

Applied Data Sciences / PCIHSD® Software Specifications

As mentioned in Section 1.3, there are two groups of I/O subroutines, namely the low-level routines described in detail in Section 2.1 and the read-write routines which are described in Section 2.2. Section 2.3 describes auxiliary routines which do not perform any data transfers but may be useful. Up to four PCIHSD's may be controlled by these subroutines, provided the user maintains a unique File Control Block (FCB) for each device. The user must not alter those portions of the FCB used by the I/O routines while the device is open. As of version 1.0, usage is limited to the first PCIHSD recognized by the BIOS.

NOTE: These routines use the "C" language calling conventions. If calling these routines from FORTRAN or other high level language, the appropriate high level language facility (e.g. the INTERFACE statement in FORTRAN) should be used.

2.1 Low-Level Routines

The low-level routines give the user the greatest control over PCIHSD operations, but are more difficult to use than the read-write (block transfer) routines. In particular, the Encore IOCB structure is not used to control data transfers, but rather the DMA Descriptor Block (DDB) native to the PLX PCI9080 bus interface is used. However, for complete emulation of the Encore HSDII, the appropriate words which would be found in the Encore IOCB must be sent to the PCIHSD's emulation logic. For example, in writing data to an external device, the first word of the IOCB (containing User-device-dependent bits and the word count) must be written to the PCIHSD before the data is sent. This may be done by the user placing the appropriate IOCB word ahead of the data buffer and writing the entire block with a single transfer, or by using a data chained list of two DDB's. The first DDB would transfer four bytes consisting of the first IOCB word; the second DDB would transfer the actual data. A thorough understanding of the Encore HSDII is required to use the low-level routines.

2.1.1 PCIOPEN Routine

The PCIOPEN routine must be called first to install the driver in the MS-DOS interrupt structure and initialize some necessary data structures. The

user passes a File Control Block (FCB) address to the subroutine. This data structure is described in Section 3.0. The user must have filled in set-up information in the FCB. The calling sequence is:

```
sts = PCIOPEN(pfcb, mode)
```

pfcb is a far (segment:offset) pointer to a File Control Block structure which contains status and set-up information.

mode is an optional integer operating mode selection. If non-zero, PCIMODE will be called at the completion of open processing, using this argument. See Section 2.4 for a description of this value.

sts is an integer return status code which may be zero to indicate success or may be one of the following negative values (symbols are defined in PCIHSD.H):

E_NODEV No PCIHSD found in system configuration

E_BIOS Not a PCI bus system (no BIOS)

E_MANY Four PCIHSD devices open already

E_OPEN Specified PCIHSD already open

E_BADDEV Bad initial board status from PCIHSD (FIFOs are not empty, transfer counter is non-zero or End of Block or External Terminate status is asserted.)

E_NOXDEV External device not connected

The following two error codes may be returned (from PCIMODE) if the mode argument is non-zero.

E_XFER Bad data transfer mode

E_OPMODE Bad operational mode

The following flags in the FCB flags field are significant on the PCIOPEN call:

FCB_NORESET Do not reset PCIHSD or external device

FCB_NOIOR Reset PCIHSD, do not send IOReset to the external device

FCB_IGNERR Ignore a bad initial PCIHSD status

2.1.2 PCICLOS Routine

The PCICLOS routine is used to restore the MS-DOS interrupt structure to its normal configuration. The calling sequence is:

```
sts = PCICLOS(pfcb)
```

pfcb is a far (segment:offset) pointer to a File Control Block structure which contains status and set-up information.

sts is an integer return status code which may be zero to indicate success or may be one of the following negative values (symbols are defined in PCIHSD.H):

E_NOTOP Addressed FCB is not open

2.1.3 PCIMODE Routine

The PCIMODE routine is used to set the PCIHSD's operating mode. This call must not be made when any I/O transfer is in progress. The calling sequence is:

```
sts = PCIMODE(pfcb, mode)
```

pfcb is a far (segment:offset) pointer to a File Control Block.

mode is an integer encoding the desired mode selections. This value is formed by or-ing together the desired options. All option selections are non-zero; any field in this value containing zero is ignored, resulting in no mode change.

