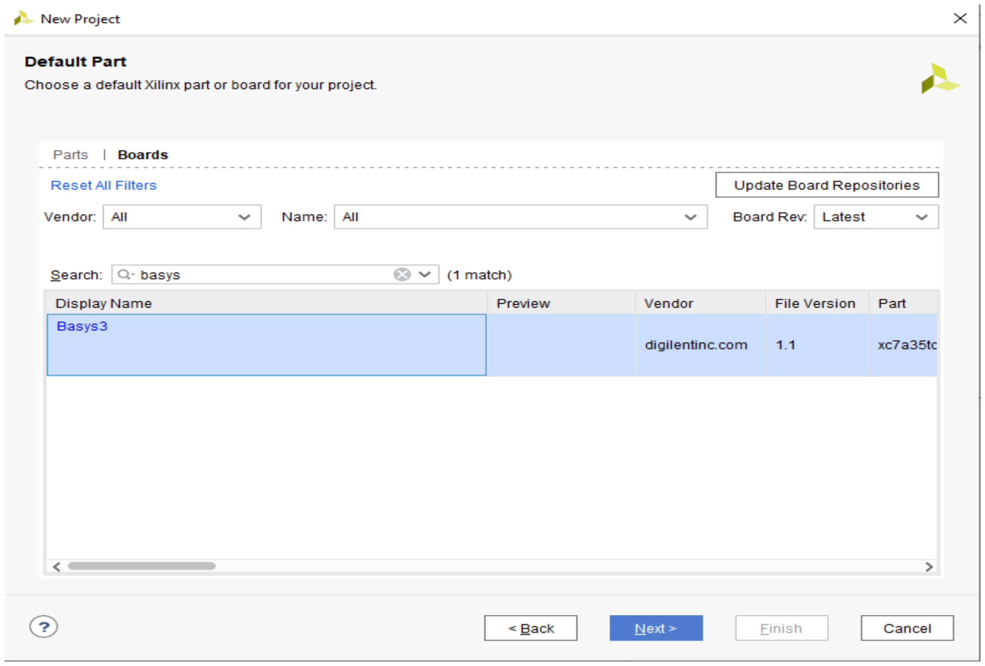
Lab 3

Tasks to be done in this lab:

1. Design and implement an 8-bit counter operating at a frequency of 1 Hz and verify its functionality.
2. Get familiar with Clocking Wizard IP and design a frequency divider capable of dividing the given frequency into the desired frequency.
3. Test the functionality of Counter on the Basys3 Board (remote access) using VIO and ILA IP.

Step 1: Create the Top Module for the Counter.

1. Create a New Project.
2. Select Project Type as RTL project.
3. In the Add Sources window, first, we will create the Verilog file for the top module(top_counter.v).
4. You don't need to add constraints at this time; we will do it in the later stages.
5. Add the board as Basys3 Board as shown in the screenshot below


New Project

Default Part

Choose a default Xilinx part or board for your project.

Parts | **Boards**

Reset All Filters

Update Board Repositories

Vendor: All

Name: All

Board Rev: Latest

Search: basys (1 match)

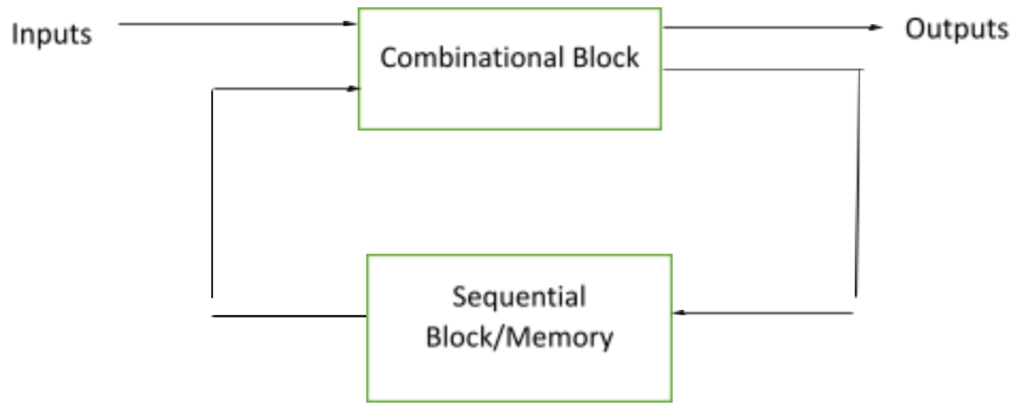
Display Name	Preview	Vendor	File Version	Part
Basys3		digilentinc.com	1.1	xc7a35tc

< Back | Next > | Finish | Cancel

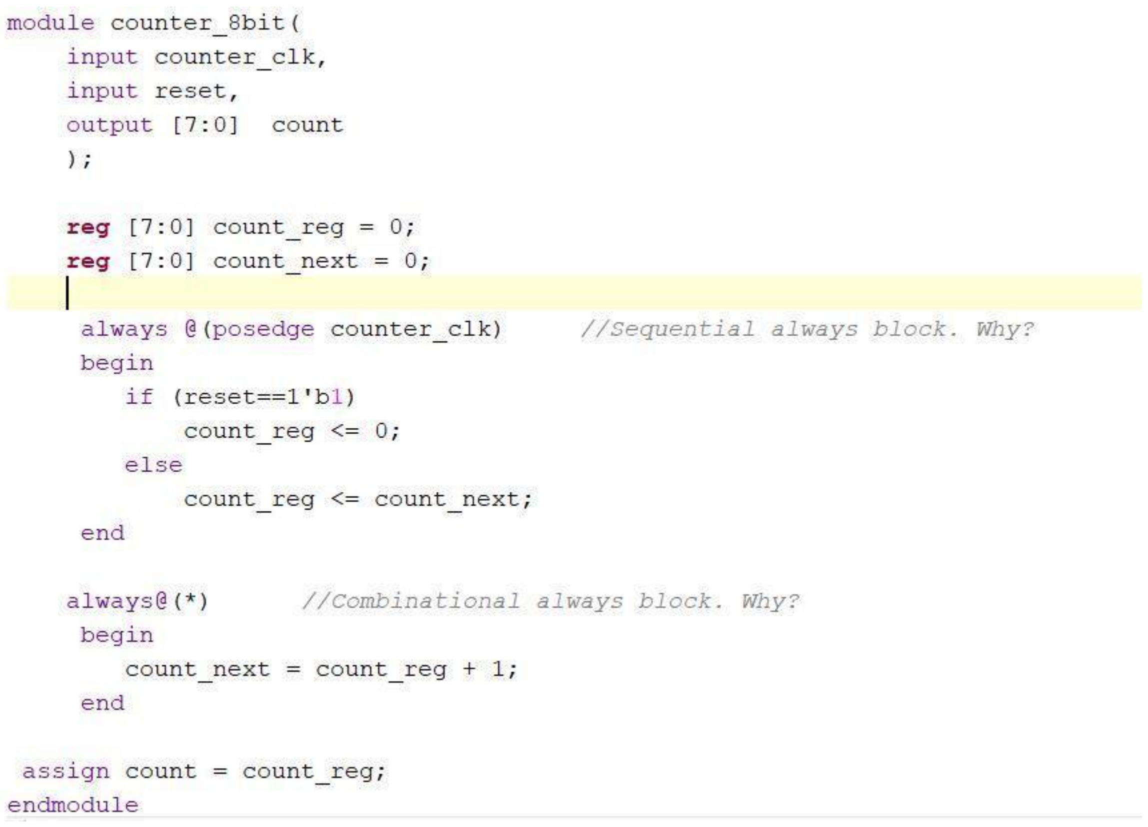
6. Make the port declaration, two input ports(clk_125M, reset) of 1 bit each, and one output port count of 8 bits.
7. After completing Step 6, your project must have a module top_counter, which should have the module and port declaration only.

Step 2: Add a new design source to define the functionality of the Counter.

1. Now we will add the functionality of the Counter. To do so, add design sources and create one more Verilog file counter_8bit.v. Add the two input ports (counter_clock, reset) of 1 bit each and one output port count of 8 bits.
2. The Counter is a sequential circuit. The block diagram of a sequential circuit is shown below:



3. We will write the Verilog code for the Counter in the same manner. Our code will consist of one combinational always block, and one sequential always block. The code for Counter is given in the screenshot below.
Note: Please look for both the blocks(combinational and sequential)



```
module counter_8bit(
    input counter_clk,
    input reset,
    output [7:0]  count
    );

    reg [7:0] count_reg = 0;
    reg [7:0] count_next = 0;

     always @(posedge counter_clk)     //Sequential always block. Why?
     begin
        if (reset==1'b1)
            count_reg <= 0;
        else
            count_reg <= count_next;
     end

    always@(*)       //Combinational always block. Why?
     begin
        count_next = count_reg + 1;
     end

 assign count = count_reg;
endmodule
```

Please note that we have used a synchronous reset in our design.

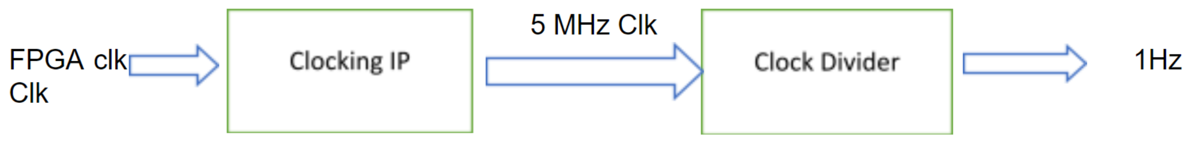
Step 3: Add Clocking Wizard IP to the project:

The next step is to add the clocking IP. Why do we need this IP in our project?
We want to operate our Counter at 1 Hz clock frequency. The frequency of the Clock on the FPGA board is 100 MHz. Now to scale it down to 1 Hz, we need to make use of a clock divider. There are two options to do so:

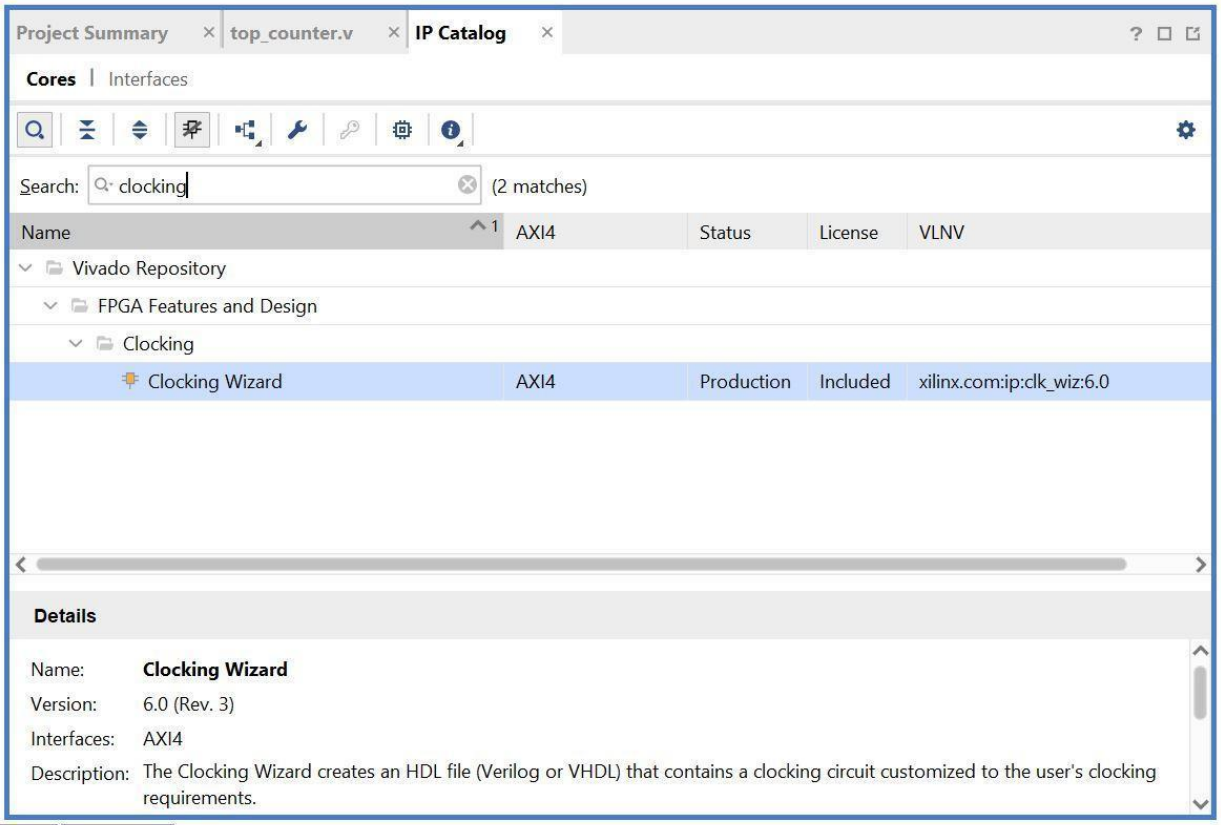
Option 1: Give the 100 MHz Clock as the input to the clock divider and obtain 1 Hz clock.
Option 2. Give a few MHz (For Eg: 5 MHz) clock input to the clock divider and obtain a 1 Hz clock.

Option 2 is a better option because we will need fewer resources to convert from a few MHz (For Eg: 5MHz) to Hz than 100 MHz to 1 Hz. Also, the synthesized Clock will have less margin of errors.

To synthesize a clock of lower frequencies from the input clock frequency of 100 Mhz, we can make use of Clocking IP.
Note: The minimum clock frequency that can be synthesized from Clock IP is 4.7 MHz. So we will first synthesize a clock of 5MHz frequency from Clock IP and then feed it to the clock divider to make a clock of 1Hz frequency.



1. Open the IP Catalog and search for Clocking Wizard IP. Double click on it.



Project Summary | top_counter.v | IP Catalog

Cores | Interfaces

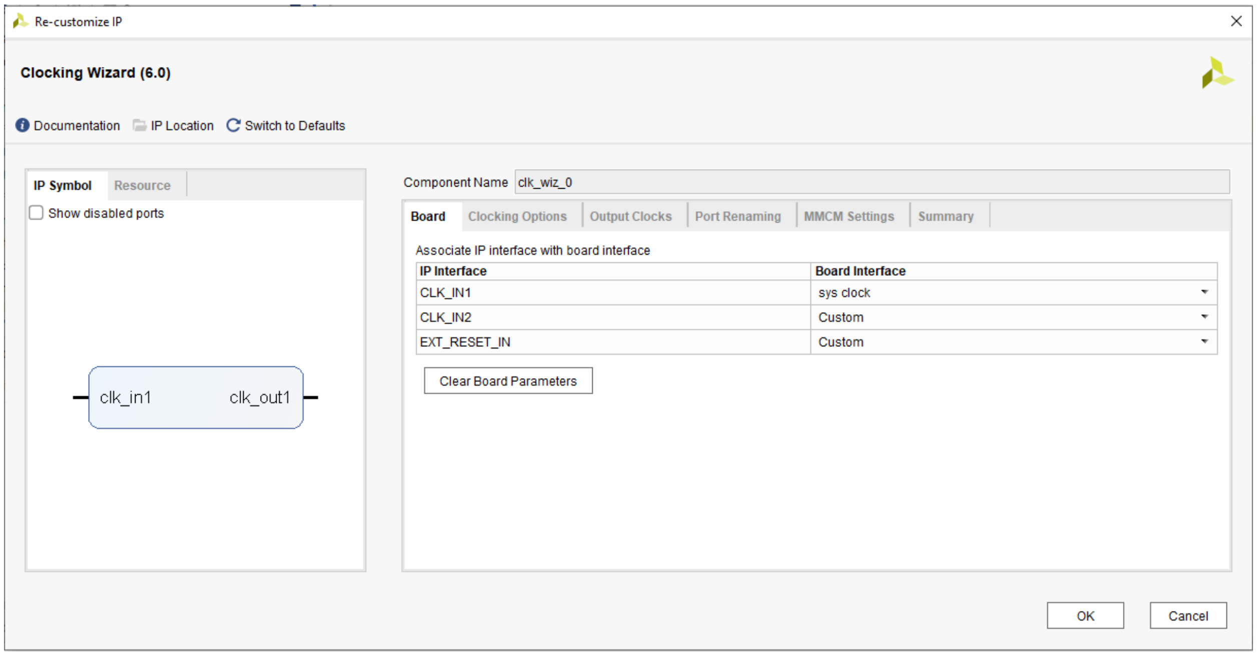
Search: clocking (2 matches)

Name	AXI4	Status	License	VLNV
Vivado Repository				
FPGA Features and Design				
Clocking				
Clocking Wizard	AXI4	Production	Included	xilinx.com:ip:clk_wiz:6.0

Details

Name: Clocking Wizard
Version: 6.0 (Rev. 3)
Interfaces: AXI4
Description: The Clocking Wizard creates an HDL file (Verilog or VHDL) that contains a clocking circuit customized to the user's clocking requirements.

2. You will get the first window which will look like the one shown below. For the CLK_IN1 field, select sysclock form the dropdown as shown below.



Re-customize IP

Clocking Wizard (6.0)

Documentation | IP Location | Switch to Defaults

IP Symbol | Resource

Show disabled ports

Component Name clk_wiz_0

Board | Clocking Options | Output Clocks | Port Renaming | MMCM Settings | Summary

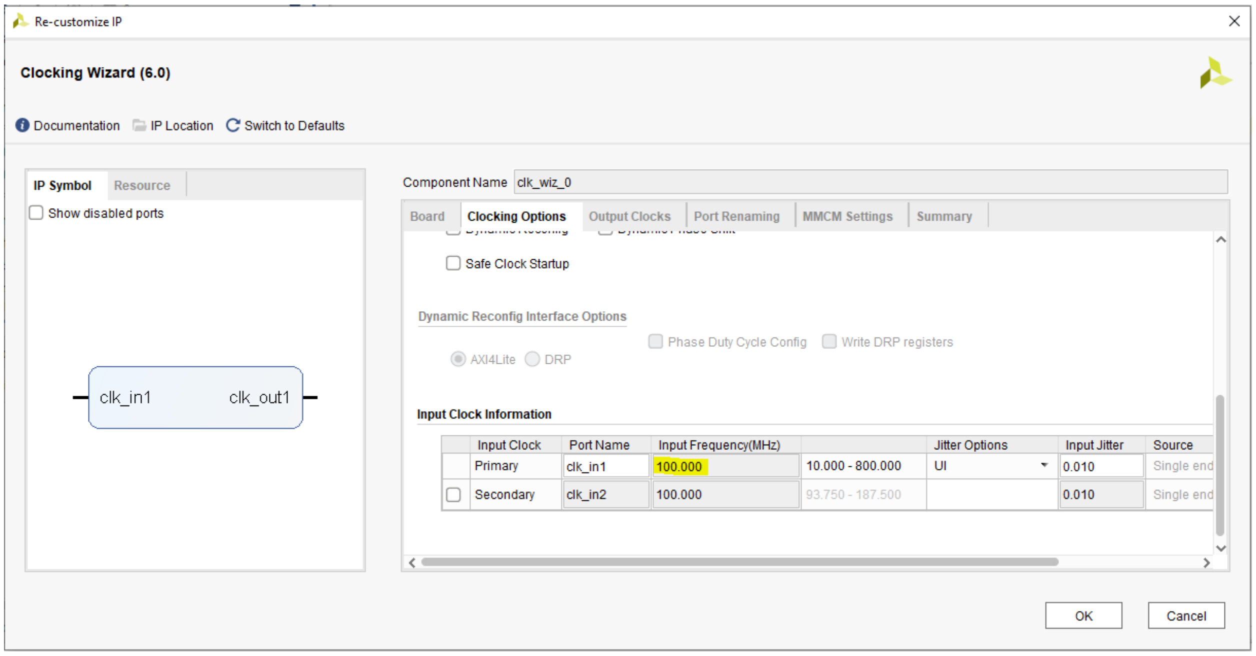
Associate IP interface with board interface

IP Interface	Board Interface
CLK_IN1	sys clock
CLK_IN2	Custom
EXT_RESET_IN	Custom

Clear Board Parameters

OK | Cancel

3. In the clocking options window, scroll down to bottom and you can see that the primary input clock frequency is 100 MHz.


Re-customize IP

Clocking Wizard (6.0)

Component Name clk_wiz_0

Safe Clock Startup

Dynamic Reconfig Interface Options

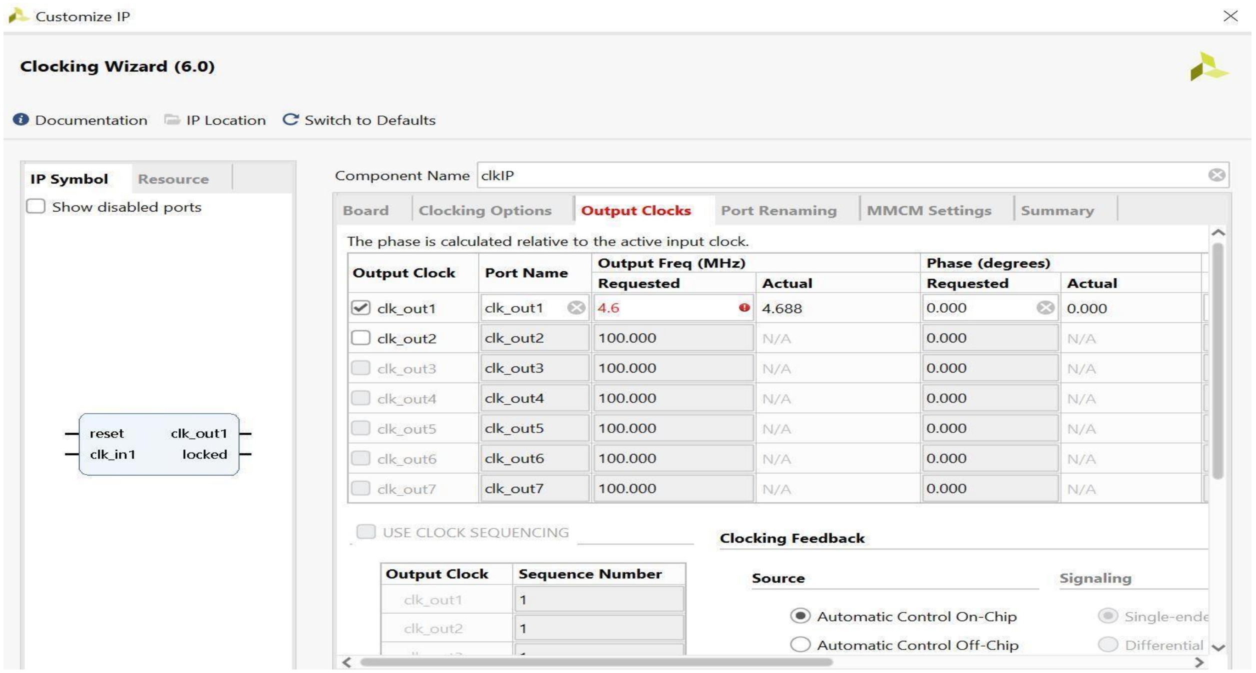
Phase Duty Cycle Config | Write DRP registers

AXI4Lite | DRP

Input Clock Information

	Input Clock	Port Name	Input Frequency(MHz)		Jitter Options	Input Jitter	Source
	Primary	clk_in1	100.000	10.000 - 800.000	UI	0.010	Single end
☐	Secondary	clk_in2	100.000	93.750 - 187.500		0.010	Single end

4. Next, go to the Output Clocks window. You can see we can synthesize multiple output clocks of different frequencies. We need only one Clock for our purpose. Try any clock frequency lower than 4.7 MHz; you will see that it cannot be synthesized as shown below.


Customize IP

Clocking Wizard (6.0)

Component Name clkIP

The phase is calculated relative to the active input clock.

Output Clock	Port Name	Output Freq (MHz)		Phase (degrees)	
		Requested	Actual	Requested	Actual
✓ clk_out1	clk_out1	4.6	4.688	0.000	0.000
☐ clk_out2	clk_out2	100.000	N/A	0.000	N/A
☐ clk_out3	clk_out3	100.000	N/A	0.000	N/A
☐ clk_out4	clk_out4	100.000	N/A	0.000	N/A
☐ clk_out5	clk_out5	100.000	N/A	0.000	N/A
☐ clk_out6	clk_out6	100.000	N/A	0.000	N/A
☐ clk_out7	clk_out7	100.000	N/A	0.000	N/A

USE CLOCK SEQUENCING

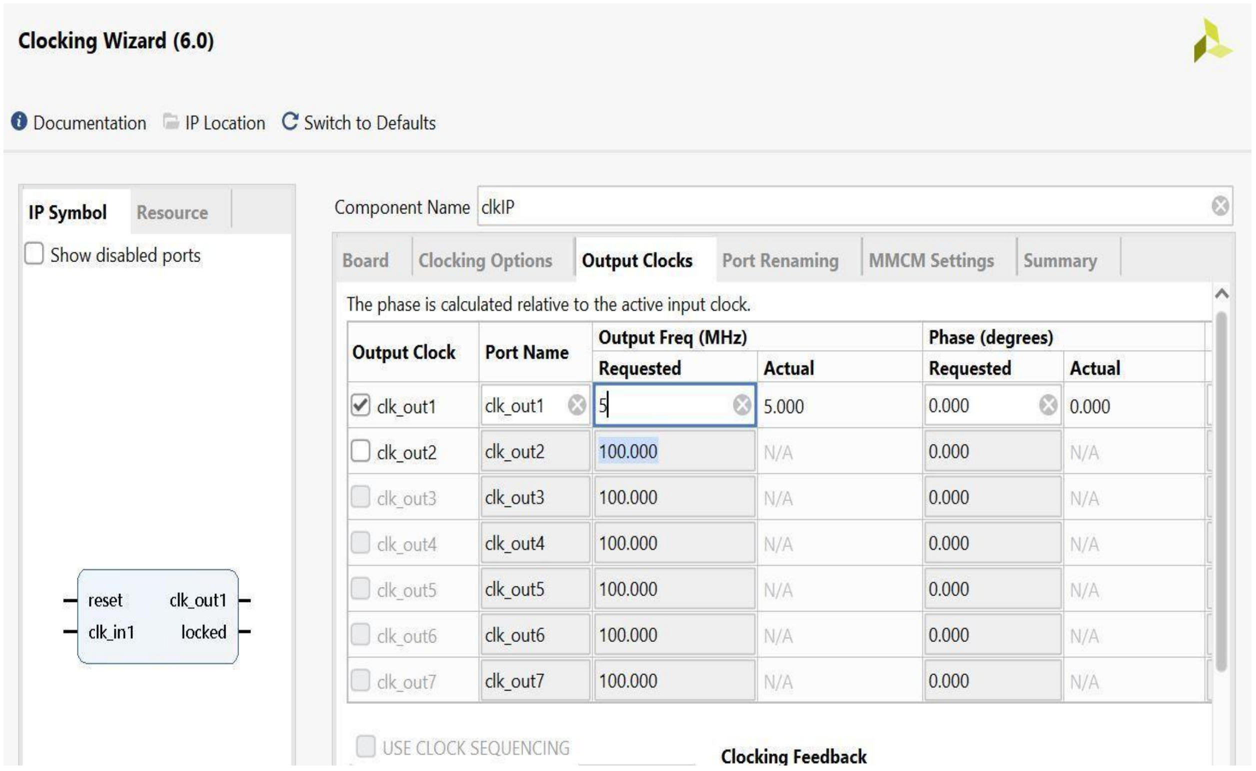
Output Clock	Sequence Number
clk_out1	1
clk_out2	1

Clocking Feedback

Source: [✓ Automatic Control On-Chip] | Automatic Control Off-Chip

Signaling: Single-ende | Differential

4. Enter the 5 Mhz frequency as the output clock frequency. Scroll down to the bottom of the page and untick reset and locked options as shown below:


Clocking Wizard (6.0)

[ ] Show disabled ports

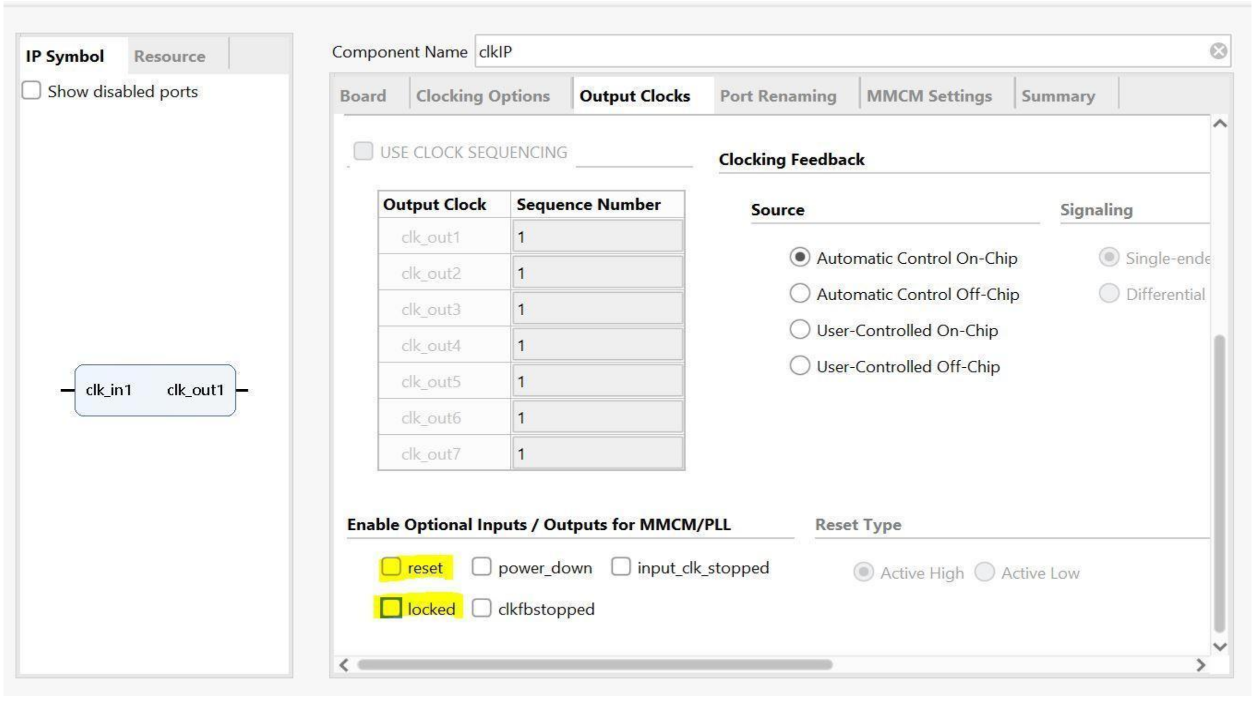
Component Name: clkIP

The phase is calculated relative to the active input clock.

Output Clock	Port Name	Output Freq (MHz)		Phase (degrees)	
		Requested	Actual	Requested	Actual
[✓] clk_out1	clk_out1	5	5.000	0.000	0.000
[ ] clk_out2	clk_out2	100.000	N/A	0.000	N/A
[ ] clk_out3	clk_out3	100.000	N/A	0.000	N/A
[ ] clk_out4	clk_out4	100.000	N/A	0.000	N/A
[ ] clk_out5	clk_out5	100.000	N/A	0.000	N/A
[ ] clk_out6	clk_out6	100.000	N/A	0.000	N/A
[ ] clk_out7	clk_out7	100.000	N/A	0.000	N/A

[ ] USE CLOCK SEQUENCING

Clocking Feedback



[ ] Show disabled ports

Component Name: clkIP

[ ] USE CLOCK SEQUENCING

Output Clock	Sequence Number
clk_out1	1
clk_out2	1
clk_out3	1
clk_out4	1
clk_out5	1
clk_out6	1
clk_out7	1

Clocking Feedback

Source: [✓ Automatic Control On-Chip] Automatic Control Off-Chip / User-Controlled On-Chip / User-Controlled Off-Chip

Signaling: [✓ Single-ende] Differential

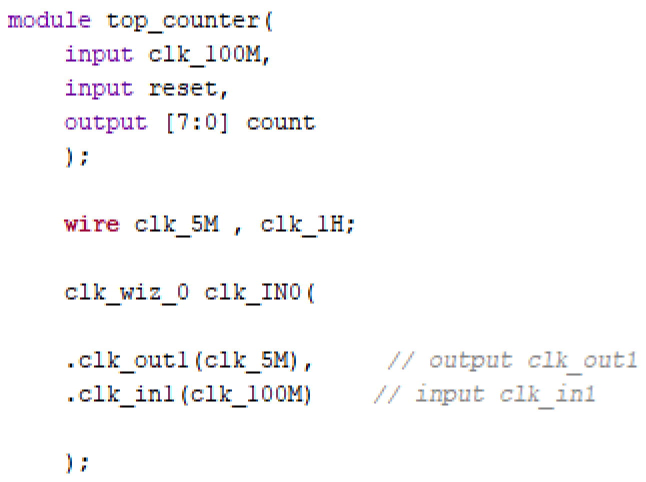
Enable Optional Inputs / Outputs for MMCM/PLL

[ ] reset [ ] power_down [ ] input_clk_stopped

[ ] locked [ ] clkfbstopped

Reset Type: [✓ Active High] Active Low

5. Click OK. Next, we will instantiate the Clocking IP in the top module. Use the same steps as we did for VIO IP to get the instantiation template.
6. Now the top module will look as below:

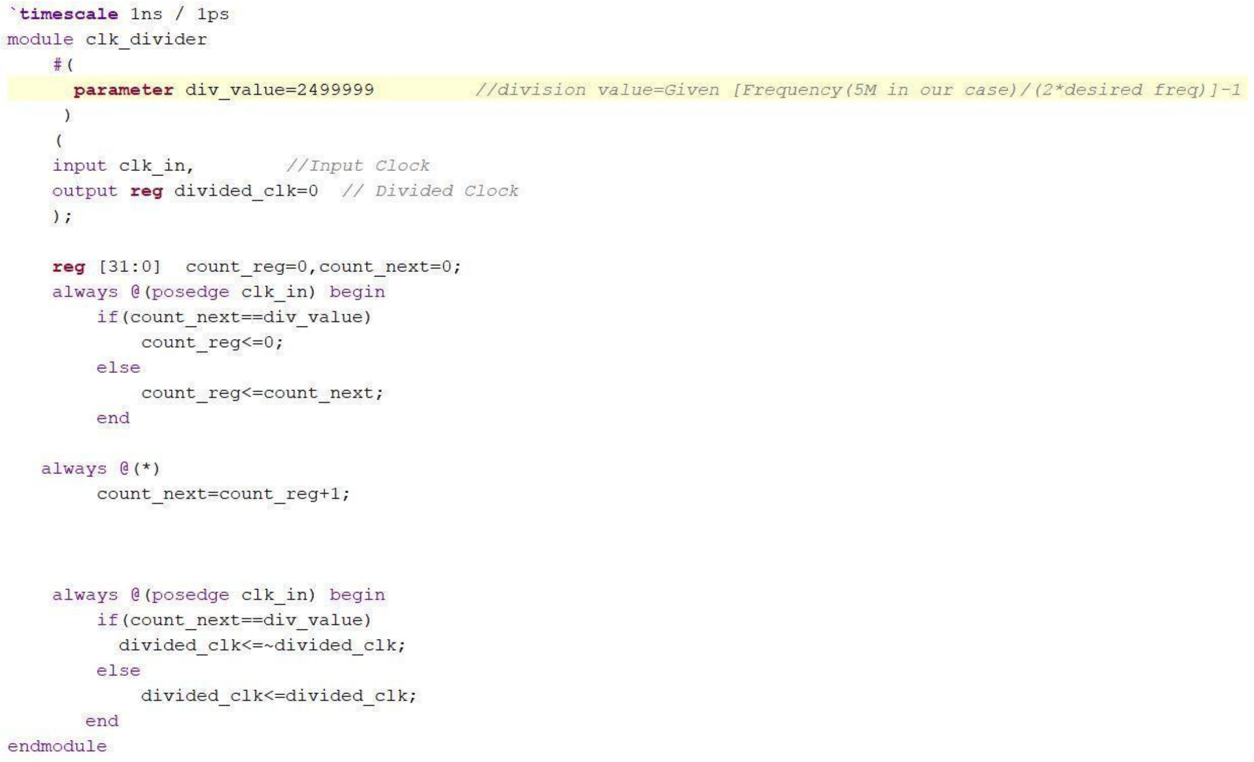


```
module top_counter(
    input clk_100M,
    input reset,
    output [7:0] count
    );

    wire clk_5M , clk_1H;

    clk_wiz_0 clk_IN0(

    .clk_out1(clk_5M),      // output clk_out1
    .clk_in1(clk_100M)     // input clk_in1

    );
```

Step 4: Make a clock divider module:

Now we have to synthesize 1 Hz clock from 5 MHz clock. For doing so, we will make use of a Counter based frequency divider.

1. Add a new design source clk_divider.v with one input (clk_in) and one output (divided_clk).
2. The code for the clock divider is shown below. Please note how to chose the value of div_value.



```
`timescale 1ns / 1ps
module clk_divider
    #(
      parameter div_value=2499999         //division value=Given [Frequency(5M in our case)/(2*desired freq)]-1
     )
    (
    input clk_in,          //Input Clock
    output reg divided_clk=0   // Divided Clock
    );

    reg [31:0]  count_reg=0,count_next=0;
    always @(posedge clk_in) begin
        if(count_next==div_value)
            count_reg<=0;
        else
            count_reg<=count_next;
        end

   always @(*)
        count_next=count_reg+1;



    always @(posedge clk_in) begin
        if(count_next==div_value)
          divided_clk<=~divided_clk;
        else
            divided_clk<=divided_clk;
       end
endmodule
```

3. Next, we will instantiate the counter_8bit.v and clk_divider.v in our top module.

```
module top_counter(
    input clk_100M,
    input reset,
    output [7:0] count
    );

    wire clk_5M , clk_1H;

    clk_wiz_0 clk_IN0(

    .clk_out1(clk_5M),      // output clk_out1
    .clk_in1(clk_100M)     // input clk_in1

    );

    clk_divider #(.div_value(2499999)) clk_div_IN0(.clk_in(clk_5M) , .divided_clk(clk_1H));

    counter_8bit count_IN0(.counter_clk(clk_1H), .reset(reset) , .count(count));
```

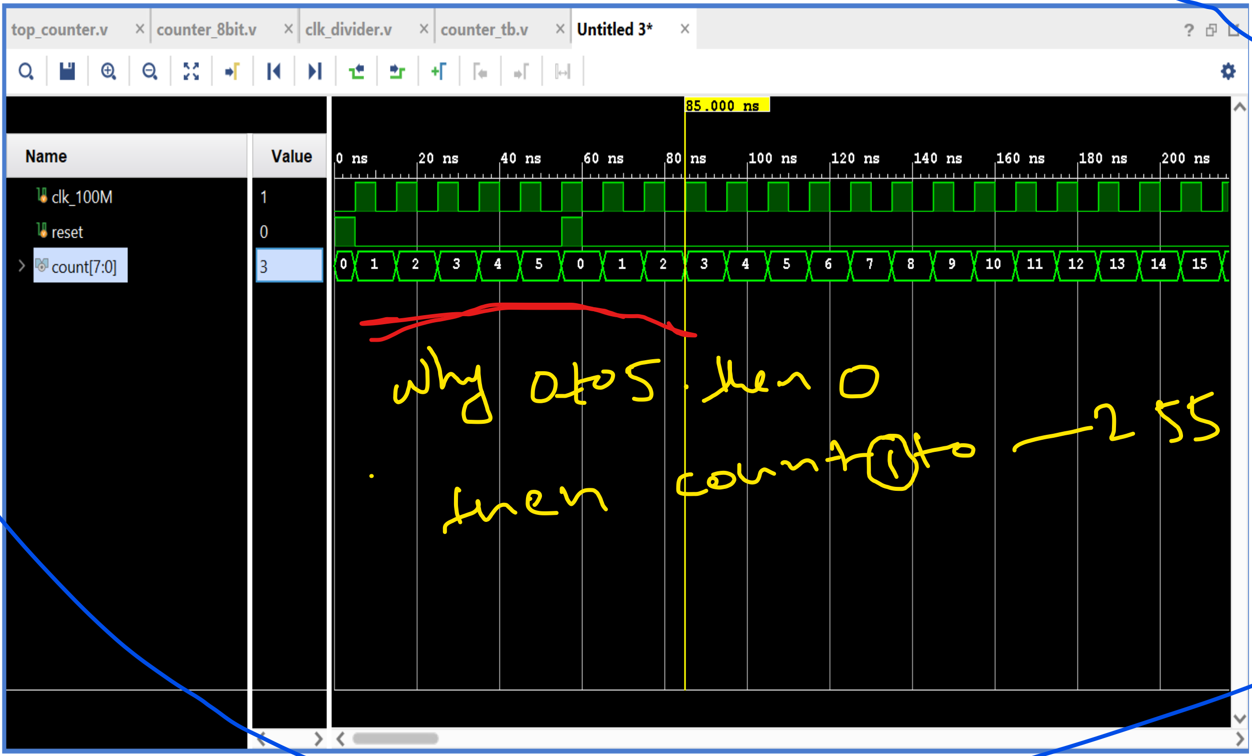
Step 5: Test the functionality of the Counter using testbench.

1. Add a simulation source and write the testbench code, as shown below:

```
module counter_tb(

    );

    reg clk,reset;                //Step 1: Define the variables
    wire [7:0] count;

    counter_8bit tb1(.counter_clk(clk),.reset(reset),.count(count));  //Step 2: Instantiate DUT

    initial      //Step 3: Initialize variables
        begin
            clk=1'b0;
            reset=1'b1;
        end

    initial   //Step 4: Change stimulus
        begin
            #5 reset=1'b0;
            #50 reset=1'b1;
            #5 reset=1'b0;
        end

    always #5 clk = ~clk;  // Clock with time period of 10ns
endmodule
```

Note: We have instantiated the counter_8bit module here and not the top_counter module. The reason being we only need to check the functionality of the Counter and not the exact working frequency. Also note that we have not given 1 Hz clock here. You can also instantiate the top_counter module, but the simulator will take so much time to increment the count due to 1Hz frequency.

2. Run the simulation and check the behavior of Counter with reset and once the reset is de-asserted.



why 0to5 then 0

then count $\boxed{1}$ to — 255

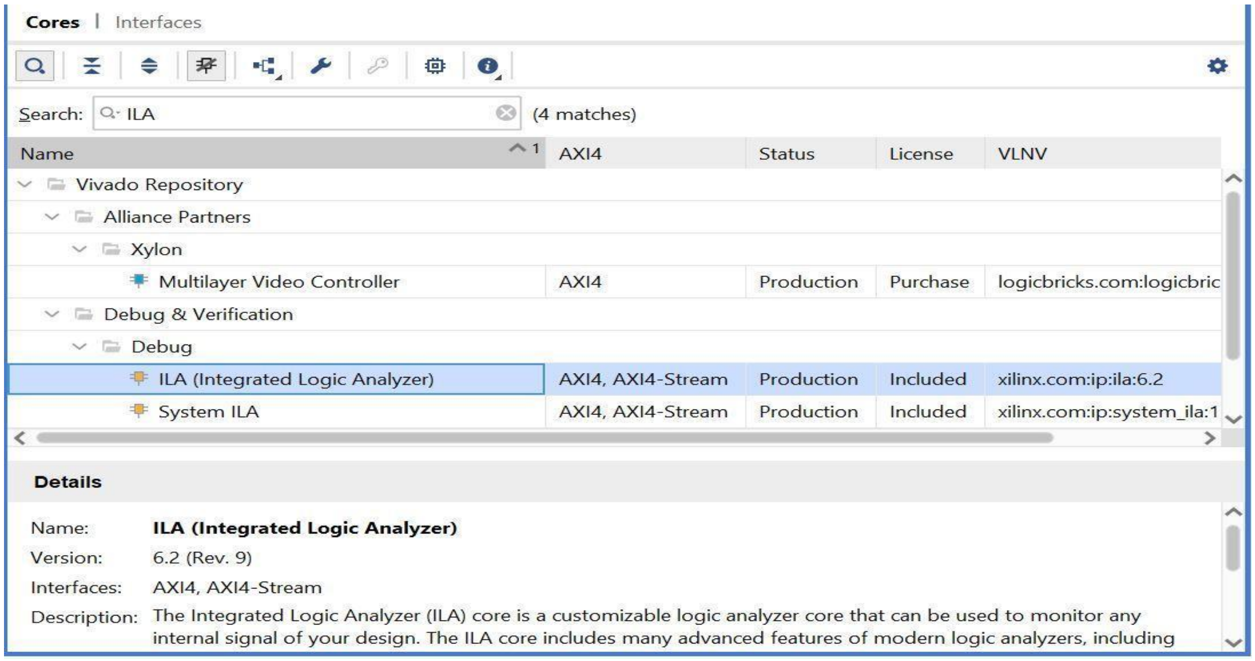
Step 6: Add VIO and ILA IP to your design to test it on hardware.

1. Create a vio_wrapper in the project in a similar way as we did in Lab2. The VIO IP will contain one input probe of 8 bits (to monitor count) and one output probe to give reset input to design. The vio_wrapper module will look like, as shown below:

```
module vio_wrapper(
    input clk
    );

    wire [7:0] count;
    wire reset;

    vio_0 in0 (
  .clk(clk),                  // input wire clk
  .probe_in0(count),      // input wire [7 : 0] probe_in0
  .probe_out0(reset)   // output wire [0 : 0] probe_out0
);

    top_counter in1(.clk_100M(clk) , .reset(reset) , .count(count));

endmodule
```

2. The next step is to add the ILA(Integrated Logic Analyzer) IP. This IP is again very useful for debugging purposes. You can get the real time waveform of the signals from the hardware using ILA. You can create triggers on the signals depending on some values or conditions. We will use ILA IP to capture the signals on the port of our top-level counter module whenever the count reaches a certain value and whenever the reset takes place.

3. To add ILA IP to your project, go to IP catalog and search for ILA and double click on it.



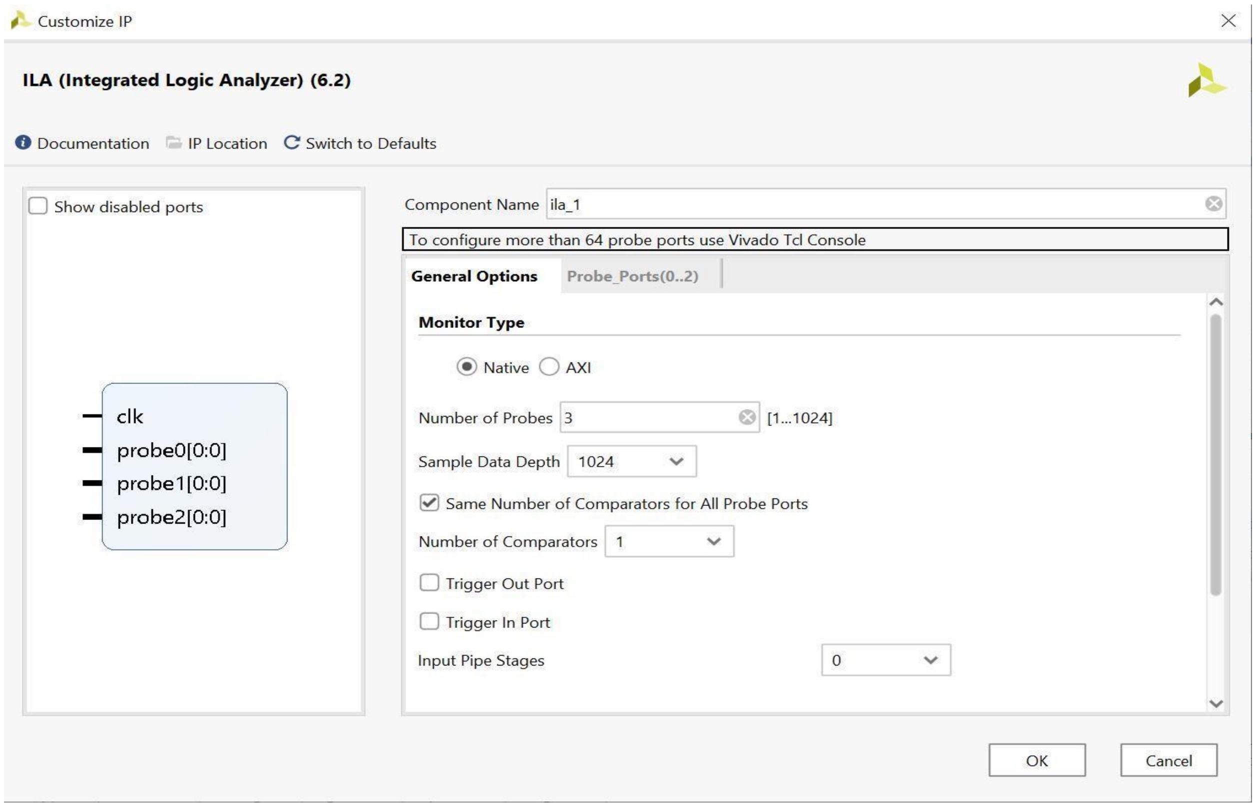
Search: ILA (4 matches)

Name	AXI4	Status	License	VLNV
Vivado Repository				
Alliance Partners				
Xylon				
Multilayer Video Controller	AXI4	Production	Purchase	logicbricks.com:logicbric
Debug & Verification				
Debug				
ILA (Integrated Logic Analyzer)	AXI4, AXI4-Stream	Production	Included	xilinx.com:ip:ila:6.2
System ILA	AXI4, AXI4-Stream	Production	Included	xilinx.com:ip:system_ila:1

Details

Name:	**ILA (Integrated Logic Analyzer)**
Version:	6.2 (Rev. 9)
Interfaces:	AXI4, AXI4-Stream
Description:	The Integrated Logic Analyzer (ILA) core is a customizable logic analyzer core that can be used to monitor any internal signal of your design. The ILA core includes many advanced features of modern logic analyzers, including

4. Select the number of probes as three(for clk_1H, reset, count). Keep the rest of the options set to default.



Customize IP

ILA (Integrated Logic Analyzer) (6.2)

Documentation | IP Location | Switch to Defaults

Show disabled ports

Component Name: ila_1

To configure more than 64 probe ports use Vivado Tcl Console

General Options | Probe_Ports(0..2)

Monitor Type

[✓ Native] AXI

Number of Probes: 3 [1...1024]

Sample Data Depth: 1024

[✓] Same Number of Comparators for All Probe Ports

Number of Comparators: 1

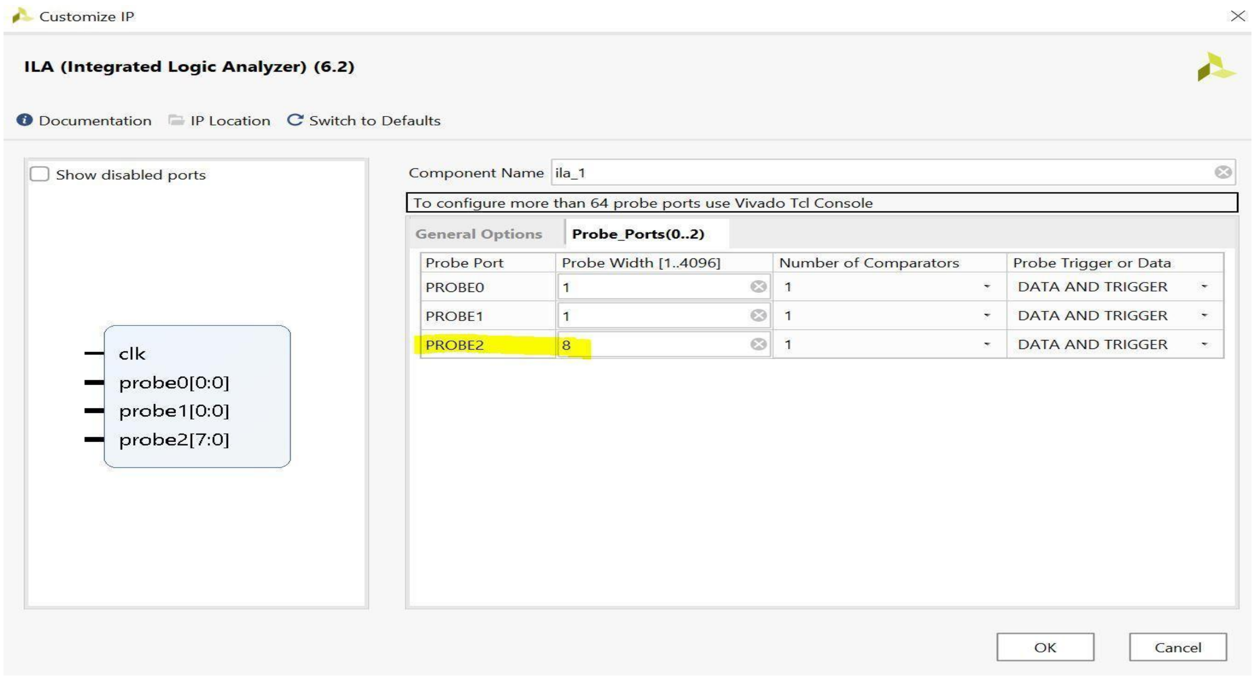
[ ] Trigger Out Port

[ ] Trigger In Port

Input Pipe Stages: 0

OK | Cancel

5. Change the width of one probe to 8 bits as shown below and then click OK.



Customize IP

ILA (Integrated Logic Analyzer) (6.2)

Documentation | IP Location | Switch to Defaults

Show disabled ports

Component Name: ila_1

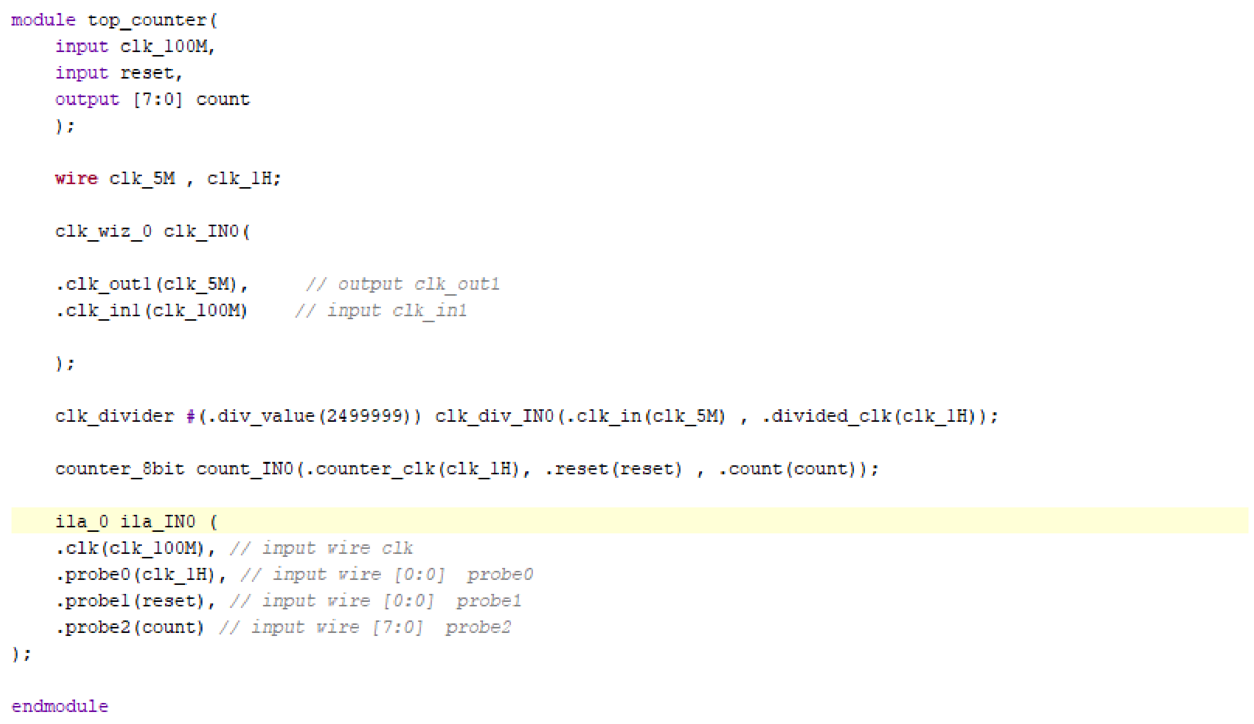
To configure more than 64 probe ports use Vivado Tcl Console

General Options | Probe_Ports(0..2)

Probe Port	Probe Width [1..4096]	Number of Comparators	Probe Trigger or Data
PROBE0	1	1	DATA AND TRIGGER
PROBE1	1	1	DATA AND TRIGGER
PROBE2	8	1	DATA AND TRIGGER

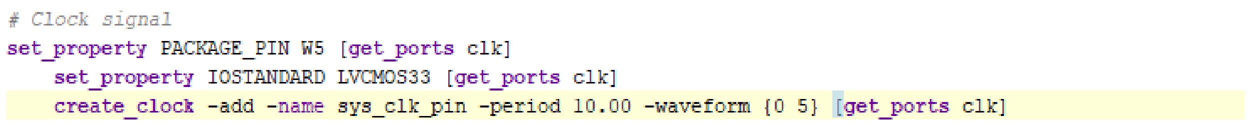
OK | Cancel

6. In the top_counter module, instantiate the ILA using the instantiate template and make the connections as shown in the code below:



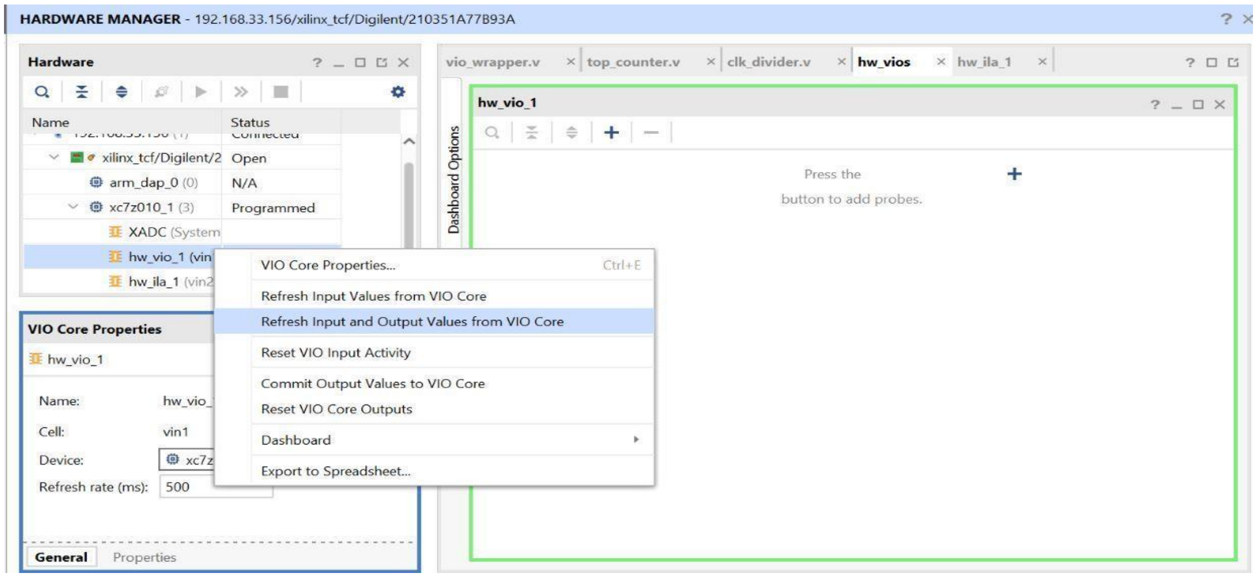
```
module top_counter(
    input clk_100M,
    input reset,
    output [7:0] count
    );

    wire clk_5M , clk_1H;

    clk_wiz_0 clk_IN0(

    .clk_out1(clk_5M),      // output clk_out1
    .clk_in1(clk_100M)     // input clk_in1

    );

    clk_divider #(.div_value(2499999)) clk_div_IN0(.clk_in(clk_5M) , .divided_clk(clk_1H));

    counter_8bit count_IN0(.counter_clk(clk_1H), .reset(reset) , .count(count));

    ila_0 ila_IN0 (
    .clk(clk_100M), // input wire clk
    .probe0(clk_1H), // input wire [0:0]  probe0
    .probe1(reset), // input wire [0:0]  probe1
    .probe2(count) // input wire [7:0]  probe2
);

endmodule
```

7. Add the constraints file.

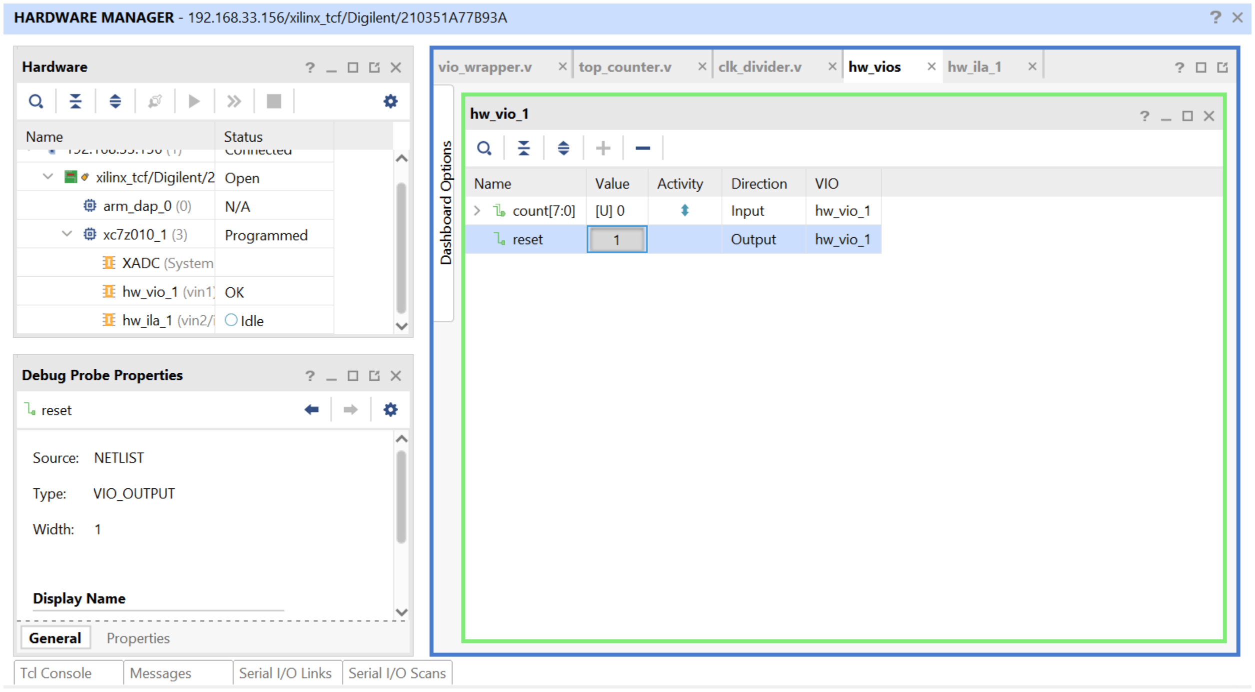


```
# Clock signal
set_property PACKAGE_PIN W5 [get_ports clk]
    set_property IOSTANDARD LVCMOS33 [get_ports clk]
    create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports clk]
```

8. Generate the bitstream.
9. Connect to the remote device using the steps mentioned in Lab 2.
10. Program the device. Once the programming is done,refresh the VIO probes.

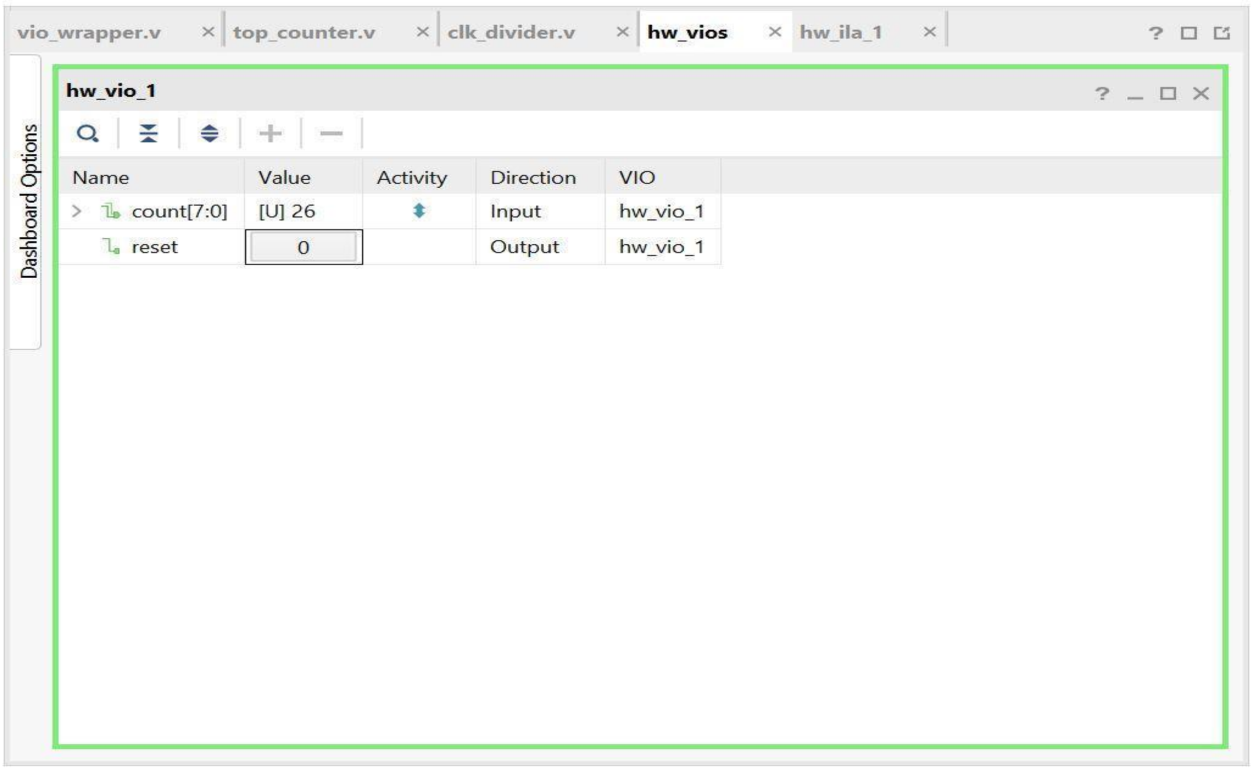


11. Add the VIO probes. Set the reset high, and you can see count is not incrementing as shown below



Name	Value	Activity	Direction	VIO
count[7:0]	[U] 0	⬍	Input	hw_vio_1
reset	1		Output	hw_vio_1

12. De-assert the reset, and you can see the Counter incrementing from 0 to 255 and back to 0.

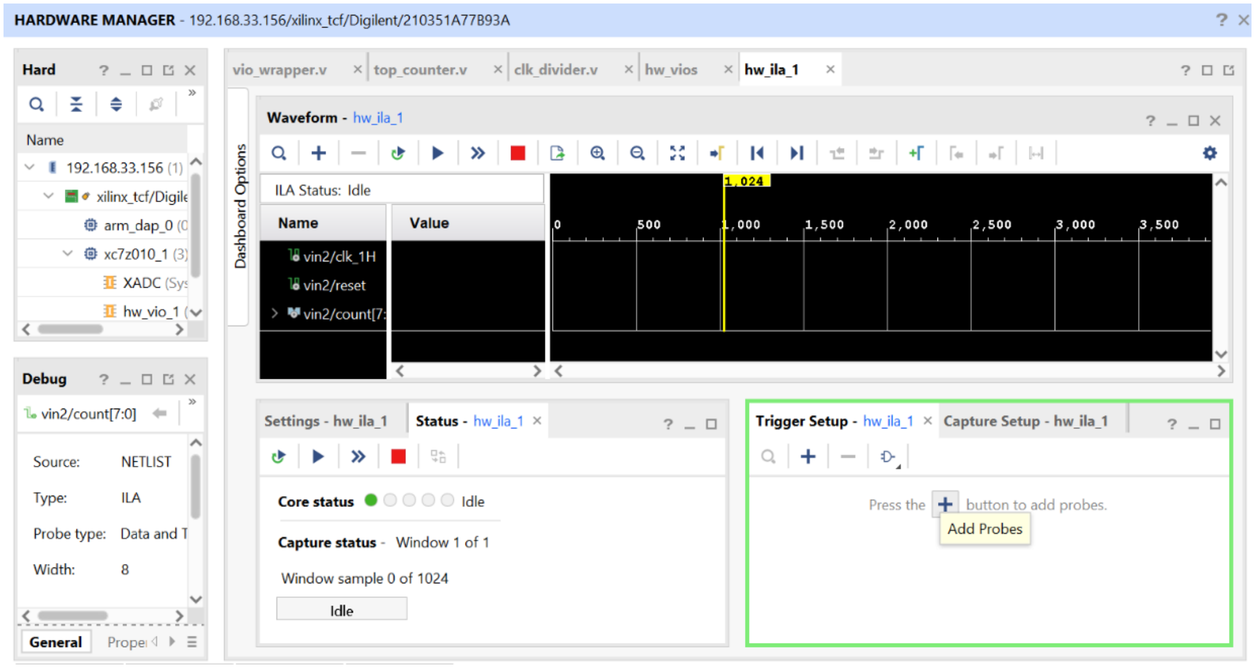

vio_wrapper.v | top_counter.v | clk_divider.v | hw_vios | hw_ila_1

Dashboard Options

hw_vio_1

Name	Value	Activity	Direction	VIO
count[7:0]	[U] 26	⬍	Input	hw_vio_1
reset	0		Output	hw_vio_1

13. Next step is to make use of ILA and trigger it in two different conditions
    - When the count reaches 15.
    - When the reset goes high.


HARDWARE MANAGER - 192.168.33.156/xilinx_tcf/Digilent/210351A77B93A

Hard

Name
- 192.168.33.156 (1)
  - xilinx_tcf/Digile
    - arm_dap_0 (0
    - xc7z010_1 (3)
      - XADC (Sys
      - hw_vio_1 (

Debug

vin2/count[7:0]

Source:	NETLIST
Type:	ILA
Probe type:	Data and T
Width:	8

General | Prope

vio_wrapper.v | top_counter.v | clk_divider.v | hw_vios | hw_ila_1

Dashboard Options

Waveform - hw_ila_1

ILA Status: Idle

Name	Value
vin2/clk_1H	
vin2/reset	
vin2/count[7:	

Settings - hw_ila_1 | Status - hw_ila_1

Core status Idle

Capture status - Window 1 of 1

Window sample 0 of 1024

Idle

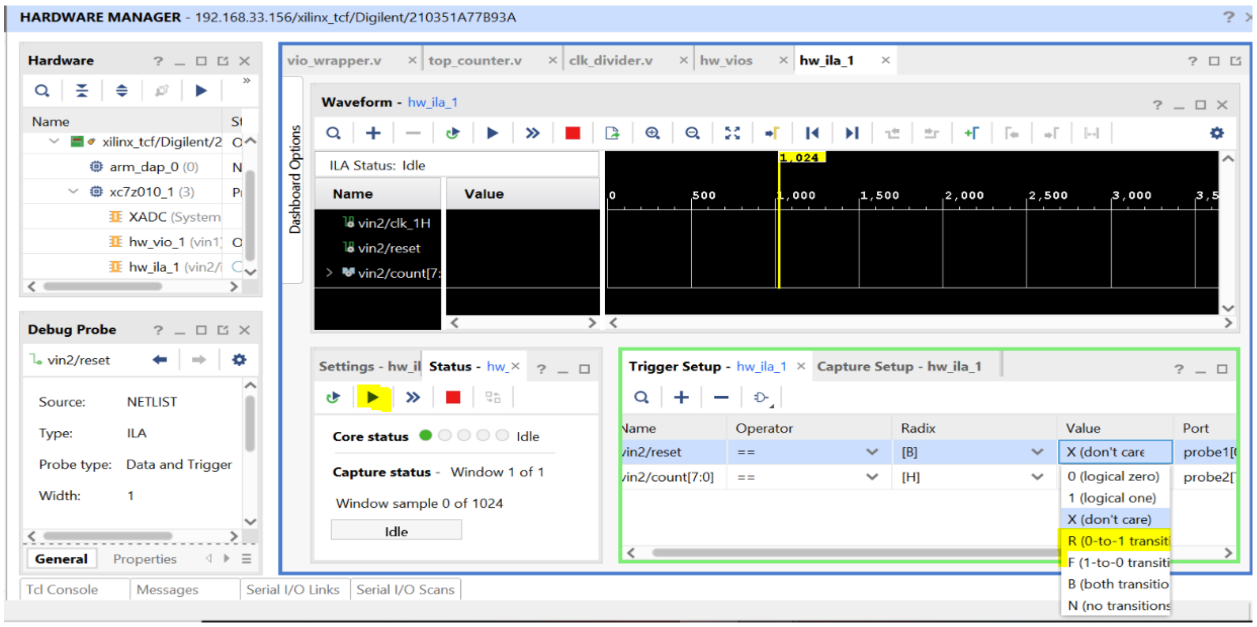
Trigger Setup - hw_ila_1 | Capture Setup - hw_ila_1

