LIST OF EXPERIMENTS

Cycle I:

Part-1

To Verify the Functionality of the following 74 Series ICs:

1. 3- 8 Decoder – 74LS138.
2. 8X1 Multiplexer– 74151 and 2X4 De-multiplexer- 74155.
3. 2-bit COMPARATOR -74LS85.
4. D-Flip- Flop – (74LS74) and JK Flip- Flop (74LS73).

Part-2

Design and simulate the following Circuits using HDL:

1. Logic Gates.
2. Adders and Subtractors
3. Code converters
4. Multiplexer and De-multiplexer.
5. Encoder and Decoder.
6. Parity generator and checker
7. Flip Flops using Truth table and FSM
8. Shift Registers
9. Asynchronous counters
10. Synchronous counters

Cycle II:

1. Development of one application which shall cover maximum no. of Experiments in Cycle I.

EXPERIMENT-1
REALIZATION OF LOGIC GATES (NOT, AND, OR, NAND, NOR, XOR, XNOR)

1. **AIM:** To Design, simulate and implement all Logic Gates in 3 different modeling styles(Data Flow, Behavioral Flow, Structural Flow Modeling)

2. **SOFTWARE USED:** Xilinx Vivado 2022.2

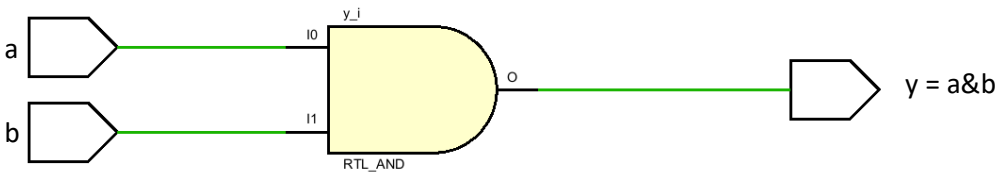
3. **PROCEDURE:**
   - Open the Xilinx Vivado project navigator.
   - Open the Design source and go to add / create sources
   - Create new file, give appropriate name save it.
   - Open the file in the editor and write the Verilog code.
   - Open the Design source and go to add / create sources to create the test bench
   - Open the editor and write the Verilog code for test bench.
   - After completion of Verilog code set your test bench as top module in simulation settings and Run Simulation.
   - To view RTL schematic, select RTL Analysis in which select open elaborate design, select Schematic.

LOGIC GATES (Data Flow Modeling)

1.1 AIM: To implement AND Gate

1.2 SOFTWARE USED: Xilinx Vivado 2022.2

1.3 SYMBOL:



1.4 LOGIC EXPRESSION:

Y = (A&B)

1.5 BOOLEAN EXPRESSION:

Y = (A.B)

1.6 TRUTH TABLE:

INPUT		OUTPUT
A	**B**	**Y**
0	**0**	**0**
0	**1**	**0**
1	**0**	**0**
1	**1**	**1**

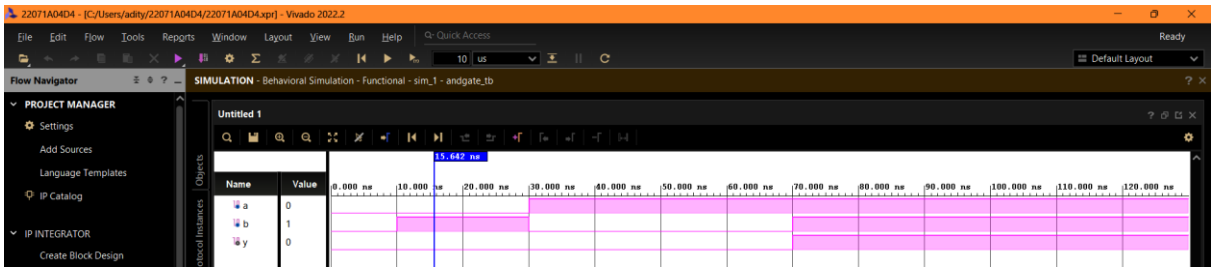
1.7 VERILOG CODE (andgate_df.v):

```
module andgate_df(y,a,b);
output y;
input a,b;

assign y=a&b;

endmodule
```

1.8 TEST BENCH CODE (andgate_df_tb.v):

```
module andgate_tb();
reg a,b;
wire y;
andgate_df x1(y,a,b);
initial
begin
   a=1'b0; b=1'b0;
#10 a=1'b0; b=1'b1;
#20 a=1'b1; b=1'b0;
#40 a=1'b1; b=1'b1;
#60 $finish;
end
endmodule
```

1.9 WAVEFORM:



1.10 OBSERVATION:

Output is high for both high inputs and low for all other cases.

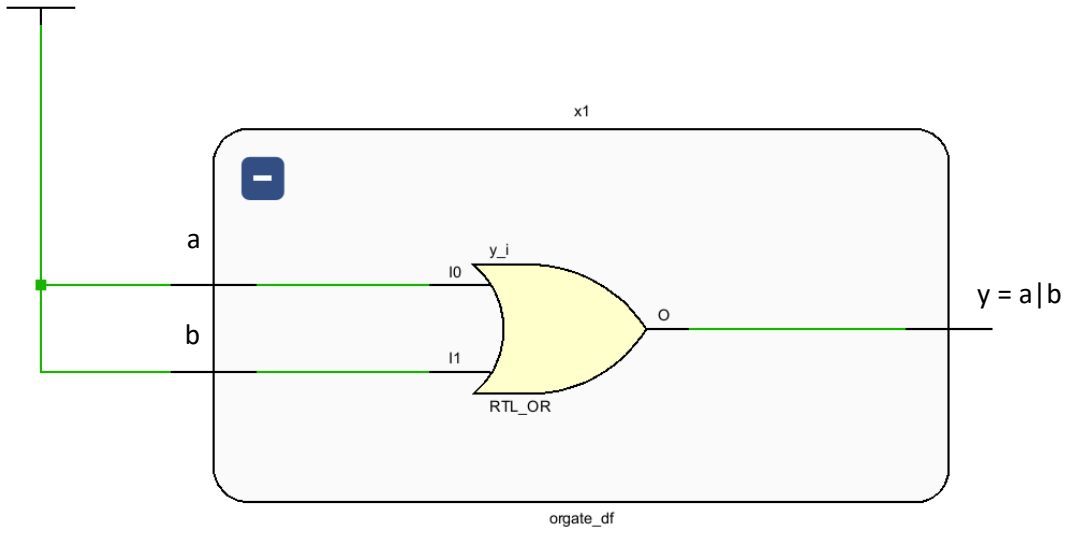
1.11 RESULT:

The Logical AND Gate is simulated and implemented in Data Flow Modeling.

2.1 AIM: To implement OR Gate

2.2 SOFTWARE USED: Xilinx Vivado 2022.2

2.3 SYMBOL:



2.4 LOGIC EXPRESSION:

Y = (A|B)

2.5 BOOLEAN EXPRESSION:

Y = (A+B)

2.6 TRUTH TABLE:

INPUT		OUTPUT
A	**B**	**Y**
0	**0**	**0**
0	**1**	**1**
1	**0**	**1**
1	**1**	**1**

2.7 VERILOG CODE (orgate_df.v):

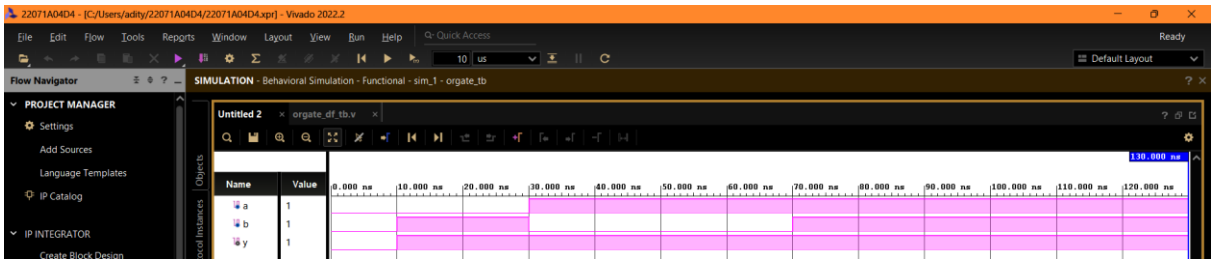
```
module orgate_df(y,a,b);
output y;
input a,b;

assign y=a|b;
```

```
endmodule
```

2.8 TEST BENCH CODE (orgate_df_tb.v):

```
module orgate_tb();
reg a,b;
wire y;
orgate_df x1(y,a,b);
initial
begin
   a=1'b0; b=1'b0;
#10 a=1'b0; b=1'b1;
#20 a=1'b1; b=1'b0;
#40 a=1'b1; b=1'b1;
#60 $finish;
end
endmodule
```

2.9 WAVEFORM:



2.10 OBSERVATION:

Output is low for both low inputs and high for all other cases.

