CIO Asynchronous FIFO Interface Programmer's Guide

HP Part No. 27114-90003
Printed in USA September 1989

First Edition

Contents

Figures

Tables

1

Introduction

Using This Manual

This manual will guide you in writing detailed code needed to match features of the HP 27114B CIO Asynchronous FIFO Interface (AFI) Device Adapter to needs of your external device. CIO stands for Channel Input/Output.

Chapter 1, "Introduction," presents a list of primary features and a product overview that includes discussions and diagrams for the:

- Frontplane
- Midplane
- Backplane

Chapter 2, "AFI Theory of Operation," discusses configuring the AFI device adapter to an external device for transferring data. The following AFI device adapter features are discussed:

- Five configurable device dependent features
- Handshake signals
- Handshake modes
- Exclusive access
- Transferring data
- Read register configuration
- Write register configuration

Chapter 3, "Using the AFI Device Adapter," discusses accessing the AFI device driver, provides an ordered set of tasks for configuring and controlling the AFI device adapter, and provides a partial programmatic example for each task and a full programming example at the end of the chapter.

A *glossary* at the end of the manual clarifies technical terms.

References

Additional information is contained in the following Hewlett-Packard manuals:

Asynchronous FIFO Interface Reference Manual
(HP part number 27114-90004)

HP-UX System Administration Concepts Manual
(HP part number 92594-90062)

HP-UX System Administration Tasks Manual
(HP part number 92453-90038)

Concepts and Tutorials: Device I/O and User Interfacing
(HP part number 97089-90054)

Hardware Features

The AFI device adapter hardware features are:

- 16 bit parallel interface
  - 16 bit frontplane data input lines
  - 16 bit frontplane data output lines
- Differential or single-ended (with ground) signal drivers/receivers
- 64 words of FIFO buffering
- Data transfer counter
- Midplane state machine to control handshaking
- Three handshaking modes
- Up to six frontplane control lines
- Up to six frontplane status lines

Figure 1-1 illustrates the Hewlett-Packard CIO bus structure (CIB) with an AFI device adapter attached.

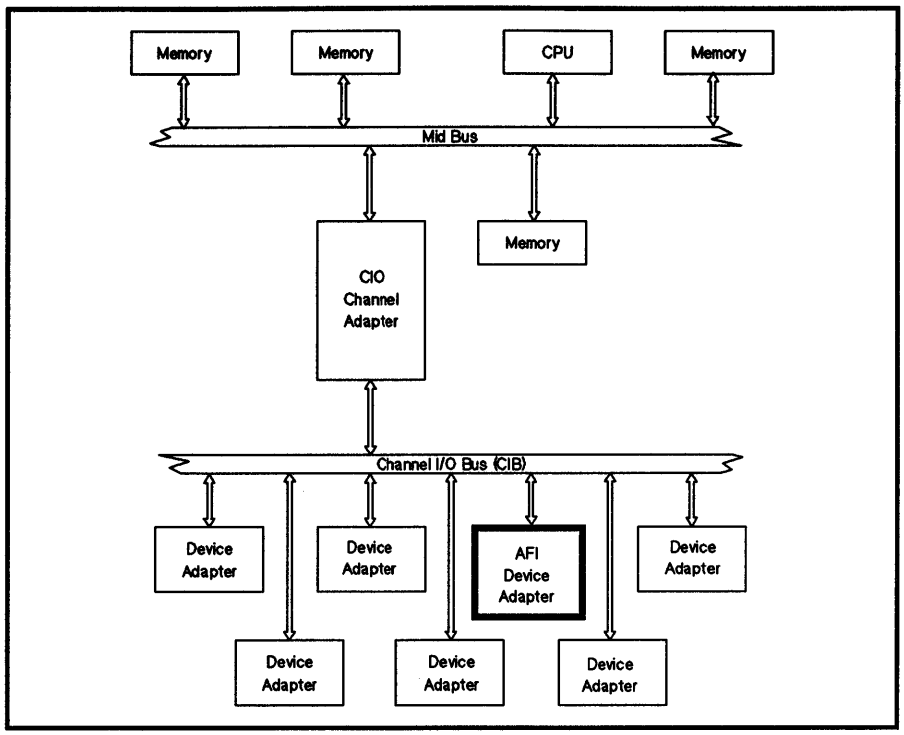

Figure 1-1.
AFI Device Adapter in a HP 9000 Series 800 CIO Computer System

Product Overview

The HP 27114B CIO Asynchronous FIFO Interface (AFI) Device Adapter provides buffered communications between external devices and HP 9000 Series 800 CIO systems using a single 64 word FIFO buffer. This device adapter is excellent for applications where high speed, 16 bit parallel transfers are required.

Figure 1-2 divides the design of the AFI device adapter into three planes of circuitry. They are:

- The frontplane (interacts with the external device and midplane)
- The midplane (interacts with the frontplane and backplane)
- The backplane (interacts with the channel adapter and midplane)

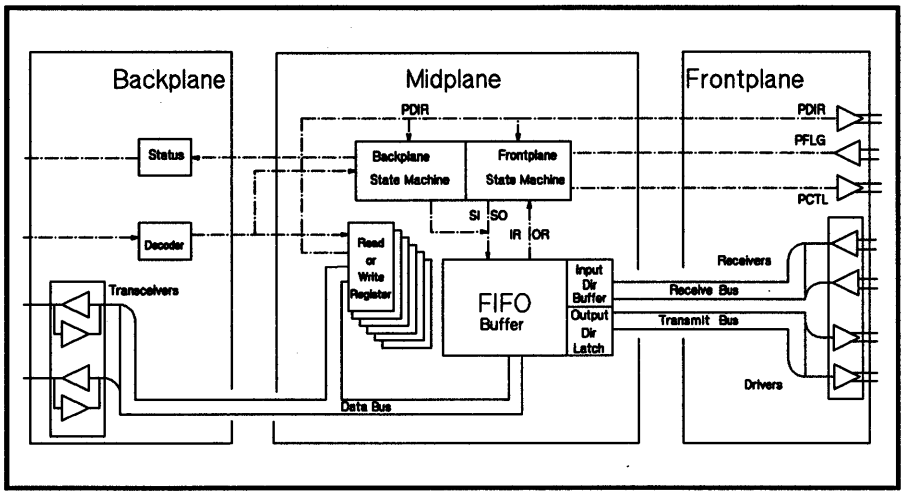

Figure 1-2. The AFI Device Adapter Planes

Each plane accepts data and control signals and converts them to the requirements of the CIO bus, the next plane, or the external device. To explain the operation of the planes, it is convenient to view the planes as working on logically independent tasks.

Frontplane

The frontplane controls or reacts to the external device and translates control signals to and from the midplane. The control signals combine in a handshake protocol with the external device to prevent data loss.

The following list contains the frontplane elements with the register representation and bit positions where 0 represents the least significant bit.

- 16-bit Data Input Bus, RD[15:0] (RD[7:0] for 8-bit data)
- 16-bit Data Output Bus, SD[15:0] (SD[7:0] for 8-bit data)
- 6-bit Control Register, CTL[5:0]
- 6-bit Status Register, STS[5:0]
- 1 interrupt signal, ATTN
- 5 handshaking signals
  1. Peripheral Control (PCTL)
  2. Peripheral Flag (PFLG)
  3. Transfer Direction (PDIR)
  4. Host (handshake) End (HEND) - optional
  5. Peripheral (handshake) End (PEND) - optional

Note

The PDIR, HEND, and ATTN signals are reflected by default in CTL[5:4] and STS[5] respectively.

This limits the number of control lines to 4 and status lines to 5. To enable all six lines for control or status you can programmatically override the default for each signal.

Frontplane handshaking signals control and monitor the sending and receiving of data to and from the external device.

Low true logic means that assertion of the signal reflects a low (0 V) state. High true logic means that assertion of the signal reflects a high (5 V) state.

To avoid confusion when discussing the assertion and deassertion of signals, Table 1-1 lists frontplane elements and their default logic sense.

Table 1-1. Logic Sense of Frontplane Elements

Frontplane Element	Logic Sense
Data Input Bus	Low True
Data Output Bus	Low True
Control Signals	Low True
Status Signals	Low True
HEND/PEND signals	Low True
PDIR	High True
PCTL/PFLG signals	High True

The *Asynchronous FIFO Interface Reference Manual* explains the logic sense and signal propagation between the hardware and software domains. The reference manual also discusses how to physically change the default logic sense of frontplane elements by crossing differential lines to external device (if the external device is differential) or by grounding one line and connecting the other to the external device (if the external device is single-ended).

Determining the logic sense of frontplane elements for your AFI device adapter is critical for configuring external device dependent features, data transfers, and interpreting status lines.

The Midplane

The midplane circuitry includes:

- the FIFO buffer and associated control circuitry
- the data, status, and control registers
- the handshake control state machine

Logically there are two state machines on the AFI device adapter. One handles data transfers between the FIFO and the frontplane and the other handles data transfers between the backplane and the FIFO.

The Backplane

The backplane contains the I/O channel interface control circuitry that controls communications between the AFI device adapter and the channel I/O bus.

2

AFI Theory of Operation

Overview

Interfacing with the AFI device adapter is divided into three categories:

1. Configuring the device adapter for an external device
2. Obtaining exclusive access to the device adapter
3. Transferring data

The following sections describe the device adapter operation with respect to the numbered categories listed above.

Configuring the Device Adapter for an External Device

There are *nine* configurable AFI device adapter attributes that are configurable for external device compatibility:

1. data path width
2. logic sense of PFLG
3. handshaking mode
4. enable or disable interrupts from external device
5. number of control lines *
6. enable PDIR and HEND signals for external device
7. control lines value
8. number of status lines *
9. enable PEND signal from external device

Note

By default, the attributes marked by an appended "*" are limited to 4 or 5 lines but are programmatically configurable to 6 lines.

The *Asynchronous FIFO Interface Reference Manual* presents an explanation of how to connect AFI device adapters to several external devices. There is a table in the AFI hardware reference manual to fill in that defines the necessary AFI device adapter configurations to consider for proper operation and use of the intended external device. Values from this table should be applied to setting the appropriate values and flags through the *gpio0* device driver. The *gpio0* device driver interface is described in section 7 of the *HP-UX Reference Manual* indexed by gpio.

Table 2-1 is a copy of the table found in the AFI hardware reference manual. You may find it more convenient to copy or fill in the values in the following table and work from this table.

Table 2-1. External Device Requirement Table

#	Attribute	External Device Requirement
1	Data Path Width	
2	PFLG Logic Sense	
3	Handshake Mode	
4	Number of Control Lines	
5	Enable PDIR/PEND in CTL[5:4]	
6	Control Lines Value	
7	Number of Status Lines	

To configure the AFI device adapter for an external device you must understand the *three* following AFI device adapter features and their operation:

- Handshake signals
- Handshaking overview with timing diagrams
- Interrupt propagation

In addition to the above listed device adapter features, the sections that follow discuss obtaining exclusive access to the device adapter and the transferring of data.

Handshake Signals

The following five signals are used for handshakes during a data transfer.

1. PCTL (Peripheral Control)
2. PFLG (Peripheral Flag)
3. PDIR (Peripheral Direction)
4. HEND (Host End)
5. PEND (Peripheral End)

Figure 2-1 shows the source and destination of these five signals.

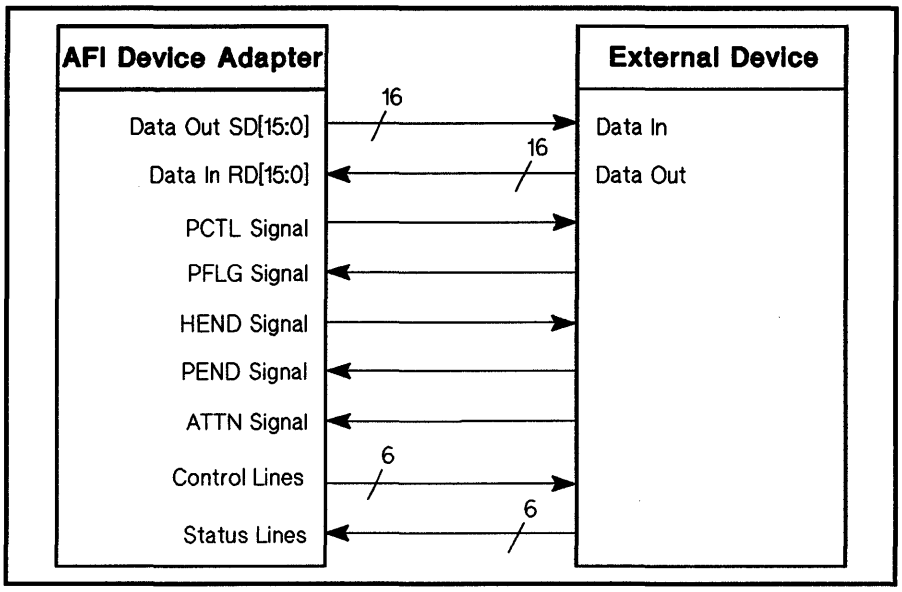

Figure 2-1. Frontplane Signals

PCTL Signal

The assertion of PCTL (peripheral control) means that the outgoing data are valid or that new incoming data can be accepted. For the FIFO_MASTER and FULL_MASTER handshaking modes, the device adapter asserts PCTL to signal readiness to start a data transfer. For FULL_SLAVE handshake mode, the device adapter asserts PCTL after the PFLG (Peripheral Flag) signal is received from the external device.

