



# Introduction to FPGA Simulation and Debug using the San Jose CR3101 Hands-On Lab

©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](http://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

## 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.
- DE1-SoC and/or Cyclone GX Starter development kits as provided through the SJ3101 Hands-On Lab, needed for Labs 2 and 3.
- Quartus Prime Standard, which is provided in the Hands-On Lab

## HINTS AND TRICKS

---

*Some helpful things to keep in mind. Refer to these if you have problems!*

- Webex Best Practices:
  - When using Webex training center, select a PC (see lower right corner). Your name is placed below the PC name. Select your own PC, not one that is already occupied.
  - If you select a PC that has a name below it, you can observe what others are doing, ok to do if agreed upon.
  - If Quartus is already open when you access your machine, close it out and follow instructions step by step to launch the correct version of Quartus
  - Audio connectivity has a mode where if you select "YES" when you connect to a machine, you have your own audio breakout room when you are working on your PC. Others can privately talk within your room but you lose audio connectivity to the main session. "NO" keeps your audio feed in the main session and you will hear background chatter from others in the training.
  - When you leave your PC session, you can return to your PC in it's same state. Leave your PC session to view the chat window within the main session or listen to the main audio feed.
  - Send a chat message in the main session if you are having problems and someone will join your session within the breakout room audio feed. When you rejoin your PC session select YES so you have the audio feed in your breakout room.
  - Open up chat within your breakout room, that way the host/TA can communicate that way with you. Look for broadcast messages to all students.

Periodically check chat the main session chat and your breakout session for announcements.

- If you are TA'ing, remind the host to make you a panelist so you can see the full chat feed.
- Lab Best Practices:
  - You must use the **18.0 Standard** version of the Intel Quartus Prime software. This version requires no license and is supported in the computer lab you are accessing. The Intel® Quartus® Prime Pro Edition software **will not work** as it does not support the target hardware.

# Accessing your lab PC

Launch your Webex session. You will select your own PC through Webex Training, the Webex package that enables connection to remote PCs to use throughout the duration of this training. Your instructor will guide you how to connect to a lab PC. In the lower right corner of Webex, you should see lab machines available, or alternatively select the labs pull down from the top screen of Webex. Note that an occupied machine has a person's name *below* the machine name. Select an unoccupied PC to connect to. You will be given a choice to hear audio from the group session or in your room with your local PC you selected. Since you will be all by yourself in your PC room, you might want to continue listening to the group feed until necessary to have a debug conversation with the leader or panelist.

Once you see a Windows login prompt, your login is student and the password is QPrime.1.

Once connected, you *might* see an existing Quartus session open from a previous student. Close that session, we will start with a fresh version of **Quartus Prime Standard 18.0**. Using the wrong version of Quartus Prime could lead to later problems in the lab. Quartus Prime Standard supports the lower complexity FPGA devices called MAX and Cyclone. The higher complexity devices offered by Intel called Arria, Stratix and Agilex use a version of tools called Quartus Prime Pro. There is a third version called Quartus Prime Lite which is entirely free and supports MAX and Cyclone class FPGA devices. This version has less optimization and IP options than the Quartus Prime Standard but is ideally suited for university level coursework and capstone projects.

- ☐ Open the desktop folder called Quartus Shortcuts. Beneath that folder open the folder called Intel FPGA 18.0.0.614 Standard Edition.
- ☐ Launch the Quartus executable under this folder: Quartus Prime Standard Edition 18.0.0.614. The Quartus GUI will launch and occupy the entirety of your screen.
- ☐ Determine which board you are connected to. This can be achieved by launching this tool: Tools → Programmer. Next to the Hardware Setup you will see either USB-Blaster [USB-0] or DE-SoC [USB-1]. Please take note of which type of development kit your remote machine is directly connected to:

Development Kit	Hardware Setup
Cyclone V GX Starter	USB-Blaster [USB-0]
DE1-SoC	DE-SoC [USB-1]

Make note of whether you are using a Cyclone V GX Starter or DE1-SoC. If that field comes up **blank**, immediately logout from your connected machine and try another computer. If you fail to follow the unique instructions per board you will be unable to complete the lab successfully. Let the instructor know via chat if the programmer fails to find a board.

## Downloading the Project Files

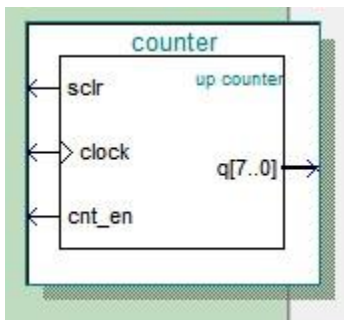
- ☐ Download the project files from the link provided here: [https://github.com/intel/FPGA-Devcloud/tree/master/main/QuickStartGuides/Workshop\\_Simulation\\_Debug\\_Remote](https://github.com/intel/FPGA-Devcloud/tree/master/main/QuickStartGuides/Workshop_Simulation_Debug_Remote)

- ☐ Copy this link to the Webex Training Center and download the starting files to somewhere on the remote machine C: drive – either downloads or documents is ok. Name the folder without spaces eg C:\Users\Student\Downloads\Jane\_SimDebug\_Workshop .
- ☐ Extract the files using zip extract command.

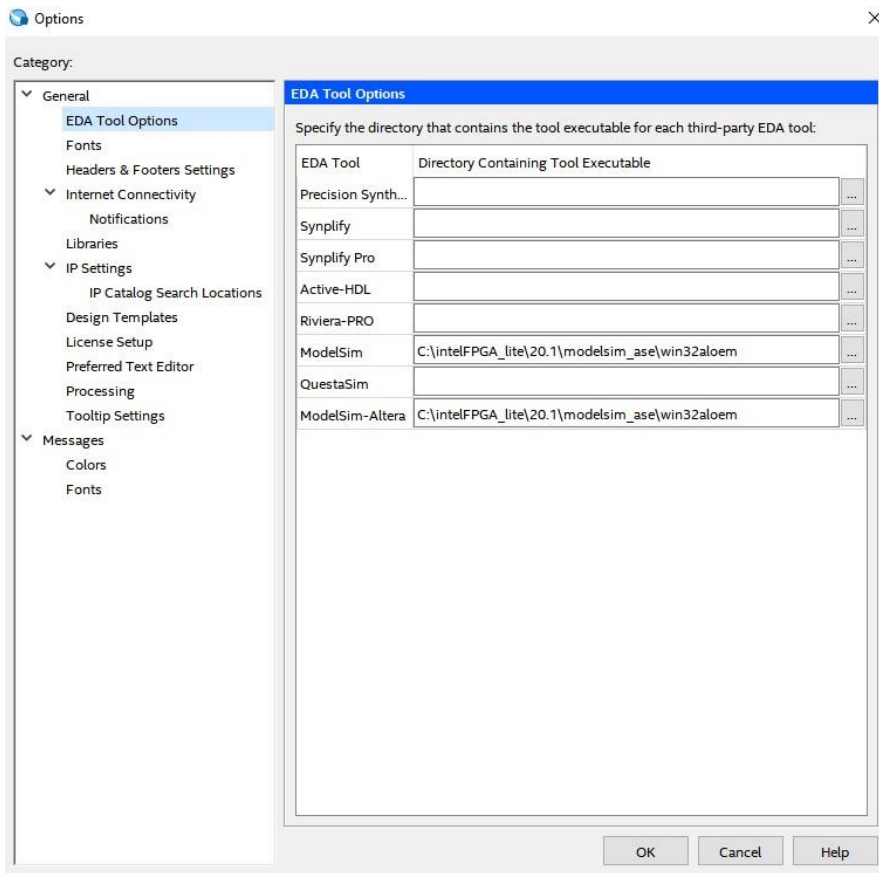
## Simulation using the University Waveform Viewer

- ☐ Launch Quartus as specified above.
- ☐ Create a new project: File → New Project
- ☐ Use the feature to copy settings from a past previous project, this will save you some time or use the File → New Project. The family is Cyclone V and device is 5CSEMA5F31C6. Select the device and finish.
- ☐ At this time, you won't have any files in the project navigator. Using the IP catalog on the right side of Quartus, search for counter. Select LPM counter.
- ☐
- ☐ Double click on the counter, name it counter8.

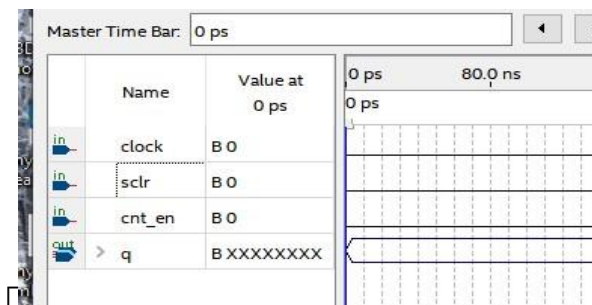
Features should include 8 bits wide with count enable and synchronous clear.



- ☐ Click finish and add the counter8 to your project (project → add/remove files to project).
- ☐ When you are done, you will 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)
- ☐ Next you need to setup the path to Modelsim: Tools → Options
- ☐ Select EDA Tool options. Fill in the path to Modelsim as follows with the appropriate path and right tools version (18.0). Note that it might already be set up from previous users of Quartus on your lab machine. The path to the tools might be different than the image below.



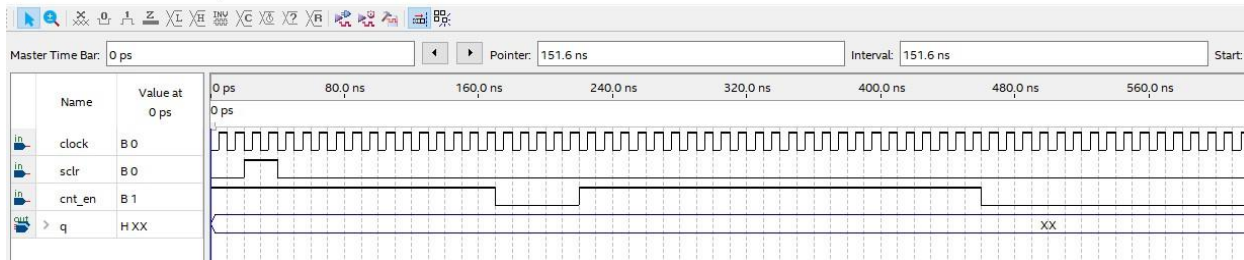
- ☐ Next, we will create waveforms and simulate. File→New→University Program VWF
- ☐ Right click on left side. Insert Node or Bus
- ☐ Click list, and you will get a list of I/O signals
- ☐ Add clock, sclr, cnt\_en and q (the bus version). Hit OK. Arrange the order like this:



- ☐ 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.

