Guide to ARC ("Leach") Controller Timing Board DSP code

Introduction

This note is primarily about the timing board code derived from the HIPO timing DSP. The systems based on the HIPO code are HIPO, of course, and NASA42, LMI, DeVeny, RC1/RC2, and the CCD57 lab test system. The Gen III (DCT) versions of this have significant differences in detail from the Gen II (HIPO/Mesa) ones. The NIHTS timing DSP is closely patterned on the HAWAII-1 code we got from Bob Leach and bears only some resemblance to the HIPO code. It is much simpler.

This note doesn't discuss the older code in NASAcam, PSST, or the astrograph cameras. If these are ever touched I'd recommend basing the new code on the HIPO code.

Some of the code can be understood without reference to the controller electronics schematics but in general it is important to understand the drawings and how the controller is intended to work. For a new installation it is essential to understand all this as well as the CCD datasheet and interconnecting cabling, and very helpful, if not essential, to have a clear understanding of how CCDs work in general.

Source Files

timhdr.s
This is a header file with many definitions in it.

**waveforms*.s*
This is a file full of constant definitions that are the basis for the clock waveforms, bias voltages, clock timing, etc. It is often changed so it isn't unusual to have many versions of it in a directory. In some cases where we were exploring noise vs speed we made several different tim.lod files with different voltages, delay times, per-pixel integration times etc.

tim.s
This is the source file that has includes for all the other source files in it. After the opening includes the file is mostly occupied by the RCCD subroutine that reads out the CCD. After this come two core clocking subroutines (CLOCK and CLOCKCT), two more includes, and assignments of values to important X and Y memory locations. The X memory part has the command table in it. Note that this command table is glued onto the end of the command table started in timboot.s. At the very end the waveforms file is included.

timboot.s
This is the limbic system of the controller and we rarely get into it. It includes subroutines related to serial communications among the three boards that have DSPs on them (PCI, Timing, and Utility). It also has some configuration bits defined that may or may not be superseded by Peter's infospec stuff. Finally it has core functions for transmitting data on the fiber optic link, read/write memory, LDAPPL, and controller initialization code. It ends with the initial X memory definitions including the initial command table.

The next two files used to have some logic with one of them containing general functions and the other specific ones but this distinction has been lost. I think they could simply be glued together into one and there wouldn't be any difference except how long they are. Indeed, I see in one of Peter's confluence postings that Bob gave up on this too for Gen III systems. Anyway, here's what we've done.

timmisc.s
High points include CLEAR, IFLE, EXPOSE and a lengthy subroutine that does the housekeeping for the amplifier configuration you've selected (e.g. one amplifier, left and right, upper left and lower right, four

amplifiers, etc.) It is called SELECT_OUTPUT_SOURCE. The other big item in here are the functions that execute the various readout modes that the HIPO DSP is capable of. The SEX command causes a JSR to START_FT_EXPOSURE which begins with a jump table depending on the readout mode previously set by a call to SET_IMAGE_PARAM (the SIP command). The readout functions make heavy use of subroutines and aren't too long or difficult to read through.

timCCDmisc.s
This is a collection of short commands and subroutines that include things like setting the DACs, shutter control, setting the exposure time for software triggered images, configuration reporting, IDL, power control, wavefront sensor LED control, the notorious SET_MUX, subarray definition, and an important subroutine, PCI_READ_IMAGE, that tells the PCI card how many pixels to expect. Better be right!

infospec.s, *timinfo.s*, and *timinfospec.s* are configuration reporting things that I've never fully understood. The perl script, info2code, is somehow related to all this. Peter wrote up a Confluence on all this so we should be able to figure it out.

tim_dsp.h is a header file that is mostly connected to this configuration business. According to the Makefile it is actually not used for assembly but is created from tim.lod by dsp2chdr using *tim_dspC.ifc* as input. I think it might actually be a symbol table.

All of the above files are assembled or created via scripts by the makefile.

