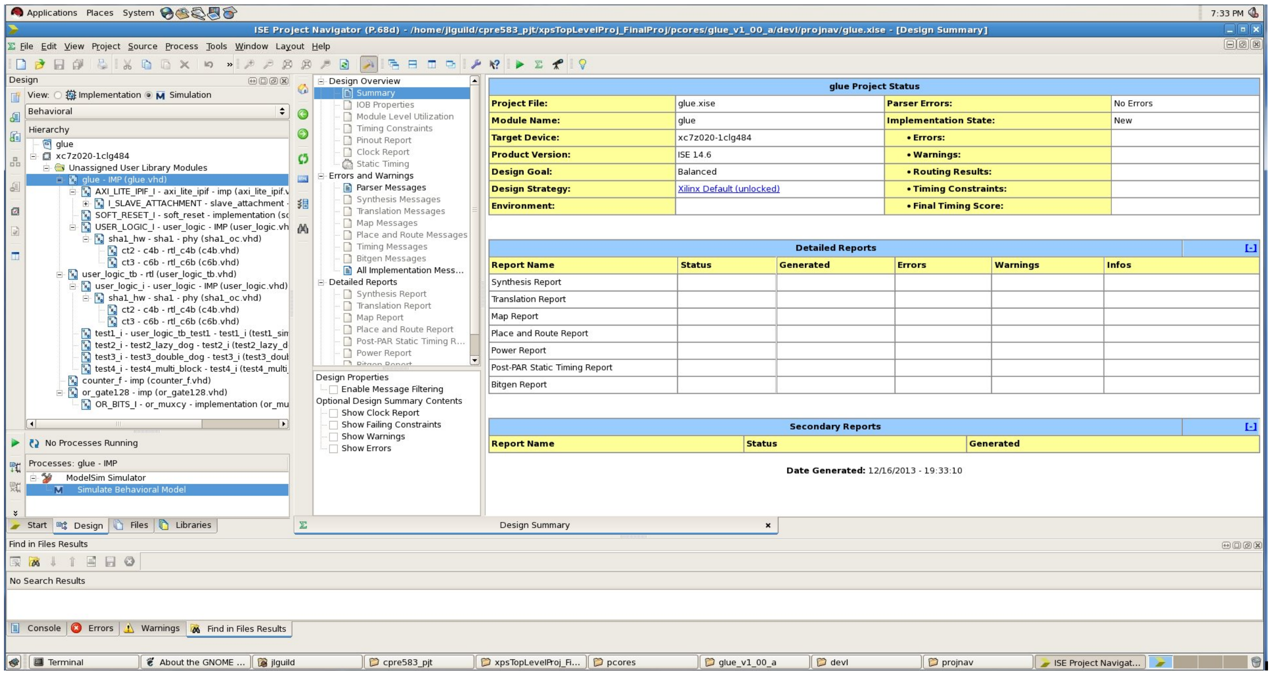
Appendix A - Simulation

Software simulation was done on a Gnome Desktop, Red Hat, Linux system running Xilinx version 14.6 ISE and ModelSIM. To launch the simulation use an emulator in bash mode and browse to the pcores/glue_v1_00_a/devl/projnav folder in the files provided. From here source the setup.sh file and then launch the ISE with the command ise glue.xise
You should have a screen as shown below [Figure A-1]:



ISE Project Navigator (P.68d) - /home/jlguild/cpre583_pjt/xpsTopLevelProj_FinalProj/pcores/glue_v1_00_a/devl/projnav/glue.xise - [Design Summary]

Hierarchy
```
glue
xc7z020-1clg484
  Unassigned User Library Modules
    glue - IMP (glue.vhd)
      AXI_LITE_IPIF_I - axi_lite_ipif - imp (axi_lite_ipif.v
        I_SLAVE_ATTACHMENT - slave_attachment -
      SOFT_RESET_I - soft_reset - implementation (sc
      USER_LOGIC_I - user_logic - IMP (user_logic.vh
        sha1_hw - sha1 - phy (sha1_oc.vhd)
          ct2 - c4b - rtl_c4b (c4b.vhd)
          ct3 - c6b - rtl_c6b (c6b.vhd)
    user_logic_tb - rtl (user_logic_tb.vhd)
      user_logic_i - user_logic - IMP (user_logic.vhd)
        sha1_hw - sha1 - phy (sha1_oc.vhd)
          ct2 - c4b - rtl_c4b (c4b.vhd)
          ct3 - c6b - rtl_c6b (c6b.vhd)
      test1_i - user_logic_tb_test1 - test1_i (test1_sin
      test2_i - test2_lazy_dog - test2_i (test2_lazy_d
      test3_i - test3_double_dog - test3_i (test3_doul
      test4_i - test4_multi_block - test4_i (test4_multi_
    counter_f - imp (counter_f.vhd)
    or_gate128 - imp (or_gate128.vhd)
      OR_BITS_I - or_muxcy - implementation (or_mu
```

No Processes Running

Processes: glue - IMP
```
ModelSim Simulator
  Simulate Behavioral Model
```

```
Design Overview
  Summary
  IOB Properties
  Module Level Utilization
  Timing Constraints
  Pinout Report
  Clock Report
  Static Timing
Errors and Warnings
  Parser Messages
  Synthesis Messages
  Translation Messages
  Map Messages
  Place and Route Messages
  Timing Messages
  Bitgen Messages
  All Implementation Mess...
Detailed Reports
  Synthesis Report
  Translation Report
  Map Report
  Place and Route Report
  Post-PAR Static Timing R...
  Power Report
  Bitgen Report
```

Design Properties
- ☐ Enable Message Filtering

Optional Design Summary Contents
- ☐ Show Clock Report
- ☐ Show Failing Constraints
- ☐ Show Warnings
- ☐ Show Errors

glue Project Status			
Project File:	glue.xise	**Parser Errors:**	No Errors
Module Name:	glue	**Implementation State:**	New
Target Device:	xc7z020-1clg484	**• Errors:**	
Product Version:	ISE 14.6	**• Warnings:**	
Design Goal:	Balanced	**• Routing Results:**	
Design Strategy:	Xilinx Default (unlocked)	**• Timing Constraints:**	
Environment:		**• Final Timing Score:**	

Detailed Reports [-]

Report Name	Status	Generated	Errors	Warnings	Infos
Synthesis Report					
Translation Report					
Map Report					
Place and Route Report					
Power Report					
Post-PAR Static Timing Report					
Bitgen Report					

Secondary Reports [-]

