EC5611

VLSI Laboratory

EXPERIMENTAL RECORD

Submitted by

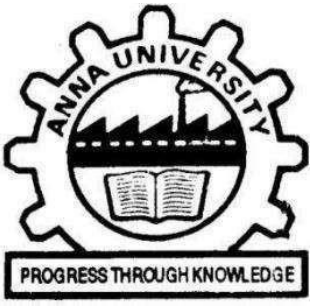
AKASHKUMAR R- 2019105507

in partial fulfilment for the award of the degree of

BACHELOR OF ENGINEERING

IN

ELECTRONICS AND COMMUNICATION ENGINEERING



DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING
ANNA UNIVERSITY, CHENNAI

ANNA UNIVERSITY
COLLEGE OF ENGINEERING GUINDY

BONAFIDE CERTIFICATE

NAME: AKASHKUMAR R
DEPT: B.E. ECE
REGISTER NUMBER: 2019105507

It is certified that this is the bonafide record of the work done by the above-mentioned student in the **VLSI LAB - EC5611** during the period December 2021 - April 2022.

Signature of Lab-In-Charge

Signature of the Head of the Department

Submitted for the practical exam held on ___/___/_____

Signature of the Internal Examiner

INDEX

S.NO	TITLE OF THE EXPERIMENT	MARKS	SIGN
1.	Realization of adder in BASYS3 FPGA		
2.	Timing Analysis of combinational circuits		
3.	Instantiation of Clock generator using IP catalogue and verifying ALU		
4.	Realisation of state machine in BASYS3 FPGA		
5.	Debugging internal signals of memory in FPGA		
6.	Designing an ASIC using openlane		
7.	Inverter layout using Magic		

EX NO: 01 22/03/2022	REALIZATION OF ADDER IN BASYS3 FPGA

AIM:

To design, implement and realize

i) Half Adder

ii) Full Adder

using Verilog and Basys3 FPGA Board. The simulation results are also obtained

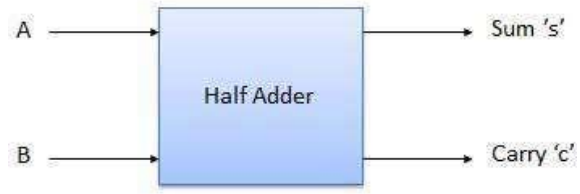
SOFTWARE REQUIRED:

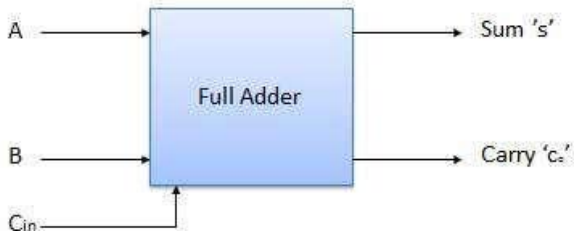
XILINX VIVADO

SOFTWARE REQUIRED:

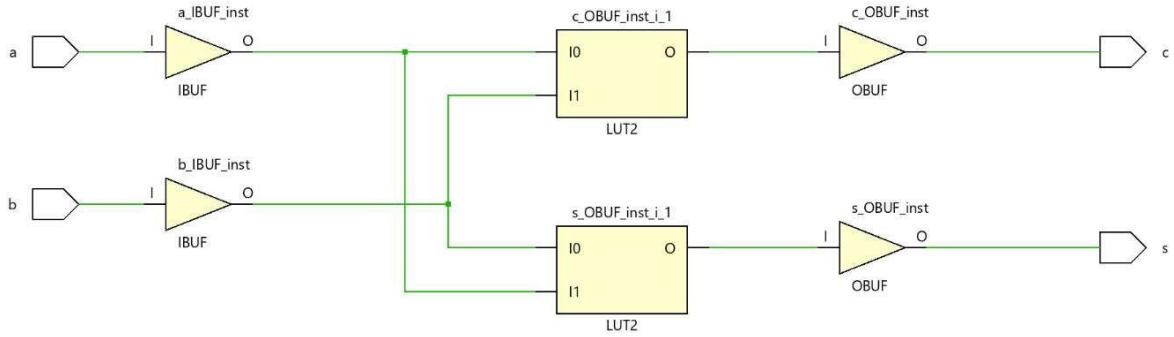
BASYS3 BOARD

BLOCK DIAGRAM:

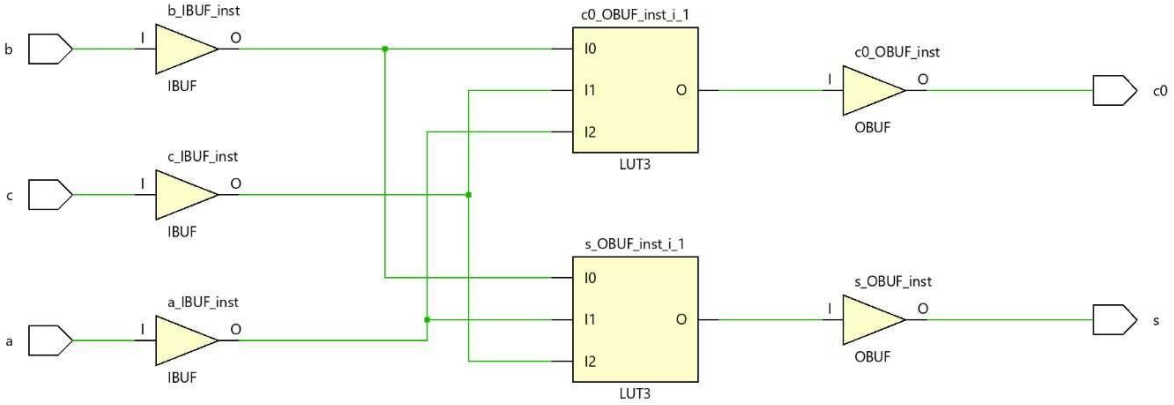




SCHEMATICS



Half Adder



Full Adder

PROCEDURE:

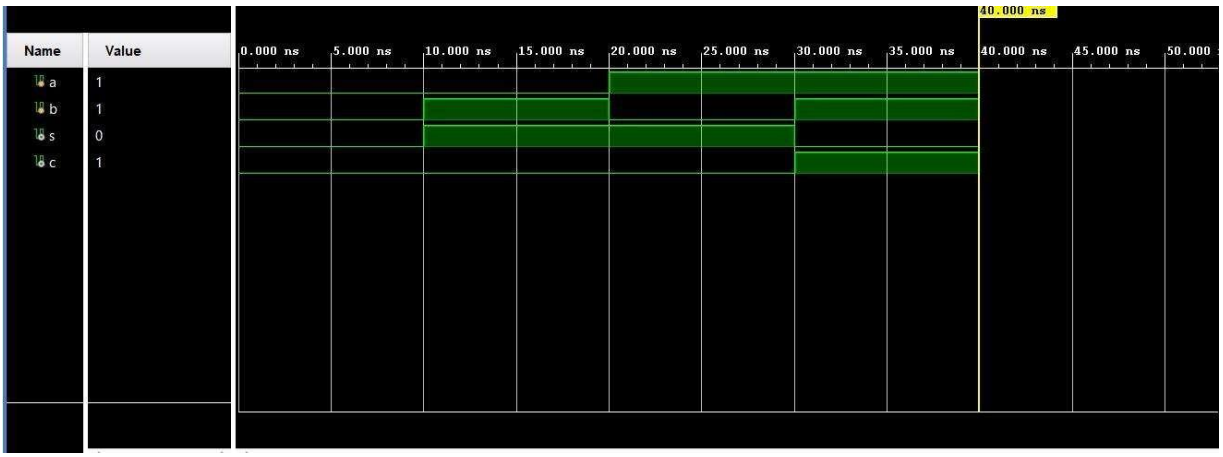
Steps for implementation of the unit in Verilog:

1. Open Vivado. Click Create New Project to start the program. Click Next in the dialog box that appears.
2. Enter Project Name and choose a location
3. Select RTL Project option in the Project Type form, and click Next.
4. Select Verilog as the Target language and Simulator language in the Add Sources form.
5. Choose Basys Board from the boards tab
6. Click Finish to create the Vivado project.
7. To simulate the design using simulator, click Add Sources under the Project Manager tasks of the Flow Navigator pane.
8. Choose "Add or Create Design Sources" to create a design file to write the code for the unit to be realised.
9. Choose "Add or Create Simulation Sources" to create a testbench file for the design file.
10. Save the files after doing the changes to be done and simulate the testbench file.
11. Simulation can be done by choosing "Run Simulation" → "Run Behaviour Simulation".
12. Observe the Timing Diagram
13. To observe the synthesis of this unit, run Synthesis and choose Schematics.
14. Record the results.

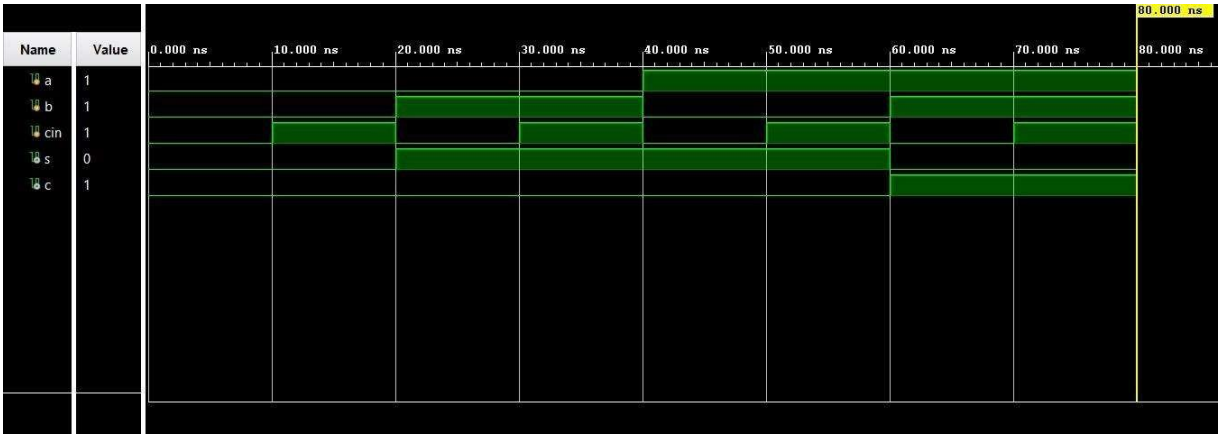
Steps to realize this unit in the Basys3 Board:

1. After successful synthesis and implementation of the unit, connect Basys3 to the computer.
2. The inputs and outputs of the module have to be mapped with the ports of the Basys3 Board. This can be done using the I/O Layout.
3. Under Scalar ports, select the appropriate ports (switches/leds) from the Layout Console.
4. Synthesis and Implement the unit again.
5. Generate Bit Stream Errors. Absence of any error states that the implementation is successful
6. Check the functioning of the unit by toggling the switches and observing the leds.

