MUH211 LAB INTRO
Kocaeli Üniversitesi Elektronik ve Haberleşme Mühendisliği

Introduction to Circuit Design in Efinity Software

PRELAB!
Read the entire lab, and **complete** the installation guide.

1.0 Overview

The Efinity® software provides a complete tool flow for designing with Efinix® products. This document describes how to install the software.
Efinity® latest version : Download

2.0 Hardware and Software Requirements

General Requirements

- Efinity full release: 64-bit, x86 instruction set architecture
- Windows Standalone Programmer.
  - Windows 10: 64-bit x86 instruction set architecture.
  - Windows 11: 64-bit x86 instruction set architecture.
- Computer with a 64-bit operating system.
  - A 64-bit Windows system is required for the Efinity standalone programmer.
  - A 64-bit Windows system is required for using the security tools in the Efinity standalone programmer.
- Your preferred text editor such as Notepad, gVim, Visual Studio
- Machine memory requirements (when running Efinity design compilations):

Table 1: Machine Memory Requirements

Product	Model	Memory
Trion	T4, T8, T13, T20, T35	8 GB
	T55, T85, T120	16 GB
Titanium	Ti35, Ti60	8 GB
	Ti90, Ti120, Ti180	16 GB
	Ti165, Ti240, Ti375	32 GB

Windows Requirements

- **Windows 10 or later, 64 bit operating system**
- **Microsoft Visual C++ 2019 x64 runtime library (or latest version) redistributable**
- **Zadig software to install USB drivers**

- **Java 64-bit runtime environment; required for configuring the Sapphire RISC-V SoC and DMA Controller in the IP Manager; available from:**
    - **https://www.java.com/en/download/manual.jsp (Java 8)**
    - **https://developers.redhat.com/products/openjdk/download (OpenJDK 8 or 11)**
    - **http://jdk.java.net/16/ (OpenJDK 16)**

Installing GTKWave

GTKWave is an open-source tool that analyzes post-simulation dumpfiles and displays the results in a graphical interface. It includes a waveform viewer and RTL source code navigator. You can use GTKWave with the iVerilog simulator to analyze and debug your simulation model, or to view any VCD waveform.

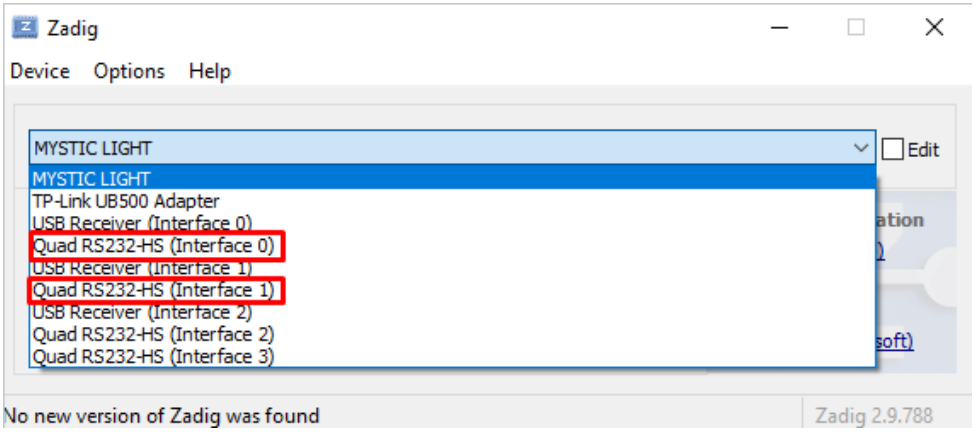
For Windows installation:

1. You can browse for the software files at gtkwave Windows files are situated lower down the page.
2. Unzip the downloaded file.
3. Optional:
   You may need to add the path to GTKWave ($GTKWave_folder$\bin\) to your System Variables path for the software to launch correctly.
4. Run the program by executing **gtkwave.exe** in the <install dir>/bin directory.

Installing the Windows USB Driver

On Windows, you use software from Zadig to install drivers. Download the Zadig software(version 2.7 or later)(You do not need to install it; simply run the downloaded executable.)

1. Connect the board to your computer with the appropriate cable and power it up.
2. Run the Zadig software.
3. Choose **Options > List All Devices.**
4. Repeat the following steps for each interface. The interface names end with (Interface N), where N is the channel number.
    i. For Kuantek TriPi board, you should selected drivers:



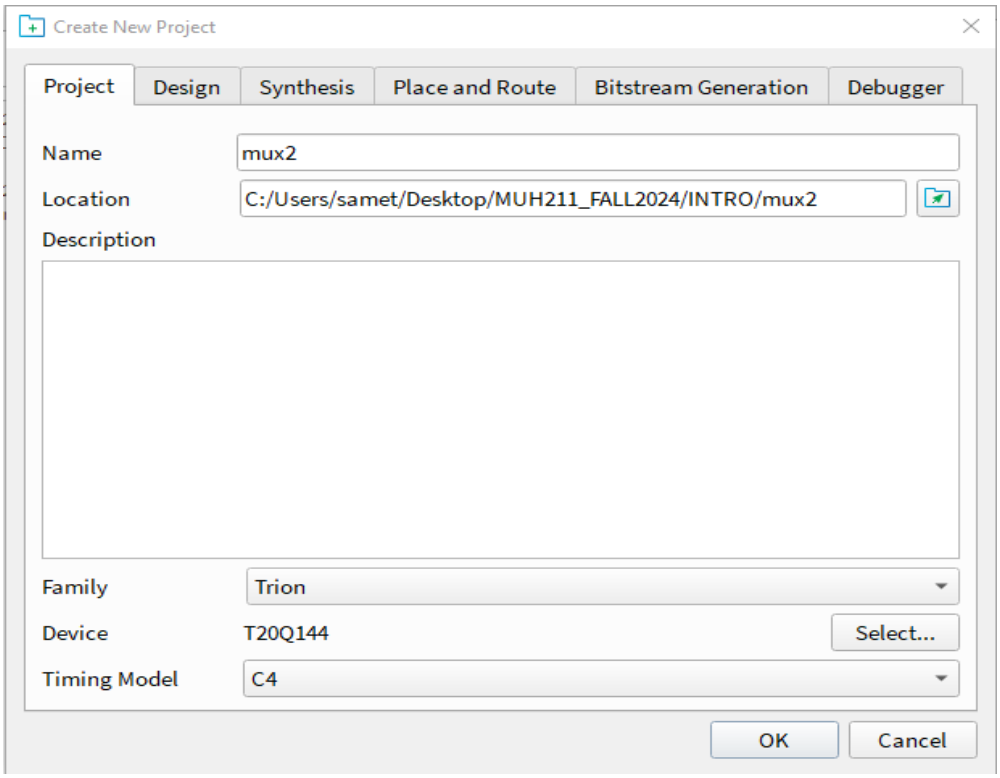
ii. Select **libusb-win32** in the Driver drop-down list.
iii. Click **Replace Driver**.