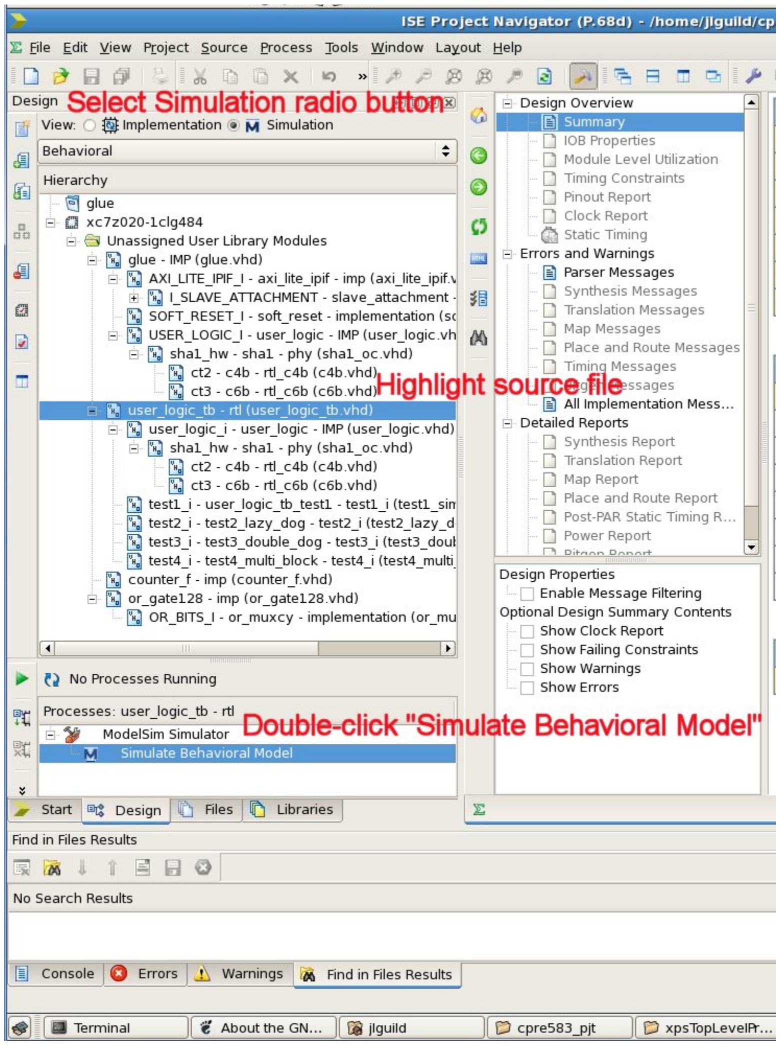
Report Name	Status	Generated

Date Generated: 12/16/2013 - 19:33:10

No Search Results

Figure A-1: ISE Project Explorer

Verify all items are listed and have no conflicts.
To run the simulation and observe results select the simulation radio button and highlight the source labeled user_logic_tb - rtl (user_logic_tb.vhd) [Figure A-2]. Then double-click the Simulate Behavioral Model in the Processes listing


ISE Project Navigator (P.68d) - /home/jlguild/cp
File Edit View Project Source Process Tools Window Layout Help
Select Simulation radio button
Highlight source file
Double-click "Simulate Behavioral Model"
Start
Design
Files
Libraries
Console
Errors
Warnings
Find in Files Results
Terminal
About the GN...
jlguild
cpre583_pjt
xpsTopLevelPr...

Design
View: ○ Implementation ◉ Simulation
Behavioral

Hierarchy

```
glue
xc7z020-1clg484
  Unassigned User Library Modules
    glue - IMP (glue.vhd)
      AXI_LITE_IPIF_I - axi_lite_ipif - imp (axi_lite_ipif.v
        I_SLAVE_ATTACHMENT - slave_attachment -
      SOFT_RESET_I - soft_reset - implementation (sc
      USER_LOGIC_I - user_logic - IMP (user_logic.vh
        sha1_hw - sha1 - phy (sha1_oc.vhd)
          ct2 - c4b - rtl_c4b (c4b.vhd)
          ct3 - c6b - rtl_c6b (c6b.vhd)
    user_logic_tb - rtl (user_logic_tb.vhd)
      user_logic_i - user_logic - IMP (user_logic.vhd)
        sha1_hw - sha1 - phy (sha1_oc.vhd)
          ct2 - c4b - rtl_c4b (c4b.vhd)
          ct3 - c6b - rtl_c6b (c6b.vhd)
      test1_i - user_logic_tb_test1 - test1_i (test1_sim
      test2_i - test2_lazy_dog - test2_i (test2_lazy_d
      test3_i - test3_double_dog - test3_i (test3_doub
      test4_i - test4_multi_block - test4_i (test4_multi_
    counter_f - imp (counter_f.vhd)
    or_gate128 - imp (or_gate128.vhd)
      OR_BITS_I - or_muxcy - implementation (or_mu
```

No Processes Running

Processes: user_logic_tb - rtl

```
ModelSim Simulator
  Simulate Behavioral Model
```

```
Design Overview
  Summary
  IOB Properties
  Module Level Utilization
  Timing Constraints
  Pinout Report
  Clock Report
  Static Timing
Errors and Warnings
  Parser Messages
  Synthesis Messages
  Translation Messages
  Map Messages
  Place and Route Messages
  Timing Messages
  ...essages
  All Implementation Mess...
Detailed Reports
  Synthesis Report
  Translation Report
  Map Report
  Place and Route Report
  Post-PAR Static Timing R...
  Power Report
  Bitgen Report
```

Design Properties
- [ ] Enable Message Filtering

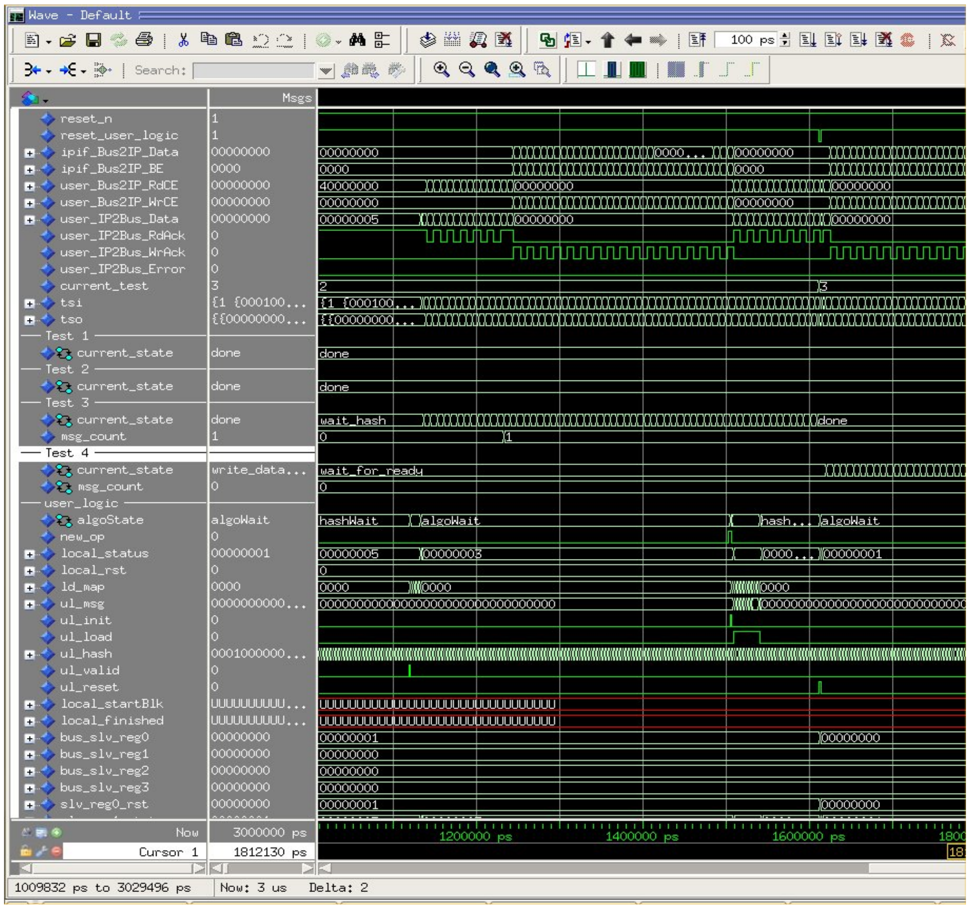
Optional Design Summary Contents
- [ ] Show Clock Report
- [ ] Show Failing Constraints
- [ ] Show Warnings
- [ ] Show Errors