PFLG Signal

The external device asserts PFLG (peripheral flag) signal to inform the AFI device adapter that the external device has accepted the data transferred or that there are valid incoming data from the device. For incoming data there is at least one CIO bus clock pulse before data transfer is complete.

The EDGE_LOGIC_SENSE flag selects the logic sense of the PFLG signal as high true or low true. It is the handshake mode selection that interprets the edge or level state to be the PFLG signal.

The default, EDGE_LOGIC_SENSE flag deasserted, defines the rising edge or level state (5 V) of PFLG to signal the device adapter. The assertion of EDGE_LOGIC_SENSE flag defines the falling edge or low level state (0 V) of PFLG to signal the device adapter.

Note 

The handshake timing diagrams assume the EDGE_LOGIC_SENSE flag is deasserted.

PDIR Signal

The PDIR (peripheral direction) signal informs the external device which direction to perform the data transfer, incoming or outgoing. PDIR is set by the read and write system calls used for requesting a data transfer. Assertion of PDIR indicates the data transfer is an outgoing write request and the deassertion of PDIR indicated the data transfer is an incoming read request.

To enable the device adapter to send the PDIR signal to the external device the PDIR_OPT_EN flag must be asserted. By default the PDIR_OPT_EN flag is asserted. The PDIR signal is sent in CTL[5].

Note 

Assertion of PDIR_OPT_EN flag limits the number of control lines to 4, CTL[3:0]. CTL[5:4] contain the PDIR and HEND signals respectively.

Note 

The HP 27114A "DIR" signal is renamed "PDIR". Although the operation is identical, the "PDIR" name better describes the signal.

HEND Signal

The AFI device adapter can be enabled to assert the HEND (host end) signal when the last word of data is being transferred. To enable the device adapter to send the HEND signal to the external device during a read or a write data transfer, the PDIR_OPT_EN flag must be asserted. By default the PDIR_OPT_EN flag is asserted. The HEND signal is sent in CTL[4].

Note 

Assertion of PDIR_OPT_EN flag limits the number of control lines to 4, CTL[3:0]. CTL[5:4] contain the PDIR and HEND signals respectively.

See the handshake timing diagrams for specific timing of the HEND signal.

PEND Signal

The external device asserts the PEND (peripheral end) signal to inform the device adapter that the last word of data is being transferred. To enable the device adapter to recognize the PEND signal from the external device during a data transfer the PEND_OPT_EN flag must be asserted.

For example, a read request is submitted from the device adapter for 30 words of data. If the external device only has 20 words, at the end of the 20th word PEND is asserted to inform the device adapter that the last word of data is being sent. If PEND_OPT_EN is not asserted, the device adapter will not recognize the PEND signal and the device adapter will wait until the external device sends the remaining 10 words to complete the request or a timeout occurs.

By default the PEND_OPT_EN flag is deasserted enabling status information in STS[4]. When PEND_OPT_EN flag is asserted the PEND signal is acknowledged in STS[4].

Note

Assertion of PEND_OPT_EN limits the number of status lines to 4 or 5. If PEND_OPT_EN is asserted, STS[4] reflects the PEND signal. If interrupts are enabled STS[5] contains the ATTN signal which limits the number of status lines to 4, STS[3:0] respectively. If interrupts are disabled, STS[5] is available for reflecting status information and the number of status line are limited to 5, STS[5] and STS[3:0] respectively.

Handshaking Modes

AFI device adapter handshake modes provide any of three handshake protocols, illustrated in Figures 2-2 through 2-7:

- FIFO_MASTER Mode
- FULL_MASTER Mode
- FULL_SLAVE Mode

The handshaking modes are device dependent.

Note

The HP 27114A PULSE_HANDSHAKE_MODE has been renamed FIFO_MASTER. Although the operation is identical , the "FIFO" name better describes the handshake.

The following sections describe each of the handshake modes in detail. Timing diagrams and a detailed sequence of events are presented for the input and output data transfer direction for each handshake mode.

FIFO_MASTER Handshake

For the FIFO_MASTER handshake, when data is valid, the data transfer occurs on the edge transition of PFLG. The PFLG edge transition is configured with the GPIO_SET_CONFIG function, EDGE_LOGIC_SENSE flag. When the EDGE_LOGIC_SENSE flag is asserted, data transfer occurs on the asserted-to-deasserted transition trailing edge of PFLG. When the EDGE_LOGIC_SENSE flag is deasserted (default), data transfer occurs on the deasserted-to-asserted transition leading edge of PFLG.

For the FIFO_MASTER handshake, the initial PCTL signal operates independently of the PFLG signal. The AFI device adapter will assert PCTL when it is ready to transfer data, without checking the state of PFLG. After recognizing the proper edge of PFLG signal from the external device (the proper edge being determined by the setting of the EDGE_LOGIC_SENSE flag), the device adapter transfers the data then deasserts PCTL. The duration of PCTL being asserted is dependent on the time it takes for the external device to assert PFLG.

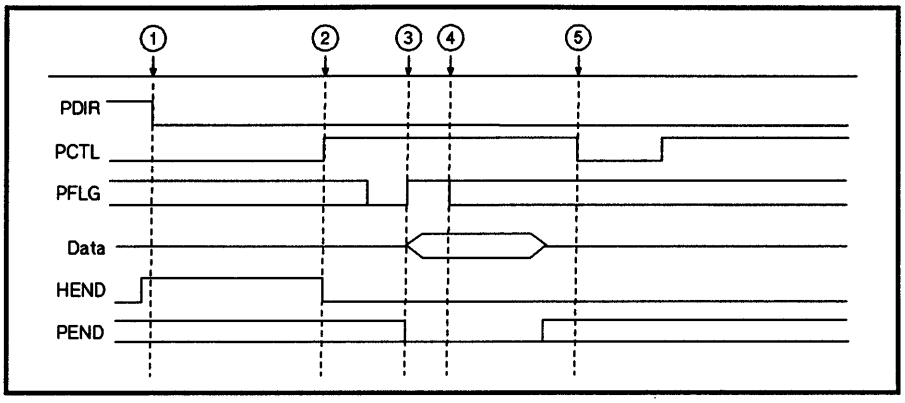

Figure 2-2. Input FIFO_MASTER Handshake

The FIFO_MASTER input handshake has the following signal sequence:

1. Device adapter deasserts PDIR (read request)
2. Device adapter asserts PCTL regardless of PFLG state
3. External device asserts PFLG (data valid and stable)
4. External device deasserts PFLG after minimum duration as configured in the hardware or PFLG may remain asserted
5. Device adapter deasserts PCTL after recognizing PFLG assertion and receiving data

Steps 2 through 5 above are repeated until last word is to be read. The last word transfer sequence is as follows:

1. PDIR remains deasserted
2. Device adapter asserts PCTL / asserts HEND (low)
3. External device asserts PFLG / asserts PEND (low)
4. External device deasserts PFLG or PFLG may remain asserted
5. Device adapter deasserts PCTL

Note

Steps 4 above are not required to happen before step 5 for input FIFO_MASTER handshake.

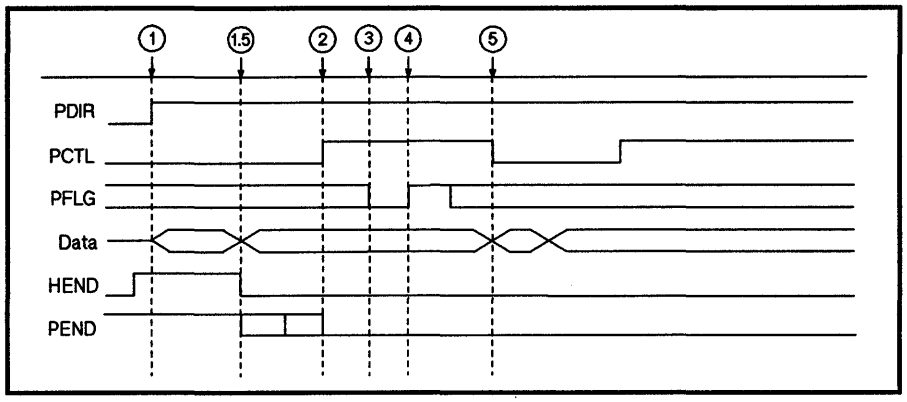

Figure 2-3. Output FIFO_MASTER Handshake

The FIFO_MASTER output handshake has the following signal sequence:

1. Device adapter asserts PDIR (write request)
2. Device adapter asserts PCTL reguardless of PFLG state (data is valid when PDIR and PCTL are asserted)
3. External device deasserts PFLG edge (data may be transferred)
4. External device asserts PFLG edge (data may be transferred)
5. Device adapter deasserts PCTL (data is invalid)

Steps 2 through 5 above are repeated until last word is to be written. The last word transfer sequence is as follows:

1. PDIR remains asserted

1.5. Device adapter asserts HEND (low)

2. Device adapter asserts PCTL / asserts HEND (low)
3. External device deasserts PFLG
4. External device asserts PFLG
5. Device adapter deasserts PCTL

Note

Steps 3 above can occur before steps 1 or 2 above, but must occur before steps 4 above for output FIFO_MASTER handshake.

PEND may be asserted by the external device if it has received the required data. If PEND is asserted low prior to the device adapter asserting PCTL, the transfer does not take place.

Note

The PEND_OPT_EN flag must be asserted for the device adapter to recognize the PEND signal from the external device.

If the PEND_OPT_EN flag is deasserted and the device adapter has more data to send after PEND has been asserted by the external

device, the device adapter will wait until the external device accepts all the data or a timeout occurs.

FULL_MASTER Handshake

For the FULL_MASTER handshake, when data is valid, the data transfer occurs when the external device signal PFLG is in a stable state. The PFLG state that triggers data to be transferred is configured with the GPIO_SET_CONFIG, EDGE_LOGIC_SENSE flag. When the EDGE_LOGIC_SENSE flag is asserted, data transfer occurs upon the PFLG signal stabilizing in a low (0 V) state. When the EDGE_LOGIC_SENSE flag is deasserted, the default, data transfer occurs upon the PFLG signal stabilizing in a high (5 V) state.

Note

The difference between FIFO_MASTER and FULL_MASTER handshakes are the PFLG requirements. For the FULL_MASTER handshake, the level state of PFLG is checked where the FIFO_MASTER handshake checks for the edge transition.

For the FULL_MASTER handshake the device adapter checks for deasserted state of PFLG before asserting PCTL to start the handshake. After PCTL is asserted, the external device asserts PFLG. The device adapter deasserts PCTL in response to the PFLG assertion. The external device then deasserts PFLG in response to the PCTL deassertion. The duration of the PCTL and PFLG signals are dependent on each other.

