Operating the UCO/Lick UCAM CCD Controller

by

Mingzhi Wei and Richard Stover

UCO/Lick Observatory
The University of California

January, 2005

Table of Contents

Index of Figures

Index of Tables

1. Introduction

This document is a guide to using the new UCAM CCD controller designed by Mingzhi Wei. The descriptions given here are based on the controllers used in the UCO/Lick guide cameras and the guide camera software written by Richard Stover. This document is written for controller users who wish to write software that communicates with the controller. A separate *UCAM Technical Reference Manua*l describes the hardware in more detail and provides diagnostic and trouble-shooting guides. If you are looking for hardware block diagrams, signal flow charts, or documentation on individual boards you want to look in the *UCAM Technical Reference Manual*.

2. Controller Components

The controller is shown in Figure 1. As the figure shows there are only three connections to the controller. The large power cable brings in DC power from a separate power supply box. The video data comes out on the optical fiber. Commands for controlling the controller are transmitted over the RS232 connection.

The BNC connector shown in Figure 1 brings out the TTL shutter control signal. The switch, seen to the left of the BNC connector, is used to turn off all of the controller's LED status lights. The switch is shown in the lights-on position. Just above the switch is one of the status LEDs controlled by the switch. This green LED lights every time a CCD readout occurs.

The controller can be configured with all of the connectors positioned on the side of the controller instead of on the back of the controller.

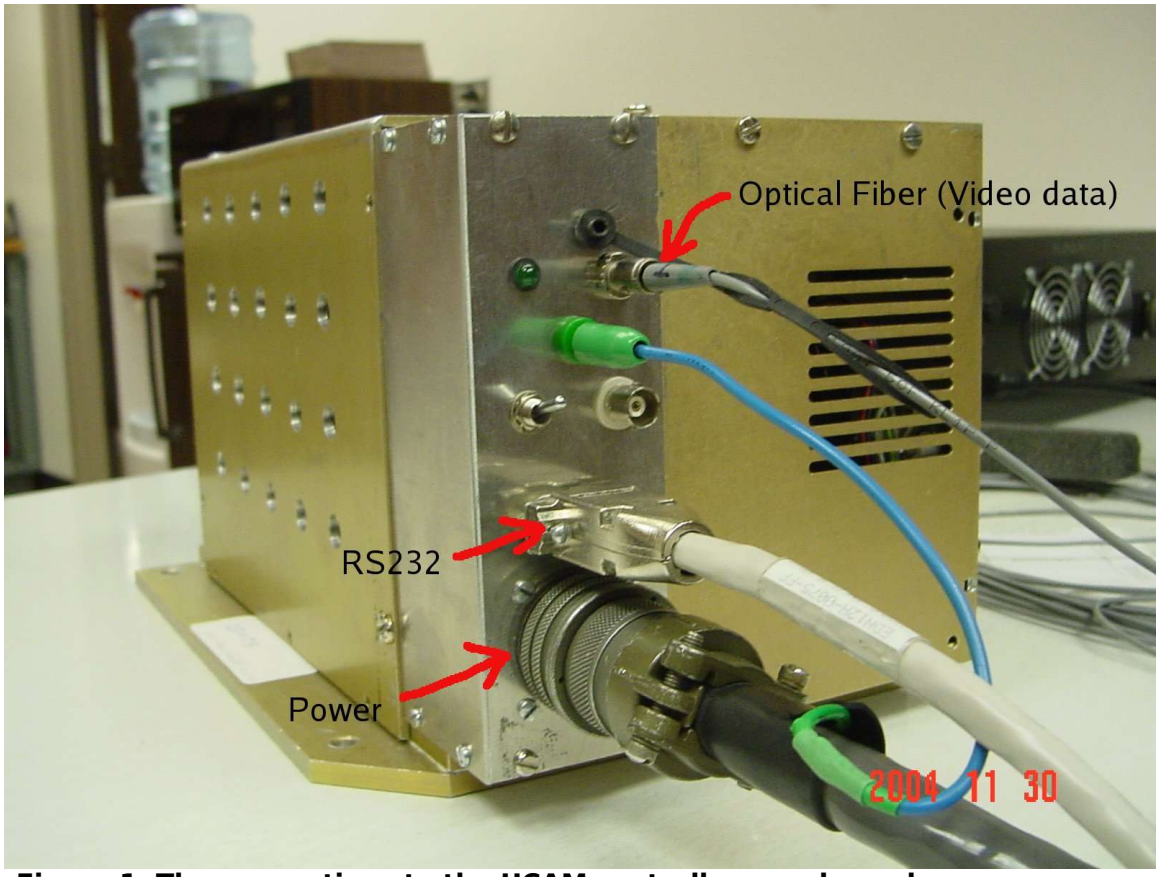

Figure 1. The connections to the UCAM controller are shown here.

The controller actually has one other connector. On the other side of the controller as viewed in Figure 1 is the connector to the CCD detector. We will not discuss the details of that connector in this document.

There are two small CPUs contained in the controller. One of the CPUs manages the controller operation. This is typically called the timing control CPU. The other CPU independently monitors and controls the CCD detector temperature. The RS232 line is connected to the temperature monitor CPU via an optical isolation circuit.

3. Controller Input and Output

The controller produces digital video. The digital data are transmitted on a 200 MHz optical fiber. All other communications with the controller uses the serial RS232. While the fiber is output-only, the RS232 line is bidirectional. Commands are sent to the controller over the RS232 line. Commands to the controller typically result in some type of response from

the controller. That response is sent over the RS232 line. The controller may also send status messages over the RS232. The RS232 line normally uses the following configuration, but these can be adjusted if necessary.

Parameter	***Value***
Baud rate (in and out)	9600
Bits	8
Parity	None

Table 1. RS232 parameters.

The RTS/CTS control lines are used to control the state of the controller CPU. This is explained in detail in Appendix IV. To avoid added electronic noise in the image data ***the RTS/CTS lines must be used properly***.

4. Command Formats

The controller is operated by sending it a series commands over the RS232 serial line. A controller command consists of a command word and zero or more parameters. All commands begin with a beginning-of-command (BOC) character and at least two additional ASCII letters. Some commands use one or more additional ASCII letters to further define the command. If a command word has parameters they must all be supplied, they must be supplied in a fixed order, and they must be supplied in a format appropriate for the command. Every command is terminated by a newline character.

The controller normally sends some type of response for every command received. If the command requests some type of information (like the value of a memory location or a voltage value) then that information is returned on the serial line as a string of ASCII characters. Controller commands which instruct the controller to take some action (like storing the next exposure time, starting an exposure, etc.) return the 2-letter ASCII string OK. Every response is terminated with a newline character.

Error checking of commands within the controller is currently minimal. Inappropriate characters received by the controller are silently thrown away. For instance, if the controller is expecting a BOC character then anything other than a BOC character is discarded until one is received. The controller assumes that any parameters received are valid. So validity checking must occur within upper level software. The controller currently does not have command checksums, so verifying the integrity of a command is not possible. This may change in the future.

5. Command Categories

Commands can be classified by the BOC character. The BOC characters can be one of these three: **$>&**. BOC characters **$** and > begin commands for the timing board CPU of the controller and the BOC character **&** begins commands for the temperature control CPU of the controller. Since the RS232 line is actually connected to the temperature control CPU any commands with the BOC characters **$** and > are forwarded by the temperature control CPU to the timing control CPU. Responses from the timing control CPU are also transparently forwarded by the temperature control CPU onto the RS232 output line. The flow of commands is shown schematically in the following figure.

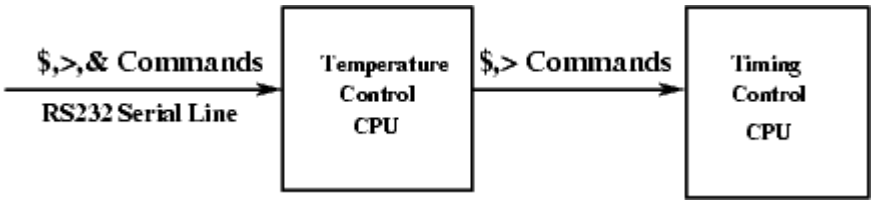

Figure 2. RS232 command flow. Commands flow first to the temperature control CPU. $ and > commands are relayed to the timing control CPU.

The difference between the **$** and > commands is in the way command parameters are transmitted to the controller. For all **$** commands the parameters are transmitted as binary data while for all > commands the parameters are transmitted as ASCII character data. The > BOC commands have the advantage that it is possible to type them in a terminal emulation program (very useful for debugging). The **$** commands, while they need a program to properly format the binary data, have the advantage of more efficient data transmission. All commands that produce some type of data response use the > form of the command and produce ASCII character responses. All of the **$** commands result in the **OK** response.

6. Exposure and Readout Sequence

The controller can read out a CCD in frame-transfer mode or non-frame-transfer mode. Figure 3 shows the sequence of events for non-frame-transfer mode. The sequence for frame-transfer is the same except the erase sequences are eliminated and after the exposure is completed the image is

rapidly shifted from the imaging half of the CCD to the storage half of the CCD. In Figure 3 time flows from the top of the diagram to the bottom of the diagram and from left to right. The colored rectangles in the middle can be thought of as a schematic representation of the CCD.