Find in Files Results

No Search Results

Figure A-2: Running Simulation

Here the simulation will run and give the signal wave results similar to this [Figure A-3]:

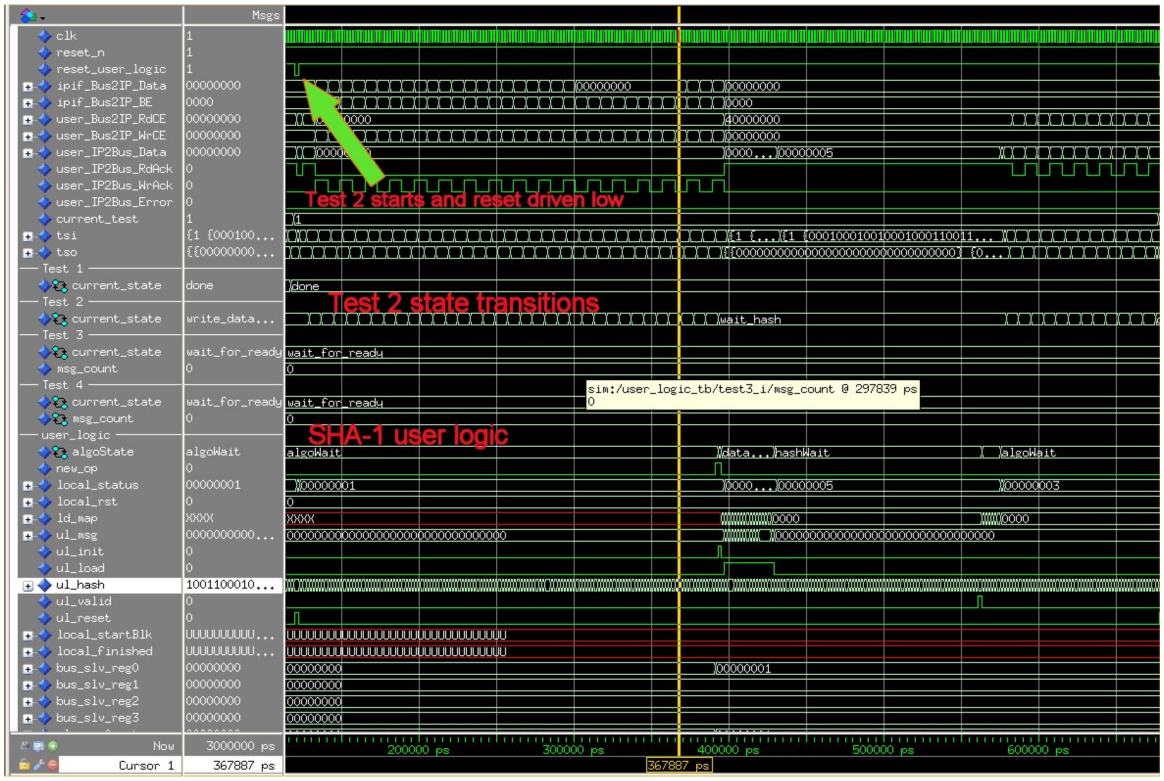


```
Wave - Default
                                   Msgs
    reset_n                        1
    reset_user_logic               1
+   ipif_Bus2IP_Data               00000000
+   ipif_Bus2IP_BE                 0000
+   user_Bus2IP_RdCE               00000000
+   user_Bus2IP_WrCE               00000000
+   user_IP2Bus_Data               00000000
    user_IP2Bus_RdAck              0
    user_IP2Bus_WrAck              0
    user_IP2Bus_Error              0
    current_test                   3
+   tsi                            {1 {000100...
+   tso                            {{00000000...
— Test 1 —
    current_state                  done
— Test 2 —
    current_state                  done
— Test 3 —
    current_state                  done
    msg_count                      1
— Test 4 —
    current_state                  write_data...
    msg_count                      0
— user_logic —
    algoState                      algoWait
    new_op                         0
+   local_status                   00000001
+   local_rst                      0
+   ld_map                         0000
+   ul_msg                         0000000000...
    ul_init                        0
    ul_load                        0
+   ul_hash                        0001000000...
    ul_valid                       0
    ul_reset                       0
+   local_startBlk                 UUUUUUUUUU...
+   local_finished                 UUUUUUUUUU...
+   bus_slv_reg0                   00000000
+   bus_slv_reg1                   00000000
+   bus_slv_reg2                   00000000
+   bus_slv_reg3                   00000000
+   slv_reg0_rst                   00000000

                          Now      3000000 ps
                     Cursor 1      1812130 ps

1009832 ps to 3029496 ps    Now: 3 us    Delta: 2
```

Figure A-3: Initial Waveform

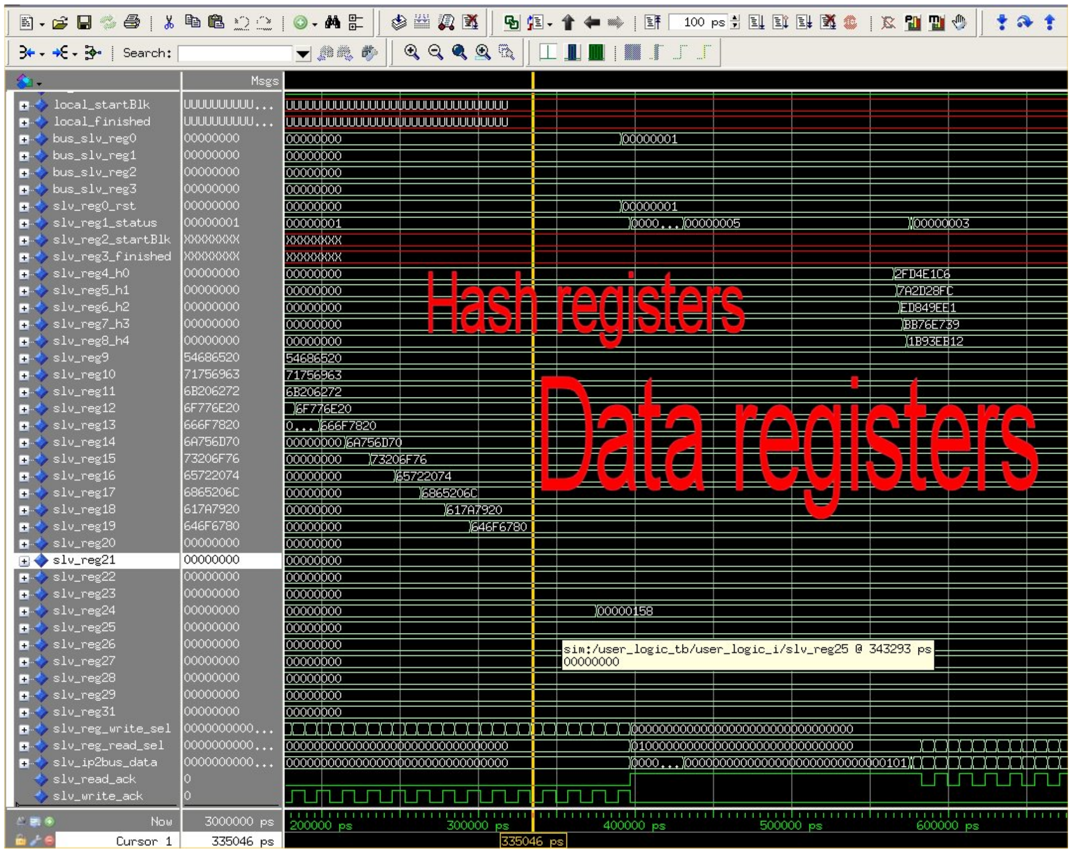
This waveform will have the initial setup of the architecture as well as all four tests . It runs for a total of 3 microseconds and contains all data registers and signals. The IP bus sends data back and forth from the micro to the FPGA doing the SHA1 encryption, current_test tracks the test being run at any given time -1, and the current_state under each test label signals where in the state machine the current test is. The user_logic maintains the local micro signals and registers with data being written for each test to the slv_reg9 through slv_reg25 registers. The resulting hash calculation is written to registers slv_reg4_h0 through slv_reg8_h4 and compared to the independently calculated hash to verify correctness. The independently calculated hash was done using the Linux command-line tool ‘sha1sum’.

