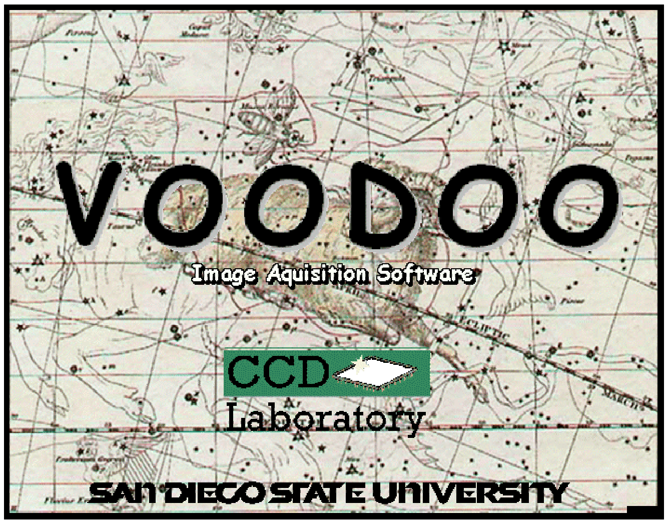
VOODOO AND DEVICE DRIVER PROGRAMMER'S REFERENCE MANUAL



…. 1101010100100010 ….

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

Scott Streit
Astronomical Research Cameras, Inc

02/26/2003

Table Of Contents

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 SOURCE DOCUMENTATION

Developers can find browsable javadoc documentation in the xx/Voodoo/Documents directory.

III. DEVICE DRIVER INSTALLATION

Solaris Installation

- ✯ Unpack the device driver files, this will create a directory called astropciV1.7 (or something similar):

```
gunzip astropci_solaris.tar.gz
tar xvf astropci_solaris.tar
```

- ✯ Become superuser and run the install script:

```
./Install
```

- ✯ Enter the PCI slot number when prompted. For multiple boards, continue to enter PCI slot numbers as prompted. Hit “return” when finished.

- ✯ To remove the driver, use the unix command: `rem_drv astropci`

Linux Installation

- ✯ This driver has been tested under Redhat 7.2 (kernel 2.4.2-2) and Redhat 9.0 (kernel 2.4.20-8). It WILL NOT work with older kernels.

- ✯ In order for this driver to function, the following line must be set in LILO (/etc/lilo.conf): append="mem=xxxM", where xxx is the amount of RAM you DO NOT want to use for an image buffer. So, if your computer has 128Mb of RAM and you want to have a 28Mb image buffer, you must have the following line in LILO (/etc/lilo.conf): append="mem=100M".

```
Example:
-------
1. Become superuser/root.

2. Edit /etc/lilo.conf to have the append="mem=xxxM" line:

     image=/boot/vmlinuz-2.4.2-2
     label=linux
     read-only
     root=/dev/hda5
     append="mem=100M"

3. From a tcsh, execute the command:  lilo -v

4. Reboot the computer.
```

✯ Create the directory (/xxx) where you want to keep the driver files.

✯ Copy the driver tar file to the new directory and unpack it:

```
gunzip astropci_x_x_x.tar.gz
tar xvf astropci_x_x_x.tar
```

✯ Become superuser and run the install script:

```
./astropci_load
```

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

✯ To unload the driver, type:

```
./astropci_unload
```

Windows 2000 Installation

✯ Unzip the device driver files.

IMPORTANT: If you are re-installing or upgrading the driver you must first delete the windows cached .inf and .pnf files. During installation, Windows stores its own copy of the driver, which it will use the next time you try to re-install. To prevent Windows from using this old copy you must delete it before re-installing or upgrading. The files are generally called oem0.inf, oem1.inf, … and oem0.pnf, oem1.pnf, … And can be found in C:\Windows\inf\. Be sure to only delete the oem# files that contain references to Astronomical Research Cameras, Inc.

✯ The Win2k driver supports Plug-N-Play. So there are two methods for installation.

1. Install the PCI board and startup the computer, the new board should be detected and you will be prompted for the driver installation.

2. From control panel, choose “Add New Hardware”, again you will be prompted for the driver installation.

3. Reboot.

✯ NOTE: The driver install will modify the Windows registry to support a large image buffer. This MUST be done. The default amount of memory to set aside is 32M (0x2000000). If you want more memory, you can modify it by doing the following: WARNING: Incorrectly modifying registry values can cause your machine to stop working.

1. Run *regedit* from *Start->Run*.

2. In the registry, modify HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\Memory Management\NonPagedPoolSize to be a value larger than the buffer you want to use. DO NOT make the value more than 80% of your total RAM. For example, to set aside 32M of nonpaged memory, set the value to 0x2000000.

3. Reboot for changes to take affect.

IV. VOODOO INSTALLATION

Solaris and Linux Installation

✯ Create a temporary directory and unpack the Voodoo files (do not untar the voodoofiles.tar file):

```
gunzip voodoo.tar.gz
tar xvf voodoo.tar
```

✯ Become superuser and run the install script:

```
./Install
```

✯ Enter the directory where you want Voodoo installed. The default is `/opt/voodoo`.

✯ Modify your .cshrc to contain the following, where $install_dir is the directory where Voodoo is installed and $system is either "unix" or "linux":

```
setenv  LD_LIBRARY_PATH $install_dir/Voodoo/Clib/$system/lib
setenv CLASSPATH $install_dir/Voodoo/classes:$install_dir/packages
```

✯ To un-install Voodoo, just delete the directory where Voodoo was installed. The default is `/opt/voodoo`.

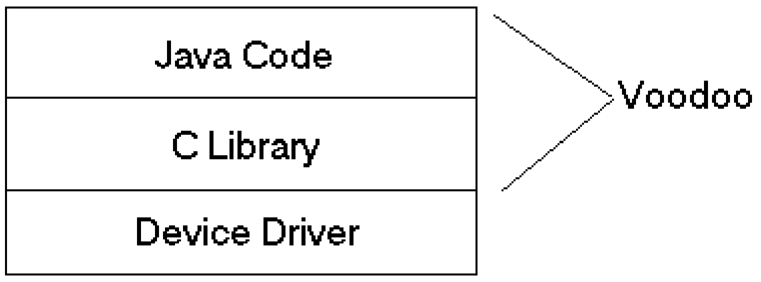
Windows 2000 Installation

✯ Create a temporary directory and unzip the Voodoo files.

✯ Run the file "*Setup.exe*" in the "*disk1*" directory.

V. APPLICATION STRUCTURE

Voodoo consists of two layers, the JAVA layer and the C library layer. Voodoo's C library supports device driver level system calls (because JAVA doesn't) and any other operations which may be difficult or too slow with JAVA (for example, memory mapping of the image buffer). While JAVA is platform independent, the C system calls are not. So the C library must be compiled separately on the supported operating systems. Likewise, the device driver is also system dependent. While the underlying system calls and structure are different for the device drivers on the UNIX, LINUX, and Windows 2000 operating systems, the interface between Voodoo and the device drivers is identical across all systems.



VI. IMAGE BUFFER

The image buffer is created by the device driver and is managed differently on the Solaris, Linux, and Windows 2000 systems. On Solaris machines, the image buffer is not created until a user application calls the memory mapping function, mmap(), at which time the device driver creates the image buffer with the size specified by the user application. The image buffer start address is then passed by the device driver to the PCI board using the INITIALIZE_IMAGE_ADDRESS (0x91) vector command. The image address is passed in two segments. The lower 16 bits of the image buffer address are sent first, followed by the upper 16 bits.

