PCIHSD® Interface Board
PCIHSD2® Interface Board
cPCIHSD™ Interface Board

Software Device Driver
for Linux – Kernel Version 2.2.x, 2.4.x
User Manual

Document Number: 0900121
Rev. 1.2

APPLIED DATA SCIENCES, INC. has contained in this subject matter proprietary material and all manufacturing, reproduction, use and sales rights pertaining to this material are expressly reserved. This material is submitted in confidence for a specified purpose and the user, by accepting this material, agrees that it will not be used, copied or reproduced in whole or in part except to meet the purpose for which it was provided.

APPLIED DATA SCIENCES, INC. reserves the right to make design changes or modification to any product to improve performance or incorporate new functions.
The material in this document is for informational purposes and is subject to change without notice.

APPLIED DATA SCIENCES, INC. assumes no responsibility for any errors that may appear in this document.

TO CONTACT US

Our mailing address is:
Applied Data Sciences, Inc.
P.O. Box 814209
Dallas, TX 75381-4209
USA

Our shipping address is:
Applied Data Sciences, Inc.
1300 North I-35E, Suite 100
Carrollton, TX 75006
USA

Our telephone numbers are:
972-242-7944 (USA & international)

Our facsimile number is:
972-242-8874 (USA & international)

Our email address is:
support@appdatsci.com

Copyright © 2000 by
APPLIED DATA SCIENCES, INC.
All Rights Reserved.
Printed in the U.S.A.

REVISION STATUS

REV	PAGES CHANGED	DESCRIPTION	APPR	DATE
1.0	----	Initial Release	RLT	08/31/00
1.1	viii,2-1, 2-5, 3-2, 5-1, 5-3,B-1,B-2	Changes in ver 1.5	RLT	08/09/01
1.2	vii, 1-2, 4-2, 5-4	Changes in ver 1.6	TFC	12/07/01

LIST OF RELATED DOCUMENTS

TITLE	DOCUMENT NUMBER
Encore CSD	
HSDII Technical Manual	303-000270-200
HSDII Hardware Reference Manual	301-320050-000
MPX-32 Reference Manual, Volume I	323-001011-300
Applied Data Sciences	
PCIHSD Interface Board-Technical Manual	0900098
PCIHSD2 Interface Board Technical Manual	0900117

TRADEMARK ACKNOWLEDGEMENTS

PCIHSD is a registered trademark of Applied Data Sciences.
PCIHSD2 is a registered trademark of Applied Data Sciences.

TABLE OF CONTENTS

SECTION		TITLE	PAGE
1.0	**INTRODUCTION**		
	1.1	PCIHSD Board Description	1-1
	1.2	Encore HSDII Board Description/Emulation	1-2
	1.3	Software Overview	1-3
	1.4	Requirements	1-3
2.0	**DRIVER FUNCTIONS**		
	2.1	General Information	2-1
	2.2	PCIHSD Devices in Linux	2-1
	2.3	IBL Mode Support	2-2
	2.4	Symbol Definitions	2-2
	2.5	OPEN Function	2-2
	2.6	CLOSE Function	2-2
	2.7	READ Function	2-3
	2.8	WRITE Function	2-4
	2.9	Data Transfer Options	2-5
	2.10	IOCTL Function	2-5
	2.10.1	HSDGETMOD [Get PCIHSD Operating Mode]	2-6
	2.10.2	HSDSETMOD [Set PCIHSD Operating Mode]	2-6
	2.10.3	HSDRESET [Perform PCIHSD Reset Operations]	2-6
	2.10.4	HSDGETSTS [Read PCIHSD Board Status]	2-7
	2.10.5	HSDGETXSTS [Get Extended Driver Status]	2-7
3.0	**DATA STRUCTURES**		
	3.1	Introduction	3-1
	3.2	Reference Documents	3-1
	3.3	pcihsd_mode_t Structure	3-1
	3.4	pcihsd_sts_t Structure	3-3

4.0 STATUS RETURNS

4.1	Status Returns	4-1
4.2	Extended Status Returns	4-1
4.3	PCIHSD Board Status Returns	4-2

5.0 PROGRAM INSTALLATION

5.1	Distribution Media	5-1
5.2	Installation	5-1
5.2.1	Completing the Installation	5-2
5.2.2	Installation Details	5-3
5.2.3	Command Line Parameters	5-4

APPENDICES

APPENDIX	TITLE	PAGE
A	Example Programs	A-1
B	SETHSD Utility	B-1

SECTION 1.0
INTRODUCTION

1.1 PCIHSD Board Description

The PCIHSD® or PCIHSD2® board provides a high-speed bi-directional link for transferring control, status, and data between a 32-bit Peripheral Component Interconnect (PCI) bus compatible system, and the Encore High-Speed Digital Interface (HSDII) or any 32-bit external device using the Encore High-Speed Digital Interface (HSDII) protocol.

The PCIHSD2 is a half-length PCI board residing in a 32-bit 5-volt PCI slot in the host computer chassis. There is one (1) 100-pin connector on the rear edge of the PCIHSD2 board. This 100-pin connector attaches to a mating 100-pin connector (with latches) which in turn has crimped to it (2) 50-pin flat ribbon cables. These two flat ribbon cables have 50-pin IDC female connectors at the opposite ends. Using the supplied 50-pin flat ribbon cable harness, connection is made between the PCIHSD2 board, and the HSDII card or HSD compatible device.

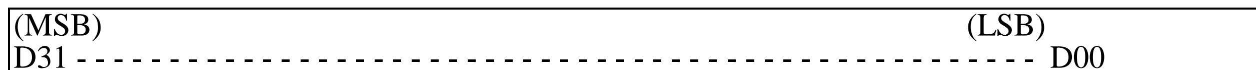
There are no hardware switches. All setup and control is handled via software. The PCIHSD can be configured for the following modes:

- HSD,
- IBL or
- Device EM (External Mode).

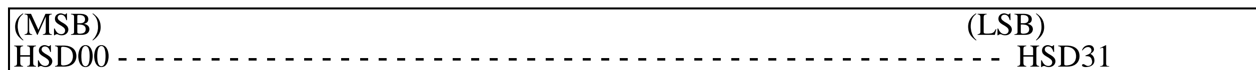
In the HSD mode the PCIHSD2 board emulates the Encore HSDII board. In the IBL mode the PCIHSD2 board can be attached directly to an Encore HSDII board or to another PCIHSD2 board. In the Device External Mode the PCIHSD2 board emulates the device driving the Encore HSDII board in External Mode.