Output files

The **.lod* files are the assembled binhex files that are uploadable to the controller. The *tim.lst* file is a very useful file that the assembler generates that includes all the code in program memory address order. It includes the assembly code, hex machine code, and all the comments as well as the contents of the X and Y memory locations that are known to the assembler. This is a good way to check if you have addressing problems (like code that goes beyond the on-chip memory of the DSP itself.

The files *tim.cld* and *tim.cln* are binary files generated during assembly that eventually are used to create the binhex *tim.lod* file.

Ancillary Scripts

There are two additional perl scripts, *dsp2chdr* and *makealldsps*, that I haven't paid attention to. I don't think they are necessary for assembling the code but are "jigs" Peter created for particular purposes.

GENERAL DETAILS

- Memory architecture and usage

  The architecture of the DSP is optimized for digital signal processing (duh), generally, and multiply-add functions specifically. It is designed to be efficient at convolving a data stream with a filter function. One optimization is that there are three sets of memory: P (program) and X and Y (for data, I imagine one is normally the filter function and the other is the data stream). In any case, that's what's there. In the Leach controllers X memory is lightly used and those locations that are used are for basic operational purposes. Y memory is much more heavily used for application-specific things like clock waveforms and DAC values. There is a significant difference between the access time for on-chip memory and off-chip memory so the serial clock waveforms, in particular, need to be located in the limited amount of fast on-chip memory. We figured out the Gen II memory boundaries (56000) but I'm not sure of the Gen III (56300) boundaries. We might be excessively worried about memory boundaries in the Gen III code.

- A few random comments

  Many of the commands that the command jump table correspond to show up in two guises. When a command is received the processor jumps to the location in the table rather than jumping to a subroutine. Wen it's done it jumps to FINISH. The second guise is as a subroutine. In this case the jump table jumps to a very short command that does a JSR to the subroutine and then jumps to finish. This is needed if the same functionality is needed in subroutine form as well as in command form.

- Argument passing for commands

  When a command is received it is processed by PRC_RCV in timboot.s. This puts the command and its succeeding arguments in a table starting at X:COM_BUF, which is 7 words long. The number of words in the command is stored in X:NWORDS. PRC_RCV decides if this is for the timing board. If not it passes the command on to the utility board. If it is for the timing board it is the program jumps to COMMAND. This looks through the command table for a match and if it finds one it jumps there. When the program jumps to the command register 3 has the address of the first argument in it. (If there's no match it goes to ERROR and sends an ERR reply back.)

In the function a syntax like

```
MOVE X:(R3)+,Y1
```

copies the first argument into the Y1 register and increments R3 to point to the next argument. Subsequent lines like this read the rest of the arguments.

As an example consider the command to set the two LEDs that are related to the S-H beamsplitter. (In HIPO there really were two of them but we only really use one at the DCT.) This command is SETVRDS in timCCDmisc.s. It gets the each of the two arguments (DAC values), masks them to 14 bits, adds an address and stores them in a tiny special DAC list at the very end of the waveforms file. At the end it loads the address of the tiny DAC table into R0 and calls SET_BIASES to set the two DACS.

CODE DETAILS

Waveforms File

There are no executable instructions in this file, just compiler directives (e.g. EQU for symbol definition) and defined constants (DC) that take space in Y memory. See the end of tim.s where this file is included. The initial 11 definitions are fixed values for a given board set. Next come some key timing values:

INT_TIM - The per-pixel reset and signal integration times

SI_DELAY - The parallel clock duration per phase

R_DELAY - The serial clock duration per phase

We generally leave ADC_TIM and SKIP_DELAY at zero to keep these signals going as fast as they can. The translation from these hex numbers to microseconds is in the PDF description of the timing board that Bob Leach wrote up.

In the more recent waveform files there is a section defining several states of the various analog switches surrounding the correlated double sampling (CDS) dual slope integrator circuit. These are defined as

binary bit patterns. In the older files the binary patterns appear directly in the waveforms. These don't exist in the NIHTS waveforms file because of the way IR array CDS is done.

