

VOODOO AND DEVICE DRIVER PROGRAMMER'S REFERENCE MANUAL



"If you think the universe is big, you should see the source code." – Frank and Ernest

Scott Streit
San Diego State University
08/16/2000

Table Of Contents

I. DESCRIPTION	4
II. VOODOO DESIGN	4
III. VOODOO API.....	4
IV. SETTING THE SHUTTER POSITION FOR AN EXPOSURE.....	4
V. THE VOODOO C LIBRARY.....	4
VI. CONTROLLER CONFIGURATION BIT DEFINITIONS	6
VII. DEVICE DRIVER INSTALLATION.....	8
VIII. PCI BOARD - DEVICE DRIVER INTERACTION	8
1. Configuration Registers.....	8
Control/Status Registers.....	9
Host Interface Control Register (HCTR)	10
Host Interface Status Register (HSTR)	10
Host Command Vector Register (HCVR).....	11
Reply Buffer.....	11
DSP Manual Command Register.....	11
Board Destination Register	12
Command Argument Registers 1-5	12
DMA Host Kernel Buffer Address 1 Register.....	12
DMA Host Kernel Buffer Address 2 Register.....	13
3. Camera Table Registers	13
4. Image Buffers.....	13
5. Interrupts	13
VIII. DEVICE DRIVER USAGE.....	14
1. open()	14
2. close().....	14
3. read().....	15
4. ioctl().....	15
X. DSP COMMANDS	17
1. Test Data Link (TDL)	17
2. Read Memory (RDM)	18
3. Write Memory (WRM)	18
4. Power On (PON).....	18
5. Start Exposure (SEX).....	19
6. Readout Image (RDI).....	19
7. Abort Exposure (ABR)	19
8. Reset Controller (RST)	19
9. Open Shutter (OSH).....	19
10. Close Shutter (CSH).....	20
11. Resume Idle (IDL)	20
12. Stop Idle (STP)	20
13. Set Temperature	20
14. Read Temperature	21
15. Load Application (LDA).....	21
16. Load Timing or Utility File.....	21
17. Load PCI File.....	22
18. Dimensions.....	22
19. Set Gain and Speed (SGN).....	22
XI. DSP VECTOR COMMAND QUICK REFERENCE.....	23
XII. DSP AND DRIVER REPLY QUICK REFERENCE.....	24

XIII. SEQUENCE OF DSP COMMANDS	24
XIV. APPENDIX A.....	25
1. Controller Setup Sequence Pseudo Code	25
2. Exposure Sequence Pseudo Code	27
XV. EXAMPLE.....	30
XVI. REVISION HISTORY.....	30

I. DESCRIPTION

This document describes the commands an application would use to obtain an image from a device connected to the new PCI board. The device driver contains four functions and a number of commands useful for communicating with the connected device. The PCI board's digital signal processor (DSP) contains a group of commands for obtaining image data. Each DSP command is a sequence of device driver commands. The commands and their device driver sequences are described here.

II. VOODOO DESIGN

Voodoo has been designed using the Model-View-Controller (MVC) architecture. Use of this object oriented design method supports component-based software development and eases future maintenance. Each class falls into one of the following three groups:

- ★ The Model, which contains the data. These classes tend to be threads and provide communications to the hardware or perform some task.
- ★ The View is the windows, which takes user input and displays data output.
- ★ The Controller, which checks the data quality and transfers it between the View and the Model.

Voodoo's file names are designed to reflect the MVC design. Model class files are named xxCommand, View class files are named xxWindow/xxDialog/etc., and Controller class files are named xxListener, where the "xx" is a developer specified name. Task classes that are not associated with a window will not have names like xxCommand, but rather will have names that are descriptive of their task.

III. VOODOO API

Developers can find a browsable javadoc API description in the xx/Voodoo/Documents directory.

IV. SETTING THE SHUTTER POSITION FOR AN EXPOSURE

To set the shutter position for an exposure, read the current controller status from the timing board address X:0 using the READ_MEMORY vector command. To open the shutter during an exposure, set bit 11 of the status word to a 1. To close the shutter during an exposure, set bit 11 of the status word to a 0. The entire status word must then be written back to timing board address X:0 using the WRITE_MEMORY (0x8089) vector command.

V. THE VOODOO C LIBRARY

Currently, Java offers no support for low level device driver system calls. To solve this problem, Voodoo has a C library that contains functions for writing image data, manipulating image data, displaying image data, creating image buffers, and device driver communications. The library files are accessed by Voodoo through the Java Native Interface (JNI). The individual libraries are discussed here.

Libpcilibc.so

The PCI device driver library. This library contains functions to communicate with the PCI board.

Copen()	Opens a connection to the device driver.
Cclose()	Closes the connection to the device driver.
Cread_image()	Reads an image from the device.
Cioctl()	Sends an ioctl() command to the device driver.
Cread_reply()	Reads the current value of the device driver's reply buffer.

Libpcimemc.so

The memory library. This library contains functions to communicate with the PCI board.

Ccreate_memory()	Creates the image buffer for Voodoo.
Cfree_memory()	Destroys the image buffer for Voodoo.
Cdata_check()	Performs a data check on synthetic images.
Cswap_memory()	Byte swaps the specified memory location, the PCI board and SUN have different endians.
Cget_memory_word()	Returns the word located at the specified index.
Cprint_memory()	Prints the contents of the memory buffer.
Cfill_memory()	Fills the image buffer with test values.

Libdisplibc.so

The display library. This library contains functions to display an image on saimage or ximtool.

Cdisplay()	Displays an image using SAOimage or Ximtool (whichever is open). Note: this function contains Copen_display() and Cclose_display(), so do not use both.
Copen_display()	Opens a connection to saimage or ximtool. Not used.
Cclose_display()	Closes a connection to saimage or ximtool. Not used.
Libcdl.a	The Solaris CDL functions library.
Linux_libcdl.a	The Linux CDL functions library.

Libfitslibc.so

The FITS library. This library contains functions to write image data to a FITS file.

Cwrite_fits_data()	Writes an image to a FITS file.
--------------------	---------------------------------

Libsetupcmdlibc.so

The setup library. This library contains functions to perform any necessary array setup features.