TIMING DIAGRAMS



Half Adder



Full Adder

CODE:

1. **HALF ADDER**

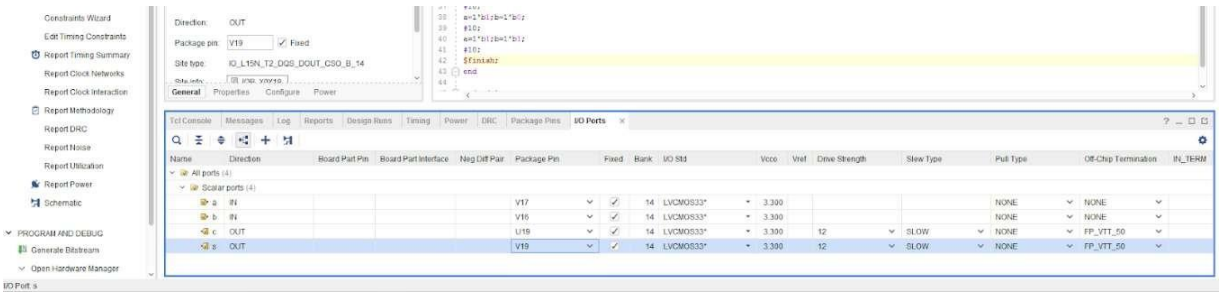
halfadder.v

```
module halfadder (
        input a, b,
        output reg s, c);
        always @ (a or b)
        begin
                s = a ^ b;
                c = a & b;
        end
endmodule
```

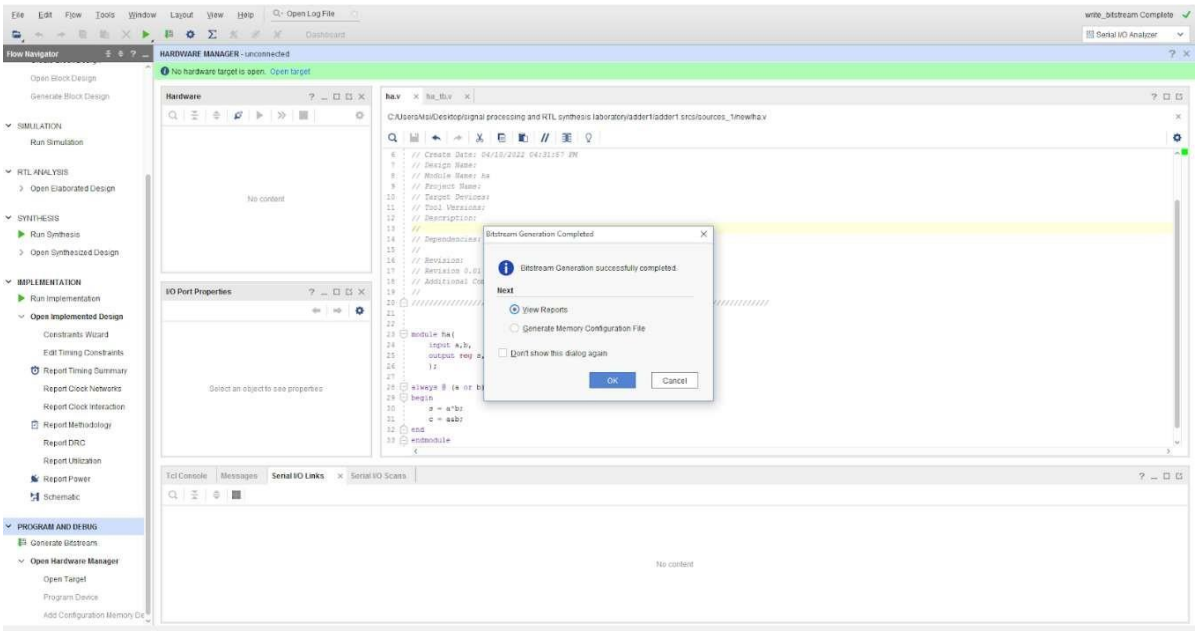
halfaddertb.v

```
module halfaddertb();
        reg a,b;
        wire s,c;

        halfadder dut(
                .a(a),
                .b(b),
                .c(c),
                .s(s));

        initial begin
                a=1'b0; b=1'b0; #10;
                a=1'b0; b=1&'; b1; #10;
                a=1'b1; b=1'b0; #10;
                a=1'b1; b=1'b1; #10;
        $finish;
        end
endmodule
```

OBSERVATION:



Assigning I/O ports



Bit Stream is generated successfully

2. FULL ADDER

fulladder.v

```
module fulladder (
        input a,b,cin,
        output reg s,c);
always @ (a or b or cin)
begin
        s = (a ^ b) ^ cin;
        c = (a & b) + (b & cin) + (cin & a);
end
endmodule
```

fulladdertb.v

```
module fulladdertb();
        reg a,b,cin;
        wire s,c;

        fulladder dut(
                .a(a),
                .b(b),
                .cin(cin),
                .s(s),
                .c(c));

        initial begin
                a=1'b0; b=1'b0; cin=1'b0; #10;
                a=1'b0; b=1'b0; cin=1'b1; #10;
                a=1'b0; b=1'b1; cin=1'b0; #10;
                a=1'b0; b=1'b1; cin=1'b1; #10;
                a=1'b1; b=1'b0; cin=1'b0; #10;
                a=1'b1; b=1'b0; cin=1'b1; #10;
                a=1'b1; b=1'b1; cin=1'b0; #10;
                a=1'b1; b=1'b1; cin=1'b1; #10;
        $finish;
        end
endmodule
```

INFERENCE:

The designed adders work and their function is verified with the help of truth table. It is realized with the help of Basys3 Board and function is checked by toggling the switches and observing the LEDs.

RESULT:

Thus, the adders are implemented and realized using the Basys3 Board.

EX NO: 02 /2022	TIMING ANALYSIS OF COMBINATIONAL CIRCUITS

AIM:

To perform timing analysis of combinational logic and register.

SOFTWARE REQUIRED :

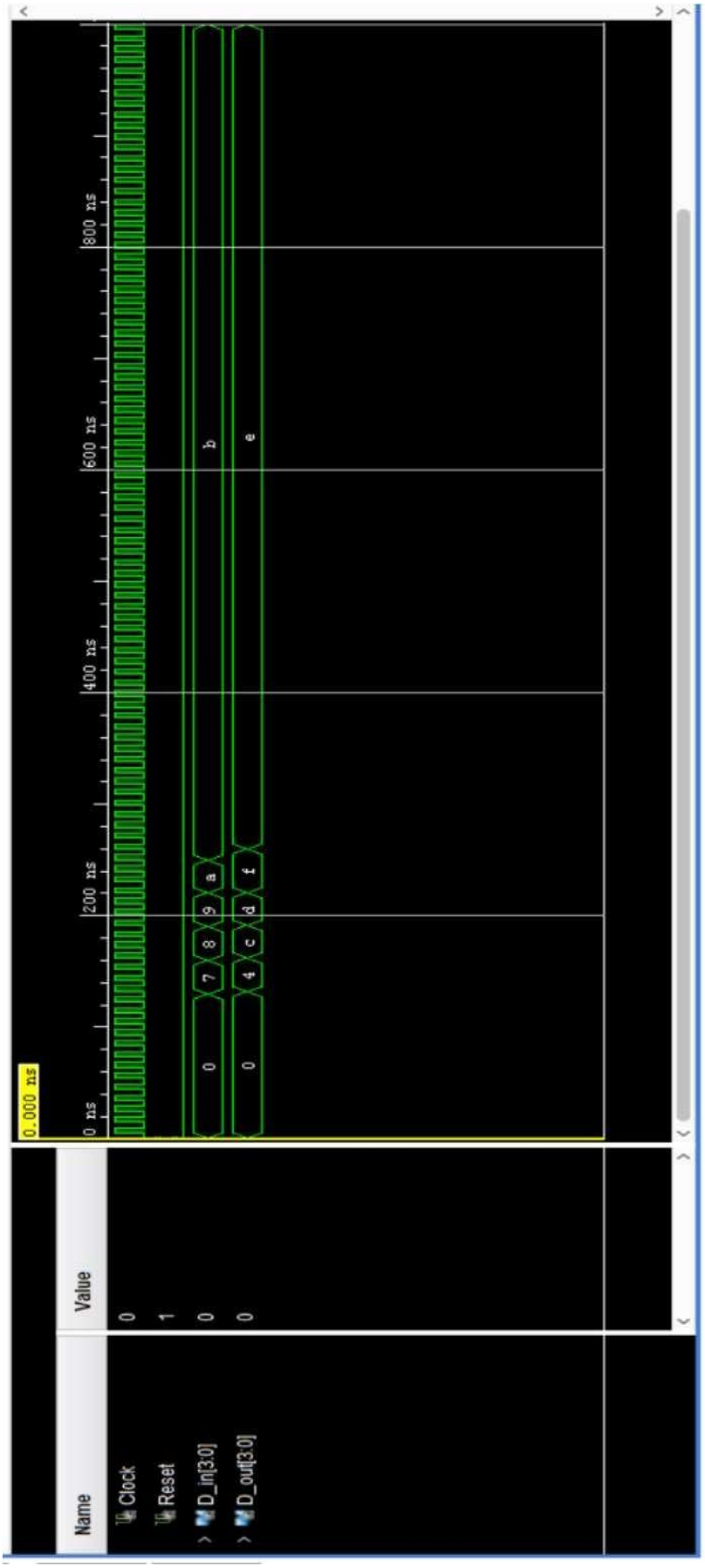
Xilinx Vivado Webpack Version 2020.2

PROCEDURE:

Steps for implementation of the unit in VHDL:

1. Open Vivado. Click Create New Project to start the program. Click Next in the dialog box that appears.
2. Enter Project Name and choose a location
3. Select RTL Project option in the Project Type form, and click Next.
4. Select VHDL as the Target language and Simulator language in the Add Sources form.
5. Choose Basys 3 Board from the boards tab
6. Click Finish to create the Vivado project.
7. To simulate the design using simulator, click Add Sources under the Project Manager tasks of the Flow Navigator pane.
8. Choose “Add or Create Design Sources” to create a design file to write the code for the unit to be realised.
9. Create new source file for testbench.
10. Simulate the design files and perform implementation
11. Give clock constraints.
12. After simulation, based on **W**orst **N**egative **S**lope (WNS), change the clock constraints to make it 0.
13. Simulate and analyse.