This high-speed data link provides a 32-bit data bus transfer between the host computer memory and the PCIHSD2 board. The data are 32-bit word transfers between the PCIHSD2 board and HSDII compatible external device. The relationship between the most significant and least significant bits for the host computer PCI bus and the HSD words are illustrated in the following sketch. Linux is designed to operate on a variety of platforms; which are either big-endian (most significant bits of a number appear in the lowest numbered byte) or little-endian (least significant bits of a number appear in the lowest numbered byte) in their architecture. The PCI bus is little-endian in its design. Under Linux, the driver is compiled to recognize the difference and to set the PCIHSD board appropriately.

******** One PCI 32-Bit Word *********



******** is equivalent to ********



******** One HSD 32-Bit Word ********

For additional technical specifications of this board refer to the PCIHSD2 Technical Manual, part number 09000117.

This software can also be used with the older PCIHSD board, Part Number 0700520, which is electrically equivalent to the PCIHSD, as well as with the cPCIHSD, Part Number 0700705, in a CompactPCI bus system. In the following sections, the term PCIHSD refers equally to the PCIHSD, PCIHSD2 and cPCIHSD.

1.2 Encore HSDII Board Description/Emulation

This information is presented to help you understand the purpose of the PCIHSD board - that is to emulate the functions of the Encore HSDII board. The HSDII board is a high-speed general purpose interface board, which is installed in an Encore computer chassis. This board is manufactured by Encore, Inc. Computer Systems Division, Fort Lauderdale, Florida.

The Encore HSDII board provides a full 32-bit parallel interface to an external device which operates autonomously under control of an Input/Output Control List (IOCL). A detailed description of the HSDII board is found in the HSDII Technical Manual and HSDII Hardware Reference Manual. The document part numbers of these two manuals are contained in the Index under LIST OF RELATED DOCUMENTS.

The PCIHSD also provides a 32-bit parallel interface to an external device with the capability of both Programmed I/O (PIO) and Direct Memory Access (DMA) data transfers. Some functions performed by the Encore HSDII must be emulated by host software in the context of the PCIHSD. Once the host has initiated a DMA data transfer, the external device executes transfers between the PCIHSD's onboard FIFO and the external device independent of the host's operation. The handshake seen by the external device appears functionally identical to the Encore HSDII card.

1.3 Software Overview

This document provides all the information necessary to integrate the user's software with the PCIHSD.

The PCIHSD driver, "**PCIHSD**", is intended for operation under Linux, version 2.2.x. The driver handles all communication with the PCIHSD on behalf of the calling program. It allows for reading and setting the operating mode of the PCIHSD. It provides synchronous (wait mode) data transfers via **read**(2) and **write**(2) calls. Device command and status request operations are performed by reads and writes at unique file offset addresses, operations that are easily performed using the **pread**(2) and **pwrite**(2) calls.

Details of the driver installation are in Section 5.0.

1.4 Requirements

Software:	Linux Operating System, version 2.2.x, including "C" compiler.
Hardware:	Personal Computer with PCI bus. PCIHSD Card.

Knowledge of the Encore HSDII IOCB structure will aid in using the PCIHSD board and software.

SECTION 2.0
DRIVER FUNCTIONS

2.1 General Information

The PCIHSD driver is installed as a standard Linux device driver. It can be compiled to handle any number of PCIHSDs configured in either HSD or IBL mode. The following system calls are supported:

OPEN
CLOSE
READ
WRITE
IOCTL

Arguments to the IOCTL call provide for reading and/or setting the device operation mode, for reading status from the PCIHSD, and resetting the PCIHSD.

2.2 PCIHSD Devices in Linux

Under Linux, a PCIHSD board is represented as character device accessed through a node found the **/dev** directory of the filesystem. The name of the node is used in the **open**(2) call to access the device. When the driver is loaded into the kernel, it allows the kernel to assign a major number for the nodes. The minor number for a node reflects the relative position of the board on the bus. The driver assumes the minor number is of the form 4*U+M, where U is the board (unit) number and M indicates the board's operating mode. Unit numbers start at zero and increment by one for each PCIHSD board in the system. When the driver is loaded, it searches for all the PCIHSD boards in the system and numbers them as they are found. Currently defined modes are 0 for normal operation, and 3 for "configuration/diagnostic" device. The configuration device is for diagnostic functions and may not be used for any I/O operation.

The following is a typical entry for a two, PCIHSD board system:

/dev/pcihsd0	Character device, minor number 0 -- first board.
/dev/pcihsd0c	Character device, minor number 3 -- first board configuration
/dev/pcihsd1	Character device, minor number 4 -- second board.
/dev/pcihsd1c	Character device, minor number 7 -- second board configuration

2.3 IBL Mode Support

The PCIHSD driver provides three protocols for initiating a link for data transfers in the InterBus Link (IBL) configuration. The desired mode is specified in the *link* field of a *pcihsd_mode_t* structure passed to an HSDSETMOD call. If PCIHM_LREQ is specified, the PCIHSD will initiate the link request for all transfers. If PCIHM_LACK is specified, the PCIHSD does not initiate any link request, but expects to wait for, and acknowledge a link request before each data transfer. A third option uses the protocol of the Encore H.IBLG handler under MPX and initiates the link request when the operation is a Write or waits for and acknowledges a link request when the operation is a Read. This method is specified by PCIHM_LIBLG in the *link* field or the *pcihsd_mode_t* structure.

2.4 Symbol Definitions

The header file **pcihsdio.h** contains definitions of symbols used with the **ioctl**(2) system call to perform various HSD functions, as well as declarations for data structures used with those calls.

2.5 OPEN Function

The standard **open**(2) function must be called to associate a file descriptor with the desired device file. No specific meaning is attached to the "**flag**" or "**mode**" arguments. Only one process may open any one physical PCIHSD device at a time. In addition to errors which may normally be returned from the **open** call, a value of EBUSY (in *errno*) may be returned if the addressed device is already in use. Opening the PCIHSD device does not configure the board for either HSD or IBL operations. Any user should set all values for the operating mode of the board to ensure that the board will operate in the manner the user expects.