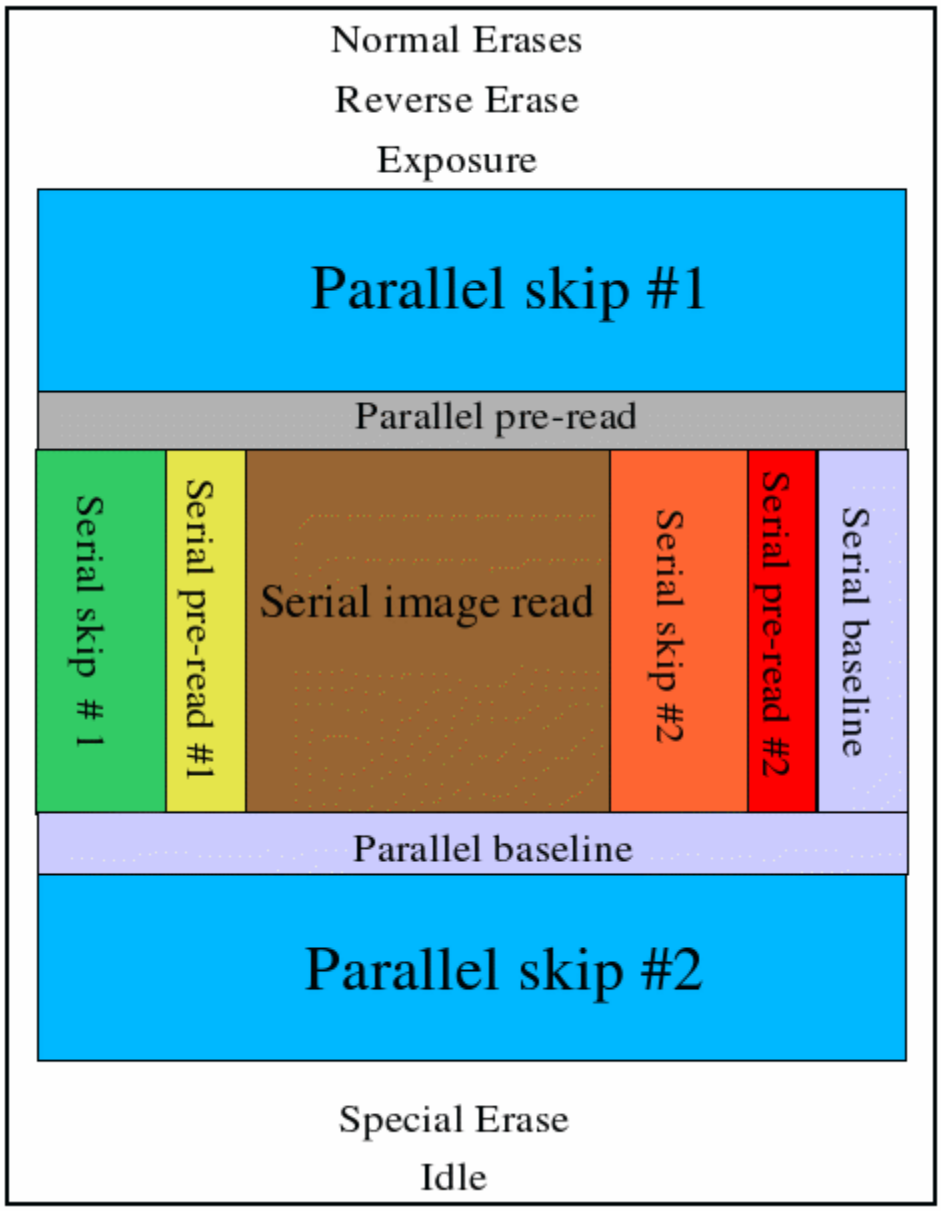

Figure 3. Schematic representation of typical exposure sequence.

Table 2 lists the sequence of events that occurs in chronological order and

provides details on each event.

Event	***Notes***	***Command parameters***
Normal Erase	A “normal” erase is a sequence of rapid parallel and serial transfers to flush any remaining charge.	The number of erases is set by **$DE** (data0)(data1). Zero is an acceptable value. Each erase produces enough parallel clocking to flush the entire CCD.
Reverse Erase	A reverse erase is a sequence of rapid reverse-direction parallel transfers.	Parameter **$DE** (data2) is a flag. 0 means no reverse erase. Non-0 means one reverse erase.
Exposure	The shutter may be opened and then a period of time elapses. If the shutter was opened it is closed and the readout sequence begins.	The state of the shutter during exposure is set by **$DT** (data3). 0 means the shutter remains closed. 1 means it is opened. **$DT** (data0)(data1)(data2) defines the exposure time.
Parallel Skip #1	If the user-defined readout window does not begin at the first row of the detector then some of the unwanted rows will be skipped at high speed.	The controller determines how many rows to skip based in the readout window defined in the **$DA** command.
Parallel pre-read #1	As a transition from the high-speed skipping of Parallel Skip #1 a number of pseudo-rows are read at normal clocking speeds prior to the actual image readout.	**$DD3** (data3)(data4) sets the number of pseudo rows.
Serial Skip #1	If the user-defined readout window does not begin at the first column of the detector then some of the unwanted columns will be skipped at high speed.	The controller determines how many columns to skip based in the readout window defined in the **$DA** command. Some details of the serial skipping are specified by **$DD2** (data0)(data1)(data2).

Event	*Notes*	*Command parameters*
Serial pre-read #1	As a transition from the high-speed skipping of Serial Skip #1 a number of columns are skipped at normal clocking speeds prior to the actual image readout.	**$DD1** (data1)(data2) set the number of pre-read serial transfer, where a transfer is equal to the current binning. Hence a value of 4 would be 4 CCD pixels for no binning and 8 CCD pixels for serial binning of 2.
Serial image read	This is the readout of the image data which are processed and transmitted over the fiber optic cable.	**$DA** defines the readout window.
Serial Skip #2	If the user-defined readout window does not end at the last column of the detector then some of the unwanted columns will be skipped at high speed.	The controller determines how many columns to skip based in the readout window defined in the **$DA** command. Some details of the serial skipping are specified by **$DD2** (data0)(data1)(data2).
Serial pre-read #2	As a transition from the high-speed skipping of Serial Skip #2 a number of columns are skipped at normal clocking speeds prior to the baseline pixel readout. After these pixels are shifted out there are no more actual image pixels left in the serial register.	**$DD1** (data3)(data4) set the number of pre-read serial transfers. See the description for Serial pre-read #1.
Serial baseline	These pixels are all baseline pixels.	**$DC** (data3)(data4) specifies the number of serial baseline pixels. These pixels use the same binning as the image pixels.

Event	*Notes*	*Command parameters*
Parallel baseline	After reading all of the rows that constitute the user defined readout window additions pseudo-rows can be transmitted. No parallel clocking is done for these rows of data. They just provide an alternate way to obtain baseline values.	**$DC** (data1)(data2) specifies the number of rows of parallel baseline. Since parallel clocking does not happen here the parallel binning does not matter.
Parallel skip #2	The remainder of the CCD image pixels are shifted out of the CCD.	
Special Erase	If a special erase procedure is selected it is done immediately following the CCD readout.	**$DC** (data5)(data6) specifies the duration spent in the special erase state.
Idle	Until the next exposure sequence the controller enters an idle state.	**$DC** (data7)(data8) specifies what happens during the idle state.

Table 2. Typical sequence of events in a non-frame-transfer exposure.

The controller sends messages to identify significant points in the exposure sequence. The messages are show in Table 3.

Message	*Event*
_ER	Erase begins
_EB	Exposure begins
_EE	Exposure ends
_RB	Readout begins
_RE	Exposure ends

Table 3. Messages sent by controller to mark significant exposure events.

These messages are transmitted on the RS232 line by the controller when the events occur. They are not responses to particular commands sent to the controller. Therefore, upper level software should be prepared to receive these messages at any time. **_ER** is sent at the start of normal erases. **_EB** is sent just before exposure timing is begun. **_EB** is sent even if the exposure time is 0 seconds. **_EE** is sent at the end of exposure timing and may overlap the beginning of readout . **_RB** is sent as image data are sent. **_RE** is sent

when image transmission is completed.

7. Digital Image Data

Each image produced by the controller consists of four elements:

- Start-of-image flag
- Image header
- Image pixel data
- End-of-image flag

The start-of-image flag allows the fiber optic computer interface to locate and synchronize on the start of a new image. This flag is consumed by the hardware and is never seen by the user. Likewise the end-of-image flag signals to the hardware that the last image pixel datum has been transmitted. The user sees only the image header and the image pixel data.

The image header is a fixed length block of data. The content of the header is described in Table 4. The digital image data immediately follows the header and consists of 16-bit unsigned integers. The exact format of the digital data depends on the manner in which the CCD detector is read out. Sufficient information is transmitted in the header to allow the complete reconstruction of the image from the transmitted data and to construct a FITS header that describes the image.

Word #	***Low Byte***	***High Byte***
0	Image geometry ID	Header size in bytes
1	Numeric image ID	
2	Columns per amplifier low byte	
3	Columns per amplifier high byte	
4	Rows per amplifier low byte	
5	Rows per amplifier high byte	
6	Exposure time low byte	
7	Exposure time middle byte	
8	Exposure time high byte	
9	Shutter state	
10	Overscan columns per amplifier low byte	
11	Overscan columns per amplifier high byte	
12	Overscan rows per amplifier low byte	
13	Overscan rows per amplifier high byte	
14	Column origin data window low byte	
15	Column origin data window high byte	
16	Row origin data window low byte	
17	Row origin data window high byte	
18	Data window columns low byte	
19	Data window columns high byte	
20	Data window rows low byte	
21	Data window rows high byte	
22	Column origin transmitted image low byte	
23	Column origin transmitted image high byte	
24	Row origin transmitted image low bytes	
25	Row origin transmitted image high bytes	

Table 4. Image header bytes defined.

