SCHOOL OF ENGINEERING AND TECHNOLOGY

Department of Electronics and Communication Engineering

EC631 - VLSI DESIGN LABORATORY

NAME: PREM KUMAR R

REG. NO: 1960628

YEAR/BRANCH: 3/ECE

SEMESTER: 6th

CHRIST (DEEMED TO BE UNIVERSITY)
SCHOOL OF ENGINEERING AND TECHNOLOGY
Mysore Road, Kanmanike, Bangalore – 560074

LABORATORY CERTIFICATE

This is to certify that **Mr./Ms**. …Prem Kumar R……..…… has satisfactorily completed the course of experiments in **EC631 - VLSI Design Laboratory** prescribed by the department of **Electronics and Communication Engineering, CHRIST (Deemed to be University)** for **VI** semester **Bachelor of Technology** Course in the laboratory of this university during the year **2021 – 2022**.

Signature of Staff-In charge

Signature of Head of the Department

Name of the Candidate ……Prem Kumar R…….

Register Number ……1960628……...

Examination Center ……Kengeri……...

Date of Practical Examination ……28/04/22……...

Signature of the Examiners:

1.

2.

SYLLABUS

EC631 - VLSI DESIGN LABORATORY

LIST OF EXPERIMENTS:

1) Design Entry and Simulation of Combinational Logic circuits

a) Basic logic gates

b) Multiplexer and Demultiplexer

c) Encoder and Decoder

d) Half adder and Full adder

e) Half Subtractor and Full Subtractor

f) 8 bit adder

g) 4 bit multiplier

2) Design Entry and Simulation of Sequential Logic Circuits

a) Flip-Flops

b) Counters

c) Registers

3) Synthesis, P&R and Post P&R simulation for all the blocks/codes developed in Expt. No. 1 and No. 2.

4) Design Entry and Simulation of traffic signal controller using Xilinx ISE Design suite and implementing the same on Spartan FPGA.

5) Schematic and Layout of a simple CMOS inverter, parasitic extraction and simulation.

6) Design and simulation of pipelined serial and parallel adder to add/ subtract 8 number of size, 12 bits each in 2's complement.

7) Design and Implement a 4 digit seven segment display.

CONTENTS

Completed/Incomplete Staff In-Charge Signature

EX.NO: 1 DESIGN ENTRY AND SIMULATION OF COMBINATIONAL LOGIC CIRCUITS

DATE: 12/01/2022

AIM:

To verify the functionality and timing of your design or portion of your design. To interpret Verilog code into circuit functionality and displays logical results of the described HDL to determine correct circuit operation and to create and verify complex functions in a relatively small amount of time.

TOOLS REQUIRED:

1. Xilinx Vivado Design Suite: WebEdition

PROCEDURE:

1. Start by clicking **Vivado Design Suite: WebEdition**

2. Go to **File → new project → Enter project name**, select the top level source as **HDL** & click **next**.

3. Enter **device properties** as

Product Category : General
Purpose Family : Kintex 7
Device :
xc7k70t Package
:fbg484 Speed :
-1
Top Level Source Type : HDL
Synthesis Tool : XST (VHDL/Verilog)
Simulator : ISim (VHDL/Verilog)
Preferred Language : Verilog

& **Click next**.

4. **Right click** the device name (XCS3540) in the source window to create

new source.

5. Select **Verilog module** and enter **file name** in the new source window & click **next**.

6. Write the **Verilog code** in the **Verilog editor window**.

7. Write the **testbench** by create **new source simulation source.**

8. Run Check syntax through **Process window→ synthesize→** double click **Behavioral Check Syntax→** and removes error if present, with proper syntax & coding.

9. Click on the symbol of FPGA device and then right click→ click on **new source**.

10. Select the desired parameters for simulating the design. In this case **combinational circuit** and **simulation time** click **finish**.

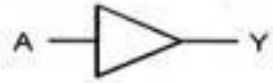
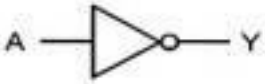
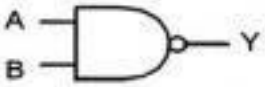
11. Assign **all input signal** (**high or low**) using just **click** on this and save file.

12. From the **source process window**. Click **Behavioral simulation** from drop-down menu.

13. Double click the **Simulation Behavioral Model**.

14.Verify your design in **wave window** by seeing behavior of output signal with respect to input signal.

BASIC LOGIC GATES:

Logic function	Logic symbol	Truth table	Boolean expression
Buffer		A | Y 0 | 0 1 | 1	$Y = A$
Inverter (NOT gate)		A | Y 0 | 1 1 | 0	$Y = \overline{A}$
2-input AND gate		A B | Y 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1	$Y = A \bullet B$
2-input NAND gate		A B | Y 0 0 | 1 0 1 | 1 1 0 | 1 1 1 | 0	$Y = \overline{A \bullet B}$
2-input OR gate		A B | Y 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1	$Y = A + B$
2-input NOR gate		A B | Y 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 0	$Y = \overline{A + B}$
2-input EX-OR gate		A B | Y 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 0	$Y = A \oplus B$
2-input EX-NOR gate		A B | Y 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 1	$Y = \overline{A \oplus B}$

PROGRAM:

a) BASIC LOGIC GATES:

1) GATE LEVEL MODELING:

```
22  //1960628 prem
23  module basic_gates(
24      input i1,
25      input i2,
26      output not_out,
27      output and_out,
28      output or_out,
29      output xor_out,
30      output nor_out,
31      output nand_out,
32      output xnor_out,
33      output buf_out
34      );
35
36  not no(not_out,i1);
37  and an(and_out,i1,i2);
38  or ort(or_out,i1,i2);
39  xor xort(xor_out,i1,i2);
40  nor nort(nor_out,i1,i2);
41  nand nandt(nand_out,i1,i2);
42  xnor (xnort,i1,i2);
43  buf buffet(buf_out,i1);
44  endmodule
45
```

2) DATAFLOW LEVEL MODELING:

```
47  //DATAFLOW
48  //1960628 PREM
49  module basic_gates(
50      input i1,
51      input i2,
52      output not_out,
53      output and_out,
54      output or_out,
55      output xor_out,
56      output nor_out,
57      output nand_out,
58      output xnor_out,
59      output buf_out
60      );
61
62  assign not_out =~i1;
63  assign and_out =i1&i2;
64  assign or_out=i1|i2;
65  assign xor_out=i1^i2;
66  assign nor_out=~(i1|i2);
67  assign nand_out= ~(i1&i2);
68  assign xnor_out=~(i1^i2);
69  assign buf_out = i1;
70  endmodule
```

BEHAVIORAL LEVEL MODELING:

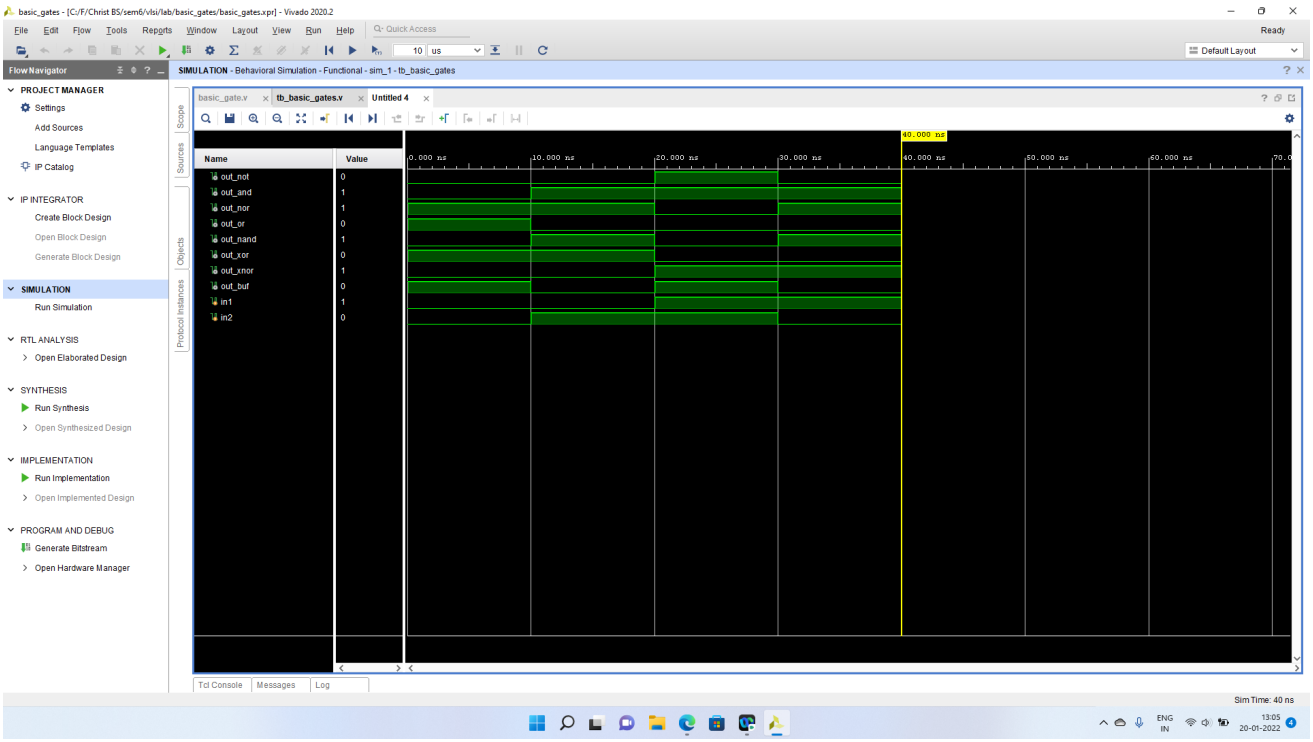
```
77  //1960628 Prem
78  //behavioural basic gate
79  module basic_gate(
80      input in1,
81      input in2,
82      output reg out_and,
83      output reg out_or,
84      output reg out_nand,
85      output reg out_nor,
86      output reg out_xor,
87      output reg out_not,
88      output reg out_buf,                                  _out, buf_out;
89      output reg out_xnor
90
91      );
92
93  always @(in1,in2)
94  begin
95  if(in1==0&&in2==0)
96  begin
97  out_not = 1;
98  out_and = 0;
99  out_nand = 1;
100 out_or = 0;
101 out_nor = 1;
102 out_xor = 0;
103 out_xnor = 1;
104 out_buf = 0;
105 end
106 if(in1==1&&in2==0)
107 begin
108 out_not = 0;
109 out_and = 0;
110 out_nand = 1;
111 out_or = 0;
112 out_nor = 1;
113 out_xor = 0;
114 out_xnor = 1;
115 out_buf = 1;
116 end
```

```
 99  end
100  if(i1==0&&i2==1)
101  begin
102   not_out =1;
103   and_out =0;
104   or_out=0;
105   xor_out=0;
106   nor_out=1;
107   nand_out=1;
108   xnor_out=1;
109   buf_out =0;
110  end
111  if(i1==1&&i2==1)
112  begin
113   not_out =0;
114   and_out =0;
115   or_out=1;
116   xor_out=0;
117   nor_out=0;
118   nand_out=0;
119   xnor_out=1;
120   buf_out =1;
121  end
122  endmodule
```

TEST BENCH:

```
21
22  //1960628 PREM
23  module tb_basic_gates();
24  wire not_out,and_out,or_out,xor_out,nor_out,nand_out,xnor_out, buf_out;
25  reg i1,i2;
26
27  basic_gates I1( not_out,and_out,or_out,xor_out,nor_out,nand_out,xnor_out, buf_out,i1,i2);
28  initial
29  begin
30  i1 = 1'b0;
31  i2 = 1'b0;
32  #10
33  i1 = 1'b0;
34  i2 = 1'b1;
35  #10
36  i1 = 1'b1;
37  i2 = 1'b0;
38  #10
39  i1 = 1'b1;
40  i2 = 1'b1;
41  #10
42  $finish;
43  end
44  endmodule
```

SIMULATION OUTPUT:



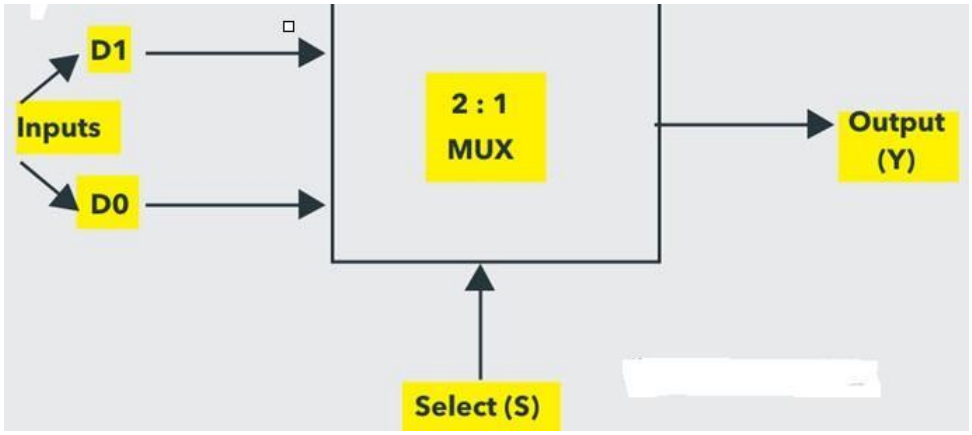
Result:

Thus the Verilog code was scripted, simulated and waveforms were verified using Xilinx Vivado Webpack.

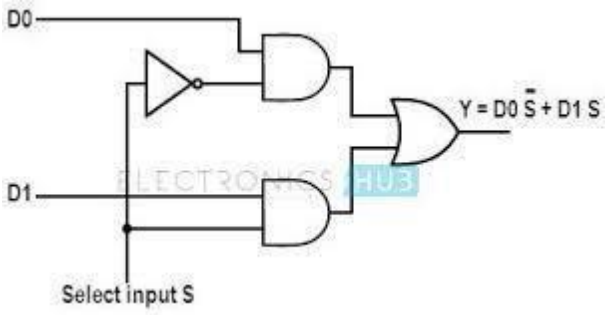
D) Multiplexer and Demultiplexer:

MULTIPLEXER (2x1):

Logic Symbol



Logic Circuit



Truth Table

Select	Inputs		Output
0	0	0	0
0	0	1	1
1	1	0	1
1	1	1	1

PROGRAM

1) GATE LEVEL MODELING:

```
15      //
16      // Revision:
17      // Revision 0.01 - File Created
18      // Additional Comments:
19      //
20      //////////////////////////////////////////////////////////////////////////////////
21
22      //1960628 PREM
23      module mux_demux(d0,d1,s,y);
24
25      input d0,d1,s;
26      output y;
27
28      wire w0,w1,w3;
29      not not_0(w0,s);
30      and and_1(w2,w0,d0);
31      and and_2(w3,d1,s);
32      or out(y,w2,w3);
33
34      endmodule
35
```

2) DATAFLOW LEVEL MODELING:

```
21
22      //1960628 PREM
23      // DATAFLOW MODELLING
24      module mux_demux(d0,d1,s,y);
25      input d0,d1,s;
26      output y;
27      wire w0,w1,w3;
28      assign not_0 = ~(s);
29      assign and_0 = not_0&d0;
30      assign and_1 = d1&s;
31      assign y = and_0 + and_1;
32      endmodule
33
```

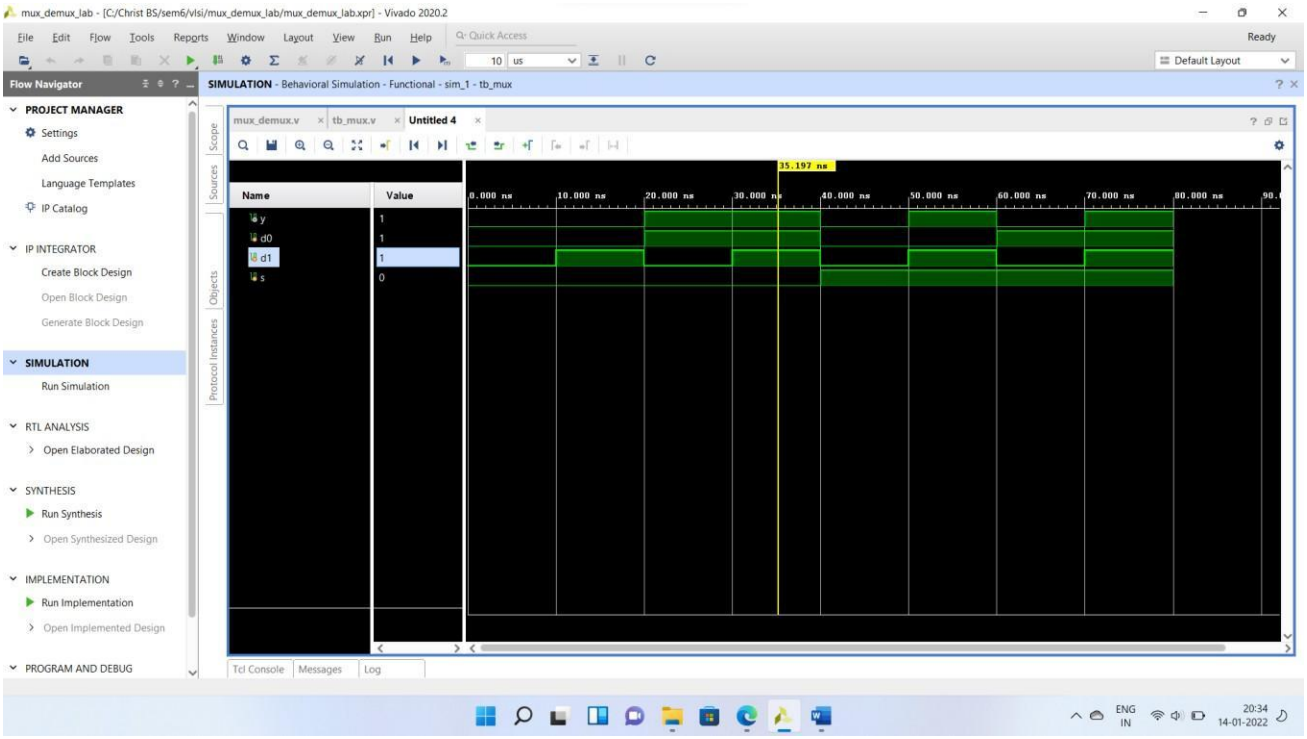
3) BEHAVIORAL LEVEL MODELING:

```
21
22      //1960628 PREM
23      // BEHAVIORAL,DATAFLOW,MODELLING
24      module mux_demux(d0,d1,s,y);
25      input d0,d1,s;
26      output y;
27      /*data flow commented out
28      wire w0,w1,w3;
29      assign not_0 = ~(s);
30      assign and_0 = not_0&d0;
31      assign and_1 = d1&s;
32      assign y = and_0 + and_1;*/
33      assign y = (s==1)?d1:d0;
34      endmodule
35
```