A valid mode selection is formed from one of:

PCIM_HSD HSD operating mode

PCIM_IBL IBL operating mode

Plus one of the following

PCIM_CNRM Normal connectors

PCIM_CREV Reversed connectors

PCIM_CLB1 Loopback configuration #1

PCIM_CLB2 Loopback configuration #2

Plus any of the following

PCIM_BSWAP Byte swap data (convert big- to little-endian)

PCIM_NSWAP Do not swap data (no endian conversion)

PCIM_HIPRI High IBL link priority

PCIM_LOPRI Low IBL link priority

PCIM_LREQ Always request IBL link

PCIM_LACK Always acknowledge IBL link

PCIM_LIBLG Use Encore H.IBLG link protocol (Request link if operation is a write; acknowledge if operation is a read.)

sts is an integer return status code which may be one of the following negative values (symbols are defined in PCIHSD.H):

E_NOTOP Addressed FCB is not open

E_BUSY Addressed device is busy

E_OPMODE Specified new operational mode is invalid

If the mode set operation was successful, the previous mode setting (always a non-negative number) is returned.

2.1.4 PCISTRTR Routine

The PCISTRTR routine is used to start an HSD transfer. The address of an FCB containing the address of a DMA Descriptor Block list is passed to this subroutine. In the NOWAIT mode, once the transfer is started, control is passed back to the calling routine. In the WAIT mode, the PCISTRTR routine waits for list execution to finish. In the NOWAIT mode the calling routine may determine when the transfer is complete by monitoring the device busy bit in the FCB. A preferred method of doing this is to call PCITEST to determine the status of the transfer. The calling sequence is.

```
sts = PCISTRTR(pfcbl)
```

pfcbl is a far (segment:offset) pointer to a File Control Block structure which contains a pointer to the DMA Descriptor Block (DDB) list.

sts is an integer return status code which may be zero to indicate success or may be one of the following negative values (symbols are defined in PCIHSD.H):

E_NOTOP Addressed FCB is not open

E_MODE Bad MODE setting for current operation. This generally means that PCIMODE has not been called.

E_XFER Bad data transfer mode

E_BUSY Addressed device is busy

E_CMD Command error

E_TOOLONG The DMA descriptor list is greater than 50 entries.

The following error codes will be returned only in WAIT mode.

E_KILLED I/O killed by PCITERM

E_TIMO Device time-out

2.1.5 PCITEST Routine

The PCITEST routine is used to determine whether an I/O transfer is still in progress. It monitors both the "busy" status bit in the FCB and the user-specified time-out. If the NOWAIT bit is set in the FCB, PCITEST will return immediately with a status. If the NOWAIT bit is not set, it will not return until the current transfer is complete. The calling sequence is:

```
sts = PCITEST(pfcf)
```

pfcf is a far (segment:offset) pointer to a File Control Block structure which contains status and set-up information.

sts is an integer return status code which may be zero to indicate successful completion of the transfer or may be one of the following negative values (symbols are defined in PCIHSD.H):

E_NOTOP Addressed FCB is not open

E_BUSY Transfer is still in progress

E_TIMO Device time-out before transfer complete

E_CMD Command error

E_IO I/O error on data xfer

E_KILLED I/O killed by PCITERM

2.1.6 PCISTAT Routine

The PCISTAT routine reads the current status from the PCIHSD and updates the bdstat field in the FCB. The calling sequence is:

```
sts = PCISTAT(pfcf)
```

pfcb is a far (segment:offset) pointer to a File Control Block structure which contains status and set-up information.

sts is an integer return status code which may be zero to indicate success or may be one of the following negative values (symbols are defined in PCIHSD.H):

E_NOTOP Addressed FCB is not open

2.1.7 PCITERM Routine

The PCITERM routine first terminates any existing transfer in progress and stores the PCIHSD status in the pre-reset status field of the FCB. It then resets the PCIHSD as indicated by the second argument in the call. The calling sequence is:

```
sts = PCITERM(pfcb, trm_cod)
```

pfcb is a far (segment:offset) pointer to a File Control Block structure which contains status and set-up information.

trm_cod is a code indicating the type of reset desired. There are seven types of Reset which have the values:

RS_GEN General reset (HSD/FIFO/External Device)

RS_HSD Reset PCIHSD's HSD emulation logic

RS_FIFO Reset FIFOs

RS_BOTH Reset FIFOs & HSD logic

RS_DVC Reset external device (Encore IOR signal)