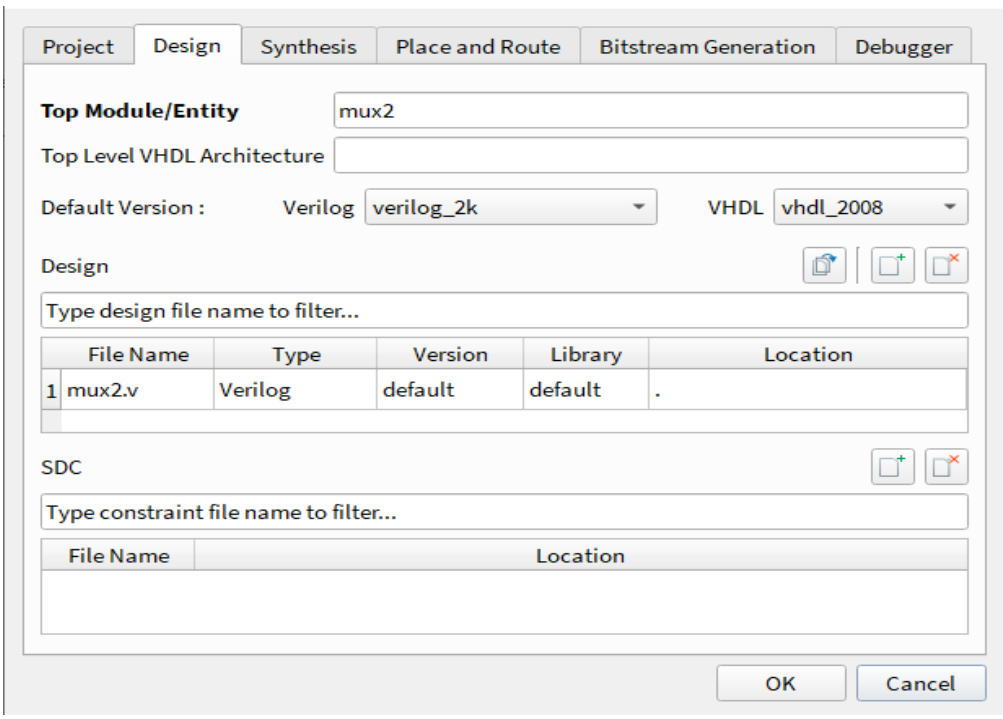
5. Close the Zadig software.

3.0 Create Your Project

In this step, you create a new project based on the mux2 example design using the Project and Design tabs in the Project Editor.

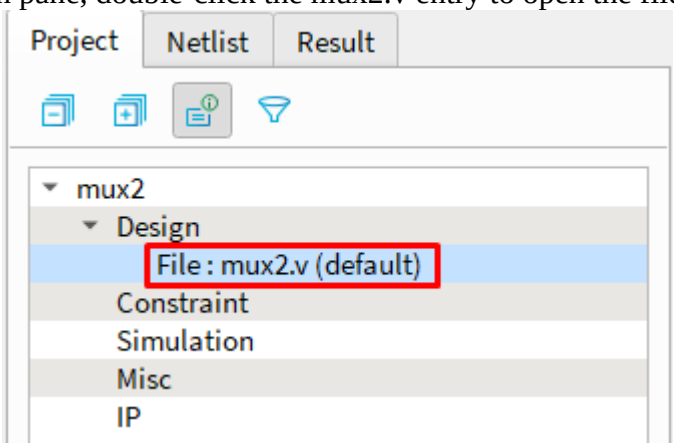
1. Open the Efinity GUI.
2. Create a new project (**File** > **Create project** or click the new project button). The **Project Editor** opens to the **Project** tab.
3. Type *mux2* in the **Name** box.
4. In the **Location** box, click the button Choose a directory to store project data.
5. Select the MUH211_FALL2024/INTRO/mux2 directory.
6. Choose Trion® as the family.
7. Choose T20Q144 as the device. Keep the default timing model selection.
8. Type mux2 in the Top Module/Entity box.
9. Next to the **Design** box, click the button **Add design** files.
10. Browse to the **Introduction to Verilog zyLabs** directory.
11. Select **mux2.v** file. **Tick** the **Copy the Project** and click **Open**. Click **OK** to close the **Project Editor**.
12. Project is created.


Create New Project
Project
Design
Synthesis
Place and Route
Bitstream Generation
Debugger
Name
mux2
Location
C:/Users/samet/Desktop/MUH211_FALL2024/INTRO/mux2
Description
Family
Trion
Device
T20Q144
Select...
Timing Model
C4
OK
Cancel

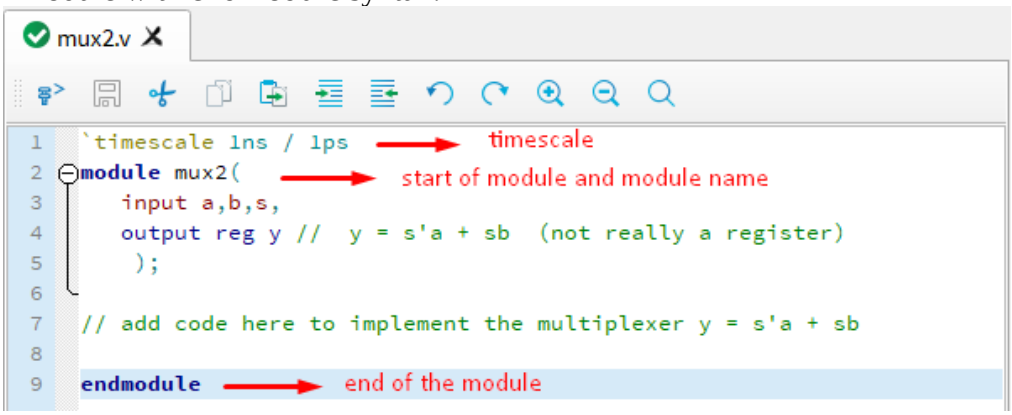

Project
Design
Synthesis
Place and Route
Bitstream Generation
Debugger
Top Module/Entity
mux2
Top Level VHDL Architecture
Default Version :
Verilog
verilog_2k
VHDL
vhdl_2008
Design
Type design file name to filter...
File Name
Type
Version
Library
Location
1
mux2.v
Verilog
default
default
.
SDC
Type constraint file name to filter...
File Name
Location
OK
Cancel

4.0 Edit the Design File

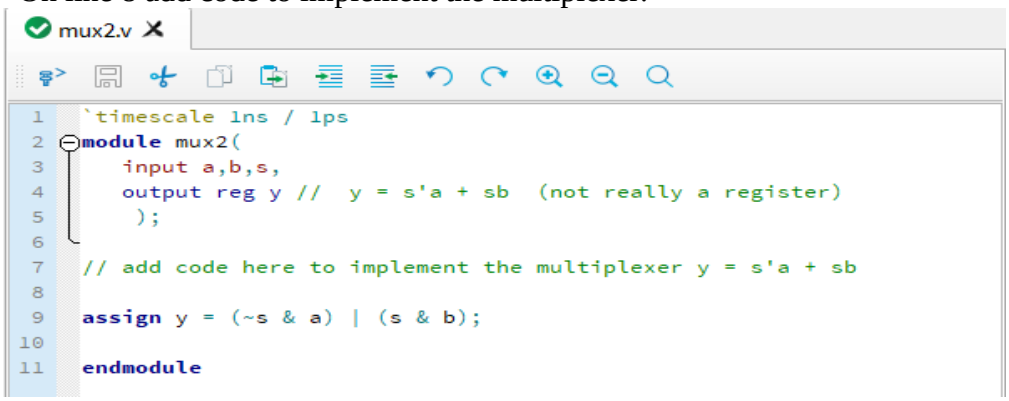
1. In the Design pane, double-click the mux2.v entry to open the file in text mode.



