CHETTINAD COLLEGE OF ENGINEERING AND TECHNOLOGY

PULIYUR, KARUR – 639114

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

EC3561 VLSI LABORATORY

ACADEMIC YEAR: 2024-25

III YEAR/V TH Semester Lab Manual

PROGRAM OUTCOMES:

1. Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

2. Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

3. Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

4. Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

5. Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.

6. The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

7. Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

8. Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

9. Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

10. Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

11. Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

12. Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PROGRAM SPECIFIC OUTCOME (PSOs)

PSO1: Design, develop and analyze electronic systems through application of relevant electronics, mathematics and engineering principles

PSO2: Design, develop and analyze communication systems through application of fundamentals from communication principles, signal processing, and RF System Design & Electromagnetics.

PSO3: Adapt to emerging electronics and communication technologies and develop innovative solutions for existing and newer problems

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

EC3561 VLSI LABORATORY

L T P C
0 0 4 2

OBJECTIVES:

The student should be made to:

- To learn Hardware Descriptive Language (Verilog/VHDL).
- To learn the fundamental principles of Digital System Design using HDL and FPGA.
- To learn the fundamental principles of VLSI circuit design in digital domain
- To learn the fundamental principles of VLSI circuit design in analog domain
- To provide hands on design experience with EDA platforms.

.

LIST OF EXPERIMENTS:

1. Design of basic combinational and sequential (Flip-flops) circuits using HDL. Simulate it using Xilinx/Altera Software and implement by Xilinx/Altera FPGA
2. Design an Adder ; Multiplier (Min 8 Bit) using HDL. Simulate it using Xilinx/Altera Software and implement by Xilinx/Altera FPGA
3. Design and implement Universal Shift Register using HDL. Simulate it using Xilinx/Altera Software
4. Design Memories using HDL. Simulate it using Xilinx/Altera Software and implement by Xilinx/Altera FPGA
5. Design Finite State Machine (Moore/Mealy) using HDL. Simulate it using Xilinx/Altera Software and implement by Xilinx/Altera FPGA
6. Design 3-bit synchronous up/down counter using HDL. Simulate it using Xilinx/Altera Software and implement by Xilinx/Altera FPGA
7. Design 4-bit Asynchronous up/down counter using HDL. Simulate it using Xilinx/Altera Software and implement by Xilinx/Altera FPGA
8. Design and simulate a CMOS Basic Gates &amp; Flip-Flops. Generate Manual/Automatic Layout .
9. Design and simulate a 4-bit synchronous counter using a Flip-Flops. Generate Manual/Automatic Layout

10. Design and Simulate a CMOS Inverting Amplifier.

11. Design and Simulate basic Common Source, Common Gate and Common Drain Amplifiers.

12. Design and simulate simple 5 transistor differential amplifier.

TOTAL: 30 PERIODS

OUTCOMES:

At the end of the course, the student should be able to

- Write HDL code for basic as well as advanced digital integrated circuit
- Import the logic modules into FPGA Boards
- Synthesize Place and Route the digital Ips
- Design, Simulate and Extract the layouts of Digital & Analog IC Blocks using EDA tools
- Test and Verification of IC design

INDEX

S.No	Name of the Experiment	Page No	Marks Obtained	Signature of the faculty member.
1	Design of basic combinational and sequential (Flip-flops) circuits using HDL. Simulate it using Xilinx/Altera Software and implement by Xilinx/Altera FPGA			
2	Design an Adder ; Multiplier (Min 8 Bit) using HDL. Simulate it using Xilinx/Altera Software and implement by Xilinx/Altera FPGA			
3	Design and implement Universal Shift Register using HDL. Simulate it using Xilinx/Altera Software			
4	Design Memories using HDL. Simulate it using Xilinx/Altera Software and implement by Xilinx/Altera FPGA			
5	Design Finite State Machine (Moore/Mealy) using HDL. Simulate it using Xilinx/Altera Software and implement by Xilinx/Altera FPGA			
6	Design 3-bit synchronous up/down counter using HDL. Simulate it using Xilinx/Altera Software and implement by Xilinx/Altera FPGA			
7	Design 4-bit Asynchronous up/down counter using HDL. Simulate it using Xilinx/Altera Software and implement by Xilinx/Altera FPGA			
8	Design and simulate a CMOS Basic Gates &amp; Flip-Flops. Generate Manual/Automatic Layout .			
9	Design and simulate a 4-bit synchronous counter using a Flip-Flops. Generate Manual/Automatic Layout			
10	Design and Simulate a CMOS Inverting Amplifier.			
11	Design and Simulate basic Common Source, Common Gate and Common Drain Amplifiers.			
12	Design and simulate simple 5 transistor differential amplifier.			

VALUE ADDED EXPERIMENTS				
S.No	**Name of the Experiment**	**Page No**	**Marks Obtained**	**Signature of the faculty member.**
1	Design and Implementation of Encoder using HDL .Simulate it using Xilinx /Altera Software & Implement by Xilinx Alter aFPGA			
2	Design and Implement of Decoder using HDL. Simulate it using Xilinx /Altera Software & Implement by Xilinx Alter aFPGA			

DIGITAL SYSTEM DESIGN USING HDL & FPGA

EX.NO.1 A DESIGN OF BASIC COMBINATIONAL CIRCUITS USING XILINX SOFTWARE

AIM :

To write a Verilog program for basic Combinational logic circuit to synthesize and simulate using Xilinx software tool.

TOOLS REQUIRED:

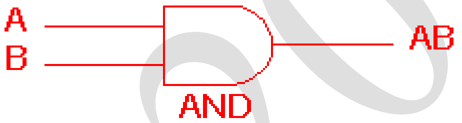
1. Xilinx ISE Design Suite 8.1i
2. PC

PRE – LAB QUESTIONS :

1. What is Digital Gate?
2. What is the principle of logic gates?
3. What is Logic gates type?
4. What is truth table?
5. What is the Universal gates?

THEORY:

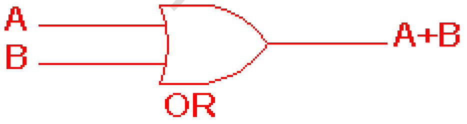
AND GATE



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

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

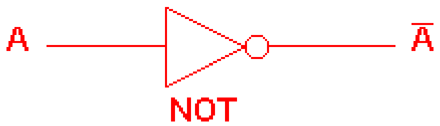
OR GATE

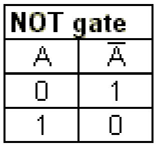


2 Input OR gate		
A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

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

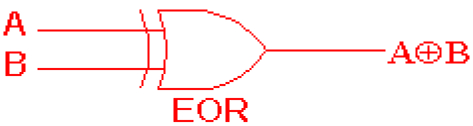


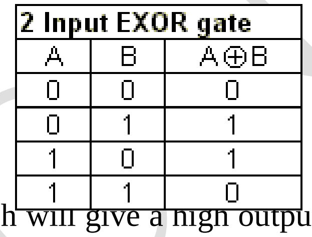


NOT gate	
A	$\overline{A}$
0	1
1	0

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. 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.

EXOR GATE

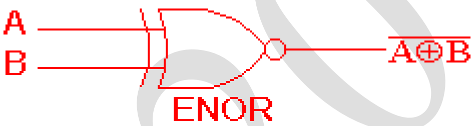


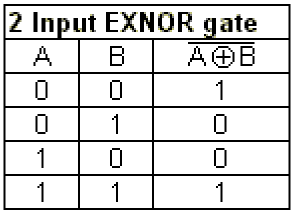


2 Input EXOR gate		
A	B	A⊕B
0	0	0
0	1	1
1	0	1
1	1	0

The 'Exclusive-OR' gate is a circuit which will give a high output if either, but not both, of its two inputs are high. An encircled plus sign ( ) is used to show ⊕ the EXOR operation.

EXNOR GATE





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

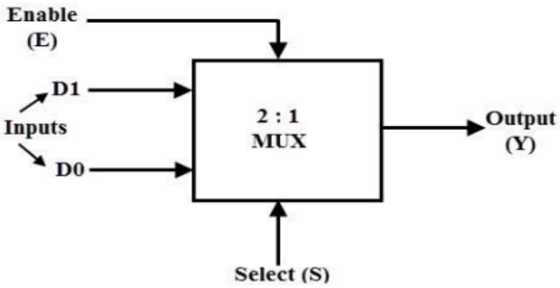
The 'Exclusive-NOR' gate circuit does the opposite to the EOR gate. It will give a low output if either, but not both, of its two inputs are high. The symbol is an EXOR gate with a small circle on the output.

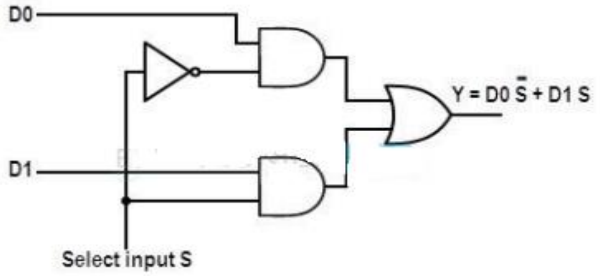
MULTIPLEXER:

A multiplexer is a combinational digital logic switching device that has multiple inputs and one output.

In addition to the input and output lines,the multiplexer has data select lines through which the data passed from an input line to output line

LOGIC DIAGRAM: MULTIPLEXER





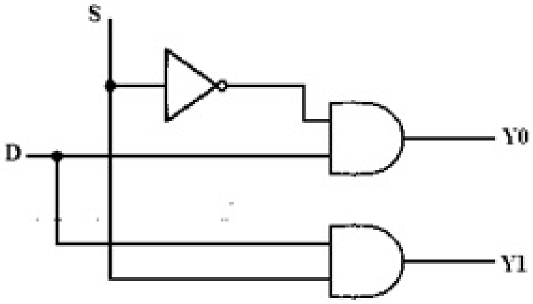
TRUTH TABLES :

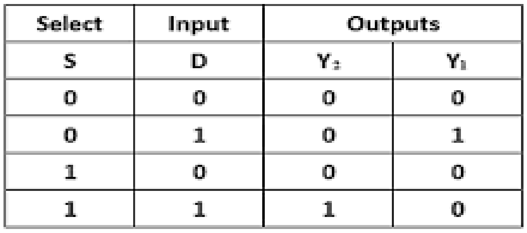
D0	D1	Y
0	0	I0
0	1	I1
1	0	I2
1	1	I3

DEMULTIPLER:

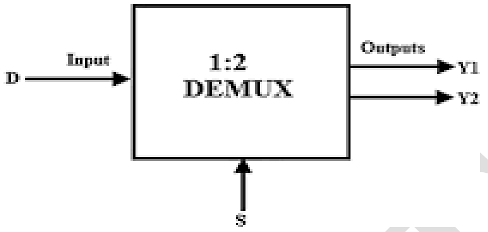
A demultiplexer is a combinational digital logic switching device the has single input and multiple output .

In addition to the input and output lines ,the demultiplexer has data select a lines through which the data passed from an input line to output line.



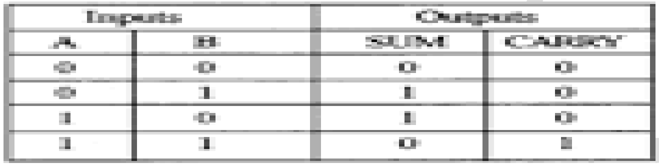


Select	Input	Outputs	
S	D	Y_2	Y_1
0	0	0	0
0	1	0	1
1	0	0	0
1	1	1	0

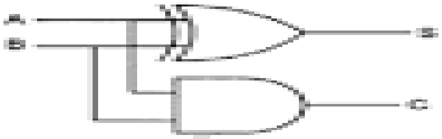


HALF ADDER:

The half adder consists of two input variables designated as Augends and Addend bits. Output variables produce the Sum and Carry. The ‘carry’ output is 1 only when both inputs are 1 and ‚sum’ is 1 if any one input is 1.

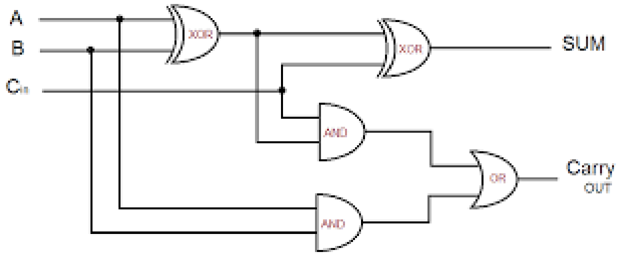


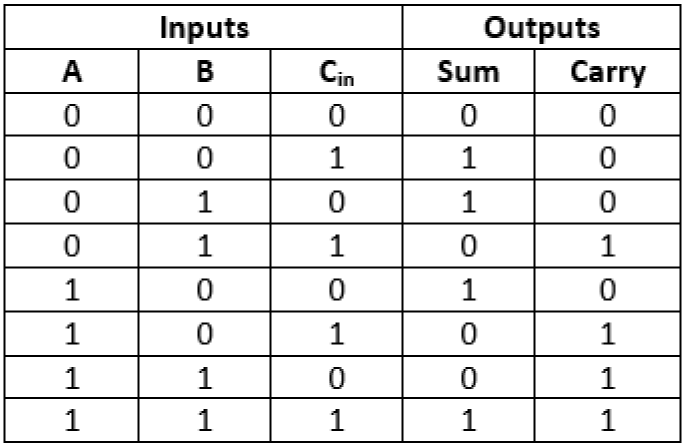
Inputs		Outputs	
A	B	SUM	CARRY
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



FULL ADDER:

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





Inputs			Outputs	
A	B	C_{in}	Sum	Carry
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:

VERILOG CODE FOR BASIC LOGIC GATES

```
module basic1(a,b,YAND,YOR,YNOT,YXOR,YXNOR);
input a,b;
output YAND,YOR,YNOT,YXOR,YXNOR;
assign YAND = a & b;
assign YOR = a | b;
assign YNOT =~ a ;
assign YXOR = a ^ b;
assign YXNOR = ~(a ^ b);
endmodule
```

VERILOG CODES FOR MULTIPLEXER

```
module mux(a,b,s1,out)
input a,b,s1;
output out;
reg out;
always@(a,b,s1)begin
if(~s1)
```

```
out=a;
else
out=b;
end
 endmodule
```

VERILOG CODES FOR DEMULTIPLEXER

```
module demux(D,select,y2,y1);
input D, select;
output y2,y1;
reg y2,y1;
always@(D or  select)begin
if(select==1’b0)
begin
y2=d;y1=1’b0;
end
else
begin
y2=1’b0;
y1=D;
end
end
endmodule
```

VERILOG CODE FOR HALF ADDER

```
module halfadder(a, b, sum, carry);
 input a;
input b;
output sum;
output carry;
assign sum = x ^ y ;
```

```
assigncarry = x & y
 endmodule
```

VERILOG CODE FOR FULL ADDER

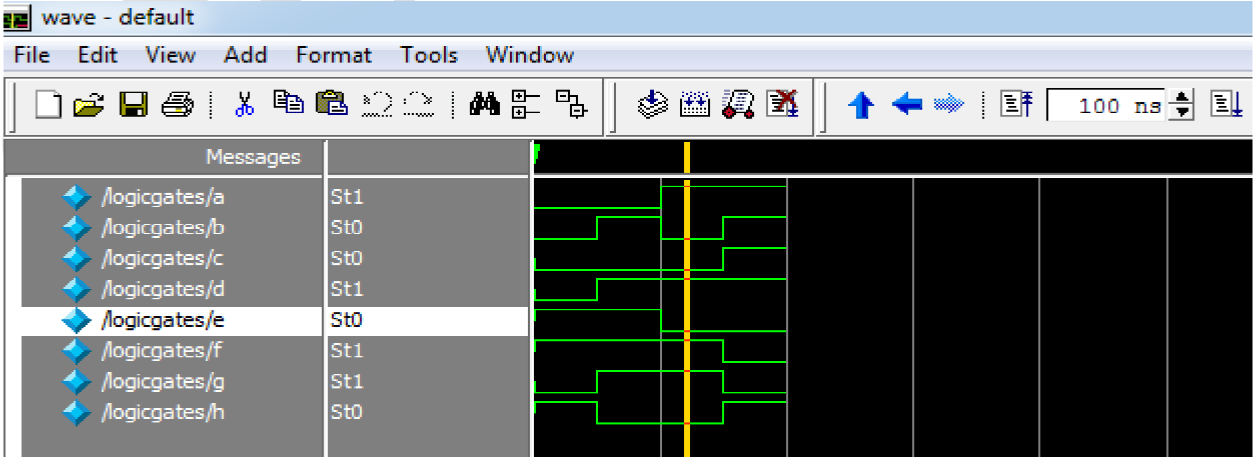
```
module fulladder(a,b,c,d,e,f,sum,carry);
 input a,b,c;
output sum,carry,d,e,f;
assign sum = x ^ y ^ z;
assign carry= (x & y) | (y & z) | (x & z);
 endmodule
```

PROCEDURE:

1. Click on the Xilinx ISE Design Suite 12.1or Xilinx Project navigator icon on the desktop ofPC.
2. Write the Verilog code by choosing HDL as top level source module.
3. Check syntax, view RTL schematic and note the device utilization summary by double
4. clicking on the synthesis in the process window.
5. Perform the functional simulation using Xilinx ISE simulator.
6. The output wave form can be observed in model sim.