2.6 CLOSE Function

The standard **close**(2) function must be called to terminate access to the desired device file. Any pending operations for the open device are cancelled and the associated memory areas released when the device is closed.

2.7 READ Function

The standard **read**(2) function is used to transfer data from the PCIHSD to the user's buffer or to retrieve device status. Data transfers are buffered into kernel space before being moved to the user's buffer. The user's transfer byte count must be a multiple of four. The maximum length of a single transfer is 262,140 bytes (65535 or $FFFF_{16}$ 32-bit words). This is the maximum that can be transferred by a single IOCB. If the requested transfer size exceeds the maximum, the driver will return an error (EINVAL). The extended error code, which may be retrieved via an HSDGETXSTS call, will be EXLONG (transfer count too long).

The IOCB command word presented to the external device to initiate the data transfer contains 8 bits (bits 16-23) of user-device-dependent (UDD) information. Unless the PCIHM_NOUDD flag bit is set (see Section 2.9) the least-significant 8 bits of the current file position will be used as the UDD byte for the read IOCB. The **pread**(2) call is a convenient way to set the file position and transfer data, effectively combining calls to **lseek**(2) and **read**(2). The UDDBYTE macro, defined in **pcihsdio.h**, is useful in conjunction with these calls, for example:

```
nread = pread (fd, &buffer, bfrlen, UDDBYTE(0x20));
```

An external device status request may be performed by issuing a **read**(2) or **pread**(2) call with the file position set to the hexadecimal value 20xxxxxx (where the x's are arbitrary values). The file position is used as the IOCB command word for the request and the external device status word (4 bytes only) is returned to the user's buffer. The R_DSR macro defined in **pcihsdio.h** generates the required file position word:

```
uint        devstat;

status = pread(fd, &devstat, 4, R_DSR(0x440000));
```

This call will issue a device status request with 44_{16} in the UDD byte, returning the status word to **devstat**.

In addition to the errors, which may normally be returned from the **read** call, the following additional conditions may be reported by *errno* values:

EINVAL	Requested transfer length is greater than the maximum allowed or is not a multiple of four bytes (1 longword).
EIO	An error occurred in the data transfer.

Further explanation may be obtained by issuing the **ioctl**(HSDGETXSTS) call and examining the returned value. (See Section 2.10.5.)

2.8 WRITE Function

The standard **write**(2) function is used to transfer data from the user's buffer to the PCIHSD or to send device commands. Both device command transfers and data transfers are buffered through the driver. The user's transfer byte count must be a multiple of four for data transfers. Device command transfers must be a multiple of 8 bytes long. The maximum length of a single transfer is 262,140 bytes (65535 or $FFFF_{16}$ 32-bit words). This is the maximum that can be transferred by a single IOCB. If the requested transfer size exceeds the maximum, the driver will return an error (EINVAL). The extended error code, which may be retrieved via an HSDGETXSTS call, will be EXLONG (transfer count too long).

The IOCB command word presented to the external device to initiate the data transfer contains 8 bits (bits 16-23) of user-device-dependent (UDD) information. Unless the PCIHM_NOUDD flag bit is set (see Section 2.9) the least-significant 8 bits of the current file position will be used as the UDD byte for the read IOCB. The **pwrite**(2) call is a convenient way to set the file position and transfer data, effectively combining calls to **lseek**(2) and **write**(2). The UDDBYTE macro, defined in **pcihsdio.h**, is useful in conjunction with these calls, for example:

```
nwrt = pwrite (fd, &buffer, bfrlen, UDDBYTE(0x20));
```

One or more commands may be sent to the external device by issuing a **write**(2) or **pwrite**(2) call with the file position set to the hexadecimal value 40xxxxxx (where the x's are arbitrary values) and the transfer count a multiple of 8 bytes. Each successive 8 bytes (2 HSD words) from the user's buffer are sent to the PCIHSD as the two IOCB words of a device command. The W_CMD macro defined in **pcihsdio.h** generates the required file position word:

```
uint            devcmd[2]={0x40010002, 0x12345678};

status = pwrite(fd, devcmd, 8, W_CMD);
```

This call will send the device command contained in the **devcmd** array.

In addition to the errors, which may normally be returned from the **write**(2) call, the following additional conditions may be reported by *errno* values:

EINVAL	Requested transfer length is greater than the maximum allowable or is not a multiple of four bytes (1 longword).
EIO	An error occurred in the data transfer.

Further explanation may be obtained by issuing the **ioctl**(HSDGETXSTS) call and examining the returned value. (See Section 2.10.5.)

2.9 Data Transfer Options

Read or **write** options are specified by flag bits set in the *flags* field of a *pcihsd_mode_t* structure passed to an **ioctl**(HSDSETMOD) call:

PCIHM_HZTO	Interpret timeout values as clock ticks (HZ ticks per second) instead of seconds.
PCIHM_NOUDD	Do not set the UDD byte in read/write transfers from the current file position.

2.10 IOCTL Function

The **ioctl**(2) call is used to read PCIHSD status and perform reset operations. The **ioctl** call requires the user to pass the file descriptor of an open device file, a function code and an argument. All **ioctl** calls are performed synchronously. That is, the operation is completed and any status information to be returned to the caller is stored before control returns from the **ioctl** call.

Any of these functions may return a value of -1, with the variable *errno* set to one of the following error codes:

EFAULT	If some part of a data structure passed as an argument is not accessible to the user.
EINVAL	If some field in a data structure passed as an argument, is invalid or inappropriate.
ENXIO	If the requested operation is illogical or impossible.
EIO	If some I/O error occurred on the PCIHSD.
EBUSY	If active I/O precludes performing the current request.
EPERM	If the caller does not have the requisite privilege for the requested operation.
EINTR	If a signal was caught during the course of the current function call.

Each subsection below discusses one of the **ioctl** functions and describes the required argument(s). Detailed information about the various data structures may be found in Section 3.0.

The only operations valid on the configuration device are the following **ioctl**(2) calls:

HSDGETMOD
HSDSETMOD
HSDGETXSTS

2.10.1 HSDGETMOD [Get PCIHSD Operating Mode]

Argument: pointer to a *pcihsd_mode_t* structure

This call returns the current operating mode information to the referenced *pcihsd_mode_t* structure. This call may be performed on a device, provided no I/O activity is in progress.

