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

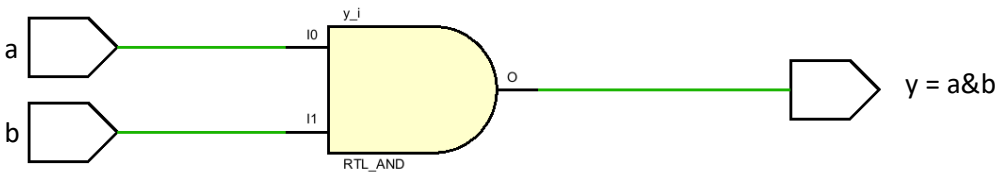
3. **PROCEDURE:**
   - Open the Xilinx Vivado project navigator.
   - Open the Design source and go to add / create sources
   - Create new file, give appropriate name save it.
   - Open the file in the editor and write the Verilog code.
   - Open the Design source and go to add / create sources to create 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**

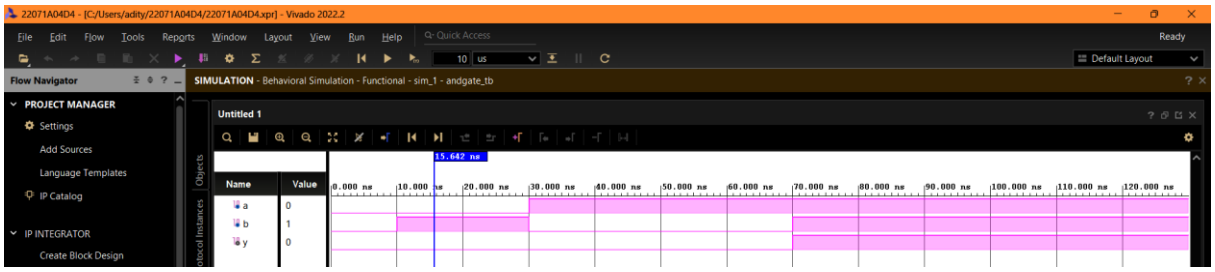
1.7 VERILOG CODE (andgate_df.v):

```
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.

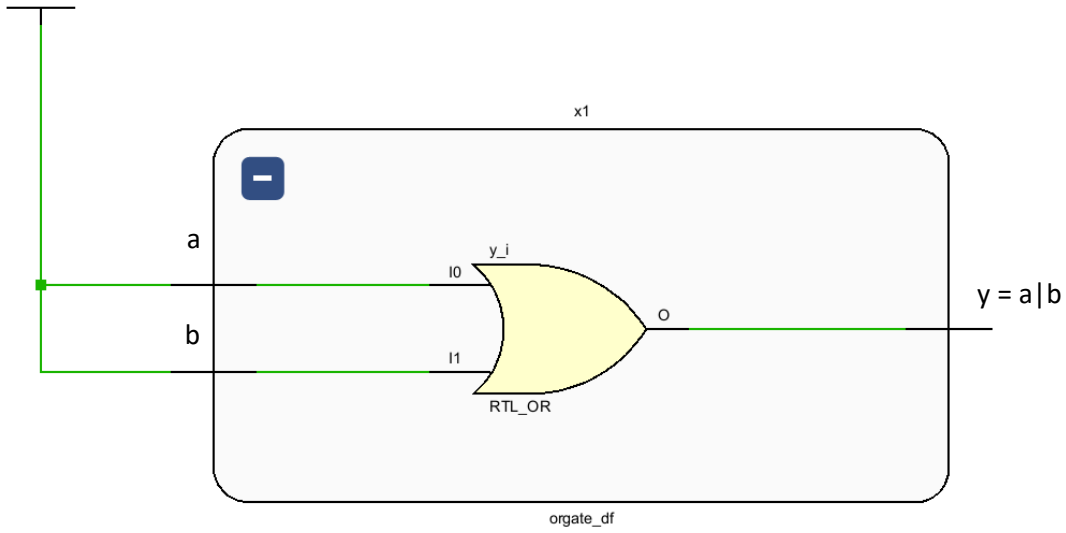
1.11 RESULT:

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):

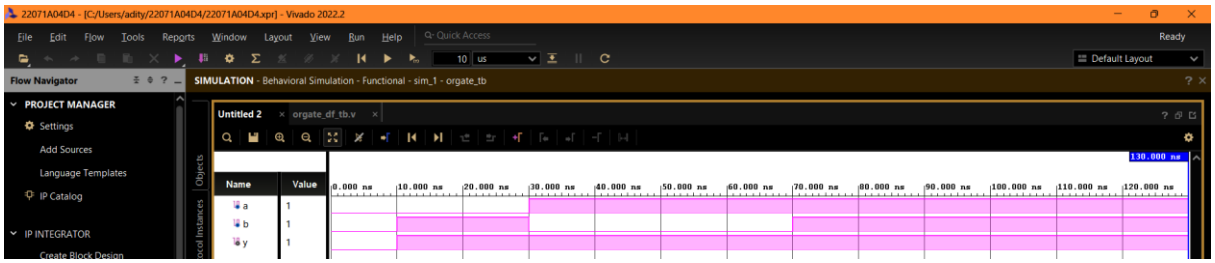
```
module orgate_df(y,a,b);
output y;
input a,b;

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.

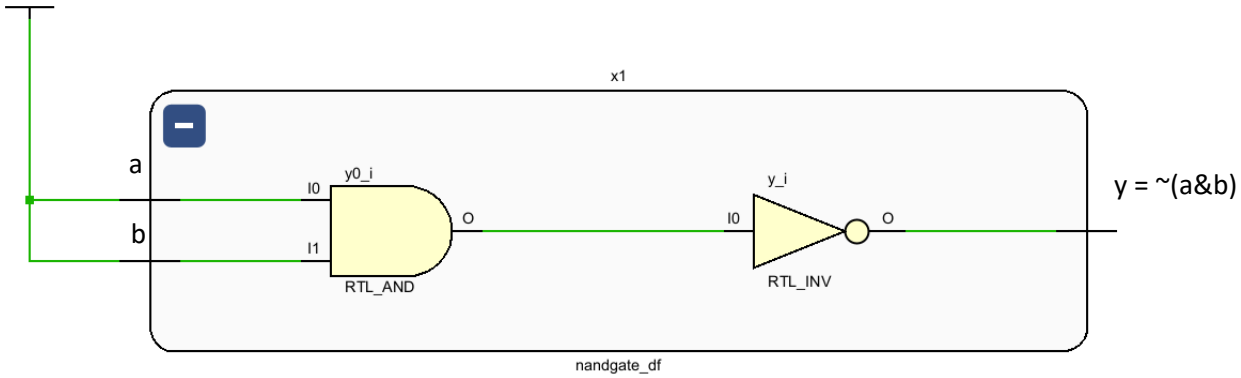
2.11 RESULT:

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**

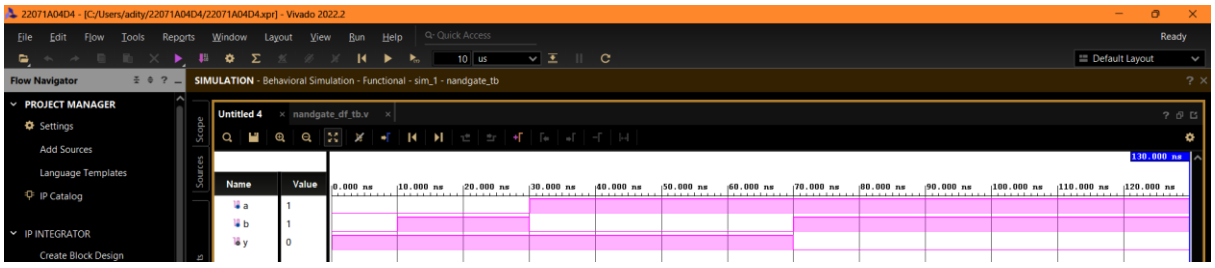
3.7 VERILOG CODE (nandgate_df.v):

```
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.

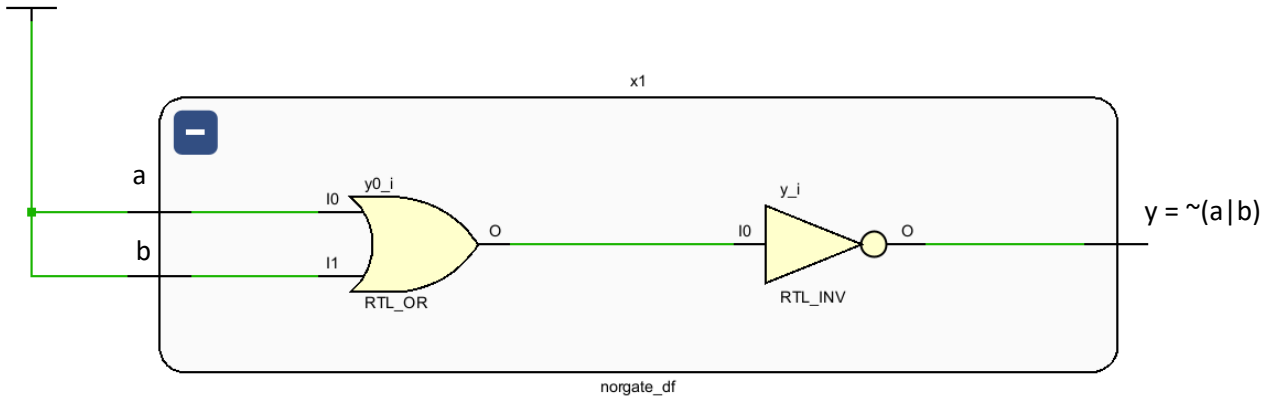
3.11 RESULT:

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**

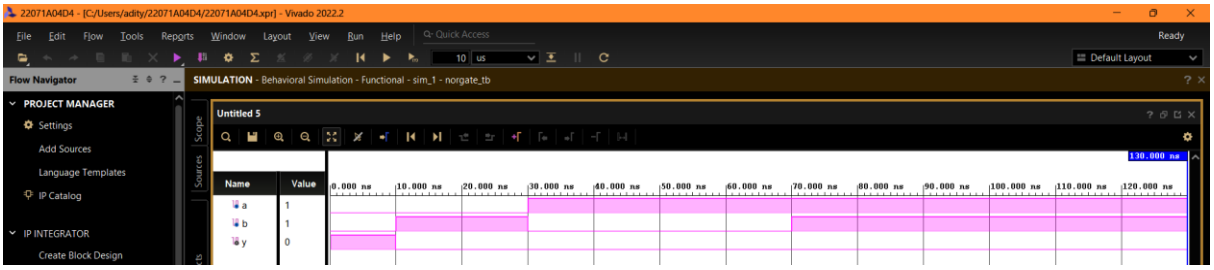
4.7 VERILOG CODE (norgate_df.v):

```
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.

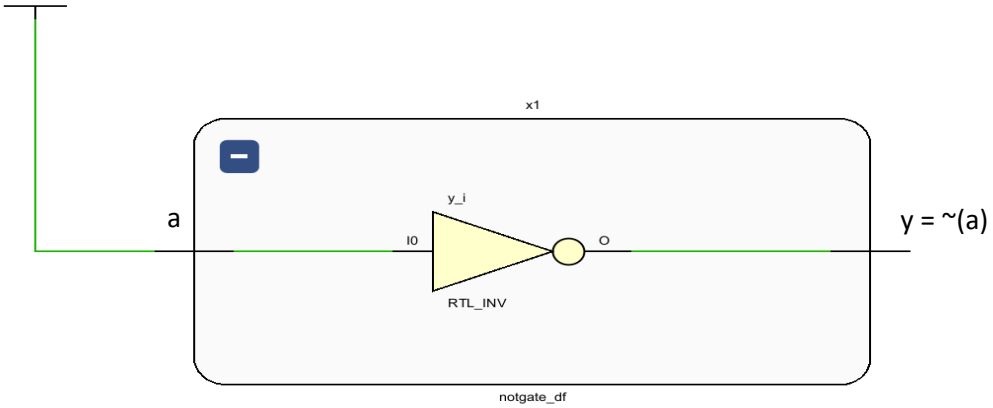
4.11 RESULT:

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**

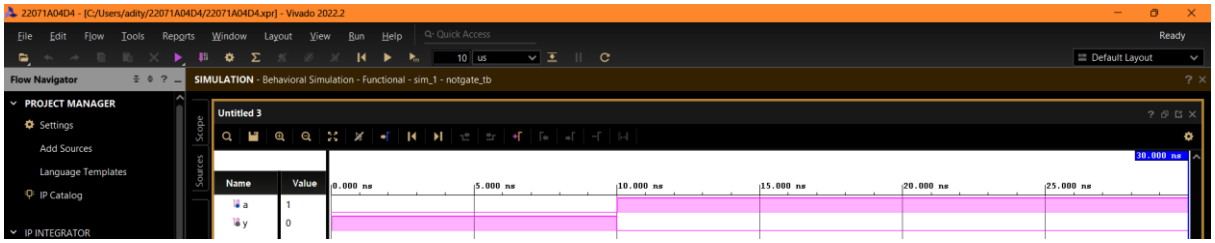
5.7 VERILOG CODE (notgate_df.v):

```
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.

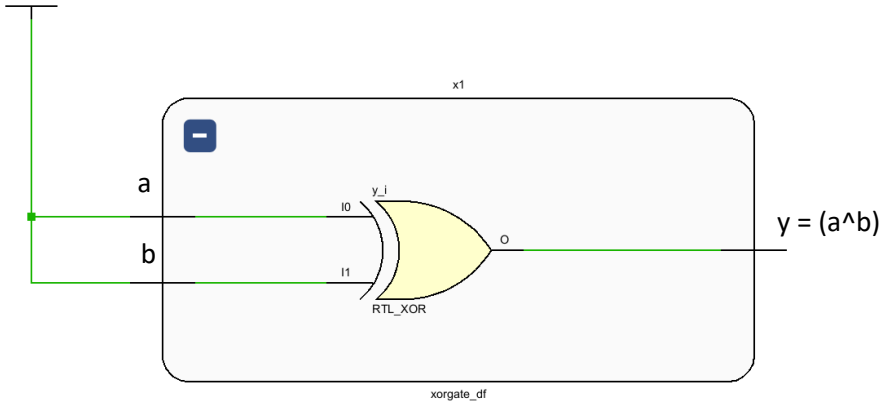
5.11 RESULT:

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**

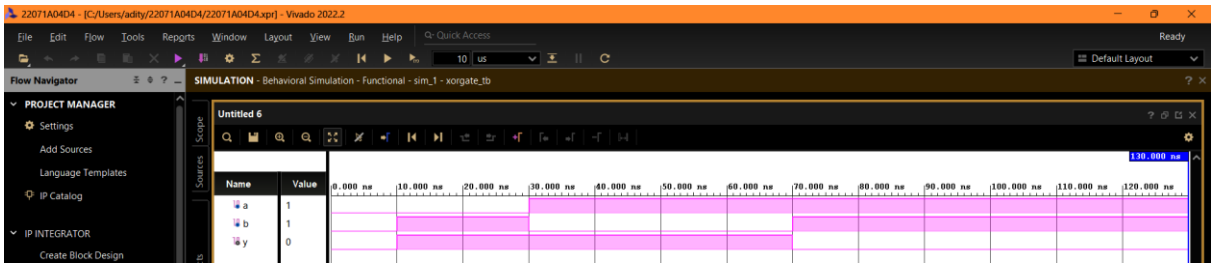
6.7 VERILOG CODE (xorgate_df.v):

```
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.

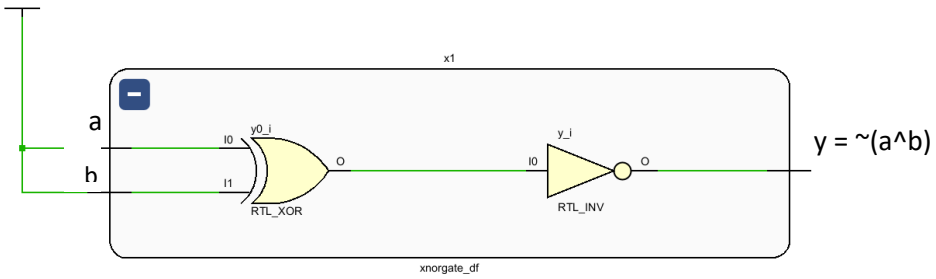
6.11 RESULT:

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**

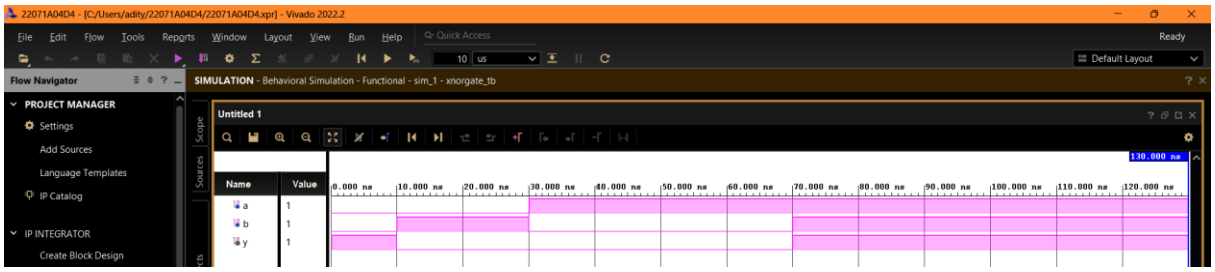
7.7 VERILOG CODE (xnorgate_df.v):

```
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:

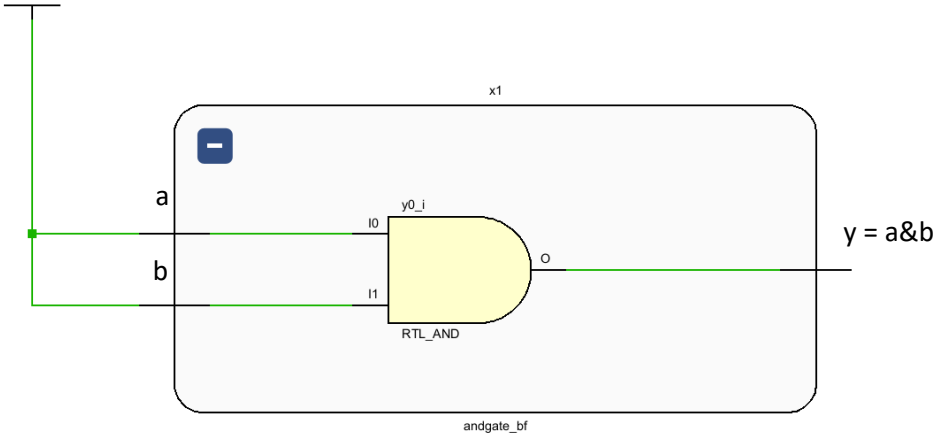
The Logical XNOR Gate is simulated and implemented in Data Flow Modeling.

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):

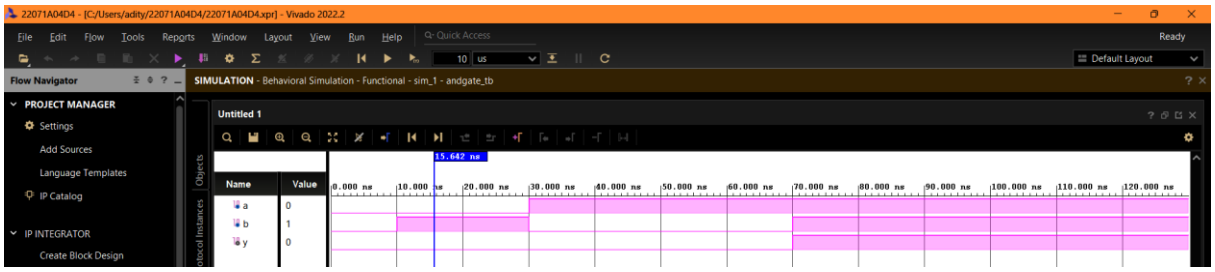
```
module andgate_bf(y,a,b);
output y;
input a,b;
reg y;
always@(a,b)
begin
if(a==1'b1 & b==1'b1)
```

```
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.

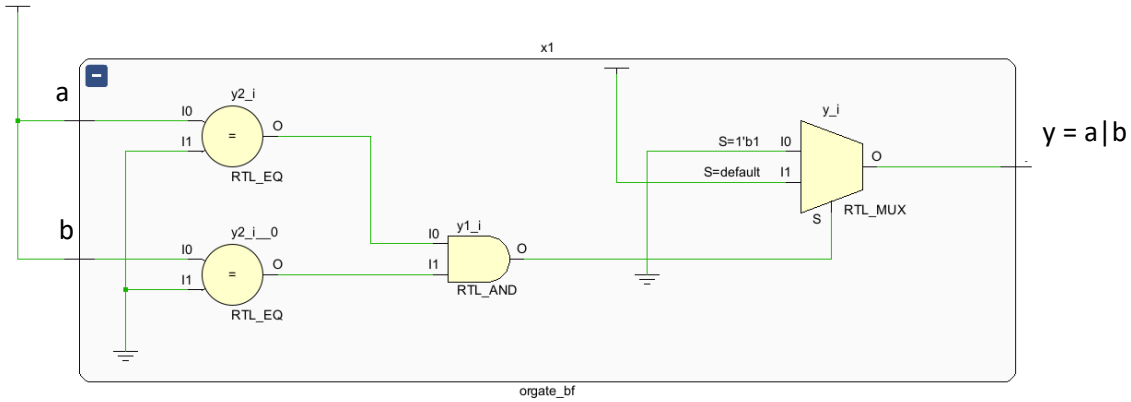
1.22 RESULT:

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**

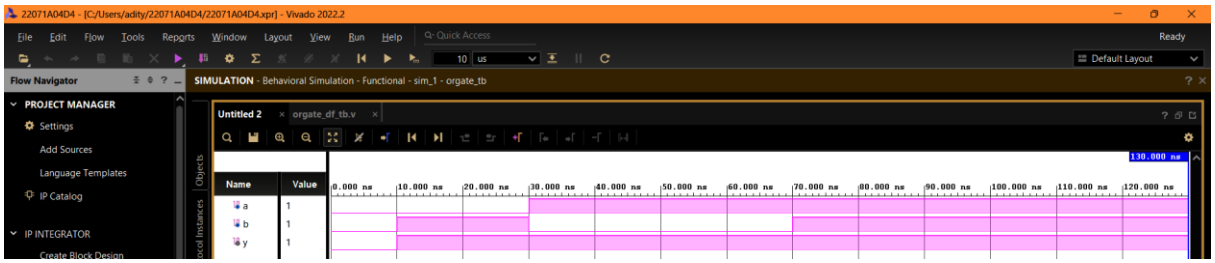
2.7 VERILOG CODE (orgate_bf.v):

```
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.

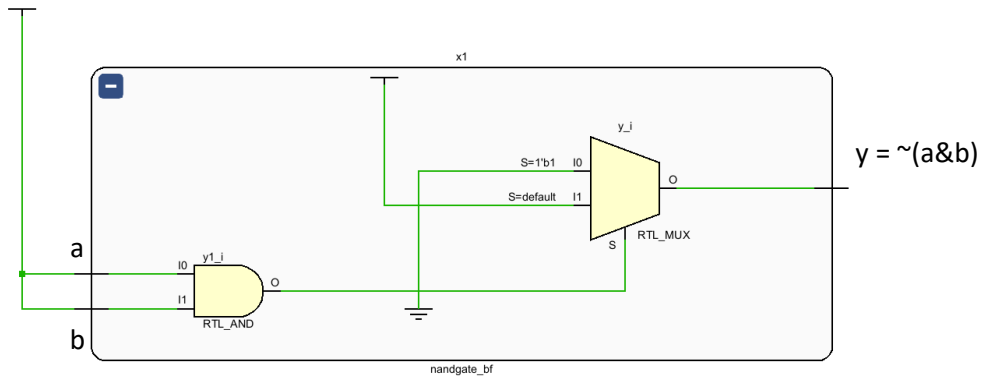
2.11 RESULT:

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):

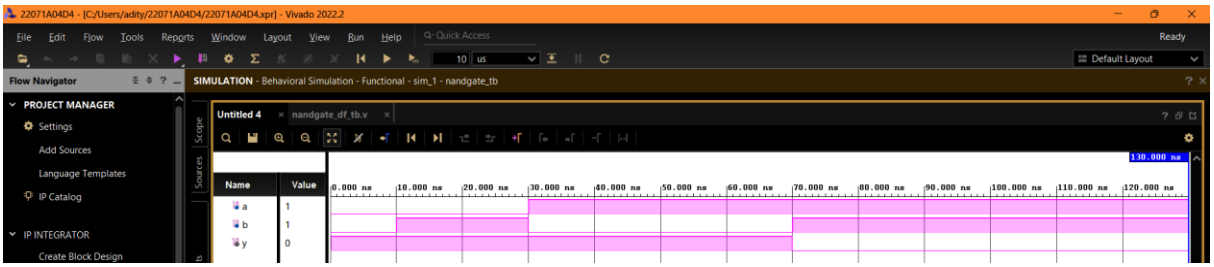
```
module nandgate_bf(y,a,b);
output y;
input a,b;
reg y;
always@(a,b)
begin
if(a==1'b1 & b==1'b1)
```

```
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.

