Lab 6

Tasks to be done in this Lab:

1. Design and implement a ROM using Block Memory Generator. Create a coe file to store the 10 numbers of 4 bits each and find the maximum amongst them.
2. Design and implement a FIFO (common block RAM) capable of storing 16 numbers of 4 bits each. Provide the input data using a 4-bit input port. Use two separate push buttons to read and write. Also, see the working of empty, full,almost_empty,almost_full, and data_count signals.

Part -1

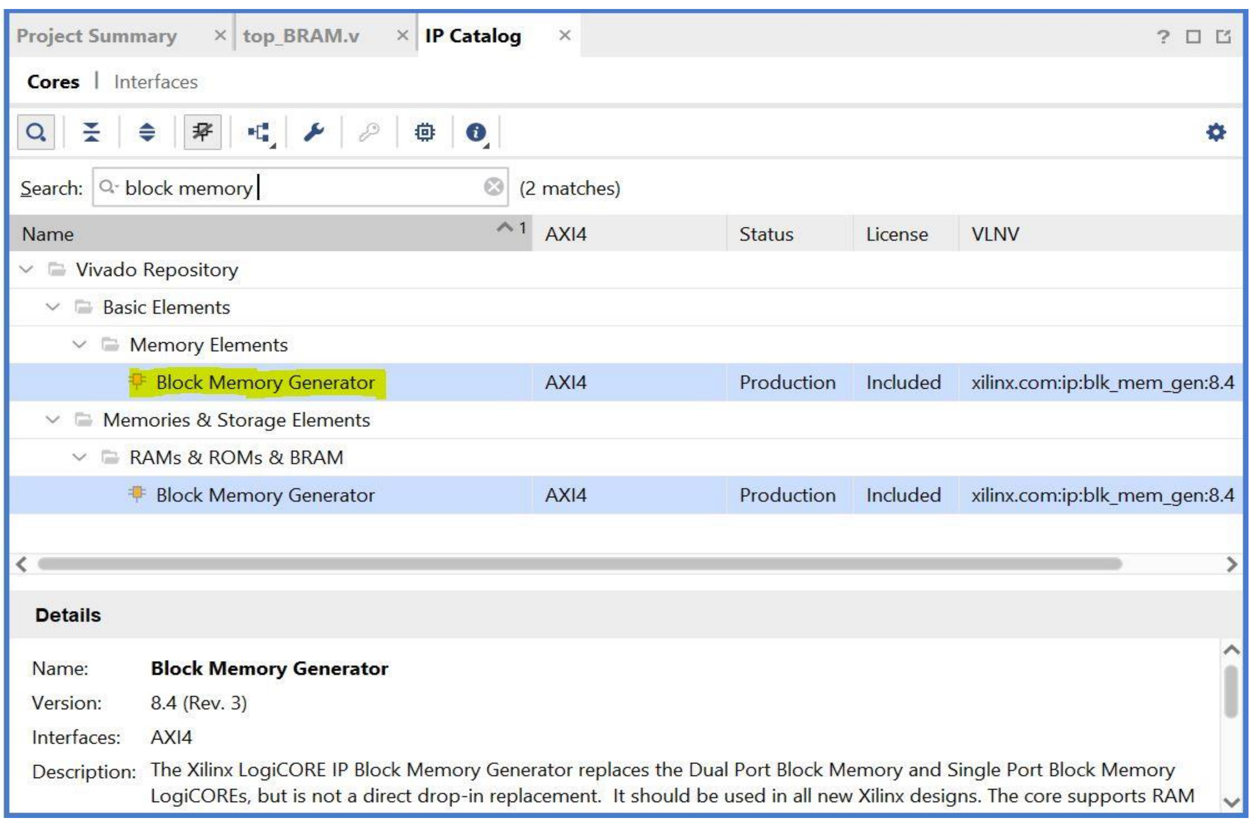
In this part, we will learn about how to implement a ROM using the Block Memory Generator IP and look at how we can customize the IP according to our needs.

Step-1

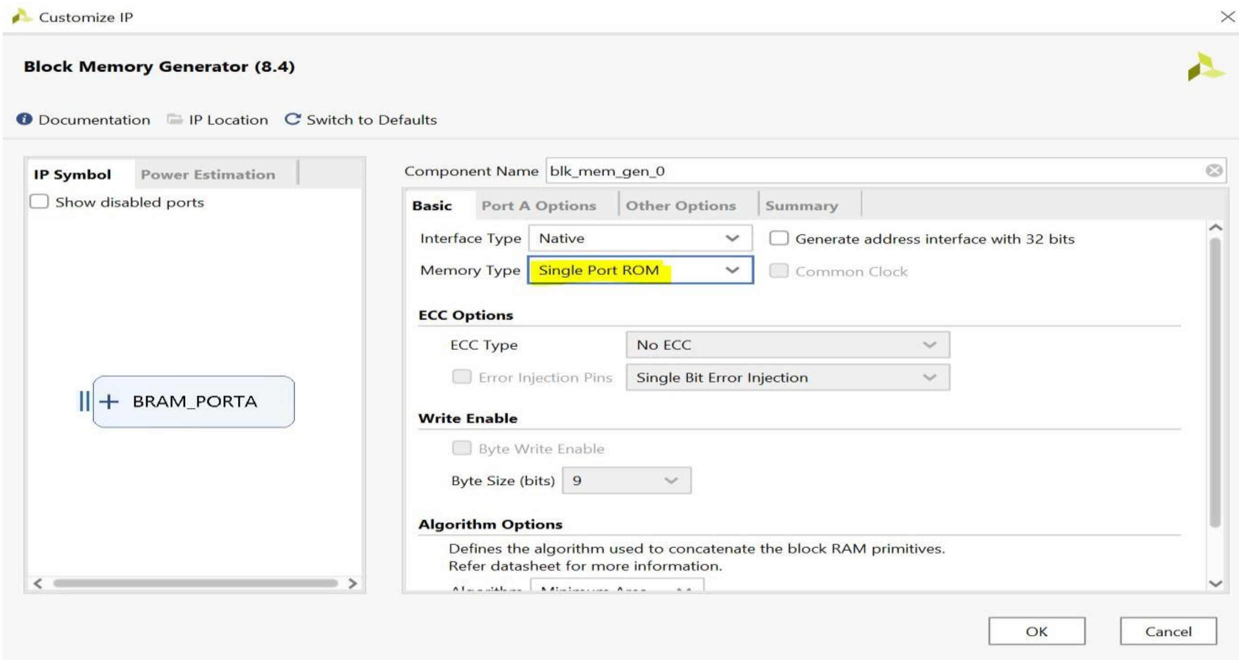
Create a top module top_BRAM with two ports. One input port of 1 bit for the clock and one output port of 4 bits to get the maximum number among all the numbers stored in the ROM.