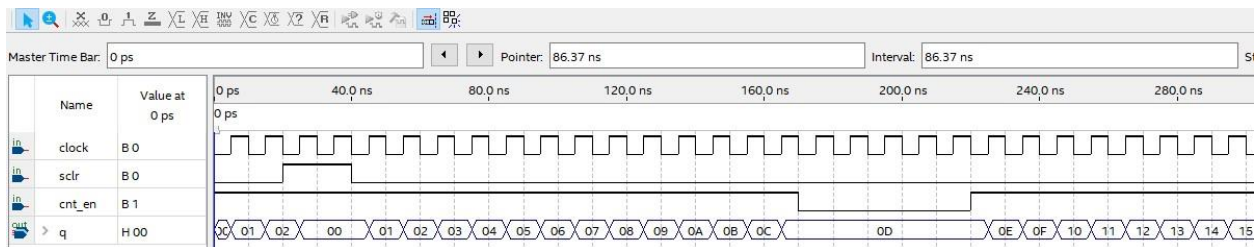


- ☐ The clock will utilize the pocket watch icon, set sclr and cnt\_en are set high or low by highlighting the time you want to change the waveform and make it look something like the following (note select one signal at a time, the tool ):



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 (now part of Siemens). 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.

- ☐ If your prior Quartus project is open: File → Close Project. After Un-zipping the files. Open the Quartus 18.0 Standard Software if not already open.
- ☐ In the main menu, File→Open Project.
- ☐ Navigate to the folder where you unzipped the files (In C:// or Documents). Go to Lab\_1 Folder and select Simulation\_Example.qar
- ☐ 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) .



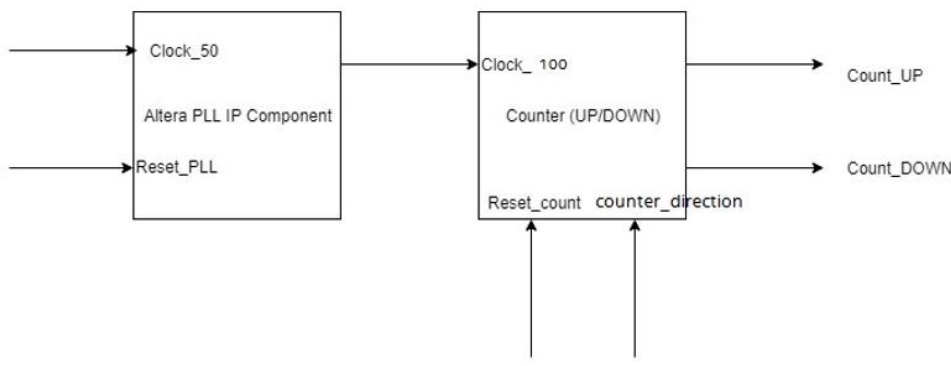
*Objective:*

*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.*

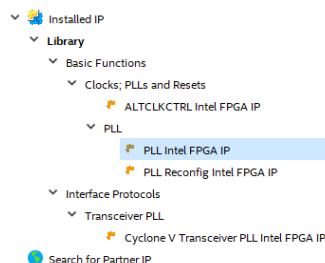
This lab exercise consists of a simple digital logic design of an up/down counter and phase locked loop (PLL) as shown in the block diagram below. The PLL is used to increase the frequency of the input clock from 50 MHz to 100 Mhz.

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 a fairly lengthy startup time associated with it. The specification for the Cyclone V 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.

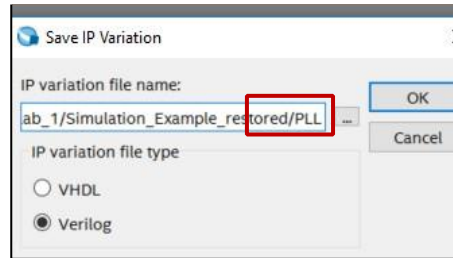


### Add IP Component to the design

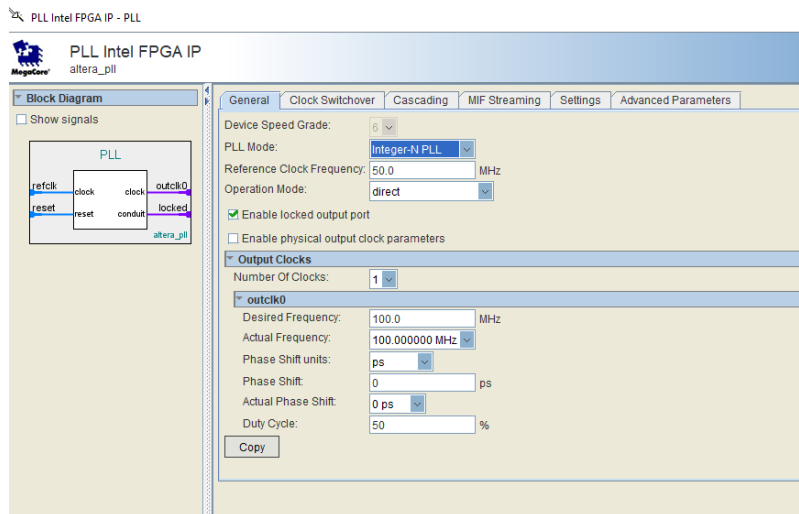
- ☐ In the IP Catalog, search for PLL. Double click to add Intel PLL FPGA IP (aka ALTPLL). If you do not have the IP Catalog already open, go to View menu→IP Catalog (IP megawizard) to view the window.



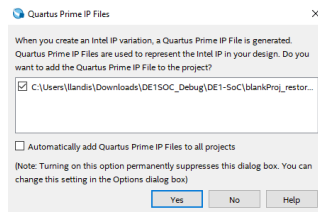
- Name the file **PLL** in the Save IP Variation box that came up and click **OK**. Please make sure to name it **PLL**.



- In the MegaWizard dialogue box, make the reference clock frequency **50 MHz** and outclk0 100 MHz.



- 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**.



## Observing the Test-bench for the design

- ☐ In the Main Menu, Go to File → Open... → Lab 1 and Open **tb\_counter.v** file.
- ☐ This is a simple testbench that stimulates the top\_counter.
- ☐ 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.
- ☐ 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.
- ☐ Next, all the outputs are declared as type **wire** i.e count\_up and count down.
- ☐ In the instantiation of the **top\_counter.v** module, the connections of the modules are made.
- ☐ 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.
- ☐ 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**

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
```

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

```

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

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

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

        #10 reset_pll=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;