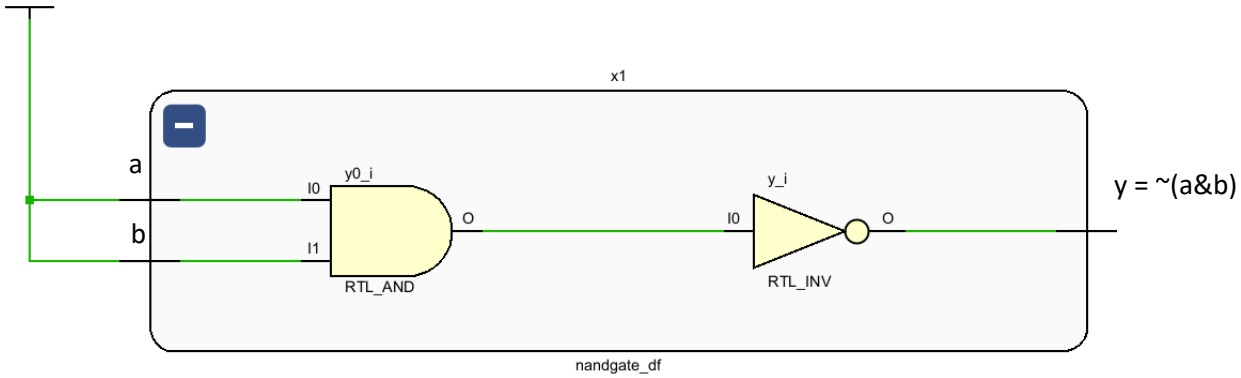
2.11 RESULT:

The Logical OR Gate is simulated and implemented in Data Flow Modeling.

3.1 AIM: To implement NAND Gate

3.2 SOFTWARE USED: Xilinx Vivado 2022.2

3.3 SYMBOL:



3.4 LOGICAL EXPRESSION:

$$Y = \sim(A\&B)$$

3.5 BOOLEAN EXPRESSION:

$$Y = \overline{(A.B)}$$

3.6 TRUTH TABLE:

INPUT		OUTPUT
A	**B**	**Y**
0	**0**	**1**
0	**1**	**1**
1	**0**	**1**
1	**1**	**0**

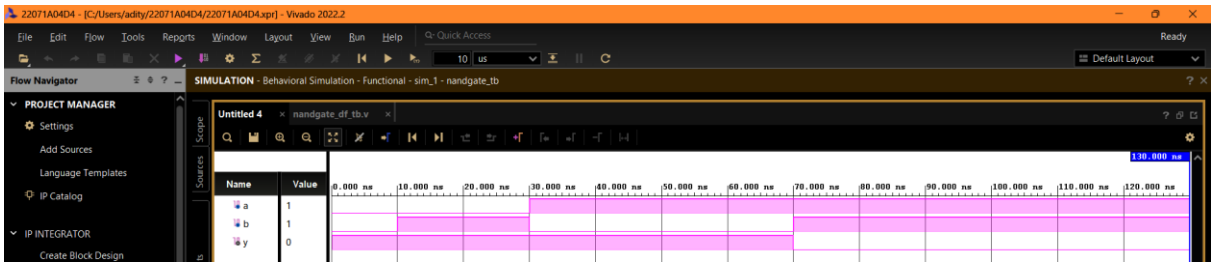
3.7 VERILOG CODE (nandgate_df.v):

```
module nandgate_df(y,a,b);
output y;
input a,b;

assign y=~(a&b);

endmodule
```

3.8 TEST BENCH CODE (nandgate_df_tb.v):

```
module nandgate_tb();
reg a,b;
wire y;
nandgate_df x1(y,a,b);
initial
begin
   a=1'b0; b=1'b0;
#10 a=1'b0; b=1'b1;
#20 a=1'b1; b=1'b0;
#40 a=1'b1; b=1'b1;
#60 $finish;
end
endmodule
```

3.9 WAVEFORM:



3.10 OBSERVATION:

Output is low for both high inputs and high for all other cases.

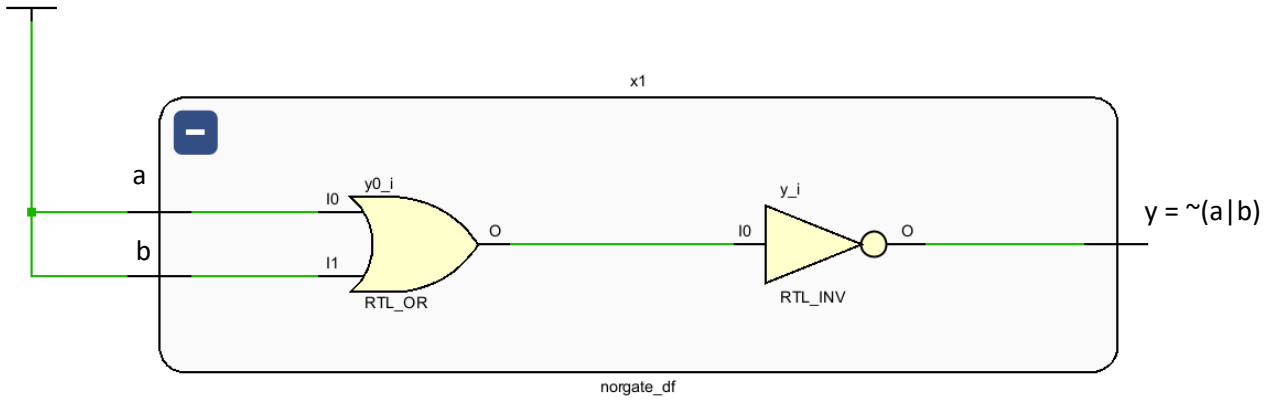
3.11 RESULT:

The Logical NAND Gate is simulated and implemented in Data Flow Modeling.

4.1 AIM: To implement NOR Gate

4.2 SOFTWARE USED: Xilinx Vivado 2022.2

4.3 SYMBOL:



4.4 LOGICAL EXPRESSION:

$$Y = \sim(A|B)$$

4.5 BOOLEAN EXPRESSION:

$$Y = \overline{(A+B)}$$

4.6 TRUTH TABLE:

INPUT		OUTPUT
A	**B**	**Y**
0	**0**	**1**
0	**1**	**0**
1	**0**	**0**
1	**1**	**0**

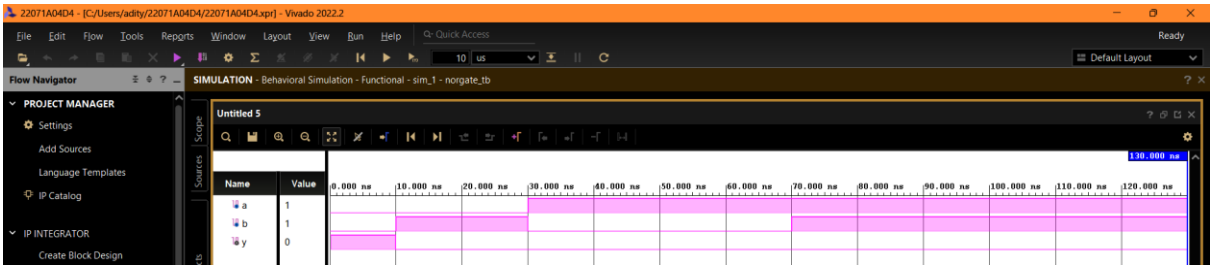
4.7 VERILOG CODE (norgate_df.v):

```
module norgate_df(y,a,b);
output y;
input a,b;

assign y=~(a|b);

endmodule
```

4.8 TEST BENCH CODE (norgate_df_tb.v):

```
module norgate_tb();
reg a,b;
wire y;
norgate_df x1(y,a,b);
initial
begin
   a=1'b0; b=1'b0;
#10 a=1'b0; b=1'b1;
#20 a=1'b1; b=1'b0;
#40 a=1'b1; b=1'b1;
#60 $finish;
end
endmodule
```

4.9 WAVEFORM:



4.10 OBSERVATION:

Output is high for both low inputs and low for all other cases.

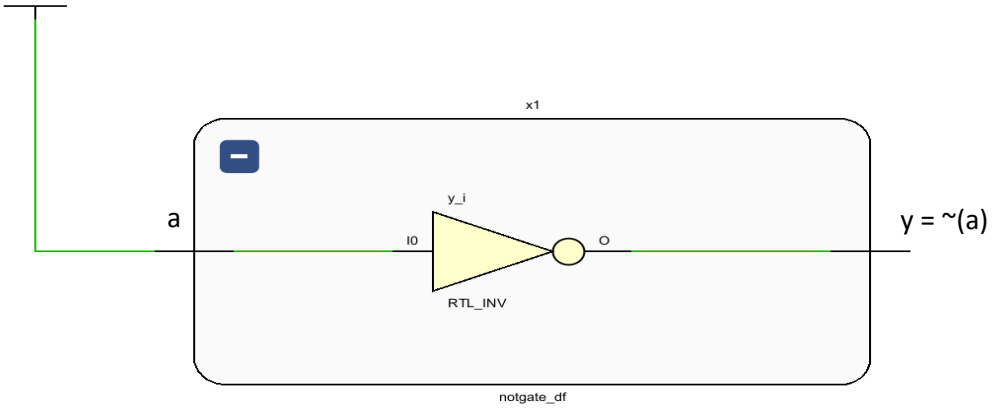
4.11 RESULT:

The Logical NOR Gate is simulated and implemented in Data Flow Modeling.