RS_TDV Terminate Device (send External Terminate)

RS_PCI Reset PCI bus interface chip

0 No Reset command; terminate I/O transfer only

sts is an integer return status code which may be zero to indicate success or may be one of the following negative values (symbols are defined in PCIHSD.H):

E_NOTOP Addressed FCB is not open

2.1.8 PCIINFO Routine

The PCIINFO routine is used to retrieve internal information about the PCIHSD and the I/O routines. The calling sequence is

```
sts = PCIINFO(pfcfcb, itemcode, itembfr, bfrsize)
```

pfcfcb is a far (segment:offset) pointer to a File Control Block structure which contains status and set-up information.

itemcode is an integer code defining the information to be returned. Valid codes are (symbols are defined in PCIHSD.H):

PCII_DMAST Return current DMA engine status (5 longwords). This function is used by the PCIHSD test program only.

itembfr is a far pointer to a buffer to receive the returned information.

itemcode is an integer giving the size, in bytes, of the return buffer.

sts is an integer return status code which may be zero to indicate success or may be one of the following negative values (symbols are defined in PCIHSD.H):

E_NOTOP Addressed FCB is not open

E_INVALID Invalid parameter (unknown itemcode or buffer too short to contain requested item.)

2.2 READ-WRITE Routines

The Read-Write PCIHSD routines hide much of the detail of operating the PCIHSD from the user. Conversely, they limit the operations which may be

performed (e.g. all transfers are done in WAIT mode). They are called in much the same manner as the UNIX read and write functions, using an integer unit number to designate a specific PCIHSD. The user calls `PCIOPEN_RW` with a pointer to a conventional FCB; `PCIOPEN_RW` returns the assigned unit number. Calls to low-level routines addressing the FCB may be used with read-write calls using the corresponding unit number.

2.2.1 `PCIOPEN_RW` Routine

The `PCIOPEN_RW` routine initializes a PCIHSD for use with the read-write routines. `PCIOPEN_RW` first calls `PCIOPEN`, passing it the supplied FCB address and mode arguments, then initializes the data structures unique to the read-write functions. Data transfers using the `PCIREAD` and `PCIWRITE` calls make use of a temporary storage area to hold the DMA Descriptor Blocks (DDBs) for the PCIHSD. An area for two DDBs (sufficient for a single read or write) is allocated by default. The user may supply an external buffer for this use by putting a far pointer to the desired buffer and its length in the `ddbl` and `ddblen` fields of the FCB, respectively, before calling `PCIOPEN_RW`. If the `ddbl` field is `NULL`, the default buffer is used. If a DDB list buffer is supplied, it must be aligned on a 16-byte boundary. The calling sequence is:

```
fd = PCIOPEN_RW(pfcb, mode)
```

`pfcb` is a far (segment:offset) pointer to a File Control Block structure which contains status and set-up information.

`mode` is an optional integer operating mode selection. If non-zero, `PCIMODE` will be called at the completion of open processing, using this argument. See Section 2.1.3 for a description of this value.

`fd` is an integer which may be a non-negative unit number to indicate success or may be one of the following negative values (symbols are defined in `PCIHSD.H`):

`E_NODEV` No PCIHSD found in system configuration

`E_BIOS` Not a PCI bus system (no BIOS)

E_MANY Four PCIHSD devices open already

E_OPEN Specified PCIHSD already open

E_BADDEV Bad initial board status from PCIHSD (FIFOs are not empty, transfer counter is non-zero or End of Block or External Terminate status is asserted.)

E_NOXDEV External device not connected

The following two error codes may be returned (from PCIMODE) if the mode argument is non-zero.