On Linux machines, the image buffer is created from a sequential segment of memory that is set aside during system boot by modifying /etc/lilo.conf, as specified in the Linux device driver installation instructions. The image buffer is created and mapped when a user application calls the memory mapping function mmap(). The image address is passed to the PCI board in the exactly the same way it is for the Solaris system. It is important to note that although the user application specifies how much memory to map for the image buffer in the mmap() function, it is ultimately the amount of memory that resides in the sequential memory, segment set aside at boot time, that determines the maximum size of the image buffer. The image buffer size must be less than the memory segment set aside using /etc/lilo.conf.

On Windows 2000 machines, the image buffer is created during the device driver installation. The user application then merely maps the pre-created image buffer using the ASTROPCI_MMAP DeviceIoControl() command. During driver installation the registry key "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\Memory Management\NonPagedPoolSize" is modified to be equal or larger than the image buffer size you want to use (the default is 34MB). Windows has a OS defined upper limit on the maximum size of memory that can be allocated. For more information see the document: "*Windows 2000/XP Device Driver DMA Limits*". Again, the image address is passed to the PCI board in the exactly the same way it is for the Solaris system.

VII. CAMERA CONTROL

The camera is controlled through a series of commands (see "*Command Description*" document) that are passed by a user application, such as Voodoo, to the device driver, which in turn sends the commands to the hardware (PCI board). All commands are passed from the user application to the device driver using the ioctl (DeviceIoControl on Win2K) driver function. The driver then writes all necessary command information to a mapped memory location on the PCI board (see the PCI BOARD – DEVICE DRIVER INTERACTION section). The PCI board then forwards any necessary commands to the controller. Every command returns a reply of one kind or another. A reply can be a data value or one of the following: timeout, done, error, system reset, readout, or busy. After sending a command, the device driver waits for one of the replies by polling a register (HSTR) on the PCI board. A timeout is incurred if no reply occurs after X number of polls. Before a command can be sent or a reply read, however, a bit in the host status register (HSTR) on the PCI board must be checked. This bit determines whether the PCI boards input and output FIFOs are empty or full respectively. This FIFO bit checking is automatically performed by the device driver.

Exposure Command Sequence

The following is a top-down sequence representing the interaction between the host computer (user application), PCI board, and the camera controller.

Host Computer		Controller
Send a "Start Exposure" command (SEX) to the controller.		

	Receive the “Start Exposure” and send it along to the controller.	
		Receive the “Start Exposure” command (IIA). Send a command to the PCI board to initialize its pixel counter for a new image.
	Set PCIADDR = BASEADDR and NPIXELS = 0. Reply (DON) to the host computer that the exposure has started.	
Receive a done (DON) reply that the exposure has started. Continuously interrogate the PCI board status and pixel counter. If the PCI board is not in readout and the exposure time is > 1 second, then also continuously read the elapsed exposure time from the controller. Sleep for 25ms after each interrogation. Continue on when the pixel counter on the PCI board reaches it’s target value or when an interrupt is received.		Clear the array and stop the clocks. Open the shutter if needed. Start the timer countdown for the exposure. When the timer counts down, send a “Read Array” command (RDA) to the PCI board with the dimensions of the image.
	Receive the “Read Array” command, set up to read the indicated number of pixels. Write image data over the computer bus as its received and increment the pixel counter as pixels are transferred to the host computer.	Close the shutter if needed, and delay for it to be closed. Calculate readout parameters (split serial or parallel, binning, subarray). Skip over unwanted rows (if in subarray mode). Clear out the serial shift register. Parallel shift one or more rows (if parallel binning). Skip over unwanted columns (if in subarray mode). Read desired number of pixels and transmit them to the PCI board. Skip over unwanted columns (if in subarray mode). Read desired number of bias pixels and transmit them to the PCI board. Loop back to the parallel shifting until done.
	Interrupt when done if desired.	
Continue on. Deinterlace and save the image data.		

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 two segments: the configuration registers, and control/status 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. 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	Subsytem 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 Command Data Register. The HCVR is used to send vector commands, whereas the Command Data register will take any of the ascii commands supported by the DSP. The Command Data register is also used to send vector command arguments to the PCI board and for setting the image buffer address.

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	Command Data

Host Interface Control Register (HCTR)

Only four bits of this register are used. Bits 8, 9, 11 and 12 are used to set the PCI bus data size for commands and replies. Setting bit 8 to 1 and bit 9 to 0 converts 32-bit PCI commands into 24-bit DSP data. So the most significant byte (MSB) of the 32-bit word is lost. Setting bit 11 to 1 and bit 12 to 0 converts 24-bit DSP reply data into 32-bit PCI data. So the most significant byte (MSB) is filled with zeros.

Host Interface Status Register (HSTR)

This register communicates status information about the current state of the system. Three bits are used for reply values and two bits are used to determine if the PCI DSP fifo is available for input or output.

Summary:

Bit 1 = 1 Host input FIFO is empty, can send command
0 Host input FIFO is not empty, cannot send command

Bit 2 = 1 DSP output FIFO is not empty, can read data
0 DSP output FIFO is empty, no data

Bits 3, 4, 5

= 0	TIMEOUT	(Timeout)
= 1	DON	(Done)
= 2	RDR	(Read Reply Value)
= 3	ERR	(Error)
= 4	SYR	(System Reset)
= 5	READOUT	(Readout)
= 6	BUSY	(PCI Busy)

VIIII. VECTOR COMMANDS

Vector commands are those that must be executed immediately. These commands interrupt the DSP processor and are serviced immediately. The command values, arguments and descriptions are listed in the table below. Depending on the command, there may be optional data values. The data values are sent with separate ioctl (DeviceIoControl on Win2K) calls before the command call.

Description	**Command**	**Argument1**	**Argument2**
CLEAR_INTERRUPT	0x8073		
READ_PIXEL_COUNT	0x8075		
PCI_PC_RESET	0x8077		
ABORT_READOUT	0x8079		
BOOT_EEPROM	0x807B		
READ_NUMBER_OF_FRAMES_READ	0x807D		

READ_HEADER	0x81		
RESET_CONTROLLER	0x87		
INITIALIZE_IMAGE_ADDRESS	0x91	Image Address Least Significant 16 Bits	Image Address Most Significant 16 Bits
WRITE_COMMAND	0xB1		
PCI_DOWNLOAD	0x802F		

The vector commands are used with the following function calls:

Unix & Linux
Data contains the optional data value. *Command* contains a valid vector command value.

```
ioctl(pci_fd, ASTROPCI_HCVR_DATA, &data);   [optional data value]
…
ioctl(pci_fd, ASTROPCI_SET_HCVR, &command);
```

Windows 2000
In the first DeviceIoControl call, *inBuffer* contains the optional data value and *outBuffer* contains any returned data value. In the second DeviceIoControl call, *inBuffer* contains a valid vector command value and *outBuffer* contains any returned value.

```
[optional data value]
pci_request = CTL_CODE(ASTROPCI_DEVICE, (0x800 | ASTROPCI_ HCVR_DATA),
                      METHOD_BUFFERED,   FILE_ANY_ACCESS);
DeviceIoControl((HANDLE)pci_fd, pci_request, &inBuffer, sizeof(inBuffer),
                &outBuffer, sizeof(outBuffer), &bytesReturned, NULL));
…
pci_request = CTL_CODE(ASTROPCI_DEVICE, (0x800 | ASTROPCI_SET_HCVR),
                      METHOD_BUFFERED,   FILE_ANY_ACCESS);
DeviceIoControl((HANDLE)pci_fd, pci_request, &inBuffer, sizeof(inBuffer),
                &outBuffer, sizeof(outBuffer), &bytesReturned, NULL));
```