Step-2

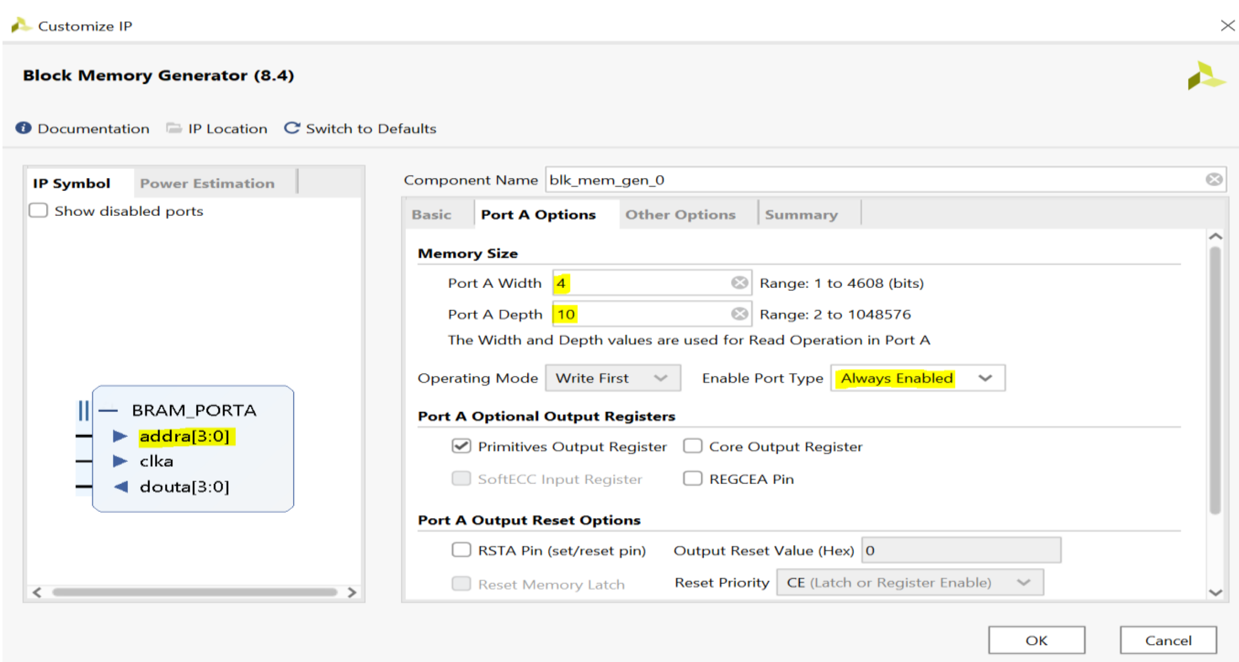
- Add the Block Memory Generator IP.



- In the Memory type, select “Single Port ROM” and keep the rest of the options as default.



- In the Port A Options you need to make three changes:
  - Width of the port→ This field is used to define the width of the word to be stored. In our case, it is 4 bits.
  - Depth of the Port→ This field is used to define the number of locations to be allotted. In our case, as we need to store 10 words. The address lines assigned will be the multiple of 2, and hence we can see that the address bus is of 4 bits.
  - Select the always enabled option for “Enable Port Type.”



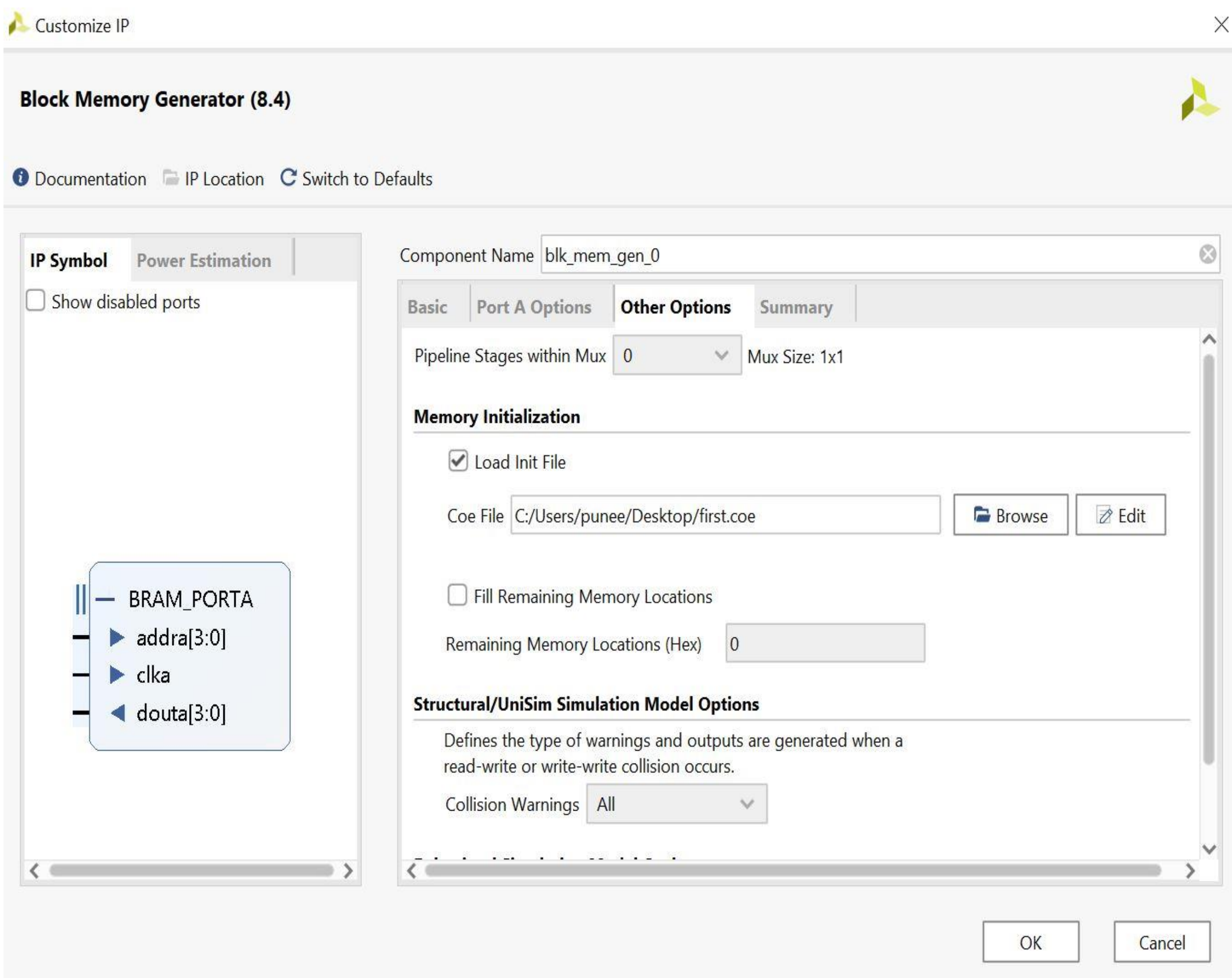
- What are the signals addra,clka,douta?

  addra is the address bus using which we can access a particular memory location. For example, a value “0000” at the bus means that we need to access the 0^{th} location.
  clka is the clock signal at which the memory will be operating
  douta is the data accessed at a particular location using addra.