As seen in Table 4, the header consists of twenty six 16-bit words. The high byte (most significant 8 bits) of the first word contains the number of bytes in the header (52). Upper-level software should use the size given in this byte rather than assume the header size, which is subject to change. All other words in the header contain 0 in the high byte of the word. The low byte of each word is interpreted as an unsigned 8-bit value.

8. Turning the Controller On

Whenever the controller is powered up it sends out the message **_IN**. Upper-level software can monitor for this command and can take appropriate power-up procedures. Typical power-up procedures may check all controller parameters and voltages (see Appendix II) against expected values that have been saved in a database. If all parameters and voltages match expected values then voltages and waveforms can be applied to the CCD. Otherwise some user intervention may be required. The current UCO/Lick software reads all of the DD parameters (**>DD1**, **>DD2**, **>DD3**, **>DD4**) and compares them to stored database values. If differences are found the user is given the option to make the database values match the controller values, make the controller values match the database values, or make no changes. This can be done on an individual parameter basis.

9. Reading the Detector

The commands described here are those sent to the controller as part of an exposure sequence. Some of the examples are based on the UCO/Lick guider software.

Start an exposure

To start an exposure sequence one command has to be sent to the controller. That command is one of those show in Table 5.

Command	*Action*
$RO	Do one full-frame exposure sequence and one readout.
$RC	Do continuous full-frame exposures and readouts.
$FO	Do one frame-transfer exposure and one readout.
$FC	Do continuous frame-transfer exposures and readouts.
$RP	Do one charge-shifting exposure and one readout.

Table 5. Readout commands.

The **$RO** command is used in typical instrumental applications while the **$FC** command is used in the UCO/Lick guider application. In the case of

$FC this command only has to be sent after the controller receives a parameter that actually changes the CCD clocking. For instance, one can change exposure times without sending a new **$FC** command. The other commands, **$RC**, **$FO**, and **$RP** are used only in unusual observing situations. **$RP** is a diagnostic readout mode.

A careful reading of Figure 3 and Table 2 would suggest that readout of the detector immediately follows the end of the exposure. In fact this is not necessarily the case. What happens following exposure depends on which of the commands in Table 6 was last sent to the controller.

Command	*Action*
$RI1	Begin readout of detector immediately after the end of exposure.
$RI0	Begin readout of the detector after receiving the **$RB** command.

Table 6. Commands that control when readout begins.

In the UCO/Lick guider we use **$RI1** and in most instrumental applications this may be the case too. One may need to use **$RI0** to delay readout if, for instance, one wanted to take multiple exposures before doing a final readout.

Sample Readout Commands

To start each exposure the UCO/Lick software sends a series of commands to the controller. These commands are shown in Table 7.

Command	*Action*
$RI1	Set controller to begin readout of the detector immediately after exposure end.
&RTD	Read temperature of detector.
&RTR	Read room temperature.
$DC	Define the secondary readout characteristics.
$GB	Define the amplifier gain and baselines.
$DT	Define exposure time and shutter state.
$DA	Define readout window.
$RO or **$FC**	Begin exposure and readout sequence.

Table 7. Typical sequence of commands to start an exposure sequence.

In the UCO/Lick software we always send these commands. However, if

none of the parameters for the **$DC**, **$GB**, **$DT**, or **$DA** commands has changed since the last time these were transmitted to the controller then there is no need to send them again. We send the **&RTD** and **&RTR** commands to capture the temperatures at the start of the exposure. These values will end up in the FITS header, but these commands are not required for readout.

Appendix I
Alphabetical List of User-Level Controller Commands

In this appendix we list many of the controller commands used in normal operation. In Appendix II we give details on the commands used to read and write voltages. In Appendix III we give details on the commands for monitoring and controlling temperatures. For each command we show the command and the number of parameters for the command. Then there is a short description of the command and a note on how to read back the current controller parameters (if applicable). Then there is a brief one-line listing of each parameter. Finally there is a detailed description of each parameter.

$AB

Abort an exposure.

When this command is sent to the controller any exposure sequence is immediately aborted. No readout of the CCD occurs. This command can be sent while image readout is in progress. Image data transmission stops immediately. Use the **$SE** command to stop an exposure and force CCD readout to begin.

$DA(data0)(data1)...(data19)

Define the primary readout parameters.
Use **>DA** to read back current controller parameters.

data0: Readout descriptor
data1: Image ID number
data2: DCS integration time
data3: binning
data4: start column (low)
data5: start column (high)
data6: start row (low)
data7: start row (high)
data8: number of unbinned columns to read (low)
data9: number of unbinned columns to read (high)
data10: number of unbinned rows to read (low)
data11: number of unbinned rows to read (high)
data12: user window start column (low)
data13: user window start column (high)

data14: user window start row (low)
data15: user window start row (high)
data16: user window number of columns (low)
data17: user window number of columns (high)
data18: user window number of rows (low)
data19: user window number of rows (high)

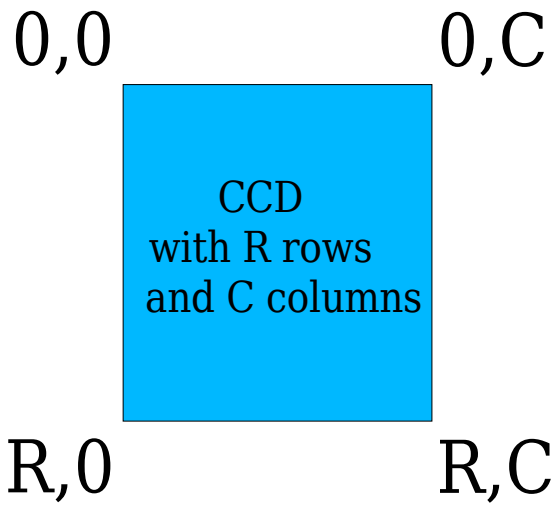

Figure 4. The naming of CCD amplifiers according to their nominal row and column position.

Details:

data0: Readout descriptor. This 1-byte parameter tells the controller which combination of CCD amplifiers to use during the readout. Assume there is a CCD with the geometry shown in Figure 4. The CCD has **R** rows and **C** columns and there is a readout amplifier at each corner. The amplifiers are named by their row and column coordinates, (row,column). So the four amplifiers are (0,0), (0,C), (R,0), and (R,C).

Given this geometry and naming convention, the meaning of data0 is shown in the following table.

Value	***Meaning***
0	Read entire CCD through (0,0)
1	Read entire CCD through (0,C)
2	Read entire CCD through (R,0)
3	Read entire CCD through (R,C)

Value	*Meaning*
4	Read CCD through (0,0) and (0,C)
5	Read CCD through (0,0) and (R,C)
6	Read CCD through (0,0) and (R,0)
7	Read CCD through (0,C) and (R,0)
8	Read CCD through (0,0), (0,C), (R,0), and (R,C)

Table 8. The definition of $DA *data0.*

Additional values for data0 may be defined later for other readout geometries.

data1: Image ID number. This number is assigned by the upper level software and is simply copied to the low byte of word 1 of the header of each image. In the UCO/Lick guider software this number is used by the image-reading software to associate the image with additional image data. For instance, the object name is effectively stored in a table which is indexed by the ID number and the image-reading software can match the image with the associated object name using the ID number.

data2: DCS integration time. This sets the double-correlated sampling amplifier integration time. Values from 1 to 255 are in units of 100 nanoseconds (ns). A value of 0 is taken to be 50ns. For each pixel the controller's does this:

- Reset the CCD amplifier (or amplifiers if reading more than 1).
- Measure the CCD output, taking an average for a time given by **data2**.
- Transfer the pixel's charge to the CCD amplifier.
- Measure the CCD output, taking an average for a time given by **data2**.
- Subtract the first measurement from the second to get the net signal.

Small values of **data2** produce fast readout while large values of data2 can result in lower noise. In the UCO/Lick guider software we typically provide three options, *Fast*, *Medium*, *and Slow*, corresponding to **data2** values of 10, 40, and 80.

data3: binning. This one-byte parameter encodes the row and column binning as shown in Figure 5.

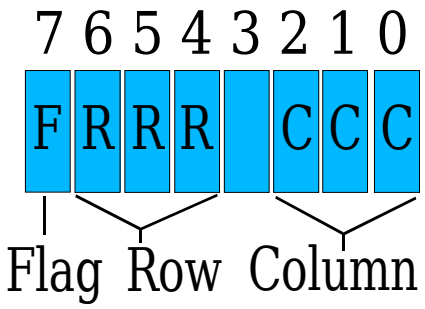

Figure 5. Row and column binning encoding.