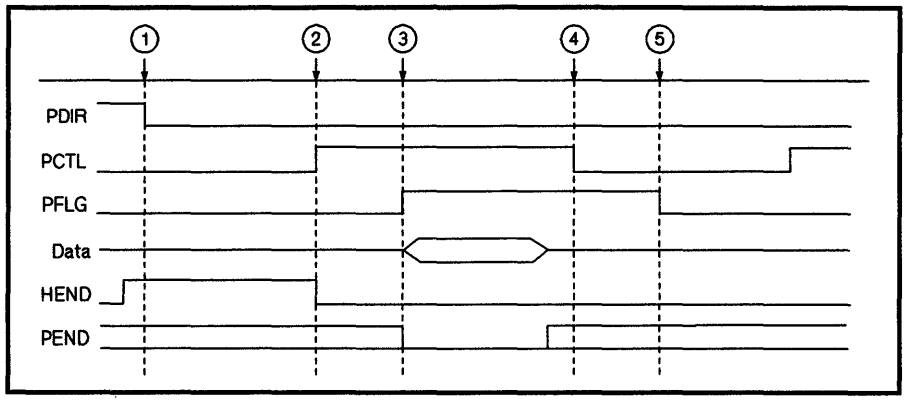

Figure 2-4. Input FULL_MASTER Handshake

The FULL_MASTER input handshake has the following signal sequence:

1. Device adapter deasserts PDIR (read request)
   (device adapter waits for PFLG to be in a deasserted state)
2. Device adapter asserts PCTL
3. External device asserts PFLG to acknowledge PCTL
4. Device adapter deasserts PCTL to acknowledge PFLG
5. External device deasserts PFLG to end handshake and enable next transfer

Steps 2 through 5 above are repeated until last word is to be read. The last word transfer sequence is as follows:

1. PDIR remains asserted
   (device adapter waits for PFLG to be in a deasserted state)
2. Device adapter asserts PCTL / asserts HEND (low)
3. External device asserts PFLG / asserts PEND (low)
4. Device adapter deasserts PCTL
5. External device deasserts PFLG to terminate transfer

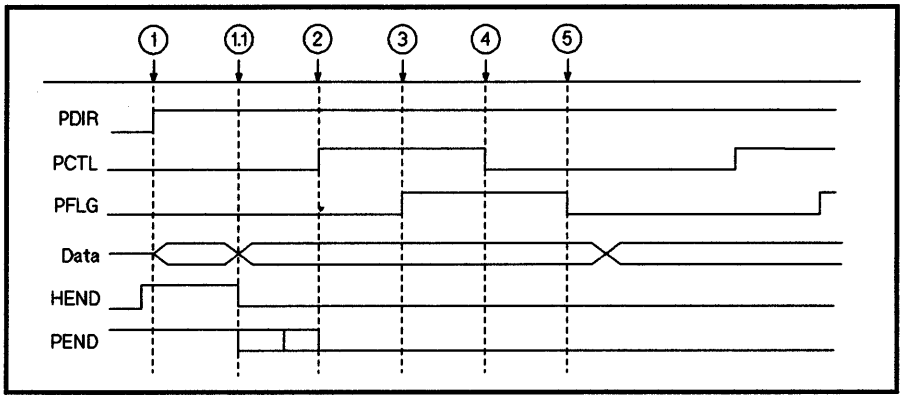

Figure 2-5. Output FULL_MASTER Handshake

The FULL_MASTER output handshake has the following signal sequence:

1. Device adapter asserts PDIR (write request)
   (device adapter waits for PFLG to be in a deasserted state)
2. Device adapter asserts PCTL
3. External device asserts PFLG to acknowledge PCTL
4. Device adapter deasserts PCTL to acknowledge PFLG
5. External device deasserts PFLG to end handshake and enable next transfer

Steps 2 through 5 above are repeated until last word is to be written. The last word transfer sequence is as follows:

1. PDIR remains asserted
   (device adapter waits for PFLG to be in a deasserted state)

1.1. Device adapter asserts HEND (low)

2. Device adapter asserts PCTL
3. External device asserts PFLG
4. Device adapter deasserts PCTL
5. External device deasserts PFLG

PEND may be asserted by the external device if it has received the required data. If PEND is asserted low prior to the device adapter asserting PCTL, the transfer does not take place.

Note

The PEND_OPT_EN flag must be asserted for the device adapter to recognize the PEND signal from the external device.

If the PEND_OPT_EN flag is deasserted and the device adapter has more data to send after PEND has been asserted by the external device, the device adapter will wait until the external device accepts all the data or a timeout occurs.

FULL_SLAVE Handshake

For the FULL_SLAVE handshake, when data is valid, the data transfer occurs when device adapter has recognized the PFLG by asserting the PCTL signal. The GPIO_SET_CONFIG function, EDGE_LOGIC_SENSE flag configures the PFLG state that triggers the data transfer.

For the FULL_SLAVE handshake, the device adapter PCTL signal is dependent upon receiving the external device PFLG signal. When the external device PFLG signal is received stating that the external device is ready to transfer data, the device adapter acknowledges the PFLG signal by asserting PCTL and the data transfer occurs. The external device deasserts PFLG to invalidate the data on the bus and the device adapter acknowledges the PFLG deassertion with the deassertion of PCTL which terminates the handshake.

The default, EDGE_LOGIC_SENSE flag deasserted, configures the data transfer to occur when PFLG stabilizes in a low (0 V) state. EDGE_LOGIC_SENSE flag asserted configures the data transfer to occur when PFLG stabilizes in a high (5 V) state.

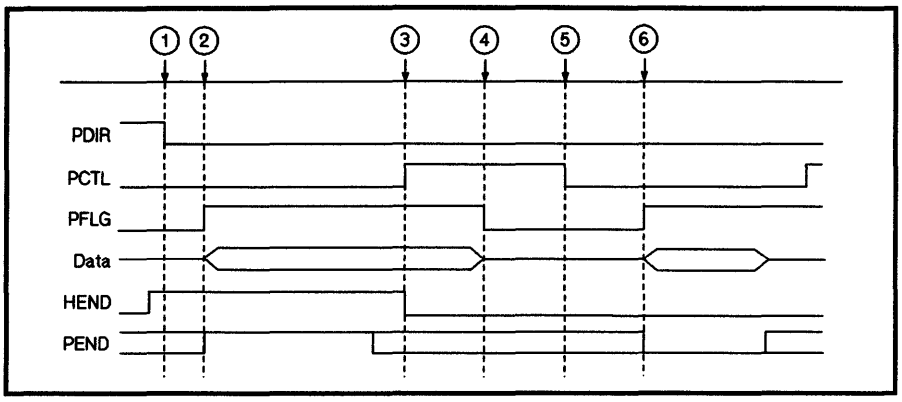

Figure 2-6. Input FULL_SLAVE Handshake

The FULL_SLAVE input handshake has the following signal sequence:

1. Device adapter deasserts PDIR (read request)
2. External device asserts PFLG
3. Device adapter asserts PCTL
4. External device deasserts PFLG
5. Device adapter deasserts PCTL

Steps 2 through 5 above are repeated until last word is to be read . The last word transfer sequence is as follows:

1. PDIR remains deasserted
2. External device asserts PFLG / deasserts PEND (high)
3. Device adapter asserts PCTL / asserts HEND (low)
4. External device deasserts PFLG / deasserts PEND (high)
5. Device adapter deasserts PCTL
6. External device asserts PEND (low)

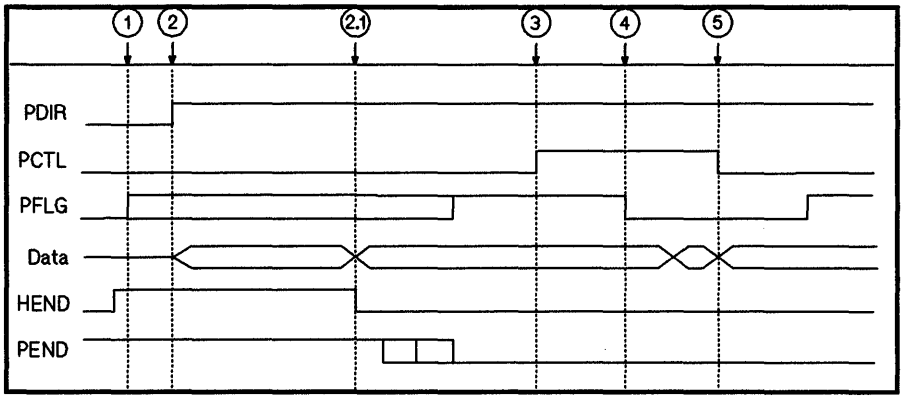

Figure 2-7. Output FULL_SLAVE Handshake

The FULL_SLAVE output handshake has the following signal sequence:

1. External device asserts PFLG
2. Device adapter asserts PDIR (write request)
3. Device adapter asserts PCTL
4. External device deasserts PFLG
5. Device adapter deasserts PCTL

Steps 1 through 5 above are repeated until last word is to be transferred. The last word transfer sequence is as follows:

1. External device asserts PFLG
2. PDIR remains asserted

2.1. Device adapter asserts HEND (low)

3. Device adapter asserts PCTL
4. External device deasserts PFLG
5. Device adapter deasserts PCTL

Note

Steps 1 above could occur after steps 2 above, but before steps 3 above, for output FULL_SLAVE handshake.

PEND may be asserted by the external device if it has received the required data. If PEND is asserted low prior to the device adapter asserting PCTL, the transfer does not take place.

Note

The PEND_OPT_EN flag must be asserted for the device adapter to recognize the PEND signal from the external device.

If the PEND_OPT_EN flag is deasserted and the device adapter has more data to send after PEND has been asserted by the external device, the device adapter will wait until the external device accepts all the data or a timeout occurs.

Interrupt Propagation

The ATTN (attention) signal is an asynchronous interrupt from the external device. Enabling and disabling interrupts enable or disable the device adapter to acknowledge the ATTN signal on the frontplane. Interrupts are enabled and disabled with the GPIO_SIGNAL_MASK function. While interrupts are enabled, the ATTN signal is reflected in STS[5].

Note

Enabling interrupts limits the number of status lines available. If the PEND_OPT_EN flag is asserted, reflecting the PEND signal in STS[4], and interrupts are enabled, the number of status lines are limited to 4, STS[3:0] respectively. If the PEND_OPT_EN flag is deasserted and interrupts are enabled, the number of status lines are limited to 5, STS[4:0] respectively.

For the AFI device adapter, the only frontplane interrupt possible is the assertion of ATTN by the external device that propagates through the AFI device adapter, backplane, *gpio0* device driver, and finally sent to the process as a SIGEMT signal. Figure 2-8 illustrates the propagation of the interrupt from the external device to the process. The interrupt signal propagation occurs independently of the handshaking and data transfer operations on the AFI device adapter.

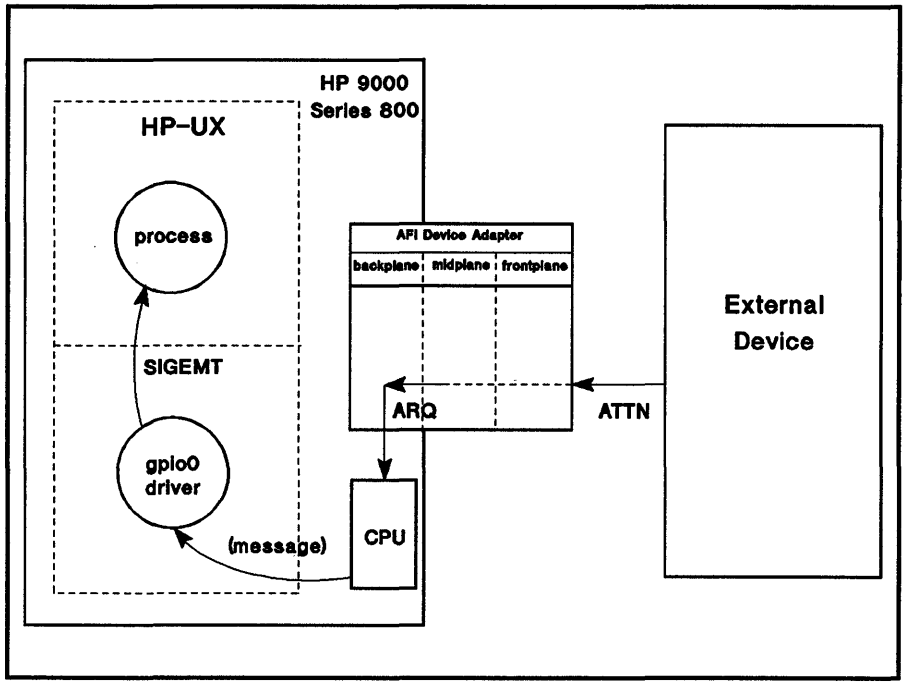

Figure 2-8. External Device Interrupt Propagation Process

Note

If interrupts are enabled you should use the signal system call to trap the SIGEMT signal sent by the *gpio0* device driver. The signal system call is described in section 2 of the *HP-UX Reference Manual.*

Exclusive Access

To guarantee exclusive access to the device adapter it must be locked. When a process locks the device adapter, all subsequent requests are denied and the requesting processes are suspended until the device adapter is unlocked or until the timeout value of the requesting process is reached. When a device adapter is unlocked and there are suspended processes waiting on the device adapter, these processes simultaneously wake up to resubmit their lock request. Only one process is given permission to lock the device adapter and subsequent requests to lock the device adapter are suspended again. There is no method for determining which process, of many awakened from a suspended state, is permitted to lock the device adapter next.