Cdeinterlace()	Deinterlaces the array image according to the specified selection, which may be one of: 1) serial split, 2) parallel split, 3) CCD quad split, or 4) IR quad split.
----------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------

Libverlibc.so

The version library. This library contains a function to obtain the current date and time used for the “about” dialog boxes.

Cdate() Returns the current date and time.

VI. CONTROLLER CONFIGURATION BIT DEFINITIONS

Voodoo provides the user with the ability to set controller properties. The controller setup window may be displayed by clicking *Setup / Controller Setup* on Voodoo’s main window menu bar. The setup procedure follows the window components from top to bottom. A minimal setup of the image dimensions must be applied before an exposure can be started.

Part of the controller setup is a 24 bit word called the controller configuration. The bits of this word determine what hardware options are available. These options are ONLY available after a Timing board file or application has been applied to the controller. After such time, a tabbed window can be displayed with all the current available options. Subsequent Timing board downloads will only result in an update of the existing configuration window.

The controller configuration word is read by sending the vector command `READ_CONTROLLER_STATUS` (0x8079) to the PCI board. These bits are subject to change and can be found in DSPLib in the TIMHDR file. If no configuration word exists, the default values are used by Voodoo and are listed in the table below. The bit configurations for this word are as follows:

VII. DEVICE DRIVER INSTALLATION

Installation of Voodoo involves one of two methods. You may either manually install the files or use the automatic install script.

Solaris Installation

- ★ Create the directory (/xxx) where you want to keep the driver files.
- ★ Copy the driver tar file to the new directory and unpack it:

```
uncompress astropci_x_x_x.tar.Z  
tar xvf astropci_x_x_x.tar
```
- ★ Become superuser and run the install script:

```
./Install
```
- ★ Enter the PCI slot number when prompted.
- ★ To unload the driver, use the unix `modunload` command.

Linux Installation

- ★ Create the directory (/xxx) where you want to keep the driver files.
- ★ Copy the driver tar file to the new directory and unpack it:

```
uncompress astropci_x_x_x.tar.Z  
tar xvf astropci_x_x_x.tar
```
- ★ Become superuser and run the install script:

```
./astro_load
```

IMPORTANT NOTE: The Linux driver currently has no support for loading the driver at system startup. So the `./astro_load` script must be run after every system boot.

- ★ To unload the driver, type:

```
./astro_unload
```

VIII. PCI BOARD - DEVICE DRIVER INTERACTION

The device driver communicates with the PCI board through a series of registers located on the PCI board. The registers are "mapped" by the device driver to produce an equivalent set of virtual registers that the device driver can access as though it were communicating with the PCI board registers directly. The registers are broken down into three segments: the configuration registers, the control/status registers, and the timing table registers.

1. Configuration Registers

This is a set of 64 registers (DWORDS) used for configuration, initialization, and error handling for the PCI DSP. These registers are not manipulated by the user. There is, however, a driver supplied command that will read a pre-selected subset of these registers purely for identification and test purposes. The configuration space header is shown below.

Configuration Space Header				
ADDRESS	REGISTER			
0x0000	Device Id			Vendor Id
0x0004	Status			Command
0x0008	Class Code			Revision Id
0x000C	BIST	Header Type	Latency Timer	Cache Line Size
0x0010	Base Address Registers			
0x0014				
0x0018				
0x001C				
0x0020				
0x0024				
0x0028	Cardbus Pointer			
0x002C	Subsystem Id		Subsystem Vendor Id	
0x0030	Expansion ROM Base Address			
0x0034	Reserved			
0x0038	Reserved			
0x003C	Max_Lat	Min_Gnt	Interrupt Pin	Interrupt Line

Control/Status Registers

These 32-bit registers are used to send control commands and receive status replies to and from the PCI board. There are two command registers, the Host Command Vector Register (HCVR) and the Manual Command Register. The difference is in the command list and speed. The HCVR will only take the predefined list of commands specified in Section V, whereas the Manual Command register will take any three character ascii command that is supported by the DSP. Sending the predefined commands through the HCVR is faster since some or all of the parameters (board destination, arg#1, arg#2, etc.) may not change when sending the same command multiple times, while all parameters must be specified for every use of the Manual Command Register. The Manual Command Register is useful for developing new commands.

The complete list of registers is as follows:

Host/DSP Control and Status Registers	
ADDRESS	Register
0x0000	DSP Reserved
0x0004	DSP Reserved
0x0008	DSP Reserved
0x000C	DSP Reserved
0x0010	Host Interface Control Register (HCTR)
0x0014	Host Interface Status Register (HSTR)
0x0018	Host Command Vector Register (HCVR)
0x001C	Reply Buffer
0x0020	DSP Manual Command
0x0024	Board Destination
0x0028	Command Argument 1

0x002C	Command Argument 1
0x0030	Command Argument 1
0x0034	Command Argument 1
0x0038	Command Argument 1
0x003C	Unused
0x0040	DMA Kernel Buffer 1 Address
0x0044	DMA Kernel Buffer 2 Address

Host Interface Control Register (HCTR)

Only three bits of this register are used. Two are control bits to set the mode of the PCI board and the other is a flag indicating the progress of image data transfer to the user's application.

Bit 3 is used during image readout to help synchronize the DMA transfer. Bit 3 is set to 1 while the device driver is transferring image data from the kernel's internal buffers to the user's buffer. The bit is cleared to 0 while the device driver is waiting for image data. Bits 8 and 9 of this register are used to set the PCI board mode. If Bits 8 and 9 are set to 0x2, then the PCI board is put into slave mode. This must be done at the start of a PCI file download and must be completed before any other commands are sent to the PCI board. Bits 8 and 9 are cleared (0x0) to set the PCI board back to processing mode at the end of a PCI file download.

Summary:

Bit 3 = 1 Image buffer busy transferring to user space

0 Image buffer not transferring to user space

Bit 8 = 0 & Bit 9 = 1 PCI board set to slave mode for PCI file download

Bit 8 = 0 & Bit 9 = 0 PCI board set to normal processing

Host Interface Status Register (HSTR)

This register communicates status information about the PCI DSP. Four bits are used to determine if an interrupt is pending and one bit is used to determine if the PCI DSP fifo is available for input. The four interrupt bits are used to distinguish between reply and image buffer interrupts. A reply interrupt is generated when the DSP communicates a reply from a command back to the user application. An image interrupt is generated upon the completion of the DSP filling one of the device driver image buffers.

