

**AIM:**

To simulate and synthesis Logic Gates, Adders and Subtractors using Xilinx ISE.

**APPARATUS REQUIRED:**

Xilinx 14.7 Spartan6 FPGA

**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 Behavioural 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.

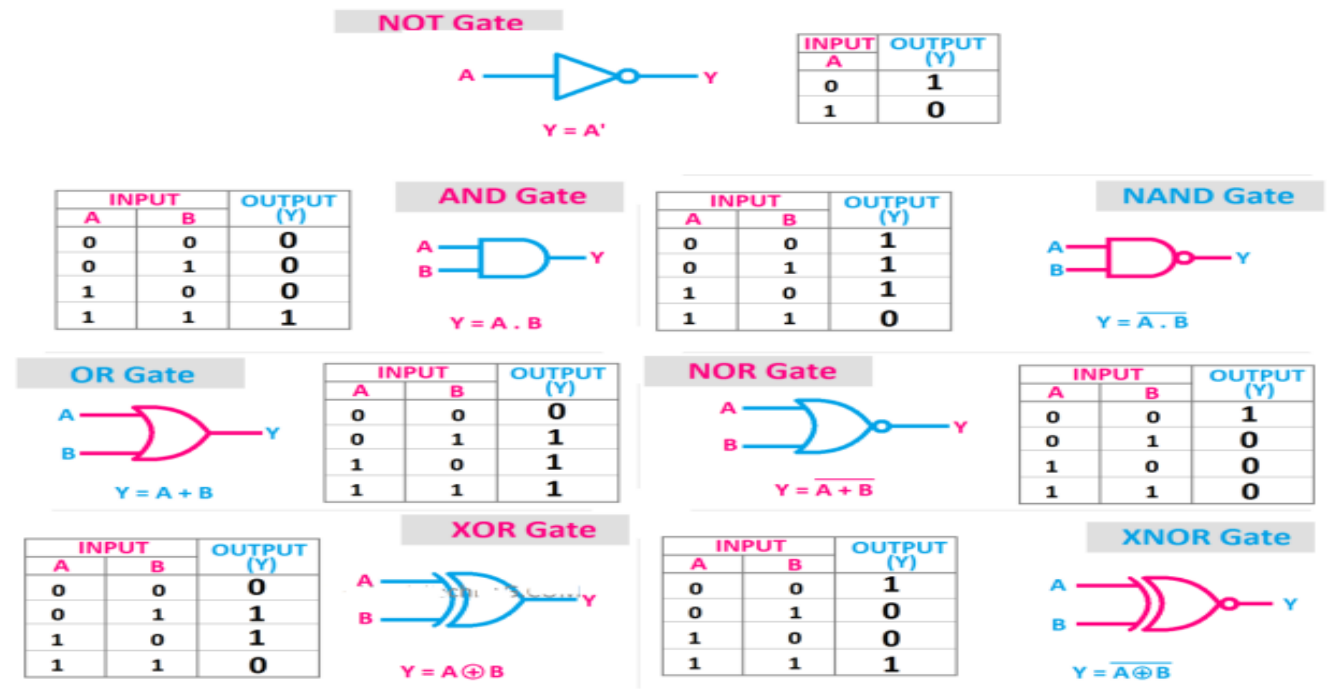
STEP:9 In the Design Object List Window, enter the pin location for each pin in the Loc column Select save from the File menu.

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:11 Load the Bit file into the SPARTAN 6 FPGA

STEP:12 On the board, by giving required input, the LEDs starts to glow light, indicating the output.

## LOGIC DIAGRAM:

### LOGIC GATES:



### VERILOG CODE:

```
module allgates(a,b,x1,x2,x3,x4,x5,x6,x7);
```

```
input a,b;
```

```
output x1,x2,x3,x4,x5,x6,x7;
```

```
and g1(x1,a,b);
```

```
or g2(x2,a,b);
```

```
xor g3(x3,a,b);
```

```
nand g4(x4,a,b);
```

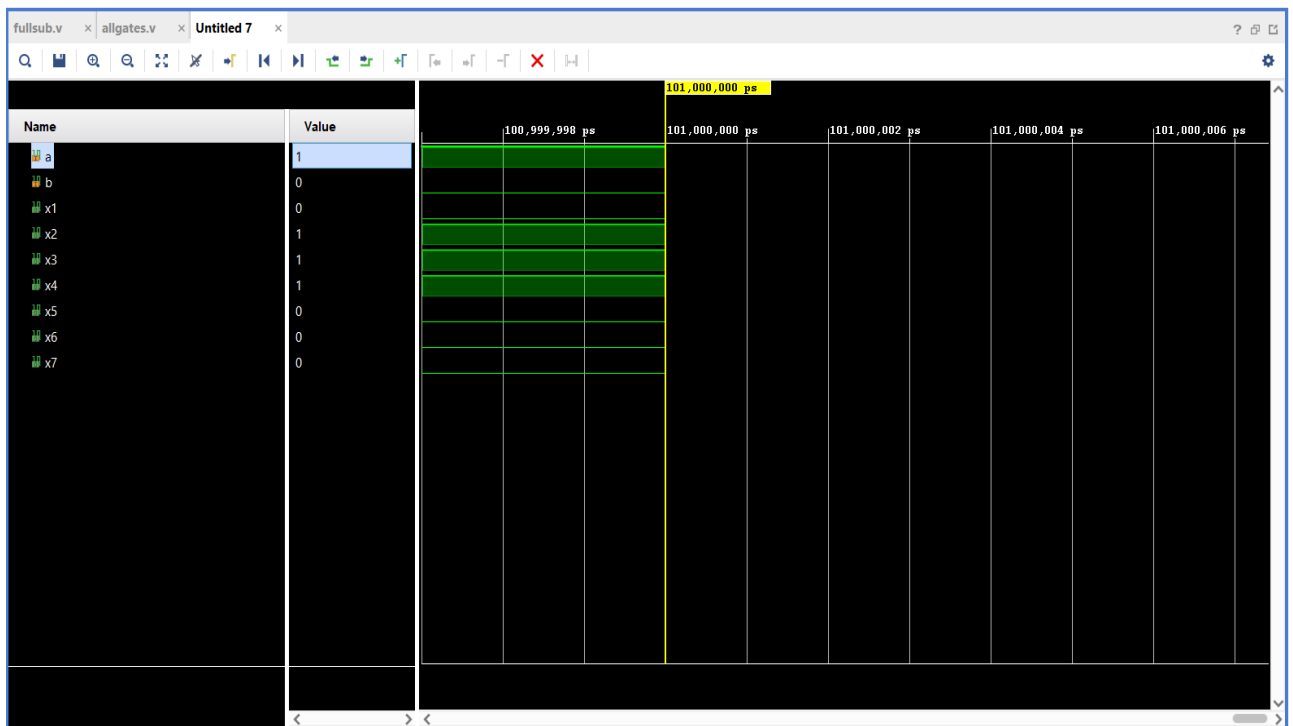
```
nor g5(x5,a,b);
```

```
xnor g6(x6,a,b);
```

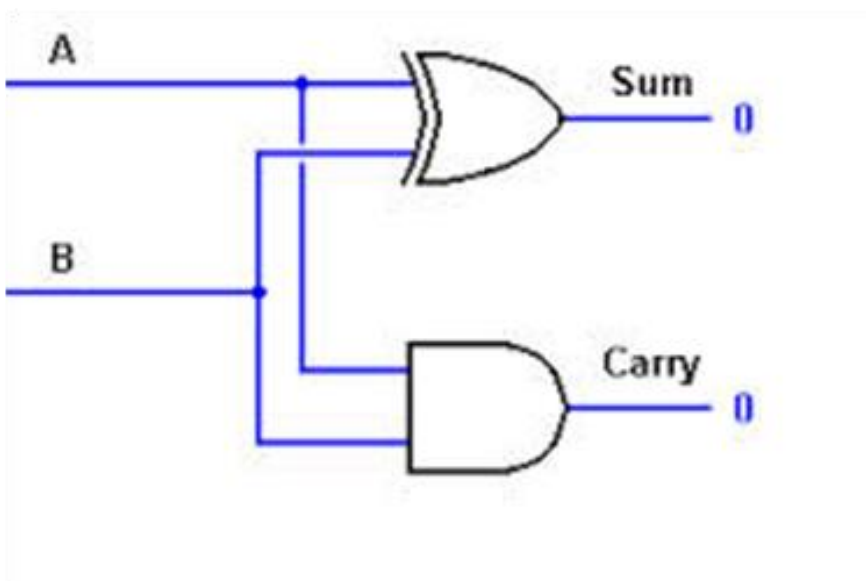
```
not g7(x7,a);
```

```
endmodule
```

OUTPUT:



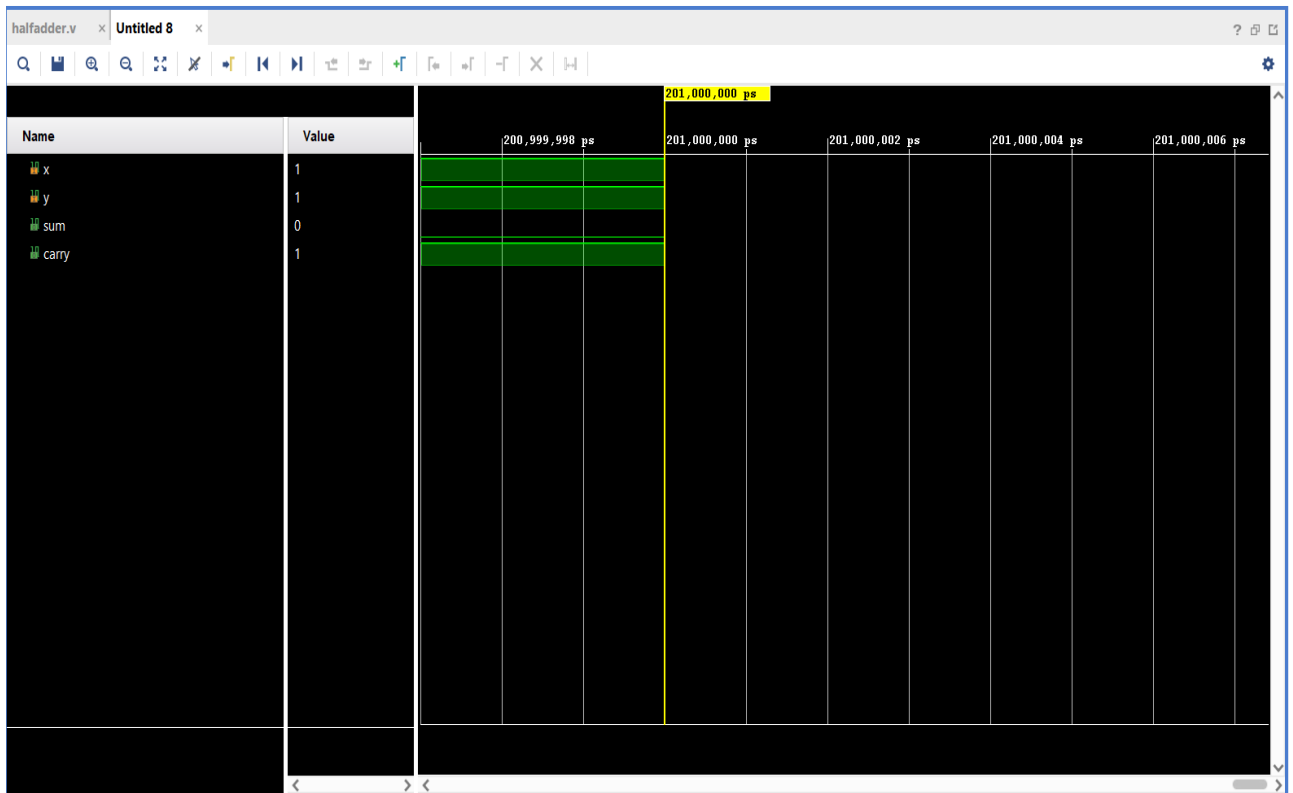
HALF ADDER:



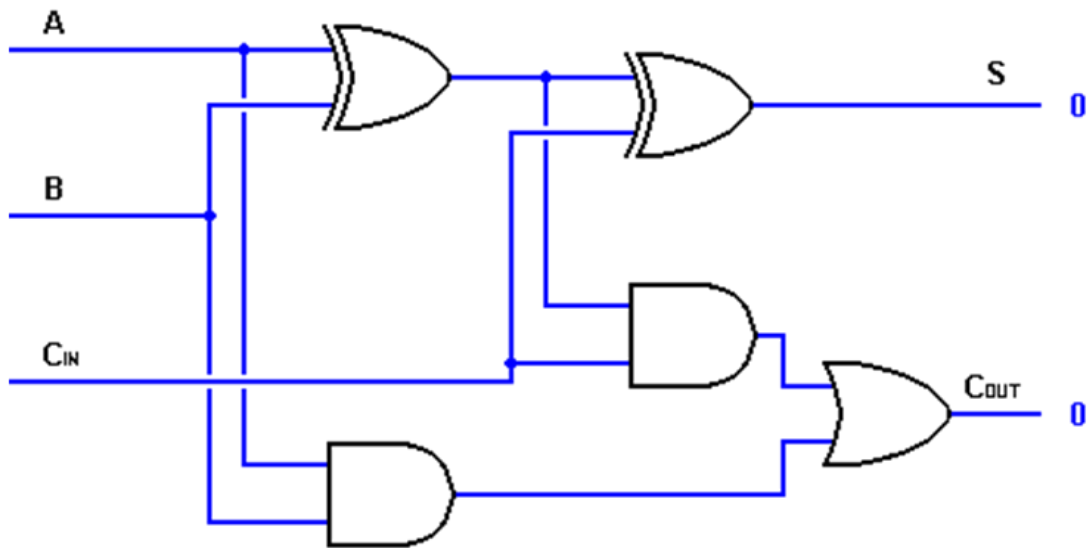
## VERILOG CODE:

```
module halfadder(x,y,sum,carry);  
  
input x,y;  
  
output sum,carry;  
  
xor g1(sum,x,y);  
  
and g2(carry,x,y);  
  
endmodule
```

## OUTPUT:



FULL ADDER:



VERILOG CODE:

```
module fulladder(a,b,cin,sum,carry);
```

```
input a,b,cin;
```

```
output sum,carry;
```

```
wire w1,w2,w3;
```

```
xor g1(w1,a,b);
```

```
xor g2(sum,w1,cin);
```

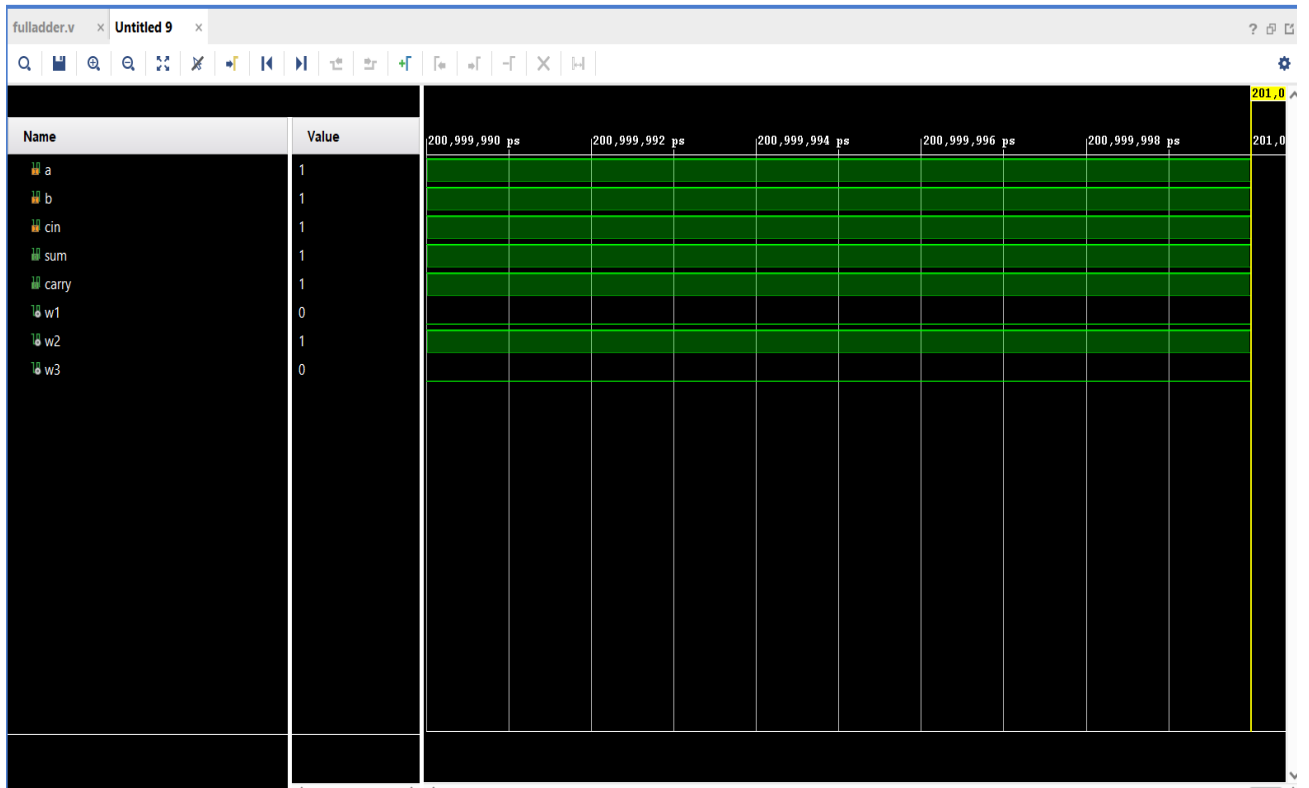
```
and g3(w2,a,b);
```

```
and g4(w3,w1,cin);
```

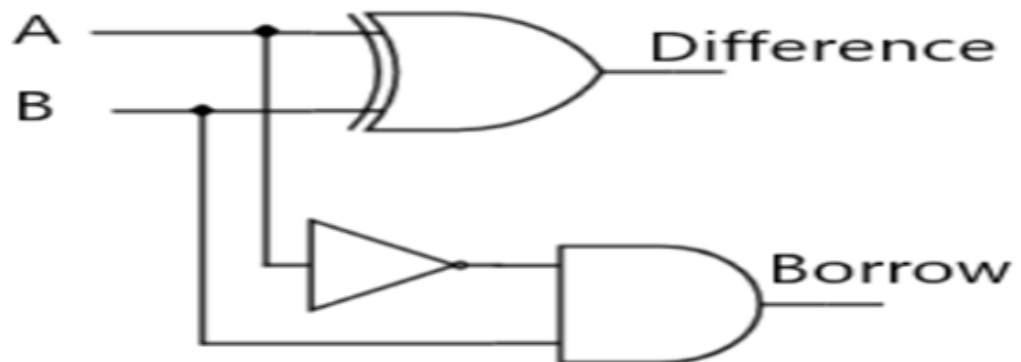
```
or g5(carry,w3,w2);
```

```
endmodule
```

OUTPUT:



HALF SUBTRACTOR:



## VERILOG CODE:

```
module halfsub(x,y,d,b);
```

```
input x,y;
```

```
output d,b;
```

```
wire x1;
```

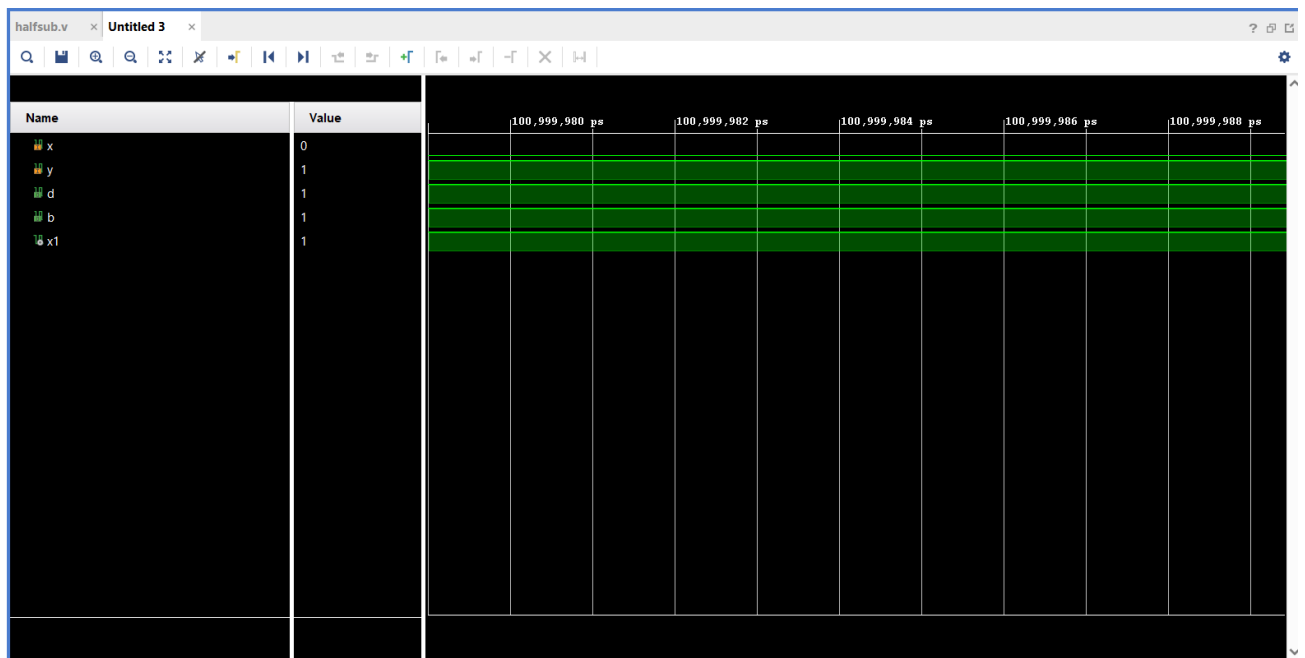
```
xor g1(d,x,y);
```

```
not g2(x1,x);
```

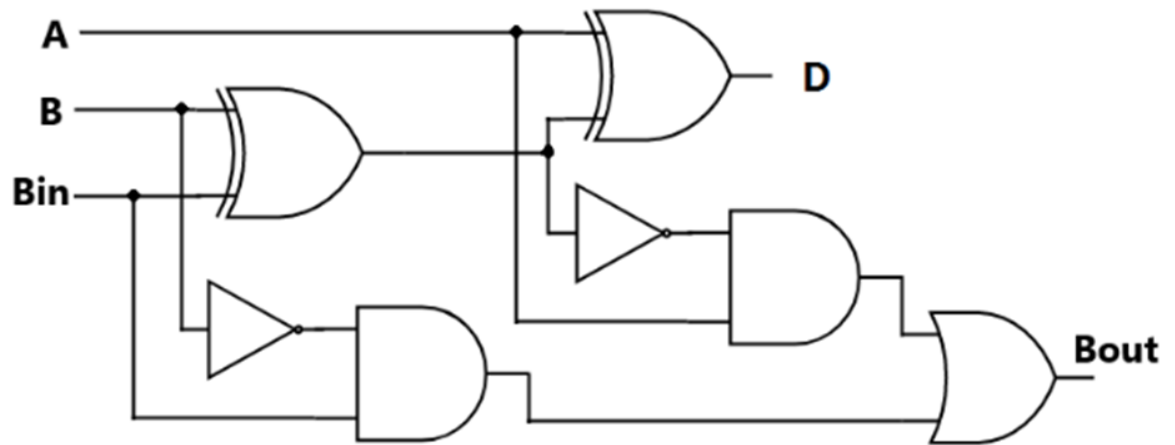
```
and g3(b,x1,y);
```

```
endmodule
```

## OUTPUT:



FULL SUBTRACTOR:



VERILOG CODE:

```
module fullsub(a,b,bin,bout,diff);
```

```
input a,b,bin;
```

```
output bout,diff;
```

```
wire w1,w2,w3,w4,w5;
```

```
xor g1(w3,a,b);
```

```
xor g2(diff,w3,bin);
```

```
not g3(w1,a);
```

```
not g4(w4,w3);
```

```
and g5(w2,b,w1);
```

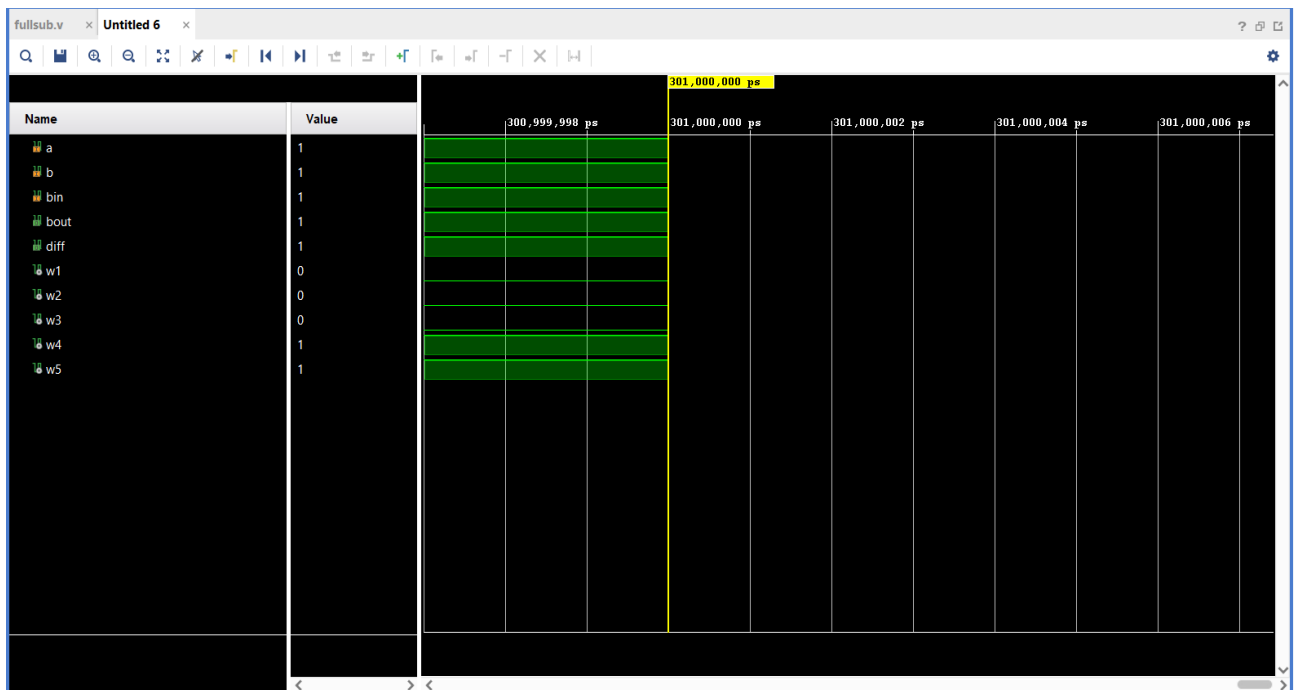
```
and g6(w5,bin,w4);
```

```
or g7(bout,w5,w2);
```

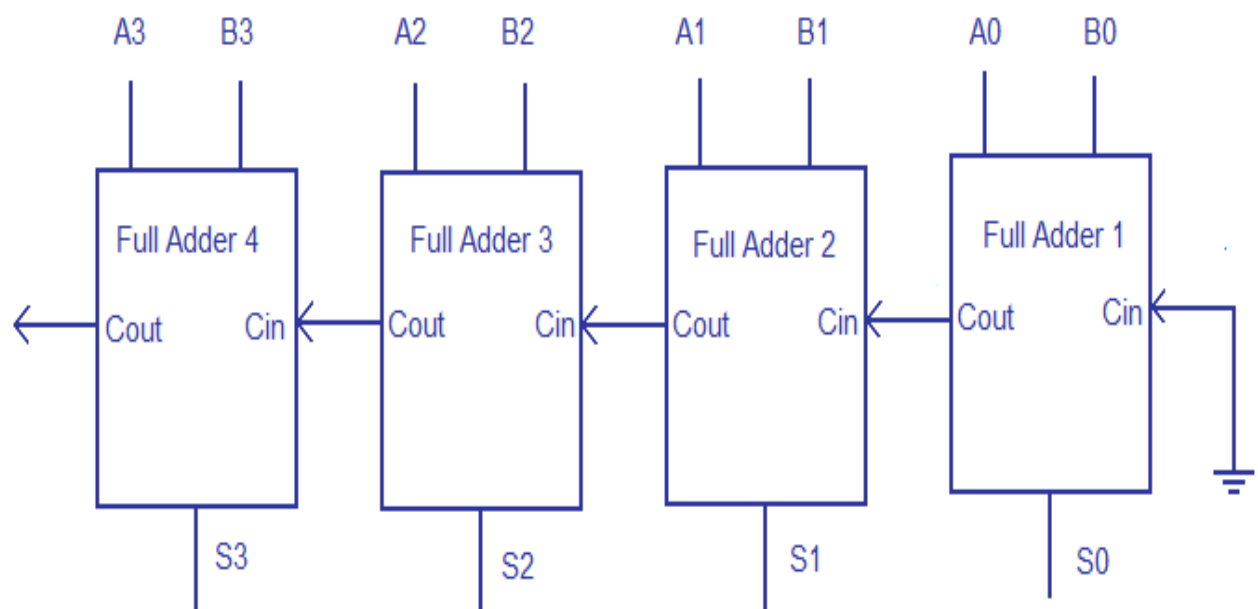
```
endmodule
```



OUTPUT:



4 BIT RIPPLE CARRY ADDER:



4 bit ripple carry adder

VERILOG CODE:

```
module fa(a,b,cin,sum,carry);
```

```
input a,b,cin;
```

```
output sum,carry;
```

```
wire w1,w2,w3;
```

```
xor g1(w1,a,b);
```

```
xor g2(sum,w1,cin);
```

```
and g3(w2,a,b);
```

```
and g4(w3,w1,cin);
```

```
or g5(carry,w3,w2);
```

```
endmodule
```

```
module rca(a,b,cin,sum,cout);
```

```
input[3:0]a,b;
```

```
input cin;
```

```
output [3:0]sum;
```

```
output cout;
```

```
wire c1,c2,c3;
```

```
fa g1(.a(a[0]),
```

```
    .b(b[0]),
```

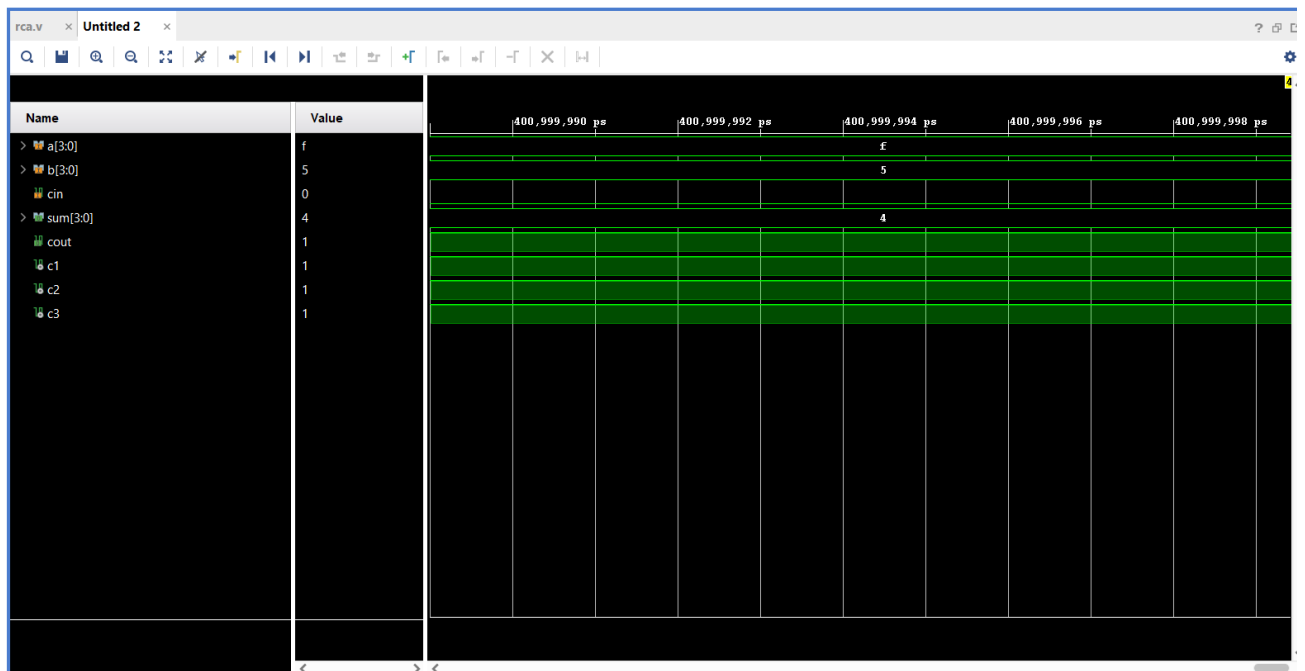
```
    .cin(cin),
```

```
    .sum(sum[0]),
```

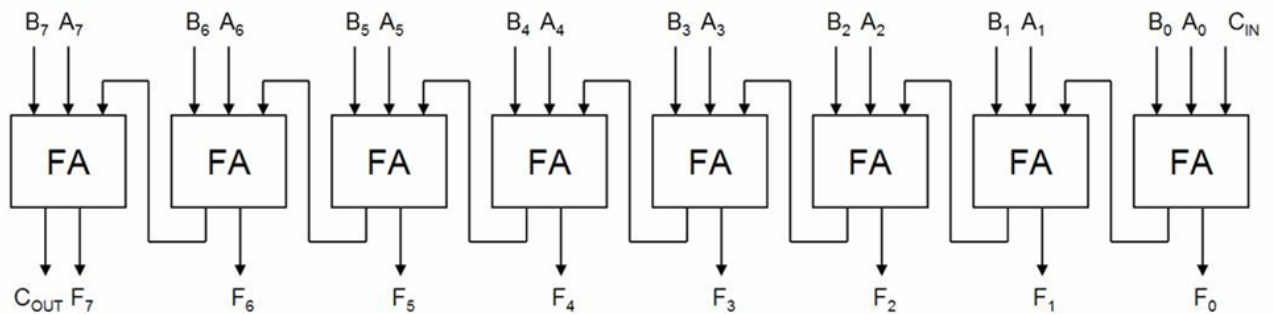
```
    .carry(c1)
```

```
);  
fa g2(.a(a[1]),  
      .b(b[1]),  
      .cin(c1),  
      .sum(sum[1]),  
      .carry(c2)  
);  
fa g3(.a(a[2]),  
      .b(b[2]),  
      .cin(c2),  
      .sum(sum[2]),  
      .carry(c3)  
);  
fa g4(.a(a[3]),  
      .b(b[3]),  
      .cin(c3),  
      .sum(sum[3]),  
      .carry(cout)  
);  
endmodule
```

