

Introduction to FPGA Simulation and Debug

Quartus Prime 17.1 Lite

©2018 Intel Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, HARDCOPY, INTEL, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Intel Corporation and registered in the U.S. Patent and Trademark Office and in other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders as described at www.altera.com/common/legal.html. Intel warrants performance of its semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or service.

Revision	Author	Date	Comments
1.0	A. Joshipura	2/2/2018	Initial Version
1.1	A. Joshipura	3/20/18	Incorporated more images & installation for Quartus
1.2	A. Joshipura	3/22/18	Added more shortcuts and tips on how to use modelsim. Updated Lab 1 with better naming convention.
1.3	A. Joshipura	3/23/18	Added a photo for final testbench. Updated the figures table with better sentence case
1.4	A. Joshipura	3/28/18	Edited names of lab 2 and added more explanation on ISSP. Added more images and explained the bug better.
1.5	A. Joshipura	3/29/18	Added more details on System Console Lab.
1.6	A. Joshipura	3/30/18	Changed ISSP explanation figure
1.7	A. Joshipura	4/2/18	Performed a few edits, wrong file names. Changed trigger condition from 2 to 1.
1.8	A. Joshipura	4/3/18	Added ISSP Source probe instantiation
1.9	A. Joshipura	4/11/18	Reduced trigger condition in Signal tap to 1.
2.0	A. Joshipura	4/19/18	Explanation about reset and clock for lab 3
2.1	A. Joshipura	5/7/18	Added DE0CV Board compatibility
2.2	L. Landis	7/16/18	Clarified signal selection in Signal Tap

Table of Contents

List of Figures	4
Objective	6
Downloading the Project Files	7
Simulation using the University Waveform Viewer	8
Simulation using ModelSim	11
Simulation for Debugging	11
ModelSim Overview.....	11
Lab Exercise 1.....	11
Some Tips & Shortcuts for ModelSim	25
Questions	32
Conclusion.....	32
In-System Sources and Probes	33
Introduction to In-System Sources and Probes	33
Lab Exercise 2.....	33
Conclusion.....	42
Signal Tap Logic Analyzer	43
Introduction to Signal Tap Logic Analyzer.....	43
Lab Exercise 3.....	43
Conclusion.....	50
System Console	52
Introduction to System Console	52
Lab Exercise 4.....	52
Conclusion.....	62

List of Figures

Figure 1 Quartus prime settings	6
Figure 2 IP catalog for altera pll	14
Figure 3 Save ip variation box	14
Figure 4 Megawizard for ALTPLL.....	15
Figure 5 Megawizard for output frequency.....	16
Figure 6 Quartus dialogue box.....	16
Figure 7 Altera PLL Cyclone V.....	17
Figure 8 Dialog Box	17
Figure 9 Altera PLL IP Parameter Editor.....	18
Figure 10 Quartus IP dialog box	18
Figure 11 Adding IP files to the design.....	19
Figure 12 Project navigator window	20
Figure 13 Testbench solution.....	21
Figure 14 Eda tool settings.....	22
Figure 15 Simulation settings from EDA tool settings	23
Figure 16 Choosing the testbench file	23
Figure 17 Running simulation from quartus	23
Figure 18 ModelSim GUI	24
Figure 19 ModelSim GUI	25
Figure 20 Waveform window ModelSim	26
Figure 21 Waveform with two cursors	27
Figure 22 Zoomed in version of waveform	28
Figure 23 Adding objects to waveform.....	28
Figure 24 Adding selected signal to the waveform.....	29
Figure 25 Waveform from ModelSim with shortcuts	30
Figure 26 Grid, timeline, cursor control.....	30
Figure 27 Selecting timescale.....	31
Figure 28 Design for ISSP and SignalTap analyzer.....	34
Figure 29 About ISSP	34
Figure 30 IP catalog.....	35
Figure 31 IP variation box	36
Figure 32 Megawizard for issp.....	36
Figure 33 Project navigator to add and remove files.....	37
Figure 34 Source probes instantiation	37
Figure 35 Instance manager.....	38
Figure 36 Programming the board.....	39
Figure 37 ISSP editor and waveforms	40
Figure 38 Changing the bus display format	41
Figure 39 Debugging the design using issp	41
Figure 40 Programmer	43

Figure 41 Signal tap instance	44
Figure 42 Node finder in signal tap.....	45
Figure 43 Rearranging signals	46
Figure 44 Node tab with all nodes	46
Figure 45 Signal configuration tab in signal tap.....	47
Figure 46 Setting triggers for all nodes	48
Figure 47 Programmer window in Signal Tap	48
Figure 48 Signal tap output log	49
Figure 49 Reformed design for 0-99 counter.....	50
Figure 50 Open dialogue box	53
Figure 51 IP catalog in platform designer	53
Figure 52 Platform designer GUI.....	54
Figure 53 Generation window for platform designer	55
Figure 54 Launching system console from platform designer.....	56
Figure 55 Verifying system connections	56
Figure 56 Programming device from system console.....	57
Figure 57 Response after programming from messages window	57
Figure 58 Command to set JTAG path	57
Figure 59 Command for verifying signal integrity.....	57
Figure 60 Command for sensing clock	58
Figure 61 Command sampling clock	58
Figure 62 Command for reset	58
Figure 63 Getting master path.....	58
Figure 64 Setting master path.....	59
Figure 65 Reading & writing memory	59
Figure 66 Snippets of TCL script.....	60
Figure 67 Launching GUI	61
Figure 68 Gui for Led's and Switch's	62

Objective

This course introduces the student to the various debugging tools available in Quartus to aid in debugging of FPGA designs. Upon completion of the lab, the student will be able to use ModelSim software for simulation debugging, In-system Sources and Probes tool, Signal Tap Analyzer as well as System Console.

Prerequisites:

- You are expected to be familiar with basics of logic and digital design
- Be familiar with Verilog/VHDL constructs.
- DE10 Lite board, needed for Lab 2, 3, and 4.
- User must have Quartus Prime Lite 17.1/or above installed.

To download Quartus Prime Lite, follow the instructions:

1. Visit the site: <https://fpgasoftware.intel.com/?edition=lite>. Select the version and your PC's operating system
2. For smallest installation, and quick download enter only the entries below.

Quartus Prime Lite Edition

Release date: November, 2017
Latest Release: v17.1

Intel® Quartus® Prime
Design Software

Select edition: Lite
Select release: 17.1

Operating System: Windows Linux

Download Method: Akamai DLM3 Download Manager Direct Download

✓ The Quartus Prime software version 17.1 supports the following device families: Arria II, Cyclone 10 LP, Cyclone IV, Cyclone V, MAX II, MAX V, and MAX 10 FPGA. [More](#)

Combined Files Individual Files Additional Software Updates

Download and install instructions: [More](#)
[Read Intel FPGA Software v17.1 Installation FAQ](#)
[Quick Start Guide](#)

Select All

☒ Quartus Prime Lite Edition (Free)
☒ Quartus Prime (includes Nios II EDS)
Size: 1.7 GB MD5: C6E662E428D1F5E93B6CC5B5076C3ED4
☒ ModelSim-Intel FPGA Edition (includes Starter Edition)
Size: 1.1 GB MD5: 9F0FCC22EB8ADD19200D3C00B20EDCC5

Devices

You must install device support for at least one device family to use the Quartus Prime software

☐ Arria II device support
Size: 499.6 MB MD5: EA15FB95662AB632F2CD95A93D995A92

☐ Cyclone IV device support
Size: 466.6 MB MD5: 09D346E4AE7AC403DF4F36563E6B7BFB

☐ Cyclone 10 LP device support
Size: 266.1 MB MD5: C9D4AC6A692BE4C3EAC15473325218BB

☐ Cyclone V device support
Size: 1.1 GB MD5: 747202966905F7917FB3B8F95228E026

☐ MAX II, MAX V device support
Size: 11.4 MB MD5: 77B086D125489CD74D05FD9ED1AA4883

☒ MAX 10 FPGA device support
Size: 325.2 MB MD5: 9B55655054A7EA1409160F27592F2358

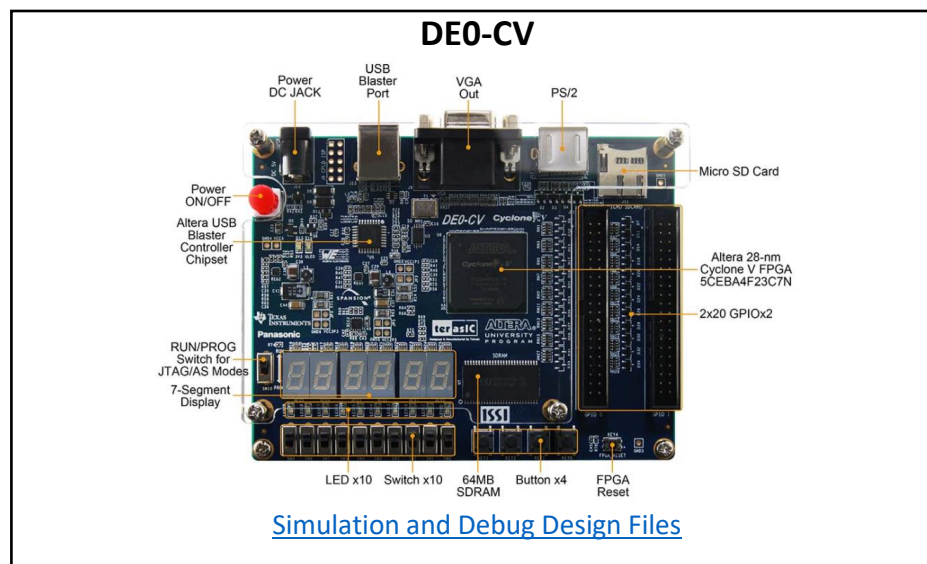
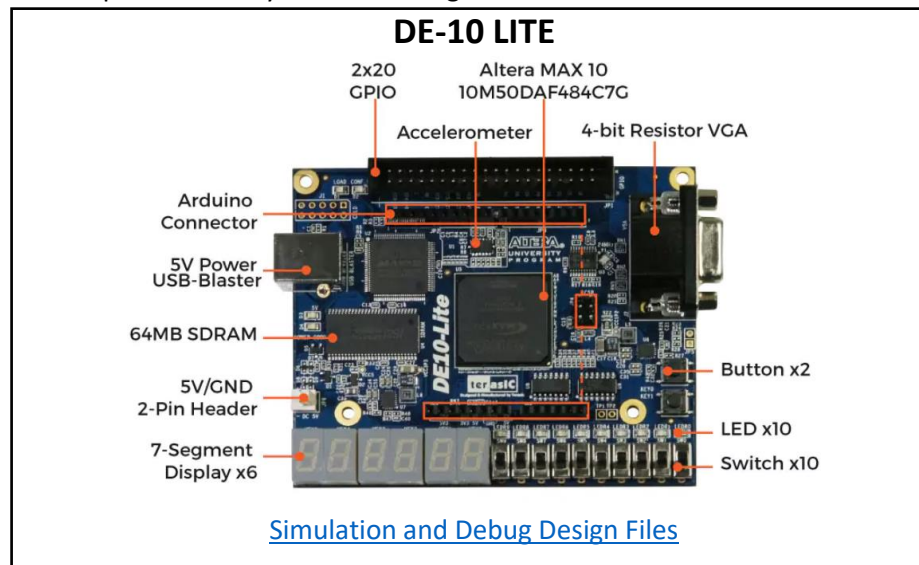
Updates Available

FIGURE 1 QUARTUS PRIME SETTINGS

3. Follow the instructions to download the design tools and you will have the Quartus Prime Lite version up and running.

Downloading the Project Files

1. Check the development board you have been given.



2. Download the project files from the links mentioned according to the board you have. Make sure the file is downloaded in C: // Drive or your Documents Folder.
3. You need to have Quartus 17.1 Lite installed on your computer. (See Prerequisites above for downloading instructions). Then, unzip the files.

Simulation using the University Waveform Viewer

Launch Quartus

Create a new project: File → New Project

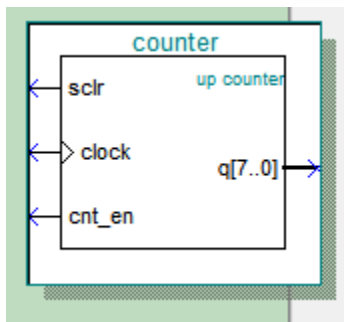
Use the feature to copy settings from the past previous project, this will save you some time

You won't have any files in the project navigator

In IP catalog, search for counter. Select LPM counter.

Double click on the counter, you can name it counter8.

Features should include 8 bits wide with count enable and synchronous clear



Click finish and add to your project

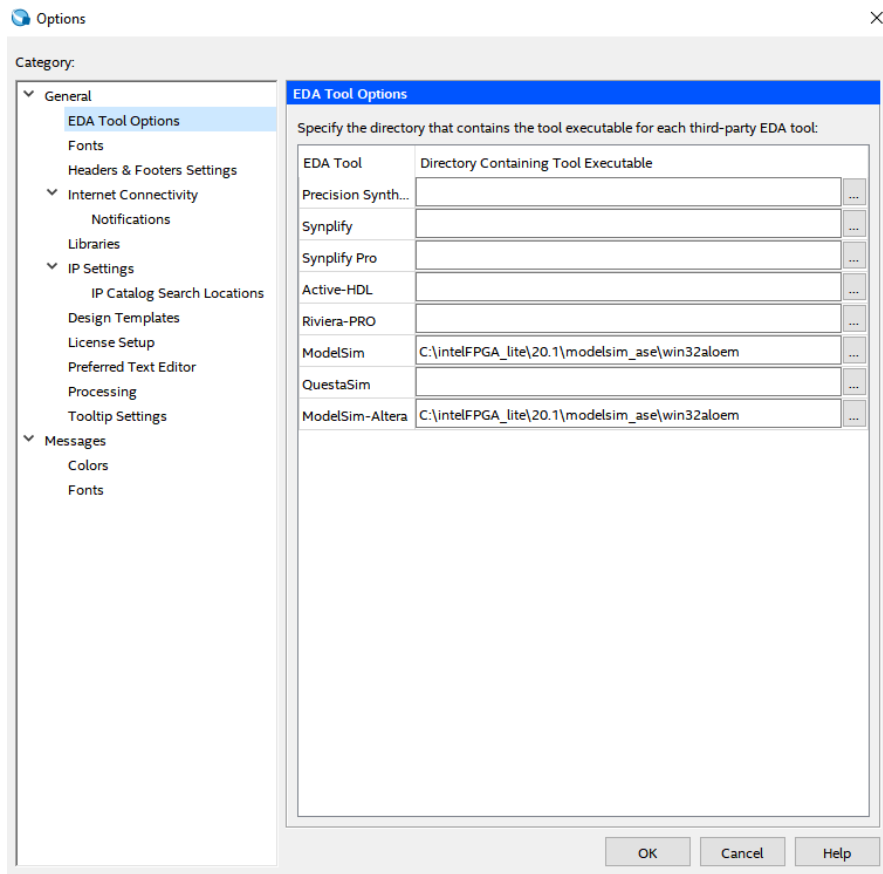
When you are done you should see counter8.qip as part of your project

Right click on counter8.qip and set as your top level entity

Compile your design (Click on the play button)

Tools → Options

Select EDA Tool options. Fill in the path to Modelsim as follows with the appropriate path.

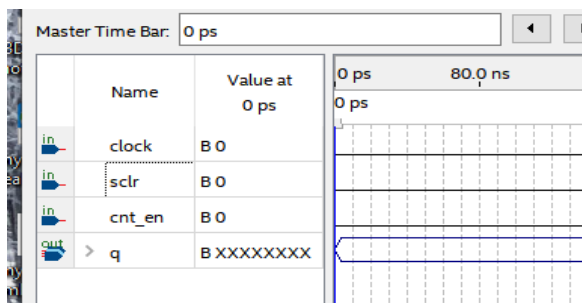


Next, we will create waveforms and simulate. File → New → University Program VWF

Right click on left side. Insert Node or Bus

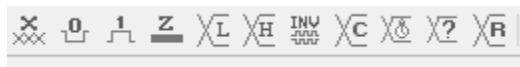
Click list, and you will get a list of I/O signals

Add clock, sclr, cnt_en and q (the bus version). Hit OK. Arrange the order like so:

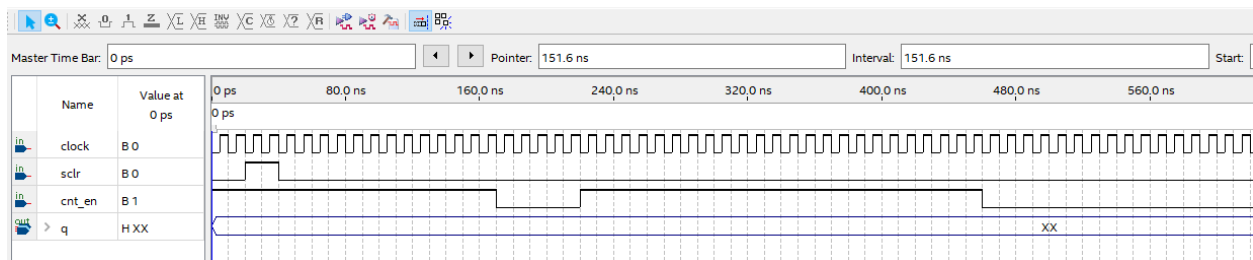


The default radix is binary. Right click on the q signal and change to hexademical.

Next you need to edit input waveforms using the waveform editing buttons.



The clock will utilize the pocket watch icon, set sclr and cnt_en are set high or low by highlighting the time you want to change the waveform and make it look something like the following:



If you use 20.1 Quartus, Simulation → Simulation Settings

Edit the vsim line from:

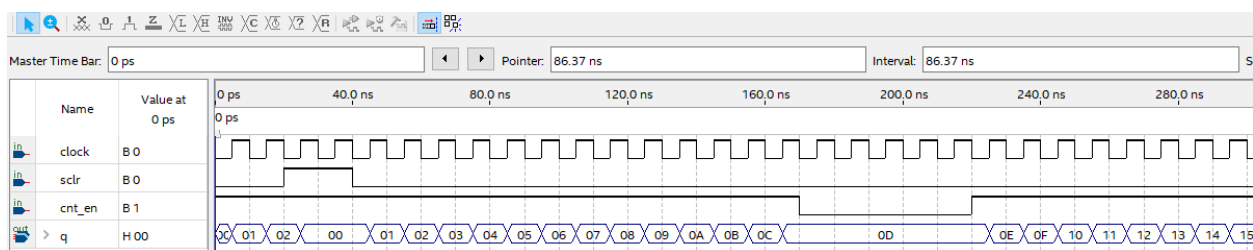
```
vsim -novopt -c -t 1ps -L fiftyfivenm_ver -L altera_ver -L altera_mf_ver -L 220model_ver -L sgate_ver -L altera_lnsim_ver work.coun
```

to

```
vsim -c -t 1ps -L fiftyfivenm_ver -L altera_ver -L altera_mf_ver -L 220model_ver -L sgate_ver -L altera_lnsim_ver work.coun
```

Launch the simulator by running Simulation → Run Functional Simulation.

