Memo

To: ObsSim Users

From: Jim Littlefield

CC:

Date: 11/13/15

Subject: Observatory Simulator Code Update - v1.7a

Preface for Version 1.7a:

This release contains updates to address problems reported by users as described below.

- *cam_hsk immediately after loading memfiles fails* : Kari traced this to noise occuring during FPE bitfile loads. To correct the problem, the FPE->ObsSim sync pattern was changed to make the ObsSim less likely to get faked out by noise on the LVDS downlink lines.
- *Frame pixel data appears to have "cut and paste" errors* : We identified an issue where the first 8 pixels of every 1K transferred were duplicated into the second 8 pixels. Kari has modified the PL to correct this issue. Based on Jerry Roberts original description of the problem we think this is the issue he had identified.
- *ObsSim 1.6c reports all short frames on Mac :* I was not able to duplicate this problem on Cosmos, however I have revised the frame buffering algorithm to allow the network data transfer to be behind by more than 1 frame. The output format of the frame statistics messages has been expanded to include some additional variables (circular buffer pointers, interrupt counts, pkts generated).

IMPORTANT NOTICE: ObsSim 1.7a and later are not compatible with FPE bit files earlier than FPE_Wrapper_6_1_V3final_1104.bin. Use of ObsSim1.7 with earlier FPE_Wrappers will result in no communications between ObsSim and the FPEs.

Preface for Version 1.6c:

The 1.6c ObsSim code version has the following changes.

- Add "dhu_rst", "cam_frinf_on", "cam_frinf_off", "cam_hm", "cam_prg", "cam_reg", "cam_seq", "cam_clv" commands. See body of document for a description of these new commands.
- Fix missing va_end() in command response formatter
- Add delay between sending of command responses to avoid host computer dropping udp

messages.

- Fix memory leak in tftp server that caused system lockup after ~200 file transfers
- Correct doumentation error for FPE2 Data port (7778 not 7776)
-

Preface for Version 1.6a:

The 1.6a code software version has the following changes.

- Add “help” command to display a list of supported command strings
- Add “registers” command to display a list of readable/writeable registers
- Correct camera reset command (previously was “camrst”). Now both “camrst” and “cam_rst” are supported. “cam_rst” is the preferred command format.
- Change behavior of memory load commands so that a downloaded tftp file can only be used once. This prevents a possible error where the tftp file is byteswapped in place 2x resulting in an incorrect byte order transfer to the FPE.
- In many places where the console command interpreter waits for hardware/firmware to do something a timeout has been added. If the target operation does not succeed within 1 sec a “Timeout” message is printed on the udp console.
- Work around a bug in memmove() which caused incorrect/garbled data transfer from the tftp file temporary buffer to BRAM if the memfile size is an odd number of words in length.

Preface for Version 1.5:

- Document new commands “version”, “cam_frinfo”
- Document udp port utilization for dual FPE's

Preface for Version 1.4 :

- Replaced OS w/ ObsSim in text
- Corrected spelling of kermit protocol
- Added section describing in overview detail the ObsSim software
- Updated to commands to reflect the forthcoming FPE2 support
- Update table 1 to show target tftp file names to direct content to FPE2

Preface for Version 1.3 :

Version 1.3 has the initial modifications for the second FPE interface. As part of this migration there are two command udp port (5555 for FPE1 and 5556 for FPE2). In addition the console prompt has changed from “?>” to “FPE1>” ofr “FPE2” depending on which command port the operator is connected to.

Since there is a dedicated command udp port for FPE1/FPE2 it is no longer necessary to include the camera number in the command syntax. All command strings have been modified to drop the numeric digit (eg “cam1_status” → “cam_status”, “cam1_start_frames” → “cam_start_frames”, etc).

The most recent code update for the TESS Observatory Simulator (ObsSim), OSExpl_1_6c_Debug.mcs, is now available and can be flashed to your hardware using the Xilinx SDK tool suite. This code version has the changes listed in the preface.

This document discusses ObsSim related procesures for loading a set of camera configuration files. The generation of these files is described in a separate document.

1 Installing the ObsSim Update

1.1 Getting OSExpl.mcs

The **OSExpl_1_6c_Debug.mcs** file contains a binary image suitable directly loading into the ObsSim flash. A copy of this file can be downloaded from https://github.com/TESScience/Observatory_Simulator_MCS.