Once the hash is validated the simulation starts the next test and resets the micro.


Test 2 starts and reset driven low
Test 2 state transitions
SHA-1 user logic
sim:/user_logic_tb/test3_i/msg_count @ 297839 ps
0

```
clk                     1
reset_n                 1
reset_user_logic        1
ipif_Bus2IP_Data        00000000
ipif_Bus2IP_BE          0000
user_Bus2IP_RdCE        00000000
user_Bus2IP_WrCE        00000000
user_IP2Bus_Data        00000000
user_IP2Bus_RdAck       0
user_IP2Bus_WrAck       0
user_IP2Bus_Error       0
current_test            1
tsi                     {1 {000100...
tso                     {{00000000...
Test 1
  current_state         done
Test 2
  current_state         write_data...
Test 3
  current_state         wait_for_ready
  msg_count             0
Test 4
  current_state         wait_for_ready
  msg_count             0
user_logic
  algoState             algoWait
new_op                  0
local_status            00000001
local_rst               0
ld_map                  XXXX
ul_msg                  000000000...
ul_init                 0
ul_load                 0
ul_hash                 1001100010...
ul_valid                0
ul_reset                0
local_startBlk          UUUUUUUUUU...
local_finished          UUUUUUUUUU...
bus_slv_reg0            00000000
bus_slv_reg1            00000000
bus_slv_reg2            00000000
bus_slv_reg3            00000000
Now                     3000000 ps
Cursor 1                367887 ps
```

Figure A-4: Test State Signals



```
local_startBlk          UUUUUUUUUU...
local_finished          UUUUUUUUUU...
bus_slv_reg0            00000000
bus_slv_reg1            00000000
bus_slv_reg2            00000000
bus_slv_reg3            00000000
slv_reg0_rst            00000000
slv_reg1_status         00000001
slv_reg2_startBlk       XXXXXXXX
slv_reg3_finished       XXXXXXXX
slv_reg4_h0             00000000
slv_reg5_h1             00000000
slv_reg6_h2             00000000
slv_reg7_h3             00000000
slv_reg8_h4             00000000
slv_reg9                54686520
slv_reg10               71756963
slv_reg11               6B206272
slv_reg12               6F776E20
slv_reg13               666F7820
slv_reg14               6A756D70
slv_reg15               73206F76
slv_reg16               65722074
slv_reg17               6865206C
slv_reg18               617A7920
slv_reg19               646F6780
slv_reg20               00000000
slv_reg21               00000000
slv_reg22               00000000
slv_reg23               00000000
slv_reg24               00000000
slv_reg25               00000000
slv_reg26               00000000
slv_reg27               00000000
slv_reg28               00000000
slv_reg29               00000000
slv_reg31               00000000
slv_reg_write_sel       0000000000...
slv_reg_read_sel        0000000000...
slv_ip2bus_data         0000000000...
slv_read_ack            0
slv_write_ack           0
Now                     3000000 ps
Cursor 1                335046 ps
```

Figure A-5: Registers

Measurements for process time were taken at the test state transitions for metrics. Some tests revisited a state and the time for both periods were added together for the data sets. [Table A-1]

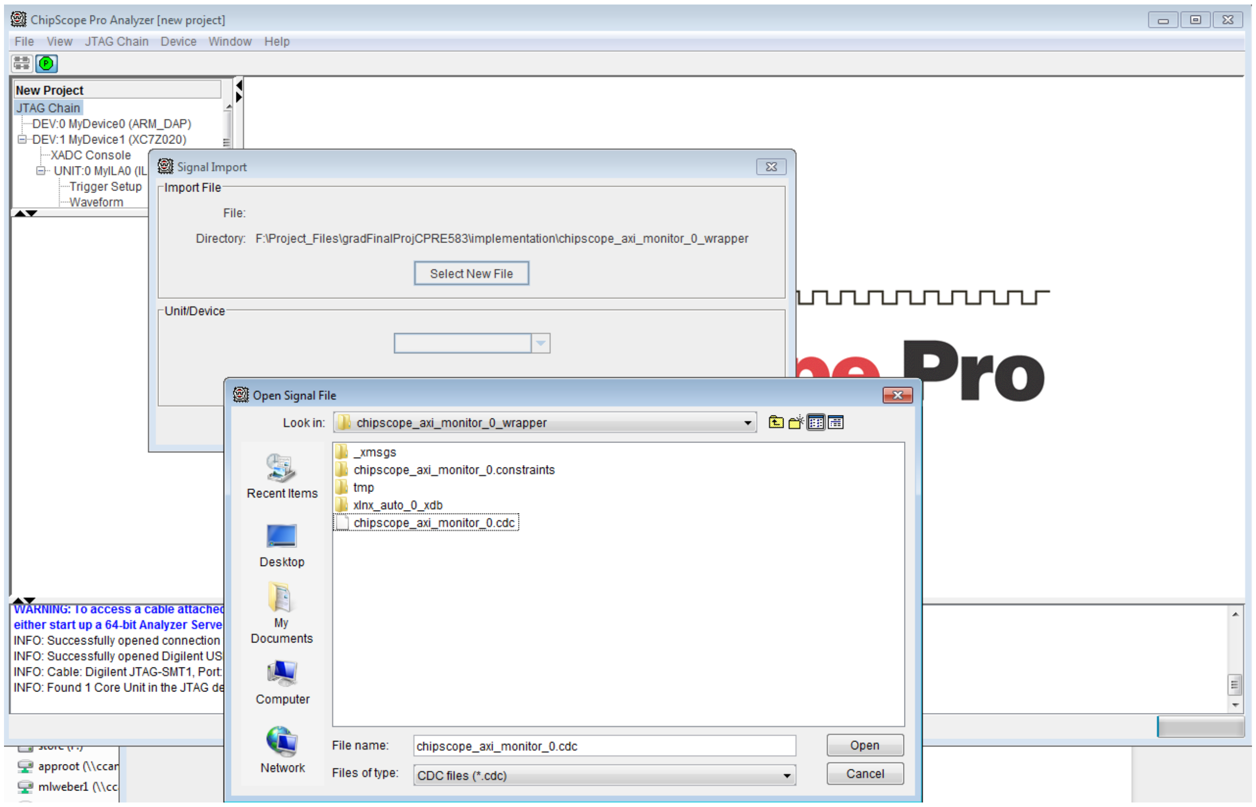
	Test 1	Test 2	Test 3	Test 4
Test Type	Word Write	Single Chunk (4bit)	Single Chunk(6bit)x2	Multi-Chunk (6bit)
String Length (Bytes)	4	44	44	88
Process Start (ns)	101	119	677	1611
Process End (ns)	118	676	1610	2476
Process Length (ns)	17	557	933	865
Process Length (ns)/ 4 Bytes	17	50.64	42.41	39.32
Bandwidth (MB/s)	235.29	78.99	94.32	101.73
Reset start (ns)		120	678	1612
Rest end (ns)		129	687	1621
Reset Length (ns)		9	9	9
% Process time		1.62	0.96	1.04
Start write (ns)	101	129	687	1621
End write (ns)	109	385	943	1877
Start write (ns)(2)			1241	1927
End write (ns)(2)			1497	2183
Write Length (ns)	8	256	512	512
Write Length (ns)/ 4 Bytes	8	23.27	46.55	23.27
% Process time	47.06	45.96	54.88	59.19
Begin hash (ns)		385	943	1877
End hash (ns)		579	1137	1927