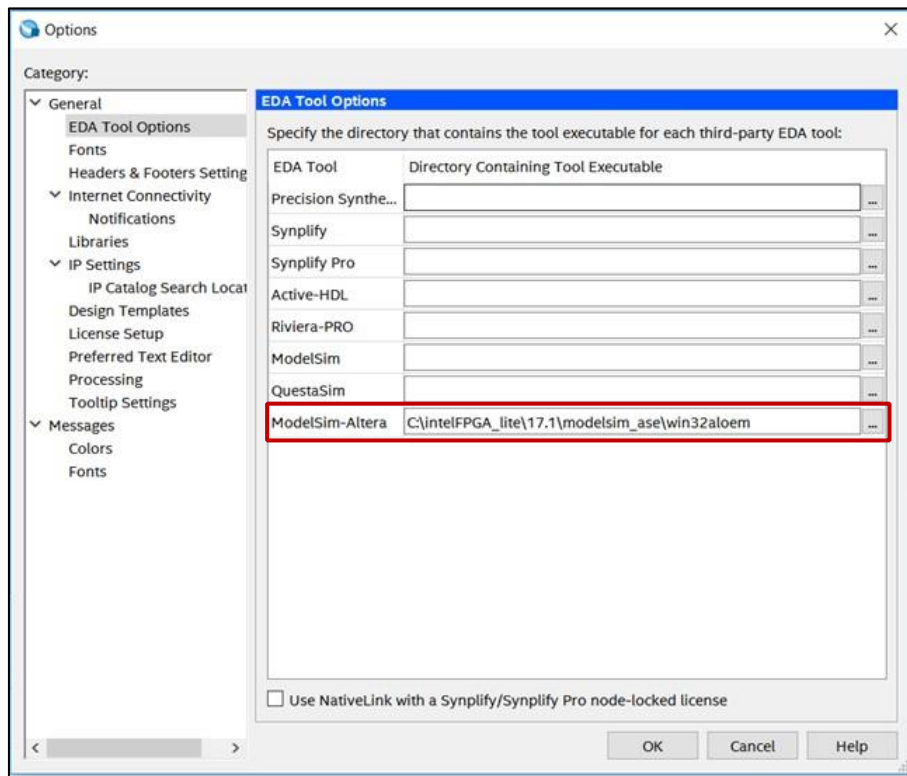
        #1000 $stop;
    end

    always #10 clock = ~clock;
endmodule

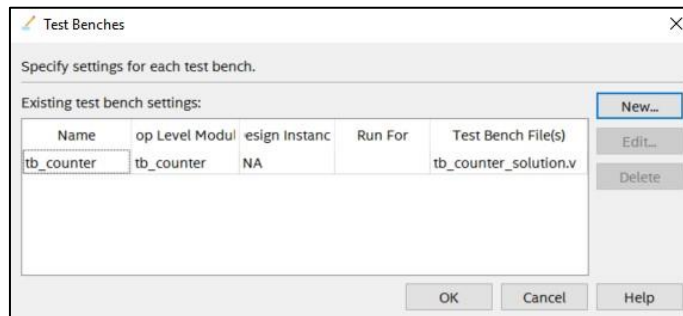
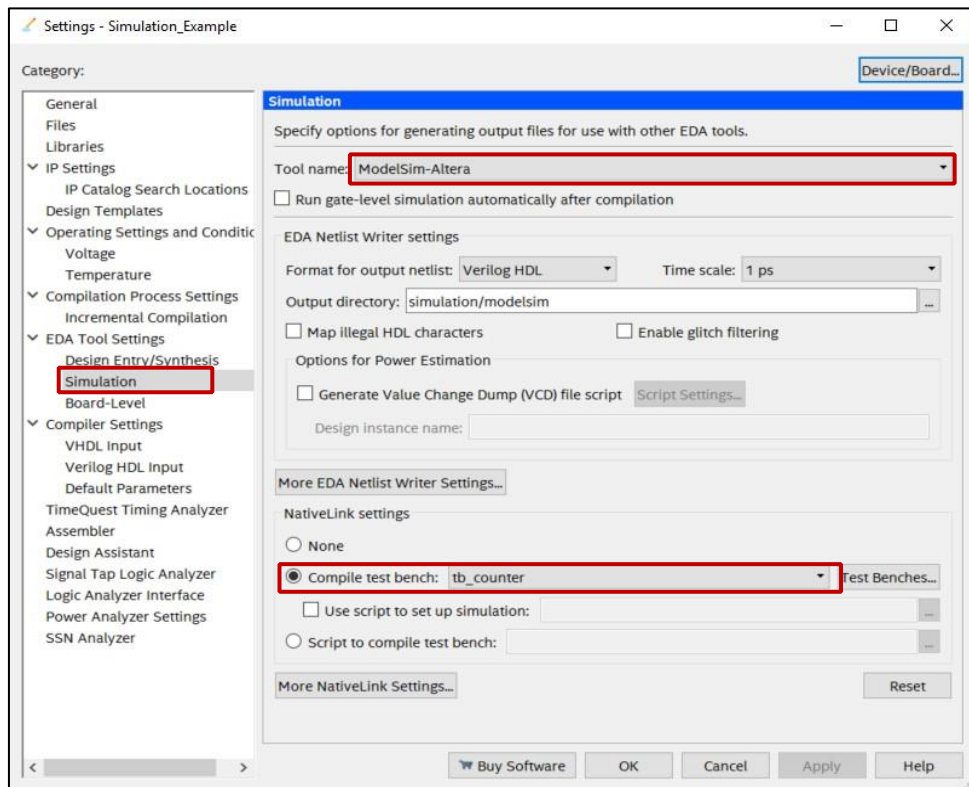
```


## Setting up ModelSim from Quartus

- Once you have completed section 1 and 2. You need to launch ModelSim for simulations. If you haven't already done 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.



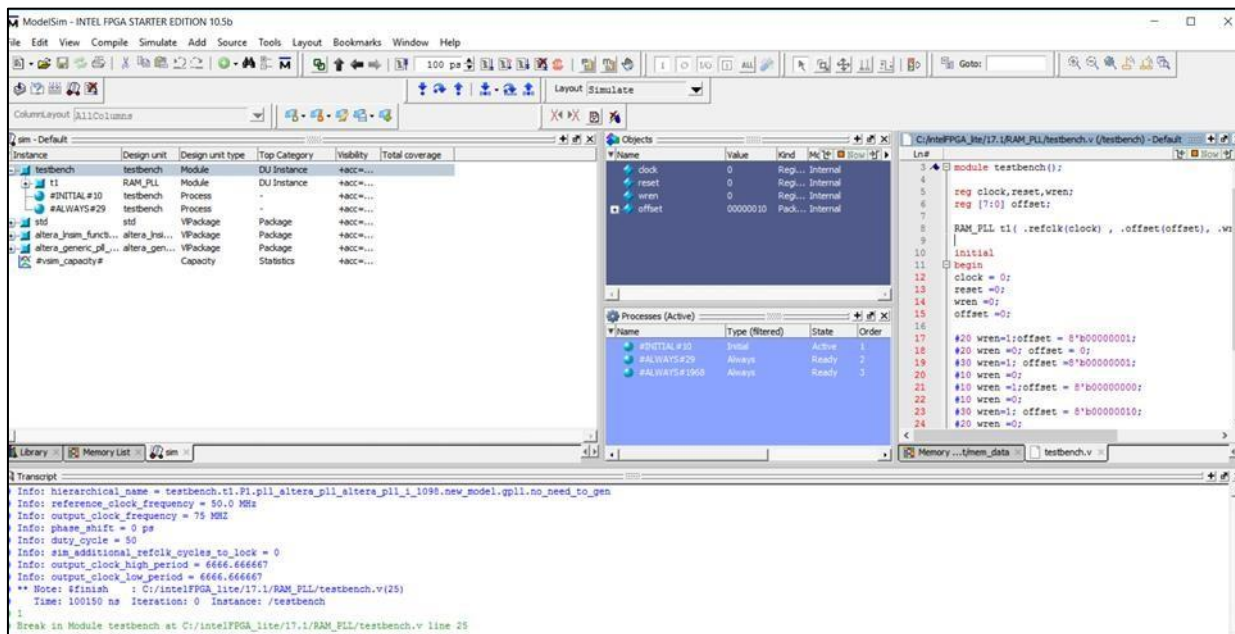
- ☐ Next step is to make sure that the test bench is compiled. Go to Assignments→Settings→EDA Tool Settings→Simulation. Select ModelSim-Altera in the Tool Name.
- ☐ Under NativeLink Settings→Select Compile Test Benches and click on Test Benches..
- ☐ In the Test Benches Dialog Box, Click on New.. Select 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.



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



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

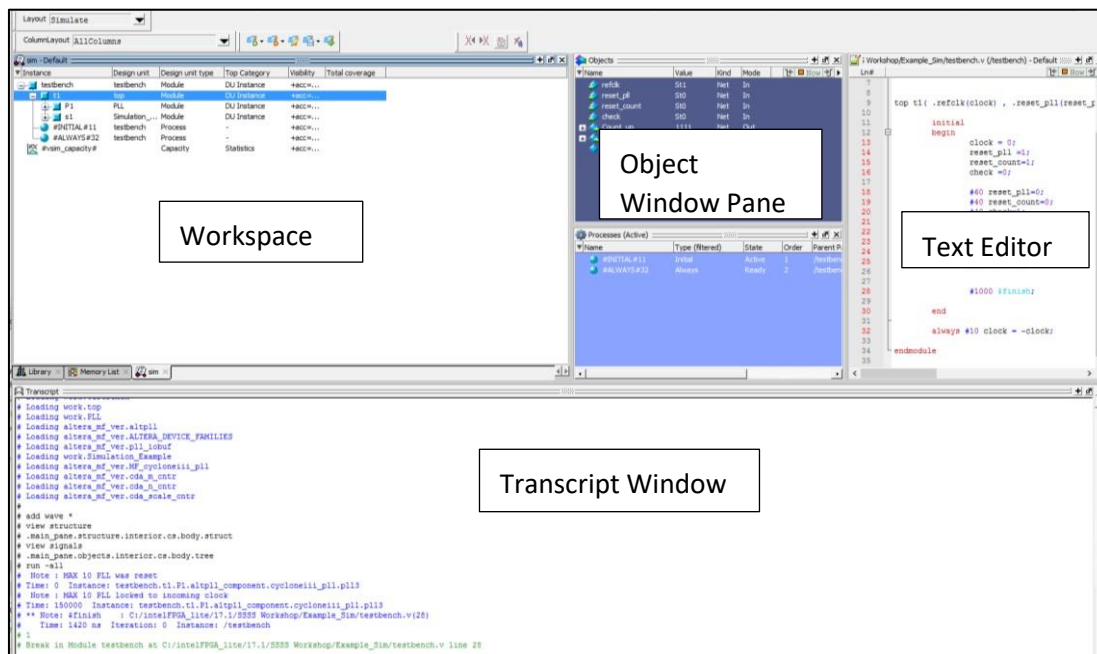


*Note: If you get error loading the design related to PLL.vo file. You might have to go to your folder directory where the project is stored and copy the PLL.vo file from the PLL\_sim folder to the main project folder. And repeat step 3, 4, 5.*

- ☐ You can now observe the waveform and verify the functionality of the design.
- ☐ 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



### *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.



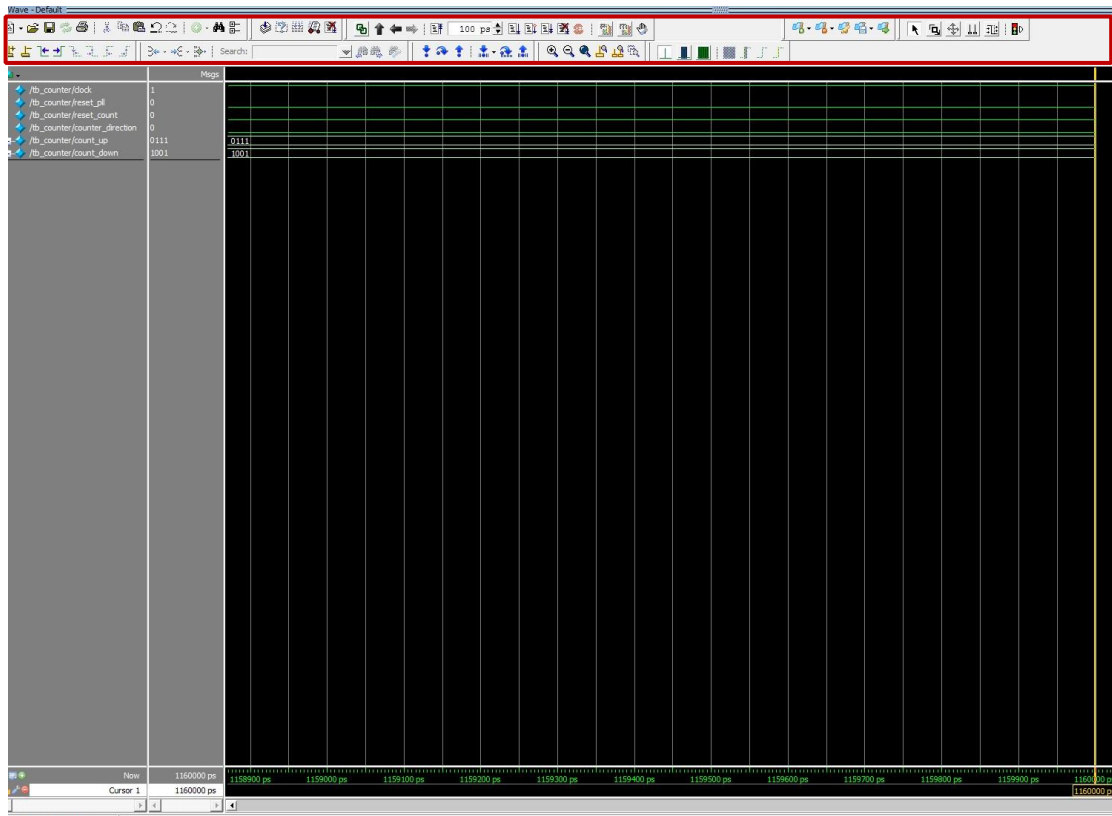



FIGURE 20 WAVEFORM WINDOW MODELIM


### 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.

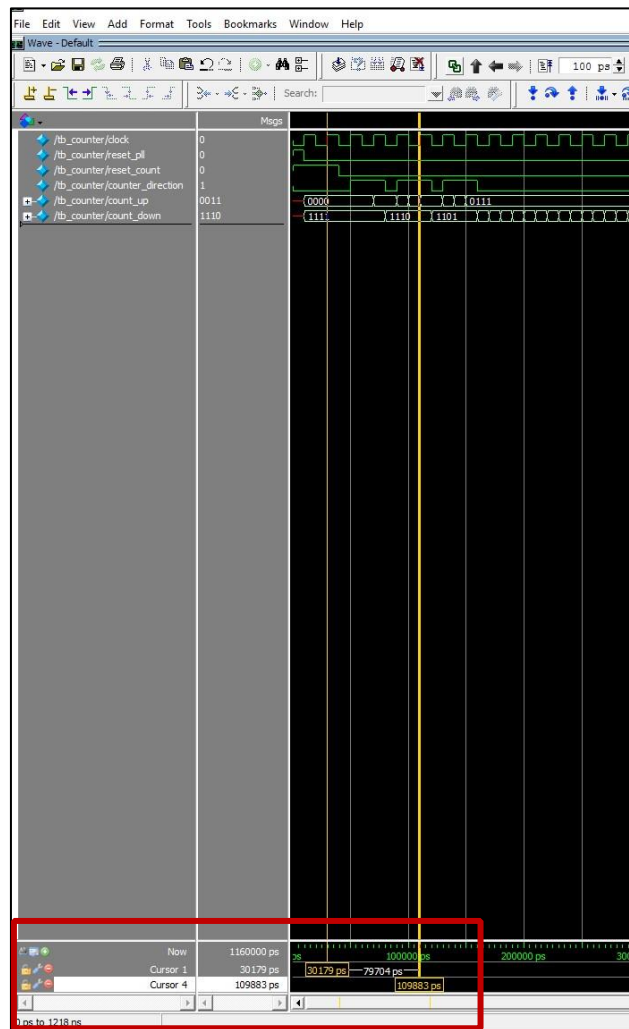
#### a. Zooming the waveform display:


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

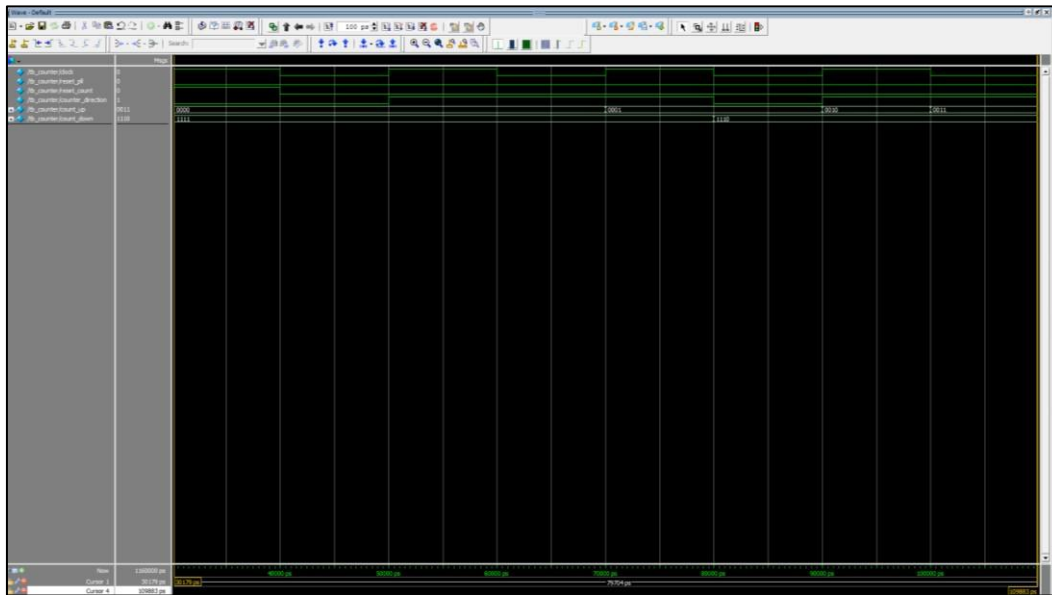
- ☐ If you want to completely zoom-out of the waveform. Click on .

- ☐ 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.



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



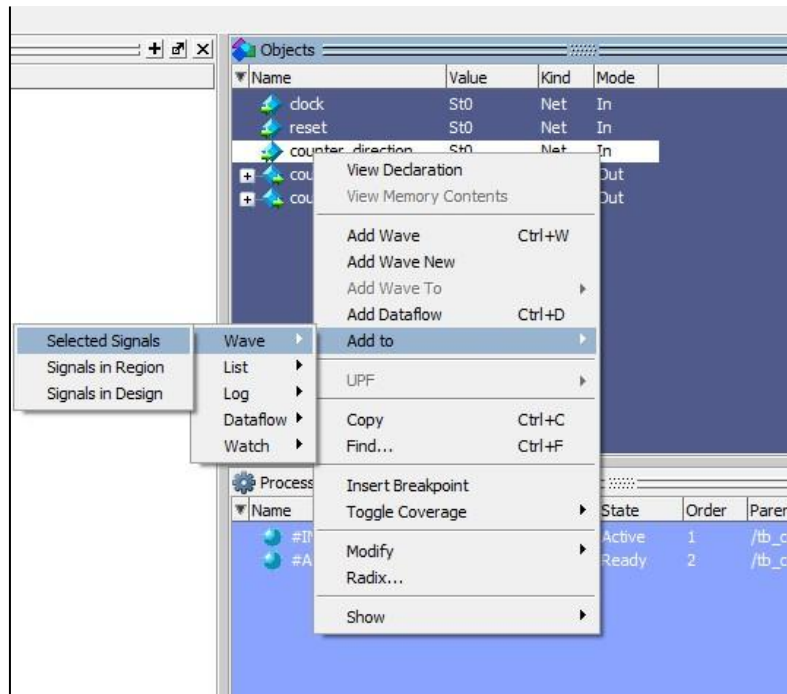
## Adding objects to Waveforms:

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

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

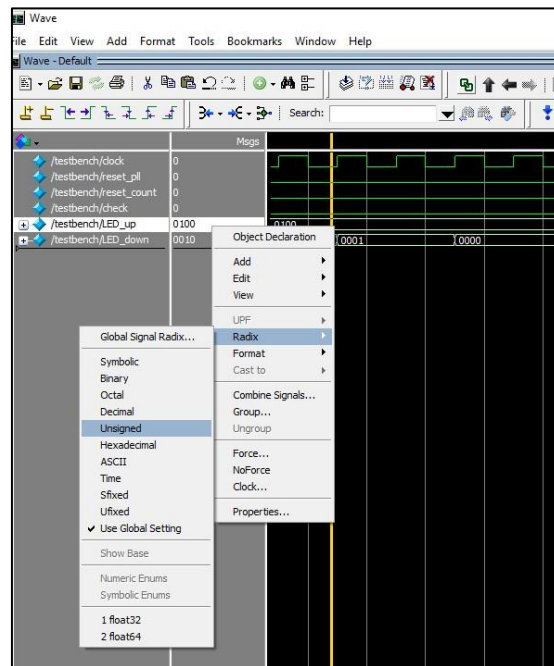
Instance	Design unit	Design unit type	Top Category	Visibility	Total cover
tb_counter	tb_counter	Module	DU Instance	+acc=...	
DUT	top_counter	Module	DU Instance	+acc=...	
P1	P1	Module	DU Instance	+acc=...	