You will get a second waveform window with the results of Q changing state



Congratulations, you now are able to run the University Waveform Viewer.

Simulation using ModelSim

Simulation for Debugging

Design simulations are able to include wide range of analyses that virtually test behavior of a product under various operating and environmental conditions. As opposed to trial-and-error, a smart simulation process allows targeted implementation of design choices in various stages of the development cycle. Proper functional and timing simulation is important to ensure design functionality and success.

This drastically reduces the need for recurrent, time-consuming testing on expensive physical prototypes, and subsequently shortens the total development time.

ModelSim Overview

ModelSim is a multi-language HDL simulation environment offered by Mentor Graphics. It can be used independently or with Intel Quartus Prime, which can help create libraries and link the designs to ModelSim. Using the **ModelSim-Altera** software simplifies the debugging with help of simulations.

Lab Exercise 1

Two Methods to Restore Project Files:

Method 1:

1. After Un-zipping the files. Navigate to the **Lab_1 Folder** and double click on **Simulation_Example.qar**. Also, note that the lab is done in windows environment.
2. A window will pop up asking to restore project. Click **OK**. Make sure the destination folder is set as the location you want to restore your project in. (C:// or Documents)

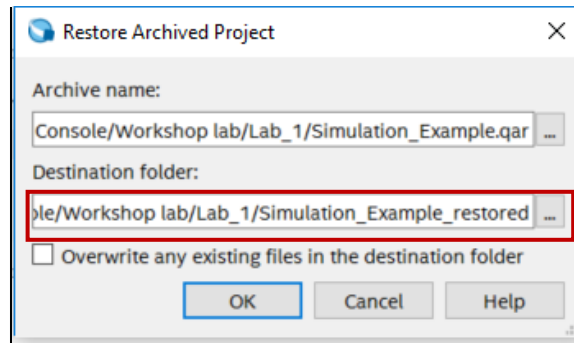


FIGURE 2 RESTORED ARCHIVE PROJECT WINDOW

Note: You will restore other labs in similar fashion.

Method 2:

1. After Un-zipping the files. Open Quartus 17.1 Lite Software.
2. In the main menu, File → Open Project.
3. Navigate to the folder where you unzipped the files (In C:// or Documents). Go to **Lab_1** Folder and select **Simulation_Example.qar**
4. A window will pop up asking to restore project. Click OK. Make sure the destination folder is set as the location you want to restore your project in. (C:// or Documents) Refer Figure .

Objective:

We will use this lab to familiarize the student with following:

1. *The IP Catalog of Quartus and how to use it. It will also familiarize the student on how to design an IP according to the needs of the design.*
2. *How to open ModelSim for simulation purposes from Quartus for an FPGA design.*
3. *Some helpful shortcuts for ModelSim.*

This lab exercise consists of a simple digital logic design of an up/down counter and phase locked loop (PLL) as shown in the block diagram below. The PLL is used to increase the frequency of the input clock from 50 MHz to 100 Mhz which is given to the up/down counter.

Generally, phased locked loops are used for multiplying clock frequencies and there is a startup time associated with it, until the output signal is locked. Here, in simulation you may not observe a significant startup time before the clock frequency is stable, but in silicon there is generally some startup time associated with it. The specification for the MAX 10 FPGA is to lock within 1 ms. That is equivalent to 50,00 clock cycles for this design which would take a long time to simulate. Take note of how fast the PLL locks in simulation.

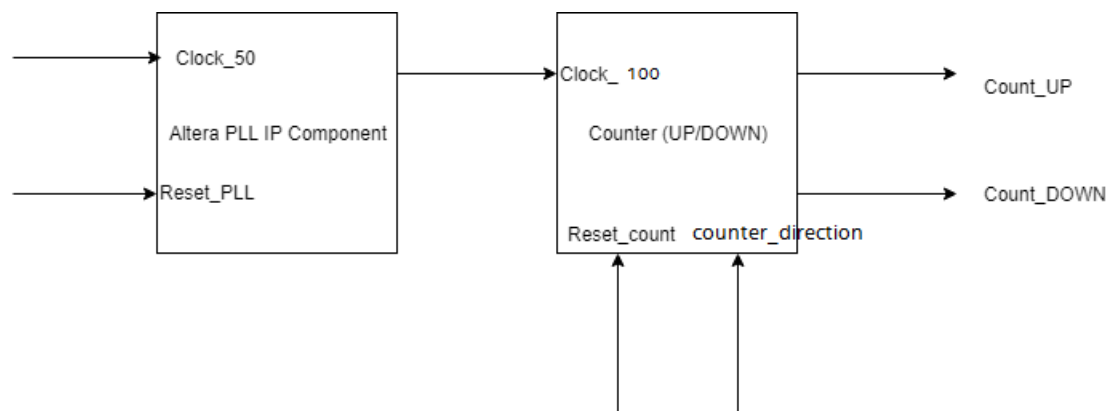


FIGURE 3 DESIGN FOR SIMULATION

Section 1: Add IP Component to the design

If you are using a **DE-10 Lite development kit**, go to [DE-10 Lite Development Kit](#) section to add the MAX 10 PLL IP.

If you are using a **DE0-CV Development kit**, got to [DE0-CV Development Kit](#) section to add the Cyclone V PLL IP.

DE-10 Lite Development Kit

1. In the IP Catalog, search for PLL. Double click to add ALTPLL in the PLL Library. The path where you can find the PLL is Library → Basic Functions → PLL → ALTPLL. If you do not have the IP Catalog already open, go to View menu → IP Catalog to view the window.

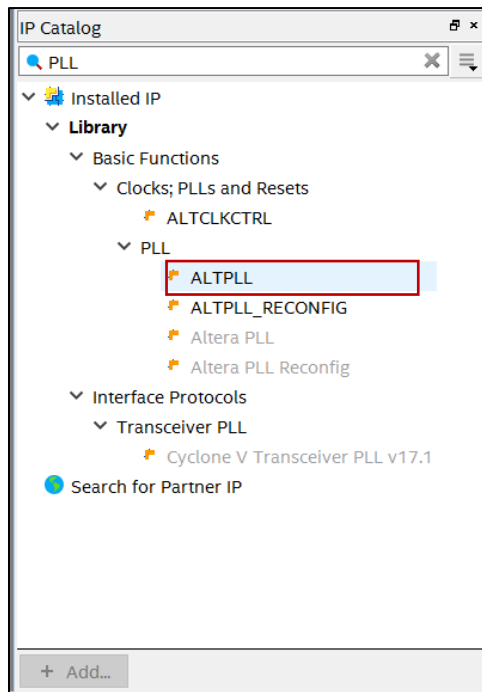


FIGURE 2 IP CATALOG FOR ALTERA PLL

2. Name the file **PLL** in the Save IP Variation box that came up and click **OK**. Please make sure to name it **PLL**.

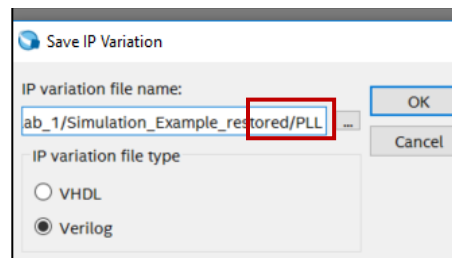


FIGURE 3 SAVE IP VARIATION BOX

3. In the MegaWizard dialogue box, choose **inclk0** to **50 MHz** and click **Next**.

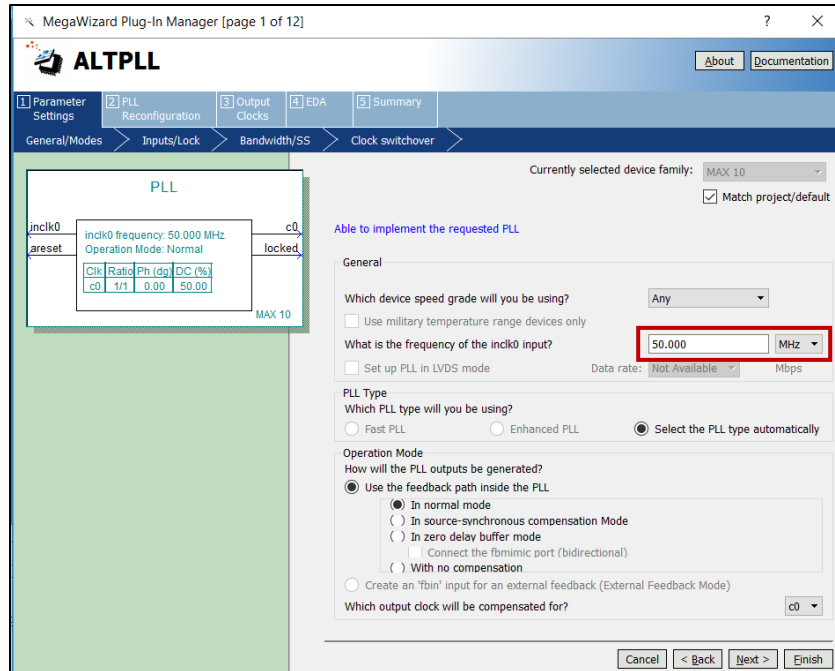


FIGURE 4 MEGAWIZARD FOR ALTPLL

4. Continue to click next and leave the default values until you reach the screen called output clocks, leave all settings to default. In the output clocks tab, for c0 select the **Enter output frequency** as **100 MHz** and leave remaining settings to default. Click **Finish**.

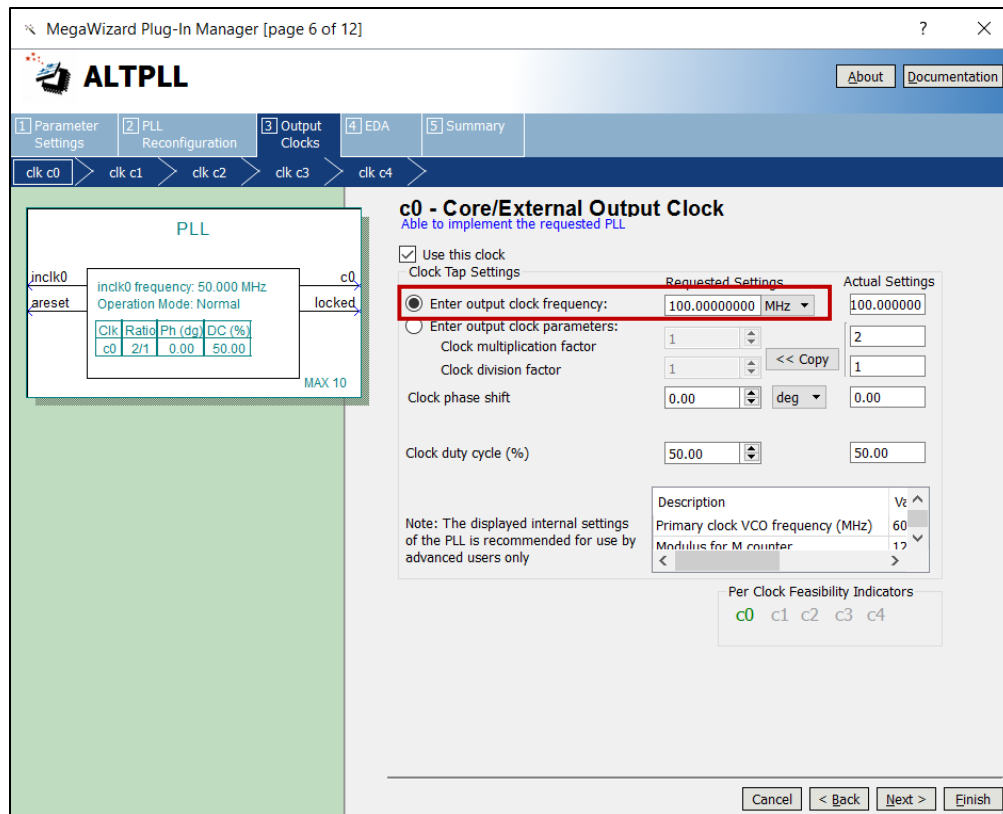


FIGURE 5 MEGAWIZARD FOR OUTPUT FREQUENCY

- After you click finish. You will get a dialog box asking you if you want to add the IP files generated to the design. Click **Yes**.

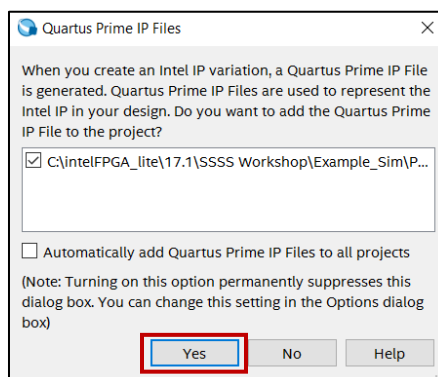


FIGURE 6 QUARTUS DIALOGUE BOX

- Continue to the section [here](#), once you are done creating the IP.

DE0-CV Development Kit

1. In the IP Catalog, search for PLL. Double click to add Altera PLL in the PLL Library. The path where you can find the PLL is Library → Basic Functions → PLL → Altera PLL. If you do not have the IP Catalog already open, go to View menu → IP Catalog to view the window.

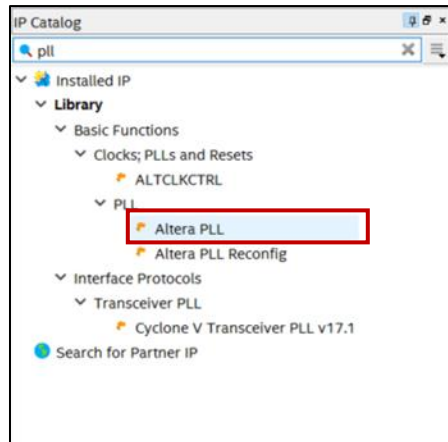


FIGURE 7 ALTERA PLL CYCLONE V

2. Name the file **PLL** in the Save IP Variation box that came up and click **OK**. Please make sure to name it **PLL**.

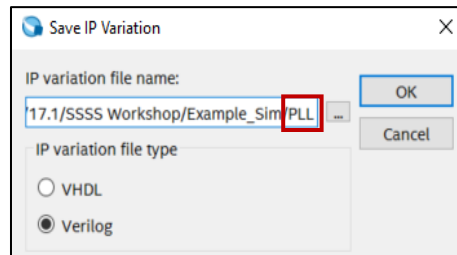


FIGURE 8 DIALOG BOX

3. In the MegaWizard, choose **reference frequency** at **50 MHz** and **output frequency** as **100 MHz** and leave remaining settings to default. Click **Finish**.

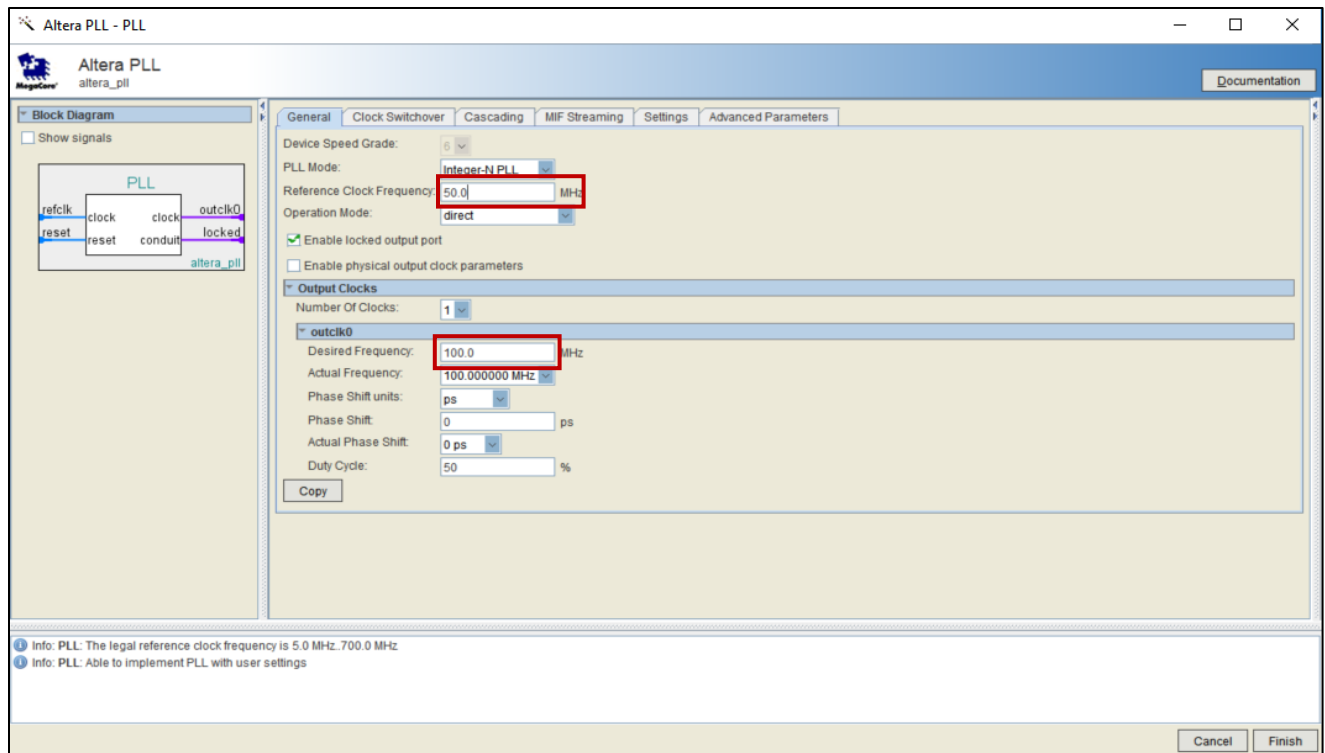


FIGURE 9 ALTERA PLL IP PARAMETER EDITOR

4. After you click finish. You will get a dialog box asking you if you want to add the IP files generated to the design. Click **Yes**.

