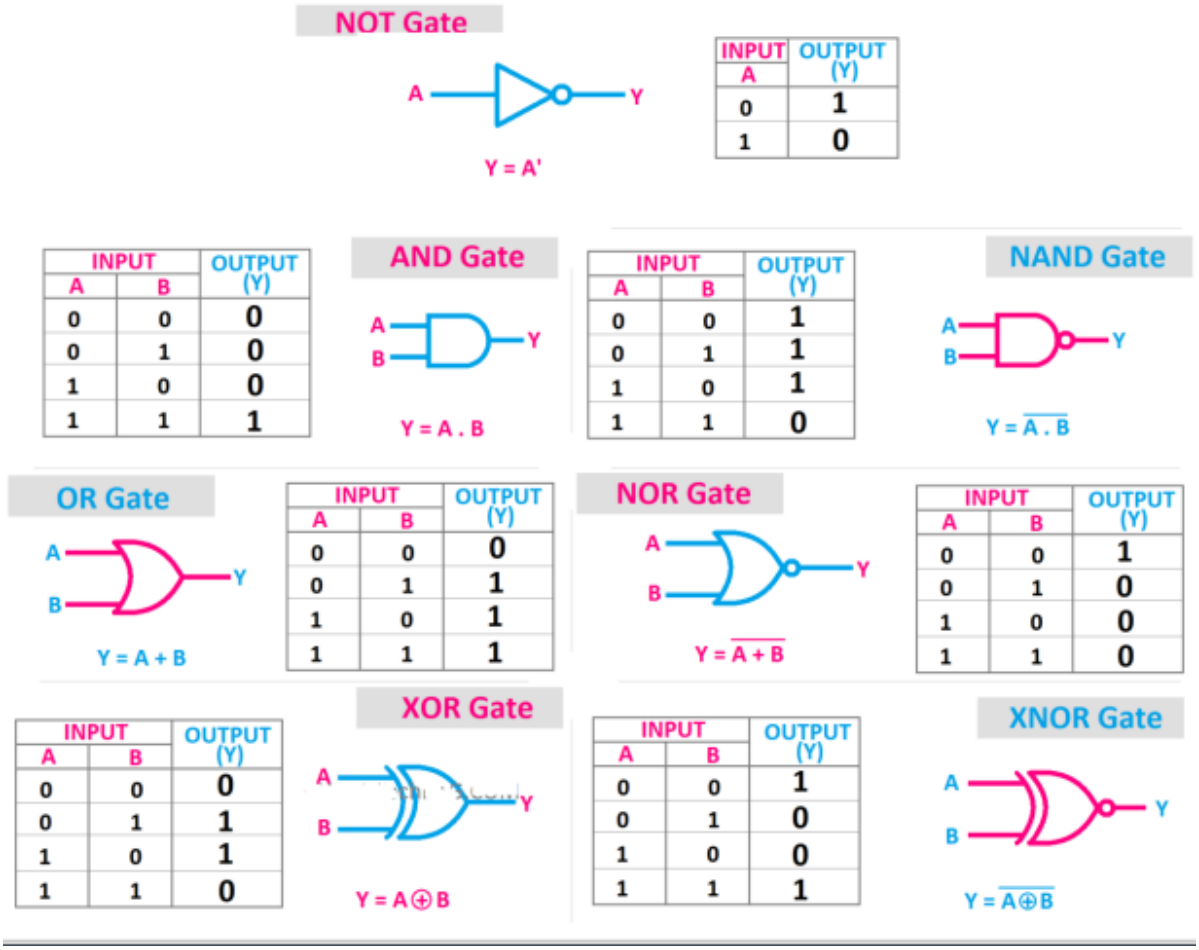
LOGIC GATES

AIM:

To simulate and synthesis Logic Gates using Xilinx ISE.