RESULT:



8 BIT RIPPLE CARRY ADDER:



VERILOG CODE:

```
module fa(a,b,cin,sum,carry);
```

```
input a,b,cin;
```

```
output sum,carry;
```

```
wire w1,w2,w3;
```

```
xor g1(w1,a,b);
```

```
xor g2(sum,w1,cin);

and g3(w2,a,b);

and g4(w3,w1,cin);

or g5(carry,w3,w2);

endmodule

module rca(a,b,cin,sum,cout);

input[7:0]a,b;

input cin;

output [7:0]sum;

output cout;

wire c1,c2,c3,c4,c5,c6,c7;
```

```
fa g1(.a(a[0]),
      .b(b[0]),
      .cin(cin),
      .sum(sum[0]),
      .carry(c1)
      );

fa g2(.a(a[1]),
      .b(b[1]),
      .cin(c1),
      .sum(sum[1]),
      .carry(c2)
      );
```

```
fa g3(.a(a[2]),  
      .b(b[2]),  
      .cin(c2),  
      .sum(sum[2]),  
      .carry(c3)  
    );
```

```
fa g4(.a(a[3]),  
      .b(b[3]),  
      .cin(c3),  
      .sum(sum[3]),  
      .carry(c4)  
    );
```

```
fa g5(.a(a[4]),  
      .b(b[4]),  
      .cin(c4),  
      .sum(sum[4]),  
      .carry(c5)  
    );
```

```
fa g6(.a(a[5]),  
      .b(b[5]),  
      .cin(c5),  
      .sum(sum[5]),  
      .carry(c6)  
    );
```

```

fa g7(.a(a[6]),

    .b(b[6]),

    .cin(c6),

    .sum(sum[6]),

    .carry(c7)

);

fa g8(.a(a[7]),

    .b(b[7]),

    .cin(c7),

    .sum(sum[7]),

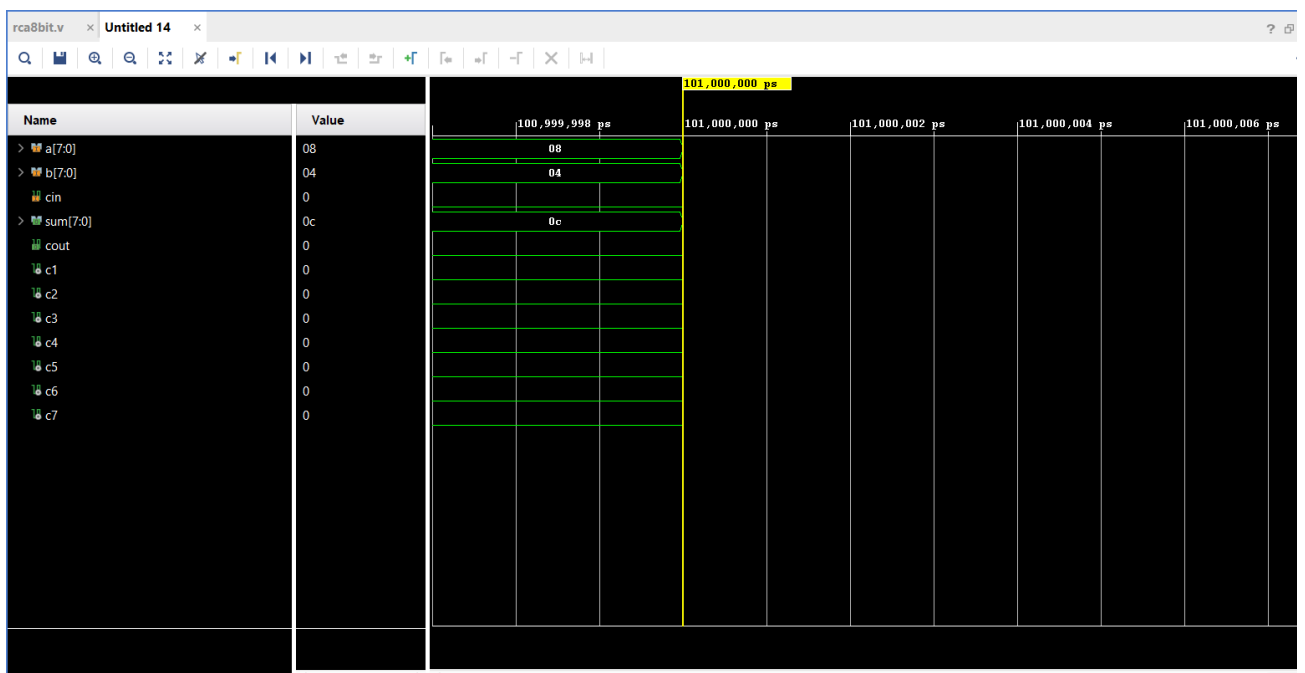
    .carry(cout)

);

endmodule

```

OUTPUT:



**RESULT:**

Hence, the simulation and synthesis of Logic Gates, Adders and Subtractors is verified using Xilinx ISE.