counter_1	counter	Module	DU Instance	+acc=...	
#ALWA...	counter	Process	-	+acc=...	
#INITIAL#16	tb_counter	Process	-	+acc=...	
#ALWAYS#37	tb_counter	Process	-	+acc=...	
vsm_capacity#		Capacity	Statistics	+acc=...	

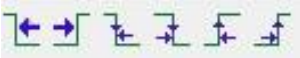
- In the objects pane, select the signal you want to view in the waveform. Right-click the selected signal. Select Add to Wave → Selected Signals as shown in the figure below.



### Changing the radix of signals:

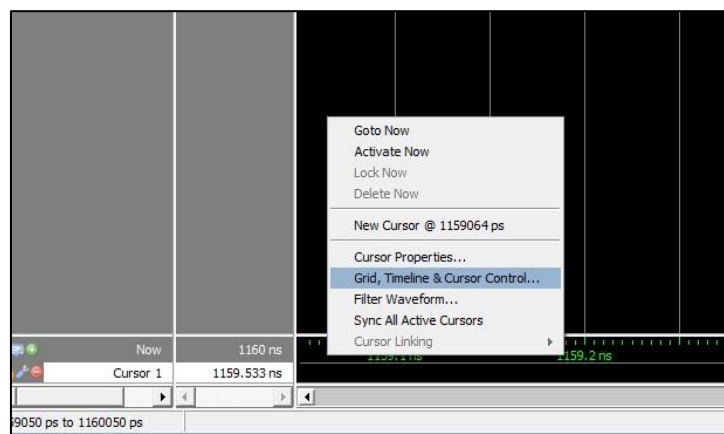
- You can select each signal in the waveform window and right click it to change its radix type as shown in the figure below. The default is binary, and this allows you to change to decimal or hexadecimal.



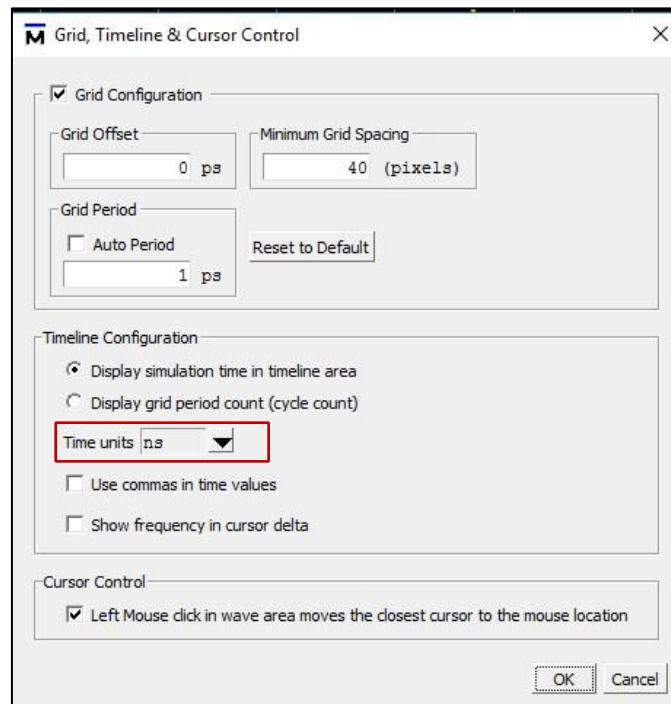
- ☐ If you want to get to the next rising or falling edge of a particular signal. Select that signal in the waveform and click on  in the toolbar for respective actions.

#### Timescale in waveform:

- ☐ 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..



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



### **Saving the Waveform:**

- ☐ If you want to save this particular set of signals in the waveform. Go to File → Save Format in the wave window.
- ☐ Name it “wave\_1.do” and click Save.
- ☐ 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.*

### **Running design independently on Modelsim:**

- ☐ When you Run RTL Simulation for the first time. In the Project folder, there is a folder called simulation is generated.
- ☐ 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.
- ☐ You can run this do file in Modelsim independently. Open ModelSim using the start menu. Once the software loads.

- In the transcript window, go to the project folder location. Once in the project folder, type `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.

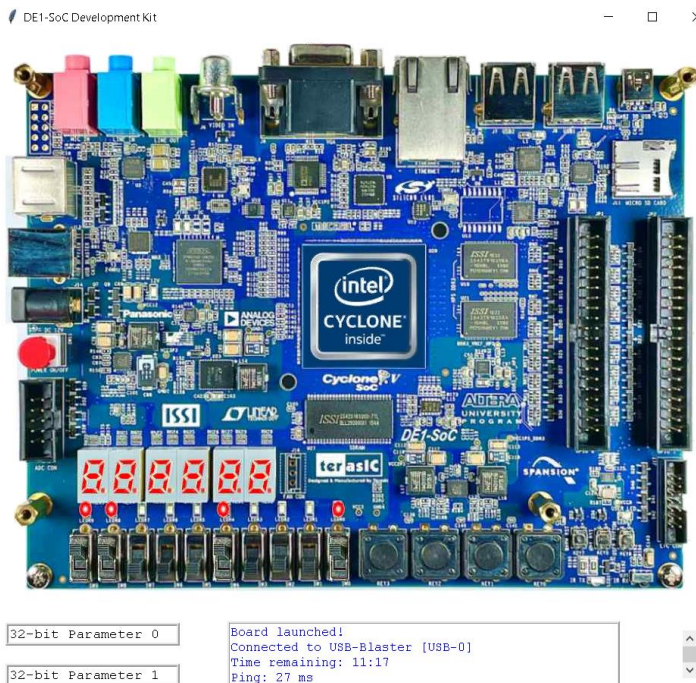
# In-System Sources and Probes

## 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 up to 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.

