GRT INSTITUTE OF ENGINEERING AND TECHNOLOGY, Tiruttani

(Approved by AICTE, New Delhi, Affiliated to Anna University, Chennai)

(An ISO 9001- 2008 certified Institution)

GRT Mahalakshmi Nagar, Chennai-Tirupathi Highway, Tiruttani-631209, Tiruvallur Dt.TN

DEPARTMENT OE ELECTRONICS AND COMMUNICATION ENGINEERING

LAB MANUAL

Year & Semester: **III Year/VI Semester**

Subject Code/Name: **EC6612 VLSI DESIGN LAB**

Prepared by

Mr.P.G.Gopinath,AP/ECE

Mr.A.S.Loganathan,AP/ECE

Approved by

Dr.P.Sivakumar

LIST OF EXPERIMENTS

1. Study of simulation and FPGA implementation of Xilinx tool
2. Design & FPGA Implementation of Logic Gates
3. Design & FPGA Implementation of Half Adder and Full Adder
4. Design & FPGA Implementation of Half Subtractor and Full Subtractor
5. Design & FPGA Implementation of 8-bit Adders (Simple Adder & Ripple Carry Adder)
6. Design & FPGA Implementation of 4 bit Multiplier (Simple Multiplier & Array Multiplier)
7. Design & FPGA Implementation of Up and Down Counters
8. Design & FPGA Implementation of Finite State Machine (Moore Machine)
9. Design & FPGA Implementation of Finite State Machine (Mealy Machine)
10. Layout Extraction & Simulation of CMOS Inverter
11. Layout Extraction & Simulation of CMOS NAND and NOR Gate
12. Layout Extraction & Simulation of Differential Amplifier
13. Design of CMOS INVERTER using Tanner
14. Design of CMOS NAND AND NOR using Tanner
15. Design of DIFFERENTIAL AMPLIFIER using Tanner

Content Beyond Syllabus

1. Design & FPGA Implementation of Flip Flops (D & T Flip Flop)
2. Design and Analysis of Half Adders using Tanner
3. Design and Analysis of Dynamic CMOS logic circuits

Exp. No.: 1	**STUDY OF SIMULATION AND IMPLEMENTATION OF XILINX TOOL**

AIM:

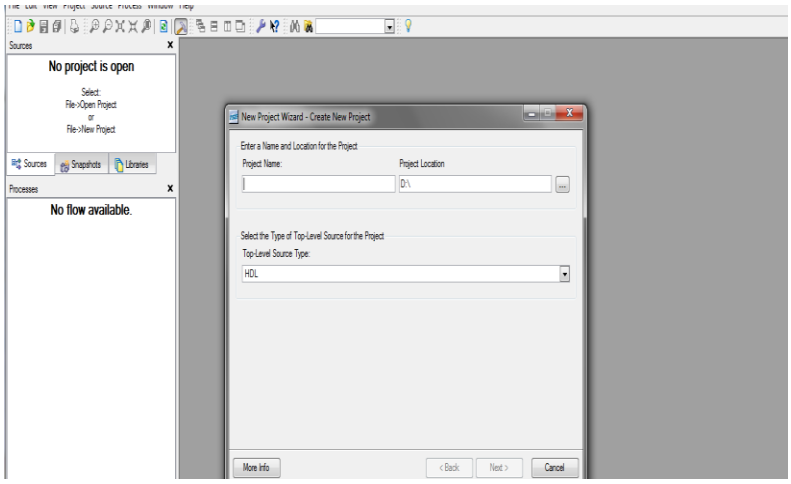
To study the simulation and implementation procedures of Xilinx tool and FPGA

STEP1:

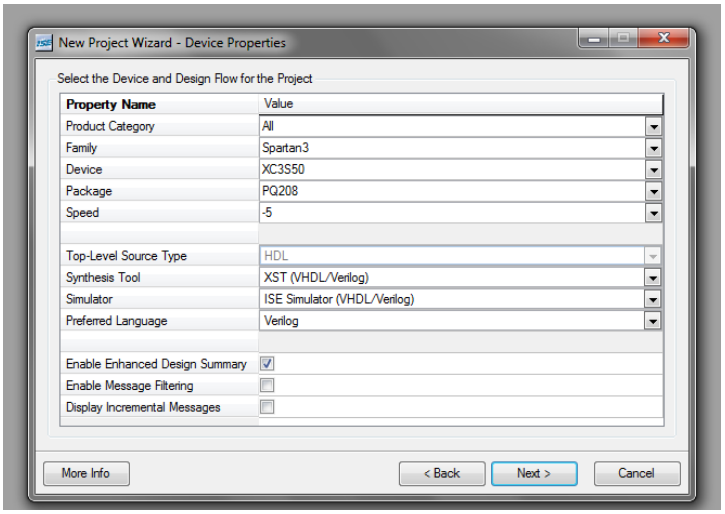
Click Xilinx ISE9.1

STEP2:

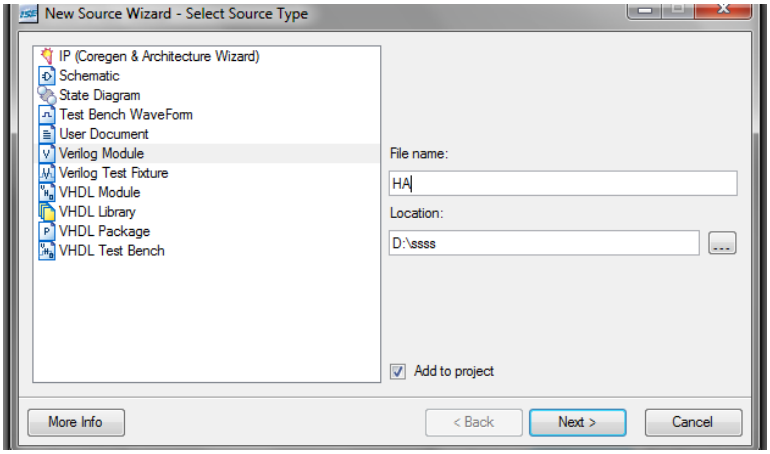
File ->New project and type the project name and check the top level source type as HDL



STEP3: Check the device properties and click Next

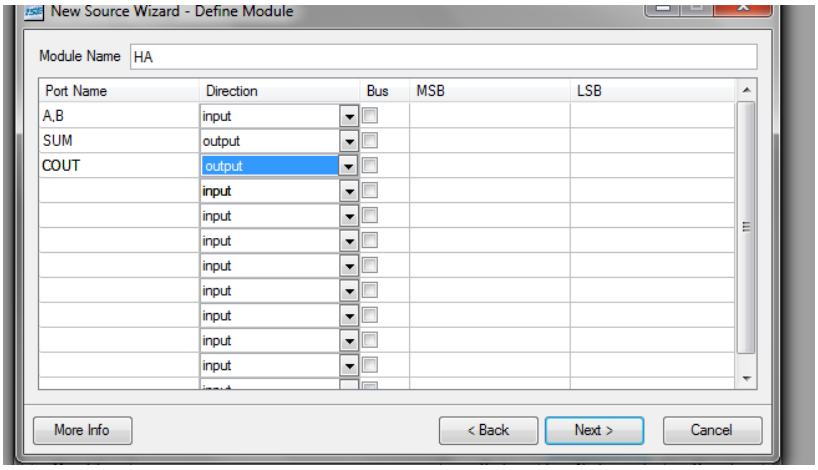


STEP4:Click New Source And Select the Verilog Module and then give the file name



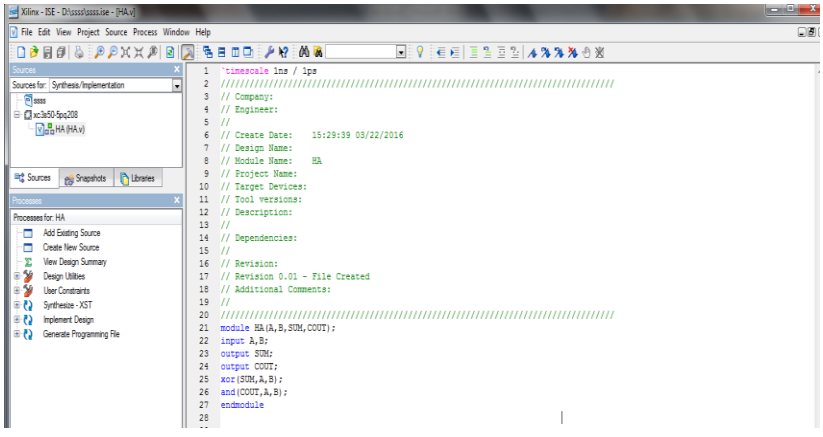
STEP5:

Select the Input,Output port names and click finish.

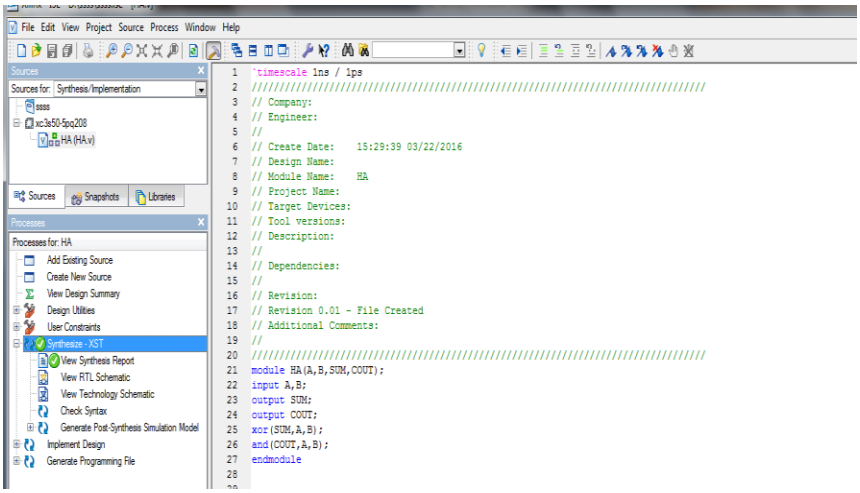


STEP6:

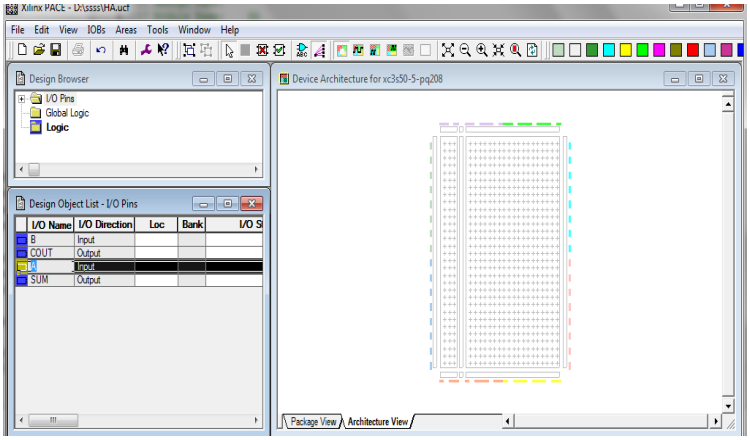
Type the program and save it



STEP7: Check the synthesize XST and check syntax

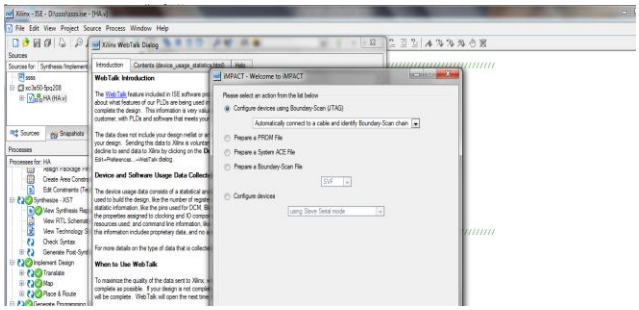


STEP8: Select user constraints-> assign package pins,set port numbers and save it then select IO Bus delimiter as XST default<>->click ok



STEP9:

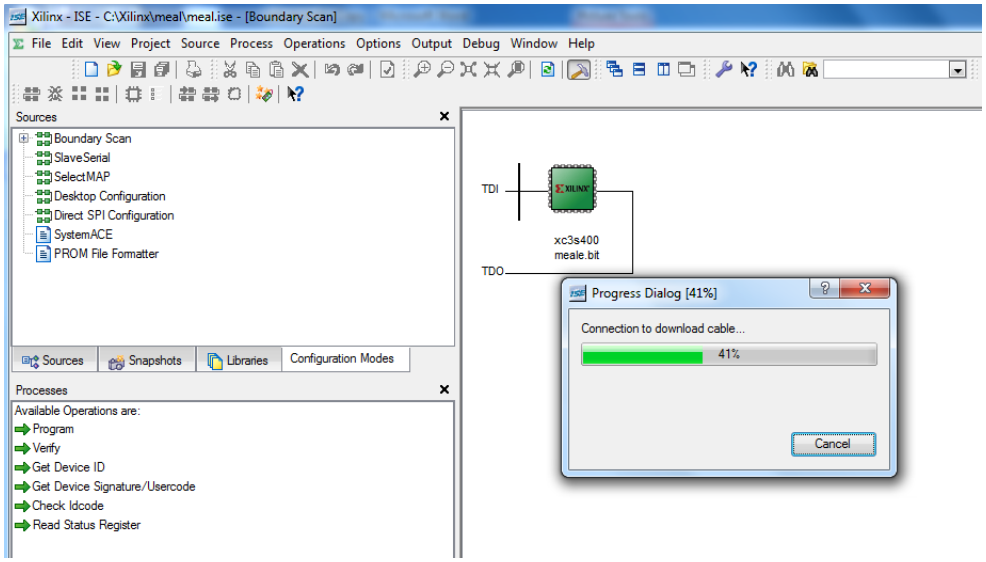
Double click implement design and click generate programming file->configure device(impact)->finish then select bit file



STEP10:

Right click on the xc3s400 figure->program->filename then click finish

and Finally check the functionality in hardware



Result:

Thus the simulation and implementation procedure of Xilinx and FPGA is studied

Exp. No.: 2	**DESIGN & FPGA IMPLEMENTATION OF LOGIC GATES**

AIM:

To design, simulate and implement basic logic gates using Verilog HDL

APPARATUS REQUIRED:

- PC with Windows XP.
- XILINX 9.2i
- FPGA-SPARTAN-3 KIT
- PARALLEL TO JTAG CABLE

THEORY:

AND GATE:

The AND gate performs logical multiplication which is most commonly known as the AND junction. The operation of AND gate is such that the output is high only when all its inputs are high and when any one of the inputs is low the output is low.

$$Y = a \& b$$

OR GATE:

The OR gate performs logical addition which is most commonly known as the OR junction. The operation of OR gate is such that the output is high only when any one of its input is high and when both the inputs are low the output is low.

$$Y = a \mid b$$

NOT GATE:

The Inverter performs a basic logic gate function called Inversion or Complementation. The purpose of an inverter is to change one logic level to opposite level. When a high level is applied top an inverter, the low level will appear at the output and vice versa.

$$Y = \sim a$$

NAND GATE:

The term NAND is derived from the complement of AND. It implies the AND junction with an inverted output. The operation of NAND gate is such that the output is low only when all its inputs are high and when any one of the inputs is low the output is high.

$$Y = \sim(a \& b)$$

NOR GATE:

The term NOR is derived from the complement of OR. It implies the OR junction with an inverted output. The operation of NOR gate is such that the output is high only when all its inputs are low and when any one of the inputs is high the output is low.

$$Y = \sim(a \mid b)$$

EX-OR GATE:

The output is high only when the inputs are at opposite level.

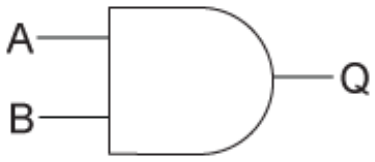
$Y = a \wedge b$

EX-NOR GATE:

The output is high only when the inputs are at same level.

$$Y = \sim(a \wedge b)$$

AND Gate:

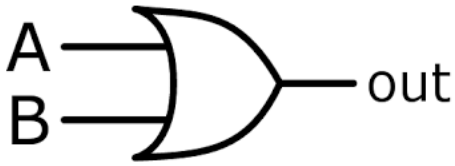


Truth table:

AND Gate

Input1	Input2	Output
0	0	0
0	1	0
1	0	0
1	1	1

OR Gate:

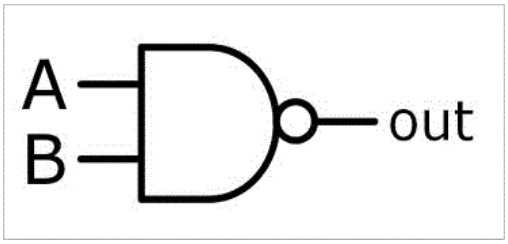


Truth table:

OR Gate

Input1	Input2	Output
0	0	0
0	1	1
1	0	1
1	1	1

NAND Gate:

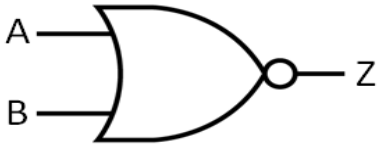


Truth table:

NAND Gate

Input1	Input2	Output
0	0	1
0	1	1
1	0	1
1	1	0

NOR Gate:

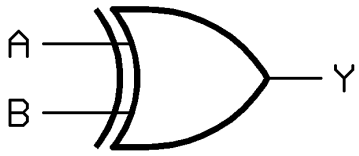


Truth table:

NOR Gate

Input1	Input2	Output
0	0	1
0	1	0
1	0	0
1	1	0

XOR Gate:



Truth table:

XOR Gate

Input1	Input2	Output
0	0	0
0	1	1
1	0	1
1	1	0

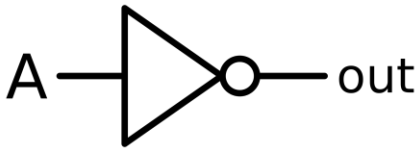
XNOR Gate:

Truth table:

XNOR Gate

Input1	Input2	Output
0	0	1
0	1	0
1	0	0
1	1	1

Not Gate:



Truth table:

NOT Gate

Input	Output
0	1
1	0

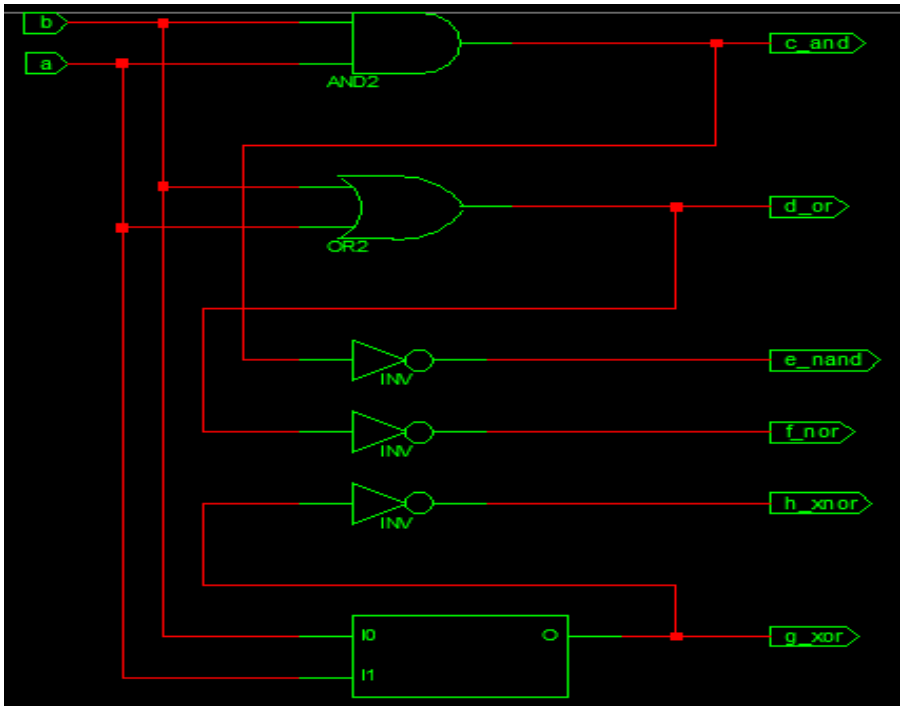
ALGORITHM:

- New project and type the project name and check the top level source type as HDL
- Enter the device properties and click Next
- Click New Source And Select the Verilog Module and then give the file name
- Give the Input and Output port names and click finish.
- Type the Verilog program and save it
- Double click the synthesize XST and check syntax
- Simulate the waveform by behavioral simulation
- For implementation Select User constraints and give input and output port pin number
- Click Implement design for Translate, map and place & route
- Generate .bit file using programming file
- Implement in FPGA through parallel-JTAG cable
- Check the behavior of design in FPGA by giving inputs

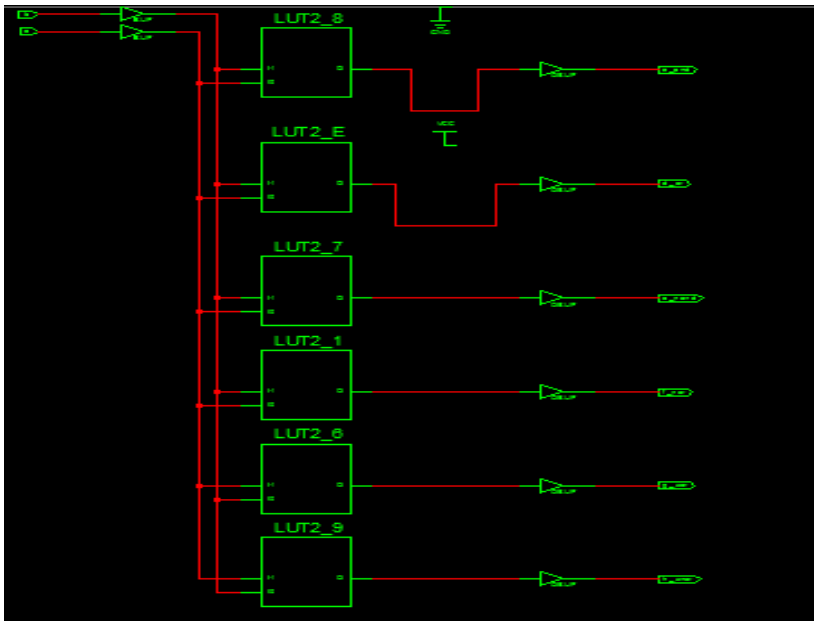
Program:

```
module gate(a, b, c_and, d_or, e_nand, f_nor, g_xor, h_xnor);
  input a;
  input b;
  output c_and;
  output d_or;
  output e_nand;
  output f_nor;
  output g_xor;
  output h_xnor;
  and (c_and,a,b);
       or (d_or,a,b);
       nand (e_nand,a,b);
       nor (f_nor,a,b);
       xor (g_xor,a,b);
       xnor (h_xnor,a,b);
       endmodule
```

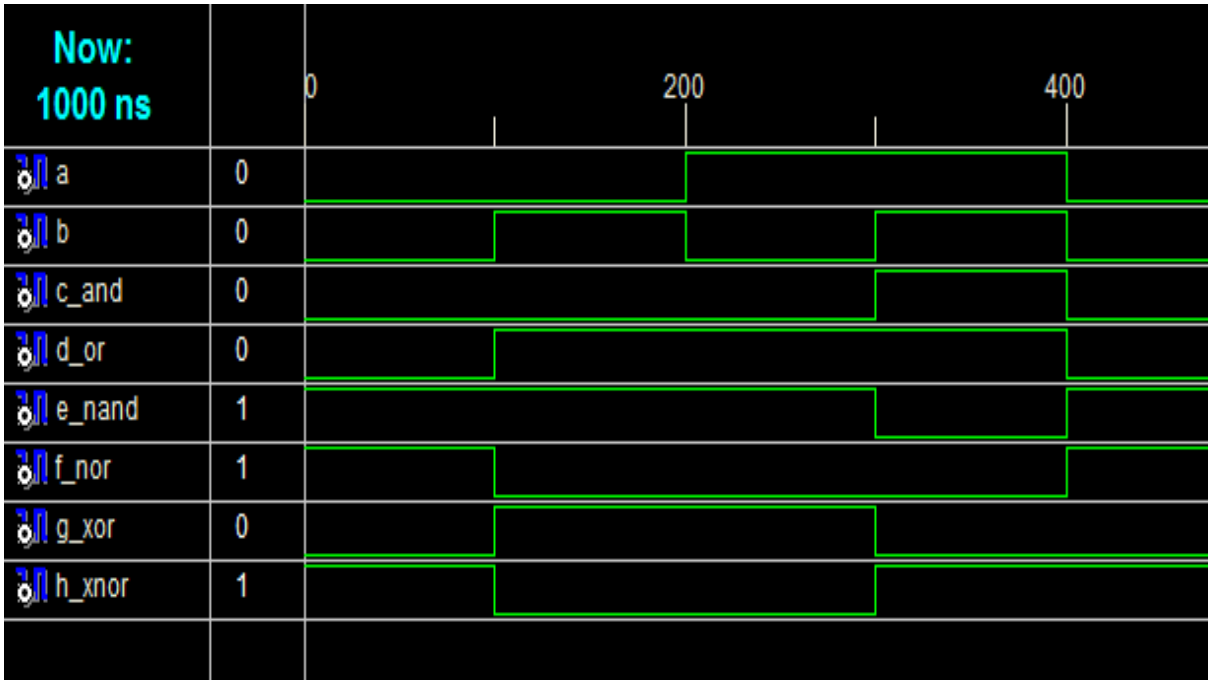
RTL Schematic:



Technological Schematic:



Output Waveform:



Result:

Thus the basic gates are designed, simulated and implemented using Verilog HDL.