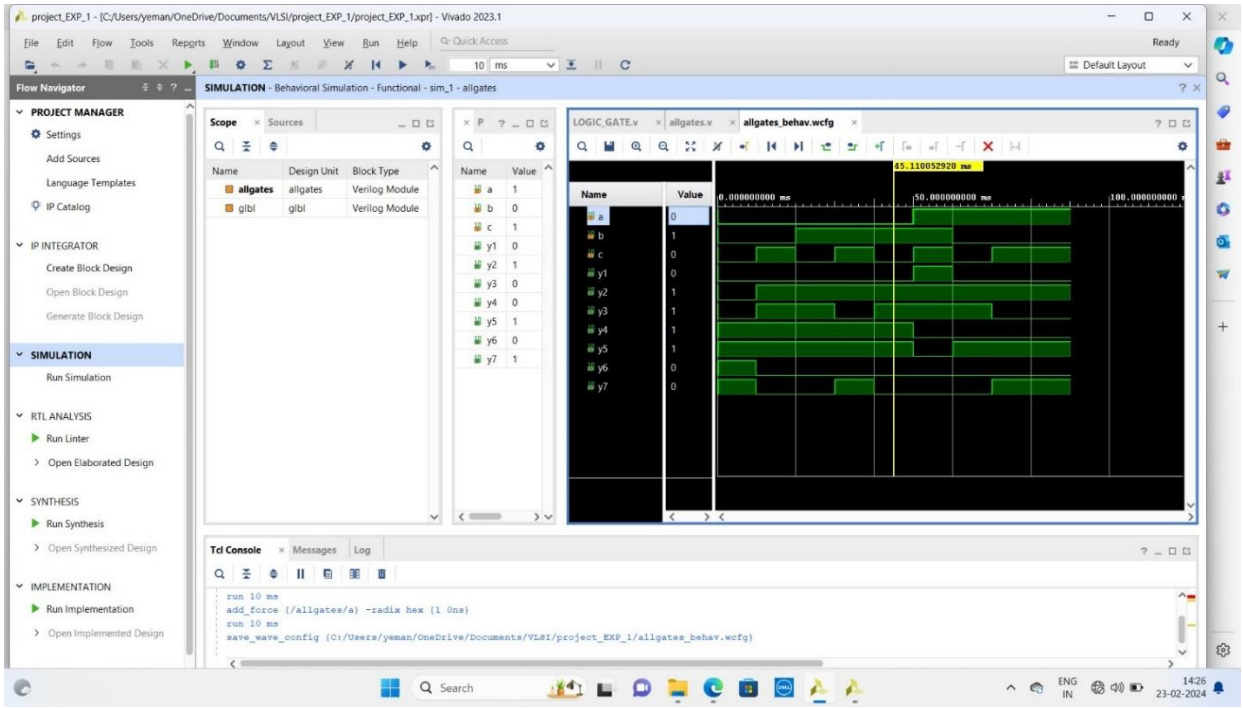
LOGIC DIAGRAM:



VERILOG CODE:

```
module allgates(a,b,c,y1,y2,y3,y4,y5,y6,y7);
input a,b,c;
output y1,y2,y3,y4,y5,y6,y7;
and x1(y1,a,b,c);
or x2(y2,a,b,c);
xor x3(y3,a,b,c);
not x4(y4,a);
nand x5(y5,a,b,c);
nor x6(y6,a,b,c);
xnor x7(y7,a,b,c);
endmodule
```

OUTPUT:



RESULT:

Thus The Simulation And Synthesis Of Logic Gates using Xilinx ISE are Verified Successfully.

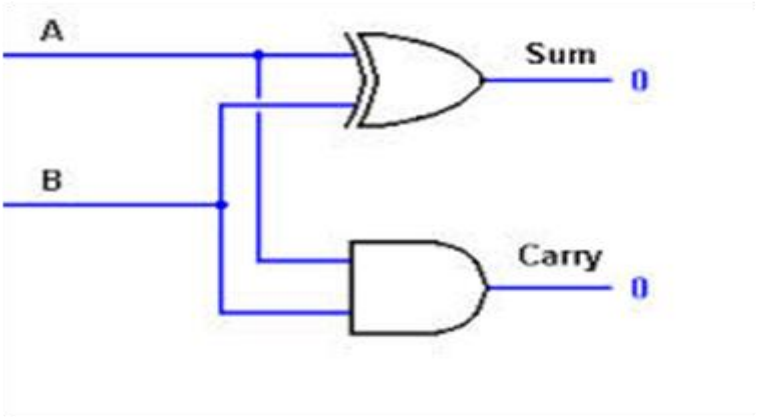
ADDERS

AIM:

To simulate and synthesis Adders using Xilinx ISE.

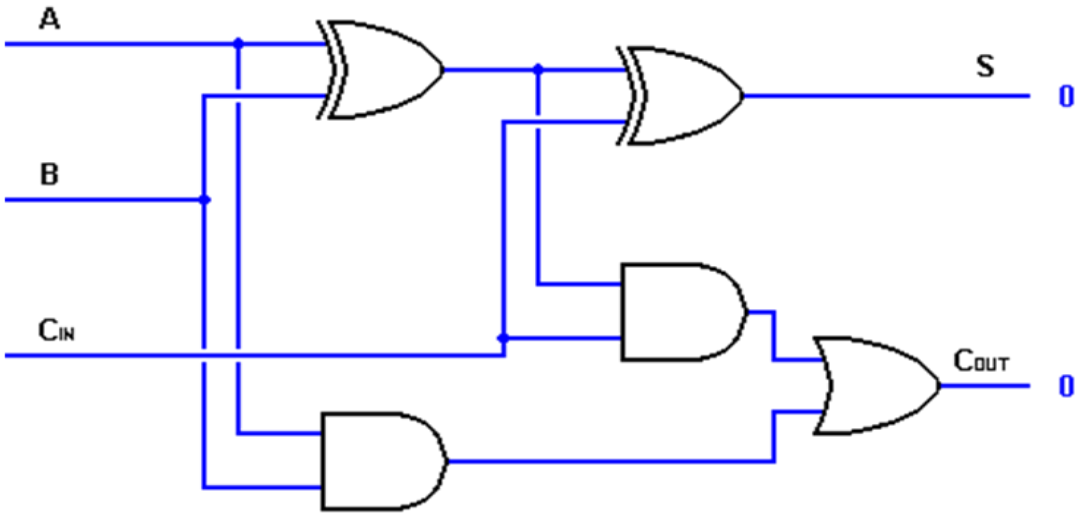
LOGIC DIAGRAM:

Half Adder



:

Full adder:



VERILOG CODE:

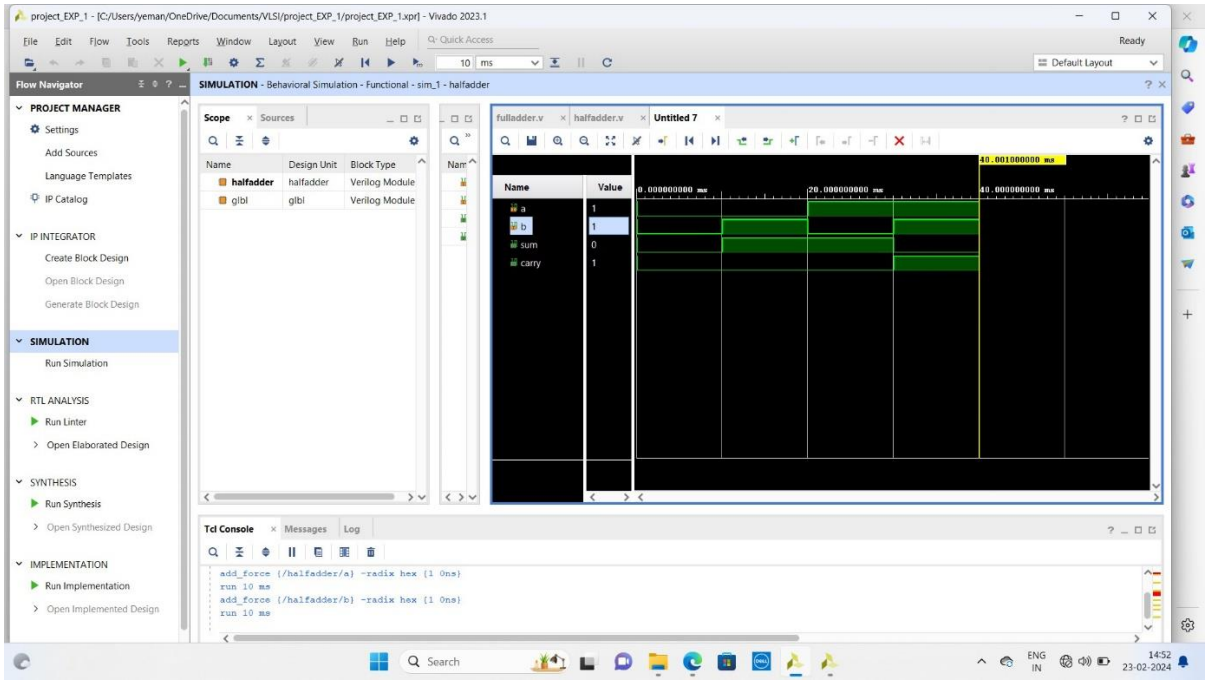
\\HALFADDER

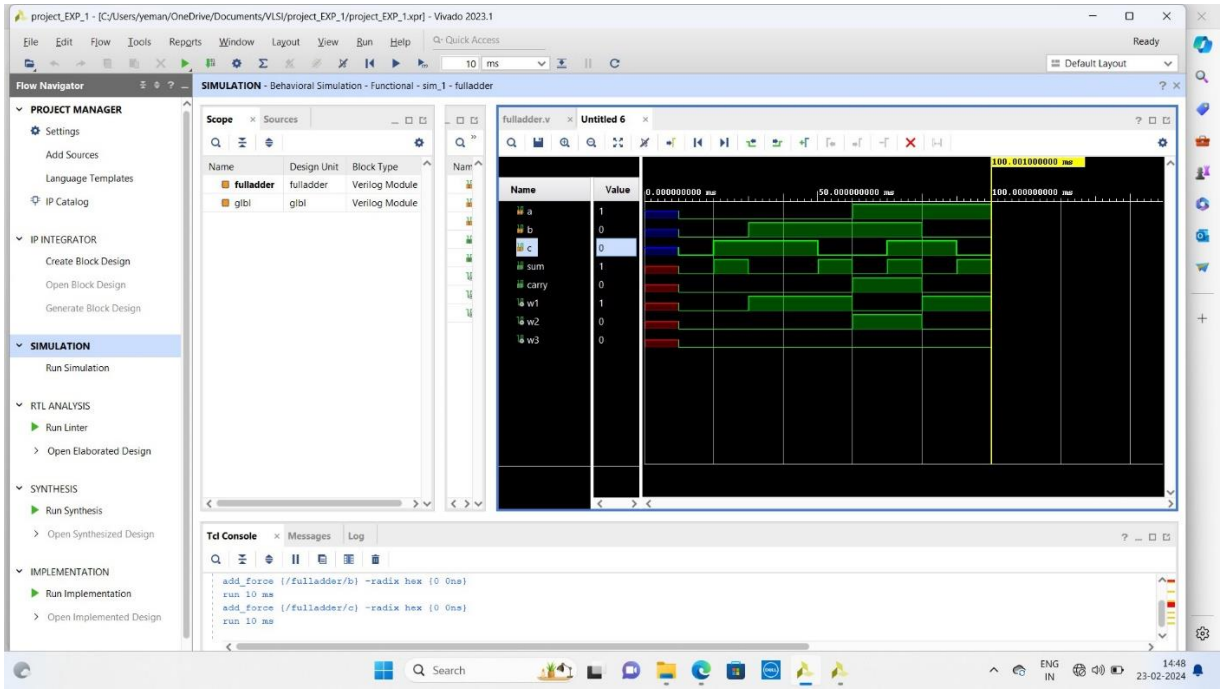
```
module halfadder(a,b,sum,carry);
input a,b;
output sum,carry;
xor x1(sum,a,b);
and x2(carry,a,b);
endmodule
```

\\FULLADDER

```
module fulladder(a,b,c,sum,carry);
input a,b,c;
output sum,carry;
wire w1,w2,w3;
xor x1(w1,a,b);
xor x2(sum,w1,c);
and x3(w2,a,b);
and x4(w3,w1,w2);
or x5(carry,w3,w2);
endmodule
```

OUTPUT:





RESULT:

Thus The Simulation And Synthesis Of Adders using Xilinx ISE are Verified Successfully.

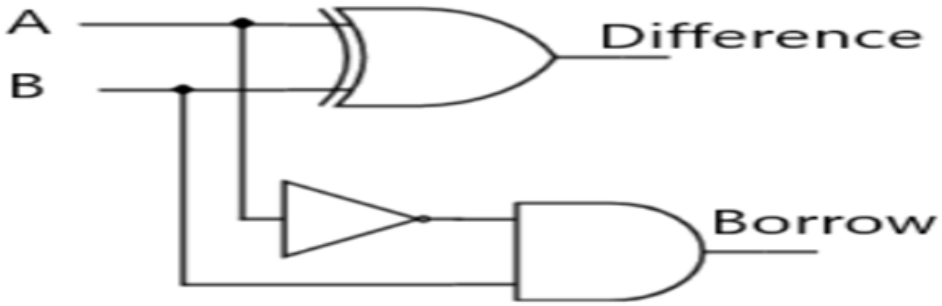
SUBTRACTOR

AIM:

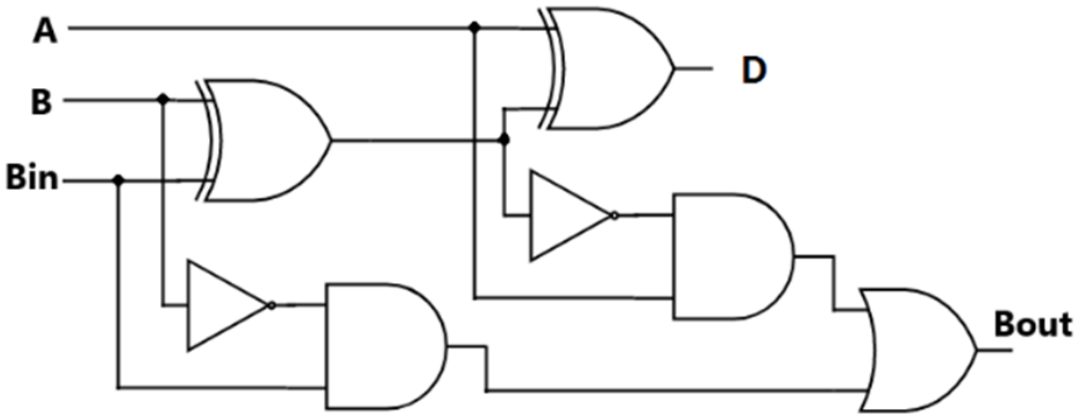
To simulate and synthesis Subtractor using Xilinx ISE.

LOGIC DIAGRAM

Half Subtractor:



Full Subtractor:



VERILOG CODE

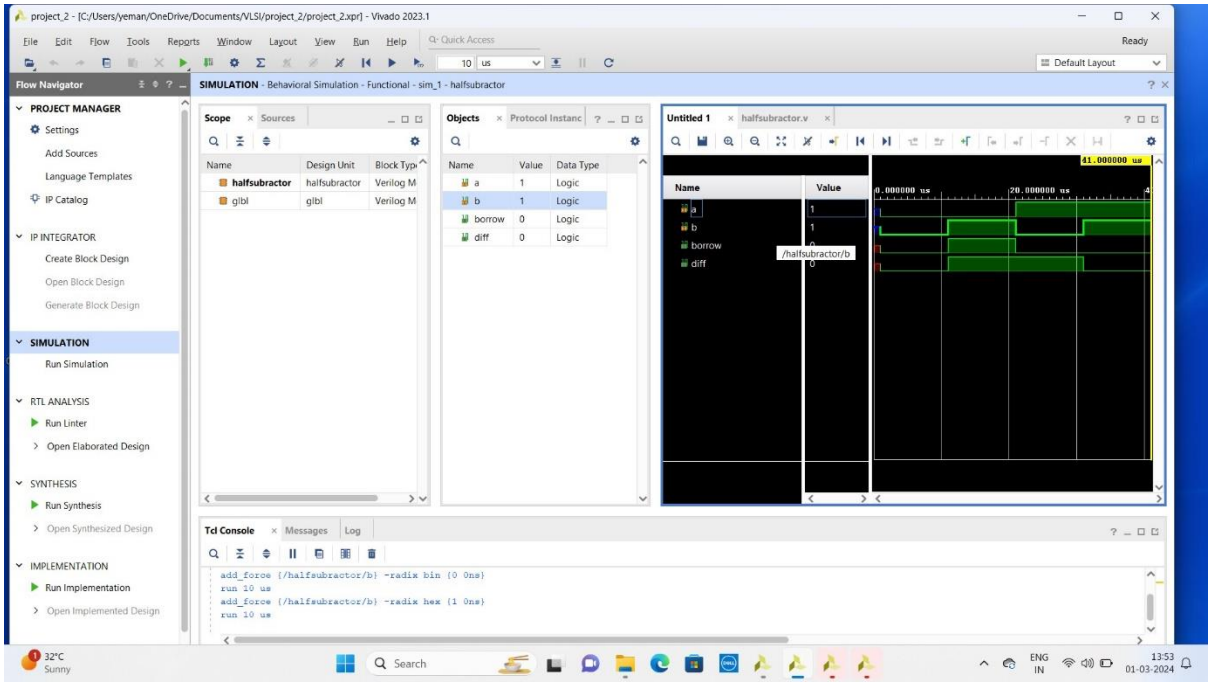
\\HALFSUBRACTOR

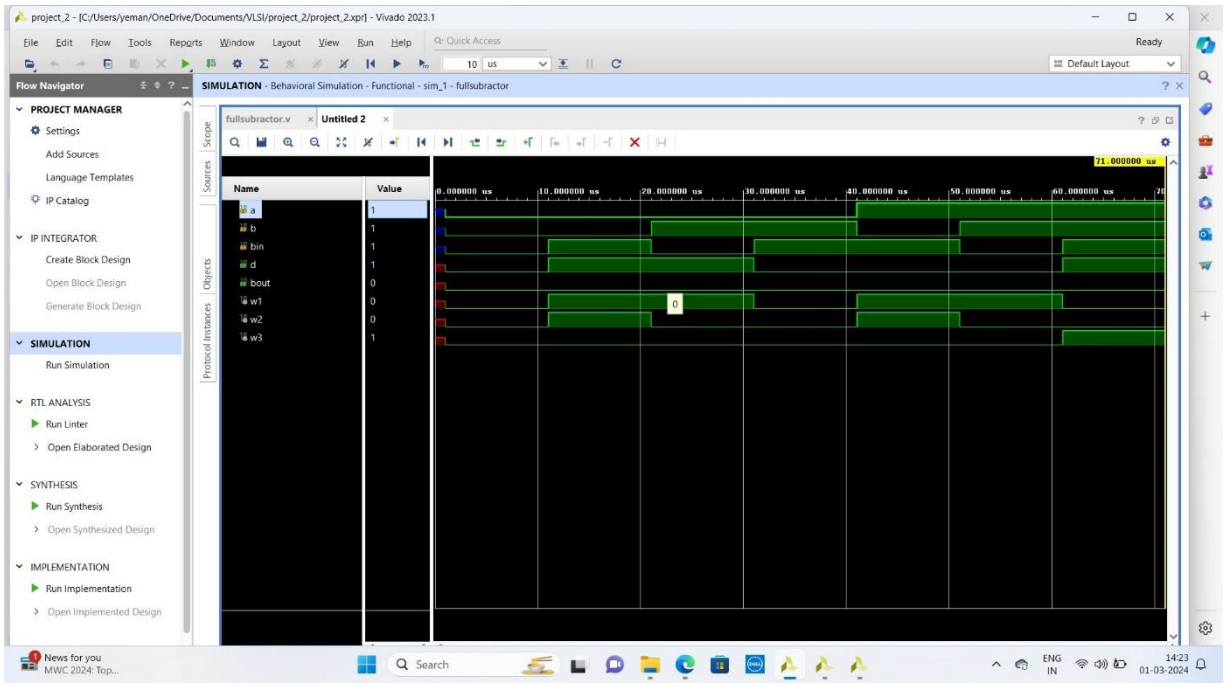
```
module halfsubractor(a,b,borrow,diff);
input a,b;
output borrow,diff;
xor g1(diff,a,b);
and g2(borrow,~a,b);
endmodule
```

\\FULLSUBRACTOR

```
module fullsubractor(a,b,bin,d,bout);
input a,b,bin;
output d,bout;
wire w1,w2,w3;
xor (d,a,w1);
xor(w1,b,bin);
and(w2,~b,bin);
and (w3,~w1,a);
or (bout,w1,w2);
endmodule
```

OUTPUT





RESULT

Thus The Simulation And Synthesis Of Subtractor using Xilinx ISE are Verified Successfully.

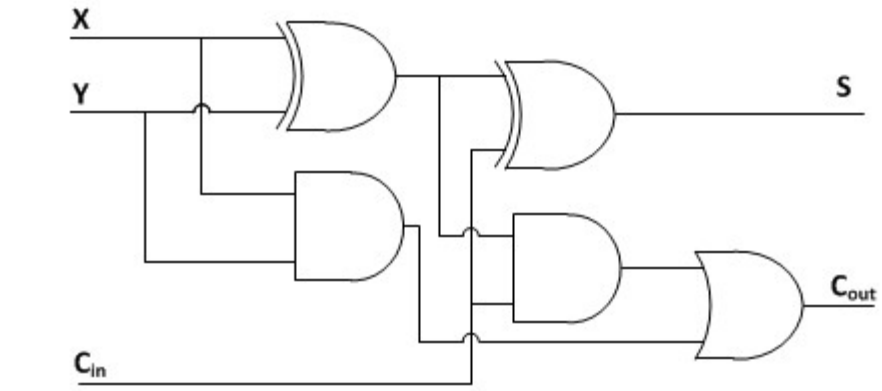
RIPPLE CARRY ADDER

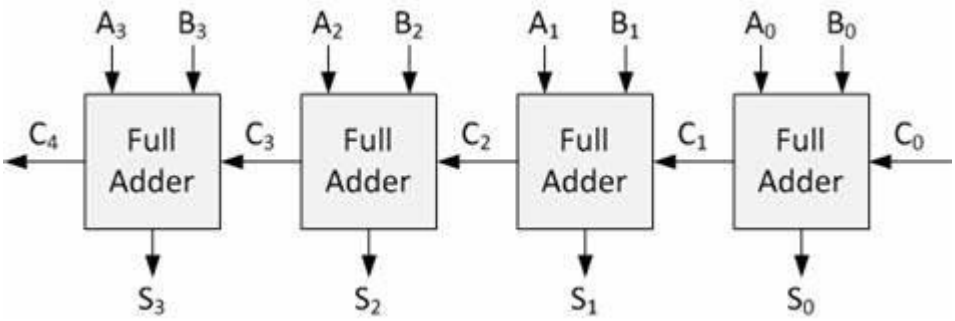
AIM:

To simulate and synthesis Ripple Carry Adder using Xilinx ISE.

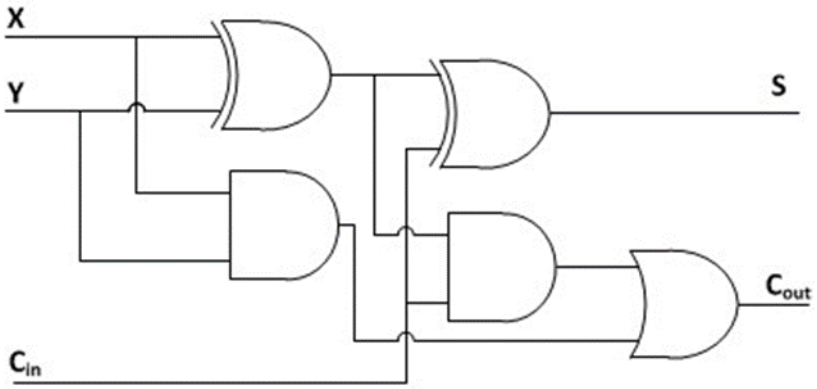
LOGIC DIAGRAM

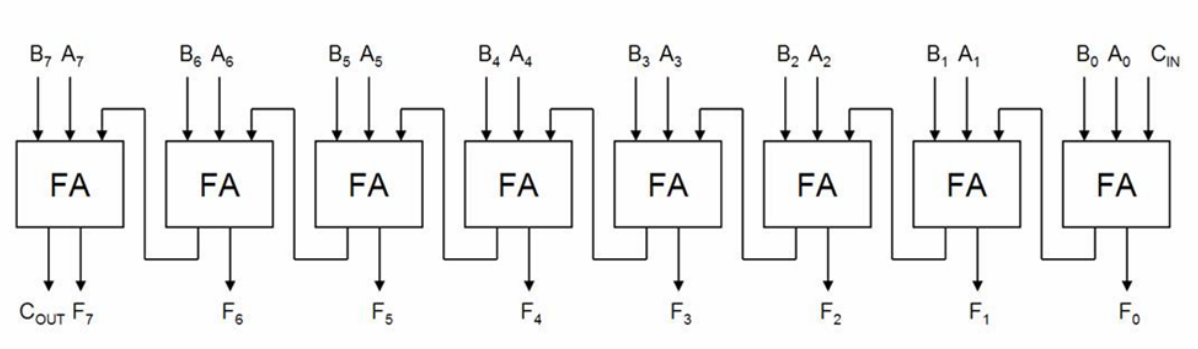
4 Bit Ripple Carry Adder:





8 Bit Ripple Carry Adder:





VERILOG CODE

\\4 Bit Ripple Carry Adder

```
module fulladder(a,b,c,sum,carry);
input a,b,c;
output sum,carry;
wire w1,w2,w3;
xor x1(w1,a,b);
xor x2(sum,w1,c);
and x3(w2,a,b);
and x4(w3,w1,w2);
```

```
or x5(carry,w3,w2);
endmodule
module ripplecarry(a,b,cin,sum,cout);
input [3:0]a,b;
input cin;
output [3:0]sum;
output cout;
wire c1,c2,c3;
fulladder(a[0],b[0],cin,sum[0],c1);
fulladder(a[1],b[1],c1,sum[1],c2);
fulladder(a[2],b[2],c2,sum[2],c3);
fulladder(a[3],b[3],c3,sum[3],c4);
endmodule
```

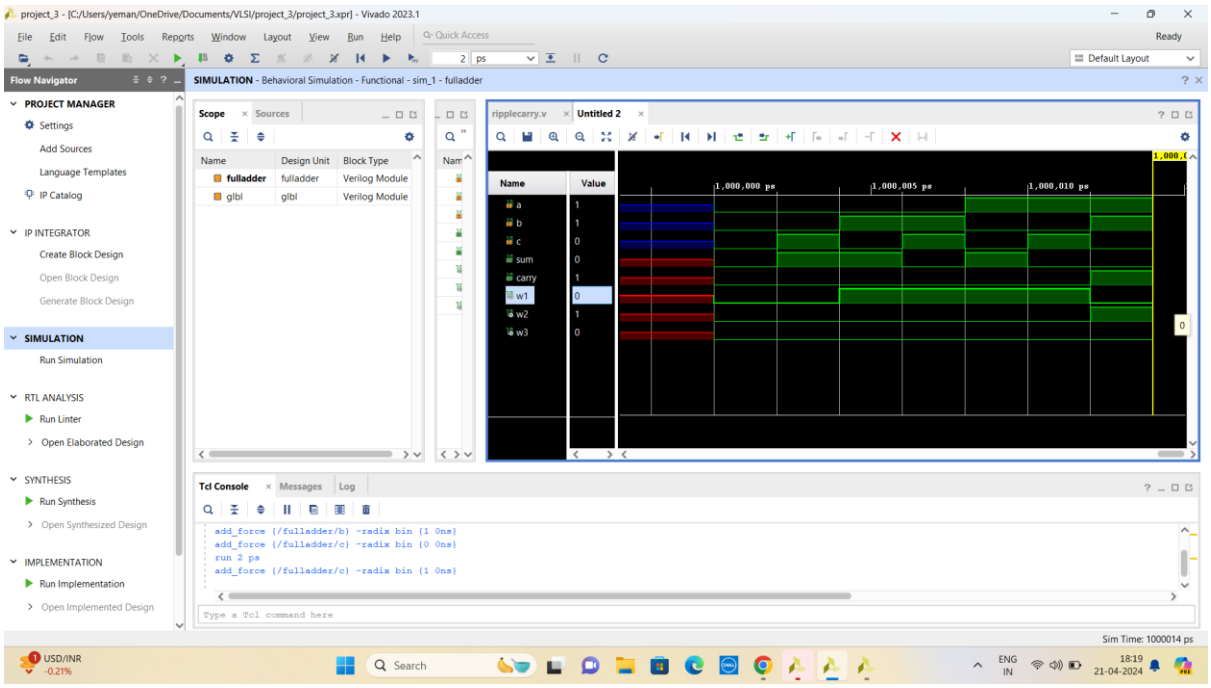
\\ 8 Bit Ripple Carry Adder\\

```
module fulladder(a,b,c,sum,carry);
input a,b,c;
output sum,carry;
wire w1,w2,w3;
xor x1(w1,a,b);
xor x2(sum,w1,c);
and x3(w2,a,b);
and x4(w3,w1,w2);
or x5(carry,w3,w2);
endmodule
module ripplecarry(a,b,cin,sum,cout);
input [7:0]a,b;
input cin;
output [7:0]sum;
output cout;
```

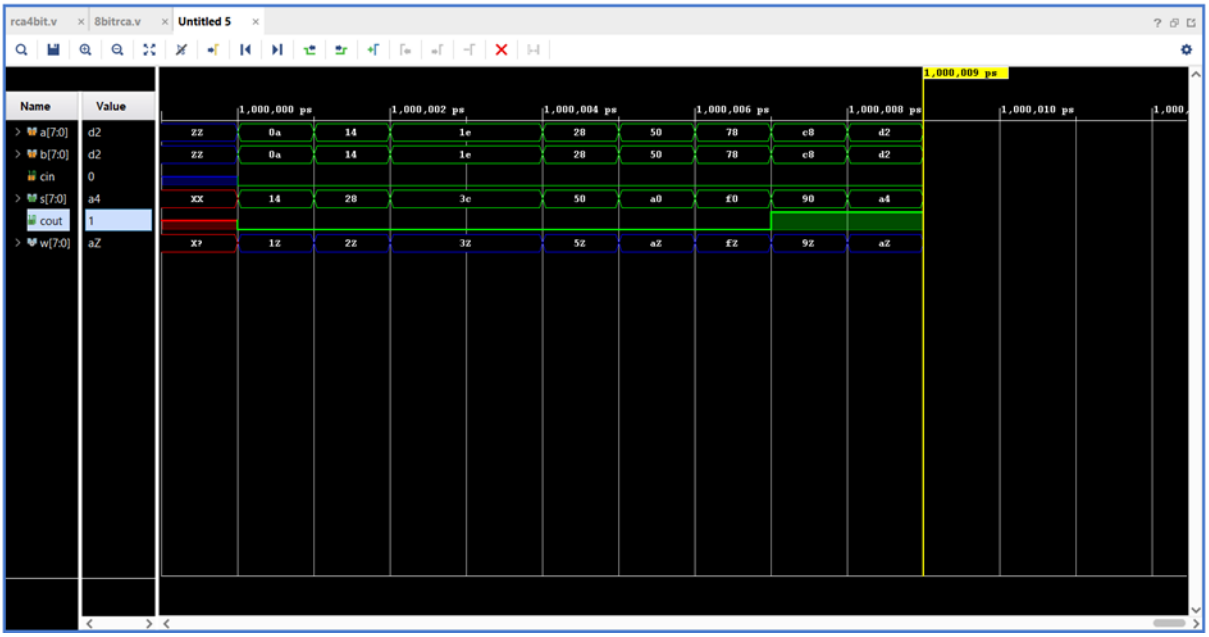
```
wire c1,c2,c3;
fulladder(a[0],b[0],cin,sum[0],c1);
fulladder(a[1],b[1],c1,sum[1],c2);
fulladder(a[2],b[2],c2,sum[2],c3);
fulladder(a[3],b[3],c3,sum[3],c4);
fulladder(a[4],b[4],c4,sum[4],c5);
fulladder(a[5],b[5],c5,sum[5],c6);
fulladder(a[6],b[6],c6,sum[6],c7);
fulladder(a[7],b[7],c7,sum[7],c8);
fulladder(a[8],b[8],c8,sum[8],cout);
endmodule
```

OUTPUT

\\4 Bit Ripple Carry Adder\\



\\ 8 Bit Ripple Carry Adder\\



PROCEDURE:

STEP:1 Start the Xilinx navigator, Select and Name the New project.

STEP:2 Select the device family, device, package and speed.
STEP: 3 Select new source in the New Project and select Verilog Module as the Source type.

STEP:4 Type the File Name and Click Next and then finish button. Type the code and save it.

STEP:5 Select the Behavioral Simulation in the Source Window and click the check syntax.

STEP:6 Click the simulation to simulate the program and give the inputs and verify the outputs as per the truth table.

STEP:7 Select the Implementation in the Sources Window and select the required file in the Processes Window.

STEP:8 Select Check Syntax from the Synthesize XST Process. Double Click in the Floorplan Area/IO/Logic-Post Synthesis process in the User Constraints process group. UCF(User constraint File) is obtained.

STEP:9 In the Design Object List Window, enter the pin location for each pin in the Loc column Select save from the File menu.

STEP:10 Double click on the Implement Design and double click on the Generate Programming File to create a bitstream of the design.(.v) file is converted into .bit file here. STEP:12 Load the Bit file into the SPARTAN 6 FPGA

STEP:11 On the board, by giving required input, the LEDs starts to glow light, indicating the output.

RESULT

Thus The Simulation And Synthesis Of The Ripple Carry Adder using Xilinx ISE are Verified Successfully.