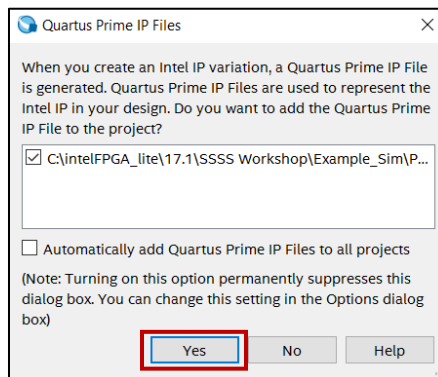


FIGURE 10 QUARTUS IP DIALOG BOX

5. Continue to the section [here](#), once you are done creating the IP.

To check if the IP is added in the design. Check the **Project navigator** window, select files in the drop down menu, if the **PLL.qip** files are added in the project. **If yes, skip this step.** To add PLL.qip files in the project.

- Right-click the files folder and select Add/Remove files.
- Select ... button, go to PLL.qip and add it to the project files.
- Once files are added, click **OK**.

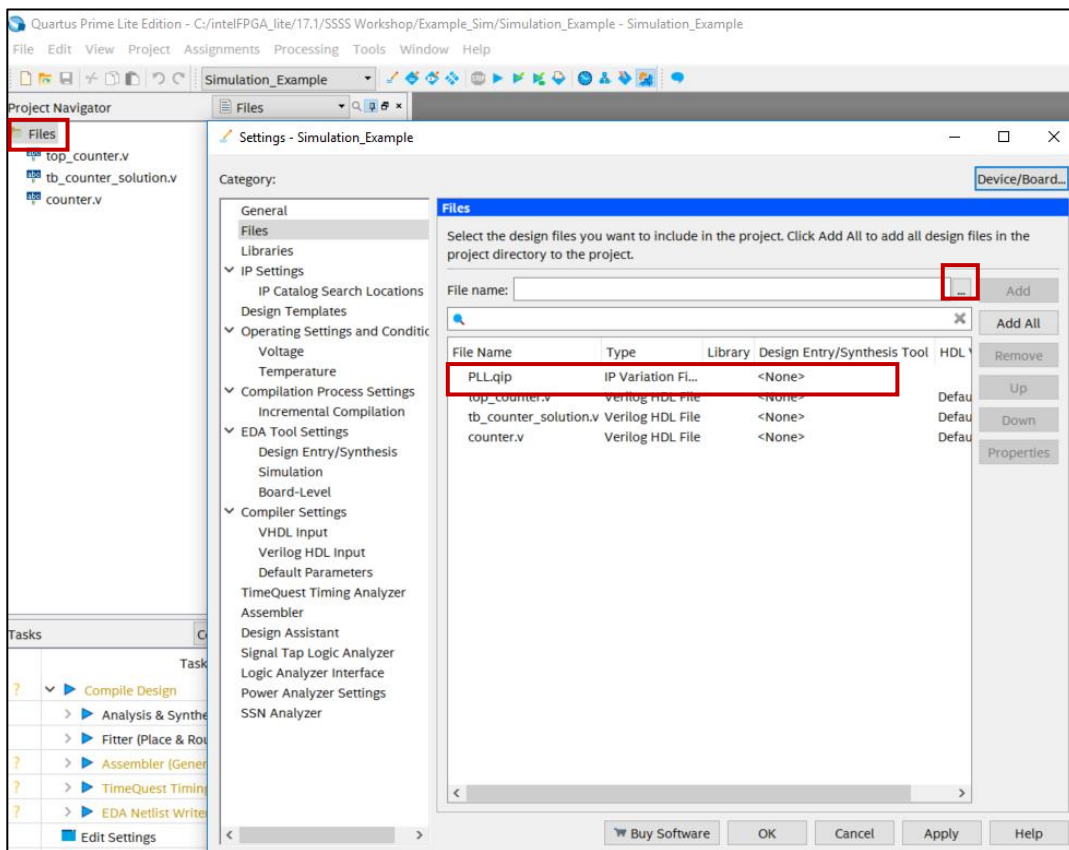


FIGURE 11 ADDING IP FILES TO THE DESIGN

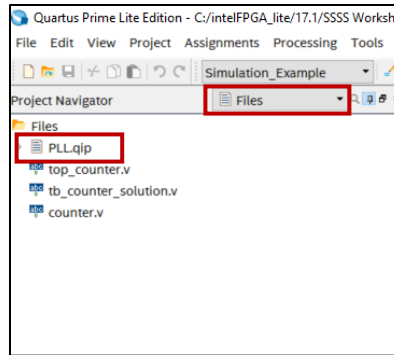


FIGURE 12 PROJECT NAVIGATOR WINDOW

Note: If you choose IP Components in the project navigator window, you should be able to see PLL IP component. If you double click on the component, you can edit the parameters using MegaWizard.

Section 2: Observing the Test-bench for the design

1. In the Main Menu, Go to File → Open.. → Lab 1 and Open **tb_counter.v** file.
2. This is a simple testbench.
3. Open file **top_counter.v** by double clicking on the file and take a look at how the connections are made between the module and the IP.
4. First, notice all the inputs from the **top_counter.v** module (clock, reset_pll, reset_count, counter_direction) are declared as type **reg** in the **tb_counter.v** file.
5. Next, all the outputs are declared as type **wire** i.e count_up and count down.
6. In the instantiation of the **top_counter.v** module, the connections of the modules are made.
7. In the **initial block**, first all the inputs are declared as 0. The initial block in Verilog is used generally in test benches. It is a block that only runs once unlike the always block.
8. To create a clock of **50 Mhz** for the PLL. There are three common ways listed below to create the clock. The most common one is (a).
 - a. Using always block


```
always #10 clock =~ clock;
```
 - b. Using forever block


```
forever begin  
#10 clock =~ clock;  
end
```
 - c. Using parameter as constant clock period


```
localparam CLK_PERIOD = 20;  
  
always
```

```

begin : CLK_GEN

    tb_clock = 1'b0;

    #(CLK_PERIOD/2);

    tb_clock = 1'b1;

    #(CLK_PERIOD/2);

end

```

9. At the end, you test bench should look like the figure shown below.

```

timescale 1ns/10ps
module tb_counter();
|
reg clock, reset_p11, reset_count, counter_direction;
wire [3:0] count_up , count_down;

top_counter DUT ( .refclk(clock) ,
                  .reset_p11(reset_p11) ,
                  .reset_count(reset_count),
                  .counter_direction(counter_direction),
                  .count_up(count_up),
                  .count_down(count_down) );

initial
begin
    clock = 0;
    reset_p11 = 1;
    reset_count = 1;
    counter_direction = 0;

    #10 reset_p11 = 0;
    #30 reset_count = 0;
    #10 counter_direction = 1;
    #30 counter_direction = 0;
    #10 counter_direction = 1;
    #30 counter_direction = 0;
    #10 counter_direction = 1;
    #30 counter_direction = 0;

    #1000 $stop;
end

always #10 clock = ~clock;
endmodule

```

FIGURE 13 TESTBENCH SOLUTION

Section 3: Setting up ModelSim from Quartus

1. Once you have completed section 1 and 2. You need to launch ModelSim for simulations. To do so you need to change settings so that Modelsim can open through Quartus Prime Lite. Go to **Tools → Options → EDA Tool Options**. In ModelSim-Altera, enter the executable pathway. For finding the executable pathway, locate where the Quartus was installed by you (Usually in C:// drive in intelFPGA_Lite folder). Select **OK** when finished.

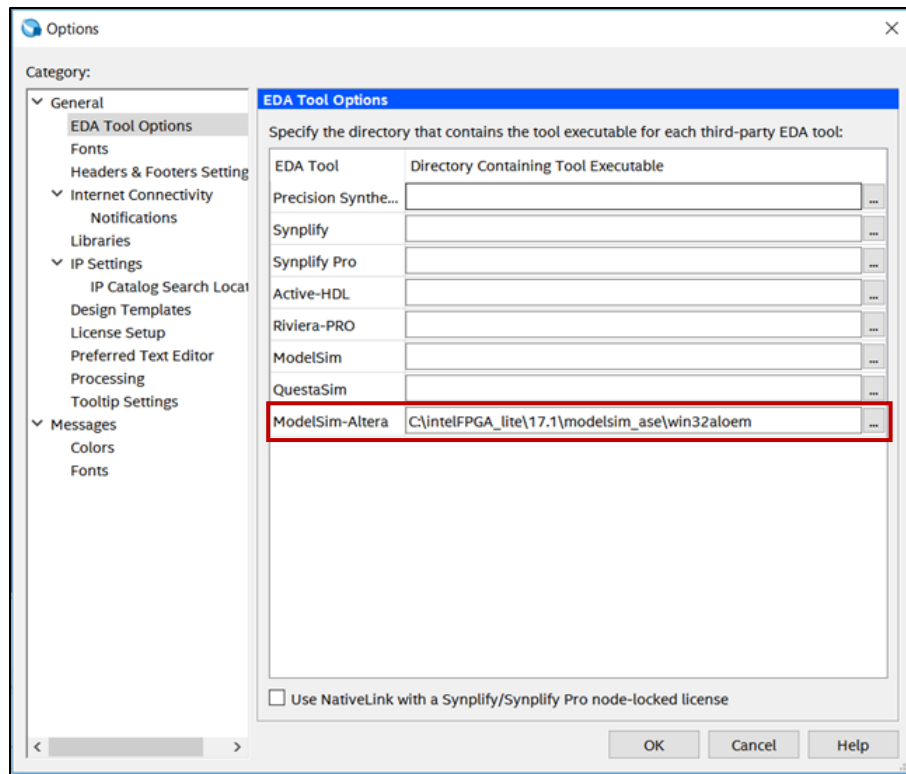


FIGURE 14 EDA TOOL SETTINGS.

2. Next step is to make sure that the test bench is compiled. Go to **Assignments → Settings → EDA Tool Settings → Simulation**. Select ModelSim-Altera in the Tool Name.
3. Under **NativeLink Settings → Select Compile Test Benches and click on Test Benches..**
4. In the Test Benches Dialog Box, Click on New.. Select the Testbench file to be added and Click **Add**. Name the testbench **tb_counter** as that is the top module of the testbench. Click **OK**.

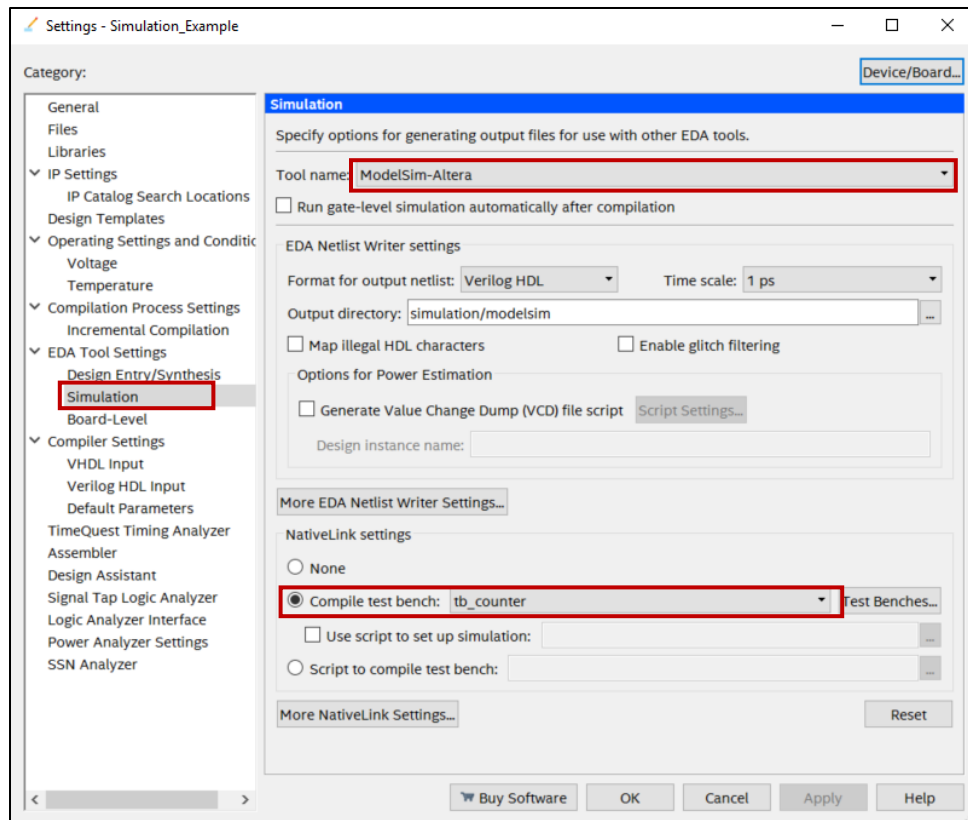


FIGURE 15 SIMULATION SETTINGS FROM EDA TOOL SETTINGS

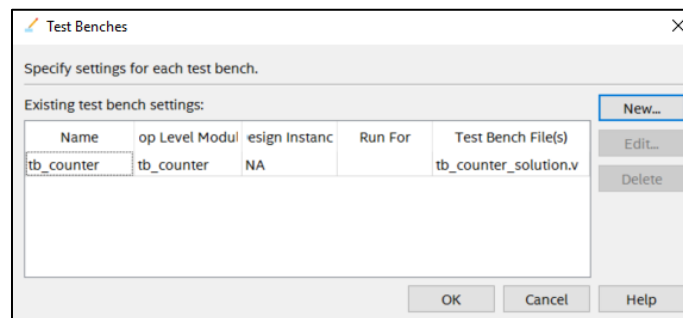



FIGURE 16 CHOOSING THE TESTBENCH FILE

5. Once all the settings are in place. Go to  to compile the design in the toolbar.
6. Next, from Quartus. **Tools → Run Simulation → RTL Simulation**

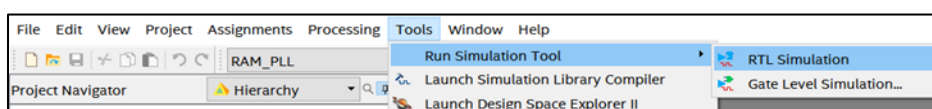


FIGURE 17 RUNNING SIMULATION FROM QUARTUS

7. Once this step is completed, ModelSim should have opened up on the screen. The GUI for ModelSim is shown below.

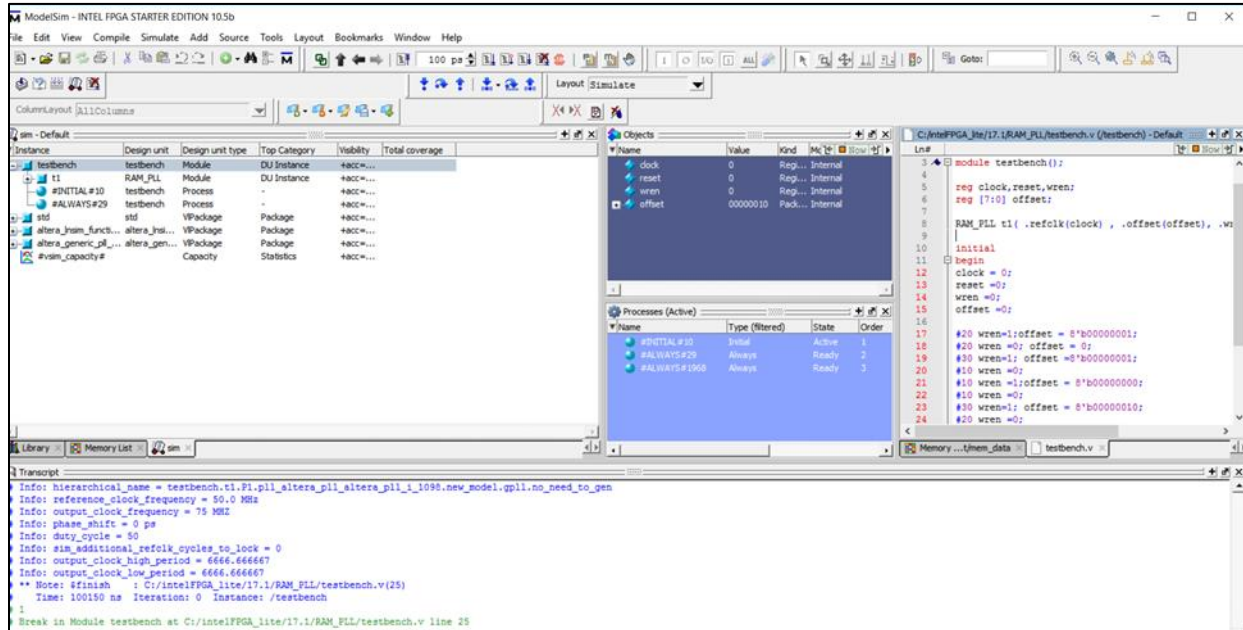


FIGURE 18 MODELIM GUI

Note: If you get error loading the design related to PLL.vo file. You might have to go to your folder directory where the project is stored and copy the PLL.vo file from the PLL_sim folder to the main project folder. And repeat step 3, 4, 5.

8. You can now observe the waveform and verify the functionality of the design.
9. If you have used ModelSim before and know how to move around the various tabs, you may not see the same layout as shown in Figure 18. You can go to Layout → Reset to get the same layout of the GUI.