Begin hash (ns)(2)			1497	2183
End hash (ns)(2)			1513	2379
Hash Length (ns)		194	210	246
Hash Length (ns)/ 4 Bytes		17.64	19.09	22.36
% Process time		34.83	22.51	28.44
Start validate (ns)	109	587	1145	2387
End validate (ns)	117	675	1233	2475
Start validate (ns)(2)			1521	
End validate (ns)(2)			1609	
Validate Length (ns)	8	88	176	88
Validate Length (ns)/ 4 Bytes	8	8	16	4
% Process time	47.06	15.80	18.86	10.17
Added overhead/4 Bytes (ns)	0.000	33.636	25.409	22.318
% increase process time	0	198	149	131

Table A-1: Metric Data Analysis

Appendix B - Chipscope

The initial chipscope setup involve connecting the JTAG to the target hardware and scanning for chipscope monitors. In this case we have a AXI bus monitor added to our design.



New Project
JTAG Chain
DEV:0 MyDevice0 (ARM_DAP)
DEV:1 MyDevice1 (XC7Z020)
XADC Console
UNIT:0 MyILA0 (IL
Trigger Setup
Waveform

WARNING: To access a cable attached
either start up a 64-bit Analyzer Serve
INFO: Successfully opened connection
INFO: Successfully opened Digilent US
INFO: Cable: Digilent JTAG-SMT1, Port
INFO: Found 1 Core Unit in the JTAG de

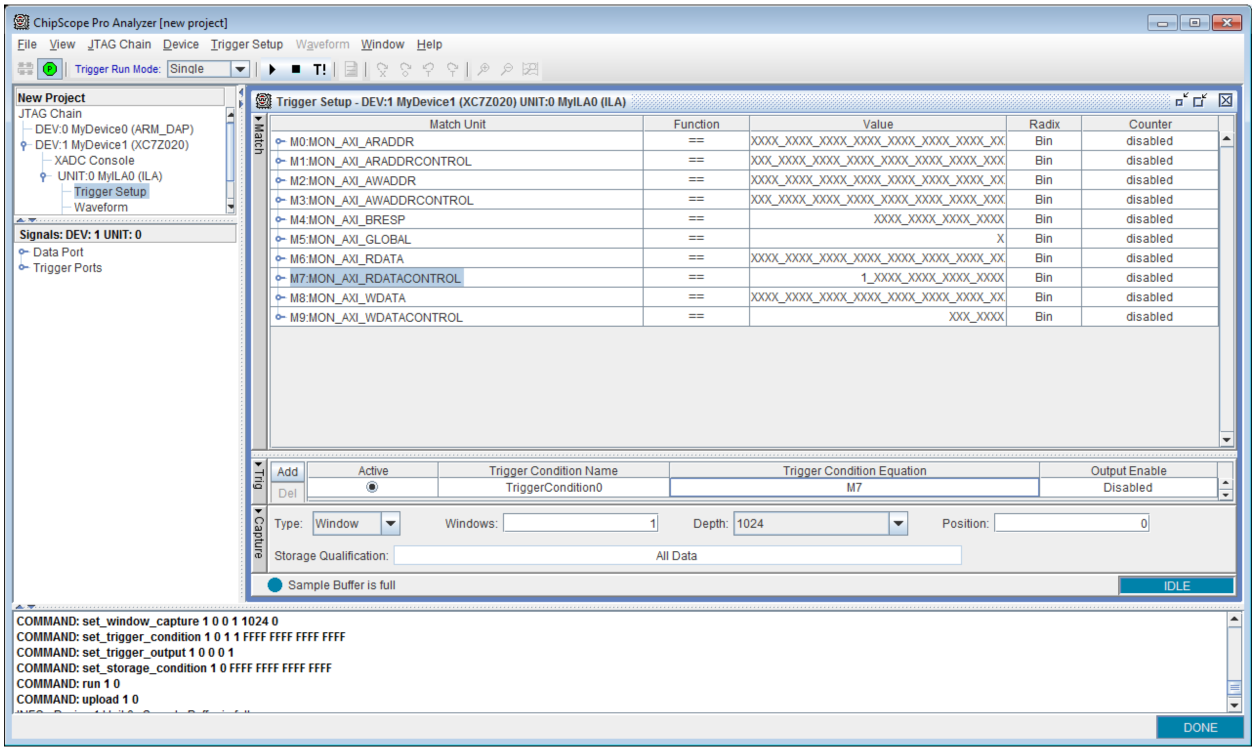
Import File
File:
Directory: F:\Project_Files\gradFinalProjCPRE583\implementation\chipscope_axi_monitor_0_wrapper
Unit/Device

Look in: chipscope_axi_monitor_0_wrapper
_xmsgs
chipscope_axi_monitor_0.constraints
tmp
xlnx_auto_0_xdb
chipscope_axi_monitor_0.cdc
File name: chipscope_axi_monitor_0.cdc
Files of type: CDC files (*.cdc)

approot (\\ccar
mlweberl (\\cc

Figure B-1 : Chipscope signal setup

The cdc file referenced in Figure B-1 is provided in the <git repo>/chipscope/ folder. A .cpj file is also checked in with the cdc and includes the signal renaming captured in the other screenshots.


ChipScope Pro Analyzer [new project]
File View JTAG Chain Device Trigger Setup Waveform Window Help
Trigger Run Mode: Single
New Project
JTAG Chain
DEV:0 MyDevice0 (ARM_DAP)
DEV:1 MyDevice1 (XC7Z020)
XADC Console
UNIT:0 MyILA0 (ILA)
Trigger Setup
Waveform
Signals: DEV: 1 UNIT: 0
Data Port
Trigger Ports
Trigger Setup - DEV:1 MyDevice1 (XC7Z020) UNIT:0 MyILA0 (ILA)
Match
Trig
Capture
Add
Del
Sample Buffer is full
IDLE
DONE

Match Unit	Function	Value	Radix	Counter
M0:MON_AXI_ARADDR	==	XXXX_XXXX_XXXX_XXXX_XXXX_XXXX_XXXX_XX	Bin	disabled
M1:MON_AXI_ARADDRCONTROL	==	XXX_XXXX_XXXX_XXXX_XXXX_XXXX_XXXX_XXX	Bin	disabled
M2:MON_AXI_AWADDR	==	XXXX_XXXX_XXXX_XXXX_XXXX_XXXX_XXXX_XX	Bin	disabled
M3:MON_AXI_AWADDRCONTROL	==	XXX_XXXX_XXXX_XXXX_XXXX_XXXX_XXXX_XXX	Bin	disabled
M4:MON_AXI_BRESP	==	XXXX_XXXX_XXXX_XXXX	Bin	disabled
M5:MON_AXI_GLOBAL	==	X	Bin	disabled
M6:MON_AXI_RDATA	==	XXXX_XXXX_XXXX_XXXX_XXXX_XXXX_XXXX_XX	Bin	disabled
M7:MON_AXI_RDATACONTROL	==	1_XXXX_XXXX_XXXX_XXXX	Bin	disabled
M8:MON_AXI_WDATA	==	XXXX_XXXX_XXXX_XXXX_XXXX_XXXX_XXXX_XX	Bin	disabled
M9:MON_AXI_WDATACONTROL	==	XXX_XXXX	Bin	disabled