X. NORMAL (ASCII) COMMANDS

All "normal" commands are 3 character ascii sequences passed to the device driver through the ioctl (DeviceIoControl on Win2K) command. The commands are sent via a 6 element array containing all the necessary data. All unused elements must be set to -1 (undefined). The structure of the array is as follows:

cmd_data[0] = header
cmd_data[1] = command
cmd_data[2] = argument1
cmd_data[3] = argument2
cmd_data[4] = argument3
cmd_data[5] = argument4

where:

header = 0xssddnn

ss = source byte = 0

dd = destination byte

= 1 for PCI board

= 2 for timing board

= 3 for utility board
nn = number of words in command (>=2)
= (header + command + number_of_arguments) >= 2

command = 24 bit 3-character ASCII command
argument1 = -1 if unused (optional)
argument2 = -1 if unused (optional)
argument3 = -1 if unused (optional)
argument4 = -1 if unused (optional)

Unix & Linux
Cmd_data contains is the above command data structure.

```
ioctl(pci_fd, ASTROPCI_COMMAND, &cmd_data);
```

Windows 2000
Cmd_data is the above command data structure and *outBuffer* contains any returned data or reply.

```
pci_request = CTL_CODE(ASTROPCI_DEVICE, (0x800 | ASTROPCI_COMMAND),
                       METHOD_BUFFERED,  FILE_ANY_ACCESS);
DeviceIoControl((HANDLE)pci_fd, pci_request, &cmd_data, sizeof(cmd_data),
                &outBuffer, sizeof(outBuffer), &bytesReturned, NULL));
```

The following pages contain the full ascii command list along with all required arguments.

AEX	=	Abort Exposure
CDS	=	Correlated Double Sampling
CLK	=	mnemonic that means clock driver board
CLR	=	Clear Array
CSH	=	Close Shutter
DCA	=	Download Coadder
HGN	=	Set High Gain
IDL	=	Idle
LDA	=	Load Application
LGN	=	Set Low Gain
MH1	=	Move NIRIM Filter Wheel 1 Home
MH2	=	Move NIRIM Filter Wheel 2 Home
MM1	=	Move NIRIM Filter Wheel 1
MM2	=	Move NIRIM Filter Wheel 2
MPP	=	Multi-Pinned Phase Mode
OSH	=	Open Shutter
PEX	=	Pause Exposure

POF	=	Power Off
PON	=	Power On
RCC	=	Read Controller Configuration
RDI	=	Read Image
RDM	=	Read Memory
RET	=	Read Elapsed Time
REX	=	Resume Exposure
SBN	=	Set Bias Number
SBV	=	Set Bias Voltage
SET	=	Set Exposure Time
SEX	=	Start Exposure
SFS	=	Send Fowler Sample
SGN	=	Set Gain
SMX	=	Select Multiplexer
SNC	=	Set Number of Coadds
SOS	=	Select Output Source
SPT	=	Set Pass Through Mode
SRM	=	Set Readout Mode - either CDS or single
SSP	=	Set Subarray Positions
SSS	=	Set Subarray Sizes
STP	=	Stop Idle
SUR	=	Set Up The Ramp Mode
TDL	=	Test Data Link
VID	=	mnemonic that means video board
WRM	=	Write Memory
FPB	=	Frames Per Buffer

ASCII Command	Header	Argument1	Argument2	Argument3	Argument4	Argument5	Reply
'AEX'	(board_id << 8) | 2	-1	-1	-1	-1	-1	DON
'CDS'	(TIM_ID << 8) | 3	mode	-1	-1	-1	-1	DON
'CLR'	(TIM_ID << 8) | 2	-1	-1	-1	-1	-1	DON
'CSH'	(board_id << 8) | 2	-1	-1	-1	-1	-1	DON
'DCA'	(TIM_ID << 8) | 2	-1	-1	-1	-1	-1	DON
'HGN'	(TIM_ID << 8) | 4	gain	speed	-1	-1	-1	DON
'IDL'	(TIM_ID << 8) | 2	-1	-1	-1	-1	-1	DON
'LDA'	(TIM_ID << 8) | 3	application number	-1	-1	-1	-1	DON
'LGN'	(TIM_ID << 8) | 4	gain	speed	-1	-1	-1	DON
'MH1'	(UTIL_ID << 8) | 2	-1	-1	-1	-1	-1	DON
'MH2'	(UTIL_ID << 8) | 2	-1	-1	-1	-1	-1	DON
'MM1'	(UTIL_ID << 8) | 3	number of steps	-1	-1	-1	-1	DON
'MM2'	(UTIL_ID << 8) | 3	number of steps	-1	-1	-1	-1	DON
'MPP'	(TIM_ID << 8) | 3	mpp_mode	-1	-1	-1	-1	DON
'OSH'	(board_id << 8) | 2	-1	-1	-1	-1	-1	DON
'PEX'	(board_id << 8) | 2	-1	-1	-1	-1	-1	DON
'POF'	(board_id << 8) | 2	-1	-1	-1	-1	-1	DON
'PON'	(board_id << 8) | 2	-1	-1	-1	-1	-1	DON
'RCC'	(TIM_ID << 8) | 2	-1	-1	-1	-1	-1	config word
'RDI'	(board_id << 8) | 2	-1	-1	-1	-1	-1	DON
'RDM'	(board_id << 8) | 3	type | address	-1	-1	-1	-1	data
'RET'	(board_id << 8) | 2	-1	-1	-1	-1	-1	elapsed time
'REX'	(board_id << 8) | 2	-1	-1	-1	-1	-1	DON
'SBN'	(TIM_ID << 8) | 6	board ID	dac number	0x00564944	voltage	-1	DON
'SBV'	(TIM_ID << 8) | 2	-1	-1	-1	-1	-1	DON
'SET'	(TIM_ID << 8) | 3	exposure time	-1	-1	-1	-1	DON
'SEX'	(board_id << 8) | 2	-1	-1	-1	-1	-1	DON
'SFS'	(TIM_ID << 8) | 3	number of samples	-1	-1	-1	-1	DON
'SGN'	(TIM_ID << 8) | 4	gain	speed	-1	-1	-1	DON
'SMX'	(TIM_ID << 8) | 5	clock driver input	clock select 1 position	clock select 2 position	-1	-1	DON
'SNC'	(TIM_ID << 8) | 3	Number of coadds	-1	-1	-1	-1	DON
'SOS'	(TIM_ID << 8) | 3	amplifier	-1	-1	-1	-1	DON
'SPT'	(TIM_ID << 8) | 3	pass through mode	-1	-1	-1	-1	DON
'SRM'	(TIM_ID << 8) | 3	array reset mode	-1	-1	-1	-1	DON
'SSP'	(TIM_ID << 8) | 5	box Y offset	box X offset	bias offset	-1	-1	DON
'SSS'	(TIM_ID << 8) | 5	bias width	box width	box height	-1	-1	DON
'STP'	(TIM_ID << 8) | 2	-1	-1	-1	-1	-1	DON
'SUR'	(TIM_ID << 8) | 3	number of up the ramps	-1	-1	-1	-1	DON
'TDL'	(board_id << 8) | 3	data	-1	-1	-1	-1	data
'WRM'	(board_id << 8) | 4	type | address	data	-1	-1	-1	DON
'FPB'	(TIM_ID << 8) | 3	Number of FPB	-1	-1	-1	-1	DON

TABLE KEY:

Parameter	**Values**
address	Any relevant address.
amplifier	May be one of the following: A (0x5F5F41) Upper left amp B (0x5F5F42) Upper right amp C (0x5F5F43) Lower left amp D (0x5F5F44) Lower right amp AB (0x5F4142) Top two amps CD (0x5F4344) Bottom two amps ALL (0x414C4C) Quad readout L (0x5F5F4C) Lower left amp R (0x5F5F52) Lower right amp LR (0x5F4C52) Split serial readout
application number	Any valid application number in the range: 0 - 3
array reset mode	May be either: GLOBAL_RESET (0) or ROWBYROW_RESET (1)
bias offset	The number of (column) pixels to the left edge of the desired bias region.
bias width	Width of the desired bias region (in pixels).
board_id	May be one of the following: TIM_ID (0x2) Timing board UTIL_ID (0x3) Utility board
box height	Height of the desired subarray region (in pixels).
box width	Width of the desired subarray region (in pixels).
box X offset	Number of (column) pixels to the lower left corner of the desired subarray region.
box Y offset	Number of (row) pixels to the lower left corner of the desired subarray region.
clock select 1 position	
clock select 2 position	
dac_number	Must be 1 for channel A, and 3 for channel B.
data	Any relevant value.
gain	Must be one of the following: 1, 2, 5, 10
mode	May be either: SINGLE_READOUT (0) or DOUBLE_CORR_READOUT (1)
mpp_mode	May be either: ON (1) or OFF (0)
number of coadds	Must be in the range: 0 < NC < 65536
number of samples	Must be in the range: 0 < N < 65536
number of steps	Depends on which filter is selected.
number of up the ramps	Must be in the range: 2 < N < 65536
pass through mode	Must be either: PASS_THROUGH (1) or NO_PASS_THROUGH (0)
speed	Must be either: SLOW (0) or FAST (1)
type	Memory type. May be one of the following: P (0x100000) Program X (0x200000) Y (0x400000) R (0x800000) ROM
board ID	Must be in the range: 1<= N <= 16
voltage	24-bit voltage. Must be in the range: 0<= N <= 4095
config word	Controller configuration parameter word. Specifies what controller option are available. See section VII.
Number of FPB	The number of frames-per-buffer that will fit into the kernel image buffer. Used for co-addition.

XI. 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 saving image data, manipulating image data, displaying image data, mapping image buffers, PTC statistics, and device driver communications. The library files are accessed by Voodoo through the Java Native Interface (JNI). There are three separate libraries, one for unix, one for linux, and one for win2k. The libraries can be found in *xx/Voodoo/Clib/unix*, *xx/Voodoo/Clib/linux*, and *xx/Voodoo/Clib/win2k,* respectively. The function calls are the same for all three platforms, only the contents are different. The individual libraries are discussed here.

Libcdllibc.so

This is a unix/linux only library. This library supports the use of the iraf client display library to automatically display images in SAOImage, DS9, and Ximtool. Requires iraf to be installed.

Copen_cdl()	Opens a connection to the client display library.
Cclose_cdl()	Closes a connection to the client display library.

Libpcilibc.so, pcilibc.dll

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

Copen()	Opens a connection to the device driver.
Cclose()	Closes the connection to the device driver.
Cioctl()	Sends an ioctl() command to the device driver.

Libpcimemc.so, pcimemc.dll

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

Cmap_memory()	Maps the device driver image buffer to Voodoo.
Cunmap_memory()	Unmaps the device driver image buffer from Voodoo.
Cdata_check()	Performs a data check on synthetic images.
Cswap_memory()	Byte swaps the spedified 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, displibc.dll

The display library. This library contains functions to display an image on saoimage or ximtool. All functions are stubs in win2k.

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 saoimage or ximtool. Not used.

Cclose_display()	Closes a connection to saoimage or ximtool. Not used.
Libcdl.a	The Solaris CDL functions library.
Linux_libcdl.a	The Linux CDL functions library.

Libfitslibc.so, fitslibc.dll

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.
CcreateFitsFile()	Creates an open-ended FITS file. For use by continuous readout modes.
CwriteToFitsFile()	Write data to an open-ended FITS file. For use by continuous readout modes.
CcloseFitsFile()	Closes an open-ended FITS file. For use by continuous readout modes.

Libsetupcmdlibc.so, setupcmdlib.dll

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.

Libptclibc.so, ptclibc.dll

This library contains functions to calculate the statistics for the Photon Transfer Curve utility.

CdarkStats()	Calculates the dark image statistics.
CflatStats()	Calculates the flat field image statistics.

Libverlibc.so, verlibc.dll

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.

Libcrlibc.so, crlibc.dll

The continuous readout library. Contains functions that support the continuous readout modes of Voodoo.

CzeroBuffer()	Clears the specified buffer by filling it with 0’s.
Cadd()	Adds two 16-bit images together and saves them in a single 32-bit image.

XII. CONTROLLER CONFIGURATION PARAMETERS

The timing board DSP code on every controller contains a 24-bit word known as 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 downloaded on the controller.

Voodoo reads the controller configuration word by sending an RCC command to the timing board at the end of a controller setup sequence. A tabbed window is then created based on the word bits. Each tab pane corresponds to one of the available hardware options and allows parameters to be modified. For the parameters to be sent to the timing board, users must click “Apply Above” for each parameters tab they wish to modify. This prevents unintentional modification of values that may potentially cause harm to the system.

NOTE: If no configuration word exists, the default values are used by Voodoo, which are NONLINEAR TEMPERATURE CONVERSION, TIMING BOARD REV4B, UTILITY BOARD REV3, and SHUTTER available.
The DSP file *TIMHDR.ASM* contains the bit specification for the controller configuration and is:

```
; The bit is set (=1) if the capability is supported by the controller
BIT #'s         FUNCTION
2,1,0           Video Processor
                        000     CCD Rev. 3
                        001     CCD Gen I
                        010     IR Rev. 4
                        011     IR Coadder

4,3             Timing Board
                        00      Rev. 4, Gen II
                        01      Gen I

6,5             Utility Board
                        00      No utility board
                        01      Utility Rev. 3

7               Shutter
                        0       No shutter support
                        1       Yes shutter support

9,8             Temperature readout
                        00      No temperature readout
                        01      Polynomial Diode calibration
                        10      Linear sensor calibration

10              Subarray readout
                        0       Not supported
                        1       Yes supported

11              Binning
                        0       Not supported
                        1       Yes supported

12              Split-Serial readout
                        0       Not supported
                        1       Yes supported
```

13	Split-Parallel readout		
		0	Not supported
		1	Yes supported
14	MPP = Inverted parallel clocks		
		0	Not supported
		1	Yes supported
16,15	Clock Driver Board		
		00	Rev. 3
		11	No clock driver board (Gen I)
19,18,17		Special implementations	
		000	Somewhere else
		001	Mount Laguna Observatory
		010	NGST Aladdin
		xxx	Other

The timing board DSP file contains default values for all available hardware configurations. Voodoo allows users to modify these default values.

- IMPORTANT: Modified values are not permanently saved on the timing board EEPROM and will need to be applied every time one of the following occurs:

    - Controller power is cycled.
    - The controller setup is applied.

- To expedite this process, Voodoo allows users to save all parameters to a file, which can later be reloaded. For this purpose there are “Save” and “Load” buttons on the “Parameters Window”. Once a saved parameters file is loaded the user must go through each configuration tab pane and click “Apply Above” for those parameters to be loaded into the controller. The parameters are NEVER automatically loaded. Again, this is to prevent unintentional harm.