Summary:

Bit 0 = 1 Host transmit FIFO is empty, can send command

0 Host transmit FIFO is not empty, cannot send command

Bit 1 = 1 Host transmit FIFO is not full

0 Host transmit FIFO is full

Bit 3 = 1 Reply Interrupt Pending

0 No Reply Interrupt Pending

- Bit 4 = 1 Image Buffer 1 Interrrupt Pending
0 No Image Buffer 1 Interrupt Pending
- Bit 5 = 1 Image Buffer 2 Interrupt Pending
0 No Image Buffer 2 Interrupt Pending
- Bit 6 = 1 Interrupt pending
0 No interrupt pending

Host Command Vector Register (HCVR)

This register is used to send DSP vector commands to the PCI board. See [DSP VECTOR COMMAND QUICK REFERENCE](#) for a list of DSP commands that are sent through this register.

Reply Buffer

The reply buffer is used to receive replies from the PCI board and controller. When a command or controller property is written to one of the registers listed below, the PCI board responds by placing a reply in this buffer and generating an interrupt. The device driver claims the interrupt, reads the reply register, saves the value internally, and signals that a reply has been received. The device driver initializes the internal representation of the reply to -1. The command `ASTROPCI_GET_REPLY` returns the internally saved value if one exists. If not, the function waits for a signal from the interrupt handler that a reply has been received or a timeout occurs. In both cases the internal reply value is re-initialized to -1 before the function returns. Writing to any of the following registers produces a reply:

- Vector command (HCVR) (address 0x0018),
- Manual command (address 0x0020),
- Board destination (address 0x0024),
- Command argument 1 (address 0x0028),
- Command argument 2 (address 0x002C),
- Command argument 3 (address 0x0030),
- Command argument 4 (address 0x0034),
- Command argument 5 (address 0x0038),
- Exposure time register (address 0x0080).
- Utility options register (address 0x0084),
- Camera status register (address 0x0088),
- Number of image columns register (address 0x008C),
- Number of image rows register (address 0x0090),
- Number of columns binned register (address 0x0094),
- Number of rows binned register (address 0x0098).

DSP Manual Command Register

This register is used to send user defined or special commands to the PCI, Timing, or Utility boards. The command may be any three character ascii sequence that is reconized by the installed

DSP program. This register is useful for developing new commands. The following sequence is used to send a manual command:

1. Set the number of arg registers and the board destination using ioctl command ASTROPCI_SET_DESTINATION.
2. Set argument register 1 if necessary, using ioctl() command ASTROPCI_SET_ARG1.
3. Set argument register 2 if necessary, using ioctl() command ASTROPCI_SET_ARG2.
4. Set argument register 3 if necessary, using ioctl() command ASTROPCI_SET_ARG3.
5. Set argument register 4 if necessary, using ioctl() command ASTROPCI_SET_ARG4.
6. Set argument register 5 if necessary, using ioctl() command ASTROPCI_SET_ARG5.
7. Send the three character ascii command using ioctl() command ASTROPCI_SET_CMDR.

Board Destination Register

This register contains two separate pieces of information. The upper 16 bits contain the number of arguments associated with the current command. The lower 16 bits specify the board destination (PCI, Timing or Utility) for the current command. The board destination register should only be used with the following commands:

<u>Command</u>		<u>Valid Boards</u>	<u>Command Type</u>
LOAD_APPLICATION	0x807F	PCI, Timing	Vector (HCVR)
MANUAL_COMMAND	0x100	PCI, Timing, Utility	Ioctl()
TEST_DATA_LINK	0x8085	PCI, Timing, Utility	Vector (HCVR)
READ_MEMORY	0x8087	PCI, Timing, Utility	Vector (HCVR)
WRITE_MEMORY	0x8089	PCI, Timing, Utility	Vector (HCVR)

The board destinations are as follows:

PCI board	0x1
Timing board	0x2
Utility board	0x3

Command Argument Registers 1-5

These five registers are used to pass manual command parameters (addresses, byte counts, etc.) . The meaning of these registers depends on the command context.

DMA Host Kernel Buffer Address 1 Register

The device driver uses two kernel allocated buffers for intermediate image storage. The address of the first buffer is passed to the PCI board DSP through this register. This address is assigned each time a user's application opens a connection to the device driver. The address does not change until the device driver is re-opened.

DMA Host Kernel Buffer Address 2 Register

The device driver uses two kernel allocated buffers for intermediate image storage. The address of the second buffer is passed to the PCI board DSP through this register. This address is assigned each time a user's application opens a connection to the device driver. The address does not change until the device driver is re-opened.

3. Camera Table Registers

The camera registers are used to communicate camera parameters such as array size and exposure time to the PCI board. These parameters are written to a set of registers that act like a table. The ascii sequence 'DON' is returned upon writing any parameters to the table. No commands are required to begin processing of this table. The PCI board retrieves these values as needed. See the following table.

Camera Table	
ADDRESS	REGISTER
0x0080	Exposure Time
0x0084	Utility Options (1 = open shutter, 0 = close shutter)
0x0088	Camera Status
0x008C	Number Of Image Columns
0x0090	Number Of Image Rows
0x0094	Number Of Columns Binned
0x0098	Number Of Rows Binned

4. Image Buffers

The device driver allocates two image buffers in the open() entry point. The buffers are 131072 bytes each in size, which is fixed by the PCI DSP. The two buffers work together to efficiently transfer data from the controller to the user application. While one buffer is transferring its contents to the user application, the other is filling with data. The buffers go back and forth until all the data has been transferred.

5. Interrupts

Interrupts are used to communicate one of four occurrences between the device driver and PCI board. 1) The PCI board has a new reply, 2) data transfer to image buffer 1 has completed, 3) data transfer to image buffer 2 has completed, and 4) an abort readout command has been received by the PCI board.

A reply interrupt is acknowledged when bits 3 and 6 of the status register (HSTR) are high (1). The device driver claims the interrupt, saves the reply in an internal variable, and signals that a reply has been received.

An image buffer interrupt is acknowledged when bits 4 and 6 of the status register (HSTR) are high (1) for image buffer 1 and when bits 5 and 6 are high (1) for image buffer 2. The device driver claims the interrupt and signals that the appropriate buffer is full and ready to be transferred to user space.