Next come a whole set of clock and bias voltage settings as well as internal offset voltages in the analog board itself. After this is the assignment of particular clock signals on the CCD to particular analog switches on the clock driver board. This assignment is defined by how the interconnecting cabling from the clock driver board to the CCD is built. The details of the clocks that exist on a given CCD are in the CCD datasheet. Just because the clocks are there doesn't mean you need to use them all independently. We sometimes gang some of them together to avoid the need for a second clock driver board, for example. NASA42 and LMI are cases like this.

Finally we get to the actual waveforms. These all have the general structure that they have a starting address and ending address, and the first defined constant in the waveform is the difference between them (with maybe a -1 depending on the controller architecture). The CLOCK or CLOCKCT function that these waveforms are designed to work with reads this length to figure out what it has to do. The following lines are sums of symbols already defined by EQU's. The bit patterns defined by these lines are sent to the clock board to operate analog switches that switch each clock signal between its high (a 1 in the bit pattern) and low (a 0 in the bit pattern) levels as set by the clock voltages already defined. Reference to the data sheet is essential to get these right.

There are many details in the serial clock sequences in particular that are best plumbed by looking at the datasheet and the waveforms and also RCCD in tim.s. There is a spot set aside for copying in the serial binned waveforms from high memory for execution in low, fast memory. It is usually big enough to handle binning by 5. Note that parallel binning doesn't suffer from the same speed problem.

Next comes a long string of DAC settings for the clocks and biases. This starts at location DACS and ends at END_DACS. Like the waveforms the first entry in this table is END_DAC-DACS-1, a count that the command SETBIAS in *timCCDmisc.s* uses.

timboot.s
In the GenIII code a lot of the EQU definitions that were in timhdr.s are now at the beginning of timboot.s. These are mostly addresses of various I/O ports and control registers in the DSP chip. After that comes a bit of initialization code, and two interrupt service routines for the asynchronous serial communication scheme. Then come command processing functions for various communication links. After this we see the simple RDMEM and WRMEM memory read/write commands and LDAPPL that loads the EEPROM boot code into P memory for execution. Next comes initialization code that I've never looked at. The last bit is some X memory definitions including a small command table. The larger one in tim.s is appended to this one, starting at END_COMMAND_TABLE. The length of the command table, NUM_COM, starts out being hardwired at 7 and is updated by tim.s when the additional commands are appended.

One puzzle is that I don't see LDAPPL being used or LOAD_APPLICATION being called except by LDAPPL. As a result I don't know how the EEPROM code gets copied into P memory. The code to do it is there but isn't called unless the LDA command is issued to the controller. Hmmm. Maybe the solution is that when we upload our code using lots of WRMEMs it is written to P memory starting from the beginning and there's no need for LDAPPL.

timboot.s is something we can't change much. By and large I've thought of it as Darkest Africa. During load it overwrites the ROM code on the fly during execution, something that can be fraught with peril if not done right. I have never taken the time to fully understand the fancy footwork that happens at startup.

tim.s
From a structural point of view tim.s is the linchpin. Before it does anything significant it includes timboot.s, timhdr.s, and various of Peter's info.s files. At the end of the program part of tim.s it includes timmisc.s and timCCDmisc.s, and finally finishes up work on X memory and includes the waveforms file to finish setting up Y memory.

The bulk of tim.s is occupied by the RDCCD subroutine that reads out the CCD. It is long and complex. Mike has suggested that it might be better to break it up - I haven't thought about this suggestion enough to know if that's possible or not.

After RDCCD ends there are two utility subroutines that need to be fast (all of tim.s is in on-chip memory). It used to be that there was only one, CLOCK, but the Gen III timing board includes a FIFO for clocking that caused Peter to introduce CLOCKCT, a modified version of CLOCK. I wasn't aware of this till recently and I haven't understood when he uses CLOCK and when he uses CLOCKCT.