TESTBENCH

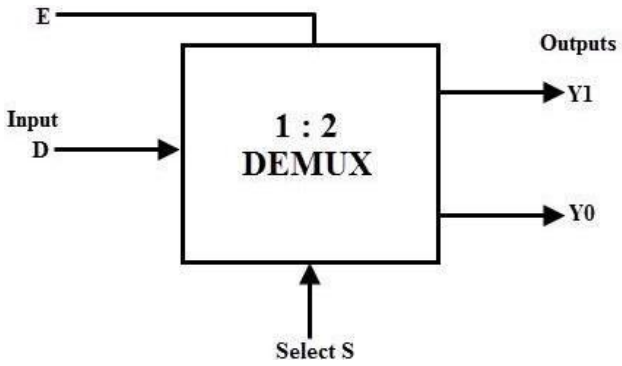
```
21
22      //1960628 PREM
23      module tb_mux();
24      wire y;
25      reg d0,d1,s;
26      mux_demux I1(d0,d1,s,y);
27      initial
30      begin
29      d0 = 1'b0;
30      d1 = 1'b0;
31      s = 1'b0;
32      #10
33      d0 = 1'b0;
34      d1 = 1'b1;
35      s = 1'b0;
36      #10
37      d0 = 1'b1;
38      d1 = 1'b0;
39      s = 1'b0;
40      #10
41      d0 = 1'b1;
42      d1 = 1'b1;
43      s = 1'b0;
44      #10
45      d0 = 1'b0;
46      d1 = 1'b0;
47      s = 1'b1;
48      #10
49      d0 = 1'b0;
50      d1 = 1'b1;
51      s = 1'b1;
52      #10
```

```
52      #10
53      d0 = 1'b1;
54      d1 = 1'b0;
55      s = 1'b1;
56      #10
57      d0 = 1'b1;
58      d1 = 1'b1;
59      s = 1'b1;
60      #10
61      $finish;
62      end
63      endmodule
64
```

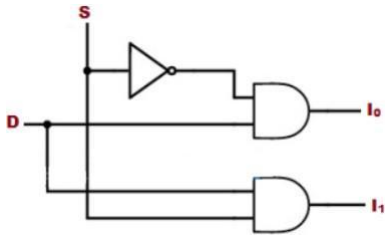


DE-MULTIPLEXER (1x2):

Logic Symbol



Logic Circuit



Truth Table

Select	Input	Outputs	
S	D	Y_2	Y_1
0	0	0	0
0	1	0	1
1	0	0	0
1	1	1	0

PROGRAM

1) **GATE LEVEL MODELING:**

```
//1960628 PREM
//GATE LEVEL MODELLING
module demux_comb(y0,y1,d,s);
input s;
input d;
output y0;
output y1;
wire w0;
not not_0(w0,s);
and and_1(y0,w0,d);
and and_2(y1,d,s);
endmodule
```

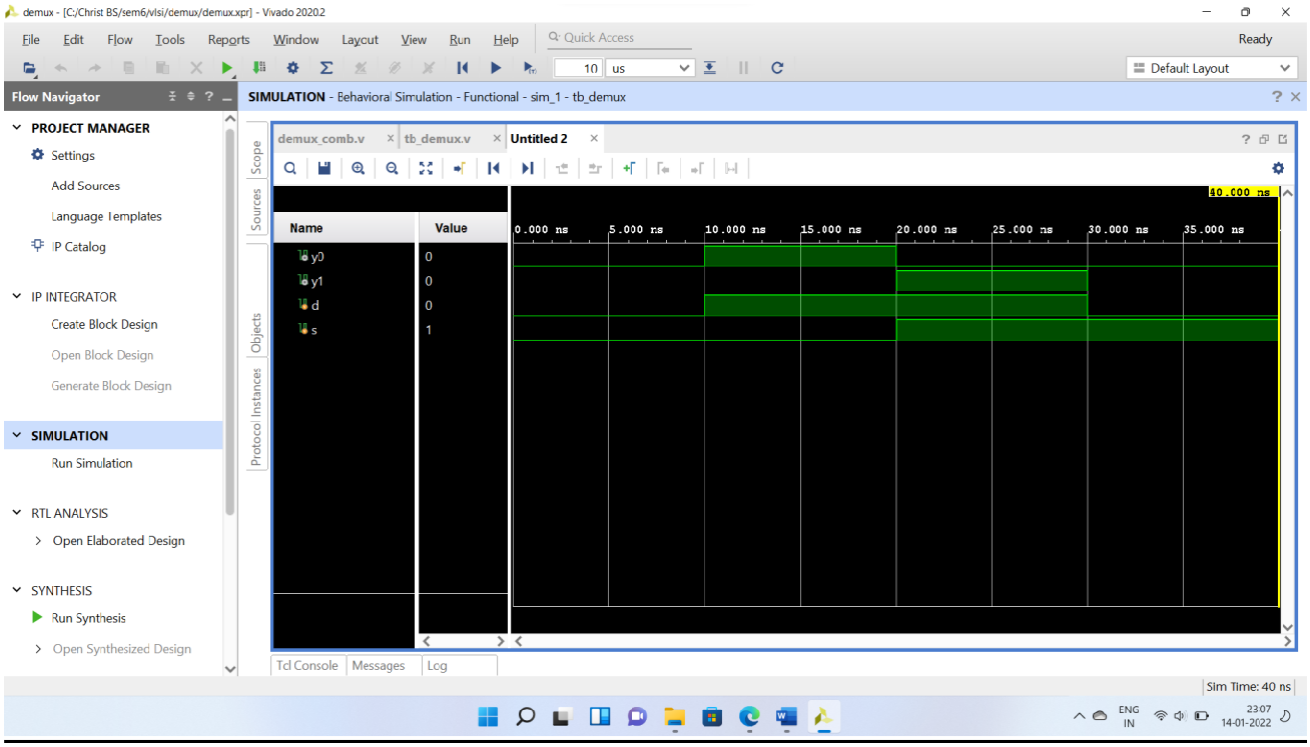
2) DATAFLOW LEVEL MODELING:

```
//DATA FLOW MODELLING
//1960628 PREM
module demux_comb(y0,y1,d,s);
input s;
input d;
output y0;
output y1;
//DATAFLOW
assign not_0 = ~s;
assign y0 = not_0&d;
assign y1 = s&d;
//GATE LEVEL
//wire w0;
//not not_0(w0,s);
//and and_1(y0,w0,d);
//and and_2(y1,d,s);
```

TESTBENCH

```
//1960628 PREM
//TEST BENCH FOR DEMUX 1X2
module tb_demux();
wire y0,y1;
reg d,s;
demux_comb I1(y0,y1,d,s);
initial
begin
d = 1'b0;
s = 1'b0;
#10
d = 1'b1;
s = 1'b0;
#10
d = 1'b1;
s = 1'b1;
#10
d = 1'b0;
s = 1'b1;
#10
$finish;
end
endmodule
```

SIMULATION OUTPUT



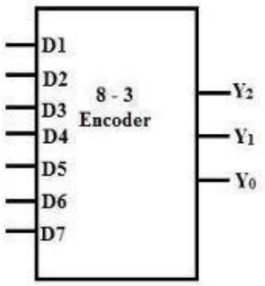
RESULT : A Mux and Demux were modelled in the Xilinx Vivado simulator and their output was verified.

ENCODERS AND DECODERS

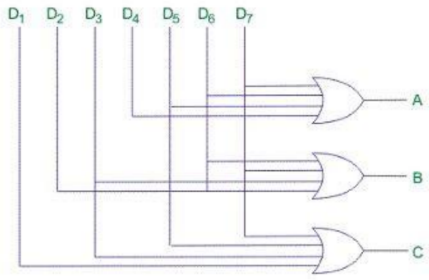
8x3 Encoder:

Encoder:

Logic Symbol



Logic Circuit



Truth Table

Inputs								Outputs		
I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7	y_2	y_1	y_0
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

Expression

$y0 = I1 + I3 + I5 + I7$

$y1 = I2 + I3 + I6 + I7$

$y2 = I4 + I5 + I6 + I7$

PROGRAM

Gate Level

```
22  //1960628 Prem
23  //encoder 8x3 Gate level
24  module encoder(
25      input d0,
26      input d1,
27      input d2,
28      input d3,
29      input d4,
30      input d5,
31      input d6,
32      input d7,
33      output y0,
34      output y1,
35      output y2
36      );
37  or out1(y0,d7,d5,d3,d1);
38  or out3(y2,d7,d6,d5,d4);
39  or out2(y1,d7,d6,d2,d3);
40  endmodule
```

Data Flow

```
43  //1960628 Prem
44  //encoder 8x3 Data Flow
45  module encoder(
46      input d0,
47      input d1,
48      input d2,
49      input d3,
50      input d4,
51      input d5,
52      input d6,
53      input d7,
54      output y0,
55      output y1,
56      output y2
57      );
58  assign y0 = d7+d5+d3+d1;
59  assign y1 = d7+d6+d2+d3;
60  assign y2 = d7+d6+d5+d4;
61  endmodule
```

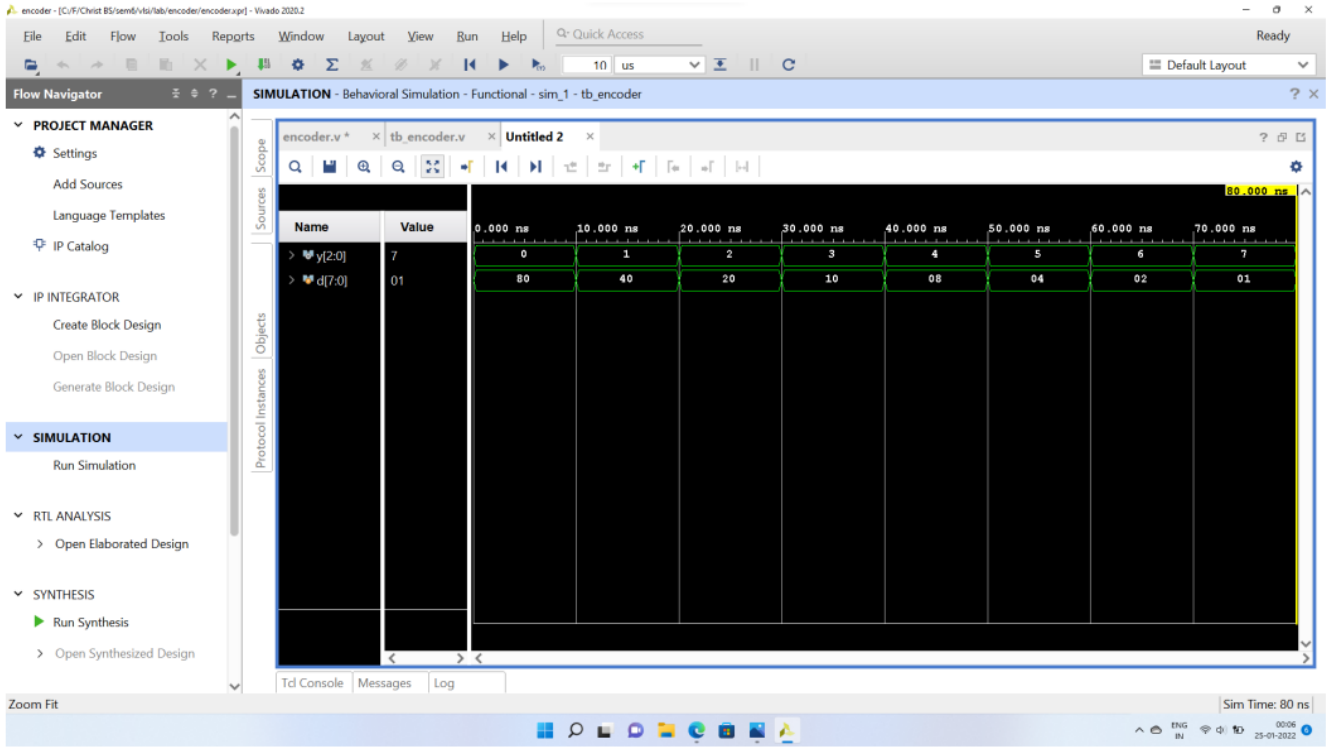
Behavioural

```
//1960628 Prem
//encoder 8x3 Behavioural
module encoder(
    output reg [2:0]y,
    input [7:0]d
    );
always @(d,y)
    begin
        case(d)
        8'b10000000:begin y=3'b000; end
        8'b01000000:begin y=3'b001; end
        8'b00100000:begin y=3'b010; end
        8'b00010000:begin y=3'b011; end
        8'b00001000:begin y=3'b100; end
        8'b00000100:begin y=3'b101; end
        8'b00000010:begin y=3'b110; end
        8'b00000001:begin y=3'b111; end
        endcase
    end
endmodule
```

TESTBENCH

```
22  //1960628 Prem
23  //encoder 8x3 Testbench
24  module tb_encoder();
25  wire [2:0]y;
26  reg [7:0]d;
27
28  encoder I1(y,d);
29  initial
30  begin
31  d = 8'b10000000;
32  #10
33  d = 8'b01000000;
34  #10
35  d = 8'b00100000;
36  #10
37  d = 8'b00010000;
38  #10
39  d = 8'b00001000;
40  #10
41  d = 8'b00000100;
42  #10
43  d = 8'b00000010;
44  #10
45  d = 8'b00000001;
46  #10
47  $finish();
48  end
49  endmodule
```

OUTPUT



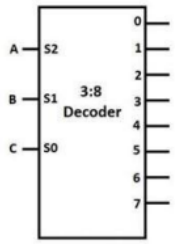
Result:

Thus the Verilog code was scripted, simulated and waveforms were verified using Xilinx Vivado Webpack.

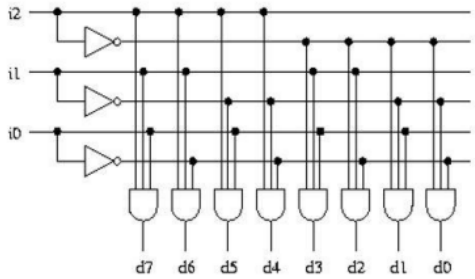
3x8 Decoder:

Decoder:

Logical Symbol



Logic Circuit



Truth Table

A_2	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

Expression:

D0 = A2' A1' A0'

D1 = A2' A1' A0

D2 = A2' A1 A0'

D3 = A2' A1 A0

D4 = A2 A1' A0'

D5 = A2 A1' A0

D6 = A2 A1 A0'

D7 = A2 A1 A0

PROGRAM

Gate Level

```
21  //1960628 Prem
22  //Demux 3x8 Gate level
23  module decoder_3x8(
24      input s0,
25      input s1,
26      input s2,
27      output d0,
28      output d1,
29      output d2,
30      output d3,
31      output d4,
32      output d5,
33      output d6,
34      output d7
35      );
36  wire w0,w1,w2;
37  not no1(w0,s0);
38  not no2(w1,s1);
39  not no3(w2,s2);
40
41  and and0(d0,w2,w1,w0);
42  and and1(d1,w2,w1,s0);
43  and and2(d2,w2,s1,w0);
44  and and3(d3,w2,s1,s0);
45  and and4(d4,s2,w1,w0);
46  and and5(d5,s2,w1,s0);
47  and and6(d6,s2,s1,w0);
48  and and7(d7,s2,s1,s0);
49  endmodule
```

Data Flow

```
52  //1960628 Prem
53  //3x8 decoder Dataflow
54  module decoder_3x8(
55      input s0,
56      input s1,
57      input s2,
58      output d0,
59      output d1,
60      output d2,
61      output d3,
62      output d4,
63      output d5,
64      output d6,
65      output d7
66      );
67
68  assign w0 = ~s0;
69  assign w1 = ~s1;
70  assign w2 = ~s2;
71  assign d0 = w2&w1&w0;
72  assign d1 = w2&w1&s0;
73  assign d2 = w2&s1&w0;
74  assign d3 = w2&s1&s0;
75  assign d4 = s2&w1&w0;
76  assign d5 = s2&w1&s0;
77  assign d6 = s2&s1&w0;
78  assign d7 = s2&s1&s0;
79  endmodule
```