E_XFER Bad data transfer mode

E_OPMODE Bad operational mode

2.2.2 PCICLOS_RW Routine

The PCICLOS_RW routine cleans up data structures specific to the read-write routines and calls PCICLOS. If the internal DDB list buffer was used, the ddbl field of the FCB is restored to its original (NULL) value. It is called as follows:

```
sts = PCICLOS_RW(fd)
```

fd is a unit number returned by PCIOPEN_RW.

sts is an integer return status code which may be zero to indicate success or may be one of the following negative values (symbols are defined in PCIHSD.H):

E_NOTOP Addressed FCB is not open

2.2.3 PCIREAD Routine

The PCIREAD routine transfers a block of data from the PCIHSD to the specified buffer. Transfers take place in WAIT mode; when PCIREAD

returns to the caller, the data transfer will have taken place, or an error status will be returned. It is called as follows:

```
l rtn = PCIREAD(fd, uddbyte, abfr, nwords)
```

fd is a unit number returned by PCIOPEN_RW.

uddbyte is an unsigned integer supplying the User-Device-Dependent field of the data transfer IOCB. Bits 0-7 of this value are inserted in the first IOCB word bits 16-23, which are passed on to the external device. This value is irrelevant in IBL mode.

abfr is a far pointer to the destination buffer.

nwords is a longword (32-bit) value containing the number of 32-bit words to transfer.

l rtn is a longword (32-bit) return status code which may be (if non-negative) the number of words successfully read or may be one of the following negative values (symbols are defined in PCIHSD.H):

E_NOTOP Specified unit is not open

E_BUSY Specified unit already has a transfer in progress.

E_MODE Bad MODE setting for current operation. This generally means that PCIMODE has not been called.

E_TOOLONG DDB list for this operation is too long to fit in supplied DDB list buffer.

E_ALIGN Supplied DDB list buffer is not 16-byte aligned. This error should not occur if using the internal DDB list buffer.

E_INVALID Transfer count (nwords) is greater than 65535 (FFFF16).

E_TIMO Timed out waiting for read to complete.

E_KILLED I/O killed by PCITERM

2.2.4 PCIWRITE Routine

The PCIWRITE routine transfers a block of data to the PCIHSD from the specified buffer. Transfers take place in WAIT mode; when PCIWRITE returns to the caller, the data transfer will have taken place, or an error status will be returned. It is called as follows:

```
l rtn = PCIWRITE(fd, uddbyte, abfr, nwords)
```

fd is a unit number returned by PCIOPEN_RW.

uddbyte is an unsigned integer supplying the User-Device-Dependent field of the data transfer IOCB. Bits 0-7 of this value are inserted in the first IOCB word bits 16-23, which are passed on to the external device. This value is irrelevant in IBL mode.

abfr is a far pointer to the source buffer.

nwords is a longword (32-bit) value containing the number of 32-bit words to transfer.

l rtn is a longword (32-bit) return status code which may be (if non-negative) the number of words successfully written or may be one of the following negative values (symbols are defined in PCIHSD.H):

E_NOTOP Specified unit is not open

E_BUSY Specified unit already has a transfer in progress.

E_MODE Bad MODE setting for current operation. This generally means that PCIMODE has not been called.

E_TOOLONG DDB list for this operation is too long to fit in supplied DDB list buffer.

E_ALIGN Supplied DDB list buffer is not 16-byte aligned. This error should not occur if using the internal DDB list buffer.

E_INVALID Transfer count (nwords) is greater than 65535 (FFFF16).

E_TIMO Timed out waiting for WRITE to complete.

E_KILLED I/O killed by PCITERM

2.2.5 PCICMD Routine

The PCICMD routine sends one or more commands to an external device. On the Encore HSDII, Device Command IOCB's are executed by sending the first two words of each IOCB to the external device. To use PCICMD, the caller places the first two words of each Device Command IOCB to be executed in a buffer. A far pointer to the buffer and the number of commands (word pairs) to send to the device are passed to PCICMD. It is called as follows:

```
sts = PCICMD(fd, clist, ncmds)
```

fd is a unit number returned by PCIOPEN_RW.

clist is a far pointer to the list of commands to be issued. The first 2 words of each Encore device command IOCB should be placed in this buffer.

ncmds is an integer giving the number of device commands to be sent. Twice this number of 32-bit words are transferred from the buffer clist.

sts is an integer return status code which may be the non-negative number of commands written successfully or may be one of the following negative values (symbols are defined in PCIHSD.H):

E_NOTOP Specified PCIHSD is not open

E_BUSY Specified device is busy with another transfer.

E_CMD Not all words in the clist buffer represent Device Command IOCB's

E_TIMO Exceeded the time-out specified in the FCB waiting for the transfer to complete.