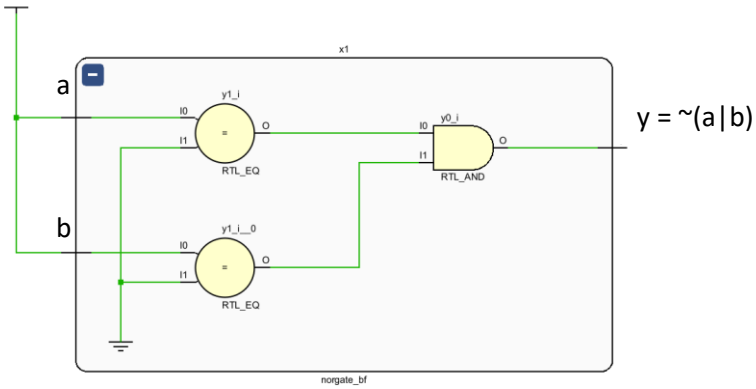
3.11 RESULT:

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):

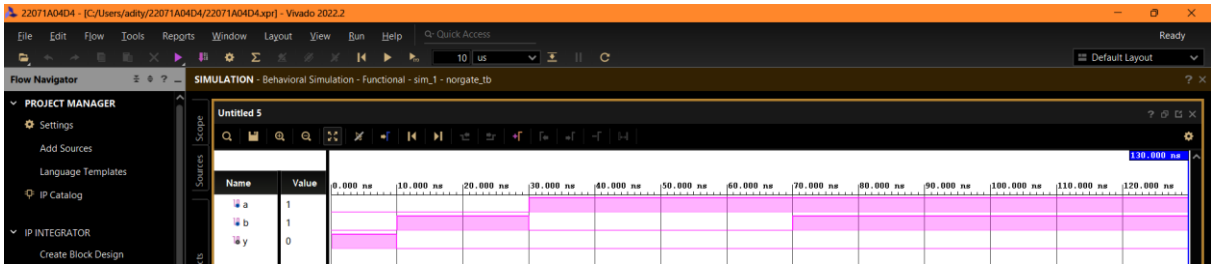
```
module norgate_bf(y,a,b);
output y;
input a,b;
reg y;
always@(a,b)
begin
if(a==1'b0 & b==1'b0)
y=1;
else
y=0;
end
```

```
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.

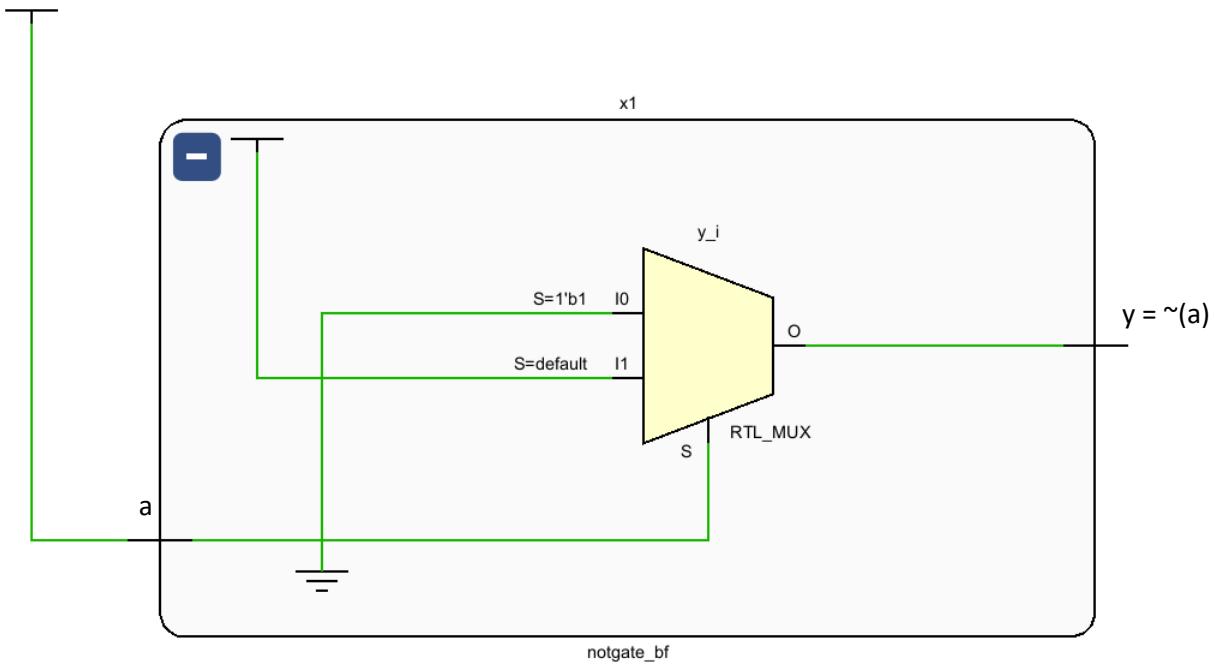
4.11 RESULT:

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):

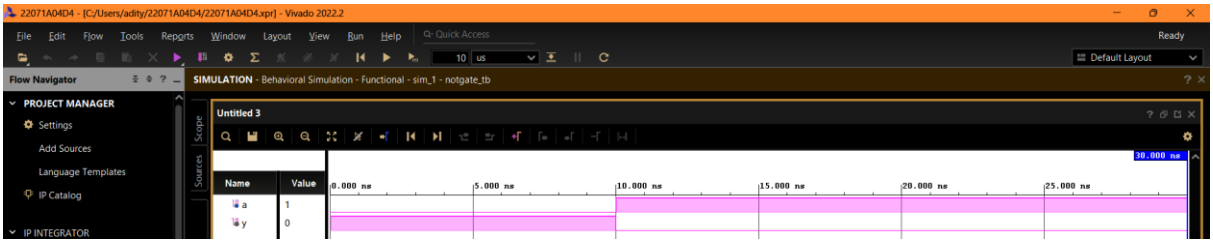
```
module notgate_bf(y,a);
output y;
input a;
reg y;
always@(a)
```

```
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.

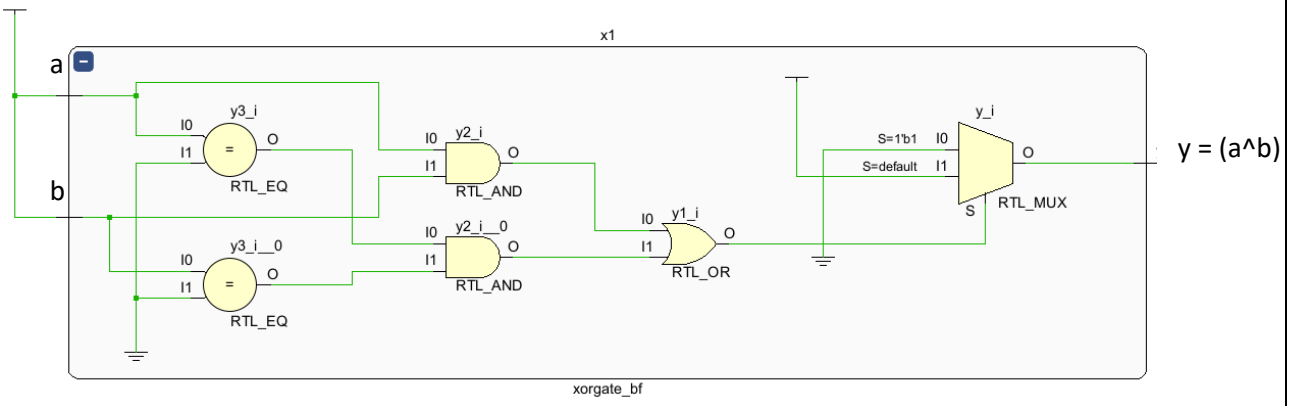
5.11 RESULT:

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**

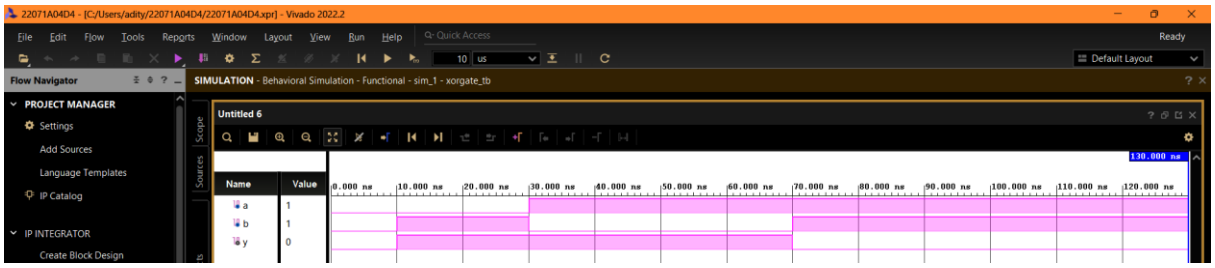
6.7 VERILOG CODE (xorgate_bf.v):

```
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.

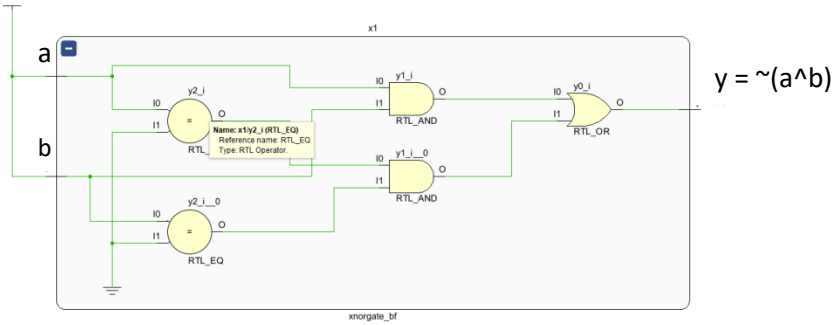
6.11 RESULT:

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:

$$Y=\sim(A\hat{}B)$$

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**

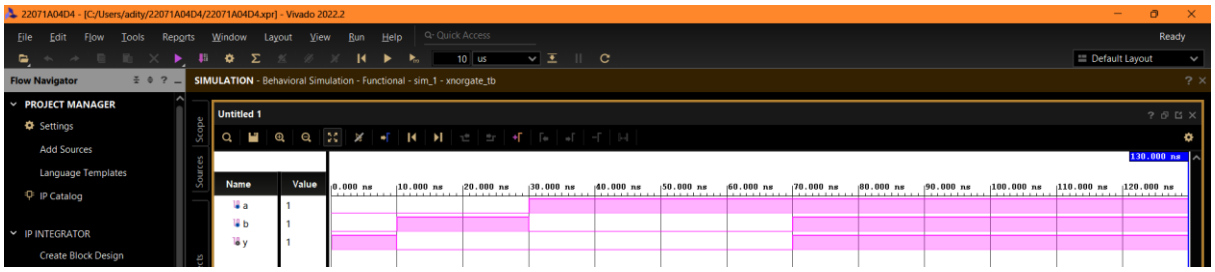
7.7 VERILOG CODE (xnorgate_bf.v):

```
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:

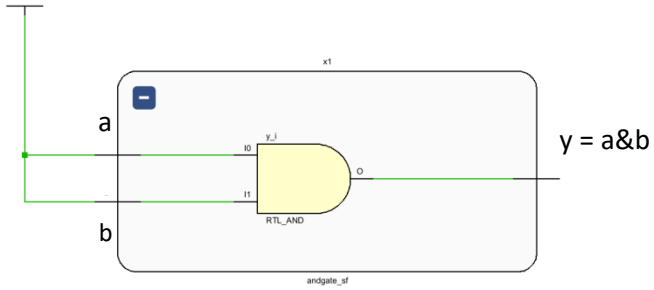
The Logical XNOR Gate is simulated and implemented in Behavioral Flow Modeling.

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**

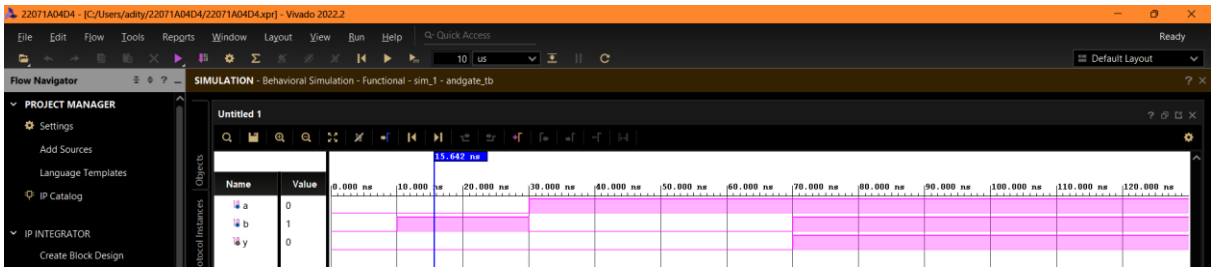
1.7 VERILOG CODE (andgate_sf.v):

```
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.

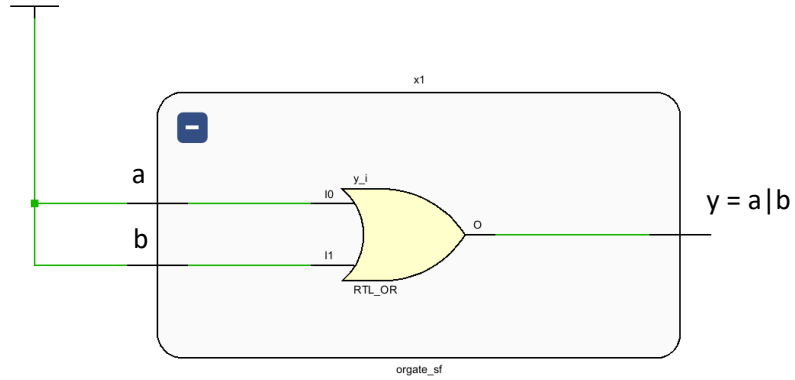
1.11 RESULT:

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):

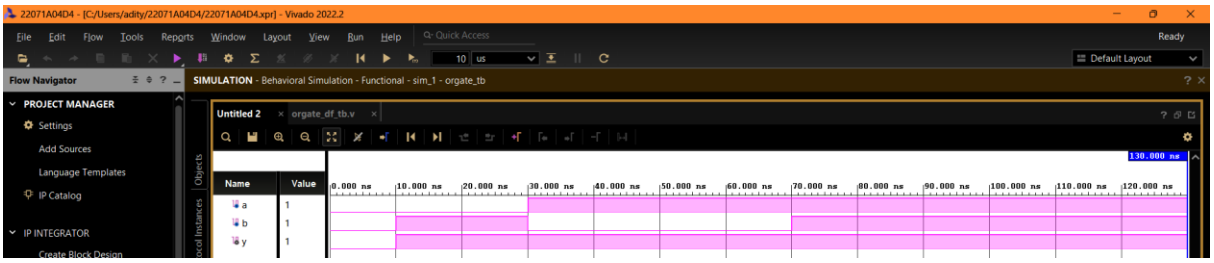
```
module orgate_sf(y,a,b);
output y;
input a,b;
wire y;
or (y,a,b);
endmodule
```

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.

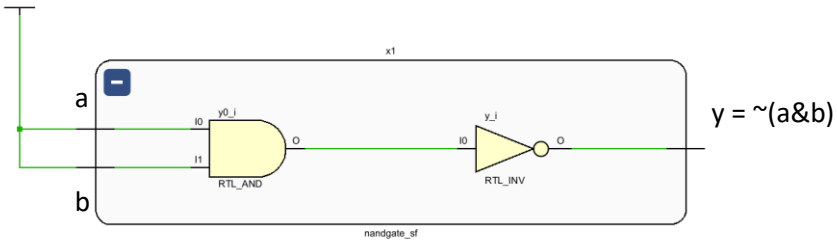
2.11 RESULT:

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):

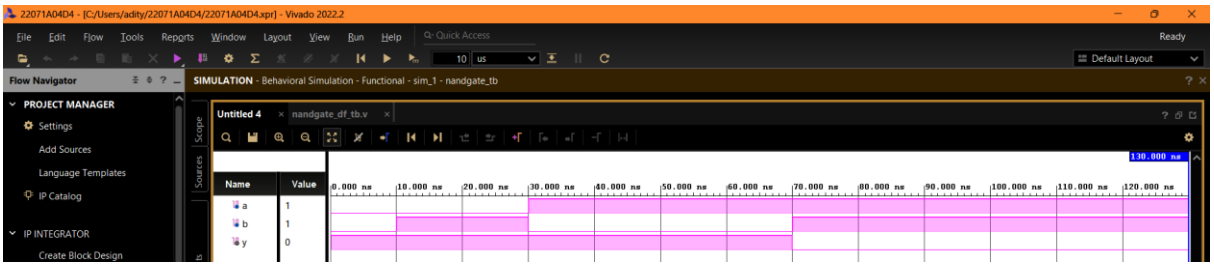
```
module nandgate_sf(y,a,b);
output y;
input a,b;
wire y;
nand (y,a,b);
endmodule
```

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.

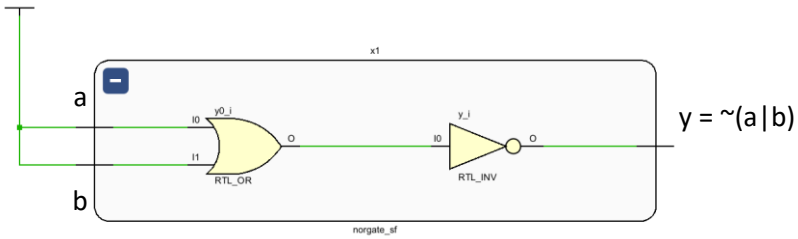
3.11 RESULT:

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):

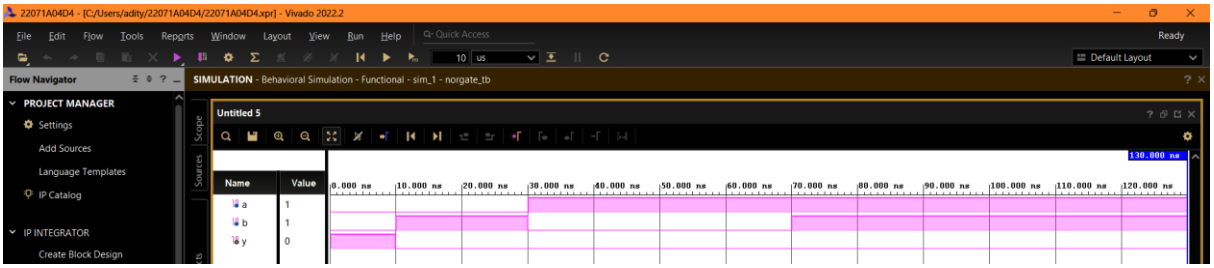
```
module norgate_sf(y,a,b);
output y;
input a,b;
wire y;
nor (y,a,b);
endmodule
```

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.

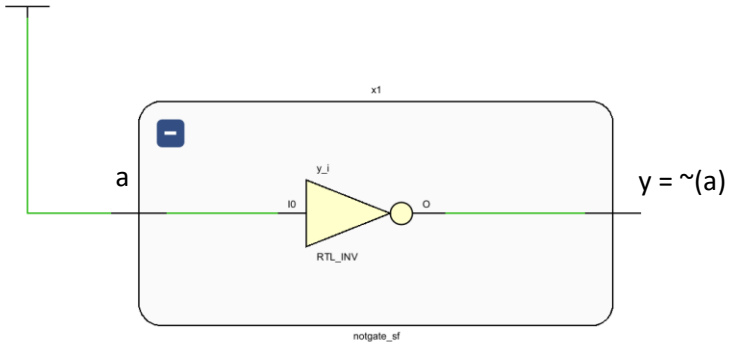
4.11 RESULT:

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):

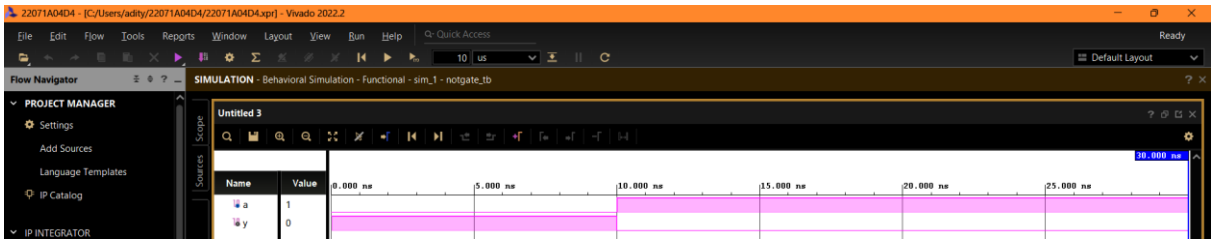
```
module notgate_sf(y,a);
output y;
input a;
wire y;
not (y,a);
endmodule
```

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.

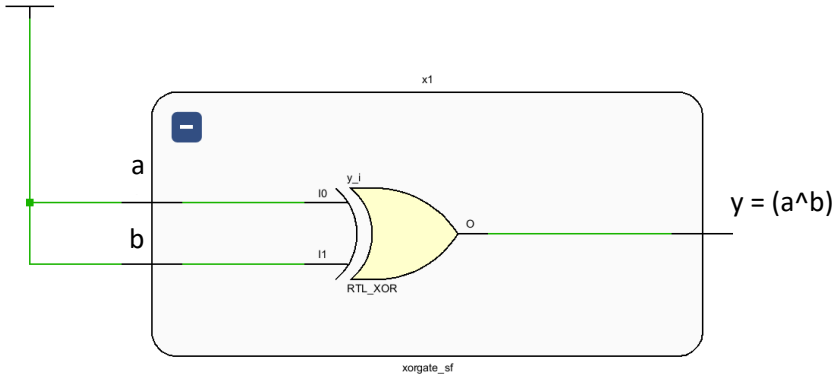
5.11 RESULT:

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):

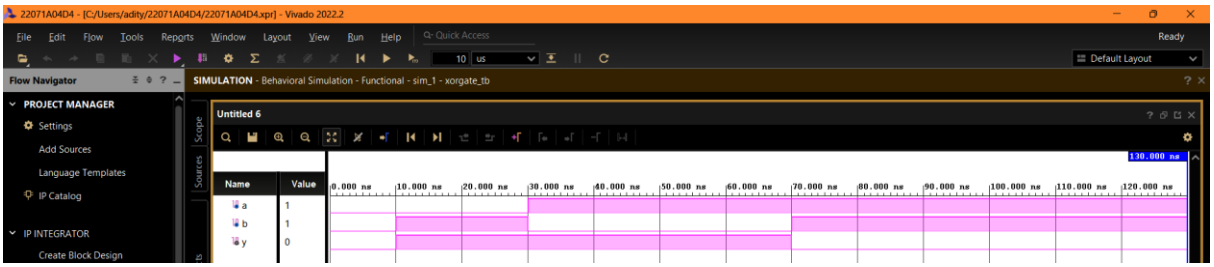
```
module xorgate_sf(y,a,b);
output y;
input a,b;
wire y;
xor (y,a,b);
endmodule
```

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.

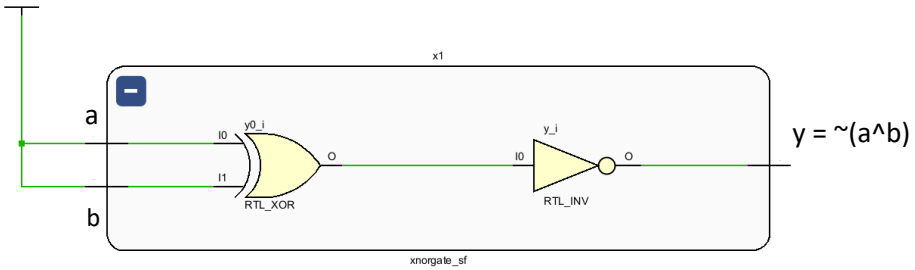
6.11 RESULT:

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):

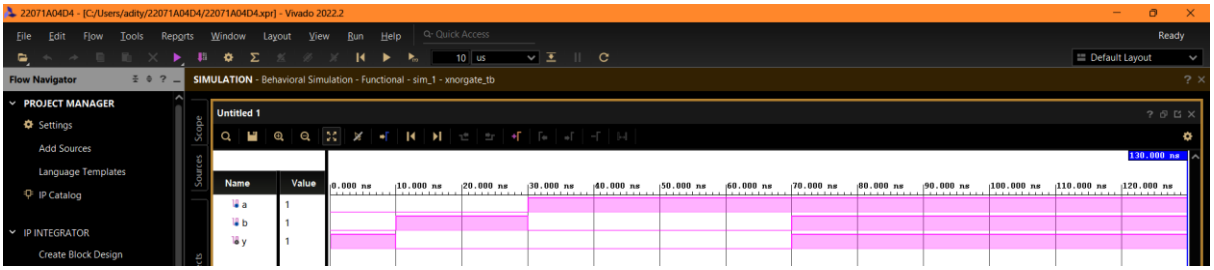
```
module xnorgate_sf(y,a,b);
output y;
input a,b;
wire y;
xnor (y,a,b);
endmodule
```

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.

2

EXPERIMENT-2
REALIZATION OF ADDERS AND SUBTRACTORS (HALF, FULL)

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

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

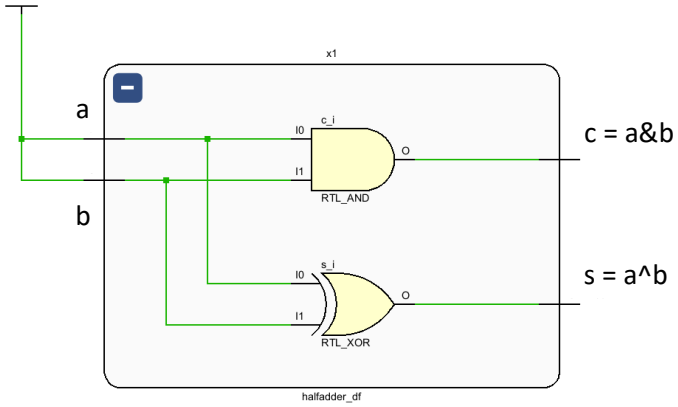
3. **PROCEDURE:**
    - Open the Xilinx Vivado project navigator.
    - Open the Design source and go to add / create sources
    - Create new file, give appropriate name save it.
    - Open the file in the editor and write the Verilog code.
    - Open the Design source and go to add / create sources 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.

ADDERS (Data Flow Modeling)

1.1 AIM: To implement Half Adder

1.2 SOFTWARE USED: Xilinx Vivado 2022.2

1.3 SYMBOL:



1.4 LOGIC EXPRESSION:

S = A^B

C = A&B

1.5 BOOLEAN EXPRESSION:

S = $\bar{A}B+A\bar{B}$

C = A.B

1.6 TRUTH TABLE:

INPUT		OUTPUT	
A	**B**	**S**	**C**
0	**0**	**0**	**0**
0	**1**	**1**	**0**
1	**0**	**1**	**0**
1	**1**	**0**	**1**

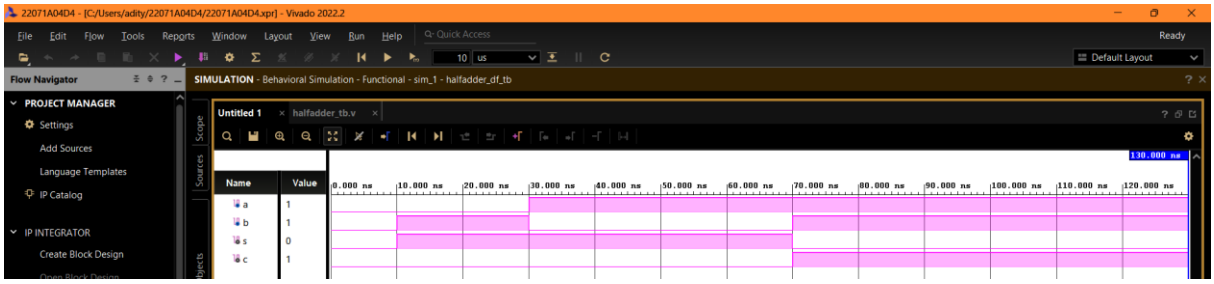
1.7 VERILOG CODE (halfadder_df.v):