If the flag bit (bit 7) is set to 1 then the lower 3 bits (bits 0, 1, and 2) specify the binning in columns and bits 4, 5, and 6 specify binning in rows. If the flag bit is set to 0 then bits 0, 1, and 2 specify both the row and column binning and the other upper bits are undefined and should be set to 0. The number of rows or columns binned is given by 2^{b} where ***b*** is the binning value.

data4,data5: start column. These bytes specify the first column of the CCD to be transmitted in the image. Column numbers start at 0.

data6,data7: start row. These bytes specify the first row of the CCD to be transmitted in the image. Row numbers start at 0.

data8,data9: number of columns. These bytes specify the number of CCD columns, prior to binning, that are to be read out of each amplifier of the CCD. The number of columns actually transmitted on the fiber cable will be the number of columns read divided by the binning in columns and multiplied by 2 if readout is through two amplifiers at opposite ends of the serial register. **(data8,data9)** should be an integer multiple of the binning in columns. For multiple amplifier readout (**data0** greater than 3) this parameter may be larger than the window defined by the user because the number of pixels read out through each amplifier must be the same. Parameters **data12** through **data19** are used to extract the user-specified data window from the transmitted image.

data10,data11: number of rows. These bytes specify the number of CCD rows, prior to binning, that are to be read out of the CCD. The number of rows actually transmitted on the fiber cable will be the number of rows

read divided by the row binning factor. (**data10,data11)** should be an integer multiple of the binning in rows. For multiple amplifier readout (**data0** greater than 3) this parameter may be larger than the window defined by the user because the number of pixels read out through each amplifier must the same. Parameters **data12** through **data19** are used to extract the user-specified data window from the transmitted image.

data12,data13: User window start column.
data14,data15: User window start row.
data16,data17: User window number of columns.
data18,data19: User window number of rows.
Because of constraints on the way CCDs are read out the actual number of rows or columns transmitted by the controller may be greater than the number of rows or columns requested by the user. These four parameters specify the part of the transmitted image actually requested by the user. These parameters are in actual transmitted rows and columns, irrespective of the on-chip binning. The start column, (data12,data13), and the start row, (data14,data15), are counted relative to the first transmitted row and column. So, for instance, start column 0 and start row 0 specifies the first transmitted pixel.

When reading a CCD through only one amplifier the four user window parameters are simple. In this case the start column and start row are always 0 and the number of columns and number of rows are the actual number of rows and columns transmitted.

When more than one amplifier is used to read a CCD then the parameter computations are more complex. Here is one example.

Suppose we have a 1000X1000 pixel CCD. The user wishes to read a window of 525 columns and 450 rows, beginning with CCD column 350 and row 200. We are reading the CCD through two amplifiers (**A** and **B**) on opposite ends of the serial register of the CCD. In the drawing the pixel at row 0 and column 0 is the closest pixel to amplifier **A**. For a little simplicity we set binning to 1 (no on-chip binning).

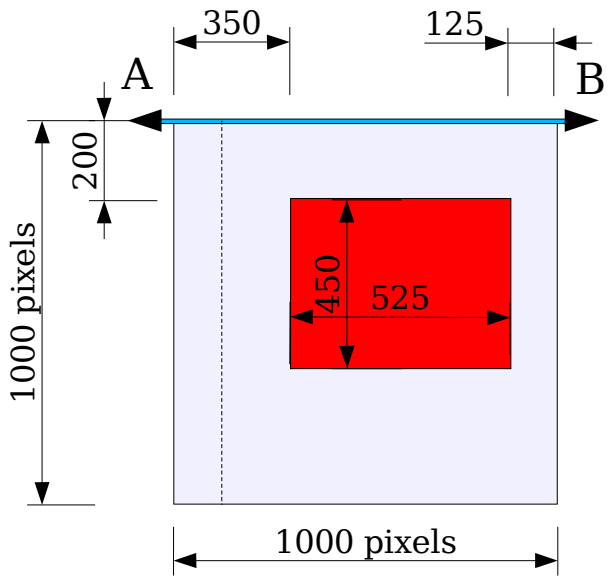

Figure 6. Example readout parameter computations.

Note that the user window is 350 columns over from amplifier **A** and 125 columns over from amplifier **B**. We can't tell the controller to skip 350 columns from amplifier **A** and 125 columns from **B** because each half of the CCD is clocked identically. We have to select a single amount to skip that applies to all amplifiers. If we skip over the first 350 columns from amplifier **A** then we skip over the first 350 columns closest to amplifier **B**. But that would be skipping over part of the image since the right edge of the user's window is only 125 columns away from amplifier **B**. Instead, we tell the controller to skip just 125 columns and we do this by setting the start column (**data4,data5**) to 125. The controller would then skip the first 125 columns from both amplifiers. This is exactly correct for amplifier **B** but leaves us with 225 extra pixels (350-125) from amplifier **A**. The extra pixels are discarded by setting the user window start column (**data12,data13**) to 225 and the user window number of columns to 525. Since each amplifier processes 500 pixels each and we are skipping 125 pixels from each amplifier, the number of columns to read per amplifier (**data8,data9**) is set to 375 (500-125).

Parameter bytes **data12** through **data19** are not used by the controller in any way. Instead they are simply copied to the header of the image. It is the responsibility of upper-level software to use these data to assemble the final image. In the example above each amplifier would produce 375 pixels of data for a total of 750 pixels. The upper level software

reassembles the image from the data stream and then uses the value of the user window start column (**data12,data13**) and discards the first 225 pixels in each row, leaving just the 525 pixels originally requested by the user.

Note that in this example the user window start row and user window number of rows is the same as in the single-amplifier case.

The following piece of pseudo-tcl code illustrates how to compute the readout and windowing parameters for several values of **data0**.

```
# Compute the $DA command readout and window parameters.
#   The user defined window is given by first_row, first_column and
#   number_of_rows, number_of_columns.

proc SetImageParameters {} {
# $bin is the binning factor (1, 2, 4, 8, etc) assumed same for rows and columns

#   sensor(idcode) controls which amplifiers are used. This is $DA command data0.

   switch $sensor(idcode) {

   0 -
   2 {
#      0 or 2 means left amplifier
#      The following four parameters define the portion of the CCD
#      to be read out and sent by the controller.

      set ReadoutParam(data4_5) $first_column
      set ReadoutParam(data6_7) $first_row
      set ReadoutParam(data8_9) $number_of_columns
      set ReadoutParam(data10_11) $number_of_rows

#      The 'window' defined by the following four parameters
#      describes the subsection of the data sent by the controller that
#      we actually want to save. The origin of the window (data12_13,data14_15)
#      is relative to (data4_5,data6_7) given above and for this case
#      is always (0,0) because we never read out more rows or
#      columns that we actually want to keep.

      set ReadoutParam(data12_13) 0
      set ReadoutParam(data14_15) 0
      set ReadoutParam(data16_17) [expr $number_of_columns/$bin]
      set ReadoutParam(data18_19) [expr $number_of_rows/$bin]
   }

   1 -
   3 {
#      1 or 3 means right amplifier

      set ReadoutParam(data4_5) \
         [expr $sensor(number_of_columns)-($first_column+$number_of_columns)]
      set ReadoutParam(data6_7) $first_row
      set ReadoutParam(data8_9) $number_of_columns
      set ReadoutParam(data10_11) $number_of_rows
      set ReadoutParam(data12_13) 0
      set ReadoutParam(data14_15) 0
      set ReadoutParam(data16_17) [expr $number_of_columns/$bin]
      set ReadoutParam(data18_19) [expr $number_of_rows/$bin]
    }
```

```
   4 {
#      4 means two amplifiers
#      With two amp readout, on opposite ends of the serial register
#      we have to construct a primary readout window that is
#      symmetric. We then use the window parameters (data12_13,data14_15) and
#      (data16_17,data18_19) to define which part of the primary readout window
#      the user really wants to keep.
#      Pixels we want skipped on the left.
       set leftskip $first_column
#      Pixels we want skipped on the right.
       set rightskip [expr $sensor(number_of_columns)-($first_column+$number_of_columns)]
#      Take the smaller of the two. This is the amount skipped for both amplifiers.
       if {$leftskip < $rightskip} {
         set skip $leftskip
       } else {
         set skip $rightskip
       }
       set ReadoutParam(data4_5) $skip
#      The number of columns we give the controller is the number of
#      columns per amplifier (2 amps in this case).
       set ReadoutParam(data8_9) [expr $sensor(number_of_columns)/2 - $skip]
#      The window origin for the final image is defined relative to
#      the primary readout.
       set ReadoutParam(data12_13) [expr ($leftskip - $skip)/$bin]
       set ReadoutParam(data16_17) [expr $number_of_columns/$bin]
#      The row parameters are just copied
       set ReadoutParam(data6_7) $first_row
       set ReadoutParam(data10_11) $number_of_rows
       set ReadoutParam(data14_15) 0
       set ReadoutParam(data18_19) [expr $number_of_rows/$bin]
     }
   }
```

$DC(data0)(data1)...(data8)

Define the secondary readout parameters.
Use >DC to read back the current controller parameters.

data0: MPP (0:nonmpp, 1:mpp)
data1: number of overscan rows (low)
data2: number of overscan rows (high)

data3: number of overscan columns (low)
data4: number of overscan columns (high)
data5: special erase time in 10ms units (low)
data6: special erase time in 10ms units (high)
data7: number of idle rows (low)
data8: number of idle rows (high)

Details:

data0: When set non-zero MPP clock levels are used during exposure. Otherwise normal clock levels are used. For CCDs that do no support MPP clock levels this parameter should be 0.

data1,data2: The number of overscan rows. These are additional rows generated after all real image rows have been read from the detector. These rows are in addition to the rows specified by the user-defined readout window. The image transmitted from the controller will include the user-defined window rows plus these additional overscan rows. The overscan rows are produced by clocking the serial register as normal but without any parallel clocking. Therefore the parallel binning used in the image rows is irrelevant for the overscan rows. In Figure 3 these rows are labeled “Parallel baseline” because the measured values should represent the digital value for no signal – the baseline.

data3,data4: The number of overscan columns. These are additional image columns read out after all real image columns have been read from the detector. These columns are in addition to the columns specified by the user-defined readout window. The image transmitted from the controller will include the user-defined window columns plus these additional overscan columns. The same serial binning is used to produce these overscan columns, but the number of overscan pixels transmitted in each row is equal to **(data3,data4)** regardless of the binning. In Figure 3 these columns are labeled “Serial baseline” because the measured values should represent the digital value for no signal – the baseline.

data5,data6: Some CCDs (LBNL and Lincoln) are best erased by placing the detector voltages in a special state. This parameter specifies how long (in 10 millisecond steps) that special state is maintained before the voltages are returned to normal operating levels. This parameter should be 0 if a special erase state is not used.

data7,data8: This two-byte value determines what happens during the idle state (see Figure 3). If it is 0 then all clocks maintain the same state they have during an exposure. If it is greater than 0 then this number of rows

are shifted out and the serials are clocked as if more rows are being read, except that no data are transmitted. For the special value 65535 (0xffff) row shifting continues indefinitely. This row-shifting idle state provides a way to continually flush the CCD between exposures. The idle state will terminate as soon as the next exposure sequence is started.

