (EFAULT);
675         return (0);
676 }

678 static int
679 aac_query_disk(struct aac_softstate *softs, intptr_t arg, int mode)
680 {
681         union aac_query_disk_align un;
682         struct aac_query_disk *qdisk = &un.d;
```



```
683         struct aac_container *dvp;

685         DBCALLED(softs, 2);

687         if (ddi_copyin((void *)arg, qdisk, sizeof (*qdisk), mode) != 0)
688                 return (EFAULT);

690         if (qdisk->container_no == -1) {
691                 qdisk->container_no = qdisk->target * 16 + qdisk->lun;
692         } else if (qdisk->bus == -1 && qdisk->target == -1 &&
693             qdisk->lun == -1) {
694                 if (qdisk->container_no >= AAC_MAX_CONTAINERS)
695                         return (EINVAL);
696                 qdisk->bus = 0;
697                 qdisk->target = (qdisk->container_no & 0xf);
698                 qdisk->lun = (qdisk->container_no >> 4);
699         } else {
700                 return (EINVAL);
701         }

703         mutex_enter(&softs->io_lock);
704         dvp = &softs->containers[qdisk->container_no];
705         qdisk->valid = dvp->dev.valid;
694         qdisk->valid = AAC_DEV_IS_VALID(&dvp->dev);
706         qdisk->locked = dvp->locked;
707         qdisk->deleted = dvp->deleted;
708         mutex_exit(&softs->io_lock);

710         if (ddi_copyout(qdisk, (void *)arg, sizeof (*qdisk), mode) != 0)
711                 return (EFAULT);
712         return (0);
713 }

715 static int
716 aac_delete_disk(struct aac_softstate *softs, intptr_t arg, int mode)
717 {
718         union aac_delete_disk_align un;
719         struct aac_delete_disk *ddisk = &un.d;
720         struct aac_container *dvp;
721         int rval = 0;

723         DBCALLED(softs, 2);

725         if (ddi_copyin((void *)arg, ddisk, sizeof (*ddisk), mode) != 0)
726                 return (EFAULT);

728         if (ddisk->container_no >= AAC_MAX_CONTAINERS)
729                 return (EINVAL);

731         mutex_enter(&softs->io_lock);
732         dvp = &softs->containers[ddisk->container_no];
733         /*
734          * We don't trust the userland to tell us when to delete
735          * a container, rather we rely on an AIF coming from the
736          * controller.
737          */
738         if (dvp->dev.valid) {
727         if (AAC_DEV_IS_VALID(&dvp->dev)) {
739                 if (dvp->locked)
740                         rval = EBUSY;
741         }
742         mutex_exit(&softs->io_lock);

744         return (rval);
745 }
```

```
747 /*
748  * The following function comes from Adaptec to support creation of arrays
749  * bigger than 2TB.
750  */
751 static int
752 aac_supported_features(struct aac_softstate *softs, intptr_t arg, int mode)
753 {
754         union aac_features_align un;
755         struct aac_features *f = &un.d;

757         DBCALLED(softs, 2);

759         if (ddi_copyin((void *)arg, f, sizeof (*f), mode) != 0)
760                 return (EFAULT);

762         /*
763          * When the management driver receives FSACTL_GET_FEATURES ioctl with
764          * ALL zero in the featuresState, the driver will return the current
765          * state of all the supported features, the data field will not be
766          * valid.
767          * When the management driver receives FSACTL_GET_FEATURES ioctl with
768          * a specific bit set in the featuresState, the driver will return the
769          * current state of this specific feature and whatever data that are
770          * associated with the feature in the data field or perform whatever
771          * action needed indicates in the data field.
772          */
773         if (f->feat.fValue == 0) {
774                 f->feat.fBits.largeLBA =
775                     (softs->flags & AAC_FLAGS_LBA_64BIT) ? 1 : 0;
776                 f->feat.fBits.JBODSupport = 1;
765                 f->feat.fBits.JBODSupport =
766                     (softs->flags & AAC_FLAGS_JBOD) ? 1 : 0;
777                 /* TODO: In the future, add other features state here as well */
778         } else {
779                 if (f->feat.fBits.largeLBA)
780                         f->feat.fBits.largeLBA =
781                             (softs->flags & AAC_FLAGS_LBA_64BIT) ? 1 : 0;
772                 if (f->feat.fBits.JBODSupport)
773                         f->feat.fBits.JBODSupport =
774                             (softs->flags & AAC_FLAGS_JBOD) ? 1 : 0;
782                 /* TODO: Add other features state and data in the future */
783         }

785         if (ddi_copyout(f, (void *)arg, sizeof (*f), mode) != 0)
786                 return (EFAULT);
787         return (0);
788 }
```

_____unchanged_portion_omitted_

```
**********************************************************
    4849 Thu Nov  1 14:48:27 2012
new/usr/src/uts/common/io/aac/aac_ioctl.h
*** NO COMMENTS ***
**********************************************************
```

```
/*
 * Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 * Copyright (c) 2010, Oracle and/or its affiliates. All rights reserved.
 */

/*
 * Copyright (c) 2010-11 PMC-Sierra, Inc.
 * Copyright (c) 2005-10 Adaptec Inc., Achim Leubner
 * Copyright 2005-06 Adaptec, Inc.
 * Copyright (c) 2005-06 Adaptec Inc., Achim Leubner
 * Copyright (c) 2000 Michael Smith
 * Copyright (c) 2000 Scott Long
 * Copyright (c) 2000 BSDi
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 *
 *    $FreeBSD: /repoman/r/ncvs/src/sys/sys/aac_ioctl.h,
 *    v 1.11 2004/12/09 22:20:25 scottl Exp $
 */

#ifndef _AAC_IOCTL_H_
#define _AAC_IOCTL_H_

#pragma ident   "@(#)aac_ioctl.h        1.3     07/10/30 SMI"

#ifdef __cplusplus
extern "C" {
#endif

/*
 * IOCTL Interface
 */

/* Macro definitions for IOCTL function control codes */
#define CTL_CODE(function, method) \
        ((4<< 16) | ((function) << 2) | (method))

/* Method codes for how buffers are passed for I/O and FS controls */
#define METHOD_BUFFERED         0
#define METHOD_NEITHER          3
```