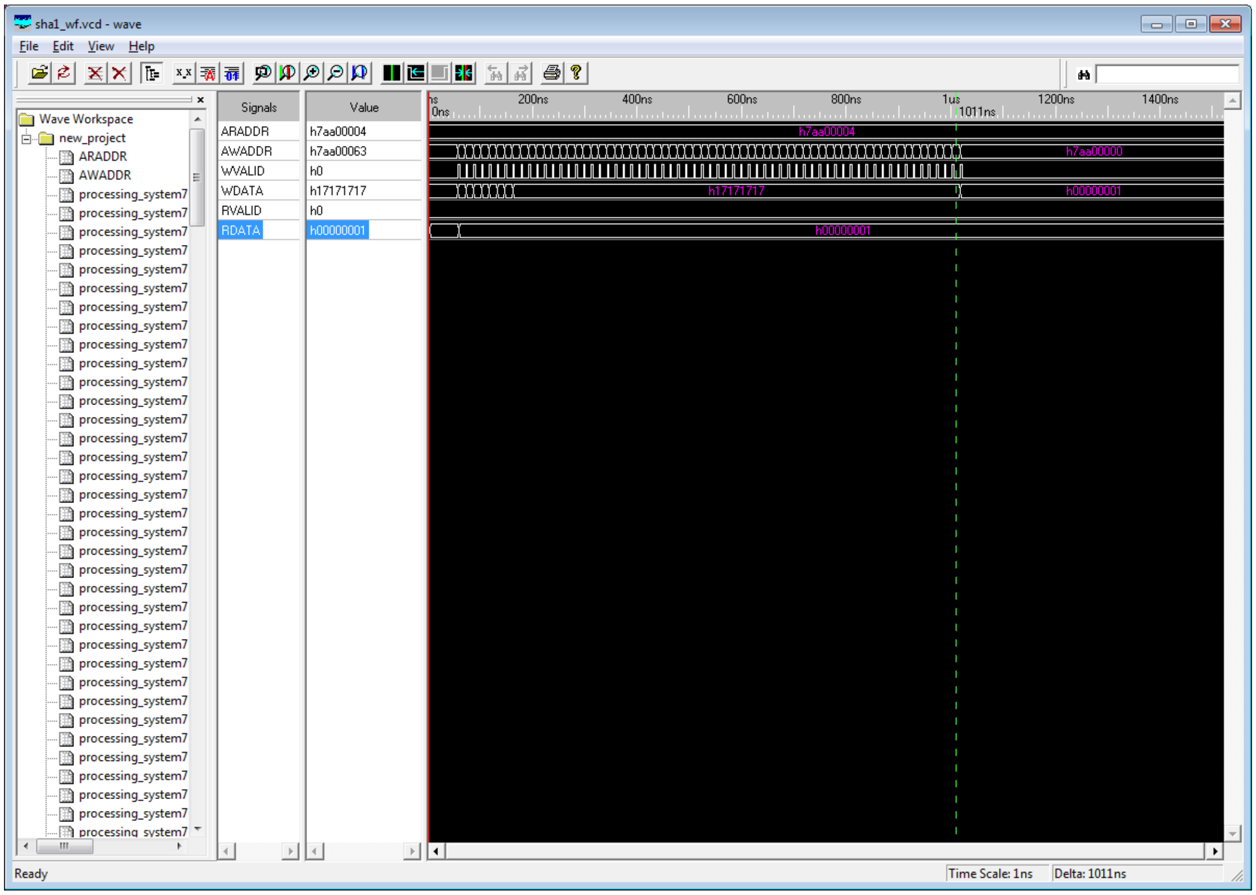
Active	Trigger Condition Name	Trigger Condition Equation	Output Enable
●	TriggerCondition0	M7	Disabled

Type: Window
Windows: 1
Depth: 1024
Position: 0
Storage Qualification: All Data

```
COMMAND: set_window_capture 1 0 0 1 1024 0
COMMAND: set_trigger_condition 1 0 1 1 FFFF FFFF FFFF FFFF
COMMAND: set_trigger_output 1 0 0 0 1
COMMAND: set_storage_condition 1 0 FFFF FFFF FFFF FFFF
COMMAND: run 1 0
COMMAND: upload 1 0
```

Figure B-2 : Chipscope trigger setup on read valid signal

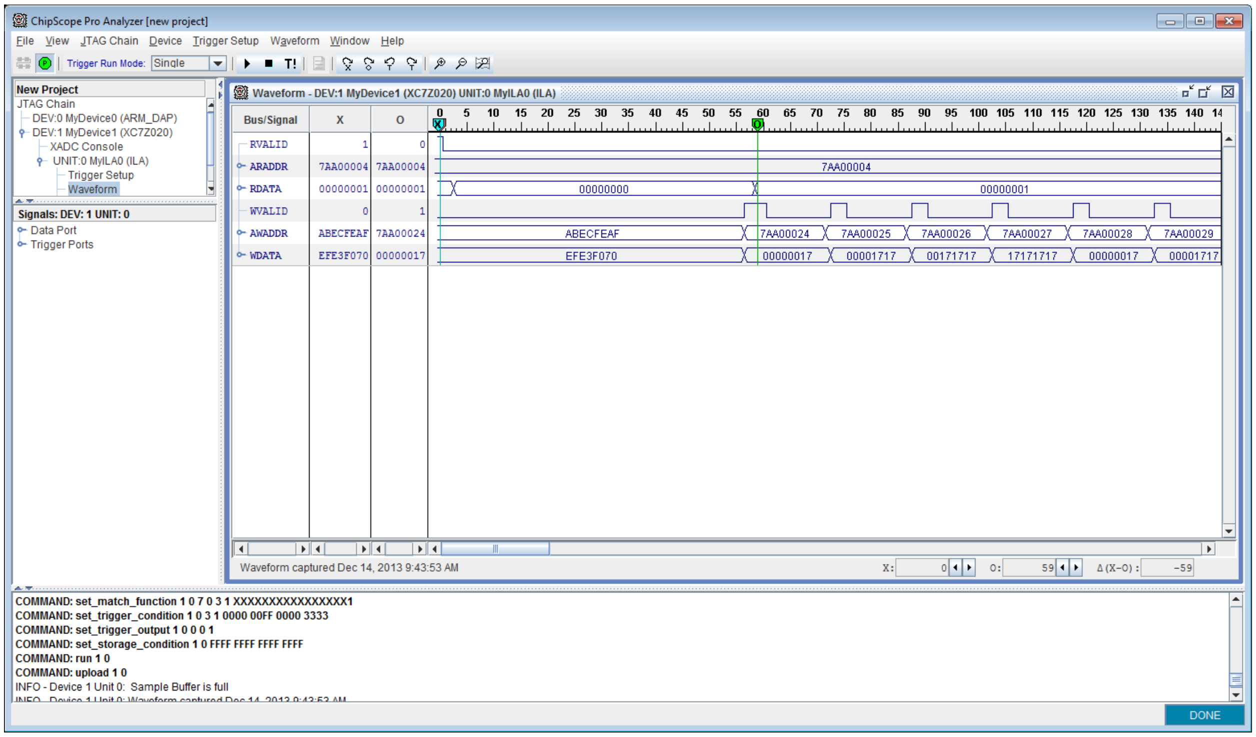
The first step after scanning the bus and loading the signal names, is to configure the trigger signals (Figure B-2). To capture our start scenario the read valid worked to trigger on the initial software read of the userlogic status register.