2.2.6 PCIDSR Routine

The PCIDSR routine requests status from an external device. The Device Status Request IOCB functions much like a Device Command, except, only the first IOCB word is sent to the device and the 32-bit status word returned by the device is stored in the fourth word of the IOCB. The calling sequence is:

```
sts = PCIDSR(fd, cmdwd, astswd)
```

fd is a unit number returned by PCIOPEN_RW.

cmdwd is an unsigned (32-bit) longword containing the first word of the Device Status Request IOCB to be sent to the device.

astswd is a far pointer to a 4-byte area to receive the external device status word.

sts is an integer return status code which may be zero to indicate success or may be one of the following negative values (symbols are defined in PCIHSD.H):

E_NOTOP Specified PCIHSD is not open

E_BUSY Specified device is busy with another transfer.

E_TIMO Exceeded the time-out specified in the FCB waiting for the transfer to complete.

2.3 Auxiliary Routines

The following functions are included in the PCIDEV.LIB library primarily for internal use by the read-write routines, but they have general applicability where the functions of an Encore HSDII are being replaced by a PCIHSD. Three functions are provided for translating an Encore HSDII IOCB list (or IOCL) to a DMA Descriptor Block List (DDBL) for use by the PCIHSD.

2.3.1 iocl_xsetup Routine

The `iocl_xsetup` routine initializes the IOCB translation functions. It is called as follows:

```
iocl_xsetup (pgsize, vtop)
```

`pgsize` is a longword (32-bit) containing the memory system page size, in bytes, currently in use. For MS-DOS, this should be set to 1MB (10000016).

`vtop` is a far pointer to a function which translates "virtual" addresses to "physical" addresses. For MS-DOS, this consists simply of converting a segment:offset format address to a 20-bit flat address in a 32-bit word. The calling sequence to `vtop` is:

```
physadr = vtop(segadr)
```

`segadr` is a far (32-bit, segment:offset) pointer to be converted.

`physadr` is an unsigned longword (32-bit) containing the equivalent 20-bit flat address of `segadr`.

2.3.2 iocl_listsize Routine

The `iocl_listsize` function calculates the number of bytes required to contain the DDBL corresponding to the given IOCL. It takes into account the page size specified in the call to `iocl_xsetup` and allows for multiple DDB's to transfer non-contiguous data, even though this situation does not occur in the MS-DOS environment. It is called as follows:

```
nbytes = iocl_listsize (iocl)
```

`iocl` is a far pointer to the IOCB list to be converted.

`nbytes` is the integer number of bytes required to contain the DDBL corresponding to the IOCB list pointed to by `iocl`.

2.3.3 iocl_xlate Routine

The iocl_xlate routine translates an Encore HSDII IOCB list (IOCL) to a DDB list usable by the PCIHSD. The iocl_xlate function is called as follows:

```
sts = iocl_xlate (iocl, ddbl, size)
```

iocl is a far pointer to the IOCB list to be translated.

ddb1 is a far pointer to a buffer to receive the converted list.

size is the integer length of the destination buffer, in bytes.

sts is an integer return status code which may be zero to indicate success or may be one of the following negative values (symbols are defined in PCIHSD.H):

E_ALIGN DDB list buffer (ddb1) is not 16-byte aligned.

E_TOOLONG DDB list buffer is not long enough to hold the translated list.

E_NOTIC Original IOCB list contains a Transfer-in-Channel (TIC) IOCB. The TIC operation cannot be emulated in the PCIHSD, since the DMA descriptors will be fetched from a FIFO on the PCIHSD.

The translations performed by iocl_xlate are as follows:

Data Transfer IOCB's (Read/Write)

DMA write of four bytes from the original IOCB (the first word), chained to DMA read(s) or write(s), as required, of four times the word count bytes to the address from the second IOCB word. The buffer address in the second word of the original IOCB is converted to a physical address for the PCIHSD's DMA engine by calling the vtop function defined in the call to iocl_xsetup. If this function returns non-contiguous physical addresses, multiple DMA descriptors will be data chained to accomplish the transfer.

Command Transfer IOCB's

DMA write of 8 bytes from the original IOCB (send first 2 IOCB words).

Device Status Request IOCB's

DMA write of 4 bytes from the original IOCB (first IOCB word), chained to a 4 byte DMA read to fourth word of original IOCB.

Note that these translations require the user to leave the original IOCB list intact until the PCIHSD operations are complete. Also note that external device status will be stored in the fourth word of the Device Status Request IOCB, just as with the Encore HSDII.

