EX.NO : 01

DATE :

AIM **DESIGN OF FPGA BASED CMOS BASIC GATES**

To design and implement FPGA based CMOS basic gates using Verilog HDL.

APPARATUS REQUIRED

Synthesis Tools	:	Xilinx ISE. Simulation
Tools	:	Model sim Simulator.
Kit	:	FPGA Board- Spartan 3E
General	:	Personal Computer (PC) with Parallel to JTAG cable.

THEORY

AND GATE

The AND gate is an electronic circuit that gives a high output (1) only if all its inputs are high. A dot (.) is used to show the AND operation i.e. A.B. Bear in mind that this dot is sometimes omitted i.e. AB

OR GATE

The OR gate is an electronic circuit that gives a high output (1) if one or more of its inputs are high. A plus (+) is used to show the OR operation.

NOT GATE

The NOT gate is an electronic circuit that produces an inverted version of the input at its output. It is also known as an inverter. If the input variable is A, the inverted output is known as NOT A. This is also shown as A', or A with a bar over the top, as shown at the outputs.

EXOR GATE

The 'Exclusive-OR' gate is a circuit which will give a high output if either, but not both, of its two inputs are high.

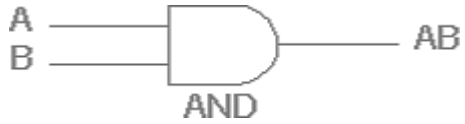
EXNOR GATE

The 'Exclusive-NOR' gate circuit does the opposite to the EOR gate. It will give a low output if either, but not both, of its two inputs are high.

PROCEDURE

- The basic gate circuit is designed and the function is found out.
- The Verilog Module Source for the circuit is written.
- It is implemented in Model Sim and Simulated.
- Signals are provided and Output Waveforms are viewed.

CIRCUIT DIAGRAM AND GATE

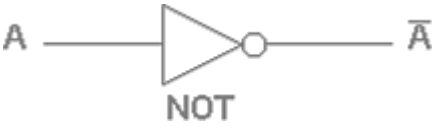


2 Input AND gate		
A	B	A.B
0	0	0
0	1	0
1	0	0
1	1	1

OR GATE

2 Input OR gate		
A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

NOT GATE

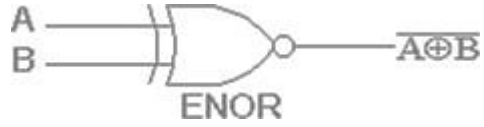


NOT gate	
A	$\overline{A}$
0	1
1	0

EXOR GATE

2 Input EXOR gate		
A	B	A⊕B
0	0	0
0	1	1
1	0	1
1	1	0

EXNOR GATE



2 Input EXNOR gate		
A	B	$\overline{A \oplus B}$
0	0	1
0	1	0
1	0	0
1	1	1

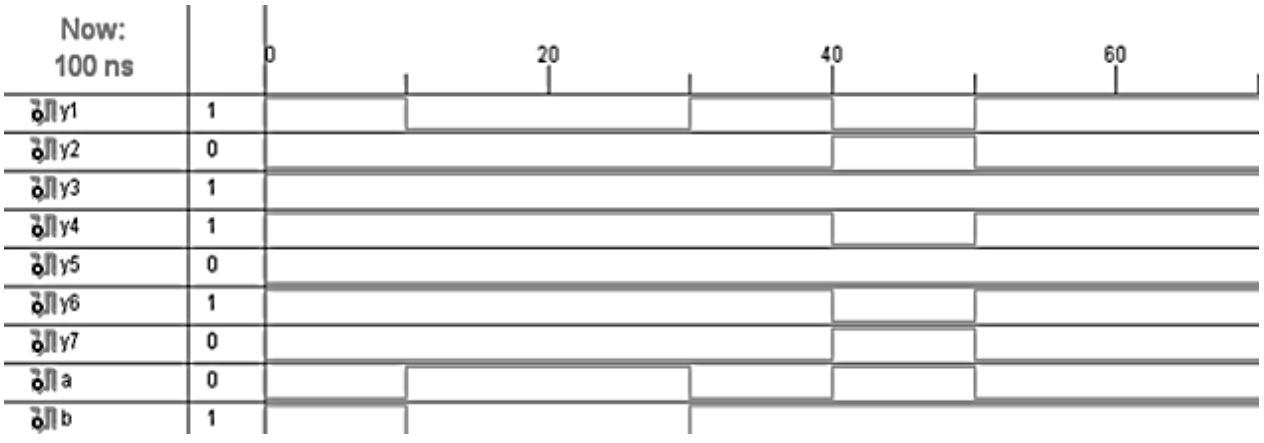
PROGRAM

```
module gates(a,b, y1,y2,y3,y4,y5,y6,y7); input
a,b;
output y1,y2,y3,y4,y5,y6,y7;
assign y1=~a;
assign y2=a&b;
assign y3=a|b; assign
y4=~(a&b); assign
y5=~(a|b); assign y6
=a^b; assign
y7=~(a^b);
endmodule
```

TEST BENCH

```
initial
begin
#100 a=0; b=0;
#100 a=0; b=1;
#100 a=1; b=0;
#100 a=1; b=1;
end endmodule
```

OUTPUT



RESULT

Thus the Verilog programs for CMOS basic gates were written, synthesized, simulated and implemented using Xilinx tools and FPGA.

EX.NO : 02

DATE :

DESIGN OF FPGA BASED INVERTER, BUFFER AND TRANSMISSION GATE

AIM

To design and implement FPGA based inverter, buffer and transmission gate using Verilog HDL.

APPARATUS REQUIRED

Synthesis Tools	:	Xilinx ISE. Simulation
Tools	:	Model sim Simulator.
Kit	:	FPGA Board- Spartan 3E
General	:	Personal Computer (PC) with Parallel to JTAG cable.

THEORY

INVERTER

The NOT gate or an inverter is an electronic circuit that produces an inverted version of the input at its output. It is also known as an inverter. If the input variable is A, the inverted output is known as NOT A. This is also shown as A', or A with a bar over the top, as shown at the outputs. The diagrams below show two ways that the NAND logic gate can be configured to produce a NOT gate. It can also be done using NOR logic gates in the same way.

BUFFER

A special logic gate called a buffer is manufactured to perform the same function as two inverters. Its symbol is simply a triangle, with no inverting “bubble” on the output terminal. Buffer gates merely serve the purpose of signal amplification: taking a “weak” signal source that isn’t capable of sourcing or sinking much current, and boosting the current capacity of the signal so as to be able to drive a load.

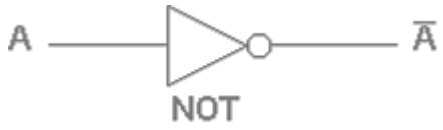
TRANSMISSION GATE

When the voltage on node A is a Logic 1, the complementary Logic 0 is applied to node active-low A, allowing both transistors to conduct and pass the signal at IN to OUT. When the voltage on node active-low A is a Logic 0, the complementary Logic 1 is applied to node A, turning both transistors off and forcing a high-impedance condition on both the IN and OUT nodes. This high-impedance condition represents the third "state" (high, low, or high-Z) that the channel may reflect downstream.

PROCEDURE

- The inverter, buffer and transmission gate circuit is designed and the function is found out.
- The Verilog Module Source for the circuit is written.
- It is implemented in Model Sim and Simulated.
- Signals are provided and Output Waveforms are viewed.

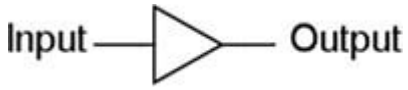
CIRCUIT DIAGRAM



NOT gate	
A	$\overline{A}$
0	1
1	0

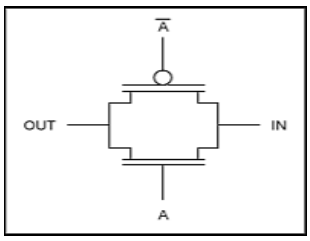
INVERTER

BUFFER



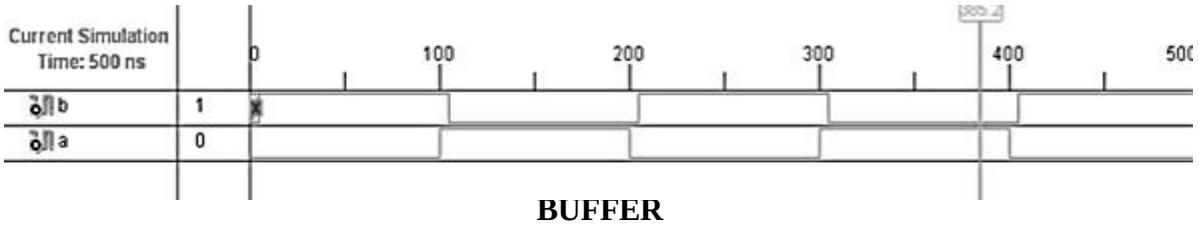
Input	Output
0	0
1	1

TRANSMISSION GATE

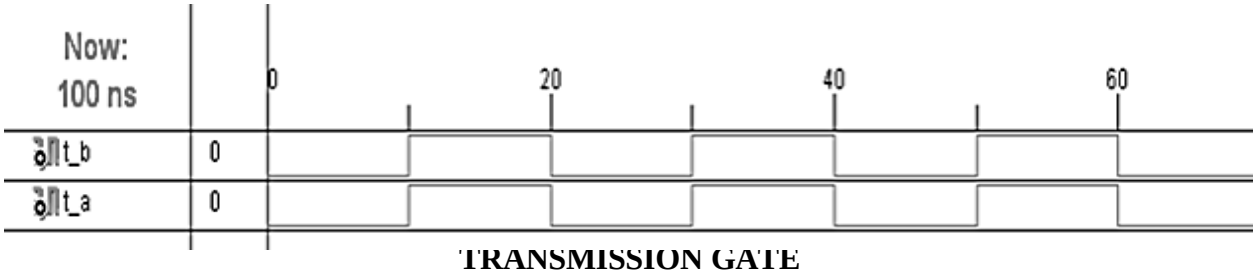


OUTPUT:

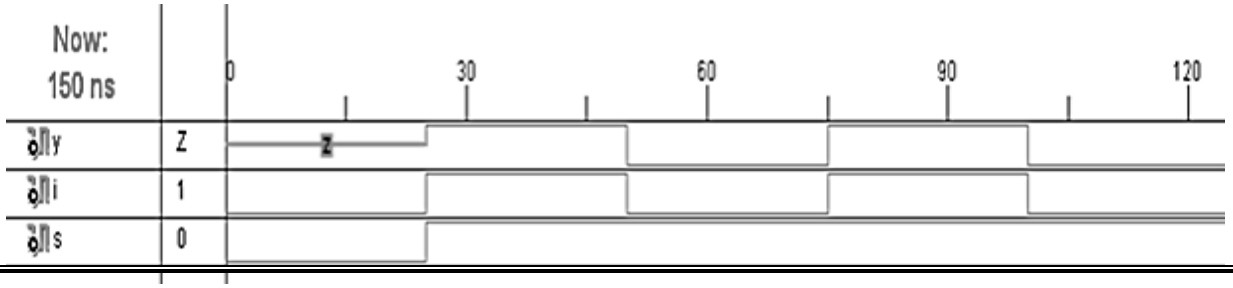
INVERTER



BUFFER



TRANSMISSION GATE



PROGRAM

INVERTER

```
module inv(a, b);
 input a;
output b;
assign b=~a;
endmodule
```

TEST BENCH

```
initial
begin
a=1'b0;
#100 a=1'b1;
#100 a=1'b0;
end
endmodule
```

BUFFER

```
module buff(a, b);
input a;
output b;
assign b=a;
endmodule
```

TEST BENCH

```
initial
begin
a=1'b0;
#100 a=1'b1;
#100 a=0'b0;
#100 a=1'b1;
end
endmodule
```

TRANSMISSION GATE

```
module tr(i,s, y);
input i,s;
output y;
reg y;
always@(i,s)
begin if(s==1)
y=i; else
y=1'bz;
end
endmodule
```

TEST BENCH

```
initial
begin
#100 i=0; s=0;
#100 i=1; s=1;
#100 i=0; s=1;
#100 i=1; s=1;
#100 i=0; s=1;
#100 i=1; s=0;
end endmodule
```

RESULT:

Thus the Verilog programs for inverter, buffer and transmission gate were written, synthesized, simulated and implemented using Xilinx tools and FPGA.

EX.NO : 03

DATE :

DESIGN OF HALF ADDER AND FULL ADDER

AIM

To design and implement FPGA based half adder and full adder using Verilog HDL.

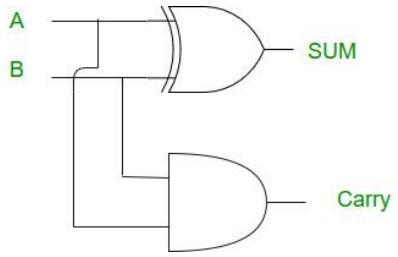
APPARATUS REQUIRED

Synthesis Tools	:	Xilinx ISE. Simulation
Tools	:	Model sim Simulator.
Kit	:	FPGA Board- Spartan 3E
General	:	Personal Computer (PC) with Parallel to JTAG cable.

PROCEDURE

1. Select the new project
2. In new project enter the project name and select location.
3. The top level module select the verilog module and click the next and finish the window.
4. Type the program and modulate the program.
5. Check the syntax and simulate the above verilog code (using ModelSim or Xilinx) and verify the output waveform as obtained.
6. Implement the above code in Spartan III using FPGA kit.

Half Adder



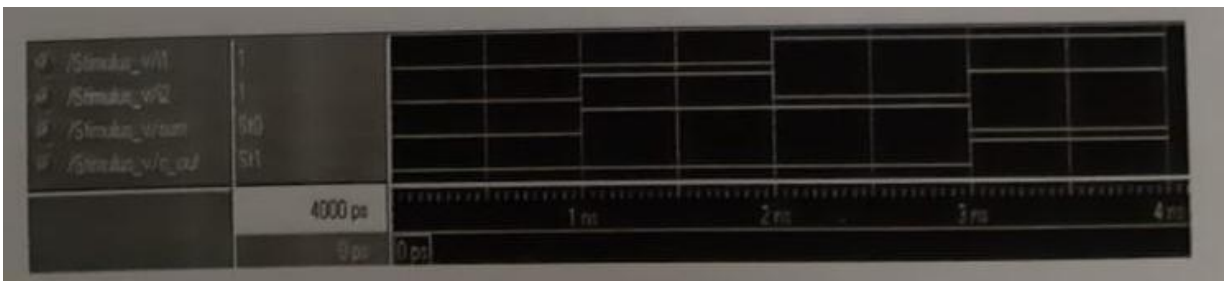
Truth Table

A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

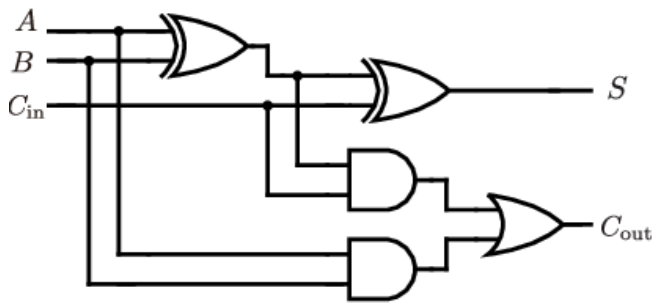
Program: Half Adder

```
// Module Name: HalfAddr
module HalfAddr(sum, c_out, il, 12);

        output sum;

        output c_out;

        input il;

        input 12:

           xor(sum.i1,12);

            and(c_out.i1,12):

        endmodule
```

Output Waveform for Half Adder



Full Adder

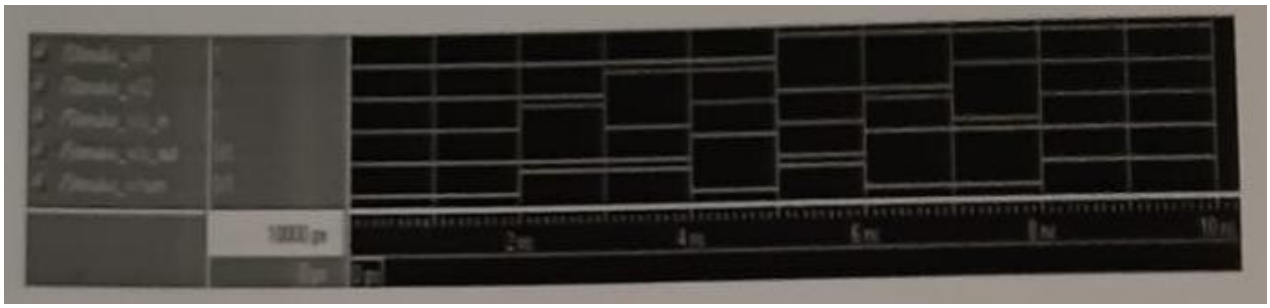


Inputs			Outputs	
A	B	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

PROGRAM: FULL ADDER:

```
// Module Name: FullAddr
module FullAddr(il, i2, c_in, c_out, sum);
input il;
input 12:
input c_in;
output c_out;
output sum;
        wire s1.cl.c2;
        xorn1(s1.1.2);
        and n2(cl.i1.12);
        xor n3(sum.s1.c_in);
        and n4(c2.s1.c_in);
        or n5(c_out,cl.c2);
endmodule
```

Output Waveform for Full Adder



RESULT

Thus the Verilog program for Half Adder and Full Adder has been simulated and output was verified.

EX.NO : 04

DATE :

DESIGN OF FPGA BASED FLIP-FLOP

AIM

To design and implement FPGA based flip-flop using Verilog HDL.

APPARATUS REQUIRED

Synthesis Tools	:	Xilinx ISE. Simulation
Tools	:	Model sim Simulator.
Kit	:	FPGA Board- Spartan 3E
General	:	Personal Computer (PC) with Parallel to JTAG cable.

THEORY

SR FLIP FLOP

It can be seen that when both inputs S = “1” and R = “1” the outputs Q and Q can be at either logic level “1” or “0”, depending upon the state of the inputs S or R before this input condition existed. Therefore the condition of S = R = “1” does not change the state of the outputs Q and Q. However, the input state of S = “0” and R = “0” is an undesirable or invalid condition and must be avoided. The condition of S = R = “0” causes both outputs Q and Q to be HIGH together at logic level “1” when we would normally want Q to be the inverse of Q.

D FLIP FLOP

It is also known as a "data" or "delay" flip-flop. The D flip-flop captures the value of the D-input at a definite portion of the clock cycle (such as the rising edge of the clock). That captured value becomes the Q output. At other times, the output Q does not change. The D flip-flop can be viewed as a memory cell, a zero-order hold, or a delay line.

T FLIP FLOP

If the T input is high, the T flip-flop changes state ("toggles") whenever the clock input is strobed. If the T input is low, the flip-flop holds the previous value.

JK FLIP FLOP

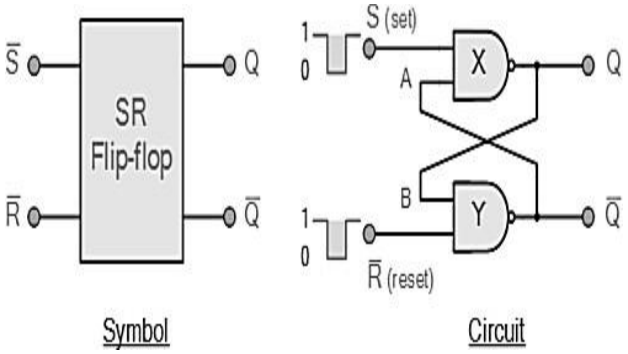
The JK flip-flop is basically an SR flip flop with feedback which enables only one of its two input terminals, either SET or RESET to be active at any one time thereby eliminating the invalid condition seen previously in the SR flip flop circuit. Also when both the J and the K inputs are at logic level “1” at the same time, and the clock input is pulsed either “HIGH”, the circuit will “toggle” from its SET state to a RESET state, or visa-versa.

PROCEDURE

- The flip-flop circuit is designed and the function is found out.
- The Verilog Module Source for the circuit is written.
- It is implemented in Model Sim and Simulated.
- Signals are provided and Output Waveforms are viewed.

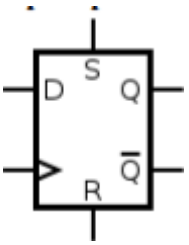
CIRCUIT DIAGRAM

SR FLIP FLOP



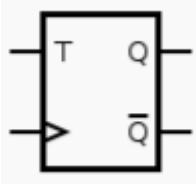
State	S	R	Q	Q	Description
Set	1	0	0	1	Set Q » 1
	1	1	0	1	no change
Reset	0	1	1	0	Reset Q » 0
	1	1	1	0	no change
Invalid	0	0	1	1	Invalid Condition

D FLIP FLOP



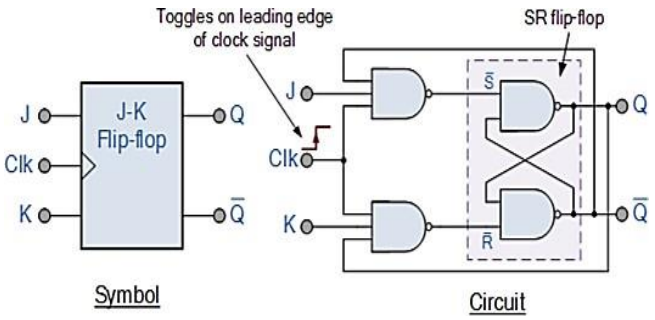
Clock	D	Q_{next}
Rising edge	0	0
Rising edge	1	1
Non-Rising	X	Q

T FLIP FLOP



T flip-flop operation[26]							
Characteristic table				Excitation table			
			Comment				Comment
0	0	0	hold state (no clk)	0	0	0	No change
0	1	1	hold state (no clk)	1	1	0	No change
1	0	1	toggle	0	1	1	Complement
1	1	0	toggle	1	0	1	Complement

JK FLIP FLOP



	Input		Output		Description
	J	K	Q	Q	
same as for the SR Latch	0	0	0	0	Memory no change
	0	0	0	1	
	0	1	1	0	Reset Q » 0
	0	1	0	1	
	1	0	0	1	Set Q » 1
	1	0	1	0	
toggle action	1	1	0	1	Toggle
	1	1	1	0	

PROGRAM:

SR FLIP FLOP

```
module srf(s,r,clk, q,qb);
input s,r,clk;
output q,qb;
reg q,qb;
always @(posedge clk)
begin
if(s==0 & r==1)
q=0;
else if(s==1 &r==0)
q=1;
else if(s==0 & r==0)
q=q;
else if (s==1 &r==1)
q=1'bz;
qb=~q;

end
endmodule
```

TEST BENCH:

```
initial
begin
#100  s=0; r=0; clk=0;
#100  s=0; r=0; clk=1;
#100  s=0; r=1; clk=0;
#100  s=0; r=1; clk=1;
#100  s=1; r=0; clk=0;
#100  s=1; r=0; clk=1;
#100  s=1; r=1; clk=0;
#100  s=1; r=1; clk=1;
end

endmodule
```

D FLIP FLOP

```
module dff(clk,d,q,qbar);
input clk;
input d; output
q,qbar;
reg temp,q,qbar; always
@(posedge clk) begin
if (d==0) temp
= 1'b0; else
temp = 1'b1; end
assign  q  =  temp;
assign  qbar  =  ~q;
endmodule
```

TEST BENCH

```
initial
begin
#100 clk=0;d=0;
#100 clk=1;d=0;
#100 clk=0;d=0;
#100 clk=1;d=0;
#100 clk=0;d=0;
end endmodule
```

T FLIP FLOP

```
module tff(t,clk, q,qb);
input t,clk;
output q,qb;
 reg qb;
reg q; initial
q=0;
always@(posedge clk)
begin
if(t==0)
q=q; else
q=~q;
qb=~q;
end
endmodule
```

TEST BENCH

```
initial begin #100
t=0;  clk=0;
#100 t=0;  clk=1;
#100 t=1;  clk=0;
#100 t=1;  clk=1;
#100 t=0;  clk=0;
#100 t=1;  clk=1;
end endmodule
```

JK FLIP FLOP

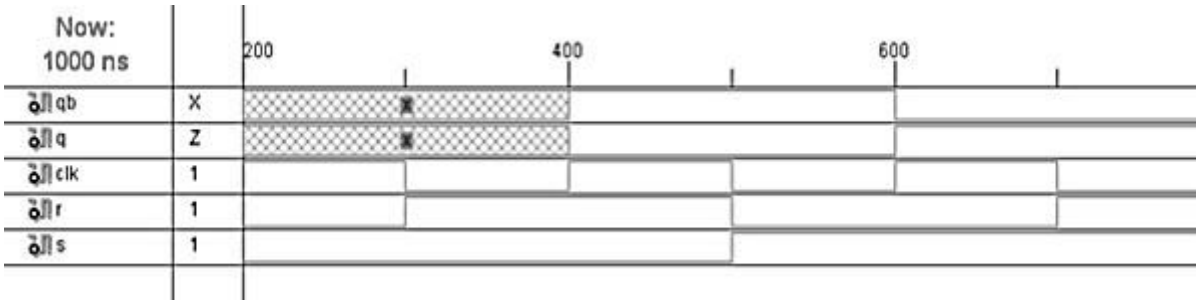
```
module jkflip(j, k, clk, q, z);
input j;
input k;
input clk;
output q;
output z;
reg q,z;
always@(posedge clk)
begin
if(j==0 & k==1) q=0;
else if(j==1 & k==0) q=1;
else if(j==0 & k==0) q=q;
else if(j==1 & k==1)
q=~q;
z=~q;
end
endmodule
```

TEST BENCH

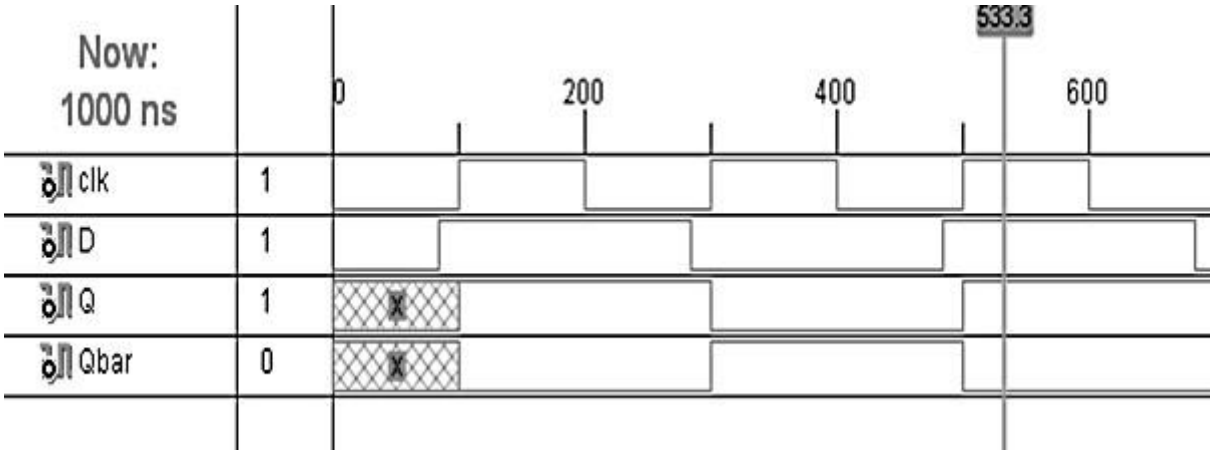
```
initial
begin
j=0; k=0; clk=0;  #100;
j=0; k=0; clk=1; #100;
j=0; k=1; clk=0; #100;
j=0; k=1; clk=1;
#100;  j=1;  k=0;  clk=0;
#100;  j=1;  k=0;  clk=1;
#100;  j=1;  k=1;  clk=0;
#100; j=1; k=1; clk=1;
end endmodule
```

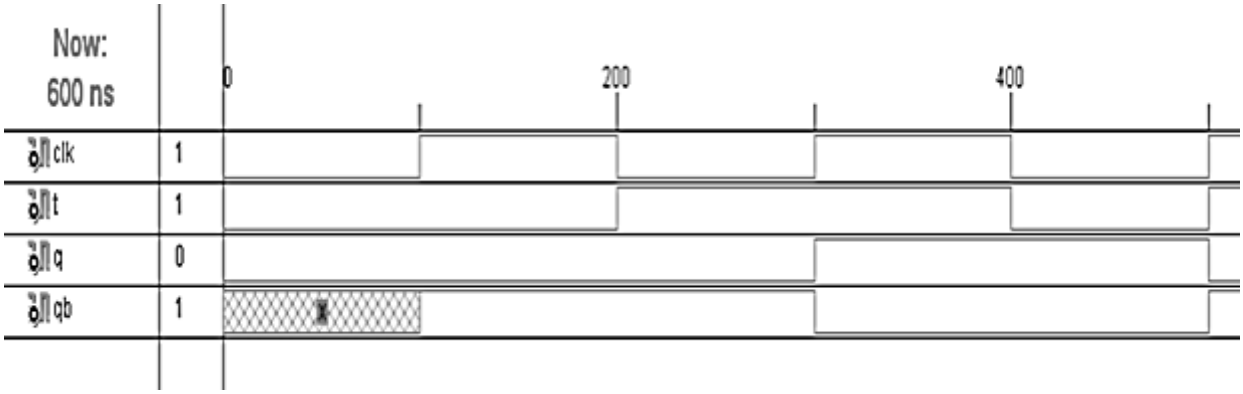
OUTPUT:

SR FLIP FLOP

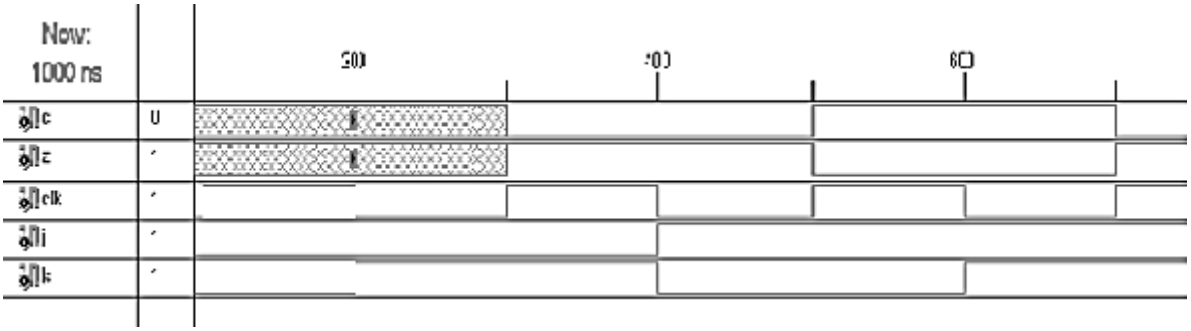


D FLIP FLOP





JK FLIP FLOP



RESULT:

Thus the Verilog programs for flip flop were written, synthesized, simulated and implemented using Xilinx tools and FPGA.

EX. NO : 05
DATE :

DESIGN OF FPGA BASED RIPPLE CARRY ADDER

AIM

To Design and Implement the FPGA based Ripple Carry Adder using Verilog HDL.

APPARATUS REQUIRED

Synthesis Tools	:	Xilinx ISE. Simulation
Tools	:	Model sim Simulator.
Kit	:	FPGA Board- Spartan 3E
General	:	Personal Computer (PC).

THEORY

The *n*-bit adder built from n one –bit full adders is known as ripple carry adder because of the carry is computed. The addition is not complete until *n-1th* adder has computed its Sn-1 output; that results depends upon ci input, n and so on down the line, so the critical delay path goes from the 0-bit inputs up through ci's to the *n-1* bit.(We can find the critical path through the n-bit adder without knowing the exact logic in the full adder because the delay through the n-bit adder without knowing the exact logic in the full adder because the delay through the n-bit carry chain is so much longer than the delay from a and b to s). The ripple-carry adder is area efficient and easy to design but it is when n is large.It can also be called as cascaded full adder.

The simplified Boolean functions of the two outputs can be obtained as below:

$$\textbf{Sum } s_i = a_i \text{ xor } b_i \text{ xor } c_i \quad \textbf{Carry}$$

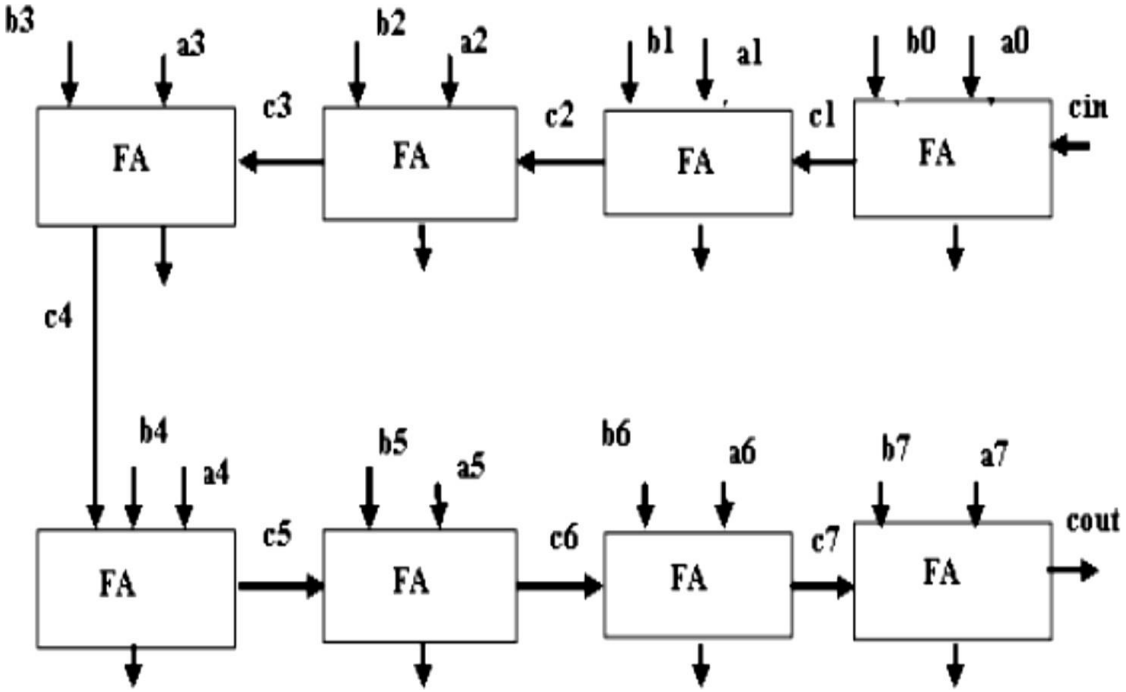
$$c_{i+1} = a_i b_i + b_i c_i + a_i c_i$$

Where x, y & z are the two input variables.

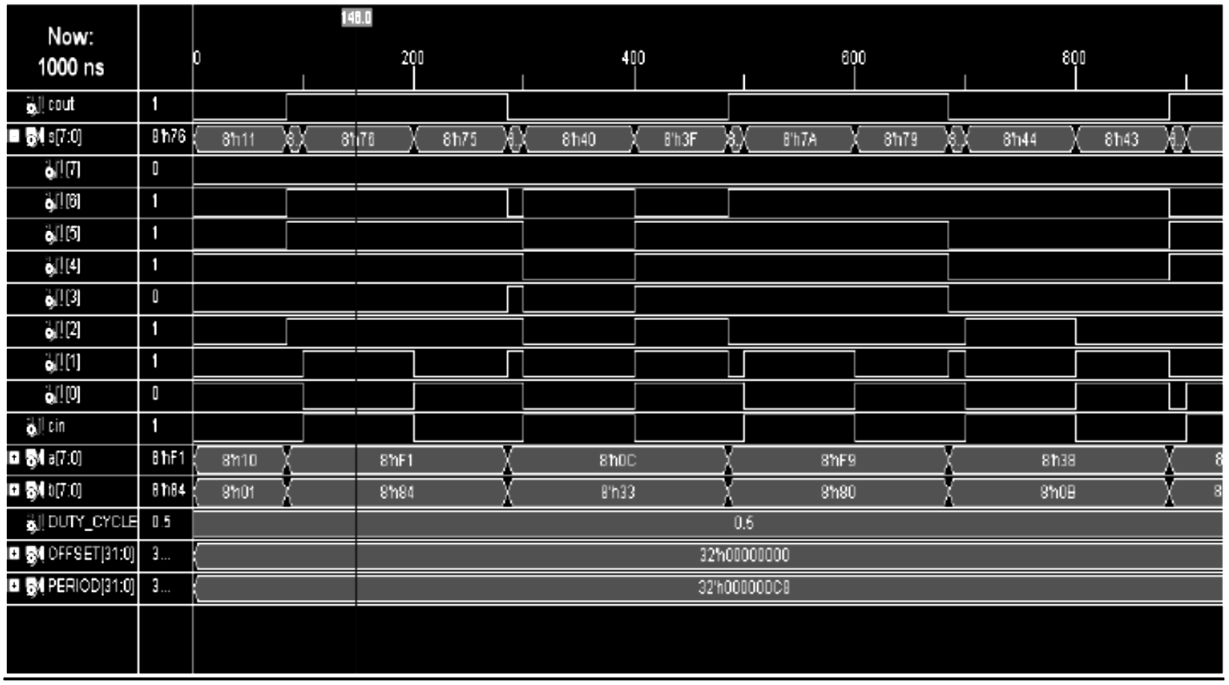
PROCEDURE

- The full-adder circuit is designed and the Boolean function is found out.
- The Verilog Module Source for the circuit is written.
- It is implemented in Model Sim and Simulated.
- Signals are provided and Output Waveforms are viewed.

CIRCUIT DIAGRAM



OUTPUT



PROGRAM

```
Module ripplecarryadder(s,cout,a,b,cin);
output[7:0]s;
outputcout;
input[7:0]a,b;
inputcin;
wire c1,c2,c3,c4,c5,c6,c7;
fulladd fa0(s[0],c1,a[0],b[0],cin);
fulladd fa1(s[1],c2,a[1],b[1],c1);
fulladd fa2(s[2],c3,a[2],b[2],c2);
fulladd fa3(s[3],c4,a[3],b[3],c3);
fulladd fa4(s[4],c5,a[4],b[4],c4);
fulladd fa5(s[5],c6,a[5],b[5],c5);
fulladd fa6(s[6],c7,a[6],b[6],c6);
fulladd fa7(s[7],cout,a[7],b[7],c7);
endmodule
modulefulladd(s,cout,a,b,cin);
outputs,cout;
inputa,b,cin;
wire s1,c1,c2;
xor(s1,a,b);
xor(s,s1,cin);
and(c1,a,b);
and(c2,s1,cin);
xor(cout,c2,c1);
endmodule
```

TEST BENCH

```
initial
begin
#100 a=8'b00000000; b=8'b00000000; cin=1'b0; #100
a=8'b00001000;  b=8'b00000001;  cin=1'b0;  #100
a=8'b01000000;  b=8'b00001000;  cin=1'b0;  #100
a=8'b00001110;  b=8'b01111000;  cin=1'b0;  #100
a=8'b10101010;  b=8'b01010100;  cin=1'b1;  #100
a=8'b00010010;  b=8'b11000000;  cin=1'b1;  #100
a=8'b11110000;  b=8'b00111100;  cin=1'b1;  #100
a=8'b00001111;  b=8'b11000011;  cin=1'b1;  #100
a=8'b11111111; b=8'b11111111; cin=1'b1;
end endmodule
```

RESULT

Thus the Verilog programs for ripple carry adder were written, synthesized, simulated and implemented using Xilinx tools and FPGA.

EX.NO : 06

DATE :

DESIGN OF FPGA BASED CARRY SAVE ADDER

AIM

To Design and Implement the FPGA based Carry Save Adder using Verilog HDL.

APPARATUS REQUIRED

Synthesis Tools	:	Xilinx ISE. Simulation
Tools	:	Model sim Simulator.
Kit	:	FPGA Board- Spartan 3E
General	:	Personal Computer (PC).

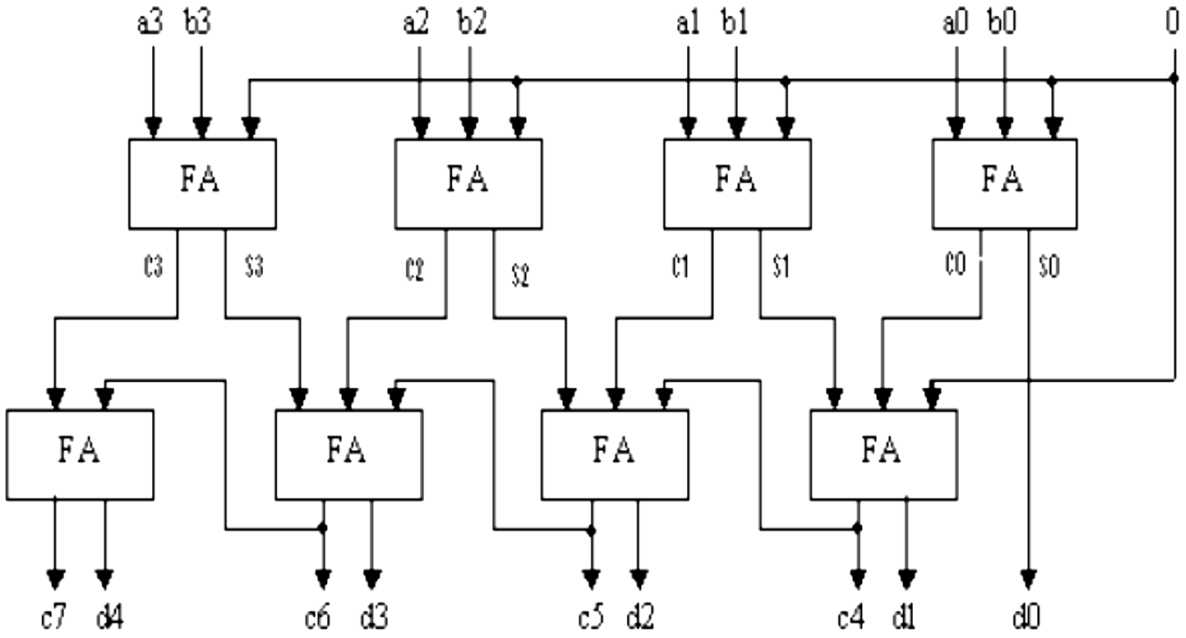
THEORY

Carry save adders are suitable when three or more operands are to be added, as in some multiplication schemes. In this adder a separate sum and carry bit is generated for partial results , except when the last operand is added. For example, when three numbers are added, the first two are added using a carry save adder. The partial result is two numbers corresponding to the sum and the carry .The last operand is added using a second carry save adder stage. The results become the sum and carry numbers. Thus a carry save adder reduces the number of operands by one for each adder stage. Finally the sum and carry are added using an adder with carry propagation- for example carry look ahead adder.

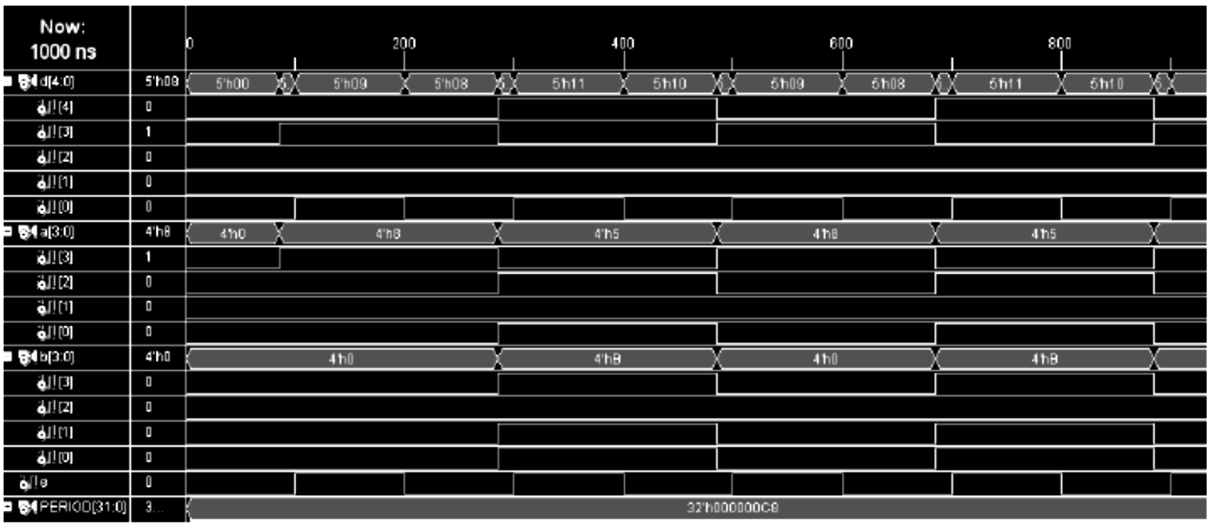
PROCEDURE

- The carry save adder is designed.
- The Verilog program source code for the circuit is written.
- It is implemented in Model Sim and Simulated.
- Signals are provided and Output Waveforms are viewed.

CIRCUIT DIAGRAM



OUTPUT



PROGRAM

```
modulecarrysaveadder(d,a,b,e); output
[4:0]d;
input e;
input [3:0]a,b;
wire s1,s2,s3,c0,c1,c2,c3,c4,c5,c6,c7;
fulladder a1(d[0],c7,a[0],b[0],e);
fulladder a2(s3,c6,a[1],b[1],e);
fulladder a3(s2,c5,a[2],b[2],e);
fulladder a4(s1,c4,a[3],b[3],e);
fulladder a5(d[1],c3,c7,s3,e);
fulladder a6(d[2],c2,c6,c3,s2);
fulladder a7(d[3],c1,c5,s1,c2);
fulladder a8(d[4],c0,c4,c1,e);
endmodule modulefulladder(s,c,
x,y,z); outputs,c;
inputx,y,z;
xor (s,x,y,z);
assign c = ((x & y )|(y & z)|( z & x)) ;
endmodule
```

TEST BENCH

```
initial
begin
#100  a=4'b0000;  b=4'b0000;  e=1'b0;
#100  a=4'b0001;  b=4'b0010;  e=1'b0;
#100  a=4'b1110;  b=4'b0011;  e=1'b0;
#100  a=4'b0110;  b=4'b0101;  e=1'b0;
#100  a=4'b1100;  b=4'b0011;  e=1'b1;
#100  a=4'b0110;  b=4'b1010;  e=1'b1;
#100 a=4'b1111; b=4'b1111; e=1'b1;
end endmodule
```

RESULT:

Thus the Verilog programs for carry save adder were written, synthesized, simulated and implemented using Xilinx tools and FPGA.

EX.NO : 07

DATE :

DESIGN OF FPGA BASED CARRY SELECT ADDER

AIM

To Design and Implement the FPGA based Carry Select Adder using Verilog HDL

APPARATUS REQUIRED

Synthesis Tools	:	Xilinx ISE. Simulation
Tools	:	Model sim Simulator.
Kit	:	FPGA Board- Spartan 3E
General	:	Personal Computer (PC).

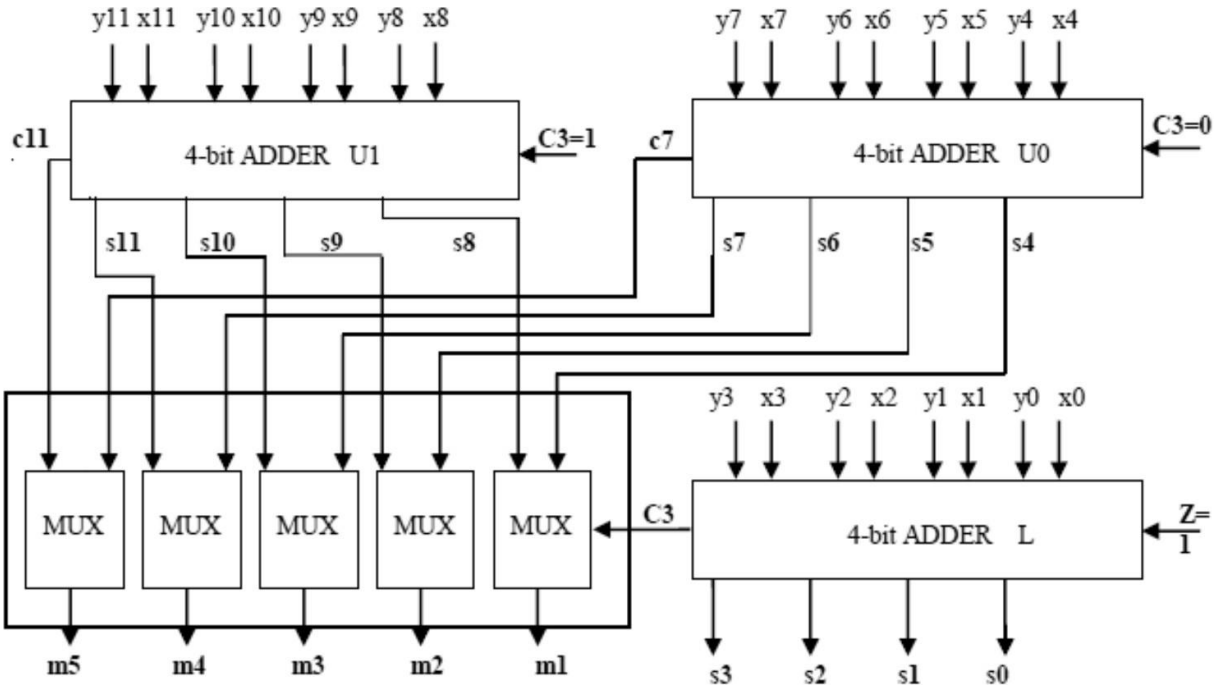
THEORY

Carry-select adders use multiple narrow adders to create fast wide adders. A carry- select adder provides two separate adders for the upper words, one for each possibility. A MUX is then used to select the valid result. Consider an 8-bit adder that is split into two 4- bit groups. The lower-order bits and are fed into the 4_bit adder l to produce the sum bits and a carry-out bit .the higher order bits and are used as input to one 4_bit adder and and are used as input of the another 4_bit adder. Adder U0 calculates the sum with a carry-in of C3=0.while U1 does the same only it has a carry-in value of C3=1.both sets of results are used as inputs to an array of 2:1 MUXes .the carry bit from the adder L is used as the MUX select signal. If =0 then the results U0 are sent to the output, while a value of =1 selects the results of U1 for . The carry-out bit is also selected by the MUX array.

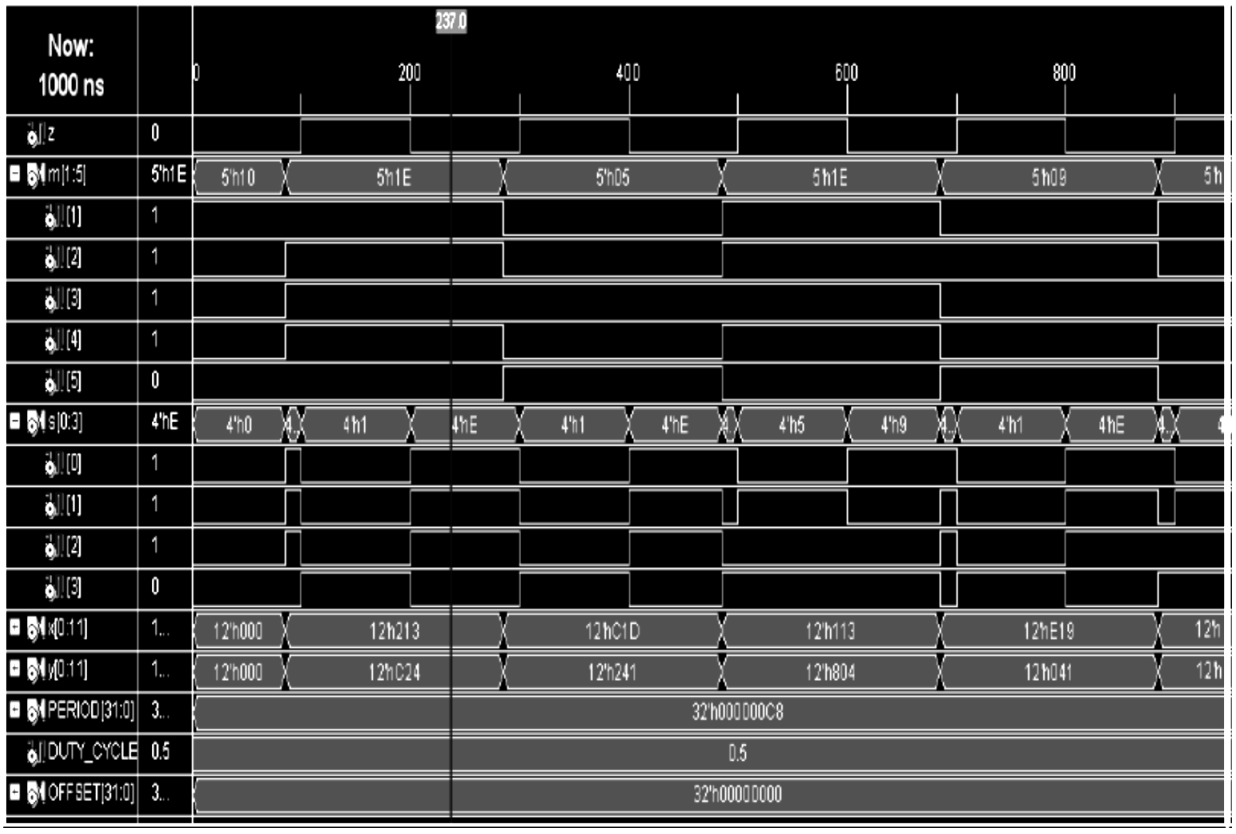
PROCEDURE

- The carry-select adder circuit is designed and the Boolean function is found out.
- The Verilog Module Source for the circuit is written.
- It is implemented in Model Sim and Simulated.
- Signals are provided and Output Waveforms are viewed.

CIRCUIT DIAGRAM



OUTPUT



PROGRAM

```
module project2(s, m, x, y, z); output
[0:3]s;
output [1:5]m;
input [0:11]x;
input [0:11]y;
input z;
wire c0,c1,c2,c3,c4,c5,c6,c7,c8,c9,c10,c11
,s4,s5,s6,s7,s8,s9,s10,s11;
fulladder f1(s[0],c0,x[0],y[0],z);
fulladder f2(s[1],c1,x[1],y[1],c0);
fulladder f3(s[2],c2,x[2],y[2],c1);
fulladder f4(s[3],c3,x[3],y[3],c2);
fulladder f5(s4,c4,x[4],y[4],c3);
fulladder f6(s5,c5,x[5],y[5],c4);
fulladder f7(s6,c6,x[6],y[6],c5);
fulladder f8(s7,c7,x[7],y[7],c6);
fulladder f9(s8,c8,x[8],y[8],~c3);
fulladder f10(s9,c9,x[9],y[9],c8);
fulladder f11(s10,c10,x[10],y[10],c9);
fulladder f12(s11,c11,x[11],y[11],c10);
muxer mu1(m[1],s4,s8,c3);
muxer mu2(m[2],s5,s9,c3);
muxer mu3(m[3],s6,s10,c3);
muxer mu4(m[4],s7,s11,c3);
muxer mu5(m[5],c7,c11,c3);
endmodule
modulefulladder (s,c,x,y,z);
outputs,c;
inputx,y,z;
xor (s,x,y,z);
assign c = ((x & y) | (y & z) | (z & x));
endmodule
modulemuxer (m,s1,s2,c);
output m;
input s1,s2,c;
wiref,g,h;  not
(f,c);
and (g,s1,c);
and (h,s2,f);
or (m,g,h);
endmodule
```

TEST BENCH

```
initial
begin
#100 x=12'b000000000000;y=12'b000000000000; z=1'b0;
#100 x=12'b000000001111;y=12'b111110000000; z=1'b0;
#100 x=12'b000011110000;y=12'b000000011111; z=1'b0;
#100 x=12'b111100001111;y=12'b011111100000; z=1'b1;
#100 x=12'b000111001111;y=12'b111111100000; z=1'b1;
#100 x=12'b111111111111;y=12'b111111111111; z=1'b1;
end endmodule
```

RESULT

Thus the Verilog programs for carry select adder were written, synthesized, simulated and implemented using Xilinx tools and FPGA.

EX.NO : 08

DATE :

DESIGN OF FPGA BASED 4 INPUTS AND 8 OUTPUTS MULTIPLIER

AIM

To design and implement FPGA based an array multiplier circuit for 4 inputs and 8 outputs using Verilog HDL.

APPARATUS REQUIRED

Synthesis Tools	:	Xilinx ISE. Simulation
Tools	:	Model sim Simulator.
Kit	:	FPGA Board- Spartan 3E
General	:	Personal Computer (PC).

THEORY

Binary multiplication can be accomplished by several approaches. The approach presented here is realized entirely with combinational circuits. Such a circuit is called an array multiplier.

The term array is used to describe the multiplier because the multiplier is organized as an array structure. Each row, called a partial product, is formed by a bit-by-bit multiplication of each operand.

For example, a partial product is formed when each bit of operand ‘a’ is multiplied by b0, resulting in a3b0, a2b0,a1b0, a0b0. The binary multiplication table is identical to the AND truth table.

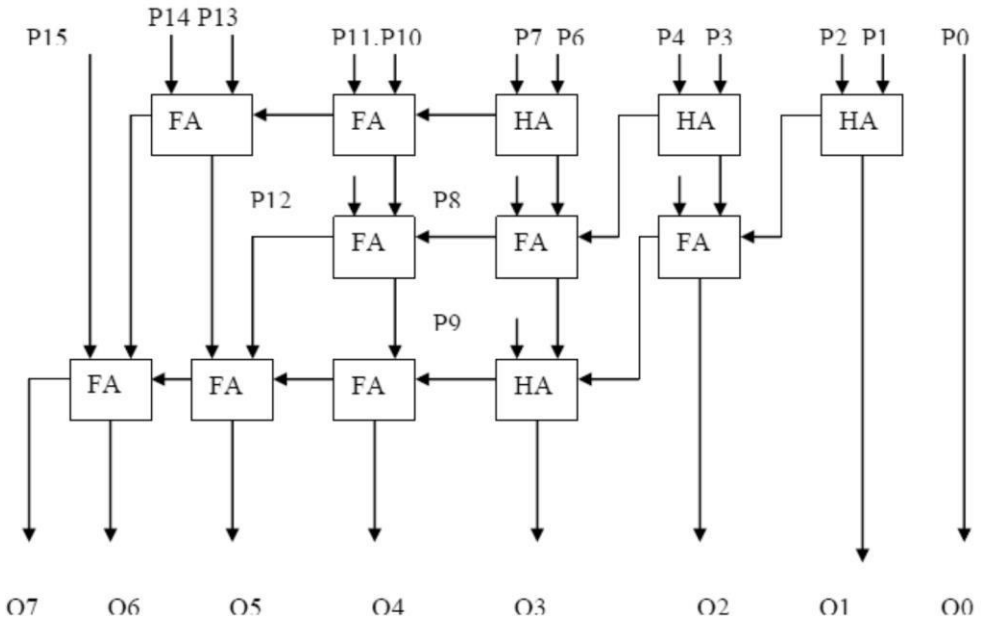
Each product bit {o(x)}, is formed by adding partial product columns. The product equations, including the carry-in {c(x)}, from column c(x-1), are (the plus sign indicates addition not OR).

Each product term, p(x), is formed by AND gates and collection of product terms needed for the multiplier. By adding appropriate p term outputs, the multiplier output equations are realized, as shown in figure.

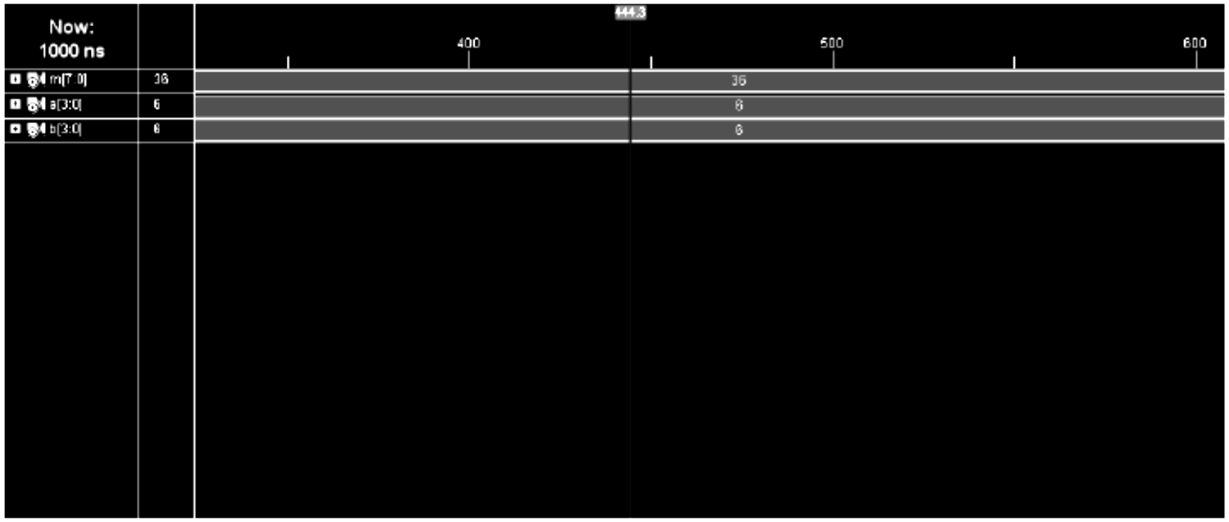
PROCEDURE

- The multiplexer circuit is designed and the Boolean function is found out.
- The Verilog Module Source for the circuit is written.
- It is implemented in Model Sim and Simulated.
- Signals are provided and Output Waveforms are viewed.

CIRCUIT DIAGRAM



OUTPUT



TEST BENCH

```
initial
begin
#100 a=4'b0000; b=4'b0000;
#100 a=4'b0001; b=4'b1100;
#100 a=4'b1100; b=4'b0011;
#100 a=4'b1111; b=4'b1111;
end endmodule
```

PROGRAM

```
Module mmmm(m,a,b);
input [3:0]a;
input [3:0]b;
output [7:0]m;
wire [15:0]p;
wire [12:1]s;
wire [12:1]c;
and(p[0],a[0],b[0]);
and(p[1],a[1],b[0]);
and(p[2],a[0],b[1]);
and(p[3],a[2],b[0]);
and(p[4],a[1],b[1]);
and(p[5],a[0],b[2]);
and(p[6],a[3],b[0]);
and(p[7],a[2],b[1]);
and(p[8],a[1],b[2]);
and(p[9],a[0],b[3]);
and(p[10],a[3],b[1]);
and(p[11],a[2],b[2]);
and(p[12],a[1],b[3]);
and(p[13],a[3],b[2]);
and(p[14],a[2],b[3]);
and(p[15],a[3],b[3]);
half ha1(s[1],c[1],p[1],p[2]);
half ha2(s[2],c[2],p[4],p[3]);
half ha3(s[3],c[3],p[7],p[6]);
full fa4(s[4],c[4],p[11],p[10],c[3]);
full fa5(s[5],c[5],p[14],p[13],c[4]);
full fa6(s[6],c[6],p[5],s[2],c[1]);
full fa7(s[7],c[7],p[8],s[3],c[2]);
full fa8(s[8],c[8],p[12],s[4],c[7]);
full fa9(s[9],c[9],p[9],s[7],c[6]);
half ha10(s[10],c[10],s[8],c[9]);
full fa11(s[11],c[11],s[5],c[8],c[10]);
full fa12(s[12],c[12],p[15],s[5],c[11]);
buf(m[0],p[0]);
buf(m[1],s[1]);
buf(m[2],s[6]);
buf(m[3],s[9]);
buf(m[4],s[10]);
buf(m[5],s[11]);
buf(m[6],s[12]);
buf(m[7],c[12]); endmodule
module half(s,co,x,y);
inputx,y;
outputs,co;
xor (s,x,y);
and (co,x,y);
endmodule
module full(s,co,x,y,ci);
inputx,y,ci;
outputs,co; wire
s1,d1,d2;
half     ha_1(s1,d1,x,y);
half     ha_2(s,d2,s1,ci);
oror_gate(co,d2,d1);
endmodule
```

RESULT

Thus the Verilog programs for multiplier were written, synthesized, simulated and implemented using Xilinx tools and FPGA.

EX.NO : 09

DATE :

DESIGN OF FPGA BASED UNIVERSAL SHIFT REGISTER

AIM:

To design and implement FPGA based Universal Shift Register using Verilog HDL.

APPARATUS REQUIRED:

Synthesis Tools	:	Xilinx ISE. Simulation
Tools	:	Model sim Simulator.
Kit	:	FPGA Board- Spartan 3E
General	:	Personal Computer (PC).

THEORY:

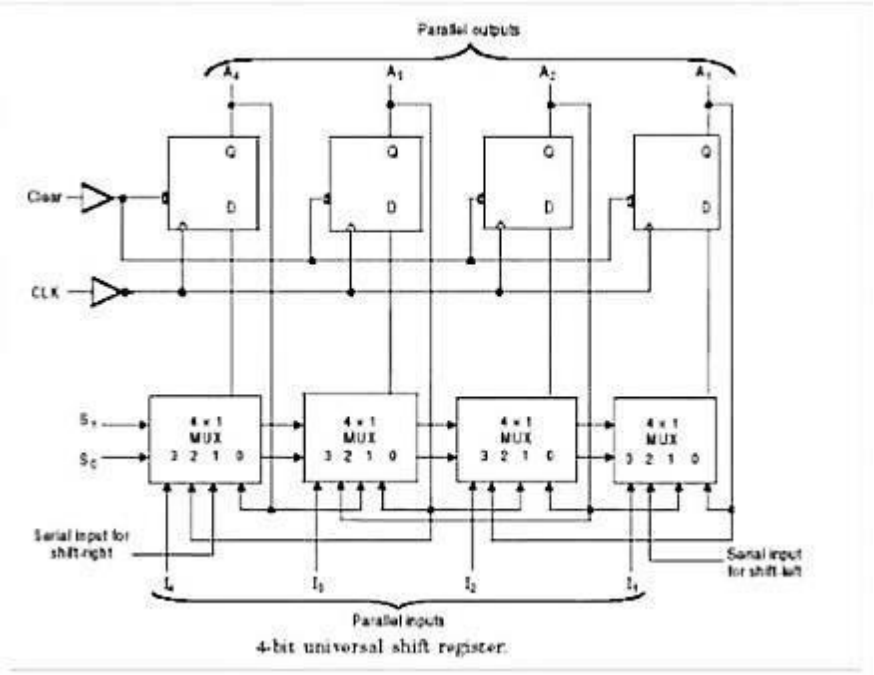
Universal shift register is capable of converting input data to parallel or serial which also does shifting of data bidirectional, unibirectional (SISO, SIPO, PISO, PIPO) and also parallel load this is called Universal shift register.

Shift Register are used as: Data storage device, Delay element, Communication lines, Digital Electronics devices (Temporary data storage, data transfer, data manipulation, counter), etc.

PROCEDURE:

- The ALU circuit is designed and the Boolean function is found out.
- The Verilog Module Source for the circuit is written.
- It is implemented in Model Sim and Simulated.
- Signals are provided and Output Waveforms are viewed.

CIRCUIT DIAGRAM



4-bit universal shift register

TABLE:

MODE CONTROL		REGISTER OPERATION
S1	S0	
0	0	NO CHANGE
0	1	SHIFT LEFT
1	0	SHIFT RIGHT
1	1	PARALLEL LOAD

PROGRAM

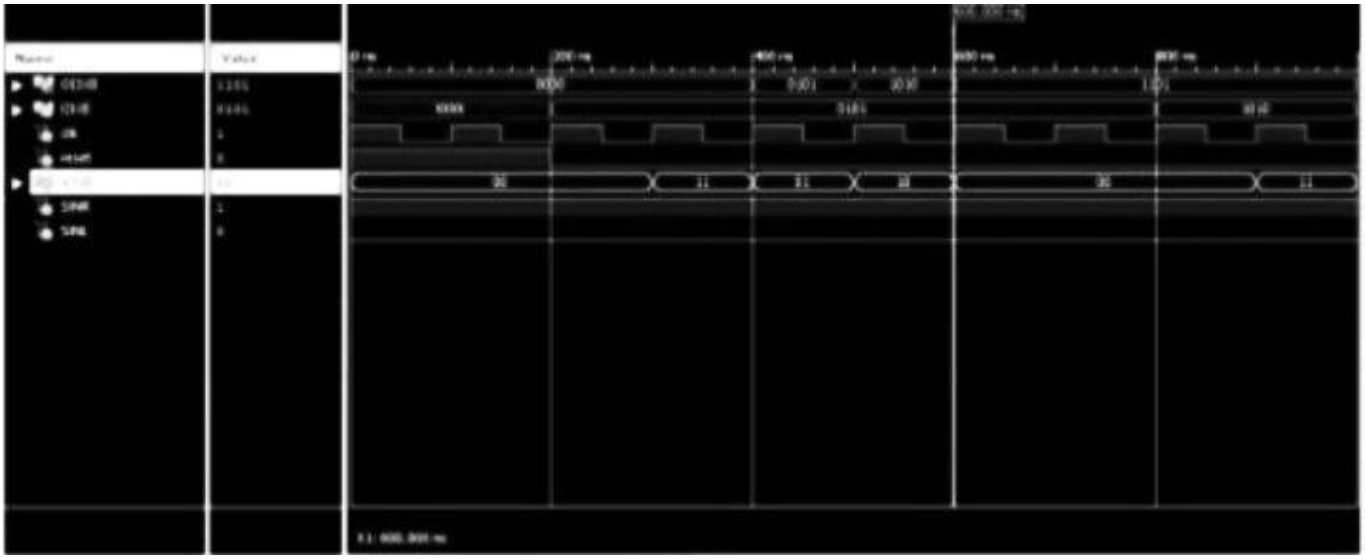
```
module usr (O, I, clk, reset, s, SINR, SINL); wire
[3:0] w;
input [3:0] I;
input [1:0] s;
input clk;
input reset, SINR, SINL;
output [3:0] O; reg[27:0]
count = 0;
mux _4_1 m1(w[0], s[1], s[0], I[0], SINL, O[1], O[0]);
mux _4_1 m2(w[1], s[1], s[0], I[1], O[0], O[2], O[1]);
mux _4_1 m3(w[2], s[1], s[0], I[2], O[1], O[3], O[2]);
mux _4_1 m4(w[3], s[1], s[0], I[3], O[2], SINR, O[3]);
D_FF d1(O[0],w[0],clk,reset);
D_FF d2(O[1],w[1],clk,reset);
D_FF d3(O[2],w[2],clk,reset);
D_FF d4(O[3],w[3],clk,reset);
Endmodule
module D_FF(q, d, clk, reset); output
reg q;
input d, clk, reset; always
@ (posedge clk) if (reset
== 1'b1)
q<=1'b0;
else
q<=d;
endmodule
module mux_4_1(y,s1,s0,i3,i2,i1,i0);
output reg y;
input i3,i2,i1,i0; input
s1,s0;
always@(s1,s0,i3,i2,i1,i0) begin
if(s0==0 & s1==0)
y=i0
```

```
elseif(s0==0 & s1==1) y=i1
elseif(s0==1 & s1==0) y=i2
elseif(s0==1 & s1==1) y=i3
end endmodule
```

TEST BENCH

```
//initialize inputs
//I=4'b0000;
clk=1'b1;
reset=1'b1;
SINR=1'b1;
SINL=1'b0;
S=2'b00;
#100 reset= 1'b1; s=2'b00;
#100 I=4'b0101; reset=2'b10;
#100 s=2'b11;
#100 s=2'b01;
#100 s=2'b10;
#100 s=2'b00;
//wait 100ns for global reset to finish #100;
//add stimulus here
#100 I=4'b1010; reset=2'b10;
#100 s=2'b11;
#100 s=2'b01;
#100 s=2'b10;
#100 s=2'b00;
end
always #50 clk=~clk;
endmodule
```

OUTPUT



RESULT

Thus the Verilog programs for universal shift register were written, synthesized, simulated and implemented using Xilinx tools and FPGA.

EX.NO : 10

DATE :

DESIGN OF FPGA BASED FINITE STATE MACHINE

AIM

To design and implement FPGA based finite state machine using Verilog HDL.

APPARATUS REQUIRED

Synthesis Tools	:	Xilinx ISE. Simulation
Tools	:	Model sim Simulator.
Kit	:	FPGA Board- Spartan 3E
General	:	Personal Computer (PC) with Parallel to JTAG cable.

THEORY

FSM consists of combinational, sequential and output logic. Combinational logic is used to decide the next state of the FSM, sequential logic is used to store the current state of the FSM. The output logic is a mixture of both combinationl and sequential logic.

Types of state machines:

There are many ways to code these state machines, but before we get into the coding styles, let's first understand the basics a bit. There are two types of state machines:

Mealy State Machine : Its output depends on current state and current inputs. In the above picture, the blue dotted line makes the circuit a mealy state machine.

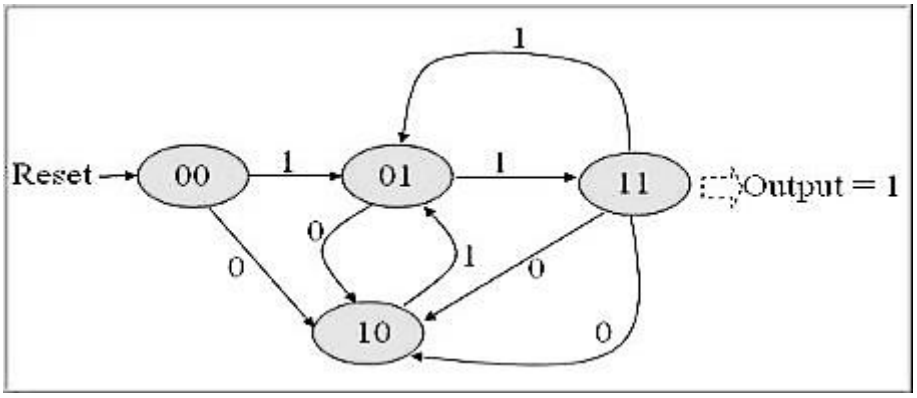
Moore State Machine : Its output depends on current state only. In the above picture, when blue dotted line is removed the circuit becomes a Moore state machine.

PROCEDURE

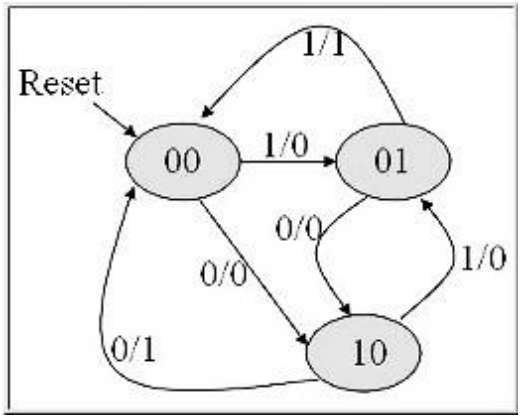
- The Finite state machine circuit is designed and the Boolean function is found out.
- The Verilog Module Source for the circuit is written.
- It is implemented in Model Sim and Simulated.
- Signals are provided and Output Waveforms are viewed.

CIRCUIT DIAGRAM

MOORE MACHINE: STATE DIAGRAM



MEALY MACHINE: STATE DIAGRAM



PROGRAM

Moore State Machine

```
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
```

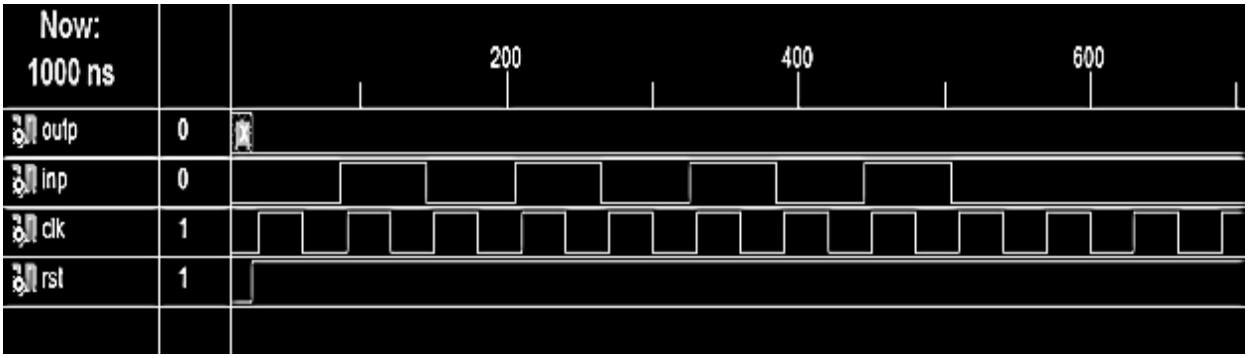
Mealy State Machine

```
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
```

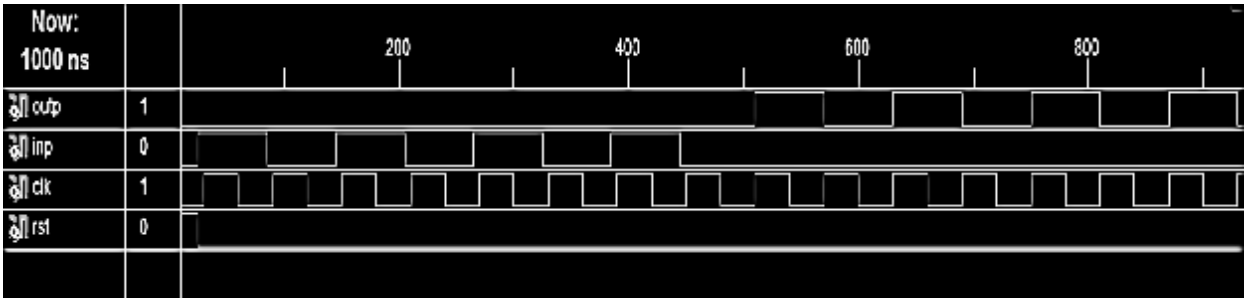
```
state <= 2'b01;
outp  <= 0;  end
else begin state
<= 2'b00; outp
<= 1;
end
end
default: begin
state <= 2'b00;
outp <= 0;
end
endcase
end
end endmodule
```

OUTPUT

Moore State Machine



Mealy State Machine:



RESULT

Thus the Verilog programs for Finite State machine were written, synthesized, simulated and implemented using Xilinx tools and FPGA.

EX.NO : 11

DATE :

DESIGN OF FPGA BASED MEMORIES (ACCUMULATOR)

AIM

To design and implement FPGA based memories (Accumulator) using Verilog HDL.

APPARATUS REQUIRED

Synthesis Tools	:	Xilinx ISE. Simulation
Tools	:	Model sim Simulator.
Kit	:	FPGA Board- Spartan 3E
General	:	Personal Computer (PC) with Parallel to JTAG cable.

THEORY

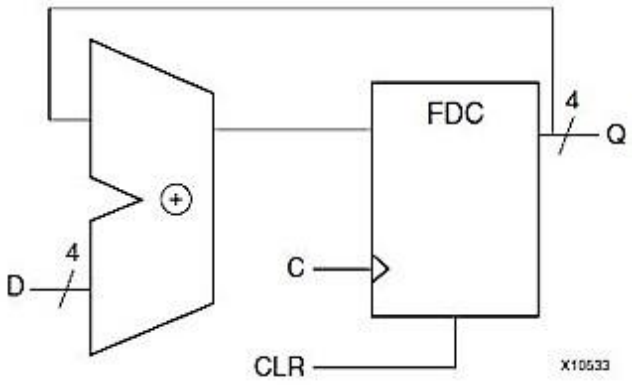
An accumulator is a register in which intermediate arithmetic and logic results are stored. Without a register like an accumulator, it would be necessary to write the result of each calculation (addition, multiplication, shift, etc.) To main memory, perhaps only to be read right back again for use in the next operation. Access to main memory is slower than access to a register like the accumulator because the technology used for the large main memory is slower (but cheaper) than that used for a register.

The canonical example for accumulator use is summing a list of numbers. The accumulator is initially set to zero, then each number in turn is read and added to the value in the accumulator. Only when all numbers have been added is the result held in the accumulator written to main memory or to another, non-accumulator, CPU register.

PROCEDURE

- The accumulator circuit is designed and the function is found out.
- The Verilog Module Source for the circuit is written.
- It is implemented in Model Sim and Simulated.
- Signals are provided and Output Waveforms are viewed.

CIRCUIT DIAGRAM



PROGRAM

```
module accu (in, acc, clk, reset); input
[7:0] in;
input clk, reset;
output [7:0] acc;
reg [7:0] acc;
always@(clk) begin
if(reset)
acc <= 8'b00000000;
else
acc <= acc + in;
end  endmodule
```

OUTPUT

RESULT

Thus the Verilog programs for memories (Accumulator) were written, synthesized, simulated and implemented using Xilinx tools and FPGA.

EX.NO : 12

DATE :

DESIGN OF FPGA BASED 4 BIT SYNCHRONOUS COUNTER USING FLIP FLOP

AIM

To design and implement FPGA based 4 bit synchronous counter using Verilog HDL.

APPARATUS REQUIRED

Synthesis Tools	:	Xilinx ISE. Simulation
Tools	:	Model sim Simulator.
Kit	:	FPGA Board- Spartan 3E
General	:	Personal Computer (PC) with Parallel to JTAG cable.

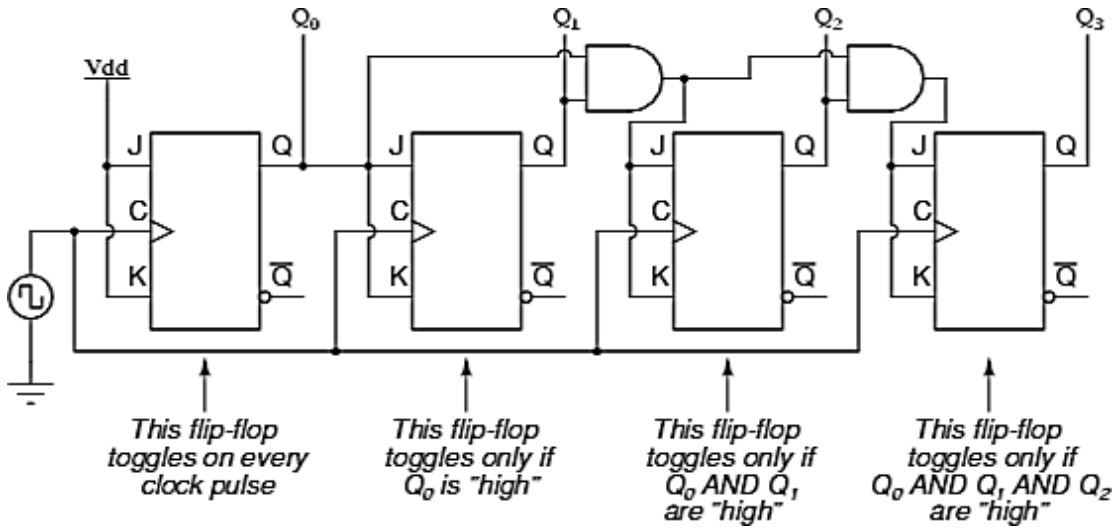
THEORY

A synchronous counter, in contrast to an asynchronous counter, is one whose output bits change state simultaneously, with no ripple. The only way we can build such a counter circuit from J-K flip-flops is to connect all the clock inputs together, so that each and every flip-flop receives the exact same clock pulse at the exact same time. The figure shows a four- bit synchronous "up" counter. Each of the higher-order flip-flops are made ready to toggle (both J and K inputs "high") if the Q outputs of all previous flip-flops are "high." Otherwise, the J and K inputs for that flip-flop will both be "low," placing it into the "latch" mode where it will maintain its present output state at the next clock pulse. Since the first (LSB) flip-flop needs to toggle at every clock pulse, its J and K inputs are connected to V_{CC} or Vdd, where they will be "high" all the time. The next flip-flop need only "recognize" that the first flip- flop's Q output is high to be made ready to toggle, so no AND gate is needed. However, the remaining flip-flops should be made ready to toggle only when all lower-order output bits are "high," thus the need for AND gates.

PROCEDURE

- The 4 bit synchronous counter using flip flop circuit is designed and the function is found out.
- The Verilog Module Source for the circuit is written.
- It is implemented in Model Sim and Simulated.
- Signals are provided and Output Waveforms are viewed.

CIRCUIT DIAGRAM



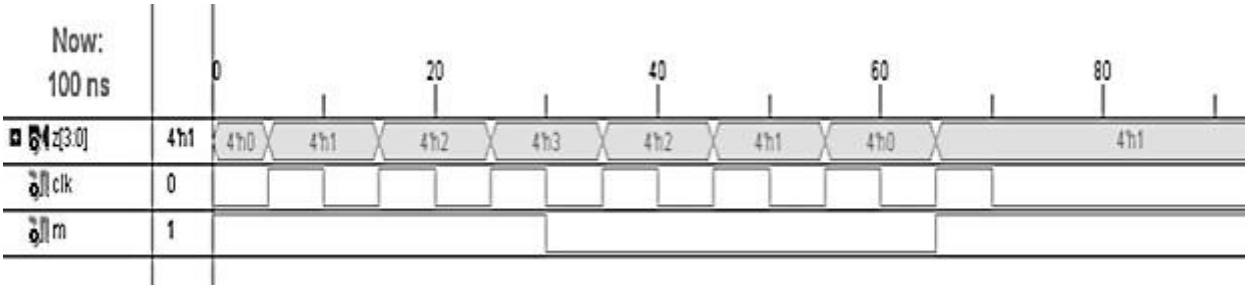
PROGRAM

```
module syn(m,clk, z);
input m,clk;
output [3:0] z;
reg[3:0] z;
initial  begin
z=4'b0000;
end
always@(posedge clk)
begin
if(m==1)
z=z+1;
else
z=z-1;
end
endmodule
```

TEST BENCH

```
initial
begin
#100 clk=0; m=1;
#100 clk=1; m=1;
#100 clk=0; m=1;
#100 clk=1; m=1;
#100 clk=0; m=1;
#100 clk=1; m=1;
#100 clk=0; m=0;
#100 clk=1; m=0;
#100 clk=0; m=0;
#100 clk=1; m=0;
#100 clk=0; m=0;
#100 clk=1; m=0;
#100 clk=0; m=0;
#100 clk=1; m=1;
#100 clk=0; m=1;
end endmodule
```

OUTPUT



RESULT

Thus the Verilog programs for 4 bit synchronous counter were written, synthesized, simulated and implemented using Xilinx tools and FPGA.

EX.NO : 13

DATE :

DESIGN OF CMOS INVERTER USING S-EDIT

AIM

To perform the functional verification of the CMOS Inverter through schematic entry.

APPARATUS REQUIRED

Synthesis Tools	:	Tanner.
Simulation Tools	:	S-EDIT.
General	:	Personal Computer (PC).

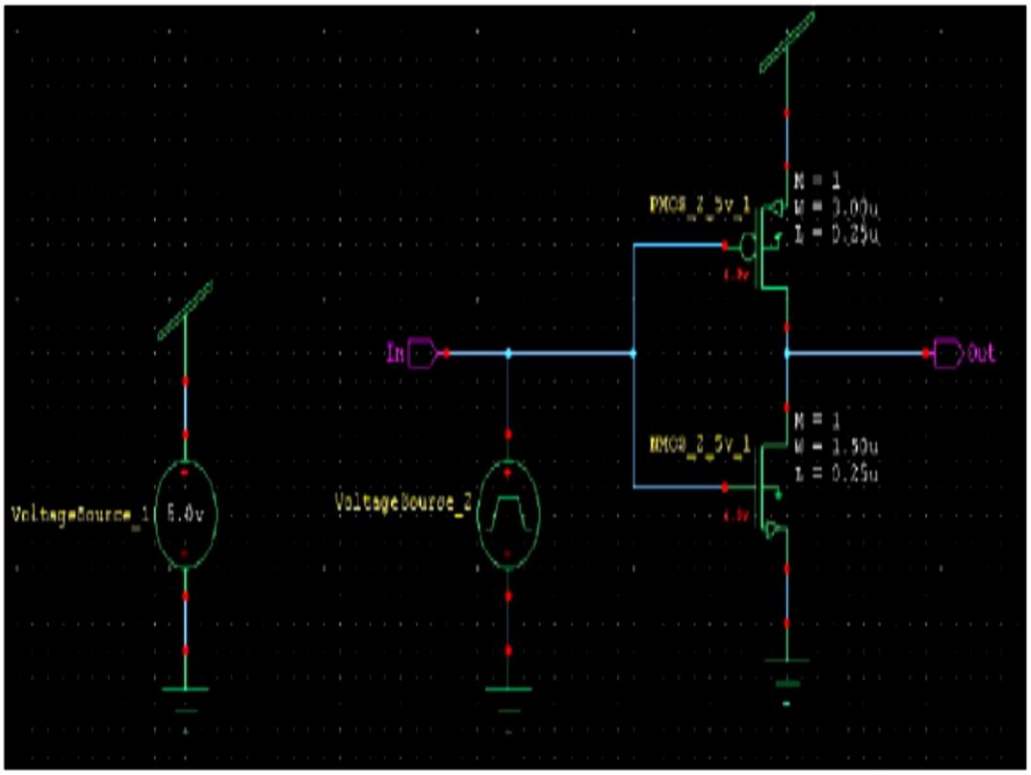
THEORY

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.

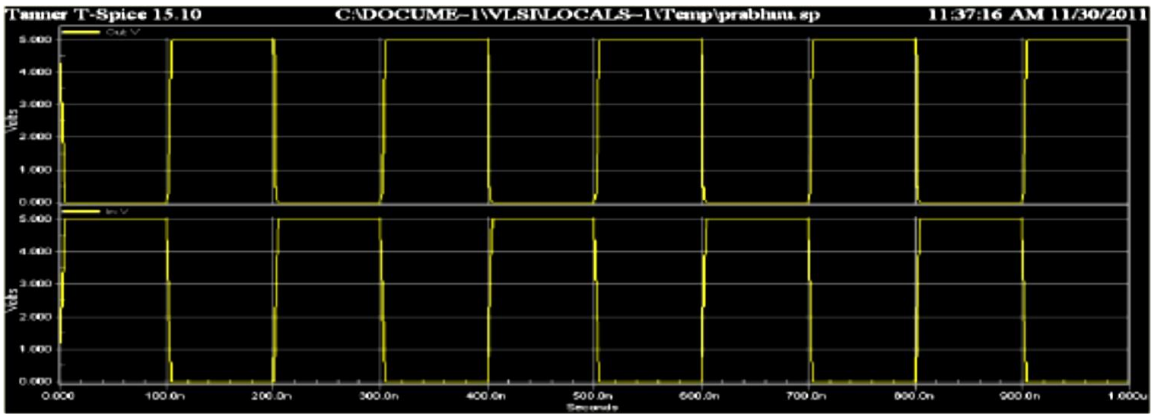
PROCEDURE

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

SCHEMATIC DIAGRAM



OUTPUT



RESULT

Thus the functional verification of the CMOS Inverter through schematic entry and the output also verified successfully.

EX.NO :14.a

DATE :

DESIGN OF COMMON SOURCE AMPLIFIER USING S-EDIT

AIM

To perform the functional verification of the common source through schematic entry.

APPARATUS REQUIRED

Synthesis Tools	:	Tanner.
Simulation Tools	:	S-EDIT.
General	:	Personal Computer (PC).

THEORY

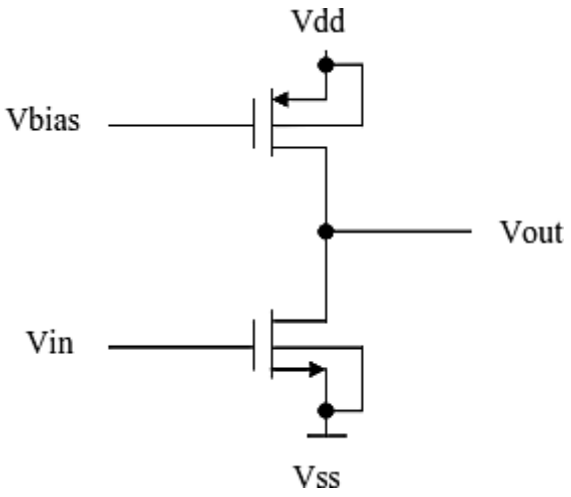
In electronics, a common-source amplifier is one of three basic single-stage field- effect transistor (FET) amplifier topologies, typically used as a voltage or transconductance amplifier. The easiest way to tell if a FET is common source, common drain, or common gate is to examine where the signal enters and leaves. The remaining terminal is what is known as "common". In this example, the signal enters the gate, and exits the drain. The only terminal remaining is the source. This is a common-source FET circuit. The analogous bipolar junction transistor circuit is the common-emitter amplifier.

The common-source (CS) amplifier may be viewed as a transconductance amplifier or as a voltage amplifier. (See classification of amplifiers). As a transconductance amplifier, the input voltage is seen as modulating the current going to the load. As a voltage amplifier, input voltage modulates the amount of current flowing through the FET, changing the voltage across the output resistance according to Ohm's law. However, the FET device's output resistance typically is not high enough for a reasonable transconductance amplifier (ideally infinite), nor low enough for a decent voltage amplifier (ideally zero). Another major drawback is the amplifier's limited high-frequency response.

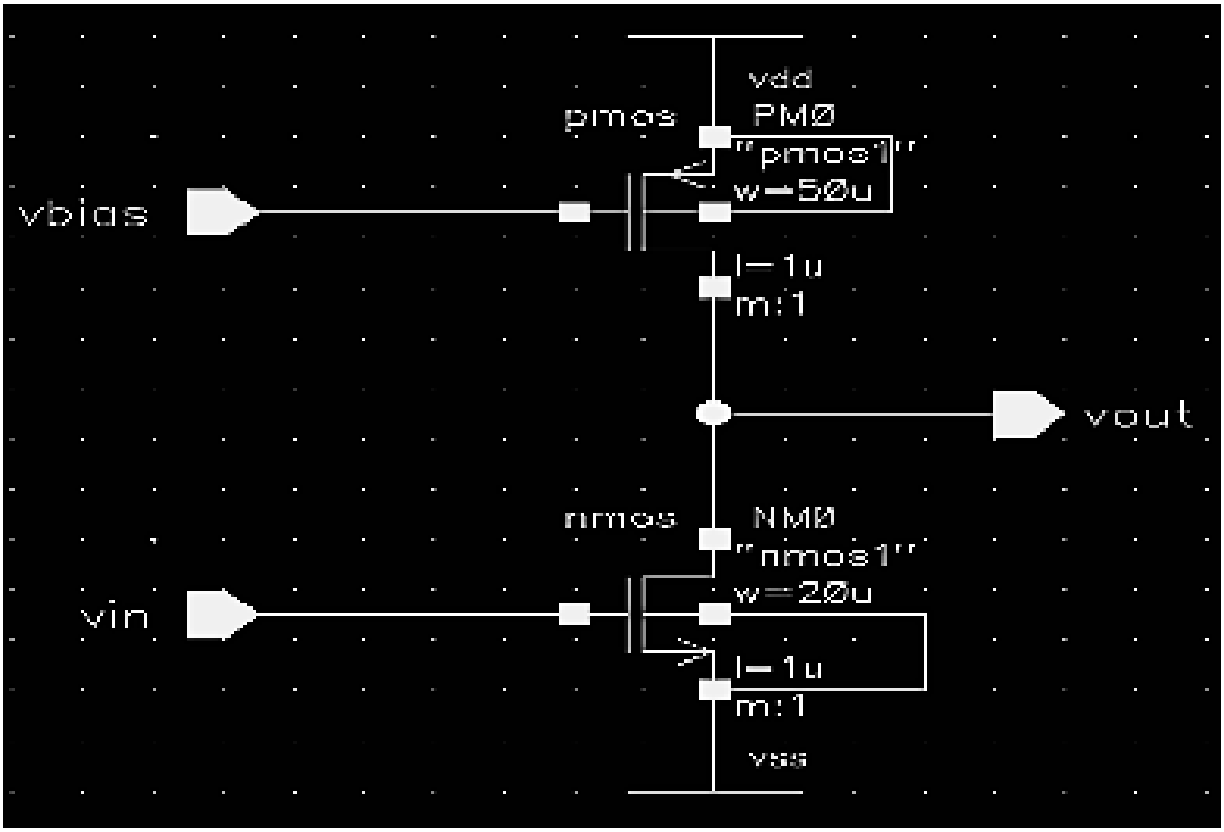
PROCEDURE

- Draw the schematic of common source using S-edit.
- Perform Transient Analysis of the common source.
- Obtain the output waveform from W-edit
- Obtain the spice code using T-edit.

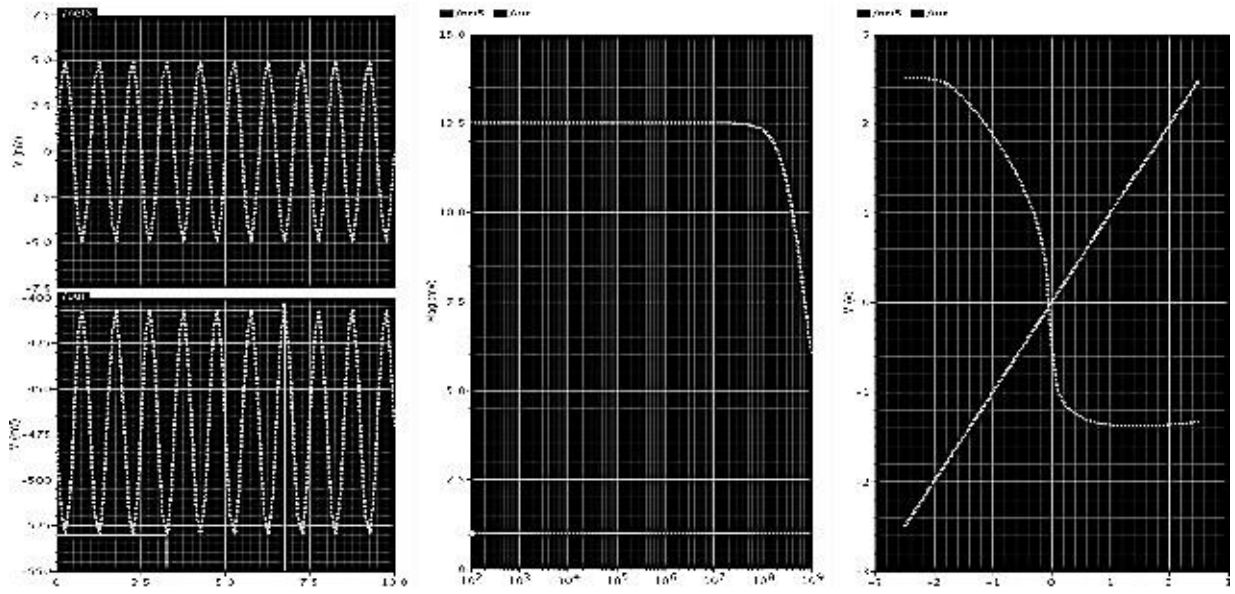
CIRCUIT DIAGRAM



SCHEMATIC DIAGRAM



OUTPUT



RESULT

Thus the functional verification of the common source amplifier through schematic entry and the output also verified successfully.

EX.NO : 14.b

DATE :

DESIGN OF COMMON DRAIN USING S-EDIT

AIM

To perform the functional verification of the common drain through schematic entry.

APPARATUS REQUIRED

Synthesis Tools	:	Tanner.
Simulation Tools	:	S-EDIT.
General	:	Personal Computer (PC).

THEORY

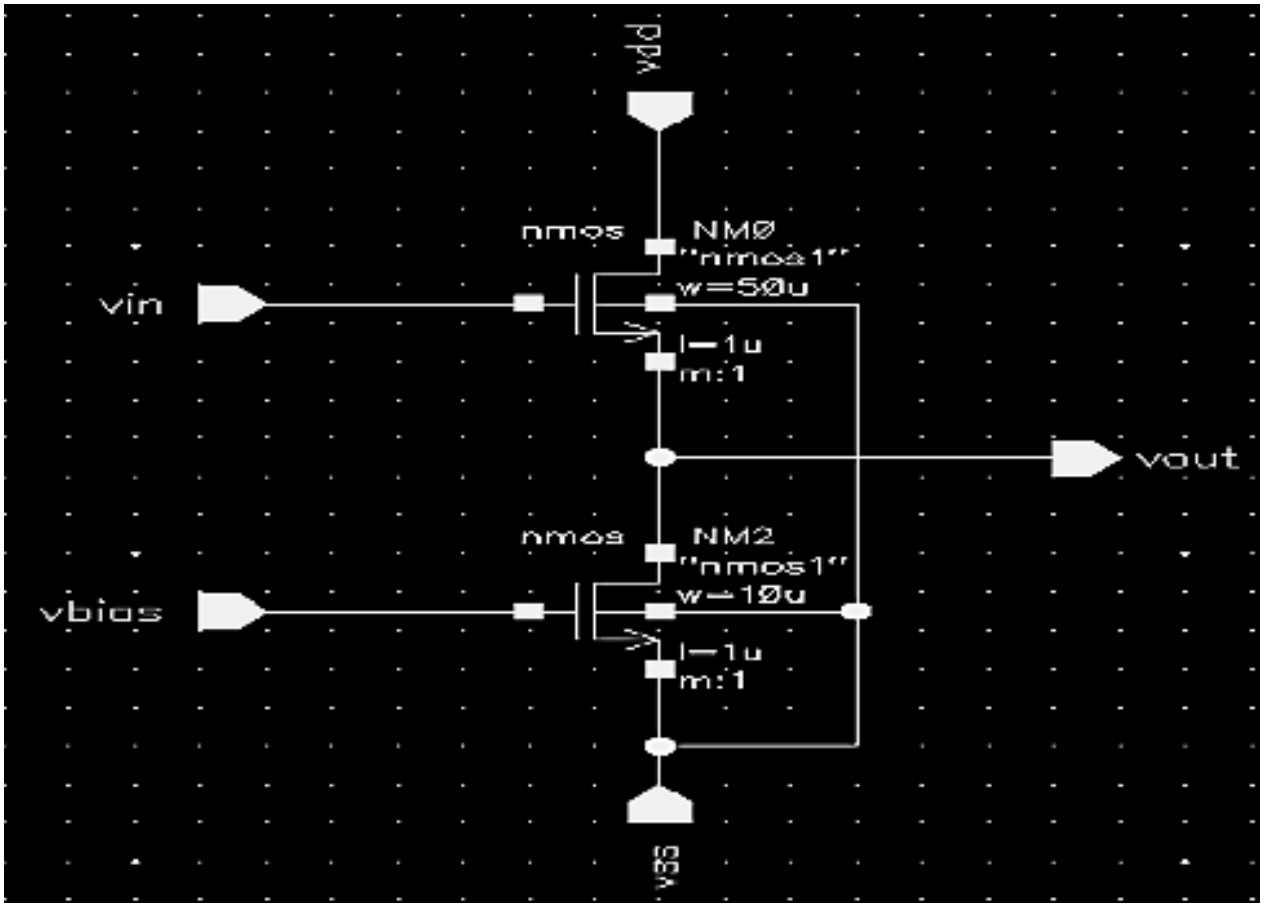
Common drain amplifier is a source follower or buffer amplifier circuit using a MOSFET. The output is simply equal to the input minus about 2.2V. The advantage of this circuit is that the MOSFET can provide current and power gain; the MOSFET draws no current from the input. It provides low output impedance to any circuit using the output of the follower, meaning that the output will not drop under load.

Its output impedance is not as low as that of an emitter follower using a bipolar transistor (as you can verify by connecting a resistor from the output to -15V), but it has the advantage that the input impedance is infinite. The MOSFET is in saturation, so the current across it is determined by the gate source voltage. Since a current source keeps the current constant, the gate-source voltage is also constant.

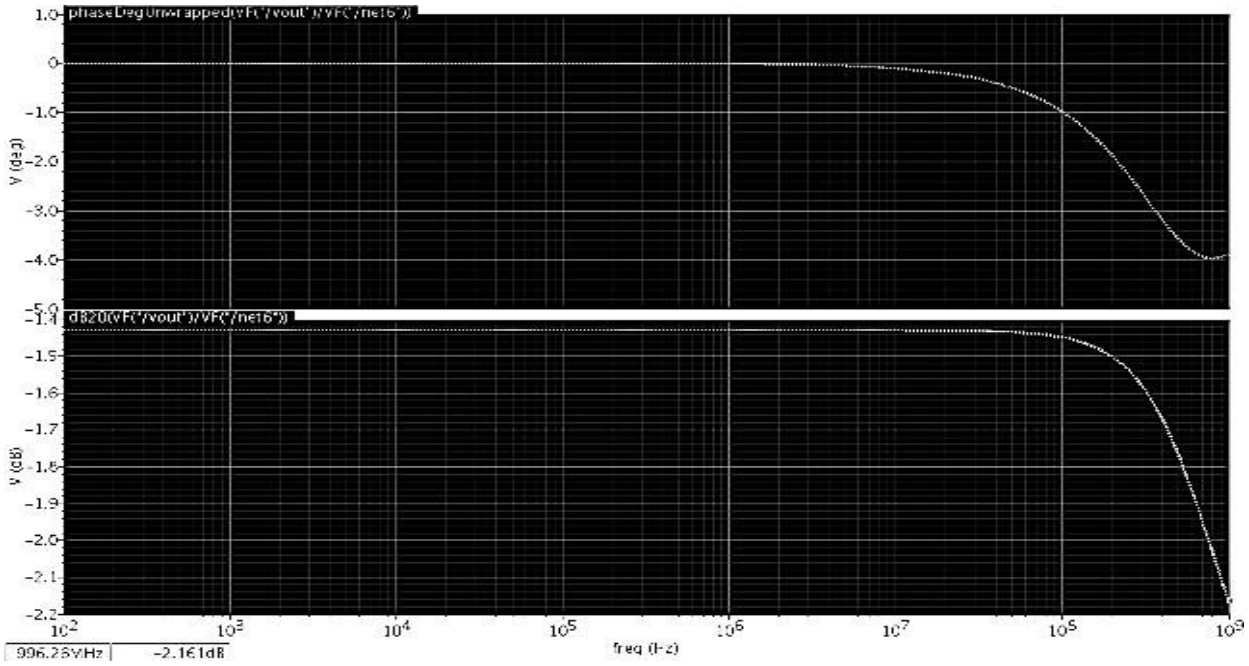
PROCEDURE

- Draw the schematic of common drain using S-edit.
- Perform Transient Analysis of the common drain.
- Obtain the output waveform from W-edit
- Obtain the spice code using T-edit.

SCHEMATIC DIAGRAM



OUTPUT



RESULT

Thus the functional verification of the common drain through schematic entry and the output also verified successfully.

EX.NO : 15

DATE :

DESIGN AND SIMULATION OF A DIFFERENTIAL AMPLIFIER. MEASURE GAIN, ICMR, AND CMRR

AIM

To calculate the gain, bandwidth and CMRR of a differential amplifier through schematic entry.

APPARATUS REQUIRED

Synthesis Tools	:	Tanner.
Simulation Tools	:	S-EDIT.
General	:	Personal Computer (PC).

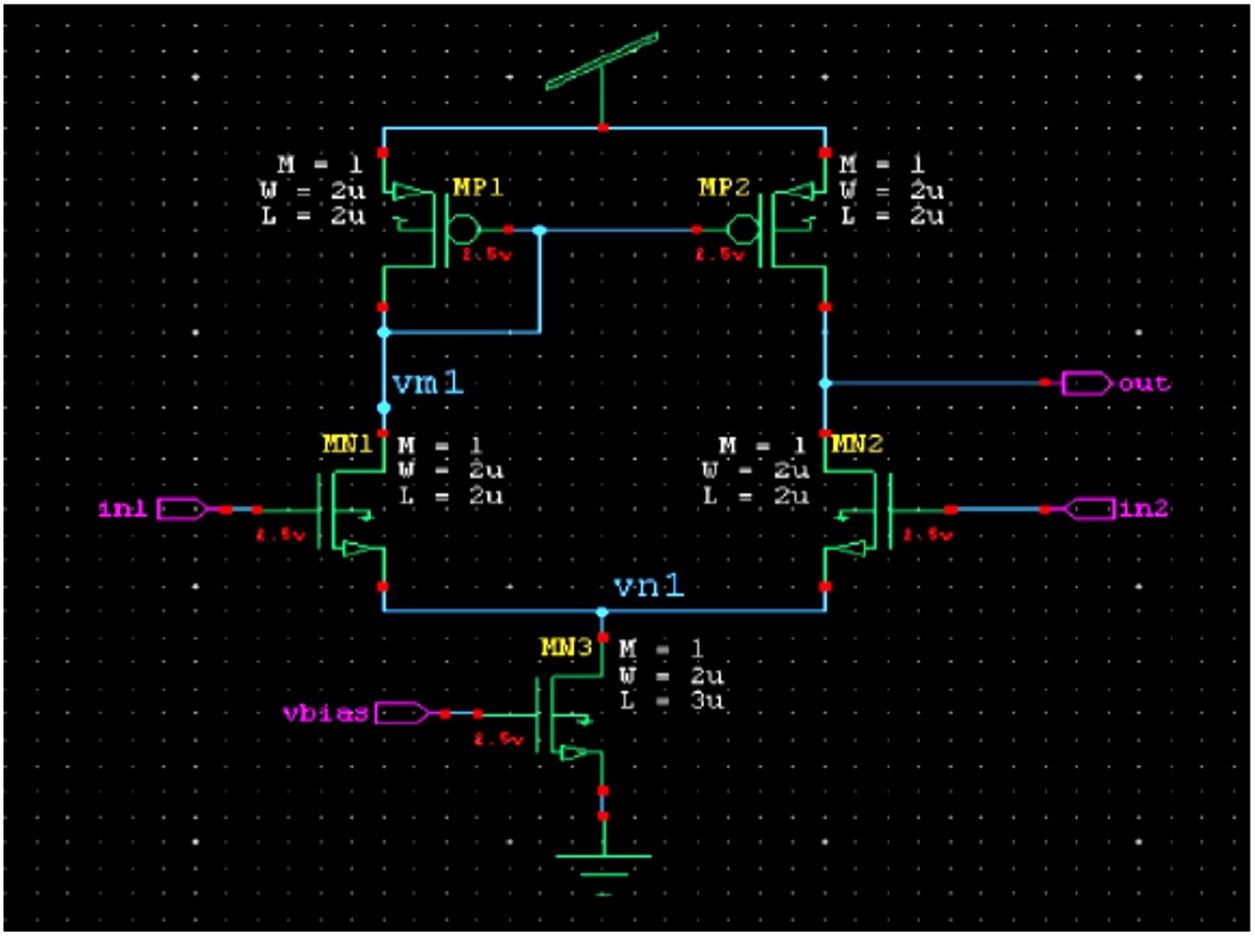
THEORY

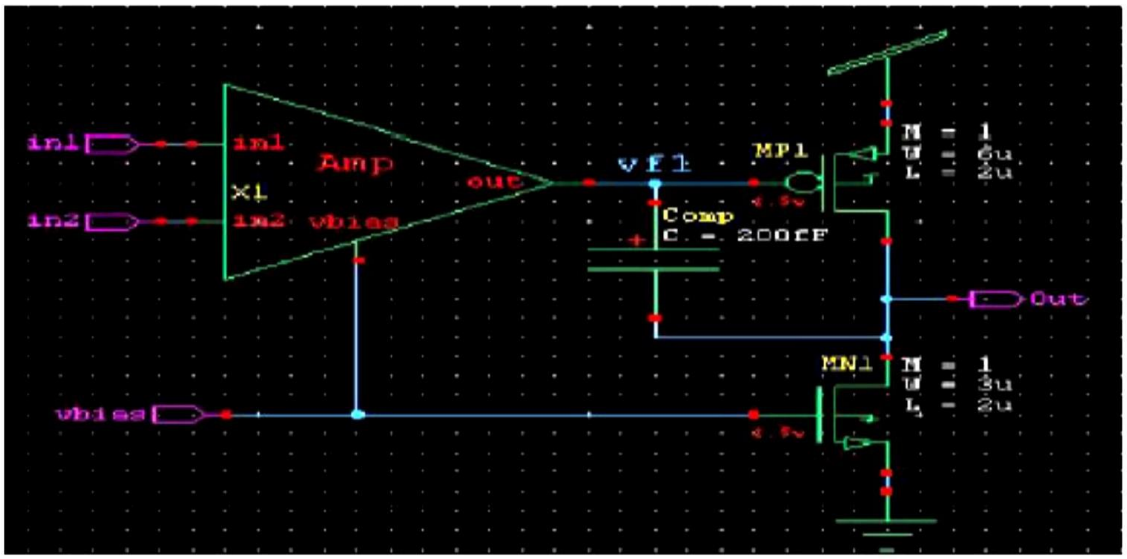
The differential amplifier is probably the most widely used circuit building block in analog integrated circuits, principally op amps. The differential amplifier can be implemented with BJTs or MOSFETs. A differential amplifier multiplies the voltage difference between two inputs (Vin+ - Vin) by some constant factor Ad, the differential gain. It may have either one output or a pair of outputs where the signal of interest is the voltage difference between the two outputs. A differential amplifier also tends to reject the part of the input signals that are common to both inputs (Vin+ + Vin-)/2. This is referred to as the common mode signal.

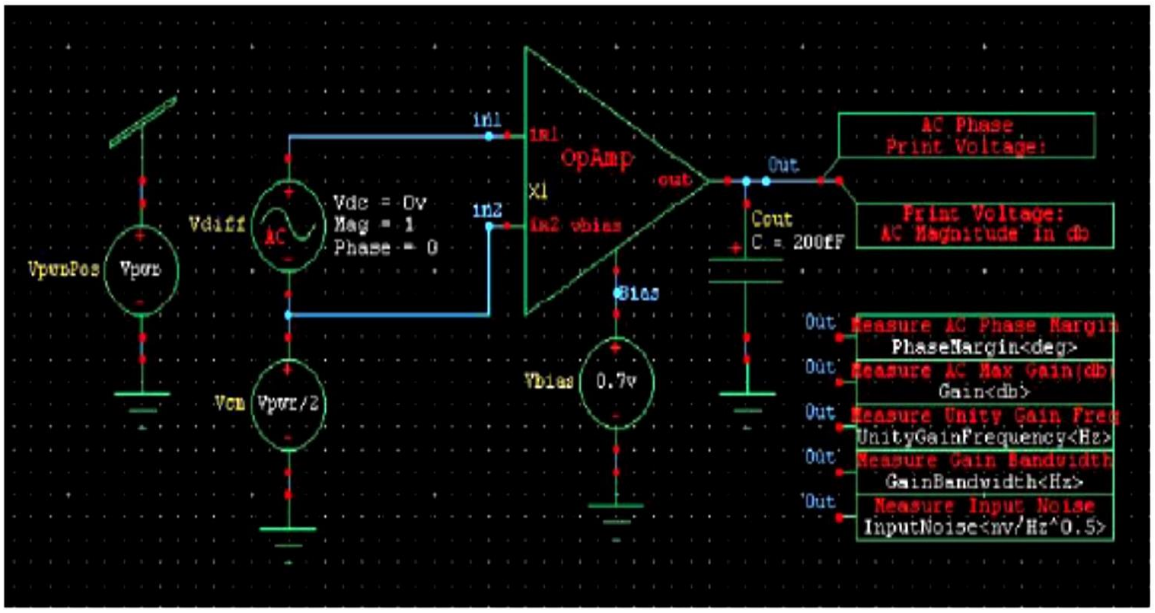
PROCEDURE

- Draw the schematic of differential amplifier using S-edit and generate the symbol.
- Draw the schematic of differential amplifier circuit using the generated symbol.
- Perform AC Analysis of the differential amplifier.
- Obtain the frequency response from W-edit.
- Obtain the spice code using T-edit.

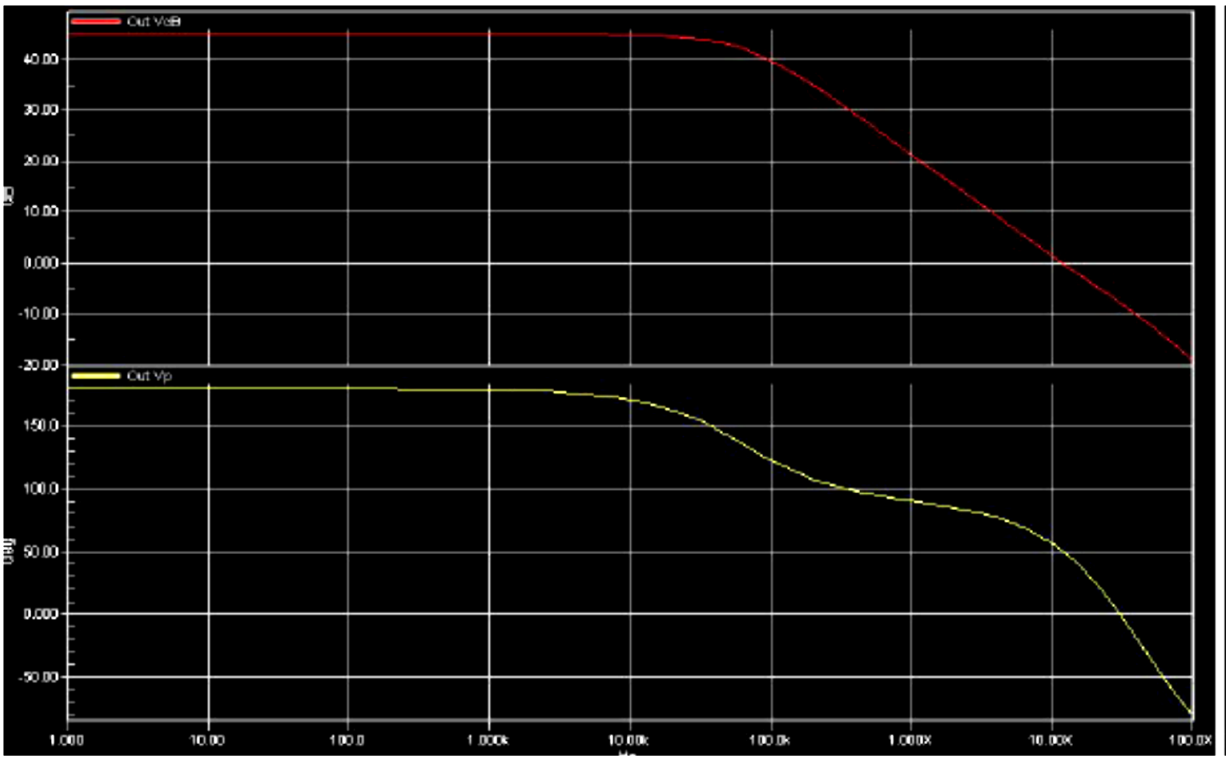
SCHEMATIC DIAGRAM







OUTPUT



RESULT

Thus the functional verification of the Differential Amplifier through schematic entry and the output also verified successfully.