A process may open the device adapter several times, thus creating several file descriptors. A process may also lock a device adapter several times without unlocking it between lock requests. Multiple open and lock sequences affect the lock counters value returned upon a successful lock request.

The device driver increments two types of lock counters upon a successful lock:

- a per-device adapter counter

  The per-device adapter lock counter increments each time a particular process locks the device adapter.

- a per-open counter

  A per-open lock counter is kept for each file descriptor created by the open system call. The per-open counter associated with a particular file descriptor increments in addition to the per-device adapter counter.

  For processes that use system calls, a per-open counter is uniquely identified for each pid and file descriptor combination.

For example, Table 2-2 displays the counter values as a particular sequence of open and lock requests are initiated.

Table 2-2. Lock Counters Example

Sequence of Calls	Per-Device Adapter Counter	Per-Open Counter fd(A)	Per-Open Counter fd(B)
open (fd A)	0	0	0
lock (fd A)	1	1	0
lock (fd A)	2	2	0
open (fd B)	2	2	0
lock (fd B)	3	2	1

Upon completion of a successful lock the per-open counter associated with the particular file descriptor is returned in arg[1] and the current per-device adapter counter is returned in arg[2].

Both counters decrement on an GPIO_LOCK, UNLOCK_INTERFACE request. The GPIO_LOCK, CLEAR_ALL_LOCKS request resets the per-device adapter counter and the associated per-open counter(s) to zero.

Transferring Data

Transfer data with the *read* and *write* HP-UX system calls. The read and write system calls are described in section 2 of the *HP-UX Reference Manual.*

Note

Before you attempt to read or write to any device adapter, you should lock the device adapter to ensure exclusive access.

The *gpio0* device driver logically changes on-board register configuration depending on the direction of data transfer. The second part of this section discusses two register configurations for a device adapter read and write data transfers.

Register Configuration

The on-board logical register configuration changes upon a read or write request. Table 2-3 lists the register number and the associated functionality upon a read or write data transfer. Figure 2-9 illustrates the AFI device adapter registers on the bus.

Table 2-3. Register Configuration

Register Number	Read	Write
0	input data	output data
1	CIO sense reg	CIO control reg
3	CIO ID reg	(not used)
7	device adapter status reg	device adapter control
9	CIO status reg	(not used)
A	transfer counter	transfer counter
B	transfer counter	transfer counter and device adapter control

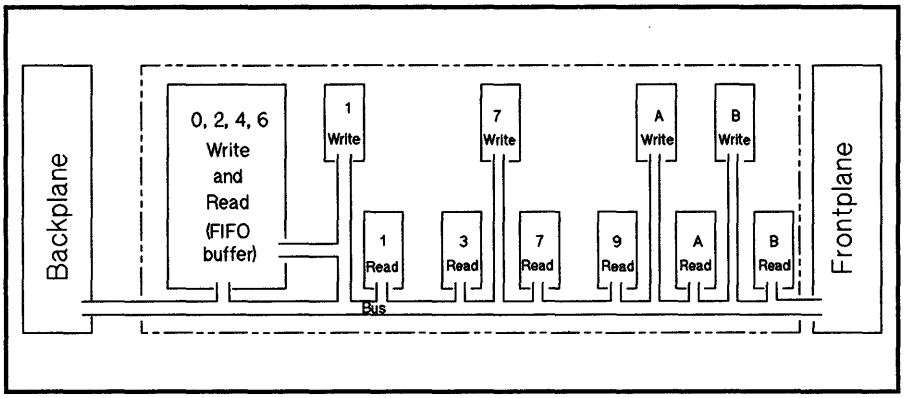

Figure 2-9. Registers on the Data Bus

Read Configuration of Registers

The read resisters contain the following information:

Register

0: input data register. The CIO bus reads a data word from this register.

1: CIO SENSE register. This register contains the interrupt pending signal bit, ARQ, and the interrupt enable signal bit ARE.

3: CIO ID register. This register contains the interface ID and revision number.

7: device adapter status register. This register contains the six status bits, the handshaking signal bits, and FIFO status. FIFO status indicates: if data exists in the FIFO and if there is room for more data, count of remaining space for data, PEND, HEND, ATTN, and ZERO signals.

9: CIO status register. This register contains a set value to be read to clear the ARQ bit.

A: transfer counter register. This register contains the low and middle bytes of the transfer counter.

B: transfer counter register. This register contains the high byte of the transfer counter.

Note

The transfer counter is 3 bytes long. Register A contains the middle and lower bytes of the transfer counter while Register B contains the upper byte of the transfer counter. The lower byte of the B Register is not used for a read request.

The following diagrams show the bit values of each of the read registers.

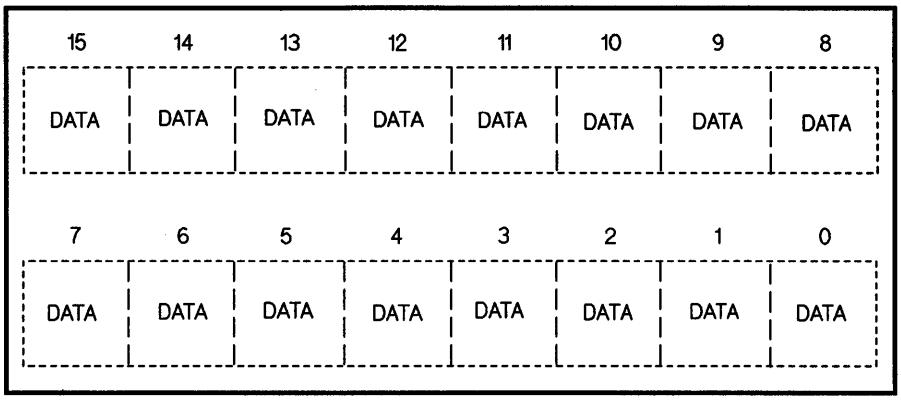

Figure 2-10. Read Register 0: Input Data

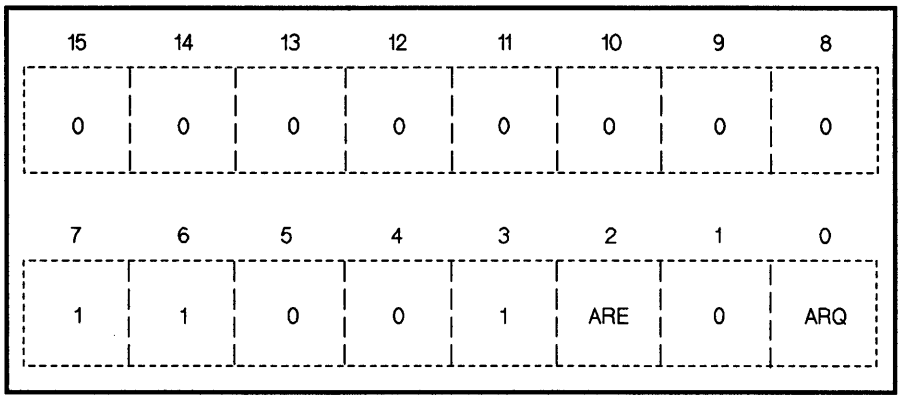

Figure 2-11. Read Register 1: CIO Sense

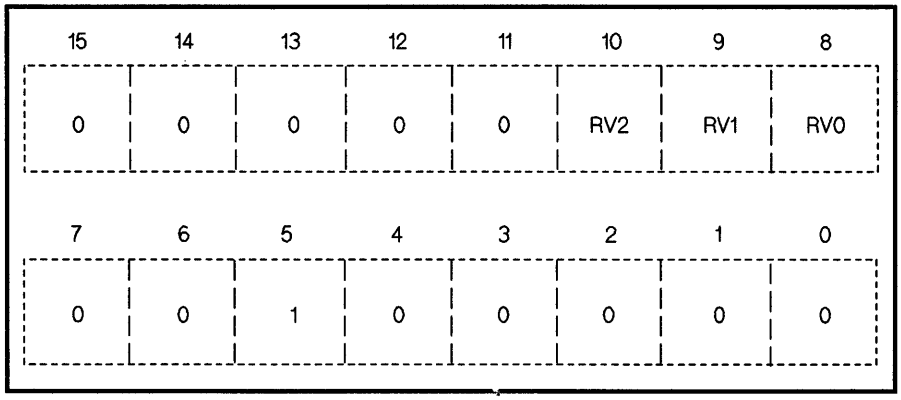

Figure 2-12. Read Register 3: CIO ID

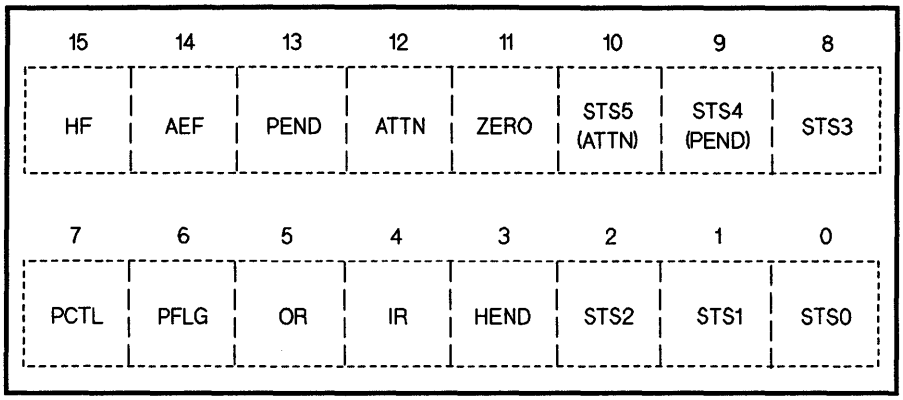

Figure 2-13. Read Register 7: Device Adapter Status

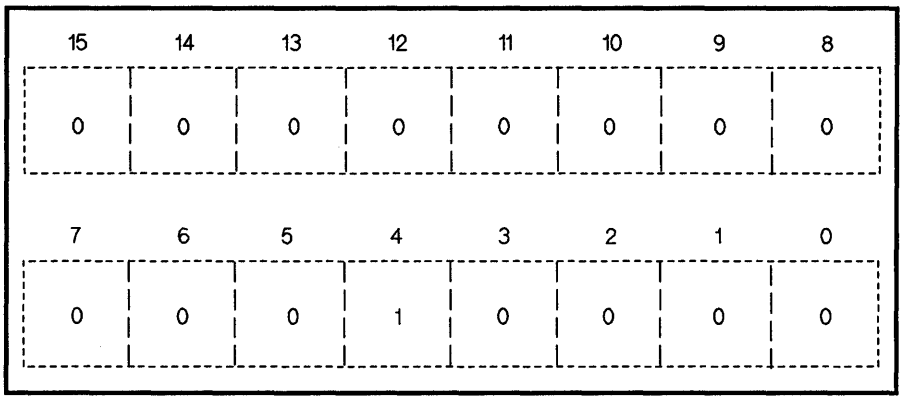

Figure 2-14. Read Register 9: CIO Status

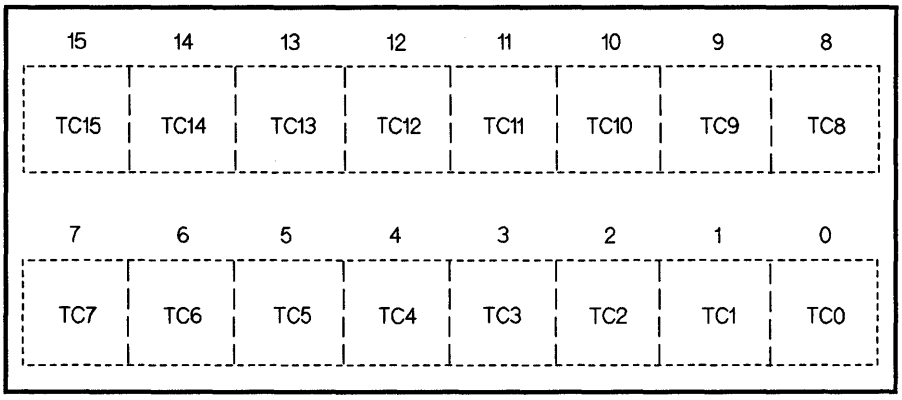

Figure 2-15. Read Register A: Transfer Counter

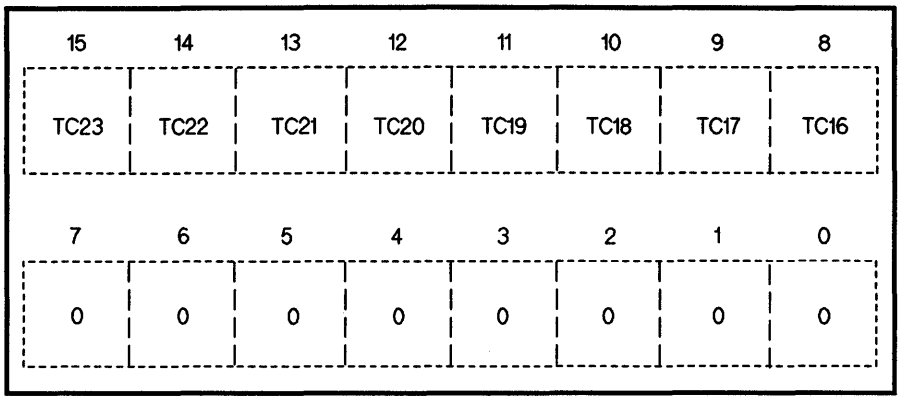

Figure 2-16. Read Register B: Transfer Counter

Write Configuration of Registers

The write resisters contain the following information:

Register

0: output data register. This register contains a data word to be put into the FIFO and ultimately put on the device adapter frontplane data lines for the external device.