5.1 AIM: To implement NOT Gate

5.2 SOFTWARE USED: Xilinx Vivado 2022.2

5.3 SYMBOL:



5.4 LOGICAL EXPRESSION:

Y=~(A)

5.5 BOOLEAN EXPRESSION:

$Y = (\overline{A})$

5.6 TRUTH TABLE:

INPUT	OUTPUT
A	**Y**
0	**1**
0	**0**

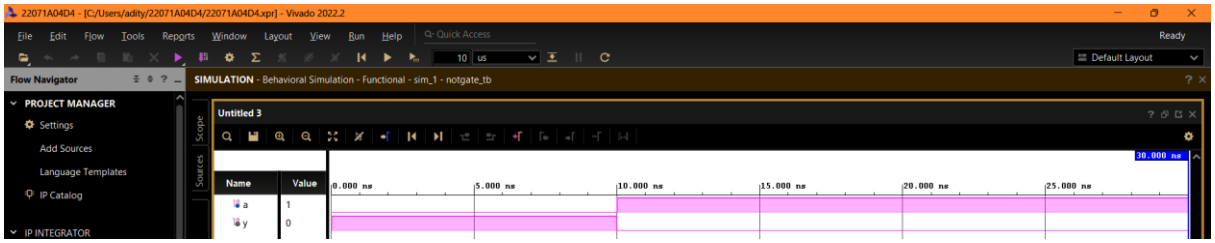
5.7 VERILOG CODE (notgate_df.v):

```
module notgate_df(y,a);
output y;
input a;

assign y=~(a);

endmodule
```

5.8 TEST BENCH CODE (notgate_df_tb.v):

```
module notgate_tb();
reg a;
wire y;
notgate_df x1(y,a);
initial
begin
   a=1'b0; b=1'b0;
#10 a=1'b0; b=1'b1;
#10 $finish;
end
endmodule
```

5.9 WAVEFORM:



5.10 OBSERVATION:

Output is high for low input and low for high input.

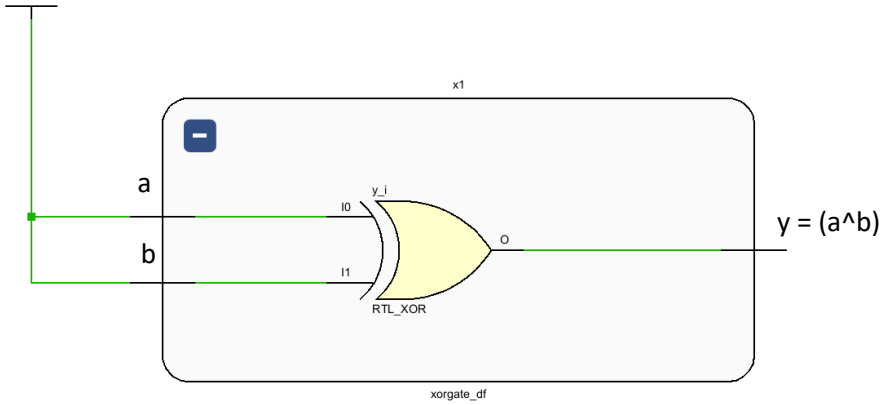
5.11 RESULT:

The Logical NOT Gate is simulated and implemented in Data Flow Modeling.

6.1 AIM: To implement XOR Gate

6.2 SOFTWARE USED: Xilinx Vivado 2022.2

6.3 SYMBOL:



6.4 LOGICAL EXPRESSION:

$$Y = A \wedge B$$

6.5 BOOLEAN EXPRESSION:

$$Y = (\overline{A}B + A\overline{B})$$

6.6 TRUTH TABLE:

INPUT		OUTPUT
A	**B**	**Y**
0	**0**	**0**
0	**1**	**1**
1	**0**	**1**
1	**1**	**0**

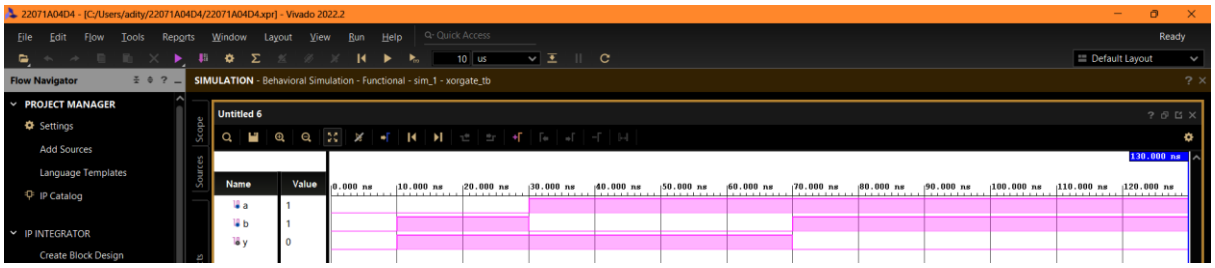
6.7 VERILOG CODE (xorgate_df.v):

```
module xorgate_df(y,a,b);
output y;
input a,b;

assign y=(a^b);

endmodule
```

6.8 TEST BENCH CODE (xorgate_df_tb.v):

```
module xorgate_tb();
reg a,b;
wire y;
xorgate_df x1(y,a,b);
initial
begin
   a=1'b0; b=1'b0;
#10 a=1'b0; b=1'b1;
#20 a=1'b1; b=1'b0;
#40 a=1'b1; b=1'b1;
#60 $finish;
end
endmodule
```

6.9 WAVEFORM:



6.10 OBSERVATION:

Output is low for both same inputs and high for all other cases.

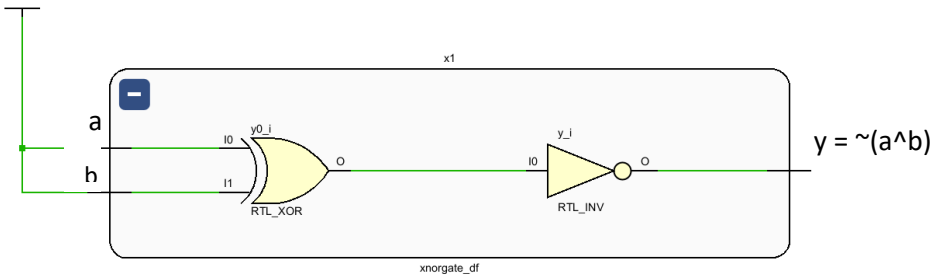
6.11 RESULT:

The Logical XOR Gate is simulated and implemented in Data Flow Modeling.

7.1 AIM: To implement XNOR Gate

7.2 SOFTWARE USED: Xilinx Vivado 2022.2

7.3 SYMBOL:



7.4 LOGICAL EXPRESSION:

$$Y = \sim(A \wedge B)$$

7.5 BOOLEAN EXPRESSION:

$$Y = (A.B+\overline{A}.\overline{B})$$

7.6 TRUTH TABLE:

INPUT		OUTPUT
A	**B**	**Y**
0	**0**	**1**
0	**1**	**0**
1	**0**	**0**
1	**1**	**1**

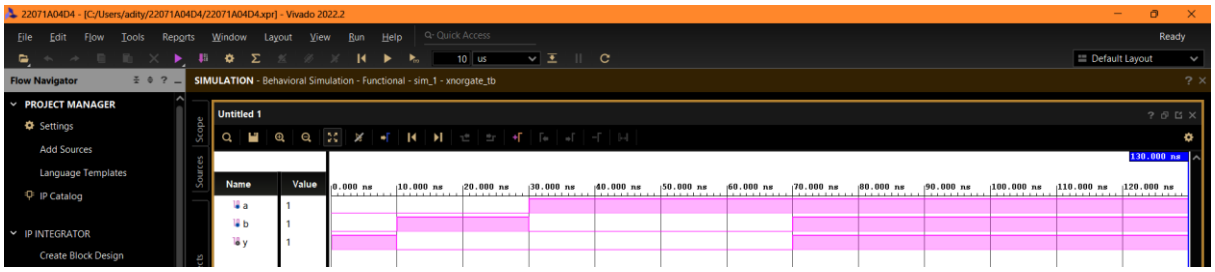
7.7 VERILOG CODE (xnorgate_df.v):

```
module xnorgate_df(y,a,b);
output y;
input a,b;

assign y=~(a^b);

endmodule
```

7.8 TEST BENCH CODE (xnorgate_df_tb.v):

```
module xnorgate_tb();
reg a,b;
wire y;
xnorgate_df x1(y,a,b);
initial
begin
   a=1'b0; b=1'b0;
#10 a=1'b0; b=1'b1;
#20 a=1'b1; b=1'b0;
#40 a=1'b1; b=1'b1;
#60 $finish;
end
endmodule
```

7.9 WAVEFORM:



7.10 OBSERVATION:

Output is high for both same inputs and low for all other cases.

7.11 RESULT:

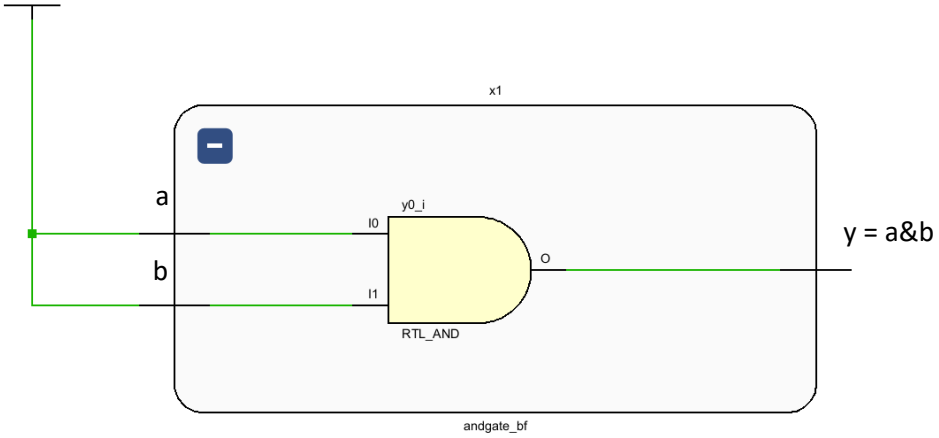
The Logical XNOR Gate is simulated and implemented in Data Flow Modeling.