```
module halfadder_df(s,c,a,b);
output s,c;
input a,b;
assign s = a^b;
assign c = a&b;
endmodule
```

1.8 TEST BENCH (halfadder_df_tb.v):

```
module halfadder_df_tb();
reg a,b;
wire s,c;
halfadder_df x1(s,c,a,b);
initial
begin
{a,b}=2'b00;
#10 {a,b}=2'b01;
#20 {a,b}=2'b10;
#40 {a,b}=2'b11;
#60 $finish;
end
endmodule
```

1.9 WAVEFORM:



1.10 OBSERVATION:

In half adder, the sum of the two inputs is represented using logical XOR Gate and carry is represented using logical AND Gate.

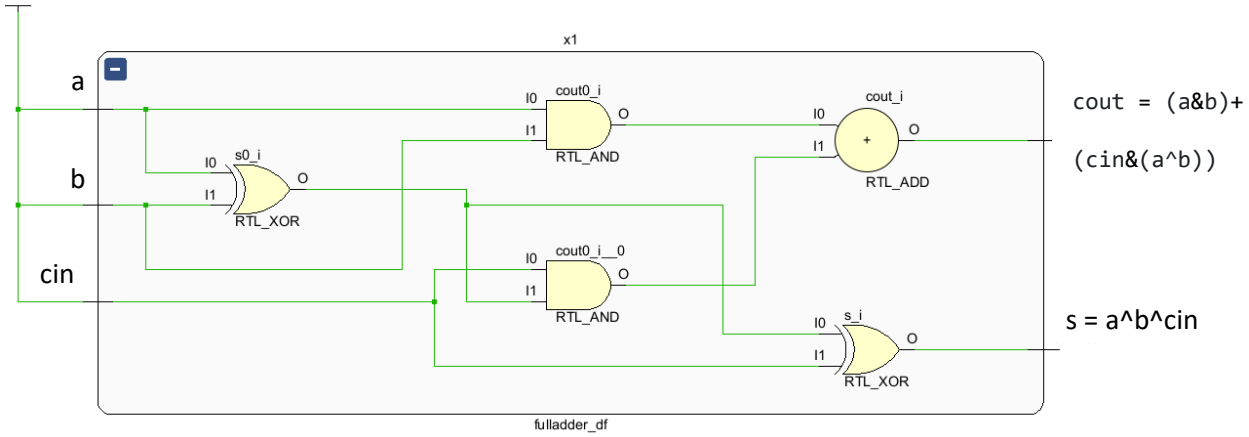
1.11 RESULT:

Half Adder is simulated and implemented in Data Flow Modeling.

2.1 AIM: To implement Full Adder

2.2 SOFTWARE USED: Xilinx Vivado 2022.2

2.3 SYMBOL:



2.4 LOGIC EXPRESSION:

S = A^B^Cin

C = (A&B) + (Cin & (A^B))

2.5 BOOLEAN EXPRESSION:

$S = \bar{A}(\bar{B}Cin + B\bar{Cin}) + A\overline{(\bar{B}Cin + B\bar{Cin})}$

$C = AB + Cin(\bar{A}B + A\bar{B})$

2.1 TRUTH TABLE:

INPUT			OUTPUT	
A	**B**	**Cin**	**S**	**C**
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**

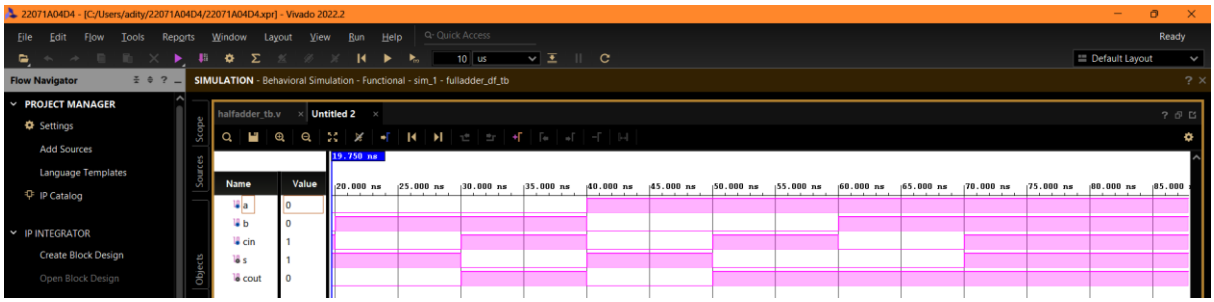
2.2 VERILOG CODE (fulladder_df.v):

```
module fulladder_df(s,cout,a,b,cin);
output s,cout;
input a,b,cin;
assign s = a^b^cin;
assign cout = (a&b)+(cin&(a^b));
endmodule
```

2.3 TEST BENCH (fulladder_df_tb.v):

```
module fulladder_df_tb();
reg a,b,cin;
wire s,cout;
fulladder_df x1(s,cout,a,b,cin);
initial
begin
   {a,b,cin}=3'b000;
#10 {a,b,cin}=3'b001;
#10 {a,b,cin}=3'b010;
#10 {a,b,cin}=3'b011;
#10 {a,b,cin}=3'b100;
#10 {a,b,cin}=3'b101;
#10 {a,b,cin}=3'b110;
#10 {a,b,cin}=3'b111;
end
endmodule
```

2.4 WAVEFORM:



2.5 OBSERVATION:

In full adder, the sum of the two inputs is represented using S = A^B^Cin and carry is represented using C = (A&B) + (Cin & (A^B))

2.6 RESULT:

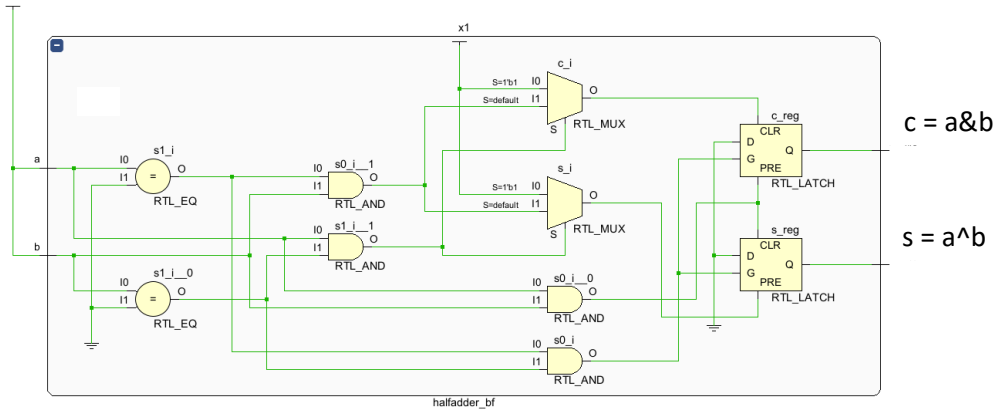
Full Adder is simulated and implemented in Data Flow Modeling.

ADDERS (Behavioral Flow Modeling)

1.1 AIM: To implement Half Adder

1.2 SOFTWARE USED: Xilinx Vivado 2022.2

1.3 SYMBOL:



1.4 LOGIC EXPRESSION:

S = A^B

C = A&B

1.5 BOOLEAN EXPRESSION:

S = $\bar{A}B+A\bar{B}$

C = A.B

1.6 TRUTH TABLE:

INPUT		OUTPUT	
A	**B**	**S**	**C**
0	**0**	**0**	**0**
0	**1**	**1**	**0**
1	**0**	**1**	**0**
1	**1**	**0**	**1**

1.7 VERILOG CODE (halfadder_bf.v):

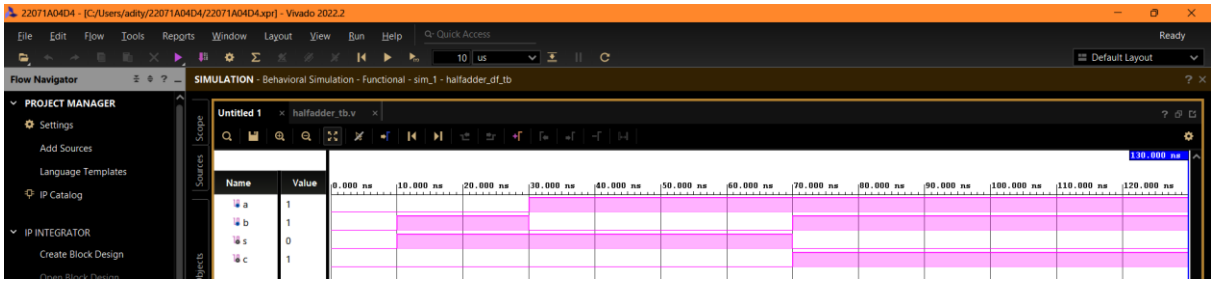
```
module halfadder_bf(s,c,a,b);
output s,c;
input a,b;
reg s,c;
always @(a,b)
begin
if(a==1'b0 & b==1'b0)
   begin
   s=0;
   c=0;
   end

else if(a==1'b0 & b==1'b1)
```

```
    begin
    s=1;
    c=0;
    end

else if(a==1'b1 & b==1'b0)
    begin
    s=1;
    c=0;
    end
else if(a==1'b1 & b==1'b1)
    begin
    s=0;
    c=1;
    end
end
endmodule
```

1.8 TEST BENCH (halfadder_bf_tb.v):

```
module halfadder_bf_tb();
reg a,b;
wire s,c;
halfadder_bf x1(s,c,a,b);
initial
begin
{a,b}=2'b00;
#10 {a,b}=2'b01;
#20 {a,b}=2'b10;
#40 {a,b}=2'b11;
#60 $finish;
end
endmodule
```

1.9 WAVEFORM:



1.10 OBSERVATION:

In half adder, the sum of the two inputs is represented using logical XOR Gate and carry is represented using logical AND Gate.

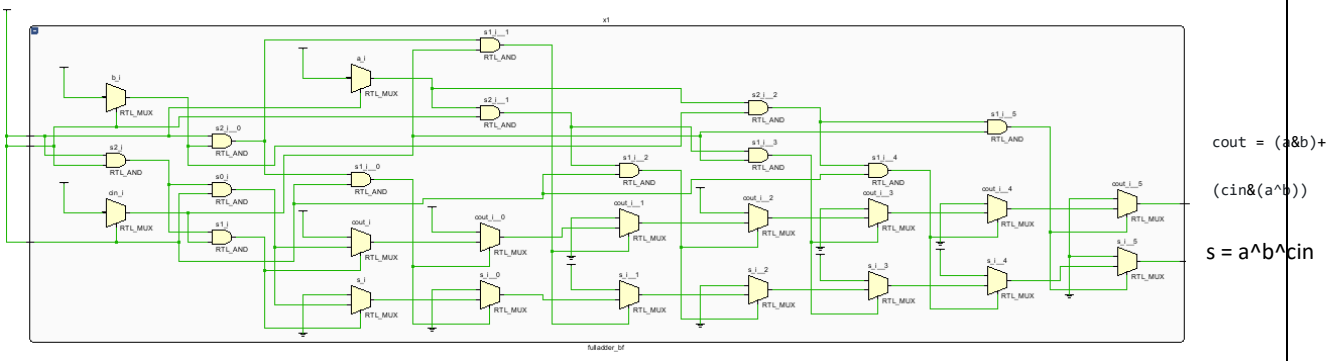
1.11 RESULT:

Half Adder is simulated and implemented in Behavioral Flow Modeling.

2.1 AIM: To implement Full Adder

2.2 SOFTWARE USED: Xilinx Vivado 2022.2

2.3 SYMBOL:



2.4 LOGIC EXPRESSION:

S = A^B^Cin

C = (A&B) + (Cin & (A^B))

2.5 BOOLEAN EXPRESSION:

$S = \bar{A}(B\bar{C}in + \bar{B}Cin) + A\overline{(\bar{B}Cin + B\bar{C}in)}$

$C = AB + Cin(\bar{A}B + A\bar{B})$

2.6 TRUTH TABLE:

INPUT			OUTPUT	
A	**B**	**Cin**	**S**	**C**
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**

2.7 VERILOG CODE (fulladder_bf.v):

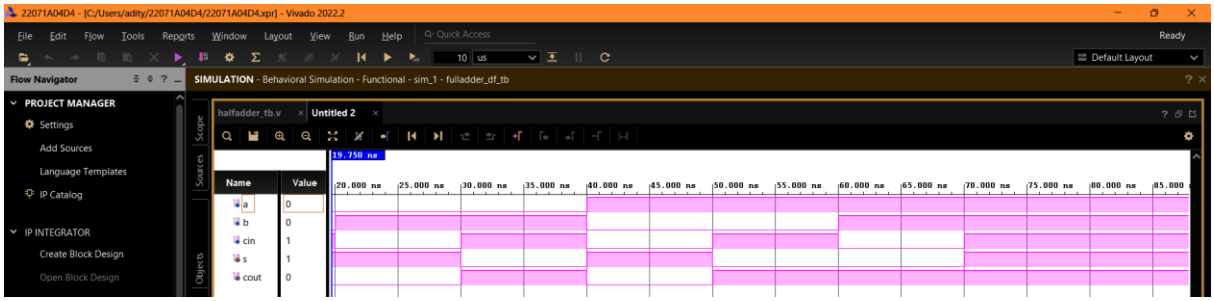
```
module fulladder_bf(s,cout,a,b,cin);
output s,cout;
input a,b,cin;
reg s,cout;
always@(a,b,cin)
begin
   if(a==1'b0 & b==1'b0 & cin==1'b0)
      begin
      s=0;
      cout = 0;
      end
```

```
    else if(a==1'b0 & b==1'b0 & cin==1'b1)
       begin
       s=1;
       cout = 0;
       end
    else if(a==1'b0 & b==1'b1 & cin==1'b0)
       begin
       s=1;
       cout = 0;
       end
    else if(a==1'b0 & b==1'b1 & cin==1'b1)
       begin
       s=0;
       cout = 1;
       end
    else if(a==1'b1 & b==1'b0 & cin==1'b0)
       begin
       s=1;
       cout = 0;
       end
    else if(a==1'b1 & b==1'b0 & cin==1'b1)
       begin
       s=0;
       cout = 1;
       end
    else if(a==1'b1 & b==1'b1 & cin==1'b0)
       begin
       s=0;
       cout = 1;
       end
    else if(a==1'b1 & b==1'b1 & cin==1'b1)
       begin
       s=1;
       cout = 1;
       end
    else
       begin
       s=0;
       cout = 0;
       end
end
endmodule
```

2.8 TEST BENCH (fulladder_bf_tb.v):

```
module fulladder_df_tb();
reg a,b,cin;
wire s,cout;
fulladder_df x1(s,cout,a,b,cin);
initial
begin
{a,b,cin}=3'b000;
#10 {a,b,cin}=3'b001;
#10 {a,b,cin}=3'b010;
#10 {a,b,cin}=3'b011;
#10 {a,b,cin}=3'b100;
#10 {a,b,cin}=3'b101;
#10 {a,b,cin}=3'b110;
#10 {a,b,cin}=3'b111;
end
endmodule
```

2.9 WAVEFORM:



2.10 OBSERVATION:

In full adder, the sum of the two inputs is represented using S = A^B^Cin and carry is represented using C = (A&B) + (Cin & (A^B))

2.11 RESULT:

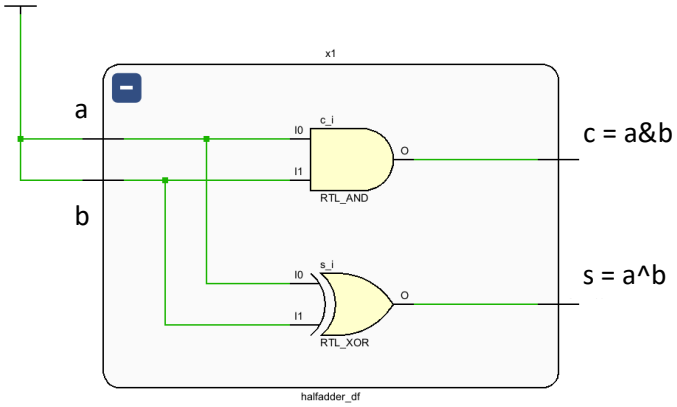
Full Adder is simulated and implemented in Behavioral Flow Modeling.

ADDERS (Structural Flow Modeling)

1.1 AIM: To implement Half Adder

1.2 SOFTWARE USED: Xilinx Vivado 2022.2

1.3 SYMBOL:



1.4 LOGIC EXPRESSION:

S = A^B

C = A&B

1.5 BOOLEAN EXPRESSION:

S = $\bar{A}B+A\bar{B}$

C = A.B

1.6 TRUTH TABLE:

INPUT		OUTPUT	
A	**B**	**S**	**C**
0	**0**	**0**	**0**
0	**1**	**1**	**0**
1	**0**	**1**	**0**
1	**1**	**0**	**1**

1.7 VERILOG CODE (halfadder_sf.v):

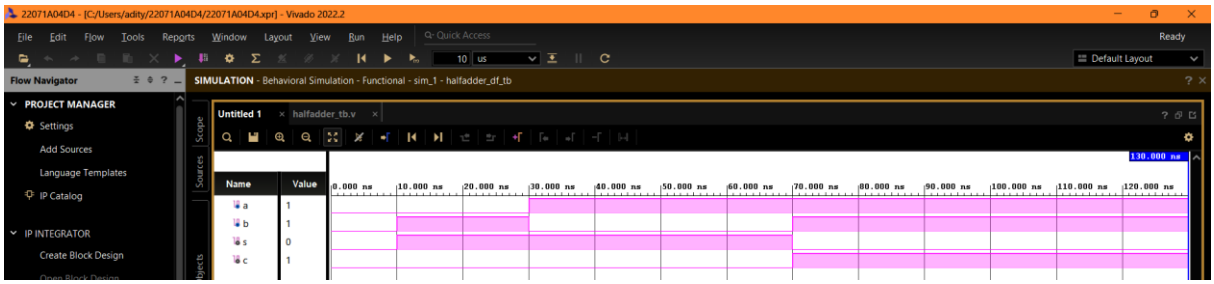
```
module halfadder_sf(s,c,a,b);
output s,c;
input a,b;
wire s,c;
xor (s,a,b);
and (c,a,b);
endmodule
```

1.8 TEST BENCH (halfadder_sf_tb.v):

```
module halfadder_sf_tb();
reg a,b;
wire s,c;
halfadder_sf x1(s,c,a,b);
```

```
initial
begin
{a,b}=2'b00;
#10 {a,b}=2'b01;
#20 {a,b}=2'b10;
#40 {a,b}=2'b11;
#60 $finish;
end
endmodule
```

1.9 WAVEFORM:



1.10 OBSERVATION:

In half adder, the sum of the two inputs is represented using logical XOR Gate and carry is represented using logical AND Gate.

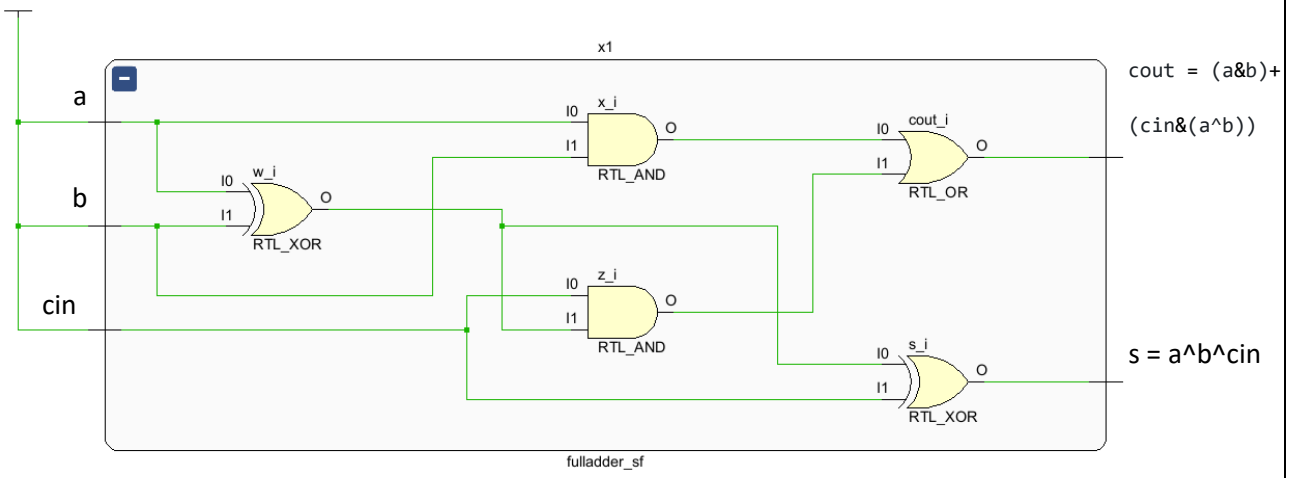
1.11 RESULT:

Half Adder is simulated and implemented in Structural Flow Modeling.

2.1 AIM: To implement Full Adder

2.2 SOFTWARE USED: Xilinx Vivado 2022.2

2.3 SYMBOL:



2.4 LOGIC EXPRESSION:

S = A^B^Cin

C = (A&B) + (Cin & (A^B))

2.5 BOOLEAN EXPRESSION:

$S = \bar{A}(\bar{B}Cin + B\overline{Cin}) + A\overline{(\bar{B}Cin + B\overline{Cin})}$

$C = AB + Cin(\bar{A}B + A\bar{B})$

2.6 TRUTH TABLE:

INPUT			OUTPUT	
A	**B**	**Cin**	**S**	**C**
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**

2.7 VERILOG CODE (fulladder_sf.v):

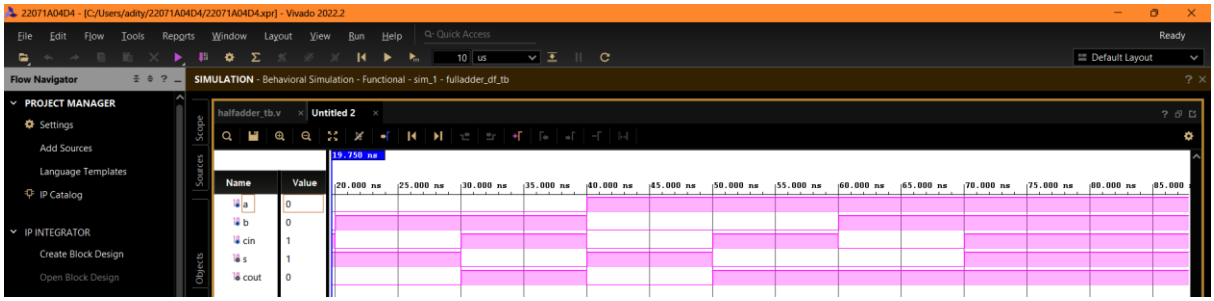
```
module fulladder_sf(s,cout,a,b,cin);
output s,cout;
input a,b,cin;
wire  w,x,y,z;
xor (w,a,b);
xor(s,w,cin);
```