```
/* IOCTL commands */
#define FSACTL_SENDFIB                  CTL_CODE(2050, METHOD_BUFFERED)
#define FSACTL_SEND_RAW_SRB             CTL_CODE(2067, METHOD_BUFFERED)
#define FSACTL_DELETE_DISK              0x163
#define FSACTL_QUERY_DISK               0x173
#define FSACTL_OPEN_GET_ADAPTER_FIB     CTL_CODE(2100, METHOD_BUFFERED)
#define FSACTL_GET_NEXT_ADAPTER_FIB     CTL_CODE(2101, METHOD_BUFFERED)
#define FSACTL_CLOSE_GET_ADAPTER_FIB    CTL_CODE(2102, METHOD_BUFFERED)
#define FSACTL_MINIPORT_REV_CHECK       CTL_CODE(2107, METHOD_BUFFERED)
#define FSACTL_GET_PCI_INFO             CTL_CODE(2119, METHOD_BUFFERED)
#define FSACTL_FORCE_DELETE_DISK        CTL_CODE(2120, METHOD_NEITHER)
#define FSACTL_REGISTER_FIB_SEND        CTL_CODE(2136, METHOD_BUFFERED)
#define FSACTL_GET_CONTAINERS           2131
#define FSACTL_GET_VERSION_MATCHING     CTL_CODE(2137, METHOD_BUFFERED)
#define FSACTL_SEND_LARGE_FIB           CTL_CODE(2138, METHOD_BUFFERED)
#define FSACTL_GET_FEATURES             CTL_CODE(2139, METHOD_BUFFERED)

#pragma pack(1)

struct aac_revision
{
        uint32_t compat;
        uint32_t version;
        uint32_t build;
};

struct aac_get_adapter_fib
{
        uint32_t context;
        int wait;
        int32_t wait;
        uint32_t aif_fib;       /* RAID config app is 32bit */
};
______unchanged_portion_omitted_

/*
 * The following definitions come from Adaptec:
 */
typedef union {
        struct {
                uint32_t largeLBA  : 1; /* disk support greater 2TB */
                uint32_t IoctlBuf  : 1; /* ARCIOCTL call support */
                uint32_t AIFSupport: 1; /* AIF support */
                uint32_t JBODSupport:1; /* firmware + driver both support JBOD *
                uint32_t JBODSupport:1; /* firmware+driver both support JBOD */
                uint32_t fReserved : 28;
        } fBits;
        uint32_t fValue;
} featuresState;
______unchanged_portion_omitted_

#pragma pack()

/*
 * Aligned structure definitions for variable declarations that require
 * alignment.
 *
 * Normally the packed structures are defined in a way that if the initial
 * member is aligned, then the following members will also be aligned. So
 * we need only to make the packed structure, ie. the first member, is
 * we need only to make sure the packed structure, ie. the first member, is
 * aligned to satisfy alignment requirement.
 */
union aac_revision_align {
        struct aac_revision d;
```

```
 146         uint32_t dummy;
 147 };
```

*unchanged_portion_omitted_*

```
**********************************************************
   53903 Thu Nov  1 14:48:28 2012
new/usr/src/uts/common/io/aac/aac_regs.h
*** NO COMMENTS ***
**********************************************************
/*
 * Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 * Copyright (c) 2010, Oracle and/or its affiliates. All rights reserved.
 */

/*
 * Copyright (c) 2010-12 PMC-Sierra, Inc.
 * Copyright (c) 2005-10 Adaptec Inc., Achim Leubner
 * Copyright 2005-06 Adaptec, Inc.
 * Copyright (c) 2005-06 Adaptec Inc., Achim Leubner
 * Copyright (c) 2000 Michael Smith
 * Copyright (c) 2000-2001 Scott Long
 * Copyright (c) 2000 BSDi
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 *
 *    $FreeBSD: src/sys/dev/aac/aacreg.h,v 1.23 2005/10/14 16:22:45 scottl Exp $
 */

#ifndef __AAC_REGS_H__
#define __AAC_REGS_H__

#pragma ident   "@(#)aac_regs.h 1.12    08/01/29 SMI"

#ifdef  __cplusplus
extern "C" {
#endif

/* Status bits in the doorbell registers */
#define AAC_DB_SYNC_COMMAND             (1<<0)  /* send/completed synchronous */
                                                /* FIB */
#define AAC_DB_COMMAND_READY            (1<<1)  /* posted one or more */
                                                /* commands */
#define AAC_DB_RESPONSE_READY           (1<<2)  /* one or more commands */
                                                /* complete */
#define AAC_DB_COMMAND_NOT_FULL         (1<<3)  /* command queue not full */
#define AAC_DB_RESPONSE_NOT_FULL        (1<<4)  /* response queue not full */
#define AAC_DB_PRINTF_READY             (1<<5)  /* adapter requests host */
                                                /* printf */
#define AAC_DB_AIF_PENDING              (1<<6)  /* pending AIF (new comm. type1)
```



```
/* PMC specific outbound doorbell bits */
#define AAC_DB_RESPONSE_SENT_NS         (1<<1)  /* response sent (not shifted) *

#define AAC_DB_INTR_BITS (AAC_DB_COMMAND_READY | \
        AAC_DB_RESPONSE_READY | AAC_DB_PRINTF_READY)
#define AAC_DB_INTR_NEW                 0x08
#define AAC_DB_INTR_NEW_TYPE1   0x04

/* Status bits in firmware status reg */
#define AAC_SELF_TEST_FAILED            0x00000004
#define AAC_MONITOR_PANIC               0x00000020
#define AAC_KERNEL_UP_AND_RUNNING       0x00000080
#define AAC_KERNEL_PANIC                0x00000100

/* aac registers definitions */
#define AAC_OMR0                0x18    /* outbound message register 0 */
#define AAC_OMR1                0x1c    /* outbound message register 1 */
#define AAC_IDBR                0x20    /* inbound doorbell reg */
#define AAC_ODBR                0x2c    /* outbound doorbell reg */
#define AAC_OIMR                0x34    /* outbound interrupt mask reg */
#define AAC_IRCSR               0x38    /* inbound dual cores reset (SRL) */
#define AAC_IQUE                0x40    /* inbound queue */
#define AAC_OQUE                0x44    /* outbound queue */
#define AAC_RX_MAILBOX          0x50    /* mailbox, size=20bytes, rx */
#define AAC_RX_FWSTATUS         0x6c    /* firmware status, rx */
#define AAC_RKT_MAILBOX         0x1000  /* mailbox, size=20bytes, rkt */
#define AAC_RKT_FWSTATUS        0x101c  /* firmware status, rkt */

/*
 * Register definitions for the Adaptec PMC SRC adapters.
 */
/* accessible via BAR0 */
#define AAC_SRC_OMR                     0xbc    /* outbound message register */
#define AAC_SRC_IDBR            0x20    /* inbound doorbell register */
#define AAC_SRC_IISR            0x24    /* inbound interrupt status register */
#define AAC_SRC_ODBR_R          0x9c    /* outbound doorbell register read */
#define AAC_SRC_ODBR_C          0xa0    /* outbound doorbell register clear */
#define AAC_SRC_OIMR            0x34    /* outbound interrupt mask register */
#define AAC_SRC_IQUE32          0x40    /* inbound queue address 32-bit */
#define AAC_SRC_IQUE64_L        0xc0    /* inbound queue address 64-bit (low) */
#define AAC_SRC_IQUE64_H        0xc4    /* inbound queue address 64-bit (high) *

#define AAC_SRC_MAILBOX         0x7fc60 /* mailbox (20 bytes) */
#define AAC_SRCV_MAILBOX        0x1000  /* mailbox (20 bytes) */