1: CIO control register. This register contains the DCL, DEN, ARE, and ARD bit values.

7: Device Adapter control register. This register contains the six control bits and other internal values.

A: transfer counter register. This register contains the low and middle bytes of the transfer counter.

B: transfer counter register. This register contains the high byte of the transfer counter, bits to define the handshake mode, PDIR_OPT_EN bit, the interrupt enable bit, counter reset bit, and the PEND reset bits.

Note

The transfer counter is 3 bytes long. Register A contains the middle and lower bytes of the transfer counter while Register B contains the upper byte of the transfer counter and device adapter control information.

The following diagrams show the bit values of each of the write registers.

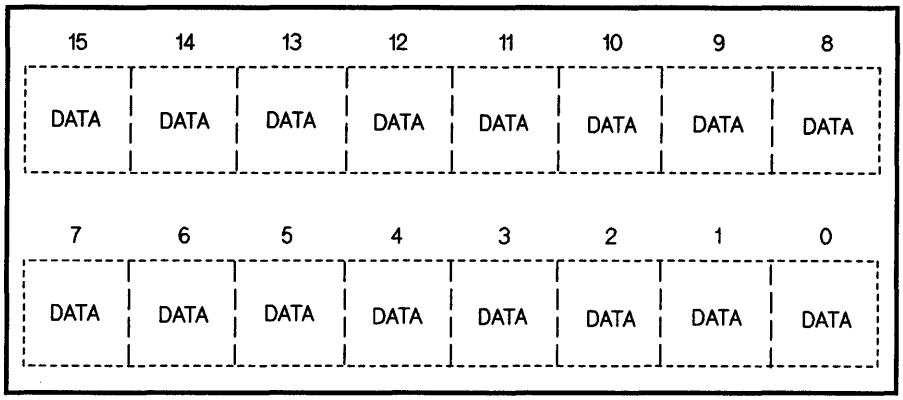

Figure 2-17. Write Register 0: Output Data

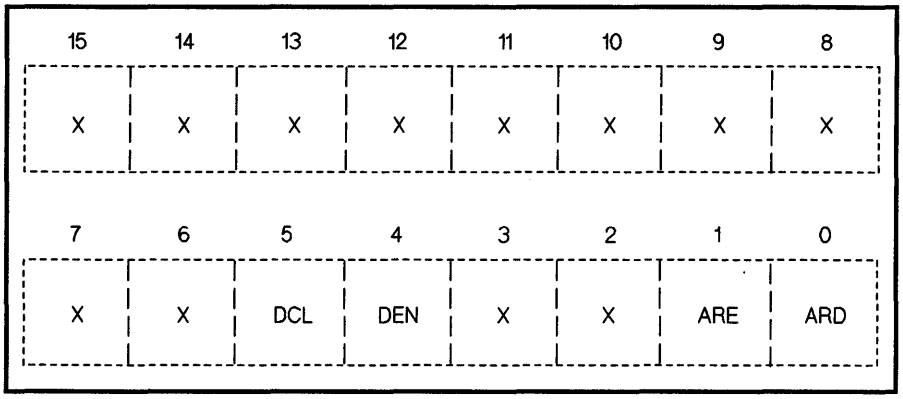

Figure 2-18. Write Register 1: CIO Control

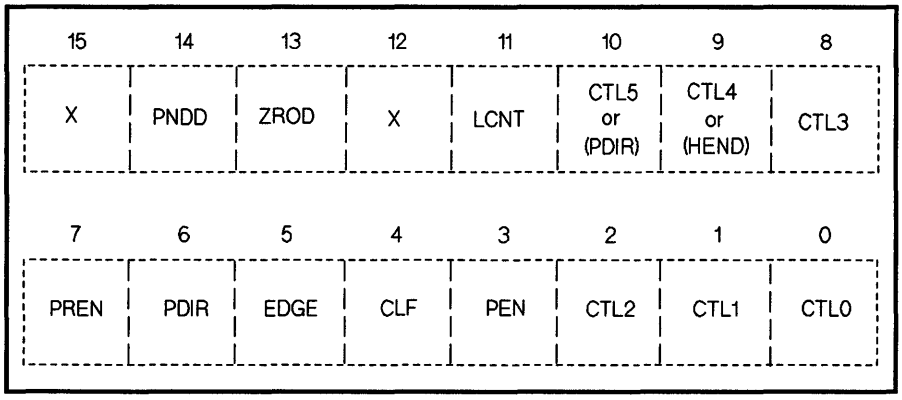

Figure 2-19. Write Register 7: Device Adapter Control

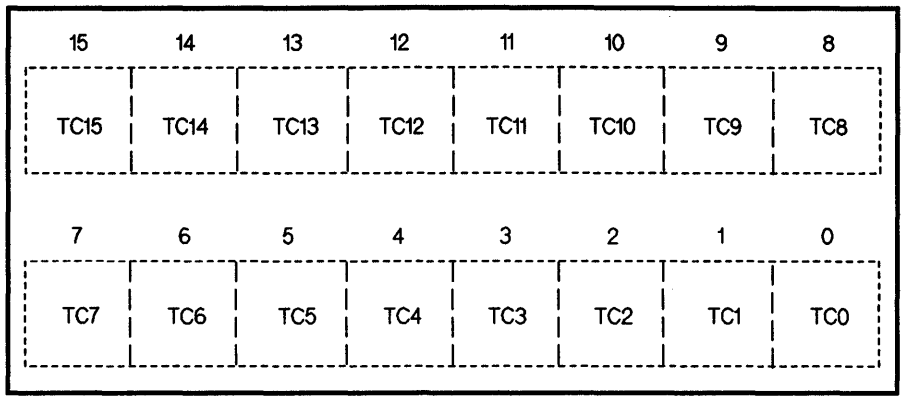

Figure 2-20. Write Register A: Transfer Counter

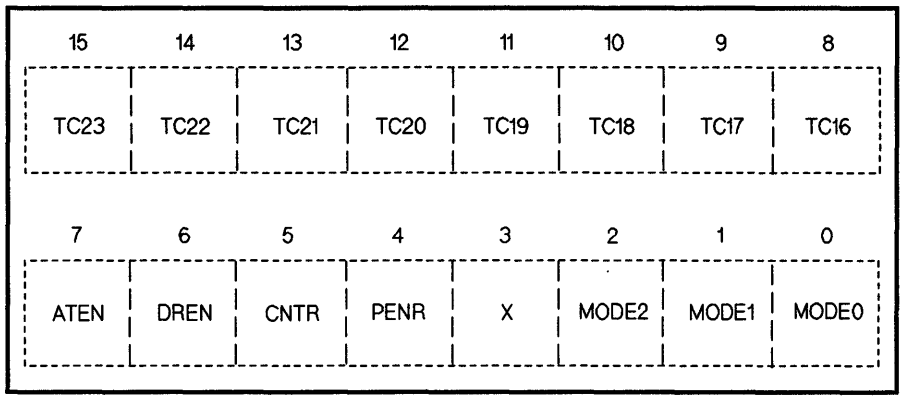

Figure 2-21.
Write Register B: Transfer Counter and Device Adapter Control

3

Using the AFI Device Adapter

There are two methods for accessing the AFI device driver, *gpio0:*

1. Calling Device I/O Library (DIL) subroutines to interface with the driver.
2. Interacting with the driver directly.

Device I/O Library Interface

The Device I/O Library (DIL) is a set of user accessible subroutines that simplify the interface between user written code and a device adapter. You must programmatically include the header file `gpio.h` to define the structures needed to talk to the AFI device adapter. The header file links are created with the C language #include statement. The DIL is located in the `/usr/lib/libdvio.a` library. Once written, compile the program and interactively link it to the DIL by entering:

```
cc filename.c -ldvio
```

Figure 3-1 shows the user access path to the AFI driver through the DIL.

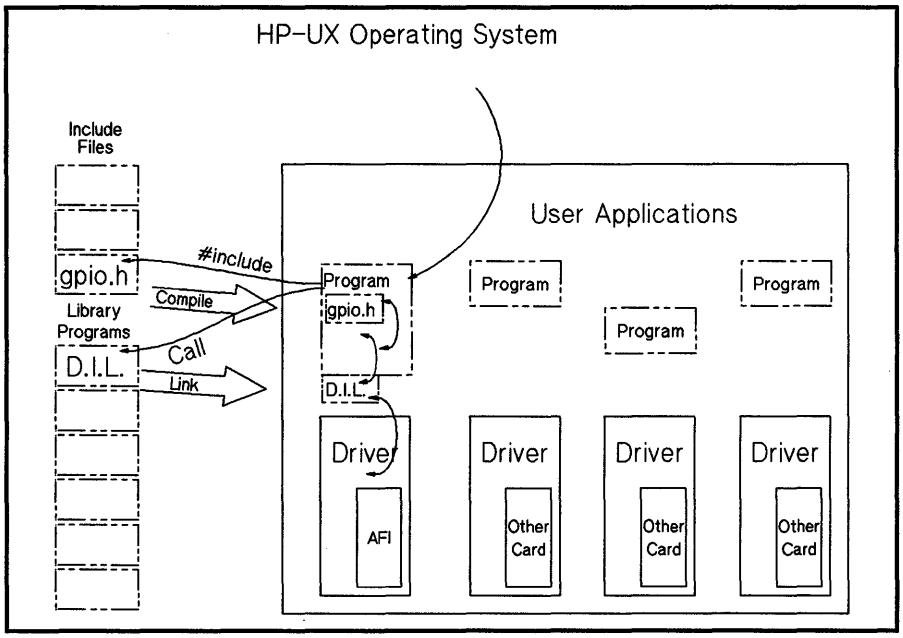

Figure 3-1. User Program Access Path Through DIL to Driver

For further information on using the DIL to access the AFI device driver, see the *Concepts and Tutorials: Device I/O and User Interfacing.*

DIL Capabilities and Limitations

While the DIL provides a common interface to communicate with device adapters, it does not allow configuration of all the AFI device adapter features.

You can use the DIL subroutines to:

- Lock and unlock the device adapter
- Reset the device adapter
- Set a timeout for transactions
- Set data path width
- Enable or disable interrupts
- Set the control lines
- Return the status lines value

You cannot use the DIL subroutines to:

- Set the edge logic sense of PFLG
- Select handshake mode.
- Enable transfer counter
- Enable PDIR and HEND flags for the external device
- Enable PEND flag value in status line
- Return process id and per-device adapter counter of a locked process
- Return the timeout value
- Return the data path width value
- Return the reason for the last interrupt
- Return device adapter specific information
- Return multiple status values in one call
- Return the configuration mask containing:
  - logic sense of PFLG
  - current handshake mode
  - transfer counter configuration
  - PDIR and HEND signals configuration
  - PEND signal configuration

Direct Device Driver Interface

There are several steps to using the AFI device adapter:

1. Creating the device file
2. Accessing and releasing the device adapter
3. Configuring the device adapter operation
4. Transferring data
5. Requesting device adapter status information

Figure 3-2 shows the user access path to the AFI driver directly.