SIMULATION:


Name
Value
Clock
Reset
D_in[3:0]
D_out[3:0]
0
1
0
0
0.000 ns
0 ns
200 ns
400 ns
600 ns
800 ns
0
7
8
9
a
b
0
4
c
d
f
e

CODE FLOW:

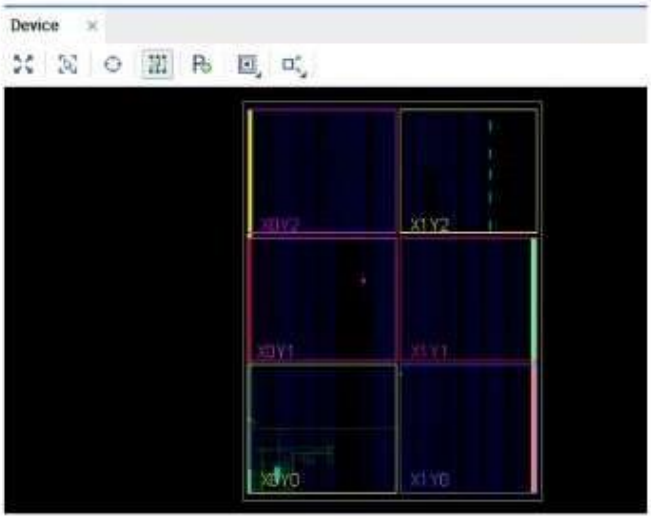
Top Level:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity top_level is
 Port (Clock, Reset :in STD_LOGIC;
       D_in:in STD_LOGIC_VECTOR (3 downto 0);
       D_out:out STD_LOGIC_VECTOR (3 downto 0));
end top_level;

architecture Behavioral of top_level is

component comb_logic is
port(A : in std_logic_vector(3 downto 0);
     Y : out std_logic_vector(3 downto 0));
end component;

component register_ff is
Port ( CLK, RST : in  STD_LOGIC;
       d_in : in STD_LOGIC_VECTOR(3 downto 0);
       d_out : out STD_LOGIC_VECTOR(3 downto 0) );
end component;

signal int_b : STD_LOGIC_VECTOR(3 downto 0);

begin

U1 : comb_logic  port map(A=>D_in,Y=>int_b);
U2 : register_ff  port map(CLK=>Clock,RST=> Reset,
d_in=>int_b,d_out=>D_out);

end Behavioral;
```

Combinational logic:

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

entity comb_logic is
port(A : in std_logic_vector(3 downto 0);
     Y : out std_logic_vector(3 downto 0));
end comb_logic;

architecture Behavioral of comb_logic is
```

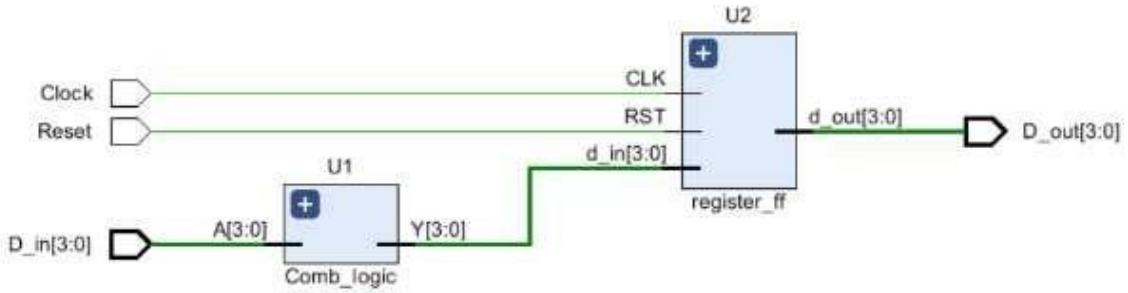
```
                          (clock Clock rise edge)       8.334      8.334 r
                          clock pessimism               0.000      8.334
                          clock uncertainty            -0.035      8.299
                          output delay                 -0.000      8.299
    ----------------------------------------------------------------------
                          required time                            8.299
                          arrival time                            -8.298
    ----------------------------------------------------------------------
                          slack                                    0.000

Slack (MET) :             0.091ns  (required time - arrival time)
```



Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	2.640 ns	Worst Hold Slack (WHS):	-1.032 ns	Worst Pulse Width Slack (WPWS):	3.667 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	-6.708 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	8	Number of Failing Endpoints:	0
Total Number of Endpoints:	12	Total Number of Endpoints:	12	Total Number of Endpoints:	5

Timing constraints are not met.



```
begin
Y(3)<= A(3);
Y(2)<= A(3) xor A(2);
Y(1)<= A(2) xor A(1);
Y(0)<= A(1) xor A(0);

end Behavioral;
```

Register flip-flop:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity register_ff is
Port ( CLK, RST : in  STD_LOGIC;
       d_in : in STD_LOGIC_VECTOR(3 downto 0);
       d_out : out STD_LOGIC_VECTOR(3 downto 0) );
end register_ff;

architecture Behavioral of register_ff is
begin
process (CLK, RST, d_in)
begin
if (RST = '1') then
d_out <= x"0";
elsif (CLK='1' and CLK'event) then

d_out <= d_in;
end if;
end process;
end Behavioral;
```

TESTBENCH

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity top_level_tb is
end top_level_tb;

architecture Bench of top_level_tb is
component top_level is
Port (Clock, Reset :in STD_LOGIC;
       D_in:in STD_LOGIC_VECTOR (3 downto 0);
       D_out:out STD_LOGIC_VECTOR (3 downto 0));
```

```
end component ;

signal Clock, Reset : STD_LOGIC;
signal D_in, D_out : STD_LOGIC_VECTOR(3 downto 0);



 begin

dut: top_level port map(Clock => Clock, Reset=> Reset, D_in => D_in, D_out =>
D_out);

D_in <= "0000","0111" after 130 ns, "1000" after 159.95 ns, "1001" after 190
ns,"1010" after 220 ns,"1011" after 250 ns;

Clk : process
begin
 Clock <= '0';
 wait for 4.167 ns;
 Clock <= '1';
 wait for 4.167 ns;
end process;

stim : process
begin
 Reset <= '1';
 wait for 2ns;
 Reset <= '0';
 wait ;
end process;

end Bench;
```

INFERENCE:

Thus, the WNS can be modified by changing the frequency of the clock.

RESULT:

Timing analysis has been performed successfully for a combinational logic.

EX NO: 03 09/04/2022	INSTANTIATION OF CLOCK GENERATOR IN THE DIGITAL DESIGN FROM IP CATALOGUE

AIM:

To perform instantiation of clock generator using IP catalogue verifying ALU

SOFTWARE REQUIRED :

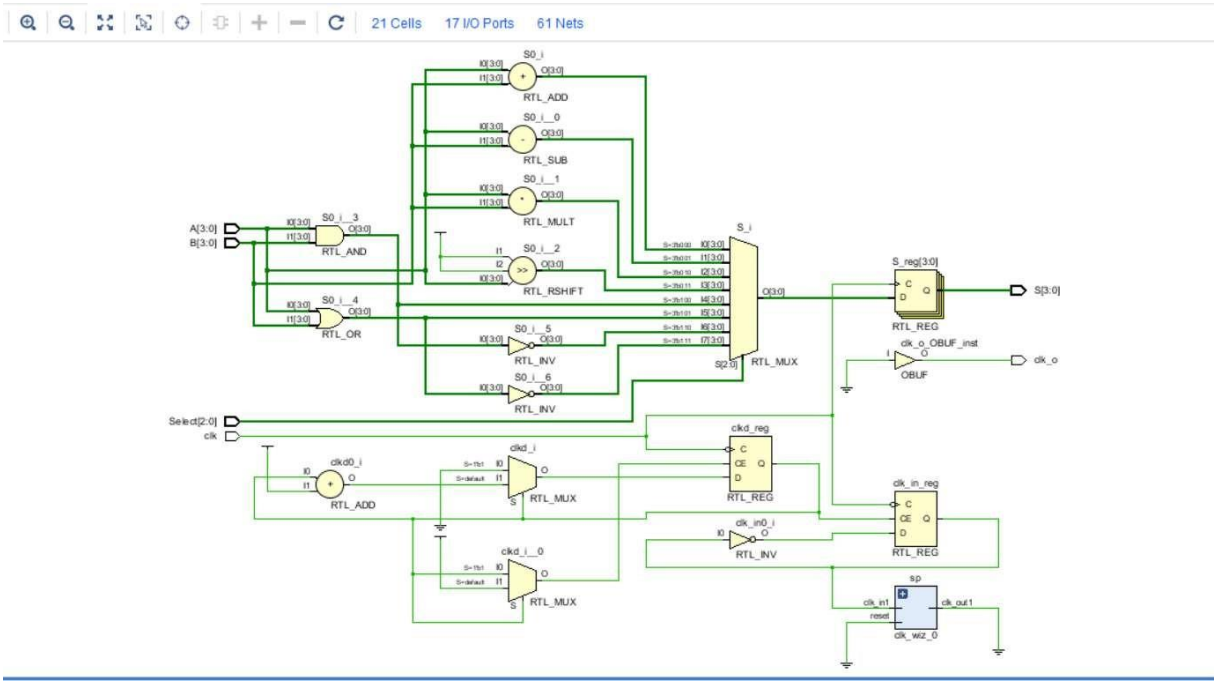
Xilinx Vivado Webpack Version 2020.2

PROCEDURE:

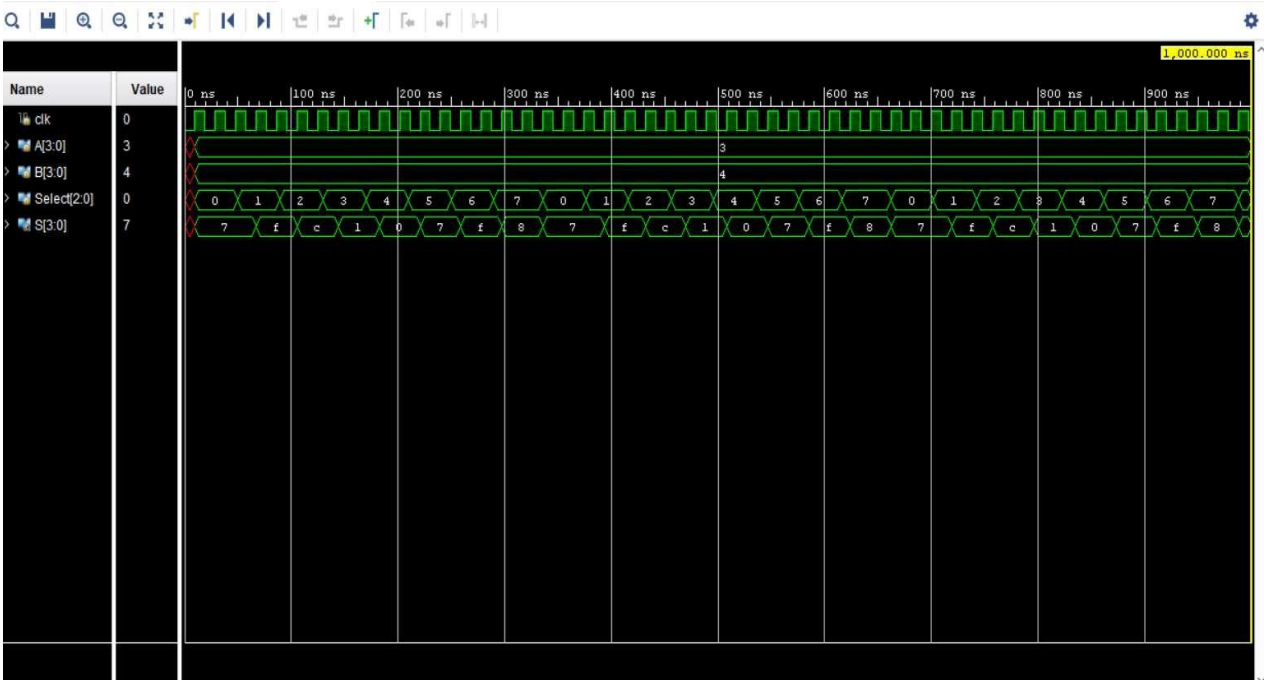
Steps for implementation of the unit in Verilog:

1. Open Vivado. Click Create New Project to start the program. Click Next in the dialog box that appears.
2. Enter Project Name and choose a location
3. Select RTL Project option in the Project Type form, and click Next.
4. Select Verilog as the Target language and Simulator language in the Add Sources form.
5. Choose Basys Board from the boards tab
6. Click Finish to create the Vivado project.
7. To simulate the design using simulator, click Add Sources under the Project Manager tasks of the Flow Navigator pane.
8. Choose “Add or Create Design Sources” to create a design file to write the code for the unit to be realised.
9. Clicking IP catalogue and select clocking wizard.In clocking options tab select primitive as PLL and set input frequency as 100MHz.
10. In output clocks ,disable reset and locked signal from IP.
11. Select out of content per ip from generate output products dialog box.