Some Tips & Shortcuts for ModelSim

ModelSim GUI

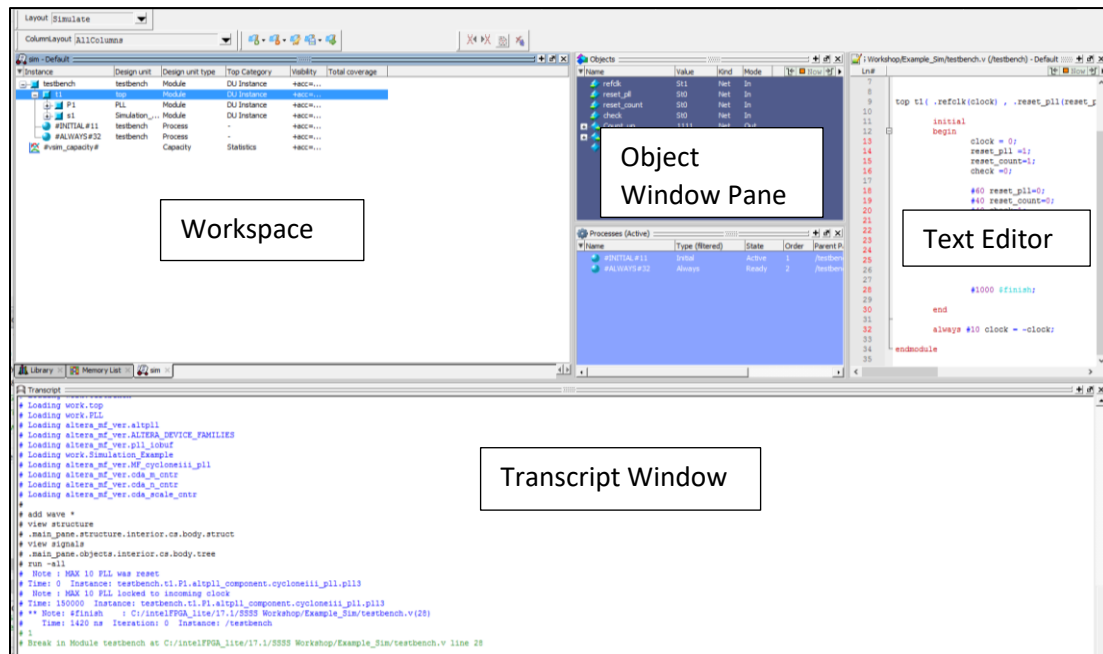


FIGURE 19 MODELSIM GUI

Workspace:

The workspace consists of various tabs, each tab having a different functionality. Library tab represents various modules and files present in the design. Memory List tab will represent the values in the memory units used in the design. Once the design is simulated, a tab called sim is created which displays the hierarchical structure of the design. You can navigate within the hierarchy by clicking on any line with a '+' (expand) or '-' (contract) icon.

Objects Window Pane:

The Objects pane shows the names and current values of data objects in the current region (selected in the Workspace). Data objects include signals, nets, registers, constants and variables not declared in a process, generics, and parameters.

Transcript Window:

The Transcript pane provides a command-line interface and serves as an activity log including status and error messages.

Text Editor:

This pane allows you to edit and read design files in ModelSim.

When you waveform window shows up the first time it may look similar to the Figure 20 below. We will now look into some methods to view the waveform shortcuts.

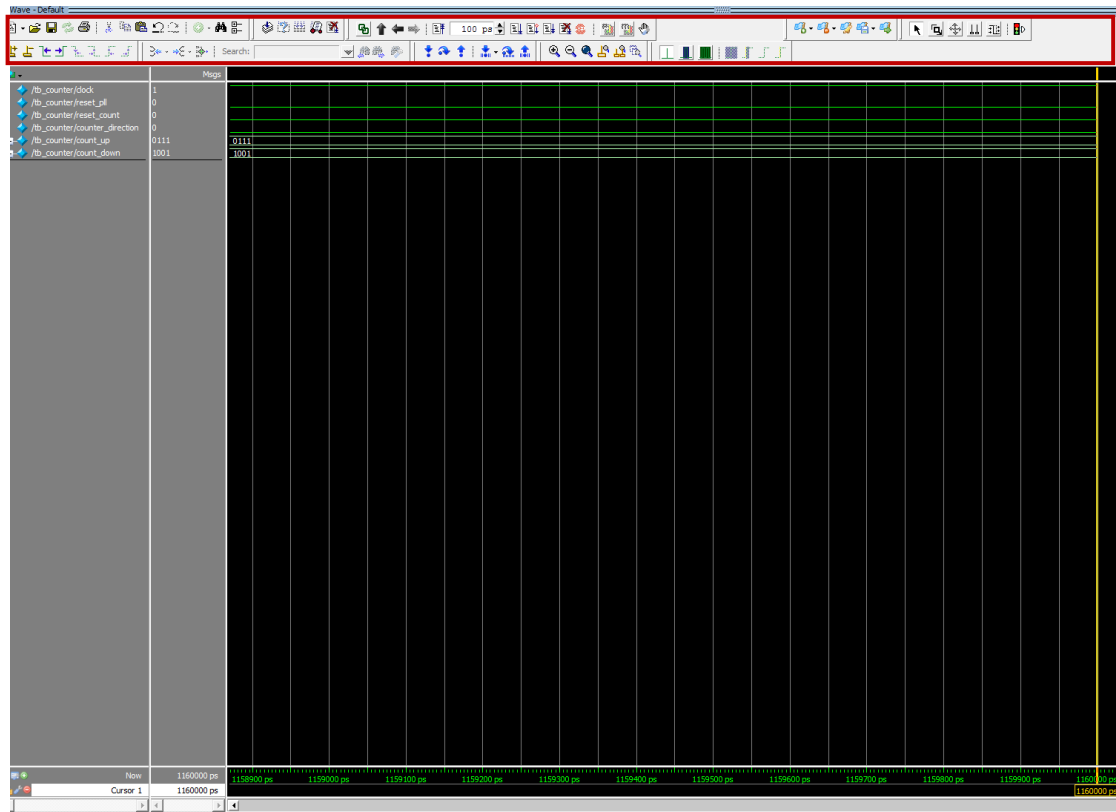




FIGURE 20 WAVEFORM WINDOW MODELIM

1. Waveform Short Cuts:


The Wave window allows you to view the results of your simulation as HDL waveforms and their values. The Wave window is divided into a number of panes. As you observe the waveform that has come up as shown in Figure 20 and verify the design functionality.

a. Zooming the waveform display:

1. In toolbar highlighted in the figure above, the icons  help in zooming in and out of the waveform.

2. If you want to completely zoom-out of the waveform. Click on .

3. For adjusting the waveform between cursors. You can click on .

First, to add another cursor, you use the icon  on the toolbar. That will add another cursor in the waveform. As shown in the figure below.

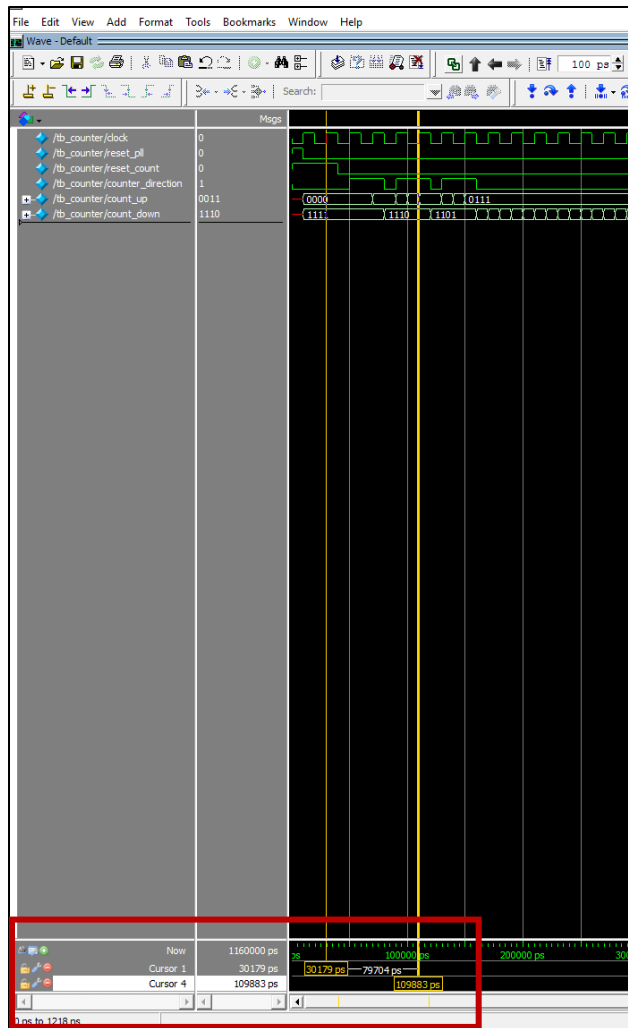



FIGURE 21 WAVEFORM WITH TWO CURSORS

4. Next, move the cursors between the regions you want to zoom in at.
5. Once you have both the cursors in the desired location. Click on  toolbar.
6. You would view a zoomed in version of the waveform as shown in the figure below.

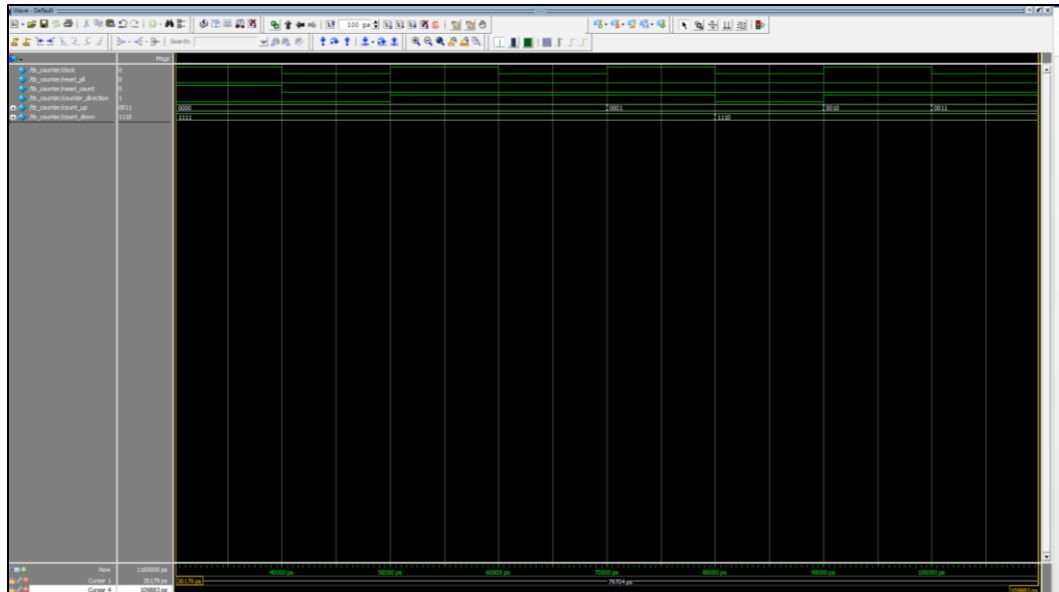


FIGURE 22 ZOOMED IN VERSION OF WAVEFORM

b. Adding objects to Waveforms:

ModelSim offers several methods for adding objects to the Wave window. If you want to add individual signals.

1. Select the instance reference name from the sim list you want to add the signals from.
In our case, it is counter_1.

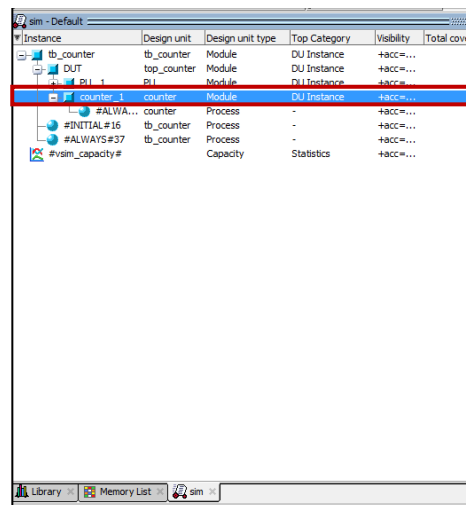


FIGURE 23 ADDING OBJECTS TO WAVEFORM

2. In the objects pane, select the signal you want to view in the waveform. Right-click the selected signal. Select Add to →Wave →Selected Signals as shown in the figure below.

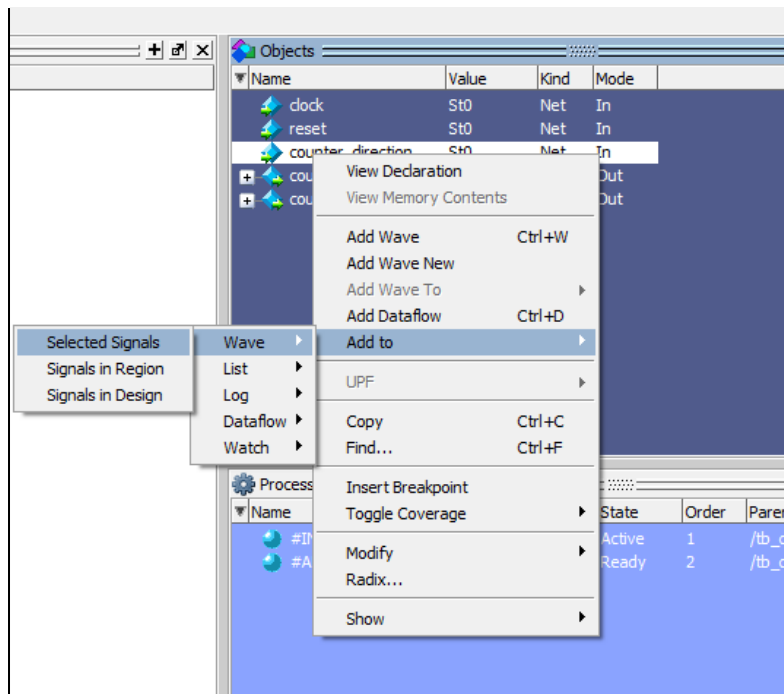


FIGURE 24 ADDING SELECTED SIGNAL TO THE WAVEFORM

c. Changing the radix of signals:

You can **select each signal** in the waveform window and **right click it to change its radix type** as shown in the figure below. The default is binary, and this allows you to change to decimal or hexadecimal.

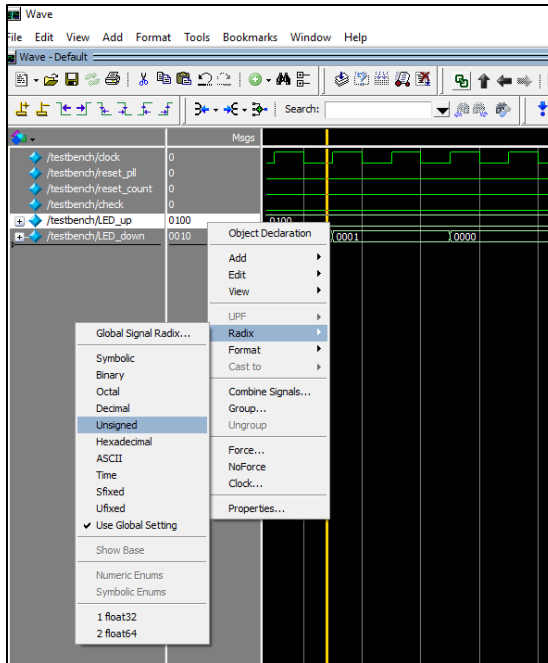
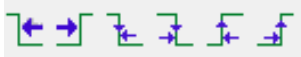


FIGURE 25 WAVEFORM FROM MODEL SIM WITH SHORTCUTS

- d. If you want to get to the next rising or falling edge of a particular signal. Select that signal in the waveform and click on  in the toolbar for respective actions.

e. Timescale in waveform:

1. If you want to change the default timescale from ps to ns. Navigate the cursor on the lower right near the timescale as shown in the figure below and right click. **Select Grid, Timeline & Cursor Control..**

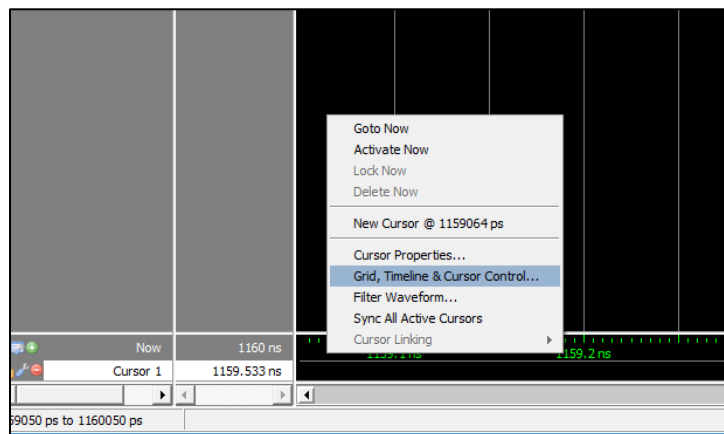


FIGURE 26 GRID, TIMELINE, CURSOR CONTROL

2. In the Grid, Timeline & Cursor Control dialogue box, select ns as the Time Units as shown in the figure below.

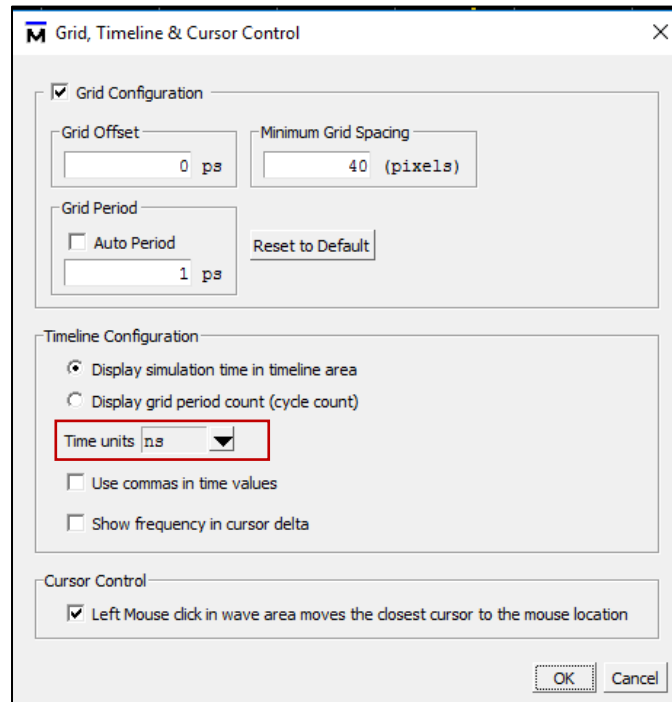


FIGURE 27 SELECTING TIMESCALE

f. Saving the Waveform:

1. If you want to save this particular set of signals in the waveform. Go to **File → Save Format** in the wave window.
2. Name it "wave_1.do" and click **Save**.
3. Next time you run commands from the Modelsim Transcript, you can type "**do wave_1.do**" before running a simulation.

Note: It is advised to name multiple waveforms with different names else it overwrites wave.do file if you have multiple waveform files.

2. Running design independently on Modelsim:

- a. When you Run RTL Simulation for the first time. In the Project folder, there is a folder called **simulation** is generated.
- b. In the **simulation** folder located in your project folder, under **Modelsim** folder look for a .do extension file. There is an "**Example_Simulation_run_msim_rtl_verilog.do**" file that is created.

- c. You can run this do file in Modelsim independently. Open ModelSim using the start menu. Once the software loads.
- d. In the transcript window, go to the project folder location. Once in the project folder, type using do **Example_Simulation_run_msim_rtl_verilog.do** the design should load without using the Quartus Software.

For more help and shortcuts on Modelsim, you can go to Help → Documentation → Tutorial.