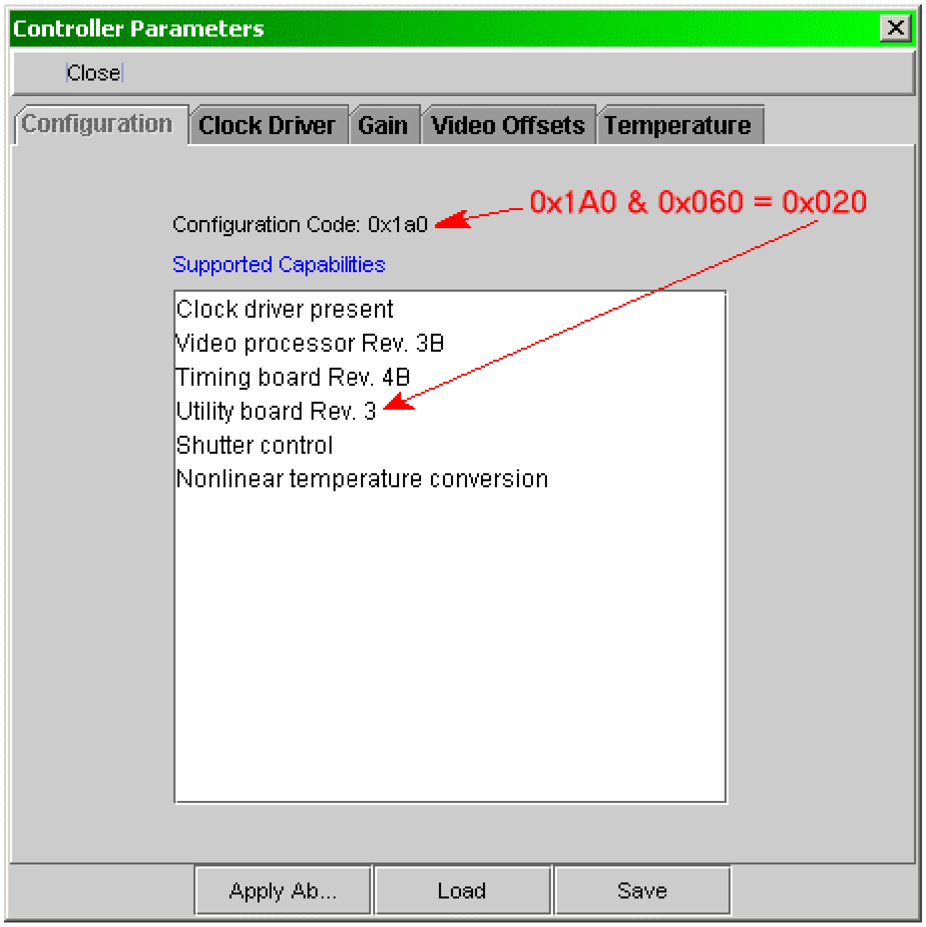
- It is important to note that the parameters only directly affect Voodoo in two minor ways, which are:

    - The deinterlace option on Voodoo’s Main Window may be changed. For example, selecting quad readout will automatically cause the deinterlace option to select quad deinterlacing.

    - The image dimensions in the Setup Window may be changed by some parameters. Binning, for example, would do this.

Example 1: Utility Board Exists

The controller returned 0x1A0 as the controller configuration. The utility board support is found by masking 0x1A0 with 0x060. A utility board is supported if: 0x1A0 & 0x060 = 0x020. And in this case it does.



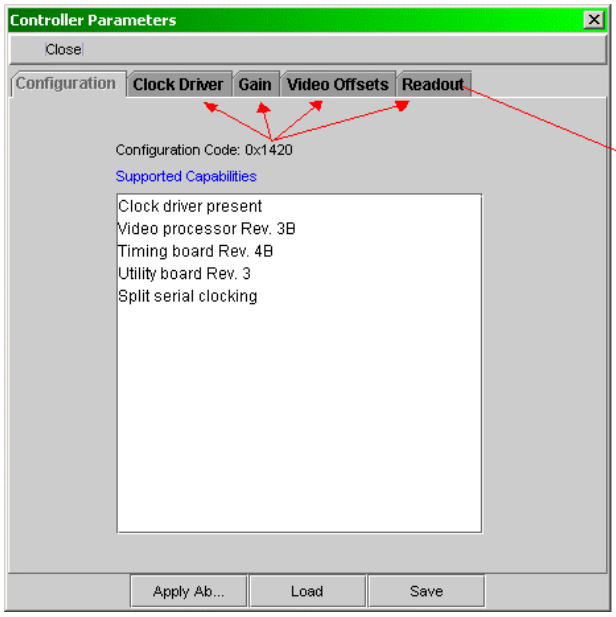
Configuration Code: 0x1a0

Supported Capabilities

Clock driver present
Video processor Rev. 3B
Timing board Rev. 4B
Utility board Rev. 3
Shutter control
Nonlinear temperature conversion

Example 2:

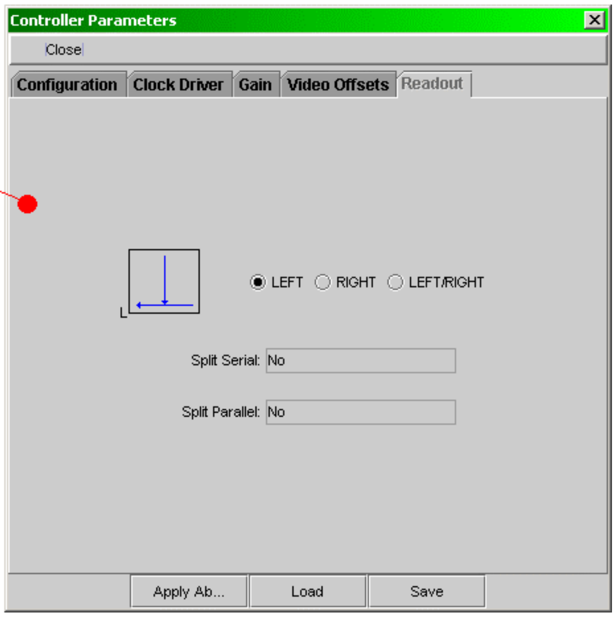
The controller has returned 0x1420, which will cause Voodoo to create the tabbed parameter panes for a Clock Driver (0x0), Video Processor Rev. 3B (0x0), Timing Board Rev. 4B (0x0), Utility Board Rev. 3 (0x20), Split Serial Readout options (0x1000), and Subarray is supported (0x400). 0x20 | 0x1000 | 0x400 = 0x1420.



Configuration Code: 0x1420

Supported Capabilities

Clock driver present
Video processor Rev. 3B
Timing board Rev. 4B
Utility board Rev. 3
Split serial clocking



[● LEFT] ○ RIGHT ○ LEFT/RIGHT

Split Serial: No

Split Parallel: No

The following sections describe the sequence of commands to set any of the above controller configuration paramters. All commands are sent via the device driver ioctl()/DeviceIoControl() function. See the command description section for details about command arguments.