The abort readout command causes the PCI board to generate an interrupt, which is claimed by the device driver only when bit 6 of the status register (HSTR) is high (1). The device driver then clears both image buffer interrupts to stop any data transfer and sets a flag that informs the read() function that the current readout has been aborted and all data transfer should cease.

VIII. DEVICE DRIVER USAGE

The device driver has four main entry points available for interaction with the camera. The four entry points are: open(), close(), read(), and ioctl(). To use these functions, the user must include the system file *fcntl.h*.

1. open()

Opens a connection to the requested device. This function must succeed before any further access to the device may occur. See the open(9E) man pages. Returns 0 for success or -1 for failure.

Usage:

```
int file descriptor = open(const char *device node, int mode)
```

device node

Is one of nodes */dev/astropci0* or */dev/astropci1*. These nodes are created during the driver installation process and correspond to pci board 1 and 2 (depending on the number of boards you have).

mode

Is the constant *O_RDWR* (*Open for reading and writing*) supplied by the system file *fcntl.h*.

file descriptor

Is an integer reference to the opened device.

2. close()

Closes a connection to the requested device. Returns 0 for success or -1 for failure.

Usage:

```
close(int file descriptor)
```

file descriptor

Is the integer returned from the open() instruction.

3. read()

Used to read an image from the camera. This instruction does not produce an image. To produce an image, a specified sequence of commands must proceed this instruction. This function merely reads the image data from the camera and passes it to the user's application. A double buffering scheme is used to facilitate the readout process. After an exposure completes, the PCI DSP writes image data to the first buffer. When the buffer is full an interrupt is generated and the DSP begins to fill the second buffer. Meanwhile, the interrupt from the first buffer being full is caught by the driver's interrupt handler and signals the read function to begin copying buffer 1 to the user application buffer. When the DSP completes writing to buffer 2, it generates an interrupt signaling the read function to copy buffer 2 to user space. Meanwhile, buffer 1 is again being filled. This process continues until all the image data has been copied to the user's application program. Note that the PCI DSP and the driver communicate during this process to prevent the buffers from being overwritten before they can be copied to the user application. The kernel buffer size is 131072 bytes (65536 pixels @ 16 bpp). This value is fixed by the PCI DSP and should NEVER be altered. Returns 0 for success or -1 for failure.

Usage:

```
read(int file descriptor, void *user buffer, int bytes)
```

file descriptor

Is the integer returned from the open() function.

user buffer

Is a pointer to a buffer where the user wants the image data to be stored as a result of this instruction.

bytes

Are the number of bytes to be read.

4. ioctl()

This is the "do all" instruction. Used to pass parameters, set controller states, receive controller status, and issue controller commands. Returns 0 for success or -1 for failure.

Usage:

```
ioctl(int file descriptor, int command, int *arg)
```

file descriptor

Is the integer returned from the open() function.

command

Is one of the commands described below.

arg

Is a variable used to send parameters and receive values associated with the execution of the specified command.

The command parameter may have one of the following values. Each value is an integer constant defined in the `astropci_io.h` file:

ASTROPCI_GET_HSTR (0x6)

Get the current value of the PCI DSP Host Status Register.

ASTROPCI_GET_HCTR (0x5)

Get the current value of the PCI DSP Host Control Register.

ASTROPCI_GET_REPLY(0x3)

Get the current reply. This command “itself” does not generate a reply, it only returns the reply from a previous command.

ASTROPCI_GET_CONFIG_INFO (0x314)

Returns the configuration space register values. No reply is generated by this command.

ASTROPCI_SET_HCTR (0x115)

Set the current value of the PCI DSP Host Control Register. No reply is generated by this command.

ASTROPCI_SET_HCVR (0x117)

Set the current value of the PCI DSP Host Control Vector Register. The reply value depends on the command sent. See [DSP VECTOR COMMAND QUICK REFERENCE](#)

ASTROPCI_SET_CMDR (0x100)

Set the current value of the manual command register. This command generates a ‘DON’ for a reply value.

ASTROPCI_SET_DESTINATION (0x111)

Set the current value of the board destination register. This command generates a ‘DON’ for a reply value.

ASTROPCI_SET_ARG1 (0x105)

Set the current value of the manual command argument 1 register. This command generates a ‘DON’ for a reply value.

ASTROPCI_SET_ARG2 (0x106)

Set the current value of the manual command argument 2 register. This command generates a ‘DON’ for a reply value.

ASTROPCI_SET_ARG3 (0x107)

Set the current value of the manual command argument 3 register. This command generates a 'DON' for a reply value.

ASTROPCI_SET_ARG4 (0x108)

Set the current value of the manual command argument 4 register. This command generates a 'DON' for a reply value.

ASTROPCI_SET_ARG5 (0x109)

Set the current value of the manual command argument 5 register. This command generates a 'DON' for a reply value.

ASTROPCI_SET_EXPTIME (0x119)

Set the current exposure time in the camera timing table. This command generates a 'DON' for a reply value.

ASTROPCI_SET_NCOLS (0x120)

Set the current number of array columns in the camera timing table. This command generates a 'DON' for a reply value.

ASTROPCI_SET_NROWS (0x121)

Set the current number of array rows in the camera timing table. This command generates a 'DON' for a reply value.

ASTROPCI_SET_IMAGE_BUFFERS (0x122)

Sets the address of the image data buffers. No reply value is generated by this command.

ASTROPCI_SET_UTIL_OPTIONS (0x123)

Supports future utility board options. This command generates a 'DON' for a reply value.

ASTROPCI_FLUSH_REPLY_BUFFER (0x124)

Clears the device reply buffer and sets the driver's internal reply value to REPLY_BUFFER_EMPTY. No reply value is generated by this command.

ASTROPCI_SET_BINNING_COLS (0x125)

Sets the image binning parameter in the columns direction. This command generates a 'DON' for a reply value.

ASTROPCI_SET_BINNING_ROWS (0x126)

Sets the image binning parameter in the rows direction. This command generates a 'DON' for a reply value.

ASTROPCI_ABORT_READ (0x302)

Aborts the current image readout. This command generates a 'DON' for a reply value.

X. DSP COMMANDS

This section describes pre-defined manual commands for the controller.

1. Test Data Link (TDL)

A user defined data value is sent to the specified board (timing, utility, pci) and returned as a reply. Used to test board communications.