#define AAC_SRC_ODR_SHIFT       12              /* outbound doorbell shift */
#define AAC_SRC_IDR_SHIFT       9               /* inbound doorbell shift */

#define AAC_MAX_MEM_SPACE       3       /* max number of PCI mem spaces */

/* Synchronous commands to the monitor/kernel. */
#define AAC_BREAKPOINT_REQ      0x04
#define AAC_MONKER_INITSTRUCT   0x05
#define AAC_MONKER_SYNCFIB      0x0c
#define AAC_MONKER_GETKERNVER   0x11
#define AAC_MONKER_GETINFO      0x19
#define AAC_MONKER_GETDRVPROP   0x23
#define AAC_MONKER_GETCOMMPREF  0x26
#define AAC_IOP_RESET           0x1000
#define AAC_IOP_RESET_ALWAYS    0x1001

/* Sunrise Lake dual core reset */
#define AAC_IRCSR_CORES_RST     3

#define AAC_SECTOR_SIZE         512
#define AAC_NUMBER_OF_HEADS     255
```

```
123 #define AAC_SECTORS_PER_TRACK   63
124 #define AAC_ROTATION_SPEED      10000
125 #define AAC_MAX_PFN             0xfffff

127 #define AAC_ADDITIONAL_LEN      31
128 #define AAC_ANSI_VER            2
129 #define AAC_RESP_DATA_FORMAT    2

131 #define AAC_MAX_LD              64      /* max number of logical disks */
132 #define AAC_MAX_PD(s)           ((s)->bus_max * (s)->tgt_max)
133 #define AAC_MAX_DEV(s)          (AAC_MAX_LD + AAC_MAX_PD((s)))
134 #define AAC_BLK_SIZE            AAC_SECTOR_SIZE
135 #define AAC_DMA_ALIGN           4
136 #define AAC_DMA_ALIGN_MASK      (AAC_DMA_ALIGN - 1)

138 #define AAC_MAX_CONTAINERS      AAC_MAX_LD

140 /*
141  * Minimum memory sizes we need to map to address the adapter. Before
142  * we know the actual size to map, minimum memory is used instead.
143  */
144 #define AAC_MAP_SIZE_MIN_RX     4096
145 #define AAC_MAP_SIZE_MIN_RKT    8192
146 #define AAC_MAP_SIZE_MIN_SRC_BAR0       0x400000
147 #define AAC_MAP_SIZE_MIN_SRC_BAR1       0x800
148 #define AAC_MAP_SIZE_MIN_SRCV_BAR0      0x100000
149 #define AAC_MAP_SIZE_MIN_SRCV_BAR1      0x400

151 /*
152  * Options supported by the adapter
153  */
154 #define AAC_SUPPORTED_SNAPSHOT                  0x01
155 #define AAC_SUPPORTED_CLUSTERS                  0x02
156 #define AAC_SUPPORTED_WRITE_CACHE               0x04
157 #define AAC_SUPPORTED_64BIT_DATA                0x08
158 #define AAC_SUPPORTED_HOST_TIME_FIB             0x10
159 #define AAC_SUPPORTED_RAID50                    0x20
160 #define AAC_SUPPORTED_4GB_WINDOW                0x40
161 #define AAC_SUPPORTED_SCSI_UPGRADEABLE          0x80
162 #define AAC_SUPPORTED_SOFT_ERR_REPORT           0x100
163 #define AAC_SUPPORTED_NOT_RECONDITION           0x200
164 #define AAC_SUPPORTED_SGMAP_HOST64              0x400
165 #define AAC_SUPPORTED_ALARM                     0x800
166 #define AAC_SUPPORTED_NONDASD                   0x1000
167 #define AAC_SUPPORTED_SCSI_MANAGED              0x2000
168 #define AAC_SUPPORTED_RAID_SCSI_MODE            0x4000
169 #define AAC_SUPPORTED_SUPPLEMENT_ADAPTER_INFO   0x10000
170 #define AAC_SUPPORTED_NEW_COMM                  0x20000
171 #define AAC_SUPPORTED_64BIT_ARRAYSIZE           0x40000
172 #define AAC_SUPPORTED_HEAT_SENSOR               0x80000
173 #define AAC_SUPPORTED_NEW_COMM_TYPE1    0x10000000  /* Tupelo new comm */
174 #define AAC_SUPPORTED_NEW_COMM_TYPE2    0x20000000  /* Denali new comm */
175 #define AAC_SUPPORTED_NEW_COMM_TYPE3    0x40000000  /* Series 8 new comm */
176 #define AAC_SUPPORTED_NEW_COMM_TYPE4    0x80000000  /* Series 9 new comm */

178 /*
179  * More options from supplement info - SupportedOptions2
180  */
181 #define AAC_SUPPORTED_MU_RESET                  0x01
182 #define AAC_SUPPORTED_IGNORE_RESET              0x02
183 #define AAC_SUPPORTED_POWER_MANAGEMENT          0x04
184 #define AAC_SUPPORTED_ARCIO_PHYDEV              0x08
185 #define AAC_SUPPORTED_DOORBELL_RESET            0x4000


188 /*
```



```
189  * FeatureBits of RequestSupplementAdapterInfo used in the driver
190  */
191 #define AAC_SUPPL_SUPPORTED_JBOD        0x08000000
151 #define AAC_FEATURE_SUPPORTED_JBOD              0x08000000

193 #pragma pack(1)

195 /* transport FIB header (PMC) */
196 struct aac_fib_xporthdr {
197         uint64_t        HostAddress;    /* FIB host address w/o xport header */
198         uint32_t        Size;                   /* FIB size excluding xport head
199         uint32_t        Handle;                 /* driver handle to reference th
200         uint64_t        Reserved[2];
201 };

203 /*
204  * FIB (FSA Interface Block) this is the data structure passed between
205  * the host and adapter.
206  */
207 struct aac_fib_header {
208         uint32_t        XferState;
209         uint16_t        Command;
210         uint8_t         StructType;
211         uint8_t         Unused;
163         uint8_t         Flags;
212         uint16_t        Size;
213         uint16_t        SenderSize;
214         uint32_t        SenderFibAddress;
215         union {
216                 uint32_t        ReceiverFibAddress;
217                 uint32_t        SenderFibAddressHigh;
218                 uint32_t        TimeStamp;
219         } a;
220         uint32_t        Handle;
221         uint32_t        Previous;
222         uint32_t        Next;
168         uint32_t        SenderData;
169         int prev;
170         int next;
223 };
```

*unchanged_portion_omitted_*