SCHEMATICS :



TIMING DIAGRAM:



12. It creates a clk-using-o folder under design sources.

13. Open the testbench of ALU and set the reset and clkin.

14. After saving the files, run behavioural simulation for the test bench and observe the timing diagram.

15. Receive the clock as input and pass it through the PLL and feed the clockout as the clock for the ALU and control unit in the ALU's RTL file.

16. View the schematics and verify that post implementation ,PLL module has been properly instantiated in the top level module.

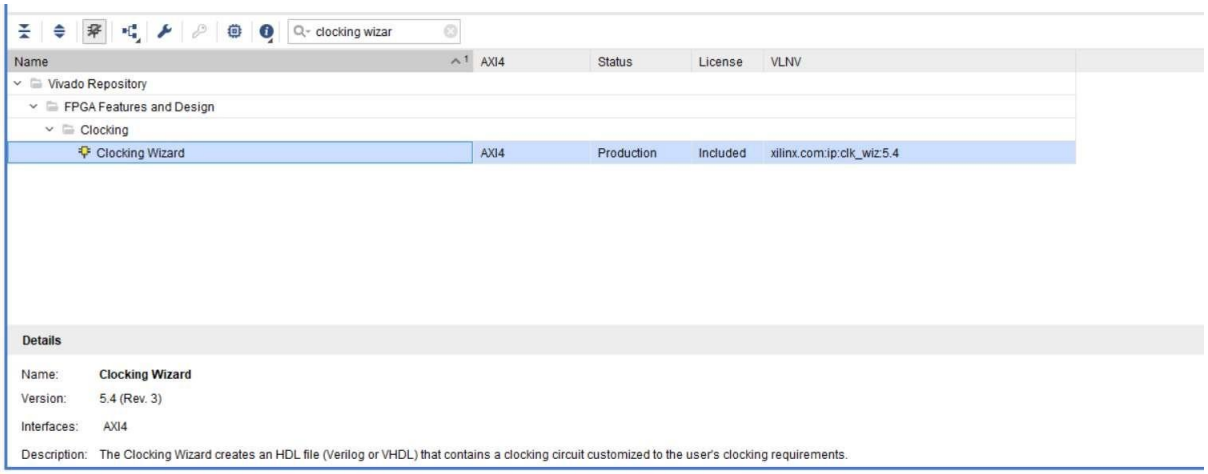
17. Run post implementation timing simulation and verify the PLL is successfully running.

CODE FLOW:

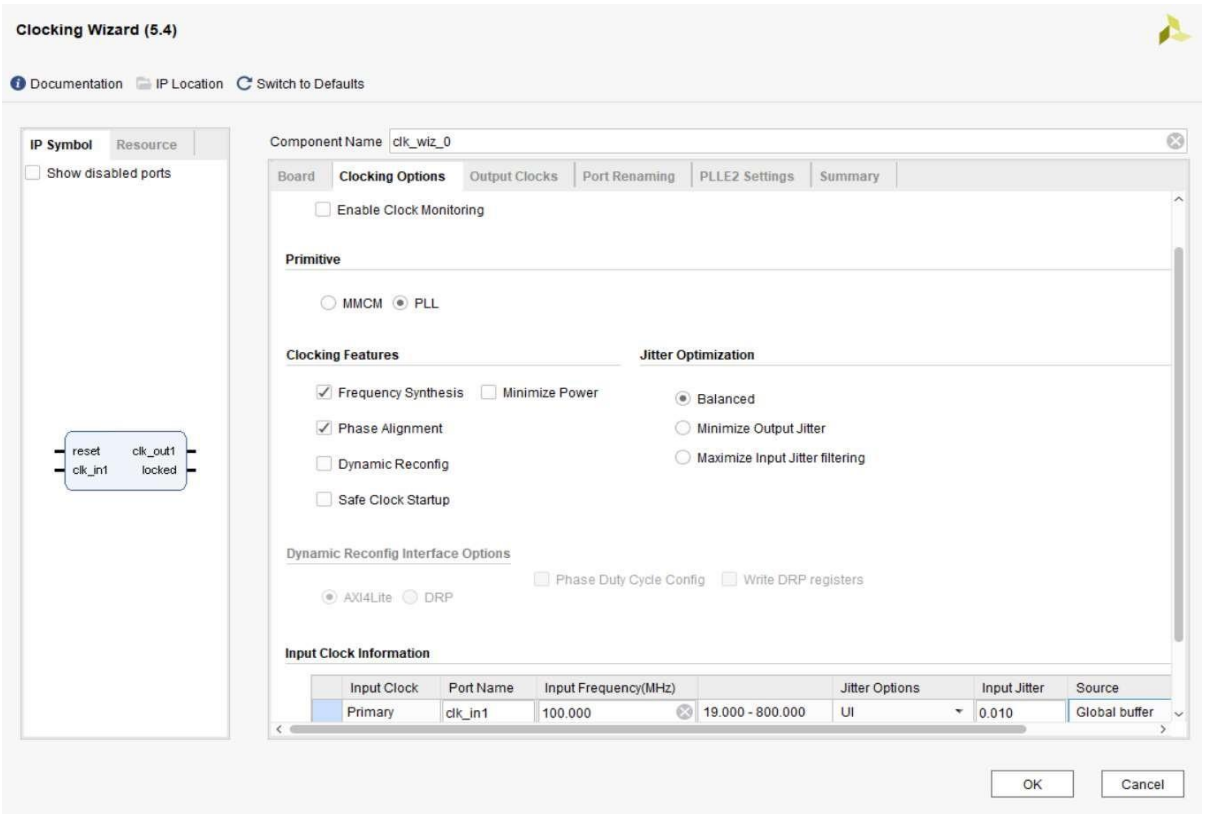
```
module alu(
    input clk,
    input [3:0] A,B,
    input  [2:0]  Select,
    output reg [3:0] S,
    output clk_o);

    reg clk_in = 0;
    wire clk_out = 0;
    wire reset = 0;
    wire locked = 0;
    reg clkd = 0;
    assign clk_o = clk_out;
    assign reset = 0;

    clk_wiz_0_clk_wiz inst(
        .clk_out1(clk_out),
        .clk_in1(clk_in)
);
```

STEPS TO INSTANTIATE CLOCK GENERATOR FROM IP CATALOGUE:

1. Select clocking wizard from IP catalogue



2. Select PLL and set input clock frequency

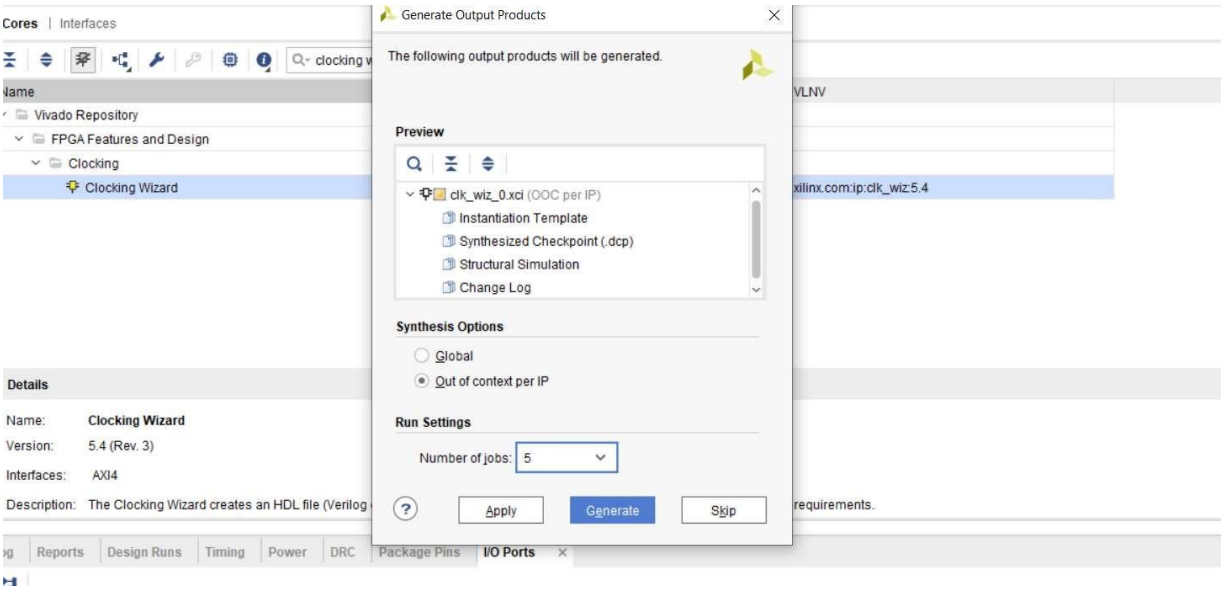


```
    always @ (negedge clk)
    begin
    if (clkd==1)
    begin
    clkd=0;
    clk_in<=~clk_in;
    end
    else
    clkd<= clkd+1;
    end

    always @ (posedge clk)
    begin
    case (Select)
        3'b000: S = A+B;
        3'b001: S = (A-B);
        3'b010: S = A*B;
        3'b011: S = A>>1;
        3'b100: S = A&B;
        3'b101: S = A|B;
        3'b110: S = ~(A&B);
        3'b111: S = ~(A|B);
    endcase
    end
endmodule
```

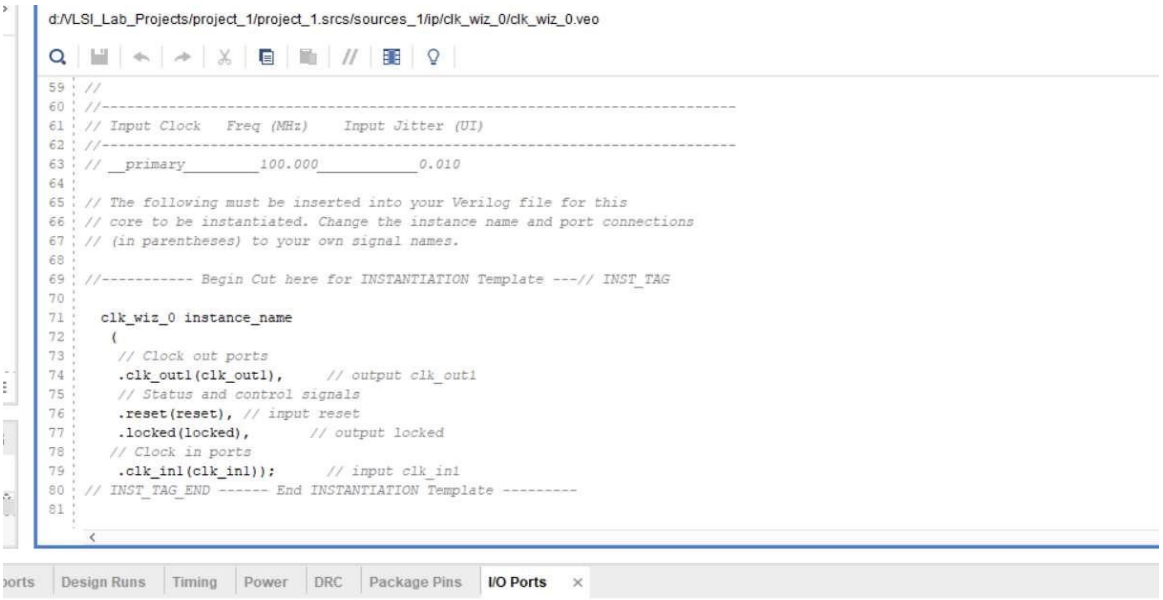
TESTBENCH

```
`timescale 1ns / 1ps
module alu_tb();
    reg clk=0;
    reg [3:0] A,B;
    reg [2:0] Select;
    wire [3:0] S;
    always #9.593 clk = ~clk;

alu dut(
    .clk(clk),
    .A(A),
    .B(B),
    .S(S),
    .Select(Select)
    );

always @ (posedge clk)
begin
    A=4'b0011; B=4'b0100; Select=3'b000;
    #40;
    A=4'b0011; B=4'b0100; Select=3'b001;
```