BINNING

Write (WRM) the number of column pixels to bin to Y:5 on the timing board and the number of row pixels to bin to Y:6.

```
P =   0x100000;  //  Bit 20
X =   0x200000;  //  Bit 21
Y =   0x400000;  //  Bit 22
R =   0x800000;  //  Bit 23

// Column pixels to bin.
cmd_data[0] = 0x000203
cmd_data[1] = WRM
cmd_data[2] = 0x400000 | 5
cmd_data[3] = number of columns to bin
cmd_data[4] = -1
cmd_data[5] = -1
ioctl (pci_fd, ASTROPCI_COMMAND, &data);


// Row pixels to bin.
cmd_data[0] = 0x000203
cmd_data[1] = WRM
cmd_data[2] = 0x400000 | 6
cmd_data[3] = number of rows to bin
cmd_data[4] = -1
cmd_data[5] = -1
ioctl (pci_fd, ASTROPCI_COMMAND, &data);
```

CLOCK DRIVER

If (doMux) then

Send SMX command with clock_driver_input, clock_select_1_positon, and clock_select_2_positon arguments to the timing board.

```
cmd_data[0] = 0x000205
cmd_data[1] = SMX
cmd_data[2] = clock driver input
cmd_data[3] = clock select 1 position
cmd_data[4] = clock select 2 position
cmd_data[5] = -1
ioctl (pci_fd, ASTROPCI_COMMAND, &data);
```

If (doClockVoltages) then {

For (all available voltages) {

Send SBN command with high voltage to timing board.

```
cmd_data[0] = 0x000203
cmd_data[1] = SBN
cmd_data[2] = high voltage value
cmd_data[3] = -1
cmd_data[4] = -1
cmd_data[5] = -1
```

```
                ioctl (pci_fd, ASTROPCI_COMMAND, &data);


                Send SBN command with low voltage to timing board.

                cmd_data[0] = 0x000203
                cmd_data[1] = SBN
                cmd_data[2] = low voltage value
                cmd_data[3] = -1
                cmd_data[4] = -1
                cmd_data[5] = -1
                ioctl (pci_fd, ASTROPCI_COMMAND, &data);

        }

    }
```

Coadder

```
    If (download coadder) then

        Send DCA to the timing board .

        cmd_data[0] = 0x000202
        cmd_data[1] = DCA
        cmd_data[2] = -1
        cmd_data[3] = -1
        cmd_data[4] = -1
        cmd_data[5] = -1
        ioctl (pci_fd, ASTROPCI_COMMAND, &data);


    If (set number of coadds) then

        Send the number of coadds using SNC to the timing board.

        cmd_data[0] = 0x000203
        cmd_data[1] = SNC
        cmd_data[2] = number of coadds
        cmd_data[3] = -1
        cmd_data[4] = -1
        cmd_data[5] = -1
        ioctl (pci_fd, ASTROPCI_COMMAND, &data);
```

Gain

```
    Send the gain and speed using SGN to the timing board.

        cmd_data[0] = 0x000204
        cmd_data[1] = SGN
        cmd_data[2] = gain
        cmd_data[3] = speed
        cmd_data[4] = -1
        cmd_data[5] = -1
        ioctl (pci_fd, ASTROPCI_COMMAND, &data);
```

Gain (GEN I)

If (setting high gain) then

Send the gain and speed using HGN to the timing board.

```
cmd_data[0] = 0x000204
cmd_data[1] = HGN
cmd_data[2] = gain
cmd_data[3] = speed
cmd_data[4] = -1
cmd_data[5] = -1
ioctl (pci_fd, ASTROPCI_COMMAND, &data);
```

Else if (setting low gain) then

Send the gain and speed using LGN to the timing board.

```
cmd_data[0] = 0x000204
cmd_data[1] = LGN
cmd_data[2] = gain
cmd_data[3] = speed
cmd_data[4] = -1
cmd_data[5] = -1
ioctl (pci_fd, ASTROPCI_COMMAND, &data);
```

MPP

If (doMPP) then

Send 1 using MPP command to timing board.

```
cmd_data[0] = 0x000203
cmd_data[1] = MPP
cmd_data[2] = 1
cmd_data[3] = -1
cmd_data[4] = -1
cmd_data[5] = -1
ioctl (pci_fd, ASTROPCI_COMMAND, &data);
```

Else

Send 0 using MPP command to timing board.

```
cmd_data[0] = 0x000203
cmd_data[1] = MPP
cmd_data[2] = 0
cmd_data[3] = -1
cmd_data[4] = -1
cmd_data[5] = -1
ioctl (pci_fd, ASTROPCI_COMMAND, &data);
```

NGST

If (number of up the ramps >= 2) then

Send DCA command to timing board.

```
cmd_data[0] = 0x000202
cmd_data[1] = DCA
cmd_data[2] = -1
```

```
cmd_data[3] = -1
cmd_data[4] = -1
cmd_data[5] = -1
ioctl (pci_fd, ASTROPCI_COMMAND, &data);
```

Send the array reset mode using SRM to the timing board.

```
cmd_data[0] = 0x000203
cmd_data[1] = SRM
cmd_data[2] = array reset mode
cmd_data[3] = -1
cmd_data[4] = -1
cmd_data[5] = -1
ioctl (pci_fd, ASTROPCI_COMMAND, &data);
```

Send the number of samples using SFS to the timing board.

```
cmd_data[0] = 0x000203
cmd_data[1] = SFS
cmd_data[2] = number of fowler samples
cmd_data[3] = -1
cmd_data[4] = -1
cmd_data[5] = -1
ioctl (pci_fd, ASTROPCI_COMMAND, &data);
```

Send the readout mode using CDS to the timing board.

```
cmd_data[0] = 0x000203
cmd_data[1] = CDS
cmd_data[2] = readout mode
cmd_data[3] = -1
cmd_data[4] = -1
cmd_data[5] = -1
ioctl (pci_fd, ASTROPCI_COMMAND, &data);
```

Send the pass through mode using SPT to the timing board.

```
cmd_data[0] = 0x000203
cmd_data[1] = SPT
cmd_data[2] = pass through mode
cmd_data[3] = -1
cmd_data[4] = -1
cmd_data[5] = -1
ioctl (pci_fd, ASTROPCI_COMMAND, &data);
```

Send the number of coadds using SNC to the timing board.

```
cmd_data[0] = 0x000203
cmd_data[1] = SNC
cmd_data[2] = number of coadds
cmd_data[3] = -1
cmd_data[4] = -1
cmd_data[5] = -1
ioctl (pci_fd, ASTROPCI_COMMAND, &data);
```

Send the number of up the ramps using SUR to the timing board.

```
cmd_data[0] = 0x000203
cmd_data[1] = SUR
cmd_data[2] = number of up-the-ramps
cmd_data[3] = -1
cmd_data[4] = -1
cmd_data[5] = -1
ioctl (pci_fd, ASTROPCI_COMMAND, &data);
```

NIRIM

```
If (move filter wheel 1) then {

    If (move to COLD PLATE) then

        Send MH1 command to utility board.

        cmd_data[0] = 0x000302
        cmd_data[1] = MH1
        cmd_data[2] = -1
        cmd_data[3] = -1
        cmd_data[4] = -1
        cmd_data[5] = -1
        ioctl (pci_fd, ASTROPCI_COMMAND, &data);

    Else if (new position < last position) then

        Send MH1 command to utility board.

        cmd_data[0] = 0x000302
        cmd_data[1] = MH1
        cmd_data[2] = -1
        cmd_data[3] = -1
        cmd_data[4] = -1
        cmd_data[5] = -1
        ioctl (pci_fd, ASTROPCI_COMMAND, &data);

        Send number of motor steps using MM1 command to utility board.

        cmd_data[0] = 0x000303
        cmd_data[1] = MM1
        cmd_data[2] = number of motor steps
        cmd_data[3] = -1
        cmd_data[4] = -1
        cmd_data[5] = -1
        ioctl (pci_fd, ASTROPCI_COMMAND, &data);

    Else if (new position > last position) then

        Send number of motor steps using MM1 command to utility board.

        cmd_data[0] = 0x000302
        cmd_data[1] = MM1
        cmd_data[2] = number of motor steps
        cmd_data[3] = -1
        cmd_data[4] = -1
        cmd_data[5] = -1
        ioctl (pci_fd, ASTROPCI_COMMAND, &data);

    If (filter wheel 2 needs to move) then

        Send number of motor steps using MM2 command to utility board.

        cmd_data[0] = 0x000302
        cmd_data[1] = MM2
        cmd_data[2] = number of motor steps
        cmd_data[3] = -1
        cmd_data[4] = -1
        cmd_data[5] = -1
        ioctl (pci_fd, ASTROPCI_COMMAND, &data);

}

Else if (move filter wheel 2) then {
        If (new position < last position) then

            Send MH2 command to utility board.

            cmd_data[0] = 0x000302
            cmd_data[1] = MH2
            cmd_data[2] = -1
            cmd_data[3] = -1
            cmd_data[4] = -1
            cmd_data[5] = -1
            ioctl (pci_fd, ASTROPCI_COMMAND, &data);

            Send number of motor steps using MM2 command to utility board.

            cmd_data[0] = 0x000302
            cmd_data[1] = MM2
            cmd_data[2] = number of motor steps
            cmd_data[3] = -1
            cmd_data[4] = -1
            cmd_data[5] = -1
            ioctl (pci_fd, ASTROPCI_COMMAND, &data);

        Else if (new position > last position) then

            Send number of motor steps using MM2 command to utility board.

            cmd_data[0] = 0x000302
            cmd_data[1] = MM2
            cmd_data[2] = number of motor steps
            cmd_data[3] = -1
            cmd_data[4] = -1
            cmd_data[5] = -1
            ioctl (pci_fd, ASTROPCI_COMMAND, &data);

        If (filter wheel 1 needs to move) then

            Send number of motor steps using MM1 command to utility board.

            cmd_data[0] = 0x000302
            cmd_data[1] = MM1
            cmd_data[2] = number of motor steps
            cmd_data[3] = -1
            cmd_data[4] = -1
            cmd_data[5] = -1
            ioctl (pci_fd, ASTROPCI_COMMAND, &data);

    }
```