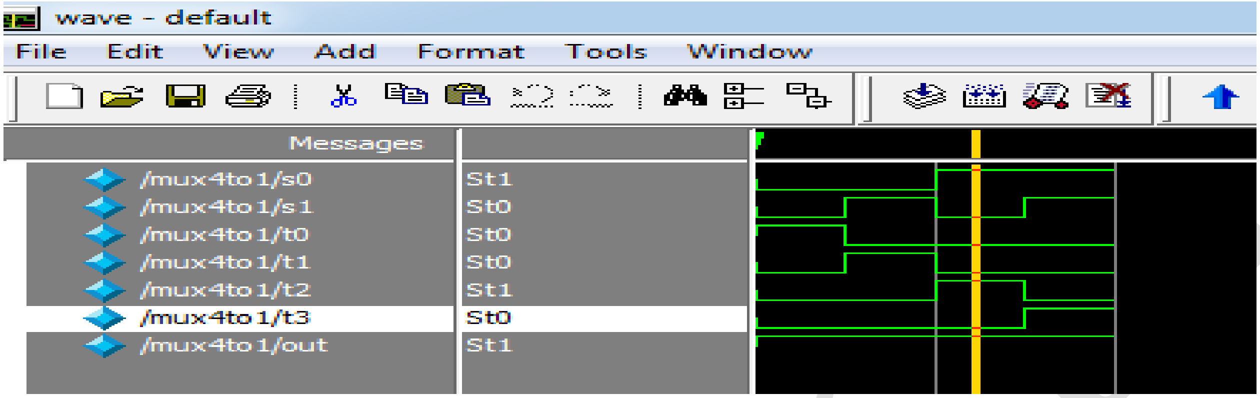
SIMULATION

LOGIC GATES



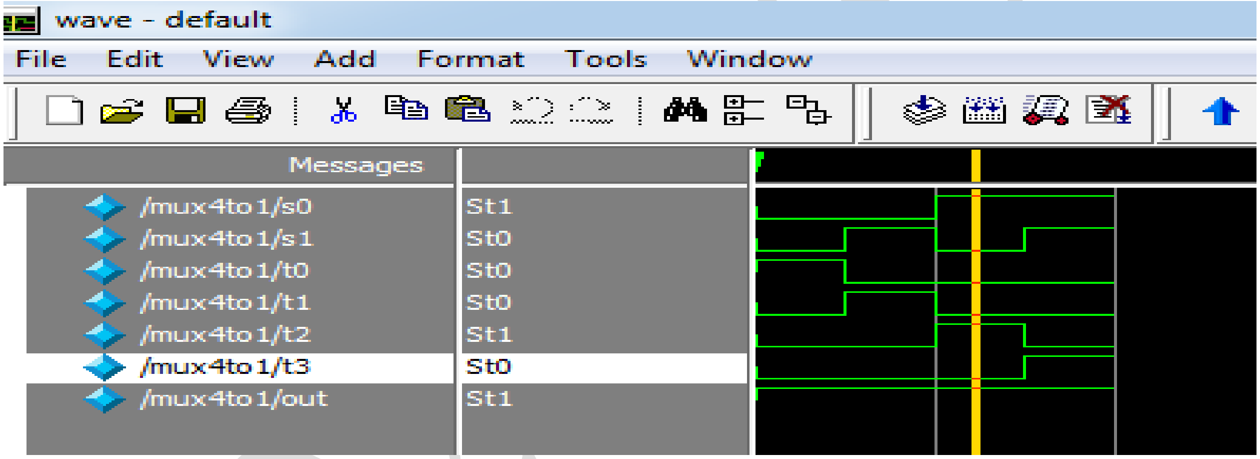
```
/logicgates/a   St1
/logicgates/b   St0
/logicgates/c   St0
/logicgates/d   St1
/logicgates/e   St0
/logicgates/f   St1
/logicgates/g   St1
/logicgates/h   St0
```

MULTIPLEXER:



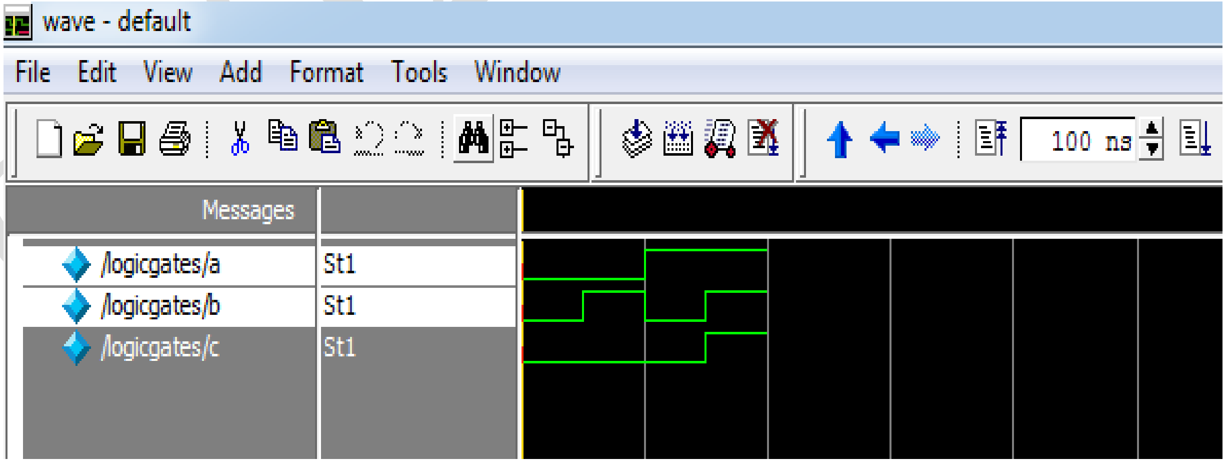
```
/mux4to1/s0    St1
/mux4to1/s1    St0
/mux4to1/t0    St0
/mux4to1/t1    St0
/mux4to1/t2    St1
/mux4to1/t3    St0
/mux4to1/out   St1
```

DEMULTIPLEXER



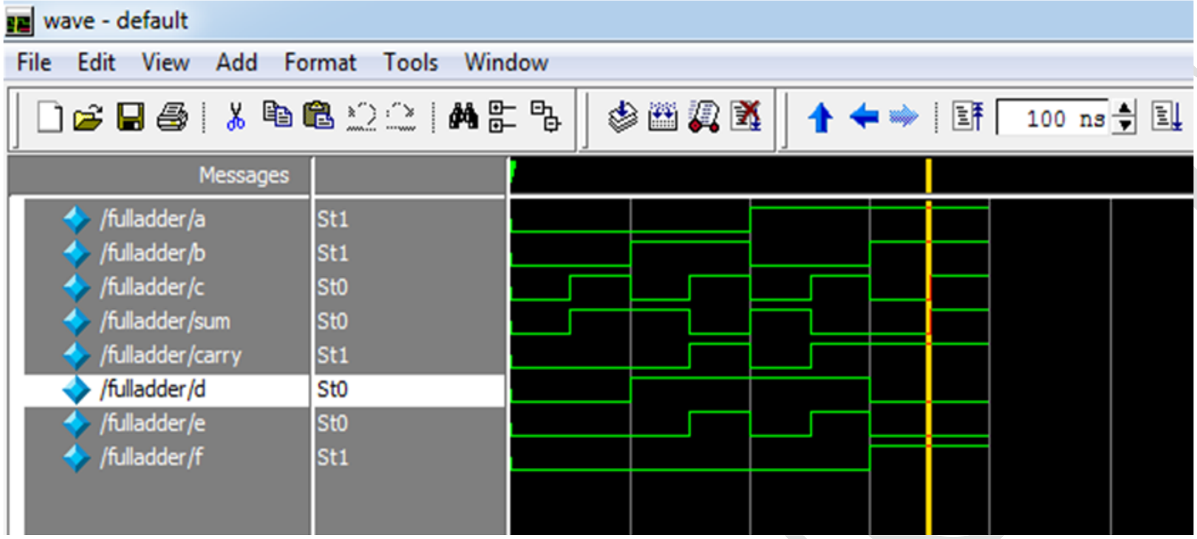
```
/mux4to1/s0    St1
/mux4to1/s1    St0
/mux4to1/t0    St0
/mux4to1/t1    St0
/mux4to1/t2    St1
/mux4to1/t3    St0
/mux4to1/out   St1
```

HALF ADDER



```
/logicgates/a   St1
/logicgates/b   St1
/logicgates/c   St1
```

FULL ADDER



Messages	
/fulladder/a	St1
/fulladder/b	St1
/fulladder/c	St0
/fulladder/sum	St0
/fulladder/carry	St1
/fulladder/d	St0
/fulladder/e	St0
/fulladder/f	St1

POST LAB QUESTIONS:

1. What is meant by combinational circuits?
2. A logic circuit requires HIGH on all its inputs to make the output HIGH. What type of logic circuit is it?
3. A basic 2-input logic circuit has a HIGH on one input and a LOW on the other input, and the output

   is HIGH. What type of logic circuit is it?
4. Develop the truth table for a 3-input AND gate and also determine the total number of possible combinations for a 4-input AND gate.
5. What is the use of half adder and full adder?

RESULT:

Verilog code for the Combinational circuits for verification is written, the waveform is observed and the code is synthesized with the technological library and is verified

EX.NO.1B DESIGN OF BASIC SEQUENTIAL CIRCUITS USING HDL AND IMPLEMENT XILINX SOFTWARE

AIM:

To write verilog code for flip flop circuit for verification, observe the waveform and synthesize the code with technological library with given Constraints.

TOOL REQUIRED:

1. Xilinx ISE Design Suite 8.1i
2. PC

PRE-LAB QUESTIONS :

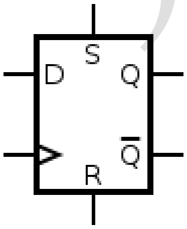
1. What is called Sequential circuits?
2. What is flip-flop?
3. How many types of flip-flop are used?
4. D flip-flop is used for?
5. What is full form of T flip-flop?

THEORY:

In electronics, a flip-flop or latch is a circuit that has two stable states and can be used to store state information. A flip-flop is a bistable multivibrator. The circuit can be made to change state by signals applied to one or more control inputs and will have one or two outputs. It is the basic storage element in sequential logic. Flip-flops and latches are fundamental building blocks of digital electronics systems used in computers, communications, and many other types of systems.

Flip-flops and latches are used as data storage elements. A flip-flop stores a single bit (binary digit) of data; one of its two states represents a "one" and the other represents a "zero". Such data storage can be used for storage of state, and such a circuit is described as sequential logic.

D FLIP-FLOP



The D flip-flop is widely used. 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.

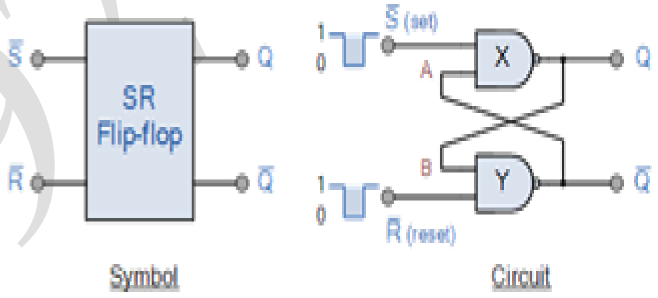
TRUTH TABLE:

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

PROGRAM :

```
module dff1(d,clk,rst,q,qb); input d,clk,rst;
output q,qb; reg q;
always @ (posedge clk) begin
if(rst) q<=1’b0;
else q<=d; end
assign qb=~q; endmodule
```

SK-FLIPFLOP:



TRUTH TABLE

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

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. The result is that the flip-floplooses control of Q and Q, and if the two inputs are now switched “HIGH” again after this condition to logic “1”, the flip-flop becomes unstable and switches to an unknown data state based upon the unbalance as shown in the following switching diagram.

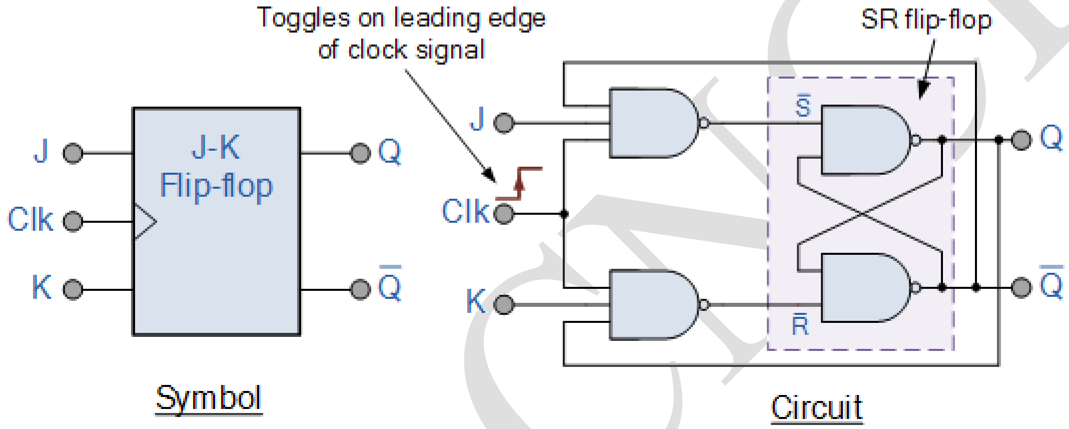
PROGRAM :

```
module srff(s,r,clk,rst,q,qb); input s,r,clk,rst;
output q,qb;
wire s,r,clk,rst,qb; reg q;
always @ (posedge clk) begin
if(rst) q<=1’b0;
else
if (s==1’b0 && r==1’b0) q<=q;
else if (s==1’b0 && r==1’b1) q<=1’b0;
else if (s==1’b1 && r==1’b0) q<=1’b1;
else if (s==1’b1 && r==1’b1) q<=1’bx;
end
```

```
assign qb=~q;
endmodule
```

JK FLIPFLOP:

Then 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. This results in the JK flip flop acting more like a T-type toggle flip-flop when both terminals are "HIGH".