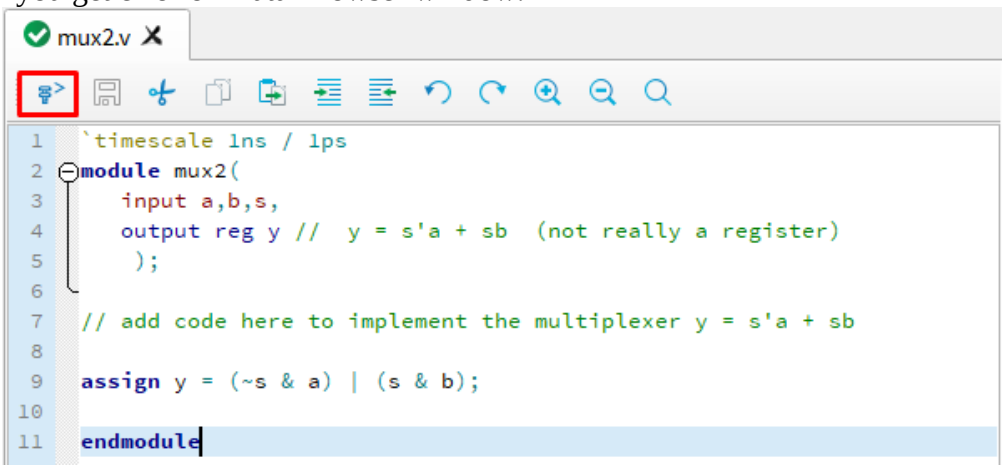
2. Notice in the Verilog code that the first line defines the timescale directive for the simulator. If there **is not exist** timescale you can add. (**`timescale 1ns / 1ps)**
3. Line 2 is defined as your **module** and **module name**. You should close the module with **endmodule** syntax.



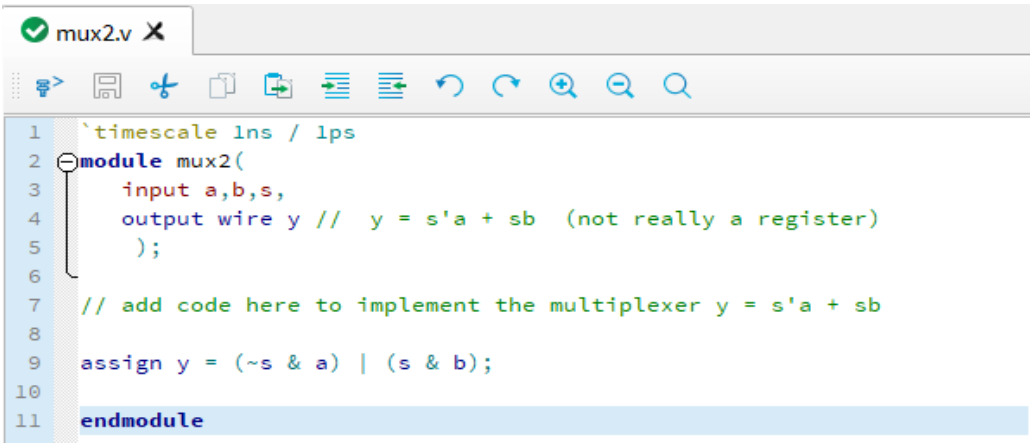
4. On line 8 add code to implement the multiplexer.



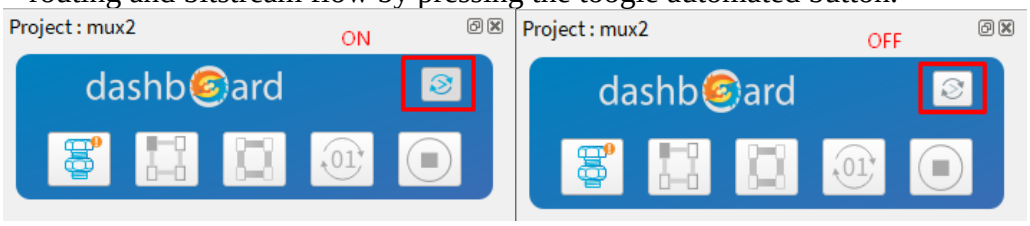
5. Save the file and to analyze the design click Analyze button. If there is a error, you get error on Data Browser window.



6. If you get "concurrent assignment to a non-net 'y' is not permitted" error, you should change type of output (y). Change reg with wire. Because wire and reg declarations in Verilog, is that a reg can be assigned to in a procedural block (a block beginning with always or initial), and a wire can be assigned in a continuous assignment (an assign statement) or as an output of an instantiated submodule.
7. The final state of the module:



8. Before performing the synthesis process, we stop the synthesis, placement, routing and bitstream flow by pressing the toogle automated button.

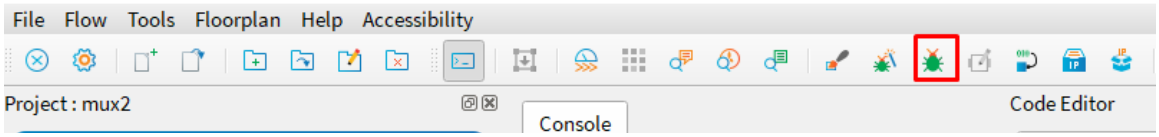


9. Click Synthesis under the dashboard.
10. When the process is completed. Click the Result tab and expand Synthesis. Open lab1.map.rpt file as we want to look at the synthesis output before progressing to the placement and routing the stage.

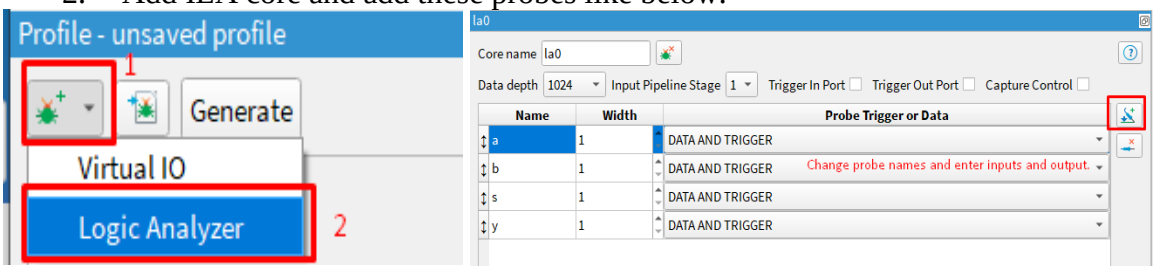
```
### ### Module Resource Usage Distribution Estimates (begin) ### ###

**Note: some resources maybe grouped under different hierarchy due to optimization and LUT mapping

Module              FFs        ADDs        LUTs       RAMs DSP/MULTs
-----------         ---        ----        ----       ---- ---------
mux2:mux2          0(0)        0(0)        1(1)       0(0)      0(0)

### ### Module Resource Usage Distribution Estimates (end) ### ###
```

5.0 Debug Settings

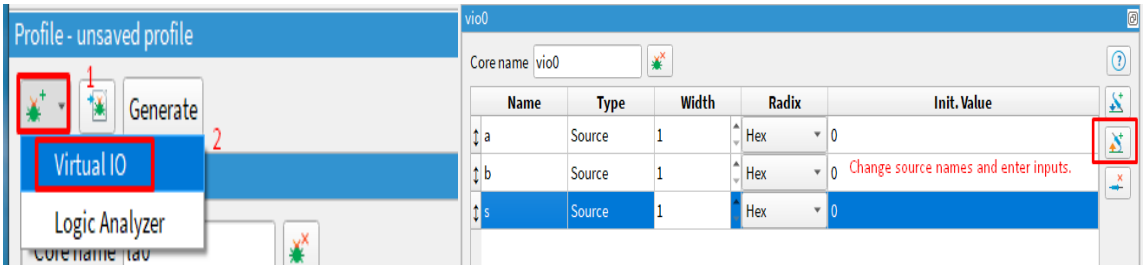
1. Open debugger.