Questions

Here are some questions you might like to answer yourself by analyzing the design.

What was the value of counter_up and counter_down at the start of the simulation?

What constructs of a testbench are not synthesizable?

Conclusion

With this exercise the student learnt to debug a design using simulations and also learnt about a few shortcuts available in ModelSim.

The next lab exercise introduces the In-system sources and probes debugging tool.

In-System Sources and Probes

Introduction to In-System Sources and Probes

Traditional debugging techniques often involve using an external pattern generator to exercise the logic and a logic analyzer to study the output waveforms during runtime. The In-System Sources and Probes Editor (ISSP) in the Quartus software extends the portfolio of verification tools. It allows you to monitor various signals in the design.

ISSP Editor consists of a probe function and interface to control the instances during run time. It is operated over JTAG. Each instance of ISSP can drive and toggle values up to 512 signals. It can create upto 128 instances of ISSP using IP Catalog.

The main difference between ISSP and Signal Tap (discussed in the next section) is that ISSP does not have a clock as reference to the output signals we will probe, it will only sample signals in the current time. Also, ISSP has a software source which can be used to monitor the outputs like a software switch.

Lab Exercise 2

Objective:

With the help of this lab the student will be able to understand how to use In-system sources and probes in debugging. The student will learn how to instantiate In-System Sources and Probes (ISSP) and observing the signal behavior of the design through the ISSP Editor.

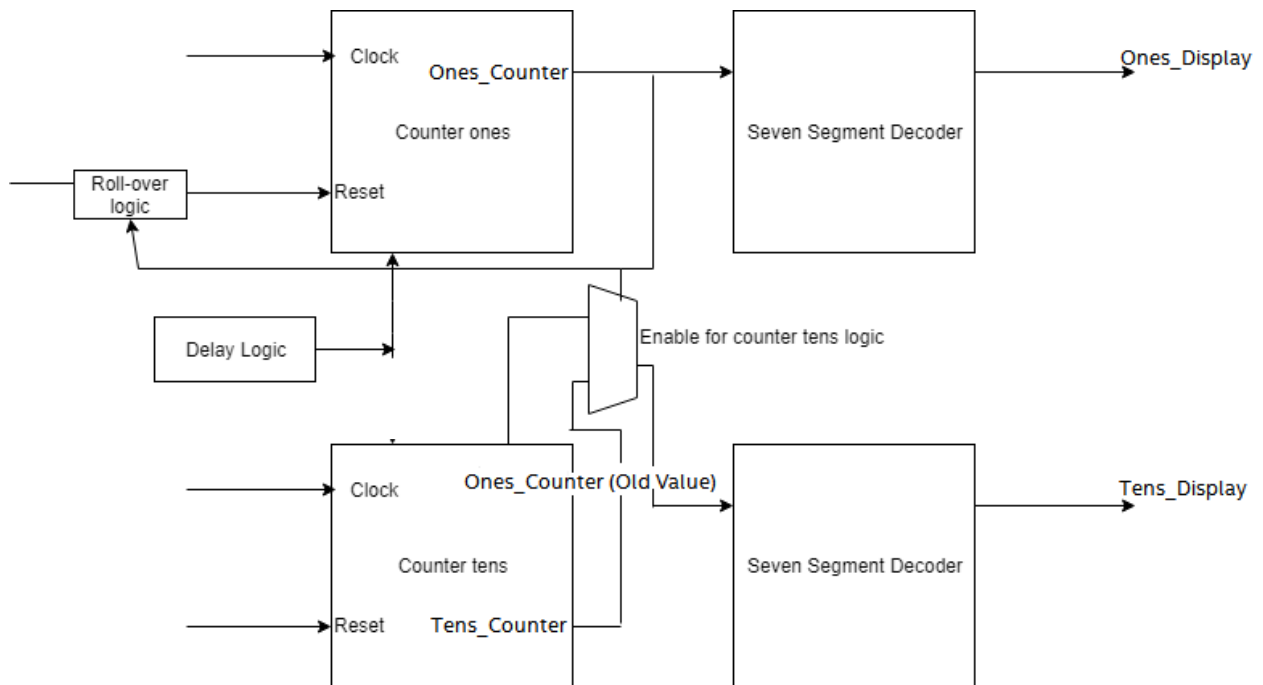


FIGURE 28 DESIGN FOR ISSP AND SIGNALTAP ANALYZER

The circuit for this exercise consists of two cascaded 4-bit counters, control logic to enable and clear the counters, and decode logic to drive two 7-segment displays. The 7-segment displays count from 0 to 99. A simple block-level diagram of the design is shown in the Figure 28.

You can see in the figure below, the In-system sources and probes is an IP present in the FPGA, which can help in debugging the design by giving the values to the sources in its interface itself, you can compare that to a software switch essentially. We can then monitor the probes in real-time. Only drawback of this is that we cannot compare the observed signals with a reference clock.

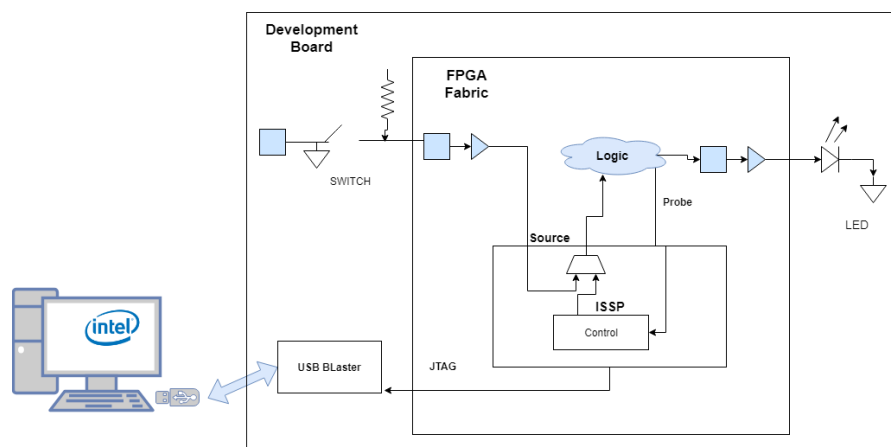


FIGURE 29 ABOUT ISSP

Section 1: Create an In-system Source and Probe Instance

1. In the downloaded **Workshop_DebuggingTools.zip** folder. Navigate to the **Lab_2** Folder. Double click on **Example_ISSP_SignalTap.qar** to restore the project.

An instance of In-System Sources and Probes (ISSP) was instantiated in the counter design. You will use ISSP Editor to program the device on the development board and to access the **altsource_probe** megafunction. You can create and configure a new ISSP Editor file from the menu, which would automatically find the altsource_probe instantiation.

2. Go to the **IP Catalog** of Quartus tool. Search for In-System Sources and Probes and double-click **Altera In-System Sources and Probes** IP.
 - a. You can also find it by going to **Basic Functions → Simulation; Debugging and Verification → Debug and Performance → Altera In-System Sources and Probes**
 - b. If the IP Catalog is not present, select it from the **Tools** menu

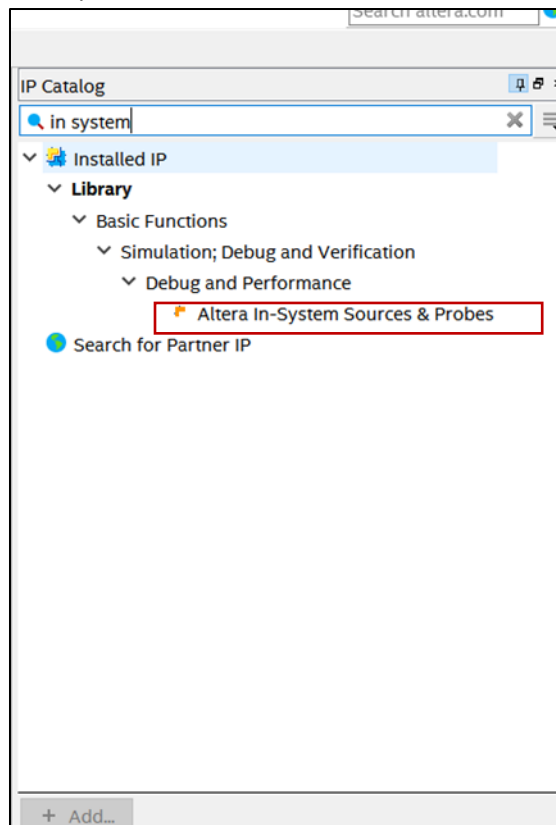


FIGURE 30 IP CATALOG

3. In the new IP Variation dialog box, set the entity name to **source_probes**. Be sure to match this **exact name** and case as it has to match the instantiation in the design files. Also, make sure device family is set to **Max 10** and device matches as well. Click **OK**.

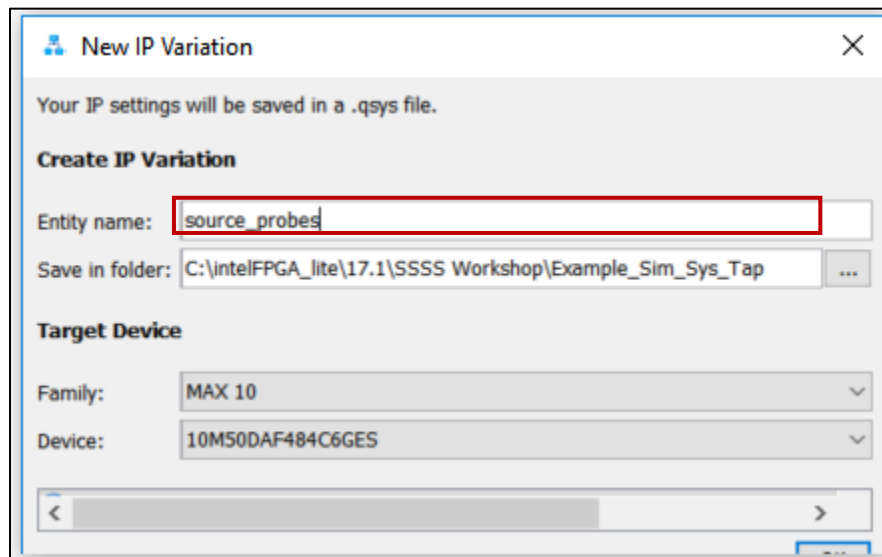


FIGURE 31 IP VARIATION BOX

4. In the IP Parameter Editor, make sure Automatic Instance Index Assignment is turned on. Set Instance ID to **SOPR**. Set Probe Port Width 14 and the Source Port Width to 1.

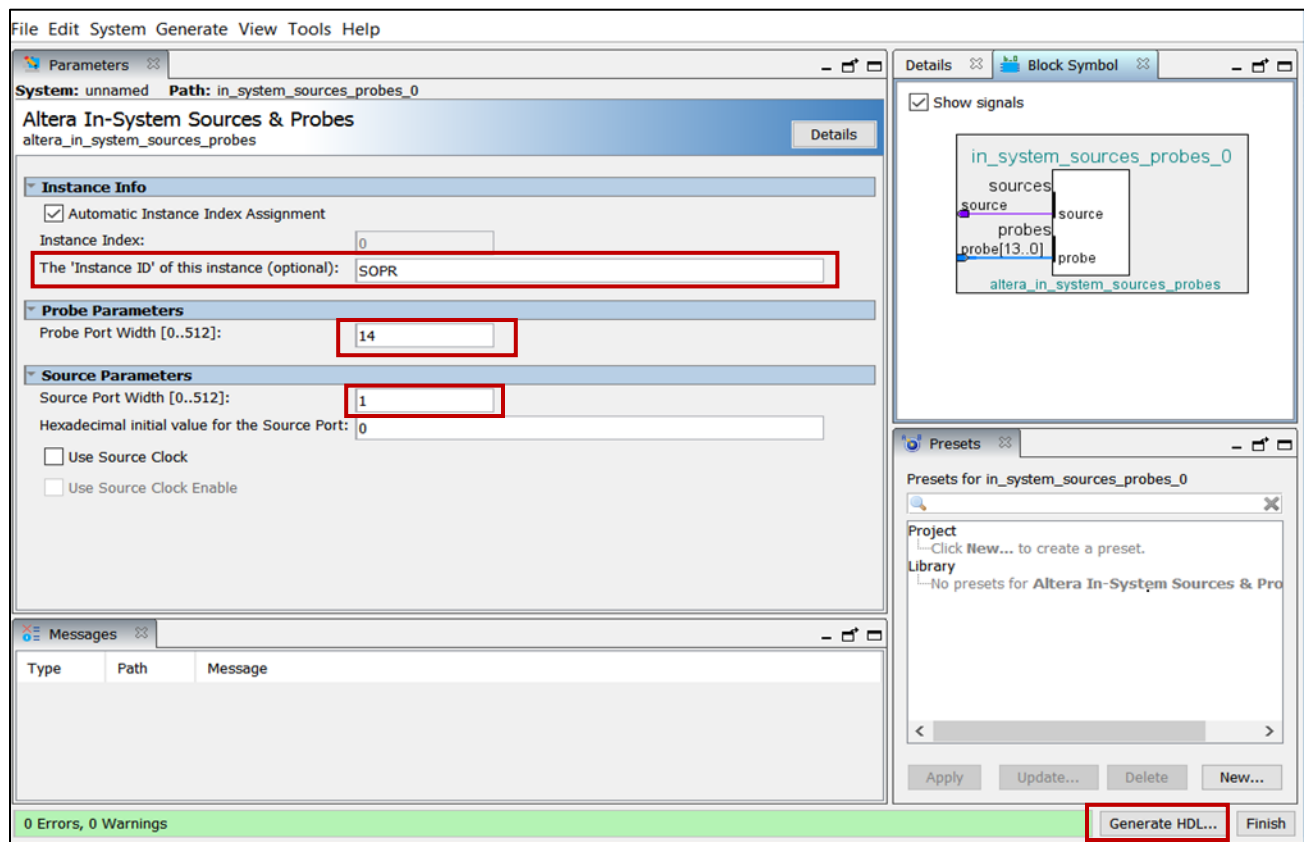


FIGURE 32 MEGAWIZARD FOR ISSP

- Now, click **Generate HDL...** In the Generation dialog box, in the **Synthesis** section, set **Create HDL** design files for synthesis to **Verilog**.
- Verify that the Output Directory Path is pointing to the **source_probes** folder in project directory. Leave all the other settings at their default and click **Generate**.
- Click **Close** when the generation of IP completes, and then close the IP Parameter Editor by clicking **Finish** at the bottom right-hand corner. Click **OK**, for the dialog box that informs about the IP File that was generated.
- Back in Quartus software, from Project Menu, select Add/Remove files in the project section. Click browse next to the file name.
- In the project directory, navigate to **source_probes** folder and then in **synthesis** folder. Click on the **source_probes.qip** file. Double click on it to add it to the project. Click **OK** to close the dialog box.

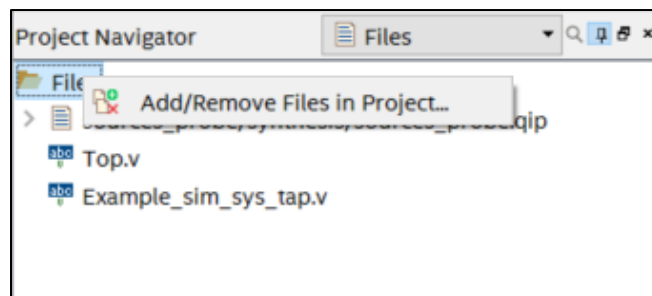


FIGURE 33 PROJECT NAVIGATOR TO ADD AND REMOVE FILES

- In the **Top.v** file, the connection for the **source_probe** IP is already made. Here, the **reset** signal will work as the **source** and the output of the counter module i.e output connecting to the seven segment display on the board (**HEX0, HEX1**) will be the signals we are probing (**probes**).


```

/* Sources and probe connections*/
source_probes source_probes_1(
    .source (reset_internal),
    .probe ({HEX0,HEX1})
);

endmodule

```

FIGURE 34 SOURCE PROBES INSTANTIATION

- Select **Start Compilation** from the Processing Menu or Click  in the toolbar to compile the project.

Section 2: Using In-system Sources and Probe Editor

- Plug in the board via the USB-Blaster cable to the PC if you have not done that already.

*If this is your first time using the board, install the USB Blaster to program your device.
To begin, open your device manager by typing “**Device Manager**” in the start menu. Navigate to the other device section and expand the section.*

Right Click the USB Blaster device and select **“Update Driver Software”**. Choose to browse your computer for driver software and navigate to the path `C://intelFPGA_lite/17.1/quartus/drivers`. Once you have the proper path selected, click on **“Next”** and the driver for the USB Blaster should be installed.

2. From the Tools menu, select **In-System Sources and Probes Editor**. You will have to click **OK**, for the dialog box that says No instances of In-system Sources and Probe were found in the project.
3. In **JTAG configuration section**, check your programming hardware. The **USB Blaster** should be selected. If there is no USB-Blaster, Click on **Setup** and select **USB-Blaster** in the drop-down menu as well. Click **OK**. (Refer to Figure 35)

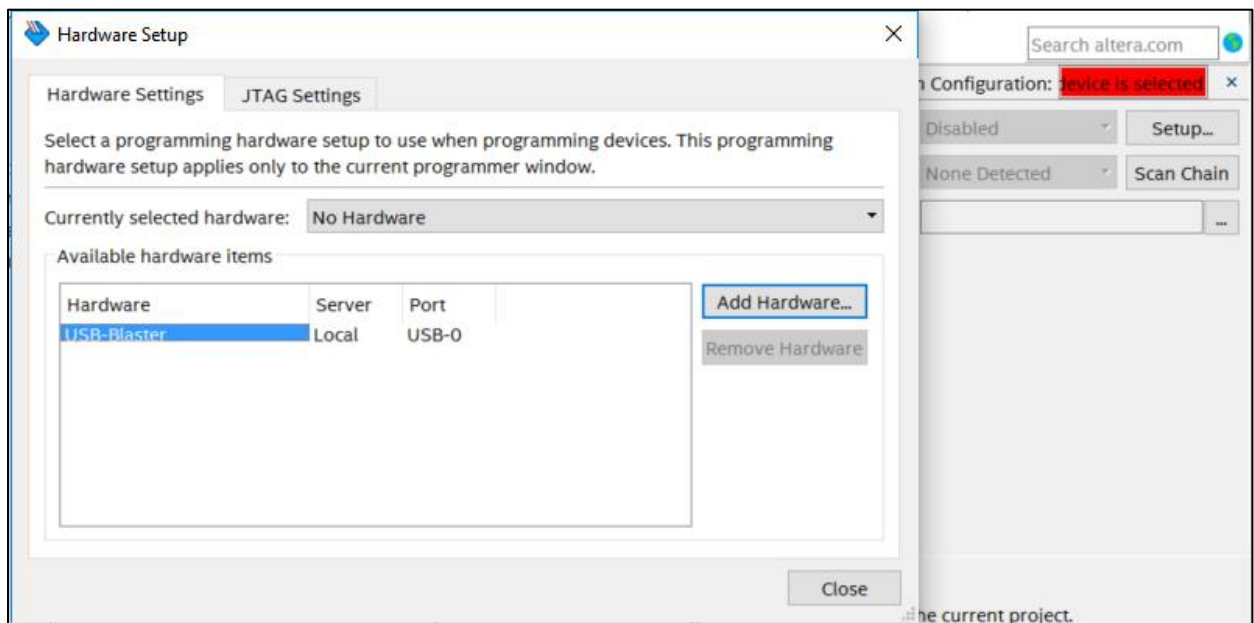


FIGURE 35 INSTANCE MANAGER

4. Click browse **button** as shown in the figure and select **Example_ISSP_SignalTap.sof** programming file in the project director.
5. Click the **Programming Button** (see Figure 36 below) to program the device. Remember that the progress bar may cycle a few times.