$DD1(data0)...(data4)

Low-level control parameters (set 1).
Use **>DD1** to read back the current parameters.

data0: serial clock period.
data1: Serial pre-read #1 transfers (low)
data2: Serial pre-read #1transfers (high)
data3: Serial pre-read #2 transfers to read before overscan (low)
data4: Serial pre-read #2 transfers to read before overscan (high)

Details:

data0: This value selects the fundamental serial clocking period from a set of fixed values.

data0 value	*Serial clock period*
0	800 ns
1	400 ns
2	100 ns
3	50 ns

*Table 9. **DD1** (data0) selects the fundamental serial clock period.*

The actual serial clock voltage waveforms produced by the controller result from applying this fundamental clocking period to a waveform table stored in the controller.
data1,data2: Serial pre-read #1 transfers (see Figure 4). To reach the first column of the readout window defined by the **$DA** command the controller may skip some pixels in each row at high speed. As a transition between this high-speed skipping and the readout window a specified number of serial transfers are done at normal clocking speed and using the image pixel binning just prior to the readout window. (**data1,data2)** sets the number of transfers. This number can be set to any value but will be internally adjusted to a smaller value if the readout window is such that there aren't **(data1,data2)*binning** columns to skip. A typical value is 4.
data3,data4: Serial pre-read #2 transfers (see Figure 3). If a baseline in each image row is being measured (true if **$DC** (data3,data4) is non-zero)

then the controller may skip some pixels in each row at high speed to flush out the remaining image pixels in each row. As a transition between this high-speed skipping and the baseline pixels a specified number of serial transfers are done at normal clocking speed and normal serial binning just prior to the baseline pixels. (data3,data4) sets the number of serial transfers. This number can any value from 0 to 65535. A typical value is 4.

$DD2(data0)...(data7)

Low-level control parameters for serial clocks, parameters for serial cleaning sequences, and serial line capacitors. Also sets DCS integration capacitor.

Use **>DD2** to read back the current parameters.

data0: Serial skipping clocking selection.
data1: Binning for serial skipped pixels
data2: DCS sample time for serial skipped pixels.
data3: Serial cleaning clocking selection.
data4: Binning for serial cleaning.
data5: DCS sample time during serial cleaning.
data6: DCS capacitor selection.
data7: Controls attachment of capacitors to serial clocks.

Details:

data0: Serial skipping clocking selection. (See Figure 3) This parameter is a flag that affects the serial clocking during Serial Skipping #1 and Serial Skipping #2. If the value is 0 then the normal serial clocks are used. If the value is 1 then the following values, data1 and data2, are used to modify the serial clocks.
data1: This is the binning. The number of pixels binned during Serial Skipping is 2 raised to the value of data1.
data2: This is the double correlated sampling (DCS) sample time for Serial Skipping in 0.1 microsecond units.
data3: Serial flushing parameters. (See Figure 3) During Parallel Skipping #1 and Parallel Skipping #2 the serial register is also run to flush charge accumulated in the serial register. This parameter is a flag that affects the serial clocking behavior during the serial flushing. If the value is 0 then the normal serial clocks are used. If the value is 1 then the following values, data4 and data5, are used to modify the serial clocks.
data4: Binning for serial flushing. The number of pixels binned during

Serial cleaning is 2 raised to the value of data4.
data5: DCS sample time during serial flushing in 0.1 microsecond units.
data6: DCS capacitor selection. The controller's Double Correlated Sampling amplifier can use one of two capacitors for charge averaging. A small capacitor provides higher system gain but smaller dynamic range while a larger capacitor provides lower gain but a larger dynamic range. Parameter *data6* selects the capacitor to use.

data6 value	***capacitor***
0	Small capacitor
1	Large capacitor
2	Autoselect using big capacitor if binning >1 or DCS time >3.9 microseconds (**$DA** data2 >39).

*Table 10. **DD2** (data6) selects the DCS capacitor.*

data7: Capacitors can be attached to the serial clocks to slow down the rise- and fall-times of the clocks. Parameter data7 controls attachment of capacitors to serial clocks.

data 7 value	***capacitor***
0	No capacitors connected
1	Capacitors connected
2	Connect capacitors if serial clock period is 1 microsecond (**$DD1** data0 is 0)

*Table 11. **DD2** (data7) selects the serial clock capacitors.*

$DD3(data0)...(data4)

Low-level control parameters for parallel clocks.

Use **>DD3** to read back the current parameters.

data0: Select fundamental parallel clock period.
data1: Parallel clock binning during normal erase (low byte).
data2: Parallel clock binning during normal erase (high byte).
data3: Number of parallel pre-read rows. (low byte).
data4: Number of parallel pre-read rows. (high byte).

Details:

data0: This value selects the fundamental parallel clocking period from a set of fixed values.

data0 value	*Parallel clock period*
0	40 µs
1	10 µs
2	1 µs
3	0.4 µs

*Table 12. **DD3** (data0) selects the parallel clock period.*

The actual parallel clock voltage waveforms produced by the controller result from applying this fundamental clocking period to a waveform table stored in the controller.
data1,data2: Parallel clock binning during normal erase. The number of rows binned is 2 raised to the value of (data1,data2). A typical value is 5.
data3,data4: To reach the first row of the readout window defined by the **$DA** command the controller may skip some rows at high speed. As a transition between this high-speed skipping and the readout window a specified number of pseudo-rows are read at normal serial clocking speed just prior to the readout window. (data3,data4) sets the number of pseudo-rows. These are called pseudo-rows because the parallel clocks are not used during this interval. This number can be set to any value. A typical value is 4.

$DD4(data0)...(data9)

Low-level control parameters for parallel clocks.

Use **>DD4** to read back the current parameters.

data0: Controls Parallel skip #2.
data1: Binning for special erasing (low byte).
data2: Binning for special erasing (high byte).
data3: Controls the substrate voltage rise-time.
data4: Controls parallel clock levels during special erasing.
data5: Controls the connection of capacitors to the parallel clocks.
data6: LBNL CCD substrate voltage during exposure (low byte).
data7: LBNL CCD substrate voltage during exposure (high byte).
data8**:** LBNL CCD substrate voltage during special erase sequence (low byte).
data9**:** LBNL CCD substrate voltage during special erase sequence (high

byte).

Details:

data0: Controls Parallel skip #2. (See Figure 3) After the user defined readout window is read out additional rows can be skipped at high speed. This parameter controls this final row skipping. A value of 0 means no skipping is done. A value of 1 means skipping is done using same parameters as Parallel skip #1.
data1,data2: Binning for special erasing. If a special erase is being done and the erase involves clocking the parallel clocks this value specifies how many rows are binned. The number of rows binned is 2 raised to the power of (data1,data2).
data3: LBNL special erasing requires lowering the substrate voltage and then raising it up to its original value slowly. The parameter controls the substrate voltage rise-time by setting the number of voltage steps used to return to the original voltage. Each step takes a fixed interval of time, so a value of 0 produces the fastest rise time and a value of 255 produces the slowest rise time. The actual duration is measured using an oscilloscope.
data4: Controls parallel clocks during special erasing according to the following table.

data4 value	***Parallel clocks***
0	Use normal parallel clocks
1	All parallel clocks are high
2	All parallel clocks are low

*Table 13. **DD4** (data4) values select state of parallel clocks during special erase.*

data5: This parameter controls the connection of wave-shaping capacitors to the parallel clocks as shown in the following table.

data5 value	***capacitor***
0	No capacitors connected
1	Capacitors connected
2	Capacitors connected when parallel clock period is 40 µs (**$DD3** data0 is 0)

*Table 14. **DD4** (data5) codes to select parallel clock capacitors*

data6,data7: When an LBNL high-resistivity CCD is used this two-byte

parameter specifies the LBNL CCD substrate voltage during exposure in units of 0.1 volts. For other CCDs these bytes are undefined and can be set to 0.

data8,data9: When an LBNL high-resistivity CCD is used this two-byte parameter specifies the LBNL CCD substrate voltage during the special erase sequence. The units are 0.1 volts. For other CCDs these bytes are undefined and can be set to 0.

$DE(data0)(data1)(data2)

Define the number of erases prior to exposure start.
Use **>DE** to read back the current parameters.

data0: number of normal erases (low)
data1: number of normal erases (high)
data2: reverse erase flag. 1 means do reverse erase, 0 means don't do reverse erase.

Details:

data0,data1: The number of normal erases. If this number is 0 then no normal erases are done. Each erase produces enough parallel clocking to transfer all rows out of the detector.

data2: 1 means do reverse erase, 0 means don't do reverse erase. A reverse erase runs the parallel clocks such that the charge is shifted in the opposite direction from normal. This type of erase is useful in some circumstances but is not appropriate for all types of CCDs. If a reverse erase is done it is done once, after all of the normal erases.