Sequence Of Commands:

1. Write *data* to Arg1 register (*data* = any 24 bit hexadecimal value) using `ioctl()`.
2. Write *TEST_DATA_LINK* command (0x8085) to HCVR register using `ioctl()`.
3. Read reply buffer using `ioctl()`, should equal the data written to Arg1 register

2. Read Memory (RDM)

Used to read the contents of a DSP memory location from one of the controller boards (timing, utility, pci).

Sequence of Commands:

1. Write *type* to Arg1 register using `ioctl()`.
type = `'__R'` (ROM)
`'__X'` (DSP X memory space)
`'__Y'` (DSP Y memory space)
`'__P'` (DSP program memory space)
2. Write *address* to Arg2 register using `ioctl()`.
address = any hexadecimal address in the specified *type* range
3. Write *READ_MEMORY* command (0x8087) to the HCVR register using `ioctl()`.
4. Read reply buffer using `ioctl()`, which will contain the data value.

3. Write Memory (WRM)

Used to write to a DSP memory location on one of the controller boards (timing, utility, pci).

Sequence of Commands:

1. Write *type* to Arg1 register using `ioctl()`.
type = `'__R'` (ROM)
`'__X'` (DSP X memory space)
`'__Y'` (DSP Y memory space)
`'__P'` (DSP program memory space)
2. Write *address* to Arg2 register using `ioctl()`.
address = any hexadecimal address in the specified *type* range
3. Write *data* to Arg3 register using `ioctl()`. (*data* = any 24 bit hex value)
4. Write *WRITE_MEMORY* command (0x8089) to the HCVR register using `ioctl()`.
5. Read reply buffer using `ioctl()`, should equal ascii string 'DON'.

4. Power On (PON)

Used to turn the controller power on. This command must be done before any other commands are sent to the controller.

Sequence of Commands:

1. Write *POWER_ON command* (0x808D) to the HCVR register using `ioctl()`.
2. Read reply buffer using `ioctl()`, should equal ascii string 'DON'.

5. Start Exposure (SEX)

Used to start an exposure.

Sequence of Commands:

1. Write *START_EXPOSURE command* (0x809B) to the HCVR register using `ioctl()`.
2. Read reply buffer using `ioctl()`, should equal ascii string 'DON'.

6. Readout Image (RDI)

Used to readout the image data from the controller into the user's application.

Sequence of Commands:

1. Write *READ_IMAGE command* (0x809D) to the HCVR register using `ioctl()`.
2. Call `read()` command with the number of image data bytes to receive.
3. Read reply buffer using `ioctl()`, should equal ascii string 'DON'.

7. Abort Exposure (ABR)

Used to abort the current exposure. This command only aborts the exposure and not the readout.

Sequence of Commands:

1. Write *ABORT_EXPOSURE command* (0x80A1) to the HCVR register using `ioctl()`.
2. Read reply buffer using `ioctl()`, should equal ascii string 'DON'.

8. Reset Controller (RST)

Used to reset the controller.

Sequence of Commands:

1. Write *RESET_CONTROLLER command* (0x80A1) to the HCVR register using `ioctl()`.
2. Read reply buffer using `ioctl()`, should equal 'SYR' (system reset).

9. Open Shutter (OSH)

Opens the camera shutter.

Sequence of Commands:

1. Write `_OPEN_SHUTTER` command (0x80A9) to the HCVR register using `ioctl()` command `ASTROPCI_SET_UTIL_OPTIONS`.
2. Read reply buffer using `ioctl()`, should equal 'DON'.

10. Close Shutter (CSH)

Closes the camera shutter.

Sequence of Commands:

1. Write `_CLOSE_SHUTTER` command (0x80AB) to the HCVR register using `ioctl()` command `ASTROPCI_SET_UTIL_OPTIONS`.
2. Read reply buffer using `ioctl()`, should equal 'DON'.

11. Resume Idle (IDL)

Puts the controller in idle mode.

Sequence of Commands:

1. Write `RESUME_IDLE_MODE` command (0x8097) to the HCVR register using `ioctl()` command `ASTROPCI_SET_HCVR`.
2. Read reply buffer using `ioctl()`, should equal 'DON'.

12. Stop Idle (STP)

Takes the controller out of idle mode.

Sequence of Commands:

1. Write `STOP_IDLE_MODE` command (0x8095) to the HCVR register using `ioctl()` command `ASTROPCI_SET_HCVR`.
2. Read reply buffer using `ioctl()`, should equal 'DON'.

13. Set Temperature

Sets the array target temperature.

Sequence of Commands:

1. Write the desired temperature value to the ARG1 register using `ioctl()`.
2. Write `SET_ARRAY_TEMPERATURE` command (0x80AD) to the HCVR register using `ioctl()` command `ASTROPCI_SET_HCVR`.
3. Read reply buffer using `ioctl()`, should equal 'DON'.

14. Read Temperature

Reads the current array temperature.

Sequence of Commands:

1. Write READ_ARRAY_TEMPERATURE command (0x80AF) to the HCVR register using ioctl() command ASTROPCI_SET_HCVR.
2. Read reply buffer using ioctl(), should equal 'DON'.

15. Load Application (LDA)

Loads a ROM application. The timing and utility boards have up to four user defined applications stored in their ROM's. These may be used in place of a file download.

Sequence of Commands:

1. Set the board destination (timing or utility) using ioctl() command ASTROPCI_SET_DESTINATION.
2. Write the desired application number to the arg1 register using ioctl().
3. Write LOAD_APPLICATION command (0x807F) to the HCVR register using ioctl() command ASTROPCI_SET_HCVR.
4. Read reply buffer using ioctl(), should equal 'DON'.

16. Load Timing or Utility File

Load a timing or utility file. This is not a single command, but rather a sequence of write memory commands.

Sequence of Commands:

1. Open the file.
2. Search the file for the TIMBOOT or UTILBOOT keywords. If found, set the board destination (timing or utility) using ioctl() command ASTROPCI_SET_DESTINATION. Else, exit with an unknown file.
3. Search file for keyword "_DATA". If found, extract address and memory type (X, Y, P, R, D). If address less than 0x4000, continue. Else resume search.
4. While next value is not a "_", perform *Write Memory (WRM)* command. Increment the address and check that a 'DON' was received for a reply. If not, then report an error and stop download.
5. Go to 4 and repeat until all the data in the current block has been written.
6. Go to 3 and repeat for the rest of the file.
7. Close the file.