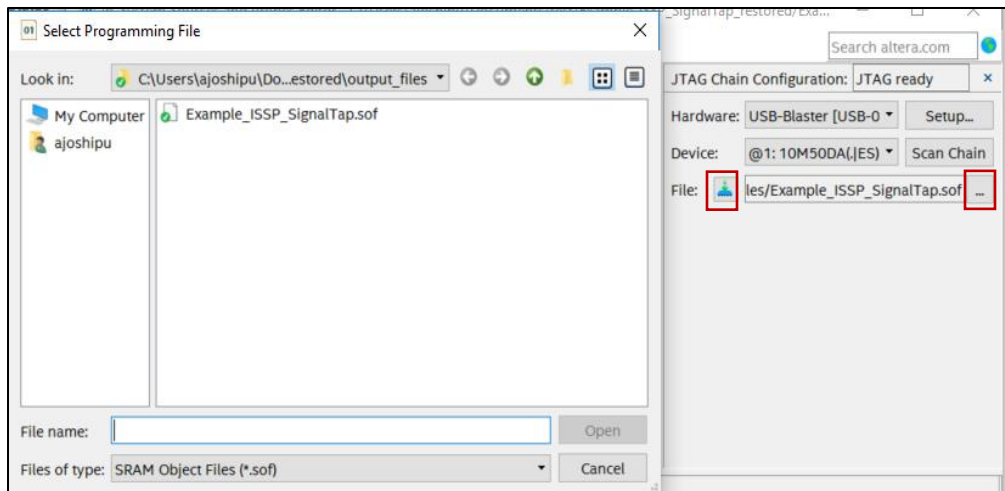



FIGURE 36 PROGRAMMING THE BOARD

6. Observe the HEX1 and HEX0 seven segment displays on the development board. They should be displaying 00.
7. Select **SOPR** instance in the **Instance Manager**. Refer to Figure 37 below.
8. Click to start continuously  reading the signals that feed the display.

- At the bottom of the window, click the 0 in the data column for the S0 source. This will take the design out of reset mode as shown in the figure below.

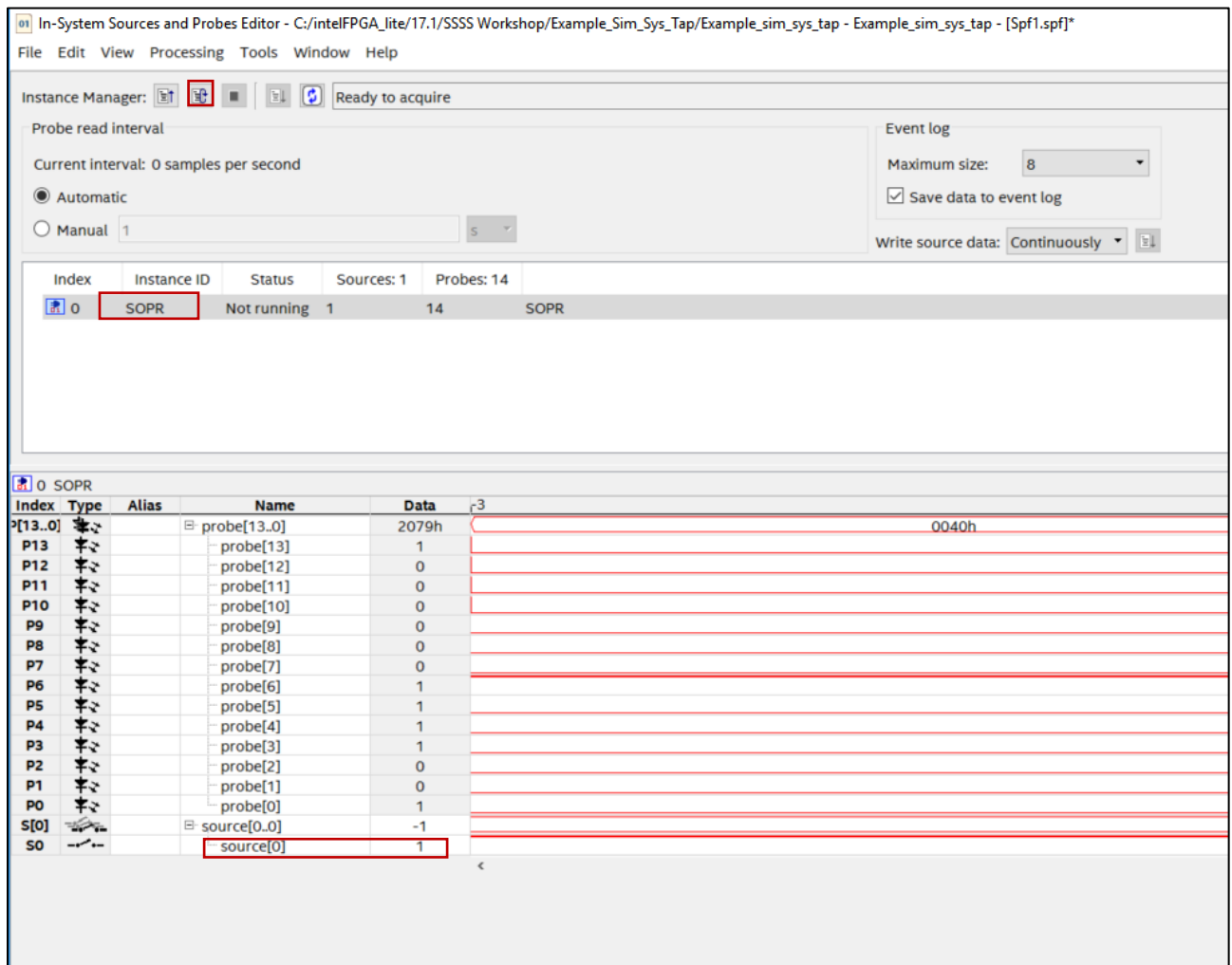


FIGURE 37 ISSP EDITOR AND WAVEFORMS

When the 0 is clicked, the level of source is changed to 1 and written to the reset of the system, taking the design out of reset as this acts as a source to get the system out of reset. You can change the 1 back to 0 to reset the counter just by clicking at it. On the seven segment display, the design should count sequentially from 0 to 99. The patterns used for displaying the numbers can be seen in the event log (waveform window) in the Sources and Probes editor.

Note: You should notice, on the seven segment display that numbers starting or ending in 9 do not get displayed on the HEX0 display. Also, if you observe the waveform window the pattern for 9 gets skipped. After 8 we get rolled over to 0 in the HEX0 display (Ones_Display signal). We will use Signal Tap logic analyzer to investigate the problem further.

Here, Probe [13] to Probe [7] are HEX0 and Probe [6] TO Probe [0] are HEX1 display.

See in Figure 39, Probe [13] to Probe [7] go from 7'b0 (decoder value to display an 8) directly to 7'b111101 (decoder value to display a 0).

Conclusion

With this exercise the student got introduced to In-system sources and probes and how to use it.

The next lab exercise introduces the signal tap logic analyzer and further helps in debugging the bug in the current lab exercise.

Signal Tap Logic Analyzer

Introduction to Signal Tap Logic Analyzer

Signal Tap logic Analyzer captures the logic state of FPGA internal signals using a defined clock signal. It helps monitor internal signals of the design in real time (with a clock as reference).

Lab Exercise 3

Objective:

With this lab exercise, the student will familiarize themselves with Signal Tap Analyzer and how to use it for debugging. The student will be able to create a signal tap instance, add nodes, create triggers and analyze the signals on the data log to debug a design.

Here, we continue to debug the design from Lab Exercise 2 to solve the issue in the design where it skips the number 9 for the first display. Also note, in the design the SW0 switch can be used to get the design out of reset.

Programming the board

If you are continuing from previous lab, we need to make sure we have reprogrammed the board. To reprogram the board follow the steps below.

1. Go to Tools → Programmer.
2. If the Example_ISSP_SignalTap.sof file is already present, click on start.

You have now reprogrammed the board.

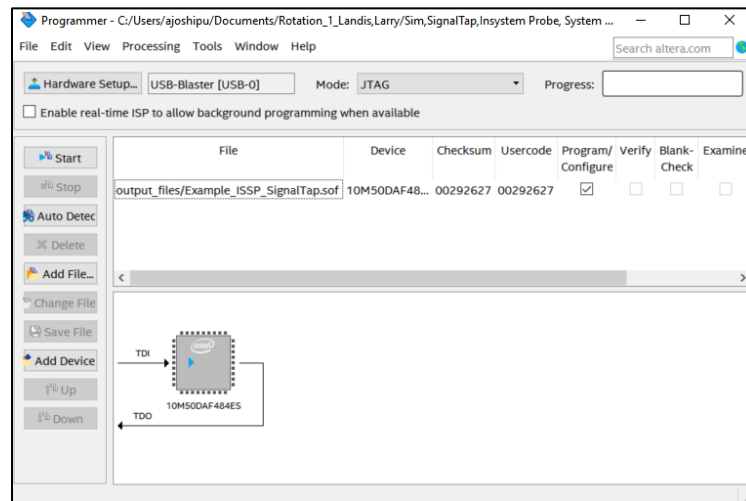


FIGURE 40 PROGRAMMER

Section 1: Create Signal Tap Logic Analyzer instances and compile with design

1. Choose **Signal Tap Logic Analyzer** from the Quartus **Tools** menu to create a new “.stp” file.
2. Set up a Signal Tap instance in design using the following steps.

The instance set up will be used to monitor the signals that drive both the seven segment displays.

In the **Instance Manager**, right click to **Create Instance**

3. Double-click to add nodes to the design. Refer figure below.

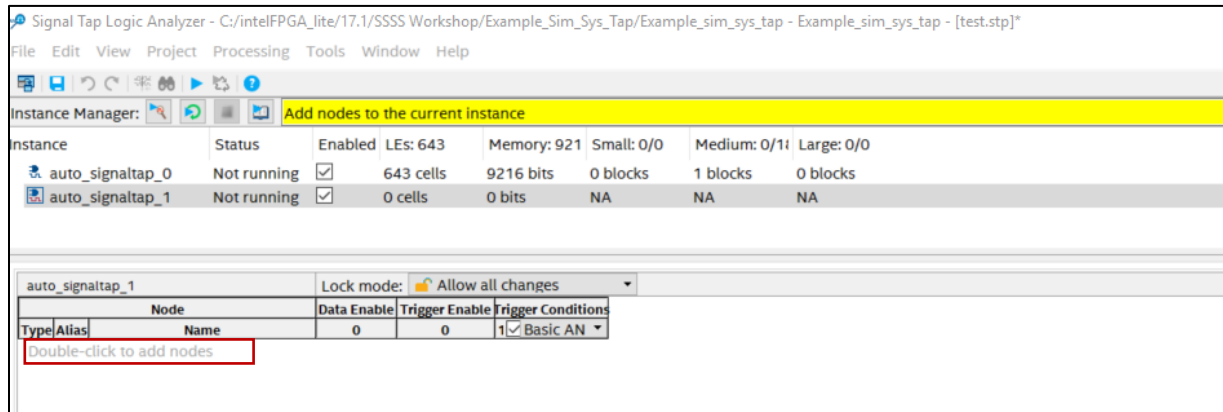


FIGURE 41 SIGNAL TAP INSTANCE

4. Find and select nodes for analysis using the following sub-steps:
 - a. In the Node Finder, click List. It will list all the components in the design.
 - b. Make sure the filter is set to **Signal Tap: Post- fitting** by selecting the arrow next to the **List** button.. Also, check box **Hierarchy view**. Click the arrow to view the hierarchy view if not seen.
 - c. Select **Counter: counter_0** and expand the view.
 - d. Add **Ones_Counter** and **Tens_Counter** signals into the Nodes Found column by clicking on the single right arrow so it faces down and expanding the selection. The selected signals should not contain the “~” character, those are other intermediate signals. If you cannot view all the signals, click on the downwards arrow as highlighted in the Figure 42. Also, add **SW [0]** from the SW instance.

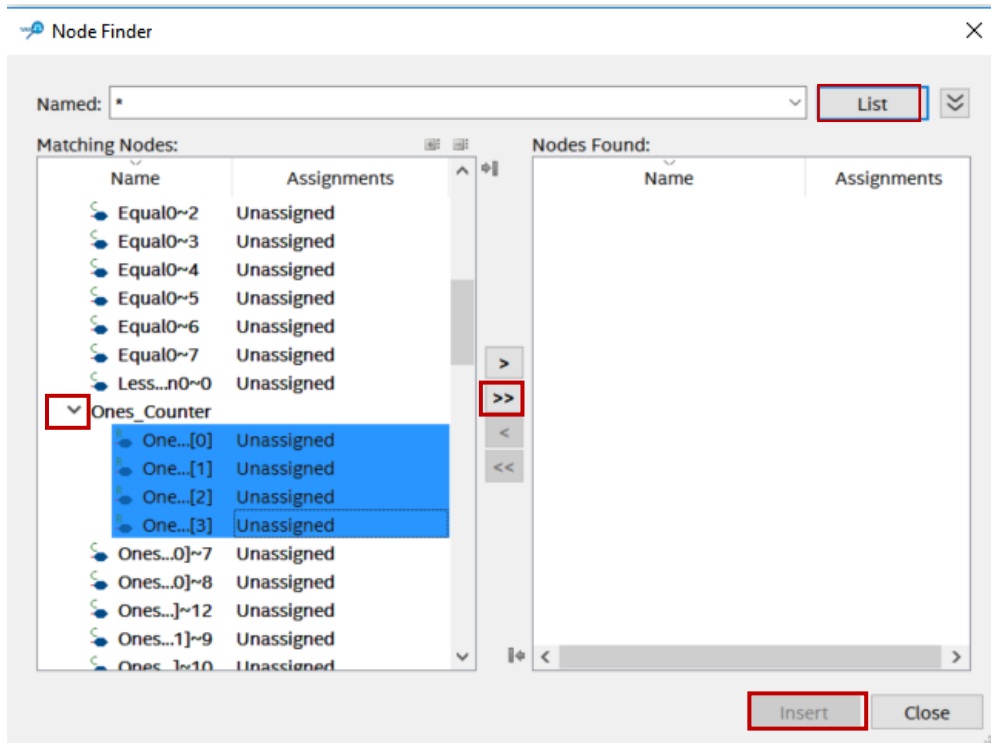


FIGURE 42 NODE FINDER IN SIGNAL TAP

- e. Click **Insert** to add nodes as shown in the figure above. If a window pops up asking to change netlist type to post-fit, click **OK**.
- f. Now, you can rearrange your signals by clicking them and dragging them to the position you want it at. Refer to Figure 43 below. One you have the order of the signals as shown in Figure 44 i.e Ones_Counter [3:0]/ Tens_Counter[3:0], you can move to next step of grouping the signals.

Note: If you cannot view the signals, click on setup tab in the tool. (Highlighted in the figure below)

5. Now, we will set up the trigger conditions for the nodes selected.
 - a. For the trigger conditions, go to **Signal Configuration Tab** present on the left side of the tabs pane. Now, in the Clock tab, search for **MAX10_CLK1_50** and add it the same way we added the nodes from before. Select the **sample depth** to be **1k**. The sample depth here helps specify the number of samples the tool can capture.
 - b. Set the **Trigger position** to be **Center trigger position** and the **Trigger condition** to be **1**. The trigger condition when set as 1, you can specify the trigger conditions at which you want tool which start to capture. Centre trigger position will give you the data 512 samples before the trigger specified and 512 sample after the trigger has been set off. (as sample depth is 1k(1024). You can observe this in your data log.

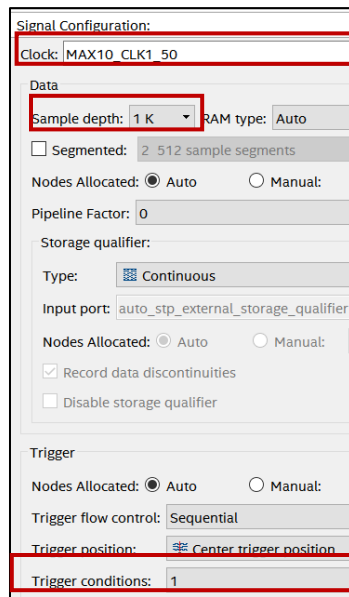


FIGURE 45 SIGNAL CONFIGURATION TAB IN SIGNAL TAP

If Signal Configuration Tab is not seen, go to View Menu and make sure signal configuration is checked


- c. In the nodes tab, change the trigger conditions of **Counter: counter_0 Ones_Counter** and **Tens_Counter** by right clicking each signal in the trigger condition tab. Set 9h (0x1001) as the trigger condition.

We choose, 9h as our trigger conditions to observe why we are skipping 9 in the HEX0 display while counting from 0 to 99.