$DP(data0)(data1)(data2)(data3)

Define the “polarimetry” charge shifting parameters.
Use >DP to read back current controller parameters.

data0: Number of times to shift charge (low byte).
data1: Number of times to shift charge (high byte).
data2: Number of rows to shift (low byte).
data3: Number of rows to shift (high byte).

$DT(data0)(data1)(data2)(data3)

Exposure time (10 ms units) and state of shutter during exposure.
Use **>DT** to read back current controller parameters.

data0: exposure time(low)
data1: exposure time(mid)

data2: exposure time(high)
data3: 0:shutter close/1:shutter open

Details:

data0,data1,data2: The exposure time is a three-byte quantity in units of 0.01 seconds. Therefore the maximum possible exposure time is $(2^{24}-1)*0.01=167772.15$ seconds (46.6 hours).
data3: 0 means the shutter remains closed during the exposure time and 1 means the shutter is opened during the exposure time. What physically happens is that a TTL-level signal is supplied by the controller. A high TTL level means the shutter should open and a low TTL level means the shutter should close. What actually happens with this TTL signal is up to external hardware. No confirmation of shutter open/close is provided by the controller.

>EC(n)

Control the analog switches on the CDB.

(n) can be either ascii 0 or 1. The clock driver board (CDB) has a built-in set of analog switches to disconnect the clocks generated on the board from the detector. **>EC0** opens the analog switches and disconnect the clocks and **>EC1** closes the analog switches. This command does not control any analog switches for bias voltages. That must be done with external switches and the **>EV(n)** command. If there is more than one CDB they all react to this command simultaneously.

>EV(n)

Control signal for external analog switches.

(n) can be either ascii 0 or 1. **>EV0** will set the signal to a TTL low state. **>EV1** will set the signal to a TTL high state. The signal controlled is pin 1 of the 8-bit parallel output byte provided by the controller. See the Technical Reference Manual for details.

$FC

Run the controller in continuous frame-transfer mode. This command is sent to the controller to run in a typical guider configuration. In continuous frame-transfer mode new exposure sequences are started automatically. A start exposure command, **$ST**, does not have to be issued. Before sending **$FC** the following commands would normally be sent: **$DE**, **$DT**, **$GB**, **$DA**, and **$DC**. Note there is a similar command, **$RC**, for non-frame-transfer readout.

$FO

Run the controller in single frame-transfer mode. In single frame-transfer mode a new exposure sequence is started when the **$ST** start exposure command is sent to the controller. Before sending **$FO** the following commands would normally be sent: **$DE**, **$DT**, **$GB**, **$DA**, and **$DC**. Note there is a similar command, **$RO**, for non-frame-transfer readout.

$GB(data0)(data1)(data2)(data3)(data4)

Set the gain and baseline.

Use **>GB** to read back the current controller parameters.

data0: gain for all amplifiers.
data1: baseline offset for amplifier A (low byte).
data2: baseline offset for amplifier A (high byte).
data3: baseline offset for amplifier B (low byte).
data4: baseline offset for amplifier B (high byte).

Details:

data0: Legal values for the gain are 0 through 3. 0 is the lowest gain and 3 is the highest gain. The gain applies to all amplifiers. Note that there is also a command **$GN(data0)** that can be used to set just the gain. Also, the baseline offsets can be set individually with the **>OA** and **>OB** commands.

data1,data2: This two-byte parameter sets the baseline offset for amplifier A. The baseline offset is added to the signal in the amplifier signal chain and therefore sets the digital signal value corresponding to zero-signal.

data3,data4: This is the same as (data1,data2) except it applies to amplifier B.

$GN(data0)

Set amplifier gain.

data0: Legal values for the gain are 0 through 3. 0 is the lowest gain and 3 is the highest gain. The gain applies to all amplifiers. Note that there is also a command **$GB** that can be used to set the gain and baseline offset at the same time.

Details:

This command sets the same parameter as (data0) of the **$GB** command.

>ID

Tell the controller to send the controller ID number.

When this command is sent the controller responds with _CID(NN) where (NN) is a two-character hexadecimal value. This is the ID number defined by a set of jumpers in the controller.

>OA(n)

Baseline offset for amplifier A.

(n) is the 4-character hexadecimal value to use for the baseline offset for video channel A. Example: >OA03f9

>OB(n)

Baseline offset for amplifier B.

(n) is the 4-character hexadecimal value to use for the baseline offset for video channel B. Example: >OB0f2a

>PT

Tell the controller to send the remaining exposure time.

When this command is sent the controller responds with **_EXT**(EEE) where (EEE) is a 6-character hexadecimal number representing the elapsed exposure in 0.01 second units.

>PU1

Pause exposure.

When this command is sent any exposure in progress is paused. This means the shutter is closed and the exposure time is stopped.

>PU0

Resume exposure.

When this command is sent an exposure in progress that was already paused will be resumed. This means the shutter is opened (if **$DT(data3)** is 1) and the exposure time begins counting again.

$RB

Read out detector.

This command tells the controller to beginning the detector readout following an exposure. Use this command only in conjunction with the **$RI0** command.

$RC

Run the controller in non-frame-transfer mode continuously. In continuous non-frame-transfer mode new exposure sequences are started automatically. A start exposure command, **$ST**, does not have to be issued. Before sending **$RC** the following commands would normally be sent: **$DE**, **$DT**, **$GB**, **$DA**, and **$DC**. Note that there is a similar command, **$FC**, to run the controller in a typical guider application using frame-transfer.

$RI0

Readout detector when commanded with $RB command.

This command tells the controller to wait for the $RB command before beginning the detector readout following an exposure.

$RI1

Readout detector immediately following the end of an exposure.

This command tells the controller to beginning the detector readout immediately following an exposure. The **$RB** command is not needed to begin the readout.

$RO

Run the controller in single non-frame-transfer mode. In single non-frame-transfer mode a new exposure sequence is started when the **$ST** start exposure command is sent to the controller. This is the typical mode for running a CCD non-guider applications. Before sending **$RO** the following commands would normally be sent: **$DE**, **$DT**, **$GB**, **$DA**, and **$DC**.

$RP

Run the controller in single “polarimetry” mode.

This is the charge shifting mode used to test for low-level traps. When the **$ST** command is issued a single non-frame-transfer charge shifting exposure sequence will be initiated. Before sending **$RP** the following commands would normally be sent: **$DE**, **$DT**, **$GB**, **$DA**, **$DC**, and **$DP**.

$SE

Stop exposure.

This command tells the controller to stop an exposure immediately and

begin the readout sequence (see Figure). To stop an exposure and not read out the abort command, **$AB** should be used.

>SL

Put the CPU to sleep

This command puts the controller's timing board CPU to sleep. If the CPU is running during CCD readout noise is introduced into the resulting image. Therefore it is important to make sure the CPU is not running during an exposure. This command can be used to make sure the CPU is asleep. The CPU sends the normal **OK** response before going to sleep. Note that this command puts only the timing board CPU to sleep. The temperature control CPU never sleeps.

Except during image readout the timing board CPU does not have to be asleep. In the UCO/Lick guider, upper level software does not send anything to the timing board CPU (like requests for remaining exposure time) if the remaining exposure time is less than 0.4 second.

Appendix II
Voltage Commands

Reading or writing controller voltages is one of the most complex aspects of the controller because the number of different voltages is large. The controller knows about three types of voltages: clocks, biases, and raw power supply voltages. Clocks typically change between two programmable levels. Biases have a single programmable level. Power supply voltages can be monitored but not changed under software control.

Voltage type	***Description***	***Read Command***	***Write command***
CLOCK A	Main clocks	RVCA	WVCA/WDCA
CLOCK B	Alternate clocks	RVCB	WVCB/WDCB
BIAS 0	Main bias	RVB0	WVB0/WDB0
BIAS 1	Alternate bias	RVB1	WVB1/WDB1
POWER	Main controller power	RVP	read only
TCBPOWER	Temperature controller power	RVPB	read only

Table 15. Commands for reading or writing controller voltages.

The controller maintains parameters for six sets of voltages. These voltage sets are described in Table 15. CLOCK A is the set of voltages that defines the high and low levels for all of the normal clocks produced by the controller. These typically include the parallel and serial clocks as well as the transfer gate, reset gate and summing well. CLOCK B is an alternate set of voltage specifications for the normal controller clocks and are used during special erase clocking sequences. BIAS 0 and BIAS 1 define the levels for two independent sets of bias voltages. Bias voltages are generated on the controller's Signal Processing Board (SPB) and the SPB has a jumper that determines whether the board corresponds to BIAS 0 or BIAS 1 voltages. The controller can support two SPBs, so with two SPBs both sets of BIAS voltages can be used if needed. POWER is the set of raw power supply voltages used by the controller and TCBPOWER is the set of raw power supply voltages used by the temperature control board within the controller. The POWER and TCBPOWER voltages are read-only.

Table 15 lists a command for reading each of the voltage sets and two

commands for writing the voltage sets that are not read-only. Reading and writing is defined here from the viewpoint of the upper-level software running the controller. To read a voltage means to ask the controller to return the current value of a voltage. To write a voltage means to send the controller a command that will change the actual value of a voltage produced by the controller. All of the voltage commands use the > beginning of command (BOC) character and this means that all parameters are ASCII.

Special note: Reading or writing clock voltages also causes the controller to stop running the clocks. Therefore reading of voltages must not be done during image readout.