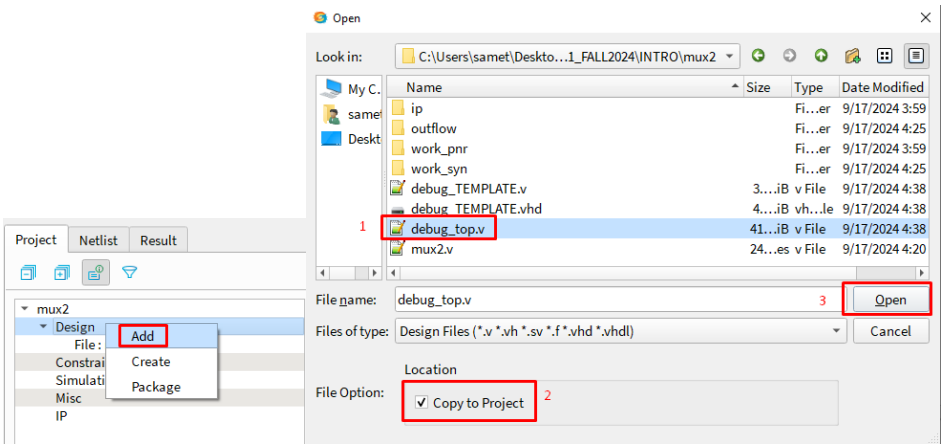
2. Add ILA core and add these probes like below.



3. Add VIO core and add these sources like below.



4. Then click generate and close window.
5. Add Debug Design File to project and edit mux2.v file.
6. Click right Design then add , select debug_top.v file from source folder. Select Copy to Project and click open.



7. Then go to project folder and open "debug_TEMPLATE.v" file and copy these lines, and paste the top module:

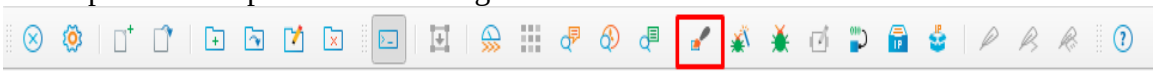
```
edb_top edb_top_inst (
    .bscan_CAPTURE      ( jtag_inst1_CAPTURE ),
    .bscan_DRCK         ( jtag_inst1_DRCK ),
    .bscan_RESET        ( jtag_inst1_RESET ),
    .bscan_RUNTEST      ( jtag_inst1_RUNTEST ),
    .bscan_SEL          ( jtag_inst1_SEL ),
    .bscan_SHIFT        ( jtag_inst1_SHIFT ),
    .bscan_TCK          ( jtag_inst1_TCK ),
    .bscan_TDI          ( jtag_inst1_TDI ),
    .bscan_TMS          ( jtag_inst1_TMS ),
    .bscan_UPDATE       ( jtag_inst1_UPDATE ),
    .bscan_TDO          ( jtag_inst1_TDO ),
    .la0_clk            ( $INSERT_YOUR_CLOCK_NAME ),
    .la0_a      ( la0_a ),
    .la0_b      ( la0_b ),
    .la0_s      ( la0_s ),
    .la0_y      ( la0_y ),
    .vio0_clk       ( $INSERT_YOUR_CLOCK_NAME ),
    .vio0_a         ( vio0_a ),
    .vio0_b         ( vio0_b ),
    .vio0_s         ( vio0_s )
);
```

8. Then edit the module like:

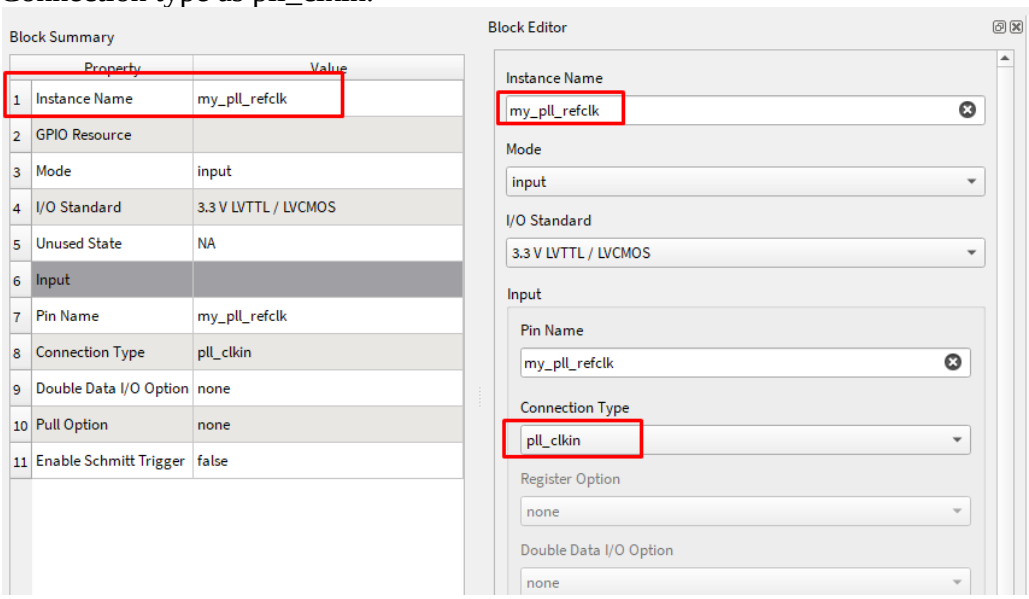
```
edb_top edb_top_inst (
    .bscan_CAPTURE      ( jtag_inst1_CAPTURE ),
    .bscan_DRCK         ( jtag_inst1_DRCK ),
    .bscan_RESET        ( jtag_inst1_RESET ),
    .bscan_RUNTEST      ( jtag_inst1_RUNTEST ),
    .bscan_SEL          ( jtag_inst1_SEL ),
    .bscan_SHIFT        ( jtag_inst1_SHIFT ),
    .bscan_TCK          ( jtag_inst1_TCK ),
    .bscan_TDI          ( jtag_inst1_TDI ),
    .bscan_TMS          ( jtag_inst1_TMS ),
    .bscan_UPDATE       ( jtag_inst1_UPDATE ),
    .bscan_TDO          ( jtag_inst1_TDO ),
    .la0_clk            ( $INSERT_YOUR_CLOCK_NAME ),
    .la0_a      ( la0_a ),
    .la0_b      ( la0_b ),
    .la0_s      ( la0_s ),
    .la0_y      ( la0_y ),
    .vio0_clk       ( $INSERT_YOUR_CLOCK_NAME ),
    .vio0_a         ( vio0_a ),
    .vio0_b         ( vio0_b ),
    .vio0_s         ( vio0_s )
);
```

```
edb_top edb_top_inst (
    .bscan_CAPTURE      ( jtag_inst1_CAPTURE ),
    .bscan_DRCK         ( jtag_inst1_DRCK ),
    .bscan_RESET        ( jtag_inst1_RESET ),
    .bscan_RUNTEST      ( jtag_inst1_RUNTEST ),
    .bscan_SEL          ( jtag_inst1_SEL ),
    .bscan_SHIFT        ( jtag_inst1_SHIFT ),
    .bscan_TCK          ( jtag_inst1_TCK ),
    .bscan_TDI          ( jtag_inst1_TDI ),
    .bscan_TMS          ( jtag_inst1_TMS ),
    .bscan_UPDATE       ( jtag_inst1_UPDATE ),
    .bscan_TDO          ( jtag_inst1_TDO ),
    .la0_clk            ( io_systemClk ),
    .la0_a      ( a ),
    .la0_b      ( b ),
    .la0_s      ( s ),
    .la0_y      ( y ),
    .vio0_clk       ( io_systemClk ),
    .vio0_a         ( a ),
    .vio0_b         ( b ),
    .vio0_s         ( s )
);
```