Although this circuit is an improvement on the clocked SR flip-flop it still suffers from timing problems called "race" if the output Q changes state before the timing pulse of the clock input has time to go "OFF". To avoid this the timing pulse period (T) must be kept as short as possible (high frequency). As this is sometimes not possible with modern TTL IC's the much improved Master- Slave JK Flip-flop was developed.

TRUTH TABLE:

	Input		Output		Description
	J	K	Q	Q	
	0	0	0	0	Memory no change
same as	0	0	0	1	

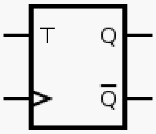
for the SR Latch	0	1	1	0	Reset Q » 0
	0	1	0	1	
	1	0	0	1	Set Q » 1
	1	0	1	0	
toggle action	1	1	0	1	Toggle
	1	1	1	0	

PROGRAM :

```
Module jkff (j,k,clk,rst,q,qb); Input j,k,clk,rst;
Output q,qb;
Wire j,k,clk,rst,qb;
Reg q;
Always @ (posedge clk)
Begin if(rst) q<=1’b0;
else if (j==0 && k==0) q<=q;
else if (j==0 && k==1) q<=1’b0;
else if (j==1 &&k==0) q<=1’b1;
else If (j==1 &&kr==1) q<=~q;
end
assign qb=~q;
endmodule
```

T FLIP-FLOP

T flip-flop



If the T input is high, the T flip-flop changes state ("toggles") whenever the clock input is strobe. If the T input is low, the flip-flop holds the previous value. This behavior is described by the characteristic equation:

TRUTH TABLE:

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

When T is held high, the toggle flip-flop divides the clock frequency by two; that is, if clock frequency is 4 MHz, the output frequency obtained from the flip-flop will be 2 MHz. This "divideby" feature has application in various types of digital counters. A T flip-flop can also be built using a JK flip-flop (J & K pins are connected together and act as T) or a D flip-flop (T input XORQ$_{previous}$ drives the D input).

PROGRAM :

```
module Tff (T,clk,rst,q,qb); input T,clk,rst;
output q,qb;
wire T,clk,rst,qb; reg tq;
always @ (posedge clk) begin
if(rst) tq<=1’b0; else begin
if(T==1’b1)
tq<=~tq; end
end
assign q=~tq; assign qb=~q;
endmodule
```

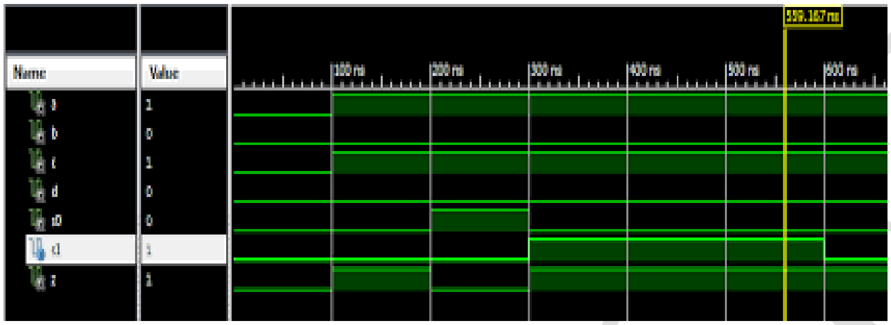
PROCEDURE:

1. Click on the Xilinx ISE Design Suite 12.1or Xilinx Project navigator icon on the desktop of PC.
2. Write the Verilog code by choosing HDL as top level source module.
3. Check syntax, view RTL schematic and note the device utilization summary by double
4. clicking on the synthesis in the process window.

5. Perform the functional simulation using Xilinx ISE simulator.
6. The output wave form can be observed in model sim.

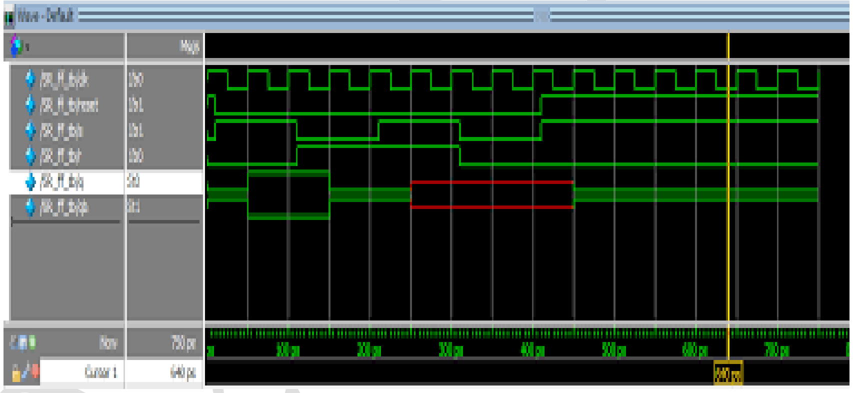
SIMULATION OUTPUT

D FLIPFLOP:



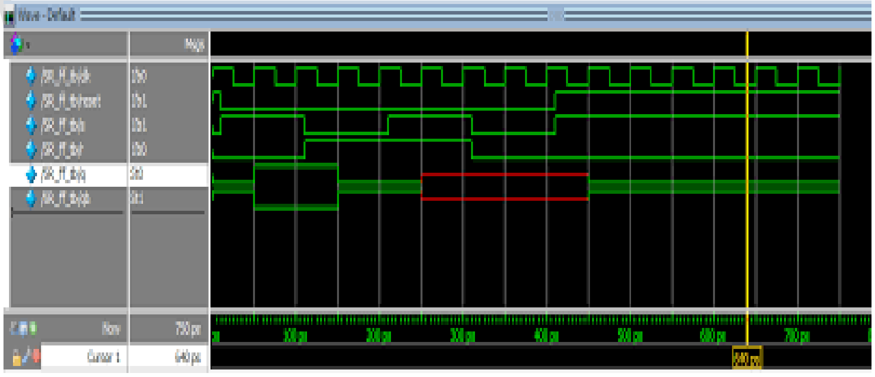
SIMULATION OUTPUT

SR FLIPFLOP:



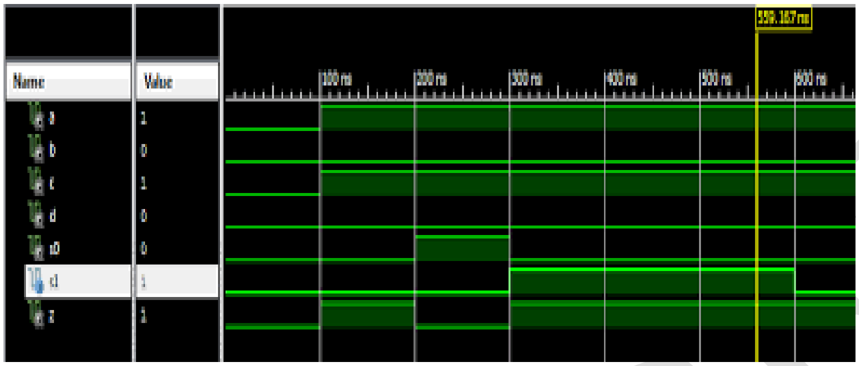
SIMULATION OUTPUT

JK FLIPFLOP:



SIMULATION OUTPUT

T FLIPFLOP:



POST LAB QUESTIONS:

1. What is disadvantage of SR flip-flop?
2. What is advantage of SR flip-flop?
3. What is disadvantage of JK flip-flop?
4. Which Gates are used in SR flip flops to a JK flip-flop?
5. Differentiate synchronous and asynchronous signal?

RESULT:

Verilog code for the flip-flop circuit verification is written, the waveform is observed and the code is synthesized with the technological library and is verified.

EX.NO.2 DESIGN AN 8 BIT ADDER ; 8 BIT MULTIPLIER USING XILINX SOFTWARE

AIM

To write a verilog program for basic 8bit adder and 8 bit multiplier to synthesize and simulate using Xilinx software tool.

TOOLS REQUIRED:

1. Xilinx ISE Design Suite 8.1i
2. PC

PRELAB QUESTIONS :

1. How does an 8-bit adder differ from a 4-bit adder?
2. What logic gates are used to implement an 8-bit adder?
3. What is the use of 8 bit multiplier?
4. How does an 8-bit multiplier differ from an adder?
5. What types of logic gates are used in 8-bit multiplier?

THEORY:

8 BIT ADDER :

When you add large numbers carefully together the addition is done digit by digit. The counter does the same. In the illustration, two 8 –digit binary numbers are being added. The top row contains the first number and the second row the other. Working from the right-hand side, therecan be no 'carry' to add to the sum of the first two digits, so a half adder is sufficient. But for the second and subsequent pairs of digits, full adders must be use (any carry' is indicated by a f below the adder). The output will be an 8-bit and if the carry is formed that will be shown in cout output value.

8 BIT MULTIPLIER :

Multiplication of two elements in the polynomial basis can be accomplished in the normalway of multiplication, but there are a number of ways to speed up multipication, especially in hardware. In this type the multiplication can done parallel counter and it is generate carry. The multiplication is independent of the carry so we can perform N number of multiplication independent of carry.

PROGRAM:

VERILOG CODE FOR 8 BIT ADDER

```
module adderm(a, b, sum, carry);
input [7:0] a;
input [7:0] b;
output [7:0] sum;
 output carry;
wire[8:0]temp;
assign temp=(a+b);
assign sum=temp[7:0];
assign carry=temp[8];
endmodule
```

VERILOG CODE FOR 8 BIT MULTIPLIER

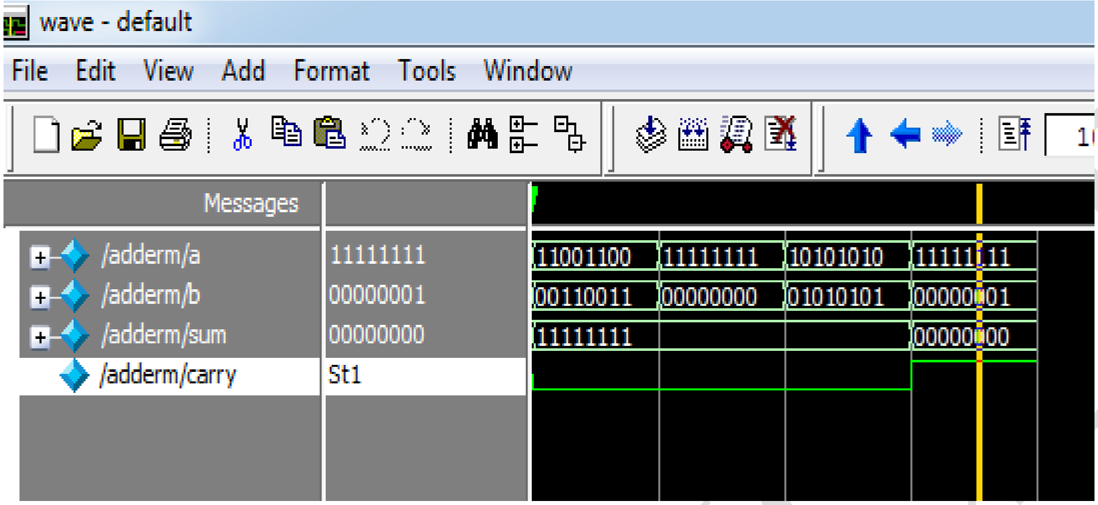
```
module multiplierm(a, b, out);
 input [4:0] a;
input [4:0] b;
output [9:0] out;
assign out=(a*b);
endmodule
```

PROCEDURE:

1. Click on the Xilinx ISE Design Suite 12.1or Xilinx Project navigator icon on the desktop of PC.
2. Write the Verilog code by choosing HDL as top level source module.
3. Check syntax, view RTL schematic and note the device utilization summary by double
4. clicking on the synthesis in the process window.
5. Perform the functional simulation using Xilinx ISE simulator.
6. The output wave form can be observed in model sim.

SIMULATION OUTPUT

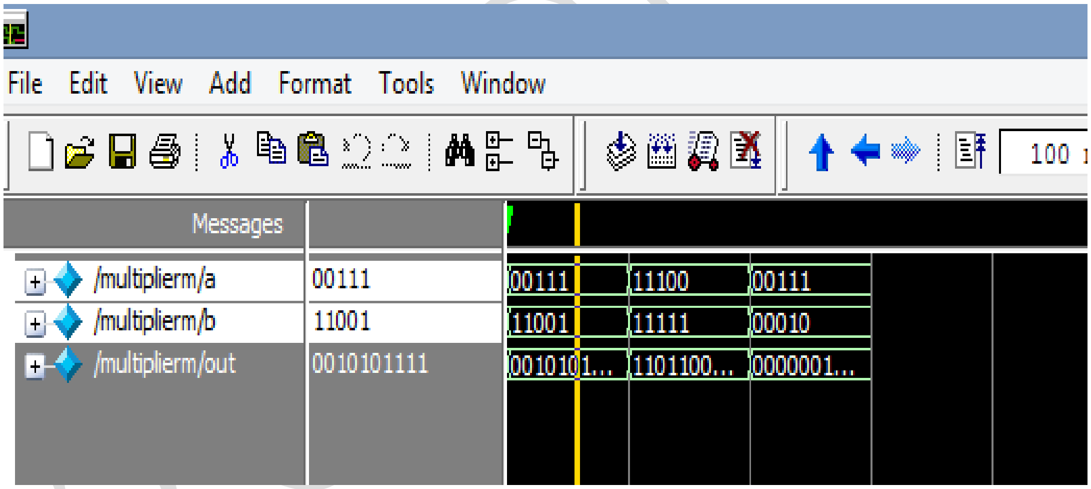
8BIT ADDER:



```
/adderm/a      11111111   11001100  11111111  10101010  11111111
/adderm/b      00000001   00110011  00000000  01010101  00000001
/adderm/sum    00000000   11111111                      00000000
/adderm/carry  St1
```

SIMULATION OUTPUT

8BIT MULTIPLIER:



```
/multiplierm/a    00111       00111       11100       00111
/multiplierm/b    11001       11001       11111       00010
/multiplierm/out  0010101111  0010101...  1101100...  0000001...
```

POST LAB QUESTIONS

1. What challenges did you encounter during the design of the multiplier?

2. How did the performance of the adder compare to that of the multiplier?

3. What did you learn from the simulation results of both circuits?

4. What were the key challenges faced during the design of the adder?

5. What performance metrics did you analyze for both the adder and multiplier?

RESULT:

Verilog code for the 8bit adder and 8bit multiplier circuit verification is written, the waveform is observed and the code is synthesized with the technological library and is verified.