Behavioural

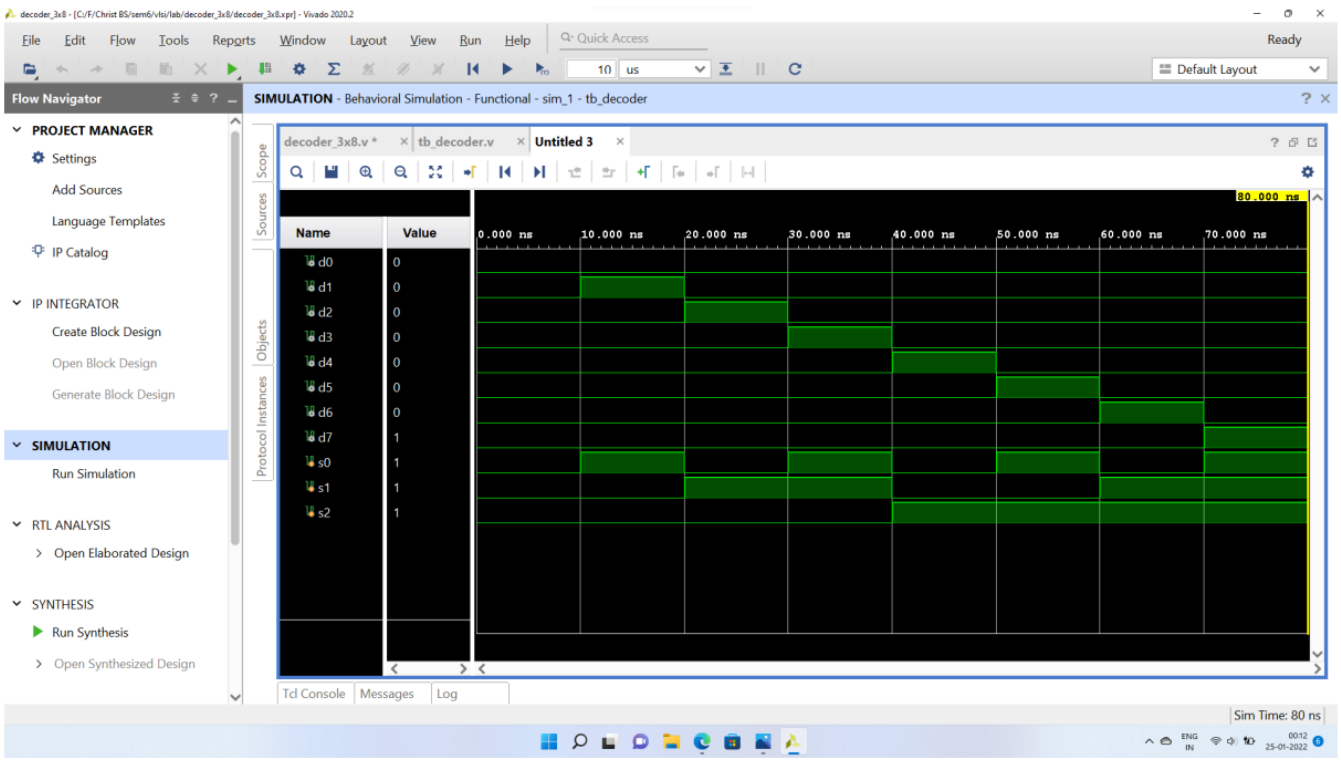
```
83  //1960628 Prem
84  //3x8 Decoder Behavioural Modelling
85  module decoder_3x8(
86      input s0,
87      input s1,
88      input s2,
89      output reg d0,
90      output reg d1,
91      output reg d2,
92      output reg d3,
93      output reg d4,
94      output reg d5,
95      output reg d6,
96      output reg d7
97      );
98  always @(s0,s1,s2)
99      begin
100         case ({s2,s1,s0})
101         3'b000: begin d0=0;d1=0;d2=0;d3=0;d4=0;d5=0;d6=0;d7=0;end
102         3'b001: begin d0=0;d1=1;d2=0;d3=0;d4=0;d5=0;d6=0;d7=0;end
103         3'b010: begin d0=0;d1=0;d2=1;d3=0;d4=0;d5=0;d6=0;d7=0;end
104         3'b011: begin d0=0;d1=0;d2=0;d3=1;d4=0;d5=0;d6=0;d7=0;end
105         3'b100: begin d0=0;d1=0;d2=0;d3=0;d4=1;d5=0;d6=0;d7=0;end
106         3'b101: begin d0=0;d1=0;d2=0;d3=0;d4=0;d5=1;d6=0;d7=0;end
107         3'b110: begin d0=0;d1=0;d2=0;d3=0;d4=0;d5=0;d6=1;d7=0;end
108         3'b111: begin d0=0;d1=0;d2=0;d3=0;d4=0;d5=0;d6=0;d7=1;end
109         default: begin d0=0;d1=0;d2=0;d3=0;d4=0;d5=0;d6=0;d7=0;end
110         endcase
111     end
112
113 endmodule
```

TESTBENCH

```
22  //1960628 Prem
23  //3x8 decoder testbench
24  module tb_decoder();
25  wire d0,d1,d2,d3,d4,d5,d6,d7;
26  reg s0,s1,s2;
27
28  decoder_3x8 I1(s0,s1,s2,d0,d1,d2,d3,d4,d5,d6,d7);
29  initial
30  begin
31  s0 = 1'b0;
32  s1 = 1'b0;
33  s2 = 1'b0;
34  #10
35  s0 = 1'b1;
36  s1 = 1'b0;
37  s2 = 1'b0;
38  #10
39  s0 = 1'b0;
40  s1 = 1'b1;
41  s2 = 1'b0;
42  #10
43  s0 = 1'b1;
44  s1 = 1'b1;
45  s2 = 1'b0;
46  #10
47  s0 = 1'b0;
48  s1 = 1'b0;
49  s2 = 1'b1;
50  #10
51  s0 = 1'b1;
52  s1 = 1'b0;
53  s2 = 1'b1;
54  #10
55  s0 = 1'b0;
56  s1 = 1'b1;
57  s2 = 1'b1;
```

```
58    #10
59    s0 = 1'b1;
60    s1 = 1'b1;
61    s2 = 1'b1;
62    #10
63    $finish();
64  end
65  endmodule
```

OUTPUT

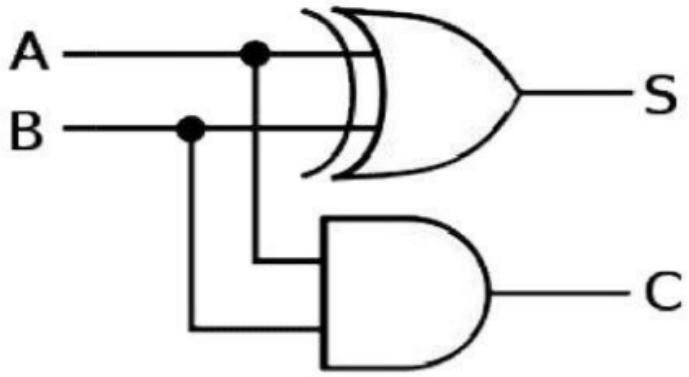


Result: Thus the Verilog code was scripted, simulated and waveforms were verified using Xilinx Vivado Webpack

D) HALF ADDER AND FULL ADDER

b) HALF ADDER:

Logic Circuit



Truth Table

Input		Output	
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Expression

$$\text{Sum} = A \oplus B$$

$$\text{Carry} = A.B$$

Gate Level

```
22  //1960628 Prem half adder gate level
23  module hagate(
24      input a,
25      input b,
26      output s,
27      output c
28      );
29  xor xort(s,a,b);
30  and and1(c,a,b);
31  endmodule
```

Data Flow

```
22  //1960628 Prem Half adder dataflow
23  module hadataflow(
24      input a,
25      input b,
26      output s,
27      output c
28      );
29  assign s = a^b;
30  assign c = a&b;
31
32  endmodule
33
```

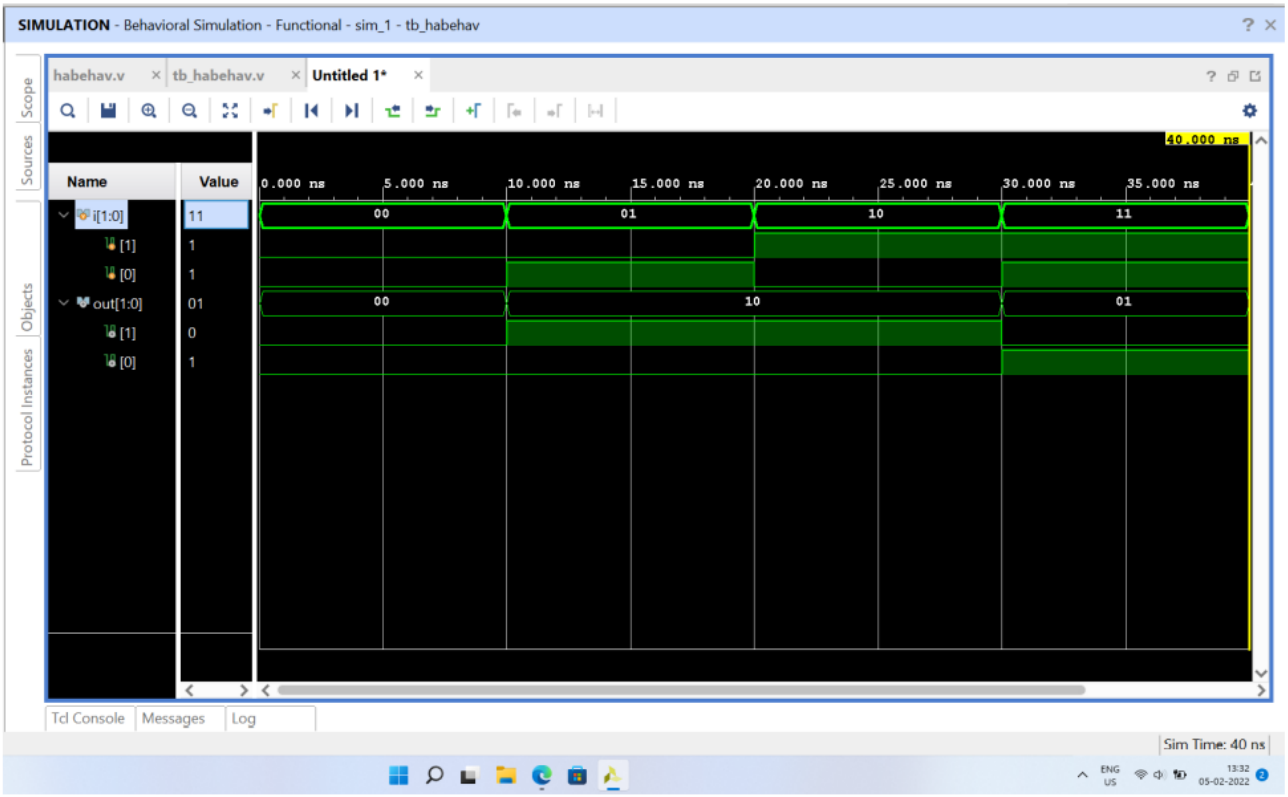
Behavioural

```
22 //1960628 prem half adder behavioural
23 module hsbehav(
24     input [1:0]a,
25     output reg [1:0]d
26     );
27 always @(d,a)
28     begin
29         case(a)
30         2'b00: begin d = 2'b00; end
31         2'b01: begin d = 2'b11; end
32         2'b10: begin d = 2'b10; end
33         2'b11: begin d = 2'b00; end
34         endcase
35     end
36 endmodule
37
```

TESTBENCH

```
22  //1960628 Prem Half Adder testbench
23  module tb_hsbehav();
24  wire [1:0]d;
25  reg [1:0]a;
26
27  hsbehav I1(a,d);
28  initial
29  begin
30  a = 2'b00;
31  #10
32  a = 2'b01;
33  #10
34  a = 2'b10;
35  #10
36  a = 2'b11;
37  #10
38  $finish();
39  end
40
41  endmodule
42
```

OUTPUT

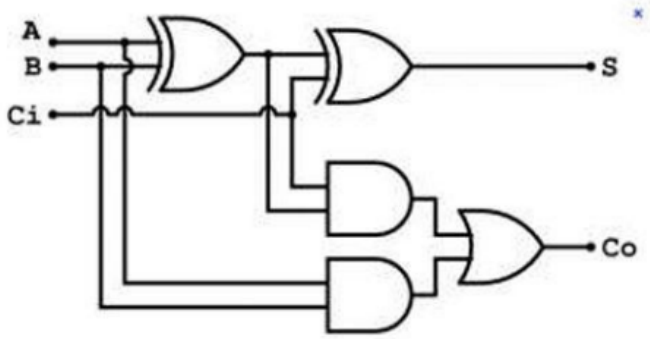


Result:

Thus the Verilog code was scripted, simulated and waveforms were verified using Xilinx Vivado Webpack.

b) FULL ADDER:

Logic Circuit



Truth Table

Input			Output	
A	B	Cin	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Expression

$$Sum = A \oplus B \oplus Cin$$

$$Cout = AB + BCin + CinA$$

PROGRAM

Gate Level

```
22      //1960628 Prem Full Adder Gate Level
23      module fulladder(
24          input a,
25          input b,
26          input ci,
27          output s,
28          output co
29          );
30      wire w0,w1,w2;
31      xor xort(w0,a,b);
32      xor xort1(s,w0,ci);
33      and and1(w1,ci,w0);
34      and and2(w2,a,b);
35      or ort(co,w1,w2);
36
37      endmodule
```

Data Flow

```
22  //1960628 Prem Full adder Dataflow
23  module fa_dataflow(
24      input a,
25      input b,
26      input ci,
27      output s,
28      output co
29      );
30  assign s = a^b^ci;
31  assign co = (a&b) | (b&ci) | (ci&a);
32  endmodule
```

Behavioural

```
22  //1960628 Prem full adder behav
23  module fa_behav(
24      input a,
25      input b,
26      input ci,
27      output reg s,
28      output reg co
29      );
30  always @(a,b,ci)
31  begin
32  if (a == 1'b0 & b == 1'b0 & ci == 1'b0)
33  begin
34  s = 1'b0;
35  co = 1'b0;
36  end
37  if (a == 1'b0 & b == 1'b0 & ci == 1'b1)
38  begin
39  s = 1'b1;
40  co = 1'b0;
41  end
42  if (a == 1'b0 & b == 1'b1 & ci == 1'b0)
43  begin
44  s = 1'b1;
45  co = 1'b0;
46  end
47  if (a == 1'b0 & b == 1'b0 & ci == 1'b1)
48  begin
49  s = 1'b0;
50  co = 1'b1;
51  end
52  if (a == 1'b1 & b == 1'b0 & ci == 1'b0)
```

```
54      s = 1'b1;
55      co = 1'b0;
56    end
57    if (a == 1'b1 & b == 1'b0 & ci == 1'b1)
58    begin
59      s = 1'b0;
60      co = 1'b1;
61    end
62    if (a == 1'b1 & b == 1'b1 & ci == 1'b0)
63    begin
64      s = 1'b0;
65      co = 1'b1;
66    end
67    if (a == 1'b1 & b == 1'b1 & ci == 1'b1)
68    begin
69      s = 1'b1;
70      co = 1'b1;
71    end
72  end
73  endmodule
```

TESTBENCH

```
22      //1960628 Prem Testbench Full adder
23      module tb_fa();
24      wire s,co;
25      reg a,b,ci;
26
27      fulladder I1(a,b,ci,s,co);
28      initial
29      begin
30      a = 1'b0;
31      b = 1'b0;
32      ci = 1'b0;
33      #10
34      a = 1'b0;
35      b = 1'b1;
36      ci = 1'b0;
37      #10
38      a = 1'b1;
39      b = 1'b0;
40      ci = 1'b0;
41      #10
42      a = 1'b1;
43      b = 1'b1;
44      ci = 1'b0;
45      #10
46      a = 1'b0;
47      b = 1'b0;
48      ci = 1'b1;
49      #10
50      a = 1'b0;
51      b = 1'b1;
52      ci = 1'b1;
53      #10
54      a = 1'b1;
55      b = 1'b0;
56      ci = 1'b1;
57      #10
```

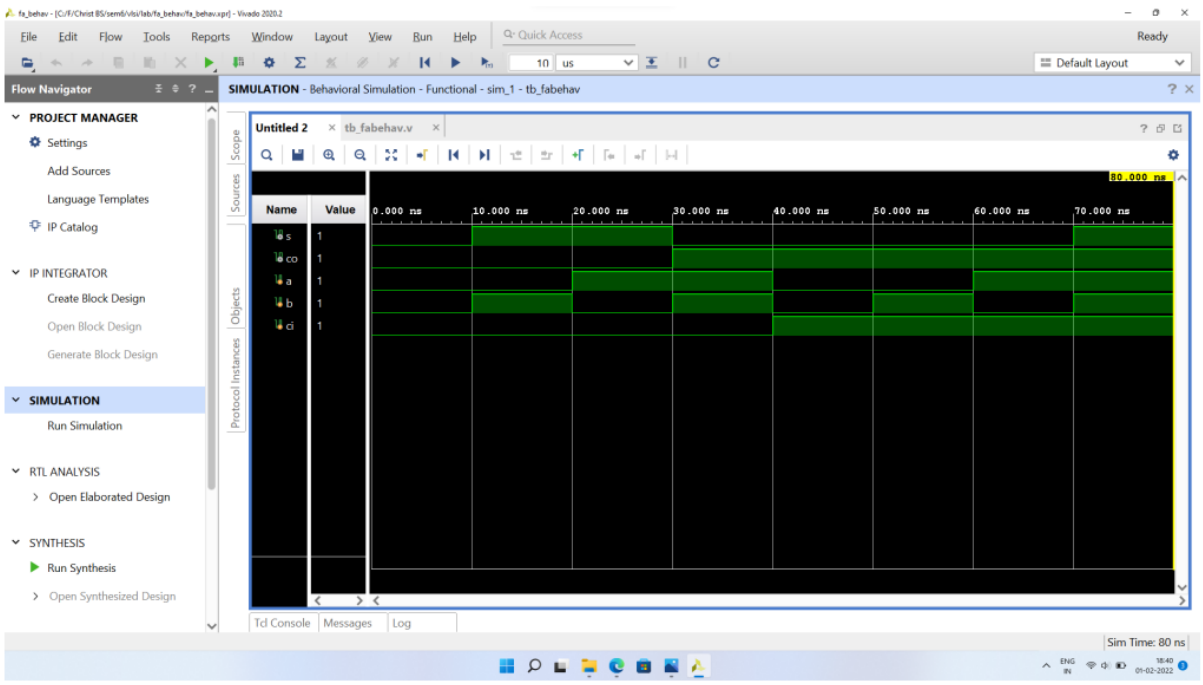
```
58        a = 1'b1;
59        b = 1'b1;
60        ci = 1'b1;
61        $finish();
62      end
63    endmodule
```

Behavioural testbench

```
23  //1960628 Prem Testbench Full adder
24  module tb_fabehav();
25  wire s,co;
26  reg a,b,ci;
27
28  fa_behav I1(a,b,ci,s,co);
29  initial
30  begin
31  a = 1'b0;
32  b = 1'b0;
33  ci = 1'b0;
34  #10
35  a = 1'b0;
36  b = 1'b1;
37  ci = 1'b0;
38  #10
39  a = 1'b1;
40  b = 1'b0;
41  ci = 1'b0;
42  #10
43  a = 1'b1;
44  b = 1'b1;
45  ci = 1'b0;
46  #10
47  a = 1'b0;
48  b = 1'b0;
49  ci = 1'b1;
50  #10
51  a = 1'b0;
52  b = 1'b1;
53  ci = 1'b1;
```

```
54  #10
55  a = 1'b1;
56  b = 1'b0;
57  ci = 1'b1;
58  #10
59  a = 1'b1;
60  b = 1'b1;
61  ci = 1'b1;
62  #10
63  $finish();
64  end
65  endmodule
```

OUTPUT



Result:

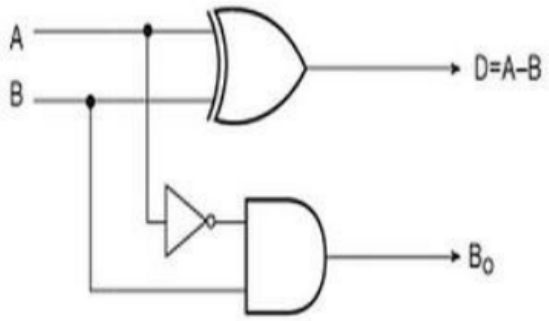
Thus the Verilog code was scripted, simulated and waveforms were verified using Xilinx Vivado Webpack.