Signals	Value
ARADDR	h7aa00004
AWADDR	h7aa00063
WVALID	h0
WDATA	h17171717
RVALID	h0
RDATA	h00000001

Figure B-3: Post analysis software

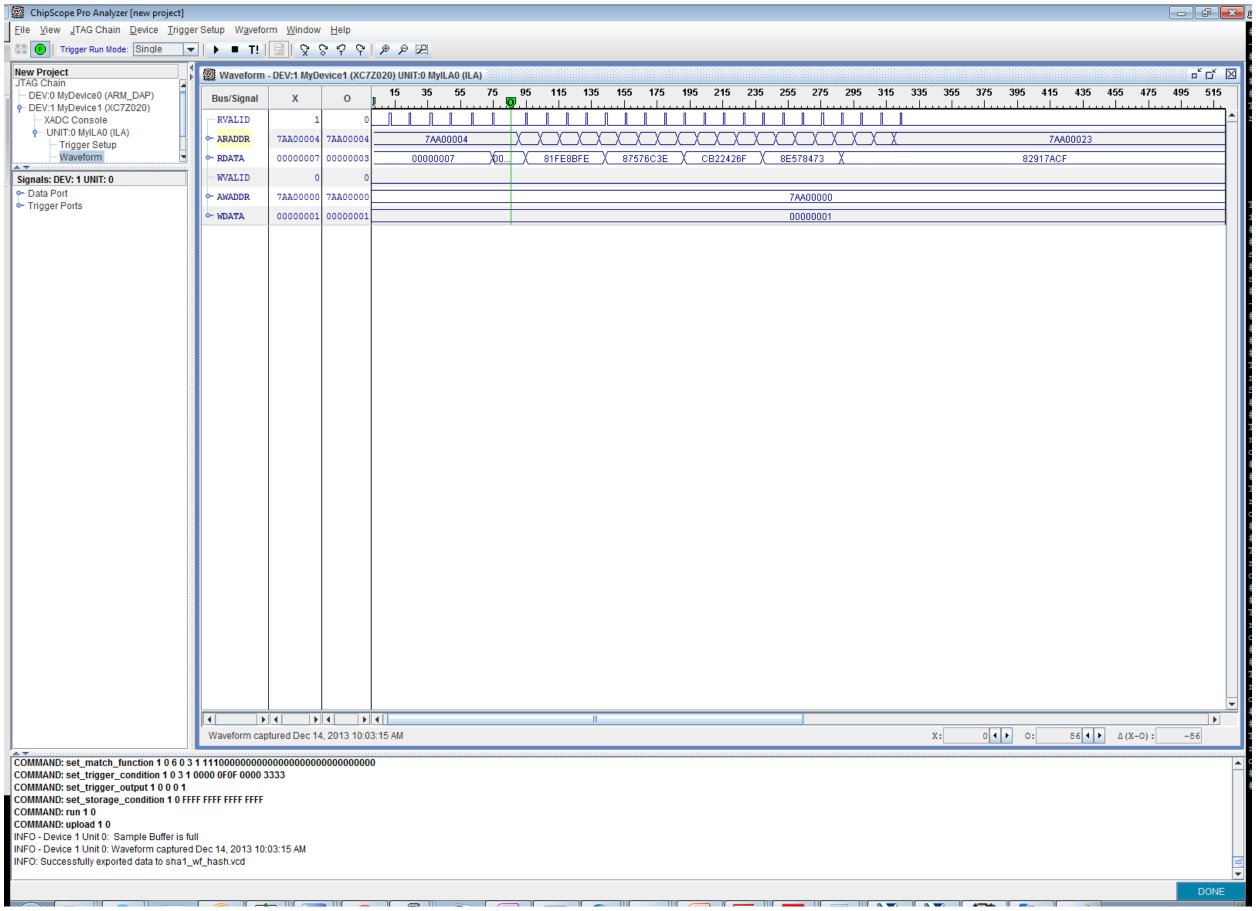
For any post signal analysis we used wavevcd viewer (http://www.iss-us.com/wavevcd/Wave_msi.zip). The vcd files are included in our project submission/chipscope or <git repo>/chipscope folder. Figure B-3 shows the initial software status read and then the write of 64bytes (a block) of msg data to be hashed. Another view of the same sequence (in chipscope) is found in Figure B-4. The spacing between blocks was larger than our chipscope window so the end of the hash sequence was captured in another sequence (Figure B-5).


ChipScope Pro Analyzer [new project]
File View JTAG Chain Device Trigger Setup Waveform Window Help
Trigger Run Mode: Single
New Project
JTAG Chain
DEV:0 MyDevice0 (ARM_DAP)
DEV:1 MyDevice1 (XC7Z020)
XADC Console
UNIT:0 MyILA0 (ILA)
Trigger Setup
Waveform
Signals: DEV: 1 UNIT: 0
Data Port
Trigger Ports
Waveform - DEV:1 MyDevice1 (XC7Z020) UNIT:0 MyILA0 (ILA)
Waveform captured Dec 14, 2013 9:43:53 AM
X: 0
O: 59
Δ (X-O) : -59
DONE

Bus/Signal	X	O
RVALID	1	0
ARADDR	7AA00004	7AA00004
RDATA	00000001	00000001
WVALID	0	1
AWADDR	ABECFEAF	7AA00024
WDATA	EFE3F070	00000017

```
COMMAND: set_match_function 1 0 7 0 3 1 XXXXXXXXXXXXXXXX1
COMMAND: set_trigger_condition 1 0 3 1 0000 00FF 0000 3333
COMMAND: set_trigger_output 1 0 0 0 1
COMMAND: set_storage_condition 1 0 FFFF FFFF FFFF FFFF
COMMAND: run 1 0
COMMAND: upload 1 0
INFO - Device 1 Unit 0:  Sample Buffer is full
INFO - Device 1 Unit 0: Waveform captured Dec 14, 2013 9:43:53 AM
```