EX.NO.3 DESIGN AND IMPLEMENTATION OF UNIVERSAL SHIFT REGISTERS USING XILINX SOFTWARE

AIM

To write a Verilog program for basic Combinational logic circuit to synthesize and simulate using Xilinx software tool.

TOOLS REQUIRED:

1.Xilinx ISE Design Suite 8.1i

2. PC

POST LAB QUESTIONS :

1. What is a universal shift register?
2. How does a universal shift register differ from a regular shift register?
3. What are the main components of a universal shift register?
4. Can you explain the shift operations in detail?
5. What control signals are necessary for the operation of a universal shift register?

THEORY:

UNIVERSAL SHIFT REGISTER:

A Unidirectional shift register is a register that can capable of transferring data in only one direction. Whereas the register that is capable of transferring data in both left and right direction is called a „bidirectional shift register.‟ Now let we have a register which can capable to transfer data in both the shift-right and shift- left, along with the necessary input and output terminals for parallel transfer, then it is called a *shift register* with *parallel load* or „universal shift register.‟

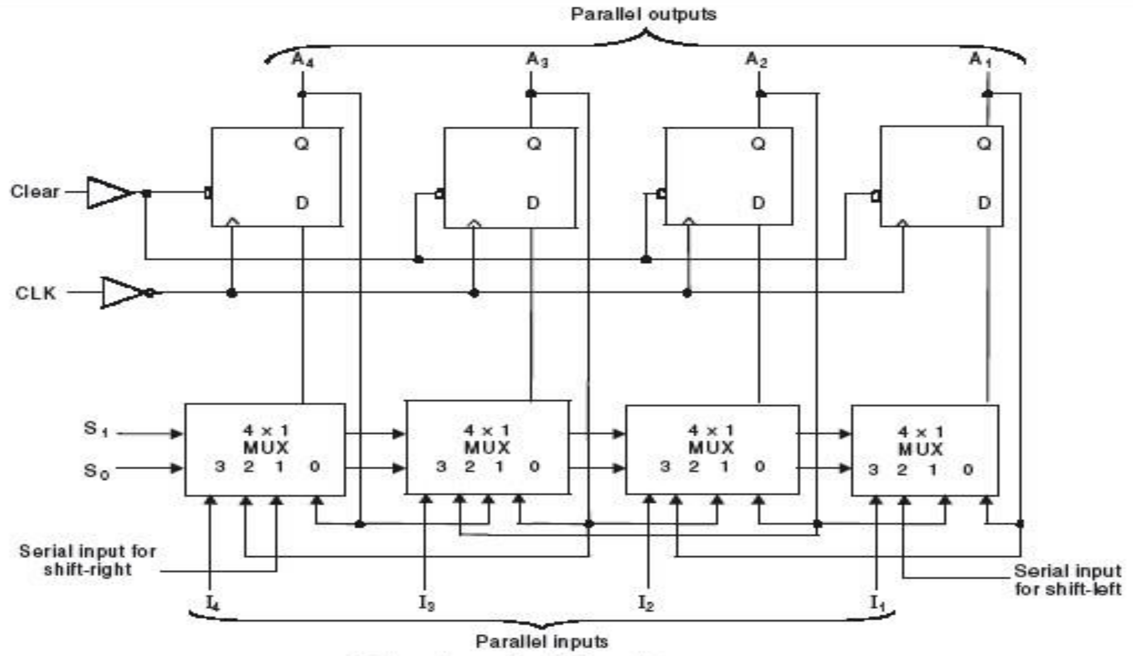
A shift-right control to enable the shift-right operation and the serial input and output lines associated with the shift-right. A shift- left control to enable the shift- left operation and the serial input and output lines associated with the shift- left.

A parallel- load control to enable a parallel transfer and the *n* input lines associated with the parallel transfer. *n* parallel output lines.

A *clear* control to clear the register to 0. A *CLK* input for clock pulses to synchronize all operations.

A control state that leaves the information in the register unchanged even though clock pulses are continuously applied.

LOGIC DIAGRAM:



4-bit universal shift register.

TRUTH TABLE:

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

PROCEDURE:

1. Click on the Xilinx ISE Design Suite 12.1or Xilinx Project navigator icon on the desktop of PC.

2. Write the Verilog code by choosing HDL as top level source module.

3. Check syntax, view RTL schematic and note the device utilizatio n summary by double Clicking on the synthesis in the process window.

4. Perform the functional simulation using Xilinx ISE simulator.

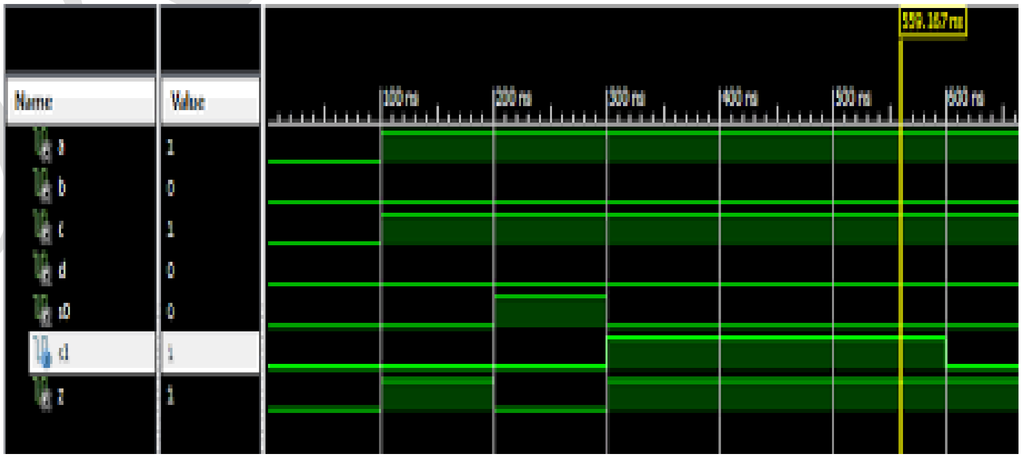
5. The output can be observed by using ISIM Simulator.

PROGRAM:

```
module universal shift(a,s,clk,p);
 input [3:0]a;
 input [1:0]s;
 input clk;
output reg [3:0]p;
initial
p<=4'b0110; always@(posedge clk)
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
```

SIMULATION OUTPUT

SHIFT REGISTER



Name	Value
a	1
b	0
c	1
d	0
s0	0
s1	1
z	1

POST LAB QUESTIONS :

1. What applications can benefit from using a universal shift register?
2. What are the potential pitfalls in designing a universal shift register?
3. How would you explain the importance of a universal shift register to someone unfamiliar with digital design?
4. How does a universal shift register interface with other digital components?
5. What enhancements can be made to improve the functionality of the universal shift register?

RESULT:

Verilog code for the universal shift registers circuit verification is written, the waveform is observed and the code is synthesized with the technological library and is verified

EX.NO.4 DESIGN MEMORIES USING HDL AND SIMULATE USING XILINX SOFTWARE

AIM

To write a verilog program for Memories using HDL and simulate using Xilinx software tool.

TOOLS REQUIRED:

1.Xilinx ISE Design Suite8.1i

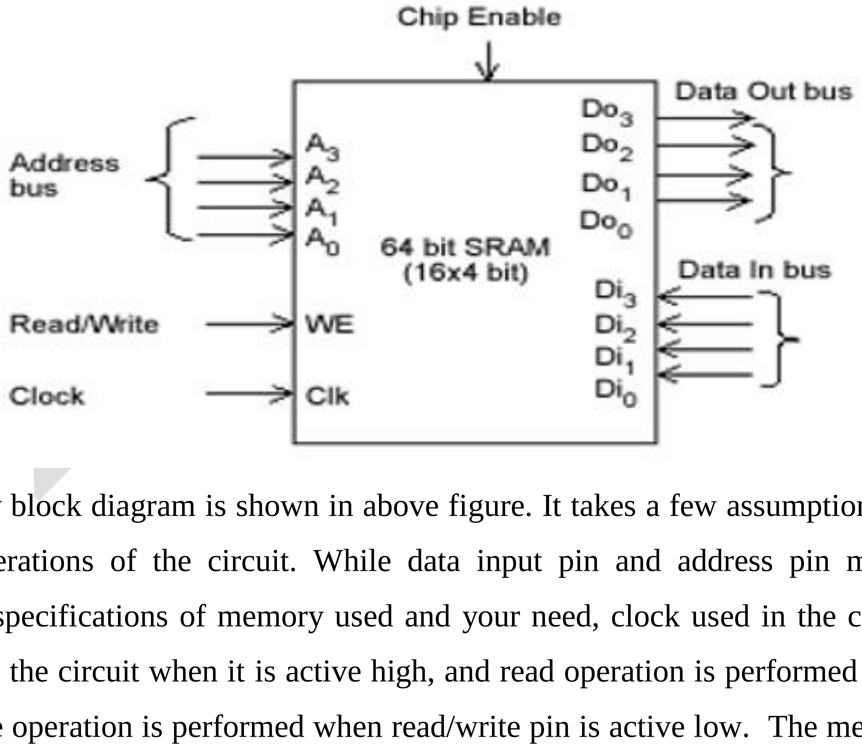
2.PC

PRE-LAB QUESTIONS :

1. What is memory in the context of VLSI design?
2. What are the main types of memory used in VLSI circuits?
3. What is the difference between volatile and non-volatile memory?
4. What is a memory cell, and how is it constructed in SRAM and DRAM?
5. What is SRAM, and how does it differ from other types of memory?

THEORY:

SRAM MEMORY :



The memory block diagram is shown in above figure. It takes a few assumptions into consideration for easing the operations of the circuit. While data input pin and address pin may have any value depending on the specifications of memory used and your need, clock used in the circuit is active high. Enable pin triggers the circuit when it is active high, and read operation is performed when read/write pin is high, while write operation is performed when read/write pin is active low. The memory block diagram is shown in above figure. It takes a few assumptions into consideration for easing the operations of the circuit. While data input pin and address pin may have any value depending on the specifications of memory used and your need, clock used in the circuit is active high. Enable pin triggers the circuit when it is active high, and read operation is performed when read/write pin is high, while write operation is performed when read/write pin is active low.

PROCEDURE:

1. Click on the Xilinx ISE Design Suite 12.1or Xilinx Project navigator icon on the desktop of PC.

2. Write the Verilog code by choosing HDL as top level source module.

3. Check syntax, view RTL schematic and note the device utilizatio n summary by double Clicking on the synthesis in the process window.

4. Perform the functional simulation using Xilinx ISE simulator.

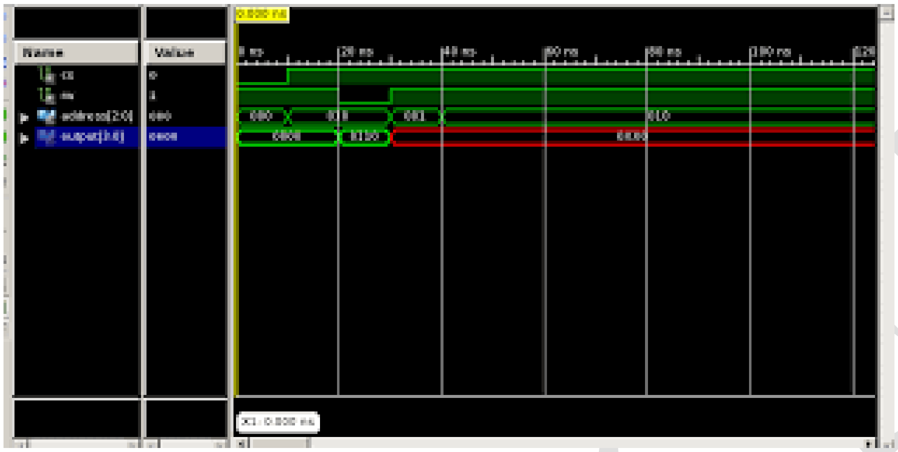
5. The output can be observed by using ISIM Simulator.

PROGRAM:

```
module memory_16x4(op,ip,rd_wr,clk,address);
output reg [3:0] op; input [3:0] ip; input [3:0]
address; input rd_wr,clk; reg [3:0]
memory[0:15]; always @(posedge clk) begin if
(rd_wr) op=memory[address]; else
begin
memory[address]=ip;
end
end endmodule
```

SIMULATION OUTPUT

MEMORY



POST LAB QUESTIONS :

1. How did you optimize the area for your 16x4 SRAM design?
2. What is the function of cache memory in a computer system?
3. Can you differentiate between static and dynamic memory?
4. What is memory latency, and why is it important?
5. What are the trade-offs involved in choosing between SRAM and DRAM for a specific application?

RESULT:

Verilog code for the memories using HDL verification is written, the waveform is observed and the code is synthesized with the technological library and is verified

EX.NO.5 DESIGN FINITE STATE MACHINE (MOORE ND MELAY) USING HDL AND SIMULATE USING XILINX SOFTWARE

AIM

To write a Verilog program for finite state machine (Moore and Melay) using HDL and simulate using Xilinx software tool.

TOOLS REQUIRED:

1. Xilinx ISE Design Suite 8.1i
2. PC

PRE- LAB QUESTIONS :

1. What is Finite state Machine?
2. What are the different types of FSMs?
3. Explain Moore and Mealy machines and the differences between them.
4. How do you represent an FSM using a state diagram and state table?
5. What are the real-world applications of FSMs?

THEORY:

MOORE MEALY MODEL:

The analysis of a sequential circuit consists of obtaining a table or a diagram for the time sequence of inputs, outputs, and internal states. It is also possible to write Boolean expressions that describe the behavior of the sequential circuit. A logic diagram is recognized as a clocked sequential circuit if it includes flip-flops with clock inputs. The flip- flops may be of any type, and the logic diagram may or may not include combinational logic gates.