2.10.2 HSDSETMOD [Set PCIHSD Operating Mode]

Argument: pointer to *pcihsd_mode_t* structure

This call sets the current operating mode of the addressed PCIHSD from information in the referenced *pcihsd_mode_t* structure. This call may be performed on a device, provided no I/O activity is in progress. To change modes, use HSDGETMOD to obtain the current settings, then issue HSDSETMOD after altering the desired items.

2.10.3 HSDRESET [Perform PCIHSD Reset Operations]

Argument: reset mode selection

This call performs the requested reset operation(s) on the addressed PCIHSD. The argument to this function is a reset mode selection which may be one of the following (defined in **pcihsdio.h**):

HSD_RS_HSD	Reset HSD controller
HSD_RS_FIFO	Reset FIFO's
HSD_RS_BOTH	Reset FIFO's and HSD controller
HSD_RS_DVC	Reset external device (IOReset)
HSD_RS_GEN	General reset (HSD/FIFO/External Device)
HSD_RS_TDV	Terminate Device (External Terminate)
HSD_RS_HALT	Do HALTIO (stop any active DMA) only
HSD_RS_PCI	Reset PCI bus interface chip
HSD_RS_LINK	Reset "IBL Link Request Received" flag (see Section 5.2 for more information)

Unless the argument is HSD_RS_HALT, there must be no currently active I/O operation on the device.

2.10.4 HSDGETSTS [Read PCIHSD Board Status]

Argument: pointer to a *pcihsd_sts_t* structure

This call stores into the addressed structure the current board status and HSD sequencer state words, as well as the values of those words immediately prior to the last reset operation (whether it resulted from an HSDRESET call or from internal driver operations).

2.10.5 HSDGETXSTS [Get Extended Driver Status]

Argument: optional pointer to an unsigned integer

This call returns information about the last operation performed on the addressed device. If the argument is non-zero, the extended status word is stored at that address. The extended status is returned as the value of the **ioctl** call in any case. The returned value contains both the system *errno* value and any PCIHSD-specific extended error code. The macros **HSDsyserr** and **HSDexterr**, defined in **pcihsdio.h** may be used to extract the respective error codes. Extended error codes are defined in **pcihsdio.h** with names of the form EXzzzz. The return from this call may also be -1, with an *errno* value of EFAULT if the argument contains an invalid address.

SECTION 3.0
DATA STRUCTURES

3.1 Introduction

This section describes the data structures used with the **ioctl**(2) function calls to perform I/O operations with the PCIHSD. These data structures and the associated constants are defined in the file **pcihsdio.h**.

3.2 Reference Documents

Refer to the respective Encore MPX-32 Operating System Reference and Technical software manuals for a detailed description of the software requirements for the IOCB structure and the HSDII board operation. For a comprehensive hardware description of the Encore HSDII compatible card refer to:

Encore HSDII Technical Manual, Document # 303-329131-000
Encore MPX-32 Volume 2, Reference Manual, Document #323-001011-300

Refer to the following documents for assistance programming the PCIHSD card:

ADS PCIHSD Technical Manual, Document # 0900098
ADS PCIHSD2 Technical Manual, Document # 0900117

3.3 pcihsd_mode_t Structure

The *pcihsd_mode_t* structure is used with the HSDGETMODE and HSDSETMODE calls to retrieve or change the PCIHSD's operating mode. This structure contains the following items:

opmode — Board operating mode code containing one of the following values defined in **pcihsdio.h**:

PCIHM_HSD	operating as HSD
PCIHM_IBL	operating as IBL

conn — Board connector configuration code containing one of the following values defined in **pcihsdio.h**:

PCIHM_CNRM	Normal (HSD) connectors
PCIHM_CREV	Reverse (IBL) connectors

Note: on an IBL link, certain signals must be transposed from one HSD device to the other. This is accomplished by either 1) using a cable with the appropriate wires interchanged and normal connectors on both HSD's or 2) using a straight-through cable and setting one of the HSD devices in a Reverse connector configuration.

link — IBL link request mode containing a code specifying the desired IBL link protocol and a single-bit field specifying the hardware link priority desired.

PCIHM_LREQ	Always request IBL link when initiating a transfer.
PCIHM_LACK	Never request IBL link when initiating transfer.
PCIHM_LIBLG	Use link protocol of the Encore H.IBLG handler for MPX. (Request link if writing, don't request if reading.)
PCIHM_HIPRI	Set high link priority. This bit may be or'd with any of the preceding values.

flags — Flag bits specifying options:

PCIHM_HZTO	Interpret timeout values as clock ticks instead of seconds.
PCIHM_NOUDD	Do not use file position as UDD byte on read/write calls.
PCIHM_CONTIBL	("Continuous IBL Mode", used with PCIHM_LACK) Wait for link request only on first read/write call.
PCIHM_NOLINK	(Used with PCIHM_LACK) Do not wait for IBL link request before starting data transfer on read/write calls.

timeout — Timeout for data transfer operations. The value is normally interpreted as a number of seconds. If PCIHM_HZTO flag bit is set in *flags*, then *timeout* is interpreted as a number of clock ticks (HZ per second). If this field contains zero, timeout operations are suppressed.

3.4 pcihsd_sts_t Structure

The *pcihsd_sts_t* structure is used with the HSDGETSTS **ioctl** call to return PCIHSD status. This structure contains the following information (all are 32-bit words):

bdstat_pre	PCIHSD status before last reset operation.
hstate_pre	HSD emulation sequencer state before last reset operation.
bdstat	Current PCIHSD status.
hstate	Current HSD emulation sequencer state word.

SECTION 4.0
STATUS RETURNS

4.1 Status Returns

Status returns from the system calls used to access the PCIHSD driver are consistent with other UNIX device drivers. In addition, extended status information is available by making the **ioctl**(HSDGETXSTS) call.

In case of an error, the system function called will return -1 and the variable *errno* will be set to an error code. Some status returns are specific to the PCIHSD driver and may differ from the meaning associated with the same return from another device.

EPERM Caller is attempting a diagnostic operation with insufficient privilege.

EBUSY Returned by OPEN if the addressed device is already in use. Returned by **read**(2), **write**(2), or **ioctl**(2) if an I/O operation is already in progress (perhaps initiated by another process thread).