Figure B-4: Chipscope trigger on start of a new hash msg



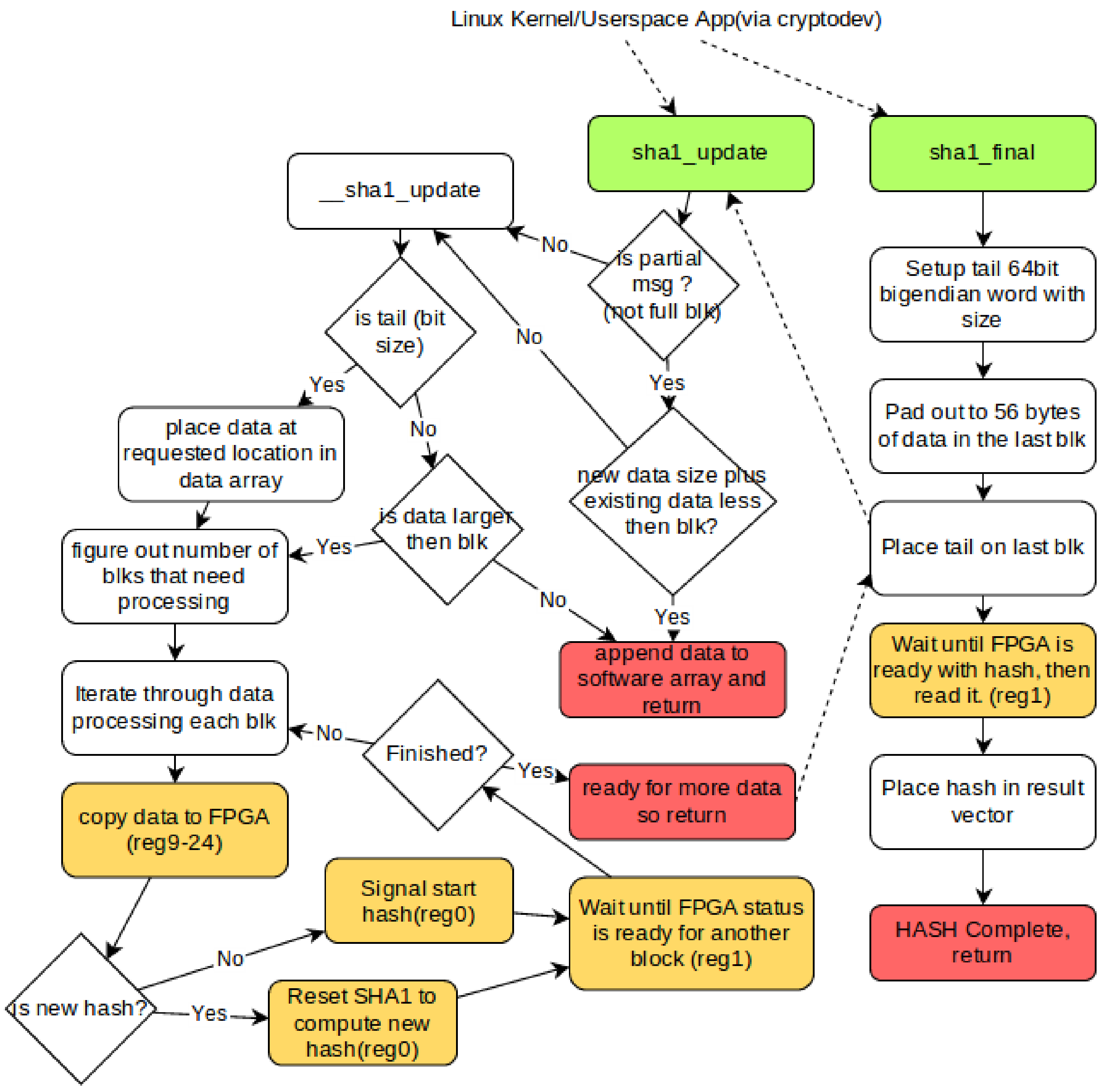
```
Waveform captured Dec 14, 2013 10:03:15 AM
COMMAND: set_match_function 1 0 6 0 3 1 1110000000000000000000000000000
COMMAND: set_trigger_condition 1 0 3 1 0000 0F0F 0000 3333
COMMAND: set_trigger_output 1 0 0 0 1
COMMAND: set_storage_condition 1 0 FFFF FFFF FFFF FFFF
COMMAND: run 1 0
COMMAND: upload 1 0
INFO - Device 1 Unit 0:  Sample Buffer is full
INFO - Device 1 Unit 0: Waveform captured Dec 14, 2013 10:03:15 AM
INFO: Successfully exported data to sha1_wf_hash.vcd
```

Figure B-5: Chipscope trigger on the end of a hash sequence when hash is ready

The end of a data msg hash sequence results in the software waiting for a status signaling hash complete. This is shown in Figure B-5 where the trigger was a read value of 0x7 signaling the software should start reading out the hash value (5 words).

In all the chipscope captures it's fairly obvious that the amount of bus time the software spends checking status could definitely be optimized by utilizing interrupts to request the software to read status when there's a update/change.

Appendix C - Extra Reference Material



Linux Kernel/Userspace App(via cryptodev)
sha1_update
sha1_final
__sha1_update
is partial msg ? (not full blk)
No
Yes
is tail (bit size)
Yes
No
place data at requested location in data array
new data size plus existing data less then blk?
Yes
No
is data larger then blk
Yes
No
figure out number of blks that need processing
append data to software array and return
Iterate through data processing each blk
Finished?
No
Yes
ready for more data so return
copy data to FPGA (reg9-24)
Signal start hash(reg0)
Wait until FPGA status is ready for another block (reg1)
is new hash?
No
Yes
Reset SHA1 to compute new hash(reg0)
Setup tail 64bit bigendian word with size
Pad out to 56 bytes of data in the last blk
Place tail on last blk
Wait until FPGA is ready with hash, then read it (reg1)
Place hash in result vector
HASH Complete, return

Figure C-1: SHA1 State Machine

Figure C-1 describes the overall operation of SHA1 PL design. All flow starts from the (top) userspace application running on the ARM. The boxes in green are the kernel function calls that implement the custom SHA1 interfacing to the PL (sha1_update/final as discussed in Section: “PL SHA1 Interface Module”). The boxes in yellow are interactions with the PL register map (FSM). The boxes in red are return/final states. Bringing this all together, the diagram shows the software control access and PL status interactions, as well as the data load and hash retrieval steps.

Address	Name	Reset Value	Notes
0x7AA00000	ZYNQ_SHA1_RST	0x0	Controls: start has of new msg (0x2), continue current hash(0x1)
0x7AA00004	ZYNQ_SHA1_STATUS	0x0	Status of userlogic FSM: ready for data input (0x1), Hash ready (0x2), hash busy (0x4)
0x7AA00010	ZYNQ_SHA1_H0	0x0	First word of 20 byte hash
0x7AA00014	ZYNQ_SHA1_H1	0x0	
0x7AA00018	ZYNQ_SHA1_H2	0x0	
0x7AA0001C	ZYNQ_SHA1_H3	0x0	
0x7AA00020	ZYNQ_SHA1_H4	0x0	Last work of 20byte hash
0x7AA00024	ZYNQ_SHA1_D0	0x0	First word of 64byte block
0x7AA00028	ZYNQ_SHA1_D1	0x0	
0x7AA0002C	ZYNQ_SHA1_D2	0x0	
0x7AA00030	ZYNQ_SHA1_D3	0x0	
0x7AA00034	ZYNQ_SHA1_D4	0x0	
0x7AA00038	ZYNQ_SHA1_D5	0x0	
0x7AA0003C	ZYNQ_SHA1_D6	0x0	
0x7AA00040	ZYNQ_SHA1_D7	0x0	
0x7AA00044	ZYNQ_SHA1_D8	0x0	
0x7AA00048	ZYNQ_SHA1_D9	0x0	
0x7AA0004C	ZYNQ_SHA1_D10	0x0	
0x7AA00050	ZYNQ_SHA1_D11	0x0	
0x7AA00054	ZYNQ_SHA1_D12	0x0	
0x7AA00058	ZYNQ_SHA1_D13	0x0	
0x7AA0005C	ZYNQ_SHA1_D14	0x0	
0x7AA00060	ZYNQ_SHA1_D15	0x0	Last word of 64byte block

Table C-1: Control/status/data registers

Resource Links:

Notes on SHA1 - Overview of padding and tail format
http://www.itl.nist.gov/fipspubs/fip180-1.htm

Crypto dev sha testing example
http://lists.freebsd.org/pipermail/freebsd-security/2013-August/007115.html

Notes on using cryptodev-linux
https://github.com/nmav/cryptodev-linux/blob/master/INSTALL

Notes on zynq fabric loading
http://xlnx.lithium.com/t5/Embedded-Linux/download-bitfile-to-dev-xdevcfg-timeout/m-p/322419#M6137
http://forums.xilinx.com/t5/Embedded-Linux/Zynq-Loading-bitfile-into-FPGA-from-Linux-xdevcfg/td-p/237850/page/3