LOGIC GATES (Behavioral Flow Modeling)

1.12 AIM: To implement AND Gate

1.13 SOFTWARE USED: Xilinx Vivado 2022.2

1.14 SYMBOL:



1.15 LOGICAL EXPRESSION:

Y = (A&B)

1.16 BOOLEAN EXPRESSION:

Y = (A.B)

1.17 TRUTH TABLE:

INPUT		OUTPUT
A	**B**	**Y**
0	**0**	**0**
0	**1**	**0**
1	**0**	**0**
1	**1**	**1**

1.18 VERILOG CODE (andgate_bf.v):

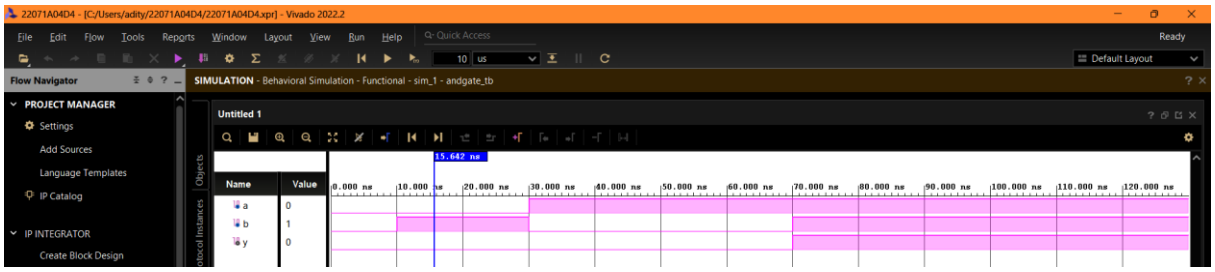
```
module andgate_bf(y,a,b);
output y;
input a,b;
reg y;
always@(a,b)
begin
if(a==1'b1 & b==1'b1)
```

```
y=1;
else
y=0;
end
endmodule
```

1.19 TEST BENCH CODE (andgate_bf_tb.v):

```
module andgate_bf_tb();
reg a,b;
wire y;
andgate_bf x1(y,a,b);
initial
begin
   a=1'b0; b=1'b0;
#10 a=1'b0; b=1'b1;
#20 a=1'b1; b=1'b0;
#40 a=1'b1; b=1'b1;
#60 $finish;
end
endmodule
```

1.20 WAVEFORM:



1.21 OBSERVATION:

Output is high for both high inputs and low for all other cases.

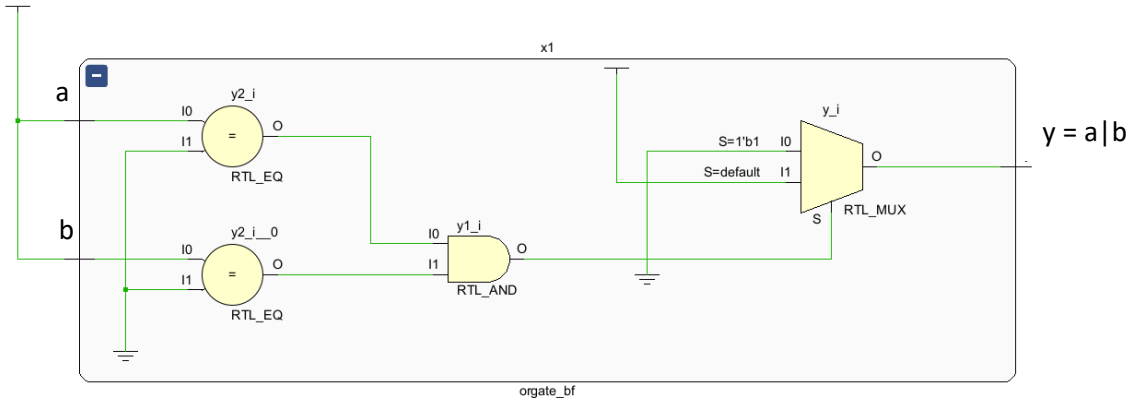
1.22 RESULT:

The Logical AND Gate is simulated and implemented in Behavioral Flow Modeling.

2.1 AIM: To implement OR Gate

2.2 SOFTWARE USED: Xilinx Vivado 2022.2

2.3 SYMBOL:



2.4 LOGICAL EXPRESSION:

Y = A|B

2.5 BOOLEAN EXPRESSION:

Y = (A+B)

2.6 TRUTH TABLE:

INPUT		OUTPUT
A	**B**	**Y**
0	**0**	**0**
0	**1**	**1**
1	**0**	**1**
1	**1**	**1**

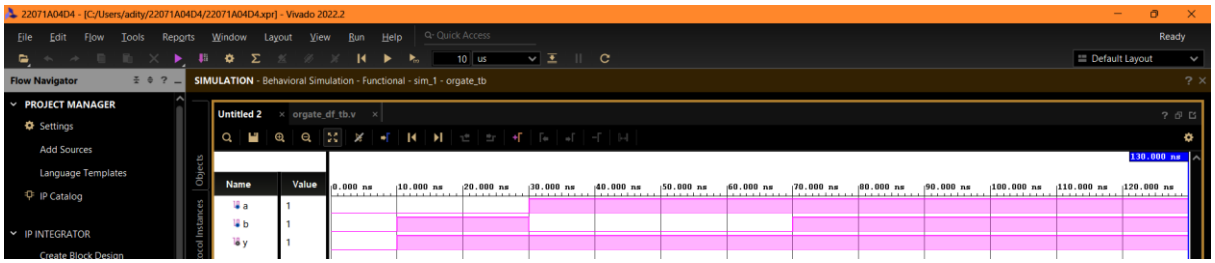
2.7 VERILOG CODE (orgate_bf.v):

```
module orgate_bf(y,a,b);
output y;
input a,b;
reg y;
always@(a,b)
begin
if(a==1'b0 & b==1'b0)
y=0;
else
y=1;
end
endmodule
```

2.8 TEST BENCH CODE (orgate_bf_tb.v):

```
module orgate_bf_tb();
reg a,b;
wire y;
orgate_bf x1(y,a,b);
initial
begin
   a=1'b0; b=1'b0;
#10 a=1'b0; b=1'b1;
#20 a=1'b1; b=1'b0;
#40 a=1'b1; b=1'b1;
#60 $finish;
end
endmodule
```

2.9 WAVEFORM:



2.10 OBSERVATION:

Output is low for both low inputs and high for all other cases.

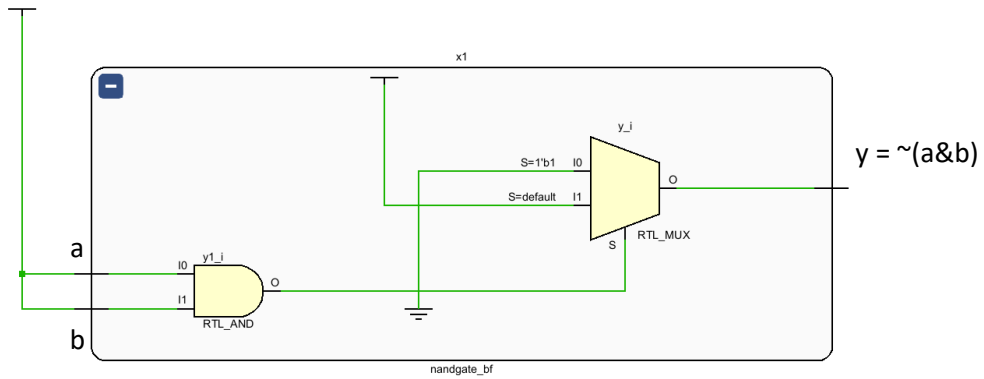
2.11 RESULT:

The Logical OR Gate is simulated and implemented in Behavioral Flow Modeling.

3.1 AIM: To implement NAND Gate

3.2 SOFTWARE USED: Xilinx Vivado 2022.2

3.3 SYMBOL:



3.4 LOGICAL EXPRESSION:

$$Y = \sim(A\&B)$$

3.5 BOOLEAN EXPRESSION:

$$Y = \overline{(A.B)}$$

3.6 TRUTH TABLE:

INPUT		OUTPUT
A	**B**	**Y**
0	**0**	**1**
0	**1**	**1**
1	**0**	**1**
1	**1**	**0**

3.7 VERILOG CODE (nandgate_bf.v):

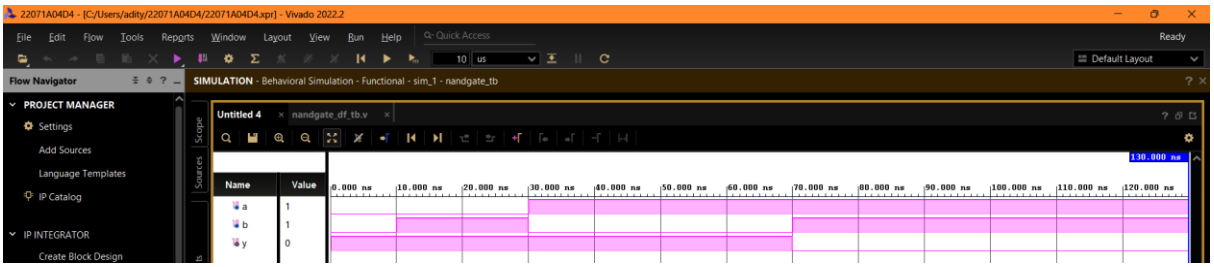
```
module nandgate_bf(y,a,b);
output y;
input a,b;
reg y;
always@(a,b)
begin
if(a==1'b1 & b==1'b1)
```

```
y=0;
else
y=1;
end
endmodule
```

3.8 TEST BENCH CODE (nandgate_bf_tb.v):

```
module nandgate_bf_tb();
reg a,b;
wire y;
nandgate_bf x1(y,a,b);
initial
begin
   a=1'b0; b=1'b0;
#10 a=1'b0; b=1'b1;
#20 a=1'b1; b=1'b0;
#40 a=1'b1; b=1'b1;
#60 $finish;
end
endmodule
```

3.9 WAVEFORM:



3.10 OBSERVATION:

Output is low for both high inputs and high for all other cases.