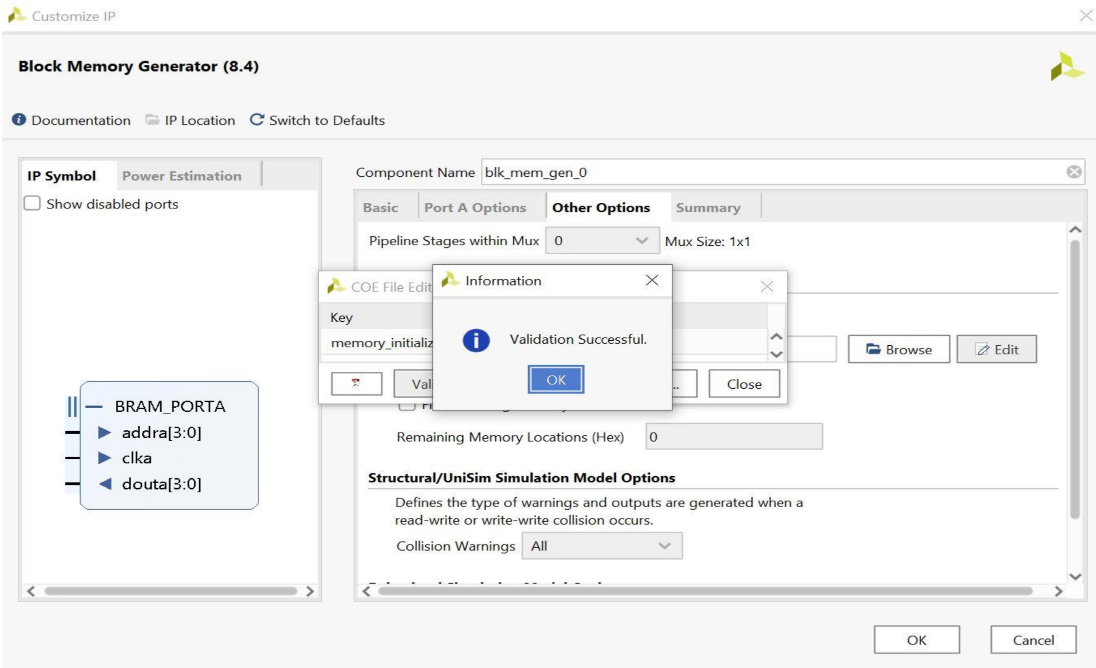
- Navigate to the “Other Options” Window. Click on the load init file. Using this option, we can load the data in memory. For this, create a file with .coe extension with the following content.

```
memory_initialization_radix=16;
memory_initialization_vector=a b c d e 1 f 3 2 1;
```

  Using the browse option, navigate to the file, and open it.



Once the file is loaded, click on edit and validate the file. If everything is fine, you will get a successful validation message.



- Finally, click on OK to synthesize the IP.

Step-3

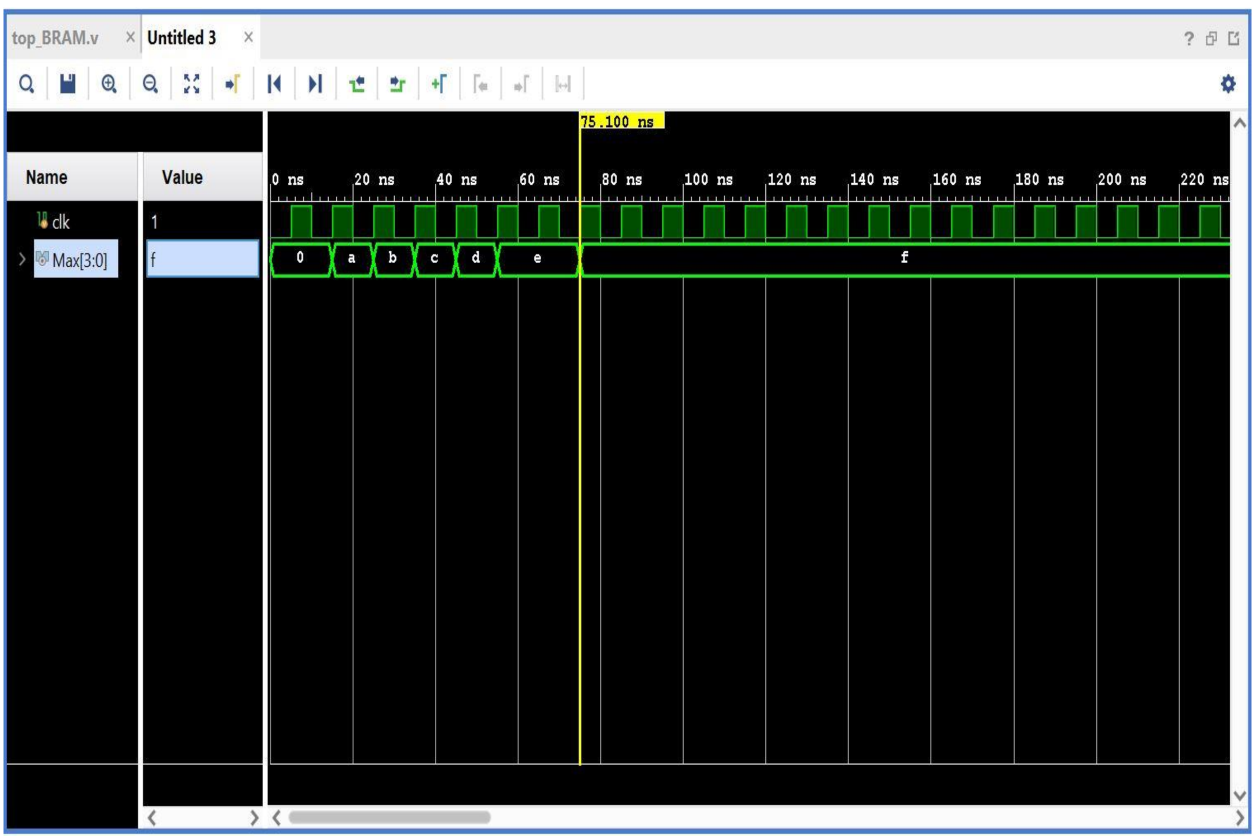
Instantiate the BRAM IP in the top module and the code to find the maximum number in the first 10 locations.

```
module top_BRAM(
    input clk,
    output reg [3:0] Max=0
    );

    wire [3:0] dout;
    reg [3:0] addr_reg = 0,addr_next;

    always@(posedge clk)
    begin
         addr_reg <= addr_next;
    end

    always@(*)
    begin
        if (addr_reg == 9)
        addr_next = addr_reg;
        else
        addr_next = addr_reg+1;
    end

         blkROM in1
         (
          .clka(clk),     // input wire clka
          .addra(addr_reg),  // input wire [3 : 0] addra
          .douta(dout)   // output wire [3 : 0] douta
         );
```

```
    always @(*)
        begin
            if(dout>Max)
                Max=dout;
            else
                Max=Max;
        end


endmodule
```

Step 4: Test the functionality of your code using the Testbench given below

```
module bram_tb(

    );

    reg clk=0;
    wire [3:0] Max;

    top_BRAM tb1(.clk(clk),.Max(Max));

    always #5 clk=~clk;
endmodule
```



Part-2

In this part, we will learn how to implement a FIFO Memory using FIFO Generator IP with given size and customize it. We will also look into various signals and their use associated with FIFO.