For each voltage set there are two commands for writing voltages. The difference in the two commands is the way in which the voltages are specified. The first form (the ones that start with "WV") express volts in numerical D-to-A converter units, ranging from **0** to hexadecimal **fff** while the second form (the ones that start with "WD") express volts in true voltage units. To use the second form the controller voltages must be calibrated so the controller can convert volts to D-to-A units. This voltage calibration is a normal part of initial controller configuration.

Each of the three types of voltages (clock, bias, and power) have a number of voltages. Table 16 lists the 16 clock voltages, table 17 list the 12 bias voltages, Table 18 lists the power supply voltages and Table 19 list the temperature control board voltages.

The voltages ID numbers shown in Table 16 are combined with the appropriate commands from Table 15 to construct the specific commands to read or write clock voltages. For instance a command to set the Set A Parallel Phase 3 Imaging high and low voltages would be something like this:

```
>WDCA20 5.0 -5.0
```

or

```
>WVCA20 0c00 03ff
```

The first form uses true voltage units and the second form uses D-to-A units which in this case assumes D-to-A value 0 corresponds to -10v and D-to-A value 0xfff corresponds to +10v.

Voltage	*Voltage ID Number (hexadecimal)*	*CDB J2 Output Pin Number*
Parallel Phase 1 Imaging	0	2
Parallel Phase 2 Imaging	10	3
Parallel Phase 3 Imaging	20	4
Transfer Gate B	30	5
Parallel Phase 1 Storage	40	6
Parallel Phase 2 Storage	50	7
Parallel Phase 3 Storage	60	8
Transfer Gate A	70	9
Serial Phase 1	80	12
Serial Phase 2R	90	13
Serial Phase 3R	a0	14
Summing Well	b0	15
Serial Phase 2L	c0	16
Serial Phase 3L	d0	17
Reset Gate	e0	18
Spare	f0	19

Table 16. The clock type voltages.

The voltages ID numbers shown in Table 16 also correspond to specific output pins on the controller's Clock Driver Board (CDB). The actual uses of those output pins, as described in the table are the ones used in the UCO/Lick guide cameras. The pin assignments are listed in Table 16 and are discussed in the UCAM Technical Reference Manual.

Descriptions of the CLOCK voltage command formats are as follows.

>RVCA or >RVCB

>cmnd(nn)

For **cmnd** substitute either **RVCA** or **RVCB**. For (nn) substitute the two-character hexadecimal number from Table 16. There is no space between **cmnd** and (nn). The response has this form:
_(VCX) (nn) (name) (dacp) (adcp) (vp) (dacn) (adcn) (vn) where:

(VCX) is either VCA for **RVCA** or VCB for **RVCB**,

(nn) is the voltage number,
(name) is the voltage name,
(dacp) is the D-to-A value used for the positive rail of the clock,
(adcp) is the A-to-D value obtained when reading the positive rail voltage,
(vp) is the positive clock rail in true volts,
(dacn) is the D-to-A value used for the negative rail of the clock,
(adcn) is the A-to-D value obtained when reading the negative rail voltage,
(vn) will be negative clock rail in true volts.

Example:

>RVCA20 returns `_VCA 20 VVI3 0C00 3830 +05.0v 03FF C630 -05.0v`

Details:

(nn) This is the same hexadecimal number used in the original **>RVCA** or **>RVCB** command.

(name) The name is only an aid for controller developers to remind them of the use for the particular signal. It is not guaranteed to be a valid name is all applications of the controller. There is no way to change these names via a controller command (*but this may be added soon*).

(dacp) This is the D-to-A value used in set the D-to-A that controls the positive rail of the clock. This is a 12-bit quantity and the first character of this 4-byte hexadecimal number is always 0.

(adcp) This is the A-to-D value obtained when reading the voltage. This is a 12-bit quantity and the least significant character of this 4-byte hexadecimal number is always 0.

(vp) This is the voltage reading converted to true volts. Note the format of the true voltage. There is always a + or – sign, two digits to the left of the decimal point (0 filled if necessary), one digit to the right of the decimal point, and then the letter 'v'.

(dacn) This is the D-to-A value used in set the D-to-A that controls the negative rail of the clock. This is a 12-bit quantity and the first character of this 4-byte hexadecimal number is always 0.

(adcn) This is the A-to-D value obtained when reading the voltage. This is a 12-bit quantity and the least significant character of this 4-byte hexadecimal number is always 0.

(vn) This is the negative voltage reading converted to true volts. Note the format of the true voltage. There is always a **+** or **–** sign, two digits to the left of the decimal point (0 filled if necessary), one digit to the right of the decimal point, and then the letter 'v'.

Each clock voltage is routed through a multiplexer to a 12-bit A-to-D converter to obtain the true voltage reading. The true voltage is a measurement of the voltage at the output connector of the CDB. The CDB has analog switches to isolate the clocks from the detector. The voltage is measured after these analog switches. If there are analog switches external to the CDB the controller does not have a mechanism for reading the voltages following those switches.

A special form of the commands, **>RVCAff** or **>RVCBff**, tells the controller to return all 16 clock voltages as if they had been requested individually.

>WVCA or >WVCB

>cmnd(nn) (hhhh) (llll)

For **cmnd** substitute either W**VCA** or W**VCB**. For (nn) substitute the two-character hexadecimal number from Table 16. There is no space between **cmnd** and (nn). (hhhh) is a 4-character hexadecimal D-to-A value for the high rail of the clock and (llll) is the 4-character hexadecimal D-to-A value for the low rail of the clock. Note that the first character of these values is always 0 since the largest possible D-to-A value is currently hexadecimal fff. The response for these commands is **OK**.

>WDCA or >WDCB

>cmnd(nn) (h) (l)

For **cmnd** substitute either WD**CA** or WD**CB**. For (nn) substitute the two-character hexadecimal number from Table 16. There is no space between **cmnd** and (nn). (h) is the value for the high rail of the clock in volts and (l) is the value for the low rail of the clock in volts. The format for (h) and (l) must be followed exactly. The format is a + or – minus sign, two characters to the left of the decimal point, the decimal point, and one character to the right of the decimal point. For example, five volts is +05.0. There should be a single space between (nn) and (h) and between (h) and (l). The response for these commands is **OK**.

Voltage	*Voltage ID Number (hexadecimal)*	*SPB J2 Output Pin Number*
Negative Amplifier Drain A	0	10
Negative Amplifier Drain B	10	11
Negative Reset Drain	20	12
(*Not connected*)	30	NC
Video offset	40	NC
N+ Substrate	50	14
Output Gate	60	13
Negative Substrate	70	6
Positive Amplifier Drain A	80	2
Positive Amplifier Drain B	90	3
Positive Reset Drain	a0	4
(*Not connected*)	b0	NC

Table 17. Bias type voltages.

The voltages ID numbers shown in Table 17 also correspond to specific output pins on the controller's Signal Processing Board (SPB). Note that there are currently nine bias voltages brought out to connector J2. The actual uses of those output pins, as described in the table are the ones used in the UCO/Lick guide cameras. The pin assignments are listed in Table 17 and are discussed in the UCAM Technical Reference Manual. The video offset voltage (code 40) is used by the video processing circuitry on the SPB and does not come to the J2 connector on the SPB.

Descriptions of the BIAS voltage command formats are as follows.

>RVB0 or >RVB1

>cmnd(nn)

For **cmnd** substitute either **RVB0** or **RVB1**. For (nn) substitute the two-character hexadecimal number from Table 17. There is no space between **cmnd** and (nn). The response has this form:
`_(VBX) (nn) (name) (dac) (adc) (v)` where:

(VBX) will be either RVB0 or RVB1,

(name) will be the voltage name,
(dac) will be the D-to-A value,
(adc) is the A-to-D value obtained when reading the voltage,
(v) will be the voltage in true volts.

Example:

>RVB120 returns _RVB1 20 -VR 0057 C520 -06.0v

Details:

(nn) This is the same hexadecimal number used in the original >**RVB0** or >**RVB1** command.

(name) The name is only an aid for controller developers to remind them of the use for the particular signal. It is not guaranteed to be a valid name is all applications of the controller. There is no way to change these names via a controller command.

(dac) This is the D-to-A value used in set the D-to-A that controls this voltage. This is a 12-bit quantity and the first character of this 4-byte hexadecimal number is always 0.

(adc) This is the A-to-D value obtained when reading the voltage. This is a 12-bit quantity and the least significant character of this 4-byte hexadecimal number is always 0.

(v) This is the voltage reading converted to true volts. Note the format of the true voltage. There is always a + or – sign, two digits to the left of the decimal point (0 filled if necessary), one digit to the right of the decimal point, and then the letter 'v'.

Each clock voltage is routed through a multiplexer to a 12-bit A-to-D converter to obtain the true voltage reading. The true voltage is a measurement of the voltage at the output connector of the SPB. The SPB has no analog switches to isolate the bias voltages from the detector. If there are analog switches external to the SPB the controller does not have a mechanism for reading the voltages following those switches.

A special form of the commands, >**RVB0ff** or >**RVB1ff**, tells the controller to return all 9 bias voltages as if they had been requested individually.

>WVB0 or >WVB1

>cmnd(nn) (vvvv)

For **cmnd** substitute either W**VB0** or W**VB1**. For (nn) substitute the two-character hexadecimal number from Table 17. There is no space between **cmnd** and (nn). (vvvv) is a 4-character hexadecimal D-to-A value for the bias voltage. Note that the first character of these values is always 0 since the largest possible D-to-A value is currently hexadecimal fff. The response for these commands is **OK**.