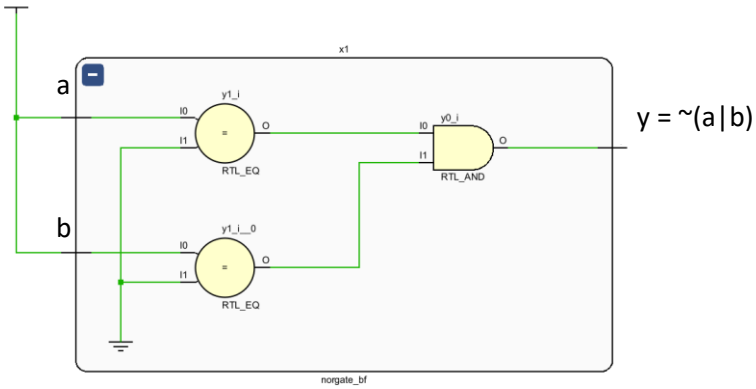
3.11 RESULT:

The Logical NAND Gate is simulated and implemented in Behavioral Flow Modeling.

4.1 AIM: To implement NOR Gate

4.2 SOFTWARE USED: Xilinx Vivado 2022.2

4.3 SYMBOL:



4.4 LOGICAL EXPRESSION:

$$Y = \sim(A|B)$$

4.5 BOOLEAN EXPRESSION:

$$Y = \overline{(A+B)}$$

4.6 TRUTH TABLE:

INPUT		OUTPUT
A	**B**	**Y**
0	**0**	**1**
0	**1**	**0**
1	**0**	**0**
1	**1**	**0**

4.7 VERILOG CODE (norgate_bf.v):

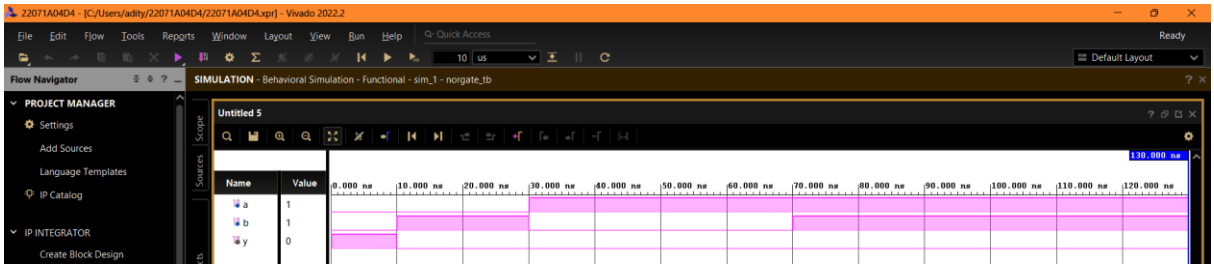
```
module norgate_bf(y,a,b);
output y;
input a,b;
reg y;
always@(a,b)
begin
if(a==1'b0 & b==1'b0)
y=1;
else
y=0;
end
```

```
endmodule
```

4.8 TEST BENCH CODE (norgate_bf_tb.v):

```
module norgate_bf_tb();
reg a,b;
wire y;
norgate_bf x1(y,a,b);
initial
begin
   a=1'b0; b=1'b0;
#10 a=1'b0; b=1'b1;
#20 a=1'b1; b=1'b0;
#40 a=1'b1; b=1'b1;
#60 $finish;
end
endmodule
```

4.9 WAVEFORM:



4.10 OBSERVATION:

Output is high for both low inputs and low for all other cases.

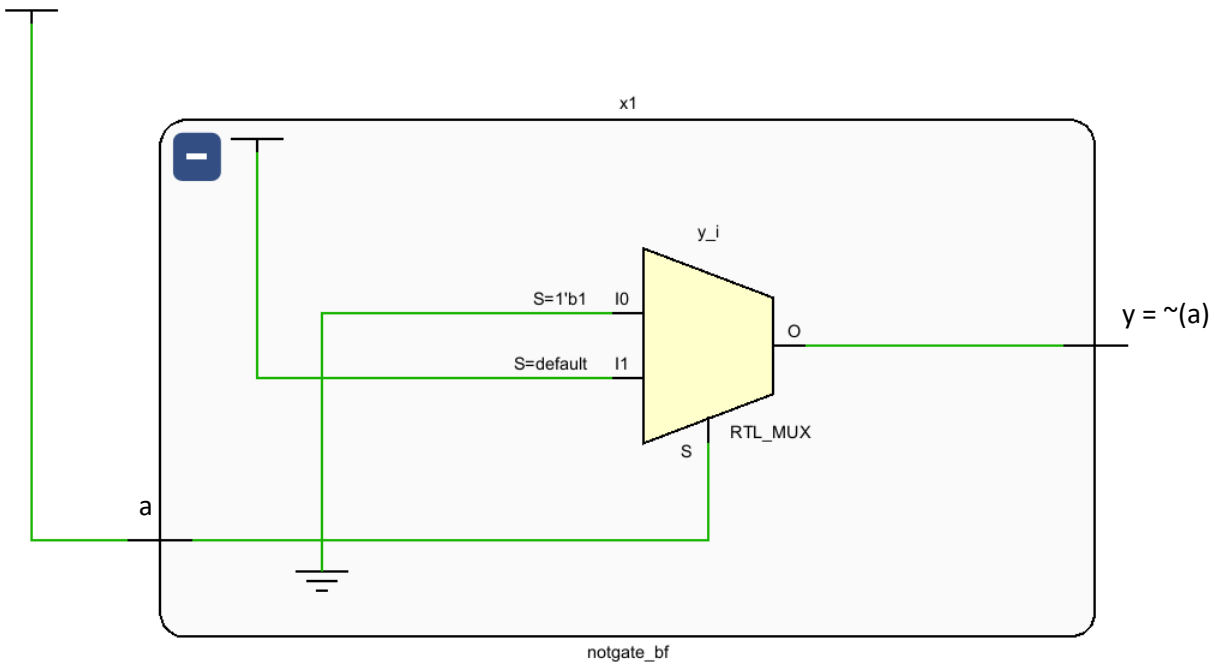
4.11 RESULT:

The Logical NOR Gate is simulated and implemented in Behavioral Flow Modeling.

5.1 AIM: To implement NOT Gate

5.2 SOFTWARE USED: Xilinx Vivado 2022.2

5.3 SYMBOL:



5.4 LOGIC EXPRESSION:

$$Y = \sim(A)$$

5.5 BOOLEAN EXPRESSION:

$$Y = (\overline{A})$$

5.6 TRUTH TABLE:

INPUT	OUTPUT
A	**Y**
0	**1**
0	**0**

5.7 VERILOG CODE (notgate_bf.v):

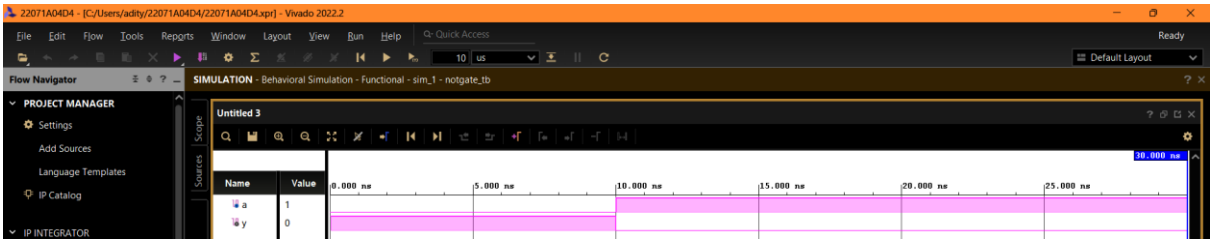
```
module notgate_bf(y,a);
output y;
input a;
reg y;
always@(a)
```

```
begin
if(a==1'b1)
y=0;
else
y=1;
end
endmodule
```

5.8 TEST BENCH CODE (notgate_bf_tb.v):

```
module notgate_bf_tb();
reg a;
wire y;
notgate_bf x1(y,a);
initial
begin
   a=1'b0; b=1'b0;
#10 a=1'b0; b=1'b1;
#10 $finish;
end
endmodule
```

5.9 WAVEFORM:



5.10 OBSERVATION:

Output is high for low input and low for high input.

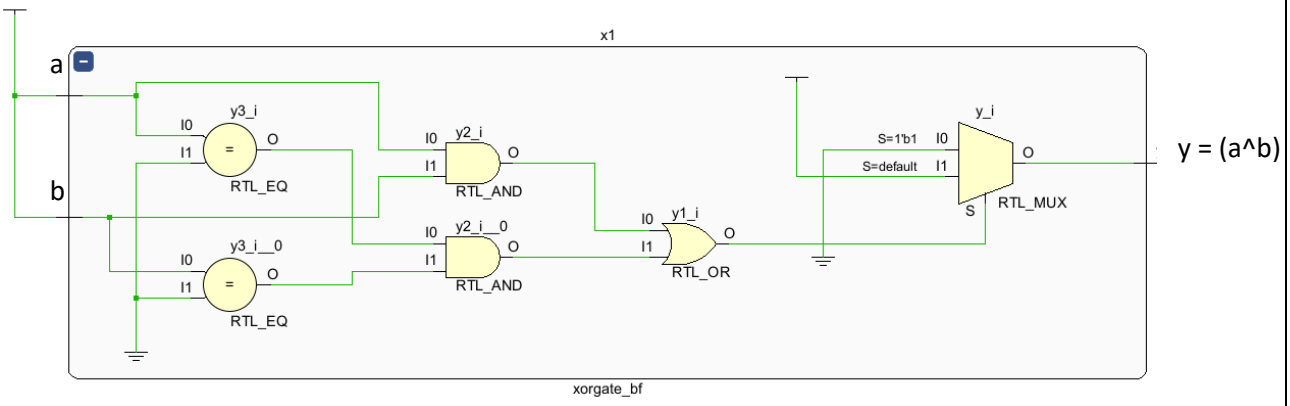
5.11 RESULT:

The Logical NOT Gate is simulated and implemented in Behavioral Flow Modeling.