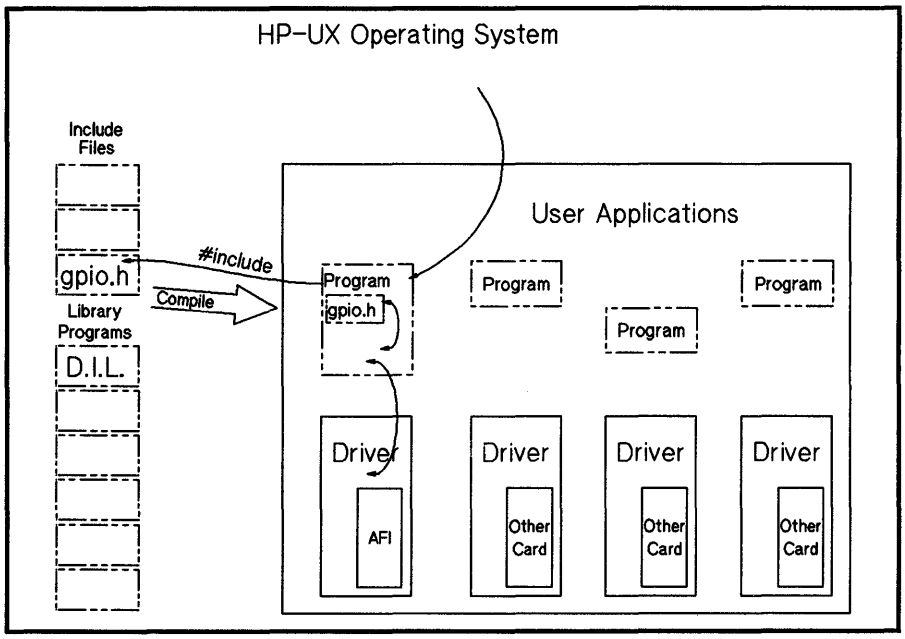

Figure 3-2. User Program Access Path Direct to Driver

Creating the Device File

The first step in accessing the AFI device adapter is to create a device file. This file modifies the original configuration file created when the computer system was installed. Create a file named `gpio0` in the `/dev` directory with the *mknod,* make node, command. The *mknod* command description is found in section 1M of the the *HP-UX Reference Manual.* For example, the following command specifies the filename to be created, character type device, the major number of 22, and the minor number of 0.

```
% mknod /dev/gpio0 c 22 0x0
```

For multiple AFI device adapters and AFI device adapter-to-AFI device adapter communication, the minor number must be changed for each device file. All the examples in this manual assume you have created a device file named **/dev/gpio0**. Refer to the *HP 9000 Series 800: System Administration Tasks Manual* for specifics on creating device files.

Accessing the Device Adapter

The second step in accessing the AFI device adapter is to communicate with the device adapter via the logically associated device file. The HP-UX operating system treats devices as files using the *open* system call on the device files to gain and release access respectively. The open call returns the file descriptor (fd) as an integer that uniquely identifies the device adapter to the process. All operations on a particular device adapter must specify the fd value. The open system call is described in section 2 of the *HP-UX Reference Manual.* The open system call syntax is as follows:

```
int fd, oflag;
char *path;
fd = open(path,oflag);
```

where:

fd	file descriptor
path	specifies the device file to be opened
oflag	specifies the access mode
	O_RDONLY for reading only
	O_WRONLY for writing only
	O_RDWR for reading and writing

For example, the following open call opens the device file for reading and writing to the device adapter.

```
int fd, oflag;
char *path;
fd = open("/dev/gpio",O_RDWR);
```

If you do not want the process to suspend upon a lock request to a currently locked device adapter, the O_NDELAY flag must be set in the open call. The O_NDELAY file status flag is set in the open call by ORing this flag with the access mode. For example, the following open call sets the O_NDELAY flag:

```
oflag = O_NDELAY | O_RDWR;
fd = open(path,oflag);
```

Note

Executing an open call does not guarantee exclusive access to the device adapter. The device adapter must be locked if you require exclusive access; see the "Locking the Device Adapter" section of this chapter.

Releasing the Device Adapter

After you have finished your I/O operations, use the close system call to release access to the device adapter. The close system call is described in section 2 of the *HP-UX Reference Manual.* The close system call syntax is as follows:

```
int fd;
close(fd);
```

where:

fd	identifies the device adapter file descriptor you wish to release.

Configuring the Device Adapter

Use the *ioctl* system call to configure device adapter attributes. The ioctl system call is described in section 2 of the *HP-UX Reference Manual.* The *ioctl* system call syntax is as follows:

```
int ret_val, fd, OP_CATGRY;
struct io_ctl_status *OP_STRUCT;
ret_val = ioctl(fd, OP_CATGRY, OP_STRUCT);
```

where:

ret_val	the status value returned after the operation request. 0 = successful operation -1 = failure
fd	integer value that uniquely identifies the device adapter you wish to operate.
OP_CATGRY	specifies operation category (status or control).
OP_STRUCT	is the data structure associated with the specified OP_CATGRY.

If the ioctl fails, indicated by a -1 value in `ret_val`, you can use the errno system call function to return a code indicating the reason for the failure. Refer to Appendix A for a list of possible errno return values. The *errno* system call is described in section 2 of the *HP-UX Reference Manual.*

AFI device adapter ioctl operations are specified in the OP_CATGRY parameter and separated into three categories:

- **IO_CONTROL.** These operations pass various configuration parameters to the device adapter. The IO_CONTROL structure includes the following fields:

```
struct io_ctl_status
{
int type;  /*control request command*/
int arg[3];/*parameters to be passed*/
} gpio_control;
```

- **IO_STATUS.** These operations request status information on a single device adapter parameter. See the "Requesting Device Adapter Status Information" section of this chapter. The IO_STATUS structure includes the following fields:

```
struct io_ctl_status
{
int type;  /*status request command*/
int arg[3];/*parameter to be received*/
} gpio_status;
```

- **IO_ENVIRONMENT.** This operation requests information on several device adapter parameters and returns their values in one structure. See the "Requesting Device Adapter Status Information" section of this chapter. The IO_ENVIRONMENT structure includes the following fields:

```
struct io_environment
{
int interface_type
int timeout;
        /* status of lines STS[5:0]*/
int status;
        /* reason for last interrupt */
int signal_mask;
        /* data path width */
int width;
        /*  locking process id */
int locking_pid;
        /*from GPIO_SET_CONFIG */
unsigned int config_mask;
} gpio_env;
```

The *type* field of the control and status structures tell the driver the particular configuration request or status request to perform. The *arg* field of the control and status structures contains a set of three integer parameters.

Each *ioctl* system call passes the configuration or status request and the associated structure containing parameters to the *gpio0* device driver.

The header file `gpio.h` defines the io_ctl_status and the io_environment structures in addition to the IO_CONTROL, IO_STATUS, and IO_ENVIRONMENT variables.

Recommended Configuration Procedure

The IO_CONTROL configurations should be performed in the following sequence after successfully opening the appropriate device file:

Note

It is recommended that you request the current configuration values with GPIO_GET_CONFIG and preserve them before reconfiguring new values.

1. Lock the device adapter (GPIO_LOCK)
2. Reset the device adapter (GPIO_RESET)
3. Set the timeout value for transactions (GPIO_TIMEOUT) * PER OPEN
4. Set the data path width value (GPIO_WIDTH)
5. Set additional configuration values (GPIO_SET_CONFIG)

   set logic sense of PFLG
   set handshake mode
   enable data transfer counter
   enable PDIR and HEND flags for external device
   enable PEND flag in a status line, STS[4]
6. Enable or disable interrupts (GPIO_SIGNAL_MASK)
7. Set the control lines value (GPIO_CTL_LINES)

AFI configurable attributes are classified into two groups:

1. Per-device adapter

   The per-device adapter configurations affect all file descriptors associated with a particular device adapter.

2. Per-open

   The per-open configurations affect only the file descriptor initiating the request.

Note 

The per-open configurations are noted in the above recommended configuration procedure by an appended asterisk '*'. The remaining configurations are per-device adapter requests.

The IO_STATUS requests have no specific order to be performed except before GPIO_SET_CONFIG if prior configuration values are to be preserved.

Note 

For all of the programmatic examples that follow, the parameters IO_CONTROL, IO_STATUS, and IO_ENVIRONMENT are defined in the gpio.h header file that you should include in your program.

Locking the Device Adapter

The GPIO_LOCK function locks or unlocks the device adapter. Once locked, only the locking process can write to or read from the device adapter. Any other process may open the device adapter, but trying to write to or read from the locked device adapter will suspend the process until the lock is removed or until a timeout occurs. However, if the O_NDELAY file flag is set, a process will not suspend if the device adapter is currently locked. Instead, the "device adapter currently locked" value, EACCESS, is returned in `ret_val` and the process resumes execution.

The O_NDELAY file flag is set during the open system call, see the "Accessing and Releasing the Device Adapter" section of this chapter.

A per-device adapter counter and a per-open counter increment upon a successful lock. For clarification of these counters see Chapter 2 of this manual. Both counters decrement on an GPIO_LOCK, UNLOCK_INTERFACE function request. The GPIO_LOCK, CLEAR_ALL_LOCKS function resets the counters to zero. When the count is zero, any process can access the device adapter. Parameters passed to lock and unlock the device adapter:

type	GPIO_LOCK
arg[0]	LOCK_INTERFACE or UNLOCK_INTERFACE or CLEAR_ALL_LOCKS
arg[2:1]	not used

Parameters returned from a successful GPIO_LOCK function:

arg[0]	not used
arg[1]	value of the per-open counter
arg[2]	value of the per-device adapter counter

For example, to lock a device adapter:

```
int fd;
struct io_ctl_status gpio_control;
gpio_control.type = GPIO_LOCK;
   /* lock device adapter    */
gpio_control.arg[0] = LOCK_INTERFACE;
ret_val = ioctl(fd, IO_CONTROL, &gpio_control);
```

Resetting the Device Adapter

The GPIO_RESET function resets all hardware and hardware related software device adapter settings to their default values. Table 3-1 lists the power up and reset default values.

Table 3-1. Power Up and Reset Configuration Values

Attribute	Power Up Value	Reset Value
Timeout	1 hour	unaltered
Interrupt	Deasserted	Deasserted
Locking	locks clear	unaltered
Control lines	000000	000000
Data path width	16 bits	unaltered
EDGE_LOGIC_SENSE	Deasserted	Deasserted
Handshake mode	FIFO_MASTER	FIFO_MASTER
TRNSFR_CTR_EN	Deasserted	Deasserted
PDIR_OPT_EN	Asserted	Asserted
PEND_OPT_EN	Deasserted	Deasserted
Data path	value unknown	value unknown
PDIR (direction)	Deasserted -read	Deasserted -read

Parameters passed to reset the device adapter:

arg[0] HW_CLR
arg[2:1] not used

GPIO_RESET does not return parameters to the calling process.

For example, to reset the device adapter:

```
int fd;
struct io_ctl_status gpio_control;
gpio_control.type = GPIO_RESET;
gpio_control.arg[0] = HW_CLR;
ret_val = ioctl(fd, IO_CONTROL, &gpio_control);
```

Setting the Timeout Value

The GPIO_TIMEOUT function sets the software timeout value for a transaction. Any DMA activity lasting longer than the time defined in arg[0] microseconds (μ) seconds will abort and return a status of ETIMEDOUT in `ret_val`. Parameters passed to set the timeout value:

type	GPIO_TIMEOUT
arg[0]	number of microseconds before a timeout occurs. (default 1 hour)
[2:1]	not used

The arg[0] value is rounded up to the next 10 μ seconds. GPIO_TIMEOUT does not return parameters to the calling process.

For example, to set the timeout to 1 second:

```
int fd;
struct io_ctl_status gpio_control;
gpio_control.type = GPIO_TIMEOUT;
   /* one second  */
gpio_control.arg[0] = 1000000;
ret_val = ioctl(fd, IO_CONTROL, &gpio_control);
```

Setting the Data Path Width Value

The GPIO_WIDTH function sets the width of the device adapter data path. The value of arg[0] must be either 8 or 16. If the width is 16, the transfer length and the buffer address specified in the read or write must be even because each transaction involves two bytes, otherwise, the error code EFAULT is returned in `ret_val`. These restrictions are based on hardware limitations of the channel adapter. Parameters passed to set the data path width:

type	GPIO_WIDTH
arg[0]	data path width (8 or 16) (default 16)
arg[2:1]	not used

GPIO_WIDTH does not return parameters to the calling process.