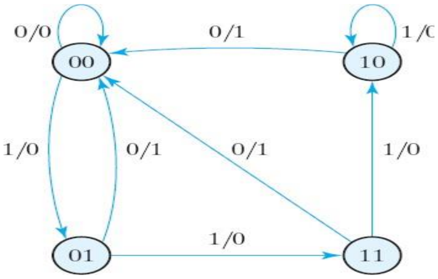
State Equations: A *state equation* (also called a *transition equation)* specifies the next state as a function of the present state and inputs.

$$A(t + 1) = A(t)x(t) + B(t)x(t) \; ; \; B(t + 1) = A'(t)x(t); \; y(t) = [A(t) + B(t)]x'(t)$$

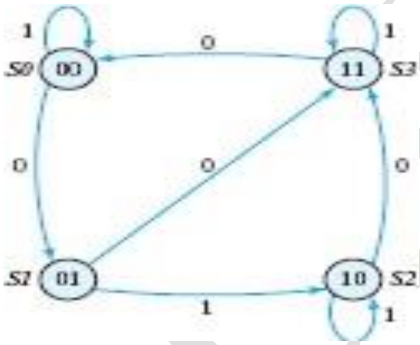
A state equation is an algebraic expression that specifies the condition for a flip- flop state transition. The left side of the equation, with (t + 1), denotes the next state of the flip- flop one clock edge later. The right side of the equation is a Boolean expression that specifies the present state and input conditions that make the next state equal to 1. Since all the variables in the Boolean expressions are a function of the present state, we can omit the designation (t) after each variable for convenience and can express the state equations in the more compact form

$$A(t + 1) = Ax + Bx \; ; \; B(t + 1) = A'x \; ; \; y = Ax' + Bx'$$

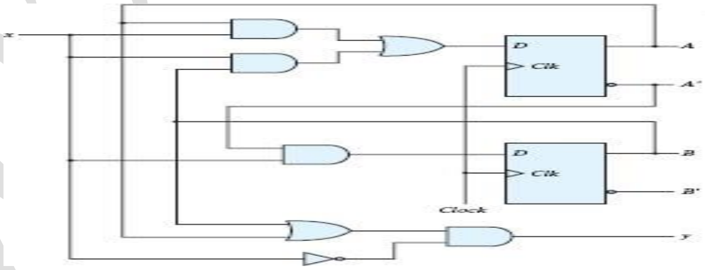
STATE DIAGRAM OF MELAY MODEL



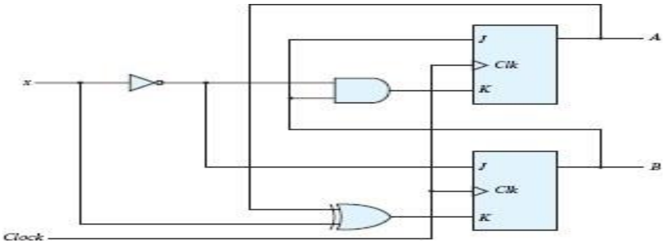
STATE DIAGRAM OF MOORE MODEL



LOGIC DIAGRAM OF MELAY MODEL:



LOGIC DIAGRAM OF MOORE MODEL



TRUTH TABLE OF MELAY MODEL:

A	B	x	A	B	J_A	K_A	J_B	K_B
0	0	0	0	1	0	0	1	0
0	0	1	0	0	0	0	0	1
0	1	0	1	1	1	1	1	0
0	1	1	1	0	1	0	0	1
1	0	0	1	1	0	0	1	1
1	0	1	1	0	0	0	0	0
1	1	0	0	0	1	1	1	1
1	1	1	1	1	1	0	0	0

TRUTH TABLE OF MOORE MODEL:

state		Present input	Next state		Output
A	B	x	A	B	y
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	1	0	0
1	1	0	0	0	1
1	1	1	1	0	0

PROCEDURE:

1. Click on the Xilinx ISE Design Suite 12.1or Xilinx Project navigator icon on the desktop of PC.
2. Write the Verilog code by choosing HDL as top level source module.
3. Check syntax, view RTL schematic and note the device utilizatio n summary by double Clicking on the synthesis in the process window.
4. Perform the functional simulation using Xilinx ISE simulator.
5. The output can be observed by using ISIM Simulator

PROGRAM FOR MEALY MODEL:

```
module Mealy_model(y, x, clk, reset);
input x,clk,reset;
output reg y;
reg [1:0] state, next_state;
parameter S0 = 2'b00, S1 = 2'b01, S2 = 2'b10, S3 = 'b11;
 always @ ( posedge clk) if (reset == 0) state <= S0;
else state <= next_state;
 always @ (state, x) case (state)
S0: if (x) next_state = S1;
else next_state = S0;
S1: if (x) next_state = S3; else next_state = S0;
S2: if (x) next_state = S2; else next_state = S0;
S3: if (x) next_state = S2; else next_state = S0;
endcase always @ (state, x) case (state) S0: y =
0; S1, S2, S3: y = ~x; endcase endmodule
```

PROGRAM FOR MOORE MODEL:

MOORE MODEL:

```
module Moore_Model(y, x, clk, reset);
input x,clk,reset;
output [1:0]y;
 reg [1:0] state;
```

```
parameter S0 = 2'b00, S1 = 2'b01, S2 = 2'b10, S3 = 'b11;
always @ (posedge clk) if (reset == 0) state <= S0;
else case (state)
S0: if (x) state <= S0; else state <= S1;
S1: if (x) state <= S2; else state <= S3;
S2: if (x) state <= S2; else state <= S3;
S3: if (x) state <= S3; else state <= S0;
endcase
assign y=state;
endmodule
```

SIMULATION :

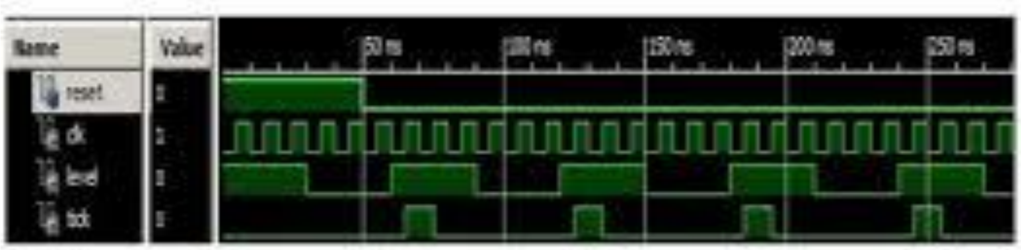


Figure 10: The Moore machine based edge detector Timing Diagram.

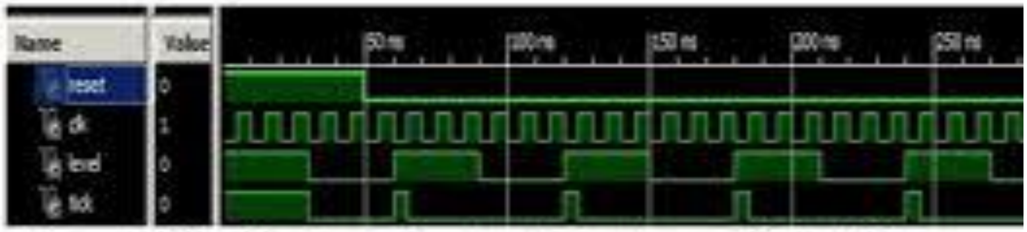


Figure 11. The Mealy machine based edge detector Timing Diagram.

POST LAB QUESTIONS :

1. What is state minimization, and why is it important?
2. What is a transition table in FSM design?
3. What is state encoding in FSM design?
4. What is the relationship between FSMs and sequential circuits?
5. Explain how FSMs can be implemented using flip-flops and combinational logic.

RESULT:

Verilog code for the Finite state machine (Moore/Melay) using HDL verification is written, the waveform is observed and the code is synthesized with the technological library and is verified

EX.NO.6 DESIGN SYNCHRONOUS UP/DOWN COUNTER USING HDL AND SIMULATE USING XILINX SOFTWARE

AIM

To write a Verilog program for synchronous up/down counter using HDL and simulate using Xilinx software tool.

TOOLS REQUIRED:

1.Xilinx ISE Design Suite 8.1i

2. PC

PRE LAB QUESTIONS :

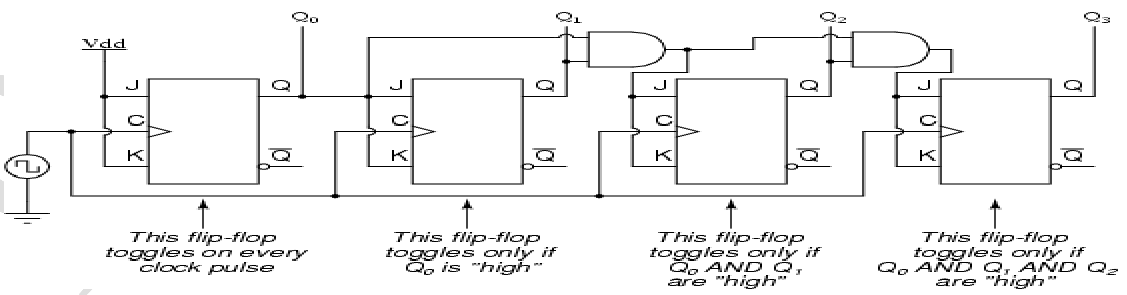
1. What is a Synchronous Counter?
2. What is an Up/Down Counter?
3. Compare the difference between Synchronous vs. Asynchronous Counters
4. How do changes in the clock signal affect the counting sequence?
5. What is the modulus of a counter, and how do you determine it for an up/down counter?

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 V_{dd}, 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

A four-bit synchronous "up" counter


Vdd
Q0
Q1
Q2
Q3
J
C
K
Q
Q̄
This flip-flop toggles on every clock pulse
This flip-flop toggles only if Q0 is "high"
This flip-flop toggles only if Q0 AND Q1 are "high"
This flip-flop toggles only if Q0 AND Q1 AND Q2 are "high"

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:

1. Click on the Xilinx ISE Design Suite 12.1or Xilinx Project navigator icon on the desktop of PC.
2. Write the Verilog code by choosing HDL as top level source module.
3. Check syntax, view RTL schematic and note the device utilizatio n summary by double Clicking on the synthesis in the process window.
4. Perform the functional simulation using Xilinx ISE simulator.
5. The output can be observed by using ISIM Simulator

PROGRAM :

```
module counter(clk,rst,count); input clk,rst;

output[3:0] count; reg [3:0] count;

always@(posedge.clk) begin

if(rst) count<=4'b0000; else

count<=count+4'b0001; end

endmodule
```

POST LAB QUESTIONS :

1. What are the design challenges associated with implementing large synchronous up/down counters?
2. How do you handle the propagation delay in synchronous counters?
3. What are the practical applications of synchronous up/down counters?
4. Which ICs are commonly used for synchronous up/down counters?
5. What is clock skew, and how does it impact the performance of a synchronous up/down counter?

RESULT:

Verilog code for the synchronous counter circuit for verification is written, the waveform is observed and the code is synthesized with the technological library and is verified.

EX.NO.7 DESIGN 4 BIT ASYNCHRONOUS UP/DOWN COUNTER USING HDL AND SIMULATE USING XILINX SOFTWARE

AIM :

To write Verilog code for asynchronous counter circuit for verification, observe the waveform and synthesize the code with technological library with given Constraints.

TOOLS REQUIRED :

1. Xilinx ISE Design Suite 8.1i
2. PC

PRE LAB QUESTIONS :

1. What is an Asynchronous Counter?
2. What are the typical applications of asynchronous up/down counters in digital electronics?
3. How do the delays accumulate in asynchronous counters?
4. Explain the state diagram for a 2-bit asynchronous up/down counter.
5. How are the flip-flops connected in an asynchronous up/down counter?

THEORY:

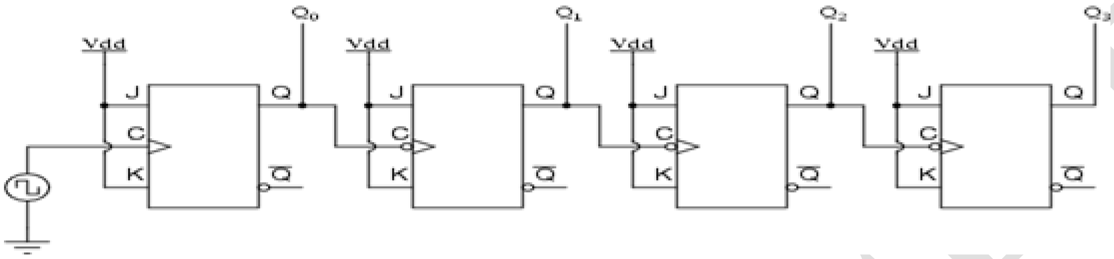
This circuit would yield the following output waveforms, when “clocked” by a repetitive source of pulses from an oscillator. A ripple counter is an asynchronous counter where only the first flip-flop is clocked by an external clock. All subsequent flip-flops are clocked by the output of the preceding flip-flop. Asynchronous Counters are also called ripple counter.

The MOD of the ripple counter or asynchronous counter is 2n if n flip-flops are used. For a 4-bit counter, the range of the count is 0000 to 1111 (24-1). A counter may count up or count down or count up and down depending on the input control. The count sequence usually repeats itself. When counting up, the count sequence goes from 0000, 0001, 0010, ... 1110 , 1111 , 0000, 0001, ... etc. When counting down the count sequence goes in the opposite manner: 1111, 1110, ... 0010, 0001, 0000, 1111, 1110, ... etc.

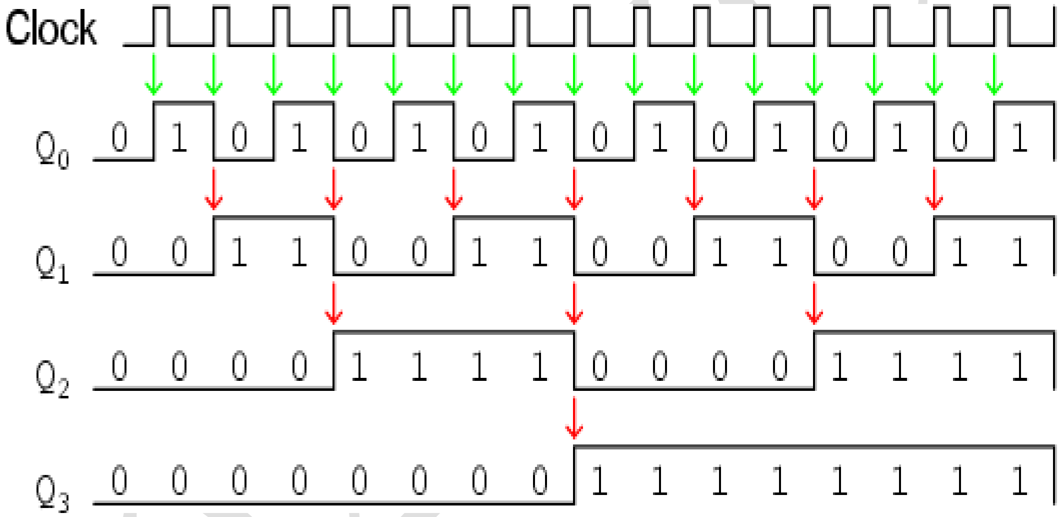
Four J-K flip-flops connected in such a way to always be in the “toggle” mode, we need to determine how to connect the clock inputs in such a way so that each succeeding bit toggles when the bit before it transitions from 1 to 0. The Q outputs of each flip-flop will serve as the respective binary bits of the final, four-bit count.

If we used flip-flops with negative-edge triggering (bubble symbols on the clock inputs), we could simply simply connect the clock input of each flip-flop to the Q output of the flip-flop before it, so that whenthe bit before it changes from a 1 to a 0, the "falling edge" of that signal would "clock" the next flip- flop to toggle the next bit.