1.2 Reflashing the board

If you have the Xilinx SDK installed on your workstation the following steps can be used to reflash the ObsSim. You must have a USB connection to from your computer to the ObsSim's real panel USB jack.

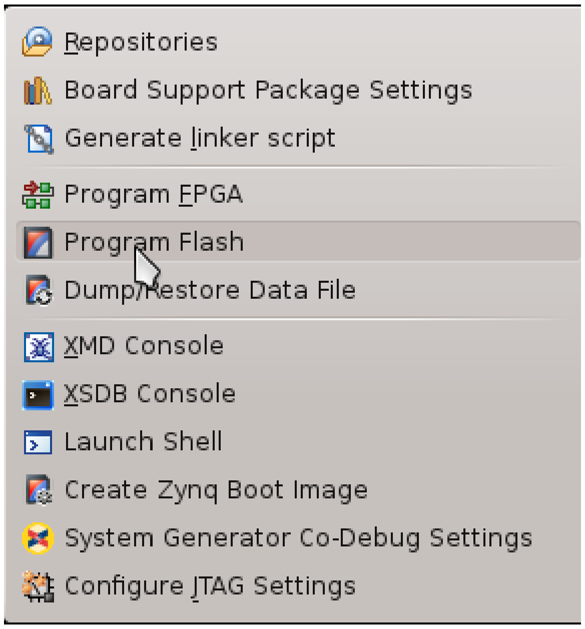

Illustration 1: Xilinx Tools Menu

From the primary menu bar select the Tools option to see the above choice list. Select the "Program Flash" option. This will bring up a second dialog where the *.mcs file can be specified as shown below.

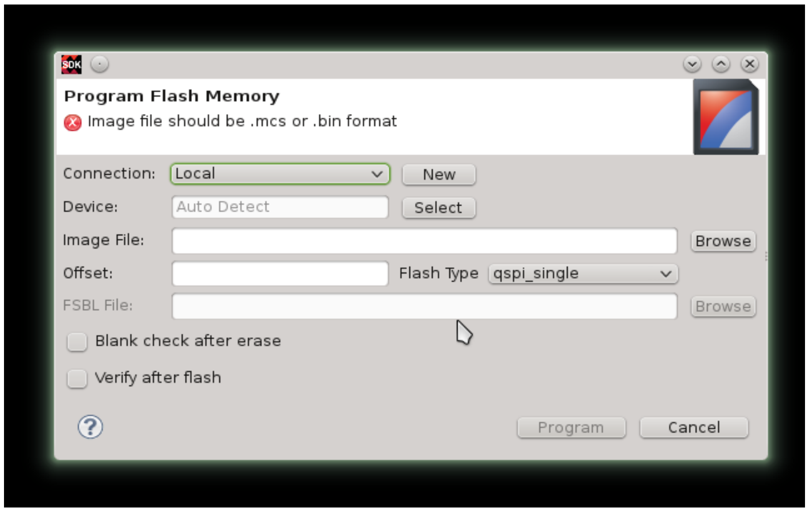

Image file should be .mcs or .bin format

Flash Type qspi_single

Illustration 1: Program Flash Options Menu

Use the Browse button next to the "Image File" selection to navigate to the OSExpl.mcs file.   Be sure the Flash Type is set for qspi_single.

Once all the necessary options are specified the "Program" button will be enabled.   Click it to begin the reflashing operation.   *Note that the reflashing operation takes 10 – 15 minutes to complete.*

Once the programming operation completes successfully it may be necessary to change the Zynq 706 board boot selection switches (SW11) so select QSPI as the boot source. The proper setting is for positions SW11.1,2,3,5 to be OFF and SW11.4 to be ON.

1.3 Local Network Configuration and UDP Ports

1.3.1 Serial Console Port

The ObsSim supports a serial console port (115200,8,n,1) via an internal USB->Serial converter.   During system boot, various sign-on, status, and possibly error messages are sent on this connection.   Although not required for general use, the serial console messages can be useful for diagnosing various configuration problems.

When first making the USB connection between the ObsSim and a workstation the serial console is likely to be assigned to /dev/ttyUSB0 (unless this device is already claimed). To access the serial console (via minicom, kermit, putty) it may be necessary to have administrative rights on the computer to change ttyUSB0 permissions to be read/write by everyone.