17. Load PCI File

Load a PCI boot file. This is not a single command, but rather a sequence of write memory commands.

Sequence of Commands:

1. Set the PCI board to "slave mode" by clearing the HTF bits 8 and 9 (Host Transfer Flags) and bit 3 of the host control register (HCTR) by using the ioctl() command ASTROPCI_SET_HCTR.
2. Write the PCI_DOWNLOAD (0x808B) command to the host vector command register (HCVR) using the ioctl command ASTROPCI_SET_HCVR.
3. Write the magic value 0x555AAA to the arg1 register using the ioctl() command ASTROPCI_SET_ARG1.
4. Open the file.
5. Look for the start of data "_DATA P".
6. If found, get the next line which contains the number of words to transfer and the starting address.
7. Write the number of words to the arg1 register using the ioctl() command ASTROPCI_SET_ARG1.
8. Write the address to the arg1 register using the ioctl() command ASTROPCI_SET_ARG1.
9. Throw away the next line and while the current word count is less than the total word count (from 7) minus 2, read in a data value and write it to the arg1 register using the ioctl() command ASTROPCI_SET_ARG1. Before writing to the arg1 register, check for an intermixed "_DATA" tag. If found throw out and continue.
10. Set bit 3 of the host control register (HCTR) by using the ioctl() command ASTROPCI_SET_HCTR.
11. Read reply buffer using ioctl(), should equal 'DON'.

18. Dimensions

Sets the array dimensions.

Sequence of Commands:

1. Write the number of array columns to the camera timing table using the ioctl() command ASTROPCI_SET_NCOLS.
2. Read reply buffer using ioctl(), should equal 'DON'.
3. Write the number of array rows to the camera timing table using the ioctl() command ASTROPCI_SET_NROWS.
4. Read reply buffer using ioctl(), should equal 'DON'.

19. Set Gain and Speed (SGN)

Sets the controller gain and speed. The gain refers to the output amplifier and speed refers to the data conversion rate.

Sequence of Commands:

1. Write the desired gain value to the arg1 register using the ioctl() command ASTROPCI_SET_ARG1.
2. Write the desired speed value to the arg2 register using the ioctl() command ASTROPCI_SET_ARG2.
3. Write the ascii string 'SGN' to the command register using the ioctl() command ASTROPCI_SET_HCVR.
4. Read reply buffer using ioctl(), should equal 'DON'.

XI. DSP VECTOR COMMAND QUICK REFERENCE

The full list of DSP vector commands and expected replies is given in the following table.

<u>Command</u>		<u>Expected Reply</u>
READ_CONTROLLER_STATUS	0x8079	Controller Config Word
WRITE_CONTROLLER_STATUS	0x807B	'DON'
RESET_CONTROLLER	0x807D	'SYR'
LOAD_APPLICATION	0x807F	'DON'
PCI_PC_RESET	0x8081	'DON'
READ_PCI_STATUS	0x8083	'DON'
TEST_DATA_LINK	0x8085	Data Value Sent
READ_MEMORY	0x8087	Data Value At Mem Location
WRITE_MEMORY	0x8089	'DON'
RESERVED_1	0x808B	n/a
POWER_ON	0x808D	'DON'
POWER_OFF	0x808F	'DON'
SET_BIAS_VOLTAGES	0x8091	'DON'
CLEAR_ARRAY	0x8093	'DON'
STOP_IDLE_MODE	0x8095	'DON'
RESUME_IDLE_MODE	0x8097	'DON'
READ_EXPOSURE_TIME	0x8099	Current Elapsed Time
START_EXPOSURE	0x809B	'DON'
READ_IMAGE	0x809D	'DON'
ABORT_READOUT	0x809F	'DON'
ABORT_EXPOSURE	0x80A1	'DON'
PAUSE_EXPOSURE	0x80A3	'DON'
RESUME_EXPOSURE	0x80A5	'DON'
RESERVED_2	0x80A7	n/a

OPEN_SHUTTER	0x80A9	'DON'
CLOSE_SHUTTER	0x80AB	'DON'
SET_ARRAY_TEMPERATURE	0x80AD	'DON'
READ_ARRAY_TEMPERATURE	0x80AF	Current Array Temperature
PCI_DOWNLOAD	0x802F	None

XII. DSP AND DRIVER REPLY QUICK REFERENCE

The full list of DSP and driver replies are given in the following table.

<u>Ascii Reply</u>	<u>Hex Equivalent</u>	<u>Description</u>
DON	0x00444F4E	Done
ERR	0x00455252	Error
SYR	0x00535952	System Reset
TOUT	0x544F5554	Timeout
NO REPLY	0xFFFFFFFF	Reply Buffer Empty

XIII. SEQUENCE OF DSP COMMANDS

To use the functions and commands described in this document, the following sequence would typically take place.

1. Open a device driver connection using *open()*.
2. Perform any necessary setup commands using *ioctl()*.
 - Load PCI File
 - Reset Controller
 - Load Application or File
 - Power On
 - Set Gain
 - Set Idle
 - Set Array Temperature
 - Set Array Dimensions
3. Take an exposure using *ioctl()* and *read()*.
 - Set Exposure Time
 - Start Exposure
 - Readout Image Data
4. Save the image.
5. Close the device driver connection using *close()*.