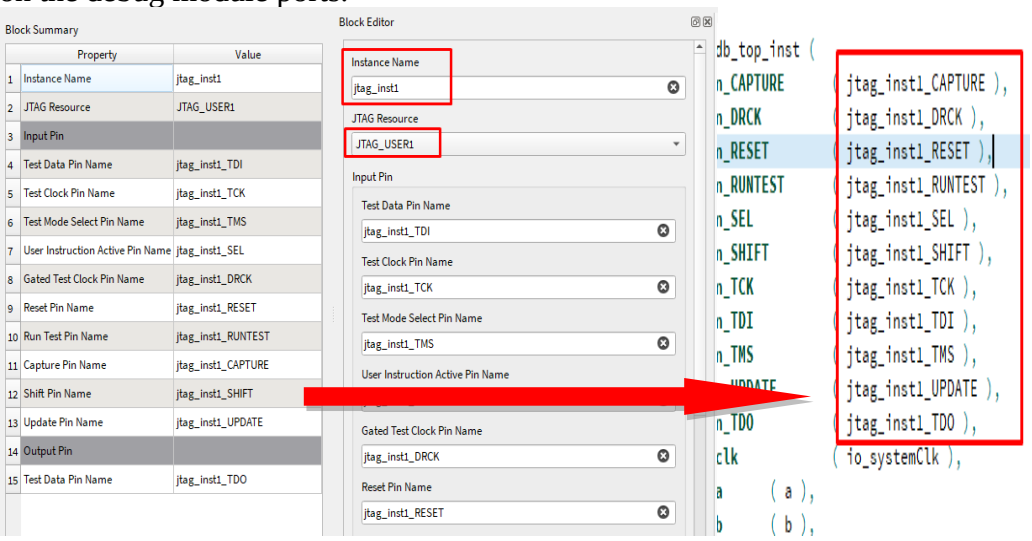
9. For use ILA and VIO with Debugger, we have to create a clock and JTAG User. JTAG User is used to connect to the Debugger. Then we have to add these ports to top module. Open Interface designer.



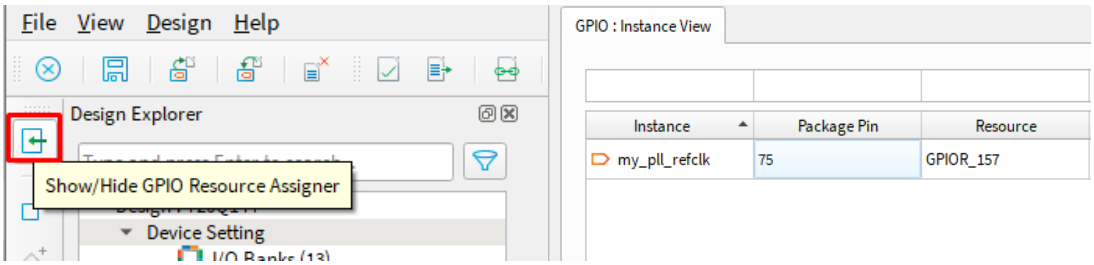
10. Right click to GPIO and select create block. Enter the instance name as my_pll_refclk. Then click the enter on the keyboard. Select mode as input and select Connection type as pll_clkin.



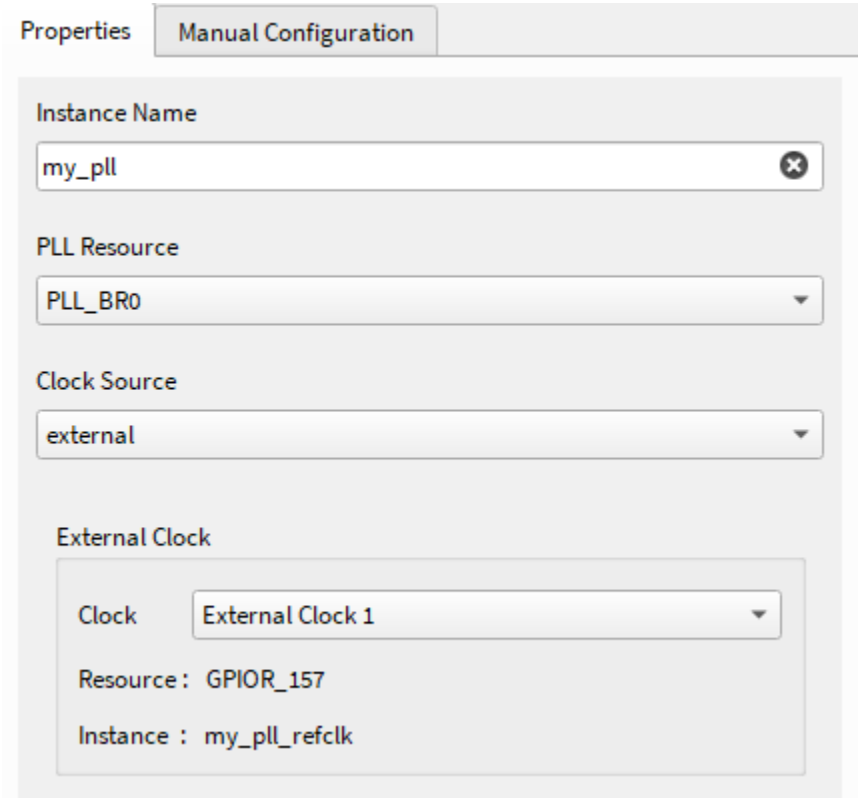
11. Right click to JTAG User Tap and select create block. Enter the Instance name as jtag_inst1. Select JTAG Resource JTAG_USER1. Pin names should be same with on the debug module ports.