In this section of the lab, you will access the remote hardware. The FPGA development kit you will use is either a [Cyclone V GX Starter Kit](#), or a development kit called the [DE1-SoC](#). Some of you in the class will end up connecting to a PC that hosts a Cyclone V GX Starter Kit and others will be connected to a PC hosting the DE1-SoC kit. The development kits are attached to Windows PCs in a lab at Intel's San Jose, CA campus. In either case of board use, you will view your operational lab on a DE1-SOC board image. Operating the board switches and viewing LEDs/7-segments looks like a video game version of the actual development kit. You will be able to see LEDs change state and move switches to run your experiments. Even if you have the Cyclone V GX Starter Kit (which you should have identified from previous steps), the GUI will look like the image below.

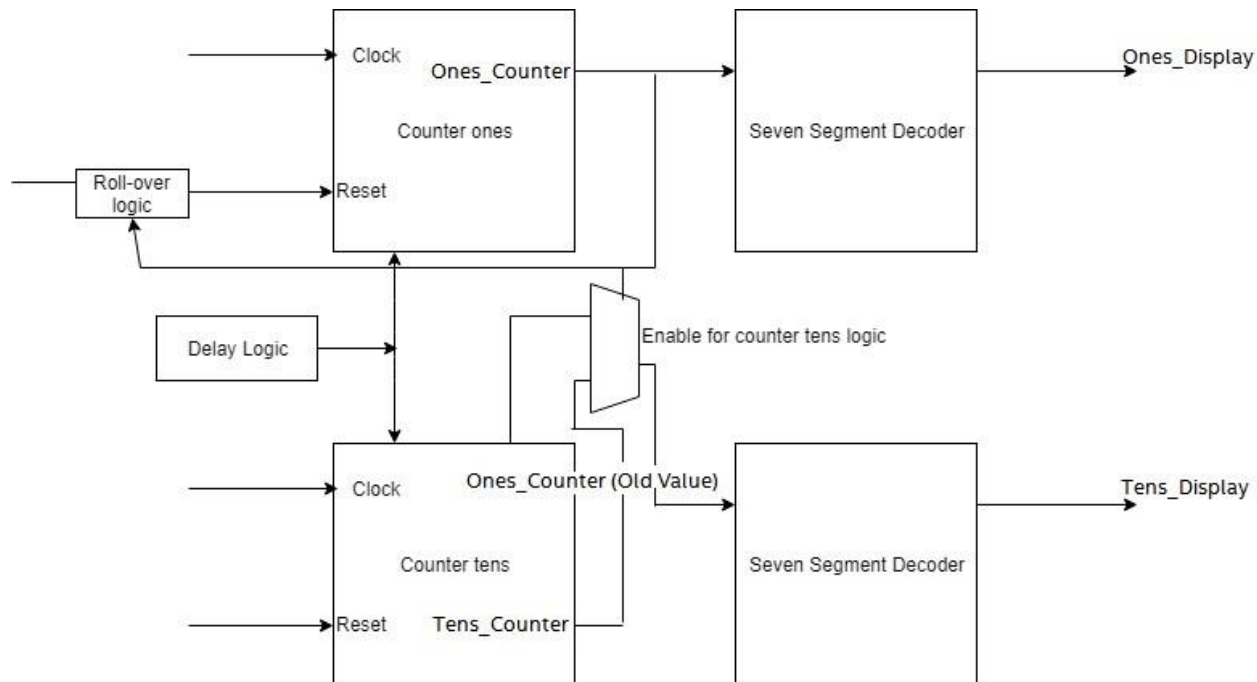




## Lab Exercise 2

### Objective:

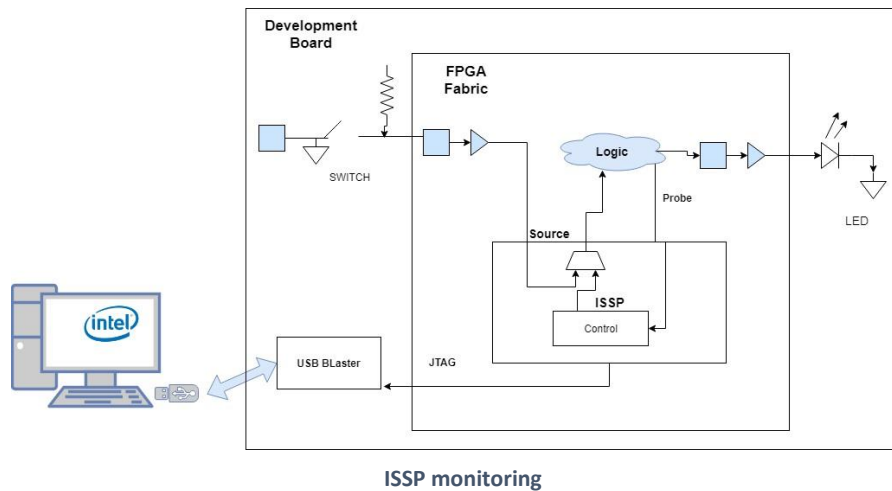
*With the help of this lab the student will be able to understand how to use In-system sources and probes in debugging. The student will learn how to instantiate In-System Sources and Probes (ISSP) and observing the signal behavior of the design through the ISSP Editor.*



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 above.

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.



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

- ☐ In the downloaded Workshop\_DebuggingTools folder. Navigate to the Lab\_2 Folder. Open Example\_ISSP\_SignalTap\_DE1SoC.qar or Example\_ISSP\_SignalTap\_CVGXStarter.qar in Quartus Standard version 18.0 to restore the project for your identified development kit.
- ☐ 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.
- ☐ 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.
- ☐ You can also find it by going to Basic Functions→Simulation; Debugging and Verification→Debug and Performance→Altera In-System Sources and Probes
- ☐ If the IP Catalog is not present, select it from the Tools menu

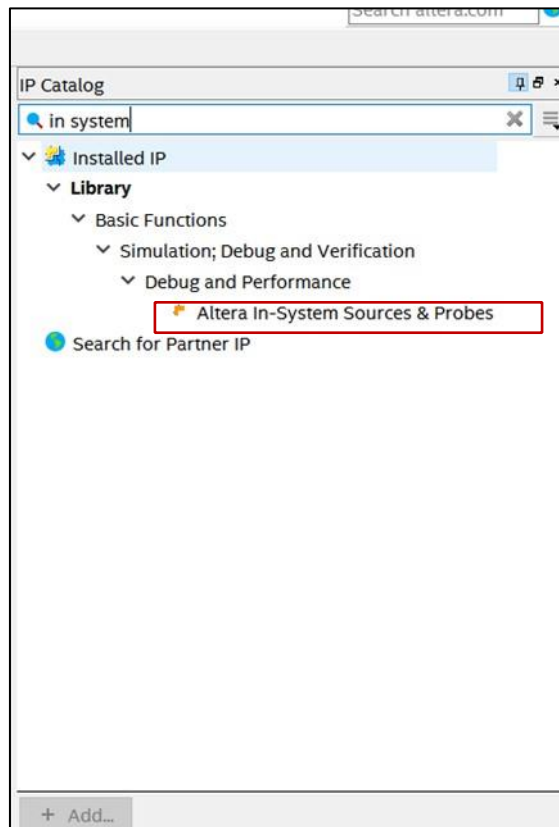
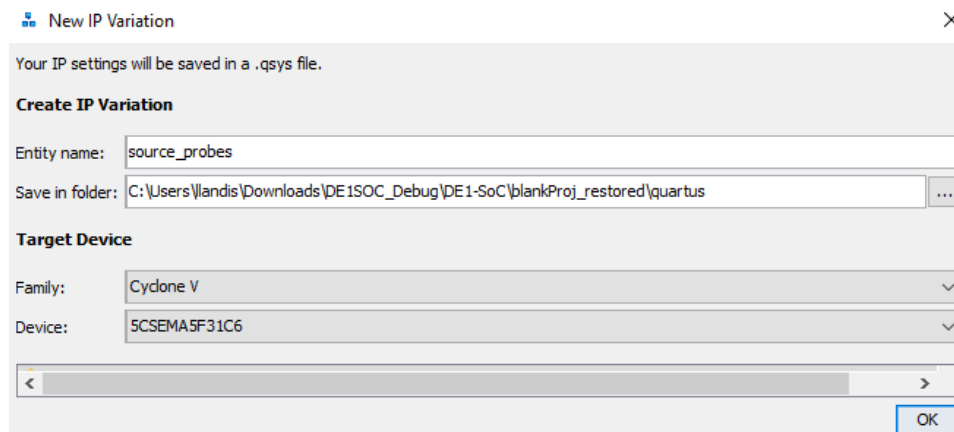
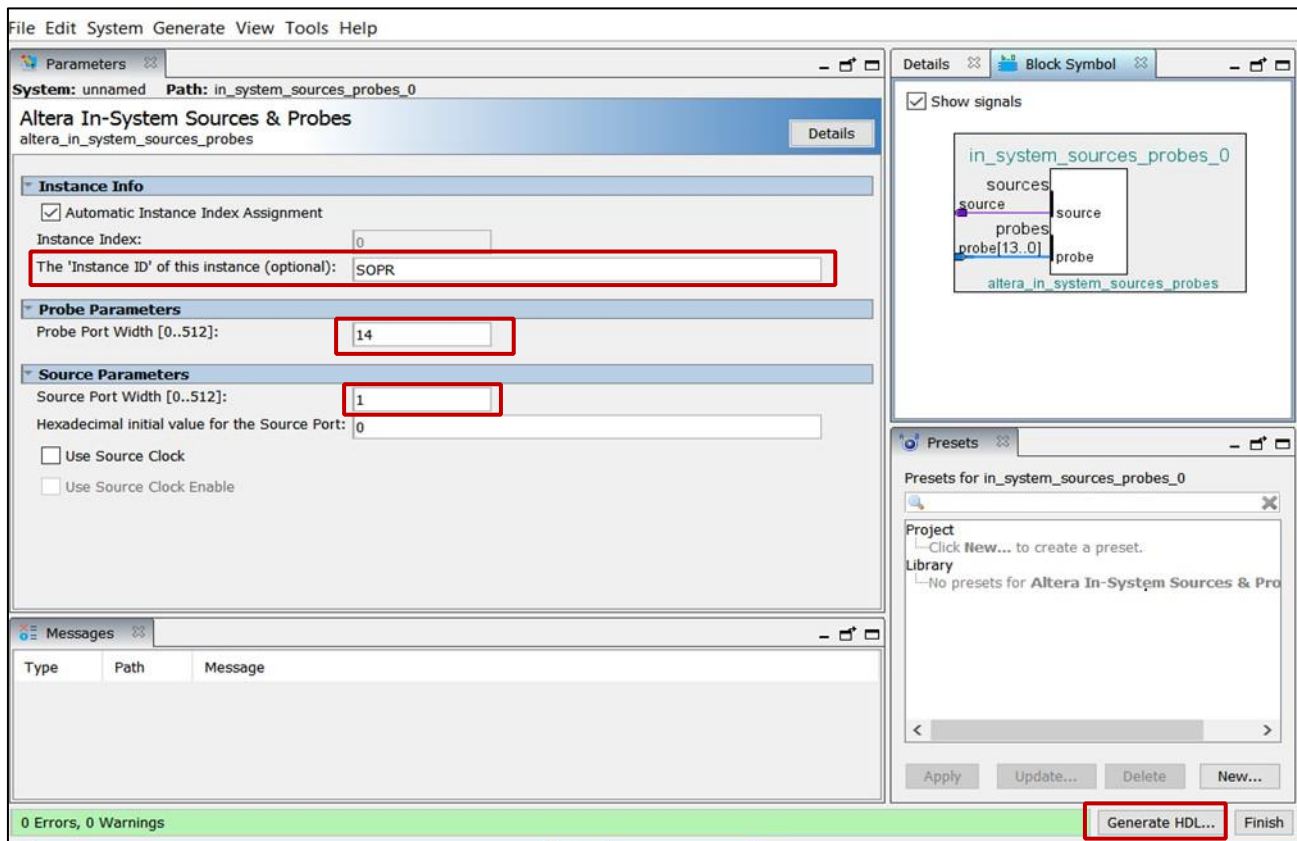


FIGURE 30 IP CATALOG

- 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 **Cyclone V** and device matches as well. Click **OK**.

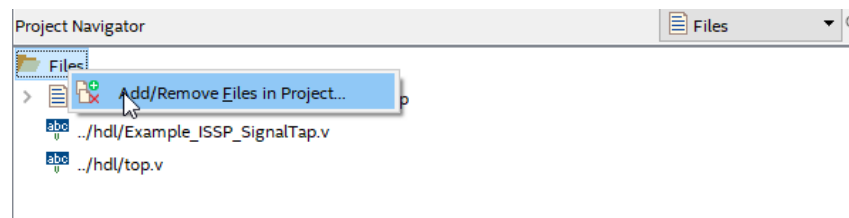


- In the IP Parameter Editor, make sure Automatic Instance Index Assignment is turned on. Set Instance ID to **SOPR**. Set Probe Port Width 14 and the Source Port Width to 1.



#### 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.



- In the **top.v** file, the connection for the **source\_probes** 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
```