READOUT

```
    Send the amp using SOS command to timing board.

        cmd_data[0] = 0x000203
        cmd_data[1] = SOS
        cmd_data[2] = amp
        cmd_data[3] = -1
        cmd_data[4] = -1
        cmd_data[5] = -1
        ioctl (pci_fd, ASTROPCI_COMMAND, &data);
```

Temperature

Write the adu temperature value to Y:1C using WRM to the utility board.

```
cmd_data[0] = 0x000303
cmd_data[1] = WRM
cmd_data[2] = (Y | 1C)
cmd_data[3] = adu temperature value
cmd_data[4] = -1
cmd_data[5] = -1
ioctl (pci_fd, ASTROPCI_COMMAND, &data);
```

Video Offsets

For (each video offset board id) {

Using the SBN command, send the video channel A offset[video offset board id] to the timing board.

```
cmd_data[0] = 0x000203
cmd_data[1] = SBN
cmd_data[2] = channel A video offset board id
cmd_data[3] = -1
cmd_data[4] = -1
cmd_data[5] = -1
ioctl (pci_fd, ASTROPCI_COMMAND, &data);
```

Using the SBN command, send the video channel B offset[video offset board id] to the timing board.

```
cmd_data[0] = 0x000203
cmd_data[1] = SBN
cmd_data[2] = channel B video offset board id
cmd_data[3] = -1
cmd_data[4] = -1
cmd_data[5] = -1
ioctl (pci_fd, ASTROPCI_COMMAND, &data);
```

}

XIII. TEMPERATURE CALIBRATION

The calibration of the temperature depends on which sensor the camera contains. This section describes how to calculate the temperature for the non-linear CY7 series silicon diodes. The curve is fit by a polynomial expansion based upon the Chebychev polynomials. This curve has four different fits based upon the temperature range required. The default values in Voodoo are set for the range of temperatures from 100K to 475K.

Recall that the Chebychev equation has the basic form:

$$T(x) = \sum_{n=0} a_n t_n(x)$$

where T(x) is the temperature in Kelvins, a_n is the nth coefficient, $t_n(x)$ is the nth Chebychev polynomial, and x is the dimensionless voltage defined by:

$$x = \frac{(V - VL) - (VU - V)}{VU - VL}$$

where VU and VL are the upper and lower voltage limits, respectively. The Chebychev polynomials were calculated using the recursion relation:

$$t_{n+1}(x) = 2xt_n(x) - t_{n-1}(x)$$

where $t_0(x) = 1$ and $t_1(x) = x$.

The DSP software determines the ADU count, not the voltage through the diode. It is therefore necessary to convert from ADUs to volts in order to make use of the above equations. The ADU count varies linearly with the voltage, so it is only necessary to determine the conversion factor from ADUs to volts and the ADU offset. The default value used for aduPerVolt is 1366.98 and the value for aduOffset is 2045. These values were calibrated from the 2048x2048 CCD at Mt. Laguna Observatory.

$$Volts = \frac{adu - aduOffset}{aduPerVolt}$$

All coefficients and constants can be found and modified in VoodooConstants.java. The maximum number of coefficients that Voodoo is capable of handling is eleven (0 – 10).

XIIII. DEVICE DRIVER USAGE (UNIX & LINUX)

The device driver has four main entry points available for interaction with the camera. The four entry points are: open(), close(), mmap(), munmap(), 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 corrospond 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. mmap()

Used to map the device driver image buffer to user space.

Usage:

mmap*(void *addr, size_t len, int prot, int flags,int fildes,off_t off);*

addr

Set to 0.

len

The size (in bytes) of the memory to be mapped.

prot

The read/write mode. Set to *(PROT_READ | PROT_WRITE).*

flags

Specifies whether or not memory is shared. Set to *MAP_SHARED.*

fildes

The PCI file descriptor returned from the open() call.

o*ffset*

Set to 0.

4. munmap()

This function frees the image buffer memory.

Usage:

munmap*(void * filde);*

fildes

The file descriptor returned from the mmap() call.

5. 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 and defined in astropci_ioctl.h.

arg

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

ASTROPCI_GET_HCTR (0x1)

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

ASTROPCI_GET_PROGRESS(0x2)

Get the current pixel transfer count.

ASTROPCI_GET_DMA_ADDR (0x3)

Returns the current start address of the image buffer.

ASTROPCI_GET_HSTR (0x4)

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

ASTROPCI_HCVR_DATA (0x10)

Set a data value for the next HCVR command.

ASTROPCI_GET_HCTR (0x11)

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

ASTROPCI_SET_HCVR (0x12)

Set the current value of the PCI DSP Host Control Vector Register.

ASTROPCI_PCI_DOWNLOAD (0x13)

Set the PCI board into download mode. Used for ROM burning.

ASTROPCI_PCI_DOWNLOAD_WAIT (0x14)

Waits for the PCI board to complete the ROM burning process.

ASTROPCI_COMMAND (0x15)

Sends the specified command to the specified board.

XV. DEVICE DRIVER USAGE (WINDOWS 2000)

The device driver has three main entry points available for interaction with the camera. The three entry points are: CreateFile(), CloseHandle(), and DeviceIoControl(). To use these functions, the user must include the system file *windows.h* and *winioctl.h*.

1. CreateFile()

Opens a connection to the requested device. This function must succeed before any further access to the device may occur. This method should be set to overlapped for asynchronous readout. Returns a handle to the PCI device driver upon success.

Usage:

```
HANDLE CreateFile(
      LPCTSTR lpFileName,           // pointer to name of the file
      DWORD dwDesiredAccess,        // access (read-write) mode
      DWORD dwShareMode,            // share mode
      LPSECURITY_ATTRIBUTES lpSecurityAttributes, // pointer to security
                                                     attributes
      DWORD dwCreationDisposition,   // how to create
      DWORD dwFlagsAndAttributes,    // file attributes
      HANDLE hTemplateFile           // handle to file with attributes to
                                        copy
);
```

Should be set as follows, where deviceName is "\\\\.\\driverName". The default value for Voodoo is "\\\.\\ARCPNP1":

```
CreateFile(deviceName,
           GENERIC_READ,
           FILE_SHARE_READ | FILE_SHARE_WRITE,
           NULL,
           OPEN_EXISTING,
           FILE_FLAG_OVERLAPPED,
           NULL);
```

2. CloseHandle()

Closes a connection to the requested device. Returns TRUE for success, FALSE otherwise.