3. Generate Output Products dialogue box opens up. Select out of context IP and click generate. This creates PLL module.



4. Navigate to IP sources ➔ clk_wiz ➔ Instantiation template ➔clk_wiz.o.veo .file and copy the instance name module clock. Integrate with ALU code.



```
    #40;
    A=4'b0011; B=4'b0100; Select=3'b010;
    #40;
    A=4'b0011; B=4'b0100; Select=3'b011;
    #40;
    A=4'b0011; B=4'b0100; Select=3'b100;
    #40;
    A=4'b0011; B=4'b0100; Select=3'b101;
    #40;
    A=4'b0011; B=4'b0100; Select=3'b110;
    #40;
    A=4'b0011; B=4'b0100; Select=3'b111;
    #40;
end
endmodule
```

INFERENCE :

Thus, the PLL can be used to decrease the clock frequency.

RESULT: Thus , the ALU has been verified using multiplied clock from ALU.

EXP. NO.: 4	Realization of Finite State Machine

Aim:

To realize a finite state machine and verify its output graph

Software used:

Vivado

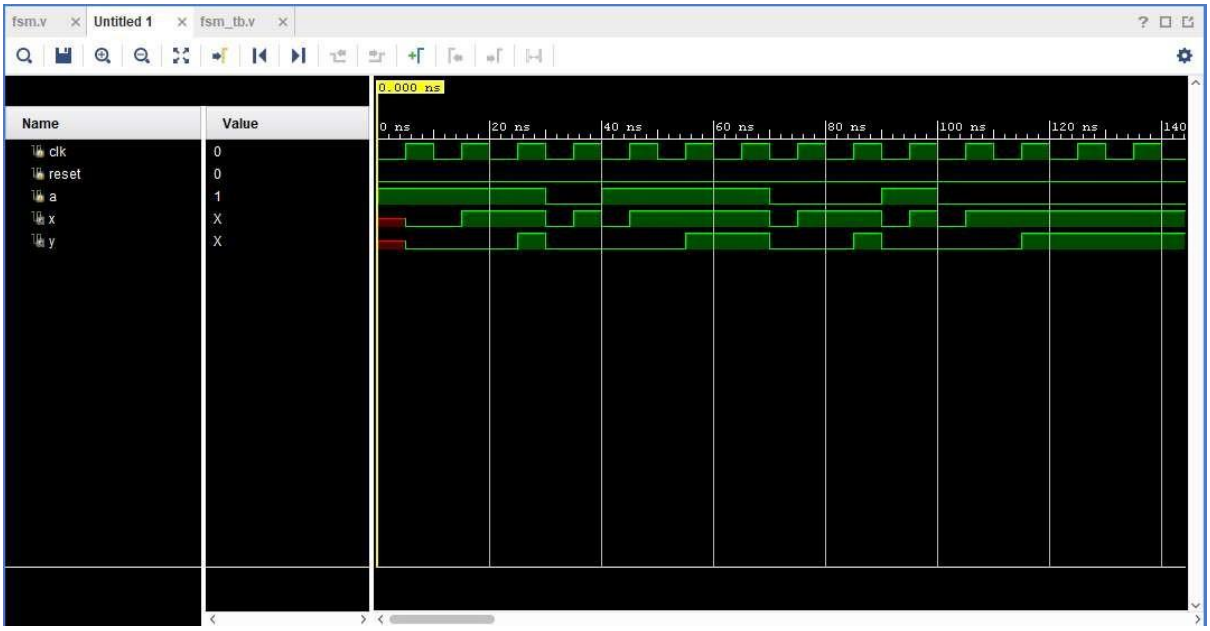
Procedure:

- Open vivado software
- Click on **create project** and click **next**
- Give project name and project location and click **next**
- Select **RTL project** option and create block design in IP integration
- Choose target language as **Verilog** and **Mixed language** as the simulator
- Select **Basys 3** board and give finish after confirming the details
- Click on the plus sign on source and click **create file** and give file name and file type and give **finish**
- Click port name for input and output and select direction of buses.
- Open verilog under **design sources**.
- Enter code and testbench
- Run behavioral simulation

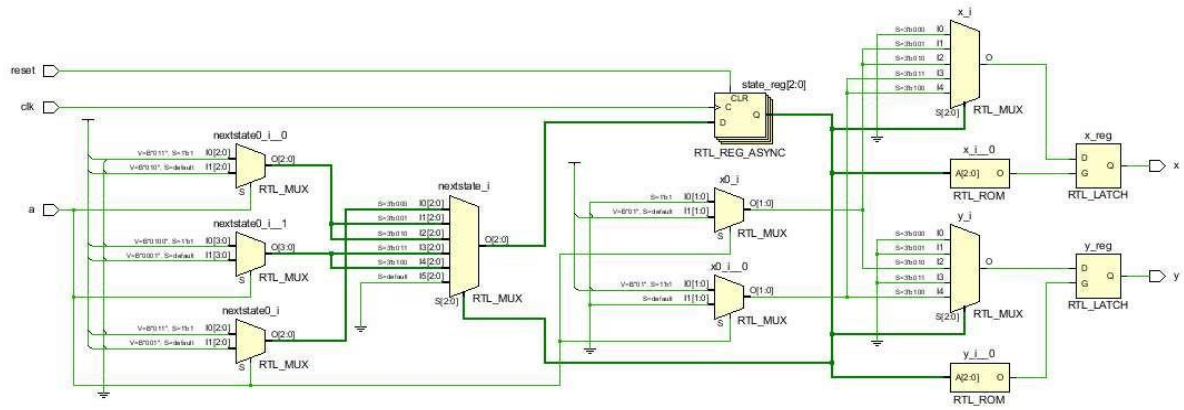
Code:

```
module fsm(
  input clk,
 input reset,
input a,
```

Timing Diagram:

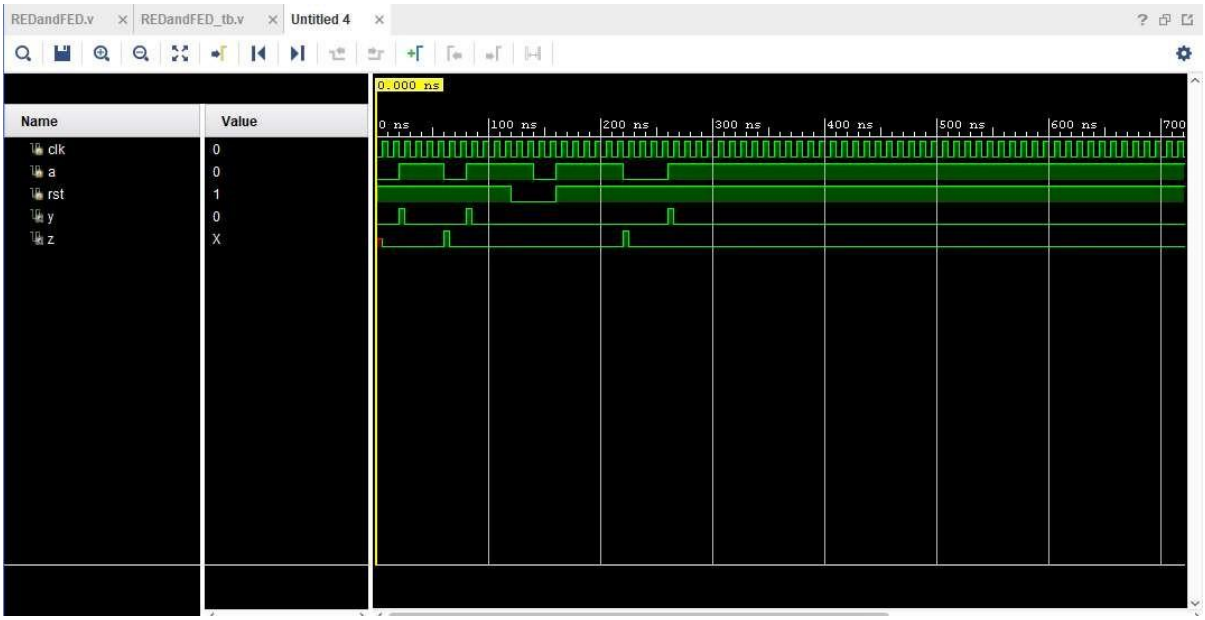


Schematic:

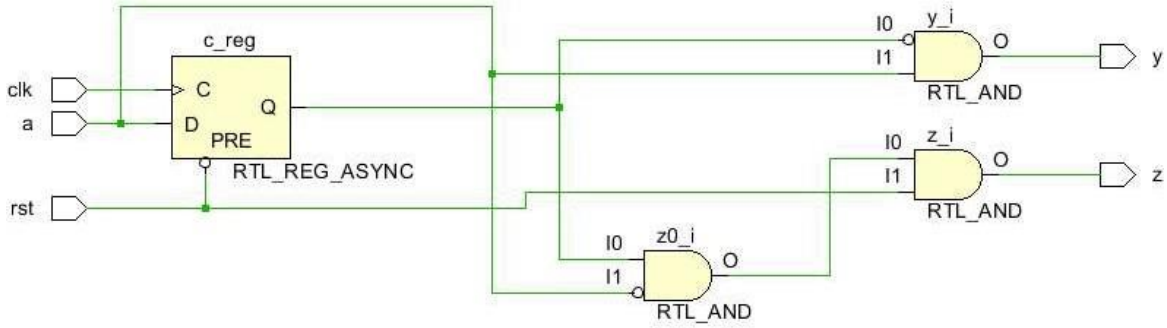


```
output reg x,
output reg y);
reg [2:0] state,nextstate;

parameter s0=0;
parameter s1=1;
parameter s2=2;
parameter s3=3;
parameter s4=4;

always @(posedge clk,posedge reset)
   if(reset) state<= s0;
   else  state<=nextstate;

always @(state,a)
case (state)
  s0: begin
  x = a ? 0 : 0;
  y = a ? 0 : 0;
  nextstate = a ? s3 : s1;
  end
  s1: begin
  x = a ? 0 : 1;
  y = a ? 0 : 0;
  nextstate = a ? s3 : s2;
  end
  s2: begin
  x = a ? 0 : 1;
  y = a ? 0 : 1;
```

Timing Diagram:



Schematic:



```
nextstate = a ? s3 : s2;
end
s3: begin
x = a ? 1 : 0;
y = a ? 0 : 0;
nextstate= a ? s4 : s1;
end
s4: begin
x = a ? 1 : 0;
y = a ? 1 : 0;
 nextstate = a ? s4 : s1;
end
default: nextstate =s0;
    endcase
Endmodule
```