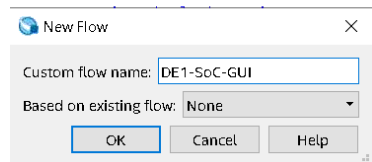
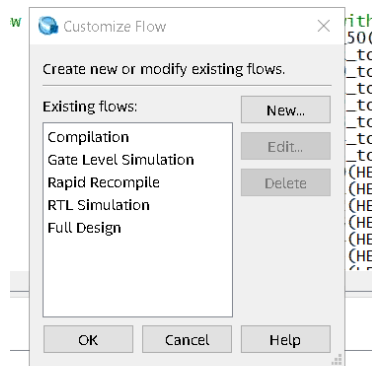
- Compile the design using the play button: 

### Using In-system Sources and Probe Editor

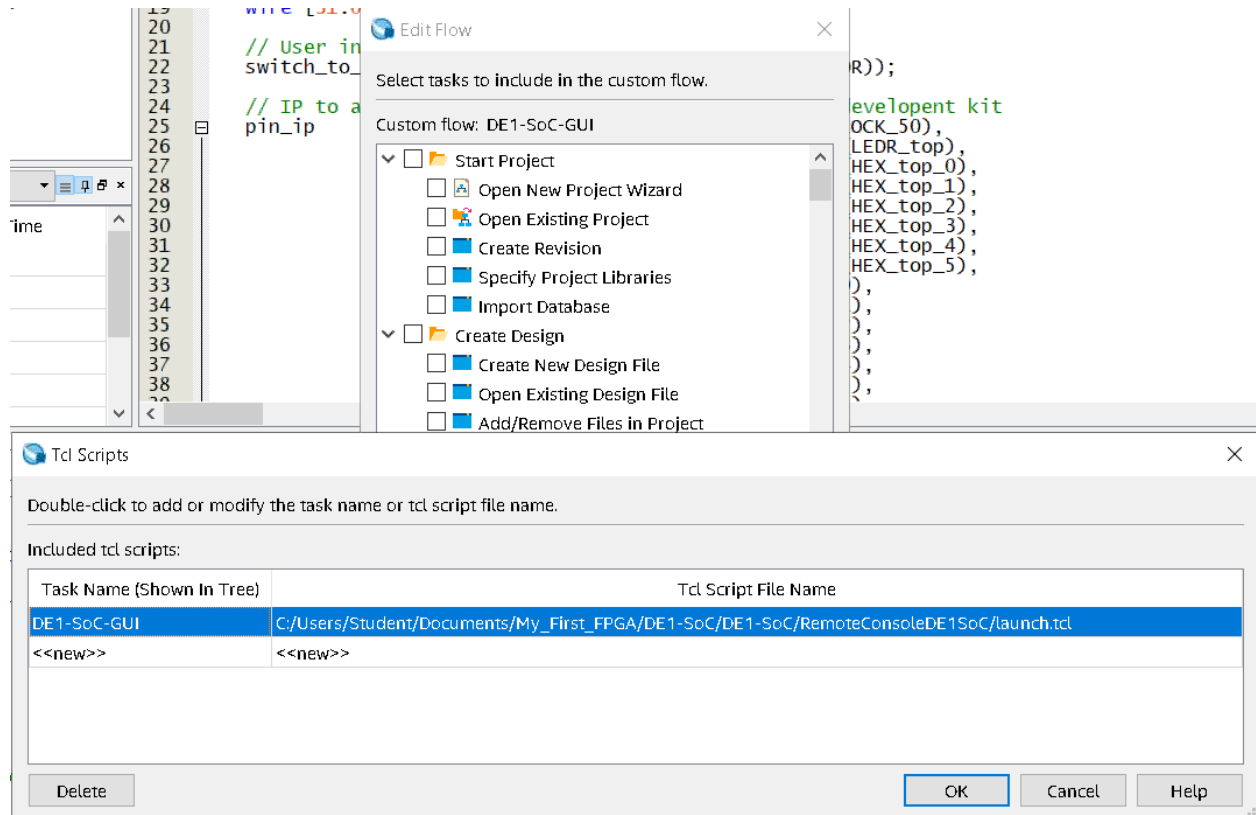
- The first step is to connect to the actual hardware through a remote connection. Under Tasks change Compilation to DE1SoC (or something similarly named as this a custom plug-in that previous users of this lab computer have setup. If this task does not exist, follow these steps. Else continue with the step on programming remote hardware.
- Navigate to this GitHub site on the Webex training web browser . Open the link <https://github.com/intel/FPGA-Devcloud/releases/tag/DE1-SoC> in the chrome browser that is part of your webex training session, not your local PC.
- Click on the zip file link, followed by download and save it. Save the DE1-SoC.zip file to a directory that does not have spaces in the path name! IMPORTANT: No spaces such as C:\Users\Student\Documents\RemoteConsole.
- Right click the zip file and extract the contents.
- In the Tasks window, there is an icon with 3 horizontal bars. Click on that and customize.



- Click on New.



- ☐ Name the Custom flow DE1-SoC-GUI. Keep existing flow as None.
- ☐ Select "TCL Scripts..."
- ☐ Enter DE1-SoC-GUI as the Task Name and the path to the launch.tcl file in the downloaded package. Click OK through windows and your Task should show up in the Task Window.
- ☐



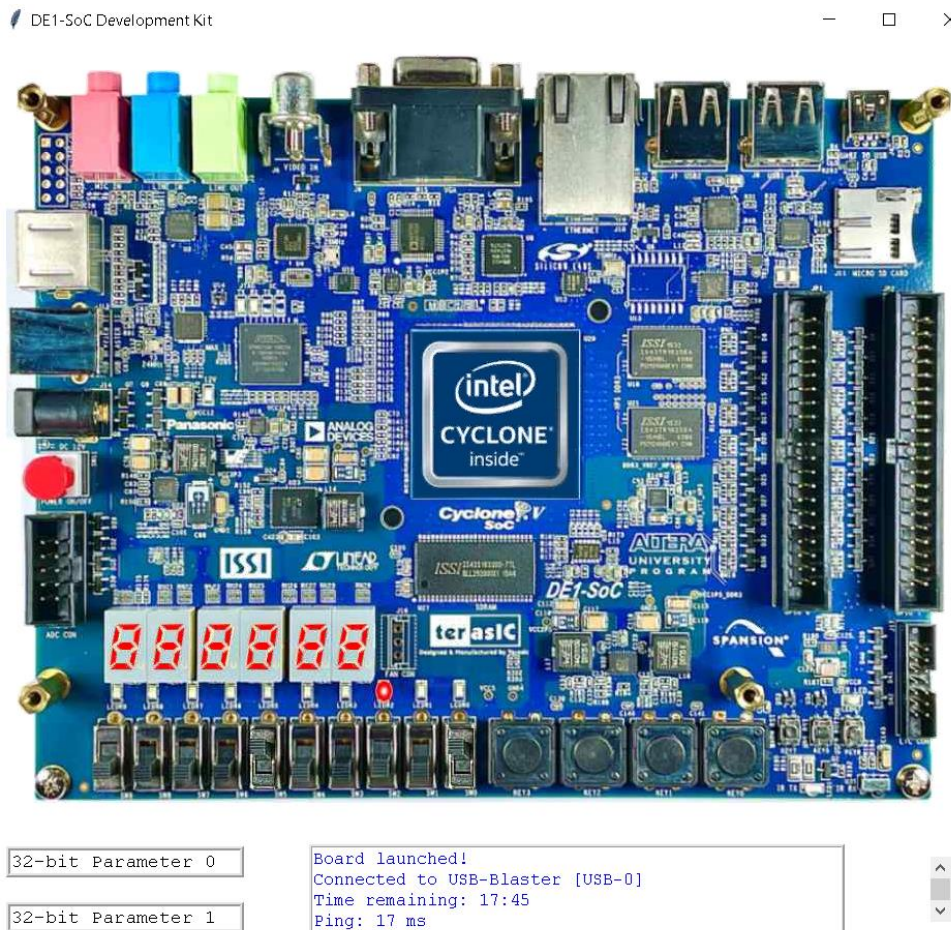
- ☐ This next step is a bit of a hack allow us to use the DE1-SoC GUI only for the Cyclone V GX Starter kit. Once we have a native Cyclone V GX Starter Kit, this step will go away. Navigate to this folder: C:\Users\Student\Documents\My\_First\_FPGA\DE1-SoC\RemoteConsoleDE1SoC using File explorer.