XIV. APPENDIX A

1. Controller Setup Sequence Pseudo Code

```
// Check for PCI download.
If (do PCI download) then
{
    If (PCI filename does not exist) then
        Print error message and continue.
    Else
        Call function to perform PCI file download.
}

If (reply not equal to 'DON' or if TIMEOUT) then
    Print error message and continue.

If (do reset controller) then
{
    Reset the controller using the ioctl() command ASTROPCI_SET_HCVR with an argument value of
    0x807D (RESET_CONTROLLER).

    If (reply not equal to 'SYR' or if TIMEOUT) then
        Print error message and continue.
}

If (do hardware test) then
{
    If (do PCI hardware test) then
    {
        Calculate data increment: data_incr = MAX_TEST_VALUE/NUM_OF_PCI_TESTS

        Loop over the NUM_OF_PCI_TESTS
        {
            Set the board destination to the PCI board using the ioctl() command
            ASTROPCI_SET_DESTINATION and argument value of 0x10000001 (# of args << 16 |
            board id).

            Set the argument 1 register using the ioctl() command ASTROPCI_SET_ARG1 with
            the current data value as the argument.

            Use the ioctl() command ASTROPCI_SET_HCVR with an argument value of 0x8085
            (TEST_DATA_LINK).

            If (reply not equal to sent data value or if TIMEOUT) then
                Print error message and continue.

            Increment data value: data = data + data_incr.
        }
    }

    If (do timing hardware test) then
    {
        Calculate data increment: data_incr = MAX_TEST_VALUE/NUM_OF_TIM_TESTS

        Loop over the NUM_OF_TIM_TESTS
        {
            Set the board destination to the timing board using the ioctl() command
            ASTROPCI_SET_DESTINATION and argument value of 0x10000002 (# of args << 16 |
            board id).

            Set the argument 1 register using the ioctl() command ASTROPCI_SET_ARG1 with
            the current data value as the argument.

            Use the ioctl() command ASTROPCI_SET_HCVR with an argument value of 0x8085
            (TEST_DATA_LINK).

            If (reply not equal to sent data value or if TIMEOUT) then
                Print error message and continue.

            Increment data value: data = data + data_incr.
        }
    }

    If (do utility hardware test) then
    {
        Calculate data increment: data_incr = MAX_TEST_VALUE/NUM_OF_UTIL_TESTS
```

```

Loop over the NUM_OF_UTIL_TESTS
{
    Set the board destination to the utility board using the ioctl() command
    ASTROPCI_SET_DESTINATION and argument value of 0x10000003 (# of args << 16 |
    board id).

    Set the argument 1 register using the ioctl() command ASTROPCI_SET_ARG1 with
    the current data value as the argument.

    Use the ioctl() command ASTROPCI_SET_HCVR with an argument value of 0x8085
    (TEST_DATA_LINK).

    If (reply not equal to sent data value or if TIMEOUT) then
        Print error message and continue.

    Increment data value: data = data + data_incr.
}
}

If (do timing file load) then
{
    If (file does not exist) then
        Print error message and continue.
    Else
    {
        Call function to load timing DSP file.

        If (timing file load failed or if TIMEOUT) then
            Print error message and continue.
    }
}

If (do timing application) then
{
    If (application number not between 0 and 3) then
        Print error message and continue.
    Else
    {
        Set the board destination to the timing board using the ioctl() command
        ASTROPCI_SET_DESTINATION and argument value of 0x10000002 (# of args << 16 | board
        id).

        Set the argument 1 register using the ioctl() command ASTROPCI_SET_ARG1 with the
        application number as the argument.

        Use the ioctl() command ASTROPCI_SET_HCVR with an argument value of 0x807F
        (LOAD_APPLICATION).

        If (reply not equal to 'DON' or if TIMEOUT) then
            Print error message and continue.
    }
}

If (do utility file load) then
{
    If (file does not exist) then
        Print error message and continue.
    Else
    {
        Call function to load utility DSP file.

        If (utility file load failed or if TIMEOUT) then
            Print error message and continue.
    }
}

If (do utility application) then
{
    If (application number not between 0 and 3) then
        Print error message and continue.
    Else
    {
        Set the board destination to the utility board using the ioctl() command
        ASTROPCI_SET_DESTINATION and argument value of 0x10000003 (# of args << 16 | board
        id).

        Set the argument 1 register using the ioctl() command ASTROPCI_SET_ARG1 with the
        application number as the argument.
    }
}

```

```

        Use the ioctl() command ASTROPCI_SET_HCVR with an argument value of 0x807F
        (LOAD_APPLICATION).

        If (reply not equal to 'DON' or if TIMEOUT) then
            Print error message and continue.
    }
}

If (do power on) then
{
    Reset the controller using the ioctl() command ASTROPCI_SET_HCVR with an argument value of
    0x808D (POWER_ON).

    If (reply not equal to 'DON' or if TIMEOUT) then
        Print error message and continue.
}

If (do set temperature) then
{
    Set the argument 1 register using the ioctl() command ASTROPCI_SET_ARG1 with the target
    temperature as the argument.

    Use the ioctl() command ASTROPCI_SET_HCVR with an argument value of 0x80AD
    (SET_ARRAY_TEMPERATURE).

    If (reply not equal to 'DON' or if TIMEOUT) then
        Print error message and continue.
}

If (do idle) then
{
    If (set to idle) then
        Use the ioctl() command ASTROPCI_SET_HCVR with an argument value of 0x8097
        (RESUME_IDLE_MODE).
    Else
        Use the ioctl() command ASTROPCI_SET_HCVR with an argument value of 0x8095
        (STOP_IDLE_MODE).

    If (reply not equal to 'DON') then
        Print error message and continue.
}

If (do set dimensions) then
{
    // Set the number of columns.
    Use the ioctl() command ASTROPCI_SET_NCOLS with the argument value set to the number of image
    columns.

    If (reply not equal to 'DON' or if TIMEOUT) then
        Print error message and continue.

    // Set the number of rows.
    Use the ioctl() command ASTROPCI_SET_NROWS with the argument value set to the number of image
    rows.

    If (reply not equal to 'DON' or if TIMEOUT) then
        Print error message and continue.

    Set a variable that lets the exposure function know that the minimum controller setup has
    been applied.

    // Setup the controller configuration.
    If (do timing (app or file) OR did timing (app or file)) then
    {
        Get the controller configuration status word using the ioctl() command
        ASTROPCI_SET_HCVR with an argument value of 0x8079 (READ_CONTROLLER_STATUS).

        If (reply equals 'ERR') then
            Set controller configuration status word to default value of 0x020000
            (DEFAULT_CONFIG_WORD).

        Call the function with the controller configuration status word to create the
        controller configuration window.
    }
}
}