Type	Alias	Node	Data Enable	Trigger Enable	Trigger Conditions
		Name	9	9	1 Basic AN
		Counter:counter_0 Ones_Counter[3..0]	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	9h
		Counter:counter_0 Ones_Counter[3]	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	1
		Counter:counter_0 Ones_Counter[2]	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	0
		Counter:counter_0 Ones_Counter[1]	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	0
		Counter:counter_0 Ones_Counter[0]	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	1
		Counter:counter_0 Tens_Counter[3..0]	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	9h
		Counter:counter_0 Tens_Counter[3]	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	1
		Counter:counter_0 Tens_Counter[2]	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	0
		Counter:counter_0 Tens_Counter[1]	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	0
		Counter:counter_0 Tens_Counter[0]	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	1
		SW[0]	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	1

AND / OR
AND
OR
NAND
NOR
XOR
XNOR
TRUE
FALSE
Compare...
Don't Care
Low
Falling Edge
Rising Edge
High
Either Edge
Insert Value...

FIGURE 46 SETTING TRIGGERS FOR ALL NODES

6. Save the Signal Tap instance.
7. Compile Design by clicking on  in the toolbar.

Section 2: Configure the device with Signal Tap instance

1. In the Signal tap window set up the JTAG Chain Configuration the same way as in the In-System Source and Probes.

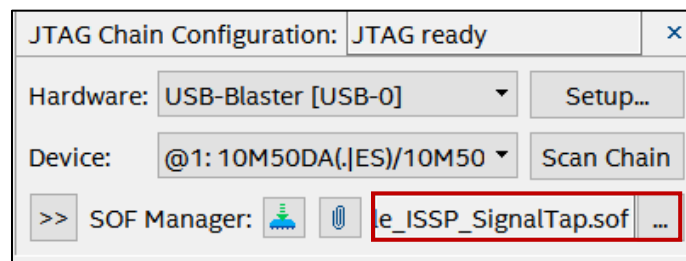




FIGURE 47 PROGRAMMER WINDOW IN SIGNAL TAP

2. Browse and select the **Example_ISSP_SignalTap.sof** file in the project directory.
3. Click  to download the configuration file that includes the Signal Tap logic analyzer to the development board.

To get the system up and running, you need to get it out of reset. By switching on the SW0 you will get the system out of reset. Also, since the clock on the board runs at 50 MHz which is really fast for the naked eye, there is a delay present in the code which helps the number to be displayed on the seven segment display.

Section 3: Run analyzer

1. In the Signal Tap Window, make sure the instance is selected.
2. Click on  to start running the analyzer. You can observe “Waiting for Trigger” in the instance manager.
3. Now, turn on the switch SW0 on the board to get the design out of reset (which is a trigger).
4. Analyze the acquired data. This might take a while as you have to wait till the counter counts till 99. You will observe that the samples start as -512 till 512 as out trigger is center triggered and sample depth is 1k. Also, the size of each capture is synchronized with the clock in the design.
5. To zoom in data, just click on the waveform. To zoom out in the waveform, Shift Key + Click.

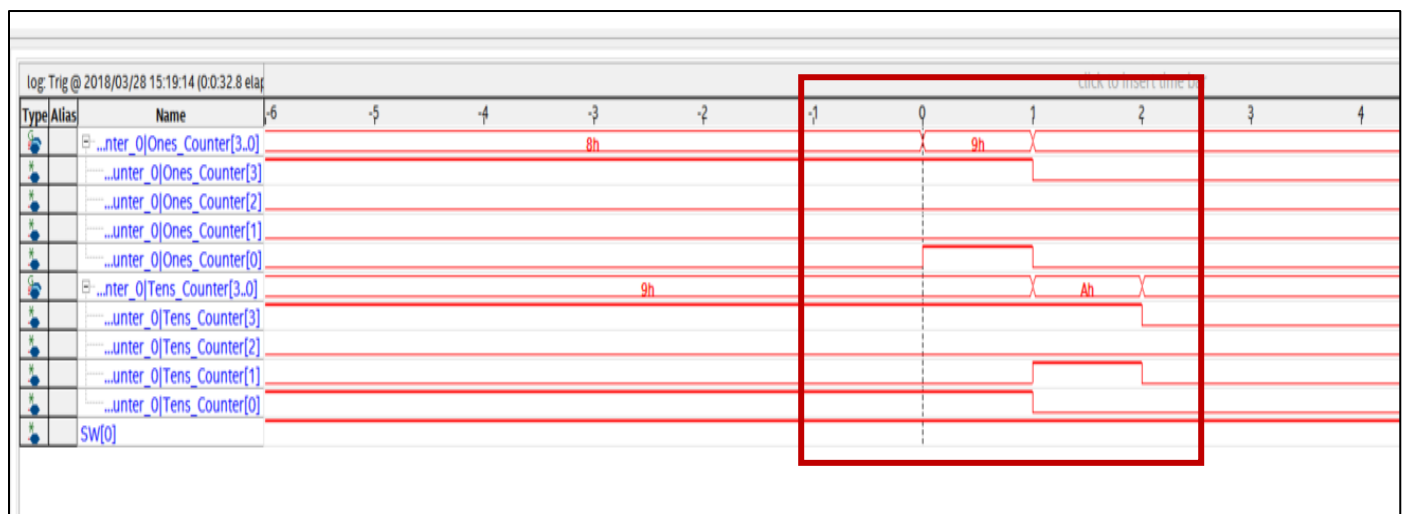


FIGURE 48 SIGNAL TAP OUTPUT LOG

Now, once all the trigger conditions occurred i.e. the instance manager will turn green. You may notice in the data log waveform that the number 9 in the counter for HEX0 (Ones_Counter signal) display occurs only once before it switches back to a zero as shown in the figure above.

So, the number nine is never displayed on the HEX0 display (as it may be occurring for a very short period of time) as it only occurs for one cycle as it is synchronously reset after it.

To be able to display “9” in the seven segment display, the code needs redesigning.

Solution:

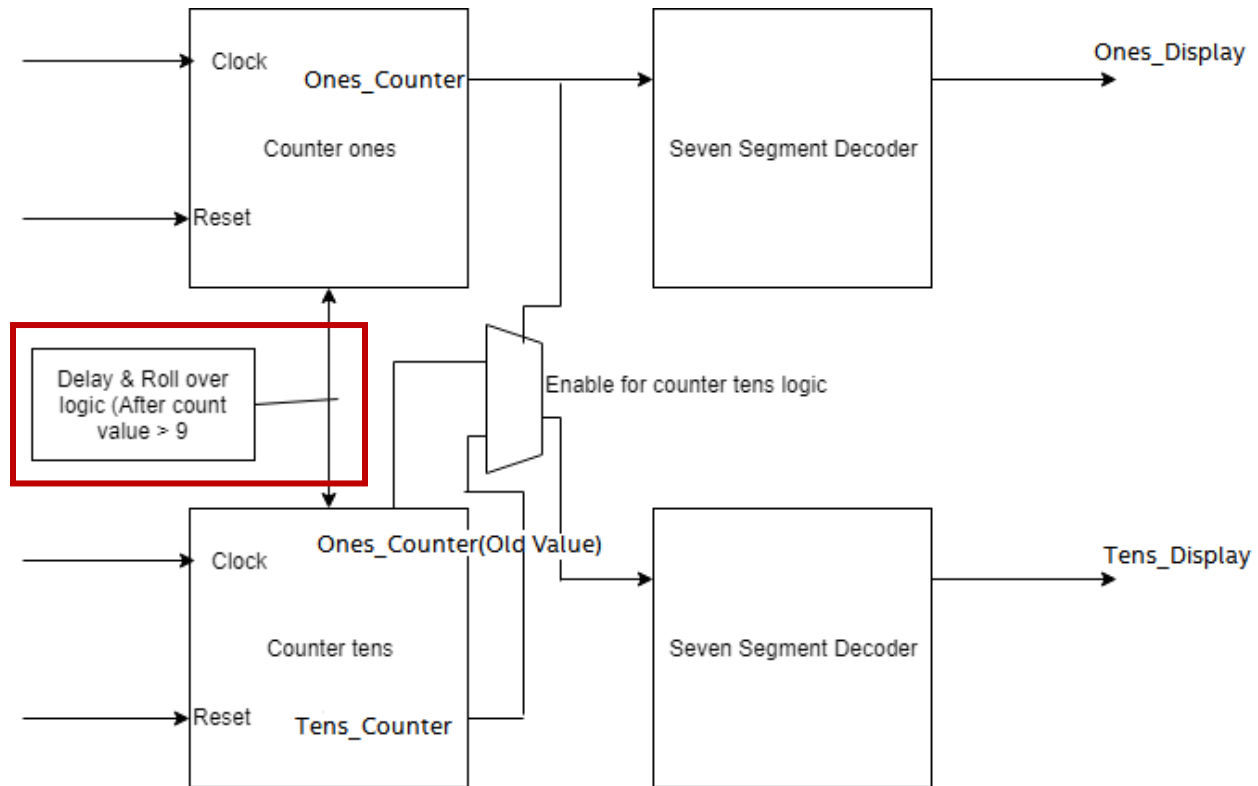


FIGURE 49 REFORMED DESIGN FOR 0-99 COUNTER

In the previous design, we could not observe “9” in the seven segment display units. If you look at the code, you will observe that “9” was skipped because we were rolling over to zero value too soon not giving enough time for the display decoder logic to work. Now, in this new design we reset the value of counter for HEX0 only after the delay logic.

There is a file named **“Example_ISSP_SignalTap_solution.v”** in the project directory, which is the correct code for the counter. If you have time, add this file to your project and remove the current **“Example_ISSP_SignalTap.v”** file. Recompile and verify the functionality using Signal Tap file or In-System Sources and Probes file.

Conclusion

With this exercise the student is introduced to signal tap analyzer and debugged a design. The next lab exercise introduces system console based on platform designer (Qsys) tool.

System Console

Introduction to System Console

System Console is an interactive console for system-level debug of Platform designer (Qsys)-based systems over JTAG, USB, and other types of connections. Based on the TCL scripting language, it has a simple set of commands for communicating with various parts of your Qsys system.

Lab Exercise 4

Objective:

In this lab exercise the student will learn how to use System Console for debugging using JTAG to Avalon Bridge. The given design consists of an On Chip Memory, PLL, LEDs and switches connected by master clock and reset. The JTAG to Avalon Bridge will act as a communication link to debug the components on the board.

The design for this lab exercise is based on Platform designer (Qsys) system. It is a PLL connected to an on-chip memory with LED's and Switch's all connected to a common clock and reset.

Get Project files

1. In the downloaded **Workshop_DebuggingTools.zip** folder. Go to **Lab_4** Folder.
2. Double-click on **systemconsole_example.qpf** to restore the project.

Section 1: Add to the JTAG to Avalon Master Bridge Platform Designer Component

1. From **Tools** menu, select **Platform Designer**.
2. A dialogue box should automatically open, click the **systemconsole_example.qsys** to open it as shown in figure below.

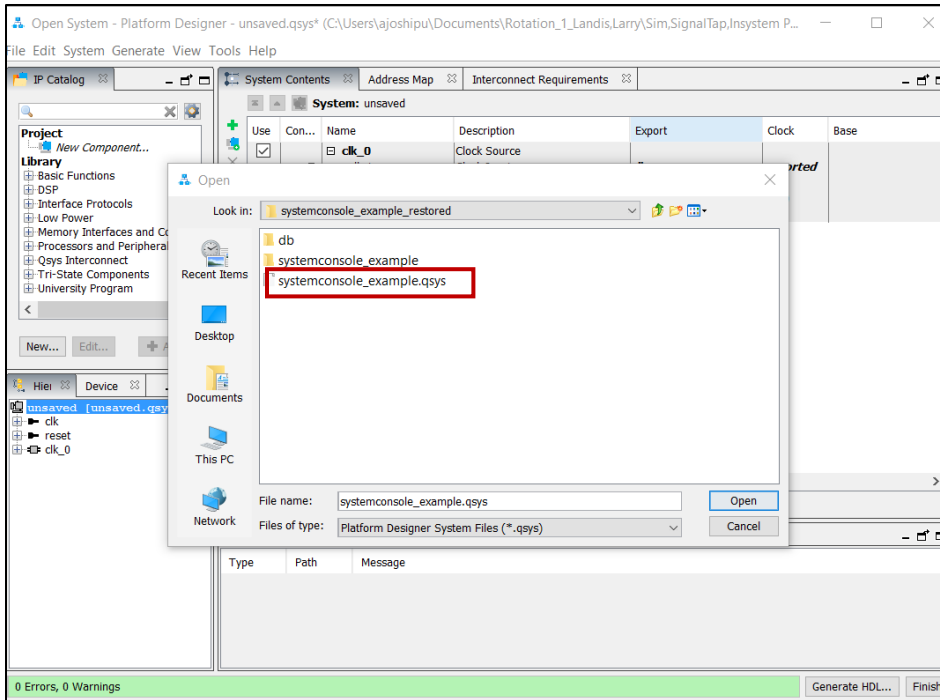


FIGURE 50 OPEN DIALOGUE BOX

3. In the IP Catalog, type JTAG in the search box. Double click on JTAG to Avalon Master Bridge. If you cannot find IP Catalog, go to View menu and select it. Click **Finish** to accept all default options. You should now see a master_0 JTAG to Avalon Master Bridge component at the bottom of the window.

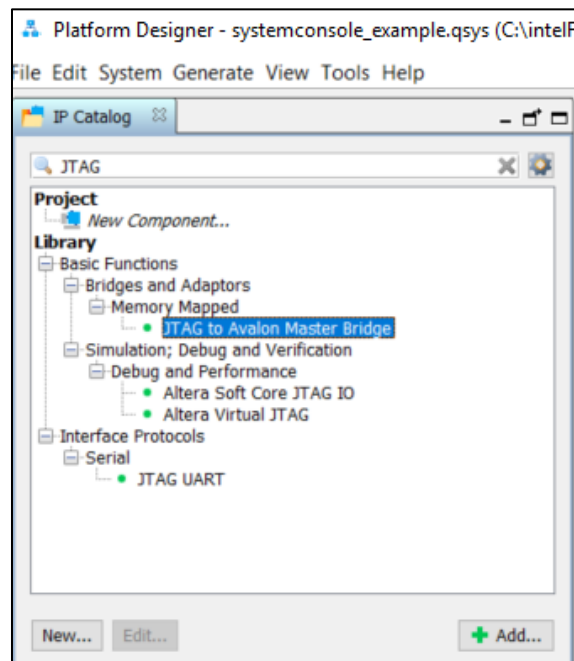


FIGURE 51 IP CATALOG IN PLATFORM DESIGNER

4. Connect the bridge to the rest of the system.
 - a. Connect master_0 clk input to the system clock. Make the connections by clicking the dots in the connections column to the left of the components.
 - b. Connect the “master” interface of the JTAG to the Avalon Master Bridge to all of the memory mapped slaves in the system by following the gray line. Connections should look as shown in the figure below.

Your system should now be error free.

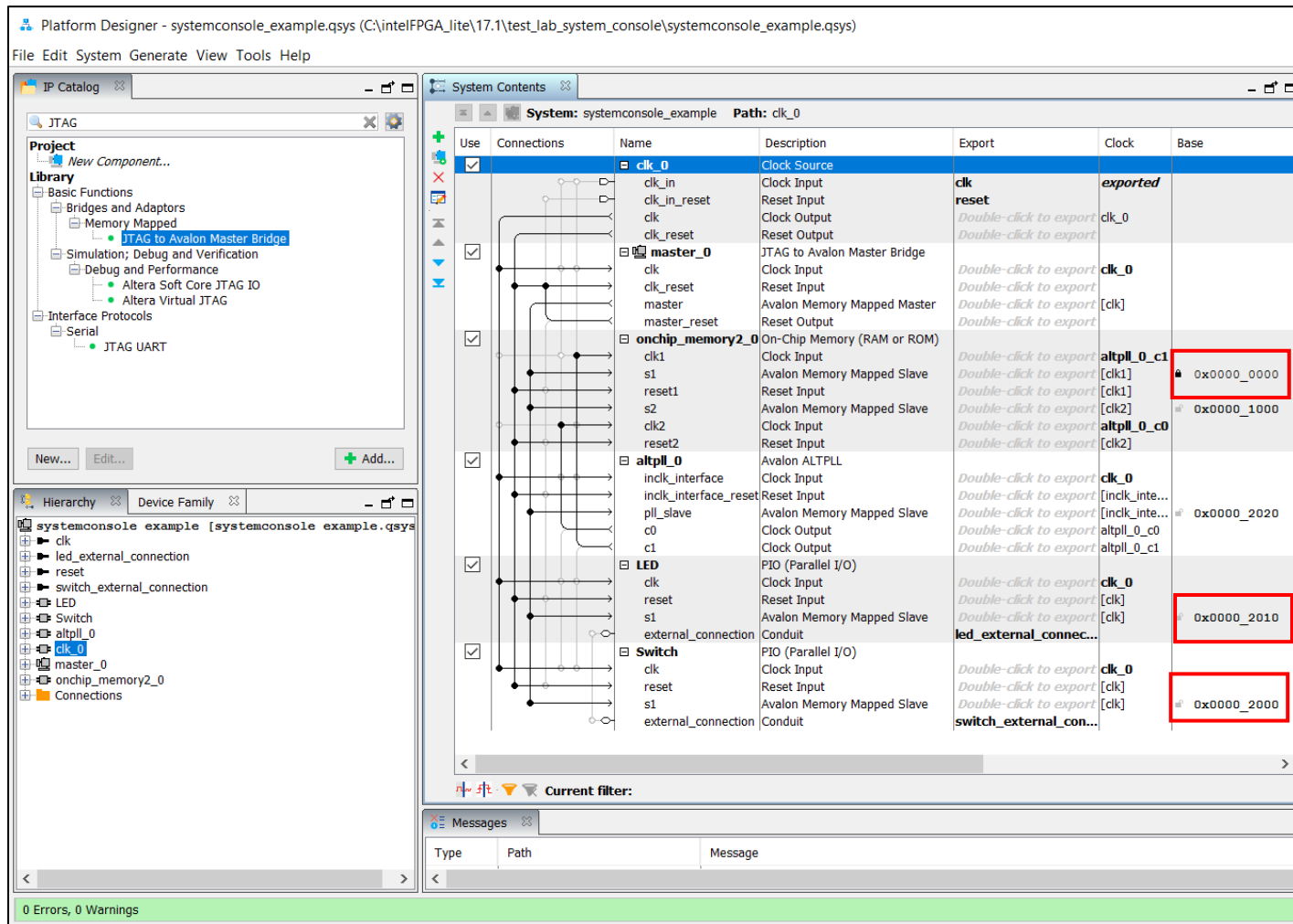


FIGURE 52 PLATFORM DESIGNER GUI

5. Generate the system by clicking on **Generate HDL..** button at the bottom.
6. Make sure HDL design is set to Verilog and click **Generate**.