3.1 Introduction

The PCIHSD card emulates the Encore HSDII card and thus all programming constraints are imposed by ensuring compatibility with the Encore HSDII card. This section provides information on the correct programming usage of data structures used with the PCIHSD driver, particularly the File Control Block (FCB), the Input/Output Control Block (IOCB) and the DMA Descriptor Block (DDB). The IOCB adheres to the Encore HSDII definitions. These data structures are defined in the header file PCIHSD.H.

3.2 Reference Documents

Refer to the respective Encore MPXÄ32 Operating System Reference and Technical software manuals for a detailed description of the software requirements for the FCB and 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 programming assistance of the IBM compatible PCIHSD card.

IBM PC/AT Technical Reference Manual 1502494

IBM DOS Technical Reference Manual 6138536

ADS PCIHSD Technical Manual 0900098

Encore MPX-32 Reference Manual, Volume 1 323-001011-300

3.3 File Control Block

The File Control Block (FCB) provides transfer parameters and working storage areas for the device driver routines. The FCB must be set up by the user to describe each PCIHSD used, and to describe certain attributes of each logical I/O operation. In addition, certain information collected during each I/O operation is made available to the user via the corresponding FCB.

The FCB format is described in detail in the following sections. The description uses the symbol names from the `pcifcb_s` structure definition, found in the header file `PCIHSD.H`. Additional fields defined in the data structure but not described here are reserved for future use.

3.3.1 User Supplied Information

The items listed in this section must be supplied by the caller to describe the hardware configuration in use and the desired options.

Field Contents

options Option flags. Any of the following values or-ed together:

FCB_NOWAIT Return control to caller after initiating data transfer operation. The caller is responsible for calling `PCITEST` or otherwise checking the status of the transfer as reported in the FCB.

FCB_IGNERR Ignore bad hardware status on open.

FCB_NORESET Do not reset the PCIHSD on open.

FCB_NOIOR Do not issue I/O Reset (IOR) to external device on open.

timeout Timeout interval to be used for data transfer operations, in units of 0.1 seconds. This is an unsigned 16-bit field; time intervals up to 6553.6 seconds may be specified.

ddb1 Far (segment:offset) pointer to DMA descriptor list to be executed by PCISTRRT. Also points to user-supplied DDB list buffer for use with read-write functions.

ddb1len Length of user-supplied DDB list buffer (used with read-write functions only).

devkey User-defined device key (currently unused).

3.3.2 Status Information

Various status information is available to the user. These fields should not be altered by the user when the FCB is open (i.e. between calls to PCIOPEN and PCICLOS). This information is presented in the following places:

Field Contents

fcbsts Current operation status. It contains the following flag bits:

FCB_OPEN This FCB is currently open for use.

FCB_BUSY Operation currently in progress.

FCB_ERR Last operation completed with an error condition.

boardid PCI bus interface chip ID and Revision number returned by successful call to PCIOPEN.

serial PCIHSD board serial number in BCD and revision level (bits 0-7).

cddb Address of current/last DDB executed by PCIHSD. This is the actual address from which the DMA engine fetched the descriptor block. Consequently, it will be a 32-bit address within the PCIHSD list FIFO. Subtracting the FIFO address (7000016) yields a byte offset to the current DDB within the list.

bdstat PCIHSD status word. See Section 3.4 for more information.

hstate HSD emulation logic state. This word is updated at the end of each transfer.

dtbl Far (segment:offset) pointer to device table used internally by PCIHSD driver. This field must not be modified.

3.4 DMA Descriptor Block Required Fill-in

Prior to a PCISTRTR call, the calling routine must first place a pointer to the DMA Descriptor Block list in FCB field ddbl.