6.1 AIM: To implement XOR Gate

6.2 SOFTWARE USED: Xilinx Vivado 2022.2

6.3 SYMBOL:



6.4 LOGIC EXPRESSOION:

$$Y = A \wedge B$$

6.5 BOOLEAN EXPRESSION:

$$Y = (\bar{A}B + A\bar{B})$$

6.6 TRUTH TABLE:

INPUT		OUTPUT
A	**B**	**Y**
0	**0**	**0**
0	**1**	**1**
1	**0**	**1**
1	**1**	**0**

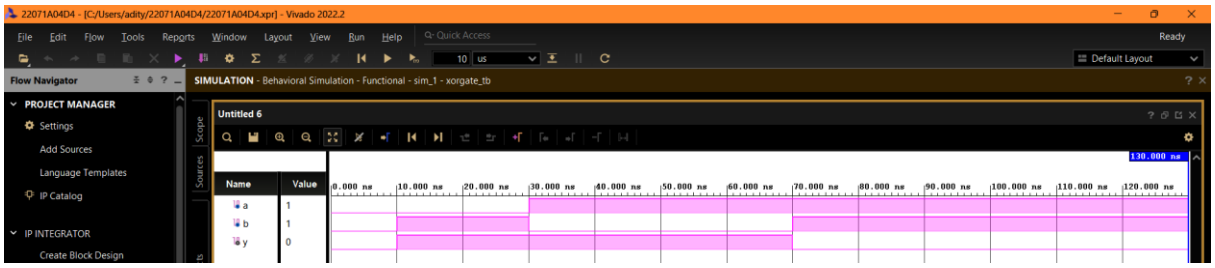
6.7 VERILOG CODE (xorgate_bf.v):

```
module xorgate_bf(y,a,b);
output y;
input a,b;
reg y;
always@(a,b)
begin
if((a==1'b1 & b==1'b1)|(a==1'b0 & b==1'b0))
y=0;
else
y=1;
end
endmodule
```

6.8 TEST BENCH CODE (xorgate_bf_tb.v):

```
module xorgate_bf_tb();
reg a,b;
wire y;
xorgate_bf x1(y,a,b);
initial
begin
   a=1'b0; b=1'b0;
#10 a=1'b0; b=1'b1;
#20 a=1'b1; b=1'b0;
#40 a=1'b1; b=1'b1;
#60 $finish;
end
endmodule
```

6.9 WAVEFORM:



6.10 OBSERVATION:

Output is low for both same inputs and high for all other cases.

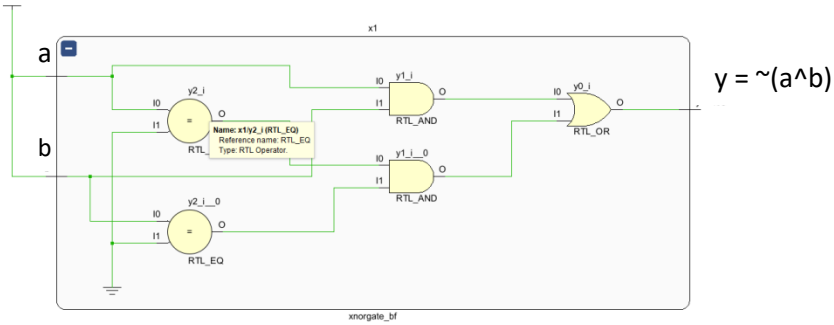
6.11 RESULT:

The Logical XOR Gate is simulated and implemented in Behavioral Flow Modeling.

7.1 AIM: To implement XNOR Gate

7.2 SOFTWARE USED: Xilinx Vivado 2022.2

7.3 SYMBOL:



7.4 LOGIC EXPRESSION:

$$Y=\sim(A\wedge B)$$

7.5 BOOLEAN EXPRESSION:

$$Y = (A.B+\overline{A}.\overline{B})$$

7.6 TRUTH TABLE:

INPUT		OUTPUT
A	**B**	**Y**
0	**0**	**1**
0	**1**	**0**
1	**0**	**0**
1	**1**	**1**

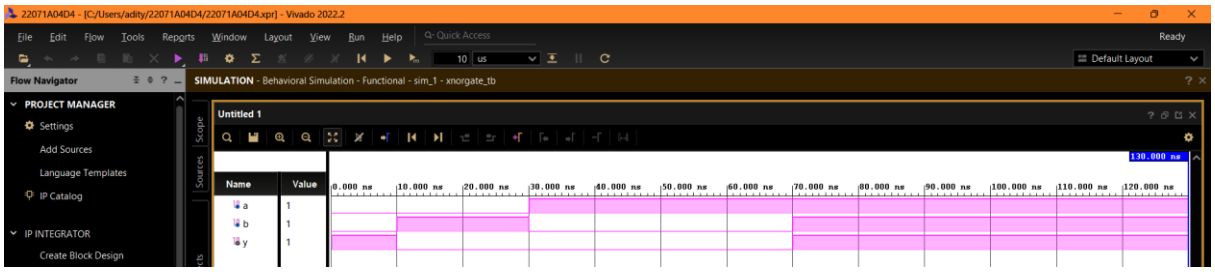
7.7 VERILOG CODE (xnorgate_bf.v):

```
module xnorgate_bf(y,a,b);
output y;
input a,b;
reg y;
always@(a,b)
begin
if((a==1'b1 & b==1'b1)|(a==1'b0 & b==1'b0))
y=1;
else
y=0;
end
endmodule
```

7.8 TEST BENCH CODE (xnorgate_bf_tb.v):

```
module xnorgate_bf_tb();
reg a,b;
wire y;
xnorgate_bf x1(y,a,b);
initial
begin
   a=1'b0; b=1'b0;
#10 a=1'b0; b=1'b1;
#20 a=1'b1; b=1'b0;
#40 a=1'b1; b=1'b1;
#60 $finish;
end
endmodule
```

7.9 WAVEFORM:



7.10 OBSERVATION:

Output is high for both same inputs and low for all other cases.

7.11 RESULT:

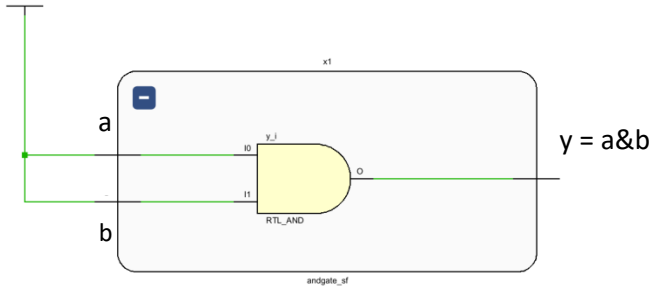
The Logical XNOR Gate is simulated and implemented in Behavioral Flow Modeling.

LOGIC GATES (Structural Flow Modeling)

1.1 AIM: To implement AND Gate

1.2 SOFTWARE USED: Xilinx Vivado 2022.2

1.3 SYMBOL:



1.4 LOGIC EXPRESSION:

Y = (A&B)

1.5 BOOLEAN EXPRESSION:

Y = (A.B)

1.6 TRUTH TABLE:

INPUT		OUTPUT
A	**B**	**Y**
0	**0**	**0**
0	**1**	**0**
1	**0**	**0**
1	**1**	**1**

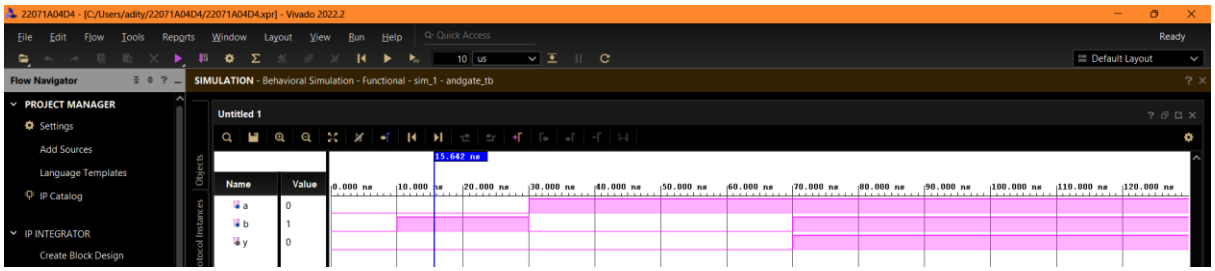
1.7 VERILOG CODE (andgate_sf.v):

```
module andgate_sf(y,a,b);
output y;
input a,b;
wire y;
and (y,a,b);
endmodule
```

1.8 TEST BENCH CODE (andgate_sf_tb.v):

```
module andgate_sf_tb();
reg a,b;
wire y;
andgate_sf x1(y,a,b);
initial
begin
   a=1'b0; b=1'b0;
#10 a=1'b0; b=1'b1;
#20 a=1'b1; b=1'b0;
#40 a=1'b1; b=1'b1;
#60 $finish;
end
endmodule
```

1.9 WAVEFORM:



1.10 OBSERVATION:

Output is high for both high inputs and low for all other cases.