Design Details:

- Build a FIFO Memory using the FIFO generator IP, having 16 memory locations of 4 bits each.
- Both Read and Write can be performed on the FIFO using two signals, “read” and “write” generated through push buttons.
- The Data to be written in the FIFO will be coming through the 4-bits input port “din”.
- To do a read operation, press the Read button, and the corresponding data should be visible on the “dout” port, which is an output port.
- To do a write operation, set the Data on the “din” port and press “write”. The Data on the Din port must be written FIFO on the press of button.
- Observe and interpret the use of various signals associated with the FIFO, namely empty, full,almost_empty,almost_full, and data_count.

Step-1

Create the top module with the ports as shown below:

```
module top_FIFO(
    input clk_125M,
    input reset,
    input read,
    input write,
    input [3:0] din,
    output [3:0] dout,
    output full,
    output empty,
    output almost_full,
    output almost_empty,
    output [3:0] data_count
    );
```

Step-2

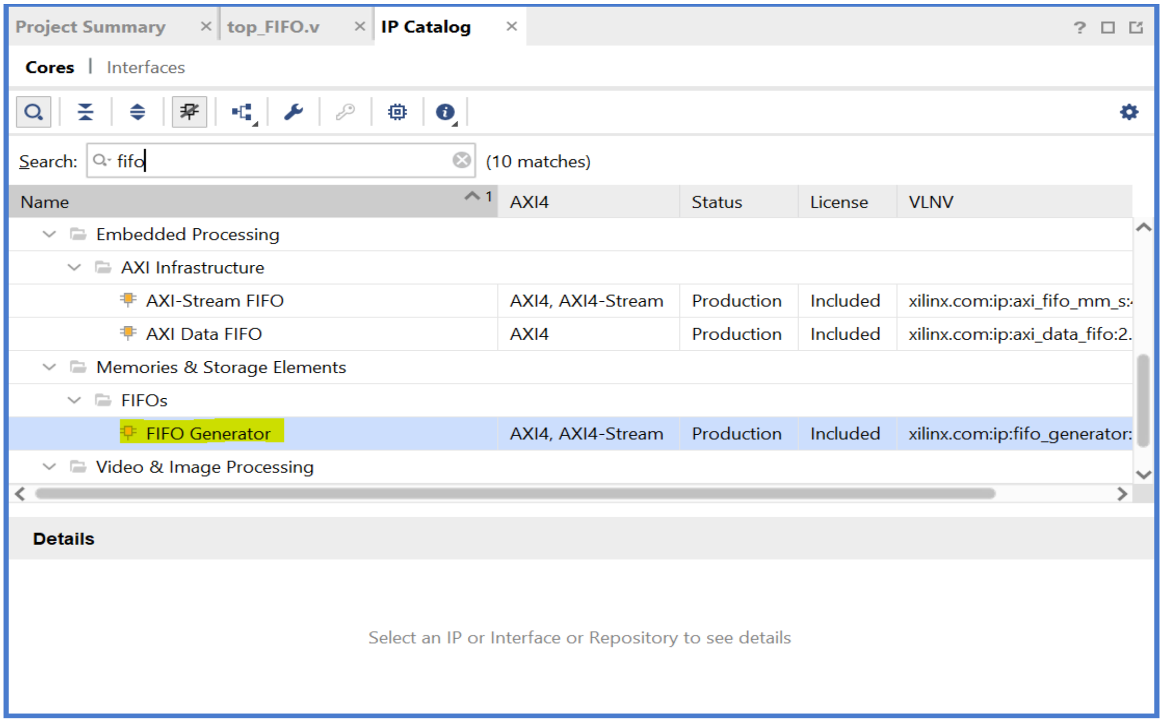
Add the Clocking IP and clock divider module and generate a 200 Hz clock, which will be used in the clock pulse generator for the FIFO Memory. Also, add the clock pulse generator module, which we made in Lab 4.

```
wire clk_5M,clk_200H,clk_pulse;

clkIP in1
(
.clk_out1(clk_5M),      // output clk_out1
.clk_in1(clk_125M)         // input clk_in1
);

clk_divider #(.div_value(12499)) in2(.clk_in(clk_5M),.divided_clk(clk_200H));
clk_pulse in3(.clk_200H(clk_200H),.inp_0(read),.inp_1(write),.clk_pulse(clk_pulse));
```

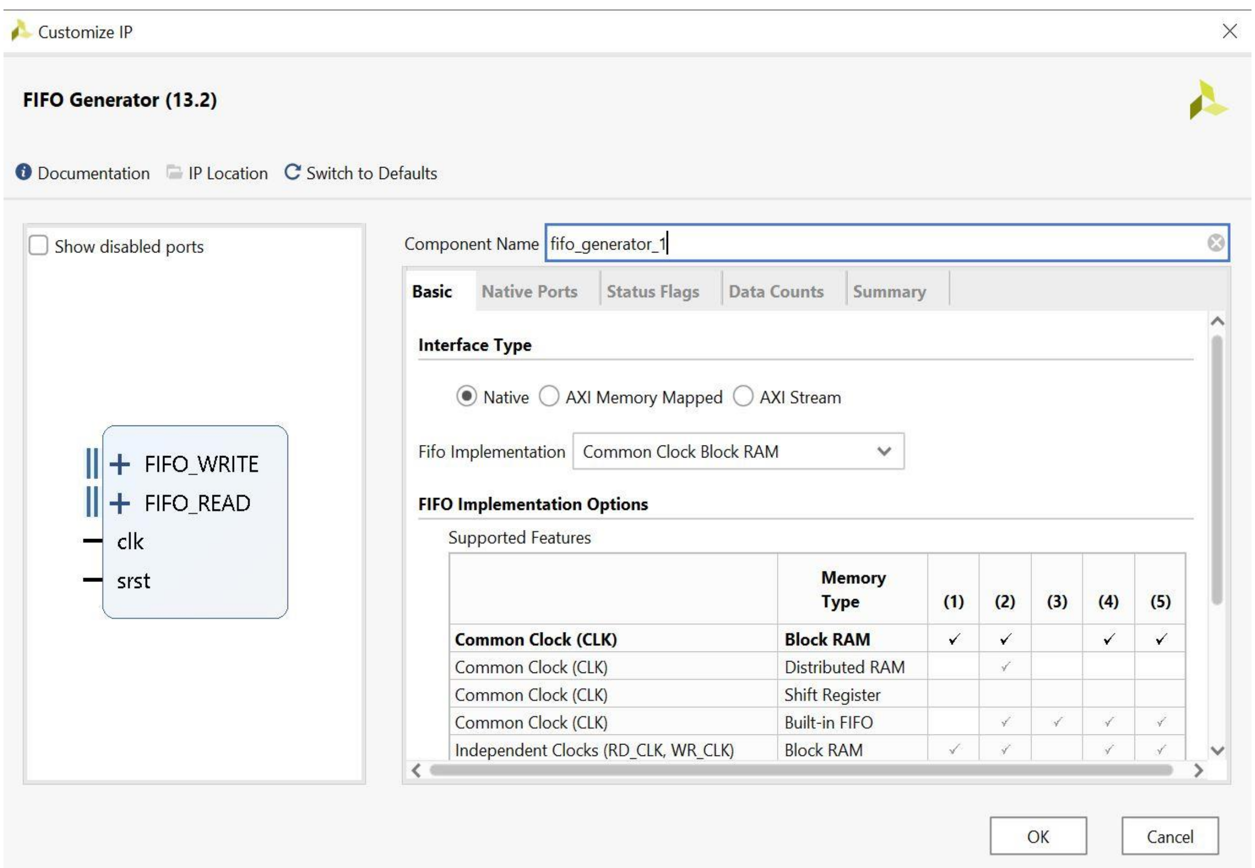
The main idea behind using the clock pulse generator is that whenever either the read or write button is pressed, the clock should get generated for the FIFO Memory.