The DDB list is an array of structures which the calling routine must fill in. The function is similar to that of the normal Encore IOCB, but the semantics are those of the PCIHSD bus interface chip. Typically, this translation is performed transparently by calling one of the read-write functions or explicitly by calling `iocl_xlate`. The DDB structure (`struct dmadsc_s`) is defined in header file `PCIHSD.H` and includes the following fields:

pciadr PCI host memory address for the transfer (a 32-bit address).

lcladr Local bus address on the PCIHSD for the transfer (a 32-bit address).

xfrsize A 32-bit byte count for the transfer

chain The 32-bit PCI memory address of the next DDB to execute, plus some control bits:

Bit Meaning

0 Next descriptor is in PCI address space

1 END of Chained DMA list

2 Interrupt after completing this DDB

3 Read: transfer PCIHSD to PCI memory

Refer to the PCIHSD Technical Manual, Document number 0900098 for a complete description of PCIHSD DMA setup. Definitions of other symbols required when setting up DDB's are in the header file PCIHSDDEV.H (PCIHSDDE.H on a DOS FAT filesystem).

To completely emulate the Encore HSDII, the appropriate parts of the usual IOCB must be sent to the PCIHSD emulation logic. This includes the first IOCB word for all transfers and the second IOCB word as well for Device Command Transfers. The following HSD functions, which are described by IOCB Word 1, opcode bits 00 - 07, are supported:

Bit 00 - HSD I/O Transfer.

Bit 01 - HSD Command Transfer.

Bit 02 - HSD Status Request.

Bit 04 - HSD Interrupt on End-of-Block (IEOB).

Bit 05 - Not used (Transfer-In-Channel)

Bit 06 - Not used (HSD Command Chain)

Bit 07 - Not used* (HSD Data Chain)

* Note that command and data chaining are handled in the DDB list in a manner analogous to the IOCB. The HSD Data Chain bit is required only if a single block of more than 65535 (FFFF16) words is to be transferred, in which case a second "IOCB word 1" must be sent to the PCIHSD to reload the transfer counter and continue the operation.

2.1 General Information

The PCIHSD driver is installed as a standard IRIX(TM) device driver. It can handle any number of PCIHSD's configured for 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 operating mode, for reading status from and resetting the PCIHSD.

2.2 PCIHSD Devices in the hwgraph

All hardware devices under IRIX 6.4 are represented as vertexes in the system hardware graph (the hwgraph) and are accessible through the /hw filesystem. The PCIHSD driver installs four vertexes in the hwgraph tree below its connection point (usually a PCI bus slot). The first (pcihsd) is for the board itself and may not be opened as a device. The other three are character device vertexes and may be used in a call to open(2). The following is a typical hwgraph structure for a PCIHSD:

```
/hw/module/1/slot/MotherBoard/node/xtalk/8/pci/6/pcihsd
```

```
/hw/module/1/slot/MotherBoard/node/xtalk/8/pci/6/pcihsd/hsd
```

```
/hw/module/1/slot/MotherBoard/node/xtalk/8/pci/6/pcihsd/ibl
```

```
/hw/module/1/slot/MotherBoard/node/xtalk/8/pci/6/pcihsd/cfg
```

pcihsd The "root" vertex, not a device

pcihsd/hsd Character device; operate in HSD mode.

pcihsd/ibl Character device; operate in IBL mode.

pcihsd/cfg Character device; used to set operating mode or check device status.

The hsd and ibl devices are created to permit read or write access by any user. The cfg device is created to be readable by any user, but writable only by root.

The system administrator may add symbolic links in the /dev directory, as required.

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/hsd or pcihsd/ibl device performs an implicit HSDSETMOD call to set the board up for HSD or IBL operation, respectively. Opening the pcihsd/cfg device does not do this, allowing the caller to examine the current board configuration. The only operations valid on the pcihsd/cfg device are the following ioctl(2) calls:

HSDGETMOD

HSDSETMOD

HSDGETXSTS

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 performed directly to the user's buffer. Therefore, the user's buffer address should be aligned on a 32-bit boundary and the transfer byte count must be a multiple of four. The maximum length of a single transfer is 262,140 bytes (65535 or FFFF16 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 4416 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. Device command transfers are buffered through the driver, but data transfers are performed directly from the user's buffer. The user's buffer address should be aligned on a 32-bit boundary and the transfer byte count must be a multiple of four. Device command transfers must be a multiple of 8 bytes long. The maximum length of a single transfer is 262,140 bytes (65535 or FFFF16 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 is 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.

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 any 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 any device open for writing, 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

This function can be issued only to the pcihsd/hsd or pcihsd/ibl device. 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). This function can be issued only to the pcihsd/hsd or pcihsd/ibl device.

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.

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 MPXÄ32 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

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.

timeout Timeout for data transfer operations. 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).

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.