E) HALF SUBTRACTOR AND FULL SUBTRACTOR

Half Subtractor:

c) HALF SUBTRACTOR:

Logic Circuit



Truth Table

A	B	D	B_0
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Expression

$$D = A \oplus B$$

$$B = A'.B$$

PROGRAM

Gate Level

```
22      //1960628 Prem Half Subtracotr Gate level
23      module hsgate(
24          input a,
25          input b,
26          output d,
27          output bo
28          );
29      wire w0;
30      xor xort(d,a,b);
31      not not1(w0,a);
32      and and1(bo,b,w0);
33      endmodule
```

Data Flow

```
22  //1960628 Prem Half Subtractor Data flow
23  module hsdata(
24      input a,
25      input b,
26      output d,
27      output bo
28      );
29  assign d = a^b;
30  assign bo = (~a)&b;
31  endmodule
```

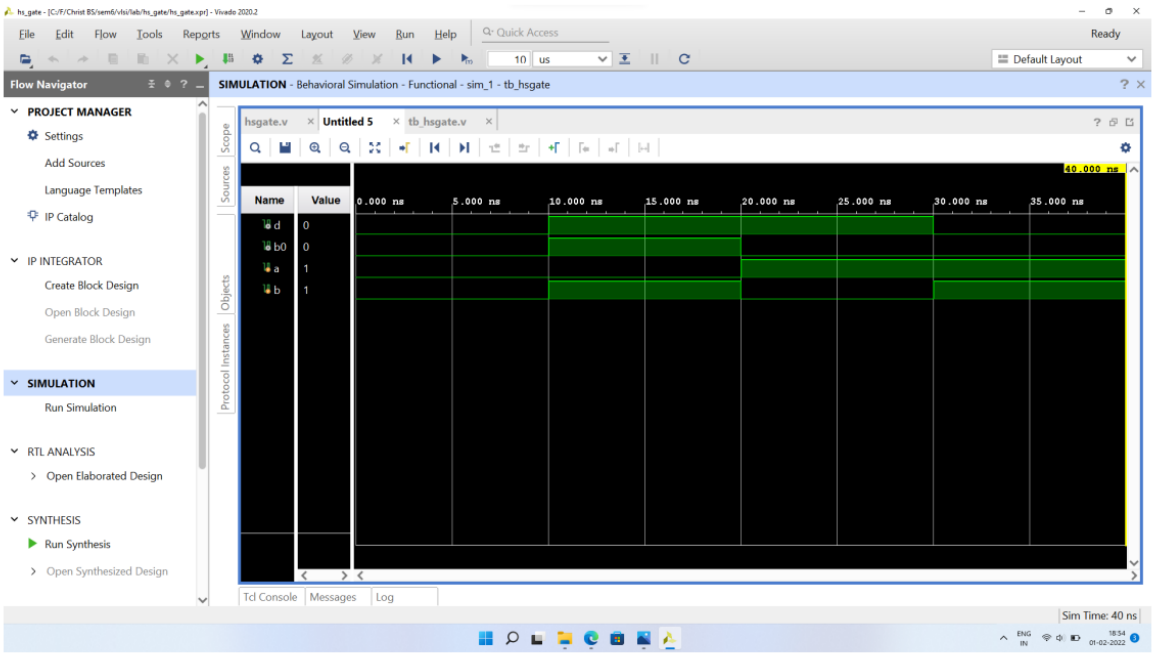
Behavioural

```
22  //1960628 prem half subtracotr behavioural
23  module hsbehav(
24      input [1:0]a,
25      output reg [1:0]d
26      );
27  always @(d,a)
28      begin
29          case(a)
30          2'b00: begin d = 2'b00; end
31          2'b01: begin d = 2'b11; end
32          2'b10: begin d = 2'b10; end
33          2'b11: begin d = 2'b00; end
34          endcase
35      end
36  endmodule
```

TESTBENCH

```
22      //1960628 Prem Half Subtracotr testbench
23      module tb_hsgate();
24      wire d,b0;
25      reg a,b;
26
27      hsgate I1(a,b,d,b0);
28      initial
29      begin
30      a = 1'b0;
31      b = 1'b0;
32      #10
33      a = 1'b0;
34      b = 1'b1;
35      #10
36      a = 1'b1;
37      b = 1'b0;
38      #10
39      a = 1'b1;
40      b = 1'b1;
41      #10
42      $finish();
43      end
44      endmodule
```

OUTPUT

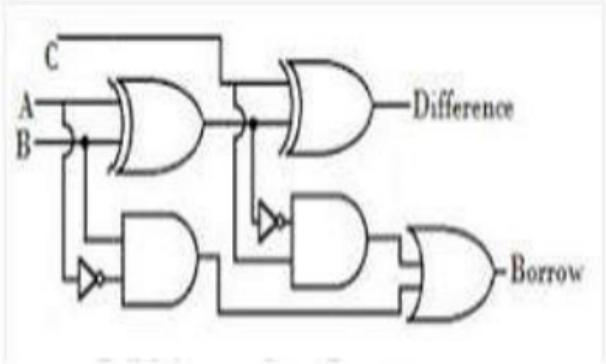


Result:

Thus the Verilog code was scripted, simulated and waveforms were verified using Xilinx Vivado Webpack.

Full Subtractor:

Logic Circuit



Truth Table

Full Subtractor-Truth Table				
Input			Output	
A	B	C	Difference	Borrow
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Expression

$$\text{Difference} = A \oplus B \oplus C$$

$$\text{Borrow} = A'B + BC + A'C$$

PROGRAM

Gate Level

```
22      //1960628 Prem full subtractor gate level
23      module fsgate(
24          input a,
25          input b,
26          input c,
27          output d,
28          output bor
29          );
30      wire w0,w1,w2,w3,w4;
31      xor xort(w0,a,b);
32      xor xort2(d,c,w0);
33      not nort(w1,w0);
34      not nort2(w2,a);
35      and and1(w3,w2,b);
36      and and2(w4,c,w1);
37      or ort(bor,w3,w4);
38      endmodule
39
```

Data Flow

```
22  //1960628 Prem full subtractor dataflow
23  module fs_dataflow(
24      input a,
25      input b,
26      input c,
27      output d,
28      output bor
29      );
30  assign d = a^b^c;
31  assign bor = ((~a)&b)|(b&c)|((~a)&c);
32  endmodule
```

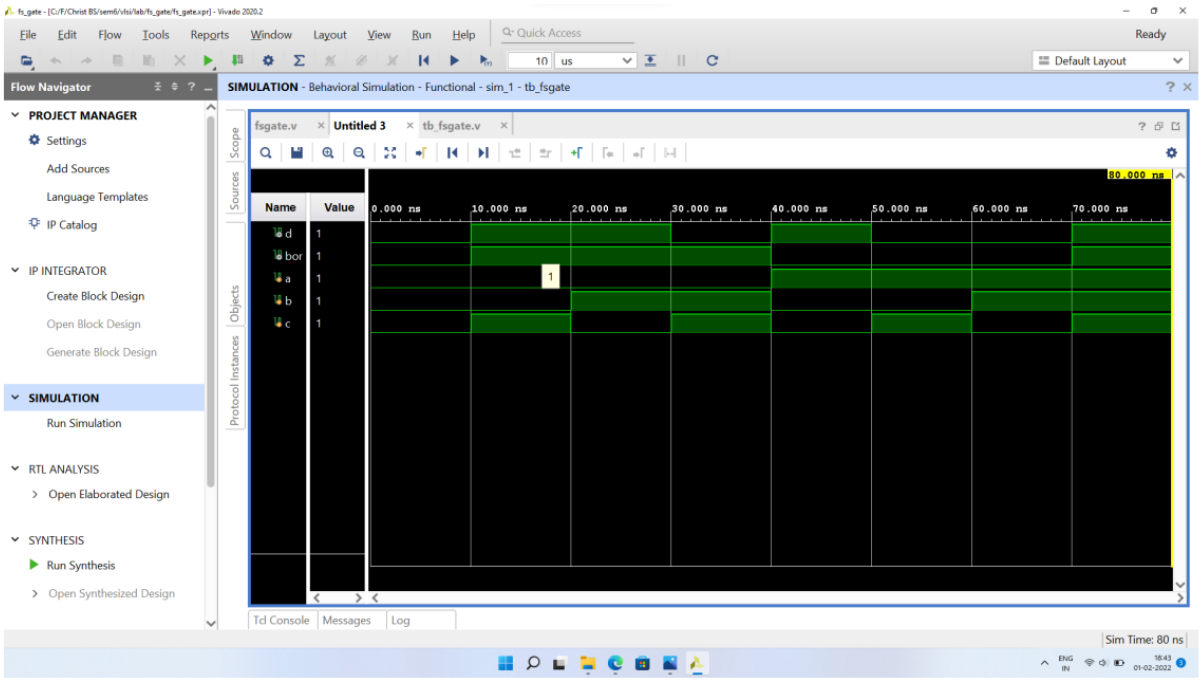
Behavioural

```
22  //1960628 Prem Full Subtractor Behavioural
23  module fsbehav(
24      input [2:0]a,
25      output reg [1:0]d
26      );
27  always@ (a,d)
28      begin
29      case(a)
30      3'b000: begin d = 2'b00; end
31      3'b001: begin d = 2'b11; end
32      3'b010: begin d = 2'b11; end
33      3'b011: begin d = 2'b01; end
34      3'b100: begin d = 2'b10; end
35      3'b101: begin d = 2'b00; end
36      3'b110: begin d = 2'b00; end
37      3'b111: begin d = 2'b11; end
38      endcase
39      end
40  endmodule
```

TESTBENCH

```
22  //1960628 Prem full adder testbemch
23  module tb_fsbehav();
24  wire [1:0]d;
25  reg [2:0]a;
26
27  fsbehav I1(a,d);
28  initial
29  begin
30  a = 3'b000;
31  #10
32  a = 3'b001;
33  #10
34  a = 3'b010;
35  #10
36  a = 3'b011;
37  #10
38  a = 3'b100;
39  #10
40  a = 3'b101;
41  #10
42  a = 3'b110;
43  #10
44  a = 3'b111;
45  #10
46  $finish();
47  end
48
49  endmodule
```

OUTPUT

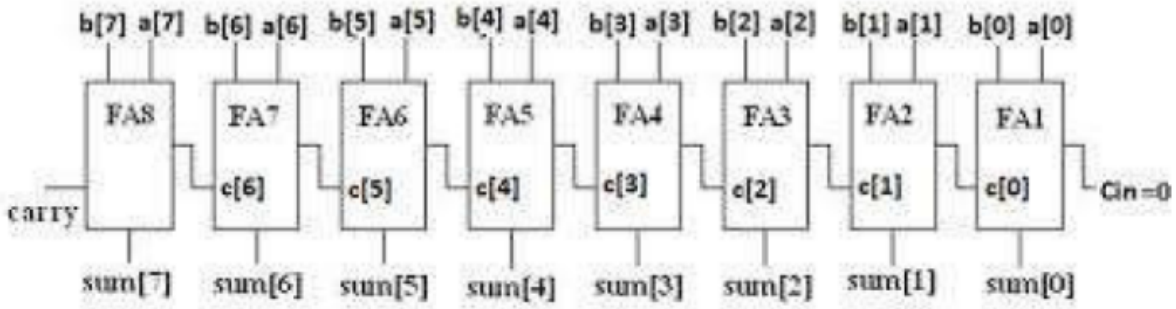


Result:

Thus the Verilog code was scripted, simulated and waveforms were verified using Xilinx Vivado Webpack.

F) 8 BIT ADDER

8 bit Full Adder:



Truth table for 2 bit Full adder:

Truth Table

Input			Output	
A	B	Cin	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Expression

$$Sum = A \oplus B \oplus Cin$$

$$Cout = AB + BCin + CinA$$

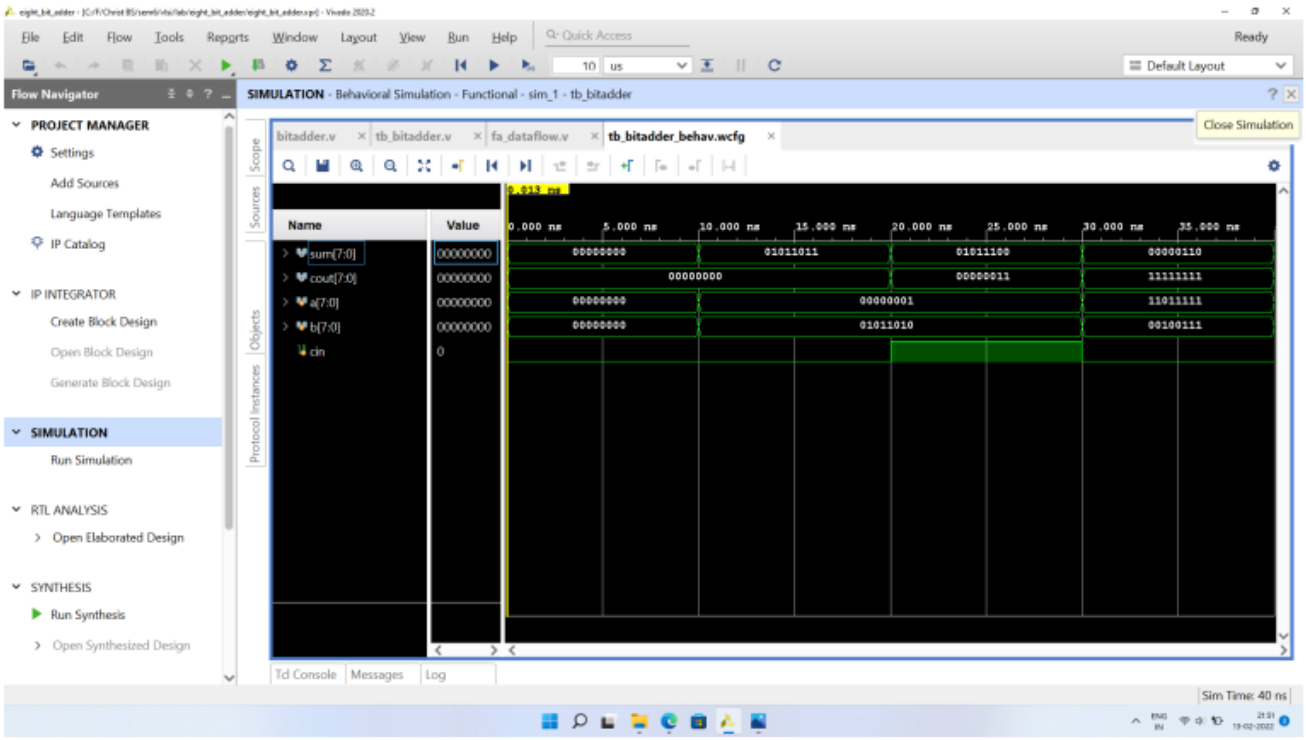
PROGRAM

```
22  //1960628 Prem 8 Bit Adder Dataflow
23  module bitadder(
24      input [7:0]a,
25      input [7:0]b,
26      input cin,
27      output [7:0]sum,
28      output [7:0]cout
29      );
30  fa_dataflow I1(a[0],b[0],cin,sum[0],cout[0]);
31  fa_dataflow I2(a[1],b[1],cout[0],sum[1],cout[1]);
32  fa_dataflow I3(a[2],b[2],cout[1],sum[2],cout[2]);
33  fa_dataflow I4(a[3],b[3],cout[2],sum[3],cout[3]);
34  fa_dataflow I5(a[4],b[4],cout[3],sum[4],cout[4]);
35  fa_dataflow I6(a[5],b[5],cout[4],sum[5],cout[5]);
36  fa_dataflow I7(a[6],b[6],cout[5],sum[6],cout[6]);
37  fa_dataflow I8(a[7],b[7],cout[6],sum[7],cout[7]);
38
39
40
41  endmodule
42
```

TESTBENCH

```
22  //1960628 Prem Testbench 8 Bit Adder
23  module tb_bitadder();
24  wire [7:0]sum;
25  wire [7:0]cout;
26  reg [7:0]a;
27  reg [7:0]b;
28  reg cin;
29  bitadder I1(a,b,cin,sum,cout);
30  initial
31  begin
32  a = 8'b00000000;
33  b = 8'b00000000;
34  cin = 1'b0;
35  #10
36  a = 8'b00000001;
37  b = 8'b01011010;
38  cin = 1'b0;
39  #10
40  a = 8'b00000001;
41  b = 8'b01011010;
42  cin = 1'b1;
43  #10
44  a = 8'b11011111;
45  b = 8'b00100111;
46  cin = 1'b0;
47  #10
48  $finish();
49  end
50  endmodule
```

OUTPUT

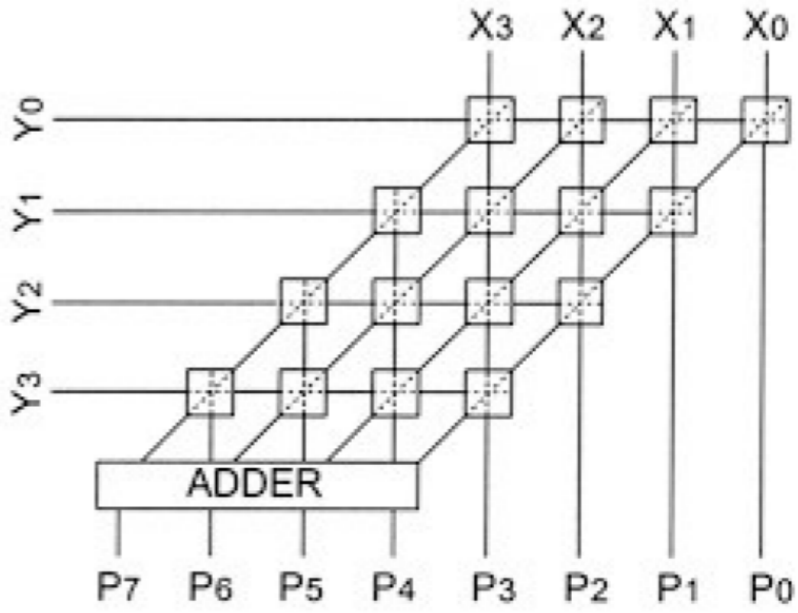


Result:

Thus the Verilog code was scripted, simulated and waveforms were verified using Xilinx Vivado Webpack.