```
and (x,a,b);
xor (y,a,b);
and (z,cin,y);
or (cout,x,z);
endmodule
```

2.8 TEST BENCH (fulladder_sf_tb.v):

```
module fulladder_sf_tb();
reg a,b,cin;
wire s,cout;
fulladder_sf x1(s,cout,a,b,cin);
initial
begin
   {a,b,cin}=3'b000;
#10 {a,b,cin}=3'b001;
#10 {a,b,cin}=3'b010;
#10 {a,b,cin}=3'b011;
#10 {a,b,cin}=3'b100;
#10 {a,b,cin}=3'b101;
#10 {a,b,cin}=3'b110;
#10 {a,b,cin}=3'b111;
end
endmodule
```

2.9 WAVEFORM:



2.10 OBSERVATION:

In full adder, the sum of the two inputs is represented using S = A^B^Cin and carry is represented using C = (A&B) + (Cin & (A^B))

2.11 RESULT:

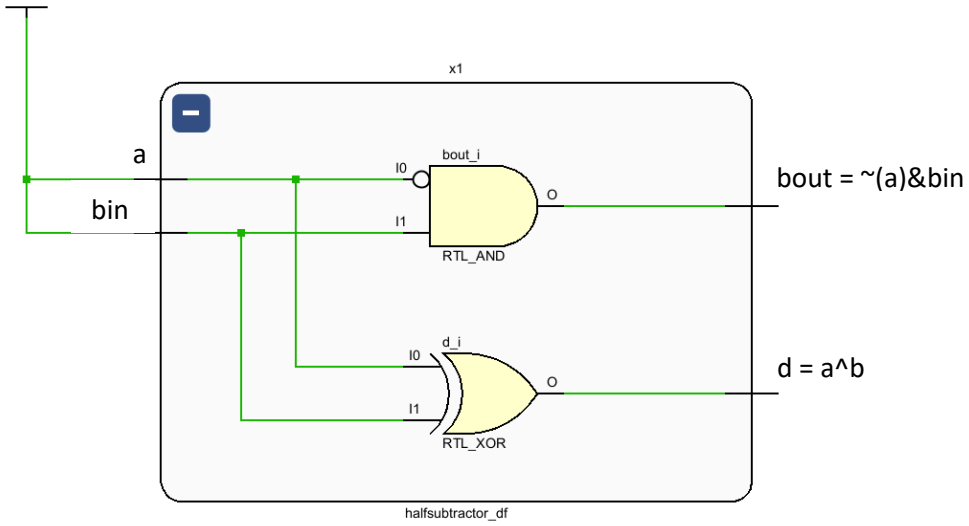
Full Adder is simulated and implemented in Structural Flow Modeling.

SUBTRACTORS (Data Flow Modeling)

1.1 AIM: To implement Half Subtractor

1.2 SOFTWARE USED: Xilinx Vivado 2022.2

1.3 SYMBOL:



1.4 LOGIC EXPRESSION:

D = A^Bin

Bout = ~(A)&Bin

1.5 BOOLEAN EXPRESSION:

$S = \bar{A}Bin + A\overline{Bin}$

$Bout = \bar{A}.B$

1.6 TRUTH TABLE:

INPUT		OUTPUT	
A	**Bin**	**D**	**Bout**
0	**0**	**0**	**0**
0	**1**	**1**	**1**
1	**0**	**1**	**0**
1	**1**	**0**	**0**

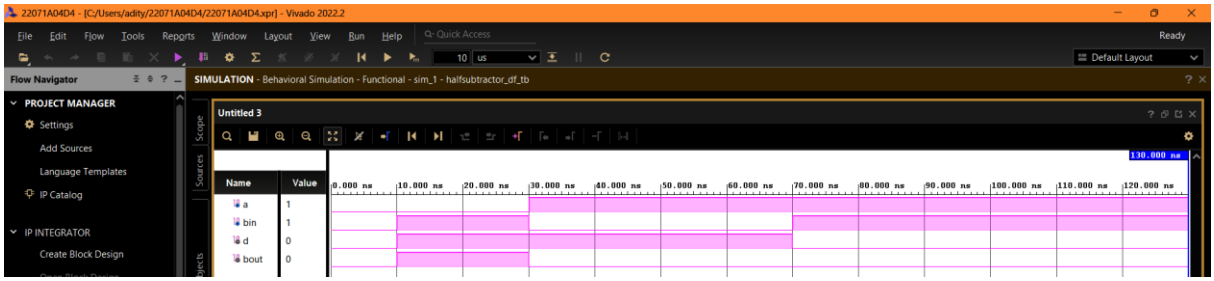
1.7 VERILOG CODE (halfsubtractor_df.v):

```
module halfsubtractor_df(d,bout,a,bin);
output d,bout;
input a,bin;
assign d = a^bin;
assign bout = ~(a)&bin;
endmodule
```

1.8 TEST BENCH (halfsubtractor_df_tb.v):

```
module halfsubtractor_df_tb();
reg a,bin;
wire d,bout;
halfsubtractor_df x1(d,bout,a,bin);
initial
begin
{a,bin}=2'b00;
#10 {a,bin}=2'b01;
#20 {a,bin}=2'b10;
#40 {a,bin}=2'b11;
#60 $finish;
end
endmodule
```

1.9 WAVEFORM:



1.10 OBSERVATION:

In half subtractor, the difference of the two inputs is represented using logical XOR of A and Bin and borrow is represented using logical AND of logical NOT of A and Bin.

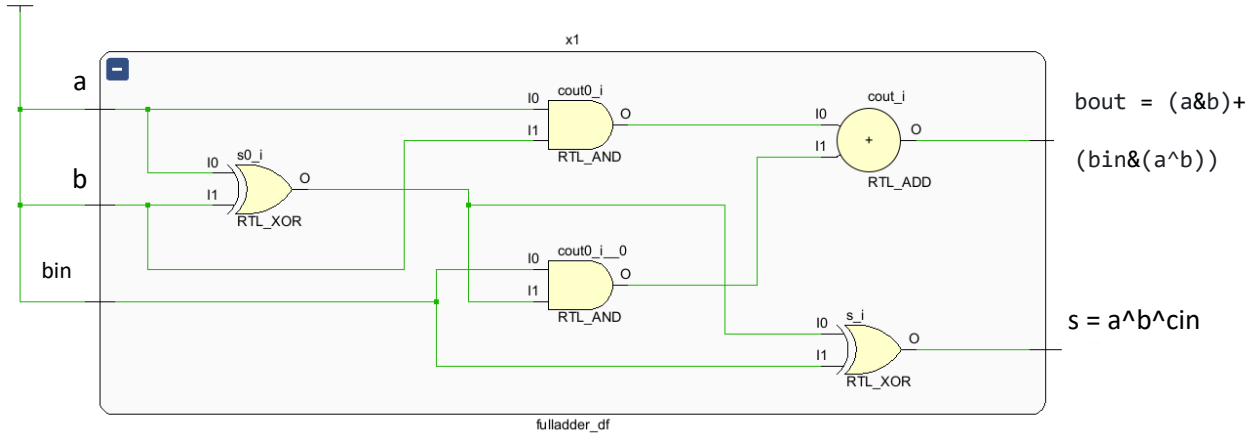
1.11 RESULT:

Half Subtractor is simulated and implemented in Data Flow Modeling.

2.1 AIM: To implement Full Subtractor

2.2 SOFTWARE USED: Xilinx Vivado 2022.2

2.3 SYMBOL:



2.4 LOGIC EXPRESSION:

D = A^B^Bin

Bout = (~(A)&B)+(Bin&(~(A^B)))

2.5 BOOLEAN EXPRESSION:

$D = \bar{A}(B\bar{C}in + \bar{B}Cin) + A\overline{(BCin + B\bar{C}in)}$

$Bout = \bar{A}B + Bin(AB + \bar{A}\bar{B})$

2.6 TRUTH TABLE:

INPUT			OUTPUT	
A	**B**	**Bin**	**D**	**Bout**
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**

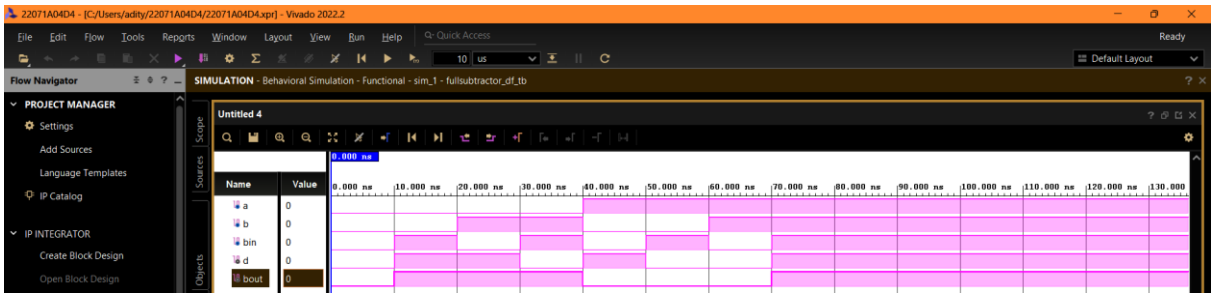
2.7 VERILOG CODE (fullsubtractor_df.v):

```
module fullsubtractor_df(d,bout,a,b,bin);
output d,bout;
input a,b,bin;
assign d = a^b^bin;
assign bout = (~(a)&b)+(bin&(~(a^b)));
endmodule
```

2.8 TEST BENCH (fullsubtractor_df_tb.v):

```
module fullsubtractor_df_tb();
reg a,b,bin;
wire d,bout;
fullsubtractor_df x1(d,bout,a,b,bin);
initial
begin
   {a,b,bin}=3'b000;
#10 {a,b,bin}=3'b001;
#10 {a,b,bin}=3'b010;
#10 {a,b,bin}=3'b011;
#10 {a,b,bin}=3'b100;
#10 {a,b,bin}=3'b101;
#10 {a,b,bin}=3'b110;
#10 {a,b,bin}=3'b111;
end
endmodule
```

2.9 WAVEFORM:



2.10 OBSERVATION:

In full subtractor, the sum of the two inputs is represented using D = A^B^Cin and carry is represented using Bout = (A&B) + (Bin & (A^B))

2.11 RESULT:

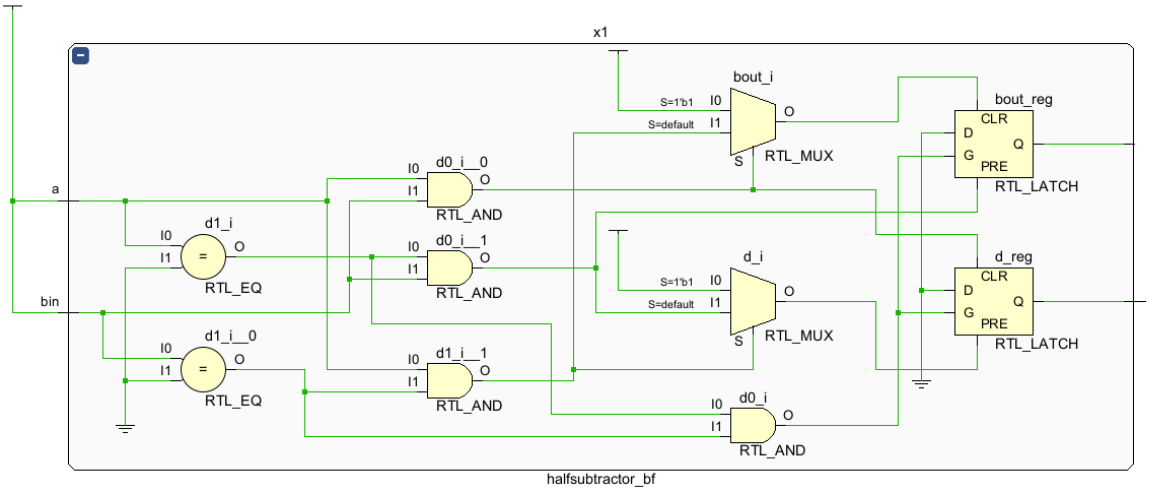
Full Subtractor is simulated and implemented in Data Flow Modeling.

SUBTRACTORS (Behavioral Flow Modeling)

1.1 AIM: To implement Half Subtractor

1.2 SOFTWARE USED: Xilinx Vivado 2022.2

1.3 SYMBOL:



bout = ~(a)

&bin

d = a^b

1.4 LOGIC EXPRESSION:

D = A^Bin

Bout = ~(A)&Bin

1.5 BOOLEAN EXPRESSION:

S = $\bar{A}$Bin+A$\bar{Bin}$

Bout = $\bar{A}$.B

1.6 TRUTH TABLE:

INPUT		OUTPUT	
A	**Bin**	**D**	**Bout**
0	**0**	**0**	**0**
0	**1**	**1**	**1**
1	**0**	**1**	**0**
1	**1**	**0**	**0**

1.7 VERILOG CODE (halfsubtractor_bf.v):

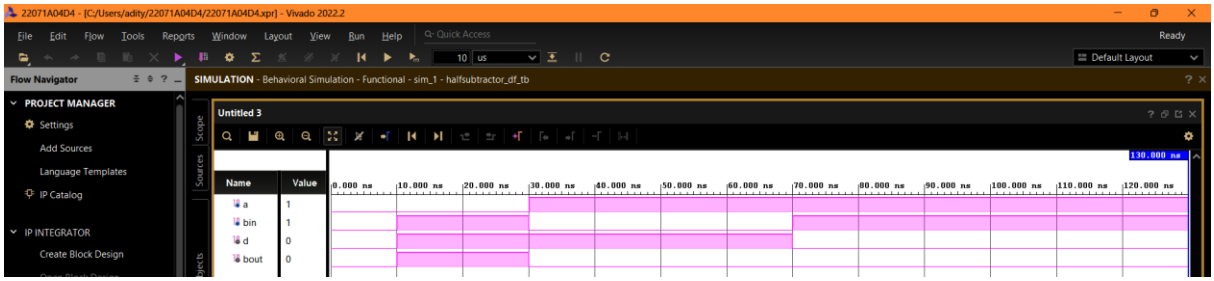
```
module halfsubtractor_bf(d,bout,a,bin);
output d,bout;
input a,bin;
reg d,bout;
always @(a,bin)
begin
if(a==1'b0 & bin==1'b0)
begin
d=0;
```

```
bout=0;
end
else if(a==1'b0 & bin==1'b1)
begin
d=1;
bout=1;
end
else if(a==1'b1 & bin==1'b0)
begin
d=1;
bout=0;
end
else if(a==1'b1 & bin==1'b1)
begin
d=0;
bout=0;
end
end
endmodule
```

1.8 TEST BENCH (halfsubtractor_df_tb.v):

```
module halfsubtractor_bf_tb();
reg a,bin;
wire d,bout;
halfsubtractor_bf x1(d,bout,a,bin);
initial
begin
{a,bin}=2'b00;
#10 {a,bin}=2'b01;
#20 {a,bin}=2'b10;
#40 {a,bin}=2'b11;
#60 $finish;
end
endmodule
```

1.9 WAVEFORM:



1.10 OBSERVATION:

In half subtractor, the difference of the two inputs is represented using logical XOR of A and Bin and borrow is represented using logical AND of logical NOT of A and Bin.

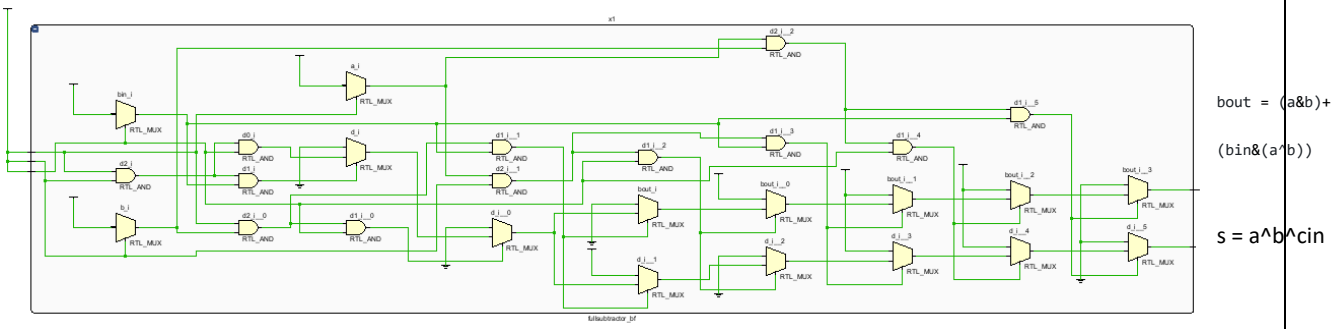
1.11 RESULT:

Half Subtractor is simulated and implemented in Behavioral Flow Modeling.

2.1 AIM: To implement Full Subtractor

2.2 SOFTWARE USED: Xilinx Vivado 2022.2

2.3 SYMBOL:



2.4 LOGIC EXPRESSION:

D = A^B^Bin

Bout = (~(A)&B)+(Bin&(~(A^B)))

2.5 BOOLEAN EXPRESSION:

$D = \bar{A}(B\bar{C}in + \bar{B}Cin) + A(\overline{BCin + B\bar{C}in})$

$Bout = \bar{A}B + Bin(AB + \overline{\bar{A}\bar{B}})$

2.6 TRUTH TABLE:

INPUT			OUTPUT	
A	**B**	**Bin**	**D**	**Bout**
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**

2.7 VERILOG CODE (fullsubtractor_bf.v):

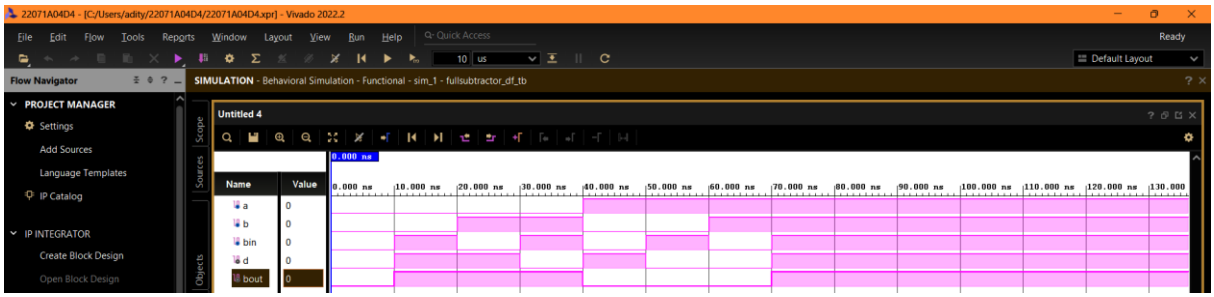
```
module fullsubtractor_bf(d,bout,a,b,bin);
output d,bout;
input a,b,bin;
reg d,bout;
always@(a,b,bin)
begin
if(a==1'b0 & b==1'b0 & bin==1'b0)
begin
d=0;
bout = 0;
end
else if(a==1'b0 & b==1'b0 & bin==1'b1)
begin
```

```
d=1;
bout = 1;
end
else if(a==1'b0 & b==1'b1 & bin==1'b0)
begin
d=1;
bout = 1;
end
else if(a==1'b0 & b==1'b1 & bin==1'b1)
begin
d=0;
bout = 1;
end
else if(a==1'b1 & b==1'b0 & bin==1'b0)
begin
d=1;
bout = 0;
end
else if(a==1'b1 & b==1'b0 & bin==1'b1)
begin
d=0;
bout = 0;
end
else if(a==1'b1 & b==1'b1 & bin==1'b0)
begin
d=0;
bout = 0;
end
else if(a==1'b1 & b==1'b1 & bin==1'b1)
begin
d=1;
bout = 1;
end
else
begin
d=0;
bout = 0;
end
end
endmodule
```

2.8 TEST BENCH (fullsubtractor_bf_tb.v):

```
module fullsubtractor_bf_tb();
reg a,b,bin;
wire d,bout;
fullsubtractor_bf x1(d,bout,a,b,bin);
initial
begin
   {a,b,bin}=3'b000;
#10 {a,b,bin}=3'b001;
#10 {a,b,bin}=3'b010;
#10 {a,b,bin}=3'b011;
#10 {a,b,bin}=3'b100;
#10 {a,b,bin}=3'b101;
#10 {a,b,bin}=3'b110;
#10 {a,b,bin}=3'b111;
end
endmodule
```

2.9 WAVEFORM:



2.10 OBSERVATION:

In full subtractor, the sum of the two inputs is represented using D = A^B^Cin and carry is represented using Bout = (A&B) + (Bin & (A^B))

2.11 RESULT:

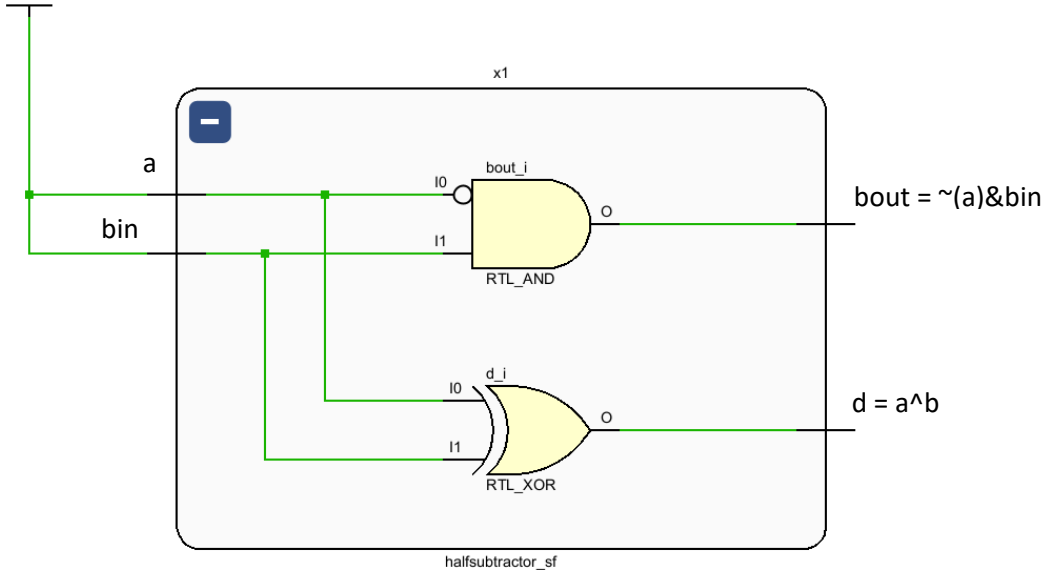
Full Subtractor is simulated and implemented in Behavioral Flow Modeling.

SUBTRACTORS (Structural Flow Modeling)

1.1 AIM: To implement Half Subtractor

1.2 SOFTWARE USED: Xilinx Vivado 2022.2

1.3 SYMBOL:



1.4 LOGIC EXPRESSION:

D = A^Bin

Bout = ~(A)&Bin

1.5 BOOLEAN EXPRESSION:

S = $\bar{A}$Bin+A$\overline{Bin}$

Bout = $\bar{A}$.B

1.6 TRUTH TABLE:

INPUT		OUTPUT	
A	**Bin**	**D**	**Bout**
0	**0**	**0**	**0**
0	**1**	**1**	**1**
1	**0**	**1**	**0**
1	**1**	**0**	**0**

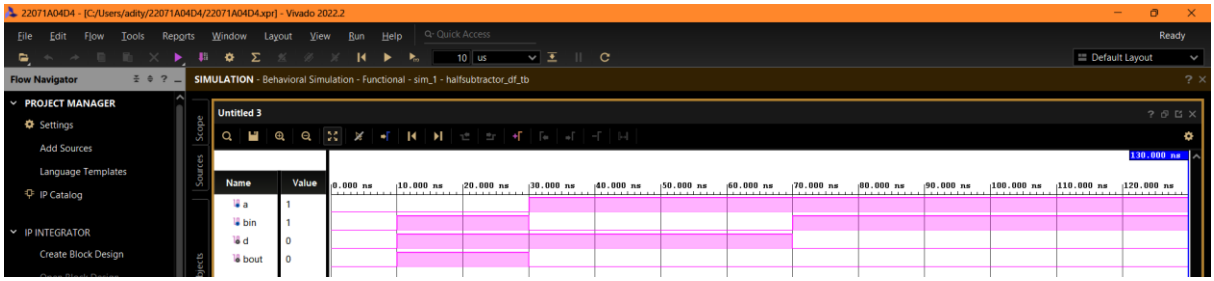
1.7 VERILOG CODE (halfsubtractor_sf.v):