- ☐ The rest of these steps in Section only apply if you are using the Cyclone V GX Starter Kit:
- ☐ Right click on setup.tcl and select your favorite editor listed.
- ☐ Replace lines 6 through 8 with the lines shown below.

```
set cfg_part "5CGXFC5C6F27C7"
set ign_part ""
set prog_regex "*Blaster"
```

## Programming Remote Hardware

- ☐ Right click on DE1-SoC-GUI and hit Start
- ☐ Click on "select .sof file" and navigate to output\_files/Remotelab.sof, select, and click link. Wait about 20 seconds for the GUI to pop up. If its hidden from view, highlight the feather icon at the bottom of your screen.



- ☐ The right most slide switch needs to be in the ON position and you should see hex displays increment from 0 to 99 and back to zero. Due to a bug in the code, it increments to 98 and back to zero.
- ☐ Tools → 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.

- Select Hardware USB-Blaster and SOPR instance in the Instance Manager.

In-System Sources and Probes Editor - C:/Users/Student/Downloads/Example\_ISSP\_SignalTap\_CVGXStarter\_restor...

File Edit View Processing Tools Window Help

Search altera.com

Instance Manager: Ready to acquire

Probe read interval  
Current interval: 9 samples per second  
☒ Automatic  
☐ Manual 1 s

Event log  
Maximum size: 8  
☒ Save data to event log  
Write source data: Continuously

JTAG Chain Configuration: JTAG ready


Hardware: USB-Blaster [USB-0] Setup...  
Device: @1: 5CGTFD5(C5|F5) Scan Chain  
File: ...

Index	Instance ID	Status	Sources: 1	Probes: 14	Name
0	SOPR	Unloading ...	1	14	SOPR

0 SOPR

Index	Type	Alias	Name	Data	-8	-7	-6	-5	-4	-3	-2	-1	0
[13..0]	probe		probe[13..0]	2340		6180			3236			2340	
S[0]	source		source[0..0]	-1					-1				

0% 00:00:00

- Click to start continuously  reading the signals that feed the display. Observe the probes displaying the values of the seven segment display LEDs.



- Turn off the SW[0] and place the design in reset state. 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.

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.

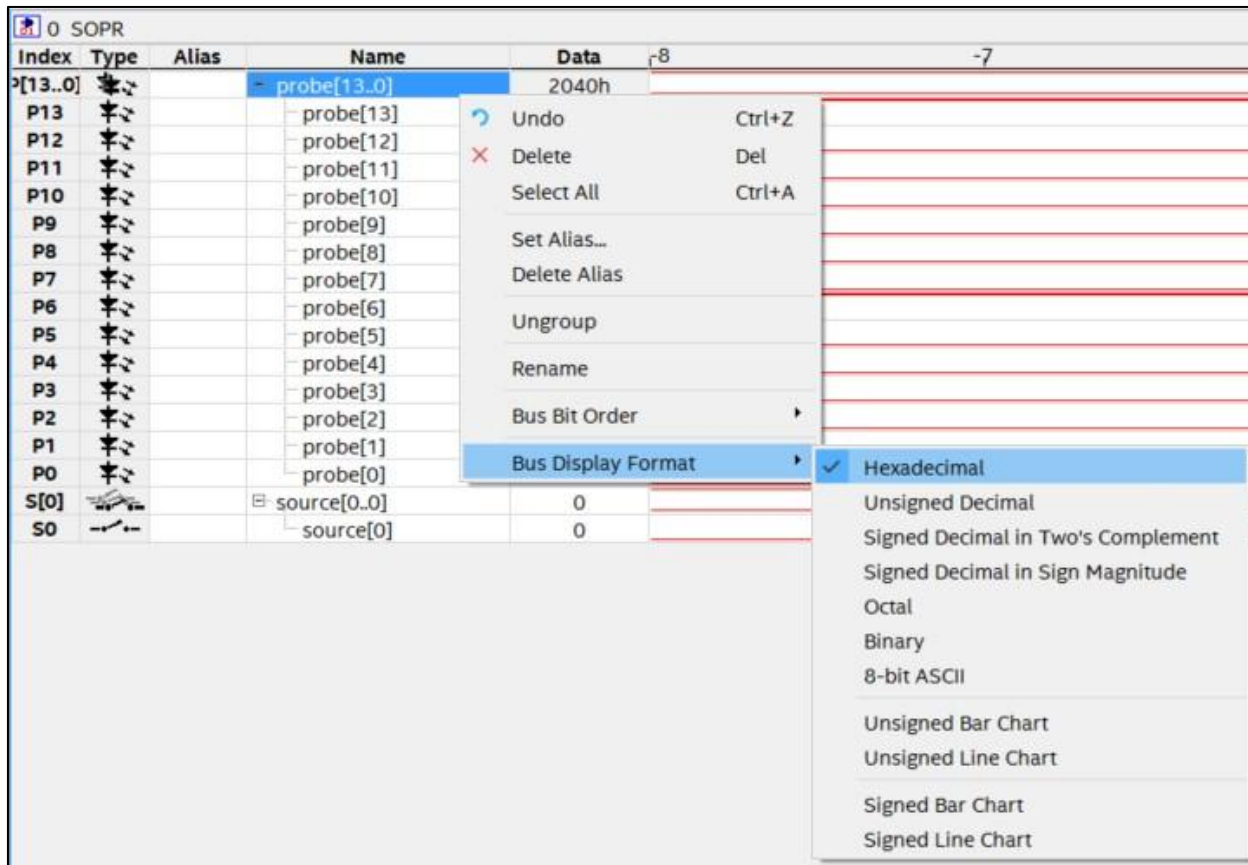


FIGURE 38 CHANGING THE BUS DISPLAY FORMAT

- If you want to view the signals in hexadecimal. You can **Right-Click** the **probe[13..0]**, go to **Bus Display Format**→**Hexadecimal**.
- Click **Stop** to stop monitoring data.
- You can hit reset timer on the remote console to give you an additional 20 minutes of interactive debugging.

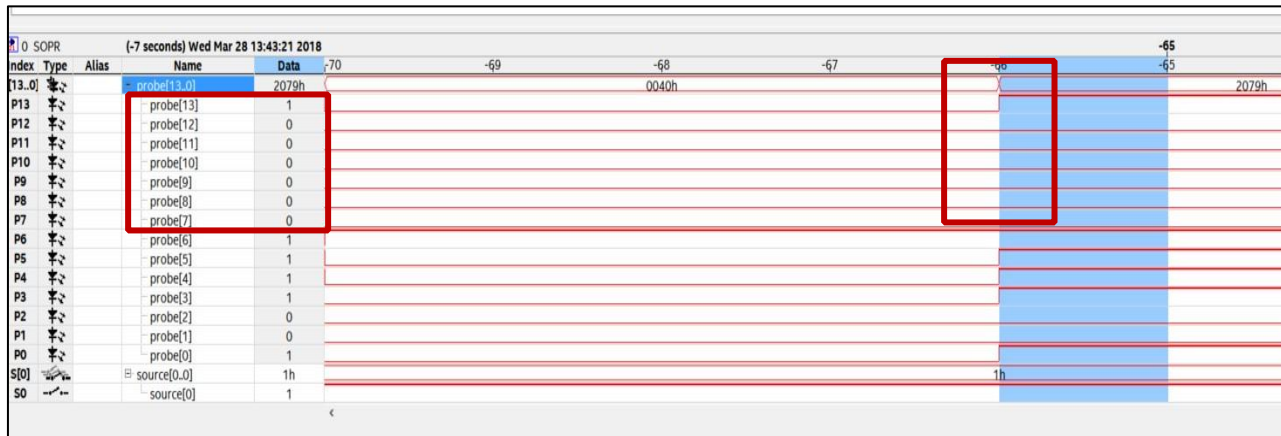


FIGURE 39 DEBUGGING THE DESIGN USING ISSP

**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.

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.

#### Create Signal Tap Logic Analyzer instances and compile with design

- ☐ Choose Signal Tap Logic Analyzer from the Quartus Tools menu to create a new “.stp” file.
- ☐ 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
- ☐ Double-click to add nodes to the design. Refer figure below.