G) 4 BIT MULTIPLIER

4 bit Multiplier:



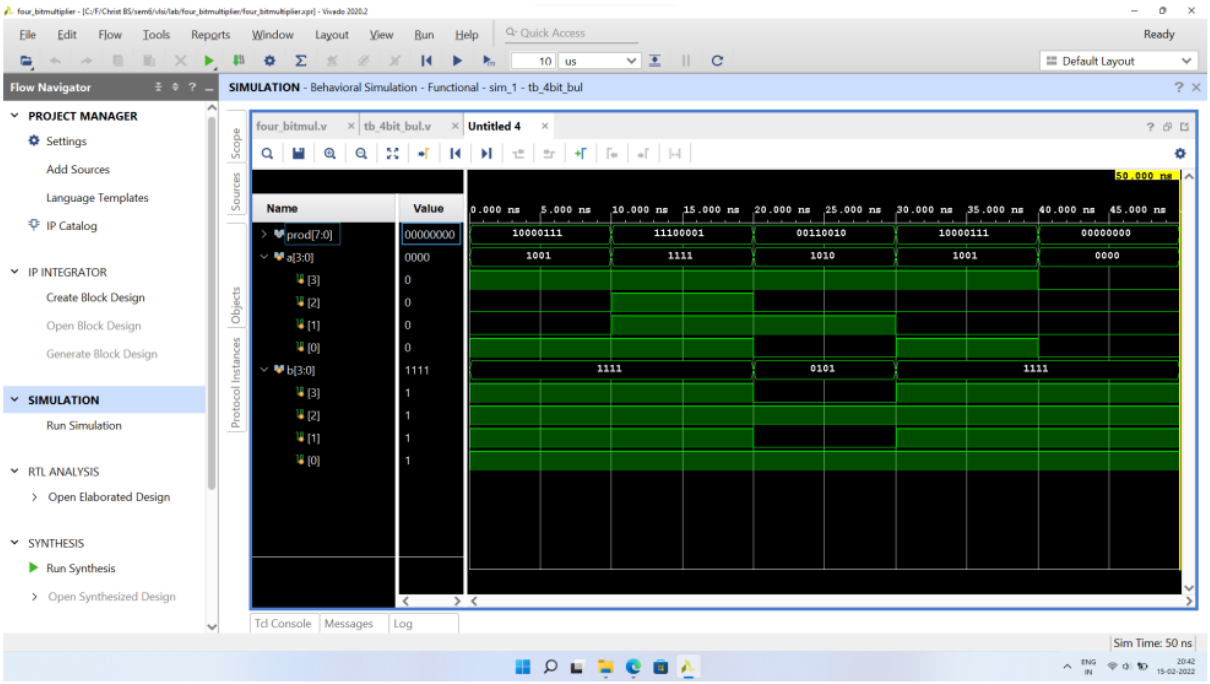
Program:

```
21
22      //1960628 Prem 4 bit multiplier
23      module four_bitmul(
24          output [7:0]prod,
25          input [3:0] a,b
26          );
27      wire [7:0] p0,p1,p2,p3;
28      wire [7:0] sum1,sum2,sum3;
29
30      assign p0 = {4{a[0]}}&b[3:0];
31      assign p1 = {4{a[1]}}&b[3:0];
32      assign p2 = {4{a[2]}}&b[3:0];
33      assign p3 = {4{a[3]}}&b[3:0];
34
35      assign sum1 = p0 + (p1<<1);
36      assign sum2 = sum1 + (p2<<2);
37      assign sum3 = sum2 + (p3<<3);
38      assign prod = sum3;
39      endmodule
40
```

Test Bench:

```
22      //1960628 Prem 4 bit multiplier testbench
23      module tb_4bit_bul();
24      wire [7:0] prod;
25      reg [3:0] a,b;
26
27      four_bitmul I1(prod,a,b);
28      initial
29      begin
30      a = 4'b1001;
31      b = 4'b1111;
32      #10
33      a = 4'b1111;
34      b = 4'b1111;
35      #10
36      a = 4'b1010;
37      b = 4'b0101;
38      #10
39      a = 4'b1001;
40      b = 4'b1111;
41      #10
42      a = 4'b0000;
43      b = 4'b1111;
44      #10
45      $finish();
46      end
47      endmodule
```

OUTPUT



Result:

Thus the Verilog code was scripted, simulated and waveforms were verified using Xilinx Vivado Webpack.

EX.NO: 2 DESIGN ENTRY AND SIMULATION OF SEQUENTIAL LOGIC CIRCUITS

DATE: 15/02/2022

AIM:

To verify the functionality and timing of your design or portion of your design. To interpret Verilog code into circuit functionality and displays logical results of the described HDL to determine correct circuit operation and to create and verify complex functions in a relatively small amount of time.

TOOLS REQUIRED:

1. Xilinx Vivado Design Suite: WebEdition

PROCEDURE:

15. Start by clicking **Vivado Design Suite: WebEdition**

16. Go to **File → new project → Enter project name**, select the top level source as **HDL** & click **next**.

17. Enter **device properties** as

Product Category : General
Purpose Family : Kintex 7
Device :
xc7k70t Package
:fbg484 Speed :
-1
Top Level Source Type : HDL
Synthesis Tool : XST (VHDL/Verilog)
Simulator : ISim (VHDL/Verilog)
Preferred Language : Verilog

& **Click next**.

18. **Right click** the device name (XCS3540) in the source window to create

new source.

19. Select **Verilog module** and enter **file name** in the new source window & click **next**.

20. Write the **Verilog code** in the **Verilog editor window**.

21. Write the **testbench** by create **new source simulation source.**

22.Run Check syntax through **Process window→ synthesize→** double click **Behavioral Check Syntax→** and removes error if present, with proper syntax & coding.

23. Click on the symbol of FPGA device and then right click→ click on **new source**.

24. Select the desired parameters for simulating the design. In this case **combinational circuit** and **simulation time** click **finish**.

25. Assign **all input signal** (**high or low**) using just **click** on this and save file.

26. From the **source process window**. Click **Behavioral simulation** from drop-down menu.

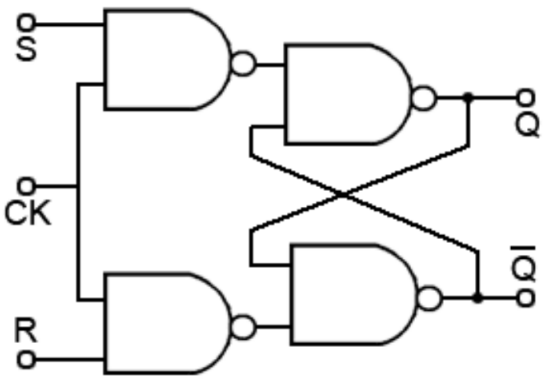
27. Double click the **Simulation Behavioral Model**.

28.Verify your design in **wave window** by seeing behavior of output signal with respect to input signal.

a) FLIP-FLOPS:

(i) SR FLIP-FLOP:

Logic Circuit



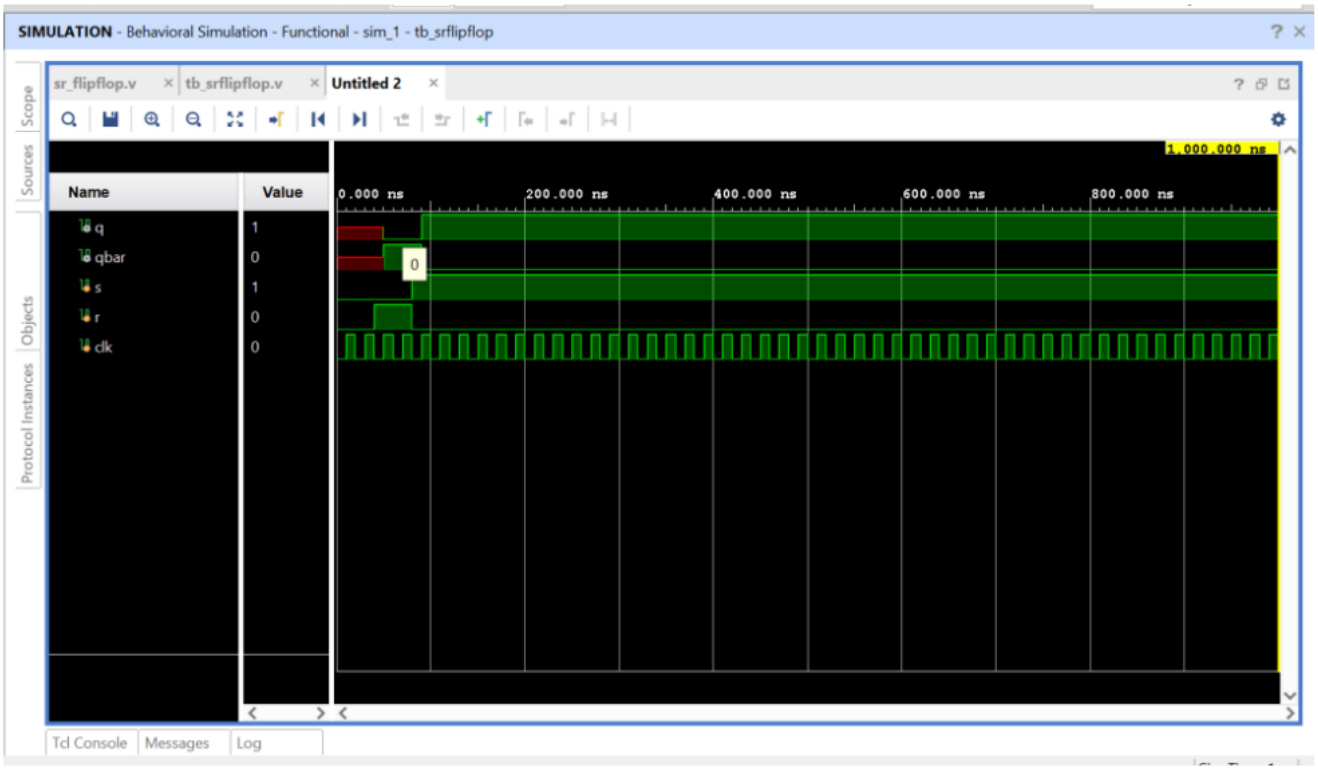
PROGRAM:

a) GATE LEVEL MODELING:

```
22  //1960628 Prem SR Flipflop gate
23  module sr_flipflop(
24      input s,
25      input r,
26      input clk,
27      output q,
28      output qbar
29      );
30  wire w0,w1;
31  nand nand1(w0,s,clk);
32  nand nand2(w1,r,clk);
33  nand nand3(q,qbar,w0);
34  nand nand4(qbar,w1,q);
35  endmodule
36
```

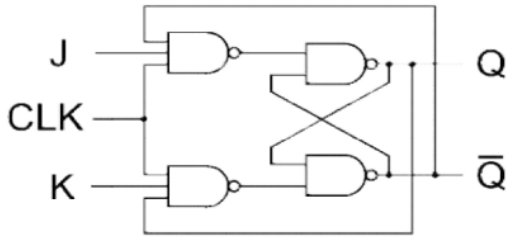
```
22  //1960628 Prem sr flip flop testbench gate
23  module tb_srflipflop();
24  wire q,qbar;
25  reg s,r,clk;
26
27  sr_flipflop I1(s,r,clk,q,qbar);
28  initial
29  begin
30  clk = 0;
31  s = 1'b0;
32  r = 1'b0;
33  #40
34  s = 1'b0;
35  r = 1'b1;
36  #40
37  s = 1'b1;
38  r = 1'b0;
39
40
41  end
42  always #10 clk=~clk;
43
44  endmodule
```

SIMULATION OUTPUT:



(ii) JK FLIP-FLOP:

Logic Circuit



Truth Table

Clk	J	K	Q	Q'	State
1	0	0	Q	Q'	No change in state
1	0	1	0	1	Resets Q to 0
1	1	0	1	0	Sets Q to 1
1	1	1	-	-	Toggles

PROGRAM:

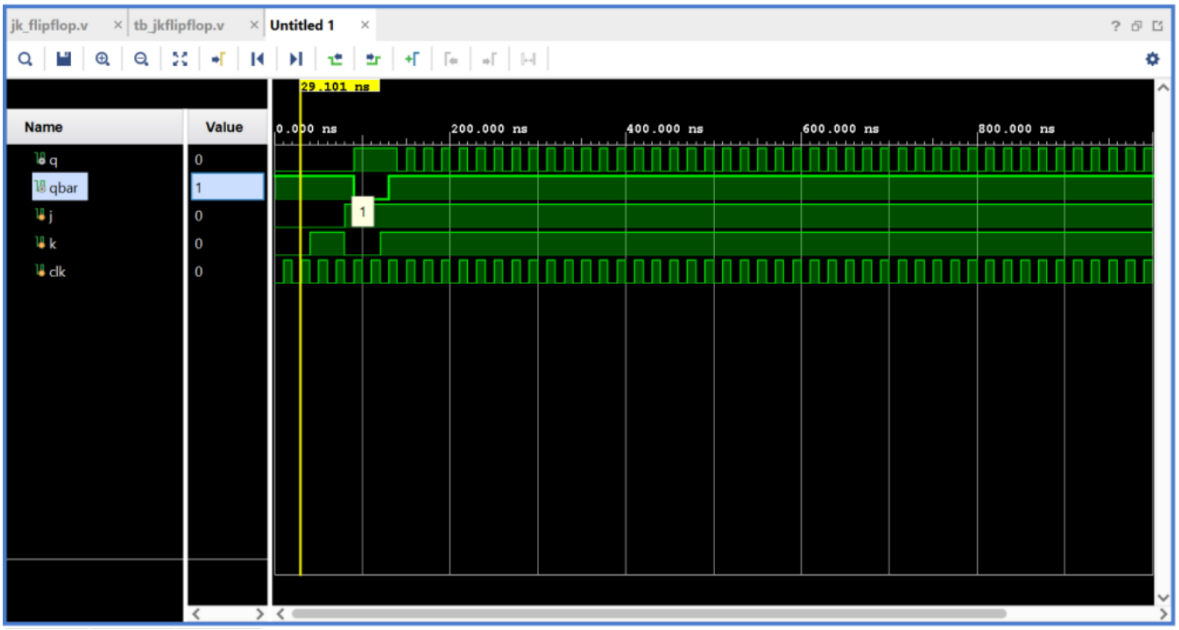
a) GATE LEVEL MODELING:

```
22      //1960628 Prem jk flipflop gate
23      module jk_flipflop(
24          input j,
25          input k,
26          input clk,
27          output q,
28          output qbar
29          );
30      wire w0,w1;
31      nand nand1(w0,qbar,j,clk);
32      nand nand2(w1,q,k,clk);
33      nand nand3(q,w0,qbar);
34      nand nand4(qbar,w1,q);
35      endmodule
```

TEST BENCH:

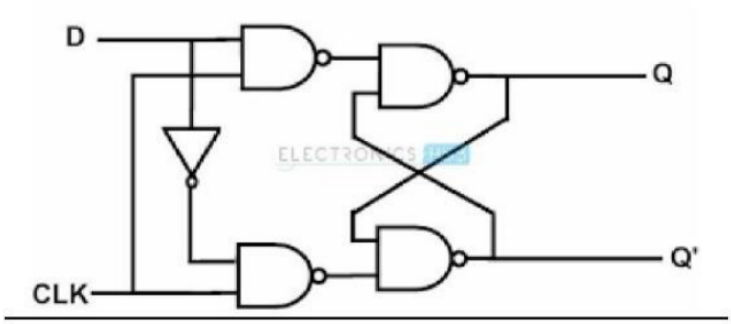
```
21
22  //1960628 Prem jk trstbench
23  module tb_jkflipflop();
24  wire q,qbar;
25  reg j,k,clk;
26  jk_flipflop I1(j,k,clk,q,qbar);
27
28  initial
29  begin
30  force q = 1'b0;
31  #2
32  release q;
33  end
34  initial
35  begin
36  clk = 1'b0;
37  j = 1'b0;
38  k = 1'b0;
39  #40
40  j = 1'b0;
41  k = 1'b1;
42  #40
43  j = 1'b1;
44  k = 1'b0;
45  #40
46  j = 1'b1;
47  k = 1'b1;
48  end
49  always #10 clk=~clk;
50
51  endmodule
```

SIMULATION OUTPUT:



(iii) D FLIP-FLOP:

Logic Circuit



Truth Table

D Flip-flop

Symbol

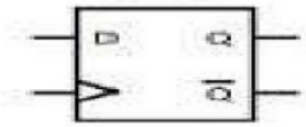


Table of truth:

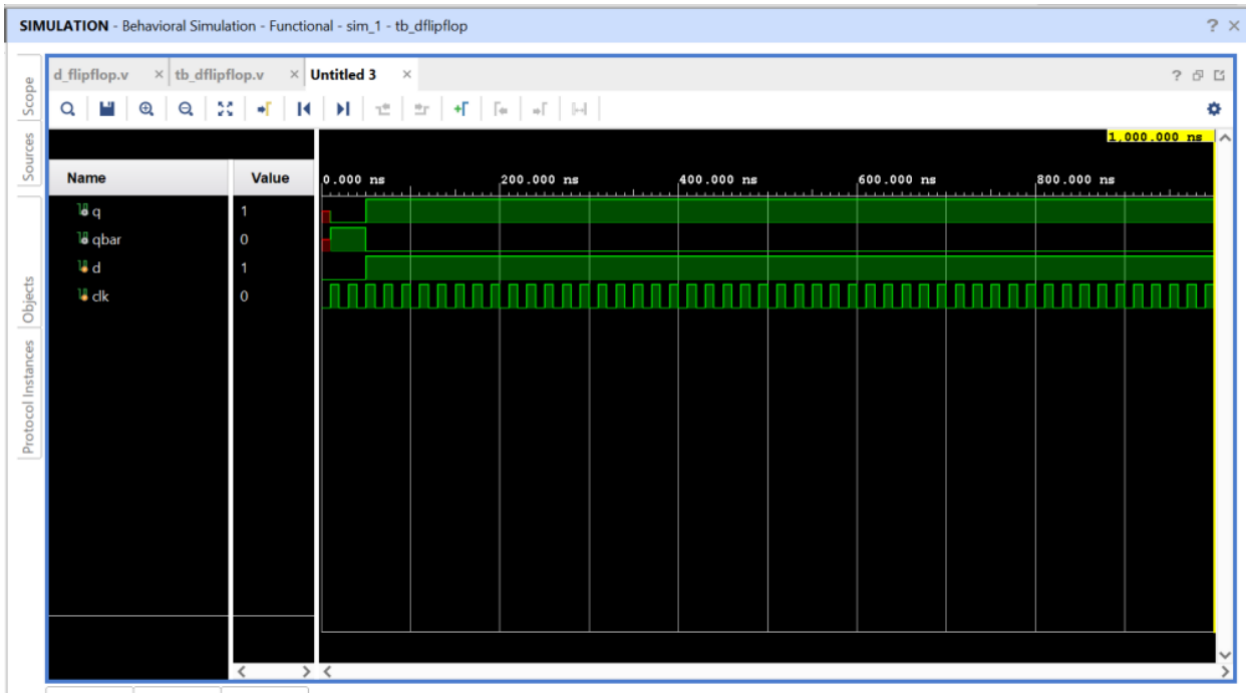
clk	D	Q	$\overline{Q}$
0	0	Q	$\overline{Q}$
0	1	Q	$\overline{Q}$
1	0	0	1
1	1	1	0

PROGRAM :
GATE LEVEL MODELING

```
// 1960628 Prem D flipflop GATE
module d_flipflop(
    input d,
    input clk,
    output q,
    output qbar
    );
wire w0,w1;
nand nand1(w0,d,clk);
nand nand2(w1,~d,clk);
nand nand3(q,w0,qbar);
nand nand4(qbar,w1,q);
endmodule
```

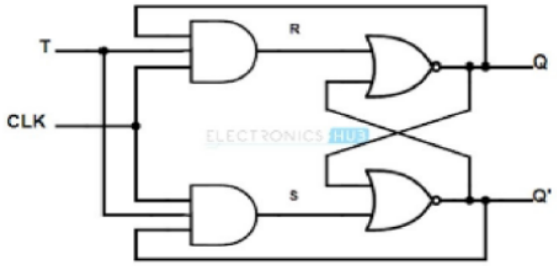
```
//1960628 Prem Testbench dflipflop
module tb_dflipflop();
wire q,qbar;
reg d,clk;
d_flipflop i1(d,clk,q,qbar);

initial
begin
clk = 0;
d = 0;
#50
d = 1;
end
always #10 clk = ~clk;
endmodule
```

SIMULATION



(iv) T FLIP-FLOP:

Logic Circuit



Truth Table

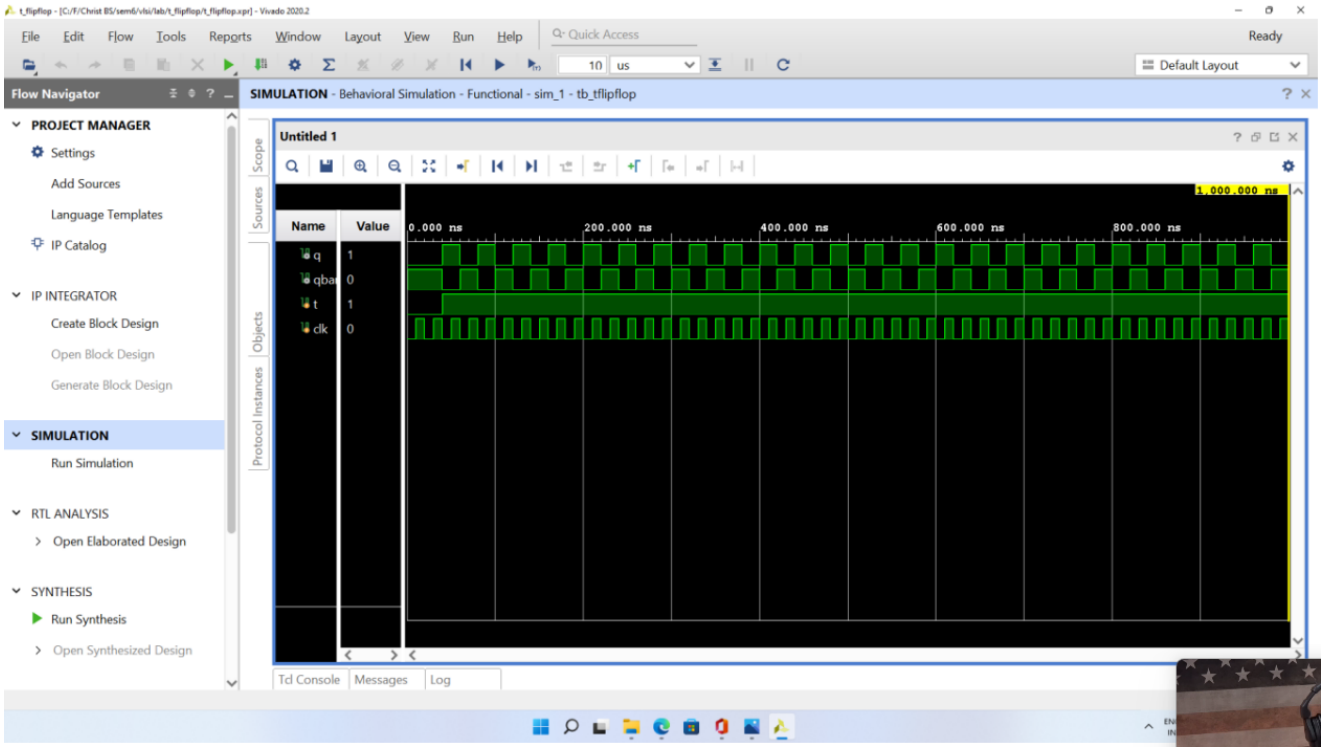
T	Q_n	Q_{n+1}
0	0	0
0	1	1
1	0	1
1	1	0

GATE LEVEL MODELING

```
//1960628 Prem  t flip fop
module t_flipflop(
    input t,
    input clk,
    output q,
    output qbar
    );
wire s,r;
and and1(r,q,t,clk);
and and2(s,qbar,t,clk);
nor nor1(q,r,qbar);
nor nor2(qbar,q,s);

endmodule
```

```
module tb_tflipflop();
wire  q,qbar;
reg t,clk;

t_flipflop I1(t,clk,q,qbar);
initial
begin
force q=1'b0;
#2
release q;
end

initial begin
clk = 0;
t = 0;
#50
t = 0;
#50
t = 1;
#50
t = 1;
end
always #10 clk = ~clk;
endmodule
```

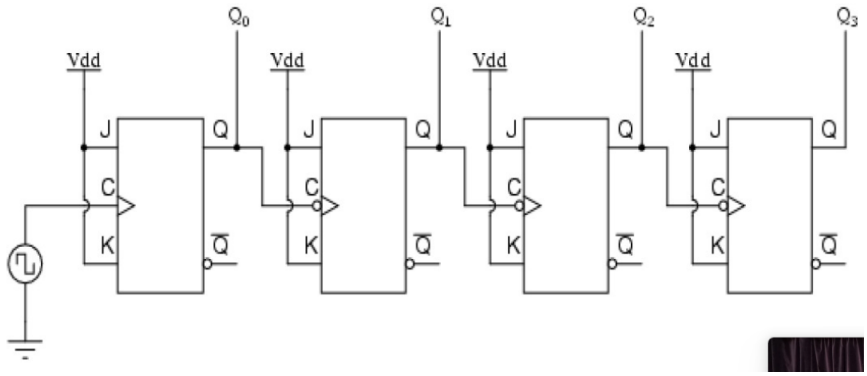
SIMULATION OUTPUT:



2 B - COUNTERS

(i) ASYNCHRONOUS COUNTERS:

A four-bit "up" counter



1) Four bit Asynchronous Counter

TOP LEVEL

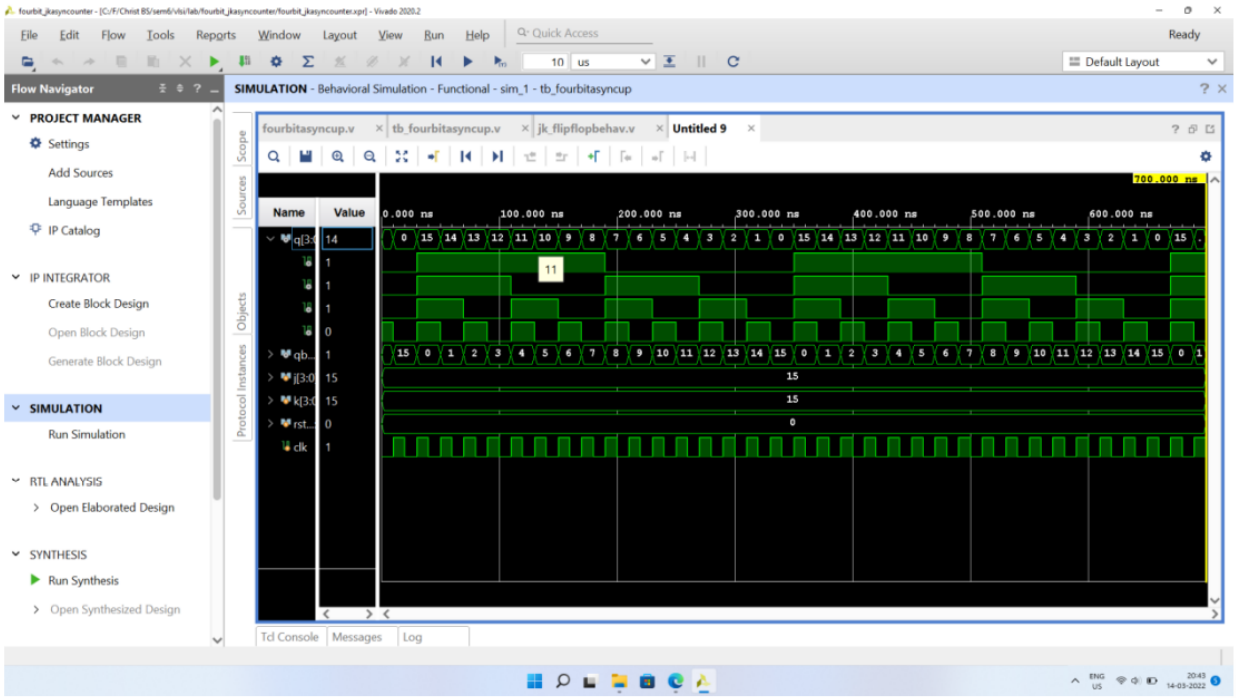
```
22  //1960628 PREM four bit async counter
23  module fourbitasyncup(
24      input [3:0]j,
25      input [3:0]k,
26      input clk,
27      input [3:0]rst,
28      output [3:0]q,
29      output [3:0]qbar
30      );
31
32  jk_flipflopbehav i1(j[0],k[0],clk,rst[0],q[0],qbar[0]);
33  jk_flipflopbehav i2(j[1],k[1],q[0],rst[1],q[1],qbar[1]);
34  jk_flipflopbehav i3(j[2],k[2],q[1],rst[2],q[2],qbar[2]);
35  jk_flipflopbehav i4(j[3],k[3],q[2],rst[3],q[3],qbar[3]);
36
37  endmodule
```

```
22  //1960628 Prem Kumar R JK Flip Flop Behav
23  module jk_flipflopbehav(
24      input j,
25      input k,
26      input clk,
27      input rst,
28      output reg  q,
29      output reg qbar
30      );
31  initial begin
32  q = 0;
33  qbar = 1;
34  end
35  always@ (posedge clk or negedge rst)
36      if (rst == 1)
37          begin
38              q <= 0;
39              qbar <= 1;
40          end
41      else
42          begin
43          case ({j,k})
44          2'b00:begin q<= q; qbar <= ~q; end
45          2'b01:begin q<= 1'b0; qbar <= 1; end
46          2'b10:begin q<= 1'b1; qbar <= 0; end
47          2'b11:begin q<= ~q; qbar <= q; end
48          endcase
49          end
50  endmodule
```

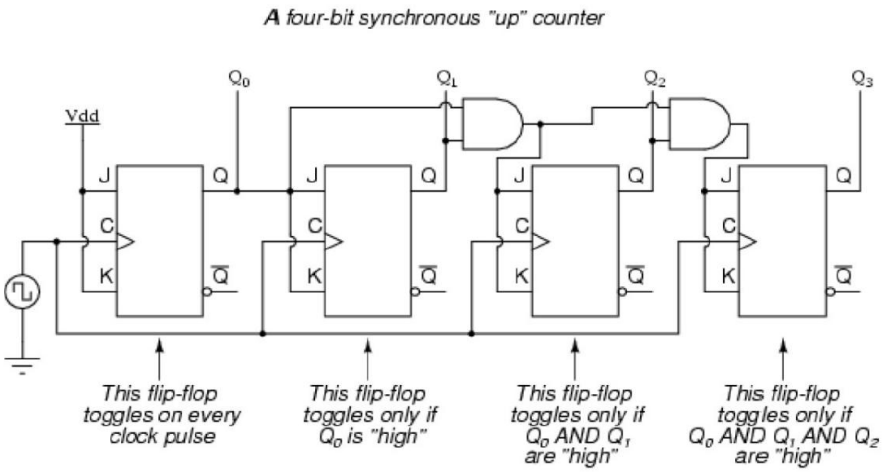
TESTBENCH

```
21
22  //1960628 PREM 4 BIT ASYNC COUNT TEMSTBENCH
23  module tb_fourbitasyncup();
24  wire [3:0]q,qbar;
25  reg [3:0]j,k,rst;
26  reg clk;
27
28  fourbitasyncup i1(j,k,clk,rst,q,qbar);
29
30  initial begin clk = 0; forever #10 clk=~clk; end
31
32  initial begin
33  j = 4'b1111;
34  k = 4'b1111;
35  rst = 4'b0000;
36  #700
37  $finish();
38  end
39
40  endmodule
41
```

OUTPUT



Logic Circuit



2) FOUR BIT SYNCHRONOUS COUNTER

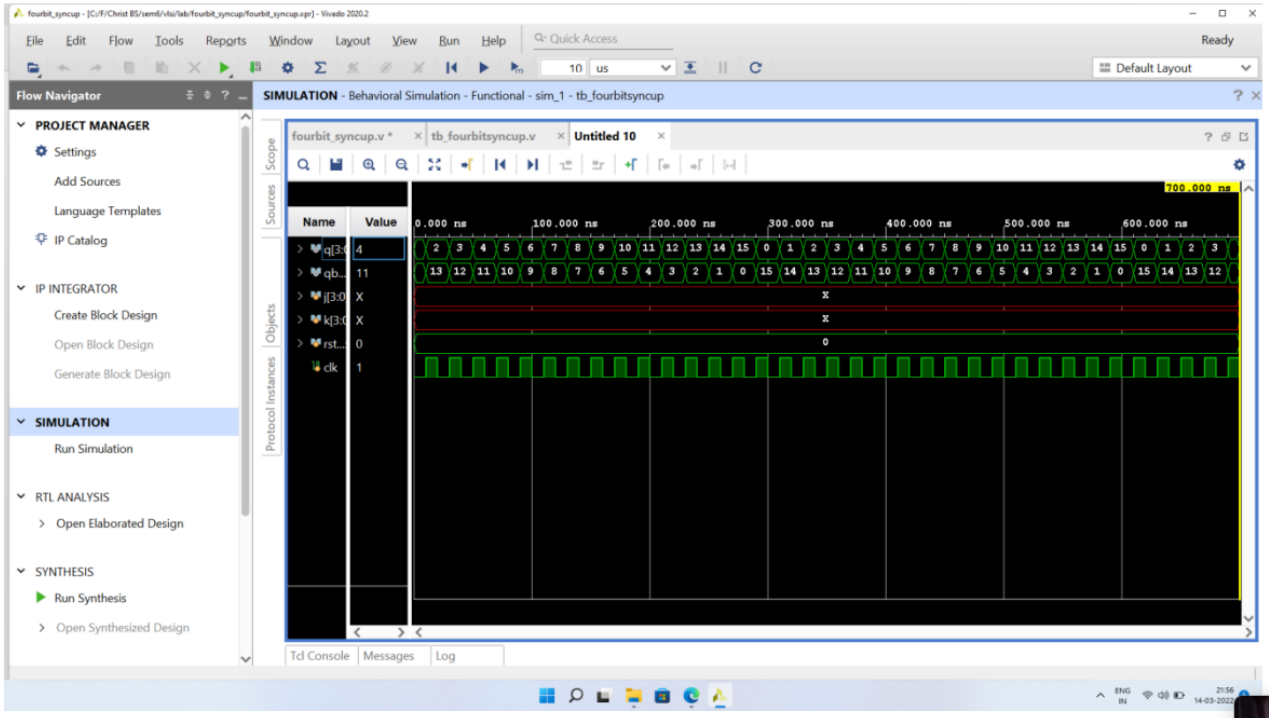
BEHAVIOURAL MODELLING

```
22  //1960628 PREM FOURBIT SYNC
23  module fourbit_syncup(
24      input [3:0]j,
25      input [3:0]k,
26      input clk,
27      input [3:0]rst,
28      output [3:0]q,
29      output [3:0]qbar
30      );
31  wire w0,w1;
32
33  and and1(w0,q[0],q[1]);
34  and and2(w1,w0,q[2]);
35  jk_flipflopbehav i1(j[0],k[0],clk,rst[0],q[0],qbar[0]);
36  jk_flipflopbehav i2(q[0],q[0],clk,rst[1],q[1],qbar[1]);
37  jk_flipflopbehav i3(w0,w0,clk,rst[2],q[2],qbar[2]);
38  jk_flipflopbehav i4(w1,w1,clk,rst[3],q[3],qbar[3]);
39  endmodule
40
```

```
22  //1960628 Prem Kumar R JK Flip Flop Behav
23  module jk_flipflopbehav(
24      input j,
25      input k,
26      input clk,
27      input rst,
28      output reg  q,
29      output reg qbar
30      );
31  initial begin
32  q = 0;
33  qbar = 1;
34  end
35  always@ (posedge clk or negedge rst)
36      if (rst == 1)
37          begin
38              q <= 0;
39              qbar <= 1;
40          end
41      else
42          begin
43          case ({j,k})
44          2'b00:begin q<= q; qbar <= ~q; end
45          2'b01:begin q<= 1'b0; qbar <= 1; end
46          2'b10:begin q<= 1'b1; qbar <= 0; end
47          2'b11:begin q<= ~q; qbar <= q; end
48          endcase
49          end
50  endmodule
```