```

2. Exposure Sequence Pseudo Code

```

// Set the shutter position. Do this by reading the current controller status from
// the timing board address X:0. If "open shutter", bit 11 of the status is set to
// a 1, otherwise, bit 11 is cleared (set to a 0). The entire status word is then
// written back to the timing board address X:0.
If (current shutter position != previous position) then
{
    Set current shutter position equal to new shutter position.

    Set the board destination to the timing board using the ioctl() command
    ASTROPCCI_SET_DESTINATION and argument value of 0x00020002 (# of args << 16 | board id).

    If (reply not equal to 'DON') then
        Print error message and return.

    Set the memory type to X space by using the ioctl() command ASTROPCCI_SET_ARG1 with an
    argument value of '__X'.

    Read the current controller status from address 0 by using the ioctl() command
    ASTROPCCI_SET_HCVR and argument 0x8087 (READ_MEMORY).

    Read the reply (ASTROPCCI_GET_REPLY), which will be the current controller status.

    Set the board destination to the timing board using the ioctl() command
    ASTROPCCI_SET_DESTINATION and argument value of 0x00030002 (# of args << 16 | board id).

    If (reply not equal to 'DON') then
        Print error message and return.

    Set the memory type to X space by using the ioctl() command ASTROPCCI_SET_ARG1 with an
    argument value of '__X'.

    If (do open shutter) then
        Write the current controller status OR'd with (1 << 11) to address 0 by using the
        ioctl() command ASTROPCCI_SET_HCVR and argument 0x8089 (WRITE_MEMORY).
    Else
        Write the current controller status OR'd with ~(1 << 11) to address 0 by using the
        ioctl() command ASTROPCCI_SET_HCVR and argument 0x8089 (WRITE_MEMORY).

    If (reply not equal to 'DON') then
        Print error message and return.
}

// Set the exposure time.
Set the board destination to the utility board using the ioctl() command ASTROPCCI_SET_DESTINATION and
argument value of 0x00010003 (# of args << 16 | board id).

If (reply not equal to 'DON') then
    Print error message and return.

Set the exposure time using the ioctl() command ASTROPCCI_SET_EXPTIME with the exposure time as the
argument.

// Get the image byte size.
Calculate image byte count from setup info.

// Check for multiple exposures.
If (do multiple exposures) then
    Set number of exposures.
Else
    Set number of exposure to 1.

// Start executing the exposures.
Loop over the number of exposures
{
    Display the elapsed time as 0.

    If (delay before starting the exposure) then
        sleep for delay*1000 seconds.
    Else continue.

    Start the exposure using the ioctl() command ASTROPCCI_SET_HCVR and argument 0x809B
    (START_EXPOSURE).

    If (reply not equal to 'DON') then
        print error message and return.

    If (exposure_time > 5 seconds) then
    {
        if (using controller exposure time counter) then

```

```

        Read the elapsed exposure time using the ioctl() command
        ASTROPCCI_SET_HCVR and argument 0x8099 (READ_EXPOSURE_TIME).

// The controllers start value may be initially less than 0.
While (elapsed_exposure_time < 0)
    Read the elapsed exposure time, as above, until the elapsed time equals 0.

Set the elapsed time to 0 and display the elapsed time.

While (elapsed_exposure_time < (exposure_time-5))
{
    Sleep for 0.5 seconds.

    While (pause equals true)
        Sleep for 0.25 seconds

    If (change exposure time equals true) then
    {
        Set the board destination to the utility board using the ioctl()
        command ASTROPCCI_SET_DESTINATION and argument value of 0x10000003 (# of
        args << 16 | board id).

        // Check for zero exposure time (and times less than 5
        // seconds) and set exposure time to 5100 ms if true. This // is to
        allow the PCI board to take control for the
        // remaining 5 seconds. The 5100 ms is required because the // PCI DSP
        expects the READ_IMAGE (0x809D) command to be
        // issued, but the PCI DSP does not accept commands when the // elapsed
        exposure time is 5 seconds or less.
        If (exposure_time < 5 seconds) then
            Exposure_time = 5100 ms.

        Set the new exposure time using the ioctl() command
        ASTROPCCI_SET_EXPTIME with the exposure time as the argument.

        Display the new elapsed exposure time.

        // Check for exposure time set to zero and break immediately // if it
        is.
        If (exposure_time <= 5 seconds) then
            Break.
    }

    If (using controller exposure time counter) then
        Read the elapsed exposure time using the ioctl() command
        ASTROPCCI_SET_HCVR and argument 0x8099 (READ_EXPOSURE_TIME).

    Else elapsed_time = elapsed_time + 0.5 seconds.

    Display the elapsed time.
}

Display a 5 for the elapsed time.
}

Print a message informing the user that the exposure has complete.

// Readout the image.
Set the board destination to the PCI board using the ioctl() command ASTROPCCI_SET_DESTINATION
and argument value of 0x1 (# of args << 16 | board id).

// This does not need to be done for exposure times greater than 5 seconds because // it is
built into the controller for times less than 5 seconds.
If (exposure_time > 5 seconds) then
    Start the data flow from the controller using the ioctl() command ASTROPCCI_SET_HCVR
    and argument 0x809D (READ_IMAGE).

Start reading image data using read() and the number of image bytes as an argument.

If (reply not equal to 'DON') then
    Print and error message and return.

If (beep after readout) then
    Ring the system bell.

// The SUN swaps the image data, so unswap it.
Call Cswap_memory() function in the libpcimemc.so library.

If (the image needs to be de-interlaced) then

```

```

        Call Cdeinterlace() function in the libsetupcmdlibc.so library.
    If (the image needs to be displayed) then
        Call Cdisplay() function in the libdisplibc.so library.

    If (cannot display) then
        Print error message and continue.

    If (the image needs to be saved) then
    {
        If (auto-increment the filename equals true) then
            Increment the filename.

        Write the FITS file.
    }

    If (image is synthetic and want to check data) then
        Call Cdata_check() function in the libpcimemc.so library.
}

```

XV. EXAMPLE

For a complete example of using the C DSPLibrary, please see the file “mreadout_pci.c” supplied with the C DSPLibrary.

XVI. REVISION HISTORY

<u>DATE</u>	<u>AUTHOR</u>	<u>CHANGE</u>
11/22/1999	Scott Streit	Initial
01/11/2000	Scott Streit	Updated example, original had bugs.
02/08/2000	Scott Streit	Added Section II subsections 4 and 5, added Section VI, and misc. minor changes.
03/02/2000	Scott Streit	Added bit 0 info for HSTR, added info for downloading timing, utility, and PCI files under section IV. Updated reply buffer info in section II.
06/27/2000	Scott Streit	Converted document to MS Word.
07/25/2000	Scott Streit	Added reply values to ioctl() and vector command lists. Updated board destination section.
08/16/2000	Scott Streit	Merged driver and voodoo docs into one programmers manual.