4.2 Extended Status Returns

The value returned by the HSDGETXSTS call contains two fields, a system error code (*errno* value) and an extended error code, which can be extracted by the **HSDsyserr** and **HSDexterr** macros, respectively. If the system error code is EINVAL or EIO, the extended codes give the following additional information:

Expansion of EINVAL code:

EXCMD Buffer containing device commands is too long for one write operation or length is not a multiple of 8.

EXCNT Transfer count on read or write is 0 or is not a multiple of 4.

EXMAPERR Driver was unable to map user's buffer for DMA (should not occur).

EXLONG Data transfer request is too long (more than 65535 32-bit words).

EXOOPS Internal driver error building DMA descriptors for PCIHSD. Should not happen.

EXSTS Buffer for a device status request is too short (must be at least 4 bytes).

EXMODE The *opmode* field of the *pcihsd_mode_t* structure passed to an ioctl (SETMODE) call is invalid.

EXCONN — The *conn* field of the *pcihsd_mode_t* structure passed to an ioctl (SETMODE) call is invalid.

EXLINK — The *link* field of the *pcihsd_mode_t* structure passed to an ioctl (SETMODE) call is invalid.

Expansion of EIO code:

EXTIMO — Request not completed before timeout occurred.

EXTDV — I/O was terminated by External Terminate signal.

EXKILL — I/O was terminated by an ioctl(RESET) call with argument HSD_RS_HALT.

The following code may be returned with a system error code of 0 (all OK) if the HSDGETXSTS call immediately follows the **open**(2) on the device.

EXNODEV — No external device is connected.

4.3 PCIHSD Board Status Returns

The *bdstat* and *bdstat_pre* fields of the *pcihsd_sts_t* structure contain the status of the addressed PCIHSD currently and immediately prior to the last reset operation, respectively. The *hstate* and *hstate_pre* fields contain the associated HSD emulation sequencer state information. Sections 3.5.1 and 3.5.2 of the ADS PCIHSD / PCIHSD2 Technical Manuals, Document # 0900098 / 0900117, describe the sequencer state and board status words, respectively.

SECTION 5.0
PROGRAM INSTALLATION

Note: The PCIHSD driver operates only in Linux kernel versions 2.2.x and 2.4.x.

5.1 Distribution Media

The PCIHSD device driver is distributed on a 3½ inch floppy in DOS format:

Device driver code: Part Number 0950113

5.2 Installation

Log onto Linux as **root**.

Create a subdirectory for the PCIHSD driver (if one is not already available). Any suitable directory can be used.

mkdir pcihsd

Change to the pcihsd directory.

cd pcihsd

Copy the installation files from the distribution floppy to the pcihsd directory:

mcopy –tm a:* .

The following files should be placed in the pcihsd directory for compiling the driver:

pcihsdrv.h	PCIHSD device table structure and other internal driver definitions.
pcihsdev.h	PCIHSD hardware definitions.
pcihsd.c	PCIHSD driver source.

pcihsdio.h	Structure and symbol definitions for users of PCIHSD driver. This file should also be placed in a suitable location for access by programs compiled to use the PCIHSD driver.
Makefile	Instructions for building driver.
ldpcihsd	Script file to dynamically install the pcihsd driver as a module in the Linux kernel and create nodes in **/dev** for accessing the PCIHSD devices.
u	Script file to unload the PCIHSD driver from the Linux kernel.
hsdexample.c	Example program using the PCIHSD driver in HSD mode.
iblexample.c	Example program using the PCIHSD driver in IBL mode.

Since the PCIHSD driver files are distributed in a DOS compatible form, it may be necessary to set the files' proper permissions. All distributed files should have read/write capabilities for user, group, and other. Use the following command to set these permissions.

chmod a+rw *

Additionally, the two script files must have execution permission. Set this permission using the following command.

chmod a+x ldpcihsd u

5.2.1 Completing the Installation

Compile the PCIHSD driver for the current Linux version by typing:

make

The module file, **pcihsd.o**, is created from the pcihsd.c, pcihsdev.h, pcihsdrv.h, and pcihsdio.h source files and is suitable for dynamically loading into the Linux kernel.

The PCIHSD driver is now available for loading into the Linux kernel. The script file, **ldpcihsd**, can be used to accomplish this task. The script file takes one parameter that specifies the number of PCIHSD boards in the system. The script file will create nodes in the **/dev** directory to access the boards. The PCIHSD driver lets Linux assign the major device number. The script file will

determine the major number from the system and create the nodes with this major number. To load the driver and create the device nodes, type the following command:

./ldpcihsd n

where **‘n’** is the number of boards in the system.

5.2.2 Installation Details

The following symbols control the compilation of options within PCIHSD:

DEBUG — If this symbol is defined as **n**, debug tracing is enabled in the driver with the trace level initially set to **n**. A trace level of 0 suppresses trace output; higher levels increase the amount of trace output.

REV_LT_D — If this symbol is defined, code will be compiled to handle a status read problem with boards prior to Revision D. This symbol should be defined unless it is known that no pre-revision D boards will ever be used with this installation of the driver.

HOLD_LREQ — If this symbol is defined, the driver will not clear its "link request received" flag at the time of an open(2) call. This should be enabled in order to communicate via IBL link with a system that may come up and send a link request before the application is able to open the PCIHSD. The "link request received" flag can be cleared at any time by the HSD_RS_LINK reset option (see Section 2.10.3). If the PCIHM_LACK mode flag is set and the PCIHM_NOLINK option flag is not set, a link request must be received after this call completes before another data transfer can take place.

SHARED_IRQ — If this symbol is defined, the driver will request the interrupt level with the SA_SHIRQ flag, allowing the interrupt level to be shared with another device.

DYNAMIC_IRQ — If this symbol is defined, the driver will allocate the board’s interrupt level and attach the service routine at Open() time. Otherwise the interrupt handler will be installed when the driver is loaded. Installing the handler only when the device is opened may facilitate sharing the interrupt level between devices which are not used concurrently. Installing the handler when the driver loads makes the PCIHSD appear in the output of **lsdev**, and makes it possible to recognize link requests arriving before the device is opened.