12. Click show/hide GPIO Resource Assigner. Enter the Package Pin as “75”. This is our PLL resource.

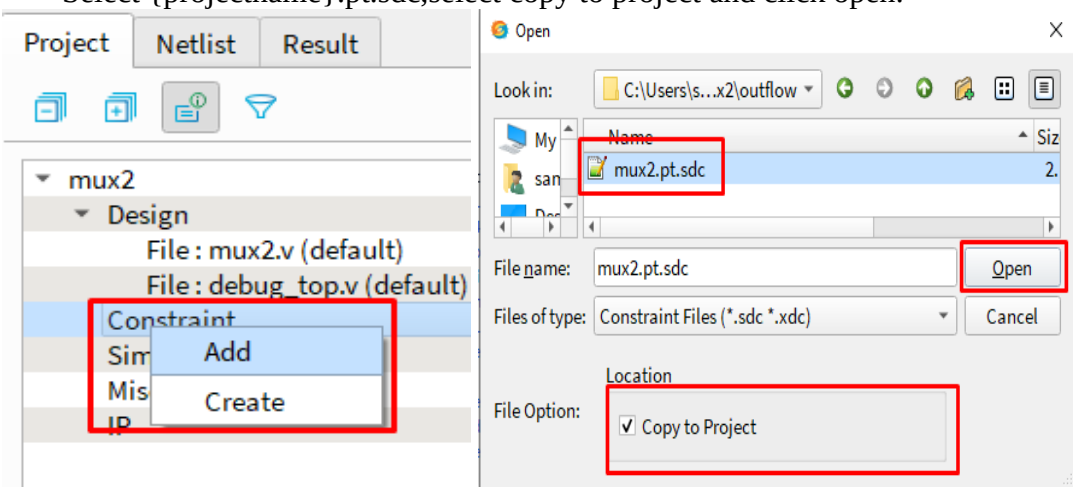


13. Then Click Save. Click again Hide GPIO Resource Assigner.
14. Right click to PLL and select create block. For 50 MHz, enter the name as my_pll, PLL Resource must select PLL_BR0 and external clock must select External Clock 1. Block editor will be like this:



15. Click Automated clock calculation. Set input frequency as 50 MHz. Set clock 0 frequency as 50 MHz and enter the name as io_systemClk. Click Finish.
16. Save and check the design then click Generate Efinity constraint file. You should not get error. You will get this prompt on the bottom:

17. Click right constraint and click add. In outflow folder, SDC file will be generated. Select {projectname}.pt.sdc,select copy to project and click open.



18. Open the mux2.v file and add this line between input and output ports.

```
input    io_systemClk,
input    jtag_inst1_TCK,
input    jtag_inst1_TDI,
output   jtag_inst1_TDO,
input    jtag_inst1_SEL,
input    jtag_inst1_CAPTURE,
input    jtag_inst1_SHIFT,
input    jtag_inst1_UPDATE,
input    jtag_inst1_RESET,
input    jtag_inst1_DRCK,
input    jtag_inst1_RUNTEST,
input    jtag_inst1_TMS,
```

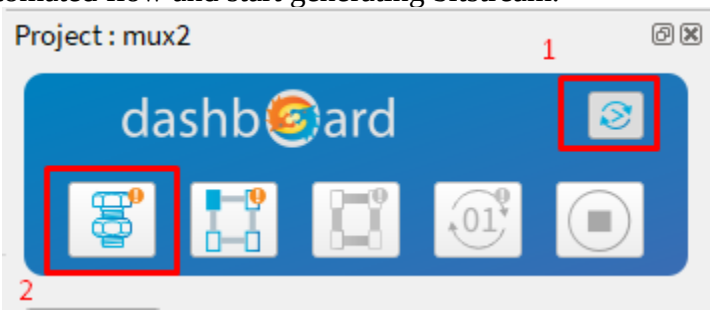
19. To use inputs with debugger, you should delete input a, b and s. Add this inputs as wire.

```
// add code here to implement the multiplexer y = s'a + sb
assign y = (~s & a) | (s & b);
wire a,b,s;
```

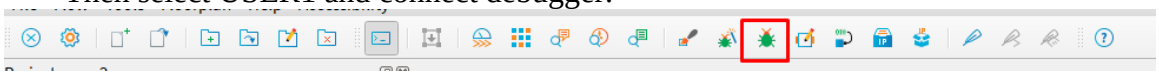
20. Mux.v file should be like:

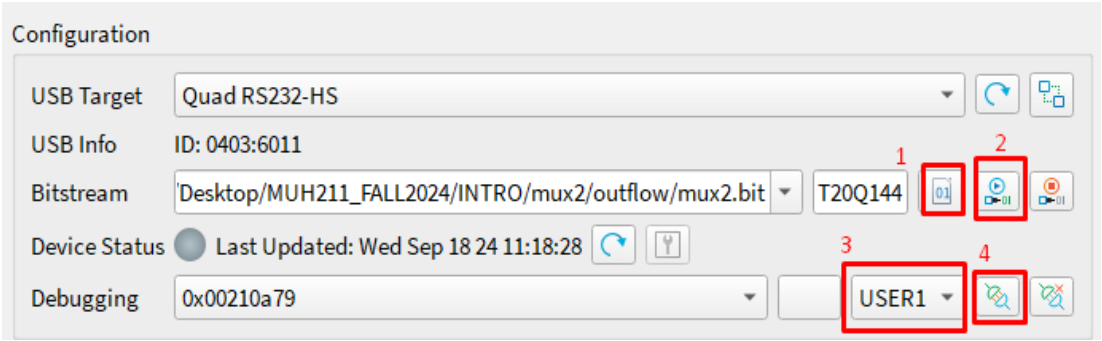
```
`timescale 1ns / 1ps
module mux2(
    input           io_systemClk,
    input           jtag_inst1_TCK,
    input           jtag_inst1_TDI,
    output          jtag_inst1_TDO,
    input           jtag_inst1_SEL,
    input           jtag_inst1_CAPTURE,
    input           jtag_inst1_SHIFT,
    input           jtag_inst1_UPDATE,
    input           jtag_inst1_RESET,
    input           jtag_inst1_DRCK,
    input           jtag_inst1_RUNTEST,
    input           jtag_inst1_TMS,
    output wire     y //  y = s'a + sb  (not really a register)
    );

// add code here to implement the multiplexer y = s'a + sb
assign y = (~s & a) | (s & b);
wire a,b,s;

edb_top edb_top_inst (
    .bscan_CAPTURE      ( jtag_inst1_CAPTURE ),
    .bscan_DRCK         ( jtag_inst1_DRCK ),
    .bscan_RESET        ( jtag_inst1_RESET ),
    .bscan_RUNTEST      ( jtag_inst1_RUNTEST ),
    .bscan_SEL          ( jtag_inst1_SEL ),
    .bscan_SHIFT        ( jtag_inst1_SHIFT ),
    .bscan_TCK          ( jtag_inst1_TCK ),
    .bscan_TDI          ( jtag_inst1_TDI ),
    .bscan_TMS          ( jtag_inst1_TMS ),
    .bscan_UPDATE       ( jtag_inst1_UPDATE ),
    .bscan_TDO          ( jtag_inst1_TDO ),
    .la0_clk            ( io_systemClk ),
    .la0_a      ( a ),
    .la0_b      ( b ),
    .la0_s      ( s ),
    .la0_y      ( y ),
    .vio0_clk       ( io_systemClk ),
    .vio0_a         ( a ),
    .vio0_b         ( b ),
    .vio0_s         ( s )
);
endmodule
```