Testbench:

```
module fsm_tb(
  );
  reg clk=0;
  reg reset=0;
  reg a; wire x,y;
```

```
fsm dut(.clk(clk),.reset(reset),.a(a),.x(x),.y(y));
   always
    #5 clk =~clk;
   initial begin
   a=1;
    #10 a=1;
    #10 a=1;
    #10 a=0;
    #10 a=1;
    #10 a=1;
    #10 a=1;
    #10 a=0;
    #10 a=0;
    #10 a=1;
    #10 a=0;
    #10 a=0;
    #10 a=0;
   end
endmodule
```

RISING EDGE AND FALLING EDGE DETECTION:

Code:

```
module REDandFED(
   input clk,
   input a,
   input rst,
   output y,
   output z
```

```
    );
    reg c;
    always @(posedge clk,negedge rst)
    begin
    if (~rst)
       c<=1;
    else
       c<=a;
    end

    assign y=(~c&a);
    assign z=(c&~a&rst);
endmodule
```

Testbench:

```
module REDandFED_tb(

    );
    reg clk=0,a,rst=1;
    wire y,z;
    REDandFED viki(.clk(clk),.a(a),.rst(rst),.y(y),.z(z));
    always
    #5 clk=~clk;

    initial
    begin
    a=0;
    #20 a=1;
```

```
    #20 a=1;
    #20 a=0;
    #20 a=1;
    #20 a=1;
    #20 rst=0;
    #20 a=0;
    #20 rst=1;a=1;
    #20 a=1;
    #20 a=1;#20 a=0;#20 a=0;#20 a=1;
    end

Endmodule
```

Result:

The realization of Finite State Machine has been successfully executed

EXP. NO.: 5	**DEBUGGING INTERNAL SIGNAL OF MEMORY IN FPGA**

Aim:

To perform debugging of internal signal of memory module in FDGA in Basys 3 board.

Tools required:

Software : VIVADO HLX 2018.4

Hardware : Basys 3 board

Procedure:

- Open vivado software and write a Verilog code and respective test bench to implement it.
- And now run simulation and check the schematic diagram for proper working.
- Now, click the IP catalog and then to debug and verification.
- Now click IPA.

  Set the number of probes and number of input ports under general option

  Next get into probe ports and set probe width.

  Now synthesise the code and perform implementation.

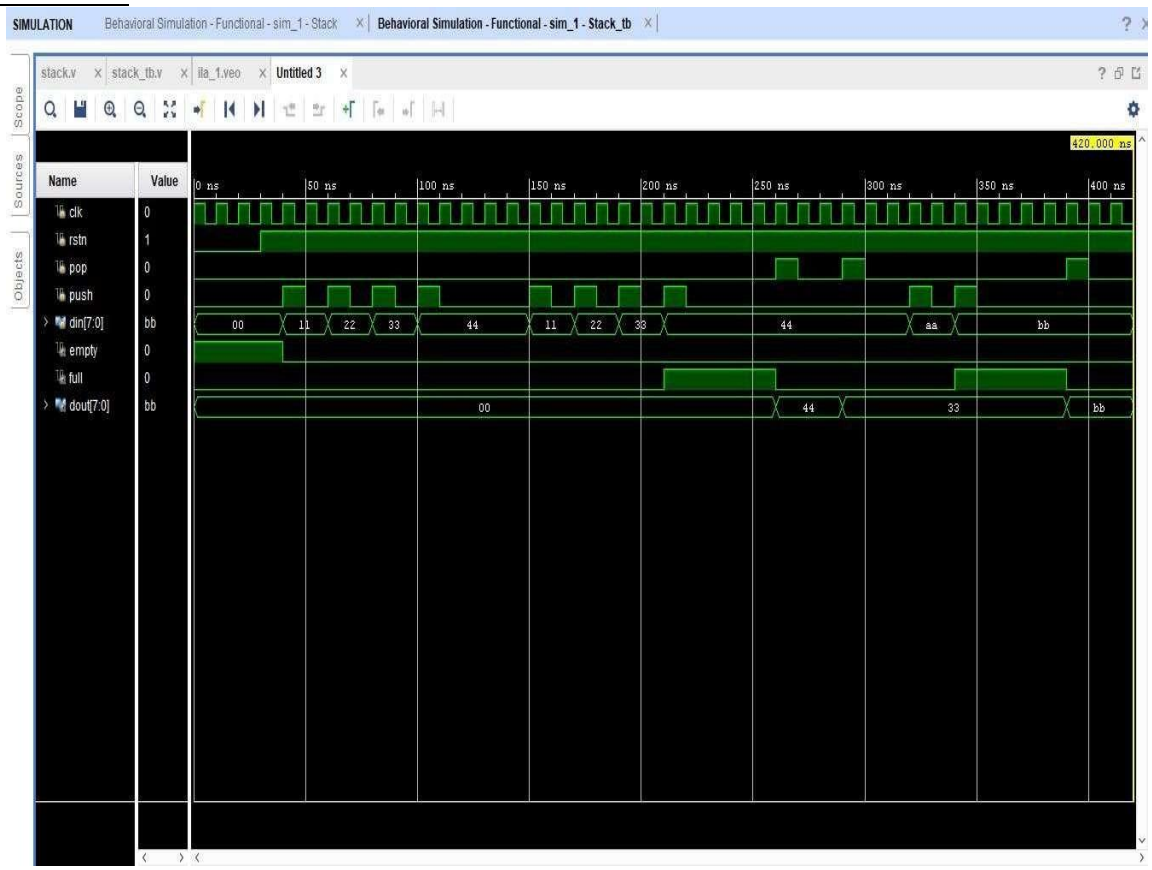
  Next and finally the hardware implementation is performed using Basys 3 board done before.

CODE:

```
module Stack(
clk, rstn, pop, push, empty, full, din, dout);
parameter WIDTH = 8;
 parameter DEPTH = 8;

input clk;
input rstn;
input pop;
input push;
input [WIDTH-1:0] din;

output [WIDTH-1:0] dout;
 output empty;
output full;
```

Output:

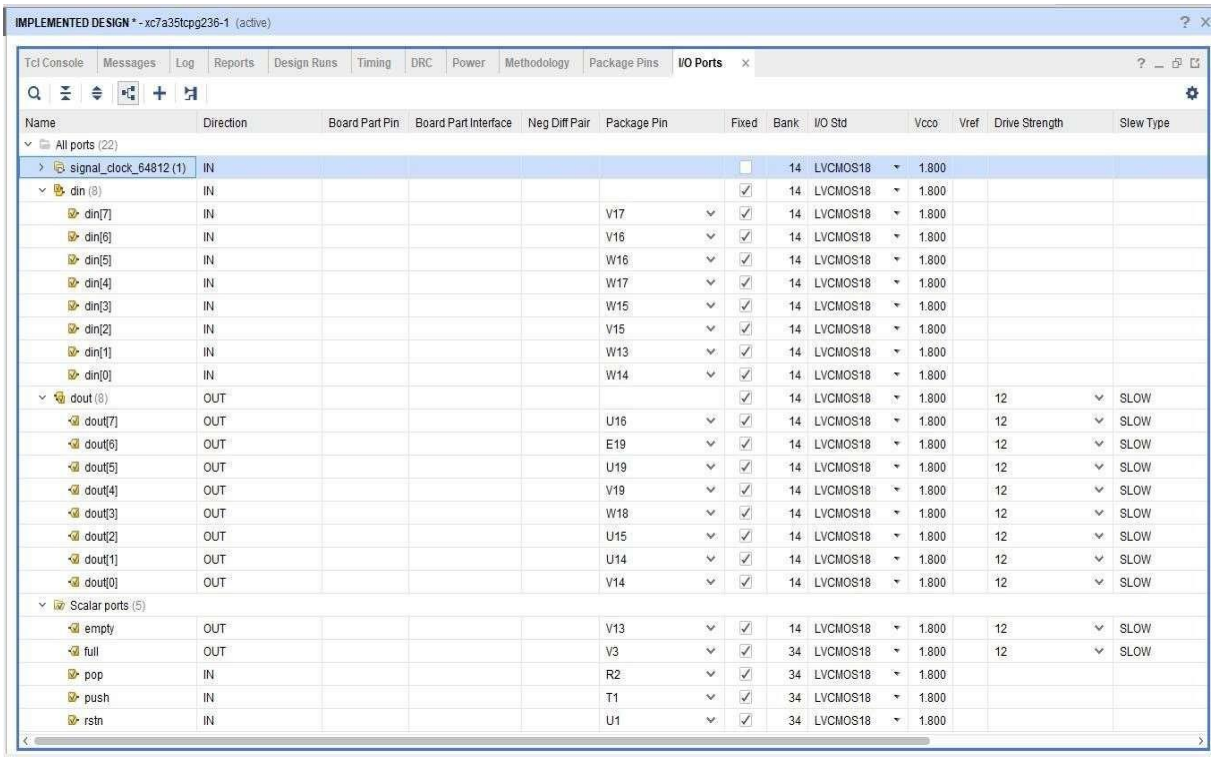
Simulation:



Timing:

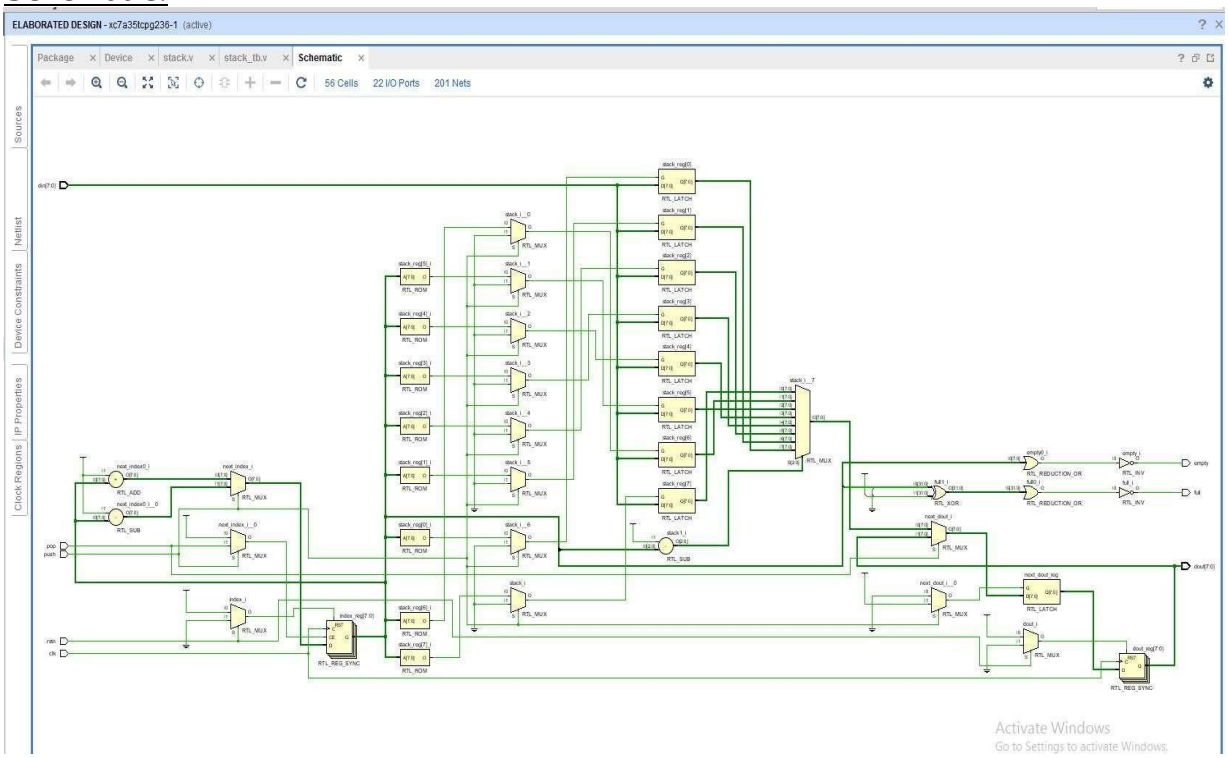
```
reg [WIDTH-1:0] stack[DEPTH-1:0]; //memory
reg [WIDTH-1:0]index, next_index;
reg [WIDTH-1:0]dout, next_dout;

wire empty, full;
ila_1 your_instance_name (
.clk(clk), // input wire clk
.probe0(clk), // input wire [7:0]           probe0
.probe1(rstn), // input wire [7:0]          probe1
.probe2(pop), // input wire [7:0]           probe2
.probe3(push), // input wire [7:0]          probe3
.probe4(empty), // input wire [7:0]         probe4
.probe5(full), // input wire [7:0]          probe5
.probe6(din), // input wire [0:0]           probe6
.probe7(dout) // input wire [0:0]           probe7
);

always @ (posedge clk) //Sequential block
 begin
if(!rstn)
 begin
dout <= 8'd0;
index <= 1'b0;
end
else
begin
dout <= next_dout;
index <= next_index;
   end
 end

assign empty = !(|index);
assign full= !(|(index ^ DEPTH));

always @ (*) //Combinational Block
 begin
```