```
module halfsubtractor_sf(d,bout,a,bin);
output d,bout;
input a,bin;
wire x;
xor (d,a,bin);
not (x,a);
and (bout,x,bin);
endmodule
```

1.8 TEST BENCH (halfsubtractor_sf_tb.v):

```
module halfsubtractor_sf_tb();
reg a,bin;
wire d,bout;
halfsubtractor_sf x1(d,bout,a,bin);
initial
begin
{a,bin}=2'b00;
#10 {a,bin}=2'b01;
#20 {a,bin}=2'b10;
#40 {a,bin}=2'b11;
#60 $finish;
end
endmodule
```

1.9 WAVEFORM:



1.10 OBSERVATION:

In half subtractor, the difference of the two inputs is represented using logical XOR of A and Bin and borrow is represented using logical AND of logical NOT of A and Bin.

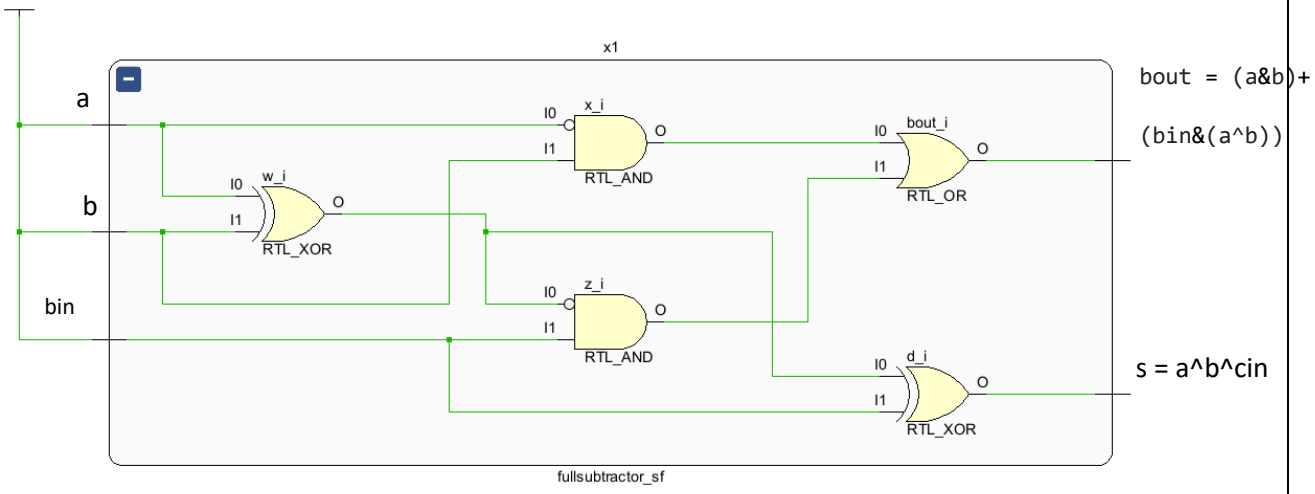
1.11 RESULT:

Half Subtractor is simulated and implemented in Structural Flow Modeling.

2.1 AIM: To implement Full Subtractor

2.2 SOFTWARE USED: Xilinx Vivado 2022.2

2.3 SYMBOL:



2.4 LOGIC EXPRESSION:

D = A^B^Bin

Bout = (~(A)&B)+(Bin&(~(A^B)))

2.5 BOOLEAN EXPRESSION:

$D = \bar{A}(\bar{B}Cin + B\bar{Cin}) + A\overline{(\bar{B}Cin + B\bar{Cin})}$

$Bout = \bar{A}B + Bin(AB + \bar{A}\bar{B})$

2.6 TRUTH TABLE:

INPUT			OUTPUT	
A	**B**	**Bin**	**D**	**Bout**
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**

2.7 VERILOG CODE (fullsubtractor_sf.v):

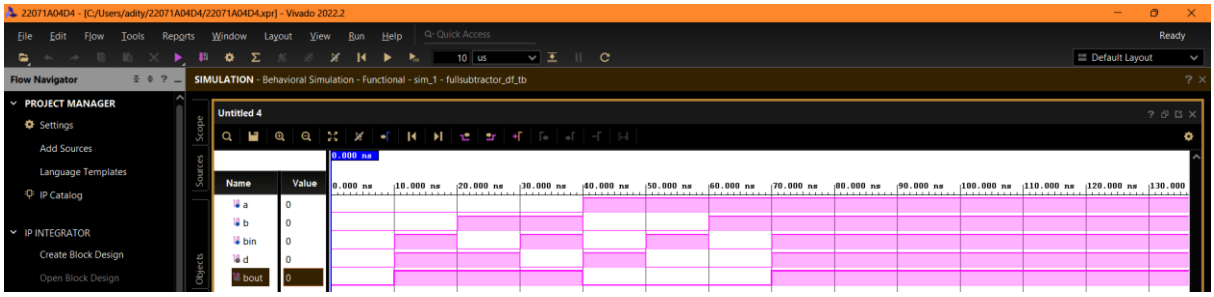
```
module fullsubtractor_sf (d,bout,a,b,bin);
output d,bout;
input a,b,bin;
wire w,x,y,z;
xor (w,a,b);
xor (d,w,bin);
and (x,~a,b);
```

```
xnor (y,a,b);
and (z,y,bin);
or (bout,x,z);
endmodule
```

2.8 TEST BENCH (fullsubtractor_sf_tb.v):

```
module fullsubtractor_sf_tb();
reg a,b,bin;
wire d,bout;
fullsubtractor_sf x1(d,bout,a,b,bin);
initial
begin
{a,b,bin}=3'b000;
#10 {a,b,bin}=3'b001;
#10 {a,b,bin}=3'b010;
#10 {a,b,bin}=3'b011;
#10 {a,b,bin}=3'b100;
#10 {a,b,bin}=3'b101;
#10 {a,b,bin}=3'b110;
#10 {a,b,bin}=3'b111;
end
endmodule
```

2.9 WAVEFORM:



2.10 OBSERVATION:

In full subtractor, the sum of the two inputs is represented using D = A^B^Cin and carry is represented using Bout = (A&B) + (Bin & (A^B))

2.11 RESULT:

Full Subtractor is simulated and implemented in Structural Flow Modeling.

3

EXPERIMENT-3
REALIZATION OF CODE CONVERTERS

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

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

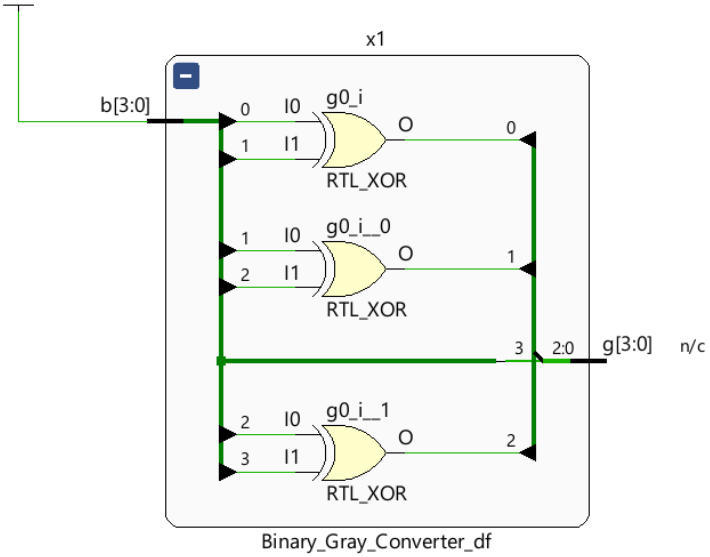
3. **PROCEDURE:**
    - Open the Xilinx Vivado project navigator.
    - Open the Design source and go to add / create sources
    - Create new file, give appropriate name save it.
    - Open the file in the editor and write the Verilog code.
    - Open the Design source and go to add / create sources 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.

Code Converters (Data Flow Modeling)

1.1 AIM: Binary to Gray Code Conversion

1.2 SOFTWARE USED: Xilinx Vivado 2022.2

1.3 SYMBOL:



1.4 LOGIC EXPRESSION:

$G_3 = B_3$

$G_2 = B_3 \wedge B_2$

$G_1 = B_2 \wedge B_1$

$G_0 = B_1 \wedge B_0$

1.5 BOOLEAN EXPRESSION:

$G_3 = B_3$

$G_2 = B_3 \oplus B_2$

$G_1 = B_2 \oplus B_1$

$G_0 = B_1 \oplus B_0$

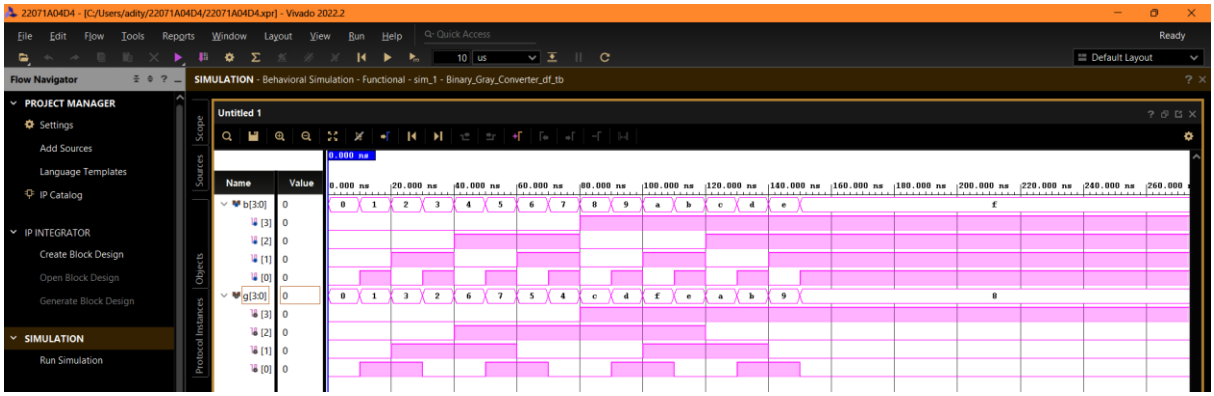
1.6 VERILOG CODE (Binary_Gray_Converter_df.v):

```
module Binary_Gray_Converter_df(g,b);
output [3:0]g;
input [3:0]b;

assign g[3] = b[3];
assign g[2] = b[2]^b[3];
assign g[1] = b[1]^b[2];
assign g[0] = b[0]^b[1];
endmodule
```

1.7 TEST BENCH (Binary_Gray_Converter_df_tb.v):

```
module Binary_Gray_Converter_df_tb();
reg [3:0]b;
wire [3:0]g;
Binary_Gray_Converter_df x1(g,b);
initial
begin
{b[3],b[2],b[1],b[0]}=4'b0000;
#10 {b[3],b[2],b[1],b[0]}=4'b0001;
#10 {b[3],b[2],b[1],b[0]}=4'b0010;
#10 {b[3],b[2],b[1],b[0]}=4'b0011;
#10 {b[3],b[2],b[1],b[0]}=4'b0100;
#10 {b[3],b[2],b[1],b[0]}=4'b0101;
#10 {b[3],b[2],b[1],b[0]}=4'b0110;
#10 {b[3],b[2],b[1],b[0]}=4'b0111;
#10 {b[3],b[2],b[1],b[0]}=4'b1000;
#10 {b[3],b[2],b[1],b[0]}=4'b1001;
#10 {b[3],b[2],b[1],b[0]}=4'b1010;
#10 {b[3],b[2],b[1],b[0]}=4'b1011;
#10 {b[3],b[2],b[1],b[0]}=4'b1100;
#10 {b[3],b[2],b[1],b[0]}=4'b1101;
#10 {b[3],b[2],b[1],b[0]}=4'b1110;
#10 {b[3],b[2],b[1],b[0]}=4'b1111;

end
endmodule
```

1.8 WAVEFORM:



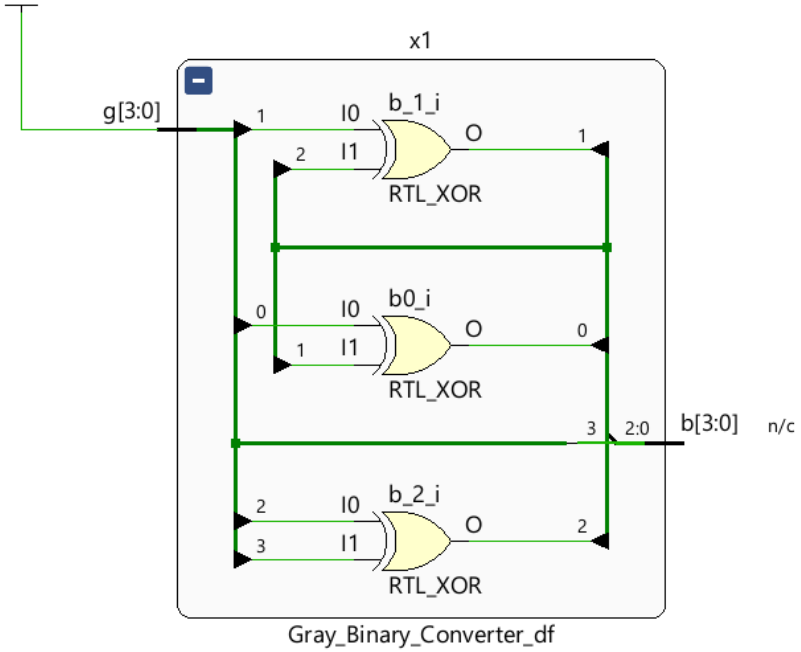
1.9 RESULT:

Binary to Gray Code Conversion is simulated and implemented in Data Flow Modeling.

2.1 AIM: Gray to Binary Code Conversion

2.2 SOFTWARE USED: Xilinx Vivado 2022.2

2.3 SYMBOL:



2.4 LOGIC EXPRESSION:

$B_3 = G_3$

$B_2 = B_3 \wedge G_2$

$B_1 = B_2 \wedge G_1$

$B_0 = B_1 \wedge G_0$

2.5 BOOLEAN EXPRESSION:

$B_3 = G_3$

$B_2 = B_3 \oplus G_2$

$B_1 = B_2 \oplus G_1$

$B_0 = B_1 \oplus G_0$

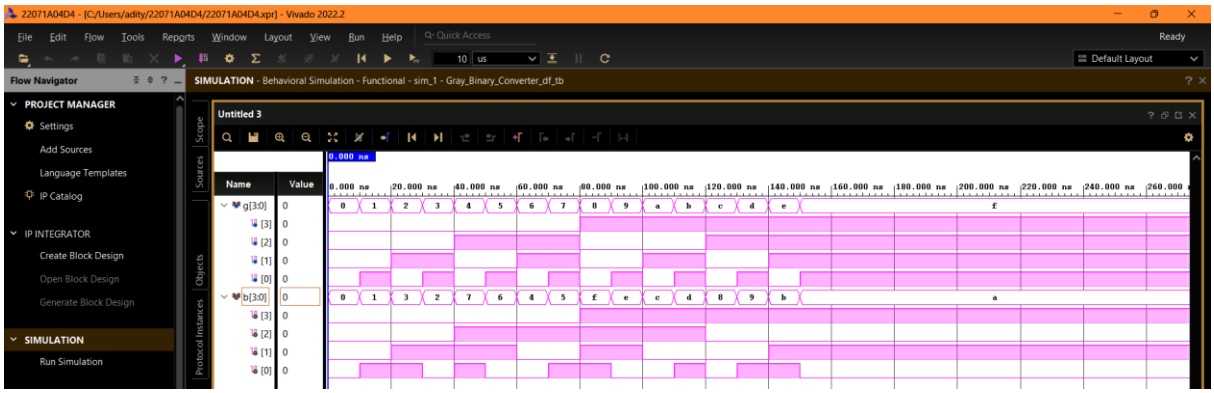
2.6 VERILOG CODE (Gray_Binary_Converter_df.v):

```
module Gray_Binary_Converter_df(b,g);
output [3:0]b;
input [3:0]g;

assign b[3] = g[3];
assign b[2] = g[2]^b[3];
assign b[1] = g[1]^b[2];
assign b[0] = g[0]^b[1];
endmodule
```

2.7 TEST BENCH (Gray_Binary_Converter_df_tb.v):

```
module Gray_Binary_Converter_df_tb();
reg [3:0]g;
wire [3:0]b;
Gray_Binary_Converter_df x1(b,g);
initial
begin
{g[3],g[2],g[1],g[0]}=4'b0000;
#10 {g[3],g[2],g[1],g[0]}=4'b0001;
#10 {g[3],g[2],g[1],g[0]}=4'b0010;
#10 {g[3],g[2],g[1],g[0]}=4'b0011;
#10 {g[3],g[2],g[1],g[0]}=4'b0100;
#10 {g[3],g[2],g[1],g[0]}=4'b0101;
#10 {g[3],g[2],g[1],g[0]}=4'b0110;
#10 {g[3],g[2],g[1],g[0]}=4'b0111;
#10 {g[3],g[2],g[1],g[0]}=4'b1000;
#10 {g[3],g[2],g[1],g[0]}=4'b1001;
#10 {g[3],g[2],g[1],g[0]}=4'b1010;
#10 {g[3],g[2],g[1],g[0]}=4'b1011;
#10 {g[3],g[2],g[1],g[0]}=4'b1100;
#10 {g[3],g[2],g[1],g[0]}=4'b1101;
#10 {g[3],g[2],g[1],g[0]}=4'b1110;
#10 {g[3],g[2],g[1],g[0]}=4'b1111;

end
endmodule
```

2.8 WAVEFORM:



2.9 RESULT:

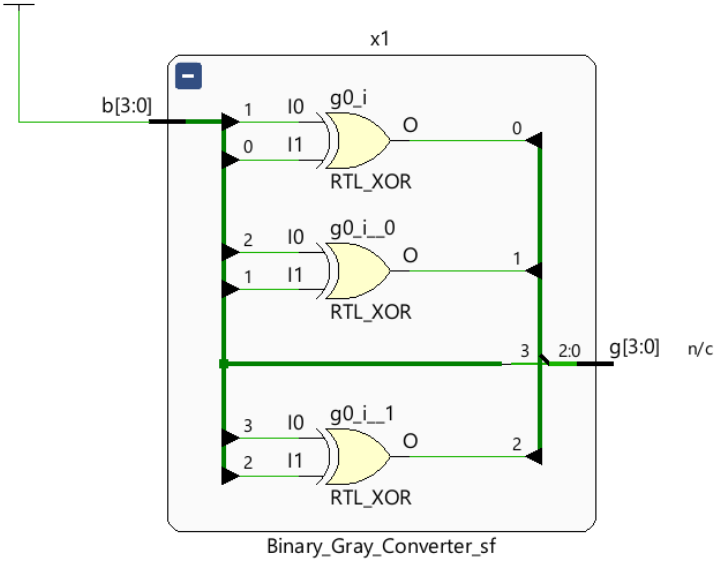
Gray to Binary Code Conversion is simulated and implemented in Data Flow Modeling.

Code Converters (Structural Flow Modeling)

1.1 AIM: Binary to Gray Code Conversion

1.2 SOFTWARE USED: Xilinx Vivado 2022.2

1.3 SYMBOL:



1.4 LOGIC EXPRESSION:

$G_3 = B_3$

$G_2 = B_3 \wedge B_2$

$G_1 = B_2 \wedge B_1$

$G_0 = B_1 \wedge B_0$

1.5 BOOLEAN EXPRESSION:

$G_3 = B_3$

$G_2 = B_3 \oplus B_2$

$G_1 = B_2 \oplus B_1$

$G_0 = B_1 \oplus B_0$

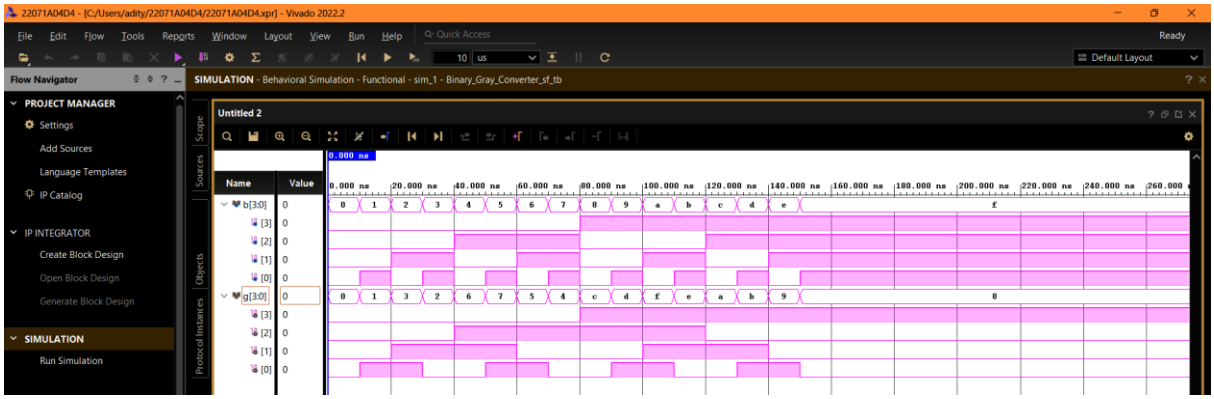
1.6 VERILOG CODE (Binary_Gray_Converter_sf.v):

```
module Binary_Gray_Converter_sf(g,b);
output [3:0]g;
input [3:0]b;

assign g[3] = b[3];
xor(g[2],b[3],b[2]);
xor(g[1],b[2],b[1]);
xor(g[0],b[1],b[0]);

endmodule
```

1.7 TEST BENCH (Binary_Gray_Converter_sf_tb.v):