```
238 /* FIB transfer state */
239 #define AAC_FIBSTATE_HOSTOWNED          (1<<0)  /* owned by the host */
240 #define AAC_FIBSTATE_ADAPTEROWNED       (1<<1)  /* owned by the adapter */
241 #define AAC_FIBSTATE_INITIALISED        (1<<2)  /* has been initialised */
242 #define AAC_FIBSTATE_EMPTY              (1<<3)  /* is empty now */
243 #define AAC_FIBSTATE_FROMHOST           (1<<5)  /* sent from the host */
244 #define AAC_FIBSTATE_FROMADAP           (1<<6)  /* sent from the adapter */
245 #define AAC_FIBSTATE_REXPECTED          (1<<7)  /* response is expected */
246 #define AAC_FIBSTATE_NOREXPECTED        (1<<8)  /* no response is expected */
247 #define AAC_FIBSTATE_DONEADAP           (1<<9)  /* processed by the adapter */
248 #define AAC_FIBSTATE_DONEHOST           (1<<10) /* processed by the host */
249 #define AAC_FIBSTATE_NORM               (1<<12) /* normal priority */
250 #define AAC_FIBSTATE_ASYNC              (1<<13)
251 #define AAC_FIBSTATE_FAST_RESPONSE      (1<<19) /* fast response capable */
252 #define AAC_FIBSTATE_NOMOREAIF          (1<<21)

254 /* FIB types */
255 #define AAC_FIBTYPE_TFIB                1
256 #define AAC_FIBTYPE_TFIB2               4
257 #define AAC_FIBTYPE_TFIB2_64    5

259 /*
260  * FIB commands
261  */
```

```
263 #define TestCommandResponse             1
264 #define TestAdapterCommand              2
265 /* Lowlevel and comm commands */
266 #define LastTestCommand                 100
267 #define ReinitHostNormCommandQueue      101
268 #define ReinitHostHighCommandQueue      102
269 #define ReinitHostHighRespQueue         103
270 #define ReinitHostNormRespQueue         104
271 #define ReinitAdapNormCommandQueue      105
272 #define ReinitAdapHighCommandQueue      107
273 #define ReinitAdapHighRespQueue         108
274 #define ReinitAdapNormRespQueue         109
275 #define InterfaceShutdown               110
276 #define DmaCommandFib                   120
277 #define StartProfile                    121
278 #define TermProfile                     122
279 #define SpeedTest                       123
280 #define TakeABreakPt                    124
281 #define RequestPerfData                 125
282 #define SetInterruptDefTimer            126
283 #define SetInterruptDefCount            127
284 #define GetInterruptDefStatus           128
285 #define LastCommCommand                 129
286 /* Filesystem commands */
287 #define NuFileSystem                    300
288 #define UFS                             301
289 #define HostFileSystem                  302
290 #define LastFileSystemCommand           303
291 /* Container commands */
292 #define ContainerCommand                500
293 #define ContainerCommand64              501
294 #define RawIo                           502
295 #define RawIo2                          503
296 /* Cluster commands */
297 #define ClusterCommand                  550
298 /* Scsi Port commands (scsi passthrough) */
299 #define ScsiPortCommand                 600
300 #define ScsiPortCommandU64              601
301 /* Misc house keeping and generic adapter initiated commands */
302 #define AifRequest                      700
303 #define CheckRevision                   701
304 #define FsaHostShutdown                 702
305 #define RequestAdapterInfo              703
306 #define IsAdapterPaused                 704
307 #define SendHostTime                    705
308 #define RequestSupplementAdapterInfo    706
309 #define LastMiscCommand                 707
310 #define OnLineDiagnostic                800
311 #define FduAdapterTest                  801

313 /*
314  * Revision number handling
315  */
316 struct FsaRev {
317         union {
318                 struct {
319                         uint8_t dash;
320                         uint8_t type;
321                         uint8_t minor;
322                         uint8_t major;
323                 } comp;
324                 uint32_t ul;
325         } external;
326         uint32_t buildNumber;
327 };
```

*unchanged_portion_omitted_*


```
423 /* Flag values for ContentState */
424 #define AAC_FSCS_NOTCLEAN       0x1     /* fscheck is necessary before mounting
425 #define AAC_FSCS_READONLY       0x2     /* possible result of broken mirror */
426 #define AAC_FSCS_HIDDEN         0x4     /* container should be ignored by driver
427 #define AAC_FSCS_NOT_READY      0x8     /* cnt is in spinn. state, not rdy for I

429 struct aac_mntobj {
430         uint32_t                ObjectId;
431         char                    FileSystemName[16];
432         struct aac_container_creation   CreateInfo;
433         uint32_t                Capacity;
434         uint32_t                VolType;
435         uint32_t                ObjType;
436         uint32_t                ContentState;
437         union {
438                 uint32_t        pad[8];
439         } ObjExtension;
440         uint32_t                AlterEgoId;

442         uint32_t                CapacityHigh; /* 64-bit LBA */
443 };
```

*unchanged_portion_omitted_*

```
514 /*
515  * Container Configuration Sub-Commands
516  */
517 typedef enum {
518         CT_Null = 0,
519         CT_GET_SLICE_COUNT,             /* 1 */
520         CT_GET_PARTITION_COUNT,         /* 2 */
521         CT_GET_PARTITION_INFO,          /* 3 */
522         CT_GET_CONTAINER_COUNT,         /* 4 */
523         CT_GET_CONTAINER_INFO_OLD,      /* 5 */
524         CT_WRITE_MBR,                   /* 6 */
525         CT_WRITE_PARTITION,             /* 7 */
526         CT_UPDATE_PARTITION,            /* 8 */
527         CT_UNLOAD_CONTAINER,            /* 9 */
528         CT_CONFIG_SINGLE_PRIMARY,       /* 10 */
529         CT_READ_CONFIG_AGE,             /* 11 */
530         CT_WRITE_CONFIG_AGE,            /* 12 */
531         CT_READ_SERIAL_NUMBER,          /* 13 */
532         CT_ZERO_PAR_ENTRY,              /* 14 */
533         CT_READ_MBR,                    /* 15 */
534         CT_READ_PARTITION,              /* 16 */
535         CT_DESTROY_CONTAINER,           /* 17 */
536         CT_DESTROY2_CONTAINER,          /* 18 */
537         CT_SLICE_SIZE,                  /* 19 */
538         CT_CHECK_CONFLICTS,             /* 20 */
539         CT_MOVE_CONTAINER,              /* 21 */
540         CT_READ_LAST_DRIVE,             /* 22 */
541         CT_WRITE_LAST_DRIVE,            /* 23 */
542         CT_UNMIRROR,                    /* 24 */
543         CT_MIRROR_DELAY,                /* 25 */
544         CT_GEN_MIRROR,                  /* 26 */
545         CT_GEN_MIRROR2,                 /* 27 */
546         CT_TEST_CONTAINER,              /* 28 */
547         CT_MOVE2,                       /* 29 */
548         CT_SPLIT,                       /* 30 */
549         CT_SPLIT2,                      /* 31 */
550         CT_SPLIT_BROKEN,                /* 32 */
551         CT_SPLIT_BROKEN2,               /* 33 */
552         CT_RECONFIG,                    /* 34 */
553         CT_BREAK2,                      /* 35 */
554         CT_BREAK,                       /* 36 */
555         CT_MERGE2,                      /* 37 */
```