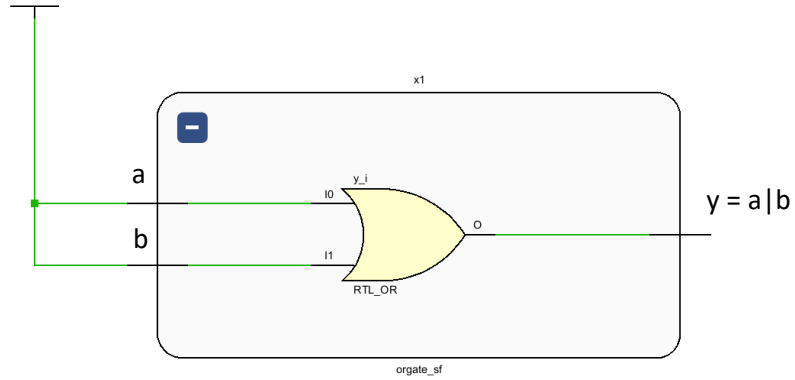
1.11 RESULT:

The Logical AND Gate is simulated and implemented in Structural Flow Modeling.

2.1 AIM: To implement OR Gate

2.2 SOFTWARE USED: Xilinx Vivado 2022.2

2.3 SYMBOL:



2.4 LOGIC EXPRESSION:

$$Y = (A|B)$$

2.5 BOOLEAN EXPRESSION:

$$Y = (A+B)$$

2.6 TRUTH TABLE:

INPUT		OUTPUT
A	**B**	**Y**
0	**0**	**0**
0	**1**	**1**
1	**0**	**1**
1	**1**	**1**

2.7 VERILOG CODE (orgate_sf.v):

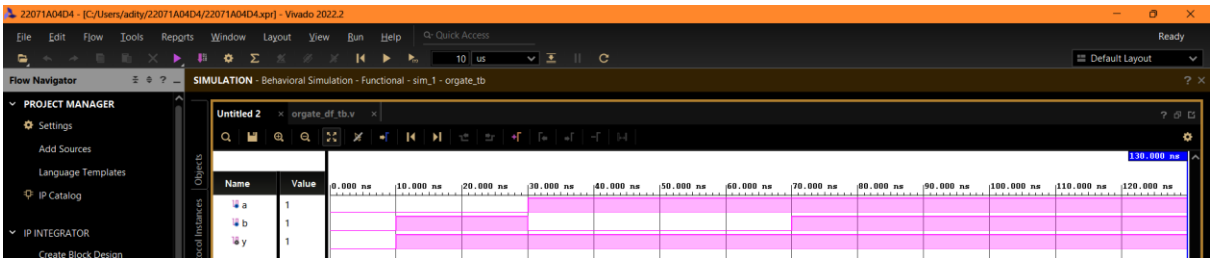
```
module orgate_sf(y,a,b);
output y;
input a,b;
wire y;
or (y,a,b);
endmodule
```

2.8 TEST BENCH CODE (orgate_sf_tb.v):

```
module orgate_sf_tb();
reg a,b;
wire y;
```

```
orgate_sf x1(y,a,b);
initial
begin
   a=1'b0; b=1'b0;
#10 a=1'b0; b=1'b1;
#20 a=1'b1; b=1'b0;
#40 a=1'b1; b=1'b1;
#60 $finish;
end
endmodule
```

2.9 WAVEFORM:



2.10 OBSERVATION:

Output is low for both low inputs and high for all other cases.

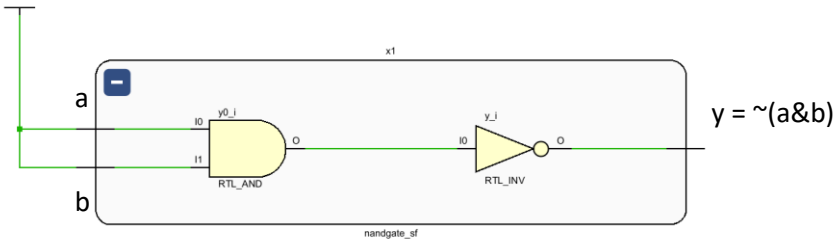
2.11 RESULT:

The Logical OR Gate is simulated and implemented in Structural Flow Modeling.

3.1 AIM: To implement NAND Gate

3.2 SOFTWARE USED: Xilinx Vivado 2022.2

3.3 SYMBOL:



3.4 LOGIC EXPRESSION:

$$Y = \sim(A\&B)$$

3.5 BOOLEAN EXPRESSION:

$$Y = \overline{(A.B)}$$

3.6 TRUTH TABLE:

INPUT		OUTPUT
A	**B**	**Y**
0	**0**	**1**
0	**1**	**1**
1	**0**	**1**
1	**1**	**0**

3.7 VERILOG CODE (nandgate_sf.v):

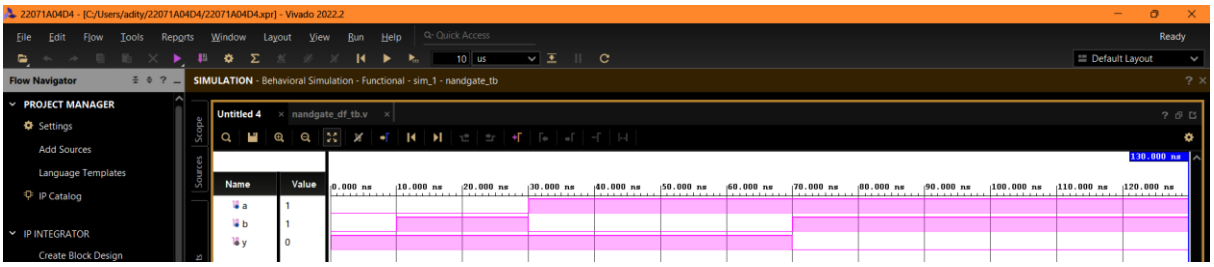
```
module nandgate_sf(y,a,b);
output y;
input a,b;
wire y;
nand (y,a,b);
endmodule
```

3.8 TEST BENCH CODE (nandgate_sf_tb.v):

```
module nandgate_sf_tb();
reg a,b;
wire y;
nandgate_sf x1(y,a,b);
initial
begin
```

```
    a=1'b0; b=1'b0;
#10 a=1'b0; b=1'b1;
#20 a=1'b1; b=1'b0;
#40 a=1'b1; b=1'b1;
#60 $finish;
end
endmodule
```

3.9 WAVEFORM:



3.10 OBSERVATION:

Output is low for both high inputs and high for all other cases.

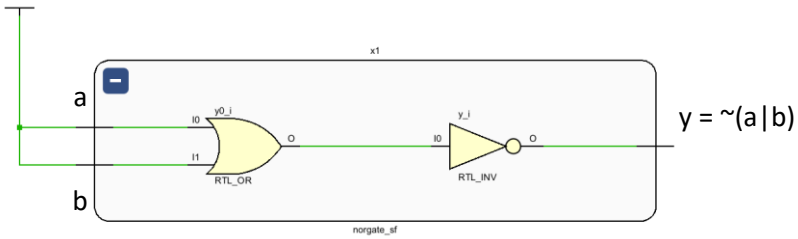
3.11 RESULT:

The Logical NAND Gate is simulated and implemented in Structural Flow Modeling.

4.1 AIM: To implement NOR Gate

4.2 SOFTWARE USED: Xilinx Vivado 2022.2

4.3 SYMBOL:



4.4 LOGIC EXPRESSION:

$$Y=\sim(A|B)$$

4.5 BOOLEAN EXPRESSION:

$$Y = \overline{(A+B)}$$

4.6 TRUTH TABLE:

INPUT		OUTPUT
A	**B**	**Y**
0	**0**	**1**
0	**1**	**0**
1	**0**	**0**
1	**1**	**0**

4.7 VERILOG CODE (norgate_sf.v):

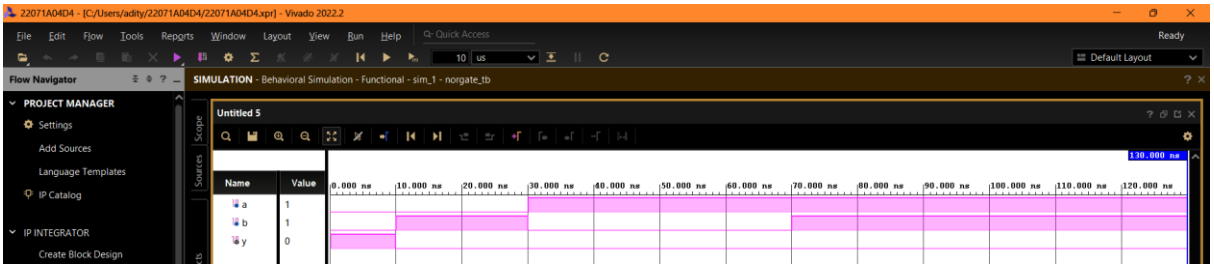
```
module norgate_sf(y,a,b);
output y;
input a,b;
wire y;
nor (y,a,b);
endmodule
```

4.8 TEST BENCH CODE (norgate_sf_tb.v):

```
module norgate_sf_tb();
reg a,b;
wire y;
norgate_sf x1(y,a,b);
initial
```

```
begin
   a=1'b0; b=1'b0;
#10 a=1'b0; b=1'b1;
#20 a=1'b1; b=1'b0;
#40 a=1'b1; b=1'b1;
#60 $finish;
end
endmodule
```

4.9 WAVEFORM:



4.10 OBSERVATION:

Output is high for both low inputs and low for all other cases.