Step-3

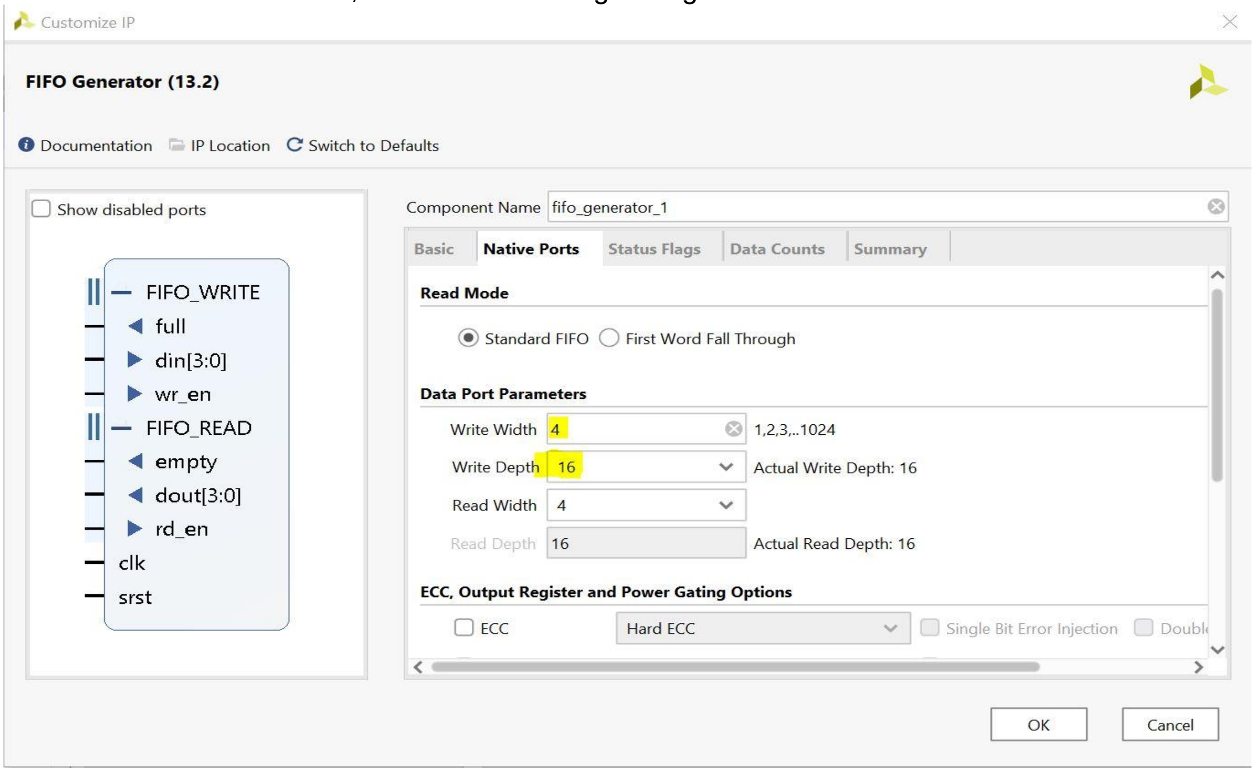
- Add and customize the FIFO Generator IP.



- Nothing needs to be changed in the Basic Window. Keep the default options.

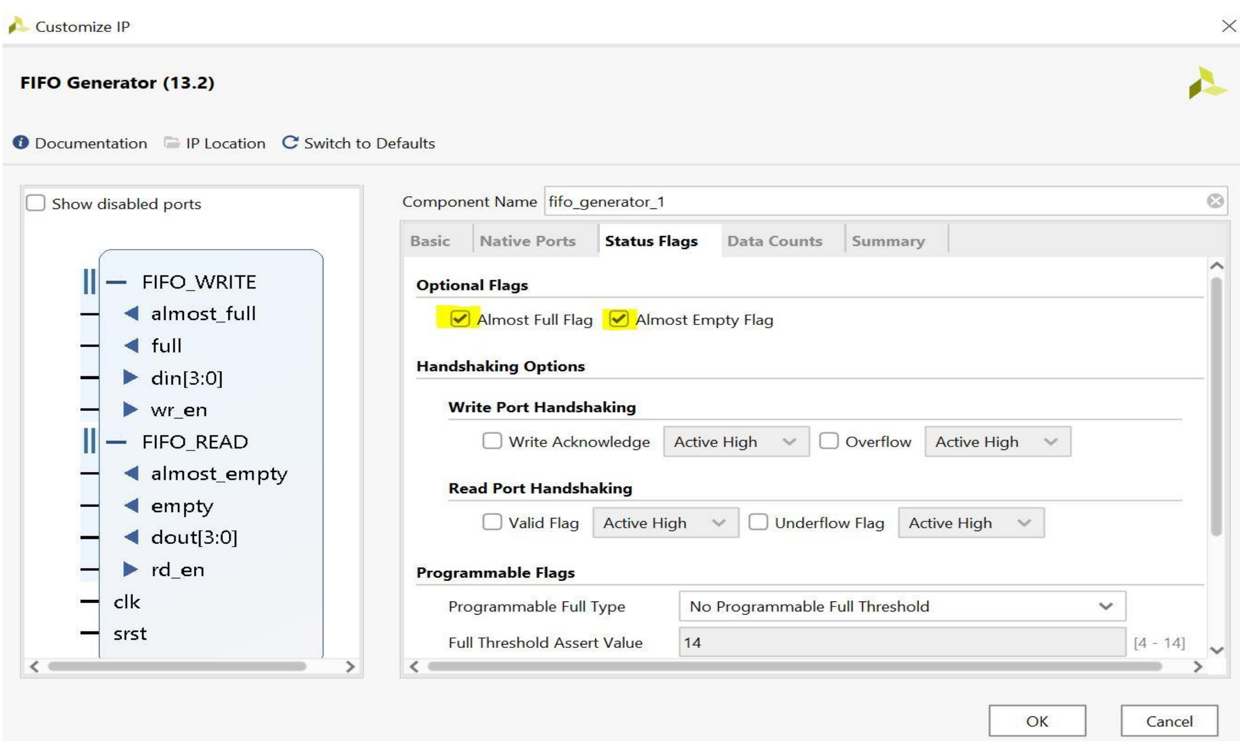


- In the Native Ports window, make the following changes:

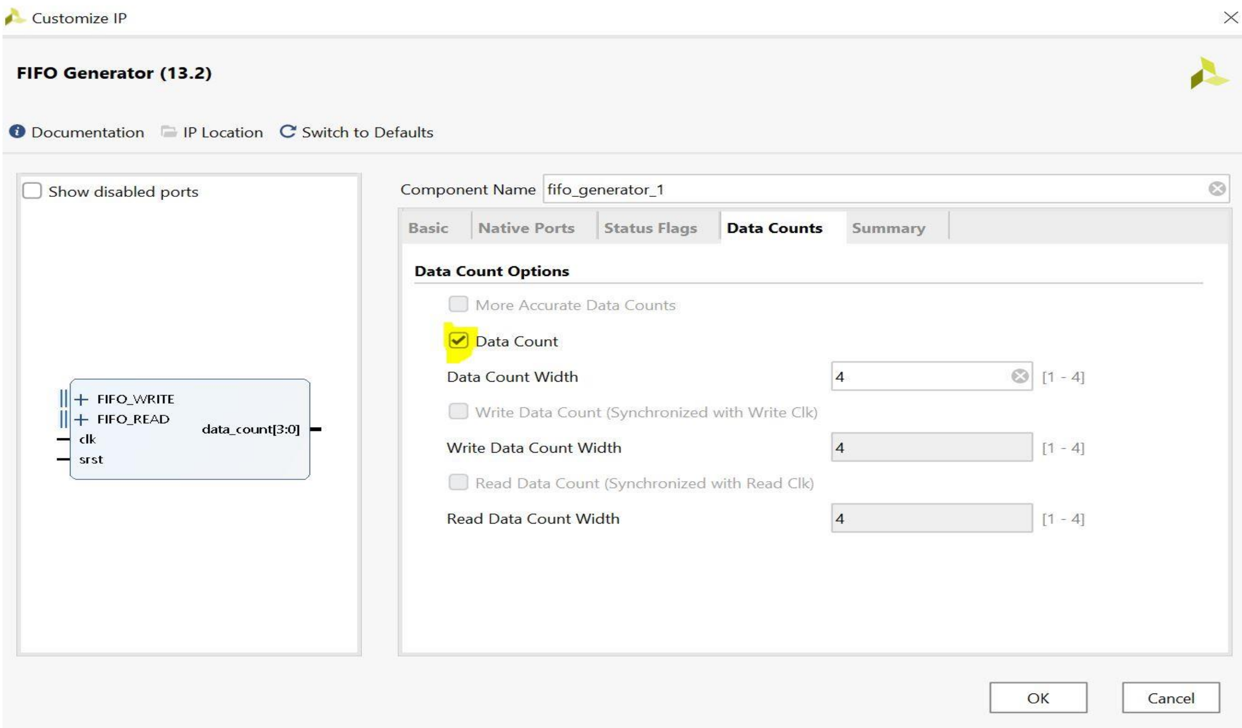


Write width is set to 4 bits as the word length is 4 bits, and we need 16 locations on the FIFO memory, and hence write depth is set to 16.

- In the Status flag window, tick the almost full and almost empty flag.



- In the Data Counts Window, tick the data count option and click OK to synthesize the IP.



Step-4

Instantiate the FIFO IP. Afterward, your top module should look like the one shown below.

```
module top_FIFO(
    input clk_125M,
    input reset,
    input read,
    input write,
    input [3:0] din,
    output [3:0] dout,
    output full,
    output empty,
    output almost_full,
    output almost_empty,
    output [3:0] data_count
    );

    wire clk_5M,clk_200H,clk_pulse;

    clkIP in1
   (
    .clk_out1(clk_5M),      // output clk_out1
    .clk_in1(clk_125M)        // input clk_in1
    );

    clk_divider #(.div_value(12499)) in2(.clk_in(clk_5M),.divided_clk(clk_200H));
    clk_pulse in3(.clk_200H(clk_200H),.inp_0(read),.inp_1(write),.clk_pulse(clk_pulse));
```

```
  fifo_generator_0 in4 (
 .clk(clk_pulse),                   // input wire clk
 .srst(reset),                 // input wire srst
 .din(din),                    // input wire [3 : 0] din
 .wr_en(write),                // input wire wr_en
 .rd_en(read),                // input wire rd_en
 .dout(dout),                  // output wire [3 : 0] dout
 .full(full),                  // output wire full
 .almost_full(almost_full),    // output wire almost_full
 .empty(empty),                // output wire empty
 .almost_empty(almost_empty),  // output wire almost_empty
 .data_count(data_count)      // output wire [3 : 0] data_count
);
endmodule
```

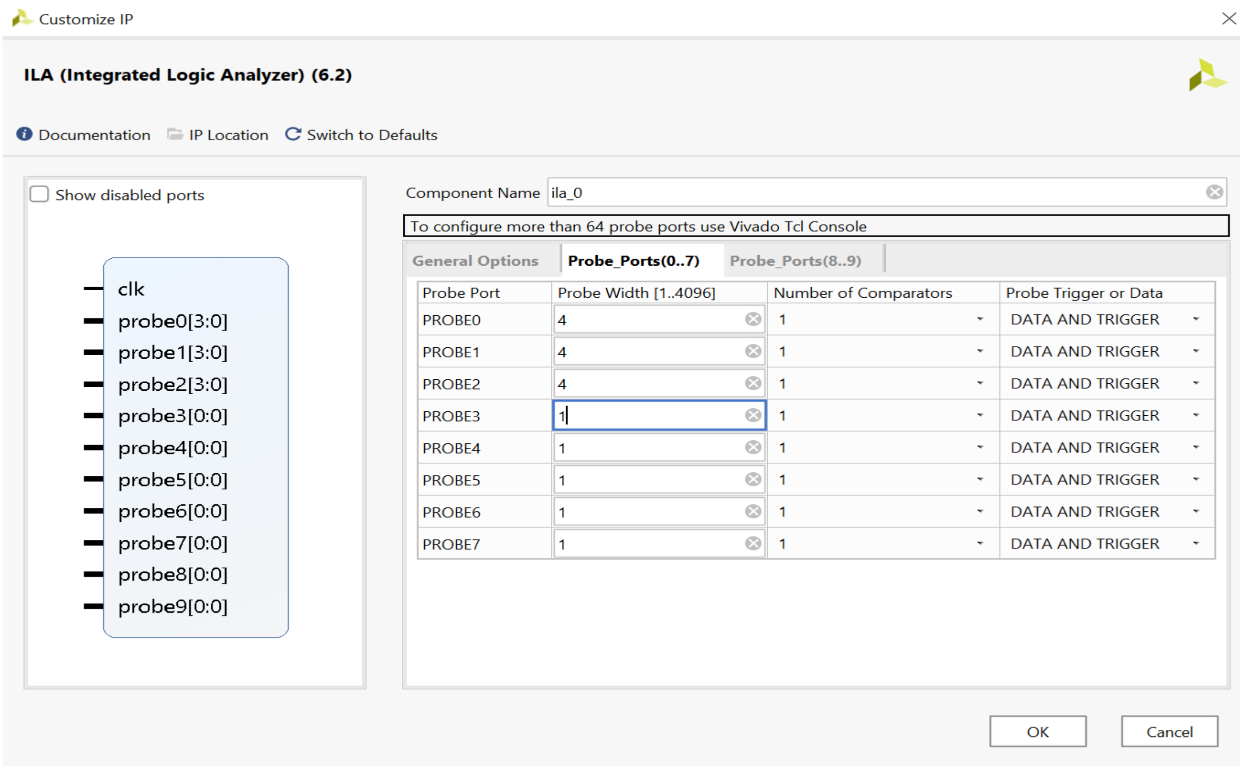
Step-5

Next, we will add the VIO and ILA IP. We will use the VIO IP to read and write into the FIFO memory and monitor the various signals associated with the FIFO Memory. Using ILA IP, we will look into other features of ILA.