Note that the JTAG programming/debug interface and the serial console share the same connection.

1.3.2 LAN Configuration

The ObsSim generates significant network traffic when image frames are being acquired. Because substantial portions of the network bandwidth are required the ObsSim must be connected to the host computer via a dedicated network segment. This is most easily accomplished if the host computer has two network interfaces one of which can be dedicated to the ObsSim ↔ Workstation link.

DHCP is not currently supported on the ObsSim. The ObsSim uses a static IP of 192.168.100.1 and gateway of 192.168.100.2.

Even when configured per the above recommendations, it is quite possible for the ObsSim udp console output to exceed the receiving capability of an attached host computer. This problem can be especially acute when a console command generates a lot of output (eg memory readbacks, cam_hsk). Version 1.6c adds a small delay between each udp console output packet decrease the probability that the host will drop console output packets.

1.3.3 UDP Connections

The primary method of communication with the ObsSim is via UDP sockets.

Port	Services
5555	FPE1 Console command/response
5556	FPE2 Console command/response
7777	FPE1 Image/housekeeping data stream
7778	FPE2 Image/housekeeping data stream
69	Tftp server/file transfer

Table 1: UDP Ports and Services

2 Workstation/PC Tools Required

Interactions with the ObsSim are conducted via UDP. It is therefore necessary to have some tools on the host to allow an operatore to send/receive text via specified UDP ports. The discussion below illustrates one group of linux tools that do the job. Windows, MacOS users will need to get equivalent tools.

2.1 Tftp Client

A tftp server runs on the ObsSim processor. This is a special limited server which supports only a single file upload. Thus, each tftp send operation overwrites the file from a previous send.

The host computer must have an available tftp client application so that FPE configuration files can be transferred from the host to the ObsSim and then on to the FPE.

An example command is ….

```
tftp -v -m binary 192.168.100.1 69 -c put <local file> <remote file>
```

Since the files being transferred will generally be binary, it is important to specify that binary transfer mode be used. On some tftp clients the argument may be "-mode binary" or "-mode octet".

Although not required the final <remote file> argument provides some convenience. If this argument is not provided, the file transfer will complete successfully *but the transferred file data will not be passed on to the FPE. In this case it would be necessary for the operator to issue a console command to transfer the unloaded file contents to one of the FPE memories.* If a valid <remote file> name is specified (see table below), this will invoke a ObsSim command to automatically transfer the uploaded file content to the FPE.

File Name	Associated ObsSim Command
"bitmem" "bitmem2"	bitmem – transfer file to FPE FPGA
"prgmem" "prgmem2"	prgmem – transfer file to FPE program memory
"seqmem" "seqmem2"	seqmem – transfer file to FPE sequencer memory
"regmem" "regmem2"	regmem – transfer file to FPE registers
"hskmem" "hskmem2"	hskmem – transfer file to FPE housekeeping setup registers
"clvmem" "clvmem2"	clvmem – transfer file to FPE voltage DACs

Table 1: Valid remote file names

2.2 UDP network client

To access the ObsSim command interface an application which provides a terminal like interface to udp port 5555 is necessary. On linux this tool is “netcat” or “nc”. The command to connect to the ObsSim is ….

```
nc -u 192.168.100.1 5555
```

When establishing an initial connection it may be necessary to hit the <enter> key a few times.

2.3 Tools for receiving image data

[Ed Morgan's Tools - TBD]

2.4 Tools for creating configuration data

The Artix_FPE_MemConfig.xls spreadsheet in the FPGA_MemConfig directory, found in the Observatory_Simulator gitlab repository, has an example of how to configure the four firmware memories for science operations. These memories are: ClvMem, which load the clock level voltages, HskMem, which determines which housekeeping will be sent, PrgMem, which specifies the program to be run on the CCDs, and SeqMem, which provides clocks to the CCDs. In addition, RegMem provides details to the firmware on how to handle housekeeping downloads, including how many housekeeping values to send per frame.

There are two ways to generate the binary files necessary to load these memories. One is to use the spreadsheet, and the other is to create ASCII files, using the spreadsheet as a guide for determining bit placement.

If the spreadsheet is used, binary files can be generated by running

```
$ ./exceltobin.py [filename]
```

This will automatically find the relevant sheets and save the outputs as separate binary files, inheriting the sheet names (e.g. ‘CLVMem6_0’). It will also save the outputs as human-readable ascii, with the same names and ‘.asc’ appended (e.g. ‘CLVMem6_0.asc’).