```
556         CT_MERGE,                       /* 38 */
557         CT_FORCE_ERROR,                 /* 39 */
558         CT_CLEAR_ERROR,                 /* 40 */
559         CT_ASSIGN_FAILOVER,             /* 41 */
560         CT_CLEAR_FAILOVER,              /* 42 */
561         CT_GET_FAILOVER_DATA,           /* 43 */
562         CT_VOLUME_ADD,                  /* 44 */
563         CT_VOLUME_ADD2,                 /* 45 */
564         CT_MIRROR_STATUS,               /* 46 */
565         CT_COPY_STATUS,                 /* 47 */
566         CT_COPY,                        /* 48 */
567         CT_UNLOCK_CONTAINER,            /* 49 */
568         CT_LOCK_CONTAINER,              /* 50 */
569         CT_MAKE_READ_ONLY,              /* 51 */
570         CT_MAKE_READ_WRITE,             /* 52 */
571         CT_CLEAN_DEAD,                  /* 53 */
572         CT_ABORT_MIRROR_COMMAND,        /* 54 */
573         CT_SET,                         /* 55 */
574         CT_GET,                         /* 56 */
575         CT_GET_NVLOG_ENTRY,             /* 57 */
576         CT_GET_DELAY,                   /* 58 */
577         CT_ZERO_CONTAINER_SPACE,        /* 59 */
578         CT_GET_ZERO_STATUS,             /* 60 */
579         CT_SCRUB,                       /* 61 */
580         CT_GET_SCRUB_STATUS,            /* 62 */
581         CT_GET_SLICE_INFO,              /* 63 */
582         CT_GET_SCSI_METHOD,             /* 64 */
583         CT_PAUSE_IO,                    /* 65 */
584         CT_RELEASE_IO,                  /* 66 */
585         CT_SCRUB2,                      /* 67 */
586         CT_MCHECK,                      /* 68 */
587         CT_CORRUPT,                     /* 69 */
588         CT_GET_TASK_COUNT,              /* 70 */
589         CT_PROMOTE,                     /* 71 */
590         CT_SET_DEAD,                    /* 72 */
591         CT_CONTAINER_OPTIONS,           /* 73 */
592         CT_GET_NV_PARAM,                /* 74 */
593         CT_GET_PARAM,                   /* 75 */
594         CT_NV_PARAM_SIZE,               /* 76 */
595         CT_COMMON_PARAM_SIZE,           /* 77 */
596         CT_PLATFORM_PARAM_SIZE,         /* 78 */
597         CT_SET_NV_PARAM,                /* 79 */
598         CT_ABORT_SCRUB,                 /* 80 */
599         CT_GET_SCRUB_ERROR,             /* 81 */
600         CT_LABEL_CONTAINER,             /* 82 */
601         CT_CONTINUE_DATA,               /* 83 */
602         CT_STOP_DATA,                   /* 84 */
603         CT_GET_PARTITION_TABLE,         /* 85 */
604         CT_GET_DISK_PARTITIONS,         /* 86 */
605         CT_GET_MISC_STATUS,             /* 87 */
606         CT_GET_CONTAINER_PERF_INFO,     /* 88 */
607         CT_GET_TIME,                    /* 89 */
608         CT_READ_DATA,                   /* 90 */
609         CT_CTR,                         /* 91 */
610         CT_CTL,                         /* 92 */
611         CT_DRAINIO,                     /* 93 */
612         CT_RELEASEIO,                   /* 94 */
613         CT_GET_NVRAM,                   /* 95 */
614         CT_GET_MEMORY,                  /* 96 */
615         CT_PRINT_CT_LOG,                /* 97 */
616         CT_ADD_LEVEL,                   /* 98 */
617         CT_NV_ZERO,                     /* 99 */
618         CT_READ_SIGNATURE,              /* 100 */
619         CT_THROTTLE_ON,                 /* 101 */
620         CT_THROTTLE_OFF,                /* 102 */
621         CT_GET_THROTTLE_STATS,          /* 103 */
```



```
622         CT_MAKE_SNAPSHOT,               /* 104 */
623         CT_REMOVE_SNAPSHOT,             /* 105 */
624         CT_WRITE_USER_FLAGS,            /* 106 */
625         CT_READ_USER_FLAGS,             /* 107 */
626         CT_MONITOR,                     /* 108 */
627         CT_GEN_MORPH,                   /* 109 */
628         CT_GET_SNAPSHOT_INFO,           /* 110 */
629         CT_CACHE_SET,                   /* 111 */
630         CT_CACHE_STAT,                  /* 112 */
631         CT_TRACE_START,                 /* 113 */
632         CT_TRACE_STOP,                  /* 114 */
633         CT_TRACE_ENABLE,                /* 115 */
634         CT_TRACE_DISABLE,               /* 116 */
635         CT_FORCE_CORE_DUMP,             /* 117 */
636         CT_SET_SERIAL_NUMBER,           /* 118 */
637         CT_RESET_SERIAL_NUMBER,         /* 119 */
638         CT_ENABLE_RAID5,                /* 120 */
639         CT_CLEAR_VALID_DUMP_FLAG,       /* 121 */
640         CT_GET_MEM_STATS,               /* 122 */
641         CT_GET_CORE_SIZE,               /* 123 */
642         CT_CREATE_CONTAINER_OLD,        /* 124 */
643         CT_STOP_DUMPS,                  /* 125 */
644         CT_PANIC_ON_TAKE_A_BREAK,       /* 126 */
645         CT_GET_CACHE_STATS,             /* 127 */
646         CT_MOVE_PARTITION,              /* 128 */
647         CT_FLUSH_CACHE,                 /* 129 */
648         CT_READ_NAME,                   /* 130 */
649         CT_WRITE_NAME,                  /* 131 */
650         CT_TOSS_CACHE,                  /* 132 */
651         CT_LOCK_DRAINIO,                /* 133 */
652         CT_CONTAINER_OFFLINE,           /* 134 */
653         CT_SET_CACHE_SIZE,              /* 135 */
654         CT_CLEAN_SHUTDOWN_STATUS,       /* 136 */
655         CT_CLEAR_DISKLOG_ON_DISK,       /* 137 */
656         CT_CLEAR_ALL_DISKLOG,           /* 138 */
657         CT_CACHE_FAVOR,                 /* 139 */
658         CT_READ_PASSTHRU_MBR,           /* 140 */
659         CT_SCRUB_NOFIX,                 /* 141 */
660         CT_SCRUB2_NOFIX,                /* 142 */
661         CT_FLUSH,                       /* 143 */
662         CT_REBUILD,     /* 144 rma, not really a command, partner to CT_SCRUB */
663         CT_FLUSH_CONTAINER,             /* 145 */
664         CT_RESTART,                     /* 146 */
665         CT_GET_CONFIG_STATUS,           /* 147 */
666         CT_TRACE_FLAG,                  /* 148 */
667         CT_RESTART_MORPH,               /* 149 */
668         CT_GET_TRACE_INFO,              /* 150 */
669         CT_GET_TRACE_ITEM,              /* 151 */
670         CT_COMMIT_CONFIG,               /* 152 */
671         CT_CONTAINER_EXISTS,            /* 153 */
672         CT_GET_SLICE_FROM_DEVT,         /* 154 */
673         CT_OPEN_READ_WRITE,             /* 155 */
674         CT_WRITE_MEMORY_BLOCK,          /* 156 */
675         CT_GET_CACHE_PARAMS,            /* 157 */
676         CT_CRAZY_CACHE,                 /* 158 */
677         CT_GET_PROFILE_STRUCT,          /* 159 */
678         CT_SET_IO_TRACE_FLAG,           /* 160 */
679         CT_GET_IO_TRACE_STRUCT,         /* 161 */
680         CT_CID_TO_64BITS_UID,           /* 162 */
681         CT_64BITS_UID_TO_CID,           /* 163 */
682         CT_PAR_TO_64BITS_UID,           /* 164 */
683         CT_CID_TO_32BITS_UID,           /* 165 */
684         CT_32BITS_UID_TO_CID,           /* 166 */
685         CT_PAR_TO_32BITS_UID,           /* 167 */
686         CT_SET_FAILOVER_OPTION,         /* 168 */
687         CT_GET_FAILOVER_OPTION,         /* 169 */
```

```
688         CT_STRIPE_ADD2,                 /* 170 */
689         CT_CREATE_VOLUME_SET,           /* 171 */
690         CT_CREATE_STRIPE_SET,           /* 172 */
691         /* 173  command and partner to scrub and rebuild task types */
692         CT_VERIFY_CONTAINER,
693         CT_IS_CONTAINER_DEAD,           /* 174 */
694         CT_GET_CONTAINER_OPTION,        /* 175 */
695         CT_GET_SNAPSHOT_UNUSED_STRUCT,  /* 176 */
696         CT_CLEAR_SNAPSHOT_UNUSED_STRUCT,        /* 177 */
697         CT_GET_CONTAINER_INFO,          /* 178 */
698         CT_CREATE_CONTAINER,            /* 179 */
699         CT_CHANGE_CREATIONINFO,         /* 180 */
700         CT_CHECK_CONFLICT_UID,          /* 181 */
701         CT_CONTAINER_UID_CHECK,         /* 182 */

703         /* 183 :RECmm: 20011220 added to support the Babylon */
704         CT_IS_CONTAINER_MEATADATA_STANDARD,
705         /* 184 :RECmm: 20011220 array imports */
706         CT_IS_SLICE_METADATA_STANDARD,

708         /* :BIOS_TEST: */
709         /* 185 :RECmm: 20020116 added to support BIOS interface for */
710         CT_GET_IMPORT_COUNT,
711         /* 186 :RECmm: 20020116 metadata conversion */
712         CT_CANCEL_ALL_IMPORTS,
713         CT_GET_IMPORT_INFO,             /* 187 :RECmm: 20020116 " */
714         CT_IMPORT_ARRAY,                /* 188 :RECmm: 20020116 " */
715         CT_GET_LOG_SIZE,                /* 189  */

717         /* Not BIOS TEST */
718         CT_ALARM_GET_STATE,             /* 190 */
719         CT_ALARM_SET_STATE,             /* 191 */
720         CT_ALARM_ON_OFF,                /* 192 */

722         CT_GET_EE_OEM_ID,               /* 193 */

724         CT_GET_PPI_HEADERS,             /* 194  get header fields only */
725         CT_GET_PPI_DATA,                /* 195  get all ppitable.data */
726         /* 196  get only range of entries specified in c_params */
727         CT_GET_PPI_ENTRIES,
728         /* 197  remove ppitable bundle specified by uid in c_param0 */
729         CT_DELETE_PPI_BUNDLE,

731         /* 198  current partition structure (not legacy) */
732         CT_GET_PARTITION_TABLE_2,
733         CT_GET_PARTITION_INFO_2,
734         CT_GET_DISK_PARTITIONS_2,

736         CT_QUIESCE_ADAPTER,             /* 201  chill dude */
737         CT_CLEAR_PPI_TABLE,             /* 202  clear ppi table */

739         CT_SET_DEVICE_CACHE_POLICY,     /* 203 */
740         CT_GET_DEVICE_CACHE_POLICY,     /* 204 */

742         CT_SET_VERIFY_DELAY,            /* 205 */
743         CT_GET_VERIFY_DELAY,            /* 206 */

745         /* 207 delete all PPI bundles that have an entry for device at devt */
746         CT_DELETE_PPI_BUNDLES_FOR_DEVT,

748         CT_READ_SW_SECTOR,              /* 208 */
749         CT_WRITE_SW_SECTOR,             /* 209 */

751         /* 210 added to support firmware cache sync operations */
752         CT_GET_CACHE_SYNC_INFO,
753         CT_SET_CACHE_SYNC_MODE,         /* 211 */
```



```
754         CT_GET_PARTITION_TABLE_500,
755         CT_GET_PARTITION_INFO_500,
756         CT_GET_DISK_PARTITIONS_500,
692         CT_PM_DRIVER_SUPPORT,           /* 212 */
693         CT_PM_CONFIGURATION,            /* 213 */

758         CT_GET_RAID_CONFIG,                     /* 215 */
759         CT_COPYBACK_ENABLE,
760         CT_CONTAINER_OPTIONS_500,

762         CT_MARK_BAD_STRIPES,            /* 218 */

764         CT_GET_RAID6_OPTIONS_500,       /* 219 */

766         CT_GET_SS_MAP_INFO,                     /* 220 */

768         CT_CREATE_CONTAINER_64,         /* 221 */
769         CT_COPY_STATUS_64,
770         CT_GET_SNAPSHOT_INFO_64,

772         CT_ENABLE_RAID_ENHANCED,        /* 224 */
773
774         CT_GET_CONTAINER_SPITFIRE_SIZE, /* 225 */
775         CT_STAMP_SPITFIRE_VM,

777         CT_GET_BST_INFO,                        /* 227 */
778         CT_ERASE_BAD_STRIPE_TABLE,

780         CT_GET_CONTAINER_LIST,          /* 229 */
781
782         CT_FREE_FOR_USE,                        /* 230 */

784         CT_GET_HIST_LOG_SIZE,           /* 231 */
785         CT_GET_HIST_LOG_ENTRY,

787         CT_CONTINUE_DATA_STOP,          /* 233 */

789         CT_GET_LOGDEV_INFO,                     /* 234 */
790         CT_GET_LOGDEV_SEGMENT_LIST,

792         CT_SET_MINIARC_HOST_MEMORY_SEGMENTS,    /* 236 */

794         CT_CREATE_LOGICAL_DRIVE,        /* 237 */

796         CT_SET_CONTROLLER_DEVICE_CACHE_POLICY,  /* 238 */
797         CT_GET_CONTROLLER_DEVICE_CACHE_POLICY,

799         CT_GET_MISC_STATUS_V3,          /* 240 */

801         CT_IDENTIFY_DEVICE,                     /* 241 */

803         CT_CREATE_JBOD,                         /* 242 */
804         CT_DELETE_JBOD,

806         CT_GET_STATISTIC_DATA,          /* 244 */

808         CT_PM_DRIVER_SUPPORT,           /* 245 */
809         CT_PM_CONFIGURATION,
810         CT_GET_PHYDEV_LIST,
811         CT_GET_PHYDEV_INFO,
812         CT_GET_LOGDEV_SEGMENT64_LIST,

814         CT_LAST_COMMAND                 /* last command */
815 } AAC_CTCommand;
```

_____***unchanged_portion_omitted_***

```
 828 struct aac_fsa_ctm {
 829         uint32_t        command;
 830         uint32_t        param[CT_FIB_PARAMS];
 831         int8_t          data[CT_PACKET_SIZE];
 832 };
______unchanged_portion_omitted_

 912 /*
 913  * Command status values
 914  */
 915 typedef enum {
 916         ST_OK = 0,
 917         ST_PERM = 1,
 918         ST_NOENT = 2,
 919         ST_IO = 5,
 920         ST_NXIO = 6,
 921         ST_E2BIG = 7,
 922         ST_ACCES = 13,
 923         ST_EXIST = 17,
 924         ST_XDEV = 18,
 925         ST_NODEV = 19,
 926         ST_NOTDIR = 20,
 927         ST_ISDIR = 21,
 928         ST_INVAL = 22,
 929         ST_FBIG = 27,
 930         ST_NOSPC = 28,
 931         ST_ROFS = 30,
 932         ST_MLINK = 31,
 933         ST_WOULDBLOCK = 35,
 934         ST_NAMETOOLONG = 63,
 935         ST_NOTEMPTY = 66,
 936         ST_DQUOT = 69,
 937         ST_STALE = 70,
 938         ST_REMOTE = 71,
 939         ST_NOT_READY = 72,
 940         ST_BADHANDLE = 10001,
 941         ST_NOT_SYNC = 10002,
 942         ST_BAD_COOKIE = 10003,
 943         ST_NOTSUPP = 10004,
 944         ST_TOOSMALL = 10005,
 945         ST_SERVERFAULT = 10006,
 946         ST_BADTYPE = 10007,
 947         ST_JUKEBOX = 10008,
 948         ST_NOTMOUNTED = 10009,
 949         ST_MAINTMODE = 10010,
 950         ST_STALEACL = 10011
 951 } AAC_FSAStatus;
______unchanged_portion_omitted_

1024 struct aac_sge_ieee1212 {
1025         uint32_t        addrLow;
1026         uint32_t        addrHigh;
1027         uint32_t        length;
1028         uint32_t        flags;          /* reserved */
1029 };

1031 struct aac_sg_table {
1032         uint32_t                SgCount;
1033         struct aac_sg_entry     SgEntry[1]; /* at least there is one */
1034                                             /* SUN's CC cannot accept [0] */
1035 };
______unchanged_portion_omitted_

1109 #define RIO2_IO_TYPE            0x0003
1110 #define RIO2_IO_TYPE_WRITE      0x0000
```



```
1111 #define RIO2_IO_TYPE_READ       0x0001
1112 #define RIO2_IO_TYPE_VERIFY     0x0002
1113 #define RIO2_IO_ERROR           0x0004
1114 #define RIO2_IO_SUREWRITE       0x0008
1115 #define RIO2_SGL_CONFORMANT     0x0010
1116 #define RIO2_SG_FORMAT          0xF000
1117 #define RIO2_SG_FORMAT_ARC      0x0000
1118 #define RIO2_SG_FORMAT_SRL      0x1000
1119 #define RIO2_SG_FORMAT_IEEE1212 0x2000

1121 struct aac_raw_io2 {
1122         uint32_t                strtBlkLow;
1123         uint32_t                strtBlkHigh;
1124         uint32_t                byteCnt;
1125         uint16_t                ldNum;
1126         uint16_t                flags;                  /* RIO2_xxx */
1127         uint32_t                sgeFirstSize;   /* size of first S/G el. */
1128         uint32_t                sgeNominalSize; /* size of 2nd S/G el. */
1129         uint8_t                 sgeCnt;
1130         uint8_t                 bpTotal;                /* following elements re
1131         uint8_t                 bpComplete;
1132         uint8_t                 sgeFirstIndex;
1133         uint8_t                 unused[4];
1134         struct aac_sge_ieee1212 sge[1]; /* S/G list */
1135 };

1137 /*
1138  * Container shutdown command.
1139  */
1140 struct aac_close_command {
1141         uint32_t                Command;
1142         uint32_t                ContainerId;
1143 };
______unchanged_portion_omitted_

1160 /* Write 'stability' options */
1161 #define CSTABLE                 1
1162 #define CUNSTABLE               2

1164 /* Number of FIBs for the controller to send us messages */
1165 #define AAC_ADAPTER_FIBS        8

1167 /* Number of FIBs for the host I/O request */
1168 #define AAC_HOST_FIBS           256

1170 /* Size of buffer for text messages from the controller */
1171 #define AAC_ADAPTER_PRINT_BUFSIZE               256

1173 #define AAC_INIT_STRUCT_REVISION                3
1174 #define AAC_INIT_STRUCT_REVISION_4              4
1175 #define AAC_INIT_STRUCT_REVISION_6              6
1176 #define AAC_INIT_STRUCT_REVISION_7              7
1177 #define AAC_INIT_STRUCT_MINIPORT_REVISION       1

1178 #define AAC_INIT_FLAGS_NEW_COMM_SUPPORTED       1
1179 #define AAC_INIT_FLAGS_DRIVER_USES_UTC_TIME     0x10
1180 #define AAC_INIT_FLAGS_DRIVER_SUPPORTS_PM       0x20
1181 #define AAC_INITFLAGS_NEW_COMM_TYPE1_SUPPORTED  0x40
1182 #define AAC_INITFLAGS_DRIVER_SUPPORTS_FAST_JBOD  0x80
1183 #define AAC_INITFLAGS_NEW_COMM_TYPE2_SUPPORTED  0x100

1184 #define AAC_PAGE_SIZE                           4096
1185 struct aac_adapter_init {
1186         uint32_t        InitStructRevision;
1187         uint32_t        MiniPortRevision;
1188         uint32_t        FilesystemRevision;
```