5.2.3 Command Line Parameters

The following command line parameters are recognized by the PCIHSD module and may be included on the **insmod** command line:

tracelvl — This parameter allows setting the debug trace level (see DEBUG symbol, above) for the driver at the time it is loaded. Typically, the DEBUG symbol is defined with a value of 0, which compiles the debug trace code, but inhibits all trace output. Once the driver is loaded, **sethsd** can be used to set the trace level. This command line parameter sets the trace level before the driver has loaded and enables debugging of driver initialization problems without re-compiling the driver with a new DEBUG value.

allow_signals — By default, the PCIHSD driver will not interrupt its sleep on a pending I/O operation if the task requesting the operation receives a signal. In some cases it is desirable that the process doing I/O be interruptible by signals. Setting this parameter to any non-zero value enables signals to interrupt the process during the I/O wait. In this case the status returned from the I/O operation is –1, with *errno* set to EINTR – interrupted system call.

APPENDIX A
EXAMPLE PROGRAMS

The following is a tested IBL example and is provided as an aid for software development purposes only. This example is for a PCIHSD board connected to a Encore HSDII board or other compatible interface (i.e., an IBL mode configuration). The PCIHSD program will write data and read data each cycle. The PCIHSD initiates all transfers.

```
/*********************************************************************
 *     file:          iblexample.c                                  *
 *     part number:   -none-                                        *
 *     contents:      Linux PCIHSD Example Program                  *
 *     created:       1-Jul-2000                                    *
 *     by:            T. F. Crandell                                *
 *                                                                  *
 *     Copyright (c) 2000 by Applied Data Sciences, Inc.                  *
 *     All Rights Reserved                                          *
 *------------------------------------------------------------------*
 *     This program is a simple example of IBL data transfers using       *
 *     the PCIHSD and Linux 2.2.x                      *
 *                                                                  *
 *     This program has no options and takes one command line             *
 *     argument: the path to the pcihsdx device.                    *
 *                                                                  *
 *     Compile it:    cc -o iblexample iblexample.c                 *
 *     Execute it:    iblexample /dev/pcihsd0                       *
 *                                                                  *
 *            or      iblexample $(find /hw -name ibl -print)             *
 *------------------------------------------------------------------*
 *     Function:                                                    *
 *     1)  Open the PCIHSD and check for the other end connected.         *
 *     2)  If no other device, switch the connectors and check again.     *
 *     3)  If still no device, quit                                 *
 *     4)  Set the link protocol to "Always Request" and the timeout      *
 *         to 2 seconds.                                            *
 *     5)  Reset the PCIHSD                                         *
 *     6)  Repeat forever:                                          *
 *            6.1)    Write a block of data (WLEN words)            *
 *            6.2)    Read a block of data (RLEN words)                   *
 *------------------------------------------------------------------*
 *     Test Environment:                                            *
 *       Personal Computer, Linux 2.2.x Operating System      *
 *       Applied Data Sciences PCIHSD, Rev. B                             *
 *                                                                  *
 *     Other end of link:                                           *
 *       486/33 EISA bus PC, MS-DOS 5.0                                   *
 *       Applied Data Sciences PCEHSD, Rev. G                             *
 *       A program to read a block of data and echo it back.              *
 *                                                                  *
 *********************************************************************/

#include       <stdio.h>
#include       <stdlib.h>
#include       <errno.h>
#include       <fcntl.h>
#include       <signal.h>
#include       <sys/types.h>
#include       "pcihsdio.h"


#define RLEN           1000          /* Number of words to read    */
#define WLEN           5000          /* Number of words to write   */
```

```
long    wbfr[WLEN],                     /* Buffers...                   */
        rbfr[RLEN];

static  int     exitflag=0;             /* so we can exit politely      */

static  void    ioerror (int fd, char *what, int exp, int got);

/*----------------------------------------------------------------------*/
static void     catcher(int number)
{
    exitflag = 1;
}
/*-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=*/
int     main (int argc, char **argv)
{
    long                pass;
    int                 hsdfd, opnsts;
    int                 rsts, wsts;
    pcihsd_sts_t        stsblk;
    pcihsd_mode_t       cmode;

    if ( argc < 2 )  {
        fputs("usage: ./iblexample  /dev/pcihsd(x)\n", stderr);
        exit(1);
    }
    signal (SIGINT, catcher);

/*--------------------------- ((( STEP 1 ))) ---------------------------*/
    hsdfd = open (argv[1], O_RDWR);
    if ( hsdfd < 0 ) {
        perror ("Error on OPEN");
        exit(2);
    }
    ioctl (hsdfd, HSDGETMODE, &cmode);
    cmode.opmode = PCIHM_IBL;
    cmode.conn = PCIHM_CNRM;
    cmode.link = PCIHM_LREQ;
    ioctl( hsdfd, HSDSETMODE, &cmode);
    opnsts = ioctl(hsdfd, HSDGETXSTS, NULL);

/*--------------------------- ((( STEP 2 ))) ---------------------------*/
    if ( HSDexterr(opnsts) == EXNODEV ) {

        if ( cmode.conn == PCIHM_CNRM )  {
            puts ("No external device with normal connectors, trying reverse.");
            cmode.conn = PCIHM_CREV;
        } else {
            puts ("No external device with reverse connectors, trying normal.");
            cmode.conn = PCIHM_CNRM;
        }
        ioctl (hsdfd, HSDSETMODE, &cmode);
        ioctl (hsdfd, HSDGETSTS, &stsblk);

/*--------------------------- ((( STEP 3 ))) ---------------------------*/
        if ( (stsblk.bdstat & HST_DVC) == 0 ) {
            puts ("Still no external device.  Abandoning");
            close (hsdfd);
            exit(3);
        }
    }

/*--------------------------- ((( STEP 4 ))) ---------------------------*/
    cmode.link = PCIHM_LREQ;
    cmode.timeout = 2;
    ioctl (hsdfd, HSDSETMODE, &cmode);

/*--------------------------- ((( STEP 5 ))) ---------------------------*/
    ioctl (hsdfd, HSDRESET, HSD_RS_PCI);

/*--------------------------- ((( STEP 6 ))) ---------------------------*/
    for (pass=1; !exitflag ; ++pass) {
```

```
        printf ("\rBegin pass %d", pass);
        fflush (stdout);

        wsts = write (hsdfd, wbfr, sizeof(wbfr));
        if ( wsts != sizeof(wbfr) )
            ioerror(hsdfd, "write", sizeof(wbfr), wsts);
        else {
            printf ("   %6d words written", wsts/sizeof(hsdwd_t));
            fflush (stdout);
        }

        rsts = read  (hsdfd, rbfr, sizeof(rbfr));
        if ( rsts != sizeof(rbfr) )
            ioerror(hsdfd, "read", sizeof(rbfr), rsts);
        else {
            printf ("   %6d words read", rsts/sizeof(hsdwd_t));
            fflush (stdout);
        }
    }
    puts ("\nexiting on break.");

/*-----------------------------------------------------------------------*/
    close (hsdfd);
    return 0;
}
/*-----------------------------------------------------------------------*
 *  ioerror - Print Error Message on Data Transfers                *
 *                                                                 *
 *  synopsis:                                                      *
 *     ioerror (int fd, char *what, int expected, int actual)              *
 *     int     fd             File descriptor open to PCIHSD               *
 *     char    *what          Ptr to character string describing   *
 *                            failed operation                             *
 *     int     expected       Expected return value                        *
 *     int     actual         Actual return value                  *
 *                                                                 *
 *  return:    none                                                *
 *                                                                 *
 *  notes:     This function amplifies _some_ of the (most likely)  *
 *             errors in I/O operations by checking the driver's   *
 *             extended status.                                            *
 *-----------------------------------------------------------------------*/
static void ioerror (int fd, char *what, int exp, int got)
{
    int        sts;
    char       *xmsg;

    fprintf (stderr, "Error on %s: expected %d, got  %d", what, exp, got);

    if ( got < 0  && errno == EIO ) {
        sts = ioctl(fd, HSDGETXSTS, NULL);

        switch ( HSDexterr(sts) ) {
        case EXTIMO:
            xmsg =  "operation timed out"; break;
        case EXTDV:
            xmsg =  "stopped by external terminate"; break;
        default:
            xmsg = "???"; break;
        }
        fprintf (stderr, ": I/O error (%s)\n", xmsg);

    } else {
        perror(" ");
    }
}
/*-----------------------------------------------------------------------*/
```