>WDB0 or >WDB1

>cmnd(nn) (v)

For **cmnd** substitute either W**VD0** or W**VD1**. For (nn) substitute the two-character hexadecimal number from Table 17. There is no space between **cmnd** and (nn). (v) is the bias voltage in volts. The format for (v) must be followed exactly. The format is a + or – minus sign, two characters to the left of the decimal point, the decimal point, and one character to the right of the decimal point. For example, 8.2 is +08.2. There should be a single space between (nn) and (v). The response for these commands is **OK**.

Voltage	***Voltage ID Number (hexadecimal)***
+48	0
+30	10
+15	20
+5	30
-5	40
-15	50
-30	60
+12	70
-12	80
VCC (+5)	90

Table 18. Power supply voltages

The ID numbers shown in Table 18 are combined with the >RVP command to read the various power supply voltages used in the controller.

>RVP(nn)

Read power supply voltage (nn). The response is:

_RVP (nn) (name) (adc) (v) where:

(nn) is the voltage ID number,
(name) is the nominal name of the voltage,
(adc) is the A-to-D digital number resulting from reading the voltage,
(v) is the voltage in volts.

Example: >**RVP50** produces `_RVP 50 -15V 8000 -14.6v`

Details:

(nn) This is the same hexadecimal number used in the original >**RVP** command.

(name) The name is only an aid for controller developers to remind them of the use for the particular signal. It is not guaranteed to be a valid name is all applications of the controller. There is no way to change these names via a controller command.

(adc) This is the A-to-D value obtained when reading the voltage. This is a 12-bit quantity and the least significant character of this 4-byte hexadecimal number is always 0.

(v) This is the voltage reading converted to true volts. Note the format of the true voltage. There is always a + or – sign, two digits to the left of the decimal point (0 filled if necessary), one digit to the right of the decimal point, and then the letter 'v'.

Voltage	***Voltage ID Number (hexadecimal)***
+15	0
-15	10
+5	20

Table 19. Temperature control board power supply voltages.

The ID numbers shown in Table 19 are combined with the >RVPB command to read the various temperature control board power supply voltages.

>RVPB(nn)

Read temperature controller power supply voltage (nn). The response is:

_RVPB (nn) (name) (adc) (v) where:

(nn) is the voltage ID number,
(name) is the nominal name of the voltage,
(adc) is the A-to-D digital number resulting from reading the voltage,
(v) is the voltage in volts.

Example: >**RVPB20** produces `_RVPB 20 +5VB 7FF0 +06.3v`

Details:

(nn) This is the same hexadecimal number used in the original >**RVPB** command.

(name) The name is only an aid for controller developers to remind them of the use for the particular signal. It is not guaranteed to be a valid name is all applications of the controller. There is no way to change these names via a controller command.

(adc) This is the A-to-D value obtained when reading the voltage. This is a 12-bit quantity and the least significant character of this 4-byte hexadecimal number is always 0.

(v) This is the voltage reading converted to true volts. Note the format of the true voltage. There is always a + or – sign, two digits to the left of the decimal point (0 filled if necessary), one digit to the right of the decimal point, and then the letter 'v'.

Appendix III
Temperature Monitoring and Control

Features

The controller has a separate CPU to monitor and control the detector temperature. This CPU is electrically isolated from the main controller, with its own power supplies and is essentially a separate subsystem. Although the temperature control CPU and the main controller CPU share the same RS232 line, optical isolators are used to keep them electrically separated.

In this appendix we describe the commands for reading temperatures and controlling temperatures. Temperatures are measured with calibrated 1N914 diodes. Control is achieved by supplying current to a resistor and the heat generated balances the cooling capacity of the cooling hardware. Additional

calibration commands are described in the **UCAM Technical Reference Manual**.

There are two temperatures available from the controller, the detector temperature and “room” temperature. Room temperature is actually a measure of the temperature of a diode mounted on the controller backplane.

Monitoring

The controller's power supply chassis has a temperature readout meter on it. Either the detector temperature or the room temperature can be displayed with one of the following commands.

&TD0

Display room temperature.

&TD1

Display detector temperature.

Reading the temperatures is done with the following commands.

&RTD

Read detector temperature.

The response is: `_RTD (adc) (tttt)` where (adc) is the hexadecimal A-to-D value and (tttt) is the temperature in degrees.

Example: `_RTD 6FF0 -100.0`

Details:

(adc) This is the value read from the 12-bit A-to-D converter. The result is transmitted as four hexadecimal digits with the least significant digit set to 0.
(tttt) The temperature in degrees format is a + or – sign, three digits to the left of the decimal point (0 filled if necessary) and one digit to the right of the decimal point.

&RTR

Read room temperature.

The response is: `_RTR (adcn) (tttt)` where (adcn) is the 16-bit hexadecimal A-to-D value and (tttt) is the temperature in degrees.
Example: `_RTR EFF0 +020.0`

Details:

(adc) This is the value read from the 12-bit A-to-D converter. The result is transmitted as four hexadecimal digits with the least significant digit set to 0.
(tttt) The temperature in degrees format is a + or – sign, three digits to the left of the decimal point (0 filled if necessary) and one digit to the right of the decimal point.

Control

The temperature controller uses simple proportional control where the current applied to the heating resistor is proportional to the error. The error is computed from a selected target temperature, T_t. If the temperature of the sense diode, T_d, is equal to or higher than the target temperature then the error is 0. If the temperature of the sense diode is less than the target temperature then the error is the difference (T_t-T_d). The calibration of the proper scaling factor is discussed in the **UCAM Technical Reference Manual**. To set the desired target temperature the following command is used.

&WTT (ttt)

Write target temperature.

(ttt) is the target temperature, T_t in degrees C. (ttt) has a fixed format. First there must be a + or – minus sign, then three digits to the left of the decimal point, the decimal point, and one digit to the right of the decimal point. Thus -30 would be formatted as **`-030.0`** and +10 would be formatted at **`+010.0`**. There is a single space between **&WTTT** and (ttt).

To read the current target temperature from the controller the following command is used.

&RTT

Read target temperature.

The response to this command is _RTT ttt where ttt is the temperature in degrees C. The temperature has a fixed format. First there is be a + or – minus sign, then three digits to the left of the decimal point, the decimal point, and one digit to the right of the decimal point. Thus -90.3 would be sent as **`-090.3`**.

Appendix IV
RTS/CTS Control

The controller has a timing board CPU which runs asynchronously relative to the CCD readout sequences. As a result it introduces digital noise into the image data if it is running during image readout. To avoid the added noise the CPU puts itself to sleep immediately prior to readout. In sleep mode the CPU is halted. This includes the RS232 send/receive hardware. To wake the CPU up the RTS control line is raised and this transition from low to high causes the CPU to wake up.

The sleep state of the CPU can be determined by testing the RS232 CTS line. The CTS line is set by the controller CPU. If CTS is high this indicates the controller CPU is awake, running and ready to receive commands. If CTS is low then the CPU is asleep.

The program *ctalk.c* should be consulted for a model on how to send and receive data from the controller and how to use RTS/CTS. Shown below is a sample C routine which waits for CTS high.

Note that this sleep/wake behavior applies only to the timing board CPU. The temperature control CPU is independent and isolated from the rest of the controller. The temperature control CPU never goes to sleep. Temperatures can be read at any time, including during image readout.

```
/*    Do controller RS232 hardware handshake. If the TCB CPU*/
/*    is asleep we can wake it up by raising RTS. The       */
/*    controller responds by raising CTS                    */
/*    If CTS is already high this should mean the CPU is    */
/*    already awake. In this case we just raise RTS.        */
/*    We return 1 if the handshaking was successful and 0   */
/*    if not.                                               */
/*    ttchan is the file descriptor of the serial port.     */


static int rts = TIOCM_RTS;
static int cts = TIOCM_CTS;

int waitforcts(void)
{
      int bits;
/*    First check to see if cts is already high. If so we   */
/*    only have to turn RTS on.                             */
      ioctl(ttchan,TIOCMGET,&bits);
      if((bits&TIOCM_CTS) == TIOCM_CTS) {
/*        Turn on RTS                                       */
          ioctl(ttchan,TIOCMBIS,&rts);
/*        Find out how many characters are available.       */
/*        Read them and throw them away. They are all trash.*/
          usleep(10000);
          ioctl(ttchan,FIONREAD,&nchar);
          if(nchar > 0)
             flogerror("waitforcts flushing %d characters",nchar);
          while(nchar > 0) {
              nchar--;
              read(ttchan,&junk,1);
          }
          return(1);
      }
/*    Make sure we have nothing in the input queue before raising RTS   */
      ioctl(ttchan,FIONREAD,&nchar);
      if(nchar > 0)
             flogerror("waitforcts flushing %d characters",nchar);
      while(nchar > 0) {
              nchar--;
              read(ttchan,&junk,1);
      }
```

```
/*    Turn on RTS                                                */
      ioctl(ttchan,TIOCMBIS,&rts);


/*    Record the current time for doing a timeout.               */
      initial_time();

#define CTS_WAIT 500

/*      Wait for CTS to go high. Give up after CTS_WAIT msec.    */

        while(delta_time() < CTS_WAIT) {
           ioctl(ttchan,TIOCMGET,&bits);

/*          When CTS goes high we are ready to communicate.      */

            if((bits&TIOCM_CTS) == TIOCM_CTS) {
                flogerror("normal return");
                return(1);
            }
        }

        flogerror("timed out after %d milliseconds waiting for cts",
            CTS_WAIT);

        return(0);

}
```

Alphabetical Index