LOGIC DIAGRAM



A four-bit "up" counter



PROCEDURE:

1. Click on the Xilinx ISE Design Suite 12.1or Xilinx Project navigator icon on the desktop of PC.
2. Write the Verilog code by choosing HDL as top level source module.
3. Check syntax, view RTL schematic and note the device utilizatio n summary by double Clicking on the synthesis in the process window.
4. Perform the functional simulation using Xilinx ISE simulator.
5. The output can be observed by using ISIM Simulator

PROGRAM :

```
module counter(clk,rst,count);
 input clk,rst;
output[3:0]count;
 reg[3:0]count;
always@(negedge clk)
if(rst) count[0]<=1’b0;
else
count[0]<=~count[0];
always @(negedge count[0]) if(rst)
count[1]<=1’b0;
else
count[1]<=~count[1];
always @(negedge count[1]) if(rst)
count[2]<=1’b0;
else
count[2]<=~count[2];
 always @(negedge count[2])
     if(rst)
    count[3]<=1’b0;
else
    count[3]<=~count[3];
 endmodule
```

POST LAB QUESTIONS :

1. How does the toggle control work for counting in both directions?
2. What is the role of the up/down control signal in configuring the flip-flops?
3. What is the counting sequence for a 2-bit or 3-bit asynchronous up/down counter?

4. How can you reset an asynchronous up/down counter to its initial state?
5. Where would you prefer an asynchronous counter over a synchronous one, and why?

RESULT:

Verilog code for the asynchronous counter circuit for verification is written, the waveform is observed and the code is synthesized with the technological library and is verified.

EX.NO.8 DESIGN AND SIMULATE A CMOS LOGIC GATES, FLIFLOPS & GENERATE LAYOUT

AIM :

To design, synthesize, simulate and implement the CMOS logic gates in layout form using Microwind.

TOOLS REQUIRED :

1. MICROWIND Tools
2. PC

PRE LAB QUESTIONS :

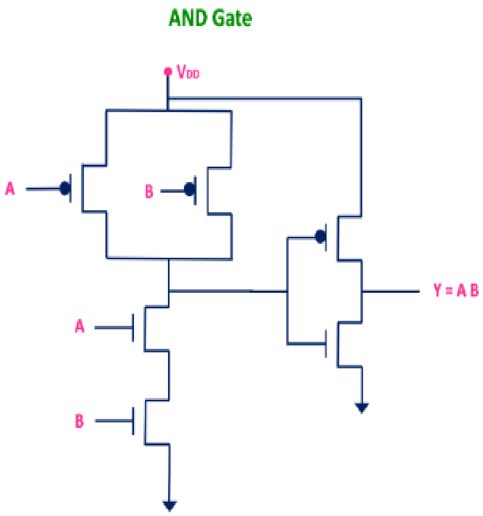
1. What is CMOS?
2. Why are CMOS circuits preferred over TTL (Transistor-Transistor Logic)?
3. What is meant by fan-in and fan-out?
4. What is a flip-flop?
5. Differentiate between latch and flip-flop.

THEORY

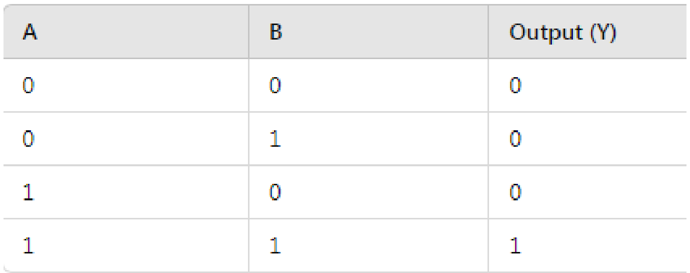
AND

A **CMOS AND gate** is a digital logic gate built using **Complementary Metal-Oxide-Semiconductor (CMOS)** technology, which combines both **nMOS** (n-channel MOSFETs) and **pMOS** (p-channel MOSFETs) transistors to implement the AND logic function. The AND gate outputs a high signal (1) only when all its inputs are high (1)

$$Y=A\cdot B$$



For a 2-input AND gate, the truth table is as follows:

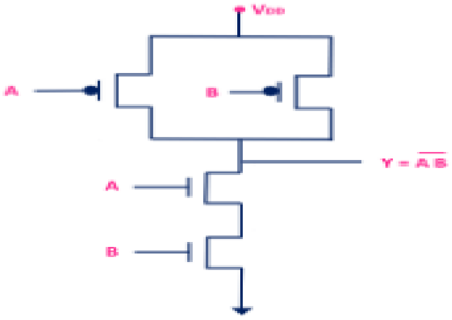


A	B	Output (Y)
0	0	0
0	1	0
1	0	0
1	1	1

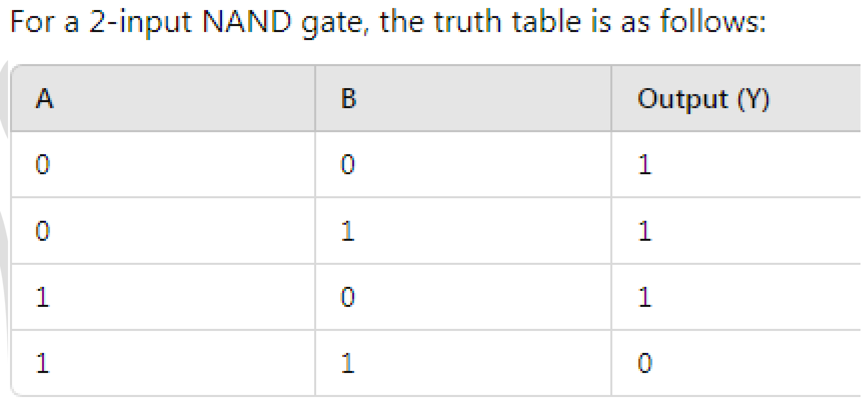
NAND

A **CMOS NAND gate** is a digital logic gate built using **Complementary Metal-Oxide-Semiconductor (CMOS)** technology to implement the **NAND function**. A NAND gate outputs a **low (0)** only when **all its inputs are high (1)**, and it outputs a **high (1)** for all other combinations of inputs.

$$Y = \overline{A \cdot B}$$



TRUTH TABLE:



For a 2-input NAND gate, the truth table is as follows:

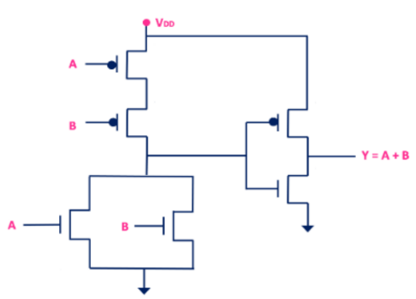
A	B	Output (Y)
0	0	1
0	1	1
1	0	1
1	1	0

OR

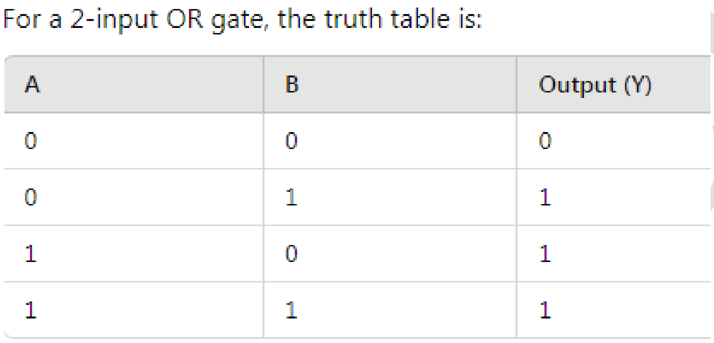
A **CMOS OR gate** is a digital logic gate that outputs a **high (1)** if **any** of its inputs are high, and outputs a **low (0)** only when **all inputs are low**. The **OR gate** is built using **Complementary Metal-Oxide-Semiconductor (CMOS)** technology, which uses both **nMOS** and **pMOS** transistors to implement the logic.

Y=A+BY = A + BY=A+B

OR Gate



TRUTH TABLE:



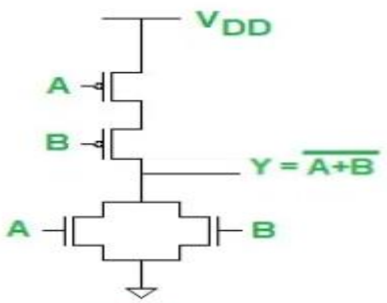
For a 2-input OR gate, the truth table is:

A	B	Output (Y)
0	0	0
0	1	1
1	0	1
1	1	1

NOR

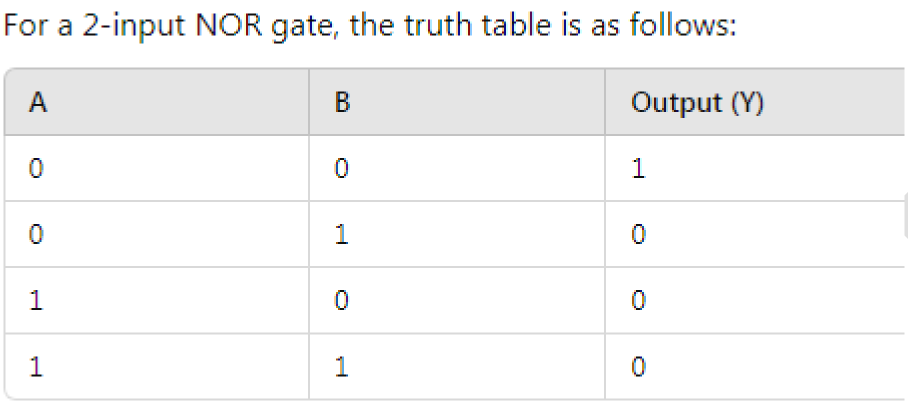
A **CMOS NOR gate** is a digital logic gate that implements the **NOR function** using **Complementary Metal-Oxide-Semiconductor (CMOS)** technology. The **NOR gate** outputs a **high (1)** only when **all inputs are low (0)**, and it outputs a **low (0)** if **any input is high (1)**. The NOR gate is the inverse of the OR gate.

$$Y = \overline{A + B}$$



NOR Gate

TRUTH TABLE:



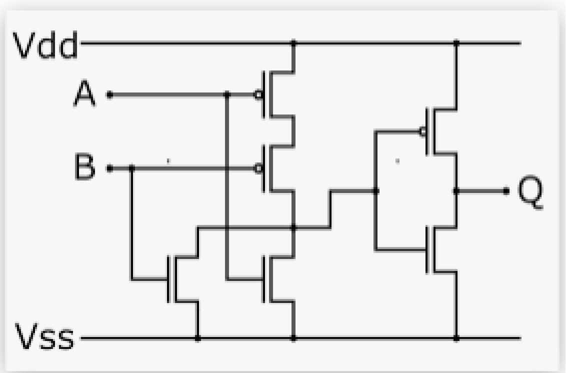
For a 2-input NOR gate, the truth table is as follows:

A	B	Output (Y)
0	0	1
0	1	0
1	0	0
1	1	0

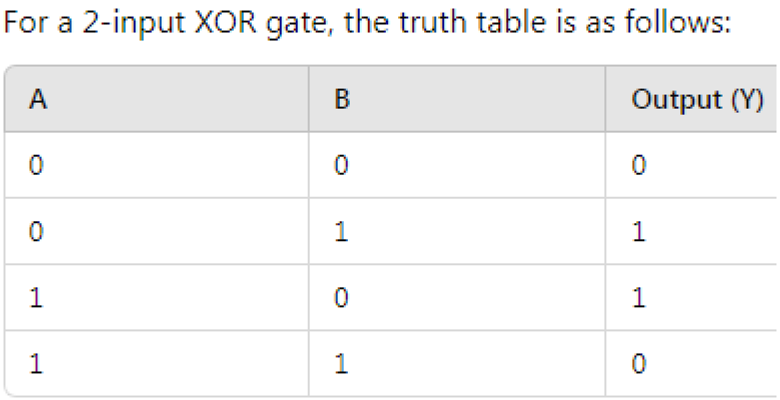
EXOR

A **CMOS XOR gate** (Exclusive OR gate) is a digital logic gate that outputs a **high (1)** when the inputs are **different**, and outputs a **low (0)** when the inputs are **the same**. It is widely used in arithmetic circuits, error detection, and digital systems where conditional logic is required.

$$Y = A \oplus B = A\overline{B} + \overline{A}B$$



TRUTH TABLE:



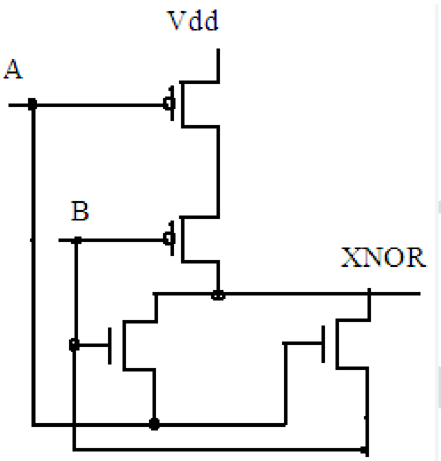
For a 2-input XOR gate, the truth table is as follows:

A	B	Output (Y)
0	0	0
0	1	1
1	0	1
1	1	0

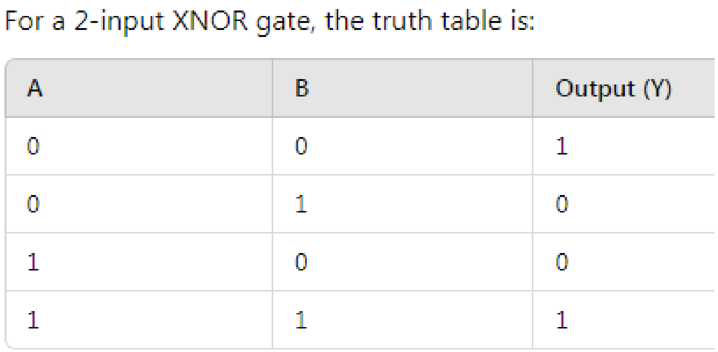
EX-NOR :

A **CMOS XNOR gate** (Exclusive-NOR gate) is a digital logic gate that outputs a **high (1)** when its inputs are **equal** (both 0 or both 1) and outputs a **low (0)** when the inputs are **different**. It is the inverse of the **XOR gate**

$$Y = \overline{A \oplus B}$$



TRUTH TABLE:



For a 2-input XNOR gate, the truth table is:

A	B	Output (Y)
0	0	1
0	1	0
1	0	0
1	1	1

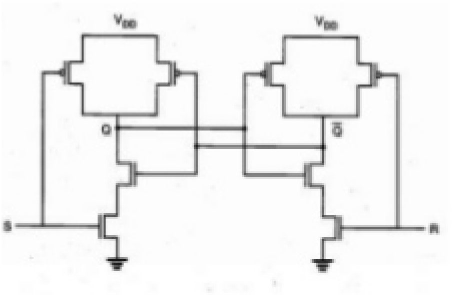
FLIPLFLOP :

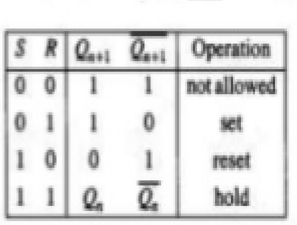
Flip-flops are fundamental building blocks in digital electronics and are used to store binary data (either 0 or 1). They are **bistable multivibrators** that can remain in one of two stable states (set or reset) until an external signal causes them to switch states. Flip-flops are primarily used in **sequential circuits**, where the output depends not only on the current input but also on previous inputs (due to the stored state).