```
module Binary_Gray_Converter_sf_tb();
reg [3:0]b;
wire [3:0]g;
Binary_Gray_Converter_sf x1(g,b);
initial
begin
{b[3],b[2],b[1],b[0]}=4'b0000;
#10 {b[3],b[2],b[1],b[0]}=4'b0001;
#10 {b[3],b[2],b[1],b[0]}=4'b0010;
#10 {b[3],b[2],b[1],b[0]}=4'b0011;
#10 {b[3],b[2],b[1],b[0]}=4'b0100;
#10 {b[3],b[2],b[1],b[0]}=4'b0101;
#10 {b[3],b[2],b[1],b[0]}=4'b0110;
#10 {b[3],b[2],b[1],b[0]}=4'b0111;
#10 {b[3],b[2],b[1],b[0]}=4'b1000;
#10 {b[3],b[2],b[1],b[0]}=4'b1001;
#10 {b[3],b[2],b[1],b[0]}=4'b1010;
#10 {b[3],b[2],b[1],b[0]}=4'b1011;
#10 {b[3],b[2],b[1],b[0]}=4'b1100;
#10 {b[3],b[2],b[1],b[0]}=4'b1101;
#10 {b[3],b[2],b[1],b[0]}=4'b1110;
#10 {b[3],b[2],b[1],b[0]}=4'b1111;

end
endmodule
```

1.8 WAVEFORM:



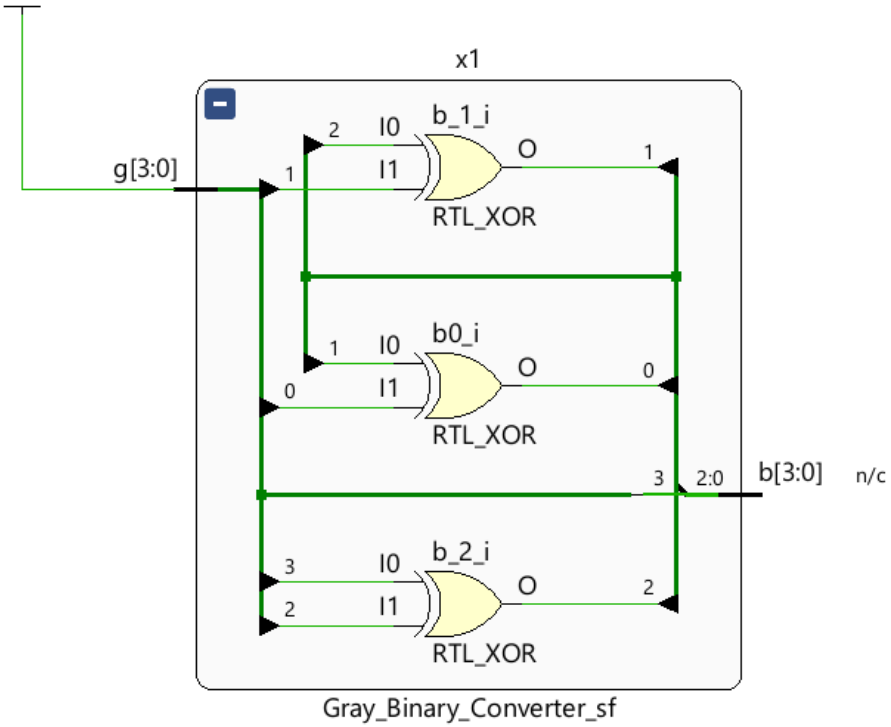
1.9 RESULT:

Binary to Gray Code Conversion is simulated and implemented in Structural Flow Modeling.

2.1 AIM: Gray to Binary Code Conversion

2.2 SOFTWARE USED: Xilinx Vivado 2022.2

2.3 SYMBOL:



2.4 LOGIC EXPRESSION:

$B_3 = G_3$

$B_2 = B_3 \wedge G_2$

$B_1 = B_2 \wedge G_1$

$B_0 = B_1 \wedge G_0$

2.5 BOOLEAN EXPRESSION:

$B_3 = G_3$

$B_2 = B_3 \oplus G_2$

$B_1 = B_2 \oplus G_1$

$B_0 = B_1 \oplus G_0$

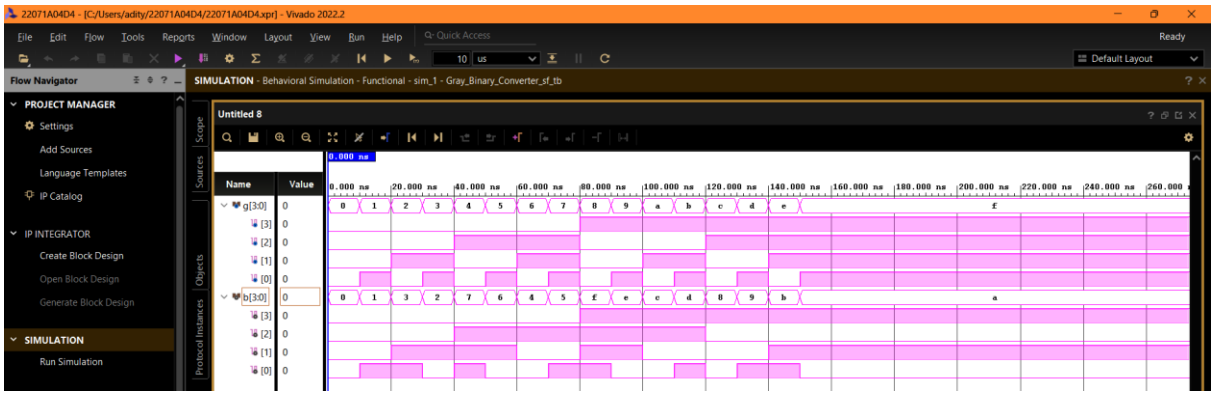
2.6 VERILOG CODE (Gray_Binary_Converter_sf.v):

```
module Gray_Binary_Converter_sf(b,g);
output [3:0]b;
input [3:0]g;

assign b[3] = g[3];
xor(b[2],b[3],g[2]);
xor(b[1],b[2],g[1]);
xor(b[0],b[1],g[0]);
endmodule
```

2.7 TEST BENCH (Gray_Binary_Converter_sf_tb.v):

```
module Gray_Binary_Converter_sf_tb();
reg [3:0]g;
wire [3:0]b;
Gray_Binary_Converter_sf x1(b,g);
initial
begin
{g[3],g[2],g[1],g[0]}=4'b0000;
#10 {g[3],g[2],g[1],g[0]}=4'b0001;
#10 {g[3],g[2],g[1],g[0]}=4'b0010;
#10 {g[3],g[2],g[1],g[0]}=4'b0011;
#10 {g[3],g[2],g[1],g[0]}=4'b0100;
#10 {g[3],g[2],g[1],g[0]}=4'b0101;
#10 {g[3],g[2],g[1],g[0]}=4'b0110;
#10 {g[3],g[2],g[1],g[0]}=4'b0111;
#10 {g[3],g[2],g[1],g[0]}=4'b1000;
#10 {g[3],g[2],g[1],g[0]}=4'b1001;
#10 {g[3],g[2],g[1],g[0]}=4'b1010;
#10 {g[3],g[2],g[1],g[0]}=4'b1011;
#10 {g[3],g[2],g[1],g[0]}=4'b1100;
#10 {g[3],g[2],g[1],g[0]}=4'b1101;
#10 {g[3],g[2],g[1],g[0]}=4'b1110;
#10 {g[3],g[2],g[1],g[0]}=4'b1111;

end
endmodule
```

2.8 WAVEFORM:



2.9 RESULT:

Gray to Binary Code Conversion is simulated and implemented in Structural Flow Modeling.

4

EXPERIMENT-4
REALIZATION OF MULTIPLEXERS AND DEMULTIPLEXERS

1. **AIM:** To Design, simulate and implement Multiplexers and Demultiplexers in 2 different modeling styles(Data Flow, Structural Flow Modeling)

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

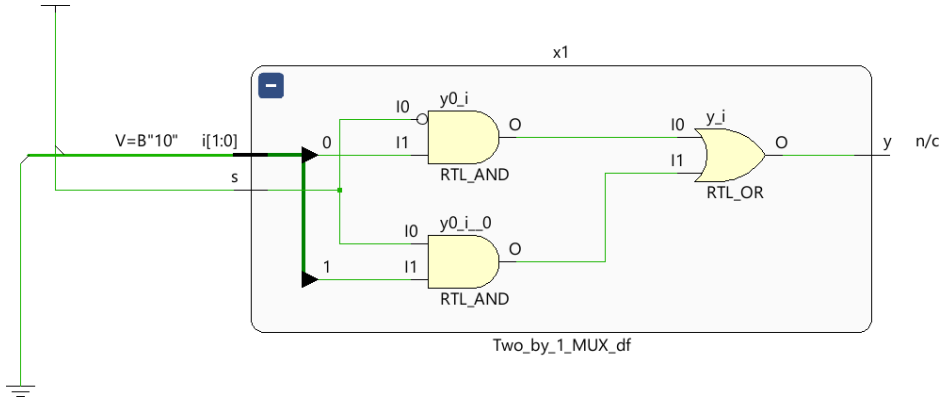
3. **PROCEDURE:**
    - Open the Xilinx Vivado project navigator.
    - Open the Design source and go to add / create sources
    - Create new file, give appropriate name save it.
    - Open the file in the editor and write the Verilog code.
    - Open the Design source and go to add / create sources 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.

Multiplexers (Data Flow Modeling)

1.1 AIM: 2x1 Multiplexer [MUX]

1.2 SOFTWARE USED: Xilinx Vivado 2022.2

1.3 SYMBOL:



1.4 LOGIC EXPRESSION:

$$Y = ((\sim S)\&I_0) \mid (S)\&I_1))$$

1.5 BOOLEAN EXPRESSION:

$$Y = \bar{S}I_0 + SI_1$$

1.6 TRUTH TABLE:

INPUT	OUTPUT
S	**Y**
0	**I_0**
1	**I_1**

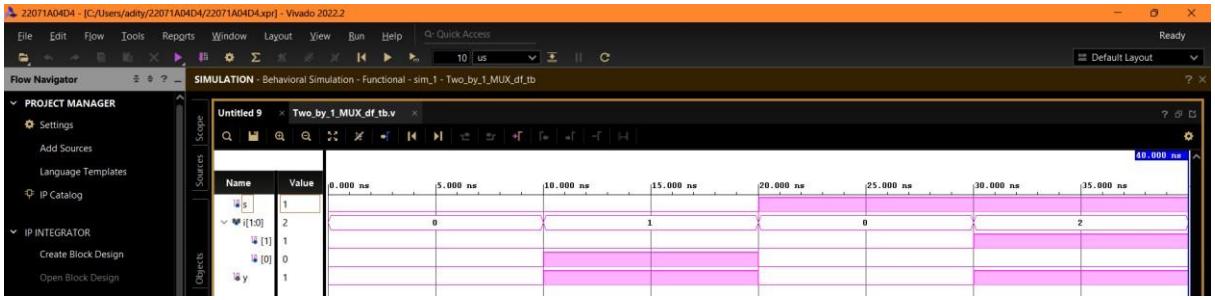
1.7 VERILOG CODE (Two_by_1_MUX_df.v):

```
module Two_by_1_MUX_df(y,s,i);
output y;
input s;
input[1:0]i;
assign y = (((~s)&i[0])|((s)&i[1]));
endmodule
```

1.8 TEST BENCH (Two_by_1_MUX_df_tb.v):

```
module Two_by_1_MUX_df_tb();
reg s;
reg[1:0]i;
wire y;
Two_by_1_MUX_df x1(y,s,i);
initial
begin
s  =  1'b0;i=2'b00;
#10 s = 1'b0;i=2'b01;

#10 s = 1'b1;i=2'b00;
#10 s = 1'b1;i=2'b10;
#10 $finish;
end
endmodule
```

1.9 WAVEFORM:



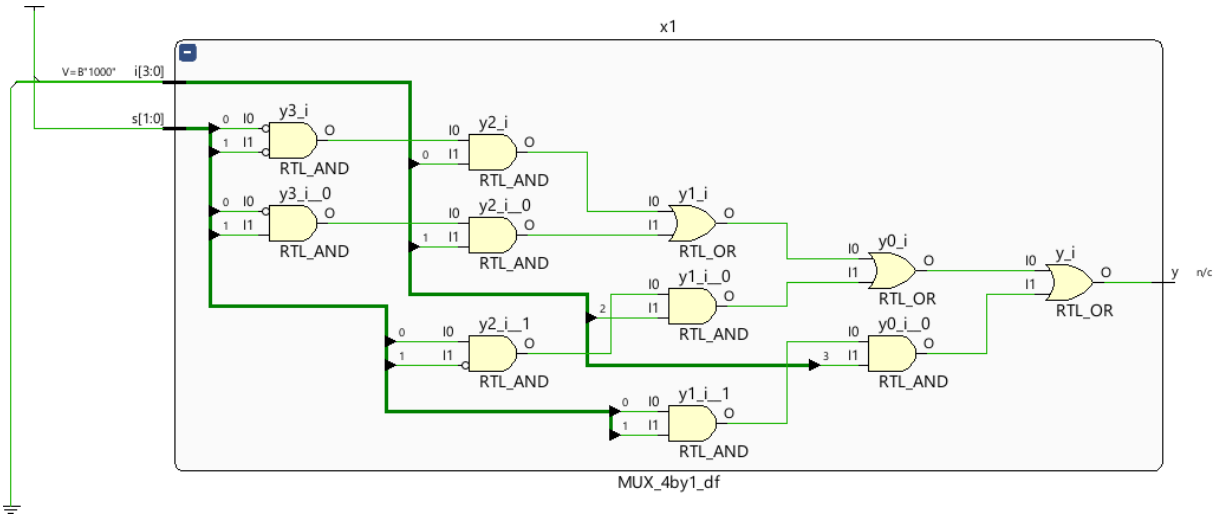
1.10 RESULT:

2x1 Multiplexer is simulated and implemented in Data Flow Modeling.

1.1 AIM: 4x1 Multiplexer [MUX]

1.2 SOFTWARE USED: Xilinx Vivado 2022.2

1.3 SYMBOL:



1.4 LOGIC EXPRESSION:

$$Y = ((\sim S_0)\&(\sim S_1)\&I_0) \mid ((\sim S_0)\&(S_1)\&I_1) \mid ((S_0)\&(\sim S_1)\&I_2) \mid ((S_0)\&(S_1)\&I_3)$$

1.5 BOOLEAN EXPRESSION:

$$Y = \overline{S_0}\,\overline{S_1}I_0 + \overline{S_0}S_1I_1 + S_0\overline{S_1}I_2 + S_0S_1I_3$$

1.6 TRUTH TABLE:

INPUT		OUTPUT
S_0	S_1	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

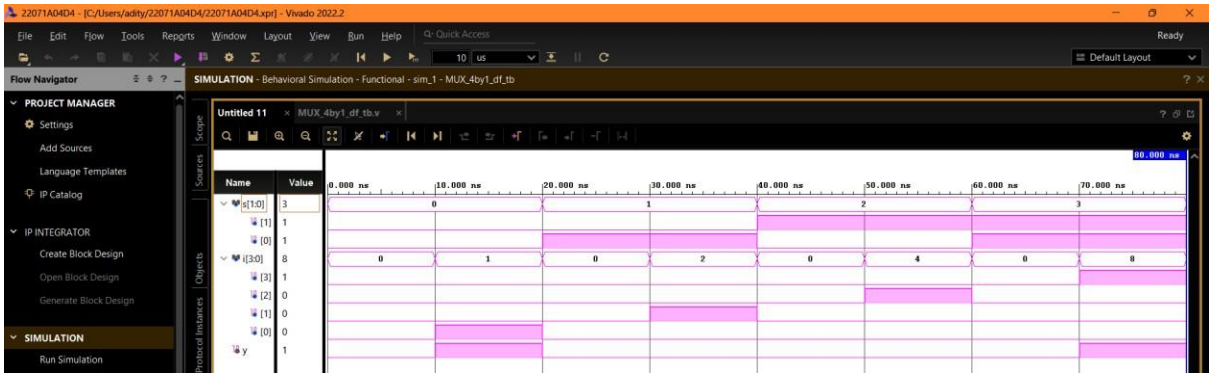
1.7 VERILOG CODE (MUX_4by1_df.v):

```
module MUX_4by1_df(y,s,i);
output y;
input [1:0]s;
input[3:0]i;

assign y = (
((~s[0])&(~s[1])&(i[0]))|
((~s[0])&(s[1])&(i[1]))|
((s[0])&(~s[1])&(i[2]))|
((s[0])&(s[1])&(i[3]))
);
endmodule
```

1.8 TEST BENCH (MUX_4by1_df_tb.v):

```
module MUX_4by1_df_tb();
reg [1:0]s;
reg[3:0]i;
wire y;

MUX_4by1_df x1(y,s,i);

initial
begin
s  =  2'b00;i=4'b0000;
#10 s = 2'b00;i=4'b0001;

#10 s = 2'b01;i=4'b0000;
#10 s = 2'b01;i=4'b0010;

#10 s = 2'b10;i=4'b0000;
#10 s = 2'b10;i=4'b0100;

#10 s = 2'b11;i=4'b0000;
#10 s = 2'b11;i=4'b1000;

#10 $finish;

end
endmodule
```

1.9 WAVEFORM:



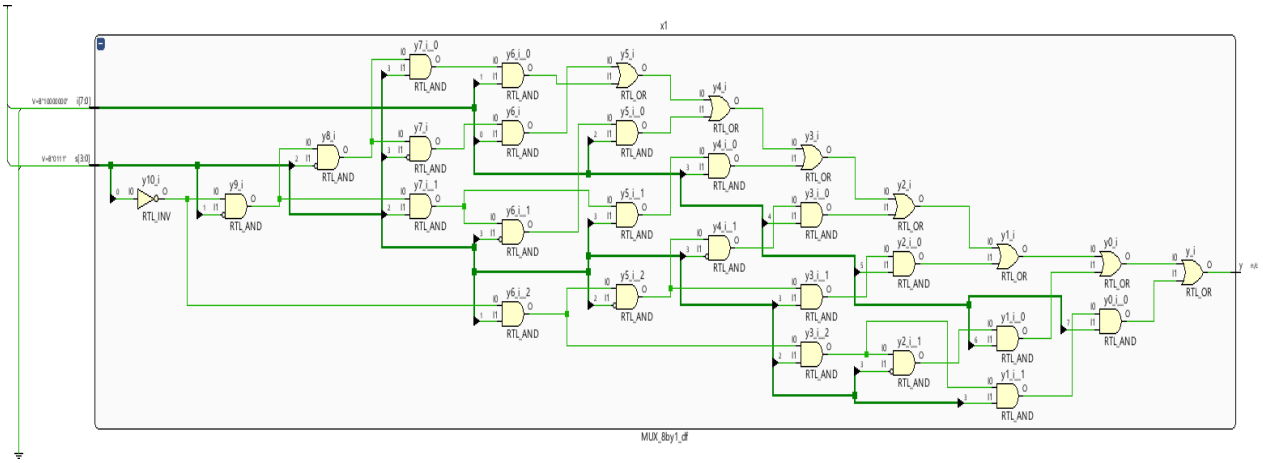
1.10 RESULT:

4x1 Multiplexer is simulated and implemented in Data Flow Modeling.

1.1 AIM: 8x1 Multiplexer [MUX]

1.2 SOFTWARE USED: Xilinx Vivado 2022.2

1.3 SYMBOL:



1.4 LOGIC EXPRESSION:

$$Y = ((\sim S_0) \,\&\, (\sim S_1) \,\&\, (\sim S_2) \,\&\, (\sim S_3) \,\&\, I_0) \,|$$
$$((\sim S_0) \,\&\, (\sim S_1) \,\&\, (\sim S_2) \,\&\, (S_3) \,\&\, I_1) \,|$$
$$((\sim S_0) \,\&\, (\sim S_1) \,\&\, (S_2) \,\&\, (\sim S_3) \,\&I_2) \,|$$
$$((\sim S_0) \,\&\, (\sim S_1) \,\&\, (S_2) \,\&\, (S_3) \,\&I_3) \,|$$
$$((\sim S_0) \,\&\, (S_1) \,\&\, (\sim S_2) \,\&\, (\sim S_3) \,\&I_4) \,|$$
$$((\sim S_0) \,\&\, (S_1) \,\&\, (\sim S_2) \,\&\, (S_3) \,\&I_5) \,|$$
$$((\sim S_0) \,\&\, (S_1) \,\&\, (S_2) \,\&\, (\sim S_3) \,\&I_6) \,|$$
$$((\sim S_0) \,\&\, (S_1) \,\&\, (S_2) \,\&\, (S_3) \,\&I_7)$$

1.5 BOOLEAN EXPRESSION:

$$Y = \overline{S_0}\overline{S_1}\overline{S_2}\overline{S_3}I_0 + \overline{S_0}\overline{S_1}\overline{S_2}S_3I_1 + \overline{S_0}\overline{S_1}S_2\overline{S_3}I_2 + \overline{S_0}\overline{S_1}S_2S_3I_3 + \overline{S_0}S_1\overline{S_2}\overline{S_3}I_4 + \overline{S_0}S_1\overline{S_2}S_3I_5 + \overline{S_0}S_1S_2\overline{S_3}I_6 + \overline{S_0}S_1S_2S_3I_7$$

1.6 TRUTH TABLE:

INPUT				**OUTPUT**
S_0	**S_1**	**S_2**	**S_3**	**I**
0	**0**	**0**	**0**	**I_0**
0	**0**	**0**	**1**	**I_1**
0	**0**	**1**	**0**	**I_2**
0	**0**	**1**	**1**	**I_3**
0	**1**	**0**	**0**	**I_4**
0	**1**	**0**	**1**	**I_5**
0	**1**	**1**	**0**	**I_6**
0	**1**	**1**	**1**	**I_7**

1.7 VERILOG CODE (MUX_8by1_df.v):

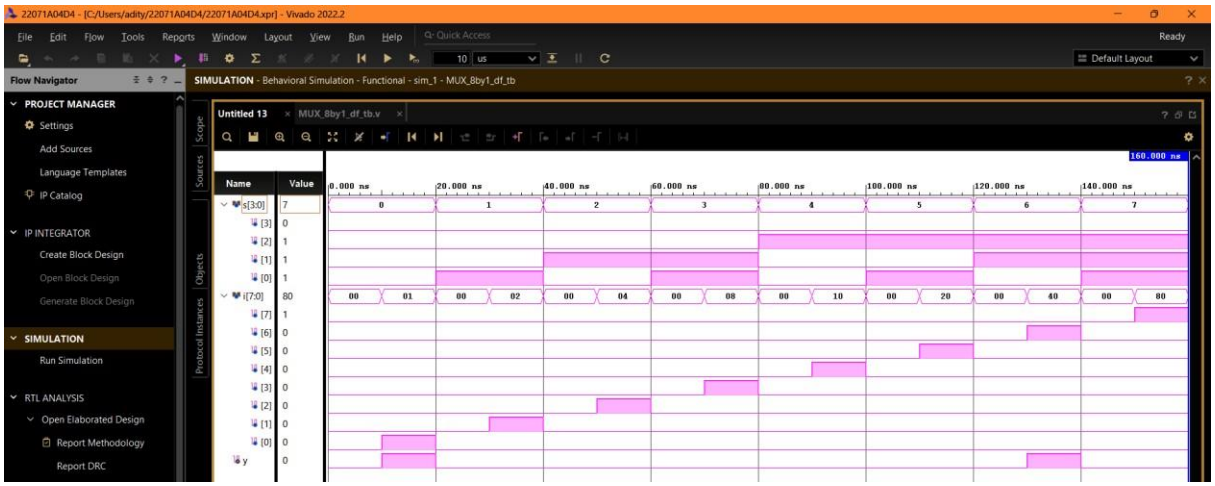
```
module MUX_8by1_df(y,s,i);
output y;
input [3:0]s;
input[7:0]i;
assign y = (
((~s[0])&(~s[1])&(~s[2])&(~s[3])&(i[0]))|
((~s[0])&(~s[1])&(~s[2])&(s[3])&(i[1]))|
((~s[0])&(~s[1])&(s[2])&(~s[3])&(i[2]))|
((~s[0])&(~s[1])&(s[2])&(s[3])&(i[3]))|
((~s[0])&(s[1])&(~s[2])&(~s[3])&(i[4]))|
((~s[0])&(s[1])&(~s[2])&(s[3])&(i[5]))|
((~s[0])&(s[1])&(s[2])&(~s[3])&(i[6]))|
((~s[0])&(s[1])&(s[2])&(s[3])&(i[7]))
);
endmodule.
```

1.8 TEST BENCH (MUX_8by1_df_tb.v):

```
module MUX_8by1_df_tb();
reg [3:0]s;
reg[7:0]i;
wire y;

MUX_8by1_df x1(y,s,i);

initial
begin
s  =  4'b0000;i=8'b00000000;
#10 s = 4'b0000;i=8'b00000001;

#10 s = 4'b0001;i=8'b00000000;
#10 s = 4'b0001;i=8'b00000010;

#10 s = 4'b0010;i=8'b00000000;
#10 s = 4'b0010;i=8'b00000100;

#10 s = 4'b0011;i=8'b00000000;
#10 s = 4'b0011;i=8'b00001000;

#10 s = 4'b0100;i=8'b00000000;
#10 s = 4'b0100;i=8'b00010000;

#10 s = 4'b0101;i=8'b00000000;
#10 s = 4'b0101;i=8'b00100000;

#10 s = 4'b0110;i=8'b00000000;
#10 s = 4'b0110;i=8'b01000000;
```

```
#10 s = 4'b0111;i=8'b00000000;
#10 s = 4'b0111;i=8'b10000000;