For example, to set the data path width:

```
int fd;
struct io_ctl_status gpio_control;
gpio_control.type = GPIO_WIDTH;
   /* data path is 16 bits wide    */
gpio_control.arg[0] = 16;
ret_val = ioctl(fd, IO_CONTROL, &gpio_control);
```

Setting Additional Configuration Values

The GPIO_SET_CONFIG function is used to:

- set the logic sense of PFLG
- set the handshaking modes
- enable and disable the transfer counter
- enable control lines CTL[5:4] for PDIR and HEND signals
- enable status line STS[4] to reflect the PEND signal

The configuration of these attributes is accomplished by bitwise ORing a list of flags to pass in the arg[0] parameter.

Note

Each GPIO_SET_CONFIG function overwrites the previous configuration. To retrieve the old configuration use the IO_STATUS, GPIO_GET_CONFIG request combination.

Parameters passed to set the flag values of GPIO_SET_CONFIG:

type	GPIO_SET_CONFIG
arg[0]	flag mask - result of ORing the flag values together
arg[2:1]	not used

GPIO_SET_CONFIG does not return parameters to the calling process.

Setting the Logic Sense of PFLG. The GPIO_SET_CONFIG function configures the edge logic sense of the PFLG signal from the external device by asserting or deasserting the EDGE_LOGIC_SENSE flag.

The deassertion of EDGE_LOGIC_SENSE flag, the default, has the following affects on the handshaking modes:

- FIFO_MASTER. The rising edge of PFLG triggers the data transfer.
- FULL_MASTER. The stabilization of the transitioning PFLG to a high (5 V) state triggers the data transfer.
- FULL_SLAVE. The stabilization of the transitioning PFLG to a high (5 V) state initiates the handshake sequence.

The assertion of EDGE_LOGIC_SENSE has the following affects on the handshaking modes:

- FIFO_MASTER. The falling edge of PFLG triggers the data transfer.
- FULL_MASTER. The stabilization of the transitioning PFLG to a low (0 V) state triggers the data transfer.
- FULL_SLAVE. The stabilization of the transitioning PFLG to a low (0 V) state initiates the handshake sequence.

Setting the Handshake Mode. The GPIO_SET_CONFIG function sets the handshake mode. The handshake mode is device dependent. The possible handshake mode flags are:

- FIFO_MASTER (default)
- FULL_MASTER
- SLAVE_MASTER

Note

The HP 27114A PULSE_HANDSHAKE_MODE has been renamed FIFO_MASTER. Although the operation is identical, the "FIFO" name better describes the handshake.

See Chapter 2 for timing diagrams and clarification of the three handshake modes.

Enabling the Data Transfer Counter. The GPIO_SET_CONFIG function enables the data transfer counter by asserting the TRNSFR_CTR_EN flag. The transfer counter monitors the number of words put in the 66 word FIFO buffer. This ensures the device adapter only accepts the requested amount of data into the FIFO and transferred. Default is TRNSFR_CTR_EN deasserted.

Note

Enabling the transfer counter is recommended.

Enabling the PDIR and HEND Signals for the External Device. The GPIO_SET_CONFIG function enables control lines 5 and 6, CTL[5:4], to reflect the current values of the PDIR and HEND signals respectively by asserting the PDIR_OPT_EN flag. Default is PDIR_OPT_EN flag asserted.

Enabling PEND Signal in Status Line. The GPIO_SET_CONFIG function enables the PEND signal to be reflected in status line 5, STS[4], by asserting the PEND_OPT_EN flag. Default is PEND_OPT_EN flag deasserted and STS[4] reflects status information.

GPIO_SET_CONFIG Example. Suppose the following configurations:

- set the falling edge of PFLG to trigger data transfer
- set the handshake mode to FULL_MASTER
- enable the transfer counter
- enable PDIR and HEND values on the control lines, CTL[5:4]
- enable device adapter to recognize the PEND signal in STS[4]

The following example performs the above configurations:

```
int fd;
struct io_ctl_status gpio_control;
gpio_control.type = GPIO_SET_CONFIG;
gpio_control.arg[0] = (EDGE_LOGIC_SENSE | FULL_MASTER |
                       TRNSFR_CTR_EN | PDIR_OPT_EN |
                       PEND_OPT_EN)
ret_val = ioctl(fd, IO_CONTROL, &gpio_control);
```

Enabling and Disabling External Device Interrupts

The GPIO_SIGNAL_MASK function enables and disables the ability of the device adapter to assert the backplane ARQ signal ARQ after the external device signal, ATTN, is detected. The ATTN signal is the only external interrupt that can cause the AFI device adapter to request the *gpio0* device adapter to send the SIGEMT signal to your executing process. You should write your code to trap the SIGEMT signal with the signal system call to diagnose the source of the interrupt . Parameters passed to enable the interrupt:

type	GPIO_SIGNAL_MASK
arg[0]	ST_ARQ2 (ATTN) (default interrupts disabled)
arg[2:1]	not used

GPIO_SIGNAL_MASK does not return parameters to the calling process.

For example, to enable the interrupt:

```
int fd;
struct io_ctl_status gpio_control;
gpio_control.type = GPIO_SIGNAL_MASK;
gpio_control.arg[0] = ST_ARQ2;
ret_val = ioctl(fd, IO_CONTROL, &gpio_control);
```

Note

After a SIGEMT signal is received, you must re-enable the interrupt with the GPIO_SIGNAL_MASK function to receive more interrupt signals.

Setting the Control Line Values

The GPIO_CTL_LINES function sets or clears the six control line outputs on the AFI device adapter frontplane. You accomplish this by using the complement of a number whose binary representation yields the bit pattern that reflects the control line states required. For instance, the bit pattern 010110 equals 16 hexidecimal (0x016) and would drive the frontplane lines: CTL[0] high (5 V), CTL[1:2] low (0 V), CTL[3] high, CTL[4] low, and CTL[5] high. There is an inversion between the software and the hardware domains.

Parameters passed to configure the control lines:

type	GPIO_CTL_LINES
arg[0]	value of control lines in decimal, binary, octal, or hexidecimal (default value 0)
arg[2:1]	not used

GPIO_SIGNAL_MASK does not return parameters to the calling process.

For example, to set the six control lines to 110011:

```
int fd;
struct io_ctl_status gpio_control;
gpio_control.type = GPIO_CTL_LINES;
 /* set software flag 001100 to reflect */
 /* 110011 on the frontplane control lines */
gpio_control.arg[0] = 0x016;
ret_val = ioctl(fd, IO_CONTROL, &gpio_control);
```

Note

Control lines 5 and 6, CTL[5:4], are not user configurable if PDIR_OPT_EN flag is asserted. Instead, CTL[5:4] will contain the PDIR and HEND signals respectively.

Transferring Data

After the device file is open and the device adapter configurations are set, the *read* and *write* system calls transfer the data to or from the device adapter. In the following example, the program has previously declared `fd`, `length`, `Num_Bytes_Trans` as integers and the character arrays `inbuf` and `outbuf`. The variable `length` holds the number of bytes to transfer.

```
if ((Num_Bytes_Trans = write(fd,outbuf,length)) == -1)
   printf("Error %d in data write to device adapter.\n" ,errno);
if ((Num_Bytes_Trans = read(fd,inbuf,length)) == -1)
   printf("Error %d in data read from file.\n" ,errno);
```

Note

The value returned in `Num_Bytes_Trans` may not equal parameter value `length` when PEND_OPT_EN is asserted and TRNSFR_CTR_EN is deasserted.

If PEND_OPT_EN is asserted and TRNSFR_CTR_EN is deasserted then not only could `Num_Bytes_Trans` not equal `length` but `Num_Bytes_Trans` may be incorrect for a write request. Therefore, we recommend to assert the TRNSFR_CTR_EN if the PEND_OPT_EN is asserted when requesting to write to the external device.

Requesting Device Adapter Status Information

Request device adapter status information by specifying `IO_STATUS` in the OP_CATGRY parameter of the ioctl call. Additionally, the status structure passes a parameter in the *type* field containing the particular status request. The returned status values are found in one or more of the *arg* integer array fields.

The IO_STATUS requests available are:

- Return the process id and the per-device adapter count of a locked device adapter
- Return the timeout value
- Return the data width value
- Return the reason for the last interrupt
- Return the status lines value
- Return device adapter specific information
- Return multiple status values
- Return the configuration mask containing:
  - logic sense of PFLG
  - current handshake mode
  - transfer counter configuration
  - PDIR and HEND signals configuration
  - PEND signal configuration

The following sections describe the IO_STATUS requests available.

Returning the Process Id and Per-Device Adapter Counter of a Locked Device Adapter

The GPIO_LOCK status request returns the pid of the process currently locking the specified device adapter and the per-device adapter lock count. The *type* field should contain the value GPIO_LOCK to request this status information. If it is not locked, the value returned will be -1. For example, to submit a status request for a locked device adapter:

```
int fd;
struct io_ctl_status gpio_status;
gpio_status.type = GPIO_LOCK;
ret_val = ioctl(fd, IO_STATUS, &gpio_status);
```

Parameters returned are:

arg[0] process id of the locked AFI device adapter or -1.
arg[1] per-device adapter lock count.
arg[2] not used

Returning the Timeout Value

The GPIO_TIMEOUT status request returns the current timeout value in microseconds (μ) seconds. The *type* field should contain the value GPIO_TIMEOUT to request this status information. For example:

```
int fd;
struct io_ctl_status gpio_status;
gpio_status.type = GPIO_TIMEOUT;
ret_val = ioctl(fd, IO_STATUS, &gpio_status);
```

Parameters returned are:

arg[0] current timeout value in microseconds.
arg[2:1] not used

A return value of zero means that the timeout will be 1 hour.

Returning the Data Path Width Value

The GPIO_WIDTH status request returns the current device adapter path width value. The *type* field should contain the value GPIO_WIDTH to request this status information. For example:

```
int fd;
struct io_ctl_status gpio_status;
gpio_status.type = GPIO_WIDTH;
ret_val = ioctl(fd, IO_STATUS, &gpio_status);
```

Parameters returned are:

arg[0] current data path width, either 8 or 16.
arg[2:1] not used

Returning Reason for Last Interrupt

Use the *type* field value GPIO_SIGNAL_MASK to return the source of the last interrupt. For the AFI device adapter, the returned value will always be the external interrupt, ST_ARQ2 flag or the value 0 (indicating that this AFI did not generate the interrupt). This request is useful when multiple AFI device adapters are being used. If ST_ARQ2 is returned in the structure, this particular AFI device adapter associated with the specified file descriptor generated the interrupt. For example:

```
int fd;
struct io_ctl_status gpio_status;
gpio_status.type = GPIO_SIGNAL_MASK;
ret_val = ioctl(fd, IO_STATUS, &gpio_status);
if (gpio_status.arg[0] != ST_ARQ)
   printf("no ARQ sent by this device adapter\n");
```

Parameters returned are:

arg[0] ST_ARQ2 or 0 (zero).
arg[2:1] not used

Returning the Status Line Values

The GPIO_STS_LINES status request returns the value of the six status lines set by the external device. The *type* field value should contain the value GPIO_STS_LINES to request this status information. For example:

```
int fd;
struct io_ctl_status gpio_status;
gpio_status.type = GPIO_STS_LINES;
ret_val = ioctl(fd, IO_STATUS, &gpio_status);
```

Parameters returned are:

arg[0] is an integer whose binary representation yields the current state of each of the six status lines, STS[5:0]. The least significant bit is STS[0]. Thus, 0x03 (00011BIN) would indicate that STS[1:0] are both low (0 V), while STS[4:2] are high (5 V) due to the hardware inversion that takes place at the frontplane.
arg[2:1] are unused.

Returning Device Adapter Specific Information

The GPIO_INTERFACE_TYPE status request returns device adapter specific identification data: The device adapter id and the revision number of the device adapter. The *type* field should contain the value GPIO_INTERFACE_TYPE to request this status information. For example:

```
int fd;
struct io_ctl_status gpio_status;
gpio_status.type = GPIO_INTERFACE_TYPE;
ret_val = ioctl(fd, IO_STATUS, &gpio_status);
```

Parameters returned are:

arg[0] device adapter identification
where:
bits[7:0] = device adapter id
bits[15:8] = device adapter revision number
bits[31:16] = 0

arg[2:1] not used

Note

In the bit descriptions above bit[0] represents the least significant bit.