Usage:

```
      BOOL CloseHandle(
            HANDLE hObject   // handle to object to close
      );
```

3. DeviceIoControl()

This is the "do all" instruction. Used to pass parameters, set controller states, receive controller status, and issue controller commands. Returns TRUE for success or FALSE for failure. See astropci_ioctl.h for a complete list of commands.

Usage:

```
      BOOL DeviceIoControl(
            HANDLE hDevice,        // handle to device of interest
            DWORD dwIoControlCode, // control code of operation to perform
            LPVOID lpInBuffer,     // pointer to buffer to supply input data
            DWORD nInBufferSize,   // size, in bytes, of input buffer
            LPVOID lpOutBuffer,    // pointer to buffer to receive output data
            DWORD nOutBufferSize,    // size, in bytes, of output buffer
            LPDWORD lpBytesReturned, // pointer to variable to receive byte
                                        count
            LPOVERLAPPED lpOverlapped // pointer to structure for asynchronous
                                         operation
            );
```

ASTROPCI_GET_HCTR (0x801)

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

ASTROPCI_GET_PROGRESS(0x802)

Get the current pixel transfer count.

ASTROPCI_GET_DMA_ADDR (0x803)

Returns the current start address of the image buffer.

ASTROPCI_GET_HSTR (0x804)

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

ASTROPCI_MEM_MAP (0x805)

Maps the device driver image buffer to user space.

ASTROPCI_HCVR_DATA (0x810)

Set a data value for the next HCVR command.

ASTROPCI_GET_HCTR (0x811)

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

ASTROPCI_SET_HCVR (0x812)

Set the current value of the PCI DSP Host Control Vector Register.

ASTROPCI_PCI_DOWNLOAD (0x813)

Set the PCI board into download mode. Used for ROM burning.

ASTROPCI_PCI_DOWNLOAD_WAIT (0x814)

Waits for the PCI board to complete the ROM burning process.

ASTROPCI_COMMAND (0x815)

Sends the specified command to the specified board.

ASTROPCI_MEM_UNMAP (0x816)

Unmaps the image buffer from user space.

XVI. 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 0x87 (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
            {
                  Use the ioctl() to send TDL command.

                  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
            {
                  Use the ioctl() to send TDL command.

                  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
            {
                  Use the ioctl() to send TDL command.

                  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
      {
            Send the STP command to the timing board using ioctl().

            Send the LDA command to the timing board using ioctl().

            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
      {
            Send the LDA command to the timing board using ioctl().

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

If (do power on) then
{
      If (timing equals master) then
            Send the PON command to the timing board using ioctl().

      Else if (utility equals master) then
            Send the PON command to the utility board using ioctl().

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

If (do set dimensions) then
{
      // Set the number of columns.
      Write (WRM) the columns to Y:1 on the timing board.

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

      // Set the number of rows.
      Write (WRM) the rows to Y:2 on the timing board.

      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.
}

If (did timing download) then
      Send RCC to timing board to get controller configuration word.
```

2. Exposure Sequence Pseudo Code

```
// ----------------------------------------------
// Set the shutter position.
// Bit 11 of the controller status is set/unset.
// ----------------------------------------------
If (timing board equals master) then
      Send RDM of X:0 to timing board. Status is returned.
Else if (utility board equals master) then
      Send RDM of X:1 to utility board. Status is returned.

If (open shutter equals true) then
      If (timing board equals master) then
            Send WRM to X:0 with an argument of (status | (1<<11)) to the timing
            board.
      Else if (utility board equals master) then
```

```
            Send WRM to X:1 with an argument of (status | 1) to the utility board.
Else
      If (timing board equals master) then
            Send WRM to X:0 with an argument of (status & ~(1<<11)) to the timing
            board.
      Else if (utility board equals master) then
            Send WRM to X:1 with an argument of (status & 0xFFFFFFFE) to the utility
            board.

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

// ---------------------------------------------
// Set the exposure time.
// ---------------------------------------------
if (timing board equals master) then
      Send SET with the exposure time argument to the timing board.

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

Else if (utility board equals master) then
      Send WRM of Y:18 with the exposure time as the argument to the utility board.

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

// 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.
      Set the current pixel count to 0.
      Set the last pixel count to 0.
      Set maximum pixel count to rows*columns.

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

      // ---------------------------------------------------
      // Start the exposure.
      // ---------------------------------------------------
      Send SEX to whichever board is master.

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

      // ---------------------------------------------------
      // Do the exposure.
      //
      // NOTES:
      // Reading the exposure time and current progress bar address
      // require different sleep times (0.5 sec - RET, 25 ms –
      // ASTROPCI_GET_PROGRESS). To compensate, the following
      // loop sleeps for 25 ms and the elapsed exposure time is only
      // read every 20*25ms = 500ms (0.5 sec).
      // ------------------------------------------------
      while (current pixel count < maximum pixel count) then {
            Read current status value using ioctl() command ASTROPCI_GET_HSTR. Then
            status = (status & HTF_BITS) >> 3.

            // ------------------------------------------------
            // Read elapsed exposure time.
            // ------------------------------------------------
            if (status not equal to 0x5 (READOUT) AND
                  exposure counter is greater or equal to 20 AND
                  elapsed time is greater or equal to 1 second) then {

                  if (timing board equals master) then
                        Send RET to timing board. Elapsed time
                        returned.

                  Else if (utility board equals master) then
                        Send RDM to Y:17. Elapsed time returned.

                  Display the elapsed time.

                  Increment exposure counter.
            }
            // ------------------------------------------------
            // Readout current byte count.
            // ------------------------------------------------
            else {
                  Send the ioctl() command ASTROPCI_GET_PROGRESS. The current pixel
                  count will be returned.

                  If (current pixel count equals last pixel count) then
                        Increment readout timeout.

                  If (readout timeout is greater than 200) then
                        Print timeout message and exit.
            }
            Sleep for 25 milliseconds.
      }

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

      // Un-swap the image data.
      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.
}
```

XVII. C APPLICATION INTERFACE

We offer a C API for users who wish to interface their own application or scripts to the device driver and controller. The API is written in C and contains the basic device driver function calls, which are the building blocks for all controller communications. There are only 5 basic function calls, one to read the PCI DSP command register (HCTR), one to write the PCI DSP command register, one to read the PCI DSP status register (HSTR), one to write a vector command to the PCI board, and one to write all regular ASCII commands. The API also contains functions for reading array temperature, deinterlacing images, storing FITS files, and loading PCI, timing, and utility DSP .lod files.

The API is bundled into the following:

- UNIX – static archive library (.a), dynamic library (.so)
- LINUX – static archive library (.a), dynamic library (.so)
- Windows 2000 – static library (.lib), Dynamic Linked Library (.dll)

The C API is supplied with a sample application called *"apiTest"*. It is fully function text based program that exercises all of the API.

```
===================================================================
MAIN MENU
===================================================================
1. Apply Controller Setup
2. Apply Controller Parameters
3. Expose
4. Let It Rip (Do Options 1, 2, 3 With Defaults)
5. Exit

Enter Option: |
```

XVIII. REVISION HISTORY

DATE	AUTHOR	CHANGE
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.
09/26/2001	Scott Streit	Updated for version 1.7
05/08/2002	Scott Streit	Updated for version 1.7-A
06/24/2002	Scott Streit	Added non-linear temperature calibration section.