The last part of tim.s is filling out X memory with a longer command table, important system-level addresses like status words, CCD size, binning factors, shutter delay, clocking details, subframes, and variables added for time series data acquisition. The last gasp of tim.s is to include the waveforms file.

timmisc.s
This file contains various utility commands and subroutines, but toward the end are the key data acquisition functions. The code starts with a collection of command functions and subroutines:

CCD clearing functions

IDLE, the function that sets up the CCD for idling between exposures

EXPOSE, start the exposure and monitor its progress

SEL_OS (command) and SELECT_OUTPUT_SOURCE (subroutine). This is a long function that organizes things to suit a given output amplifier configuration and binning factor. This is where the waveforms from high Y memory get copied into SERIAL_READ. There is also a little hocus-pocus with the clock line that is copied to the CCLK1 location in VIDEO_PROCESS (in the waveforms file). This is because everything in VIDEO_PROCESS has to be done regardless of binning factor or amplifier configuration with the exception that the final serial clock between the reset and data phases of the correlated double sample varies depending on the amplifier configuration. In the CCD57 case for left amplifier the charge is moved to phase 2 before the signal integration, to phase 1 for the right amplifier, and to phases 1R and 2L for both amplifiers. This is only used for general binning for serial binning greater than the maximum binning supported by the waveform copy approach.

WAVECPY and SET_BINBIT are small subroutines used by SEL_OS for the waveform copy.

SET_ROWS_COLUMNS sets the size of the image and binning factors to use. This needs to be called prior to SEL_OS because SEL_OS needs the binning factor to be set first.

SET_IMAGE_PARAM is a function we created for time series readout modes. It lets you set the readout mode, how many time steps there are, the value of SROWS, and the interval between time steps. SROWS is the number of (binned) rows you want to shift between images in a time series. It is usually converted to unbinned rows by UB_CONV and stored in UBSROWS before it is used.

SET_TRIGGER is how you set up to use hardware triggering instead of free-running data acquisition based on the Leach controller's exposure time approach.

SETUP_SUBROUTINE is called only once from the RDCCD function, near the beginning of the loop over the subframes. It computes important loop counters for use later on in the RDCCD function. It in turn calls FASTSKP in *timCCDmisc.s*.

Three trigger-related functions, WAIT_UNTIL_TRIGGER, WAIT_WHILE_TRIGGER, and CLEAR_WHILE_TRIGGER come next. These do the obvious things but are no-ops for systems without hardware trigger capability.

UB_CONV was discussed in SET_IMAGE_PARAM.

Next comes the real meat of the timing code, the parts that coordinate the data acquisition operations for all of the readout modes we've defined to date.

START_FT_EXPOSURE is a jump table based on the image mode previously defined by

SET_IMAGE_PARAM. This is what the 'SEX' command jumps to. Following this are four important functions that orchestrate the activity of the various readout modes.

There is a spreadsheet that shows what the control bits are for each of these modes and they are best understood by studying the code with this spreadsheet at hand.

FDOT_PROC is only used by fast dots mode.

SINGLE_PROC is used by both single and basic occultation modes.

SDOT_PROC is used by slow dots and strip scanning modes.

FPO_PROC is used by the fast and pipelined occultation modes.

Following these are some support functions:

IMG_INI does preparatory work that prepares the PCI card to receive data. It also

changes the status bit to readout status. It calls PCI_READ_IMAGE in timCCDmisc.s.

CLEANUP does some stuff that is common to the end of all four f the data acquisition

functions.

ISHIFT, SSHIFT, and RSHIFT are subroutines that deal with the frame transfer and storage area row shift manipulations needed for the various modes. RSHIFT is used by pipelined occultation mode. I remember having trouble with this and not using it, but it is called in the CCD57 code and see it in the HIPO code too. Hmmm.

timCCDmisc.s
This file contains miscellaneous utility functions. The first four are things I've never looked at. They are quite low-level.