Returning Multiple Status Values

This section presents the IO_ENVIRONMENT category request that retrieves several of the status variables in one operation. The include file <gpio.h> holds the definition of the io_environment structure. The io_environment structure has the following fields:

- interface_type
- timeout
- status
- signal_mask
- width
- locking_pid
- config_mask

Since there is only one function request for this category, there is no need to specify the *type* field value. For example:

```
int ret_val,fd;
struct io_environment gpio_env;
ret_val = ioctl(fd, IO_ENVIRONMENT, &gpio_env);
```

Parameter fields returned in the gpio_env structure are:

interface_type is the device adapter revision and id number.

timeout is the number of microseconds the host will wait for device adapter transaction to complete.

status is an integer whose binary representation gives the status of each of the six status lines STS[5:0].

signal_mask is the reason for the last interrupt. For the AFI device adapter, this will always be ST_ARQ2, the external device interrupt signal ATTN, or 0.

width is the data path width, always 8 or 16 bits.

locking_pid is the process id of the process that locked the AFI device adapter, or -1 if the device adapter is unlocked.

config_mask is the ORed flags bit pattern set by the GPIO_SET_CONFIG function.

Returning the Configuration Mask

The GPIO_GET_CONFIG status request returns the current configuration mask. This mask is an integer value that is a result of the following device adapter configuration flags ORed together:

- EDGE_LOGIC_SENSE
- FIFO_MASTER, FULL_MASTER, or FULL_SLAVE
- TRNSFR_CTR_EN
- PDIR_OPT_EN
- PEND_OPT_EN

These flags represent the following attributes respectively:

- edge logic sense of PFLG
- handshaking modes
- transfer counter enable
- control lines [5:4] as PDIR and HEND signal values
- status line [4] as the PEND signal value

The *type* field should contain the value GPIO_GET_CONFIG to request this status information.

To extract the current configuration flags from the mask, a series of logical AND operations must be performed. The returned mask must be ANDed with the flag value you are testing for. If the result of a particular AND operation reflects the value of the flag being tested for, then the current configuration reflects the assertion of that flag. For example, to request the current configuration and test for the PDIR_OPT_EN flag asserted:

```
int fd;
struct io_ctl_status gpio_status;
gpio_status.type = GPIO_GET_CONFIG;
ret_val = ioctl(fd, IO_STATUS, &gpio_status);

/* test for occurrence of PDIR_OPT_EN flag   */

if (gpio_status.arg[0] & PDIR_OPT_EN)
    /* then PDIR_OPT_EN flag is asserted */
printf("PDIR/HEND signals enable flag set\n");
```

Parameters returned are:

arg[0]	is an integer whose value equals a logical ORing of the individual binary codes used for each of the configuration flags
arg[2:1]	are not used.

Programmatic Example

The following sample program uses the direct driver interface method to request status information and configure a device adapter. The program scenario is as follows:

1. open device adapter
2. lock device adapter
3. reset device adapter
4. set timeout value to 1 second
5. get revision number
6. if device adapter is a B product enable the transfer counter, set handshaking mode to FULL_SLAVE, and set the PDIR_OPT_EN flag.
7. if device adapter is an A product set handshaking mode to FULL_SLAVE and set the PDIR_OPT_EN flag
8. enable interrupts
9. set control line value
10. get multiple status (IO_ENVIRONMENT)
11. check for TRNSFR_CTR_EN flag
12. check for PDIR_OPT_EN flag
13. check for specific status line value

```
#include </usr/include/sys/gpio.h>
#include <signal.h>
#include <errno.h>
#include </usr/include/sys/ioctl.h>
#include <fcntl.h>
#include <stdio.h>
#define revB 0x0200

int ret_val,fd, omode;
struct io_ctl_status gpio_status, gpio_control;
struct io_environment gpio_env;

fd = open("/dev/gpio",omode);

/* lock the device adapter */

gpio_control.type = GPIO_LOCK;
gpio_control.arg[0] = LOCK_INTERFACE;
if((ret_val = ioctl(fd, IO_CONTROL, &gpio_control)) == -1;
   printf("Error %d in ioctl \n",errno);

/* reset the device adapter */

gpio_control.type = GPIO_RESET;
gpio_control.arg[0] = HW_CLR;
if ((ret_val = ioctl(fd, IO_CONTROL, &gpio_control)) == -1;
   printf("Error %d in ioctl \n",errno);

/* set the timeout value to 1 second */

gpio_control.type = GPIO_TIMEOUT;
gpio_control.arg[0] = 1000000;
if((ret_val = ioctl(fd, IO_CONTROL, &gpio_control)) == -1);
   printf("Error %d in ioctl \n",errno);

/* set the data path width to 8 bits */

gpio_control.type = GPIO_WIDTH;
gpio_control.arg[0] = 8;
if((ret_val = ioctl(fd, IO_CONTROL, &gpio_control)) == -1);
   printf("Error %d in ioctl \n",errno);

/* get device adapter revision number */

gpio_status.type = GPIO_INTERFACE_TYPE;
if((ret_val = ioctl(fd, IO_STATUS, &gpio_status)) == -1);
   printf("Error %d in ioctl \n",errno);
```

```
/* revision number bits are arg[15:8]          */
/* The 27114A device adapter has 2 revision    */
/* numbers: 00000000 or 00000001.              */
/* The 27114B device adapter has revision      */
/* numbers greater than or equal to 00000010. */

gpio_control.type = GPIO_SET_CONFIG;

/* test for occurrence of the 27114B product   */

if ((gpio_status.arg[0] & 0x0700) >= revB)

/* You have a 27114B device adapter       */
/* enable transfer counter feature,       */
/* set handshake mode to FULL_SLAVE,      */
/* and enable PDIR and HEND in CTL[5:4]. */
gpio_control.arg[0] = (FULL_SLAVE | TRNSFR_CTR_EN |
                       PDIR_OPT_EN);
else

/* You have an 27114A device adapter        */
/* do not enable transfer counter feature, */
/* set handshake mode to FULL_SLAVE,        */
/* and enable PDIR and HEND in CTL[5:4].    */

gpio_control.arg[0] = (FULL_SLAVE | PDIR_OPT_EN);

/* set configuration with mask created by       */
/* ORed flags contained in gpio_control.arg[0].*/

if((ret_val = ioctl(fd, IO_CONTROL, &gpio_control)) == -1);
   printf("Error %d in ioctl \n",errno);

/* enable interrupts                */

gpio_control.type = GPIO_SIGNAL_MASK;
gpio_control.arg[0] = ST_ARQ2;
if((ret_val = ioctl(fd, IO_CONTROL, &gpio_control)) == -1);
   printf("Error %d in ioctl \n",errno);
```

```
/* set control lines to reflect 1010, CTL[3:0], */
/* on the frontplane by specifying 0101 in the  */
/* parameters passed to ioctl.                   */

gpio_control.type = GPIO_CTL_LINES;
gpio_control.arg[0] = 0x0005;
if((ret_val = ioctl(fd, IO_CONTROL, &gpio_control)) == -1);
   printf("Error %d in ioctl \n",errno);

/* Now you are ready to read or write to the    */
/* AFI device adapter.  Insert read or write    */
/* statements here.                             */

/* Checking device adapter status.              */

if((ret_val = ioctl(fd, IO_ENVIRONMENT, &gpio_env)) == -1);
   printf("Error %d in ioctl \n",errno);

/* check for occurrence of transfer counter flag */
/* THIS CHECK IS ONLY VALID FOR THE 27114B CARD. */

if (gpio_env.config_mask & TRNSFR_CTR_EN)
  printf("the transfer counter is enabled\n");
else
  printf("the transfer counter is not enabled\n");

/* check for occurrence of PDIR/HEND flag  */

if (gpio_env.config_mask & PDIR_OPT_EN)
  printf("the PDIR and HEND signals are enabled\n");
else
  printf("the PDIR and HEND signals are not enabled\n");

/* You may check for all possible flag combinations. */

/* check the value of the status lines, STS[5:0],  */
/* Assume that you are checking for a status value */
/* of 100111 from your external device.          */

if (gpio_env.status == 0x0018)

/* status line reflects 100111 on the frontplane.*/
/* Note that the complement of 100111 is         */
/* is 011000 which is 18 hexidecimal.            */

printf("status lines reflect 100111\n");
```

A

Error Code Values

This appendix contains error codes that may occur when making ioctl(2) calls to the gpio driver. The error code value may be found by calling the errno(2) function. The error code values possible when making ioctl(2) calls to the gpio driver are:

[EACCES] the access to the specific device file cannot be granted without the proper minor number or the interface is currently locked via GPIO_LOCK.

[EFAULT] I/O request specified odd byte count or odd address on 16-bit AFI data path width.

[EINTR] an interface power failure occurred during the processing of this request; the device might have lost state.

[EINVAL] an attempt was made to unlock an interface that was not locked or invalid command or invalid parameter.

[EIO] some unclassified error occured.

[EMFILE] the maximum number of simultaneous opens on this interface exceeded.

[ENXIO] there is no bus interface associeated with the device file.

[EPERM] an attempt is made to unlock when lock is not owned by this user.

[ERANGE] the interface lock count limit exceeded.

[ETINEDOUT] the transaction did not complete within the timeout specified.

Glossary

CIB
Channel I/O Bus.

CIO
Channel I/O.

DIL
Device I/O Library.

FIFO
First In First Out. The 64 word buffer that resides on the AFI device adapter midplane which is referred to in this manual as the FIFO.

assert
Forces a logical true condition to occur. The signal value upon assertion is dependent upon the logic sense of the signal. Assert is the opposite of deassert. See logic sense.

backplane
Sub-system of the AFI device adapter that interfaces with the host computer CIB.

deassert
Forces a logical false condition to occur. The signal value upon deassertion is dependent upon the logic sense of the signal. Deassert is the opposite of assert. See logic sense.

device adapter
A hardware interface between the host computer CIB and an external device. The device adapter handles the protocol for communication and data transfers with the external device. A device adapter occupies at least one I/O slot in the host computer card cage.

device driver
A kernel module that controls the operation of external devices or device adapters. The gpio0 device driver controls the operation of the AFI device adapter.

device file
Resides in the /dev directory and represents a particular device. I/O requests are submitted to the device file. The device file

contains the entry points within the device driver and the physical address of the particular device on the I/O bus.

differential driver
Sends data to a receiver over two lines, both of which are isolated from ground. The voltage difference between the two lines determines the logic sense.

differential receiver
Receives data from a driver over two lines, both of which are isolated from ground. The voltage difference between the two lines determines the logic sense.

exclusive access
Exclusive access is a condition that requires that one and only one process can access a particular resource at a given time.

file descriptor
An integer returned by the open system call which a process uses for subsequent references to the file. A file descriptor is unique for every process id/open system call combination.

frontplane
Sub-system of the AFI device adapter which directly interfaces with the external device.

handshake
A protocol for transferring data from one physical location to another. The handshake is a series of signals from both the source and destination to prevent data loss and coordinate data transfer over the frontplane data buses.

high true logic
Assertion of a signal forces the signal to a high (5 V) state and the deassertion of a signal forces the signal to a low (0 V) state.

interrupt signal
Informs the HP-UX operating system that a particular condition has occurred. The HP-UX operating system can chose to forward this signal (message) to the appropriate process for handling or the operating system can ignore the signal.

kernel
The executable modules for the HP-UX operating system.

lock
Ensures exclusive access to the device adapter.

logic sense
The signal value upon assertion and deassertion. Logic sense can be classified as high true logic or low true logic.

low true logic
Assertion of a signal forces the signal to a low (0 V) state and the deassertion of a signal forces the signal to a high (5 V) state.

major number
Identifies the entry point within the device driver. The device file contains the major number and can be specified when creating the device file.

midplane
Sub-system of the AFI device adapter which interfaces between the device adapter's frontplane and backplane.

minor number
Identifies the address of the external device on the CIB. The device file contains the minor number and can be specified when creating the device file.

process
A program that is recognized as an executable module. A process can be executing or temporarily suspended.

reset
Sets all hardware and software values to their default state or value.

single-ended driver
Sends data over one line. Single-ended configuration for the AFI device adapter is one line carrying data and the other connected to ground.

single-ended receiver
Receives data over one line. Single-ended configuration for the AFI device adapter is one line carrying data and the other connected to ground.

suspend
Temporarily stops the process from executing. A process can be suspended when denied access to a resource. The process will continue executing when it receives an awake signal from the HP-UX operating system. A suspended process waiting on a resource will receive an awake signal when the resource is released.

system call
Instructs the HP-UX operating system to perform a particular task.

Index

HP Part No. 27114-90003
Printed in USA September 1989

First Edition