I/O ports:



Name	Direction	Board Part Pin	Board Part Interface	Neg Diff Pair	Package Pin	Fixed	Bank	I/O Std	Vcco	Vref	Drive Strength	Slew Type
All ports (22)												
signal_clock_64812 (1)	IN						14	LVCMOS18	1.800			
din (8)	IN						14	LVCMOS18	1.800			
din[7]	IN				V17		14	LVCMOS18	1.800			
din[6]	IN				V16		14	LVCMOS18	1.800			
din[5]	IN				W16		14	LVCMOS18	1.800			
din[4]	IN				W17		14	LVCMOS18	1.800			
din[3]	IN				W15		14	LVCMOS18	1.800			
din[2]	IN				V15		14	LVCMOS18	1.800			
din[1]	IN				W13		14	LVCMOS18	1.800			
din[0]	IN				W14		14	LVCMOS18	1.800			
dout (8)	OUT						14	LVCMOS18	1.800		12	SLOW
dout[7]	OUT				U16		14	LVCMOS18	1.800		12	SLOW
dout[6]	OUT				E19		14	LVCMOS18	1.800		12	SLOW
dout[5]	OUT				U19		14	LVCMOS18	1.800		12	SLOW
dout[4]	OUT				V19		14	LVCMOS18	1.800		12	SLOW
dout[3]	OUT				W18		14	LVCMOS18	1.800		12	SLOW
dout[2]	OUT				U15		14	LVCMOS18	1.800		12	SLOW
dout[1]	OUT				U14		14	LVCMOS18	1.800		12	SLOW
dout[0]	OUT				V14		14	LVCMOS18	1.800		12	SLOW
Scalar ports (5)												
empty	OUT				V13		14	LVCMOS18	1.800		12	SLOW
full	OUT				V3		34	LVCMOS18	1.800		12	SLOW
pop	IN				R2		34	LVCMOS18	1.800			
push	IN				T1		34	LVCMOS18	1.800			
rstn	IN				U1		34	LVCMOS18	1.800			

Schematic:



```
  if(push) //write
  begin
  stack[index] = din;
  next_index = index+1'b1;
  end
  else if(pop)          //read
  begin
  next_dout = stack[index-1'b1];
   next_index = index-1'b1;
  end
  else
  begin
  next_dout = dout;
  next_index = index;
  end

  end
  endmodule
```

Testbench:

```
  module Stack_tb;
// Inputs
reg clk;
reg rstn;
reg pop;
reg push;
  reg [7:0] din;
// Outputs
wire empty;
wire full;
   wire [7:0] dout;
// Instantiate the Unit Under Test (UUT)
Stack uut (
  .clk(clk),
  .rstn(rstn),
  .pop(pop),
  .push(push),
```

```
  .empty(empty),
  .full(full),
  .din(din),
  .dout(dout)
  );
  always #5 clk = ~clk;
  task reset();          //reset
  task begin
  clk = 1'b1;
  rstn = 1'b0;
  pop = 1'b0;
  push = 1'b0;
  din = 8'd0;
  #30 rstn = 1'b1;
  end
  endtask
task read_stack();            //read
task begin
  pop = 1'b1;
  #10
  pop = 1'b0;
  end
  endtask

  task write_stack([7:0]din_tb); //write
  task begin
  push = 1'b1;
  din = din_tb;
  #10 push = 1'b0;
  end
  endtask
    // Main code
   initial
  begin
  reset();
  #10;
  repeat(2)
   begin
```

```
write_stack(8'h11);
 #10;
write_stack(8'h22);
 #10;
write_stack(8'h33);
 #10;
write_stack(8'h44);
 #40;
end read_stack();
 #20;
read_stack();
 #20;
write_stack(8'hAA);
#10;
write_stack(8'hBB);
#40;
read_stack();
 #20;
$finish;
end
endmodule
```

Inference:

The clock pulse signal of the output signal was verified for Sdifferent input waveforms.

Result: Thus, the debugging of internal signal of memory module in FPGA has been implemented on Basys 3 board and has been verified.

EXP. NO.: 6	**Designing an ASIC using openlane**

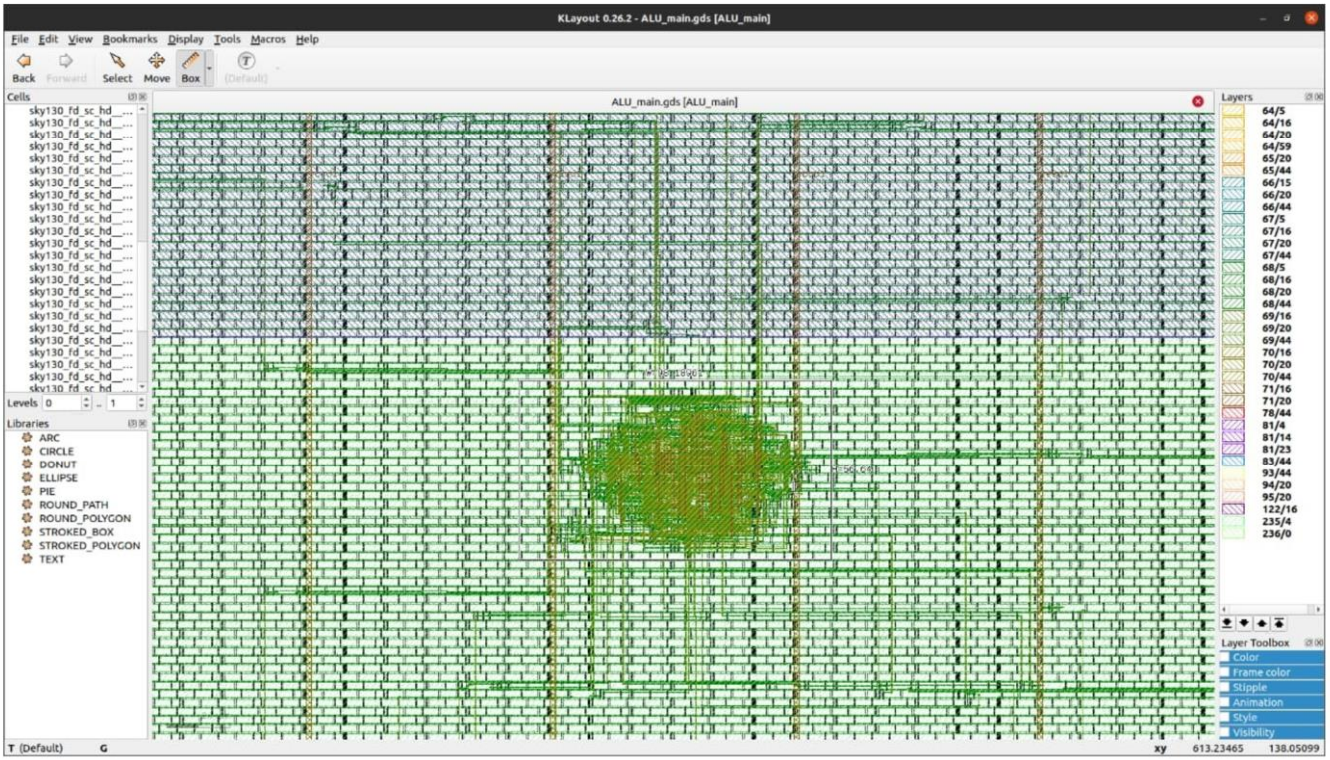
Aim: To use openlane to design an ASIC. Generating the ALU that we have designed in the previous experiments.

Software required: Vivado

Procedure:

- Install the necessary openlane packages and pdks, including docker. Pull the repository from github.
- Open the directory caravel_user_project.
- cd to verilog/rtl and paste the ALU (.v) files to this directory. In the top level file add power pin configurations (vdd1,vss1) and the io_oeb(output) pin and set it to 0.
- Go to verilog/includes and include the top level file in the 2 files and rename the *.example files to *.<alu_top_level_file>
- Go to caravel_user_proj/openlane and copy the example project (naming it the same as your top level file)
- Enter that directory and remove the runs folder.
- Open includes.json and includes.tcl and modify the paths to the Verilog rtl files. (caravel_user_proj/Verilog/rtl/*.v)
- Change the clock port to the name of the clock input in your top level file.
- Cd back to home and create a directory for openlane files. (Naming it opnlane_files).
- “export OPENLANE_ROOT=”home/openlane_files/openlane”
- “export PDK_ROOT=”home/openlane_files/pdks”
- Go to caravel_user_proj and run “make setup”
- Run “make <top_level_filename>”
- After the run is complete, open caravel_user_proj/<top_level_file>/runs/results/final/
- Using klayout open the .gds file and open the final_report.csv

Output:




```
================================================================================
 report_design_area
================================================================================
Design area 503479 u^2 98% utilization.
(END)
```

```
{
    "PDK"                          : "sky130A",
    "STD_CELL_LIBRARY"             : "sky130_fd_sc_hd",
    "CARAVEL_ROOT"                 : "../../caravel",
    "CLOCK_NET"                    : "counter.clk",
    "CLOCK_PERIOD"                 : "10",
    "CLOCK_PORT"                   : "clk",
    "DESIGN_IS_CORE"               : "0",
    "DESIGN_NAME"                  : "alu",
    "DIE_AREA"                     : "0 0 900 600",
    "DIODE_INSERTION_STRATEGY"     : "4",
    "FP_PIN_ORDER_CFG"             : "pin_order.cfg",
    "FP_SIZING"                    : "absolute",
    "GLB_RT_MAXLAYER"              : "5",
    "GND_NETS"                     : "vssd1",
    "PL_BASIC_PLACEMENT"           : "1",
    "PL_TARGET_DENSITY"            : "0.05",
    "RUN_CVC"                      : "1",
    "VDD_NETS"                     : "vccd1",
    "VERILOG_FILES"                : ["../../verilog/rtl/*.v"]
}
```


```
=======================================================================
 report_power
=======================================================================
Group                  Internal  Switching    Leakage      Total
                          Power      Power      Power      Power
-----------------------------------------------------------------------
Sequential             7.72e-05   7.61e-06   1.52e-10   8.48e-05  37.1%
Combinational          5.16e-05   9.23e-05   1.21e-07   1.44e-04  62.9%
Macro                  0.00e+00   0.00e+00   0.00e+00   0.00e+00   0.0%
Pad                    0.00e+00   0.00e+00   0.00e+00   0.00e+00   0.0%
-----------------------------------------------------------------------
Total                  1.29e-04   9.99e-05   1.21e-07   2.29e-04 100.0%
                          56.3%      43.7%       0.1%
(END)
```

```
=======================================================================
 report_worst_slack -max (Setup)
=======================================================================
worst slack 1.74

=======================================================================
 report_worst_slack -min (Hold)
=======================================================================
worst slack 0.47
27-parasitics_multi_corner_sta.worst_slack.rpt (END)
```

Inference:

- From the timing report, we observe that the worst case setup slack is 1.74 ns and hold slack as 0.49ns for a clock period of 10ns
- The area report says that it took an area of 503479 $(\mu m)^2$
- From the power report, the total power consumption is 229µm

Result:

The module passed all the stages of the ASIC flow design, was optimized and the gds file was generated.