The following is an example of HSD operation and is provided as an aid for software development purposes only. The program demonstrates a variety of different operations, and is not related to any specific external device.

```
/**********************************************************************
 *     file:         hsdexample.c                            *
 *     part number:  -none-                                  *
 *     contents:     Linux PCIHSD Example Program                   *
 *     created:      1-Jul-2000                              *
 *     by:           T. F. Crandell                                 *
 *                                                           *
 *     Copyright (c) 2000 by Applied Data Sciences, Inc.                   *
 *     All Rights Reserved                                   *
 *--------------------------------------------------------------------*
 *     This program is a simple example of HSD data transfers using   *
 *     the PCIHSD and Linux 2.2.x                                     *
 *                                                           *
 *     This program has no options and takes one command line         *
 *     argument: the path to the pcihsdx device.             *
 *                                                           *
 *     Compile it:   cc -o hsdexample hsdexample.c                    *
 *     Execute it:   hsdexample /dev/pcihsd0                 *
 *                                                           *
 *            or     hsdexample $(find /hw -name hsd -print)          *
 *--------------------------------------------------------------------*
 *     Function:                                             *
 *     1)  Open the PCIHSD and check for the external device          *
 *     2)  If no device, quit                                         *
 *     3)  Set the timeout to 2 seconds.                              *
 *     4)  Reset the PCIHSD                                  *
 *     5)  Repeat forever the equivalent of the IOCB list:            *
 *         5.1)  Send 2 device commands          42110000  00020003   *
 *                                     420100FF  00000000   *
 *         5.2)  Write WLEN words of data 0A00WLEN  0bfradrs   *
 *         5.3)  Send 1 device command           42020000  00020003   *
 *         5.4)  Read RLEN words of data         8A01RLEN  0bfradrs   *
 *         5.5)  Read the device status          20040000  00000000   *
 *                                                           *
 *     Notes:                                                *
 *     1)  pread() and pwrite() are used in HSD mode, so the "file    *
 *         position" can be controlled.                               *
 *     2)  The device command IOCB's (first 2 words only) are written *
 *         to "position" 0x40000000 which causes them to be sent as   *
 *         device commands. (W_CMD macro generates the "position").   *
 *     3)  The UDDBYTE macro serves the same purpose as W_CMD for data       *
 *         transfers, allowing the User Device Dependent byte (0 for  *
 *         the write, 1 for the read) to be specified.                *
 *     4)  The deivce status request is performed by reading 1 HSD    *
 *         word (32 bits) from "file position" 0x20000000.  The R_DSR *
 *         macro generates this position including the rest of the    *
 *         the first IOCB word for the deivce.                        *
 *--------------------------------------------------------------------*
 *     Test Environment:                                     *
 *       Personal Computer, Linux 2.2.x Operating System                     *
 *       Applied Data Sciences PCIHSD, Rev. B                         *
 *                                                           *
 *     Other end of link:                                    *
 *       Applied Data Sciences HSD Analyzer/Tester                    *
 *                                                           *
 **********************************************************************/

#include      <stdio.h>
#include      <stdlib.h>
#include      <errno.h>
#include      <fcntl.h>
#include      <signal.h>
#include      <sys/types.h>
```

```
#include      "pcihsdio.h"


#define       RLEN          1000          /* Number of words to read  */
#define       WLEN          5000          /* Number of words to write */

long   wbfr[WLEN],                 /* Buffers...               */
       rbfr[RLEN],
       cmd1[4],                           /* 2 commands = 4 words             */
       cmd2[2];                           /* 1 command  = 2 words             */


static int    exitflag=0;          /* so we can exit politely  */

static void   ioerror (int fd, char *what, int exp, int got);

/*------------------------------------------------------------------*/
static void   catcher(int number)
{
    exitflag = 1;
}
/*-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-*/
int    main (int argc, char **argv)
{
    long             pass;
    int                     hsdfd, opnsts, printnow;
    int                     rsts, wsts, csts;
    pcihsd_sts_t     stsblk;
    pcihsd_mode_t    cmode;
    hsdwd_t          devsts;              /* external device status   */

    if ( argc < 2 )  {
       fputs("usage: hsdexample  /dev/pcihsd(x)\n", stderr);
       exit(1);
    }
    signal (SIGINT, catcher);

/*--------------------------- ((( STEP 1 ))) ---------------------------*/
    hsdfd = open (argv[1], O_RDWR);
    if ( hsdfd < 0 ) {
       perror ("Error on OPEN");
       exit(2);
    }
    opnsts = ioctl(hsdfd, HSDGETXSTS, NULL);

/*--------------------------- ((( STEP 2 ))) ---------------------------*/
    if ( HSDexterr(opnsts) == EXNODEV ) {
       puts ("No external device.  Abandoning");