TEST BENCH

```
22  //1960628 PREM KUMAR R TESTBENCH SYNC UP
23  module tb_fourbitsyncup();
24  wire [3:0]q,qbar;
25  reg [3:0]j,k,rst;
26  reg clk;
27
28  fourbit_syncup i1(j,k,clk,rst,q,qbar);
29
30  initial begin
31  j = 4'b0000;
32  k = 4'b0000;
33  end
34
35  initial begin clk = 0; forever #10 clk = ~clk; end
36
37  initial begin
38  rst = 4'b0000;
39  j[0] = 1;
40  k[0] = 1;
41  #700
42  $finish();
43  end
44
45  endmodule
```

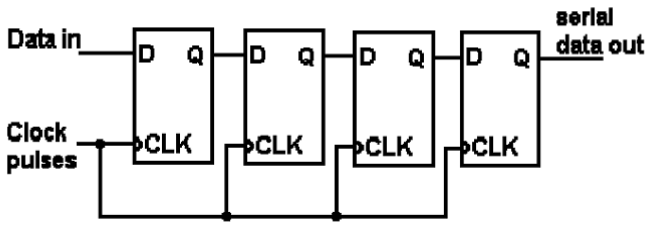


RESULT : A FOUR BIT SYNCHRONOUS AND ASYNCHRONOUS COUNTER WERE MADE USING JK FLIP FLOP LEAF CELSS IN BEHAVIOURAL MODELLING AND OUTPUTS VERIFIED

2C) REGISTERS

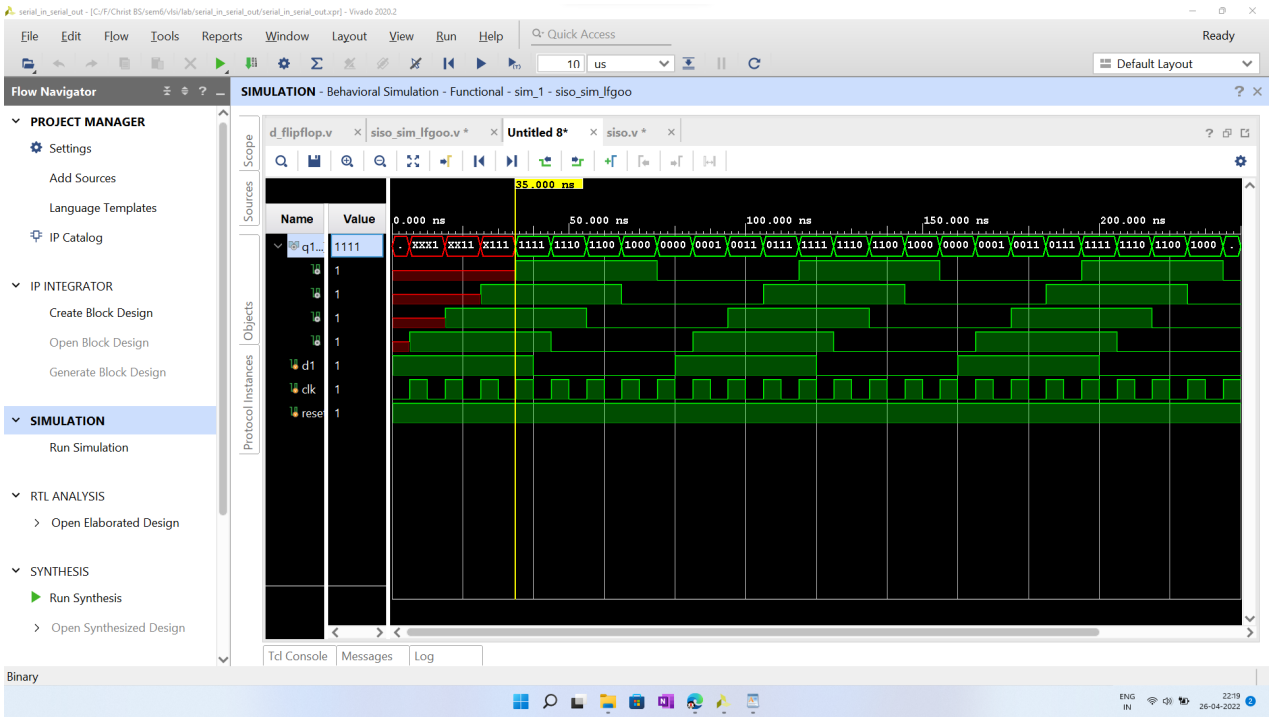
(i) Serial In Serial Out (SISO):

Logic Circuit




```
//1960628 Prem Kuma SISO
module siso(
    input d1,
    input clk,
    input reset,
    output [3:0]q1
    );
    dff i1(d1,clk,reset,q1[0]);
    dff i2(q1[0],clk,reset,q1[1]);
    dff i3(q1[1],clk,reset,q1[2]);
    dff i4(q1[2],clk,reset,q1[3]);
endmodule

module dff(
    input d,
    input clk,
    input rst,
    output reg q
    );

always @(posedge clk, negedge rst)
begin
if(!rst)
q <= 0;
else q<= d;
end
endmodule
```

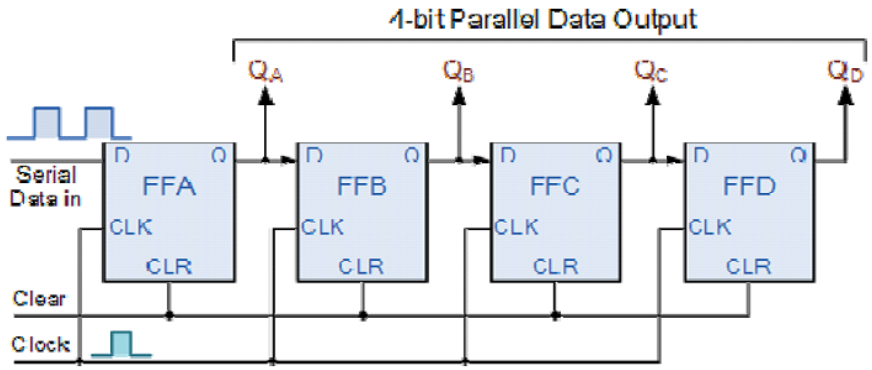
```
// TESTBENCH SISO 1960628 PREM
module siso_sim_lfgoo();
wire [3:0]q1;
reg d1;
reg clk;
reg reset;
siso i1(d1,clk,reset,q1);
initial begin clk = 0; forever #5 clk = ~clk; end
initial begin
reset = 1;
d1 = 1;
#40
d1 = 0;
#40
d1 = 1;
#40
d1 = 0;
#40
d1 = 1;
#40
d1 = 0;
#40
$finish();
end
endmodule
```



RESULT: A 4 bit Serial in Serial Out register has been verified using 4 D flipflops.

(ii) Serial In Parallel Out (SIPO):

Logic Circuit



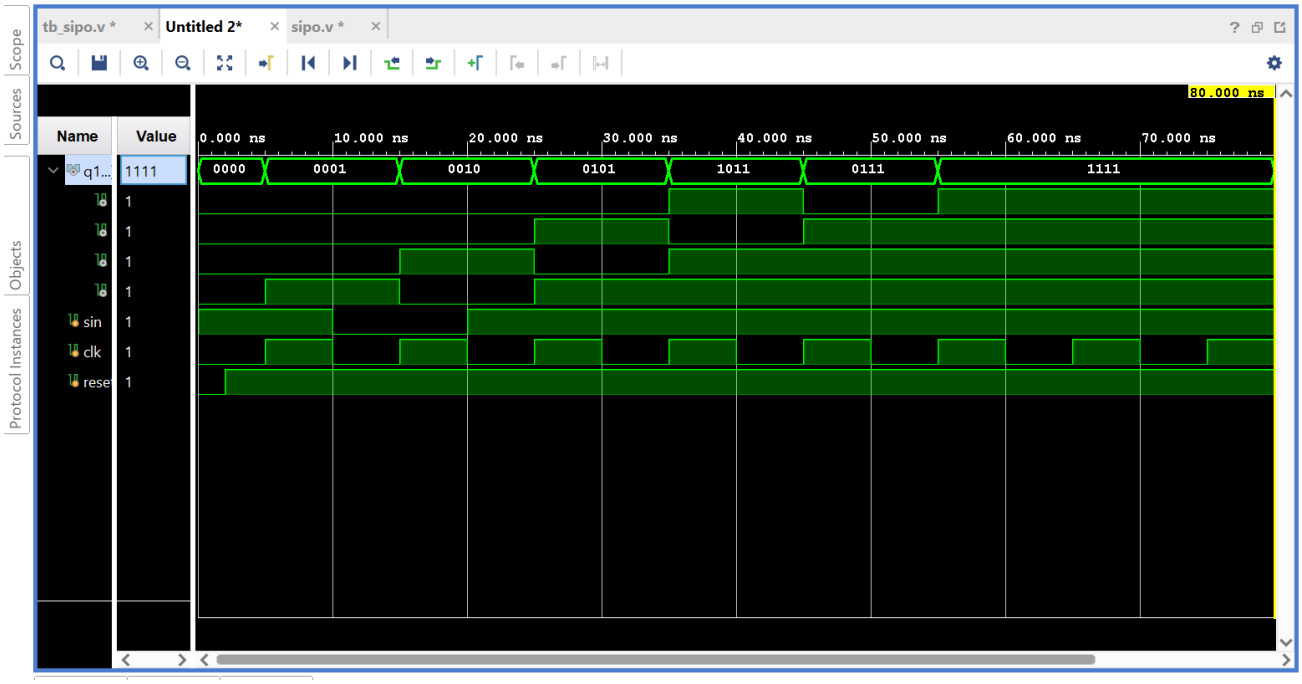
SOURCE:

```
22      //1960628 Prem Kumar R SIPO
23      module sipo(
24          input sin,
25          input clk,
26          input reset,
27          output [3:0]q1
28          );
29      dff i1(sin,clk,reset,q1[0]);
30      dff i2(q1[0],clk,reset,q1[1]);
31      dff i3(q1[1],clk,reset,q1[2]);
32      dff i4(q1[2],clk,reset,q1[3]);
33      endmodule
34
35
36      module dff(
37          input d,
38          input clk,
39          input rst,
40          output reg q
41          );
42
43      always @(posedge clk, negedge rst)
44      begin
45      if(!rst)
46      q <= 0;
47      else q<= d;
48      end
49      endmodule
```

TESTBENCH

```
22      //1960628 PREM KUMAR R TESTBENCH SIPO
23      module tb_sipo();
24      wire [3:0]q1;
25      reg sin,clk,reset;
26
27      sipo i1(sin,clk,reset,q1);
28      initial begin reset = 0; #2 reset = 1; end
29      initial begin clk = 0; forever #5 clk = ~clk; end
30      initial begin
31      sin = 1;
32      #10
33      sin = 0;
34      #10;
35      sin = 1;
36      #10
37      sin = 1;
38      #10
39      sin = 1;
40      #40
41      $finish();
42      end
43      endmodule
```

OUTPUT

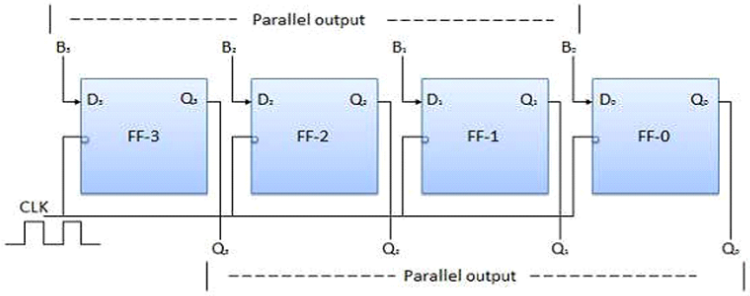


RESULT: 4 bit Serial input parallel output shift register simulated verified using D flipflops

PARALLEL INPUT PARALLEL OUTPUT SHIFT REGISTER

(iii) Parallel In Parallel Out (PIPO):

Logic Circuit



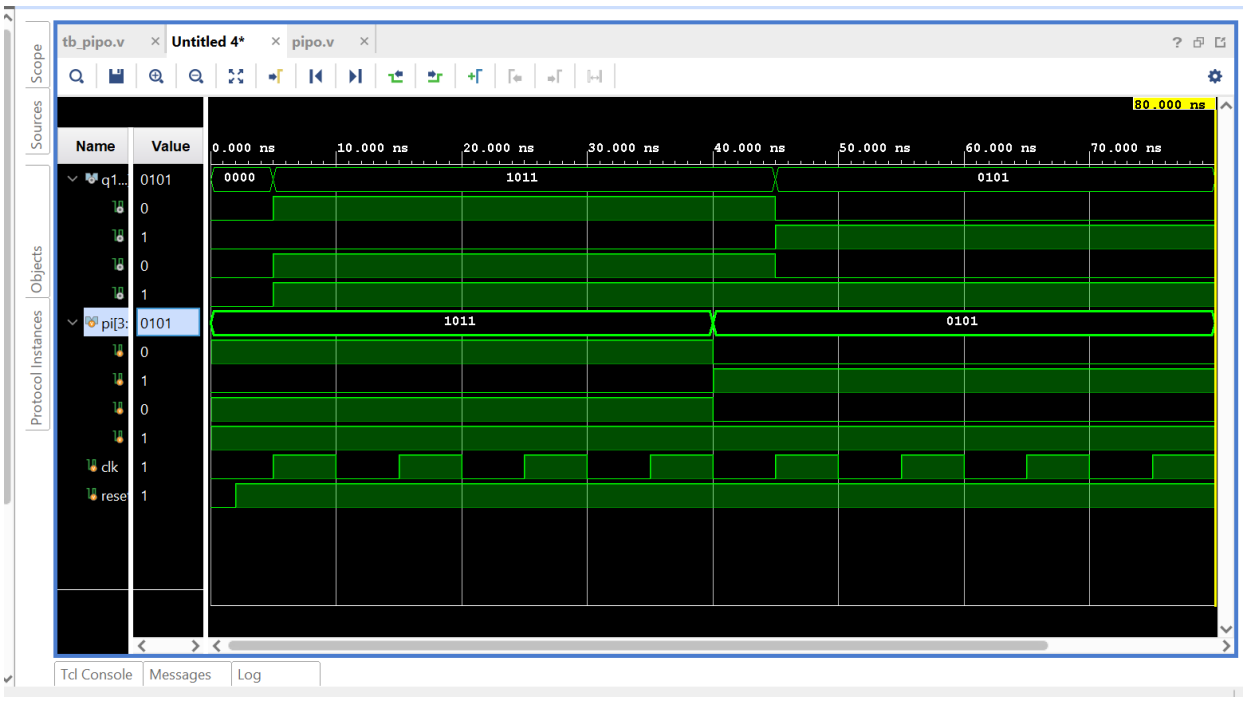
SOURCE:

```
23      //1960628 Prem Kumar R PIPO
24      module pipo(
25          input [3:0]pi,
26          input clk,
27          input reset,
28          output [3:0]q1
29          );
30      dff i1(pi[0],clk,reset,q1[0]);
31      dff i2(pi[1],clk,reset,q1[1]);
32      dff i3(pi[2],clk,reset,q1[2]);
33      dff i4(pi[3],clk,reset,q1[3]);
34      endmodule
35
36
37      module dff(
38          input d,
39          input clk,
40          input rst,
41          output reg q
42          );
43
44      always @(posedge clk, negedge rst)
45      begin
46      if(!rst)
47      q <= 0;
48      else q<= d;
49      end
50      endmodule
```

TESTBENCH:

```
23      //1960628 PREM KUMAR R TESTBENCH PIPO
24      module tb_pipo();
25      wire [3:0]q1;
26      reg [3:0]pi;
27      reg clk,reset;
28
29      pipo i1(pi,clk,reset,q1);
30      initial begin reset = 0; #2 reset = 1; end
31      initial begin clk = 0; forever #5 clk = ~clk; end
32      initial begin
33      pi[3:0] = 4'b1011;
34      #40
35      pi[3:0] = 4'b0101;
36      #40
37      $finish();
38      end
39      endmodule
```

OUTPUT:



RESULT: PARALLEL IN PARALLEL OUT REGISTER IS SIMULATED USING VERILOG THROUGH 4 FLIP FLOPS.