#10 $finish;

end
endmodule
```

1.9 WAVEFORM:



1.10 RESULT:

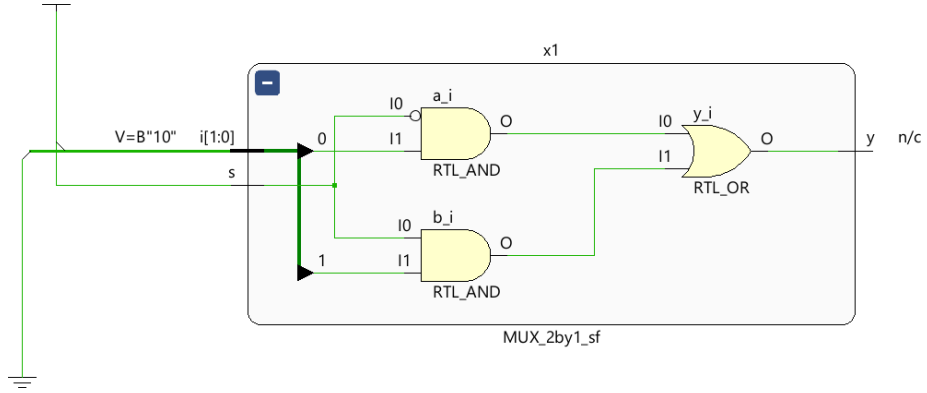
8x1 Multiplexer is simulated and implemented in Data Flow Modeling.

Multiplexers (Structural Flow Modeling)

1.1 AIM: 2x1 Multiplexer [MUX]

1.2 SOFTWARE USED: Xilinx Vivado 2022.2

1.3 SYMBOL:



1.4 LOGIC EXPRESSION:

$$Y = ((\sim S)\&I_0) \mid (S)\&I_1))$$

1.5 BOOLEAN EXPRESSION:

$$Y = \bar{S}I_0 + SI_1$$

1.6 TRUTH TABLE:

INPUT	OUTPUT
S	**Y**
0	**I_0**
1	**I_1**

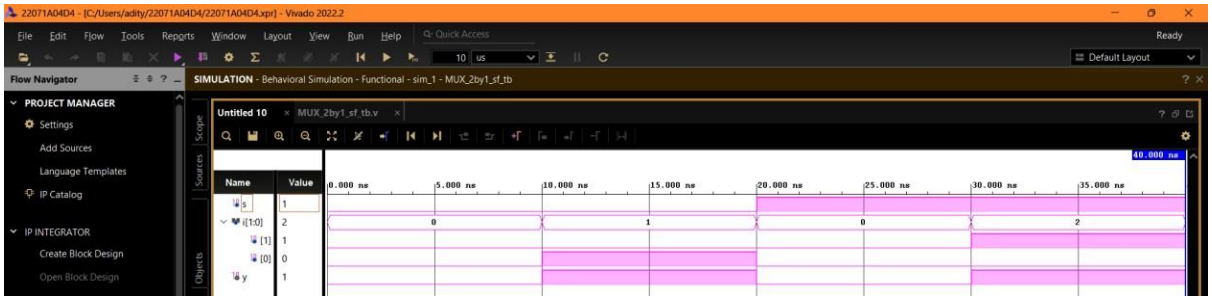
1.7 VERILOG CODE (MUX_2by1_sf.v):

```
module MUX_2by1_sf(y,s,i);
output y;

input s;
input[1:0]i;

wire a,b,c;

and (a,~s,i[0]);

and (b,s,i[1]);

or(y,a,b);

endmodule
```

1.8 TEST BENCH (MUX_2by1_sf_tb.v):

```
module MUX_2by1_sf_tb();
reg s;
reg[1:0]i;
wire y;

MUX_2by1_sf x1(y,s,i);

initial
begin
s  =  1'b0;i=2'b00;
#10 s = 1'b0;i=2'b01;

#10 s = 1'b1;i=2'b00;
#10 s = 1'b1;i=2'b10;

#10 $finish;

end
endmodule
```

1.9 WAVEFORM:



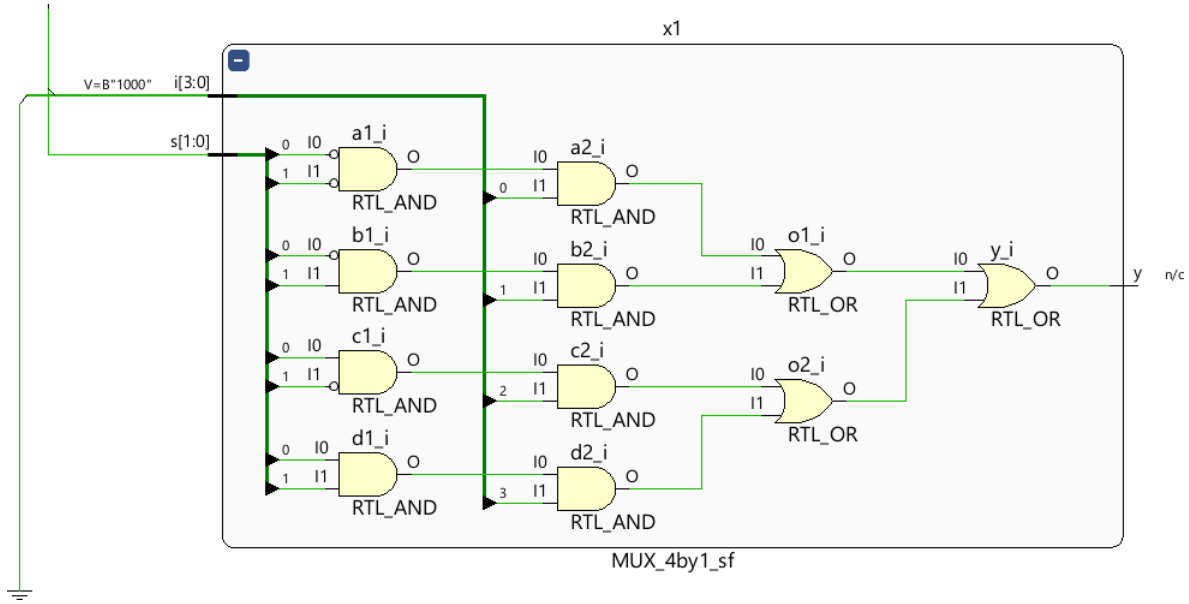
1.10 RESULT:

2x1 Multiplexer is simulated and implemented in Structural Flow Modeling.

1.1 AIM: 4x1 Multiplexer [MUX]

1.2 SOFTWARE USED: Xilinx Vivado 2022.2

1.3 SYMBOL:



1.4 LOGIC EXPRESSION:

$$Y = ((\sim S_0)\&(\sim S_1)\&I_0) \mid ((\sim S_0)\&(S_1)\&I_1) \mid ((S_0)\&(\sim S_1)\&I_2) \mid ((S_0)\&(S_1)\&I_3)$$

1.5 BOOLEAN EXPRESSION:

$$Y = \overline{S_0}\,\overline{S_1}I_0 + \overline{S_0}S_1I_1 + S_0\overline{S_1}I_2 + S_0S_1I_3$$

1.6 TRUTH TABLE:

INPUT		**OUTPUT**
S_0	**S_1**	**Y**
0	**0**	**I_0**
0	**1**	**I_1**
1	**0**	**I_2**
1	**1**	**I_3**

1.7 VERILOG CODE (MUX_4by1_sf.v):

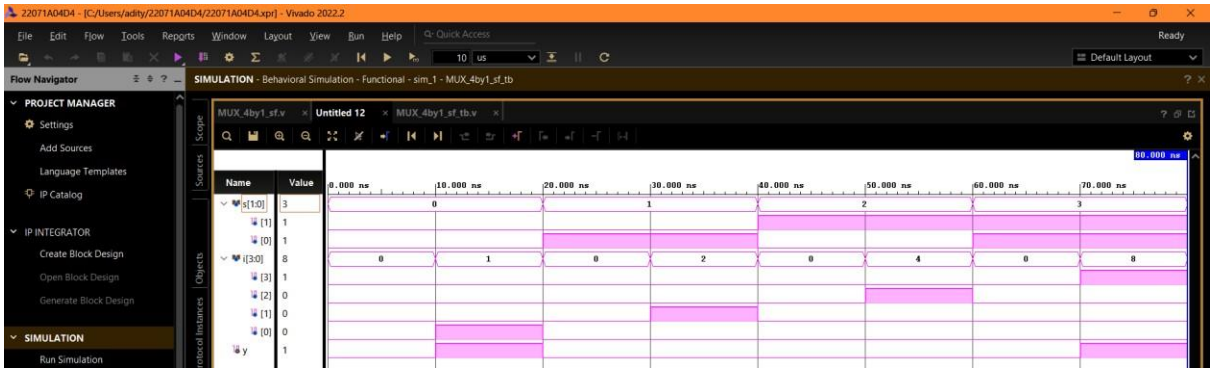
```
module MUX_4by1_sf(y,s,i);
output y;
input [1:0]s;
input[3:0]i;

wire a1,a2,b1,b2,c1,c2,d1,d2,o1,o2;

and (a1,~s[0],~s[1]);
and (a2,a1,i[0]);

and (b1,~s[0],s[1]);
```

```
and (b2,b1,i[1]);

and (c1,s[0],~s[1]);
and (c2,c1,i[2]);

and (d1,s[0],s[1]);
and (d2,d1,i[3]);

or (o1,a2,b2);

or (o2.c2,d2);

or (y,o1,o2);

endmodule
```

1.8 TEST BENCH (MUX_4by1_sf_tb.v):

```
module MUX_4by1_sf_tb();
reg [1:0]s;
reg[3:0]i;
wire y;

MUX_4by1_sf x1(y,s,i);

initial
begin
s  =  2'b00;i=4'b0000;
#10 s = 2'b00;i=4'b0001;

#10 s = 2'b01;i=4'b0000;
#10 s = 2'b01;i=4'b0010;

#10 s = 2'b10;i=4'b0000;
#10 s = 2'b10;i=4'b0100;

#10 s = 2'b11;i=4'b0000;
#10 s = 2'b11;i=4'b1000;

#10 $finish;

end
endmodule
```

1.9 WAVEFORM:



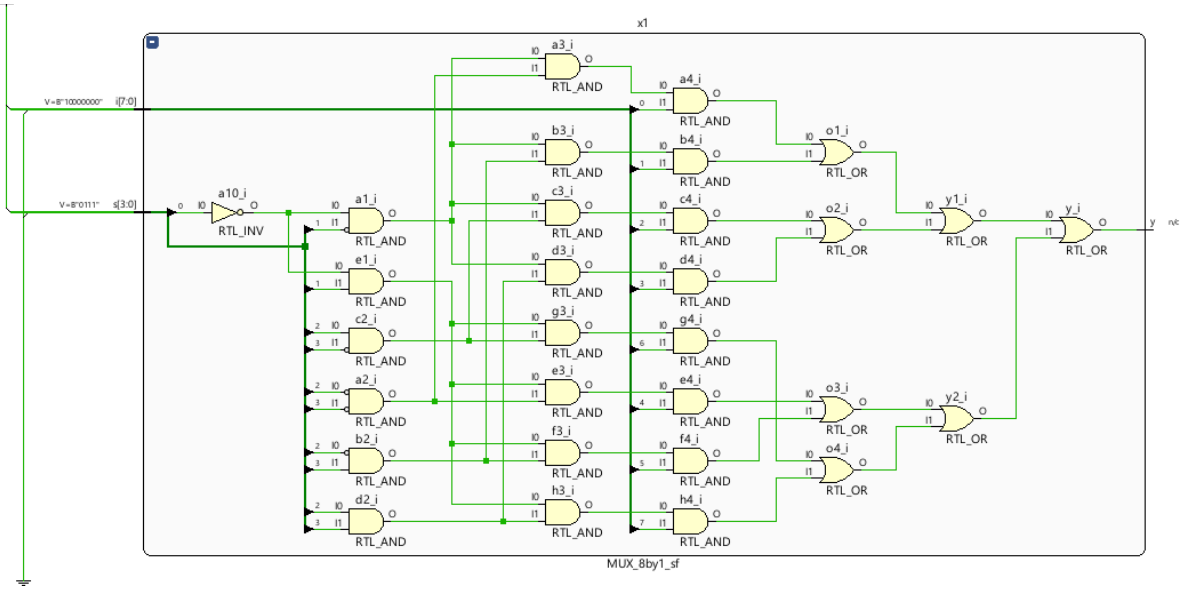
1.10 RESULT:

4x1 Multiplexer is simulated and implemented in Structural Flow Modeling.

1.1 AIM: 8x1 Multiplexer [MUX]

1.2 SOFTWARE USED: Xilinx Vivado 2022.2

1.3 SYMBOL:



1.4 LOGIC EXPRESSION:

$$Y = ((\sim S_0) \,\&\, (\sim S_1) \,\&\, (\sim S_2) \,\&\, (\sim S_3) \,\&\, I_0) \,|$$
$$((\sim S_0) \,\&\, (\sim S_1) \,\&\, (\sim S_2) \,\&\, (S_3) \,\&\, I_1) \,|$$
$$((\sim S_0) \,\&\, (\sim S_1) \,\&\, (S_2) \,\&\, (\sim S_3) \,\&I_2) \,|$$
$$((\sim S_0) \,\&\, (\sim S_1) \,\&\, (S_2) \,\&\, (S_3) \,\&I_3) \,|$$
$$((\sim S_0) \,\&\, (S_1) \,\&\, (\sim S_2) \,\&\, (\sim S_3) \,\&I_4) \,|$$
$$((\sim S_0) \,\&\, (S_1) \,\&\, (\sim S_2) \,\&\, (S_3) \,\&I_5) \,|$$
$$((\sim S_0) \,\&\, (S_1) \,\&\, (S_2) \,\&\, (\sim S_3) \,\&I_6) \,|$$
$$((\sim S_0) \,\&\, (S_1) \,\&\, (S_2) \,\&\, (S_3) \,\&I_7)$$

1.5 BOOLEAN EXPRESSION:

$$Y = \overline{S_0}S_1\overline{S_2}\overline{S_3}I_0 + \overline{S_0}\overline{S_1}\overline{S_2}S_3I_1 + \overline{S_0}\overline{S_1}S_2\overline{S_3}I_2 + \overline{S_0}\overline{S_1}S_2S_3I_3 + \overline{S_0}S_1\overline{S_2}\overline{S_3}I_4 + \overline{S_0}S_1\overline{S_2}S_3I_5 + \overline{S_0}S_1S_2\overline{S_3}I_6 + \overline{S_0}S_1S_2S_3I_7$$

1.6 TRUTH TABLE:

INPUT				**OUTPUT**
S_0	**S_1**	**S_2**	**S_3**	**I**
0	**0**	**0**	**0**	**I_0**
0	**0**	**0**	**1**	**I_1**
0	**0**	**1**	**0**	**I_2**
0	**0**	**1**	**1**	**I_3**
0	**1**	**0**	**0**	**I_4**
0	**1**	**0**	**1**	**I_5**
0	**1**	**1**	**0**	**I_6**
0	**1**	**1**	**1**	**I_7**

1.7 VERILOG CODE (MUX_8by1_sf.v):

```
module MUX_8by1_sf(y,s,i);
output y;
input [3:0]s;
input [7:0]i;

wire a1,a2,a3,a4;
wire b1,b2,b3,b4;
wire c1,c2,c3,c4;
wire d1,d2,d3,d4;
wire e1,e2,e3,e4;
wire f1,f2,f3,f4;
wire g1,g2,g3,g4;
wire h1,h2,h3,h4;
wire o1,o2,o3,o4;
wire y1,y2;

and (a1,~s[0],~s[1]);
and (a2,~s[2],~s[3]);
and(a3,a1,a2);
and(a4,a3,i[0]);

and (b1,~s[0],~s[1]);
and (b2,~s[2],s[3]);
and (b3,b1,b2);
and (b4,b3,i[1]);

and (c1,~s[0],~s[1]);
and (c2,s[2],~s[3]);
and (c3,c1,c2);
and (c4,c3,i[2]);

and (d1,~s[0],~s[1]);
and (d2,s[2],s[3]);
and (d3,d1,d2);
and (d4,d3,i[3]);

and (e1,~s[0],s[1]);
and (e2,~s[2],~s[3]);
and (e3,e1,e2);
and (e4,e3,i[4]);

and (f1,~s[0],s[1]);
and (f2,~s[2],s[3]);
and (f3,f1,f2);
and (f4,f3,i[5]);

and (g1,~s[0],s[1]);
```

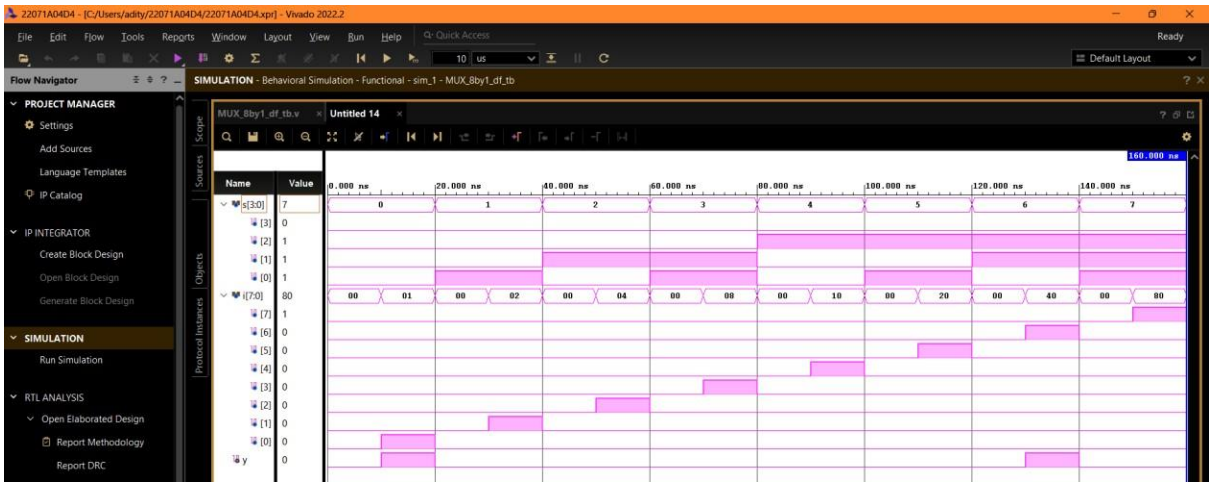
```
and (g2,s[2],~s[3]);
and (g3,g1,g2);
and (g4,g3,i[6]);

and (h1,~s[0],s[1]);
and (h2,s[2],s[3]);
and (h3,h1,h2);
and (h4,h3,i[7]);

or (o1,a4,b4);
or (o2,c4,d4);
or (o3,e4,f4);
or (o4,g4,h4);

or(y1,o1,o2);
or(y2,o3,o4);

or(y,y1,y2);

endmodule
```

1.8 TEST BENCH (MUX_8by1_sf_tb.v):

```
module MUX_8by1_sf_tb();
reg [3:0]s;
reg[7:0]i;
wire y;

MUX_8by1_sf x1(y,s,i);

initial
begin
s  =  4'b0000;i=8'b00000000;
#10 s = 4'b0000;i=8'b00000001;

#10 s = 4'b0001;i=8'b00000000;
#10 s = 4'b0001;i=8'b00000010;

#10 s = 4'b0010;i=8'b00000000;
#10 s = 4'b0010;i=8'b00000100;

#10 s = 4'b0011;i=8'b00000000;
#10 s = 4'b0011;i=8'b00001000;

#10 s = 4'b0100;i=8'b00000000;
#10 s = 4'b0100;i=8'b00010000;

#10 s = 4'b0101;i=8'b00000000;
#10 s = 4'b0101;i=8'b00100000;
```

```
#10 s = 4'b0110;i=8'b00000000;
#10 s = 4'b0110;i=8'b01000000;

#10 s = 4'b0111;i=8'b00000000;
#10 s = 4'b0111;i=8'b10000000;


#10 $finish;

end
endmodule
```

1.9 WAVEFORM:



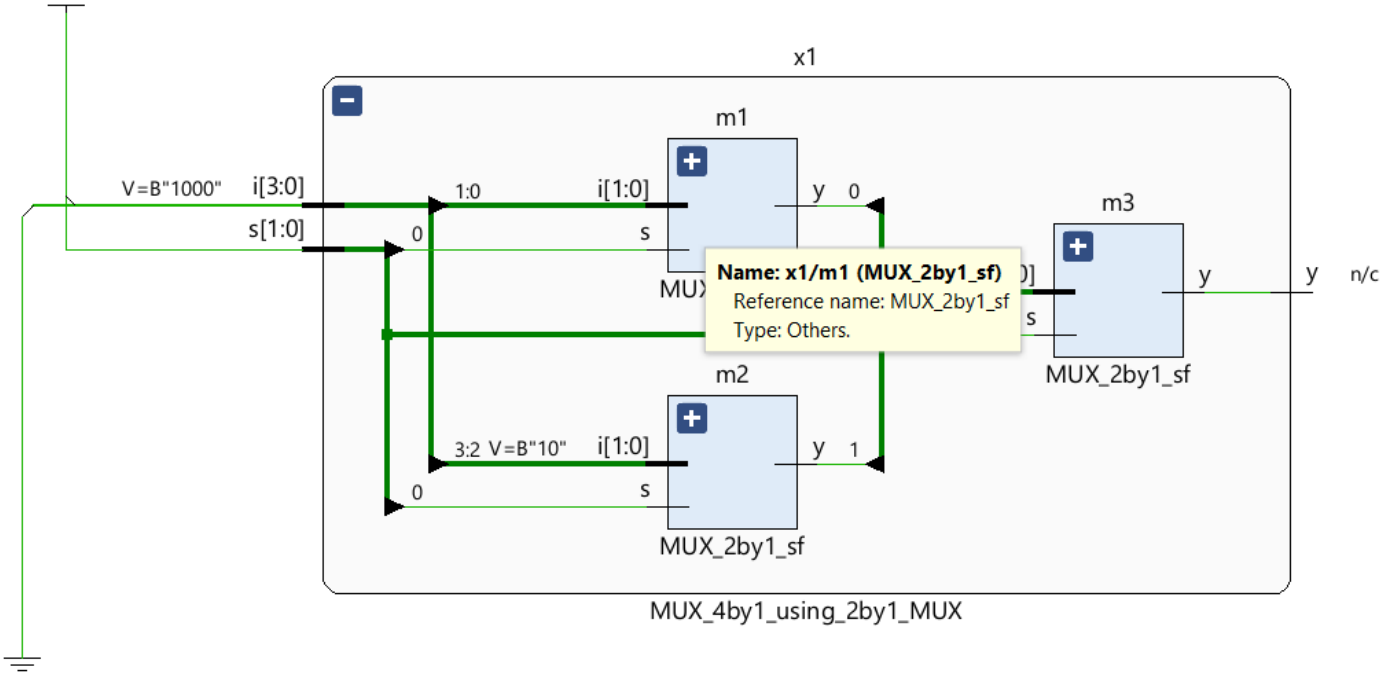
1.10 RESULT:

8x1 Multiplexer is simulated and implemented in Structural Flow Modeling.

1.1 AIM: 4x1 MUX using 2x1 MUX

1.2 SOFTWARE USED: Xilinx Vivado 2022.2

1.3 SYMBOL:



1.4 LOGIC EXPRESSION:

$$Y = ((\sim S_0)\&(\sim S_1)\&I_0) \mid ((\sim S_0)\&(S_1)\&I_1) \mid ((S_0)\&(\sim S_1)\&I_2) \mid ((S_0)\&(S_1)\&I_3)$$

1.5 BOOLEAN EXPRESSION:

$$Y = \overline{S_0}\,\overline{S_1}I_0 + \overline{S_0}S_1I_1 + S_0\overline{S_1}I_2 + S_0S_1I_3$$

1.6 TRUTH TABLE:

INPUT		OUTPUT
S_0	S_1	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

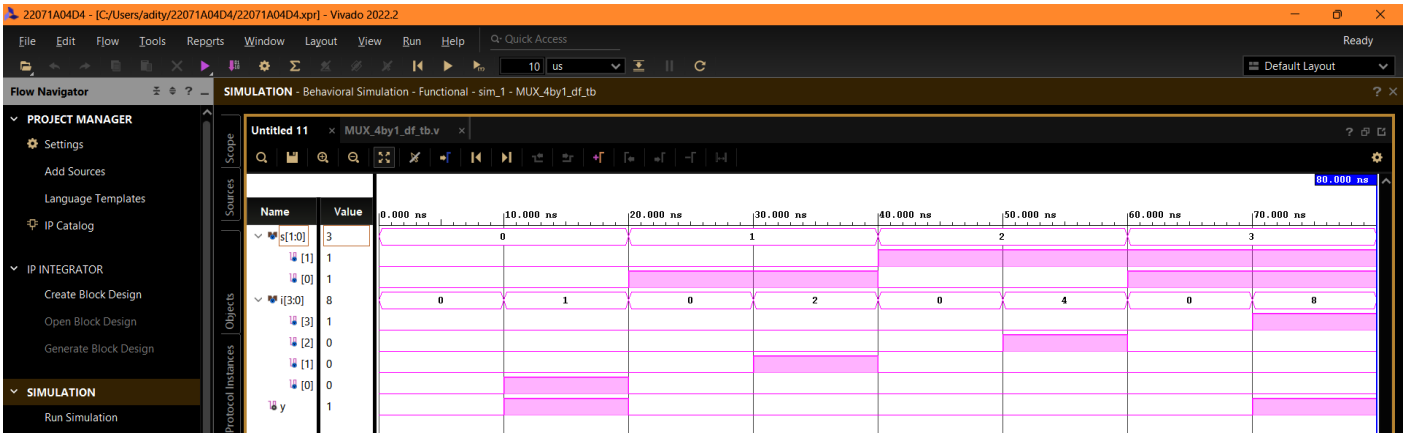
1.7 VERILOG CODE (MUX_4by1_using_2by1_MUX.v):