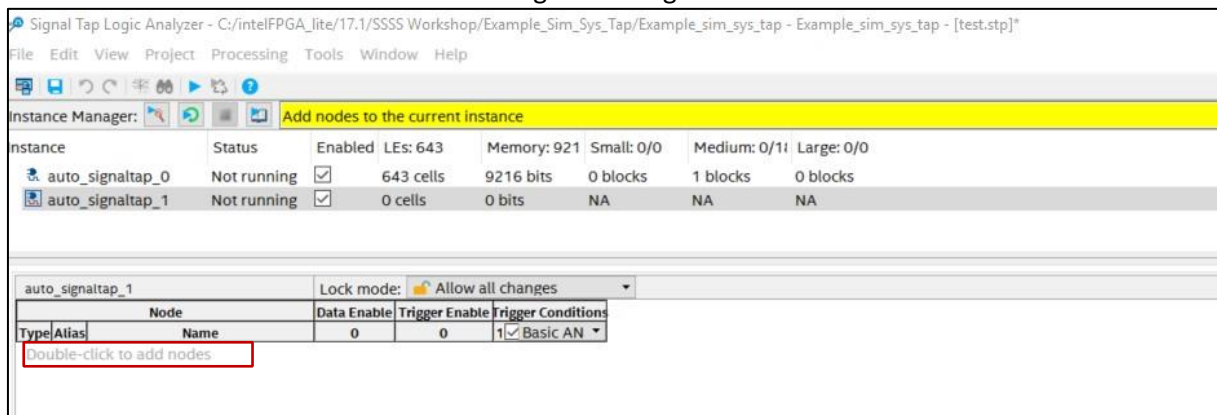

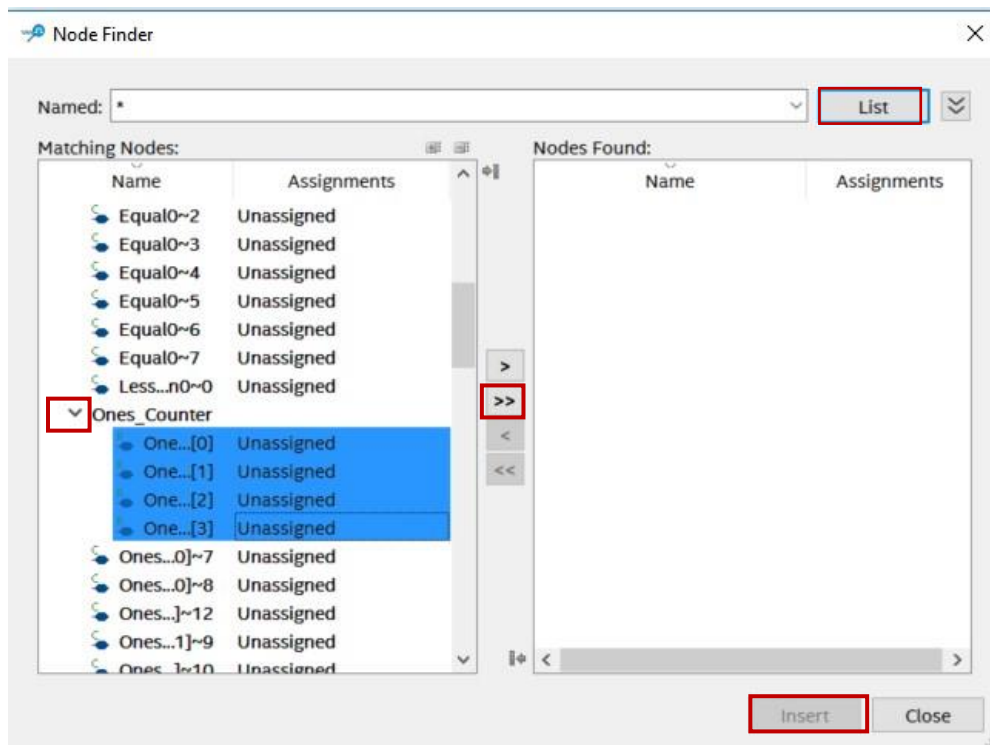


FIGURE 41 SIGNAL TAP INSTANCE

- ☐ Find and select nodes for analysis using the following sub-steps:
- ☐ In the Node Finder, click List. It will list all the components in the design.
- ☐ 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.
- ☐ Select Counter: counter\_0 and expand the view.

- 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. Also, add SW [0] from the SW instance.



- 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.
- Now, you can rearrange your signals by clicking them and dragging them to the position you want it at. Once you have the order of the signals as shown 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)*

trigger: 2018/02/07 11:13:19 #0				Lock mode:  Allow all	
		Node	Data Enable	Trigger Enable	
Type	Alias	Name	9	9	
		Counter:counter_0 Ones_Counter[1]	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
		Counter:counter_0 Ones_Counter[2]	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
		Counter:counter_0 Ones_Counter[3]	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
		Counter:counter_0 Ones_Counter[0]	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
		Counter:counter_0 Tens_Counter[0]	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
		Counter:counter_0 Tens_Counter[1]	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
		Counter:counter_0 Tens_Counter[2]	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
		Counter:counter_0 Tens_Counter[3]	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
		SW[0]	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	

Data
 Setup

- After the nodes are added to the setup, you can group the signals as well as shown in the figure below. For example, all Ones\_Counter as one group and Tens\_Counter as another group. To group the signals, Ctrl + Click all the signals you want to group and Right-Click. Select Group.

trigger: 2018/02/07 11:13:19 #0			Lock mode:  Allow all changes		Trigger Conditions	
Type	Alias	Node	Data Enable	Trigger Enable	1 Basic AN	2 Basic A
		Counter:counter_0 Ones_Counter[3:0]	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	8h	9h
		Counter:counter_0 Ones_Counter[3]	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	T	T
		Counter:counter_0 Ones_Counter[2]	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	0	0
		Counter:counter_0 Ones_Counter[1]	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	0	0
		Counter:counter_0 Ones_Counter[0]	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	0	T
		Counter:counter_0 Tens_Counter[3]	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	T	T
		Counter:counter_0 Tens_Counter[2]	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	0	0
		Counter:counter_0 Tens_Counter[1]	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	0	0
		Counter:counter_0 Tens_Counter[0]	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	0	T
		SW[0]	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	T	

Delete
 Select All
 Find...
 Find Next
 Find Bus Value...
 Find Next Bus Value
 Find Previous Bus Value
 Add Nodes with Plug-In
 Add Nodes...
 Plug-In Options...
 Add State Machine Nodes...
 Recreate State Machine Mnemonics...
 Locate Node
 **Group**
 Ungroup
 Ungroup Nested Groups
 Rename
 Mnemonic Table Setup...
 Create Signal Tap List File
 Invert Signal
 Align Left
 Align Right
 Pre-Synthesis Node
 Post-Fit Node
 MSB on Top, LSB on Bottom
 LSB on Top, MSB on Bottom
 Bus Display Format

Data
 Setup

Hierarchy Display:

- Now, we will set up the trigger conditions for the nodes selected.

- 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 CLOCK\_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.
- 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.

Signal Configuration:

Clock: MAX10\_CLK1\_50

Data

Sample depth: 1 K RAM type: Auto

☐ Segmented: 2 512 sample segments

Nodes Allocated: ☒ Auto ☐ Manual:

Pipeline Factor: 0

Storage qualifier:

Type: ☒ Continuous

Input port: auto\_stp\_external\_storage\_qualifier

Nodes Allocated: ☒ Auto ☐ Manual:

☒ Record data discontinuities

☐ Disable storage qualifier

Trigger

Nodes Allocated: ☒ Auto ☐ Manual:

Trigger flow control: Sequential

Trigger position: ☒ Center trigger position

Trigger conditions: 1


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

- 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.

Node		Data Enable	Trigger Enable	Trigger Conditions
Type	Alias			
	Counter:counter_0 Ones_Counter[3..0]	9	9	1 Basic AN
	Counter:counter_0 Ones_Counter[3]	✓	✓	9h
	Counter:counter_0 Ones_Counter[2]	✓	✓	T
	Counter:counter_0 Ones_Counter[1]	✓	✓	0
	Counter:counter_0 Ones_Counter[0]	✓	✓	0
	Counter:counter_0 Tens_Counter[3..0]	✓	✓	T
	Counter:counter_0 Tens_Counter[3]	✓	✓	9h
	Counter:counter_0 Tens_Counter[2]	✓	✓	T
	Counter:counter_0 Tens_Counter[1]	✓	✓	0
	Counter:counter_0 Tens_Counter[0]	✓	✓	1
	SW[0]	✓	✓	T

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

- ☐ Save the Signal Tap instance.
- ☐ Compile Design by clicking on  in the toolbar. In the main Quartus window, you can observe the compilation tasks and time to completion.

## Configure the device with Signal Tap instance

**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.**

## Run analyzer

- ☐ Right click on DE1-SoC-GUI and hit Start as you did in the Sources and Probes section.
- ☐ Click on “select .sof file” and navigate to output\_files/RemoteLab.sof, select, and click link. Wait about 20 seconds for the GUI to pop up. If its hidden from view, highlight the feather icon at the bottom of your screen.
- ☐ In the Signal Tap Window, make sure the instance is selected.
- ☐ Click on  to start running the analyzer. You can observe “Waiting for Trigger” in the instance manager.
- ☐ Now, turn on the switch SW0 on the board to get the design out of reset (which is a trigger).

- ☐ Analyze the acquired data. This might take a while as you have to wait until 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.
- ☐ To zoom in data, just click on the waveform. To zoom out in the waveform, Shift Key + Click.



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:

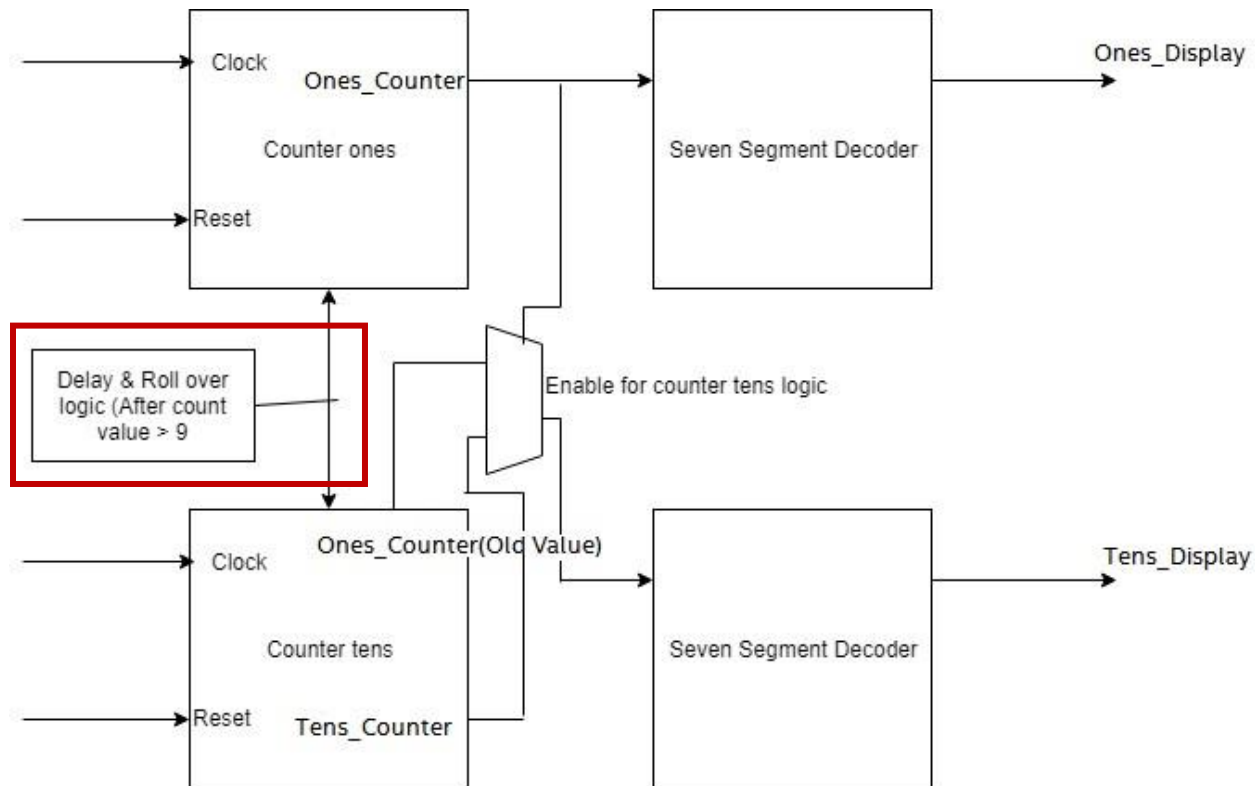


FIGURE 49 REFORMED DESIGN FOR 0-99 COUNTER

*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 InSystem 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.

Thanks for attending this workshop.

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
2.3	L. Landis	4/14/2021	Adapt to DE0-CIII board
2.4	L. Landis	4/22/2021	Adapt to DE1-SoC and CV GX Starter