       close (hsdfd);
       exit(3);
    }

/*--------------------------- ((( STEP 3 ))) ---------------------------*/
    ioctl (hsdfd, HSDGETMODE, &cmode);
    cmode.timeout = 2;
    ioctl (hsdfd, HSDSETMODE, &cmode);

/*--------------------------- ((( STEP 4 ))) ---------------------------*/
    ioctl (hsdfd, HSDRESET, HSD_RS_PCI);

/*-------------------- Set up Device Comand Buffers --------------------*/
    cmd1[0] = 0x40110000;          /* 1st command, IOCB word 0 */
    cmd1[1] = 0x00020003;          /* 1st command, IOCB word 1 */
    cmd1[2] = 0x400100FF;          /* 2nd command, IOCB word 0 */
    cmd1[3] = 0x00000000;          /* 2nd command, IOCB word 1 */

    cmd2[0] = 0x40020000;          /* 3rd command, IOCB word 0 */
    cmd2[1] = 0x00020003;          /* 3rd command, IOCB word 1 */

/*--------------------------- ((( STEP 5 ))) ---------------------------*/
```

```
    for (pass=1; !exitflag ; ++pass) {
       printnow = ((pass % 100) == 0);
       if ( printnow ) {
           printf ("\rBegin pass %d", pass);
           fflush (stdout);
       }

/*-------------------------- ((( STEP 5.1 ))) --------------------------*/
       csts = pwrite (hsdfd, cmd1, sizeof(cmd1), W_CMD);
       if ( csts != sizeof(cmd1) )
           ioerror(hsdfd, "device cmd #1", sizeof(cmd1), csts);

/*-------------------------- ((( STEP 5.2 ))) --------------------------*/
       wsts = pwrite (hsdfd, wbfr, sizeof(wbfr), UDDBYTE(0));
       if ( wsts != sizeof(wbfr) )
           ioerror(hsdfd, "write", sizeof(wbfr), wsts);
       else if (printnow) {
           printf ("   %6d words written", wsts/sizeof(hsdwd_t));
           fflush (stdout);
       }

/*-------------------------- ((( STEP 5.3 ))) --------------------------*/
       csts = pwrite (hsdfd, cmd2, sizeof(cmd2), W_CMD);
       if ( csts != sizeof(cmd2) )
           ioerror(hsdfd, "device cmd #2", sizeof(cmd2), csts);

/*-------------------------- ((( STEP 5.4 ))) --------------------------*/
       rsts = pread  (hsdfd, rbfr, sizeof(rbfr), UDDBYTE(1));
       if ( rsts != sizeof(rbfr) )
           ioerror(hsdfd, "read", sizeof(rbfr), rsts);
       else if (printnow) {
           printf ("   %6d words read", rsts/sizeof(hsdwd_t));
           fflush (stdout);
       }

/*-------------------------- ((( STEP 5.4 ))) --------------------------*/
       csts = pread  (hsdfd, &devsts, sizeof(devsts), R_DSR(0x00040000));
       if ( csts != sizeof(devsts) )
           ioerror(hsdfd, "device sts rqst", sizeof(devsts), csts);
       else if (printnow) {
           printf ("  dvc stat %08x", devsts);
           fflush (stdout);
       }
    }
    puts ("\nexiting on break.");

/*----------------------------------------------------------------------*/
    close (hsdfd);
    return 0;
}
/*---------------------------------------------------------------------*
 *  ioerror - Print Error Message on Data Transfers                   *
 *                                                             *
 *  synopsis:                                                  *
 *     ioerror (int fd, char *what, int expected, int actual)         *
 *     int    fd            File descriptor open to PCIHSD            *
 *     char   *what         Ptr to character string describing *
 *                          failed operation                          *
 *     int    expected      Expected return value                            *
 *     int    actual        Actual return value                *
 *                                                             *
 *  return:   none                                             *
 *                                                             *
 *  notes:    This function amplifies _some_ of the (most likely)     *
 *            errors in I/O operations by checking the driver's       *
 *            extended status.                                        *
 *----------------------------------------------------------------------*/
static void ioerror (int fd, char *what, int exp, int got)
{
    int              sts;
    char      *xmsg;
```

```
    fprintf (stderr, "Error on %s: expected %d, got  %d", what, exp, got);

    if ( got < 0  && errno == EIO ) {
       sts = ioctl(fd, HSDGETXSTS, NULL);

       switch ( HSDexterr(sts) ) {
       case EXTIMO:
           xmsg =  "operation timed out"; break;
       case EXTDV:
           xmsg =  "stopped by external terminate"; break;
       default:
           xmsg = "???"; break;
       }
       fprintf (stderr, ": I/O error (%s)\n", xmsg);

    } else {
       perror(" ");
    }
}
/*-----------------------------------------------------------------------*/
```

APPENDIX B
SETHSD UTILITY

The **sethsd** program is a utility which can be used to examine or alter a PCIHSD's configuration. It is invoked with command line arguments specifying the configuration to be set and the device files to be altered. If no changes are given, the current configuration for each addressed device will be displayed. The command line must list the desired options followed by the names of the devices to be affected. **Sethsd** returns zero if successful, non-zero if any requested operation fails.

B.1 Available Options

-H	Set device to HSD configuration.
-In	Set device to IBL configuration with normal (HSD) connector configuration
-Ir	Set device to IBL configuration with reverse (IBL) connector configuration
-p	(appended to -In or -Ir) Set high priority for IBL link arbitration (-Ir or -In resets to low priority)
-R	Perform power-up reset
-S	Read and display device status

B.2 EXAMPLES:

sethsd -In /dev/pcihsd0

Set PCIHSD unit 0 to IBL mode, normal (HSD) connectors, low priority.

sethsd -Ir /dev/pcihsd0

Same as above, but set connectors to reverse (IBL) configuration.

sethsd -S /dev/pcihsd1

Open PCIHSD unit 1; read and display its status.