Note that for the above script to work, the folder ‘xlrd’ (containing the Python package xlrd for reading excel files) must be within the same folder as exceltobin.py.

If the ascii files are edited, new binary files can be created by running

```
$ ./asctobin.py [*filenames]
```

where any number of ascii files are supplied as command-line arguments. The new files will be saved with the same names as the input files, but with '.bin' appended (e.g. 'CLVMem6_0.asc.bin').

3 Use Cases

This subsection describes some typical use cases.

3.1 Power Up

For initial power up observe the following steps....

- Connect AC power to the backpanel power entry module
- Connect USB cable between host workstation and the ObsSim backpanel
- Connect Ethernet cable between the ObsSim backpanel jack and the host/workstation
- Move both the survival and main power toggle switches on the ObsSim front panel to their leftmost positions.
- There is a toggle switch on the ObsSim backpanel. This switch is integrated into the power entry module. Toggle this switch to the ON position. This will apply power to the Xilinx board. Once powered up the Vadj, +12, +3.3V LEDs on the ObsSim front panel should be lit. *Note : Some ObsSim units appear to have power up timing issues wherein the +12, +3.3 supplies are enabled but the Vadj supply fails to start. This condition will be indicated by the Vadj LED on the front panel being OFF while the +12 and +3.3 LEDs are illuminated.* If this condition occurs, use the backpanel AC switch to toggle power off, wait 30 sec., and restore AC power.
- After power up, the ObsSim will boot from internal flash, configure its network interface, and begin waiting for commands received at 192.169.100.1:5555/6. During the boot process multiple status messages will be emitted to the serial console.
- Start a netcat session ("nc -u 192.168.100.1 5555") and hit enter in the terminal window. Type the command "cam_status" to verify that the connection is working. You should see a message indicating the contents of FPE 1 status register.

3.2 Load Camera FPGA

This section illustrates the steps to bring up a camera/FPE. It is assumed that all the steps in the preceding section have already been accomplished.

- Move the ObsSim front panel Main power toggle switch all the way to the right. Observe that the +5,-12,+15 LEDs on the ObsSim front panel are illuminated and the +15 Surv. LED is not illuminated.
- At a terminal window on the workstation/host, issue the command ..."**tftp -v -m binary 192.168.100.1 69 -c put FPE_Wrapper.bin bitmem".** Substitute the local filesystem name of the FPE firmware desired to be loaded.
- After a few seconds tftp will announce how many bytes were transferred and return to the command prompt.
- Observe the netcat window started in the previous section. There will be a series of "."'s displayed as the data is transferred to the FPE. Once the transfer is complete, the FPE FPGA is operational.
- Note that the bit file load/FPGA programming can only be done 1x per power cycle of the FPE.
- Continue with the next section to load configuration files for the individual FPE memories and register sets.

3.3 Load Camera Configuration Files

This section assumes that the ObsSim is powered up and the FPE FPGA has been loaded with a valid bit file. To complete camera/FPE configuration it is necessary to load 5 more files.

- At a terminal window on the workstation/host, issue the command ..."**tftp -v -m binary 192.168.100.1 69 -c put RegMem regmem".** Substitute the local filesystem name of the FPE register memory configuration desired.
- At a terminal window on the workstation/host, issue the command ..."**tftp -v -m binary 192.168.100.1 69 -c put PrgMem prgmem".** Substitute the local filesystem name of the FPE register memory configuration desired.
- At a terminal window on the workstation/host, issue the command ..."**tftp -v -m binary 192.168.100.1 69 -c put SeqMem seqmem".** Substitute the local

filesystem name of the FPE register memory configuration desired.

- At a terminal window on the workstation/host, issue the command ...”**tftp -v -m binary 192.168.100.1 69 -c put HskMem hskmem”.** Substitute the local filesystem name of the FPE register memory configuration desired.
- At a terminal window on the workstation/host, issue the command ...”**tftp -v -m binary 192.168.100.1 69 -c put ClvMem clvmem”.** Substitute the local filesystem name of the FPE register memory configuration desired.
- Each transfer should generate a command response message on the netcat window indicating that the data was successfully transferred.
- To begin collection of image frames see Sec 3.6.