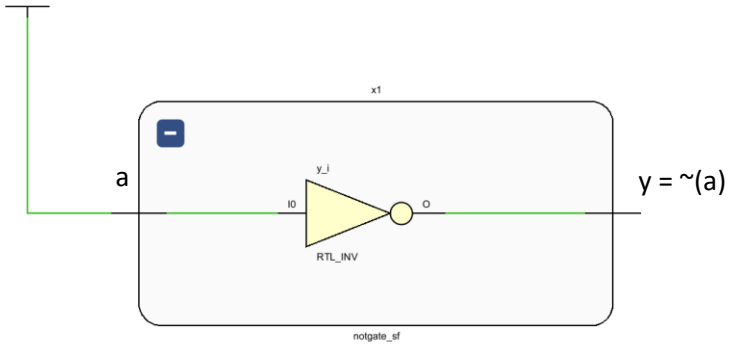
4.11 RESULT:

The Logical NOR Gate is simulated and implemented in Structural Flow Modeling.

5.1 AIM: To implement NOT Gate

5.2 SOFTWARE USED: Xilinx Vivado 2022.2

5.3 SYMBOL:



5.4 LOGIC EXPRESSION:

$$Y = \sim(A)$$

5.5 BOOLEAN EXPRESSION:

$$Y = (\overline{A})$$

5.6 TRUTH TABLE:

INPUT	OUTPUT
A	**Y**
0	**1**
0	**0**

5.7 VERILOG CODE (notgate_sf.v):

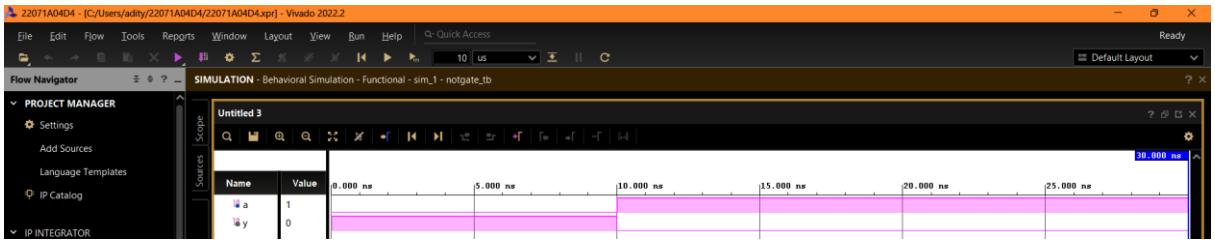
```
module notgate_sf(y,a);
output y;
input a;
wire y;
not (y,a);
endmodule
```

5.8 TEST BENCH CODE (notgate_sf_tb.v):

```
module notgate_sf_tb();
reg a;
wire y;
notgate_sf x1(y,a);
initial
begin
```

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

5.9 WAVEFORM:



5.10 OBSERVATION:

Output is high for low input and low for high input.

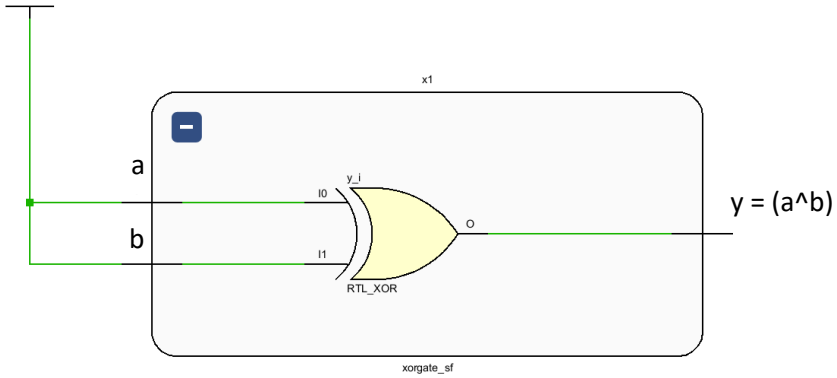
5.11 RESULT:

The Logical NOT Gate is simulated and implemented in Structural Flow Modeling.

6.1 AIM: To implement XOR Gate

6.2 SOFTWARE USED: Xilinx Vivado 2022.2

6.3 SYMBOL:



6.4 LOGIC EXPRESSION:

$$Y = (A \wedge B)$$

6.5 BOOLEAN EXPRESSION:

$$Y = (\overline{A}B + A\overline{B})$$

6.6 TRUTH TABLE:

INPUT		OUTPUT
A	**B**	**Y**
0	**0**	**0**
0	**1**	**1**
1	**0**	**1**
1	**1**	**0**

6.7 VERILOG CODE (xorgate_sf.v):

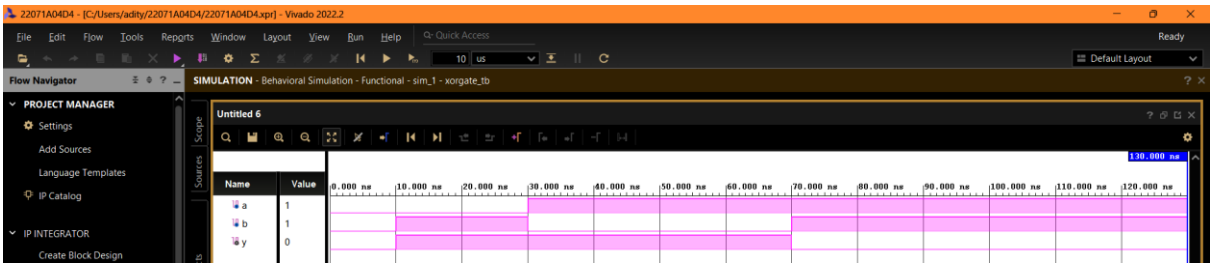
```
module xorgate_sf(y,a,b);
output y;
input a,b;
wire y;
xor (y,a,b);
endmodule
```

6.8 TEST BENCH CODE (xorgate_sf_tb.v):

```
module xorgate_sf_tb();
reg a,b;
wire y;
```

```
xorgate_sf x1(y,a,b);
initial
begin
   a=1'b0; b=1'b0;
#10 a=1'b0; b=1'b1;
#20 a=1'b1; b=1'b0;
#40 a=1'b1; b=1'b1;
#60 $finish;
end
endmodule
```

6.9 WAVEFORM:



6.10 OBSERVATION:

Output is low for both same inputs and high for all other cases.

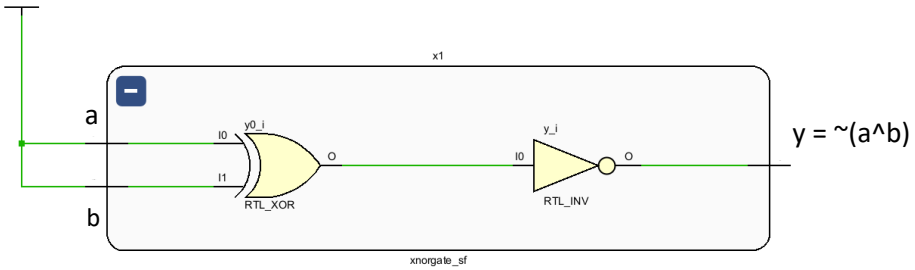
6.11 RESULT:

The Logical XOR Gate is simulated and implemented in Structural Flow Modeling.

7.1 AIM: To implement XNOR Gate

7.2 SOFTWARE USED: Xilinx Vivado 2022.2

7.3 SYMBOL:



7.4 LOGIC EXPRESSION:

$$Y = \sim(A \wedge B)$$

7.5 BOOLEAN EXPRESSION:

$$Y = (A.B+\overline{A}.\overline{B})$$

7.6 TRUTH TABLE:

INPUT		OUTPUT
A	**B**	**Y**
0	**0**	**1**
0	**1**	**0**
1	**0**	**0**
1	**1**	**1**

7.7 VERILOG CODE (xnorgate_sf.v):

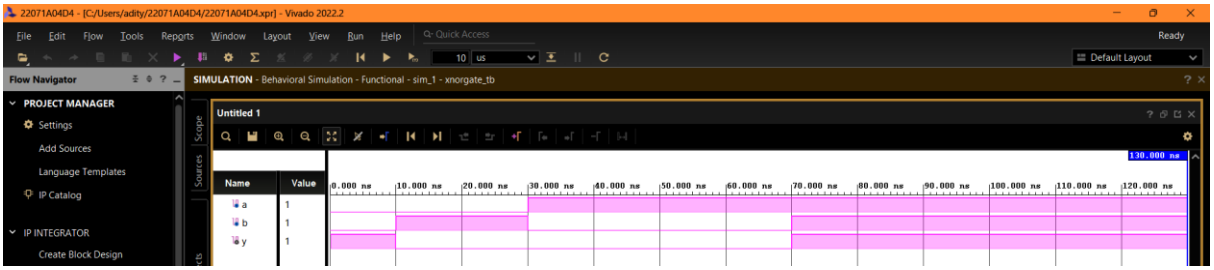
```
module xnorgate_sf(y,a,b);
output y;
input a,b;
wire y;
xnor (y,a,b);
endmodule
```

7.8 TEST BENCH CODE (xnorgate_sf_tb.v):

```
module xnorgate_sf_tb();
reg a,b;
wire y;
xnorgate_sf x1(y,a,b);
initial
```

```
begin
   a=1'b0; b=1'b0;
#10 a=1'b0; b=1'b1;
#20 a=1'b1; b=1'b0;
#40 a=1'b1; b=1'b1;
#60 $finish;
end
endmodule
```

7.9 WAVEFORM:



7.10 OBSERVATION:

Output is high for both same inputs and low for all other cases.

7.11 RESULT:

The Logical XNOR Gate is simulated and implemented in Structural Flow Modeling.