Add a VIO IP and create a vio wrapper and instantiate the VIO IP and top module in it. For the various probes of VIO, please refer to the following code of the vio wrapper.

```
module vio_wrapper(
    input clk
    );

    wire reset, read, write,full,empty,almost_full,almost_empty;
    wire [3:0] din,dout,data_count;


    vio_0 vin1 (
  .clk(clk),                // input wire clk
  .probe_in0(dout),     // input wire [3 : 0] probe_in0
  .probe_in1(data_count),    // input wire [3 : 0] probe_in1
  .probe_in2(full),     // input wire [0 : 0] probe_in2
  .probe_in3(almost_full),     // input wire [0 : 0] probe_in3
  .probe_in4(empty),     // input wire [0 : 0] probe_in4
  .probe_in5(almost_empty),     // input wire [0 : 0] probe_in5
  .probe_out0(din),  // output wire [3 : 0] probe_out0
  .probe_out1(reset),  // output wire [0 : 0] probe_out1
  .probe_out2(read),  // output wire [0 : 0] probe_out2
  .probe_out3(write)  // output wire [0 : 0] probe_out3
);

top_FIFO vin2(.clk_125M(clk),.reset(reset),.read(read),.write(write),.din(din),
                    .full(full),.empty(empty),.almost_full(almost_full),.almost_empty(almost_empty),
                    .dout(dout),.data_count(data_count));
endmodule
```

Add the ILA IP with ten probes with 3 ports of 4-bits.