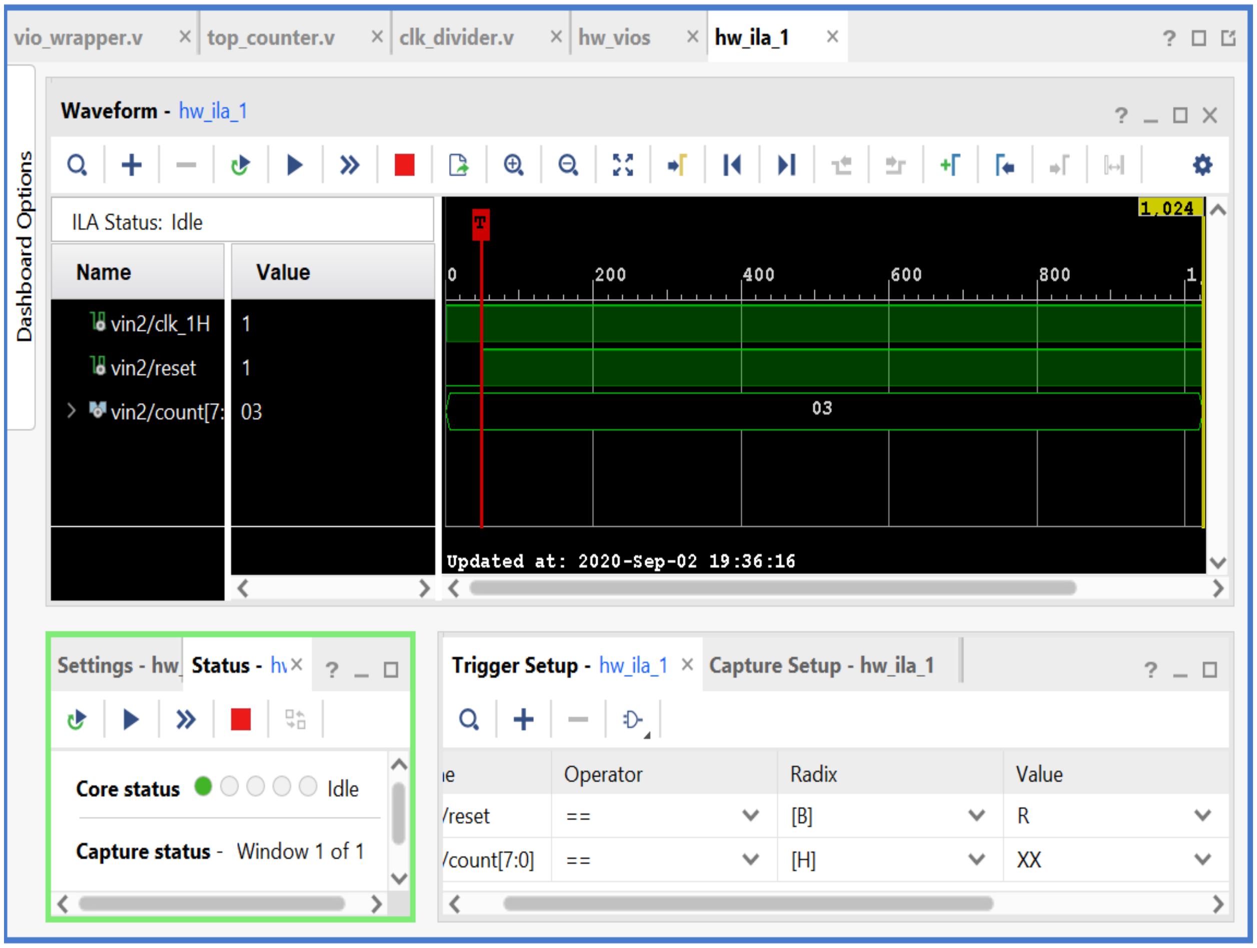
Press the + button to add probes.

Add Probes

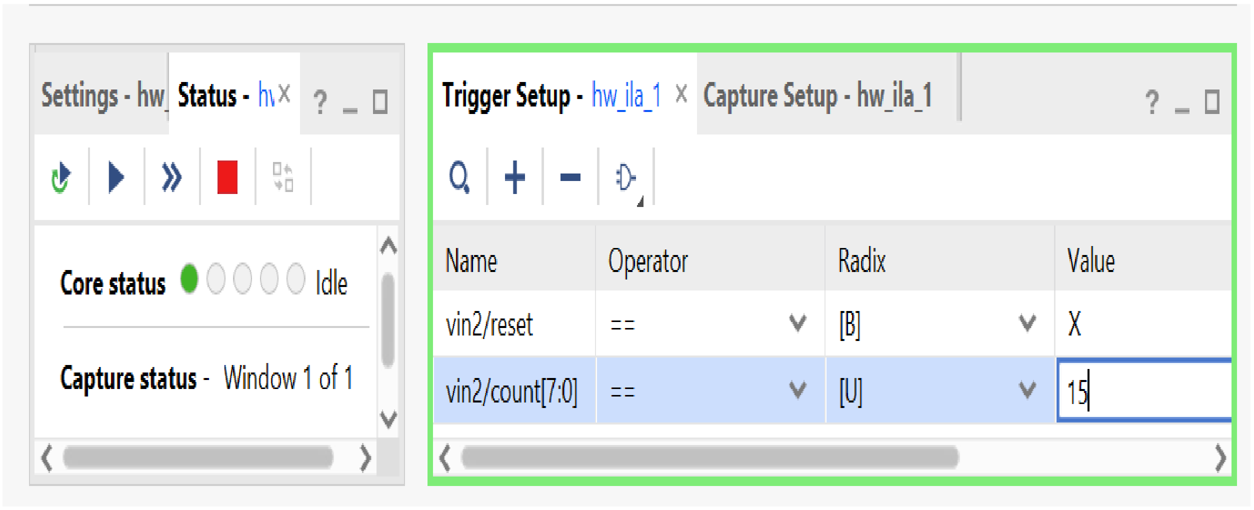
Click on add probes and add both reset and count probes.

14. Select the trigger condition for reset as 0-1 transition and click on the play button, which will create the trigger. Now from VIO, assert the reset and check that the ILA must have captured the event.



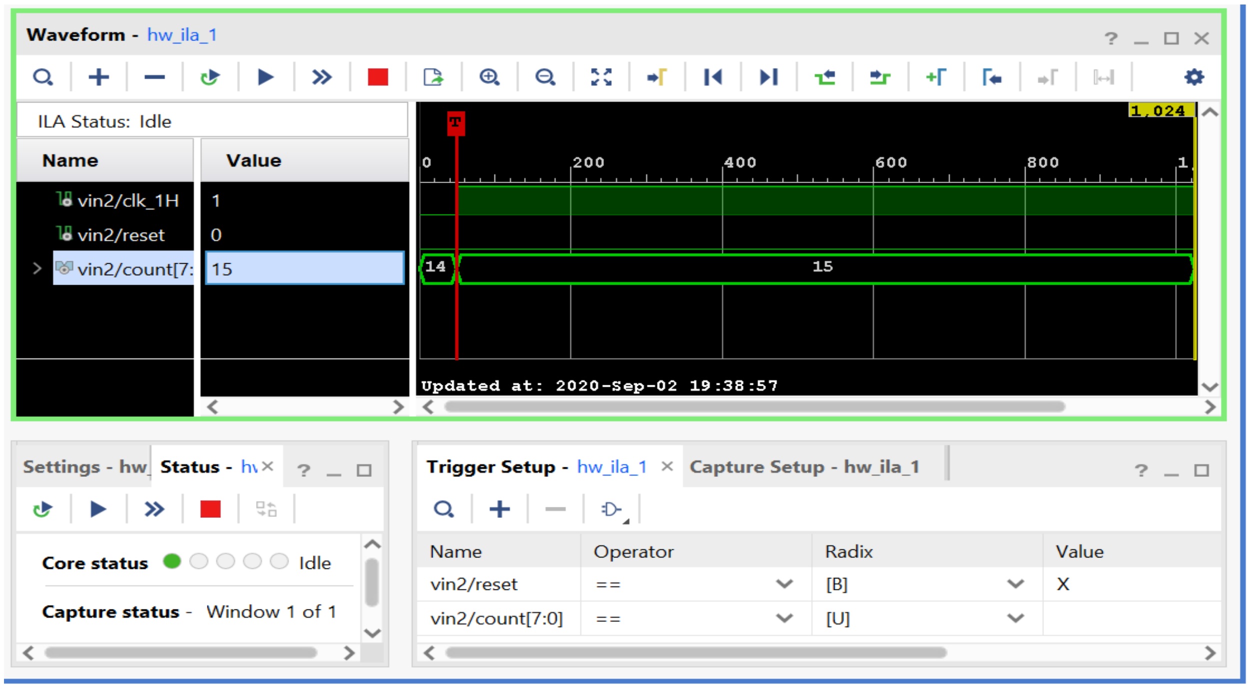


Next, we need to create a trigger when the count reaches 15. Make the following changes.



Name	Operator	Radix	Value
vin2/reset	==	[B]	X
vin2/count[7:0]	==	[U]	15

After making the changes, click on the Run trigger(blue play button). Once the count reaches 15, ILA will catch the event and show it on the wave window as shown below.



Name	Value
vin2/clk_1H	1
vin2/reset	0
vin2/count[7:	15

```
Updated at: 2020-Sep-02 19:38:57
```

Name	Operator	Radix	Value
vin2/reset	==	[B]	X
vin2/count[7:0]	==	[U]	

You can explore many options by creating composite conditions (logical AND, logical OR, etc.) using the gate icon.

Deliverables for Lab 3 Submission(Graded):

1. .bit file and .ltx file of the design
2. PDF with code, testbench, simulation screenshot, VIO, and ILA output screenshot for the design.