1. SR (Set-Reset) Flip-Flop

Theory:

- The **SR flip-flop** is the simplest type of flip-flop. It has two inputs: **Set (S)** and **Reset (R)**.
- The **Set** input sets the output to **1**.
- The **Reset** input resets the output to **0**.



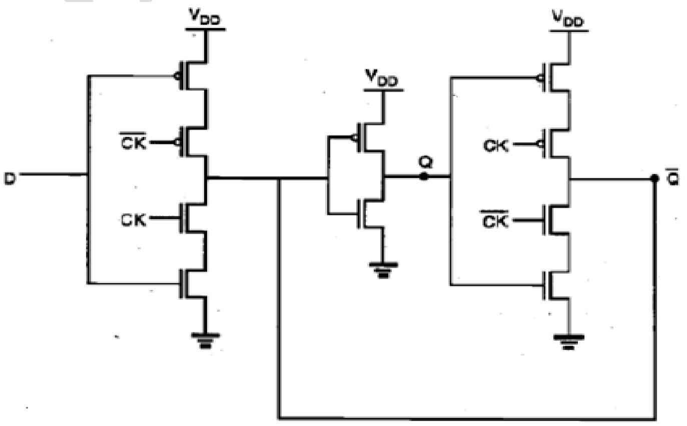


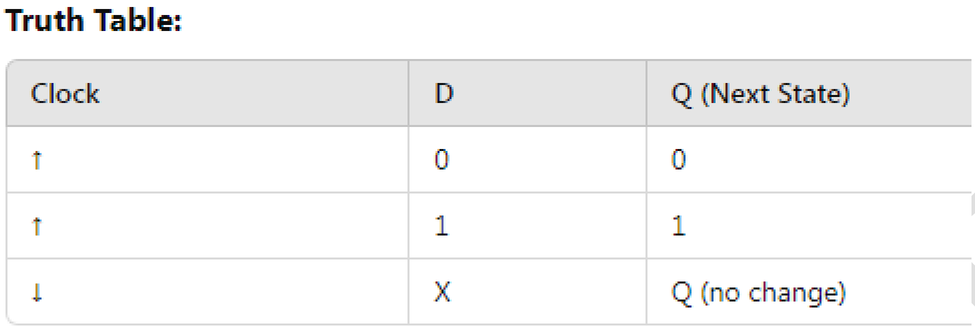
S	R	Q_{n+1}	$\overline{Q_{n+1}}$	Operation
0	0	1	1	not allowed
0	1	1	0	set
1	0	0	1	reset
1	1	Q_n	$\overline{Q_n}$	hold

2. D (Data or Delay) Flip-Flop

Theory:

- The **D flip-flop** is also called a **data** or **delay** flip-flop. It has a single data input (**D**) and a clock input (**CLK**).
- When the clock signal is active (on a rising edge or falling edge, depending on the design), the **D flip-flop** captures the value on the **D** input and holds it until the next clock edge.





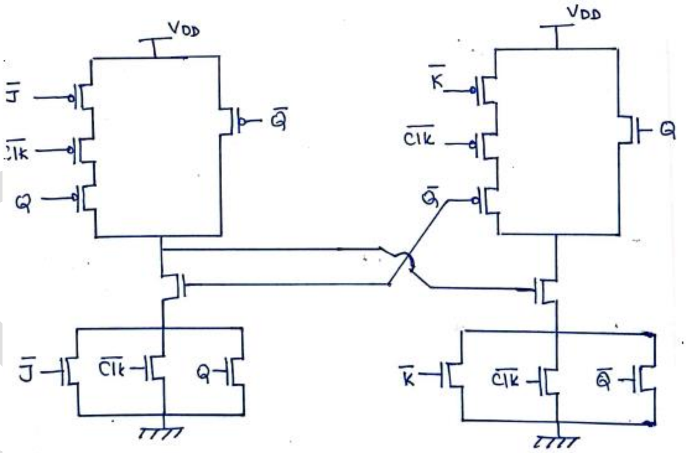
Truth Table:

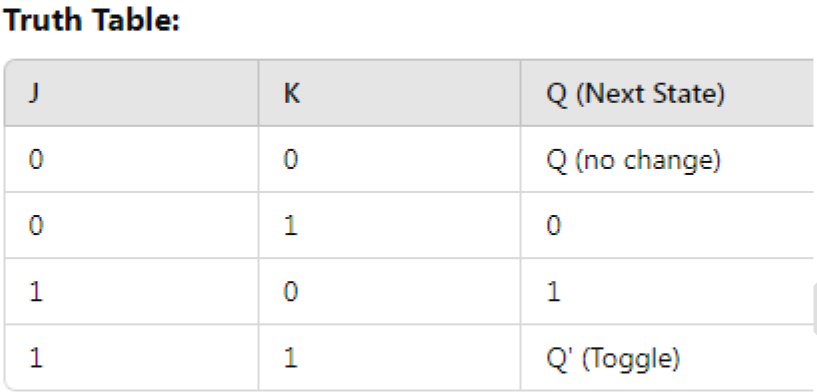
Clock	D	Q (Next State)
↑	0	0
↑	1	1
↓	X	Q (no change)

3. JK Flip-Flop

Theory:

- The **JK flip-flop** is a refinement of the SR flip-flop. It eliminates the **undefined state** by toggling the output when both inputs are **1**.
- It has two inputs: **J (Set)** and **K (Reset)**, along with a clock input.
- The **JK flip-flop** behaves like an SR flip-flop when **J = 0** and **K = 0**, but when **J = K = 1**, it toggles the output state.





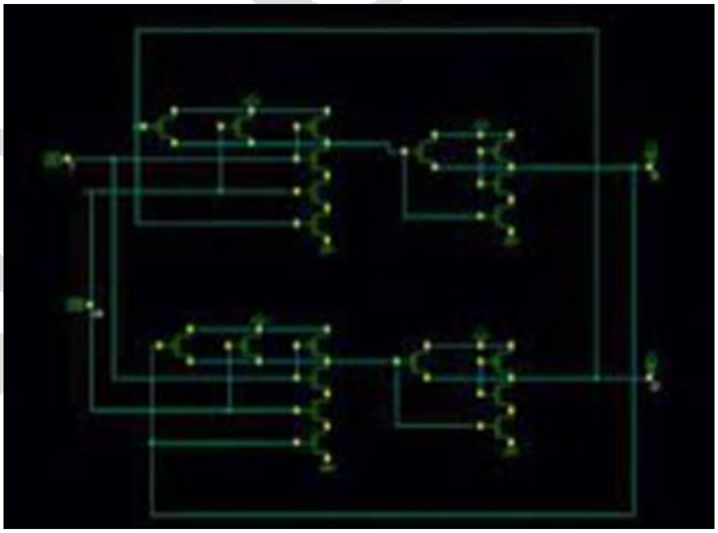
Truth Table:

J	K	Q (Next State)
0	0	Q (no change)
0	1	0
1	0	1
1	1	Q' (Toggle)

4. T (Toggle) Flip-Flop

Theory:

- The **T flip-flop** is a simplified version of the **JK flip-flop** with both **J** and **K** tied together, creating a single **T (toggle)** input.
- If **T = 0**, the output does not change.
- If **T = 1**, the output toggles with every clock pulse.



Truth Table:

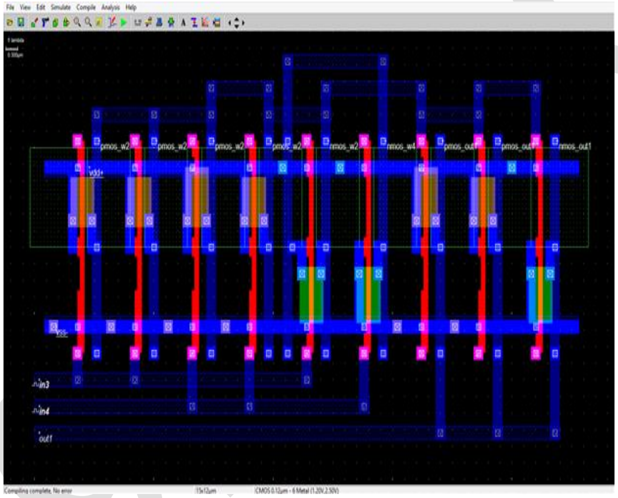
T	Q (Next State)
0	Q (no change)
1	Q' (Toggle)

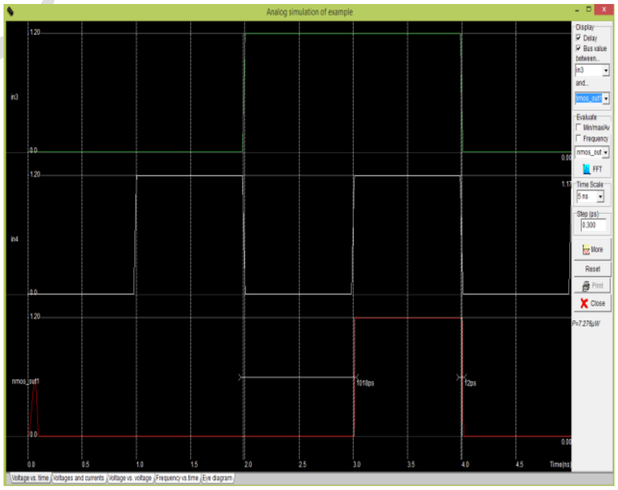
PROCEDURE:

1. Open Microwind by double clicking on Microwind.
2. Create new design by click on file-new
3. Select model file by click on file->select foundary and select the model file and save the file
4. Make Verilog file and click on generate
5. To run the simulation file
6. View the output waveform

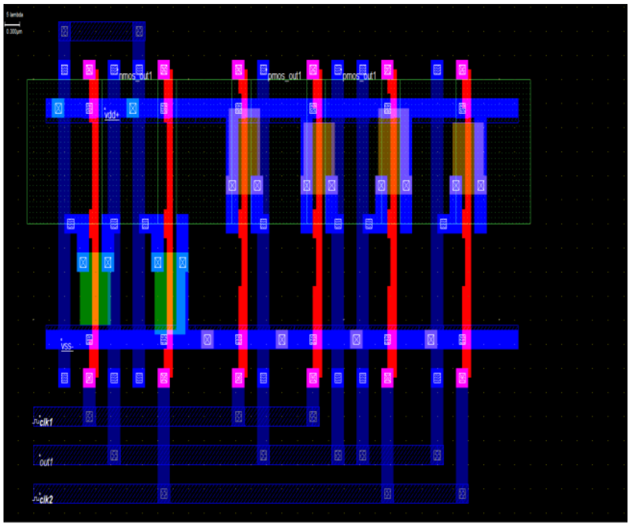
SIMULATION OUTPUT:

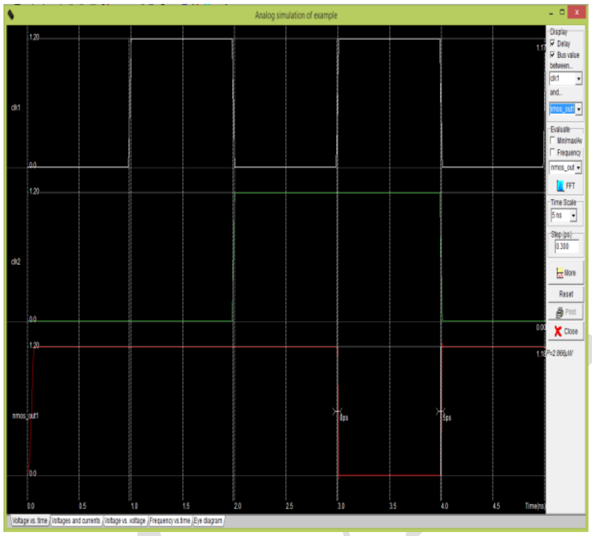
AND :


File View Edit Simulate Compile Analysis Help
pmos_w2
pmos_w2
pmos_w2
pmos_w2
nmos_w2
nmos_w4
pmos_out
pmos_out
nmos_out1
vdd+
vss-
in3
in4
out1
Compiling complete. No error
15x12µm
CMOS 0.12µm - 6 Metal (1.20V,2.30V)

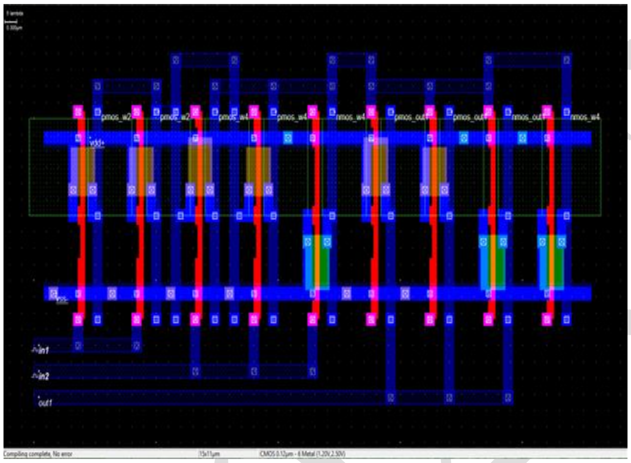


NAND :

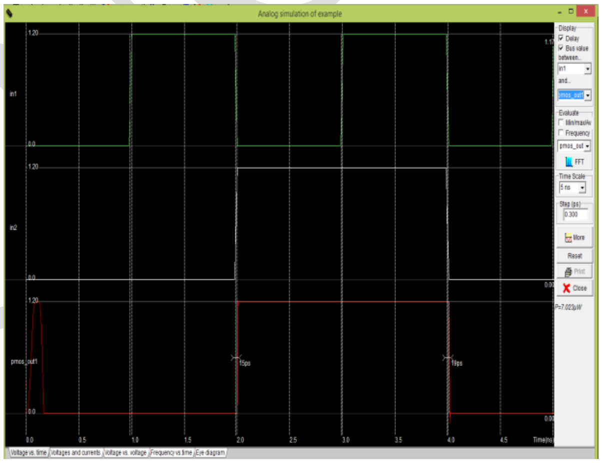




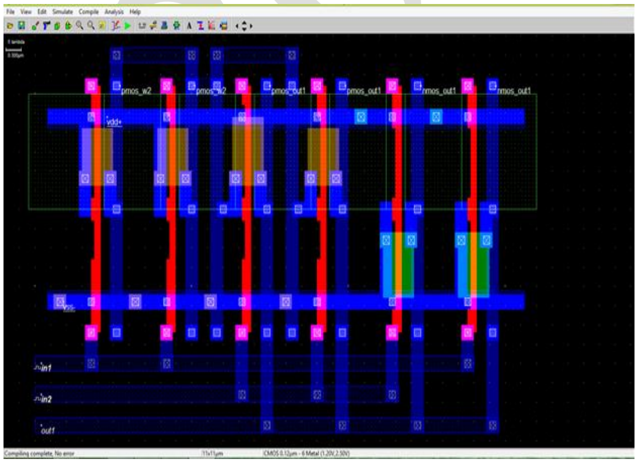
OR :