Exp. No.: 3	**DESIGN & FPGA IMPLEMENTATION OF HALF ADDER AND FULL ADDER**

AIM:

To design, simulate and implement basic half adder and full adder using Verilog HDL

APPARATUS REQUIRED:

- PC with Windows XP.
- XILINX 9.2i
- FPGA-SPARTAN-3 KIT
- PARALLEL TO JTAG CABLE

THEORY:

HALF ADDER:

The half adder consists of two input variables designated as Augends and Addend bits. Output variables produce the Sum and Carry. The 'carry' output is 1 only when both inputs are 1 and ,sum' is 1 if any one input is 1. The Boolean expression is given by,

sum = x ^ y carry = x & y

FULL ADDER:

A Full adder is a combinational circuit that focuses the arithmetic sum of three bits. It consists of 3 inputs and 2 outputs. The third input is the carry from the previous Lower Significant Position. The two outputs are designated as Sum (S) and Carry (C). The binary variable S gives the value of the LSB of the Sum. The output S=1 only if odd number of 1's are present in the input and the output C=1 if two or three inputs are 1.

sum = x ^ y ^ z

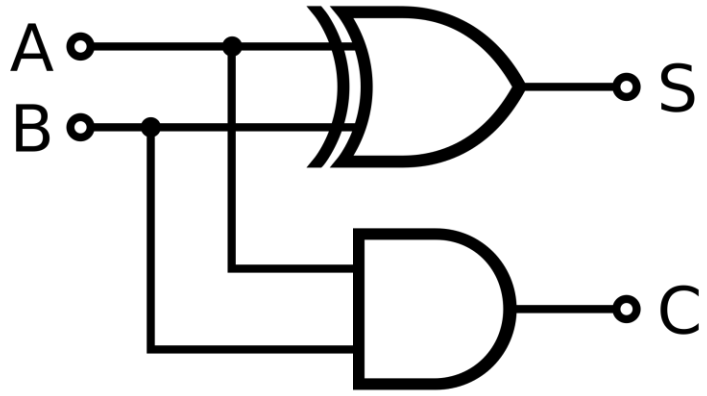
carry= (x & y) | (y & z) | (x & z)

ALGORITHM:

- New project and type the project name and check the top level source type as HDL
- Enter the device properties and click Next
- Click New Source And Select the Verilog Module and then give the file name
- Give the Input and Output port names and click finish.
- Type the Verilog program and save it
- Double click the synthesize XST and check syntax

- Simulate the waveform by behavioral simulation
- For implementation Select User constraints and give input and output port pin number
- Click Implement design for Translate, map and place & route
- Generate .bit file using programming file
- Implement in FPGA through parallel-JTAG cable
- Check the behavior of design in FPGA by giving inputs

Half Adder:



Program :

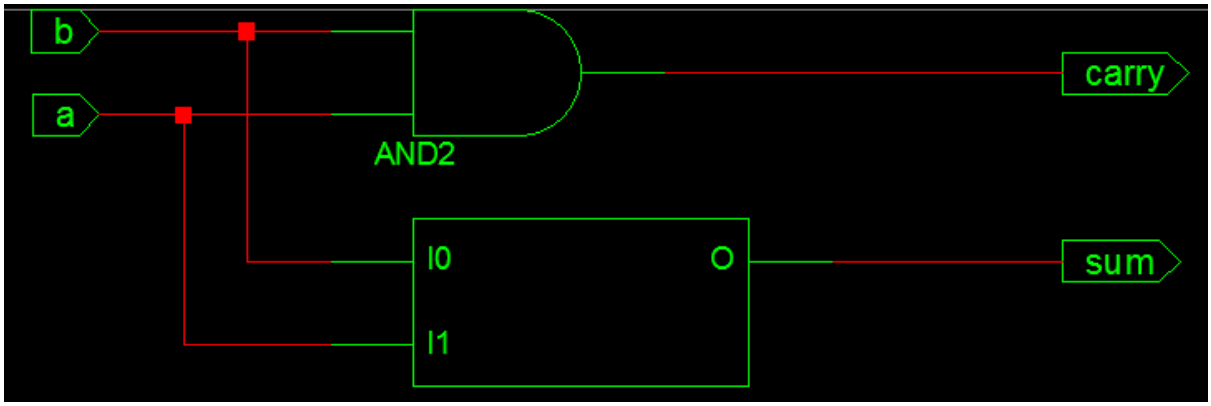
```
Module Half add(a,b,sum,carry);
input a,b;
output sum,carry;
 xor (sum,a,b);
and (carry,a,b);
endmodule
```

Truth table:

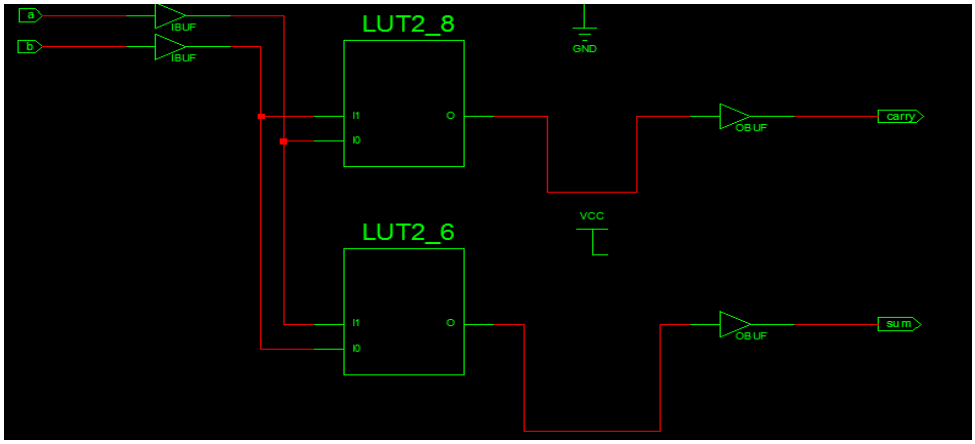
Half Adder

Input1	Input2	Carry	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

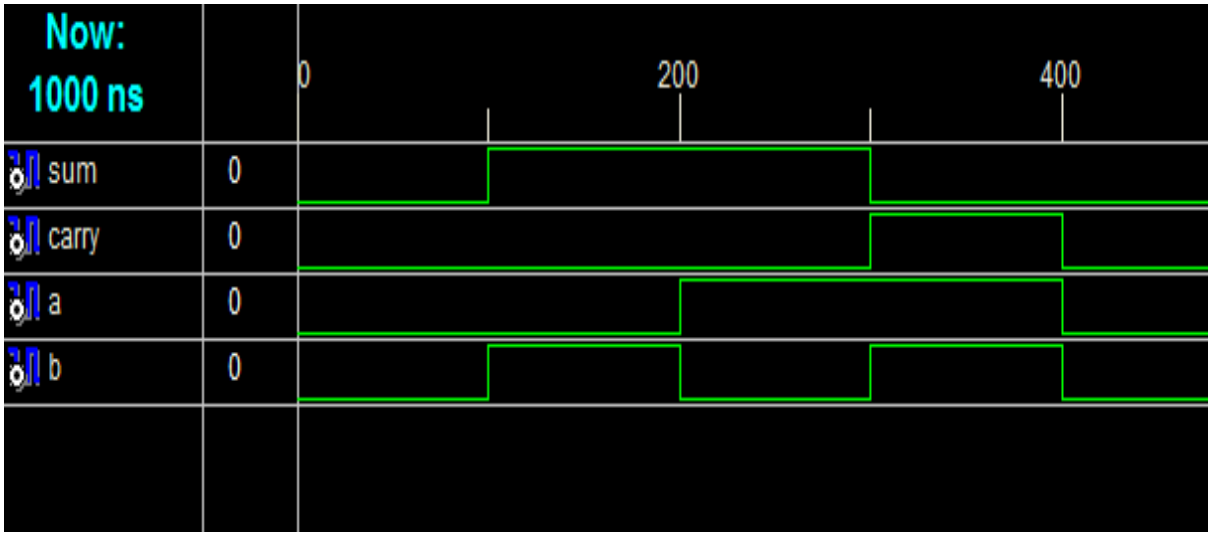
RTL SCHEMATIC:



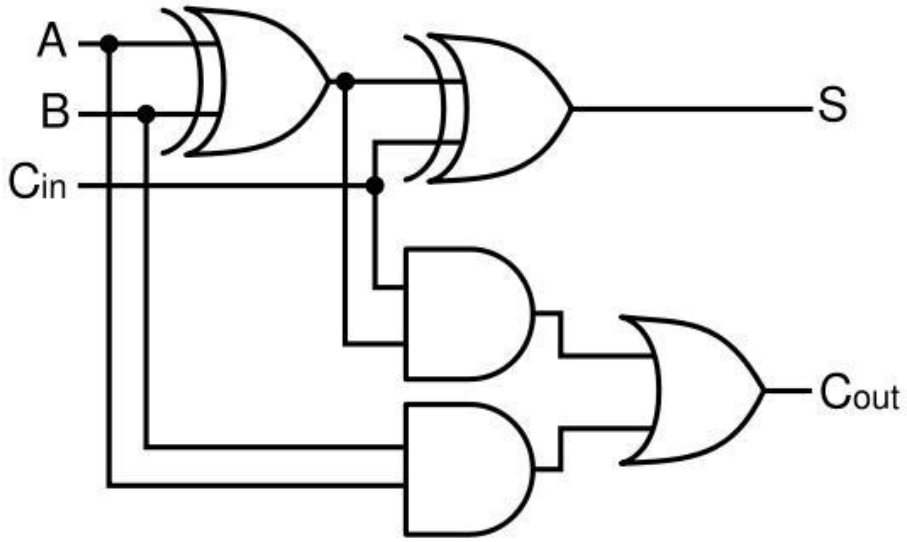
TECHNOLOGIC SCHEMATIC:



OUTPUT WAVEFORM:



Full Adder:



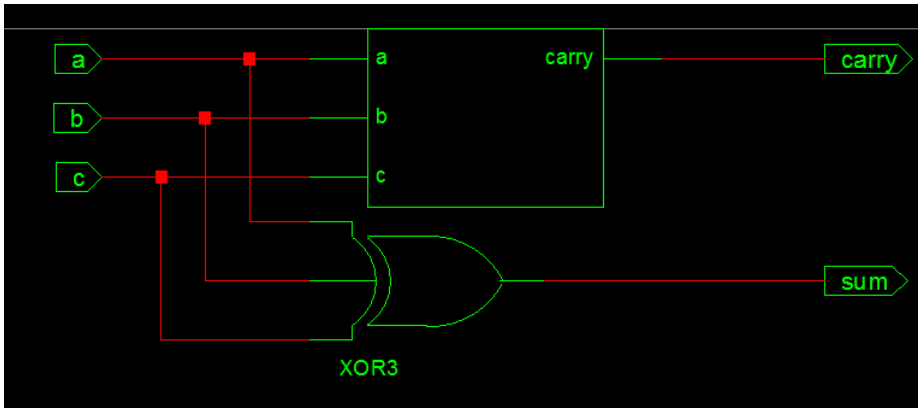
Program:

```
module fulladd (sum, carry, a, b, c);
input a, b, c;
output sum, carry;
wire w1, w2, w3;
xor (sum,a,b,c);
and (w1,a,b);
and(w2,b,c);
and(w3,c,a);
or(carry,w1,w2,w3);
endmodule
```

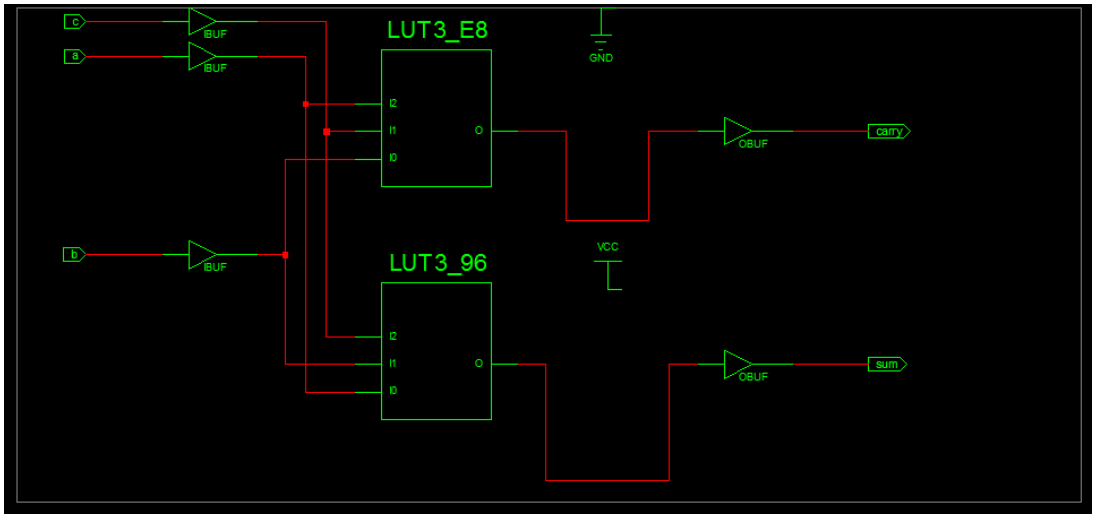
Truth Table:

a	b	c	carry	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

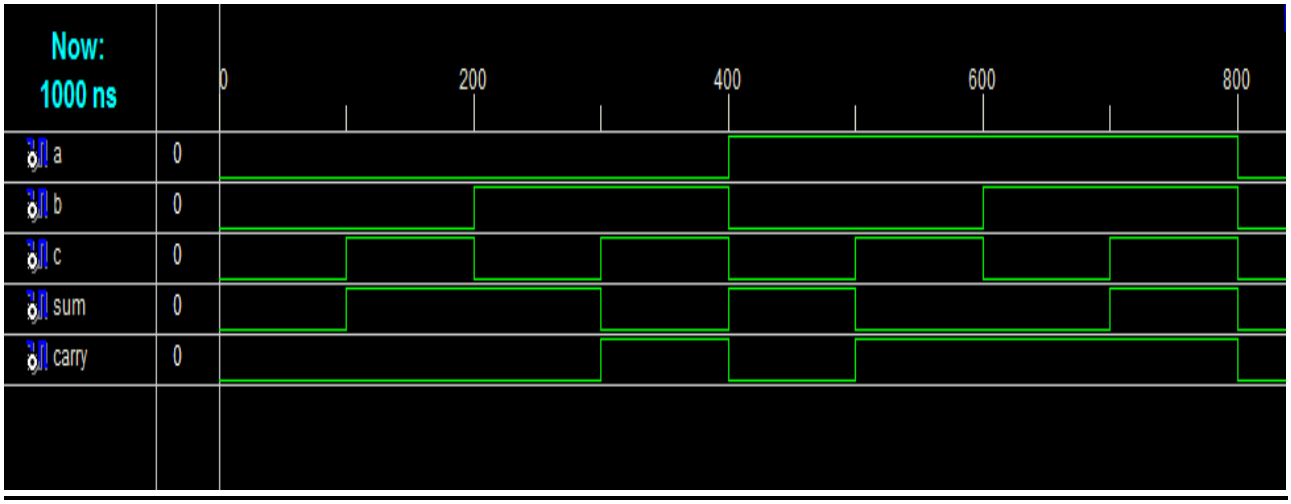
RTL SCHEMATIC:



TECHNOLOGIC SCHEMATIC:



OUTPUT WAVEFORM:



Result:

Thus the half adder and full adder was simulated and implemented successfully.

Exp. No.: 4	**DESIGN & FPGA IMPLEMENTATION OF HALF SUBTRACTOR AND FULL SUBTRACTOR**

AIM:

To design, simulate and implement half subtractor and full subtractor using Verilog HDL.

APPARATUS REQUIRED:

- PC with Windows XP.
- XILINX 9.2i
- FPGA-SPARTAN-3 KIT
- PARALLEL TO JTAG CABLE

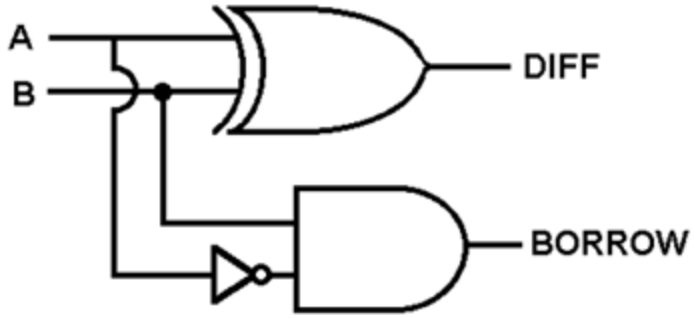
Theory:

In electronics, a subtractor can be designed using the same approach as that of an adder. The binary subtraction process is summarized below. As with an adder, in the general case of calculations on multi-bit numbers, three bits are involved in performing the subtraction for each bit of the difference: the minuend (X_{i}), subtrahend (Y_{i}), and a borrow in from the previous (less significant) bit order position (B_{i}). The outputs are the difference bit (D_{i}) and borrow bit B_{i+1}. The subtractor is best understood by considering that the subtrahend and both borrow bits have negative weights, whereas the X and D bits are positive.

ALGORITHM:

- New project and type the project name and check the top level source type as HDL
- Enter the device properties and click Next
- Click New Source And Select the Verilog Module and then give the file name
- Give the Input and Output port names and click finish.
- Type the Verilog program and save it
- Double click the synthesize XST and check syntax
- Simulate the waveform by behavioral simulation
- For implementation Select User constraints and give input and output port pin number
- Click Implement design for Translate, map and place & route
- Generate .bit file using programming file
- Implement in FPGA through parallel-JTAG cable
- Check the behavior of design in FPGA by giving inputs

HALF SUBTRACTOR:



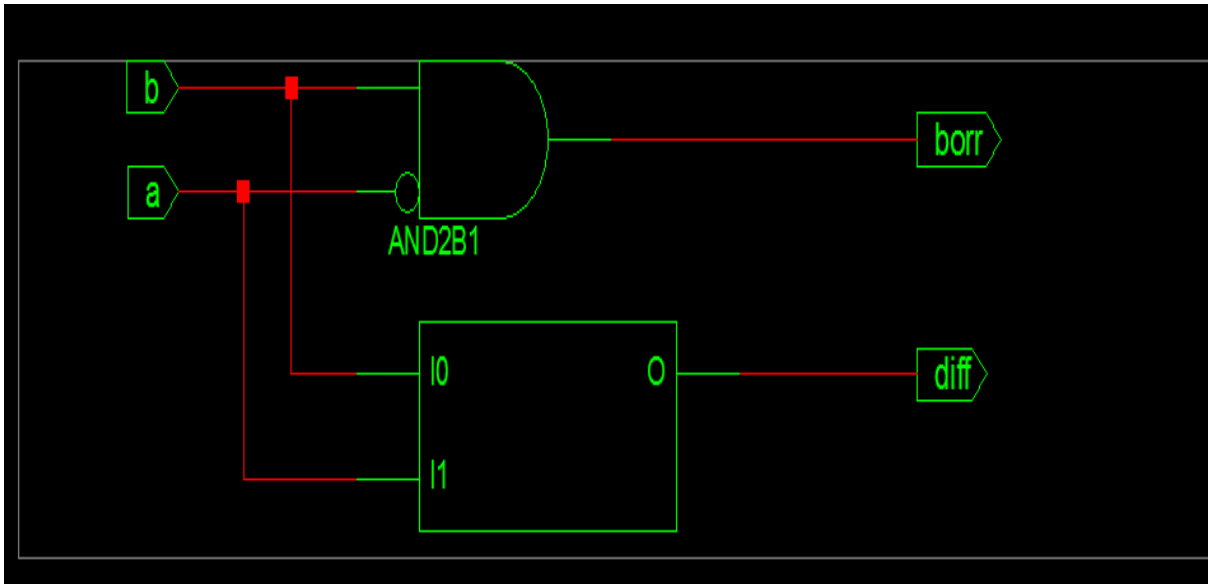
Program:

```
module halfSub(a, b, diff, borr);
input a, b;
output diff, borr;
 wire s;
 not (s, a);
 xor (diff, a, b);
 and (borr, s, b);
 endmodule
```

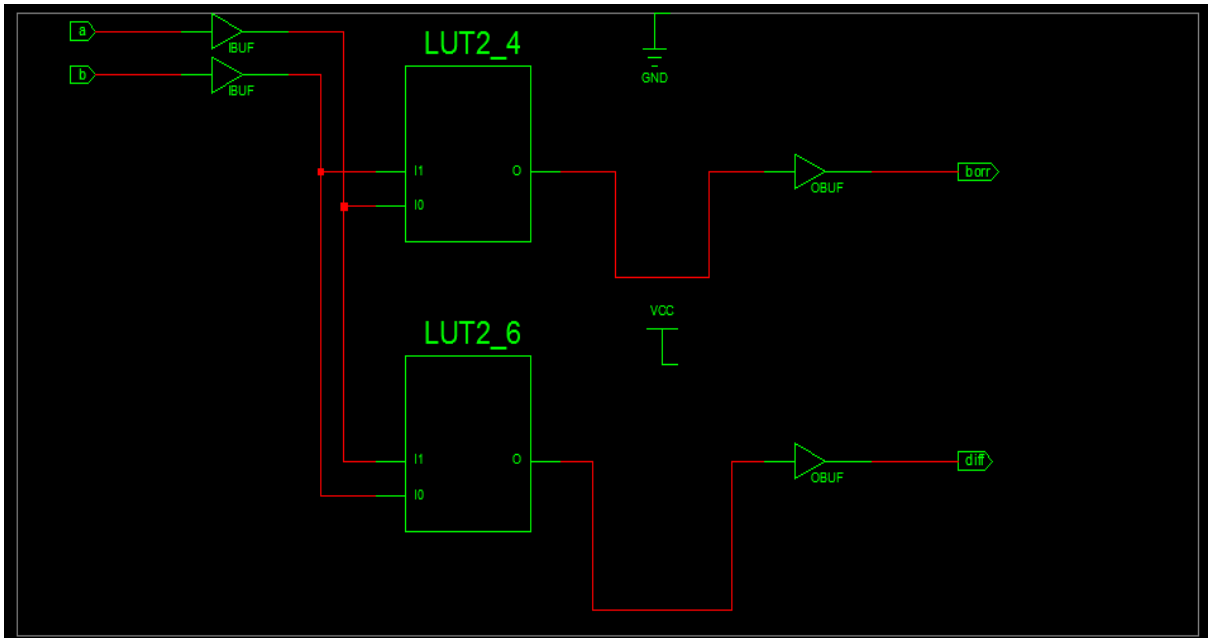
Truth Table:

Input1	Input2	Borrow	Difference
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

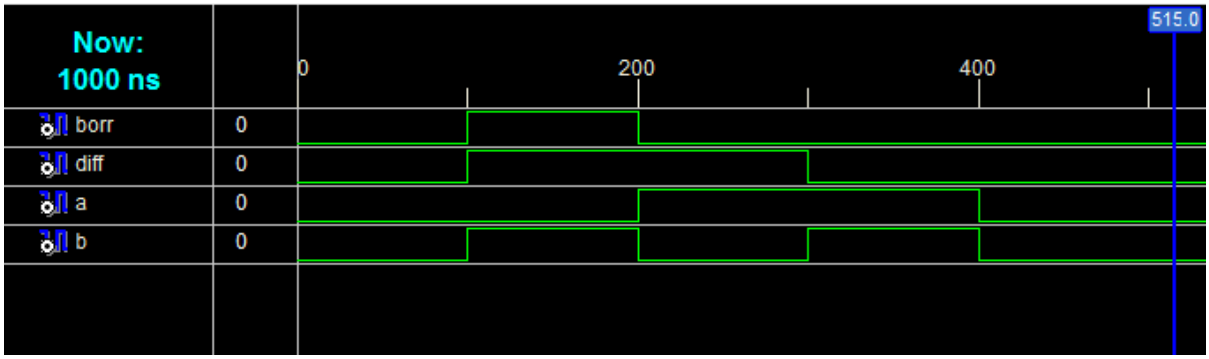
RTL SCHEMATIC



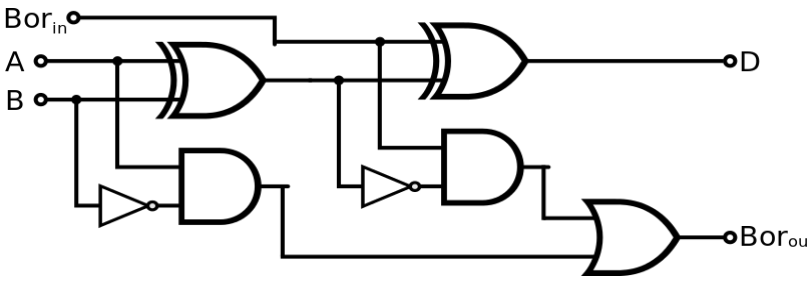
TECHNOLOGIC SCHEMATIC



Output Wave:



FULL SUBTRACTOR:



Program:

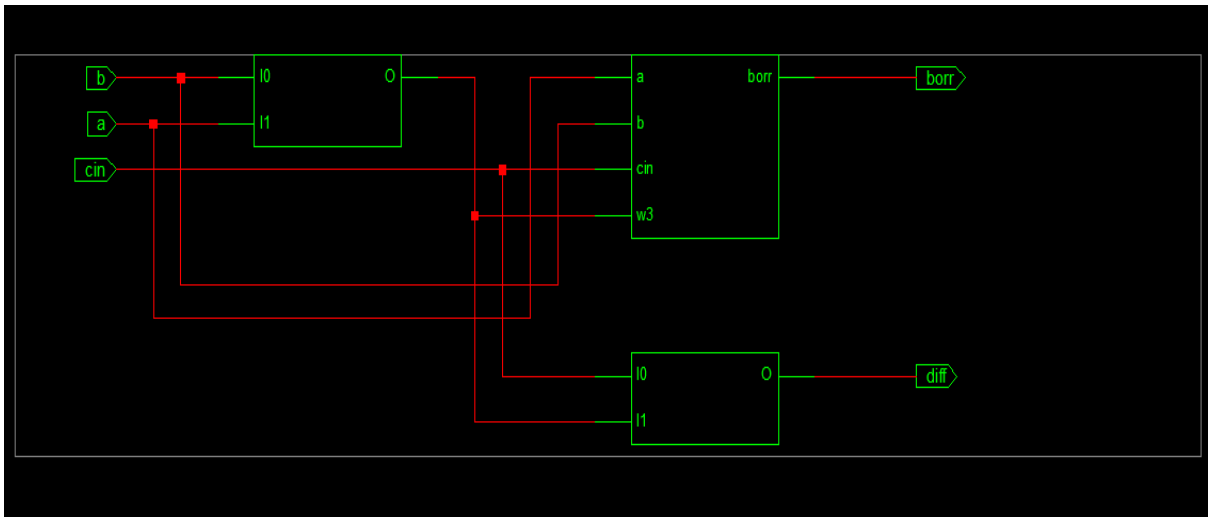
```
module fullsub (a, b, cin, diff, borr);
        input  a, b, cin;
        output diff, borr;
        wire w1, w2, w3, w4, w5;
        not n1(w1, a);
```

```
not n2(w4, w3);
xor x1(w3, a, b);
xor x2(diff, w3, cin);
and a1(w2, w1, b);
and a2(w5,w4,cin);
or g1(borr, w5, w2);
endmodule
```

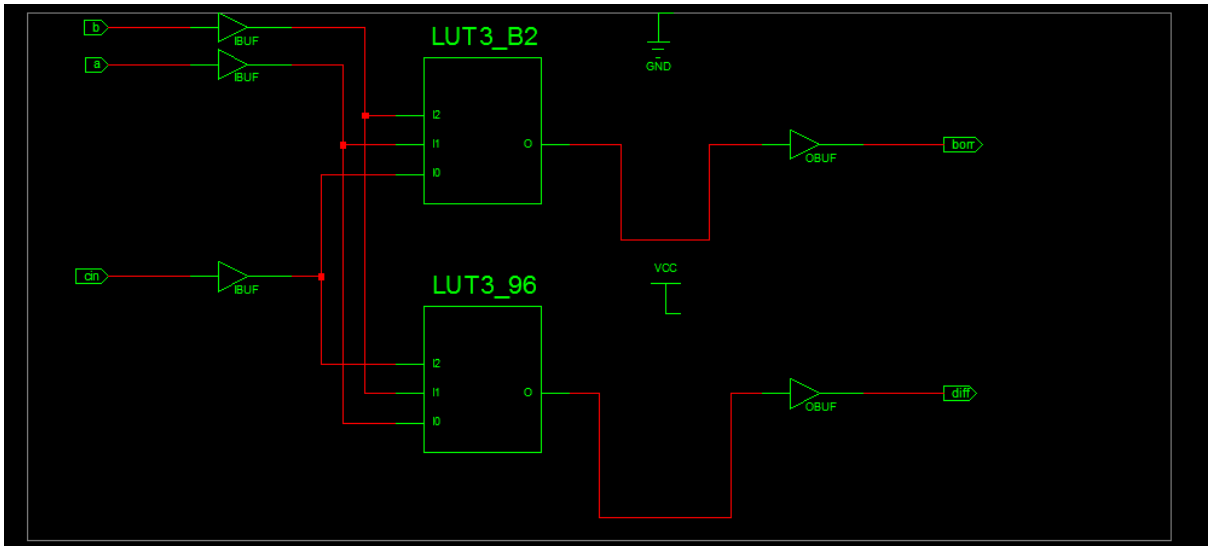
Truth Table:

A	*B*	*Cin*	*D*	B_{out}
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

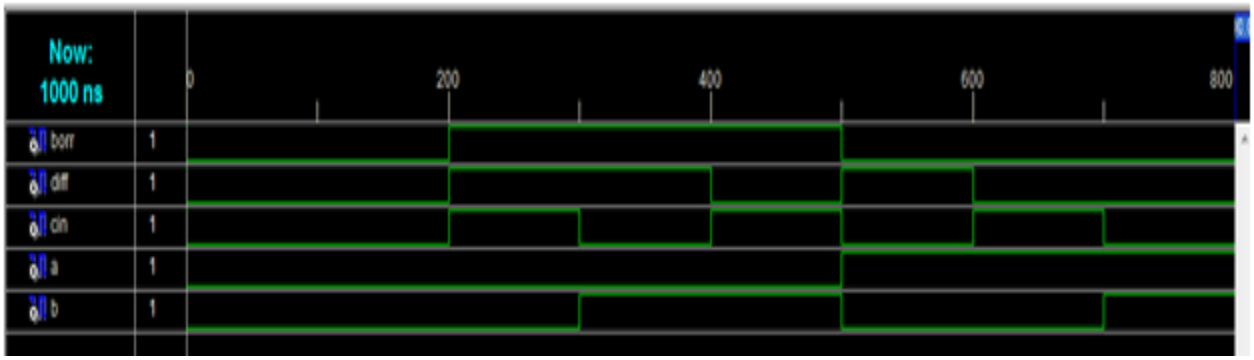
RTL Schematic



TEHNOLOGICAL SCHEMATIC



Output Wave:



Result:

Thus the half subtractor and full subtractor was designed, simulated and implemented successfully.

Exp. No.: 5	**DESIGN & FPGA IMPLEMENTATION OF 8-BIT ADDERS (SIMPLE & RIPPLE CARRY ADDER)**

AIM:

To design and to implement 8-bit adders (simple adder and ripple carry adder using Verilog HDL.

APPARATUS REQUIRED:

- PC with Windows XP.
- XILINX 9.2i
- FPGA-SPARTAN-3 KIT
- PARALLEL TO JTAG CABLE

ALGORITHM:

- New project and type the project name and check the top level source type as HDL
- Enter the device properties and click Next
- Click New Source And Select the Verilog Module and then give the file name
- Give the Input and Output port names and click finish.
- Type the Verilog program and save it
- Double click the synthesize XST and check syntax
- Simulate the waveform by behavioral simulation
- For implementation Select User constraints and give input and output port pin number
- Click Implement design for Translate, map and place & route
- Generate .bit file using programming file
- Implement in FPGA through parallel-JTAG cable
- Check the behavior of design in FPGA by giving inputs

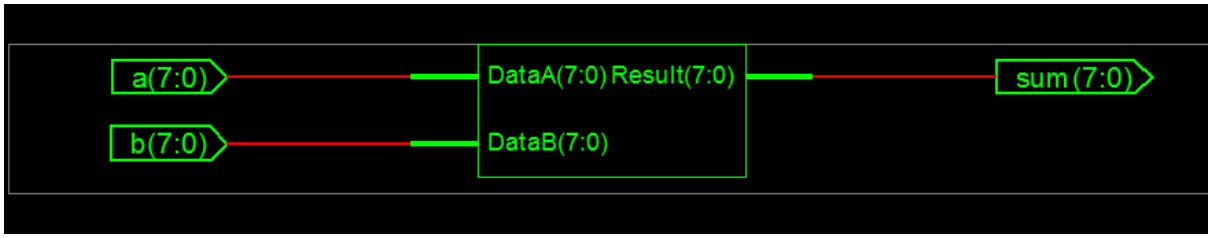
Program:

```
module ad(a,b,sum);
input [7:0]a,b;
output [7:0]sum;
assign sum=a+b;
endmodule
```

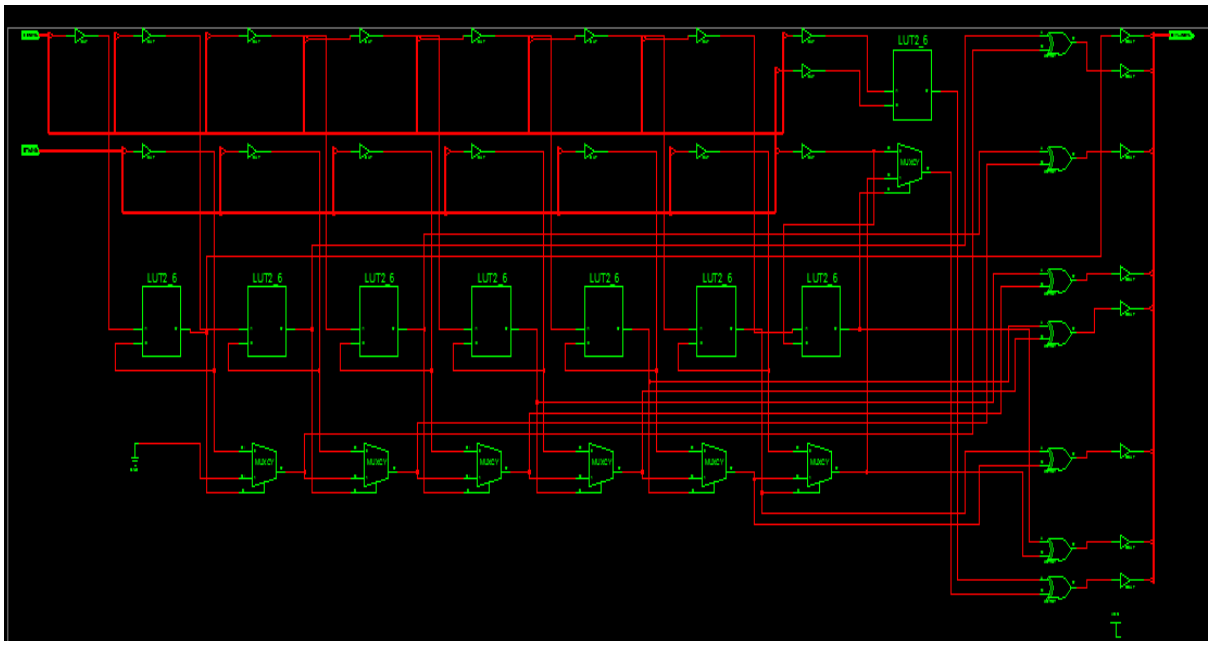
Truth Table:

A	B	RES
1111111	0000000	1111111

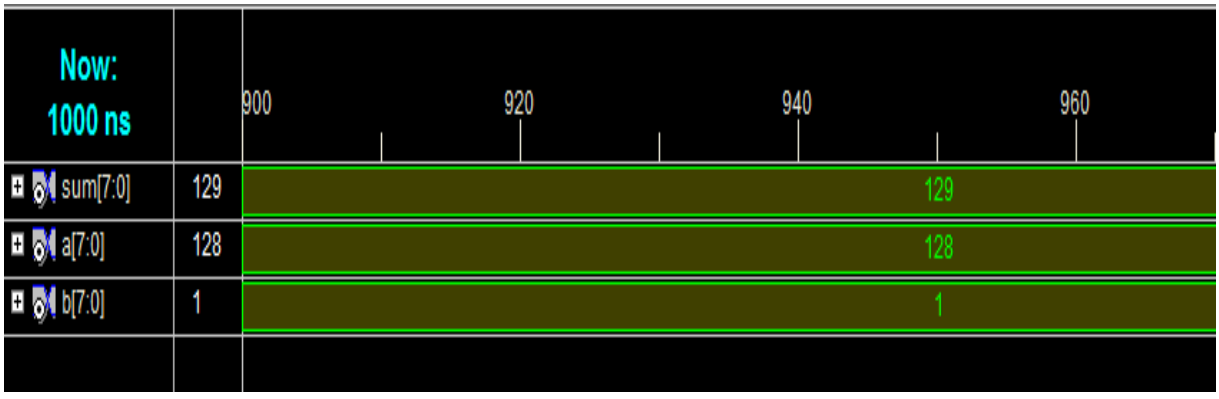
-RTL SCHEMATIC:



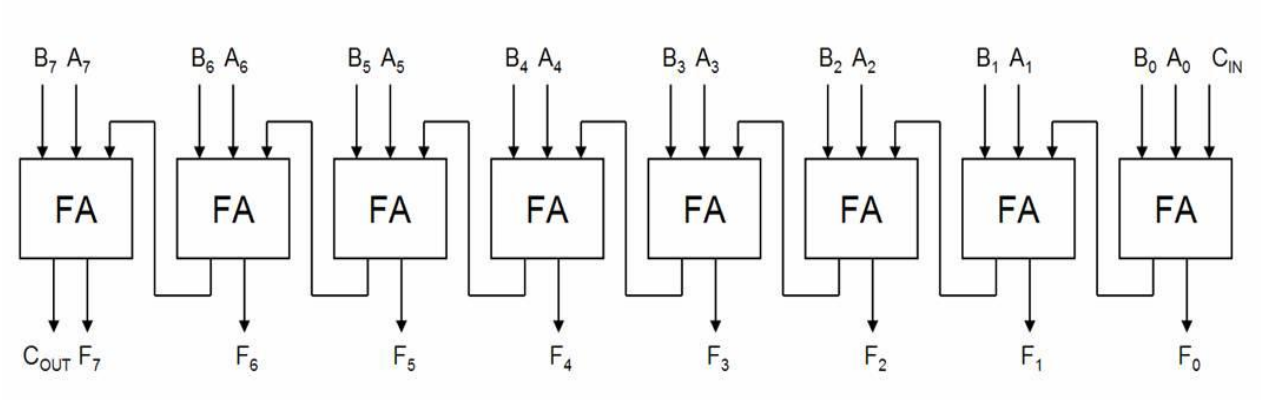
TECHNOLOGIC SCHEMATIC



OUTPUT WAVEFORM:



RIPPLE CARRY ADDER:



PROGRAM:

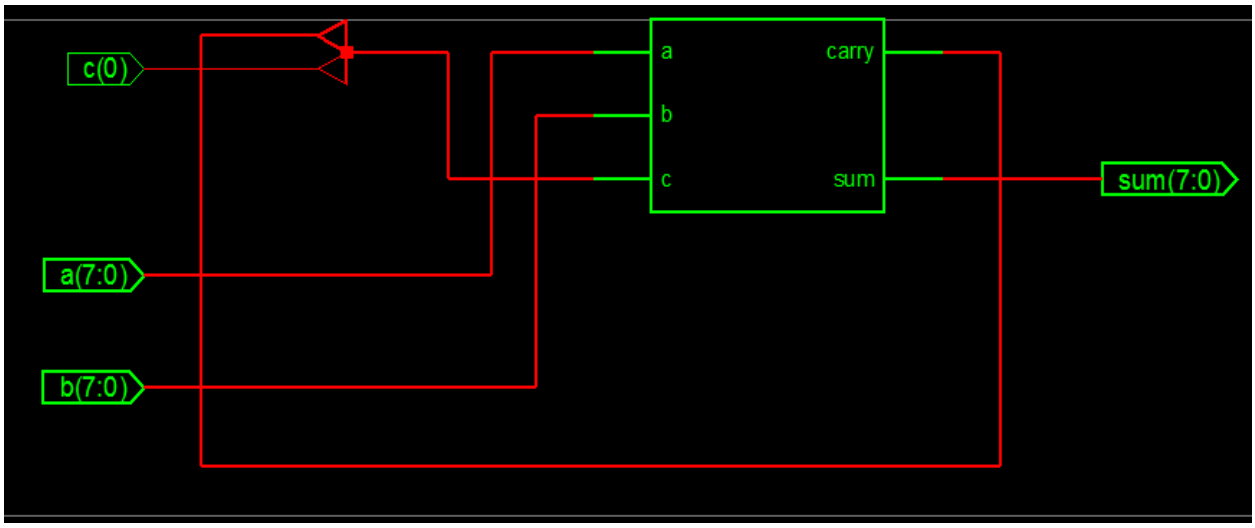
Main program:

```
module sw(a, b, c, sum, carry);
input [7:0]a;
input [7:0]b;
input c;
output [7:0]sum;
output carry;
wire [6:0]c;
gk fa0(a[0],b[0],c,sum[0],c1);
gk fa1(a[1],b[1],c1,sum[1],c2);
gk fa2(a[2],b[2],c2,sum[2],c3);
gk fa3(a[3],b[3],c3,sum[3],c4);
gk fa4(a[4],b[4],c4,sum[4],c5);
gk fa5(a[5],b[5],c5,sum[5],c6);
gk fa6(a[6],b[6],c6,sum[6],c7);
gk fa7(a[7],b[7],c7,sum[7],carry);
endmodule
```

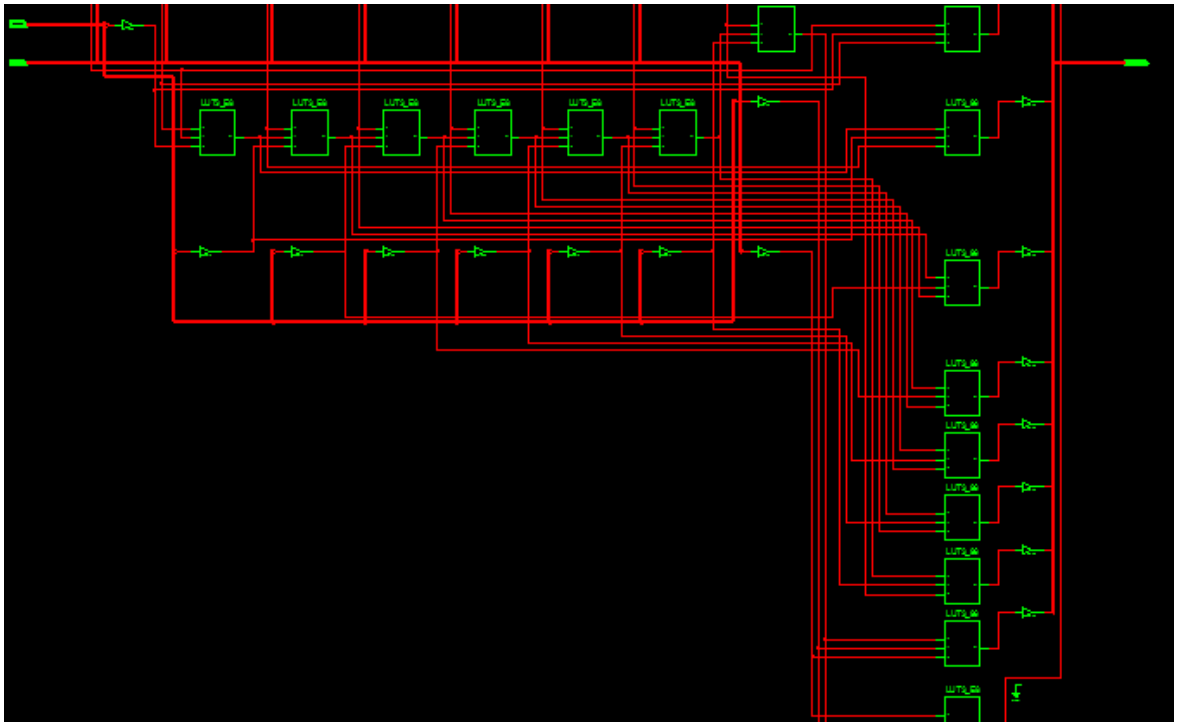
Sub-Program

```
module fulladd (sum, carry, a, b, c);
input a, b, c;
output sum, carry;
wire w1, w2, w3;
xor (sum,a,b,c);
and (w1,a,b);
and(w2,b,c,);
and(w3,c,a);
or(carry,w1,w2,w3);
endmodule
```

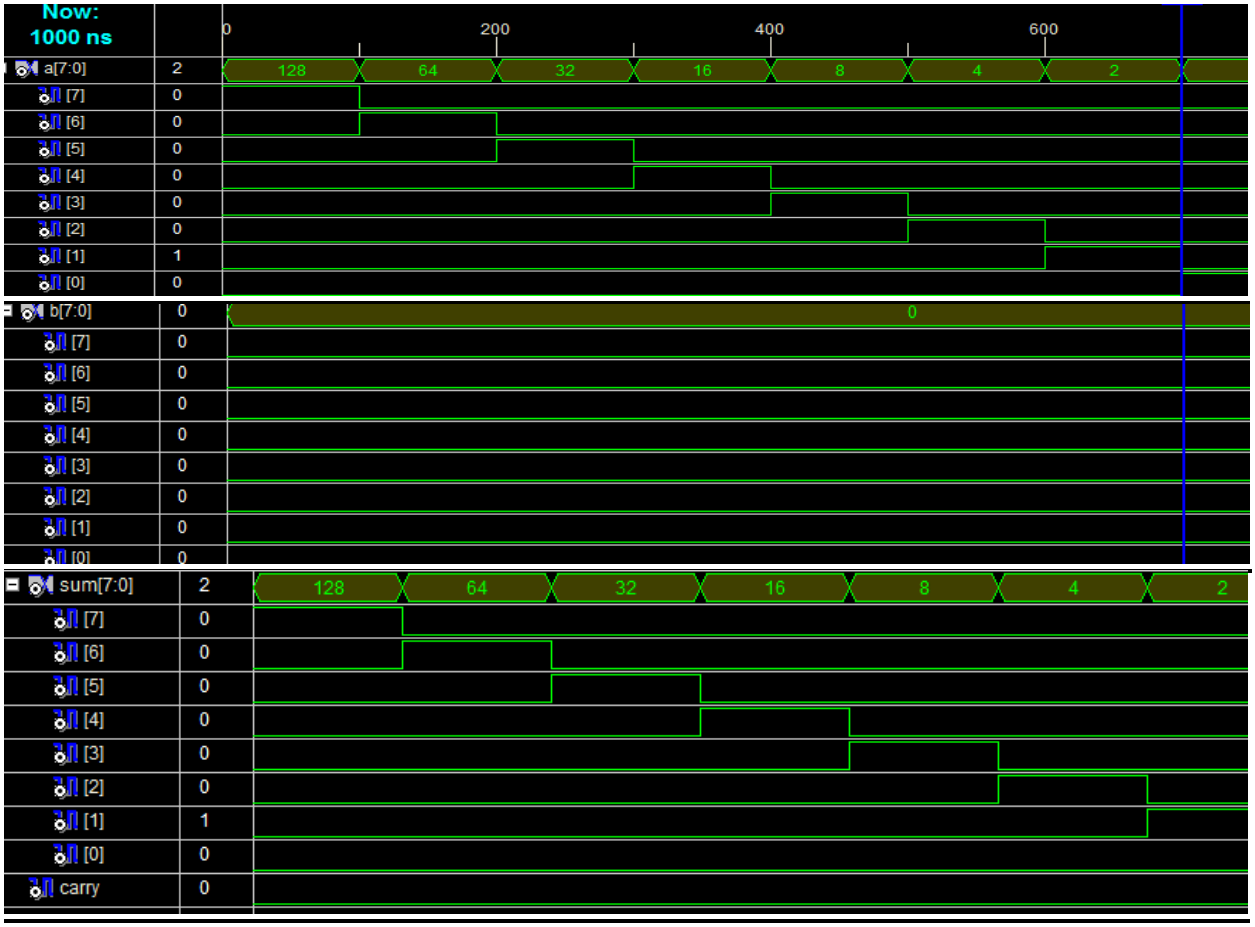
RTL SCHEMATIC:



TECHNOLOGICAL SCHEMATIC:



OUTPUT WAVEFORM:



RESULT:

Thus simple adder and ripple carry adder was designed and implemented successfully.

Exp. No.: 6	**DESIGN & FPGA IMPLEMENTATION OF 4-BIT MULTIPLIER (SIMPLE & ARRAY MULTIPLIER)**

AIM:

To implement Multiplexer & Demultiplexer using Verilog HDL.

APPARATUS REQUIRED:

- PC with Windows XP.
- XILINX 9.2i
- FPGA-SPARTAN-3 KIT
- PARALLEL TO JTAG CABLE

ALGORITHM:

- New project and type the project name and check the top level source type as HDL
- Enter the device properties and click Next
- Click New Source And Select the Verilog Module and then give the file name
- Give the Input and Output port names and click finish.
- Type the Verilog program and save it
- Double click the synthesize XST and check syntax
- Simulate the waveform by behavioral simulation
- For implementation Select User constraints and give input and output port pin number
- Click Implement design for Translate, map and place & route
- Generate .bit file using programming file
- Implement in FPGA through parallel-JTAG cable
- Check the behavior of design in FPGA by giving inputs

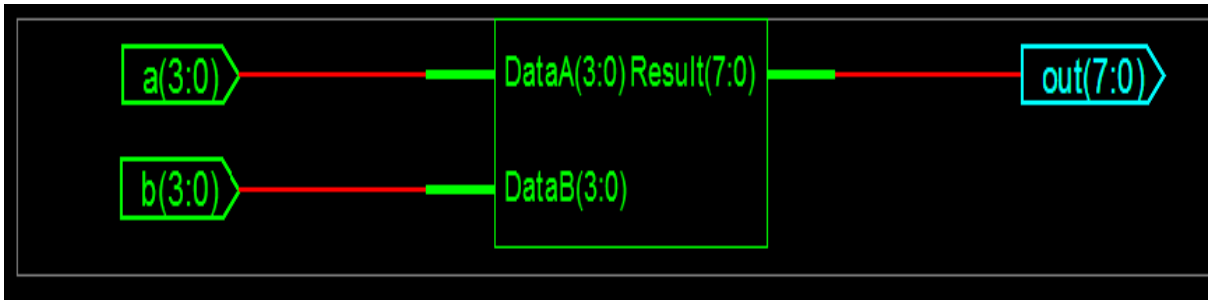
4 BIT MULTIPLIER:

```
module unsignedmult (out, a, b);
Output [7:0] out;
 Input [3:0] a;
 Input [3:0] b;
 Assign out=a*b;
 endmodule
```

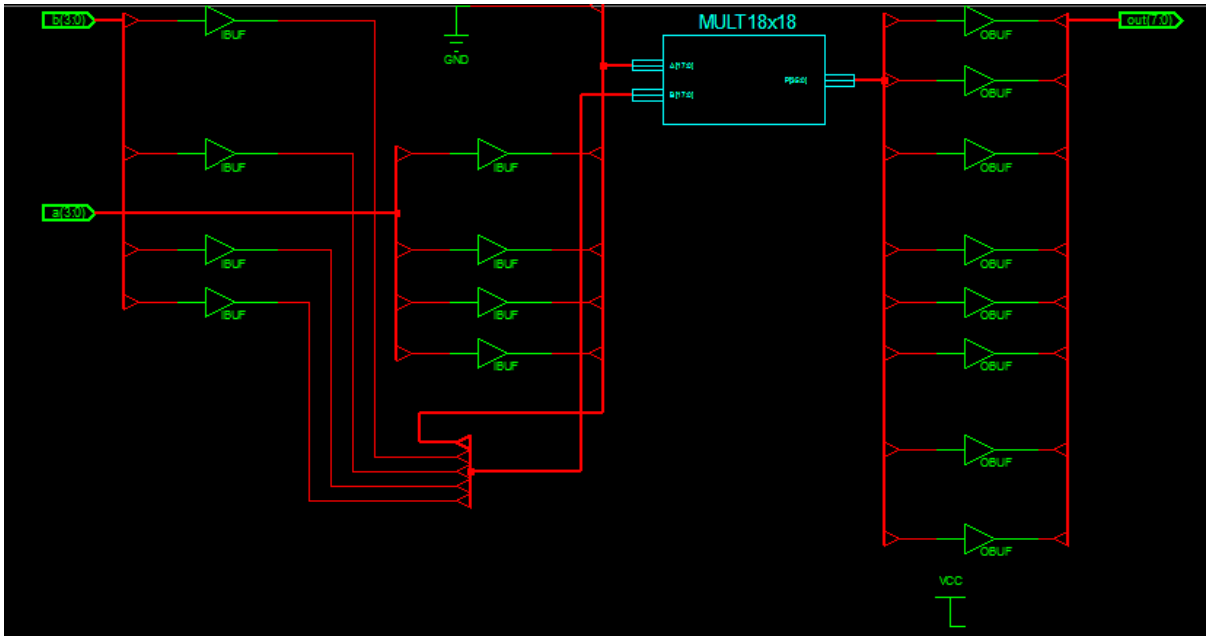
TRUTH TABLE:

A	B	RES
1000	1001	1000

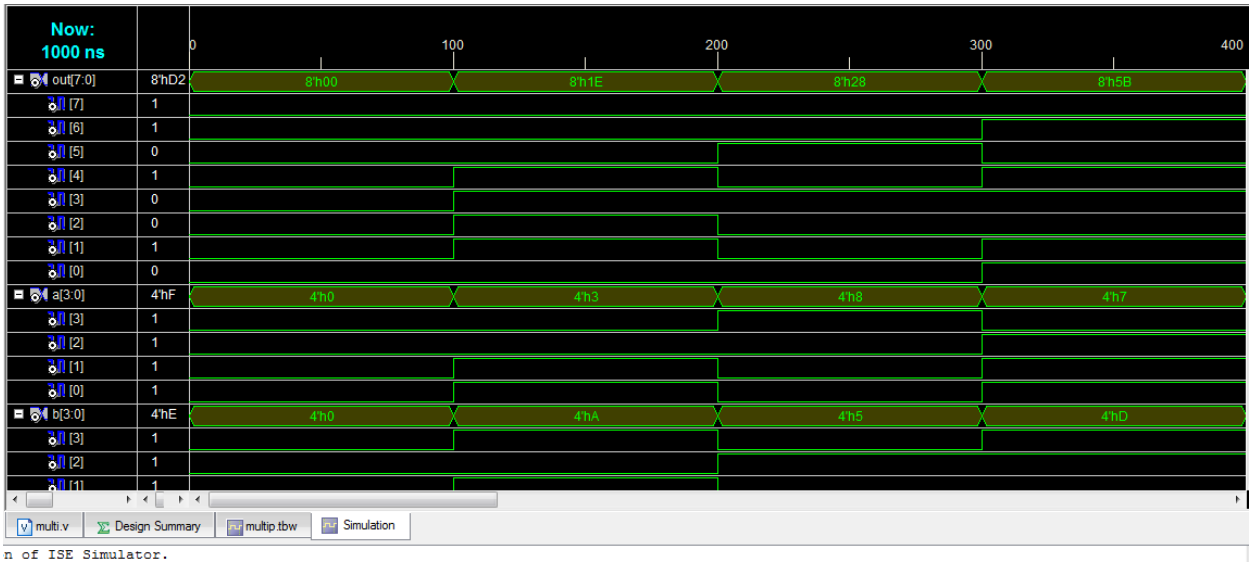
RTL SCHEMATIC:



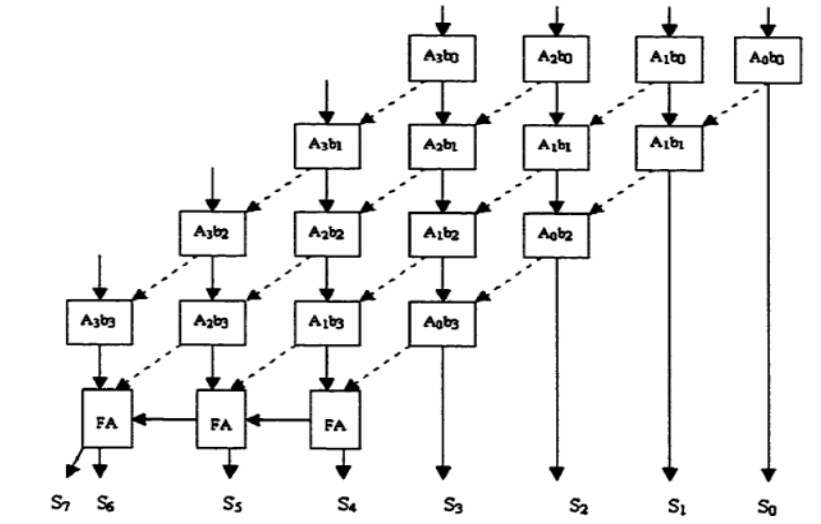
TECHNOLOGICAL SCHEMATIC:



OUTPUT WAVE:



ARRAY MULTIPLIER:

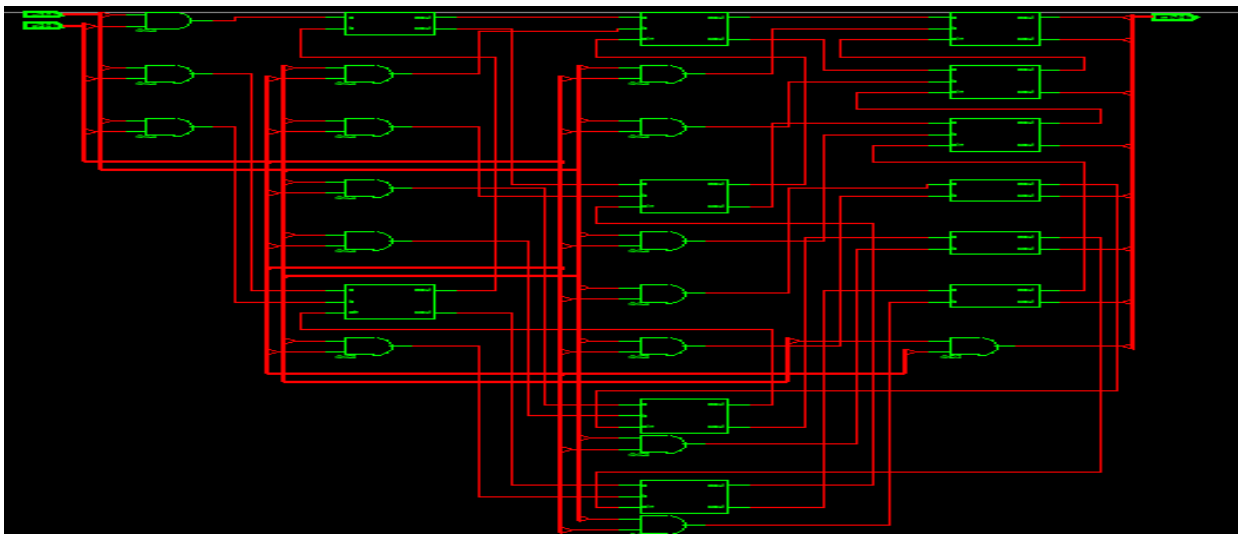


Program:

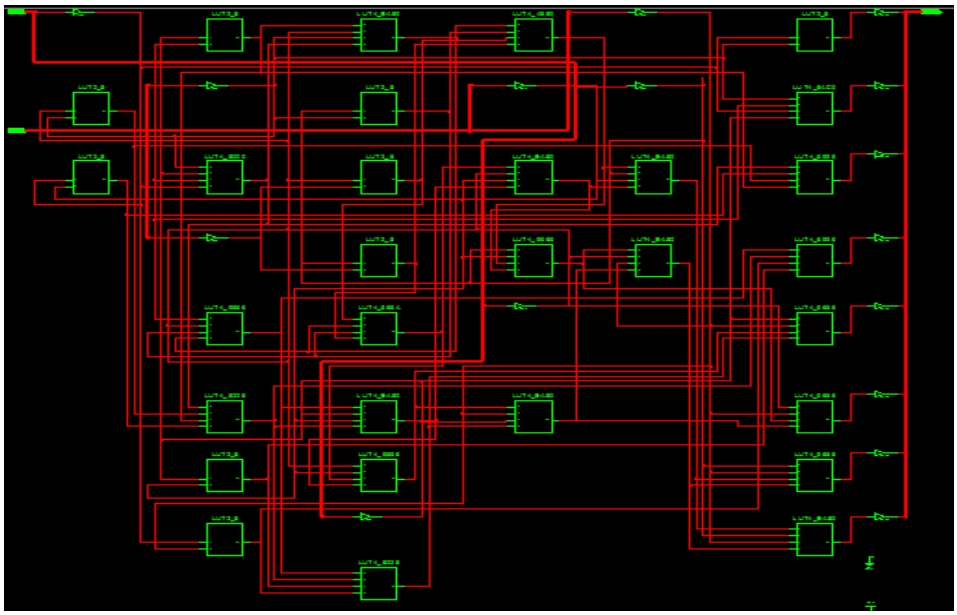
```
module  HA (sout,cout,a,b);
input  a,b;
output  sout,cout;
assign  sout=(a^b);
assign  cout=(a&b);
endmodule
```

```
module  FA (sout,cout,a,b,cin);
input  a,b,cin;
output  sout,cout;
assign  sout=(a^b^cin);
assign  cout=((a&b)|(b&cin)|(cin&a));
endmodule
module  bitmul  (m,x,y);
output  [7:0]m;
input  [3:0]x;
input  [3:0]y;
assign  m[0]=(x[0]&y[0]);
wire  x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,x13,x14,x15,x16,x17;
HA HA1 (m[1],x1,(x[1]&y[0]),(x[0]&y[1]));
FA FA1 (x2,x3,(x[1]&y[1]),(x[0]&y[2]),x1);
FA FA2 (x4,x5,(x[1]&y[2]),(x[0]&y[3]),x3);
HA HA2 (x6,x7,(x[1]&y[3]),x5);
HA HA3 (m[2],x15,x2,(x[2]&y[0]));
FA FA5 (x14,x16,x4,(x[2]&y[1]),x15);
FA FA4 (x13,x17,x6,(x[2]&y[2]),x16);
FA FA3 (x9,x8,x7,(x[2]&y[3]),x17);
HA HA4 (m[3],x12,x14,(x[3]&y[0]));
FA FA8 (m[4],x11,x13,(x[3]&y[1]),x12);
FA FA7 (m[5],x10,x9,(x[3]&y[2]),x11);
FA FA6 (m[6],m[7],x8,(x[3]&y[3]),x10);
endmodule
```

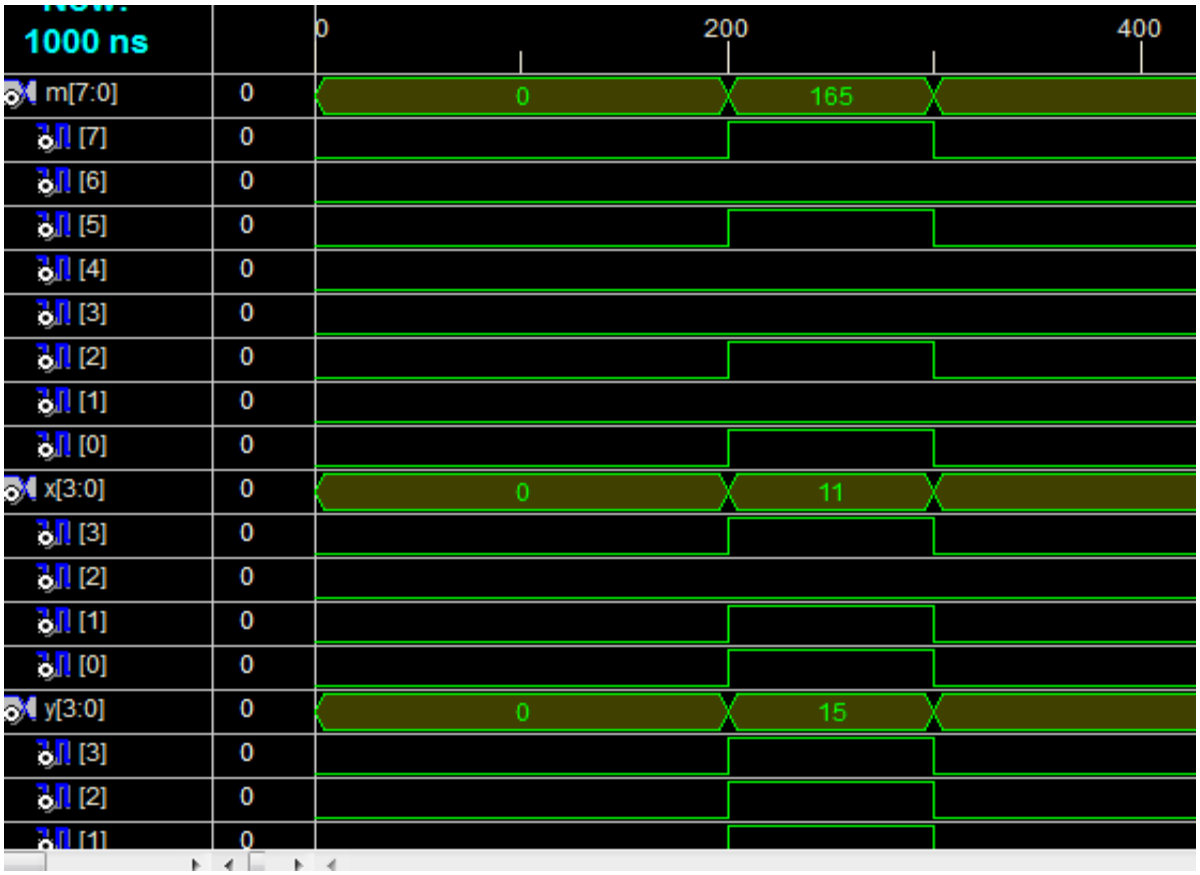
RTL SCHEMATIC:



TECHNOLOGICAL SCHEMATIC:



OUTPUT WAVEFORM:



Result:

Thus 4-bit multiplier (simple and array multiplier) was implemented successfully

Exp. No.: 7	**DESIGN & FPGA IMPLEMENTATION OF UP COUNTER AND DOWN COUNTER**

AIM:

To implement Counters using Verilog HDL

APPARATUS REQUIRED:

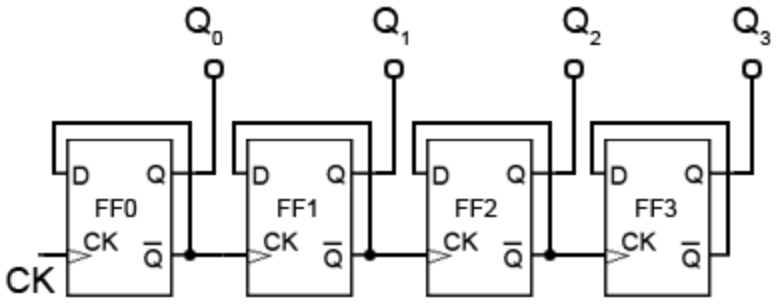
- PC with Windows XP.
- XILINX 9.2i
- FPGA-SPARTAN-3 KIT
- PARALLEL TO JTAG CABLE

.

ALGORITHM:

- New project and type the project name and check the top level source type as HDL
- Enter the device properties and click Next
- Click New Source And Select the Verilog Module and then give the file name
- Give the Input and Output port names and click finish.
- Type the Verilog program and save it
- Double click the synthesize XST and check syntax
- Simulate the waveform by behavioral simulation
- For implementation Select User constraints and give input and output port pin number
- Click Implement design for Translate, map and place & route
- Generate .bit file using programming file
- Implement in FPGA through parallel-JTAG cable
- Check the behavior of design in FPGA by giving inputs

UP COUNTER:

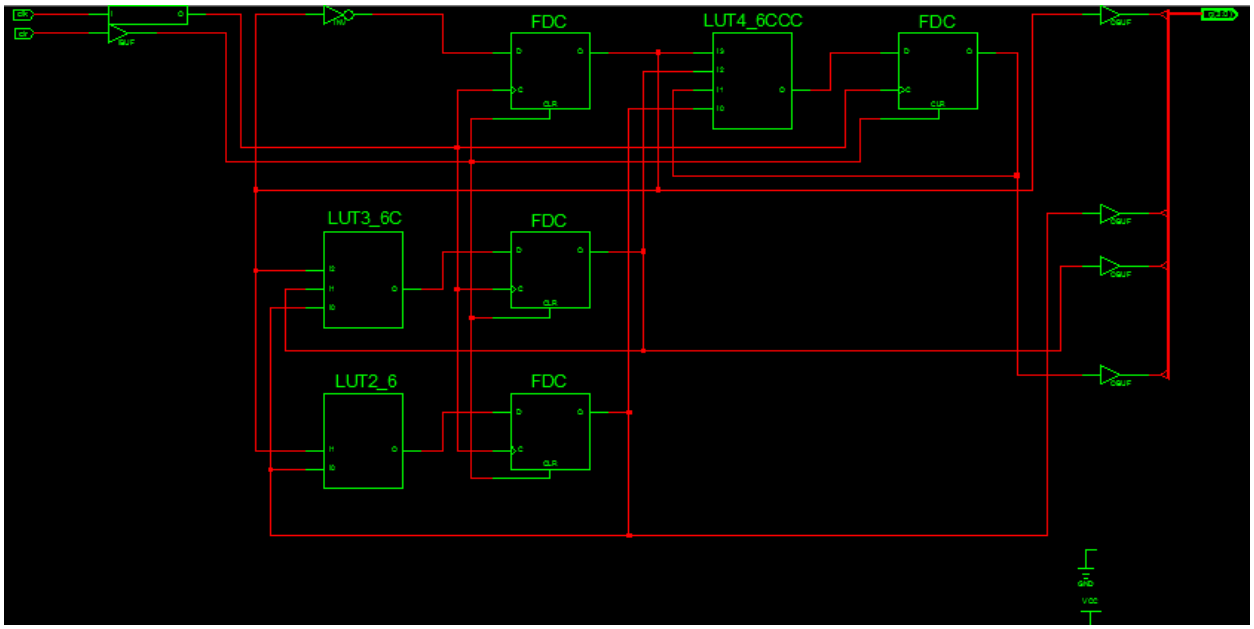


PROGRAM:

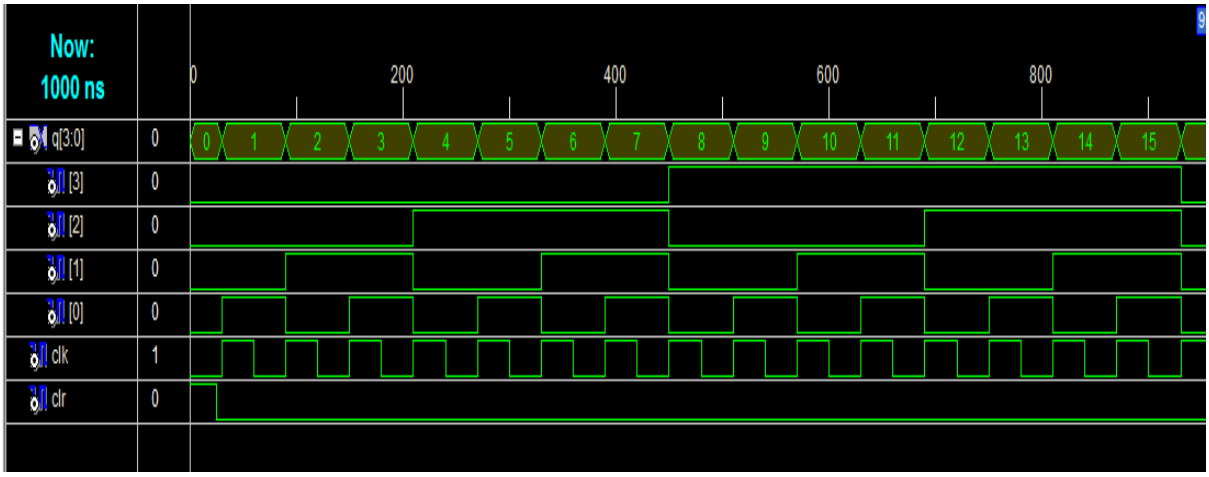
```
module upcounterr(clk,clr,q);
 input   clk, clr;
 output [3:0]q;
 reg   [3:0]tmp;
 always@(posedge  clk or posedge clr)
 begin
 if (clr)
 tmp <= 4'b0000;
 else
 tmp <= tmp + 1'b1;
 end
 assign q = tmp;
 endmodule
```

RTL SCHEMATIC:

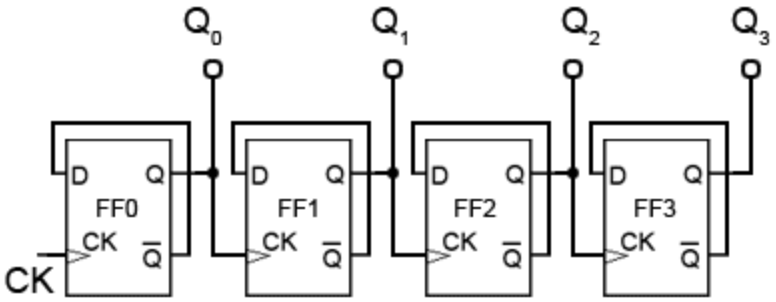
TECHNOLOGICAL SCHEMATIC:



OUTPUT WAVEFORM:



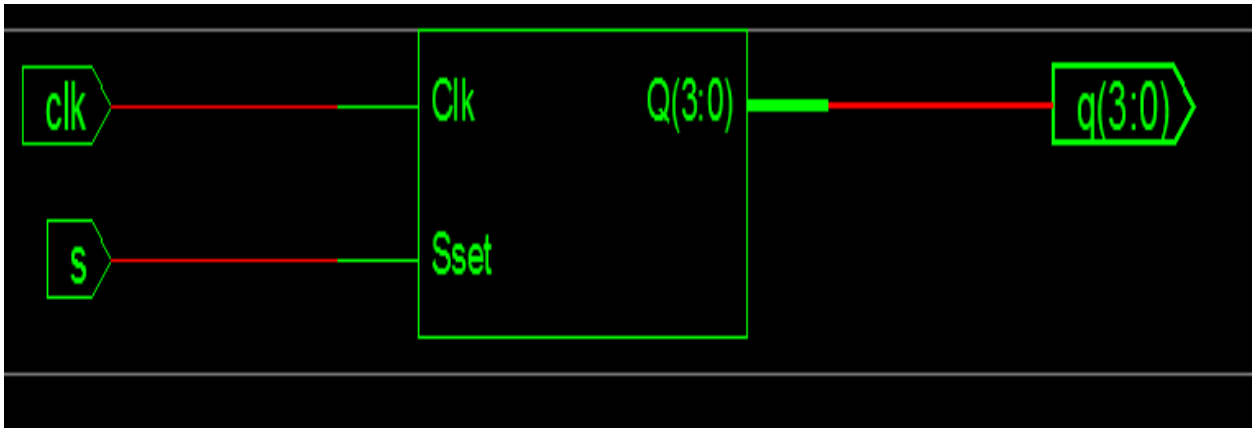
DOWN COUNTER:



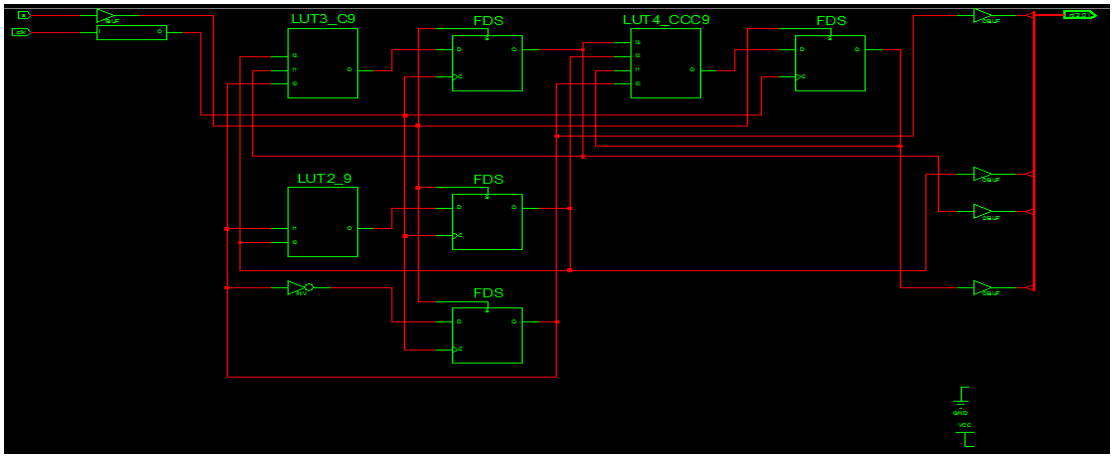
PROGRAM:

```
module downcounterr(clk,clr,q);
input clk, clr;
output [3:0]q;
reg [3:0]tmp;
always@(posedge clk or posedge clr)
begin
if (clr)
tmp <= 4'b1111;
else
tmp <= tmp - 1'b1;
end
assign q = tmp;
endmodule
```

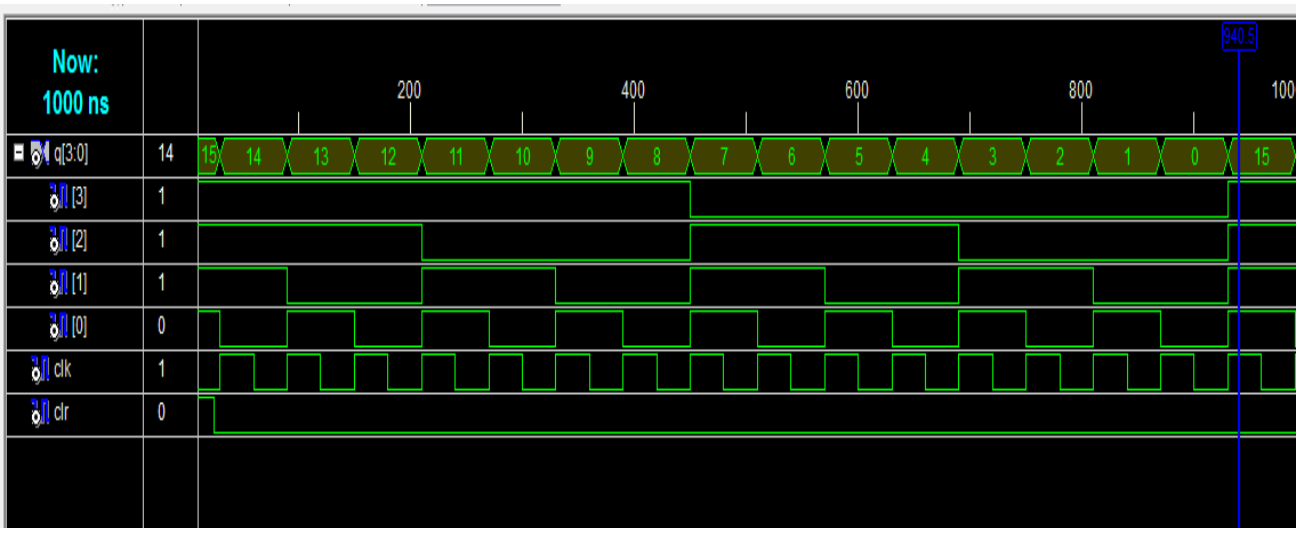
RTL SCHEMATIC:



TECHNOLOGICAL SCHEMATIC:



OUTPUT WAVEFORM:



RESULT:

Thus the up counter and down counter has been simulated and verified and implemented.

Exp. No.: 8	**DESIGN & FPGA IMPLEMENTATION OF FINITE STATE MACHINE (MOORE MACHINE)**

AIM:

To implement finite state machine (moore machine) using Verilog HDL

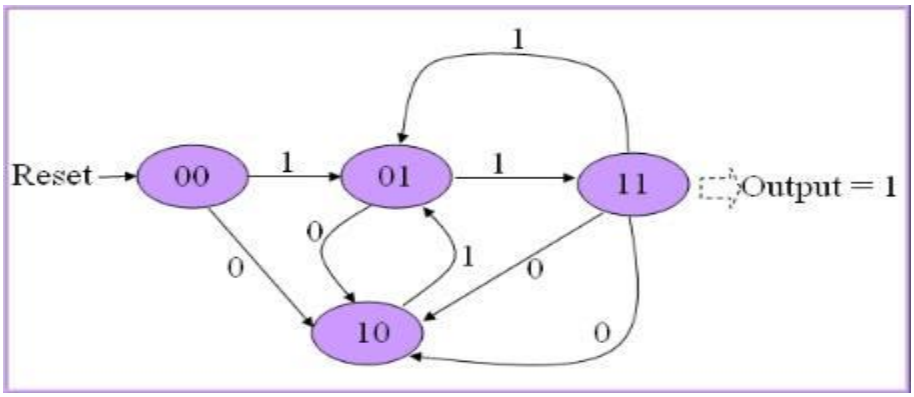
SOFTWARE REQUIRED:

- PC with Windows XP.
- XILINX 9.2i
- FPGA-SPARTAN-3 KIT
- PARALLEL TO JTAG CABLE

ALGORITHM:

- New project and type the project name and check the top level source type as HDL
- Enter the device properties and click Next
- Click New Source And Select the Verilog Module and then give the file name
- Give the Input and Output port names and click finish.
- Type the Verilog program and save it
- Double click the synthesize XST and check syntax
- Simulate the waveform by behavioral simulation
- For implementation Select User constraints and give input and output port pin number
- Click Implement design for Translate, map and place & route
- Generate .bit file using programming file
- Implement in FPGA through parallel-JTAG cable
- Check the behavior of design in FPGA by giving inputs

MOORE MACHINE:

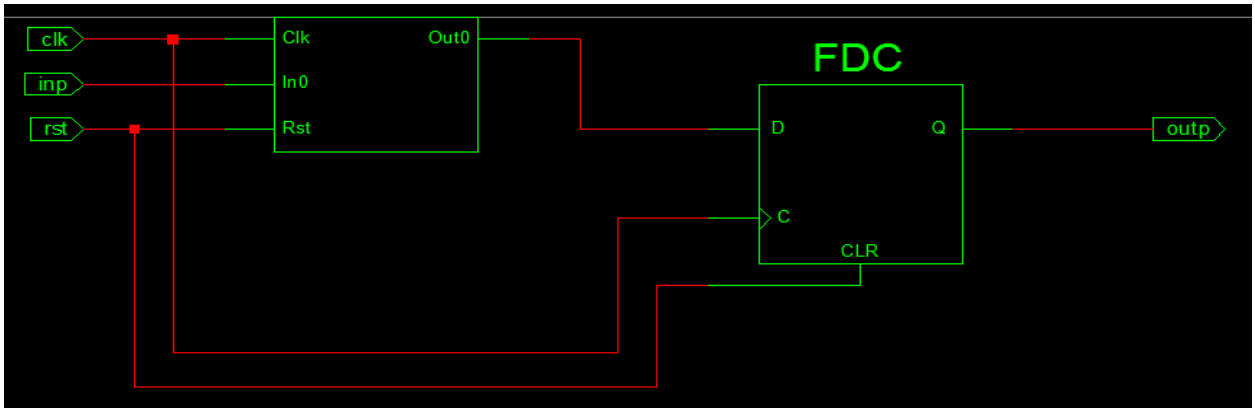


PROGRAM:

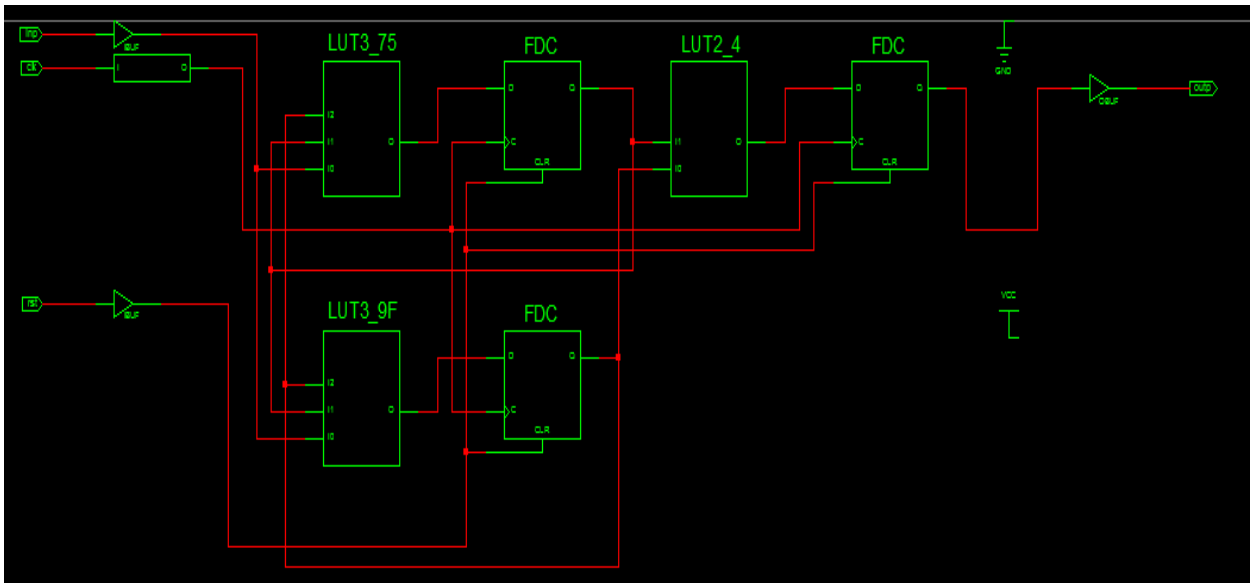
```
module  moore( clk, rst, inp, outp);

  input  clk, rst, inp;
  output  outp;

  reg [1:0]  state;
  reg outp;

  always  @( posedge clk, posedge rst )
  begin
  if( rst )
     state <= 2'b00;
  else

  begin

     case( state )

     2'b00:
     begin
        if( inp  ) state <= 2'b01;
        else state <= 2'b10;
     end

     2'b01:
     begin
        if( inp  ) state <= 2'b11;
        else state <= 2'b10;
     end

     2'b10:
     begin
        if( inp  ) state <= 2'b01;
        else state <= 2'b11;
     end

     2'b11:
     begin
        if( inp  ) state <= 2'b01;
        else state <= 2'b10;
     end
     endcase
  end
end
always @(posedge clk, posedge rst)
begin
   if( rst )
     outp <= 0;
```

```
    else if( state == 2'b11 )
      outp <= 1;
    else outp <= 0;

end

endmodule
```

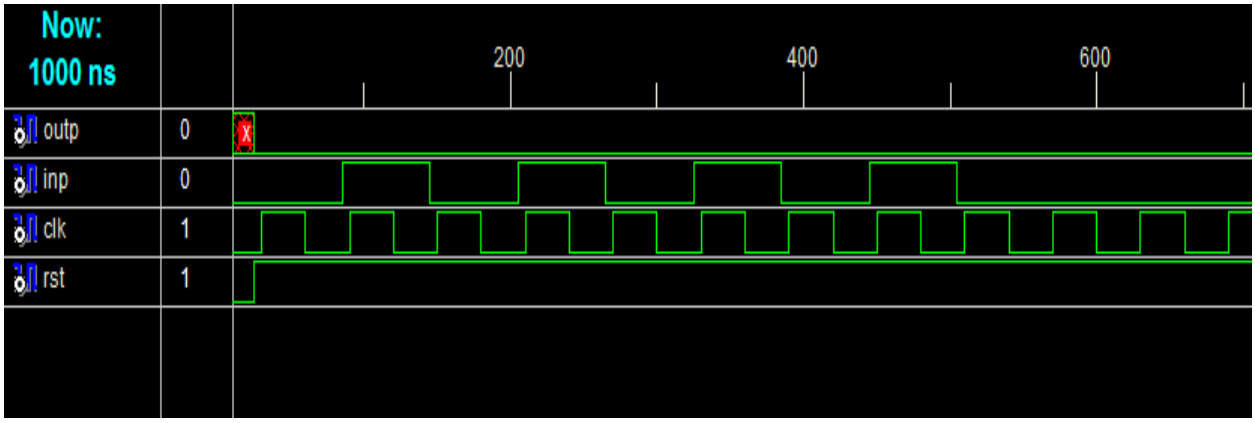
RTL SCHEMATIC:



TECHNOLOGICAL SCHEMATIC:



OUTPUT WAVEFORM:



RESULT:

Thus the finite state machine (moore machine) has been simulated and verified and implemented.

Exp. No.: 9	**DESIGN & FPGA IMPLEMENTATION OF FINITE STATE MACHINE (MEALY MACHINE)**

AIM:

To implement finite state machine (mealy machine) using Verilog HDL

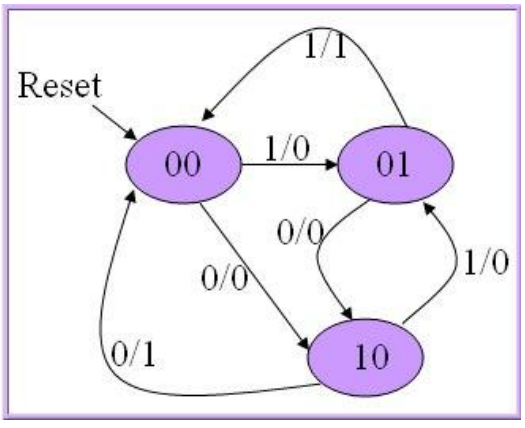
SOFTWARE REQUIRED:

- PC with Windows XP.
- XILINX 9.2i
- FPGA-SPARTAN-3 KIT
- PARALLEL TO JTAG CABLE

ALGORITHM:

- New project and type the project name and check the top level source type as HDL
- Enter the device properties and click Next
- Click New Source And Select the Verilog Module and then give the file name
- Give the Input and Output port names and click finish.
- Type the Verilog program and save it
- Double click the synthesize XST and check syntax
- Simulate the waveform by behavioral simulation
- For implementation Select User constraints and give input and output port pin number
- Click Implement design for Translate, map and place & route
- Generate .bit file using programming file
- Implement in FPGA through parallel-JTAG cable
- Check the behavior of design in FPGA by giving inputs

MEALY MACHINE: State diagram

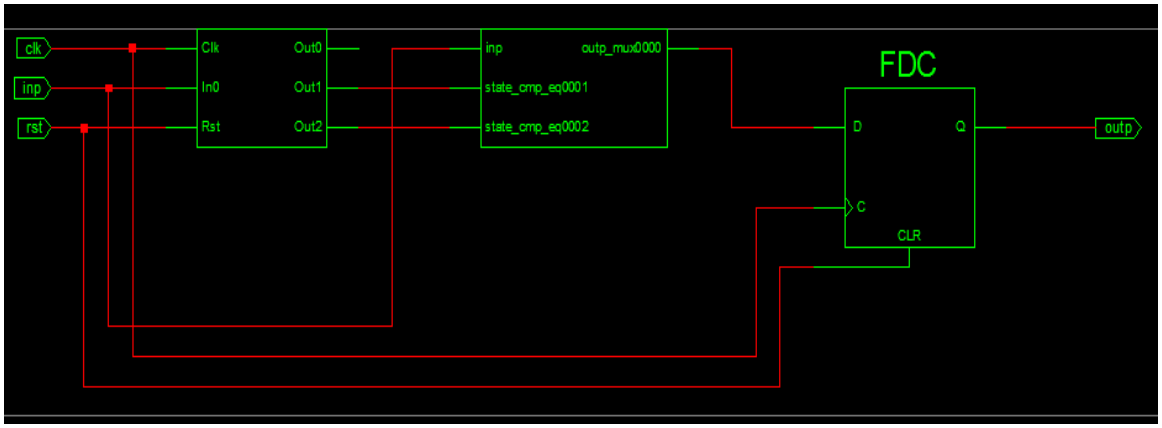


PROGRAM:

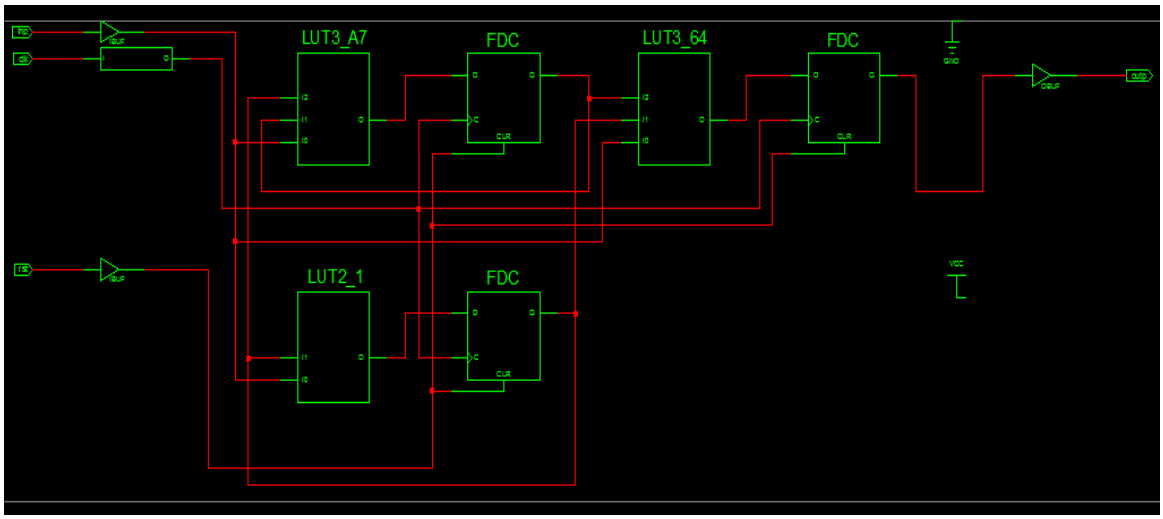
```
module mealy( clk, rst, inp, outp);

  input clk, rst, inp;
  output outp;

  reg [1:0] state;
  reg outp;

  always @( posedge clk, posedge rst ) begin
  if( rst ) begin
     state <= 2'b00;
     outp <= 0;
  end

  else  begin

     case( state )
     2'b00: begin
         if( inp ) begin
           state <= 2'b01;
           outp  <= 0;
         end
         else begin
            state <= 2'b10;
            outp <= 0;
         end
     end

     2'b01: begin
         if( inp ) begin
            state <= 2'b00;
            outp  <= 1;
         end
         else  begin
           state <= 2'b10;
           outp <= 0;
         end
     end

     2'b10: begin
         if( inp ) begin
            state <= 2'b01;
            outp  <= 0;
```

```
        end
        else begin
          state <= 2'b00;
          outp <= 1;
        end

    end

    default: begin
        state <= 2'b00;
        outp <= 0;
    end
   endcase
  end
end

endmodule
```

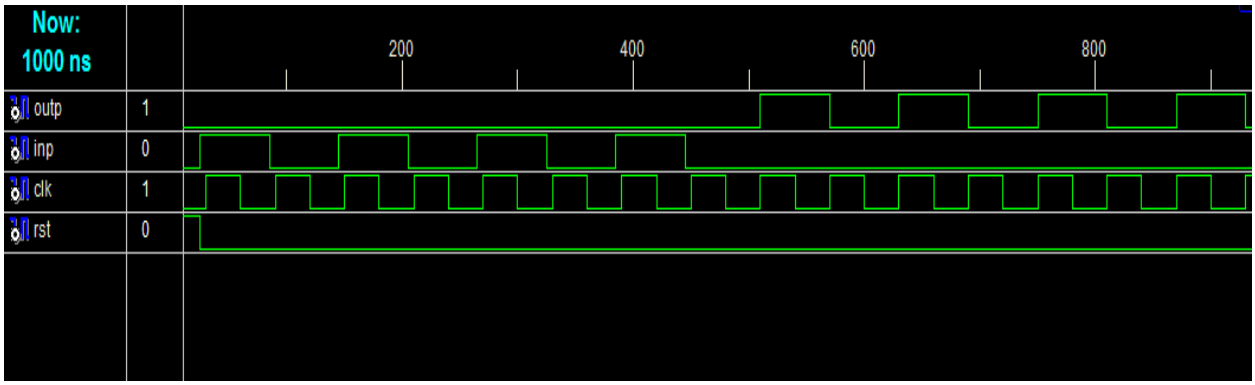
RTL SCHEMATIC:



TECHNOLOGICAL SCHEMATIC:



OUTPUT WAVEFORM:



RESULT:

Thus the finite state machine (Mealy machine) has been simulated and verified and implemented.

Exp. No.: 10	**LAYOUT EXTRACTION AND SIMULATION OF C-MOS INVERTOR**

AIM:

To draw the layout of an CMOS inverter

SOFTWARE USED:

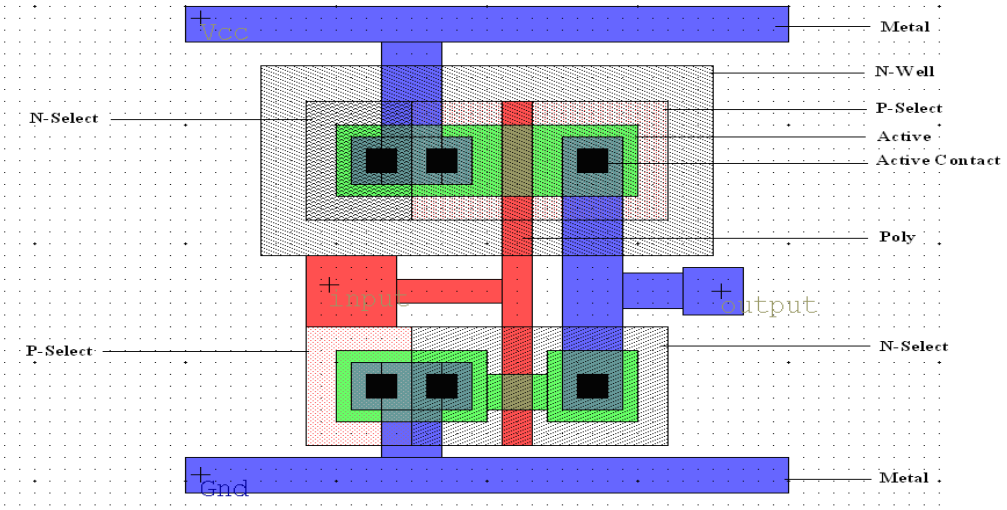
- **Microwind**
- **DSCH**

DESCRIPTION:

CMOS INVERTER:

The NMOS transistor and the PMOS transistor form a typical complementary MOS (CMOS) device. When a low voltage (0 V) is applied at the input, the top transistor (P-type) is conducting (switch closed) while the bottom transistor behaves like an open circuit. Therefore, the supply voltage (5 V) appears at the output. Conversely, when a high voltage (5 V) is applied at the input, the bottom transistor (N-type) is conducting (switch closed) while the top transistor behaves like an open circuit. Hence, the output voltage is low (0 V).

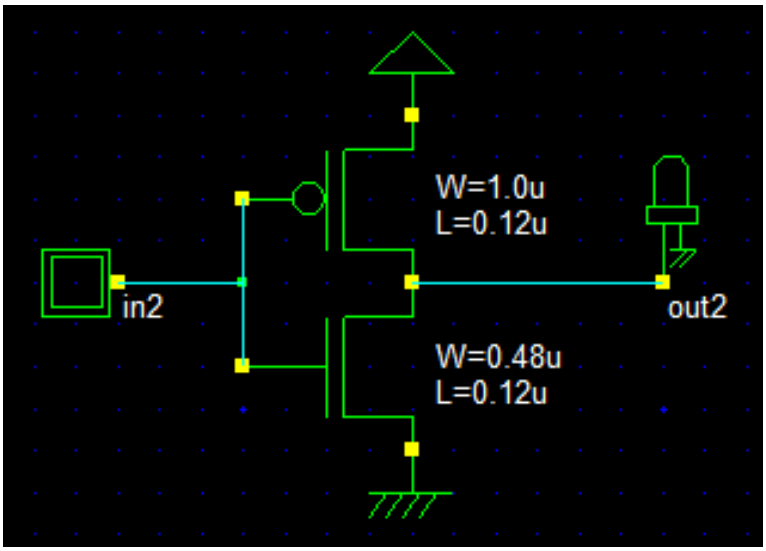
LAYOUT DIAGRAM



ALGORITHM:

- Open the DSCH2
- Drag the components like pmos,nmos,voltage source, ground, and LED from the symbol library.
- Connect the circuit as in the circuit diagram.
- Save the circuit & run the simulation
- Make verilog file go to Microwind and compile the verilog file saved in DSCH2
- Compile it and obtain the layourt diagram & draw the waveform

CIRCUIT DIAGRAM:



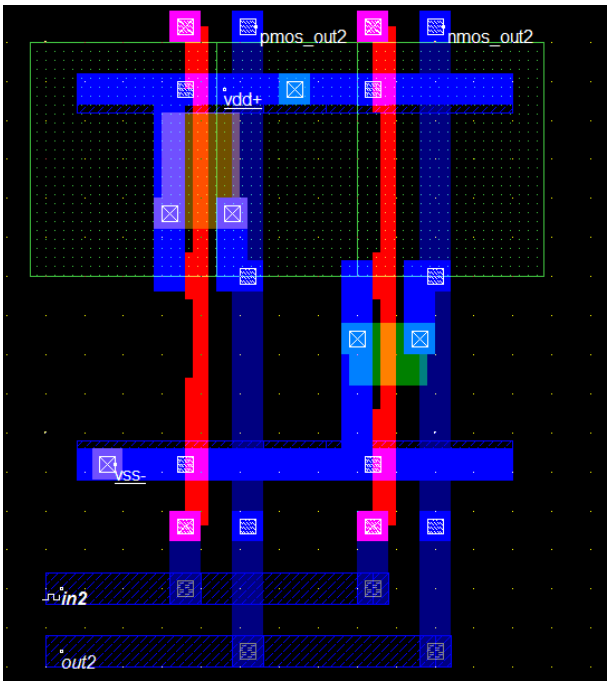
MOS LAYOUT

We use MICROWIND2 to draw the MOS layout and simulate its behavior. Go to the directory in which the software has been copied (By default MICROWIND2). Double-click on the MicroWind2 icon. The MICROWIND2 display window includes four main windows: the main menu, the layout display window,the icon menu and the layer palette. The layout window features a grid, scaled in lambda (□) units. The lambda unit is fixed to half of the minimum available lithography of the technology. The default technology is a CMOS 6-metal layers 0.25µm technology, consequently lambda is 0.125 µm.

Verilog code:

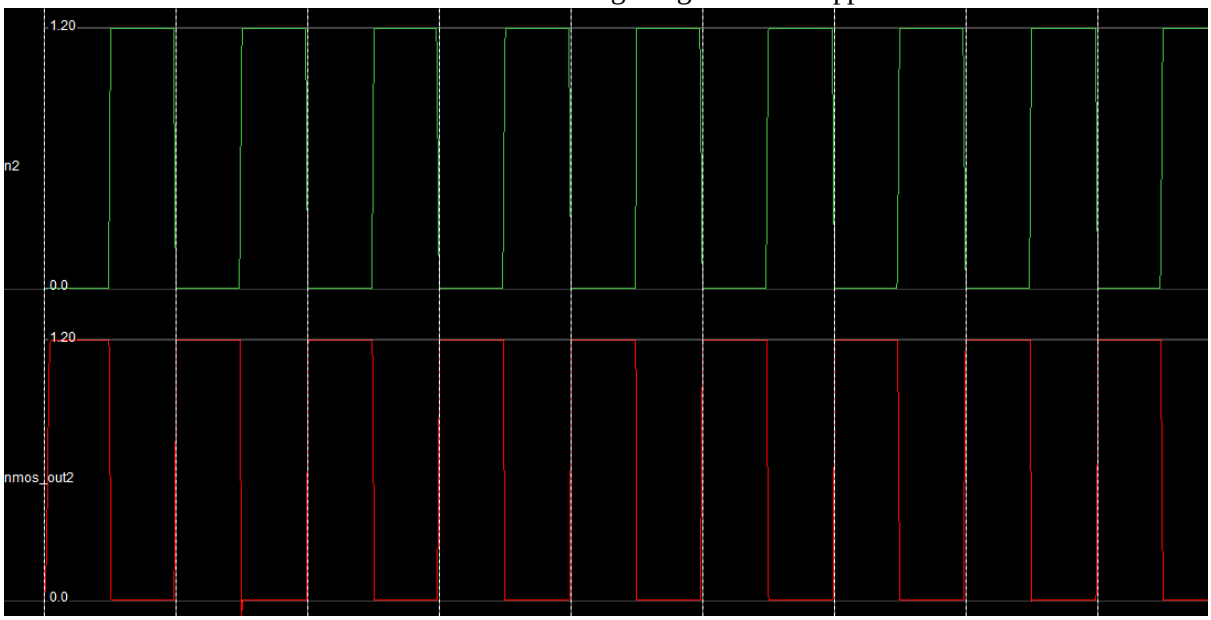
```
module cmosInv( in2,out2)
input in2;
output out2;
pmos #(17) pmos(out2,vdd,in2); // 1.0u 0.12u
nmos #(114) nmos(out2,vss,in2); // 0.48u 0.12u
endmodule
```

LAYOUT FOR C-MOS INVERTOR:



ANALOG SIMULATION:

Click on **Simulate à Start Simulation**. The timing diagrams will appears as follows



RESULT:

Thus the Layout design of a CMOS inverter has been drawn, verified and timing analysis perform.

Exp. No.: 11	LAYOUT EXTRACTION AND SIMULATION OF C-MOS NAND AND NOR GATE

AIM:

To design and simulate the cmos NAND and NOIR gate circuit.

SOFTWARE USED

- **Microwind**
- **DSCH**

THEORY:

NAND and NOR gates are known as universal gates as any function can be implemented with them

NAND functionality can be implemented by parallel combination of PMOS and series combination of NMOS transistor. When any one of the inputs is zero, then the output will be one and when both the inputs are one the output will be low.

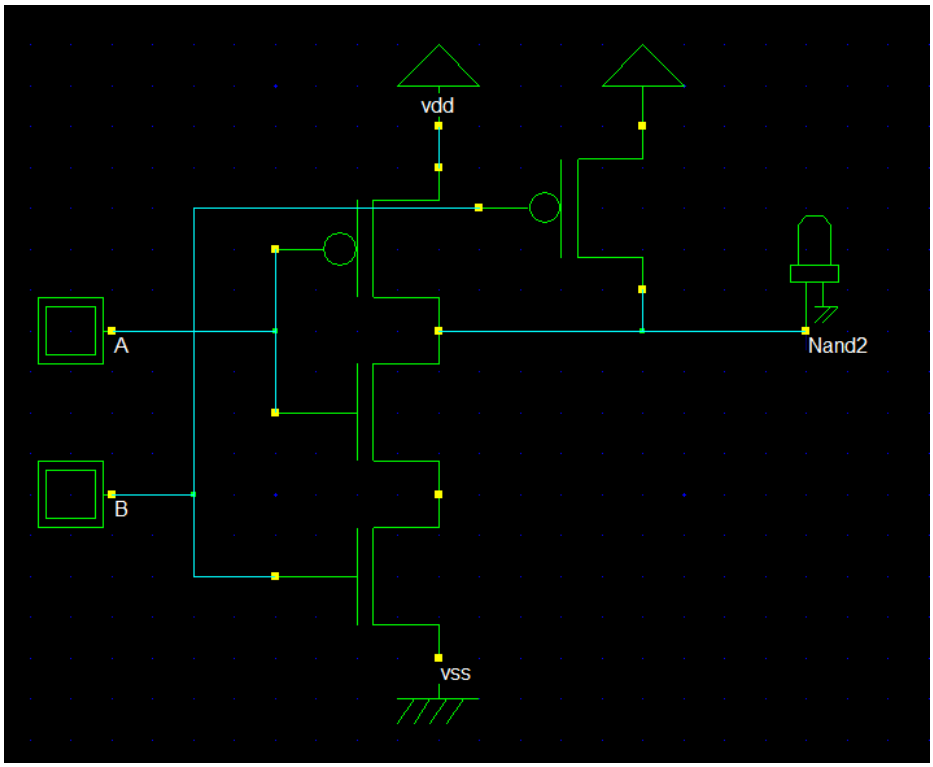
NOR functionality can be implemented by parallel combination of NMOS and series combination of PMOS transistor. When any one of the inputs is one, then the output will be one and when both the inputs are zero the output will be low.

ALGORITHM:

- Open the DSCH2
- Drag the components like pmos,nmos,voltage source, ground, and LED from the symbol library.
- Connect the circuit as in the circuit diagram.
- Save the circuit & run the simulation
- Make verilog file go to Microwind and compile the verilog file saved in DSCH2
- Compile it and obtain the layourt diagram & draw the waveform

CIRCUIT DIAGRAM:

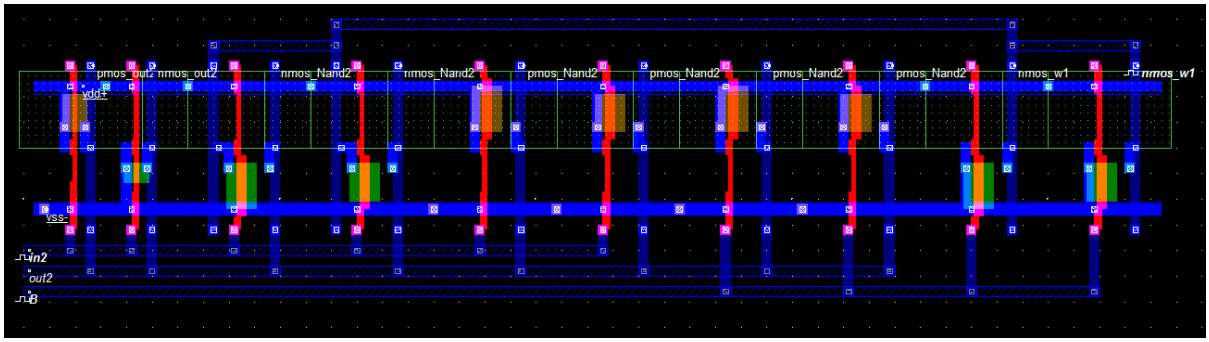
NAND gate:



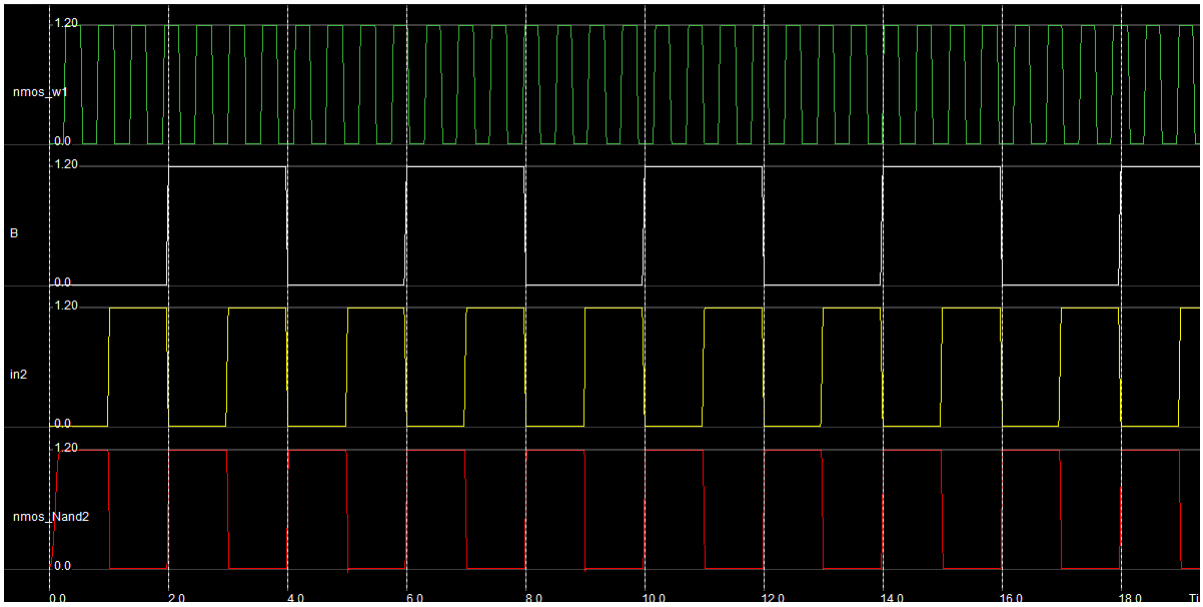
Verilog code:

```
module  cmosNand2(  A,B,Nand2);
 input  A,B;
 output  Nand2;
 nmos  #(121) nmos(Nand2,w1,A);  // 2.0u 0.25u
 pmos  #(121) pmos(Nand2,vdd,A);  // 2.0u 0.25u
 pmos  #(121) pmos(Nand2,vdd,B);  // 2.0u 0.25u
 nmos  #(107) nmos(w1,vss,B);  // 2.0u 0.25u
endmodule
```

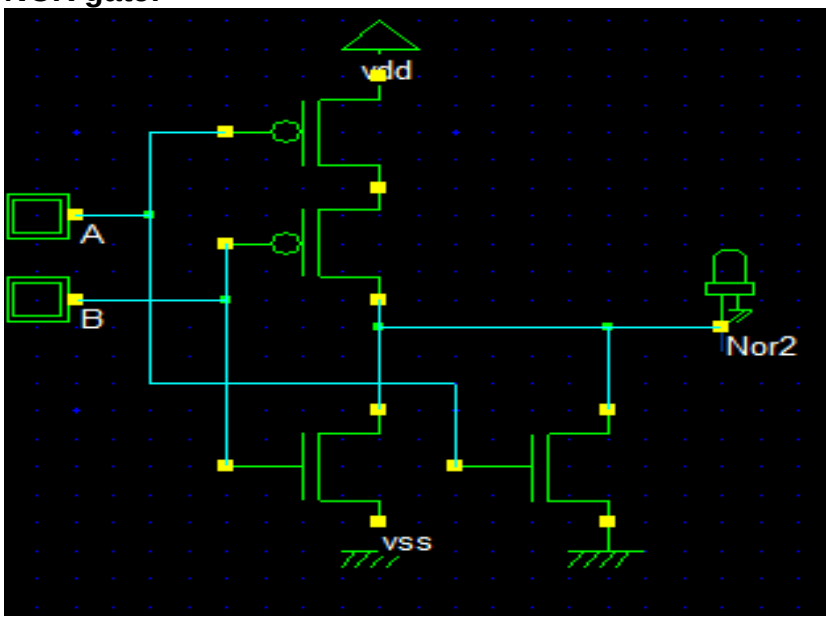
Layout:



Waveform:



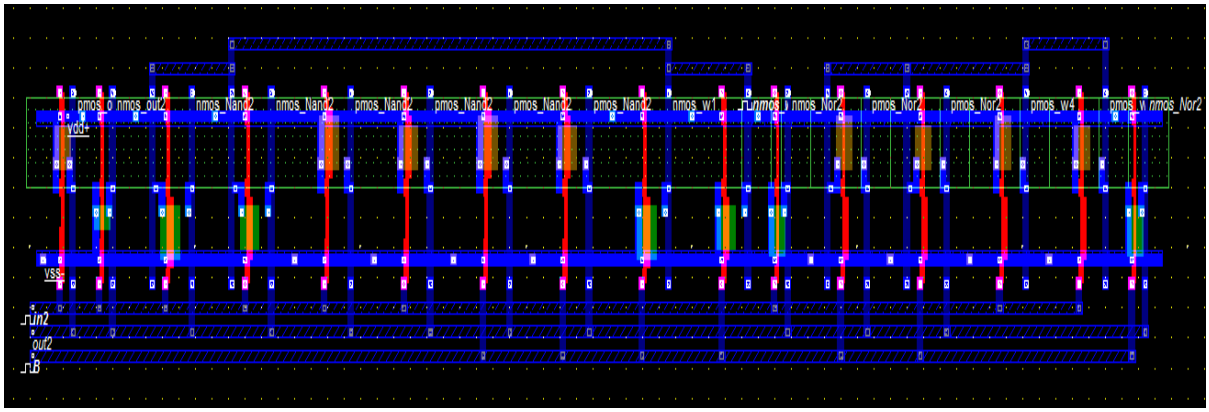
NOR gate:



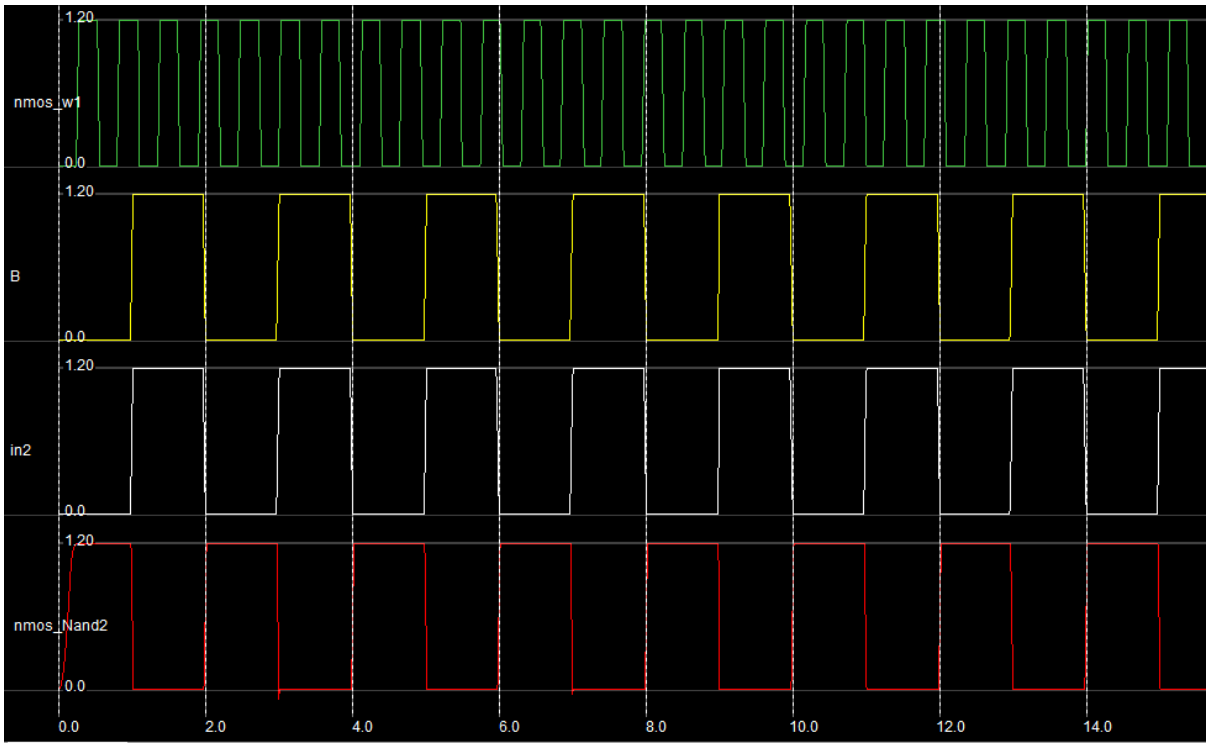
Verilog code:

```
module nor2Cmos( B,A,Nor2);
 input B,A;
 output Nor2;
 nmos #(121) nmos(Nor2,vss,A); // 1.0u 0.12u
 pmos #(121) pmos(Nor2,w4,B); // 2.0u 0.12u
 pmos #(107) pmos(w4,vdd,A); // 2.0u 0.12u
 nmos #(121) nmos(Nor2,vss,B); // 1.0u 0.12u
endmodule
```

Layout:



Waveform:



RESULT:

Thus the Layout design of a CMOS NAND and NOR gate has been drawn, verified and timing analysis performed

Exp. No.: 12	**LAYOUT EXTRACTION AND SIMULATION OF C-MOS DIFFERENTIAL AMPLIFIER**

AIM:

To design and simulate the emitter follower and differential amplifier circuit.

SOFTWARE USED

- **Microwind**
- **DSCH**

THEORY:

Differential amplifier:

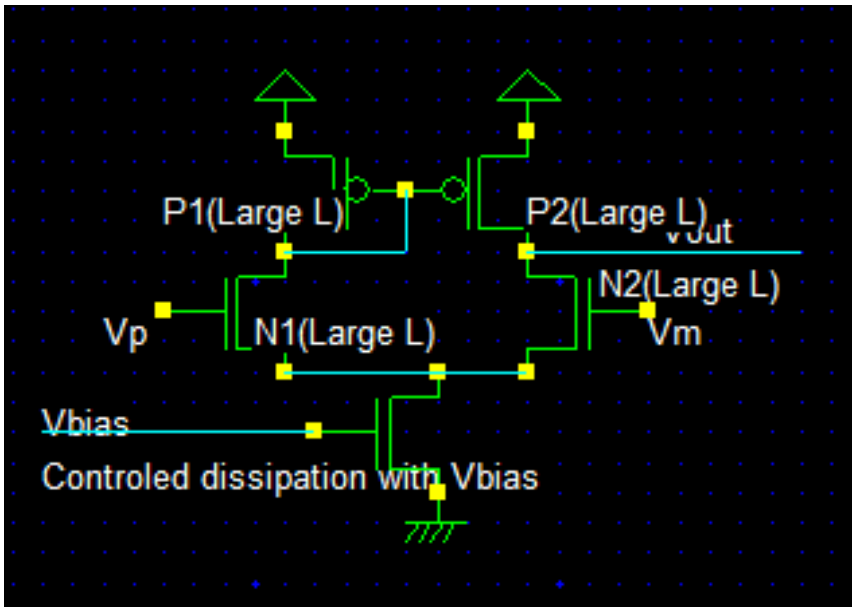
Differential Amplifier amplifies the current with very little voltage gain. It consists of two FETs connected so that the FET sources are connected together. The common source is connected to a large voltage source through a large resistor Re, forming the "long tail" of the name, the long tail providing an approximate constant current source. The higher the resistance of the current source Re, the lower Ac is, and the better the CMRR. In more sophisticated designs, a true (active) constant current source may be substituted for the long tail. The output from a differential amplifier is itself often differential.

ALGORITHM:

- Open the DSCH2
- Drag the components like pmos,nmos,voltage source, ground, and LED from the symbol library.
- Connect the circuit as in the circuit diagram.
- Save the circuit & run the simulation
- Make verilog file go to Microwind and compile the verilog file saved in DSCH2
- Compile it and obtain the layourt diagram & draw the waveform

CIRCUIT DIAGRAM:

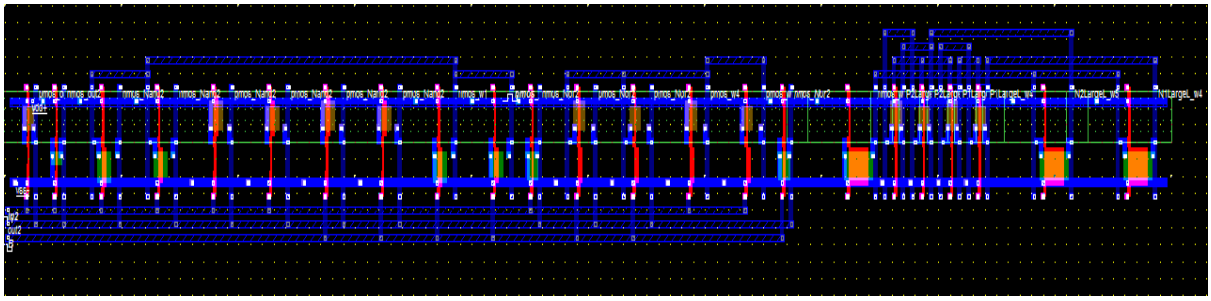
DIFFERENTIAL AMPLIFIER:



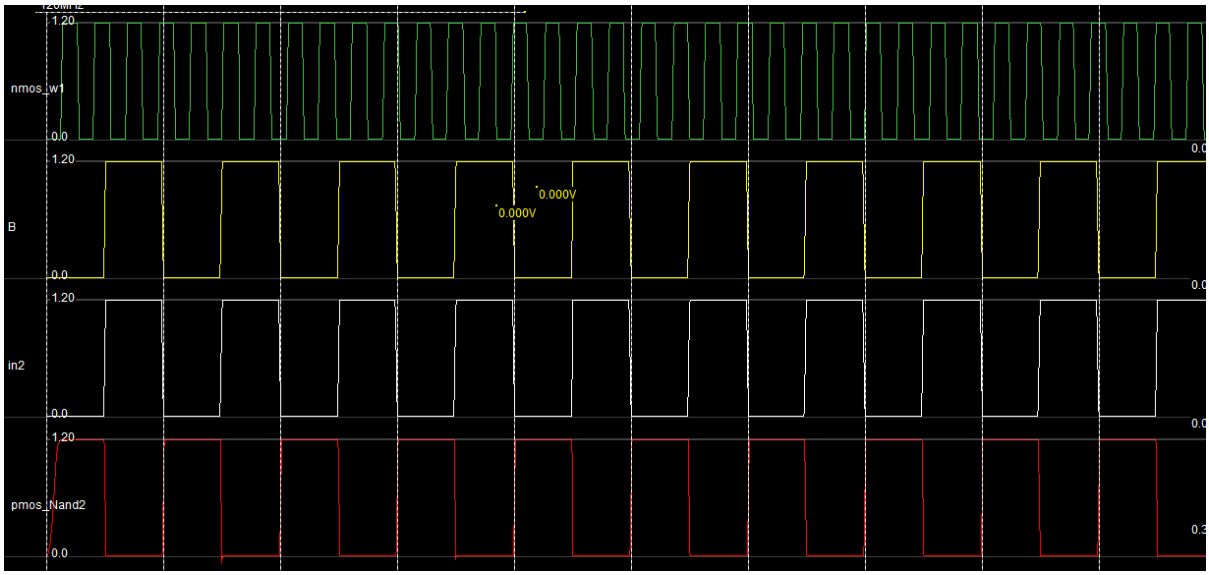
Verilog code:

```
module  diff amp( );
 nmos #(17) nmos(w2,vss,w1);  // 1.0u 1u
 pmos #(10) P2LargeL(w5,w3,w4);  // 2.0u 0.12u
 pmos #(24) P1LargeL(w4,w6,w4);  // 2.0u 0.12u
 nmos #(10) N2LargeL(w5,w2,w7);  // 1.0u 1u
 nmos #(24) N1LargeL(w4,w2,w8);  // 1.0u 1u
endmodule
```

LAYOUT:



WAVEFORM:



RESULT:

Thus the Layout design of a differential amplifier has been drawn, verified and timing analysis performed

Exp. No.: 13	DESIGN OF CMOS INVERTER USING TANNER

AIM

To design a CMOS inverter using the Schematic entry tool, Tanner and verify its functioning.

APPARATUS REQUIRED:

1. Tanner tool
2. PC

THEORY:

CMOS Inverter consists of nMOS and pMOS transistor in series connected between VDD and GND. The gate of the two transistors are shorted and connected to the input. When the input to the inverter A = 0, nMOS transistor is OFF and pMOS transistor is ON. The output is pull-up to VDD. When the input A = 1, nMOS transistor is ON and pMOS transistor is OFF. The Output is Pull-down to GND.

ALGORITHM

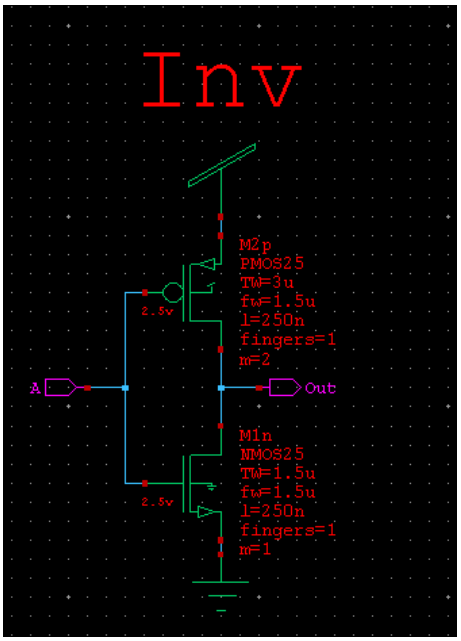
1. Draw the schematic of CMOS Inverter using S-edit.

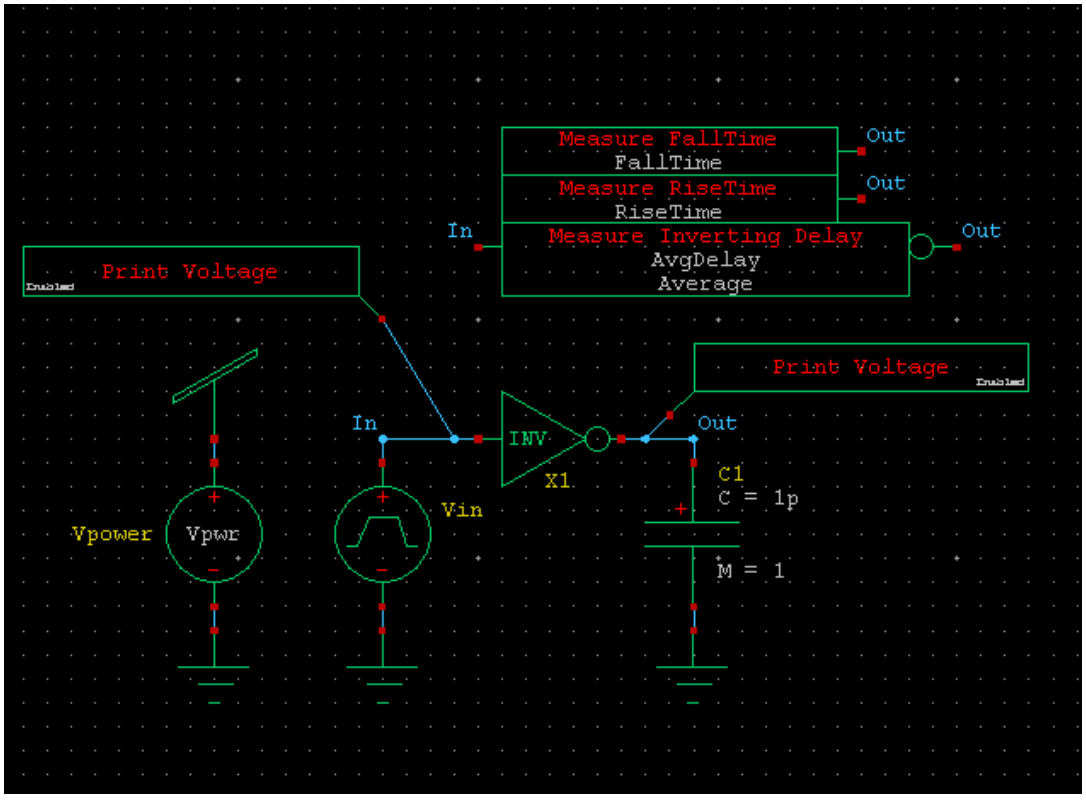
2. Perform Transient Analysis of the CMOS Inverter.

3. Obtain the output waveform from W-edit.

4. Obtain the spice code using T-edit.

CMOS Inverter

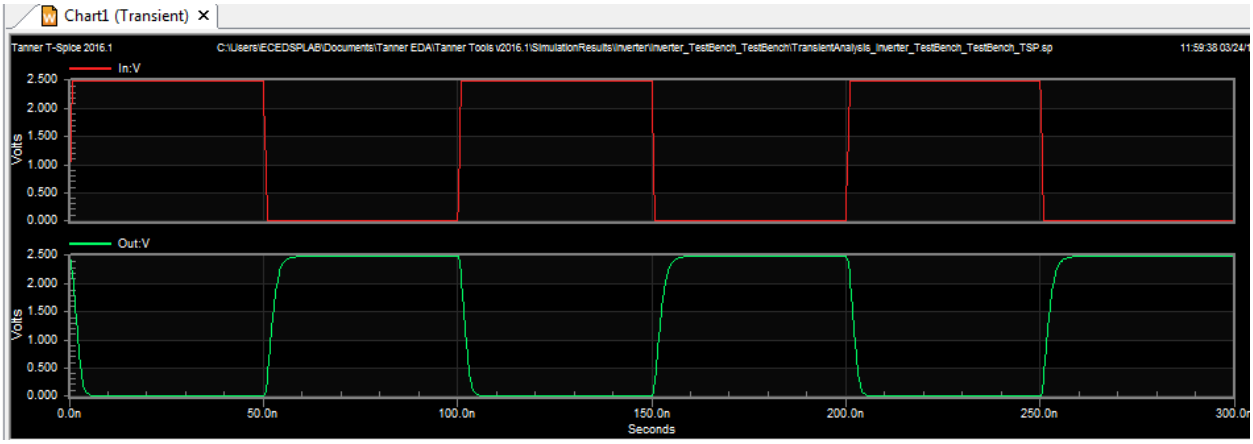




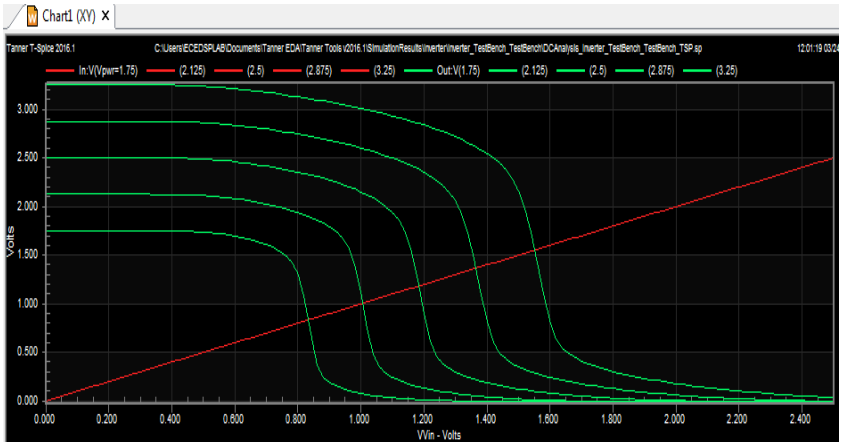
]

WAVEFORM

Transient Analysis



Dc analysis



OUTPUT

\

TSPICE - Tanner SPICE
Version 7.10
Copyright (c) 1993-2001 Tanner Research, Inc.

Parsing "C:\Tanner\S-Edit\Module011.sp"

Including "C:\Tanner\TSpice70\models\ml2_125.md"

Device and node counts:

MOSFETs - 2 BJTs - 0 MESFETs - 0 Capacitors - Inductors - 0
MOSFET geometries - 2
JFETs - 0
Diodes - 0
Resistors - 0

Mutual inductors - 0
Voltage sources - 2
VCVS - 0
CCVS - 0
V-control switch - 0
Macro devices - 0
Subcircuits - 0
Independent nodes - 1
Total nodes - 4

Current sources - 0

VCCS - 0

CCCS - 0

I-control switch - 0

Functional model instances - 0

Subcircuit instances - 0

Boundary nodes - 3

Parsing 0.01 seconds

Setup 0.00 seconds

DC operating point 0.00 seconds

Transient Analysis 0.04 seconds

Total 0.05 seconds

RESULT

Thus the design & simulation of a CMOS inverter has been carried out using S-Edit of Tanner EDA Tools

Exp. No.: 14	DESIGN OF CMOS NAND GATE AND NOR GATE USING TANNER

AIM

To design a CMOS NAND and NOR gates using the Schematic entry tool, Tanner and verify its functioning.

APPARATUS REQUIRED:

1. Tanner tool
2. PC

THEORY:

NAND and NOR gates are known as universal gates as any function can be implemented with them

NAND functionality can be implemented by parallel combination of PMOS and series combination of NMOS transistor. When any one of the inputs is zero, then the output will be one and when both the inputs are one the output will be low.

NOR functionality can be implemented by parallel combination of NMOS and series combination of PMOS transistor. When any one of the inputs is one, then the output will be one and when both the inputs are zero the output will be low.

ALGORITHM

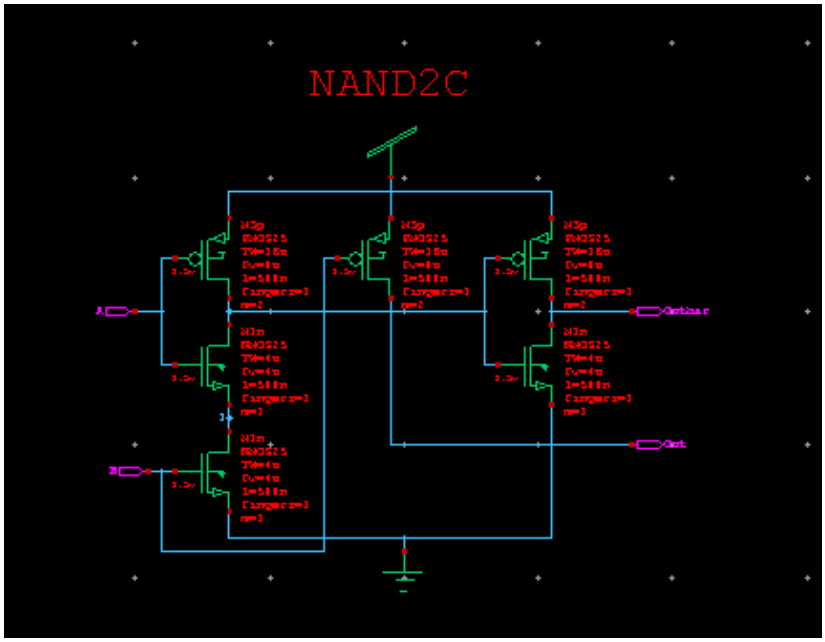
CMOS NAND

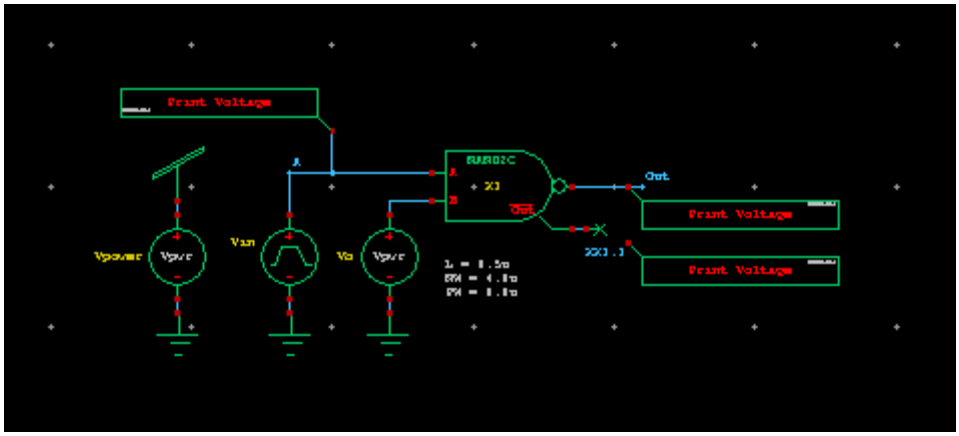
1. Draw the schematic of CMOS NAND using S-edit.
2. Perform Transient Analysis of the CMOS NAND.
3. Obtain the output waveform from W-edit.
4. Obtain the spice code using T-edit.

CMOS NOR

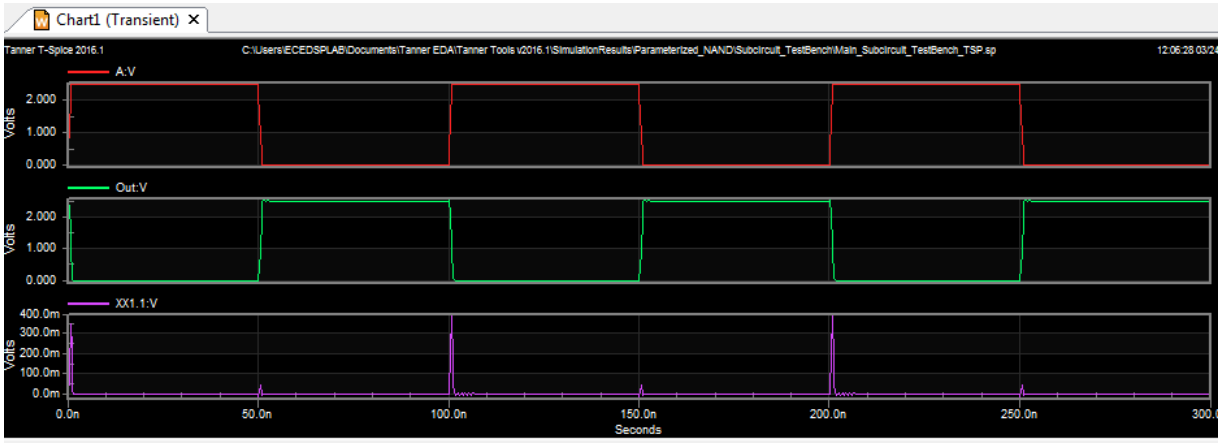
1. Draw the schematic of CMOS NOR using S-edit.
2. Perform Transient Analysis of the CMOS NOR.
3. Obtain the output waveform from W-edit.
4. Obtain the spice code using T-edit.

CMOS NAND Ciruit:





WAVEFORM NAND



OUTPUT - NAND

TSPICE - Tanner SPICE

Version 7.10

Parsing "F:\tanner\TSpice70\Module0.sp"

Including "F:\tanner\S-Edit\models\ml2_125.md"

Warning T-SPICE : DC sources have non-unique names. "vin".

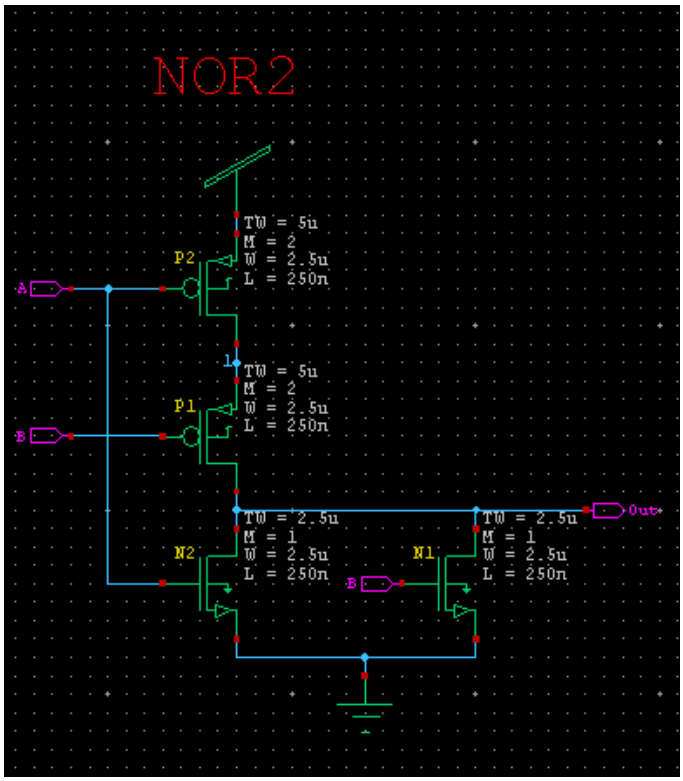
Device and node counts:
MOSFETs - 4 MOSFET geometries - 2
BJTs - 0 JFETs - 0
MESFETs - 0 Diodes - 0
Capacitors - 0 Resistors - 0
Inductors - 0 Mutual inductors - 0
Transmission lines - 0 Coupled transmission lines - 0
Voltage sources - 3 Current sources - 0
VCVS - 0 VCCS - 0
CCVS - 0 CCCS - 0
V-control switch - 0 I-control switch - 0
Macro devices - 0 Functional model instances - 0
Subcircuits - 0 Subcircuit instances - 0
Independent nodes - 2 Boundary nodes - 4
Total nodes - 6

*** 1 WARNING MESSAGES GENERATED

Parsing 0.00 seconds
Setup 0.00 seconds
DC operating point 0.01 seconds
Transient Analysis 0.03 seconds

Total 0.04 seconds

CMOS NOR Circuit



OUTPUT – NOR

TSPICE - Tanner SPICE
Version 7.10
Copyright (c) 1993-2001 Tanner Research, Inc.
Parsing "F:\tanner\TSpice70\Module0.sp"
Including "F:\tanner\S-Edit\models\ml2_125.md"
Warning T-SPICE : DC sources have non-unique names. "vin".

Device and node counts:
MOSFETs - 4 MOSFET geometries - 2
BJTs - 0 JFETs - 0
MESFETs - 0 Diodes - 0
Capacitors - 0 Resistors - 0
Inductors - 0 Mutual inductors - 0
Transmission lines - 0 Coupled transmission lines - 0
Voltage sources - 3 Current sources - 0
VCVS - 0 VCCS - 0
CCVS - 0 CCCS - 0
V-control switch - 0 I-control switch - 0
Macro devices - 0 Functional model instances - 0
Subcircuits - 0 Subcircuit instances - 0
Independent nodes - 3 Boundary nodes - 4
Total nodes - 7

*** 1 WARNING MESSAGES GENERATED

Warning T-SPICE : The vrange voltage range limit (5.5) for diode tables has been exceeded.

Warning T-SPICE : The vrange voltage range limit (5.5) for MOSFET tables has been exceeded.

Warning T-SPICE : The vrange voltage range limit should be set to at least 7.11984 for best accuracy and performance.

Parsing 0.00 seconds
Setup 0.01 seconds
DC operating point 0.00 seconds
Transient Analysis 0.06 seconds

Total 0.07 seconds

RESULT

Thus the design & simulation of a CMOS NAND and NOR gates have been carried out using S-Edit of Tanner EDA Tools

Exp. No.: 15	DESIGN OF CMOS DIFFERENTIAL AMPLIFIER USING TANNER

AIM

To design a CMOS Differential Amplifier using the Schematic entry tool, Tanner and verify its functioning.

APPARATUS REQUIRED:

1. Tanner tool
2. PC

THEORY:

A **differential amplifier** is a type of electronic amplifier that multiplies the difference between two inputs by some constant factor (the differential gain). Many electronic devices use differential amplifiers internally. The output of an ideal differential amplifier is given by:

$$V_{\text{out}} = A_{\text{d}}(V_{\text{in}}^{+} - V_{\text{in}}^{-})$$

Where Vin+ and Vin- are the input voltages and Ac is the differential gain. In practice, however, the gain is not quite equal for the two inputs. This means that if Vin+ and Vin-are equal, the output will not be zero, as it would be in the ideal case. A more realistic expression for the output of a differential amplifier thus includes a second term.

$$V_{\text{out}} = A_{\text{d}}(V_{\text{in}}^{+} - V_{\text{in}}^{-}) + A_{\text{c}}\left(\frac{V_{\text{in}}^{+} + V_{\text{in}}^{-}}{2}\right)$$

A_{c} is called the common-mode gain of the amplifier. As differential amplifiers are often used when it is desired to null out noise or bias-voltages that appear at both inputs, a low common-mode gain is usually considered good.

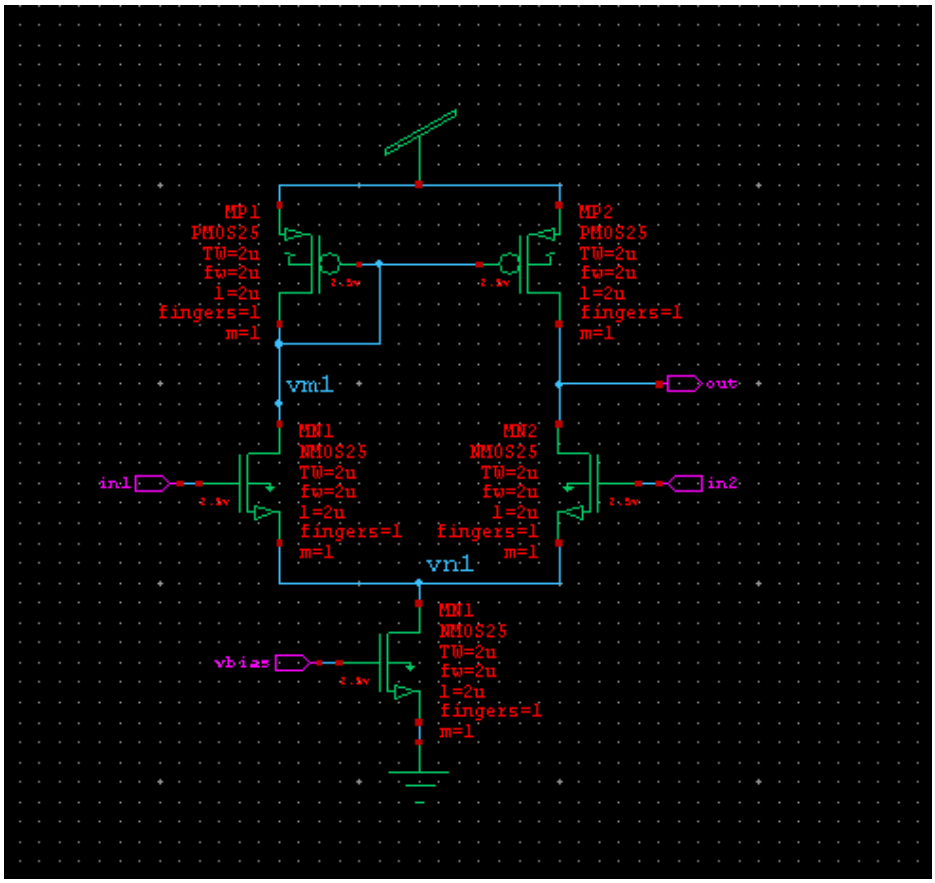
The common-mode rejection ratio, usually defined as the ratio between differential-mode gain and common-mode gain, indicates the ability of the amplifier to accurately cancel voltages that are common to both inputs. Common-mode rejection ratio (CMRR):

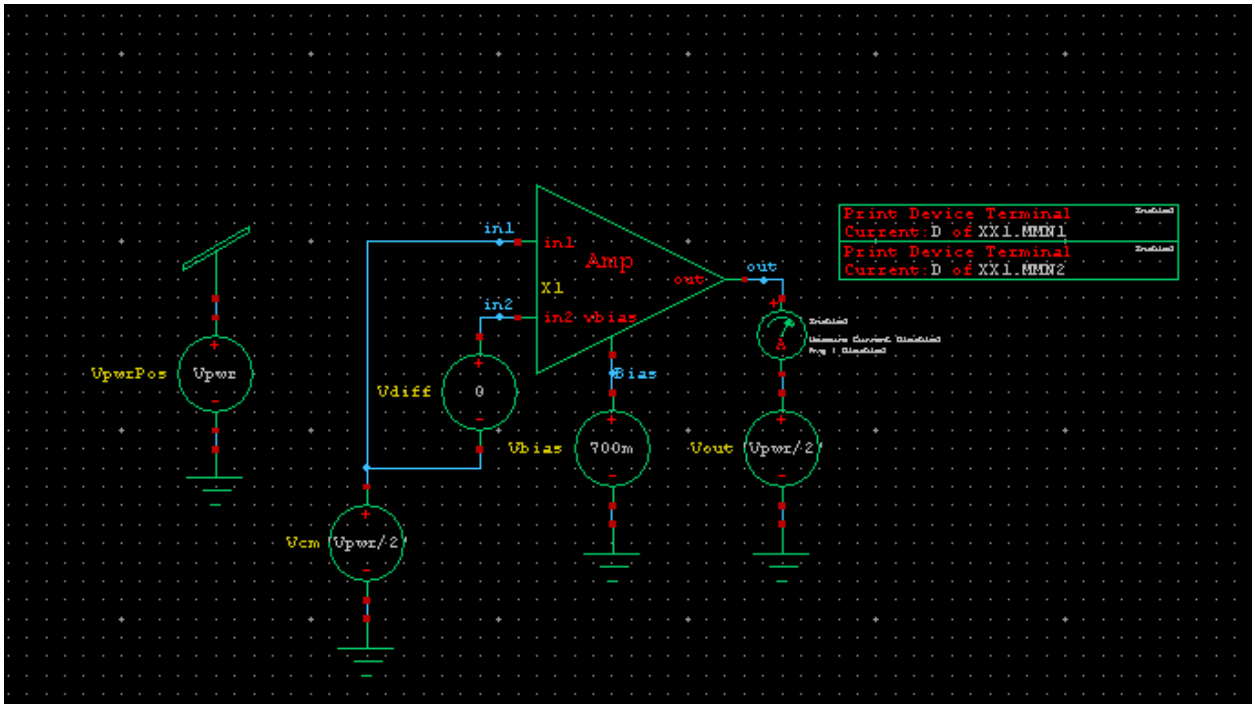
$$\text{CMRR} \triangleq \frac{A_{\text{d}}}{A_{\text{c}}}$$

ALGORITHM

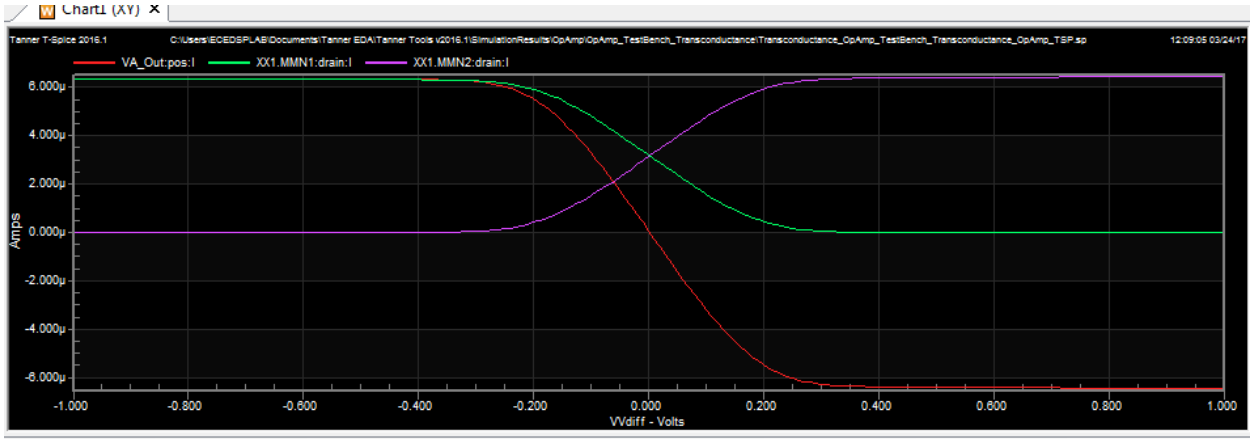
- Draw the schematic of CMOS differential amplifier using S-edit.
- Perform Transient Analysis of the CMOS Inverter.
- Obtain the output waveform from W-edit.
- Obtain the spice code using T-edit.

DIFFERENTIAL AMPLIFIER CIRCUIT:





WAVEFORM:



OUTPUT

TSPICE - Tanner SPICE
Version 7.10
Copyright (c) 1993-2001 Tanner Research, Inc.

Parsing "C:\Tanner\S-Edit\Module0.sp"

Including "C:\Tanner\TSpice70\models\ml2_125.md"

Device and node counts:

MOSFETs - 5 BJTs - 0 MESFETs - 0 Capacitors - 0 Inductors - 0

MOSFET geometries - 2
JFETs - 0
Diodes - 0

Resistors - 0
Mutual inductors - 0

Transmission lines - 0 Coupled transmission lines - 0

Voltage sources - 4
VCVS - 0
CCVS - 0
V-control switch - 0
Macro devices - 0
Subcircuits - 0

Independent nodes - 4
Total nodes - 9

Current sources - 0
VCCS - 0
CCCS - 0
I-control switch - 0
Functional model instances - 0
Subcircuit instances - 0
Boundary nodes - 5
Parsing 0.00 seconds
Setup 0.01 seconds
DC operating point 0.01 seconds
Transient Analysis 0.07 seconds

Total 0.09 seconds

RESULT

The design and simulation of Differential Amplifier has been performed using Tanner EDA Tools.

CONTENT BEYOND SYLLABUS

Exp. No.: 1	**DESIGN & FPGA IMPLEMENTATION OF FLIPFLOPS (D & T FLIPFLOPS)**

AIM:

To implement Flipflops using Verilog HDL.

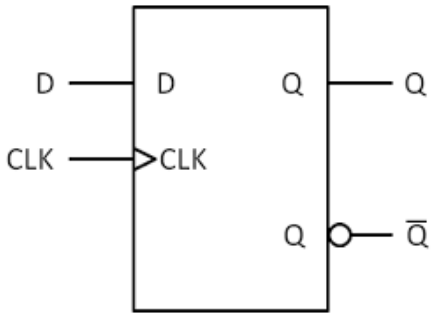
APPARATUS REQUIRED:

- PC with Windows XP.
- XILINX 9.2i
- FPGA-SPARTAN-3 KIT
- PARALLEL TO JTAG CABLE

ALGORITHM:

- New project and type the project name and check the top level source type as HDL
- Enter the device properties and click Next
- Click New Source And Select the Verilog Module and then give the file name
- Give the Input and Output port names and click finish.
- Type the Verilog program and save it
- Double click the synthesize XST and check syntax
- Simulate the waveform by behavioral simulation
- For implementation Select User constraints and give input and output port pin number
- Click Implement design for Translate, map and place & route
- Generate .bit file using programming file
- Implement in FPGA through parallel-JTAG cable
- Check the behavior of design in FPGA by giving inputs

D-FLIPFLOP:



PROGRAM:

```
Module DFF(Clock, Reset, d, q);
input Clock;
input Reset;
input d;
output q;
reg q;
always@(posedge Clock or negedge Reset)
if (~Reset)
q=1'b0;
else
q=d;
endmodule
```

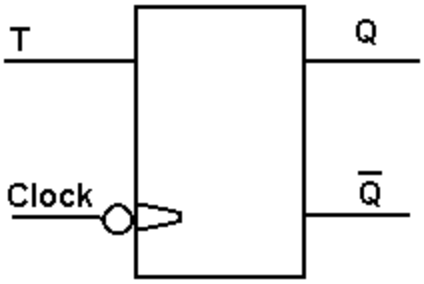
Truth Table:

D FlipFlop

Clock	Reset	Input (d)	Output q(~q)
0	0	0	0(1)
1	0	0	0(1)
0	0	1	0(1)
1	0	1	0(1)
0	0	0	0(1)
1	0	0	0(1)
0	1	1	0(1)
1	1	1	1(0)
0	1	0	1(0)
1	1	0	0(1)
0	1	1	0(1)
1	1	1	1(0)
0	0	0	0(1)
1	0	0	0(1)
0	0	0	0(1)

OUTPUT:

T-FLIPFLOP:



PROGRAM:

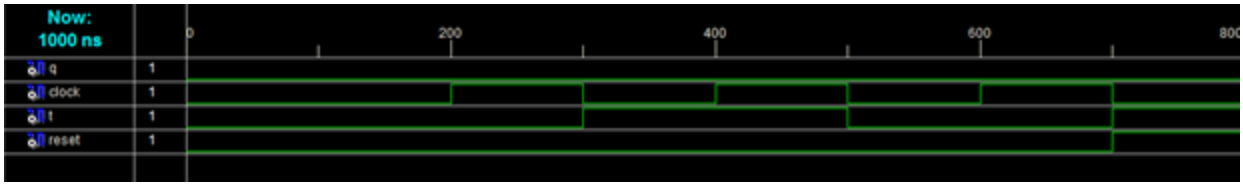
```
Module TFF(Clock, Reset, t, q);
input Clock;
input Reset;
input t;
output q;
reg q;
always@(posedge Clock , negedge Reset)
if(~Reset) q=0;
else if (t) q=~q;
else q=q;
endmodule
```

TRUTH TABLE:

Clock	Reset	Input (t)	Output q(~q)
0	0	0	0(1)
1	0	0	0(1)
0	0	1	0(1)
1	0	1	0(1)
0	0	0	0(1)
1	0	0	0(1)
0	1	1	0(1)
1	1	1	1(0)
0	1	0	1(0)
1	1	0	1(0)
0	1	1	1(0)
1	1	1	0(1)

0	0	0	0(1)
1	0	0	0(1)
0	0	0	0(1)

OUTPUT WAVEFORM:



RESULT:

Thus the D & T flipflops are designed, simulated and implemented successfully.

Exp. No.: 2	**DESIGN & ANALYSIS OF Half Adders using Tanner**

HALF ADDER

Aim:

a) To construct the Half Adder in Tanner EDA v13.1 and to do the Transient Analysis.
b) To analyze the response with appropriate wave forms. And to verify the Spice.

Tools used:

Tanner Tools v13.1
Schematic-Edit
Layout -Edit
Wave- Edit
Tanner Spice

Procedure:

Open S-Edit window.
Go to File → New → New design
Go to Cell→ New View
Add libraries file to the New Cell.
Instance the devices by using appropriate library files.
Save the design and setup the simulation.
Run design and observe waveforms.
Observe DC inputs and outputs by giving appropriate inputs.

Schematic Diagram:

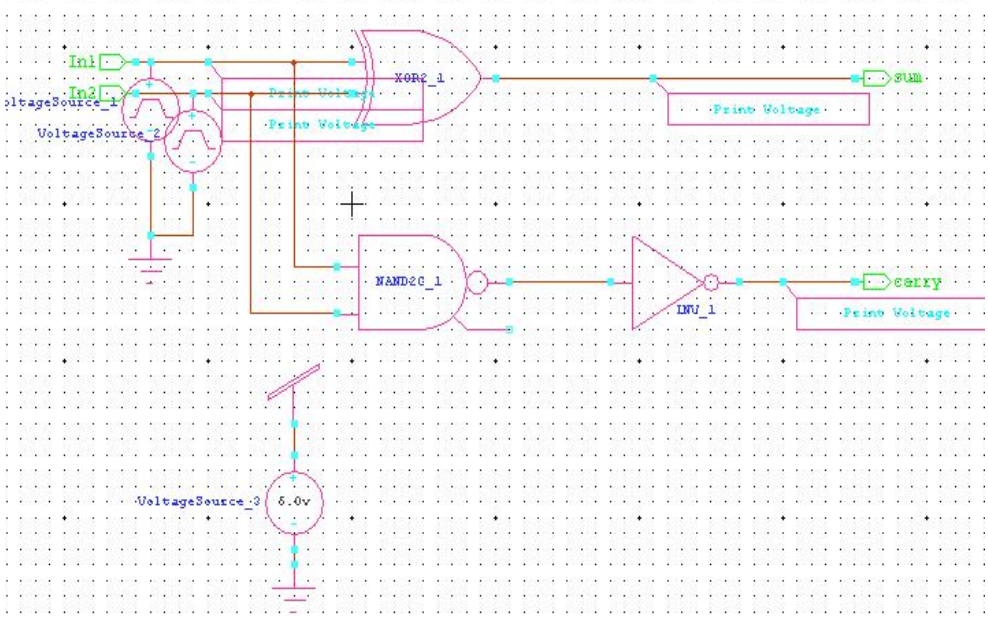


Fig (a): Half Adder Schematic

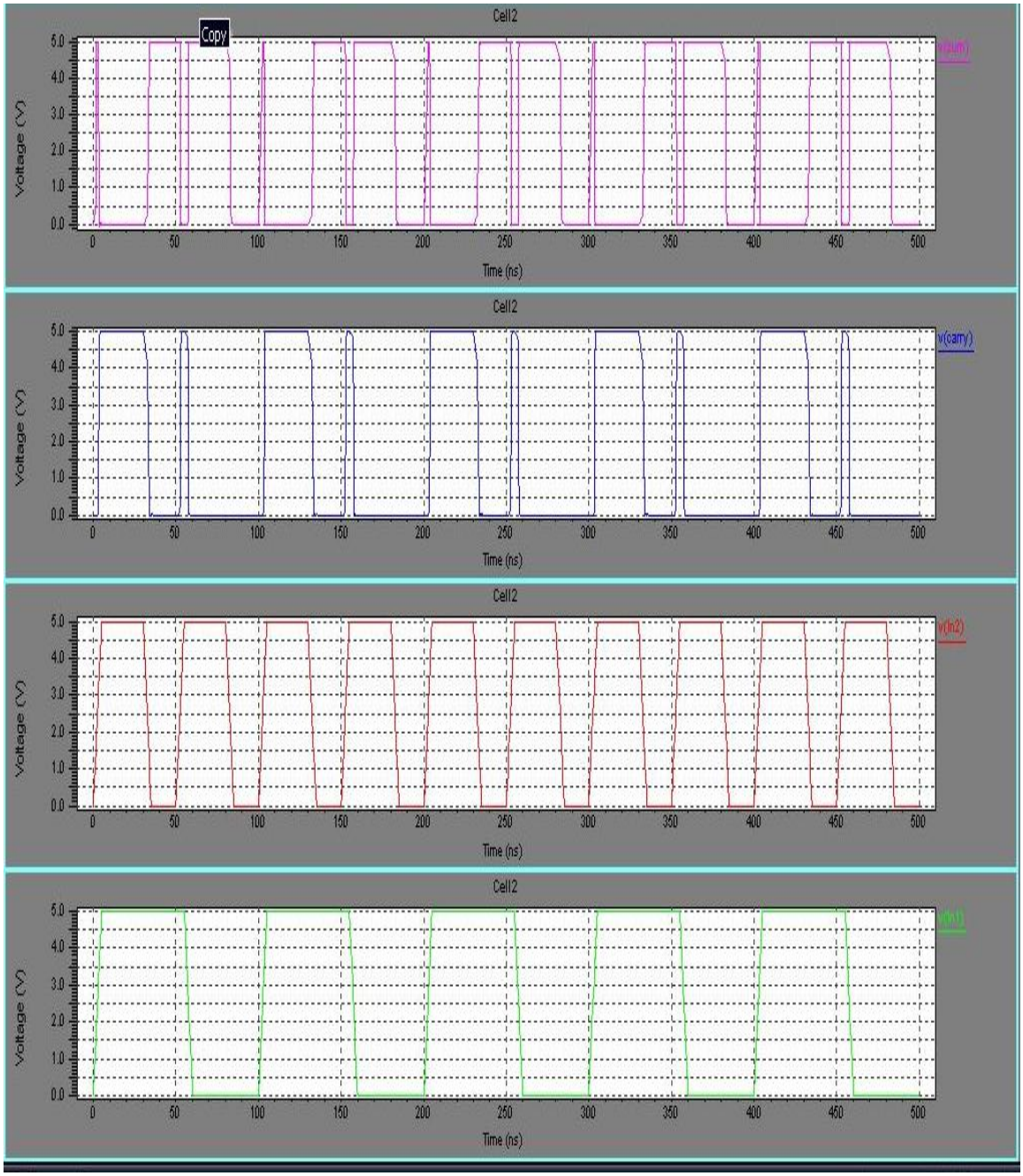
Tanner Spice Code:

```
* SPICE export by:  SEDIT 13.12
* Export time:      Fri Apr 16 11:24:00  2010
* Design:           adm705-1
* Cell:             Cell2
* View:             view0
* Export as:        top-level cell
* Export mode:      hierarchical
* Exclude .model:   no
* Exclude .end:     no
* Expand paths:     yes
* Wrap lines:       no
* Root path:        C:\Documents  and Settings\Administrator\Desktop\adm705-1
* Exclude global pins:   no
* Control property name: SPICE


********* Simulation  Settings  - General section *********
.lib "C:\Documents  and Settings\Administrator\My  Documents\Tanner  EDA\Tanner  Tools
v13.1\Libraries\Models\Generic_025.lib"  TT

*************** Subcircuits  *****************
.subckt INV A Out Gnd Vdd
*-------- Devices: SPICE.ORDER < 0 --------
* Design:  LogicGates  / Cell:  INV / View:  Main / Page:
* Designed by:  Tanner EDA Library  Development  Team
* Organization:  Tanner EDA - Tanner Research,  Inc.
* Info:  Inverter
* Date:  6/14/2007 1:47:11  AM
* Revision:  3

*-------- Devices: SPICE.ORDER > 0 --------
MN1 Out A Gnd 0 NMOS W=2.5u L=250n AS=2.25p PS=6.8u AD=2.25p PD=6.8u
MP1 Out A Vdd Vdd PMOS W=2.5u L=250n M=2 AS=1.5625p PS=3.75u AD=2.25p PD=6.8u
.ends

.subckt NAND2C A B Out1 Out2 Gnd Vdd
*-------- Devices: SPICE.ORDER < 0 --------
* Design:  LogicGates  / Cell:  NAND2C / View:  Main / Page:
* Designed by:  Tanner EDA Library  Development  Team
* Organization:  Tanner EDA - Tanner Research,  Inc.
* Info:  2 Input  NAND with  complementary  output.
* Date:  6/14/2007 1:47:11  AM
* Revision:  2

*-------- Devices: SPICE.ORDER > 0 --------
MN1 Out1 A 1 0 NMOS W=2.5u L=250n AS=2.25p PS=6.8u AD=2.25p PD=6.8u
MN2 1 B Gnd 0 NMOS W=2.5u L=250n AS=2.25p PS=6.8u AD=2.25p PD=6.8u
MN3 Out2 Out1 Gnd 0 NMOS W=2.5u L=250n AS=2.25p PS=6.8u AD=2.25p PD=6.8u
MP1 Out1 A Vdd Vdd PMOS W=2.5u L=250n M=2 AS=1.5625p PS=3.75u AD=2.25p PD=6.8u
```

```
MP2 Out1 B Vdd Vdd PMOS W=2.5u L=250n M=2 AS=1.5625p PS=3.75u AD=2.25p PD=6.8u
MP3 Out2 Out1 Vdd Vdd PMOS W=2.5u L=250n M=2 AS=1.5625p PS=3.75u AD=2.25p
PD=6.8u
.ends

.subckt XNOR2 A B Out Gnd Vdd
*-------- Devices: SPICE.ORDER < 0 --------
* Design: LogicGates / Cell: XNOR2 / View: Main / Page:
* Designed by: Tanner EDA Library Development Team
* Organization: Tanner EDA - Tanner Research, Inc.
* Info: 2 Input NOR
* Date: 7/18/2008 3:58:48 AM
* Revision: 4

*-------- Devices: SPICE.ORDER > 0 --------
MM3n 1 A 2 0 NMOS W=2.5u L=250n AS=2.25p PS=6.8u AD=2.25p PD=6.8u
MM10n Out 1 5 0 NMOS W=2.5u L=250n AS=2.25p PS=6.8u AD=2.25p PD=6.8u
MM9n 5 A Gnd 0 NMOS W=2.5u L=250n AS=2.25p PS=6.8u AD=2.25p PD=6.8u
MM8n Out 1 4 0 NMOS W=2.5u L=250n AS=2.25p PS=6.8u AD=2.25p PD=6.8u
MM7n 4 B Gnd 0 NMOS W=2.5u L=250n AS=2.25p PS=6.8u AD=2.25p PD=6.8u
MM4n 2 B Gnd 0 NMOS W=2.5u L=250n AS=2.25p PS=6.8u AD=2.25p PD=6.8u
MM5p 3 B Vdd Vdd PMOS W=2.5u L=250n M=2 AS=1.5625p PS=3.75u AD=2.25p PD=6.8u
MM11p Out 1 Vdd Vdd PMOS W=2.5u L=250n M=2 AS=1.5625p PS=3.75u AD=2.25p
PD=6.8u
MM2p 1 B Vdd Vdd PMOS W=2.5u L=250n M=2 AS=1.5625p PS=3.75u AD=2.25p PD=6.8u
MM1p 1 A Vdd Vdd PMOS W=2.5u L=250n M=2 AS=1.5625p PS=3.75u AD=2.25p PD=6.8u
MM6p Out A 3 Vdd PMOS W=2.5u L=250n M=2 AS=1.5625p PS=3.75u AD=2.25p PD=6.8u
.ends

.subckt XOR2 A B Out Gnd Vdd
*-------- Devices: SPICE.ORDER < 0 --------
* Design: LogicGates / Cell: XOR2 / View: Main / Page:
* Designed by: Tanner EDA Library Development Team
* Organization: Tanner EDA - Tanner Research, Inc.
* Info: 2 Input NOR
* Date: 7/18/2008 1:30:51 AM
* Revision: 3

*-------- Devices: SPICE.ORDER == 0 --------
XXinv N_1 Out Gnd Vdd INV
XXxnor A B N_1 Gnd Vdd XNOR2
.ends


********* Simulation Settings - Parameters and SPICE Options *********

*-------- Devices: SPICE.ORDER == 0 --------
XINV_1 N_1 carry Gnd Vdd INV
XNAND2C_1 In1 In2 N_1 N_2 Gnd Vdd NAND2C
XXOR2_1 In1 In2 sum Gnd Vdd XOR2
```

```
*-------- Devices: SPICE.ORDER > 0 --------
VVoltageSource_3  Vdd Gnd  DC 5
VVoltageSource_1  In1 Gnd  PULSE(0 5 0 5n 5n 50n 100n)
VVoltageSource_2  In2 Gnd  PULSE(0 5 0 5n 5n 25n 50n)
.PRINT TRAN  V(In1)
.PRINT TRAN  V(In2)
.PRINT TRAN  V(carry)
.PRINT TRAN  V(sum)

********* Simulation  Settings  - Analysis  section *********
.tran 350ns 500ns
.dc lin  source VVoltageSource_1  0 5 0.5 sweep lin source VVoltageSource_2  0 5 0.5
.print  dc v(XINV_1,GND)
.print  dc v(XXOR2_1,GND)

********* Simulation  Settings  - Additional  SPICE commands  *********

.end
```

Output responses:



Result:

The Half Adder is constructed in Tanner EDA v13.1, the spice code is generated and wave forms are verified.