```
1189         uint32_t        CommHeaderAddress;
1190         uint32_t        FastIoCommAreaAddress;
1191         uint32_t        AdapterFibsPhysicalAddress;
1192         uint32_t        AdapterFibsVirtualAddress;
1193         uint32_t        AdapterFibsSize;
1194         uint32_t        AdapterFibAlign;
1195         uint32_t        PrintfBufferAddress;
1196         uint32_t        PrintfBufferSize;
1197         uint32_t        HostPhysMemPages;
1198         uint32_t        HostElapsedSeconds;
1199         /* ADAPTER_INIT_STRUCT_REVISION_4 begins here */
1200         uint32_t        InitFlags;
1201         uint32_t        MaxIoCommands;
1202         uint32_t        MaxIoSize;
1203         uint32_t        MaxFibSize;
1204         /* ADAPTER_INIT_STRUCT_REVISION_5 begins here */
1205         uint32_t        MaxNumAif;              /* max number of aif */
1206         /* ADAPTER_INIT_STRUCT_REVISION_6 begins here */
1207         uint32_t        HostRRQ_AddrLow;
1208         uint32_t        HostRRQ_AddrHigh;       /* Host RRQ (response queue) for
1209 };
```

_____*unchanged_portion_omitted_*

```
1404 /*
1405  * Event Notification
1406  */
1407 typedef enum {
1408         /* General application notifies start here */
1409         AifEnGeneric = 1,       /* Generic notification */
1410         AifEnTaskComplete,      /* Task has completed */
1411         AifEnConfigChange,      /* Adapter config change occurred */
1412         AifEnContainerChange,   /* Adapter specific container cfg. change */
1413         AifEnDeviceFailure,     /* SCSI device failed */
1414         AifEnMirrorFailover,    /* Mirror failover started */
1415         AifEnContainerEvent,    /* Significant container event */
1416         AifEnFileSystemChange,  /* File system changed */
1417         AifEnConfigPause,       /* Container pause event */
1418         AifEnConfigResume,      /* Container resume event */
1419         AifEnFailoverChange,    /* Failover space assignment changed */
1420         AifEnRAID5RebuildDone,  /* RAID5 rebuild finished */
1421         AifEnEnclosureManagement, /* Enclosure management event */
1422         AifEnBatteryEvent,      /* Significant NV battery event */
1423         AifEnAddContainer,      /* A new container was created. */
1424         AifEnDeleteContainer,   /* A container was deleted. */
1425         AifEnSMARTEvent,        /* SMART Event */
1426         AifEnBatteryNeedsRecond, /* The battery needs reconditioning */
1427         AifEnClusterEvent,      /* Some cluster event */
1428         AifEnDiskSetEvent,      /* A disk set event occured. */
1429         AifEnContainerScsiEvent, /* a container event with no. and scsi id */
1430         AifEnPicBatteryEvent,   /* An event gen. by pic_battery.c for an ABM */
1431         AifEnExpEvent,          /* Exp. Event Type to replace CTPopUp mess. */
1432         AifEnRAID6RebuildDone,  /* RAID6 rebuild finished */
1433         AifEnSensorOverHeat,    /* Heat Sensor indicate overheat */
1434         AifEnSensorCoolDown,    /* Heat Sensor ind. cooled down after overheat*/
1435         AifFeatureKeysModified, /* notif. of updated feature keys */
1436         AifApplicationExpirationEvent,  /* notif. on app. expiration status */
1437         AifEnBackgroundConsistencyCheck, /* BCC notif. for NEC - DDTS 94700 */
1438         AifEnAddJBOD,           /* A new JBOD type drive was created (30) */
1439         AifEnDeleteJBOD,        /* A JBOD type drive was deleted (31) */
1265         AifEnAddJBOD = 30,      /* A new JBOD type drive was created (30) */
1266         AifEnDeleteJBOD = 31,   /* A JBOD type drive was deleted (31) */
1440         AifDriverNotifyStart = 199, /* Notifies for host driver go here */
1441         /* Host driver notifications start here */
1442         AifDenMorphComplete,    /* A morph operation completed */
1443         AifDenVolumeExtendComplete, /* Volume expand operation completed */
1444         AifDriverNotifyDelay,
```



```
1445         AifRawDeviceRemove                     /* Raw device Failure event */
1270         AifDenVolumeExtendComplete      /* Volume expand operation completed */
1446 } AAC_AifEventNotifyType;
```

_____*unchanged_portion_omitted_*

```
1521 /*
1522  * Adapter Initiated FIB command structures. Start with the adapter
1523  * initiated FIBs that really come from the adapter, and get responded
1524  * to by the host.
1525  */
1526 #define AAC_AIF_REPORT_MAX_SIZE 64

1528 typedef enum {
1529         AifCmdEventNotify = 1,  /* Notify of event */
1530         AifCmdJobProgress,      /* Progress report */
1531         AifCmdAPIReport,        /* Report from other user of API */
1532         AifCmdDriverNotify,     /* Notify host driver of event */
1533         AifReqJobList = 100,    /* Gets back complete job list */
1534         AifReqJobsForCtr,       /* Gets back jobs for specific container */
1535         AifReqJobsForScsi,      /* Gets back jobs for specific SCSI device */
1536         AifReqJobReport,        /* Gets back a specific job report or list */
1537         AifReqTerminateJob,     /* Terminates job */
1538         AifReqSuspendJob,       /* Suspends a job */
1539         AifReqResumeJob,        /* Resumes a job */
1540         AifReqSendAPIReport,    /* API generic report requests */
1541         AifReqAPIJobStart,      /* Start a job from the API */
1542         AifReqAPIJobUpdate,     /* Update a job report from the API */
1543         AifReqAPIJobFinish,     /* Finish a job from the API */
1544         AifReqEvent = 200       /* PMC NEW COMM: Request the event data */
1368         AifReqAPIJobFinish      /* Finish a job from the API */
1545 } AAC_AifCommand;
```

_____*unchanged_portion_omitted_*

```
1704 #define AAC_SENSE_BUFFERSIZE     30

1706 struct aac_srb_reply
1707 {
1708         uint32_t status;
1709         uint32_t srb_status;
1710         uint32_t scsi_status;
1711         uint32_t data_xfer_length;
1712         uint32_t sense_data_size;
1713         uint8_t sense_data[AAC_SENSE_BUFFERSIZE];    /* Can this be */
1714                                                     /* SCSI_SENSE_BUFFERSIZE */
1537         uint8_t sense_data[AAC_SENSE_BUFFERSIZE];
1715 };
```

_____*unchanged_portion_omitted_*

```
1797 /* AAC Communication Space */
1798 struct aac_comm_space {
1799         struct aac_fib adapter_fibs[AAC_ADAPTER_FIBS];
1800         struct aac_adapter_init init_data;
1801         struct aac_queue_table qtable;
1802         char qt_align_pad[AAC_QUEUE_ALIGN];
1803         char adapter_print_buf[AAC_ADAPTER_PRINT_BUFSIZE];
1804     /* response buffer for SRC (new comm. type1) - must be last element */
1805     uint32_t aac_host_rrq[1];
1806 };
```

_____*unchanged_portion_omitted_*