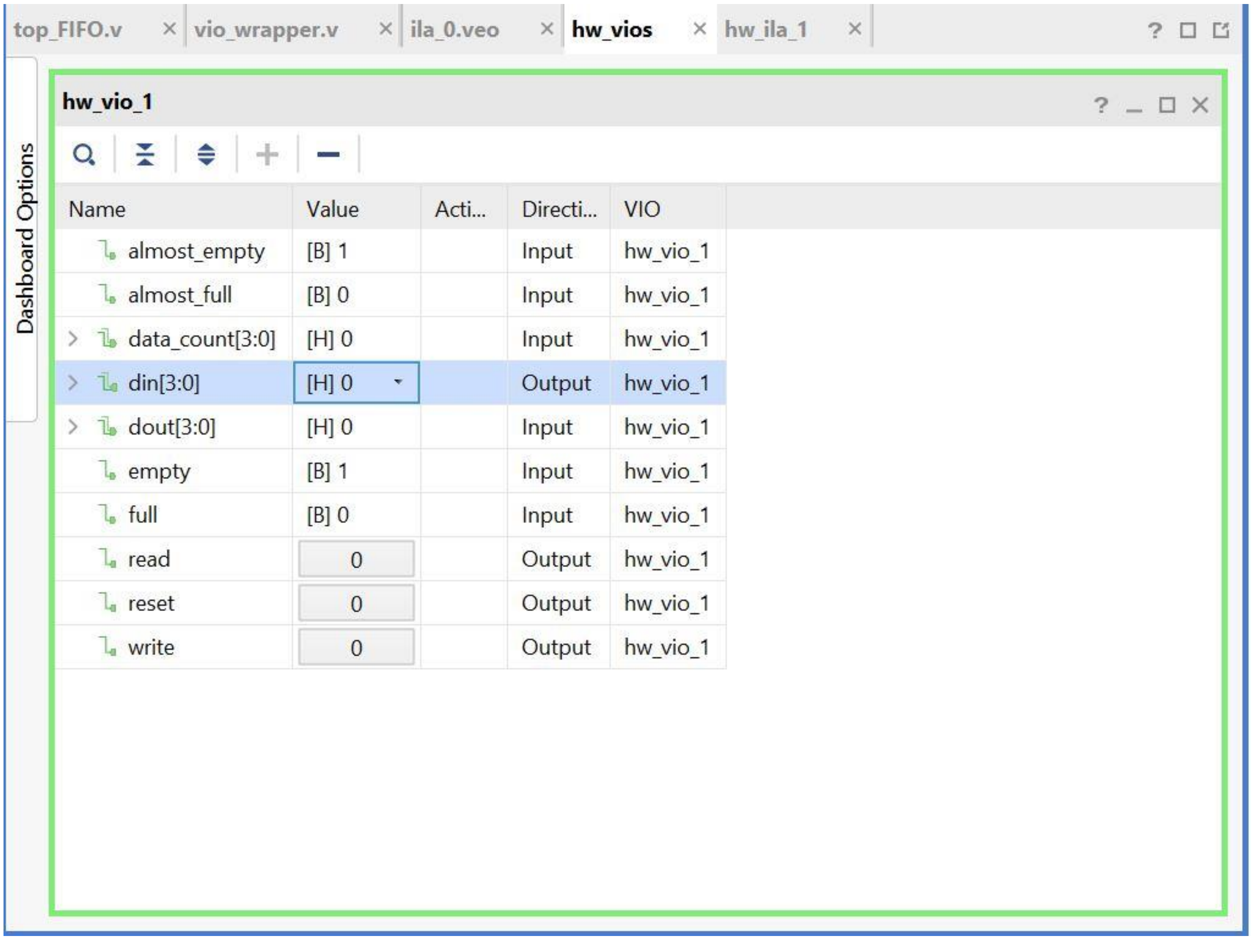


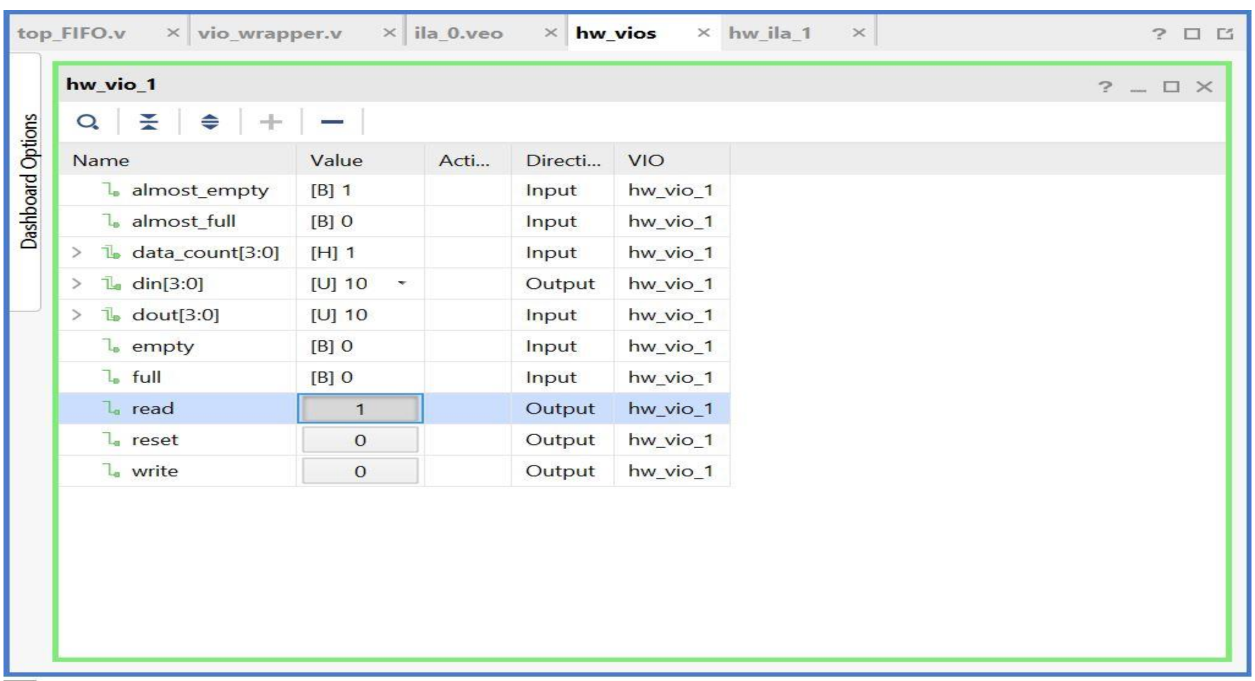
Instantiate the ILA IP in the top module top_FIFO (not in vio_wrapper).

```
ila_0 in5 (
     .clk(clk_125M), // input wire clk


     .probe0(din), // input wire [3:0]  probe0
     .probe1(dout), // input wire [3:0]  probe1
     .probe2(data_count), // input wire [3:0]  probe2
     .probe3(read), // input wire [0:0]  probe3
     .probe4(write), // input wire [0:0]  probe4
     .probe5(reset), // input wire [0:0]  probe5
     .probe6(full), // input wire [0:0]  probe6
     .probe7(almost_full), // input wire [0:0]  probe7
     .probe8(empty), // input wire [0:0]  probe8
     .probe9(almost_empty) // input wire [0:0]  probe9
);
```

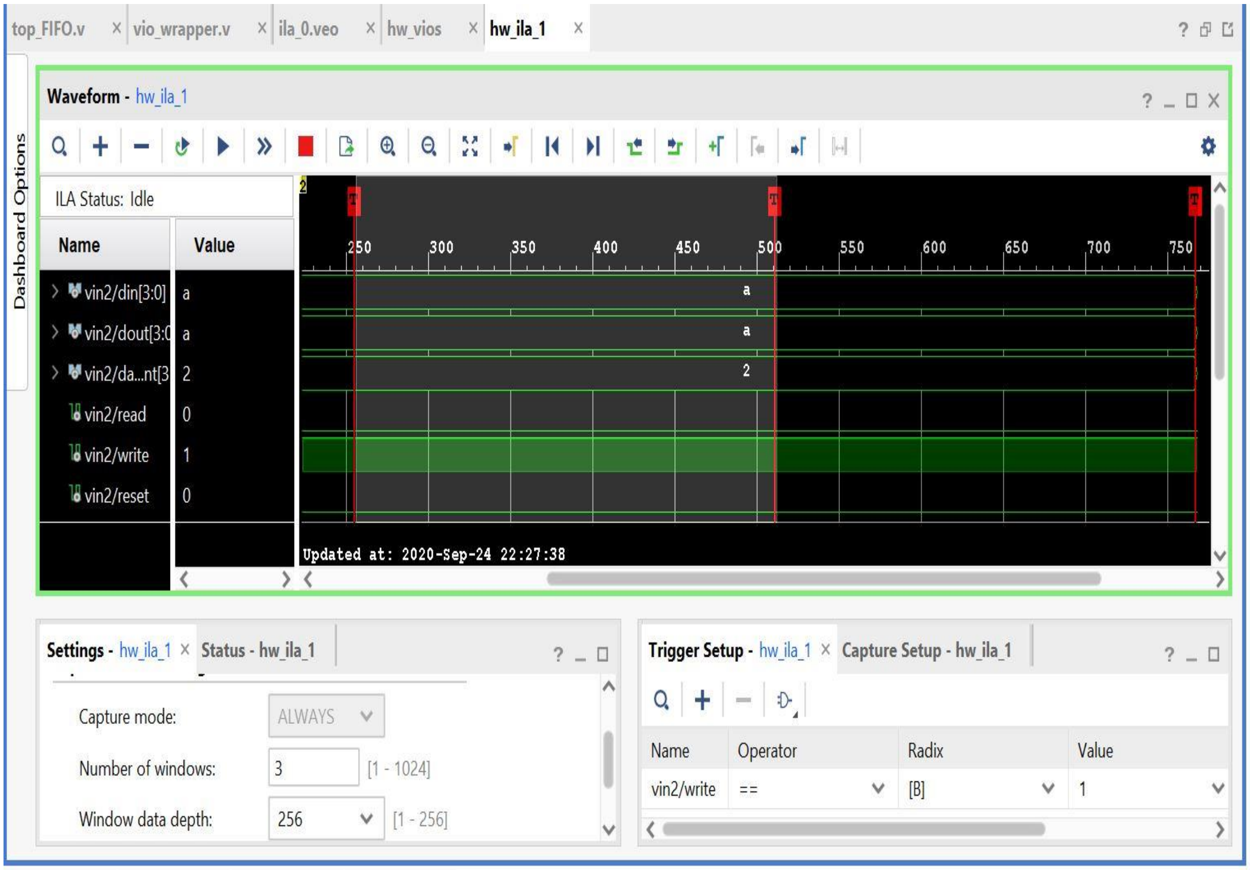
Step-6

Add the constraints file, generate the bitstream and connect to the remote hardware and program the device.


top_FIFO.v
vio_wrapper.v
ila_0.veo
hw_vios
hw_ila_1
Dashboard Options
hw_vio_1
Name Value Acti... Directi... VIO
almost_empty [B] 1 Input hw_vio_1
almost_full [B] 0 Input hw_vio_1
data_count[3:0] [H] 0 Input hw_vio_1
din[3:0] [H] 0 Output hw_vio_1
dout[3:0] [H] 0 Input hw_vio_1
empty [B] 1 Input hw_vio_1
full [B] 0 Input hw_vio_1
read 0 Output hw_vio_1
reset 0 Output hw_vio_1
write 0 Output hw_vio_1


top_FIFO.v
vio_wrapper.v
ila_0.veo
hw_vios
hw_ila_1
Dashboard Options
hw_vio_1
Name Value Acti... Directi... VIO
almost_empty [B] 1 Input hw_vio_1
almost_full [B] 0 Input hw_vio_1
data_count[3:0] [H] 1 Input hw_vio_1
din[3:0] [U] 10 Output hw_vio_1
dout[3:0] [U] 10 Input hw_vio_1
empty [B] 0 Input hw_vio_1
full [B] 0 Input hw_vio_1
read 1 Output hw_vio_1
reset 0 Output hw_vio_1
write 0 Output hw_vio_1

In the ILA settings, choose the number of windows more than one and also play around with the data depth and then run trigger for some event and see the differences. More on this will be explained in the Lab tutorial video.



Deliverables for Lab 6 Submission(Graded):

1. PDF with code, Testbench, simulation screenshot for Part 1
2. Code, VIO output screenshots, and ILA Screenshot in the same PDF for Part 2.
3. .bit and .ltx file for Part 2 and create a zip folder with PDF.