```
module MUX_4by1_using_2by1_MUX(y,s,i);
input [3:0]i;
input [1:0]s;
output y;
wire [1:0]w;

MUX_2by1_sf m1(w[0],s[0],i[1:0]);
MUX_2by1_sf m2(w[1],s[0],i[3:2]);
MUX_2by1_sf m3(y,s[1],w[1:0]);
endmodule
```

1.8 TEST BENCH (MUX_4by1_using_2by1_MUX _tb.v):

```
module MUX_4by1_using_2by1_MUX_tb();
reg [1:0]s;
reg[3:0]i;
wire y;

MUX_4by1_using_2by1_MUX x1(y,s,i);

initial
begin
s = 2'b00;i=4'b0000;
#10 s = 2'b00;i=4'b0001;

#10 s = 2'b01;i=4'b0000;
#10 s = 2'b01;i=4'b0010;

#10 s = 2'b10;i=4'b0000;
#10 s = 2'b10;i=4'b0100;

#10 s = 2'b11;i=4'b0000;
#10 s = 2'b11;i=4'b1000;

#10 $finish;

end
endmodule
```

1.9 WAVEFORM:



1.10 RESULT:

4x1 Multiplexer is simulated and implemented using 2x1 Multiplexer.

5

EXPERIMENT-5
REALIZATION OF COMPARATORS

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

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

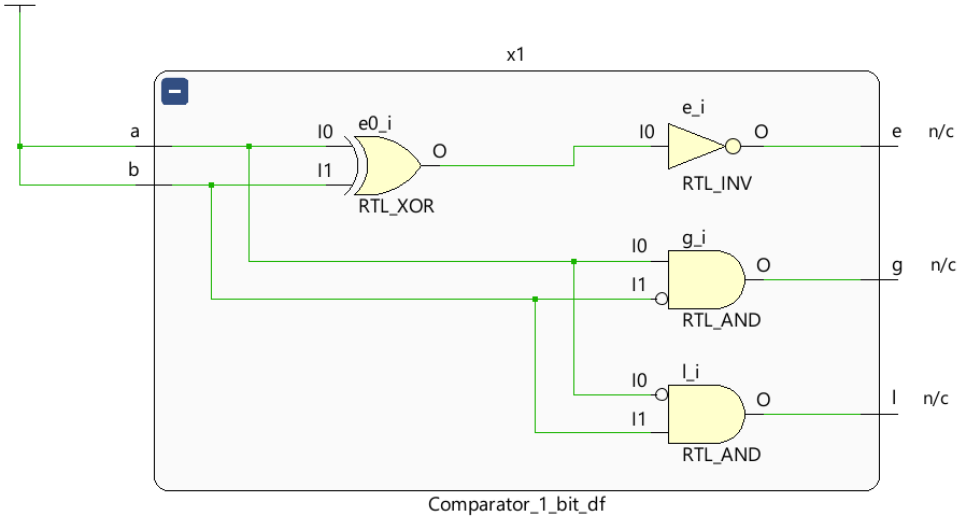
3. **PROCEDURE:**
    - Open the Xilinx Vivado project navigator.
    - Open the Design source and go to add / create sources
    - Create new file, give appropriate name save it.
    - Open the file in the editor and write the Verilog code.
    - Open the Design source and go to add / create sources 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.

Comparators (Data Flow Modeling)

1.1 AIM: 1 Bit Comparator

1.2 SOFTWARE USED: Xilinx Vivado 2022.2

1.3 SYMBOL:



1.4 LOGIC EXPRESSION:

E = ~ (A^B)
G = (~B) & A
L = (~A) & B

1.5 BOOLEAN EXPRESSION:

$E = \bar{A}\bar{B} + AB$
$G = A\bar{B}$
$L = \bar{A}B$

1.6 TRUTH TABLE:

INPUT		OUTPUT		
A	**B**	**E[$\sum_m$=(0,3)]**	**G[$\sum_m$=(2)]**	**L[$\sum_m$=(1)]**
0	**0**	**1**	**0**	**0**
0	**1**	**0**	**0**	**1**
1	**0**	**0**	**1**	**0**
1	**1**	**1**	**0**	**0**

2.1 VERILOG CODE (Comparator_1_bit_df.v):

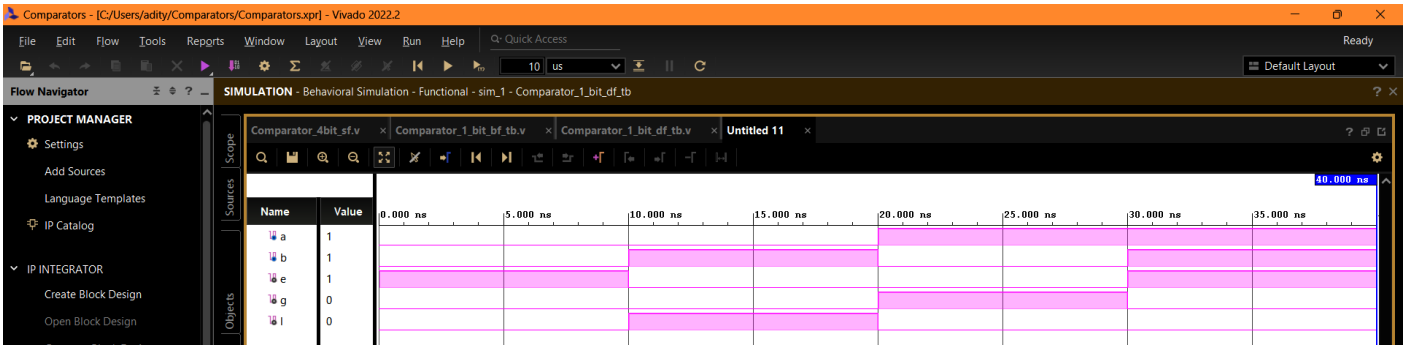
```
module Comparator_1_bit_df(e,g,l,a,b);
output e,g,l;
input a,b;

assign e = ~(a^b);  // XNOR Gate
assign g = a&(~b);
assign l = (~a)&b;

endmodule
```

.

1.8 TEST BENCH (Comparator_1_bit_df _tb.v):

```
module Comparator_1_bit_df_tb();
reg a,b;
wire e,g,l;
Comparator_1_bit_df x1(e,g,l,a,b);
initial
begin
a=1'b0;b=1'b0;
#10 a=1'b0;b=1'b1;
#10 a=1'b1;b=1'b0;
#10 a=1'b1;b=1'b1;
#10 $finish;
end
endmodule
```

1.9 WAVEFORM:



1.10 RESULT:

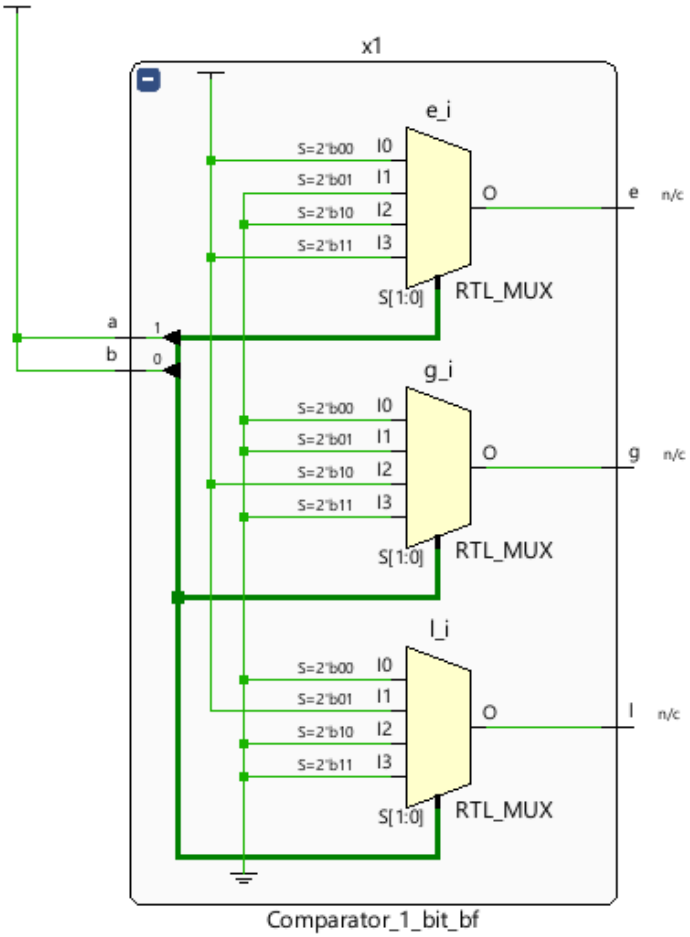
1 Bit Comparator is simulated and implemented in Data Flow Modeling.

Comparators (Behavioral Flow Modeling)

1.1 AIM: 1 Bit Comparator

1.2 SOFTWARE USED: Xilinx Vivado 2022.2

1.3 SYMBOL:



1.4 LOGIC EXPRESSION:

E = ~ (A^B)
G = (~B) & A
L = (~A) & B

1.5 BOOLEAN EXPRESSION:

$E = \bar{A}\bar{B} + AB$
$G = A\bar{B}$
$L = \bar{A}B$

1.6 TRUTH TABLE:

INPUT		OUTPUT		
A	**B**	**E[$\sum_m$=(0,3)]**	**G[$\sum_m$=(2)]**	**L[$\sum_m$=(1)]**
0	**0**	**1**	**0**	**0**
0	**1**	**0**	**0**	**1**
1	**0**	**0**	**1**	**0**
1	**1**	**1**	**0**	**0**

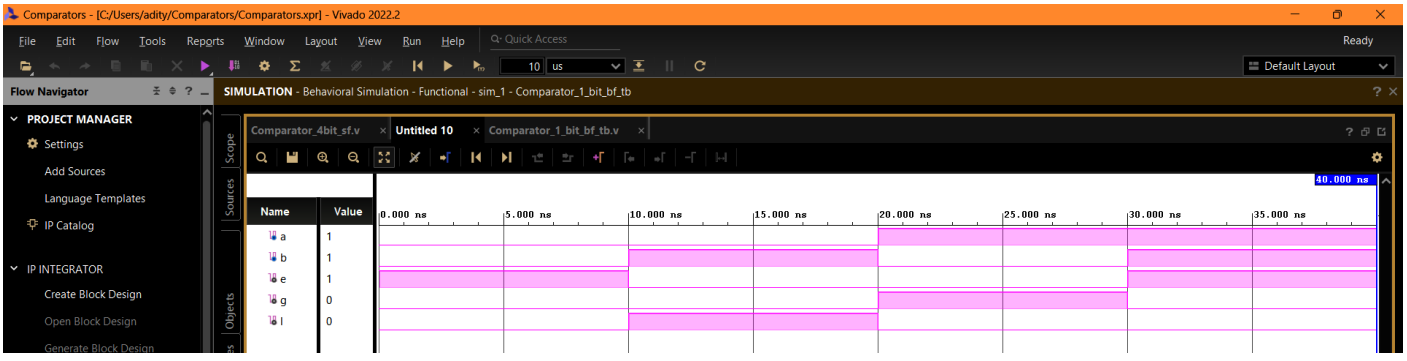
1.7 VERILOG CODE (Comparator_1_bit_bf.v):

```
module Comparator_1_bit_bf(e,g,l,a,b);
output e,g,l;
input a,b;
reg e,g,l;
always@(a,b)
case({a,b})
2'b00:begin e=1;g=0;l=0; end
2'b01:begin e=0;g=0;l=1; end
2'b10:begin e=0;g=1;l=0; end
2'b11:begin e=1;g=0;l=0; end
default:begin e=0;g=0;l=0; end
endcase
endmodule
```

1.8 TEST BENCH (Comparator_1_bit_bf _tb.v):

```
module Comparator_1_bit_df_tb();
reg a,b;
wire e,g,l;
Comparator_1_bit_bf x1(e,g,l,a,b);
initial
begin
a=1'b0;b=1'b0;
#10 a=1'b0;b=1'b1;
#10 a=1'b1;b=1'b0;
#10 a=1'b1;b=1'b1;
#10 $finish;
end
endmodule
```

1.9 WAVEFORM:



1.10 RESULT:

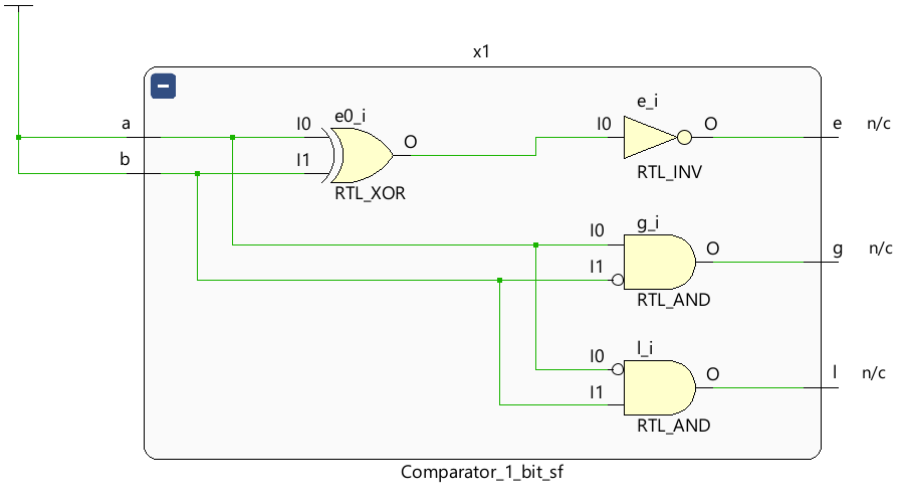
1 Bit Comparator is simulated and implemented in Behavioral Flow Modeling.

Comparators(Structural Flow Modeling)

1.1 AIM: 1 Bit Comparator

1.2 SOFTWARE USED: Xilinx Vivado 2022.2

1.3 SYMBOL:



1.4 LOGIC EXPRESSION:

E = ~ (A^B)
G = (~B) & A
L = (~A) & B

1.5 BOOLEAN EXPRESSION:

$E = \bar{A}\bar{B} + AB$
$G = A\bar{B}$
$L = \bar{A}B$

1.6 TRUTH TABLE:

INPUT		OUTPUT		
A	**B**	**E[$\sum_m$=(0,3)]**	**G[$\sum_m$=(2)]**	**L[$\sum_m$=(1)]**
0	**0**	**1**	**0**	**0**
0	**1**	**0**	**0**	**1**
1	**0**	**0**	**1**	**0**
1	**1**	**1**	**0**	**0**

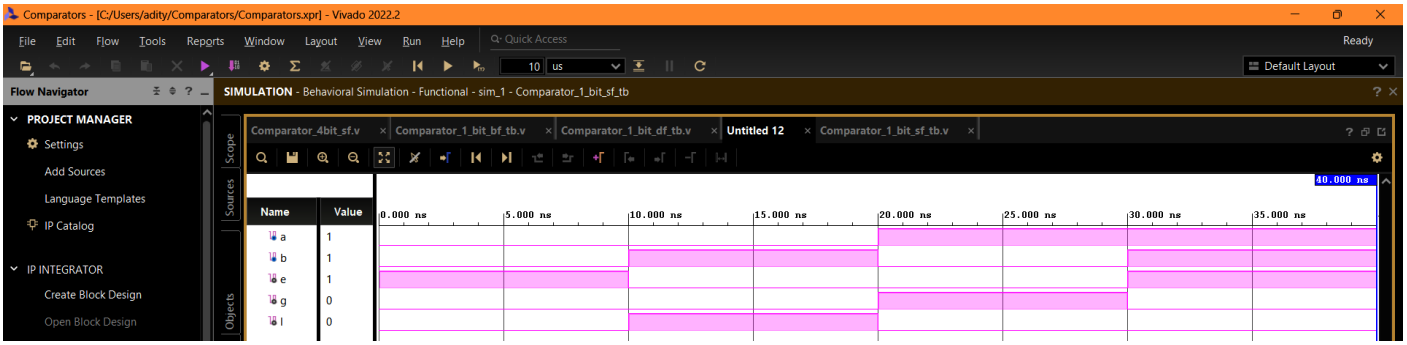
1.7 VERILOG CODE (Comparator_1_bit_sf.v):

```
module Comparator_1_bit_sf(e,g,l,a,b);
output e,g,l;
input a,b;

xnor (e,a,b);
and (g,a,~b);
and (l,~a,b);
endmodule
```

1.8 TEST BENCH (Comparator_1_bit_sf _tb.v):

```
module Comparator_1_bit_sf_tb();
reg a,b;
wire e,g,l;
Comparator_1_bit_sf x1(e,g,l,a,b);
initial
begin
a=1'b0;b=1'b0;
#10 a=1'b0;b=1'b1;
#10 a=1'b1;b=1'b0;
#10 a=1'b1;b=1'b1;
#10 $finish;
end
endmodule
```

1.9 WAVEFORM:



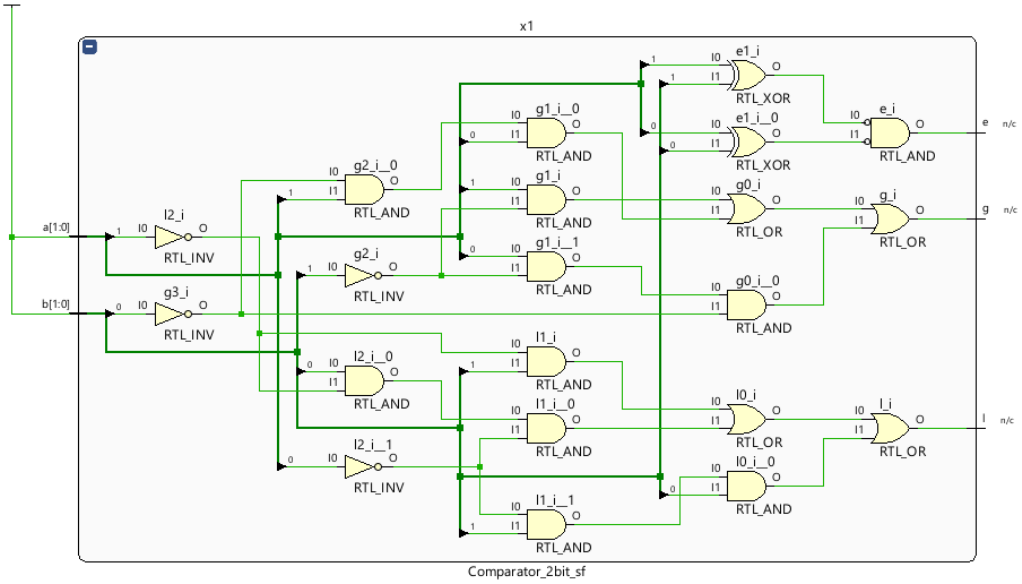
1.10 RESULT:

1 Bit Comparator is simulated and implemented in Structural Flow Modeling.

2.1 AIM: 2 Bit Comparator

2.2 SOFTWARE USED: Xilinx Vivado 2022.2

2.3 SYMBOL:



2.4 LOGIC EXPRESSION:

E = (~(A1^B1)) & (~(A0^B0))
G = (A1& (~B1)) | ((~B0) & A1&A0) | (A0& (~B1)&(~B0)))
L = (B1& (~A1)) | (B0 & (~A1)&(~A0)) | ((~A0)& B1&B0))

2.5 BOOLEAN EXPRESSION:

$E = (\overline{A1}\,\overline{B1}+A1B1)(\overline{A0}\,\overline{B0}+A0B0)$
$G = A1\overline{B1} + A0A1\overline{B0} + A0\overline{B1}\,\overline{B0}$
$L = \overline{A1}B1 + \overline{A0}\,\overline{A1}B0 + \overline{A0}B1B0$

2.6 TRUTH TABLE:

INPUT				OUTPUT		
A1	**A0**	**B1**	**B0**	**E**	**G**	**L**
0	**0**	**0**	**0**	**1**	**0**	**0**
0	**0**	**0**	**1**	**0**	**0**	**1**
0	**0**	**1**	**0**	**0**	**0**	**1**
0	**0**	**1**	**1**	**0**	**0**	**1**
0	**1**	**0**	**0**	**0**	**1**	**0**
0	**1**	**0**	**1**	**1**	**0**	**0**
0	**1**	**1**	**0**	**0**	**0**	**1**
0	**1**	**1**	**1**	**0**	**0**	**1**
1	**0**	**0**	**0**	**0**	**1**	**0**
1	**0**	**0**	**1**	**0**	**1**	**0**
1	**0**	**1**	**0**	**1**	**0**	**0**
1	**0**	**1**	**1**	**0**	**0**	**1**
1	**1**	**0**	**0**	**0**	**1**	**0**
1	**1**	**0**	**1**	**0**	**1**	**0**
1	**1**	**1**	**0**	**0**	**1**	**0**
1	**1**	**1**	**1**	**1**	**0**	**0**

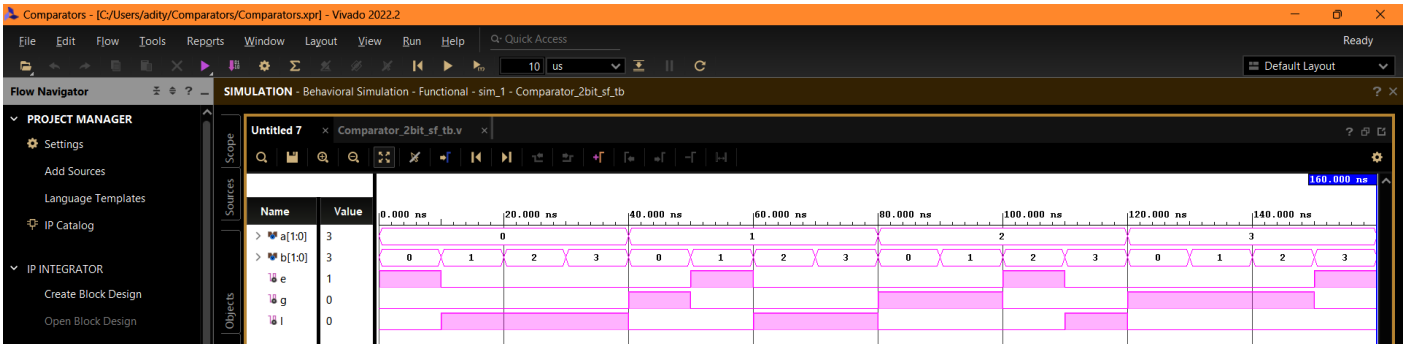
2.7 VERILOG CODE (Comparator_2bit_sfv):

```
module Comparator_2bit_sf(e,g,l,a,b);
output e,g,l;
input [1:0]a;
input [1:0]b;
wire e,g,l;
or (g,(a[1]&(~b[1])),((~b[0])&a[1]&a[0]),(a[0]&(~b[1])&(~b[0])));
or (l,(~(a[1])&b[1]),(b[0]&(~a[1])&(~a[0])),((~a[0])&b[1]&b[0]));
and(e,(~(a[1]^b[1])),(~(a[0]^b[0])));
endmodule
```

2.8 TEST BENCH (Comparator_2bit_sf_tb.v):

```
module Comparator_2bit_sf_tb();
reg [1:0]a;
reg [1:0]b;
wire e,g,l;
Comparator_2bit_sf x1(e,g,l,a,b);
initial
begin
{a,b}=4'b0000;
#10 {a,b}=4'b0001;
#10 {a,b}=4'b0010;
#10 {a,b}=4'b0011;
#10 {a,b}=4'b0100;
#10 {a,b}=4'b0101;
#10 {a,b}=4'b0110;
#10 {a,b}=4'b0111;
#10 {a,b}=4'b1000;
#10 {a,b}=4'b1001;
#10 {a,b}=4'b1010;
#10 {a,b}=4'b1011;
#10 {a,b}=4'b1100;
#10 {a,b}=4'b1101;
#10 {a,b}=4'b1110;
#10 {a,b}=4'b1111;
#10 $finish ;
end
endmodule
```

2.9 WAVEFORM:



2.10 RESULT:

2 Bit Comparator is simulated and implemented in Structural Flow Modeling.