Quoted below is a shell script which automates the process outlined above.

```
#!/bin/bash
# Script to automatically start frames...
#
tftp -v -m binary 192.168.100.1 69 -c put FPE_Wrapper.bin bitmem
sleep 10
tftp -v -m binary 192.168.100.1 69 -c put SeqMemStag seqmem
sleep 1
tftp -v -m binary 192.168.100.1 69 -c put RegMem regmem
sleep 1
tftp -v -m binary 192.168.100.1 69 -c put PrgMem prgmem
sleep 1
tftp -v -m binary 192.168.100.1 69 -c put CLVMem6_0 clvmem
sleep 1
tftp -v -m binary 192.168.100.1 69 -c put HskMem hskmem
sleep 1
```

3.4 Changing Camera Configuration

Individual configuration memories and registers in the FPE can be changed as needed however a new FPGA bit file cannot currently be uploaded without power cycling the cameras via the ObsSim front panel switch. The procedure for changing the contents of a particular FPE memory are the same as outlined in the preceding section.

- Suspend image data collection ("cam_stop_frames") if previously enabled.
- Issue a "cam_rst" command to ensure that the FPE logic state machines are reset.
- **"tftp -v -m binary 192.168.100.1 -c put ClvMem2 clvmem"** for example, to change the voltage memory from ClvMem to ClvMem2.
- Resume image data collection ("cam_start_frames") or manually request housekeeping data ("cam_hsk").

3.5 Polling for Housekeeping data

A new command "cam_hsk" allows acqusition of housekeeping data when image frames are NOT being acquired. Issueing this command at an ObsSim command prompt ("FPE1>", "FPE2>") will cause the ObsSim to poll the FPE for housekeeping data and, when acquired, will print the 128 values on the udp terminal. *Note: You must have previously uploaded a configuration to the FPE housekeeping memory and the FPE HSK_SAMPLE_PER_FRAME registers or no data will be returned and/or the command will timeout.*

3.6 Starting/Stopping Image Frames

This ObsSim code version (1.3 and later) has a command, "cam_start_frames", which enables the ObsSim's DHU firmware to generate PPS commands to the FPE to cause acquisition of image data. You must have previously uploaded a valid set of FPE configuration files. Starting image frames without a valid configuration loaded can have unpredictable results. Once image frames have been started, the acquired frames and housekeeping data will be transmitted from 192.168.100.1:7777/8.

To suspend frames issue "cam_stop_frames" (or "cam_control = 0") from the udp console.

4 ObsSim Software Description

The ObsSim hardware is based in part on the Xilinx ZC706 Kintex evaluation board. This board provides an Zynq 7000 series chip along with various I/O peripherals. The Zynq 7000 consists of processing system (PS) and programmable logic (PL) sections. The PS is an ARM based general purpose processor which is able to communicate with the PL. This document subsection provides an overview of the PS code.

4.1 PS/PL Communications

PS/PL communications consists of 6 interrupts (PL->PS) and 3 dual port memories (pcie_bram, ddr1_bram,ddr2_bram). Pcie memory is written by the PS with command/request packets which trigger register read/write operations in the PL. When the PL has completed a register read/write operation the PL updates the pcie memory to signal that the requested operation is complete (many operations also result in a PL->PS interrupt).

The ddr1/ddr2 bram memorys are reserved for bulk data transport for fpe1/2 respectively. Generally this means image data from the PL to the PS but during fpe configuration the ddr memory is also used for transport of the fpe fpga configuration bitstream from PS->PL->FPE.

The PL->PS interrupts are summarized in the table below.

Interrupt	Description
FPE 1 DHU Interrupt	Signals DHU1 command completion and other processing related events.
DDR1_Rdy	Signals that ddr1_bram has data ready to be read
DDR1_Req	Signals that previously written ddr1_bram data has been transmitted to FPE1 and more data is required.
FPE 2 DHU Interrupt	Signals DHU2 command completion and other processing related events.
DDR2_Rdy	Signals that ddr2_bram has data ready to be read
DDR2_Req	Signals that previously written ddr2_bram data has been transmitted to FPE2 and more data is required.

4.2 Implementation Choices

The ObsSim PS code is written in C using the xilinx SDK tool suite. The project type is “bare metal application” meaning that there is no operating system support incorporated into the code.