21. Open the automated flow and start generating bitstream.

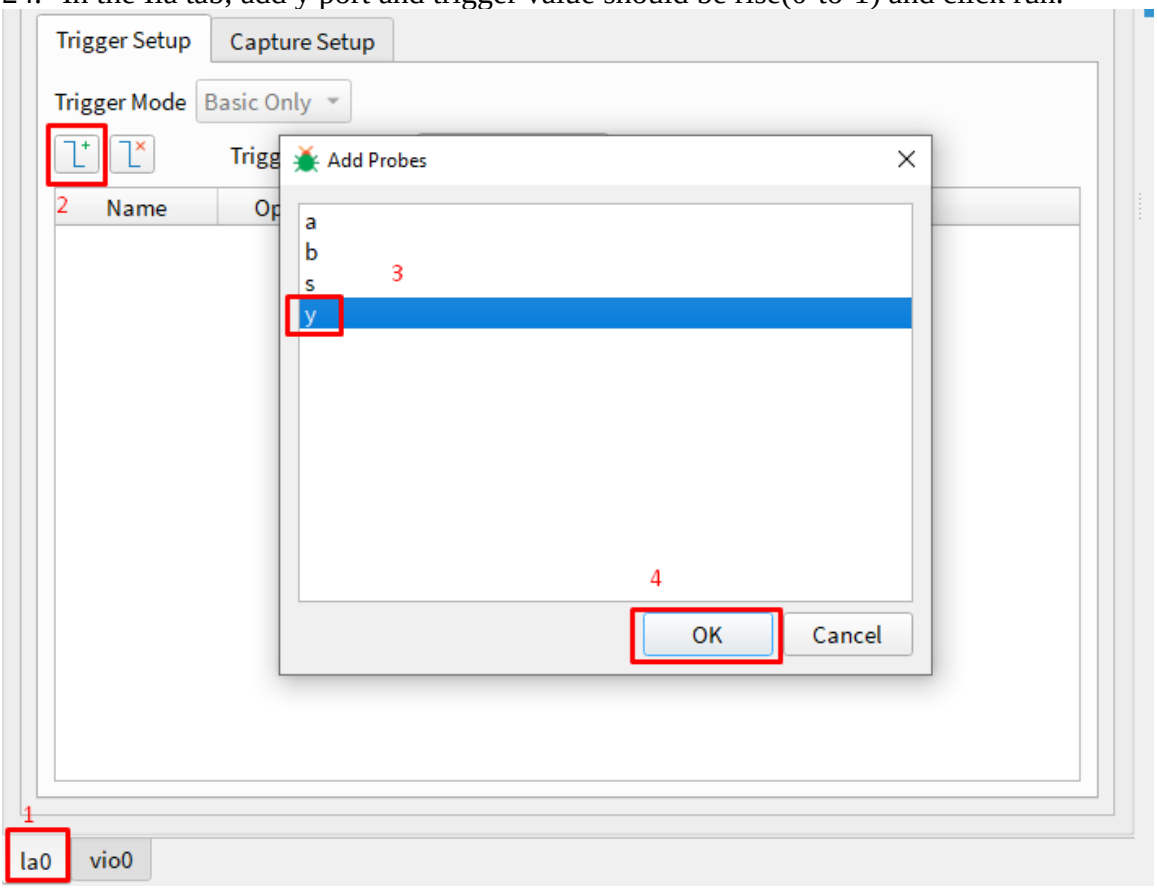


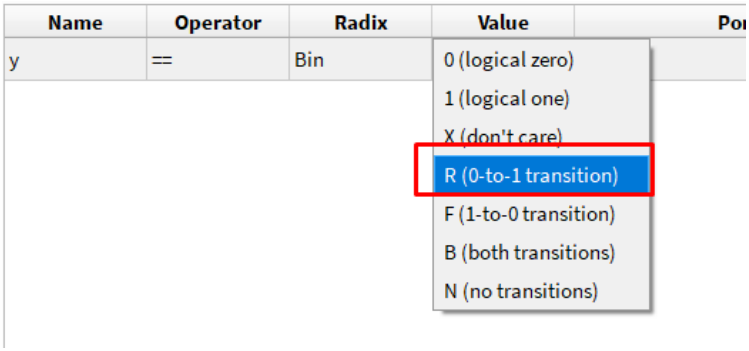
22. After generated bitstream open Debugger and load bit file in configuration window. Then select USER1 and connect debugger.



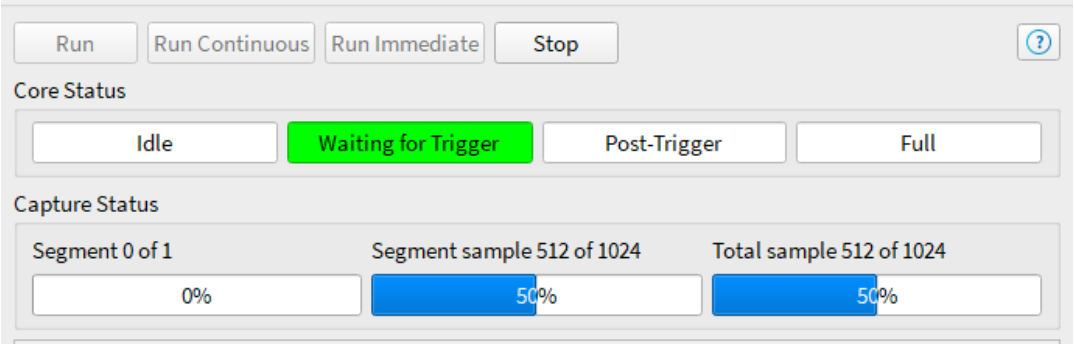


23. In the Vio tab, our source bits should be selected as 0 and you can give any number you want for a, b and s.
24. In the Ila tab, add y port and trigger value should be rise(0-to-1) and click run.