SET_DAC is an old subroutine that has been replaced by SET_BIASES. It is never called.

FASTSKP is called by SETUP_SUBROUTINE in timmisc.s to help compute the number of fast clocks needed.

SET_SHUTTER_STATE is a subroutine called by all the commands and subroutines for shutter control that follow it in this file. It sets the actual hardware bit that goes to the shutter driver. The four shutter control commands are:

OPEN_SHUTTER and CLOSE_SHUTTER, as commands, not subroutines.

C_OSHUT and OSHUT are two entry points into a subroutine to open the shutter. The

C_OSHUT entry point is conditional depending on the shutter bit in X:STATUS. The

OSHUT entry point skips this test and unconditionally opens the shutter.

C_CSHUT and CSHUT are the same except for closing the shutter.

SET_EXP_TIME is the command you use to set the exposure time.

ABORT_EXPOSURE is a command we will need to understand when it's time to get abort to work.

GET_INFO returns configuration information. I haven't plumbed this before. I think this is Peter Collins' configuration scheme.

IDL as a command is used to set the timing board back into idle mode.

READ_CONTROLLER_CONFIGURATION is another configuration function that I think is from Bob Leach.

PWR_OFF and PWR_ON turn the power off and on using analog switches to control the power MOSFETS that connect the analog power supplies to the rest of the controller.

These are commands. PWR_ERR isn't a command, just an alternate way to exit PWR_ON if PWROK isn't OK. PON is a subroutine to turn on the power. There is no POF.

Note that there is a specific order in which the power comes on with delays to be sure that certain voltages are correctly established before others are turned on. This is done to protect the CCD on power-up.

SETVRDS is a command in the CCD67 timing DSP that controls the brightnesses of the Shack-Hartmann LEDs. It isn't in the CCD57 timCCDmisc.s but is in the 20180214 version of it. This was copied from the CCD67 timCCDmisc.s. If we ever install the LEDs into the CCD57's probe assembly this will need to be added too.

SETBIAS is the command form of the subroutine SET_BIASES. This is normally used to set all the bias voltages in the big table in the waveforms file, but is also used for the two biases that are in the tiny table at the end of the waveforms file that control the LEDs for the Shack-Hartmann test.

CLR_SWS and CLEAR_SWITCHES clear the analog switches. The subroutine form is called by the PWR_ON and PWR_OFF commands but not by the PON subroutine. In practice I think this is usually an initialization function. That said, there are times when a controller can be brought to "2-light state" by electrical discharge or other interference. The usual recovery from this has been to restart LOIS, which reloads the DSP and turns the power on over again. A shorter fix might be PWR_OFF and

PWR_ON followed by whatever additional initialization is needed to set up for the next exposure. This would need some thought to get it right.

SET_MUX is a command used to direct clock signals to the two debugging multiplexers that are on the clock driver board. The version in the CCD57 code is wrong and should be replaced by the NIHTS version.

ST_GAIN is a command to set the gain in the programmable gain stage of the video board. I've been burned by having the software set the gain but not keep track of it and my general attitude is to set the gain once at the DSP level, or leave it at a default state, and never touch it again. I'm not convinced that this command is even right. The older 2-channel boards we use at the Mesa and in HIPO have a very different arrangement for the programmable amplifier stage than the newer 4-channel boards do and I fear that we have old code in some of the newer DSPs. Doesn't matter if you don't use it, but it's wrong anyway.

SET_SUBARRAY_SIZES and SET_SUBARRAY_POSITIONS are a pair of commands to set up the subarrays you want to use. It is important to set the sizes first and positions second in order to clear the NBOXES variable that becomes set by the positions function. The subarrays all have the same size and cannot overlap in the row direction.

PCI_READ_IMAGE is the function that tells the PCI card how many pixels to expect. It's a little bit tricky and absolutely has to be right.

Revision History

Date	Revision	Description	Author
2019-01-22	1.0	Initial draft	EWD
2019-01-23	1.1	Reformatted from flat text	MPC