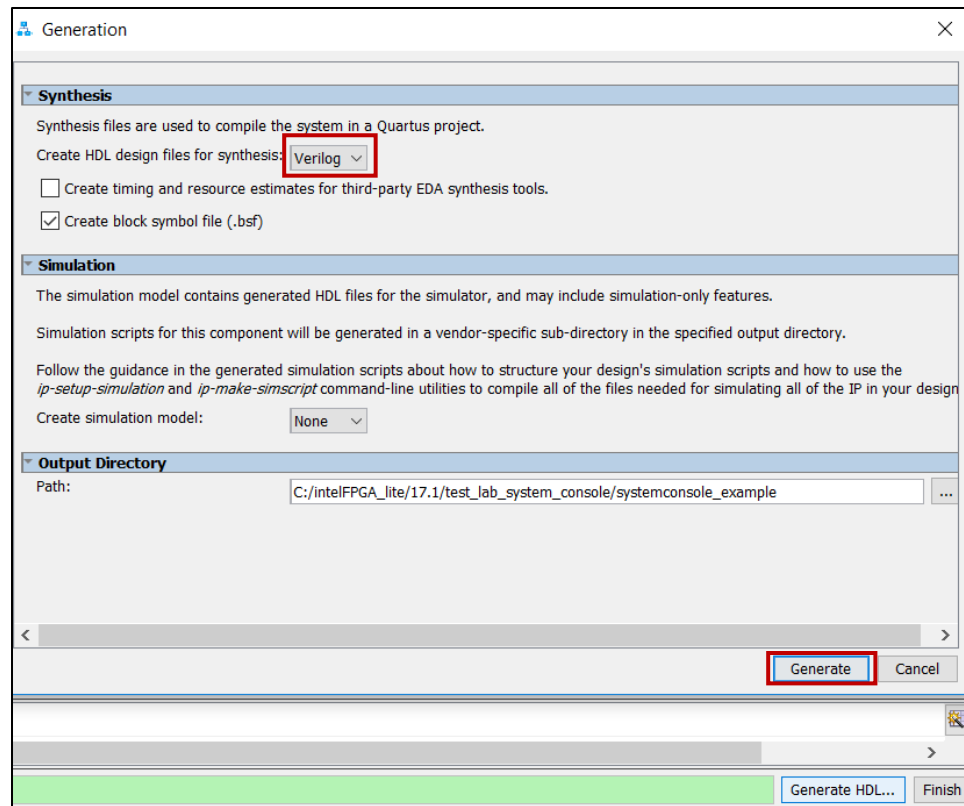



FIGURE 53 GENERATION WINDOW FOR PLATFORM DESIGNER

7. Before the generating is completed, a **Save System Complete** dialog box will appear, go ahead and select **Close**. Another dialog box will appear letting you know that the design is generating. Once that is finished, select **Close** again.
8. Once completed, compile the Quartus design by going to the **Processing** tool menu on the main Quartus window, and selecting **Start Compilation** 

Section 2: Launch System Console and Program the FPGA

1. Ensure that the board is connected and powered on.
2. In Platform Designer tool, make sure **systemconsole_example.qsys** is still open. If not, go to Tools menu in Quartus Tools → System Debugging tools → System Console and skip the next step.
3. Launch System Console tool by selecting System Console from the Platform Designer Tool menu.

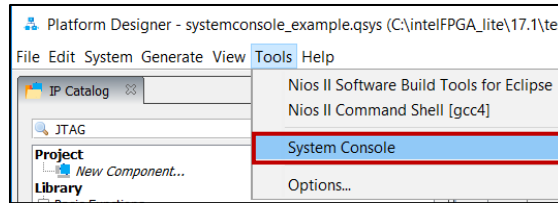


FIGURE 54 LAUNCHING SYSTEM CONSOLE FROM PLATFORM DESIGNER

4. Verify that system console can see the connections in the JTAG chain in the system console System Explorer window as shown in Figure 55 below.
If you do not see the connections, go to the system console tools menu and select **Refresh Connections**.

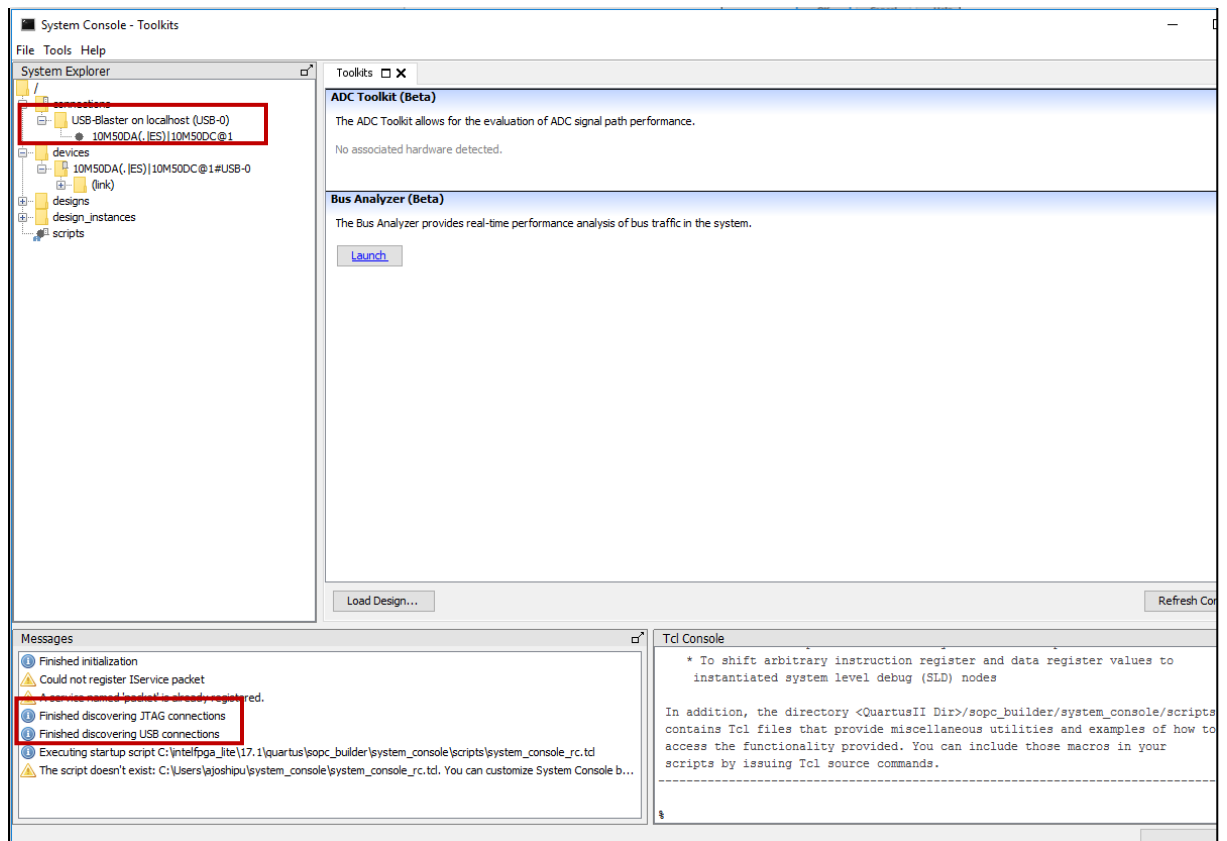


FIGURE 55 VERIFYING SYSTEM CONNECTIONS

5. To program the device, in the system explorer window, right click the device and choose Program device and **systemconsole_example.sof** file.

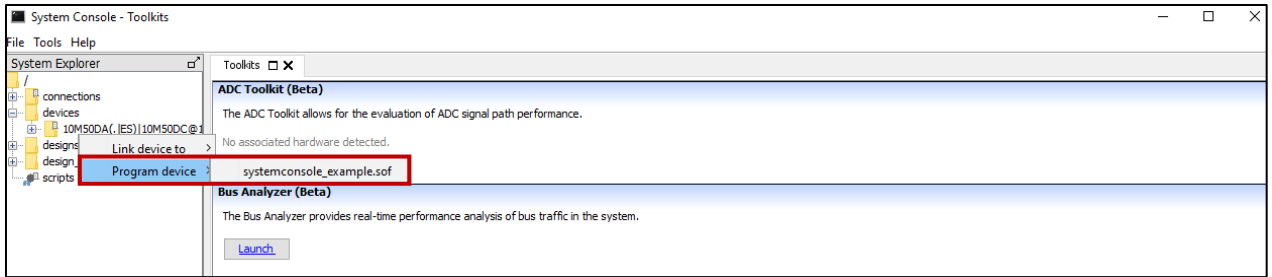


FIGURE 56 PROGRAMMING DEVICE FROM SYSTEM CONSOLE

Once the device is programmed you can confirm it by the system console response in the messages window.

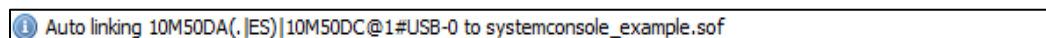


FIGURE 57 RESPONSE AFTER PROGRAMMING FROM MESSAGES WINDOW

Section 3: Verify Clock and Reset from the System Console Tool

Here we will be using the `jtag_debug` service to perform some board bring up tasks. To perform these `jtag_debug` tasks you need to include a JTAG to Avalon Master Bridge in your system which we have.

For using the `jtag_debug` service, we first will use the `get_service_paths` command to locate a jtag debug path and assign it to a variable `jd_path`, it will then open the path, and finally we will close the path when all the tasks are performed.

1. Create a **`jd_path`** variable that points to the JTAG to Avalon bridge component by typing

`set jd_path [get_service_paths jtag_debug]`

```
$ set jd_path [get_service_paths jtag_debug]
/devices/10M50DA(.|ES)|10M50DC@1#USB-0/(link)/JTAG/alt_sld_fab_sldfabric.node_0/phy_0
```

FIGURE 58 COMMAND TO SET JTAG PATH

Here, `jd_path` is a variable that is created to represent the value returned by `get_service_paths jtag_debug`.

2. Verify the signal integrity of JTAG chain by passing a test pattern through it and making sure you get the same values back from the JTAG chain.

Type:

`jtag_debug_loop $jd_path [list 1 2 3 4 5 6 7 8 9 10]` in the system console tcl Console window. This is a useful sanity check to make sure the JTAG chain is passing through the bits expected and not corrupting them. This can be useful for large JTAG chains.

```
$ jtag_debug_loop $jd_path [list 1 2 3 4 5 6 7 8 9 10]
0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0a
```

FIGURE 59 COMMAND FOR VERIFYING SIGNAL INTEGRITY

3. Verify that the clock is toggling by entering the following commands in the System Console Tcl window.

- a. **jtag_debug_sense_clock \$jd_path**

This command simply senses if the clock has ever toggled, returning a 1 if it has.

```
% jtag_debug_sense_clock $jd_path
1
```

FIGURE 60 COMMAND FOR SENSING CLOCK

- b. **jtag_debug_sample_clock \$jd_path**

This command asynchronously samples the clock signals. You will likely have to run this command several times to witness a change.

```
% jtag_debug_sample_clock $jd_path
0
% jtag_debug_sample_clock $jd_path
0
% jtag_debug_sample_clock $jd_path
0
% jtag_debug_sample_clock $jd_path
0
% jtag_debug_sample_clock $jd_path
1
```

FIGURE 61 COMMAND SAMPLING CLOCK

4. Verify that the reset signal has been released by typing **jtag_debug_sample_reset \$jd_path** command in the TCL window to sample the current value of the reset signal.

```
% jtag_debug_sample_reset $jd_path
1
```

FIGURE 62 COMMAND FOR RESET

Section 4: Perform Master Reads and Writes to Peripherals in the FPGA

Here we will be using the master service type to issue commands to read from or write to Avalon Memory Mapped slave interfaces. To use the master service type, you'll either need the JTAG to Avalon Bridge or USB debug master or a Nios II processor with debug module enable. In our case, we are using a JTAG to Avalon Bridge.

1. Create a variable **m_path**, that points to the JTAG to Avalon bridge component by typing **set m_path [get_service_paths master]**

Here, we are setting up the master service by using **get_service_paths** command to find service master paths and sets the variable **m_path** to the master service.

```
% set m_path [get_service_paths master]
/devices/10M50DA(.|ES)|10M50DC@1#USB-0/(link)/JTAG/alt_sld_fab_sldfabric.node_0/phy_0/master_0.master
```

FIGURE 63 GETTING MASTER PATH

2. Claim the master service to allow exclusive access to the master device by typing **set c_master [claim_service master \$m_path ""]** in the TCL console window.

```
% set c_master [claim_service master $m_path ""]
/channels/local/(lib)/master_1
```

FIGURE 64 SETTING MASTER PATH

With the service claimed as the c_master variable, system console then allows other users to access other memory ranges and denies access to the claimed memory range.

3. In Platform Designer, note the base address of the on chip-memory component. The base address is shown as a column in System Contents tab of Platform Designer tool. This will be the starting address in the next command in our case it is 0 (Refer to Figure 52)
4. Now let's read from and write to a memory component
 - a. Type **master_read_memory \$c_master 0 32**
 - b. Type **master_write_memory \$c_master 0 {0 1 2 3 4 5 6 7 8 9 10}**
 - c. Type **master_read_memory \$c_master 0 32**

```
% master_read_memory $c_master 0 32
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00
% master_write_memory $c_master 0 {1 2 3 4 5 6 7 8 9 10}

% master_read_memory $c_master 0 32
0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0a 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00
```

FIGURE 65 READING & WRITING MEMORY

5. Close the service by typing **close_service master \$c_master;**

As you can see it's very easy to use the system console to get access to memory-mapped components in your system.

Section 5: Run the System from the System Console Toolkit GUI

1. From the Quartus software File menu, choose file type filter to be "All Files". Open **SysConsoleScript.tcl** found in the project directory and examine it in the editor of your choice.

In the figure below, are some snippets of code from the TCL script. Make sure that the address of LED's and Switch's in the script is same as the base addresses in your Qsys system. See Figure 52, the highlighted values in the figure match with the addresses present in the script in the figure below. If that is not the case with you, edit the values with the base address and save the script.

```

proc toggle_led {led } {
    global master_path
    set led_read [master_read_16 $master_path 0x2010 1]

    if { $led == 0 } {
        set led_set [expr 1 ^ $led_read] }

    master_write_16 $master_path 0x2010 $led_set

    set led_read [master_read_16 $master_path 0x2010 1]

    if { [expr { $led_read & 1 }] == 1 } {
        toolkit_set_property LED0_led color red
    }

# allback Procedure
proc update_sw {} {
    global master_path
    set sw_data [master_read_16 $master_path 0x2000 1]

    if { [expr { $sw_data & 1 }] == 1 } {
        toolkit_set_property SW0 color green
    }
}

```

FIGURE 66 SNIPPETS OF TCL SCRIPT

2. In System Console tool TCL console, register the toolkit by typing **toolkit_register SysConsoleToolkit.toolkit** in TCL console window
3. In System Console Tools menu launch the toolkit by choosing "System Console Toolkit"

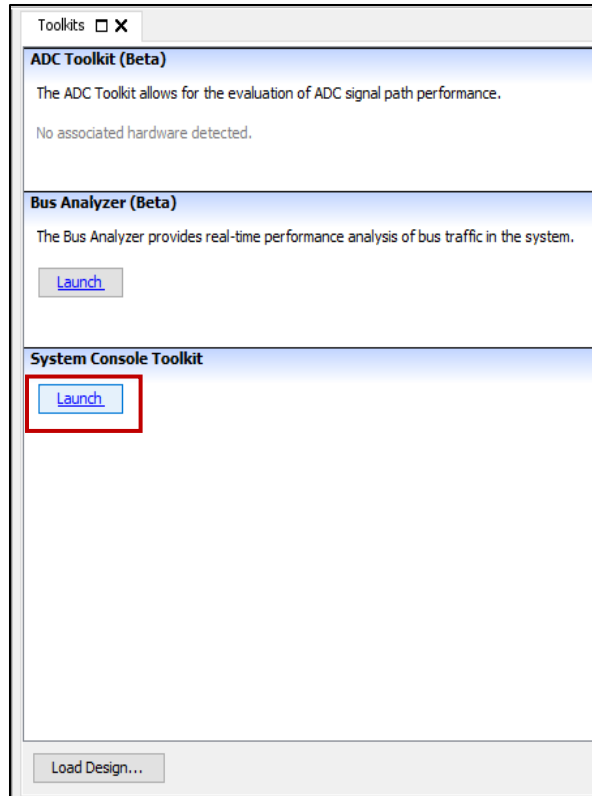


FIGURE 67 LAUNCHING GUI

4. Push the LED Toggle buttons on the dashboard and confirm that corresponding red LEDs change states on development board.
5. Flip some of SW switches on the board and push the update button. Verify that corresponding Switch setting LED changes on the toolkit window.

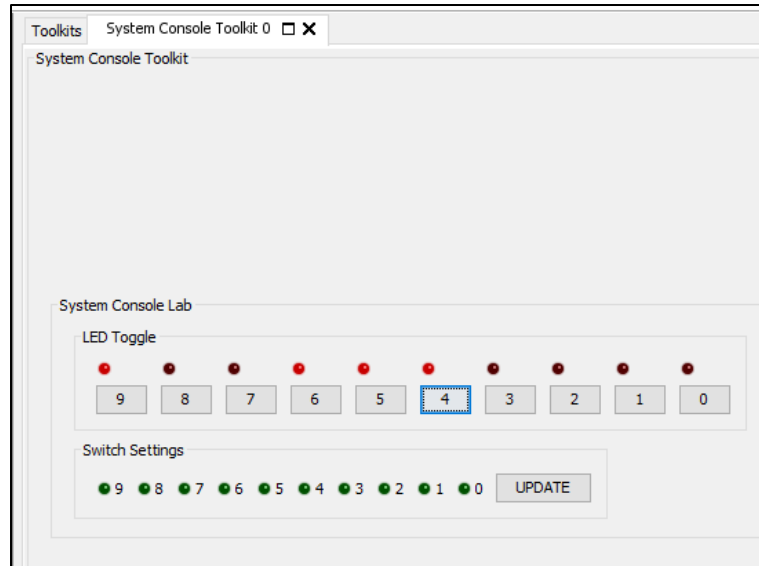


FIGURE 68 GUI FOR LED'S AND SWITCH'S

6. Exit System Console.

Conclusion

With this exercise the student were introduced to system console and learned how to use it for debugging a platform designer (Qsys design). Just like the toolkit used in the lab, there are several other toolkits available for free to use in the system console tool itself such as ADC toolkit, Bus Analyzer etc.