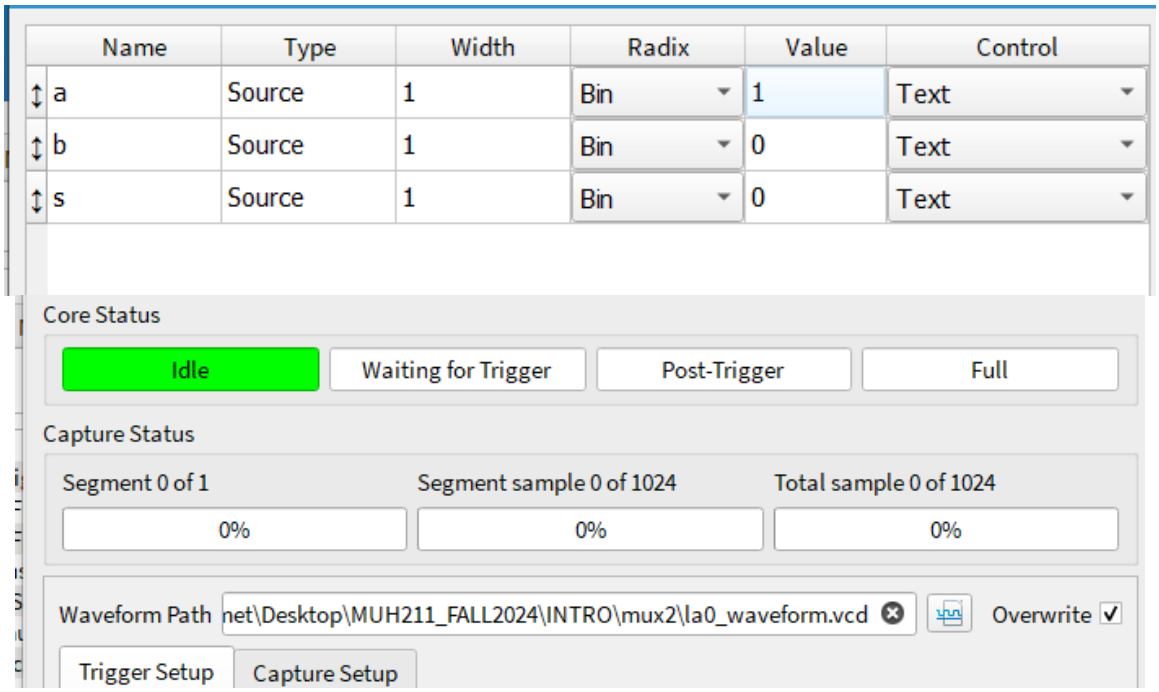




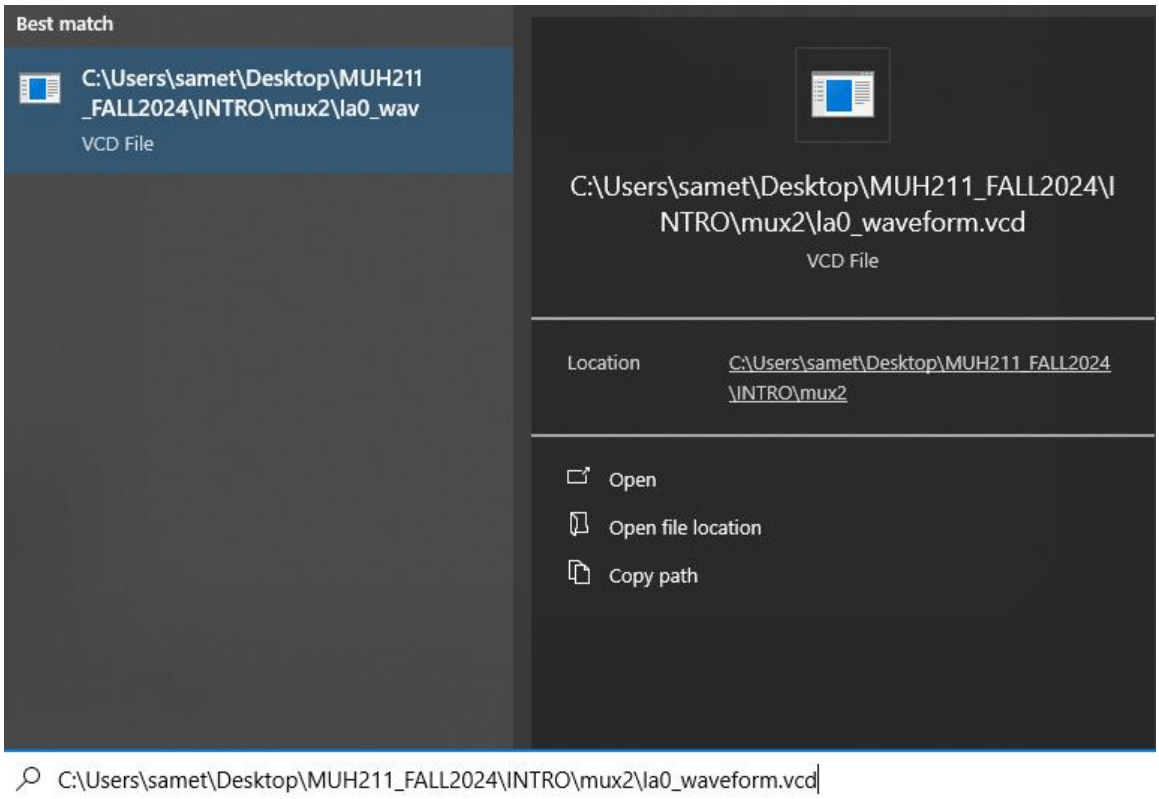
25. Core status will look like this:



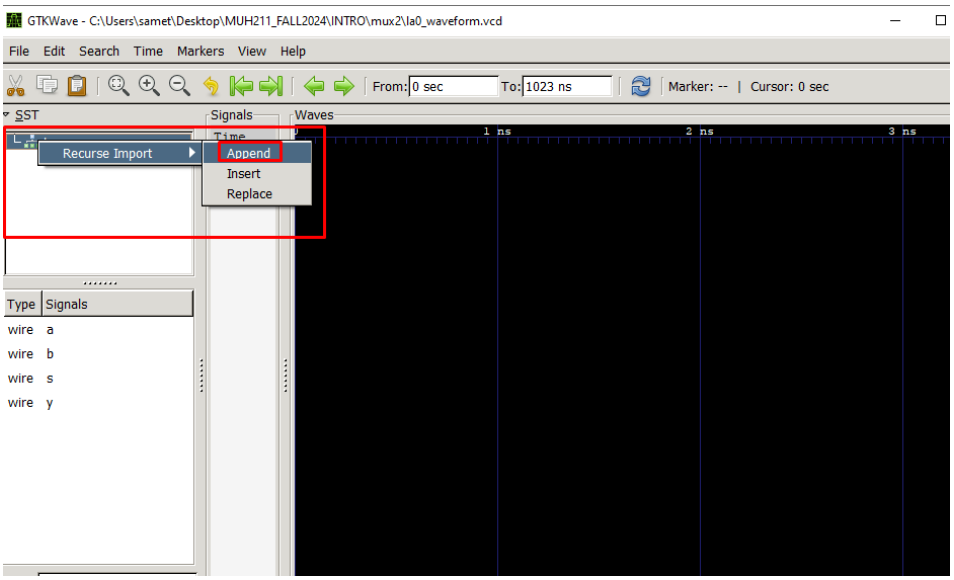
26. Then return to VIO tab. Change value of a to 1. Return the ILA tab. Core is triggered and core status is returned to IDLE.

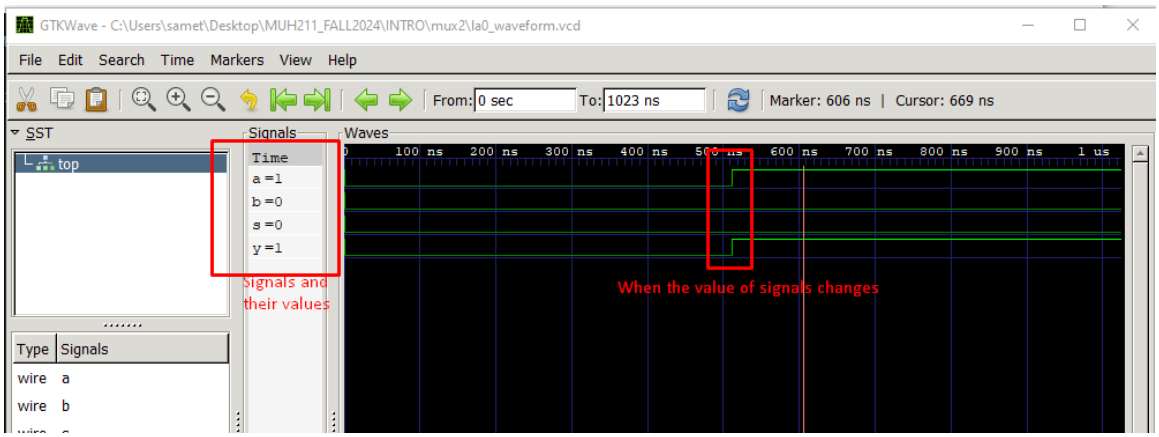


27. Copy the waveform path and open it with GTKWave.



28. Right click the top and append all signals. Click zoom out and you will see all signals and their values.





29. You can change other input values with VIO and trigger with ILA.