EX.NO: 3 DESIGN ENTRY AND SIMULATION OF COMBINATIONAL LOGIC CIRCUITS

DATE: 27/04/2022

AIM:

To verify the functionality and timing of your design or portion of your design. To interpret Verilog code into circuit functionality and displays logical results of the described HDL to determine correct circuit operation and to create and verify complex functions in a relatively small amount of time.

TOOLS REQUIRED:

1. Xilinx Vivado Design Suite: WebEdition

PROCEDURE:

29. Start by clicking **Vivado Design Suite: WebEdition**

30. Go to **File → new project → Enter project name**, select the top level source as **HDL** & click **next**.

31. Enter **device properties** as

Product Category : General
Purpose Family : Kintex 7
Device :
xc7k70t Package
:fbg484 Speed :
-1
Top Level Source Type : HDL
Synthesis Tool : XST (VHDL/Verilog)
Simulator : ISim (VHDL/Verilog)
Preferred Language : Verilog

& **Click next**.

32. **Right click** the device name (XCS3540) in the source window to create

new source.

33. Select **Verilog module** and enter **file name** in the new source window & click **next**.

34. Write the **Verilog code** in the **Verilog editor window**.

35. Write the **testbench** by create **new source simulation source.**

36.Run Check syntax through **Process window→ synthesize→** double click **Behavioral Check Syntax→** and removes error if present, with proper syntax & coding.

37. Click on the symbol of FPGA device and then right click→ click on **new source**.

38. Select the desired parameters for simulating the design. In this case **combinational circuit** and **simulation time** click **finish**.

39. Assign **all input signal** (**high or low**) using just **click** on this and save file.

40. From the **source process window**. Click **Behavioral simulation** from drop-down menu.

41. Double click the **Simulation Behavioral Model**.

42.Verify your design in **wave window** by seeing behavior of output signal with respect to input signal.

SOURCE:

```
22      //1960628 PREM KUMAR R
23      module traffic_control(n_lights,s_lights,e_lights,w_lights,clk,rst_a);
24
25        output reg [2:0] n_lights,s_lights,e_lights,w_lights;
26        input      clk;
27        input      rst_a;
28
29        reg [2:0] state;
30
31        parameter [2:0] north=3'b000;
32        parameter [2:0] north_y=3'b001;
33        parameter [2:0] south=3'b010;
34        parameter [2:0] south_y=3'b011;
35        parameter [2:0] east=3'b100;
36        parameter [2:0] east_y=3'b101;
37        parameter [2:0] west=3'b110;
38        parameter [2:0] west_y=3'b111;
39
40        reg [2:0] count;
41
42
43        always @(posedge clk, posedge rst_a)
44          begin
45            if (rst_a)
46                begin
47                    state=north;
48                    count =3'b000;
49                end
50            else
51                begin
52                    case (state)
53                    north :
54                        begin
55                            if (count==3'b111)
56                                begin
57                                count=3'b000;
58                                state=north_y;
59                                end
60                            else
61                                begin
62                                count=count+3'b001;
63                                state=north;
64                                end
65                        end
66
67                     north_y :
68                        begin
69                            if (count==3'b011)
70                                begin
71                                count=3'b000;
72                                state=south;
73                                end
74                            else
75                                begin
76                                count=count+3'b001;
77                                state=north_y;
78                                end
79                        end
80
81                    south :
82                        begin
83                            if (count==3'b111)
84                                begin
85                                count=3'b0;
86                                state=south_y;
87                                end
```

```
                    else
                        begin
                        count=count+3'b001;
                        state=south;
                    end
                end

            south_y :
                begin
                    if (count==3'b011)
                        begin
                        count=3'b0;
                        state=east;
                        end
                    else
                        begin
                        count=count+3'b001;
                        state=south_y;
                        end
                end

            east :
                begin
                    if (count==3'b111)
                        begin
                        count=3'b0;
                        state=east_y;
                        end
                    else
                        begin
                        count=count+3'b001;
                        state=east;
                        end
                end

              east_y :
                  begin
                      if (count==3'b011)
                          begin
                          count=3'b0;
                          state=west;
                          end
                      else
                          begin
                          count=count+3'b001;
                          state=east_y;
                          end
                  end

              west :
                  begin
                      if (count==3'b111)
                          begin
                          state=west_y;
                          count=3'b0;
                          end
                      else
                          begin
                          count=count+3'b001;
                          state=west;
                          end
                  end

              west_y :
                  begin
                      if (count==3'b011)
```

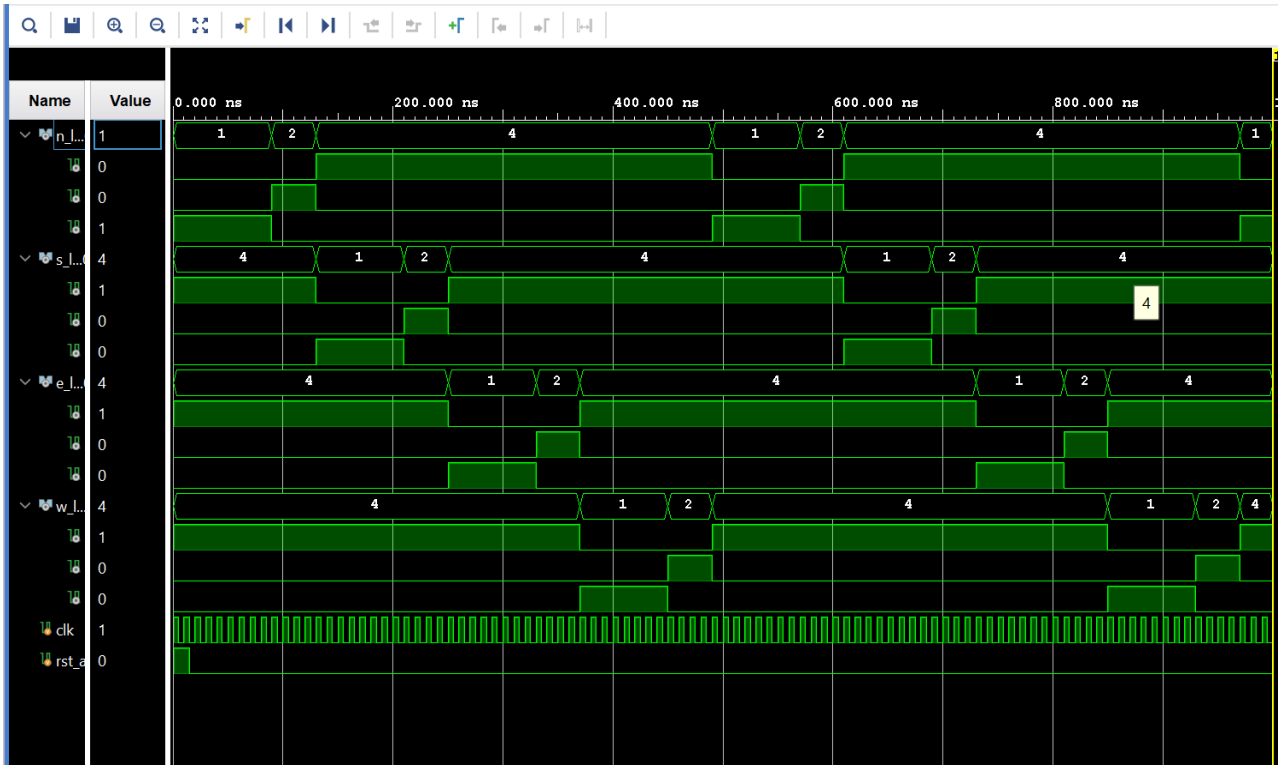
```
                        begin
                        state=north;
                        count=3'b0;
                        end
                    else
                        begin
                        count=count+3'b001;
                        state=west_y;
                        end
                    end
            endcase // case (state)
        end // always @ (state)
    end


always @(state)
    begin
        case (state)
            north :
                begin
                    n_lights = 3'b001;
                    s_lights = 3'b100;
                    e_lights = 3'b100;
                    w_lights = 3'b100;
                end // case: north

            north_y :
                begin
                    n_lights = 3'b010;
                    s_lights = 3'b100;
                    e_lights = 3'b100;
                    w_lights = 3'b100;
                end // case: north_y
            south :
                begin
                    n_lights = 3'b100;
                    s_lights = 3'b001;
                    e_lights = 3'b100;
                    w_lights = 3'b100;
                end // case: south

            south_y :
                begin
                    n_lights = 3'b100;
                    s_lights = 3'b010;
                    e_lights = 3'b100;
                    w_lights = 3'b100;
                end // case: south_y

            west :
                begin
                    n_lights = 3'b100;
                    s_lights = 3'b100;
                    e_lights = 3'b100;
                    w_lights = 3'b001;
                end // case: west

            west_y :
                begin
                    n_lights = 3'b100;
                    s_lights = 3'b100;
                    e_lights = 3'b100;
                    w_lights = 3'b010;
                end // case: west_y

            east :
                begin
```

```
                    n_lights = 3'b100;
                    s_lights = 3'b100;
                    e_lights = 3'b001;
                    w_lights = 3'b100;
                end // case: east

            east_y :
                begin
                    n_lights = 3'b100;
                    s_lights = 3'b100;
                    e_lights = 3'b010;
                    w_lights = 3'b100;
                end // case: east_y
            endcase // case (state)
        end // always @ (state)
    endmodule
```

TESTBENCH

```
//1960628 PREM KUMAR R
module traffic_control_tb;

wire [2:0] n_lights,s_lights,e_lights,w_lights;
reg clk,rst_a;

traffic_control DUT (n_lights,s_lights,e_lights,w_lights,clk,rst_a);

initial
 begin
  clk=1'b1;
  forever #5 clk=~clk;
 end

initial
 begin
  rst_a=1'b1;
  #15;
  rst_a=1'b0;
  #1000;
  $stop;
 end
endmodule
```

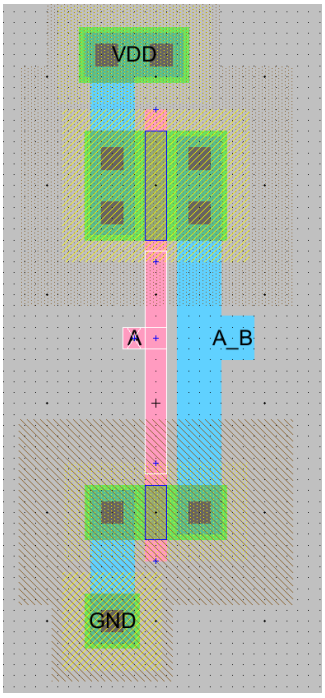
OUTPUT


Name
Value
0.000 ns
200.000 ns
400.000 ns
600.000 ns
800.000 ns
clk
rst_a

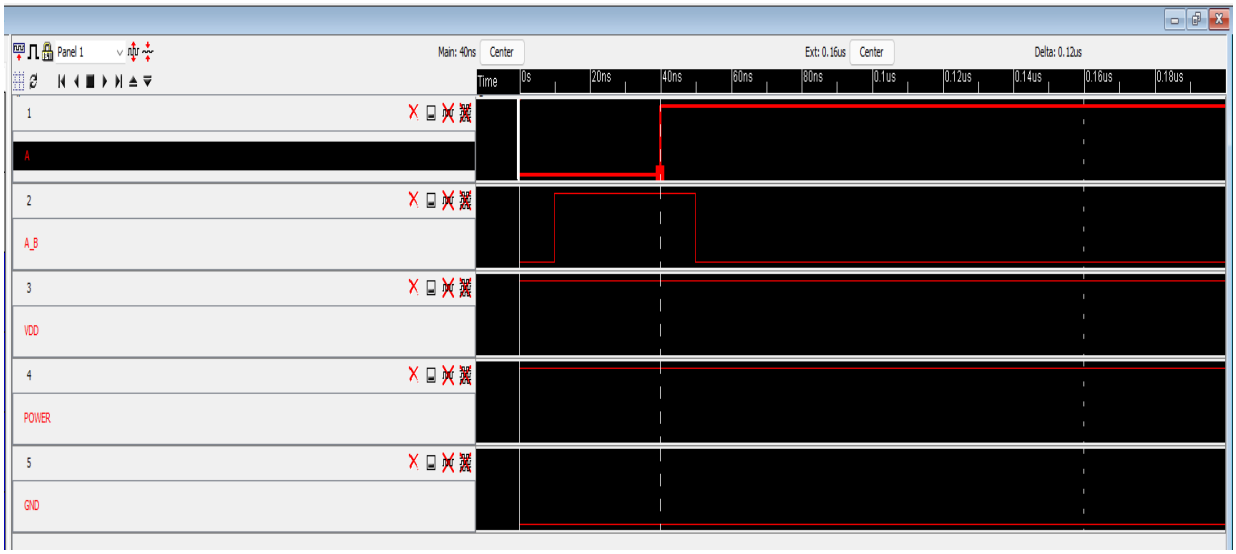
EX.NO:4 SCHEMATIC AND LAYOUT OF A SIMPLE CMOS INVERTER

AIM: To design the layout of a CMOS Inverter, NAND and NOR gate using Electric software.

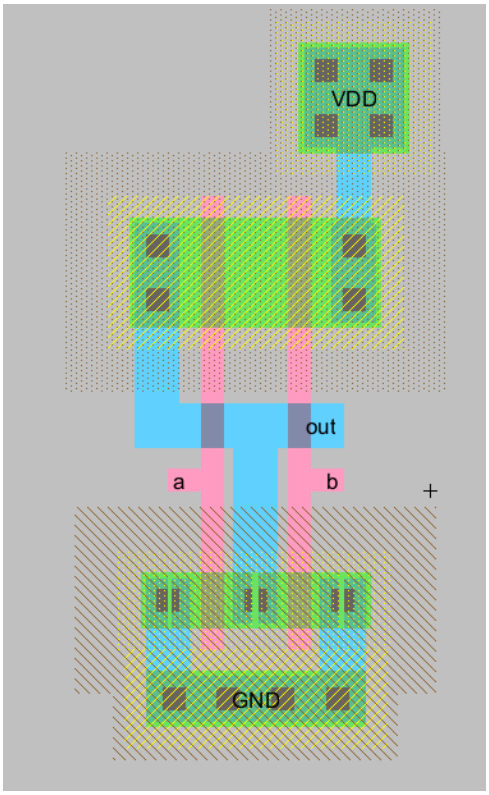
CMOS Inverter Schematic:



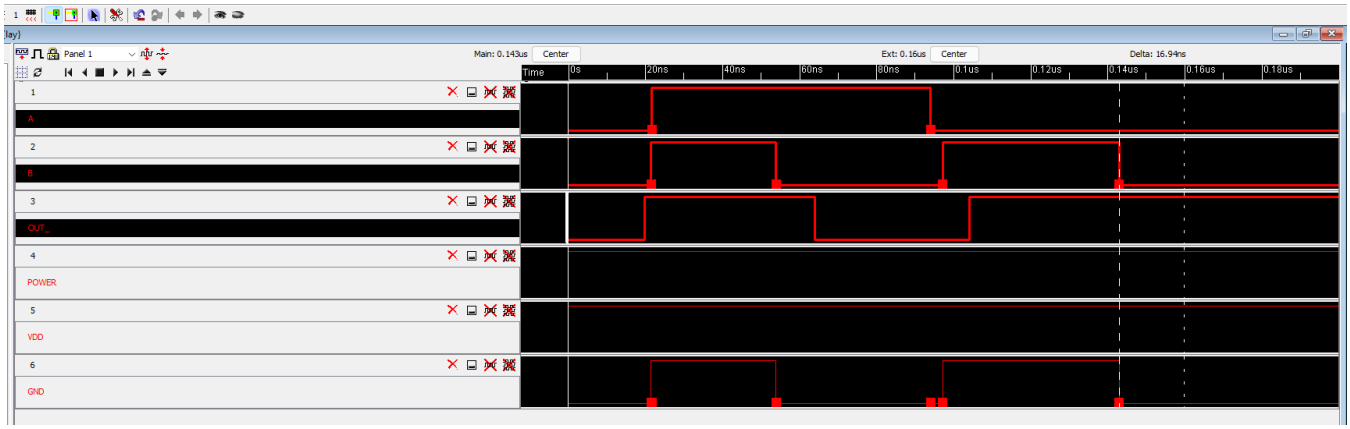
CMOS Inverter Simulated Output



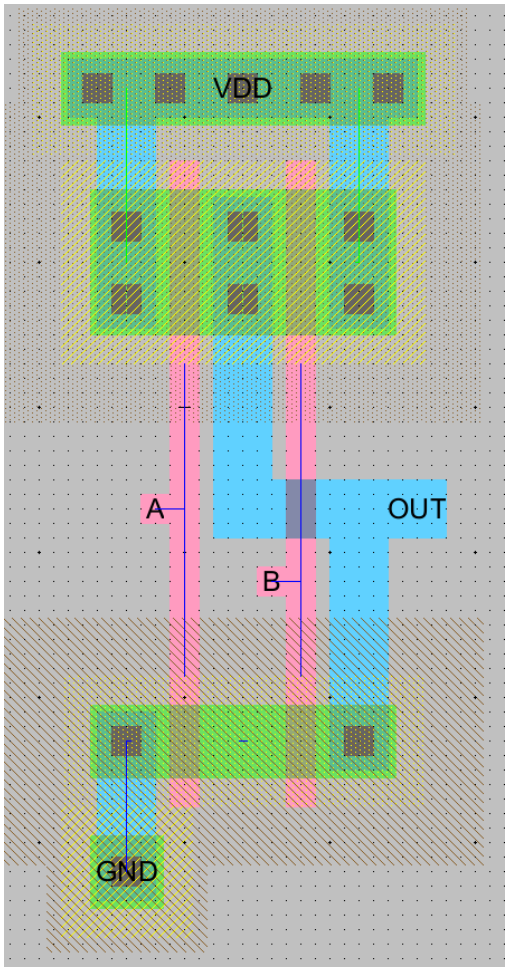
CMOS NOR Schematic:



CMOS NOR SIMULATED OUTPUT:



CMOS NAND Schematic:



CMOS NAND Simulation Output: