MAHA BARATHI ENGINEERING COLLEGE

NH-79, SALEM-CHENNAI HIGHWAY, A.VASUDEVANUR, CHINNASALEM (TK), KALLAKURICHI (DT) 606 201.
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai
Accredited by NAAC and Recognized under section 2(f) & 12(B) status of UGC, New Delhi

www.mbec.ac.in | Ph: 04151-256333, 257333 | E-mail: mbec123@gmail.com

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

EC3561-VLSI LABORATORY

III Year/ V Semester B.E ECE

Regulation 2021

(As Per Anna University, Chennai syllabus)

Prepared By	Verified By	Approved By
Staff I/C	(HoD/ECE)	(Principal)

MAHA BARATHI ENGINEERING COLLEGE

NH-79, Salem-Chennai Highway, A.Vasudevanur, Chinnasalem TK, Kallakurichi Dt – 606 201.
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai
Accredited by NAAC and Recognized under section 2(f) & 12(B) status of UGC, New Delhi.

www.mbec.ac.in | 04151-256333, 257333 | mbec123@gmail.com

Bonafide Certificate

Certified that this is a bonafide record of workdone by Selvan/Selvi……………………………………………………………. ………..

Reg.No…………………………of……Fifth……………………………Semester ……………………Electronics and Communication Engineering…………Branch

of …B.E…..Degree Examination in the subject…EC3561-VLSI Laboratory…… ………………………………………………………

Staff Incharge **Head of the Department**

Date:

Submitted for Anna University Practical Examination conducted on…………………

Internal Examiner **External Examiner**

CONTENT					
S. No.	**Date**	**Name of the Experiment**	**Page No.**	**Marks Awarded**	**Signature of the Staff Incharge**
1		Combinational and Sequential Circuits			
2		8- Bit Adder and Multiplier			
3		Universal Shift Register			
4		Random Access Memory			
5		Finite state machine			
6		3-Bit Synchronous Up/Down Counter			
7		4-Bit Asynchronous Up/Down Counter			
8		CMOS Nand, Nor Gate and Flipflop			
9		Synchronous Counter			
10		Differential Amplifier			
11		Design And Simulate The Analysis Of Source Followers			
12		CMOS Inverting Amplifier			

Expt No: 1

Date: COMBINATIONAL AND SEQUENTIAL CIRCUITS

Aim:

To Design and Simulate basic gates, half & full adder,half & full Subtractor, multiplexer, decoder, comparator and flip-flopsusing Verilog HDL and also implement in FPGA Kit.

APPARATUS REQUIRED:

- PC with Windows XP.
- XILINX software.

Theory:

A logic gate is a physical model of a boolean function that is, it performs a logical operation on one or more logic inputs and produces a single logic output. Logic gates are primarily implemented electronically using diodes or transistors, but can also be constructed using electromagnetic relays (relay logic), fluidic logic, pneumatic logic, optics, molecules, or even mechanicalelements.With amplification, logic gates can be cascaded in the same way that Boolean functions can be composed, allowing the construction of a physical model of all of Boolean logic, and therefore, all of the algorithms and mathematics that can be described with Boolean logicSynthesis is the process of constructing a gate level netlist from a register-transfer Level models of the circuit described in Verilog HDL ,VHDL or mixed language designs.

The netlist files contain both logical design data and constraints .
XILINX SYNTHESIS TOOL enable us to study

1. Utilization of LUTs & Slices
2. I/O Buffers assignment
3. RTL schematic in gate level
4. Time delay between i/Os and path

.

<u>PROCEDURE</u>:

1. Start the Xilinx ISE by using start→ program file → Xilinx ISE (8.1i) → Project navigator
2. File → New Project
3. Enter the project name and location then click next
4. select the Device and other category and click next twice and finish
5. Click on the symbol of FPGA device and then right click → click on new source
6. Select the Verilog Module and give the file name → click next and define ports → click next and finish
7. Writing the behavioral verilog code in verilog Editor
8. Run the Check syntax → process window → synthesize → double click check syntax and remove errors, if present , with proper syntax & coding.
9. synthesis your design, from the source window select, Synthesis/Implementation from the window Now double click the synthesis →XSt
10. After the HDL synthesiss phase of the synthesis process,you can display a schematic representation of your synthesized source file.This schematic shows a representation of the pre-optimized design in terms of generic symbols,such as adders,multipliers,counters,Ann gates and OR gates → double click View RTL schematic
11. Double click the schematic to internal view
12. Double click outside the schematic to move one-level back
13. This Schematic Shows a representation of the design in terms of logic elements optimized to the target device. For example,in terms of LUTs(Look Up Table),carry logic,I/O buffers and other technology-specific components

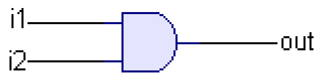
→Double ***click View technology Schematic***

14.Double click the Schematic to inner view

15.Double click the LUT to inner View.This is gate lenel view of LUT ,if you want see ***Truth Table and K-Map*** for your design just click the respective tabs

16.After finishing the synthesis,you can view number of Slices,LUT(Look Up Table),I/Os are taken by your design in Device using Design Summary

AND Gate:



PROGRAM:

AND Gate:

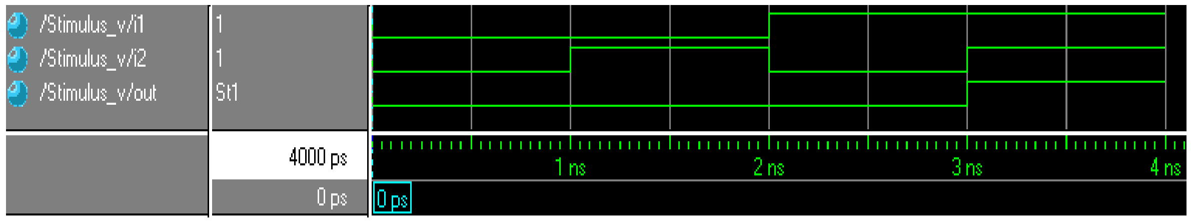
```
moduleAndgate(i1, i2, out);
input i1;
input i2;
output out;
        and (out,i1,i2);
endmodule
```

Truth table:

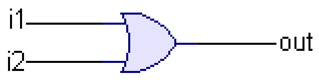
AND Gate

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

OUTPUT WAVE



OR Gate:



Program:

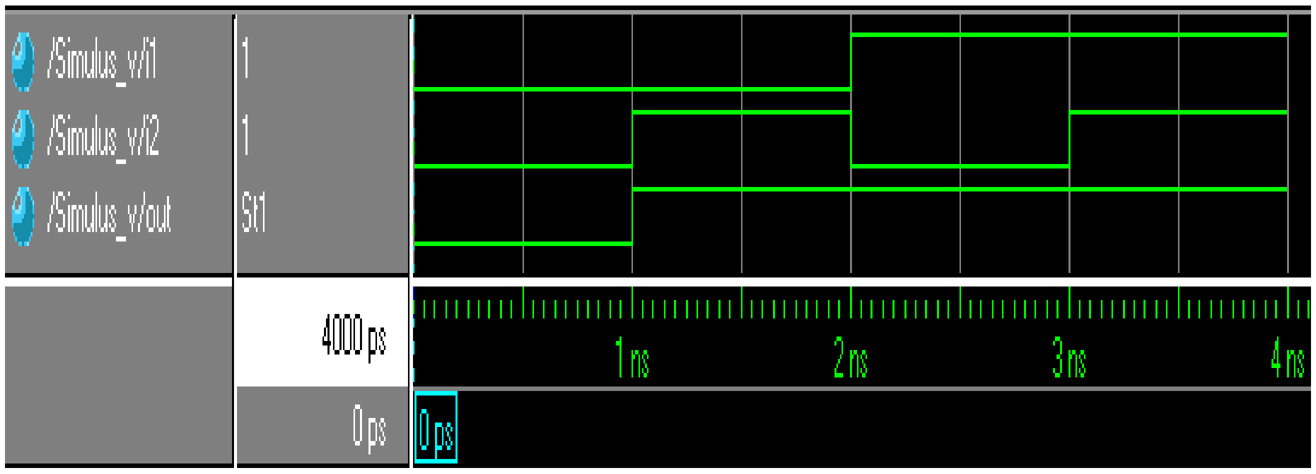
```
moduleOrgate(i1, i2, out);

input i1;

input i2;

output out;

        or(out,i1,i2);

endmodule
```

Truth table:

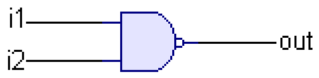
OR Gate

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

Output Wave



NAND Gate:



<u>Program</u>

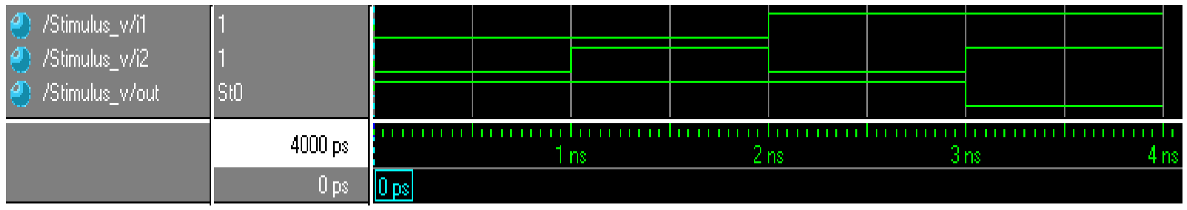
```
moduleNandgate(i1, i2, out);
input i1;
input i2;
output out;
        nand(out,i1,i2);
endmodule
```

Truth table:

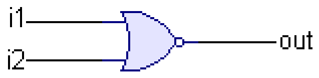
NAND Gate

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

Output Wave:



NOR Gate:



<u>Program</u>

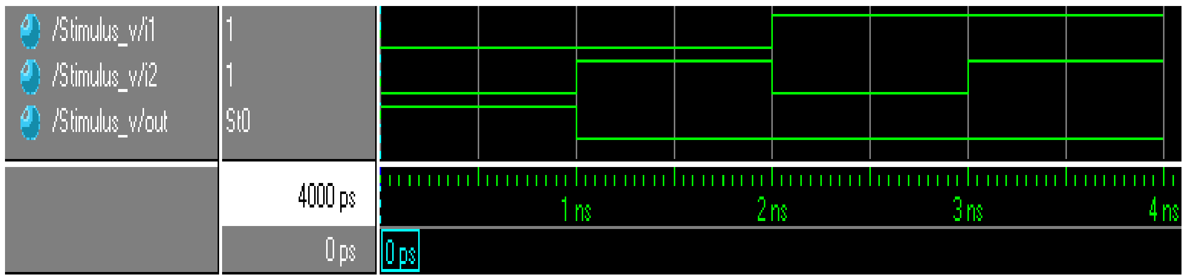
```
moduleNorgate(i1, i2, out);
input i1;
input i2;
output out;
        nor(out,i1,i2);
endmodule
```

Truth table:

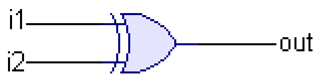
NOR Gate

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

Output wave



XOR Gate:



Program

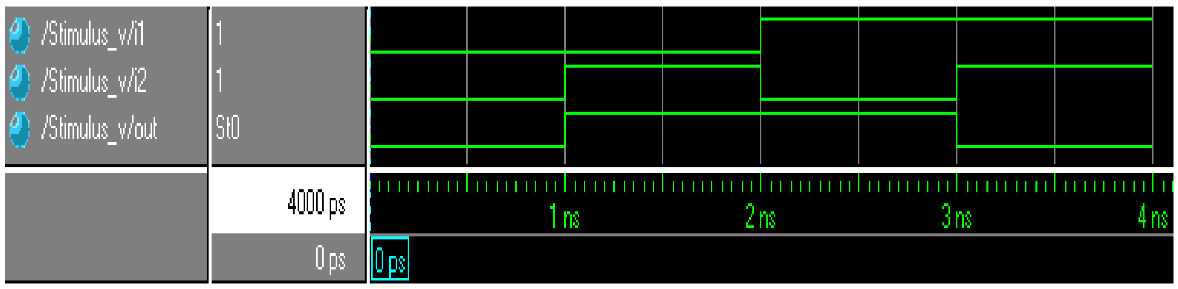
```
moduleXorgate(i1, i2, out);
input i1;
input i2;
output out;
        xor(out,i1,i2);
endmodule
```

Truth table:

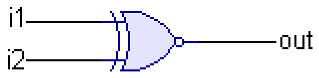
XOR Gate

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

Output Wave



XNOR Gate:



Program

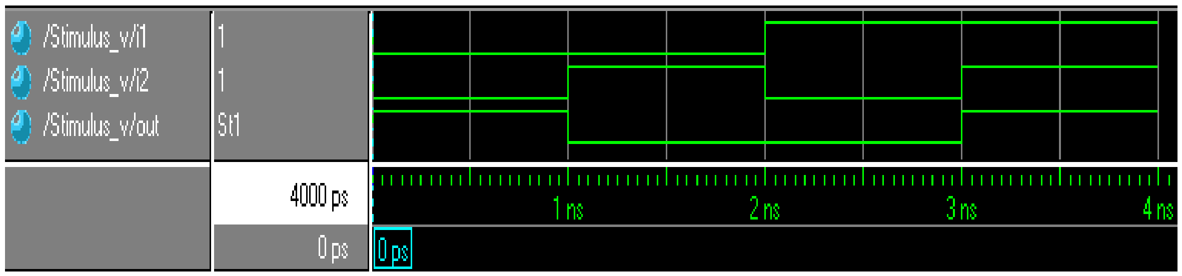
```
moduleXnorgate(i1, i2, out);

input i1;

input i2;

output out;

        xnor(out,i1,i2);

endmodule
```

Truth table:

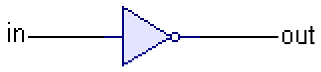
XNOR Gate

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

Output Wave:



Not Gate:



<u>Program</u>

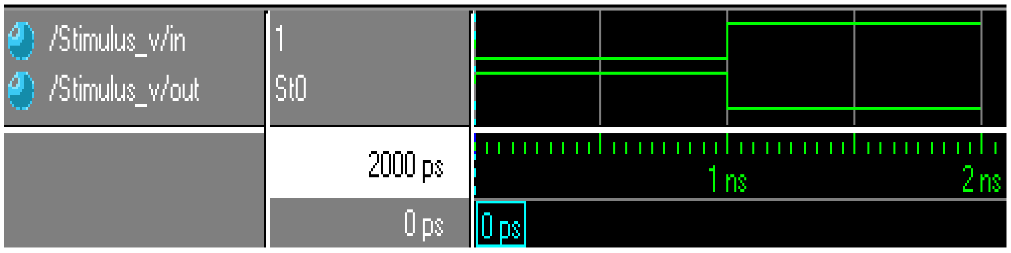
```
moduleNotgate(in, out);

input in;

output out;

        not(out,in);

endmodule
```

Truth table:

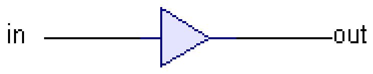
NOT Gate

Input	Output
0	1
1	0

Output Wave



Buffer:



<u>Program</u>

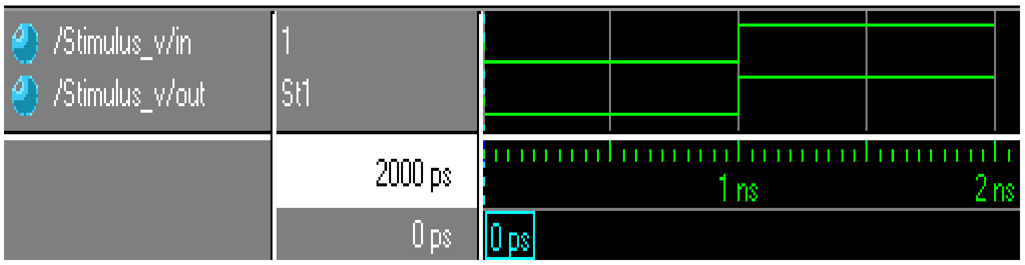
```
module Buffer(in, out);

input in;

output out;

        buf(out,in);

endmodule
```

Truth table :

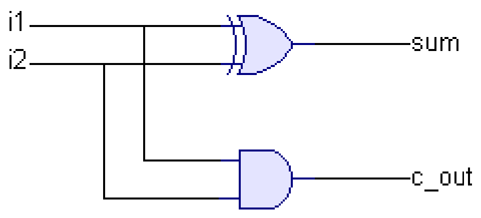
BUFFER

Input	Output
0	0
1	1

Output Wave:



Half Adder:



Program :

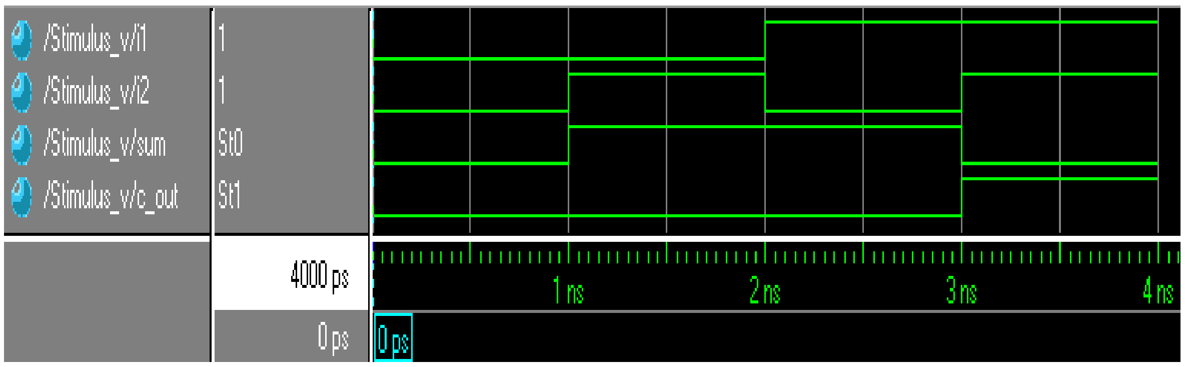
```
moduleHalfAddr(sum, c_out, i1, i2);

output sum;

outputc_out;

input i1;

input i2;

        xor(sum,i1,i2);

        and(c_out,i1,i2);

endmodule
```

Truth table:

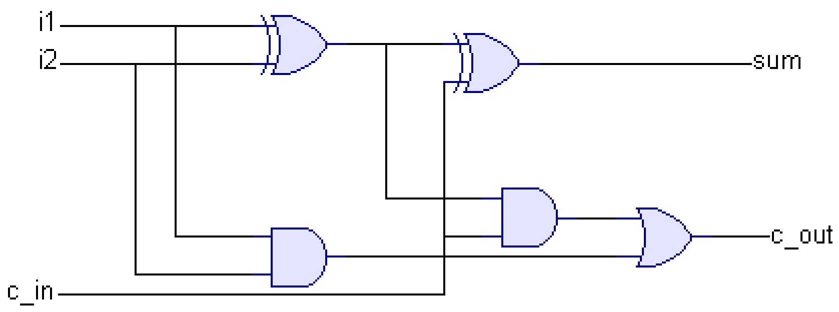
Half Adder

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

Output Wave:



Full Adder:



Program:

moduleFullAddr(i1, i2, c_in, c_out, sum);

input i1;

input i2;

inputc_in;

outputc_out;

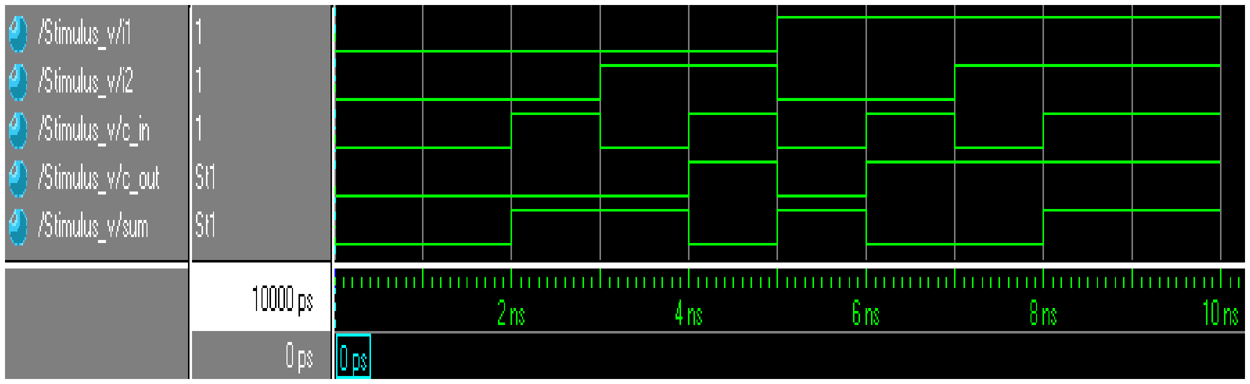
```
output sum;
        wire s1,c1,c2;
        xor n1(s1,i1,i2);
        and n2(c1,i1,i2);
        xor n3(sum,s1,c_in);
        and n4(c2,s1,c_in);
        or n5(c_out,c1,c2);
endmodule
```

Truth Table:

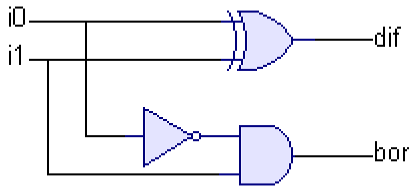
Full Adder

i1	i2	C_in	C_out	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

Output Wave:



Halfsubtractor:



Program:

moduleHalfSub(i0, i1, bor, dif);

input i0;

input i1;

outputbor;

outputdif;

        wire i0n;

        not(i0n,i0);

        xor(dif,i0,i1);

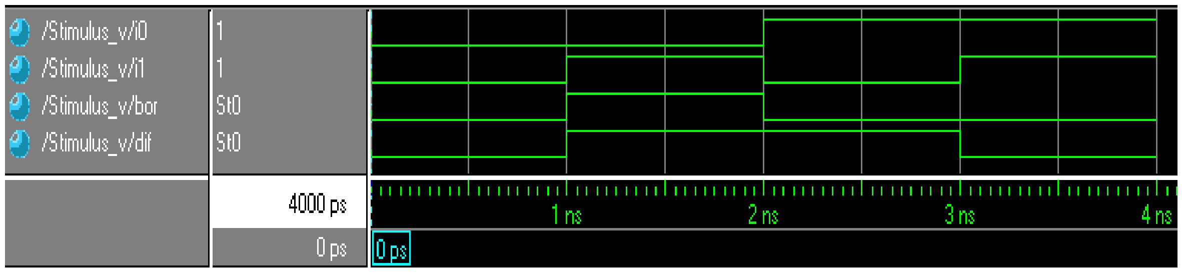
```
        and(bor,i0n,i1);
endmodule
```

Truth Table:

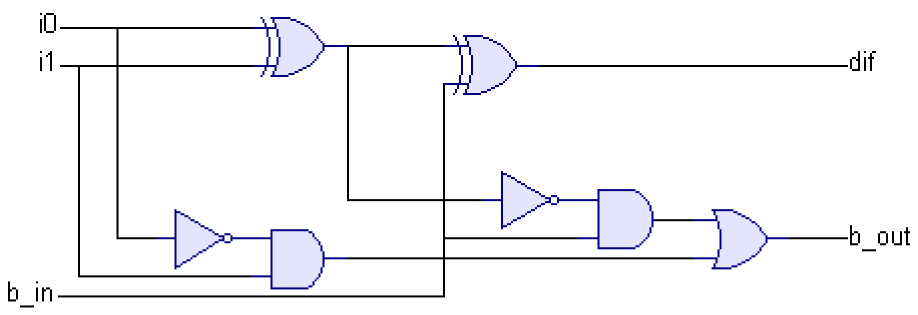
Half Subtractor

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

Output Wave:

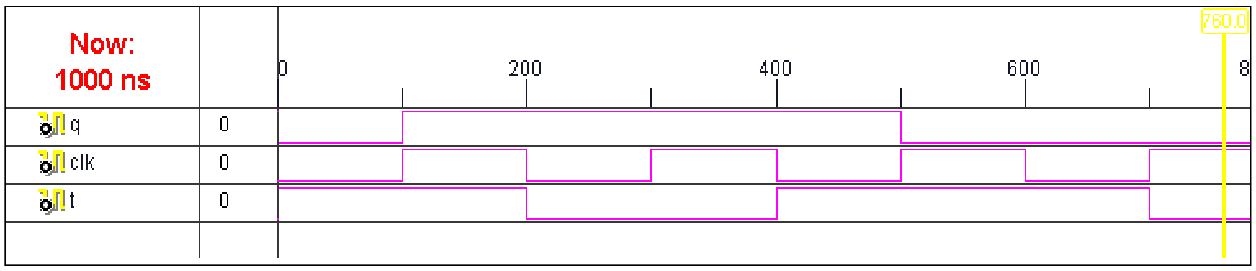


<u>FULL SUBTRACTOR:</u>



Program:

```
moduleFullSub(b_in, i1, i0, b_out, dif);
inputb_in;
input i1;
input i0;
outputb_out;
outputdif;
        assign dif= i0^i1^b_in;
assignb_out= ((~i0&i1)|((~i0|i1)&b_in)))
endmodule
```

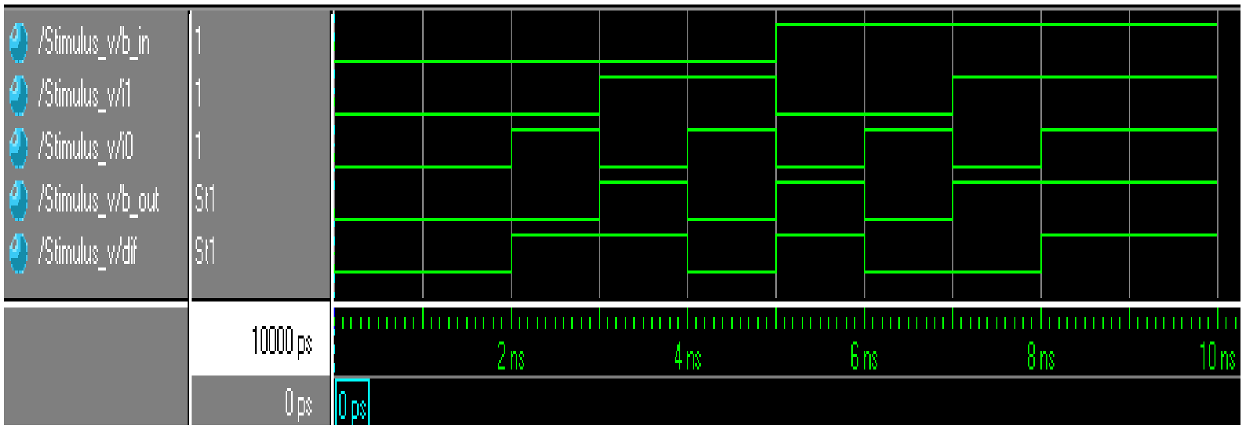


Truth Table:

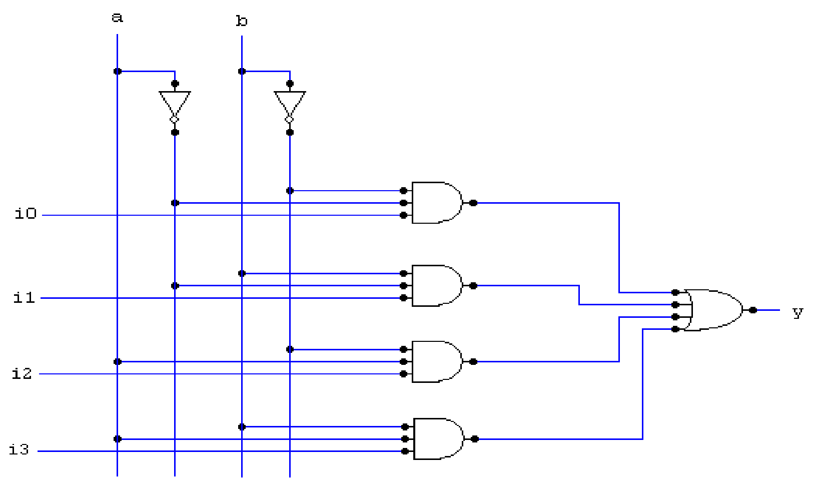
Full Subtractor

A	B	C	Difference	bout
0	0	0	0	0
0	0	1	1	1
0	1	0	1	0
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

Output Wave:



4:1 MUX:



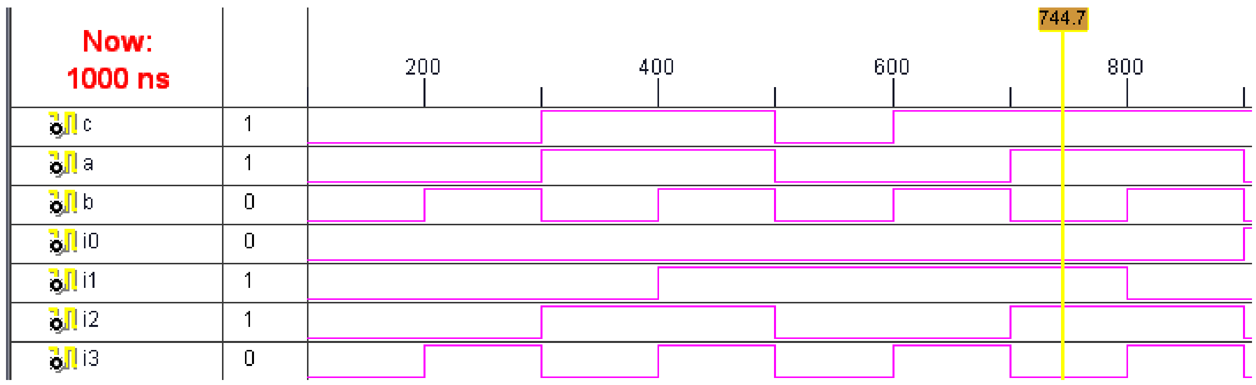
TRUTH TABLE:

INPUT		OUTPUT
S(0)	**S(1)**	**Y**
0	0	X(0)
0	1	X(1)
1	0	X(2)
1	1	X(3)

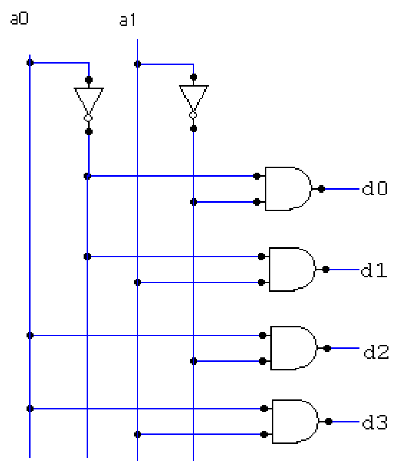
Program for 4:1 Multiplexer:

```
module mux(y,a,b,c,d,s0,s1);

input a,b,c,d,s0,s1;

output y;

reg y;

always @ (a or b or c or d or s0 or s1)

begin

case({s0,s1})

2'b00:y=a;

2'b01:y=b;

2'b10:y=c;

2'b11:y=d;

end case

end

endmoudle
```

OUTPUT:



DECODER:



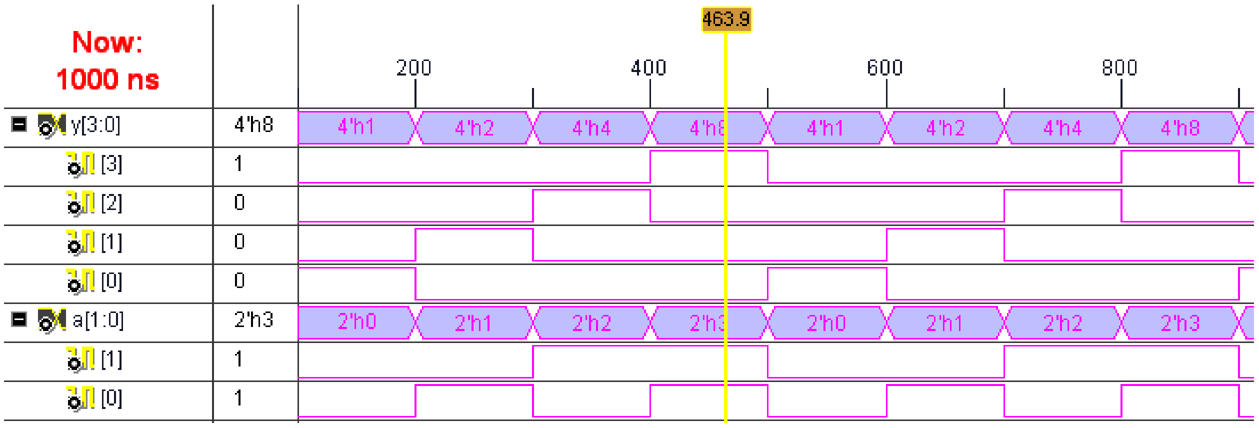
TRUTH TABLE:

INPUTS		OUTPUTS			
a0	**a1**	**d0**	**d1**	**d2**	**d3**
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

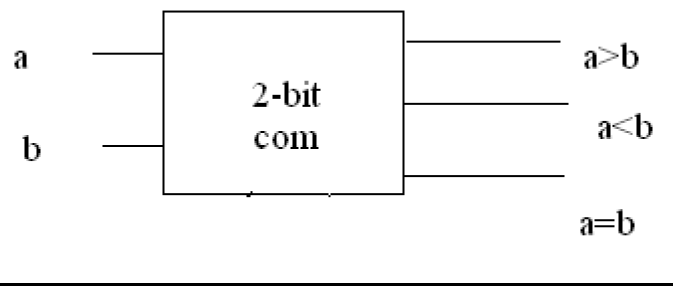
Program for 2 to 4 decoder:

```
moduledec(y0,y1,y2,y3,a,b);

inputa,b;

output y0,y1,y2,y3;

reg y0,y1,y2,y3;

always @ (a or b)

begin

case({a,b})

2’b00:{y0,y1,y2,y3}=4’b1000;

2’b01:{y0,y1,y2,y3}=4’b0100;

2’b10:{y0,y1,y2,y3}=4’b0010;

2’b11:{y0,y1,y2,y3}=4’b0001;

default:$ display (“invalid”);

end case

end

endmodule
```

OUTPUT:



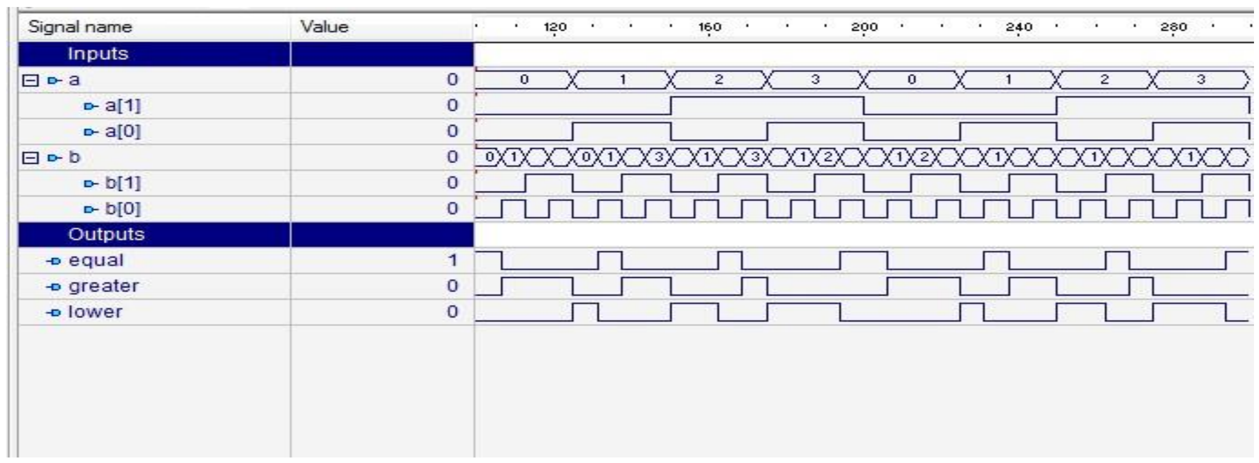
2-bit Comparator:



Program for 2-bit Comparator:

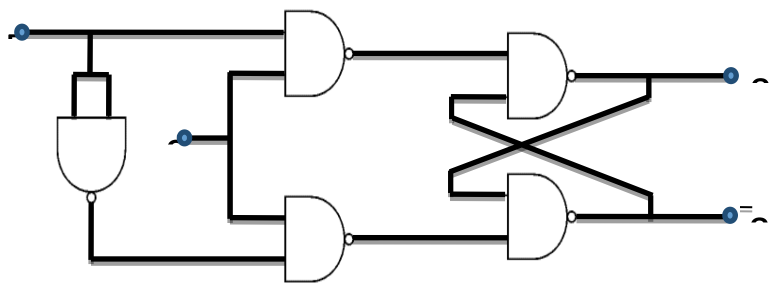
```
module comp(agtb,altb,aeqb,a0,a1,b0,b1);

input a0,a1,b0,b1;

outputagtb,altb,aeqb;

assignagtb=(a0&~b1&~b0)|(~a1&b1)|(a1&a0&~b0);

assignaltb=(~a1&~a0&b0)|(~a0&b1&b0)|(~a1&b1);

assignaeqb=(a0~^b0)&(a1~^b1);

end module
```

OUTPUT:



Sequential Circuits FLIP-FLOPS

LOGIC DIAGRAM of D-FLIP FLOP:



TRUTH TABLE:

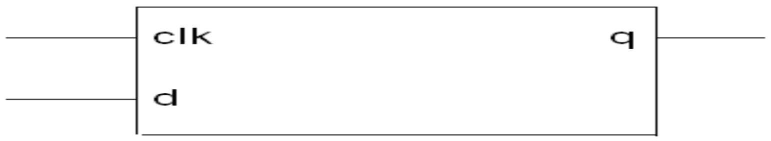
Q(t)	D	Q(t+1)
0	0	0
0	1	1
1	0	0
1	1	1

PROGRAM:

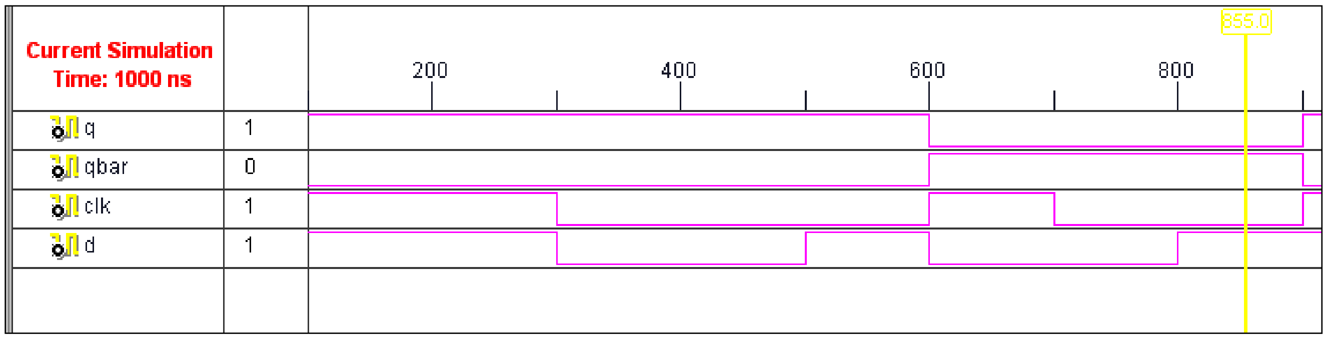
```
moduledff (d, clk, q, qbar);

input d;input clk;

output q;

outputqbar;

            regq,qbar;

            initial q=0;

            always @(posedgeclk)

            begin

             q=d;

            qbar=~d;

            end
```

```
endmodule
```

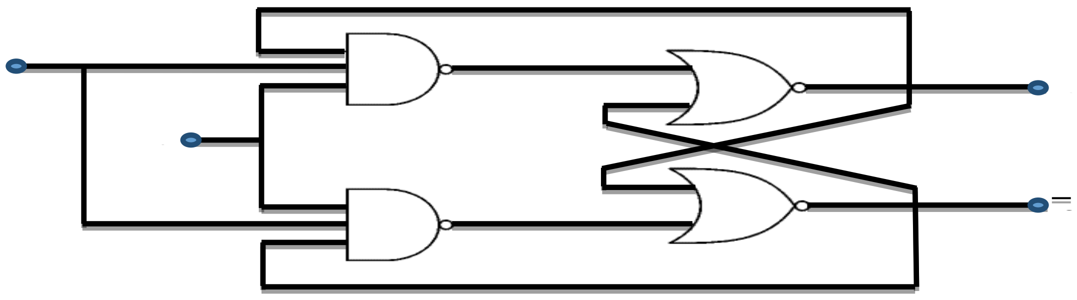
RTL - SCHEMATIC:



SIMULATED OUTPUT:



LOGIC DIAGRAM of T-FLIP FLOP :



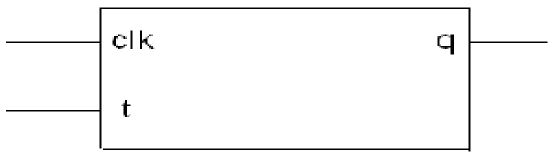
TRUTH TABLE:

Q(t)	T	Q(t+1)
0	0	0
0	1	1
1	0	1
1	1	0

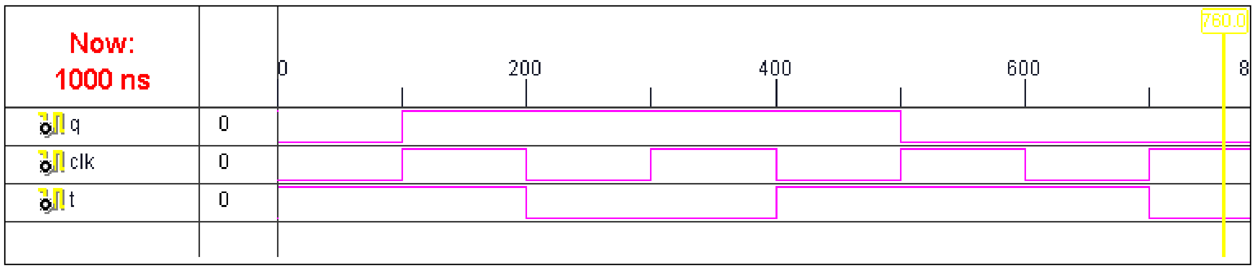
PROGRAM:

```
module TFF(q, clk, t);

output q;

inputclk;

input t;
        reg q;
        initial q=0;
        always@(posedgeclk)
begin
         q=q^t;
        end
endmodule
```

RTL - SCHEMATIC:



SIMULATED OUTPUT:



RESULT:

Thus the program of Basic Gates, half & full adder,half & full Subtractor, Multiplexer, Decoder, Comparator and Flip-flops were designed and simulated using Verilog HDL and also implemented in FPGA kit.

Expt No: 2

Date: **8- BIT ADDER AND MULTIPLIER**

AIM:

To Design and Simulate an 8-Bit Adder and Multiplier using Verilog HDL and also implement in FPGA Kit.

APPARATUS REQUIRED:

- PC,
- XILINX Software,
- Spartan 3E kit.

8- BITADDER

THEORY:

In electronics, an adder or summer is a digital circuit that performs addition of numbers. In modern computers adders reside in the arithmetic logic unit (ALU) where other operations are performed. Although adders can be constructed for many numerical representations, such as Binary-coded decimal or excess-3, the most common adders operate on binary numbers. In cases where two's complement or one's complement is being used to represent negative numbers, it is trivial to modify an adder into an adder-subtractor. Other signed number representations require a more complex adder. Synthesis is the process of constructing a gate level netlist from a register-transfer Level models of the circuit described in Verilog HDL ,VHDL or mixed language designs.Thenetlist files contain both logical design data and constraints the XILINX Synthesis tool enable us to study

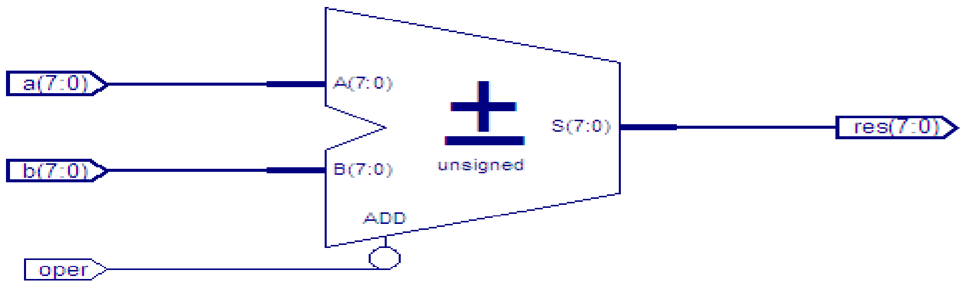
1. Utilization of LUTs & Slices
2. I/O Buffers assignment
3. RTL schematic in gate level
   Time delay between i/Os and path

PROCEDURE:

1. Start the Xilinx ISE by using start→ program file → Xilinx ISE → Project navigator
2. File → New Project
3. Enter the project name and location then click next
4. select the Device and other category and click next twice and finish
5. Click on the symbol of FPGA device and then right click → click on new source
6. Select the Verilog Module and give the file name → click next and define ports → click next and finish

7. Writing the behavioral verilog code in verilog Editor
8. Run the Check syntax → process window → synthesize → double click check syntax and remove errors, if present, with proper syntax & coding.
9. synthesis your design, from the source window select, Synthesis/Implementation from the window Now double click the synthesis →XSt
10. After the HDL synthesis phase of the synthesis process, you can display a schematic representation of your synthesized source file. This schematic shows a representation of the pre-optimized design in terms of generic symbols, such as adders,multipliers,counters,Ann gates and OR gates → double click View RTL schematic.
11. Afterr Synthesis you assign the Pin Value for your design so, → double click the Assign Package Pins
12. Enter the Pin value for your input and output signals. If you want see your Pin assignment in FPGA zoom in Architecture View or Package View
13. You see the Pins in FPGA. Save file as XST Default click ok and close the window
14. Double Click Implementation Design
15. Right click the Generate Programming file→select properties and then select the start-up options→change the clock into JTAG clock, then click apply and ok.
16. Double click the Generate Programming file.
17. Double click Configure Device→click finish →select the bit file and then click ok
18. Right click Xilinx Device→clickProgram→ok.

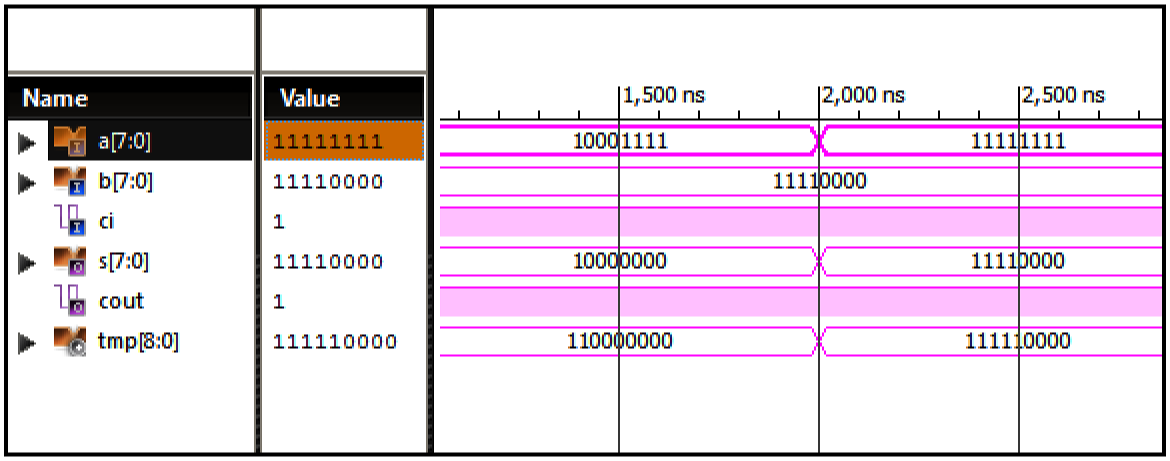
RTL-SCHEMATIC:



PROGRAM:

```
module adder(sum, cout, a, b, cin);
input  cin;
input  [7:0] a;
input  [7:0] b;
output [7:0] sum;
output  cout;
wire   [8:0] tmp;

assigntmp = a + b + cin;
assign sum = tmp [7:0];
assigncout  = tmp [8];
endmodule
```

OUTPUT:



Name	Value
a[7:0]	11111111
b[7:0]	11110000
ci	1
s[7:0]	11110000
cout	1
tmp[8:0]	111110000

8- BIT MULTIPLIER

THEORY:

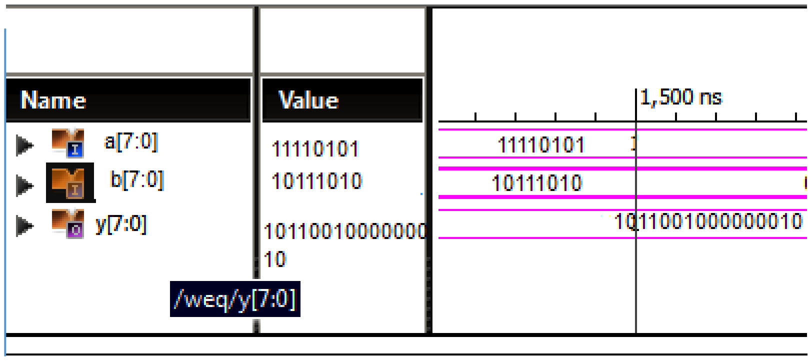
A combinational multiplier is a good example of how simple logic functions (gates, half adders and full adders) can be combined to construct a much more complex function. In particular, it is possible to construct a 4x4 combinational multiplier from an array of AND gates, half-adders and full-adders, taking what you have learned recently and extending it to a more complex circuit. The purpose of this document is to introduce how a relatively complex arithmetic function, such as binary multiplication, can be realized using simple logic building blocks.Synthesis is the process of constructing a gate level netlist from a register-transfer Level models of the circuit described in Verilog HDL ,VHDL or mixed language designs.Thenetlist files contain both logical design data and constraints, XILINX Synthesis tool enable us to study,

1. Utilization of LUTs & Slices
2. I/O Buffers assignment
3. RTL schematic in gate level
4. Time delay between i/Os and path

PROGRAM:

```
  modulemulti(y,a,b);
input [7:0]a;
input [7:0]b;
output [15:0]y;
assign y=a*b;
endmodule
```

OUTPUT:



```
a[7:0]  11110101
b[7:0]  10111010
y[7:0]  1011001000000010
```

RESULT:

Thus the Program of 8-Bit Adder and Multiplier were designed and simulated using Verilog HDL and also implemented in FPGA Kit.

Expt No: 3

Date: **UNIVERSAL SHIFT REGISTER**

AIM:

To Design and Simulate aUniversal Shift Registerusing Verilog HDL and also implement in

FPGA Kit.

APPARATUS REQUIRED:

- PC,
- XILINX Software,
- Spartan 3E kit.

THEORY

A Universal shift register is a register which has both the right shift and left shift with parallel load capabilities. Universal shift registers are used as memory elements in computers. A Unidirectional shift register is capable of shifting in only one direction. A bidirectional shift register is capable of shifting in both the directions. The Universal shift register is a combination design of bidirectional shift register and a unidirectional shift register with parallel load provision.A n-bit universal shift register consists of n flip-flops and n 4×1 multiplexers. All the n multiplexers share the same select lines(S1 and S0)to select the mode in which the shift register operates. The select inputs select the suitable input for the flip-flops.

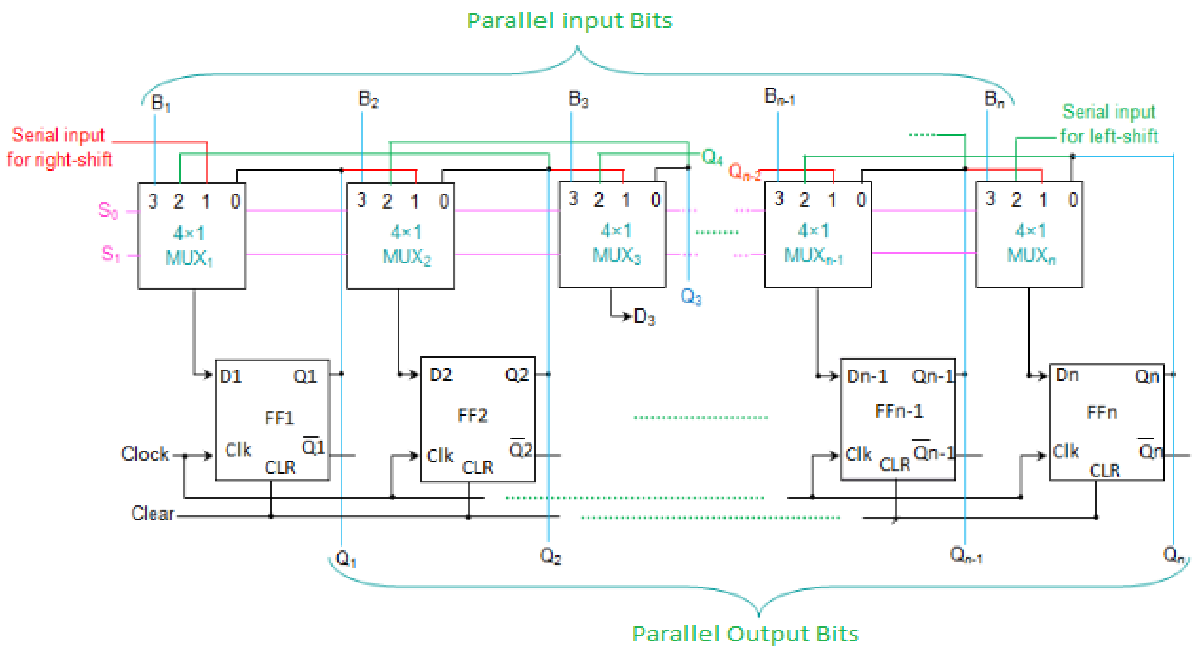
Basic connections

1. The first input (zeroth pin of multiplexer) is connected to the output pin of the corresponding flip-flop.
2. The second input (first pin of multiplexer) is connected to the output of the very-previous flip flop which facilitates the right shift
3. The third input (second pin of multiplexer) is connected to the output of the very-next flip-flop which facilitates the left shift.
4. The fourth input (third pin of multiplexer) is connected to the individual bits of the input data which facilitates parallel loading.

The working of the Universal shift register depends on the inputs given to the select lines.

The register operations performed for the various inputs of select lines are as follows:

S1	S0	REGISTER OPERATION
0	0	No changes
0	1	Shift right
1	0	Shift left
1	1	Parallel load



PROCEDURE:

1. Start the Xilinx ISE by using start→ program file → Xilinx ISE → Project navigator
2. File → New Project
3. Enter the project name and location then click next
4. select the Device and other category and click next twice and finish
5. Click on the symbol of FPGA device and then right click → click on new source

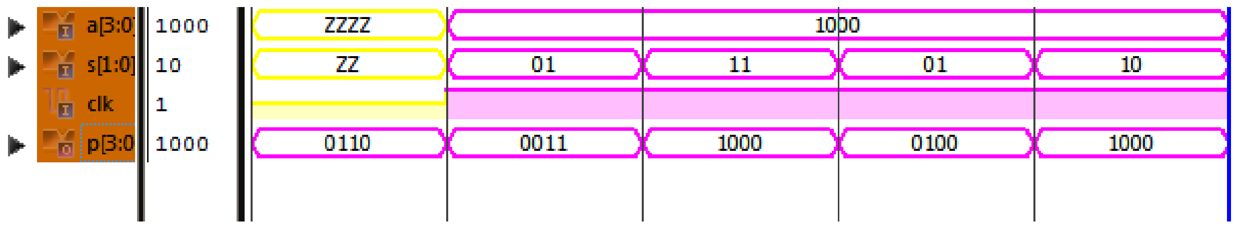
6. Select the Verilog Module and give the file name → click next and define ports → click next and finish
7. Writing the behavioral verilog code in verilog Editor
8. Run the Check syntax → process window → synthesize → double click check syntax and remove errors, if present, with proper syntax & coding.
9. synthesis your design, from the source window select, Synthesis/Implementation from the window Now double click the synthesis →XSt
10. After the HDL synthesis phase of the synthesis process, you can display a schematic representation of your synthesized source file. This schematic shows a representation of the pre-optimized design in terms of generic symbols, such as adders, multipliers, counters, Ann gates and OR gates → double click View RTL schematic.
11. Afterr Synthesis you assign the Pin Value for your design so, → double click the Assign Package Pins
12. Enter the Pin value for your input and output signals. If you want see your Pin assignment in FPGA zoom in Architecture View or Package View
13. You see the Pins in FPGA. Save file as XST Default click ok and close the window
14. Double Click Implementation Design
15. Right click the Generate Programming file→select properties and then select the start-up options→change the clock into JTAG clock, then click apply and ok.
16. Double click the Generate Programming file.
17. Double click Configure Device→click finish →select the bit file and then click ok
18. Right click Xilinx Device→clickProgram→ok.

PROGRAM:

```
moduleefffff(a,s,clk,p);
input [3:0]a;
input [1:0]s;
inputclk;
outputreg [3:0]p;
initial
p<=4'b0110;
always@(posedgeclk)
```

```
begin
case (s)
2'b00:
begin
p[3]<=p[3];
p[2]<=p[2];
p[1]<=p[1];
p[0]<=p[0];
end
2'b01:
begin
p[3]<=p[0];
p[2]<=p[3];
p[1]<=p[2];
p[0]<=p[1];
end
2'b10:
begin
p[0]<=p[3];
p[1]<=p[0];
p[2]<=p[1];
p[3]<=p[2];
end
2'b11:
begin
p[0]<=a[0];
p[1]<=a[1];
p[2]<=a[2];
p[3]<=a[3];
end
endcase
end
endmodule
```

OUTPUT:



RESULT:

Thus the Program of 4-Bit Universal Shift Registerwas designed and simulated using Verilog HDL and also implemented in FPGA Kit.

Expt No: 4

Date: **RANDOM ACCESS MEMORY**

AIM:

To Design and Simulate a Random Access Memoryusing Verilog HDL and also implement in FPGA Kit.

APPARATUS REQUIRED:

- PC,
- XILINX Software,
- Spartan 3E kit.

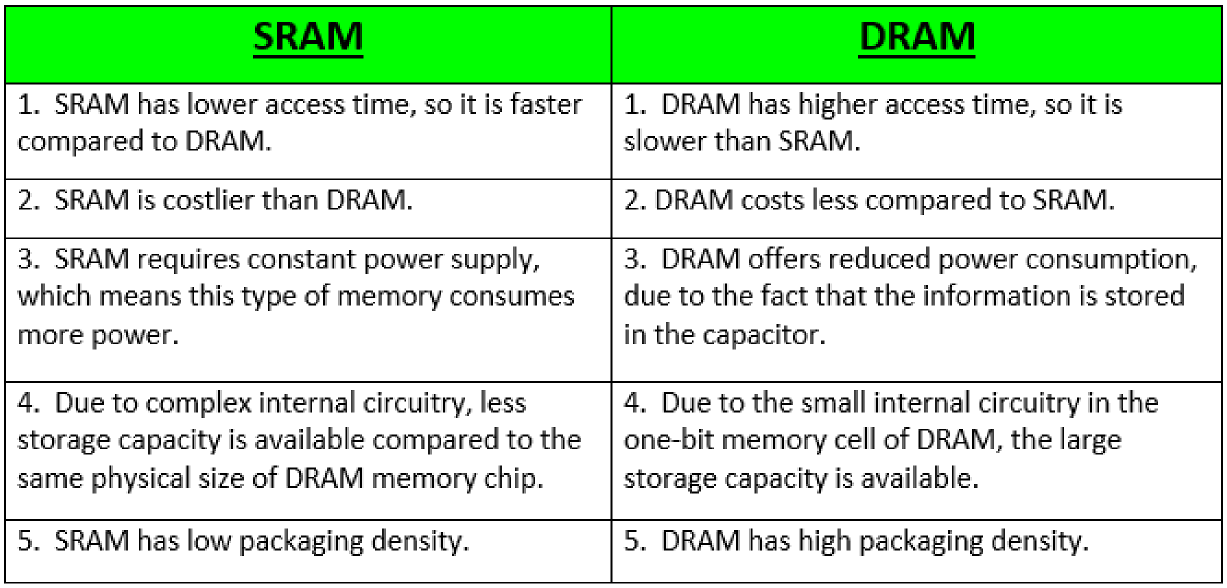
THEORY:

RAM (pronounced ramm) is an acronym for **r***andom* **a***ccess* **m***emory,* a type of computer memory that can be accessed randomly; that is, any byte of memory can be accessed without touching the preceding bytes. RAM is found in servers, PCs, tablets, smartphones and other devices, such as printers.There are two main types of RAM:

DRAM (Dynamic Random Access Memory)
SRAM (Static Random Access Memory)

DRAM (Dynamic Random Access Memory) – The term dynamic indicates that the memory must be constantly refreshed or it will lose its contents. DRAM is typically used for the main memory in computing devices. If a PC or smartphone is advertised as having 4-GB RAM or 16-GB RAM, those numbers refer to the DRAM, or main memory, in the device.More specifically, most of the DRAM used in modern systems is synchronous DRAM, or SDRAM. Manufacturers also sometimes use the acronym DDR (or DDR2, DDR3, DDR4, etc.) to describe the type of SDRAM used by a PC or server. DDR stands for double data rate, and it refers to how much data the memory can transfer in one clock cycle.In general, the more RAM a device has, the faster it will perform.

SRAM (Static Random Access Memory) – While DRAM is typically used for main memory, today SRAM is more often used for system cache. SRAM is said to be static because it doesn't need to be refreshed, unlike dynamic RAM, which needs to be refreshed thousands of times per second. As a result, SRAM is faster than DRAM. However, both types of RAM are volatile, meaning that they lose their contents when the power is turned off.



SRAM	**DRAM**
1. SRAM has lower access time, so it is faster compared to DRAM.	1. DRAM has higher access time, so it is slower than SRAM.
2. SRAM is costlier than DRAM.	2. DRAM costs less compared to SRAM.
3. SRAM requires constant power supply, which means this type of memory consumes more power.	3. DRAM offers reduced power consumption, due to the fact that the information is stored in the capacitor.
4. Due to complex internal circuitry, less storage capacity is available compared to the same physical size of DRAM memory chip.	4. Due to the small internal circuitry in the one-bit memory cell of DRAM, the large storage capacity is available.
5. SRAM has low packaging density.	5. DRAM has high packaging density.

The Difference between Memory, RAM and Storage:

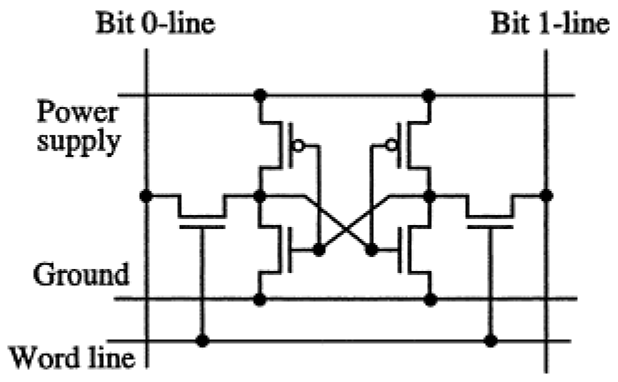
The difference between memory and storage, in part because both can be measured in megabytes (MB), gigabytes (GB) and terabytes (TB).In common usage, the term RAM is synonymous with main memory. This is where a computing system stores data that it is actively using. Storage systems, such as hard drives, network storage devices or cloud storage, are where a system saves data that it will need to access later.Computing systems can retrieve data from RAM very quickly, but when a device powers down, all the data that was in memory goes away. Many people have had the experience of losing a document they were working on after an unexpected power outage or system crash. In these cases, the data was lost because it was stored in system memory, which is volatile.By contrast, storage is slower, but it can retain data when the device is powered down. So, for example, if a document has been saved to a hard drive prior to a power outage or system crash, the user will still be able to retrieve it when the system is back up and running.Storage is usually less expensive than RAM on a per-gigabyte basis. As a result, most PCs and smartphones have many times more gigabytes of storage than gigabytes of RAM.

PROCEDURE:

1. Start the Xilinx ISE by using start→ program file → Xilinx ISE → Project navigator
2. File → New Project
3. Enter the project name and location then click next
4. select the Device and other category and click next twice and finish

5. Click on the symbol of FPGA device and then right click → click on new source
6. Select the Verilog Module and give the file name → click next and define ports → click next and finish
7. Writing the behavioral verilog code in verilog Editor
8. Run the Check syntax → process window → synthesize → double click check syntax and remove errors, if present, with proper syntax & coding.
9. synthesis your design, from the source window select, Synthesis/Implementation from the window Now double click the synthesis →XSt
10. After the HDL synthesis phase of the synthesis process, you can display a schematic representation of your synthesized source file. This schematic shows a representation of the pre-optimized design in terms of generic symbols, such as adders, multipliers, counters, Ann gates and OR gates → double click View RTL schematic.
11. Afterr Synthesis you assign the Pin Value for your design so, → double click the Assign Package Pins
12. Enter the Pin value for your input and output signals. If you want see your Pin assignment in FPGA zoom in Architecture View or Package View
13. You see the Pins in FPGA. Save file as XST Default click ok and close the window
14. Double Click Implementation Design
15. Right click the Generate Programming file→select properties and then select the start-up options→change the clock into JTAG clock, then click apply and ok.
16. Double click the Generate Programming file.
17. Double click Configure Device→click finish →select the bit file and then click ok
18. Right click Xilinx Device→clickProgram→ok.

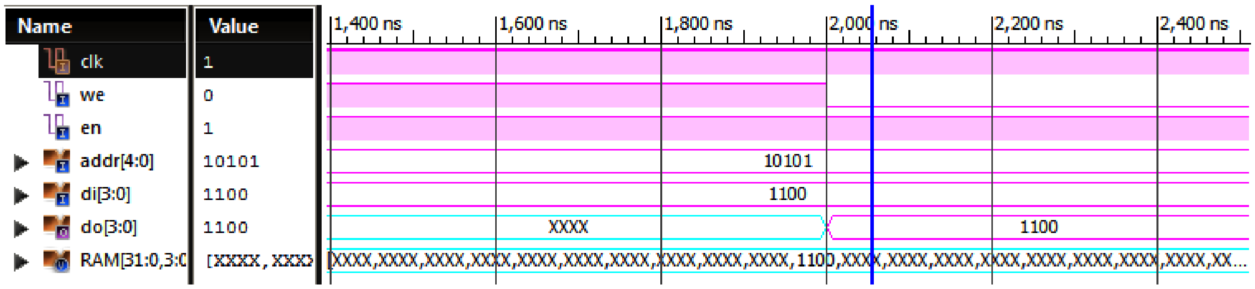
CIRCUIT DIAGRAM:



PROGRAM:

```
moduleramreadfirst(clk, en, we, addr, di, do);
inputclk;
input       we;
input       en;
input  [4:0] addr;
input  [3:0] di;
output [3:0] do;
reg    [3:0] RAM [31:0];
reg    [3:0] do;
always @(posedgeclk)
begin
if (en)
begin
if (we)
RAM[addr] <= di;
do<= RAM[addr];
end
end
endmodule
```

OUTPUT:



RESULT:

Thus the Program of Random Access Memorywas designed and simulated using Verilog HDL and also implemented in FPGA Kit.

Expt No: 5

Date: **FINITE STATE MACHINE**

AIM:

To Design and Simulate a Finite State Machineusing Verilog HDL and also implement in

FPGA Kit.

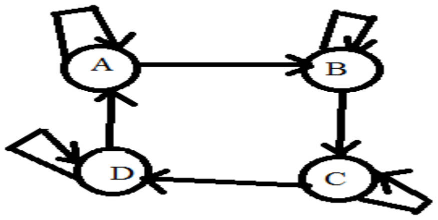
APPARATUS REQUIRED:

- PC,
- XILINX Software,
- Spartan 3E kit.

THEORY:

Finite State Machines (FSM) are sequential circuit used in many digital systems to control the behavior of systems and dataflow paths. Examples of FSM include control units and sequencers. This lab introduces the concept of two types of FSMs, Mealy and Moore, and the modeling styles to develop such machines.

A finite-state machine (FSM) or simply a state machine is used to design both computer programs and sequential logic circuits. It is conceived as an abstract machine that can be in one of a finite number of user-defined states. The machine is in only one state at a time; the state it is in at any given time is called the current state. It can change from one state to another when initiated by a triggering event or condition; this is called a transition. A particular FSM is defined by a list of its states, and the triggering condition for each transition. The behavior of state machines can be observed in many devices in modern society performing a predetermined sequence of actions depending on a sequence of events with which they are presented. Simple examples are vending machines which dispense products when the proper combination of coins are deposited, elevators which drop riders off at upper floors before going down, traffic lights which change sequence when cars are waiting, and combination locks which require the input of combination numbers in the proper order. The state machines are modeled using two basic types of sequential networks- Mealy and Moore. In a Mealy machine, the output depends on both the present (current) state and the present (current) inputs. In Moore machine, the output depends only on the present state. A general model of a Mealy sequential machine consists of a combinatorial network, which generates the outputs and the next state, and a state register which holds the present state as shown below. The state register is normally modeled as D flip-flops. The state register must be sensitive to a clock edge. The other block(s) can be modeled either using the always procedural block or a mixture of the always procedural block and dataflow modeling statements; the always procedural block will have to be sensitive to all inputs being read into the block and must have all output defined for every branch in order to model it as a combinatorial block. The two blocks Mealy machine can be viewed as,



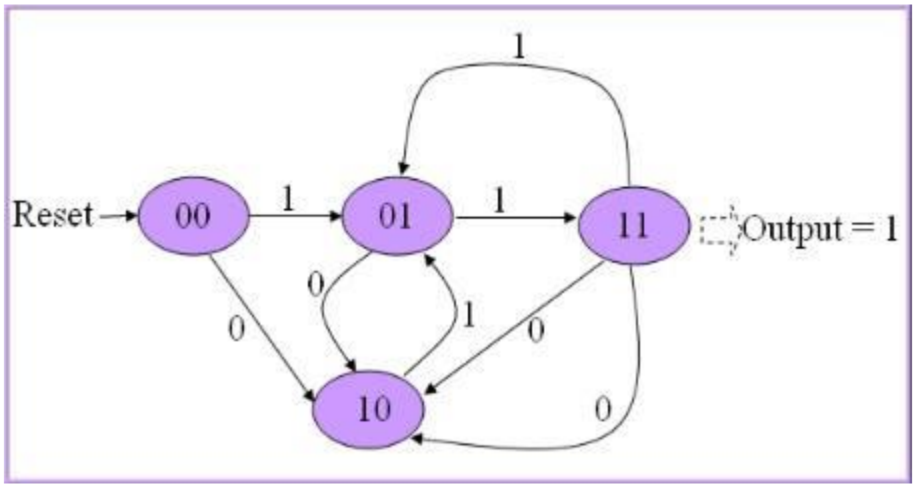
PROCEDURE:

1. Start the Xilinx ISE by using start→ program file → Xilinx ISE → Project navigator
2. File → New Project
3. Enter the project name and location then click next
4. select the Device and other category and click next twice and finish
5. Click on the symbol of FPGA device and then right click → click on new source
6. Select the Verilog Module and give the file name → click next and define ports → click next and finish
7. Writing the behavioral verilog code in verilog Editor
8. Run the Check syntax → process window → synthesize → double click check syntax and remove errors, if present, with proper syntax & coding.
9. synthesis your design, from the source window select, Synthesis/Implementation from the window Now double click the synthesis →XSt
10. After the HDL synthesis phase of the synthesis process, you can display a schematic representation of your synthesized source file. This schematic shows a representation of the pre-optimized design in terms of generic symbols, such as adders, multipliers, counters, Ann gates and OR gates → double click View RTL schematic.
11. Afterr Synthesis you assign the Pin Value for your design so, → double click the Assign Package Pins
12. Enter the Pin value for your input and output signals. If you want see your Pin assignment in FPGA zoom in Architecture View or Package View
13. You see the Pins in FPGA. Save file as XST Default click ok and close the window
14. Double Click Implementation Design
15. Right click the Generate Programming file→select properties and then select the start-up options→change the clock into JTAG clock, then click apply and ok.

16. Double click the Generate Programming file.

17. Double click Configure Device→click finish →select the bit file and then click ok

18. Right click Xilinx Device→clickProgram→ok.

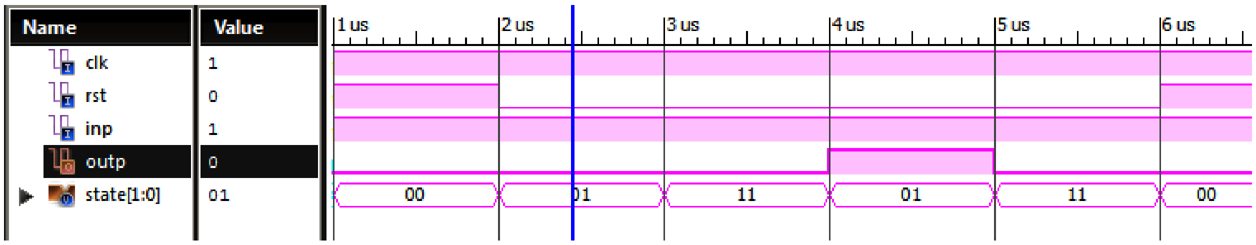


PROGRAM:

```
modulefsm( clk, rst, x, y);
inputclk, rst, x;
output y;
reg [1:0] state;
reg y;
always @( posedgeclk, posedgerst )
begin
if(rst )
state<= 2'b00;
else
begin
case( state )
      2'b00:
begin
if( x ) state <= 2'b01;
else state <= 2'b10;
end
      2'b01:
begin
if( x ) state <= 2'b11;
else state <= 2'b10;
end
      2'b10:
begin
if( x ) state <= 2'b01;
```

```
else state <= 2'b11;
end
      2'b11:
begin
if( x ) state <= 2'b01;
else state <= 2'b10;
end
endcase
end
end
always @(posedgeclk, posedgerst)
begin
if(rst )
y <= 0;
else if( state == 2'b11 )
y <= 1;
else y <= 0;
end
endmodule
```

FINITE STATE MACHINEOUTPUT:



RESULT:

Thus the Program of Finite State Machinewas designed and simulated using Verilog HDL and also implemented in FPGA Kit.

Expt No: 6 **3-BIT SYNCHRONOUS UP/DOWN COUNTER**

Date:

AIM:

To design and simulate 3-Bit Synchronous Up/Down Counter Using HDLand also implement in FPGA Kit.

APPARATUS REQUIRED:

- PC
- Xilinx software
- FPGA(Spartan 3E) Kit

PROCEDURE:

1. Start the Xilinx ISE by using start→ program file → Xilinx ISE → Project navigator
2. File → New Project
3. Enter the project name and location then click next
4. select the Device and other category and click next twice and finish
5. Click on the symbol of FPGA device and then right click → click on new source
6. Select the Verilog Module and give the file name → click next and define ports → click next and finish
7. Writing the behavioral verilog code in verilog Editor
8. Run the Check syntax → process window → synthesize → double click check syntax and remove errors, if present, with proper syntax & coding.
9. synthesis your design, from the source window select, Synthesis/Implementation from the window Now double click the synthesis →XSt
10. After the HDL synthesis phase of the synthesis process, you can display a schematic representation of your synthesized source file. This schematic shows a representation of the pre-optimized design in terms of generic symbols, such as adders, multipliers, counters, Ann gates and OR gates → double click View RTL schematic.
11. Afterr Synthesis you assign the Pin Value for your design so, → double click the Assign Package Pins
12. Enter the Pin value for your input and output signals. If you want see your Pin assignment in FPGA zoom in Architecture View or Package View

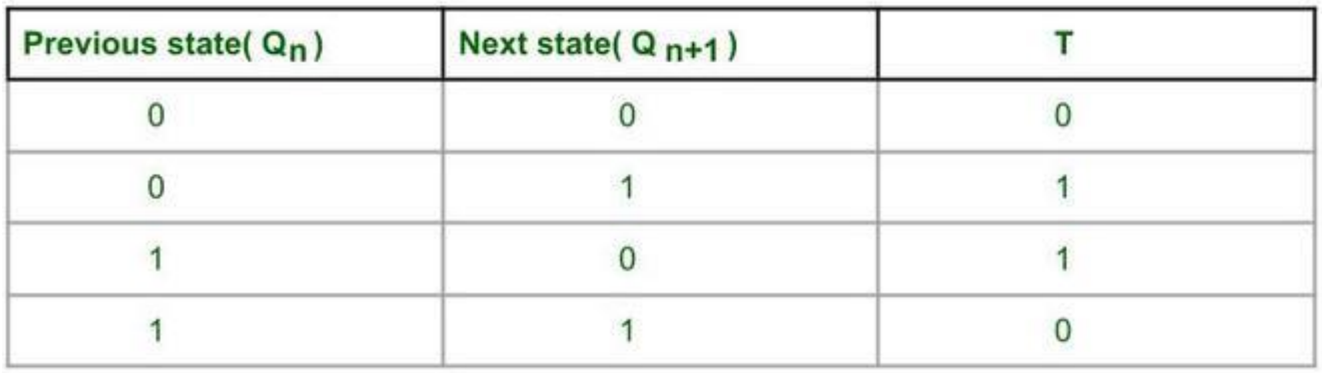
13. You see the Pins in FPGA. Save file as XST Default click ok and close the window
14. Double Click Implementation Design
15. Right click the Generate Programming file→select properties and then select the start-up options→change the clock into JTAG clock, then click apply and ok.
16. Double click the Generate Programming file.
17. Double click Configure Device→click finish →select the bit file and then click ok
18. Right click Xilinx Device→clickProgram→ok.

Steps to design Synchronous 3 bit Up/Down Counter :

1. Decide the number and type of FF –

- Here we are performing 3 bit or mod-8 **Up or Down counting,** so 3 Flip Flops are required, which can count up to $2^3-1 = 7$.
- Here T Flip Flop is used.

2. **Write excitation table of Flip Flop –**



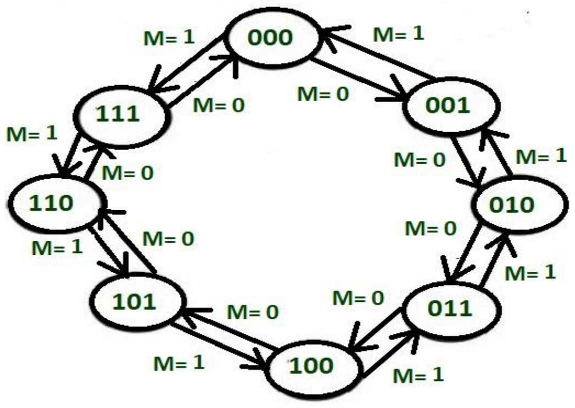
Previous state(Q_n)	Next state(Q_{n+1})	T
0	0	0
0	1	1
1	0	1
1	1	0

Excitation table of T FF

3. Decision for Mode control input M –

- When M=0 ,then the counter will perform up counting.
- When M=1 ,then the counter will perform down counting.

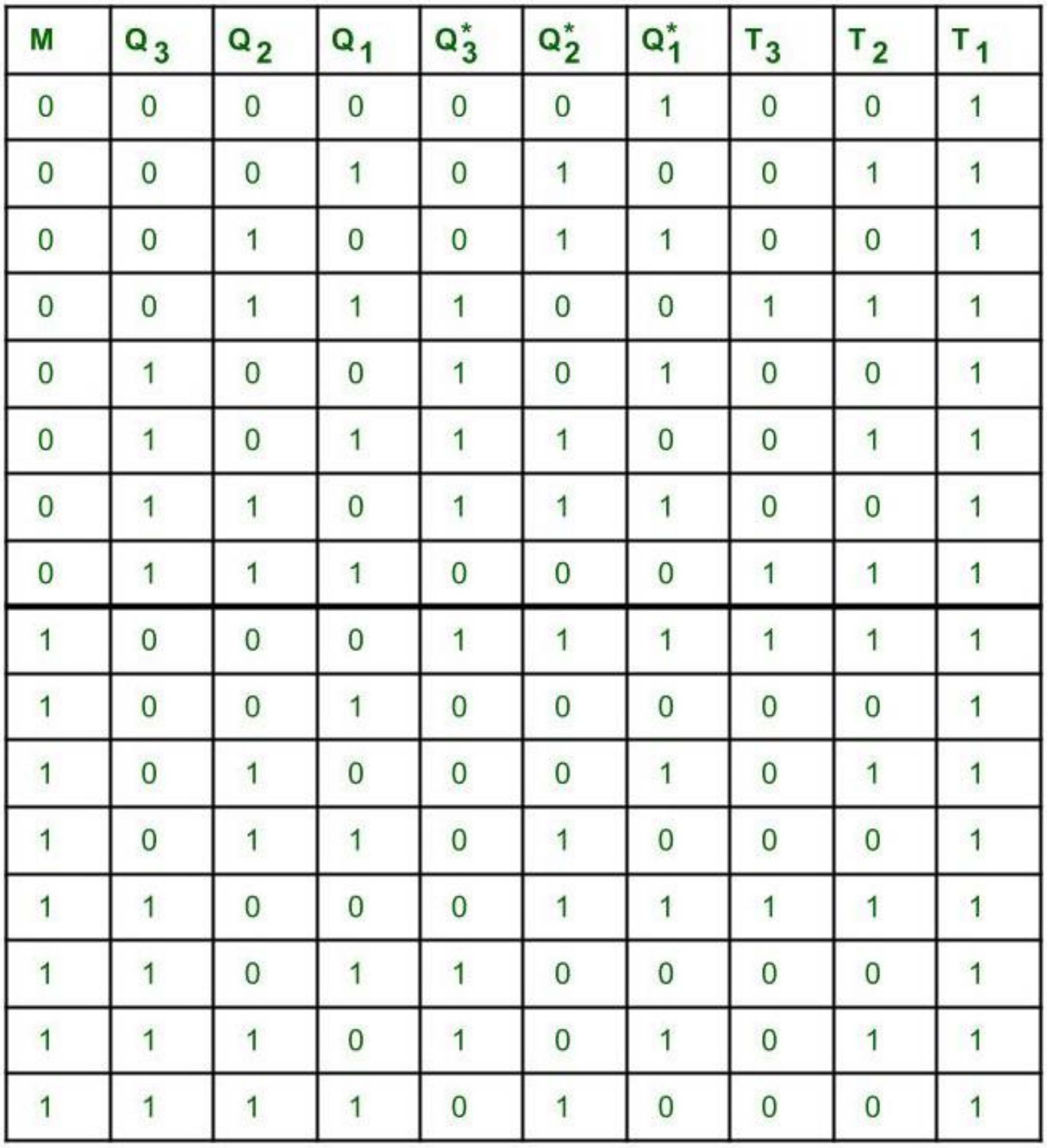
4. Draw the state transition diagram and circuit excitation table –



State transition diagram for 3 bit up/down counting.

5. Circuit excitation table –

The circuit excitation table represents the present states of the counting sequence and the next states after the clock pulse is applied and input T of the flip-flops. By seeing the transition between the present state and the next state, we can find the input values of 3 Flip Flops using the Flip Flops excitation table. The table is designed according to the required counting sequence.

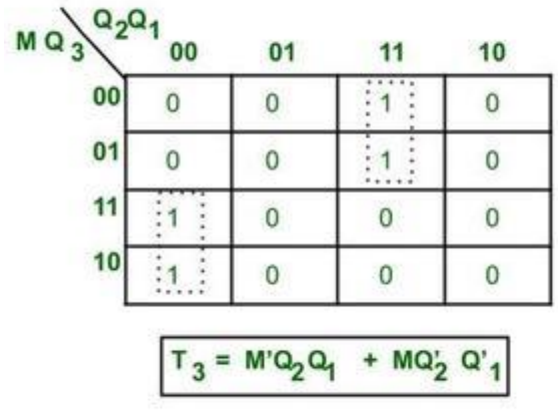


M	Q_3	Q_2	Q_1	Q_3^*	Q_2^*	Q_1^*	T_3	T_2	T_1
0	0	0	0	0	0	1	0	0	1
0	0	0	1	0	1	0	0	1	1
0	0	1	0	0	1	1	0	0	1
0	0	1	1	1	0	0	1	1	1
0	1	0	0	1	0	1	0	0	1
0	1	0	1	1	1	0	0	1	1
0	1	1	0	1	1	1	0	0	1
0	1	1	1	0	0	0	1	1	1
1	0	0	0	1	1	1	1	1	1
1	0	0	1	0	0	0	0	0	1
1	0	1	0	0	0	1	0	1	1
1	0	1	1	0	1	0	0	0	1
1	1	0	0	0	1	1	1	1	1
1	1	0	1	1	0	0	0	0	1
1	1	1	0	1	0	1	0	1	1
1	1	1	1	1	1	0	0	0	1

Circuit excitation table

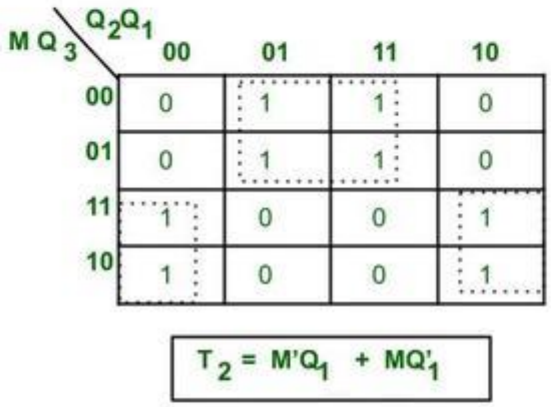
If there is a change in the output state of a flip flop (i.e. 0 to 1 or 1 to 0), then the corresponding T value becomes 1 otherwise 0.

6. Find a simplified equation using k map –
Here we are finding the minimal Boolean expression for each Flip Flop input T using k map.



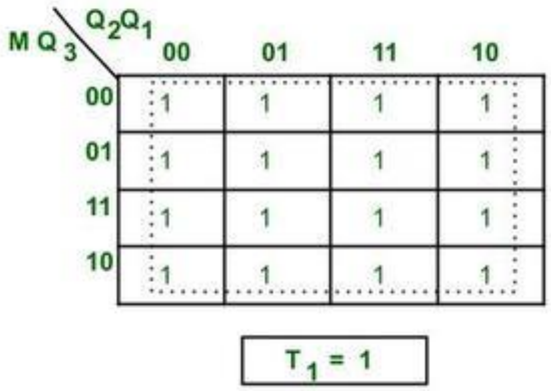
$MQ_3 \backslash Q_2Q_1$	00	01	11	10
00	0	0	1	0
01	0	0	1	0
11	1	0	0	0
10	1	0	0	0

$$T_3 = M'Q_2Q_1 + MQ'_2\,Q'_1$$



$MQ_3 \backslash Q_2Q_1$	00	01	11	10
00	0	1	1	0
01	0	1	1	0
11	1	0	0	1
10	1	0	0	1

$$T_2 = M'Q_1 + MQ'_1$$

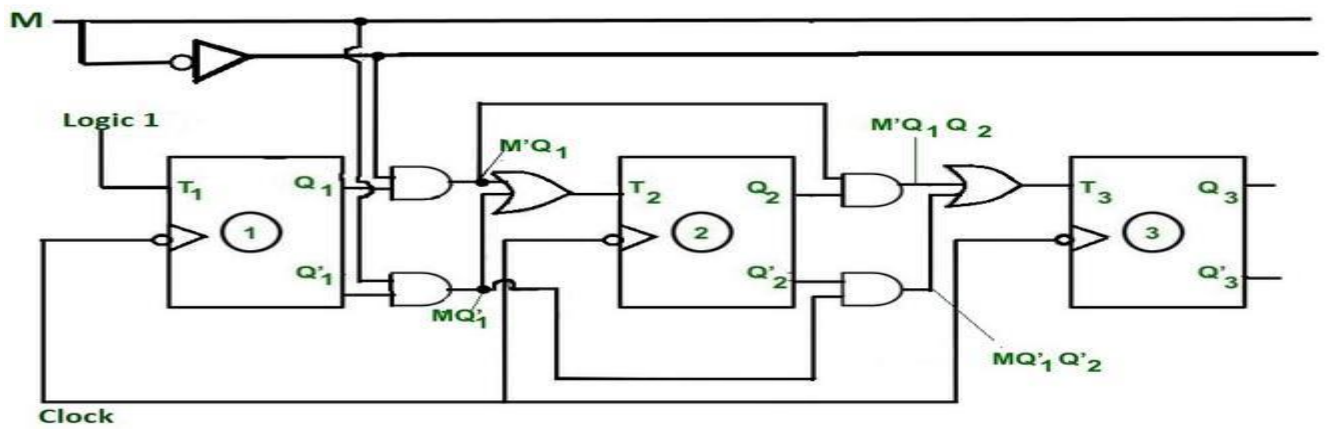


$MQ_3 \backslash Q_2Q_1$	00	01	11	10
00	1	1	1	1
01	1	1	1	1
11	1	1	1	1
10	1	1	1	1

$$T_1 = 1$$

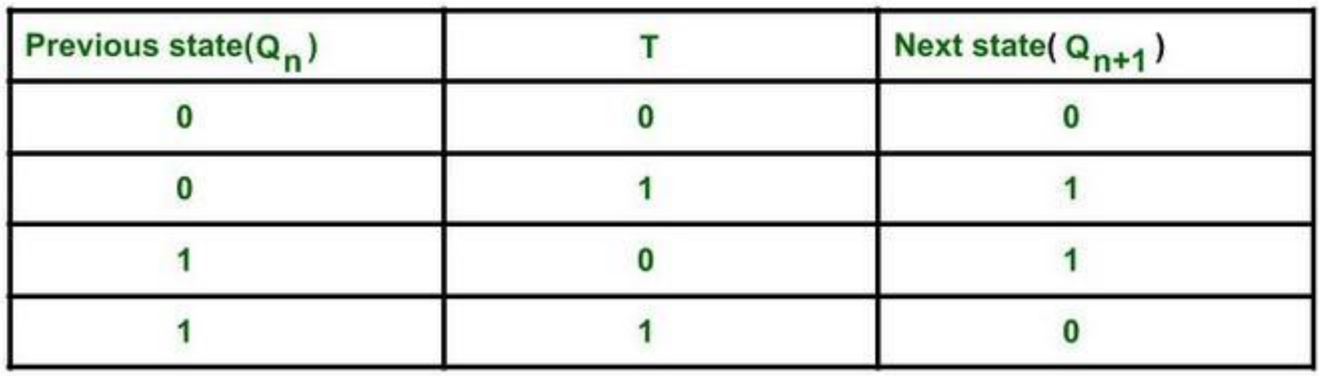
Simplified equation for K map

7. Create a circuit diagram –
The simplified expression for Flip Flops is used to design circuit diagrams. Here all the connections are made according to simplified expressions for Flip Flops.



3 bit synchronous up/down counter.

Explanation :
Here -ve edge triggered clock pulse is used for toggling purpose.



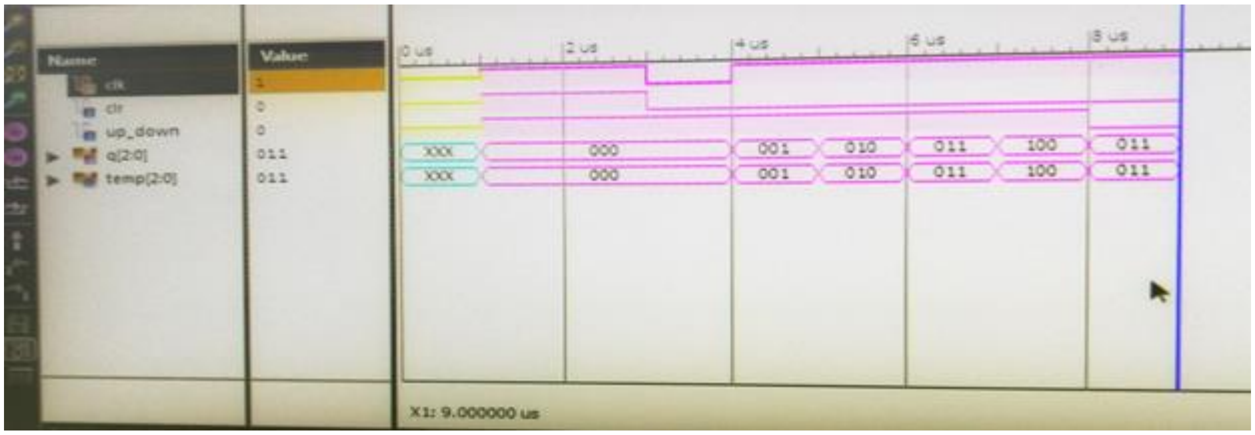
Previous state(Q_n)	T	Next state(Q_{n+1})
0	0	0
0	1	1
1	0	1
1	1	0

Characteristics table of T FF

PROGRAM:

```
module counter (q,clk,clr,up_down);
inputclk, clr, up_down;
output[2:0]q;
reg[2:0]temp;
always@(posedgeclk or posedgeclr)
begin
if(clr)
temp=3'b000;
else
if(up_down)
temp=temp+1'b1;
else
temp=temp-1'b1;
end
assign q=temp;
endmodule
```

3-BIT SYNCHRONOUS UP/DOWN COUNTER OUTPUT:



RESULT:

Thus the Program of 3-bit synchronous up/down counterwas designed and simulated using Verilog HDL and also implemented in FPGA Kit.

Expt No: 7 **4-BIT ASYNCHRONOUS UP/DOWN COUNTER**

Date:

AIM:

To design and simulate 4-Bit Asynchronous Up/Down Counter Using HDLand also implement in FPGA Kit.

APPARATUS REQUIRED:

- PC
- Xilinx software
- FPGA(Spartan 3E) Kit

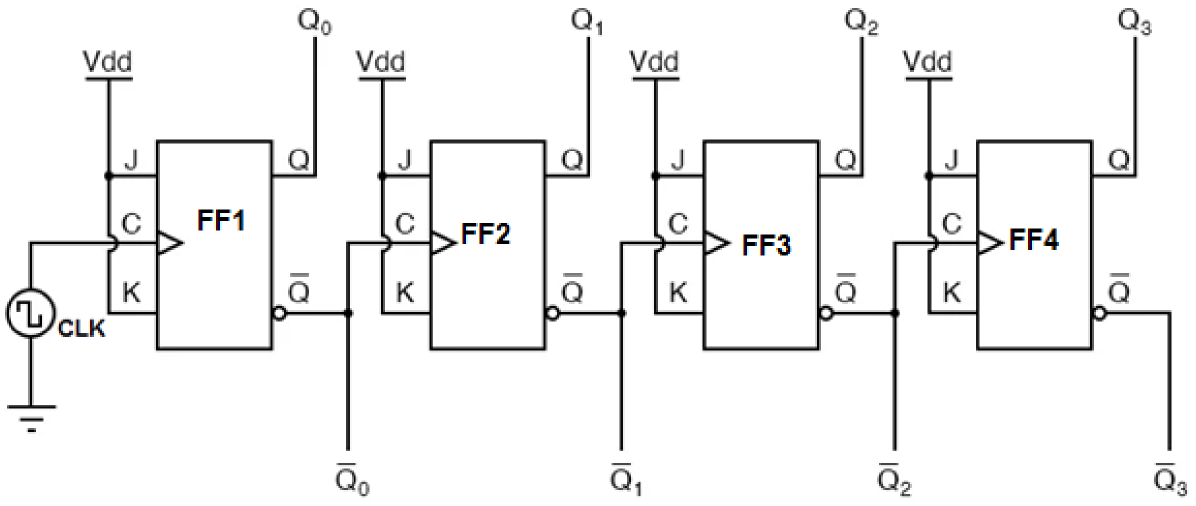
PROCEDURE:

19. Start the Xilinx ISE by using start→ program file → Xilinx ISE → Project navigator
20. File → New Project
21. Enter the project name and location then click next
22. select the Device and other category and click next twice and finish
23. Click on the symbol of FPGA device and then right click → click on new source
24. Select the Verilog Module and give the file name → click next and define ports → click next and finish
25. Writing the behavioral verilog code in verilog Editor
26. Run the Check syntax → process window → synthesize → double click check syntax and remove errors, if present, with proper syntax & coding.
27. synthesis your design, from the source window select, Synthesis/Implementation from the window Now double click the synthesis →XSt
28. After the HDL synthesis phase of the synthesis process, you can display a schematic representation of your synthesized source file. This schematic shows a representation of the pre-optimized design in terms of generic symbols, such as adders, multipliers, counters, Ann gates and OR gates → double click View RTL schematic.
29. After Synthesis you assign the Pin Value for your design so, → double click the Assign Package Pins
30. Enter the Pin value for your input and output signals. If you want see your Pin assignment in FPGA zoom in Architecture View or Package View

31. You see the Pins in FPGA. Save file as XST Default click ok and close the window
32. Double Click Implementation Design
33. Right click the Generate Programming file→select properties and then select the start-up options→change the clock into JTAG clock, then click apply and ok.
34. Double click the Generate Programming file.
35. Double click Configure Device→click finish →select the bit file and then click ok
36. Right click Xilinx Device→clickProgram→ok.

4-bit Asynchronous up/down counter using HDL

CIRCUIT DIAGRAM:

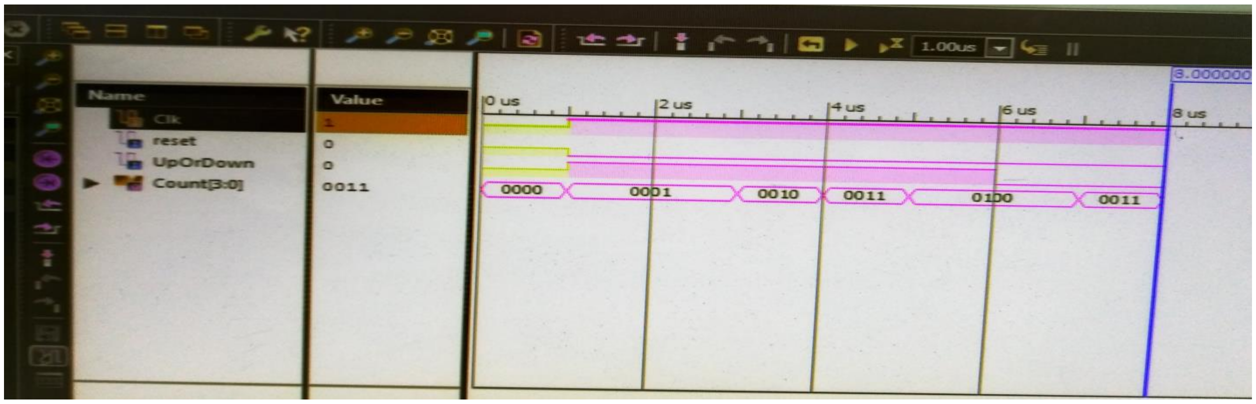


PROGRAM

```
moduleddfsfd (
Clk,
reset,
UpOrDown,  //high for UP counter and low for Down counter
   Count
   );
inputClk,reset,UpOrDown;
output [3 : 0] Count;
reg [3 : 0] Count = 0;                            //input ports and their sizes
always @(posedge(Clk) or posedge(reset))
                                          //output ports and their size
begin
if(reset == 1)
```

```
            Count <= 0;
else
if(UpOrDown == 1)   //Up mode selected
if(Count == 15)
                    Count <= 0;
else
                    Count <= Count + 1; //Incremend Counter
else//Down mode selected
if(Count == 0)
                    Count <= 15;
else
                    Count <= Count - 1; //Decrement counter
end
//Internal variables
endmodule
```

4-BIT ASYNCHRONOUS UP/DOWN COUNTEROUTPUT:



RESULT:

Thus the Program of 4-bit Asynchronous up/down counterwas designed and simulated using Verilog HDL and also implemented in FPGA Kit.

Expt No: 8

Date: **CMOS NAND, NOR GATE AND FLIPFLOP**

AIM:

To design and simulate CMOSinverter , 2-input NAND, NOR gate, Flipflopand also automaticlayout generationusing microwind.

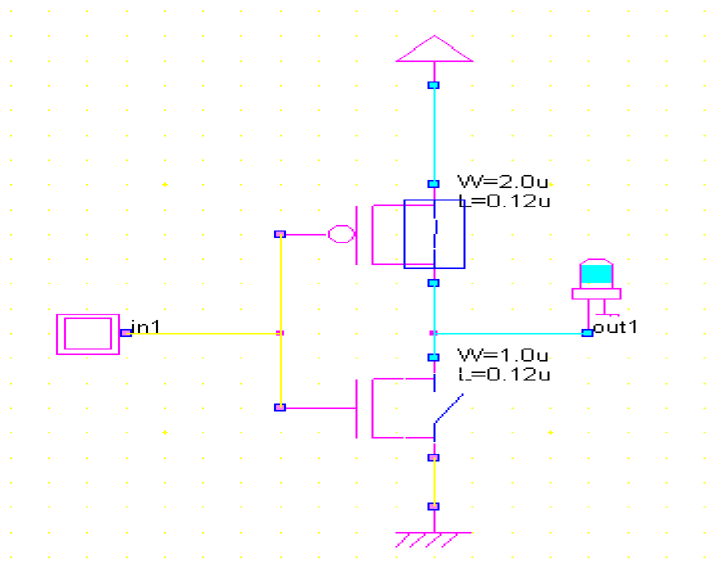
APPARATUS REQUIRED:

- PC
- Microwind
- Dsch 03
- Mw 03

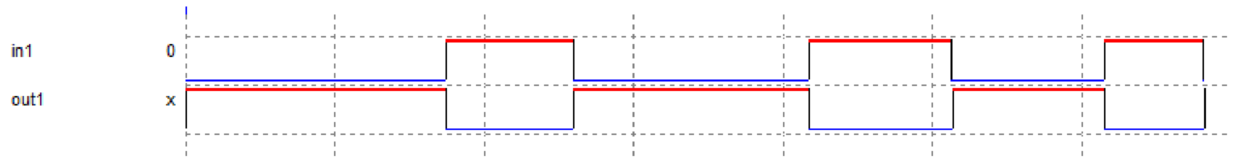
PROCEDURE:

- Open the dsch03 and design a circuit by using symbol library.
- After the commend circuits go to file and save the design and close the dsch3
- Next open the microwind 3 and go to the compile the verilog file then compile and were to editor.

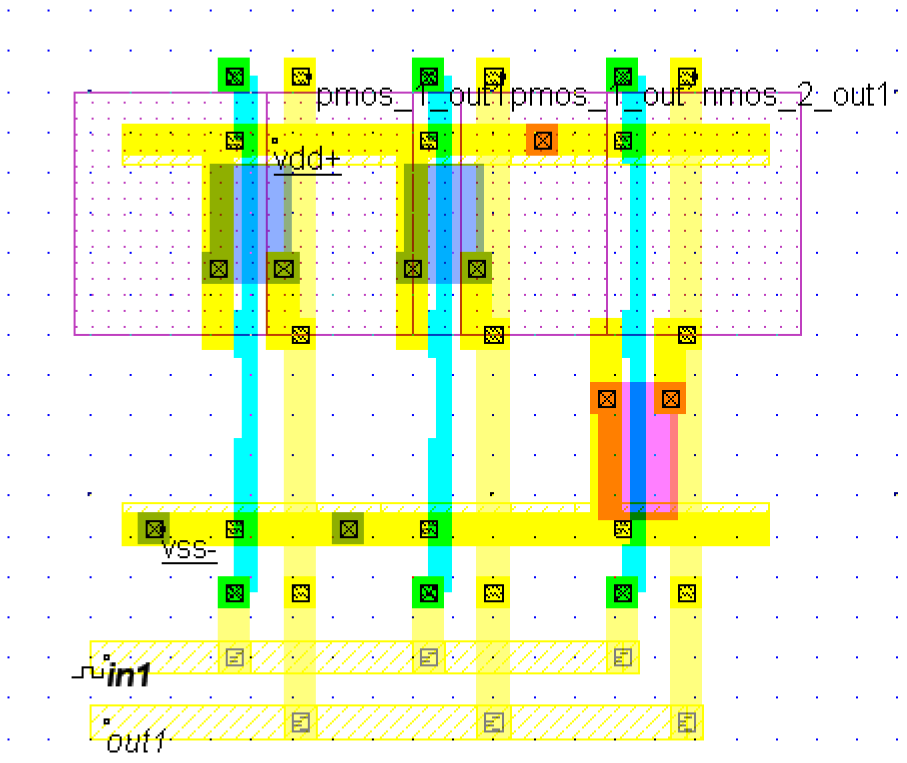
CIRCUIT FOR CMOS INVERTER:



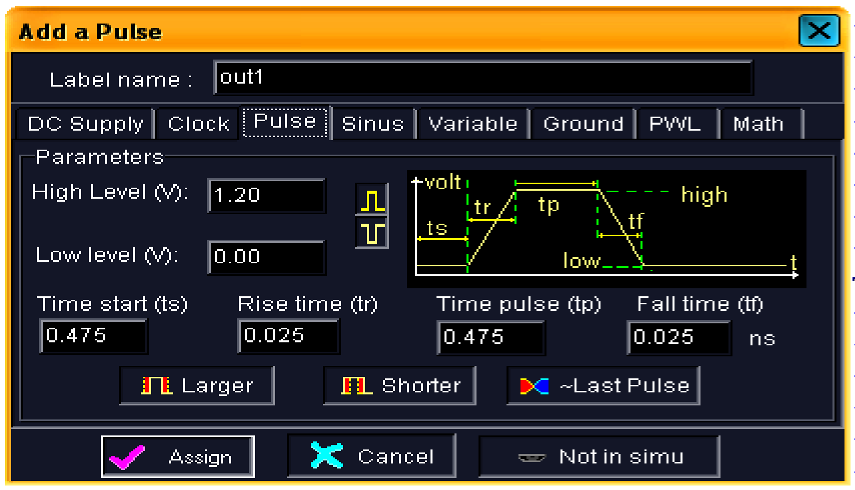
SIMULATIONOUTPUT:



LAYOUT GENERATION:



OUTPUT FOR TIMING ANALYSIS:


Add a Pulse

Label name : out1

DC Supply | Clock | Pulse | Sinus | Variable | Ground | PWL | Math

Parameters

High Level (V): 1.20

Low level (V): 0.00

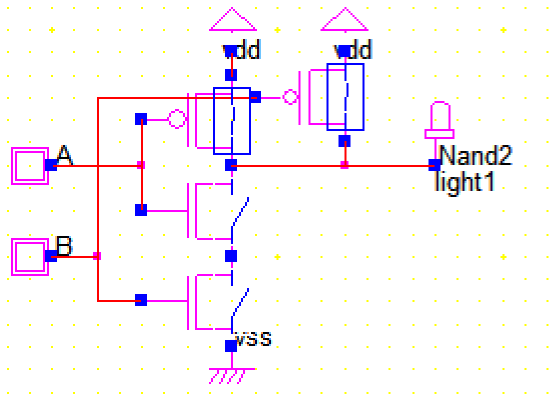
Time start (ts) 0.475

Rise time (tr) 0.025

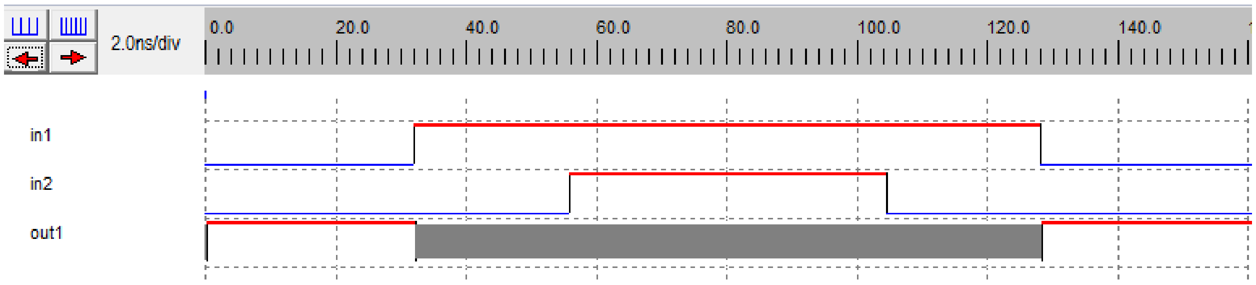
Time pulse (tp) 0.475

Fall time (tf) 0.025 ns

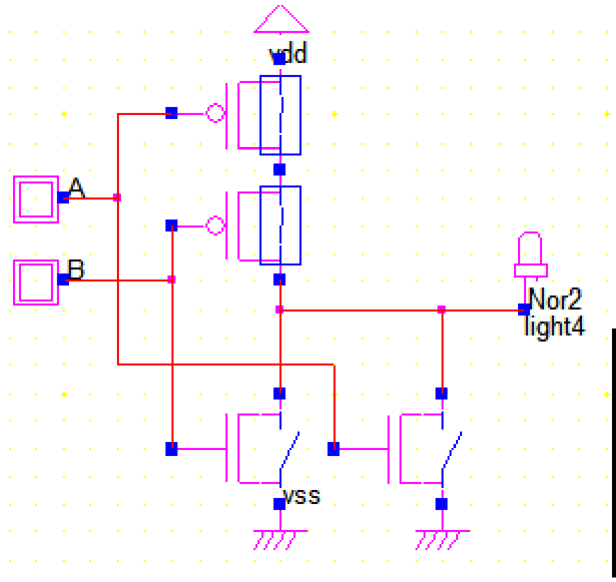
CIRCUIT DIAGRAM FOR CMOS NAND GATE:

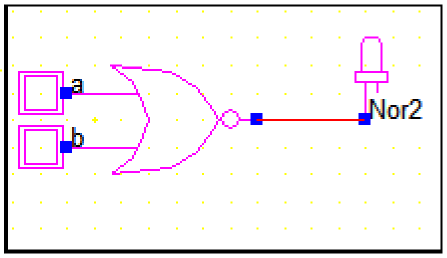


CMOS NAND GATE OUTPUT:

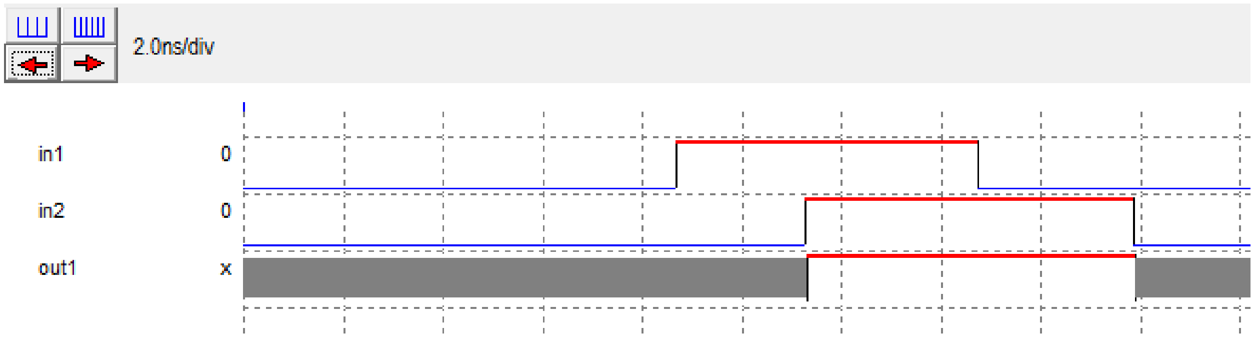


CMOS NOR GATE:

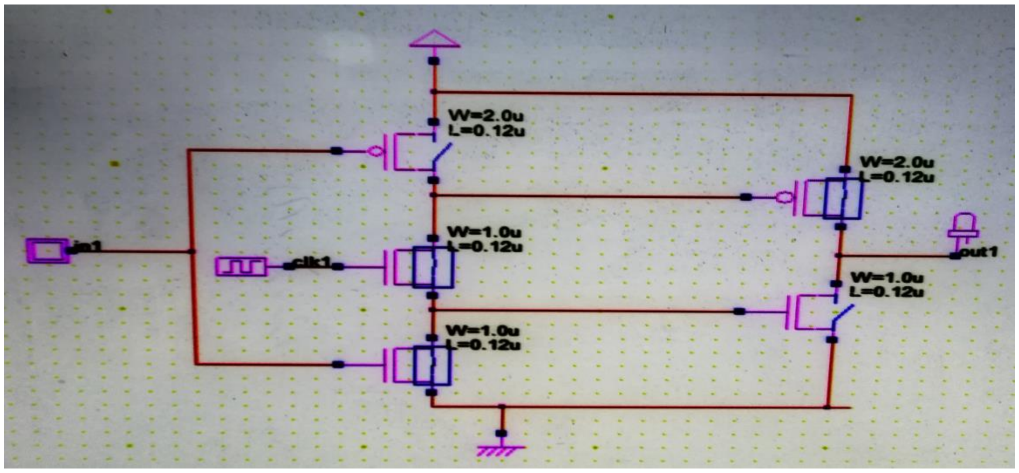




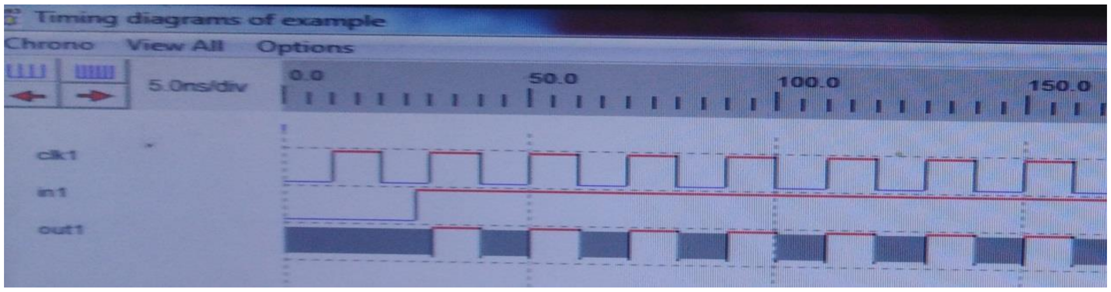
NOR GATE OUTPUT:



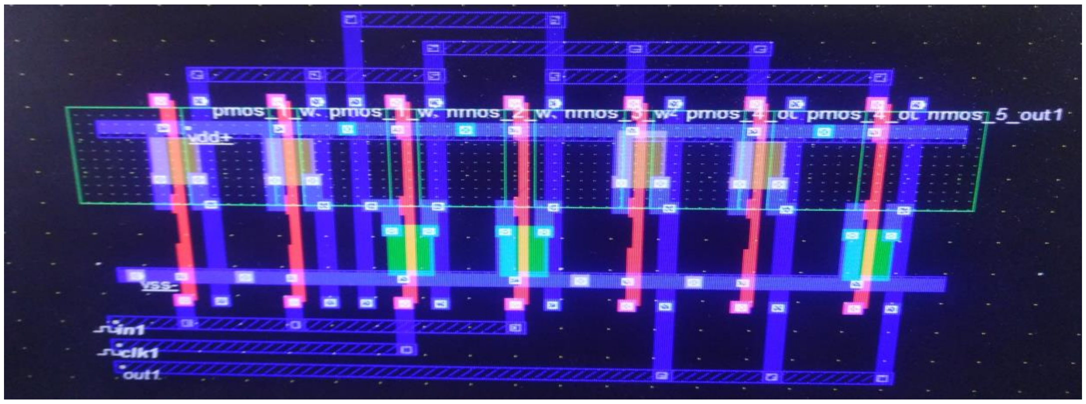
D-FLIPFLOP:



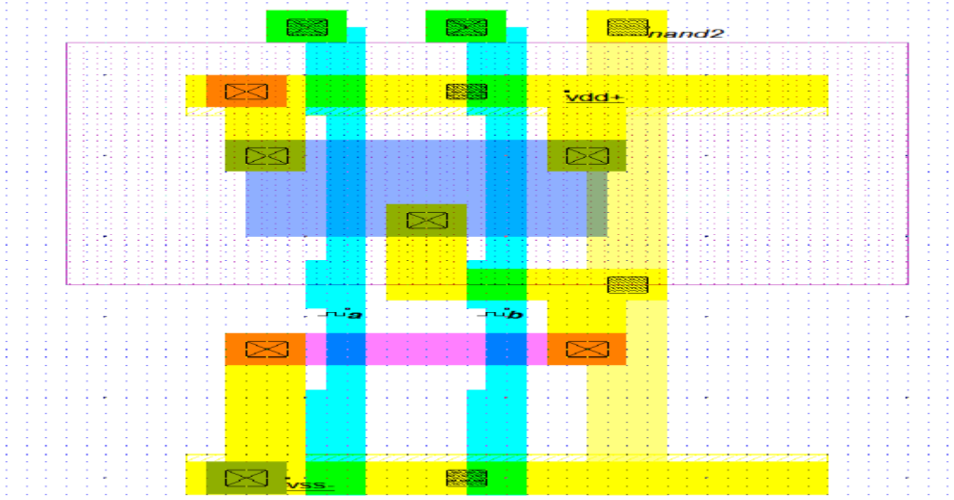
SIMULATION OF D-FLIPFLOP:



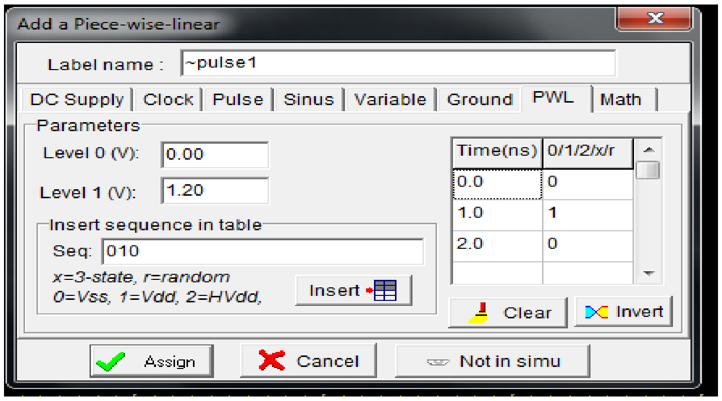
LAYOUT OF D-FLIPFLOP:



NAND GATE LAYOUT:



NAND GATE TIMING ANALYSIS:



Label name : ~pulse1

Parameters

Level 0 (V): 0.00

Level 1 (V): 1.20

Insert sequence in table

Seq: 010

x=3-state, r=random
0=Vss, 1=Vdd, 2=HVdd,

Time(ns)	0/1/2/x/r
0.0	0
1.0	1
2.0	0

RESULT:

Thus the CMOS 2-input NAND, NOR gate, Flip-flopwas designed and simulated and also automatic layout generation completed using micro wind.

Expt No: 9

Date: **SYNCHRONOUS COUNTER**

AIM:

To design and simulate 4-bit Synchronous Counter and also automatic layout generation usingmicrowind.

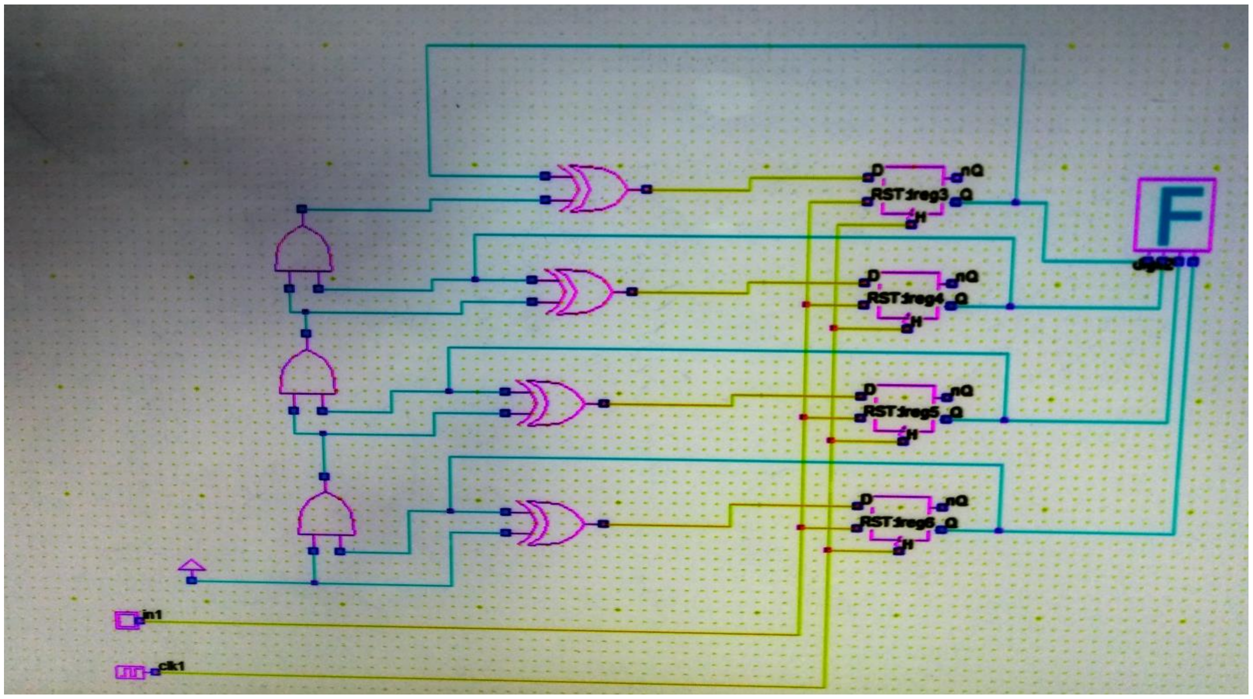
APPARATUS REQUIRED:

- PC
- Microwind
- Dsch 03
- Mw 03

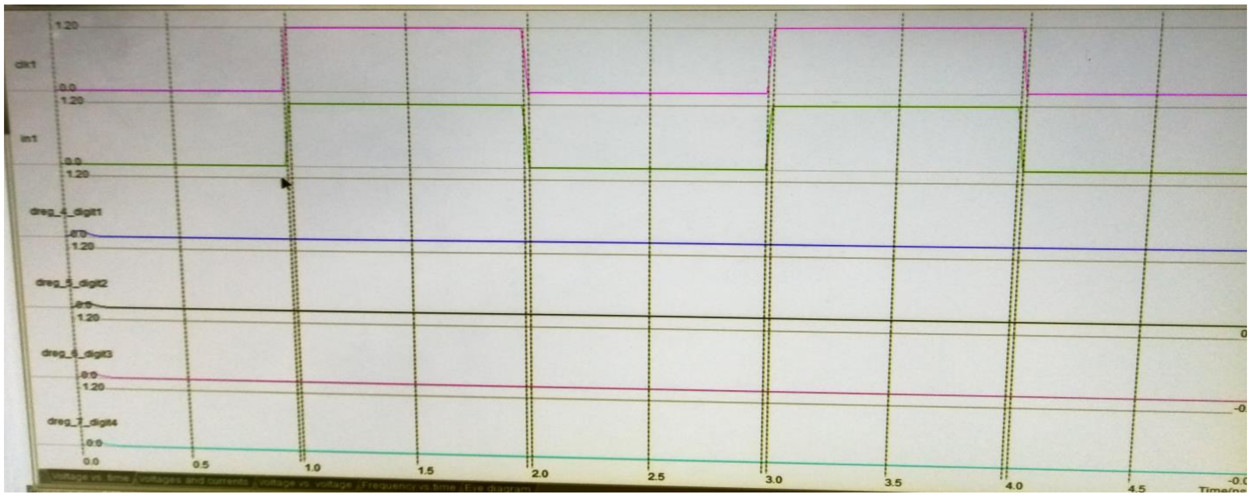
PROCEDURE:

- Open the dsch03 and design a circuit by using symbol library.
- After the commend circuits go to file and save the design and close the dsch3.
- Next open the micro wind 3 and go to the compile the Verilog file then compile and were to editor.

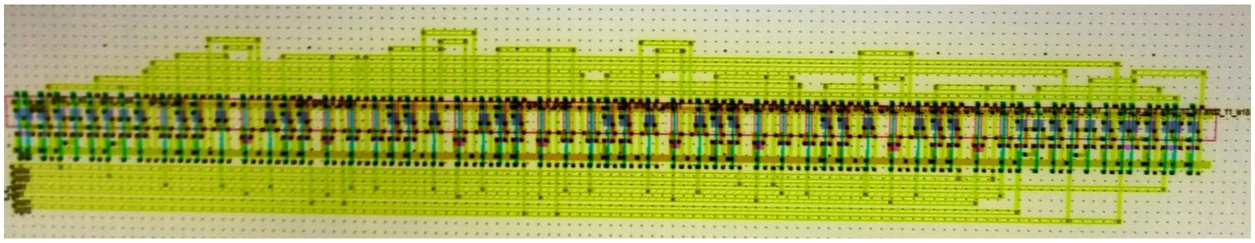
CIRCUIT DIAGRAM:



SIMULATION OUTPUT:



Layout of synchronous counter:



RESULT:

Thus the 4-bit Synchronous Counter was designed and simulated and also automatic layout generation completed using microwind.

EXPT NO: 10

DATE:

DIFFERENTIAL AMPLIFIER

OBJECTIVE

To design and simulate the differential amplifier circuits using Tanner EDA tool.

SOFTWARE USED

Tanner EDA Tools

(i) S-Edit

(ii) T-Edit

(iii) W-Edit

THEORY:

Differential amplifier:

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:

$\mathbf{V_{out}= A_d(V_{in1} - V_{in2})}$, Where V_{in1} and V_{in2} are the input voltages and A_d is the differential gain. In practice, however, the gain is not quite equal, for the two inputs. This means that if V_{in1} and V_{in2} 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. $\mathbf{V_{out}= (A_d(V_{in1} - V_{in2}) + A_c(V_{in1} + V_{in2}))/2}$

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): $\mathbf{CMRR= A_d / A_c}$

The input common-mode range (ICMR) is the range of common-mode voltages over which the differential amplifier continues to sense and amplify the difference signal with the same gain.

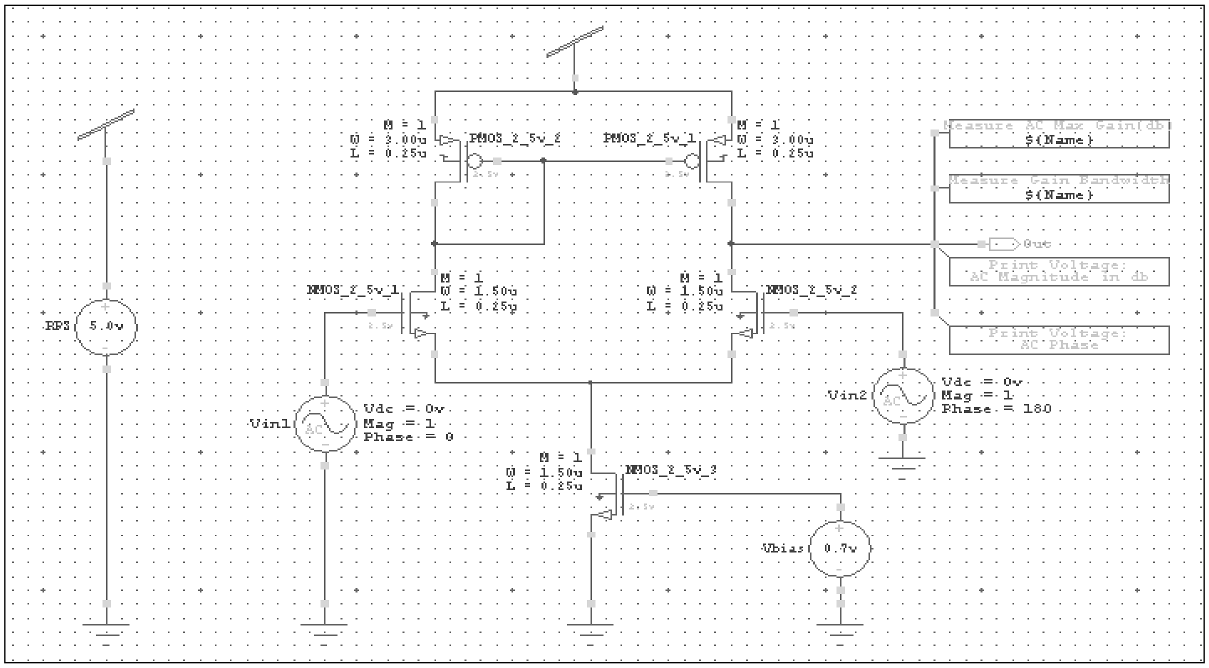
Typically, ICMR is defined by the common-mode voltage range over which all MOSFETS remain in the saturation region..

PROCEDURE:

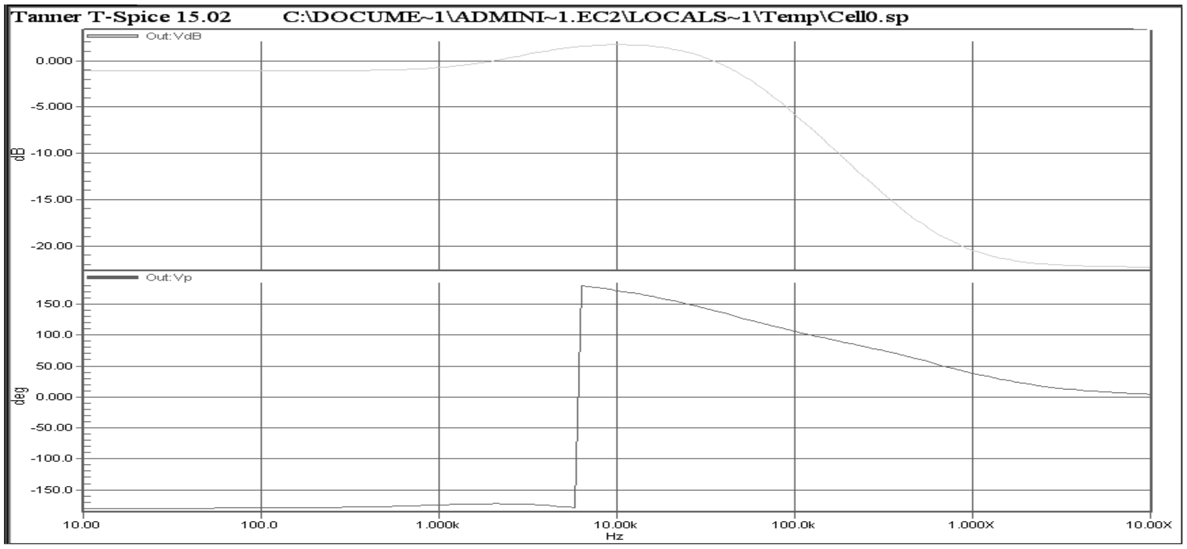
1. Open a schematic editor(S-Edit) from the Tanner EDA Tools.
2. Select the required components from the symbol browser and design given circuit using S-Edit.
3. Write the program in T-Edit and run the simulation to simulate the given program to view the result.
4. Output waveform is viewed in the waveform viewer.

CIRCUIT DIAGRAM:

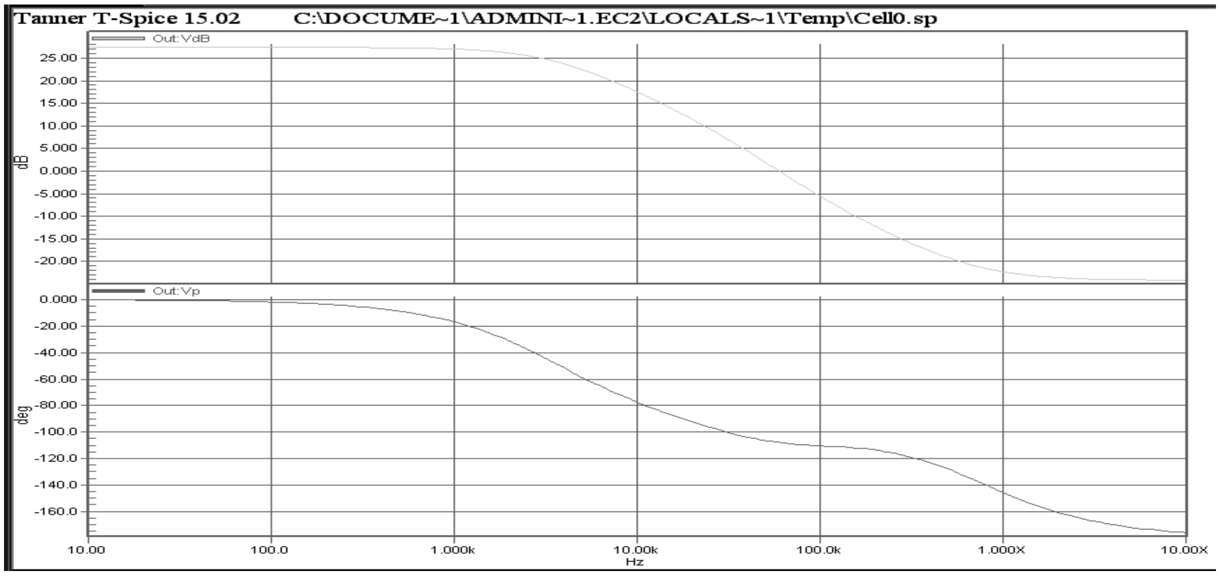
SCHEMATIC DIAGRAM:



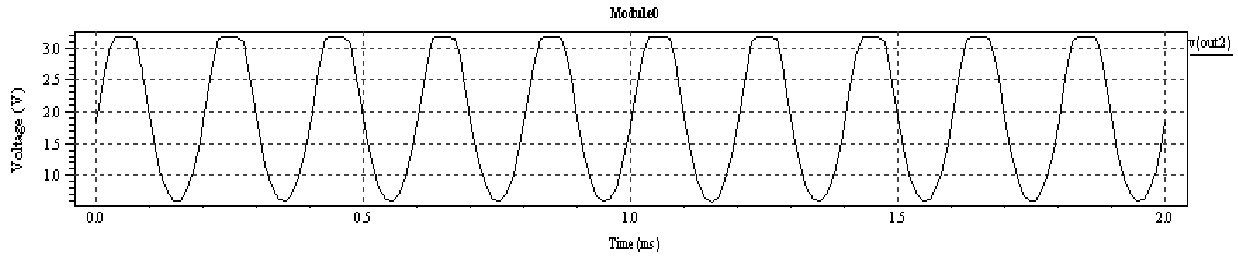
COMMON MODE OUTPUT:

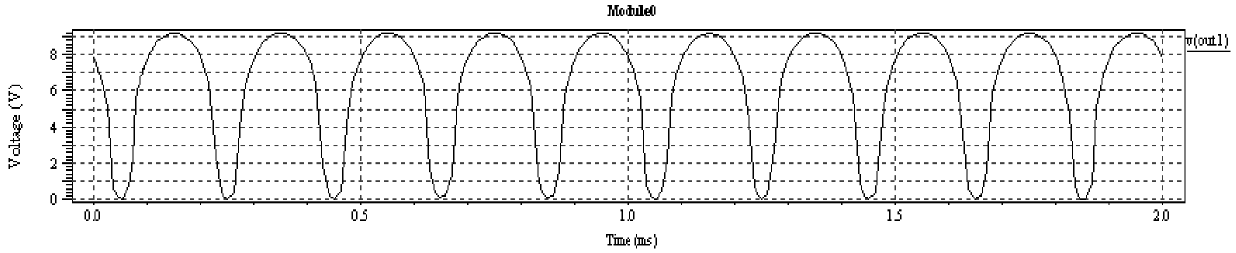


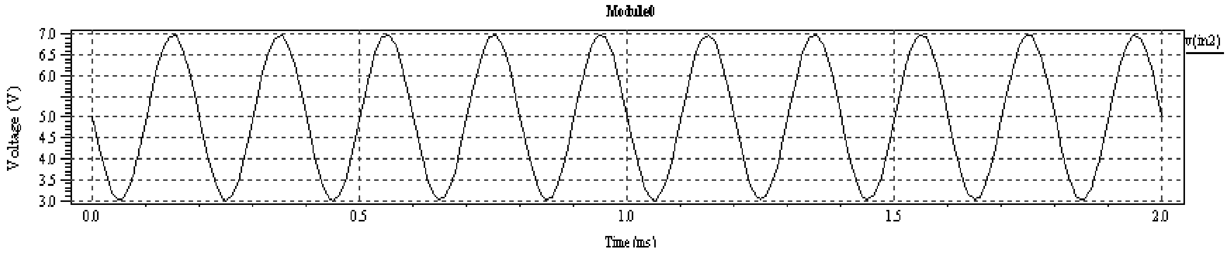
DIFFERENTIAL MODE OUTPUT:

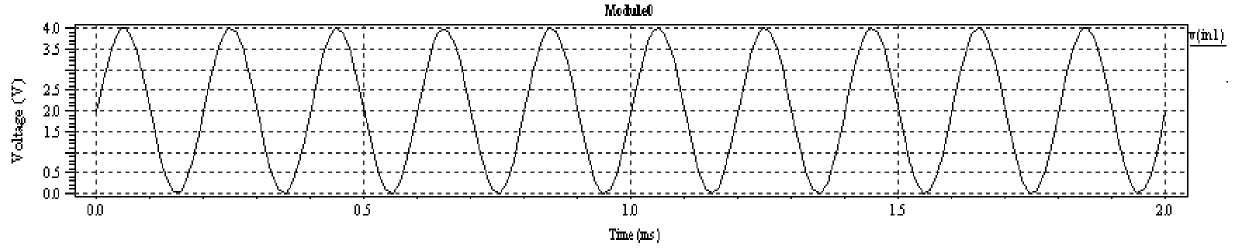


OUTPUT:









RESULT:

Thus theDifferential Amplifier was designed and simulated using Tanner EDA Tools.