This choice was made for the following reasons...

a) Petalinux is available for the hardware platform but the existing driver suite did not incorporate bram driver/interrupt handlers.

b) Due to the tight timing constraints during image acqusition it is essential to provide low latency, low jitter transport from bram to main memory and then from main memory to the network stack.

It may be worth revisiting the “bare metal” choice at some point in the future. Certainly there are downsides to the lack of an operating system in terms of flexibility, modularity, and ease of customization.

The following drawing show an overview of data flows through the PS.

Drawing 1: ObsSim Frame Buffer, BRAM interface

4.3 Network Interfaces

The ObSim communicates with a command/control application via 5 udp sockets. Image/housekeeping data is also transferred via socket connections.

UDP Port	Interface Function
5555	FPE1 command port
5556	FPE2 command port
7777	FPE1 frame data/housekeeping port
7778	FPE2 frame data/housekeeping port
69	TFTP server (configuration memory loads)

The tftp port is used for communications with both FPE1/2 with the target camera distinguished by the target filename to which the transfer is directed.

5 Command Reference

This section lists all commands supported by ObsSim version 1.3.

5.1 Configuration/Convenience Commands

bitmem - Transfer the last received tftp file content to the FPE FPGA as a configuration bit stream.

seqmem - Transfer the last received tftp file content to the FPE's sequencer memory

prgmem - Transfer the last received tftp file content to the FPE's sequencer memory

hskmem - Transfer the last received tftp file content to the FPE's housekeeping configuration memory

clvmem - Transfer the last received tftp file content to the FPE's voltage level memory

regmem - Transfer the last received tftp file content to the FPE's registers

dhu_rst - Reset the ObsSim internal interface between application code (PS) and PL. Note that this hardware interface is shared between both FPE1/2 so sending this command to either Udp console will affect both camera interfaces.

cam_rst, camrst - Reset FPE state machine logic.

cam_hsk - Poll for housekeeping data and display the values on the udp console. Note: This is for debug only. It is to be used only when image frames are NOT being acquired.

cam_hm - Dump housekeeping selection memory from FPE. The memory printed to the udp console as a sequence of 32 bit values.

cam_reg - Dump register memory from FPE. The memory is printed to the udp console as a sequence of 32 bit values.

cam_prg - Dump program memory from FPE. The memory is printed to the udp console as a sequence of 32 bit values.

cam_seq - Dump sequencer memory from FPE. The memory is printed to the udp console as a sequence of 32 bit values.

cam_clv - Dump voltage memory from FPE. The memory is printed to the udp console as a sequence of 32 bit values.

cam_start_frames – Configure DHU/SPM registers to enable PPS generation and image acquisition.

cam_stop_frames – Stop generation of PPS messages from the ObsSim to the camera. This will stop frame collection.

cam_frinfo – Toggle on/off the printing on the udp console port of per frame statistics during frame collection.

cam_frinf_on – Enable printing on the udp console port of per frame statistics during frame collection.

cam_frinf_off – Disable printing on the udp console port of per frame statistics during frame collection.

version - Print on the udp console port the ObsSim software version and build date/time.

help – Print on the udp console a list of supported ObsSim commands

registers - Print on the udp console a list of readable/writeable registers (See below)

5.2 Register Access Commands

In addition to the general commands which perform sequences of register read/write operations, the ObsSim supports individual register read/write commands.   The supported list of register names is given in the structure quoted below....

```
DHURegisterType DHURegisters[] = {
      {"CAM_CONTROL", CAM_CONTROL},
      {"CAM_STATUS", CAM_STATUS},
      {"CAM_INTERRUPT",CAM_INTERRUPT},
      {"CAM_INTERRUPT_ENA",CAM_INTERRUPT_ENA},
      {"CAM_INTERRUPT_FORCE",CAM_INTERRUPT_FORCE},
      {"CAM_INTERRUPT_SPM",CAM_INTERRUPT_SPM},
      {"CAM_INTERRUPT_SPM_ENA",CAM_INTERRUPT_SPM_ENA},
      {"CAM_INTERRUPT_SPM_FORCE",CAM_INTERRUPT_SPM_FORCE},
      {"CAM_PPS_DELAY",CAM_PPS_DELAY},
      {"CAM_FRAME_COUNT",CAM_FRAME_COUNT},
      {"CAM_ALL_PIXELS",CAM_ALL_PIXELS},
      {"CAM_DATA_PIXELS",CAM_DATA_PIXELS},
      {"CAM_OC_PIXELS",CAM_OC_PIXELS},
      {"CAM_NUM_BAD_PKTS_A",CAM_NUM_BADPKTS_A},
      {"CAM_NUM_BAD_PKTS_B",CAM_NUM_BADPKTS_B},
      {"CAM_TEST_DATA",CAM_TEST_DATA},
      {"CAM_FPE_WDATA_BAR",CAM_FPE_WDATA_BAR},
      {"CAM_FPE_WCTRL",CAM_FPE_WCTRL},
      {"CAM_FPE_RCTRL",CAM_FPE_RCTRL},
      {"CAM_FPE_BITFILE_SIZE",CAM_FPE_BITFILE_SIZE},
      {"CAM_SPM0_CTRL",CAM_SPM0_CTRL},
      {"CAM_SPM0_CURRENT_INT",CAM_SPM0_CURRENT_INT},
      {"CAM_SPM0_FRAME_START",CAM_SPM0_FRAME_START},
      {"CAM_SPM0_FRAME_END",CAM_SPM0_FRAME_END},
      {"CAM_SPM0_BM_NUM_PXLS",CAM_SPM0_BM_NUM_PXLS},
      {"CAM_SPM0_INT_NUM",CAM_SPM0_INT_NUM},
      {"CAM_SPM0_CHUNK_NUM",CAM_SPM0_CHUNK_NUM},
      {"CAM_SPM0_PAUSE_NUM",CAM_SPM0_PAUSE_NUM},
      {"CAM_SPM0_SATURATION",CAM_SPM0_SATURATION},
      {"CAM_SPM0_BM_BAR",CAM_SPM0_BM_BAR},
      {"CAM_SPM0_MIN_BAR",CAM_SPM0_MIN_BAR},
      {"CAM_SPM0_MAX_BAR",CAM_SPM0_MAX_BAR},
      {"CAM_SPM0_BAR_DATA_BUFA",CAM_SPM0_BAR_DATA_BUFA},
      {"CAM_SPM0_BAR_DATA_BUFB",CAM_SPM0_BAR_DATA_BUFB},

      {"CAM_SPM1_CTRL",CAM_SPM1_CTRL},
      {"CAM_SPM1_CURRENT_INT",CAM_SPM1_CURRENT_INT},
      {"CAM_SPM1_FRAME_START",CAM_SPM1_FRAME_START},
      {"CAM_SPM1_FRAME_END",CAM_SPM1_FRAME_END},
      {"CAM_SPM1_BM_NUM_PXLS",CAM_SPM1_BM_NUM_PXLS},
      {"CAM_SPM1_CHUNK_NUM",CAM_SPM1_CHUNK_NUM},
      {"CAM_SPM1_PAUSE_NUM",CAM_SPM1_PAUSE_NUM},
      {"CAM_SPM1_SATURATION",CAM_SPM1_SATURATION},
      {"CAM_SPM1_BM_BAR",CAM_SPM1_BM_BAR},
      {"CAM_SPM1_MIN_BAR",CAM_SPM1_MIN_BAR},
      {"CAM_SPM1_MAX_BAR",CAM_SPM1_MAX_BAR},
      {"CAM_SPM1_BAR_DATA_BUFA",CAM_SPM1_BAR_DATA_BUFA},
      {"CAM_SPM1_BAR_DATA_BUFB",CAM_SPM1_BAR_DATA_BUFB},
      {"CAM_SPM2_CTRL",CAM_SPM2_CTRL},
      {"CAM_SPM2_CURRENT_INT",CAM_SPM2_CURRENT_INT},
      {"CAM_SPM2_FRAME_START",CAM_SPM2_FRAME_START},
      {"CAM_SPM2_FRAME_END",CAM_SPM2_FRAME_END},
      {"CAM_SPM2_BM_NUM_PXLS",CAM_SPM2_BM_NUM_PXLS},
      {"CAM_SPM2_INT_NUM",CAM_SPM2_INT_NUM},
      {"CAM_SPM2_CHUNK_NUM",CAM_SPM2_CHUNK_NUM},
      {"CAM_SPM2_PAUSE_NUM",CAM_SPM2_PAUSE_NUM},
      {"CAM_SPM2_SATURATION",CAM_SPM2_SATURATION},
      {"CAM_SPM2_BM_BAR",CAM_SPM2_BM_BAR},
      {"CAM_SPM2_MIN_BAR",CAM_SPM2_MIN_BAR},
      {"CAM_SPM2_MAX_BAR",CAM_SPM2_MAX_BAR},
      {"CAM_SPM2_BAR_DATA_BUFA",CAM_SPM2_BAR_DATA_BUFA},
      {"CAM_SPM2_BAR_DATA_BUFB",CAM_SPM2_BAR_DATA_BUFB},
      {"CAM_SPM3_CTRL",CAM_SPM3_CTRL},
      {"CAM_SPM3_CURRENT_INT",CAM_SPM3_CURRENT_INT},
      {"CAM_SPM3_FRAME_START",CAM_SPM3_FRAME_START},
      {"CAM_SPM3_FRAME_END",CAM_SPM3_FRAME_END},
      {"CAM_SPM3_BM_NUM_PXLS",CAM_SPM3_BM_NUM_PXLS},
      {"CAM_SPM3_INT_NUM",CAM_SPM3_INT_NUM},
      {"CAM_SPM3_CHUNK_NUM",CAM_SPM3_CHUNK_NUM},
      {"CAM_SPM3_PAUSE_NUM",CAM_SPM3_PAUSE_NUM},
      {"CAM_SPM3_SATURATION",CAM_SPM3_SATURATION},
      {"CAM_SPM3_BM_BAR",CAM_SPM3_BM_BAR},
      {"CAM_SPM3_MIN_BAR",CAM_SPM3_MIN_BAR},
      {"CAM_SPM3_MAX_BAR",CAM_SPM3_MAX_BAR},
      {"CAM_SPM3_BAR_DATA_BUFA",CAM_SPM3_BAR_DATA_BUFA},
      {"CAM_SPM3_BAR_DATA_BUFB",CAM_SPM3_BAR_DATA_BUFB},
      {"CAM_SPM4_CTRL",CAM_SPM4_CTRL},
      {"CAM_SPM4_CURRENT_INT",CAM_SPM4_CURRENT_INT},
      {"CAM_SPM4_FRAME_START",CAM_SPM4_FRAME_START},
      {"CAM_SPM4_FRAME_END",CAM_SPM4_FRAME_END},
            {"CAM_SPM4_BM_NUM_PXLS",CAM_SPM4_BM_NUM_PXLS},
            {"CAM_SPM4_INT_NUM",CAM_FPE_HSK},
            {"CAM_SPM4_CHUNK_NUM",CAM_SPM4_CHUNK_NUM},
            {"CAM_SPM4_PAUSE_NUM",CAM_SPM4_PAUSE_NUM},
            {"CAM_SPM4_SATURATION",CAM_SPM4_SATURATION},
            {"CAM_SPM4_BM_BAR",CAM_SPM4_BM_BAR},
            {"CAM_SPM4_MIN_BAR",CAM_SPM4_MIN_BAR},
            {"CAM_SPM4_MAX_BAR",CAM_SPM4_MAX_BAR},
            {"CAM_SPM4_BAR_DATA_BUFA",CAM_SPM4_BAR_DATA_BUFA},
            {"CAM_SPM4_BAR_DATA_BUFB",CAM_SPM4_BAR_DATA_BUFB},
            {"CAM_FPE_HSK",CAM_FPE_HSK},  // 0x600  - 0x67F  Base of 127 registers
            {"CAM_FPE_MEM_DUMP",CAM_FPE_MEM_DUMP} // 0x800  - 0xFFF  Base of 2048
      memory dump registers
      };
```

Each line defines a command string and specifies the register referenced by the command. Looking at the first line …

```
      {"CAM_CONTROL", CAM_CONTROL},
```

… this specifies that the command “cam_control” issued from the udp console (netcat) will reference the CAM_CONTROL register. The command(s) can be issued in two forms. To query the current value of the register simply enter the command name (eg “cam_control<enter>”). ObsSim will read the register, format the value as ascii hex, and display the register contents in the form “cam_control = 0x12345678”. To write to a register the command is issued along with “=” followed by <value>. For example “cam_control = 2” would change the contents of the cam_control register for the FPE to 0x00000002.

For information on the details of each register please refer to DHU Firmware Design Specification Dwg. No. 37-14010.