EXP. NO.: 7	**Inverter Design using Magic**

AIM:

To make the layout of CMOS inverter in Magic VLSI Layout Tool using the technology sky130A and to extract its SPICE netlist and compare it with the netlist from Xschem(LVS). To also make sure the layout is DRC clean.

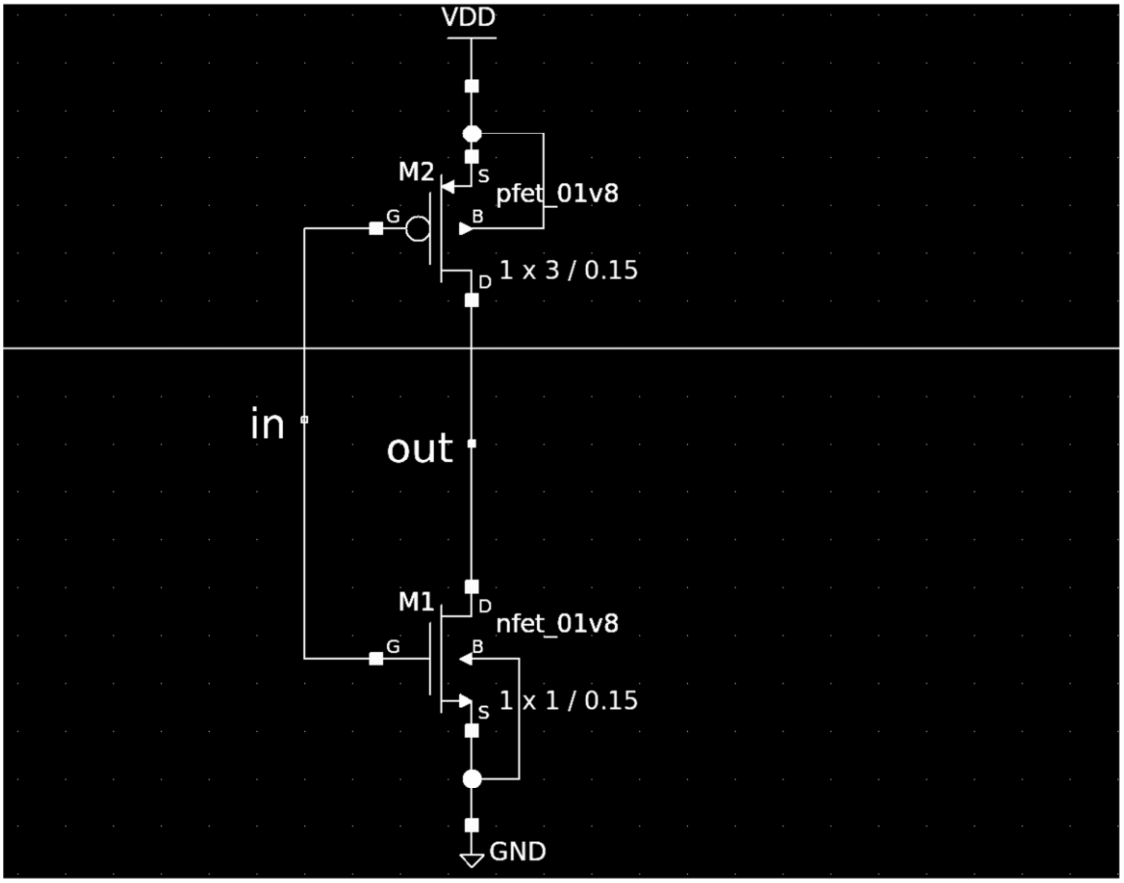
SOFTWARE REQUIRED:

- Magic VLSI Layout Tool
- Xschem schematic editor
- Netgen
- Skywater 130 nm PDK

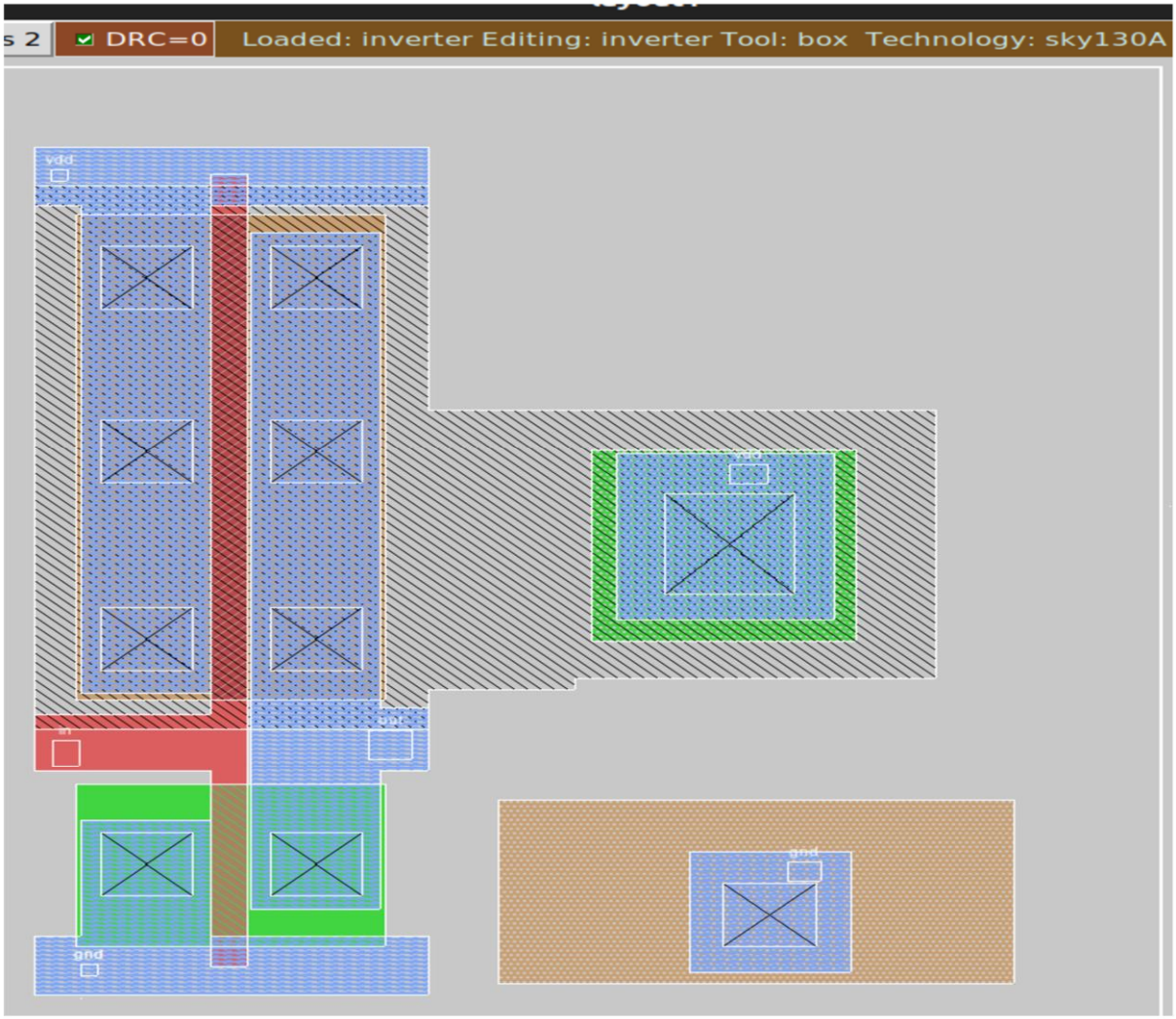
PROCEDURE:

1) Launch Xschem and create a new schematic of the CMOS inverter with labels for input and output and name the file as "inverter.sch"
2) Use the minimum size devices – nmos 1um/0.15um and pmos 3 um/0.15 um
3) Generate netlist by clicking on 'Netlist' button in Xschem and name it as "inverter_schem.spice"
4) Launch Magic and draw the layout of CMOS inverter with specified size and following the design rules put forward by the PDK
5) Label the input, output, supply and ground
6) Make nwell contact and psubstrate contact and attach them to 'vdd' and 'gnd' respectively
7) Use 'li' layer for Vdd and Gnd
8) Check for Design Rule violation by making sure "DRC=0" is shown or issuing the command "drc check" in Magic's tcl prompt.
9) Extract the netlist from the layout using the commands "extract", "ext2spice" in succession. Rename the netlist generated from extraction as "inverter_extracted.spice"
10) Launch Netgen and issue the command "lvs inverter_schem.spice inverter_extracted.spice". This will generate a file with the name "comp.out" with LVS report. Make sure that the extracted and schematic netlist match with each other.

SCHEMATIC:


VDD
M2
S
pfet_01v8
G
B
D
1 x 3 / 0.15
in
out
M1
D
nfet_01v8
G
B
S
1 x 1 / 0.15
GND

LAYOUT:



LVSREPORT:

```
Equate elements:  no current cell.
Equate elements:  no current cell.

Subcircuit summary:
Circuit 1: inverter_schem.spice           |Circuit 2: inverter_extracted.spice
------------------------------------------|------------------------------------------
sky130_fd_pr__nfet_01v8 (1)               |sky130_fd_pr__nfet_01v8 (1)
sky130_fd_pr__pfet_01v8 (1)               |sky130_fd_pr__pfet_01v8 (1)
Number of devices: 2                      |Number of devices: 2
Number of nets: 4                         |Number of nets: 4
------------------------------------------------------------------------------------
Circuits match uniquely.
Property errors were found.
Netlists match uniquely.
```

INFERENCE:

The schematic and layout for the inverter is done in Xschem and Magic respectively. Netlist is extracted from both and they match exactly. The layout is carefully done to avoid Design Rule violation and is shown with a green tick in the layout window. Netgen is used to compare the netlists and generate LVS report.

RESULT:

Thus the layout for inverter is successfully done in Magic. The layout has passed DRC check and LVS check.