Expt No: 11

Date: **DESIGN AND SIMULATE THE ANALYSIS OF SOURCE FOLLOWERS**

AIM:

Todesign and simulate the analysis of source followersusing Tanner EDA T-spice (S-Edit).

APPARATUS REQUIRED:

- PC
- Tanner EDA
- T-spice
- S-Edit

PROCEDURE FOR TANNER TOOLS:

STEP1:open s-edit window

STEP2:go to file- > new- > new design

STEP3:go to cell- > new view

STEP4:add libraries file to the new cell

STEP5: Instance the devices by using appropriate library files

STEP6:save the design & setup the simulation

STEP7:run design &observe waveforms

STEP8:observe the input & output waveform by given appropriate inputs.

CIRCUIT DESCRIPTION:

Common-Source amplifier is one of the three basic single stage field effect transistor topologies typically used as a voltage or trans-conductance amplifier. As a trans-conductance 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 Ohms law. Because of its high input-impedance, relatively high gain, low noise, speed and simplicity, common source amplifiers find different applications from sensor signal amplifications to RF noise amplification.

SPECIFICATIONS:

L = 250nm; Tox = 5.7e-9; Vth = 0.388582; Un = 304.684e-4;

Vs = 200mV (max voltage drop across degeneration resistor)

Gain = 10dB; Rin = 50K; Power Budget = 5mW; Kn = 0.1876e-3; Vdd = 2.5V

Fl = 1Khz; Fh = 3Ghz; NF = 3dB; γ = 0.8

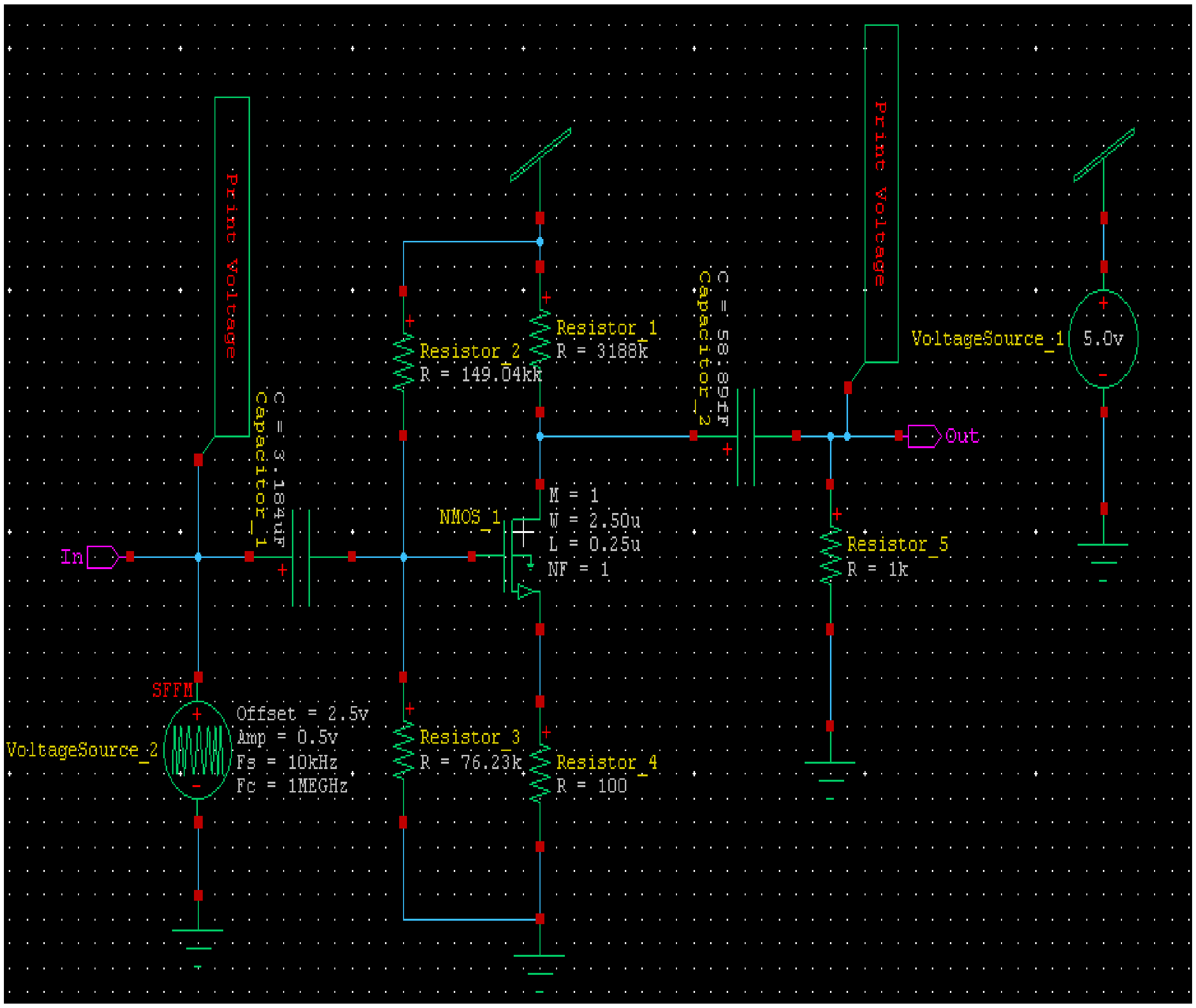
DESIGNED VALUES:

R1= 149.04K; R2= 76.23K

W = 21.321um; Rd = 3188K

Cin = 3.184uF; Cout = 58.89nF

CIRCUIT DIAGRAM:

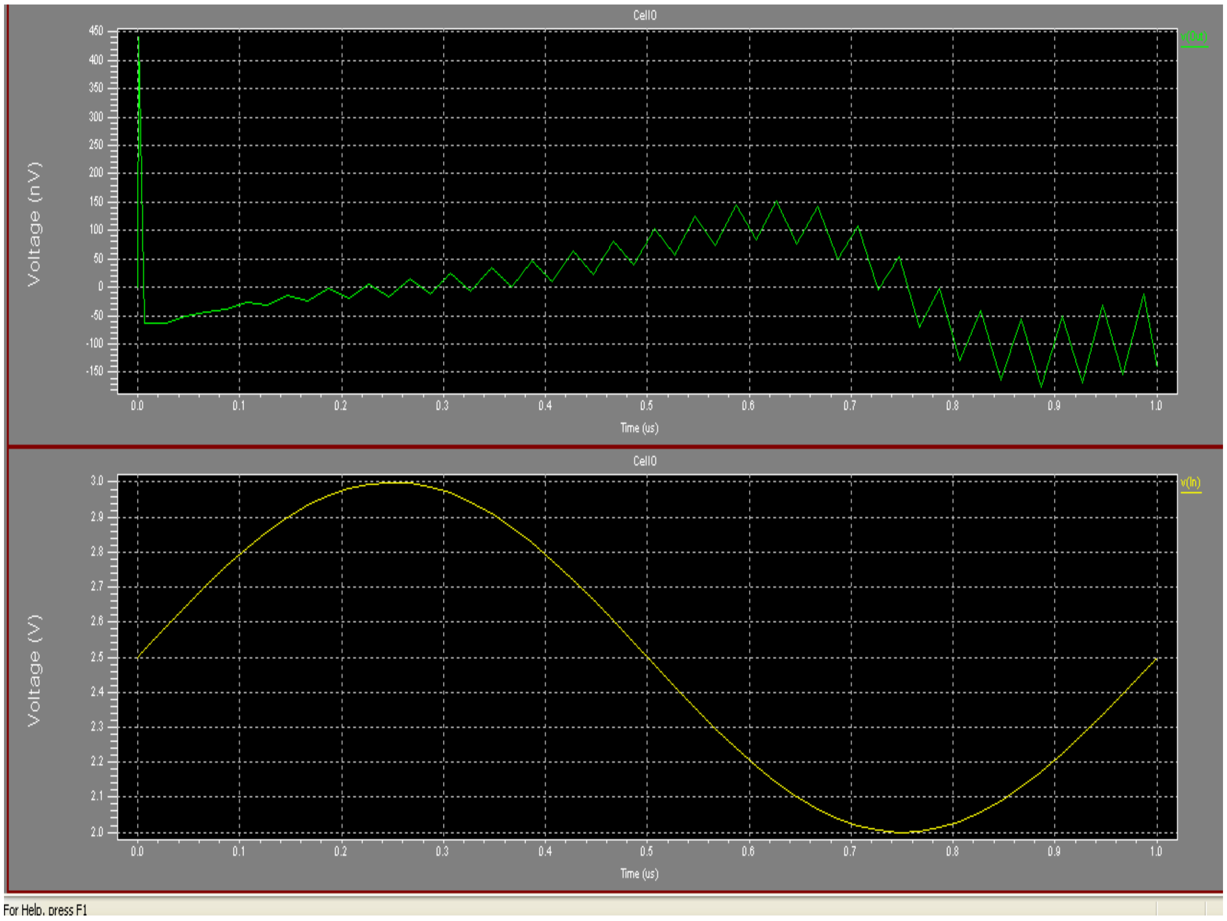


T-SPICE NETLIST:

```
SPICE export by:  SEDIT 13.00

* Export time:      Tue May 13 12:23:36 2014

* Design:           opi

* Cell:             Cell0

* 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\ece\My Documents\Tanner EDA\Tanner Tools
v13.0\Libraries\All\opi

* Exclude global pins:   no

* Control property name: SPICE

********* Simulation Settings - General section *********
.lib "C:\Documents and Settings\ece\My Documents\Tanner EDA\Tanner Tools
v13.0\Libraries\Models\Generic_025.lib"tt

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

*-------- Devices: SPICE.ORDER > 0 --------
CCapacitor_1 In N_2  3.184u
CCapacitor_2 N_1 Out  58.89f
RResistor_1 Vdd N_1  R=3.188meg
RResistor_2 Vdd N_2  R=149.04k
RResistor_3 N_2 Gnd  R=76.23k
RResistor_4 N_3 Gnd  R=100
RResistor_5 Out Gnd  R=1k
MNMOS_1 N_1 N_2 N_3 0 NMOS W=2.5u L=250n AS=2.25p PS=6.8u AD=2.25p PD=6.8u
VVoltageSource_1 VddGnd  DC 5
VVoltageSource_2 In Gnd  SFFM(2.5 500m 1meg 0 10k)
```

```
.PRINT TRAN V(In)
.PRINT TRAN V(Out)

********* Simulation Settings - Analysis section *********
.tran 10ns 1000ns
********* Simulation Settings - Additional SPICE commands *********
.end
```

OUTPUT WAVEFORM:



RESULT:

Thus the analysis of source followers was designed and simulated using Tanner EDA Tools.

EX.NO: 12

DATE:

CMOS INVERTING AMPLIFIER

AIM:

To design and simulate a single stage operational amplifiers using Tanner EDA tools.

.

APPARATUS REQUIRED:

- Tanner EDA Tools
- S-Edit
- PC

PROCEDURE FOR TANNER TOOLS:

STEP1: open s-edit window

STEP2: go to file- > new- > new design

STEP3: go to cell- > new view

STEP4: add libraries file to the new cell

STEP5: instance the devices by using appropriate library files

STEP6: save the design & setup the simulation

STEP7: run design &observe waveforms

STEP8: observe the input & output waveform by given appropriate inputs.

THEORY:

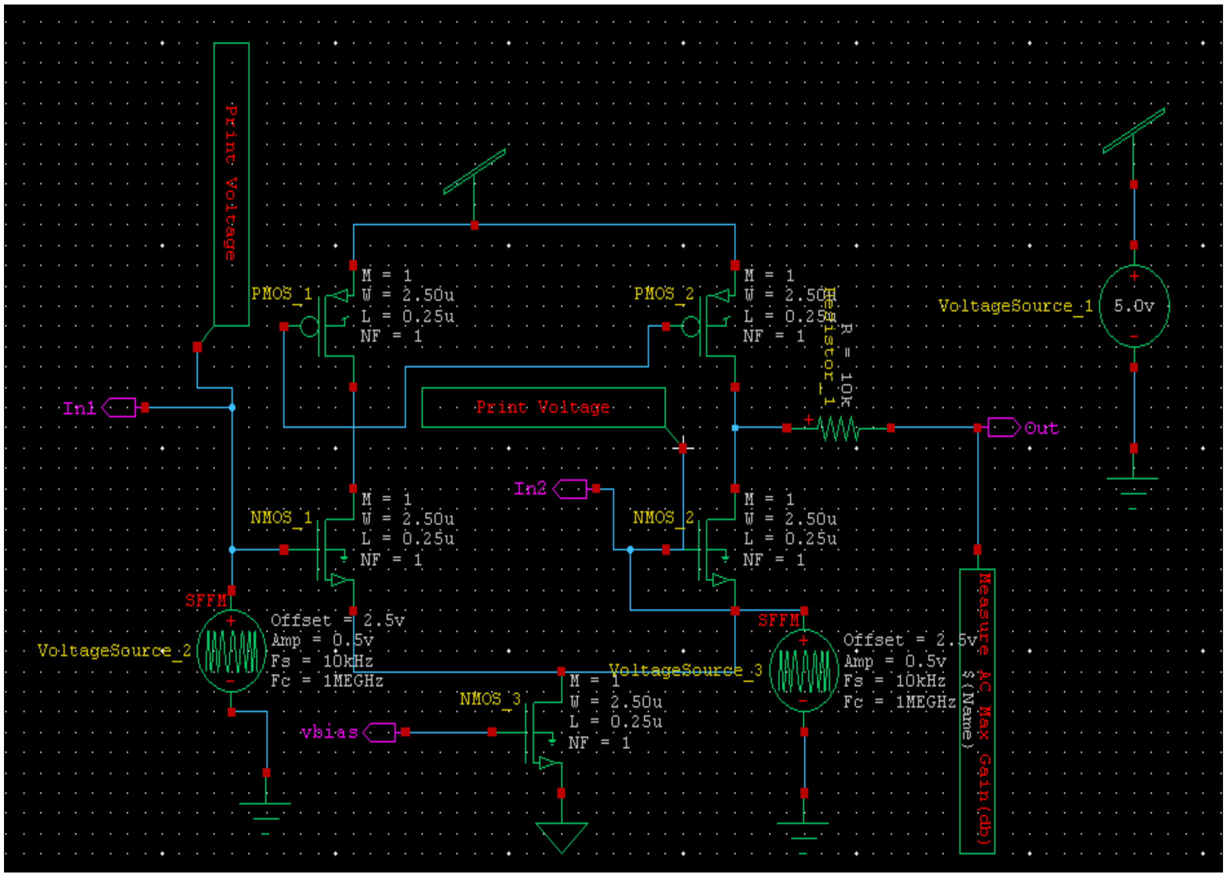
The single stage operational amplifier produces an output with a reduced voltage swing. A two stage operational amplifier increases the voltage swing. The differential amplifier constitutes the first stage of the two stage operational amplifier. The differential input signal applied across the two terminals will be amplified according to the gain of the differential stage. The current mirror topology performs the differential to single-ended conversion of the input signal and also, the load helps with the common mode rejection ratio. The second stage is a current sink load inverter (common source) which provides additional gain.

SPECIFICATIONS:

$V_{DD} = 2.5V$,,$V_{SS} = -2.5V$,,$GB = 0.5V$,,$ICMR = -0.5\ to\ 2$,,$C_L = 4.5fF$,$|V_{Tp} = 0.42|$, $V_{Tn} = 0.399$,$K_n = 302 * 10^{-6}$, $K_p = 65.77 * 10^{-6}$

SINGLE STAGE OPERATIONAL AMPLIFIER:

SCHEMATIC:



T-SPICE Netlist:

* SPICE export by: S-Edit 15.00

* Export time: Fri Feb 28 02:26:05 2014

* Design: OPAMP4_SINGLE

* Cell: Cell0

* View: view0

* Export as: top-level cell

* Export mode: hierarchical

```
* Exclude empty cells: yes

* Exclude .model:   no

* Exclude .end:     no

* Exclude simulator commands:     no

* Expand paths:     yes

* Wrap lines:       no

* Root path:        C:\software\tanner\OPAMP4_SINGLE

* Exclude global pins:   no

* Control property name: SPICE

********* Simulation Settings - General Section *********

.include"C:\software\tanner\LibraryfilesforTannerEDAV15\Libraries\Libraries\Models\ptm_180nm.md"

VDD VDD VSS 2.5

VBiasVBias VSS 1

*-------- Devices With SPICE.ORDER >0.0  --------

***** Top Level *****

RResistor_1 N_2 Out  R=100k $ $x=6000 $y=4614 $w=600 $h=149

MNMOS_1 N_1 In- N_4 0 NMOS W=5u L=180n AS=4.5p PS=11.8u AD=4.5p PD=11.8u $ $x=2800
$y=3900 $w=400 $h=600

MNMOS_2 N_2 In+ N_4 0 NMOS W=5u L=180n AS=4.5p PS=11.8u AD=4.5p PD=11.8u $ $x=5200
$y=3900 $w=400 $h=600

MNMOS_3 N_4 VBiasVss 0 NMOS W=34u L=180n AS=30.6p PS=69.8u AD=30.6p PD=69.8u $
$x=4000 $y=2900 $w=400 $h=600

MPMOS_1 N_1 N_1VddVdd PMOS W=720n L=180n AS=648f PS=3.24u AD=648f PD=3.24u $
$x=2800 $y=5400 $w=400 $h=600

MPMOS_2 N_2 N_1 VddVdd PMOS W=720n L=180n AS=648f PS=3.24u AD=648f PD=3.24u $
$x=5200 $y=5400 $w=400 $h=600

.MEASURE AC AC_Measure_Gain_1 MAX vdb(Out) ON $ $x=7150 $y=3800 $w=1500 $h=200

********* Simulation Settings - Analysis Section *********

VIn+ In+ GND AC 1 SIN (0 2 80K)

VIn- In- GND AC 1 SIN (0 1.5 80K)

*.op
```

```
*VIn+ In+ GND PULSE (0 1 0 0 0 1n 2n)
*VIn- In- GND PULSE (0 0.5 0 0 0 1n 2n)
********* Simulation Settings - Additional SPICE Commands *********
.tran 50p 100u
.print tranv(In+) v(In-)v(Out)
.ac dec 10 100 10G   .print ac vdb(Out) vp(Out).end
```

RESULT:

Thus the single stage operational amplifier was designed and simulated using tanner EDA.