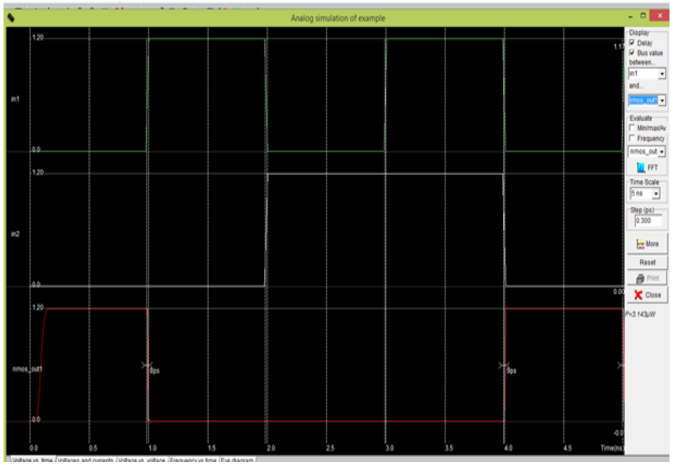
Compiling complete, No error



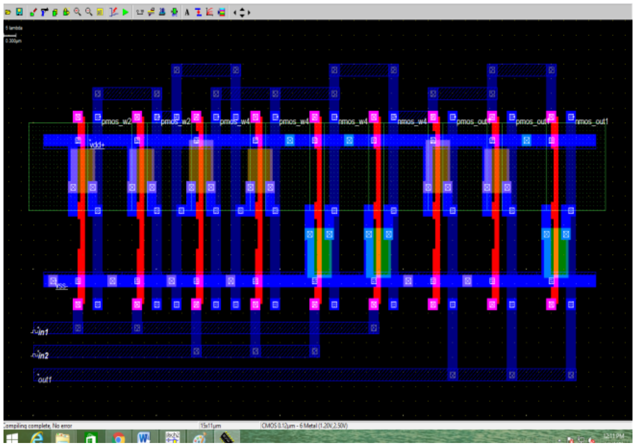
NOR :



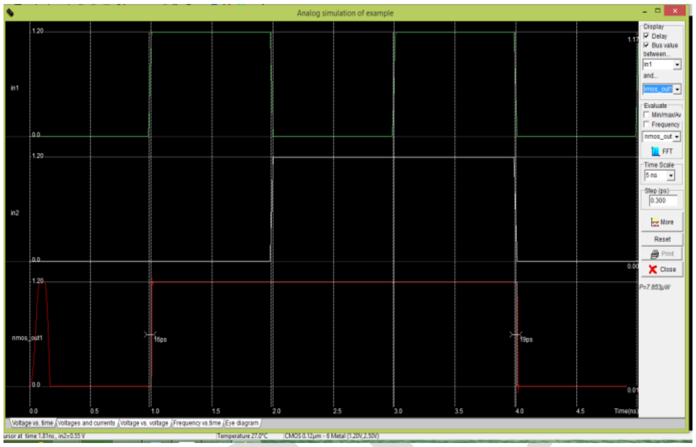
Compiling complete, No error



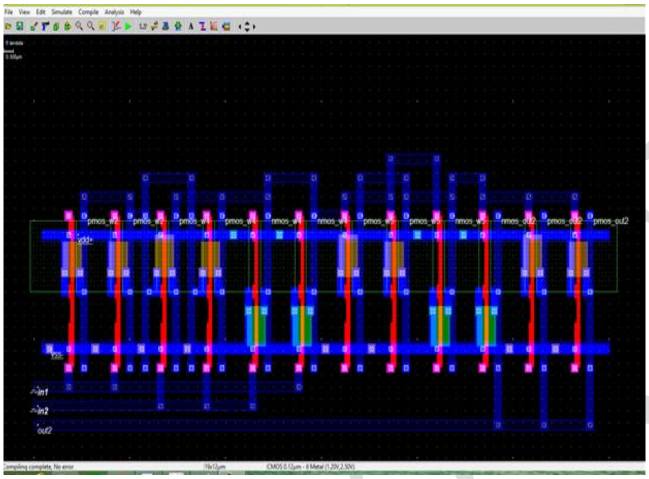
EXOR :



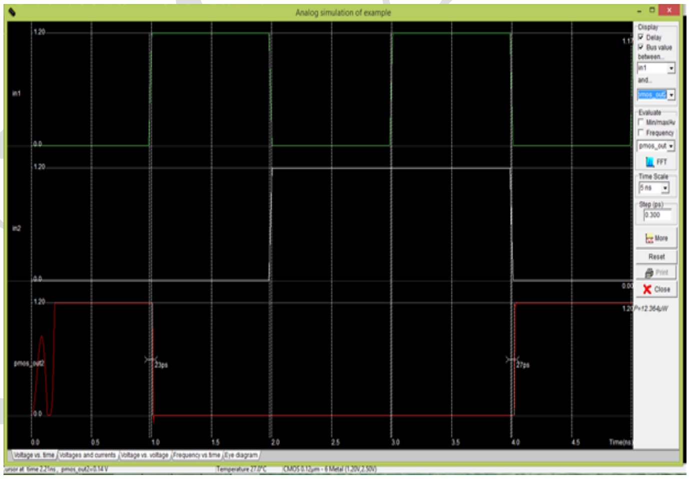
Compiling complete, No error



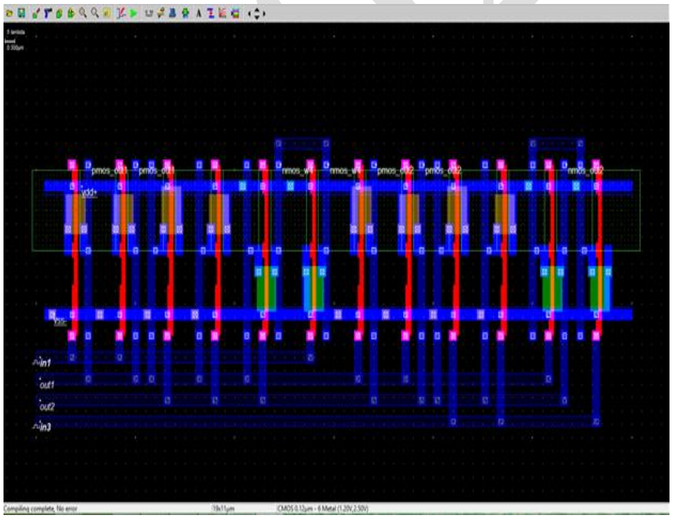
EXNOR :



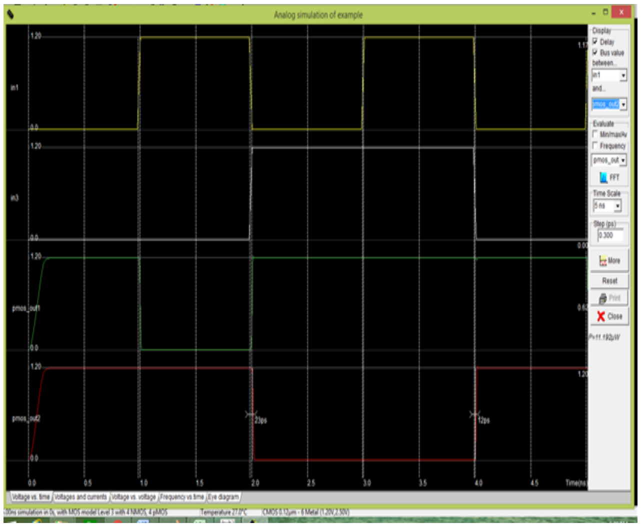
Compiling complete, No error



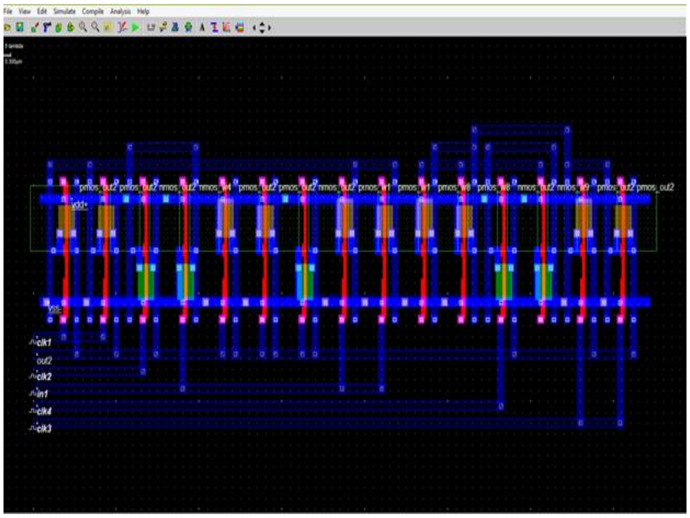
SR FLIPFOP :

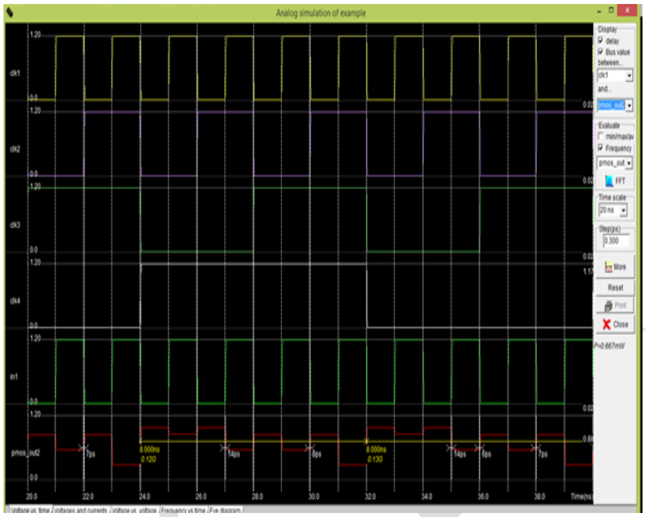


Compiling complete, No error

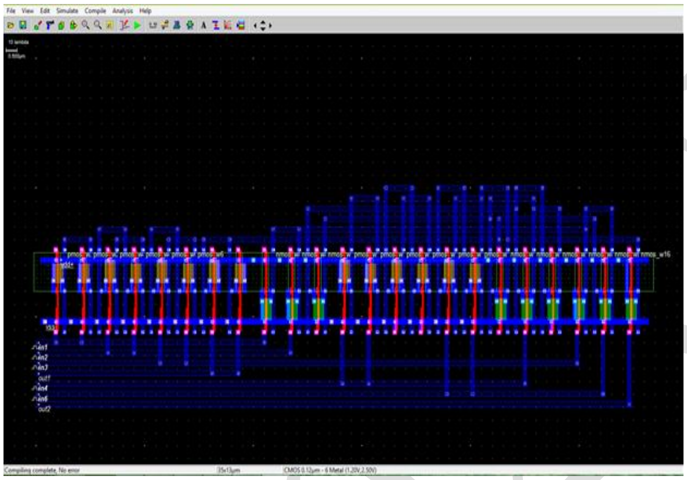


D FLIPFLOP :

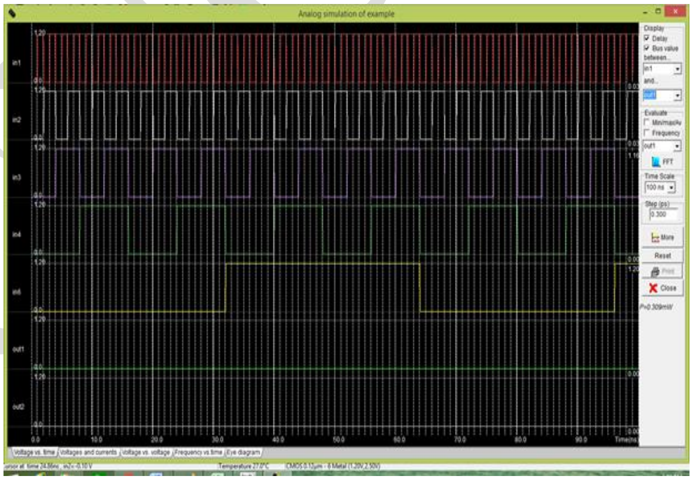




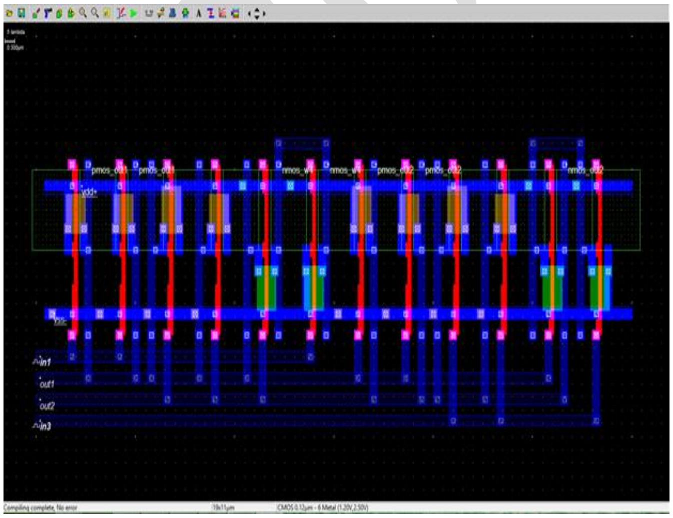
JK FLIPFLOP :



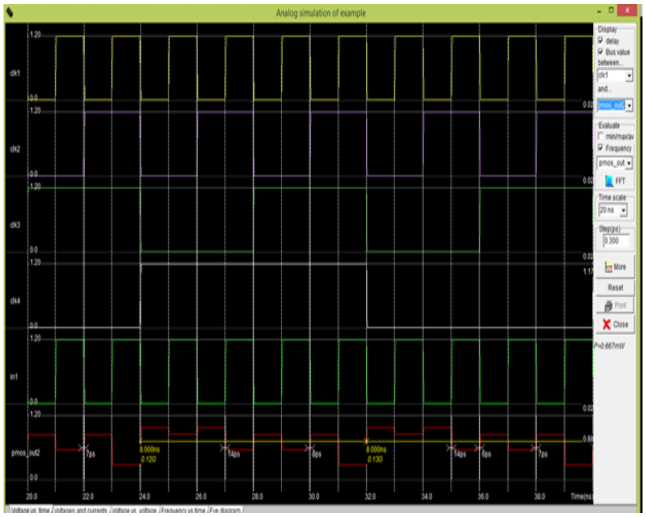
Compiling complete, No error



T FLIPFLOP :



Compiling complete, No error



POST LAB QUESTIONS :

1. What is the difference between a D flip-flop and an SR flip-flop?
2. Explain the working of a JK flip-flop.
3. What is the race-around condition in a JK flip-flop?
4. What is a master-slave flip-flop?
5. What is metastability in flip-flops?

RESULT:

Thus the CMOS logic gates schematic form was designed and simulated using Microwind.

EX.NO. 9 DESIGN 4-BIT SYNCHRONOUS COUNTER USING A FLIP-FLOPS SIMULATE GENERATE LAYOUT

AIM :

To design, synthesize, simulate and implement the 4-bit synchronous counter using a Flip-Flops in layout form using Microwind.

TOOLS REQUIRED :

1. MICROWIND Tools
2. PC

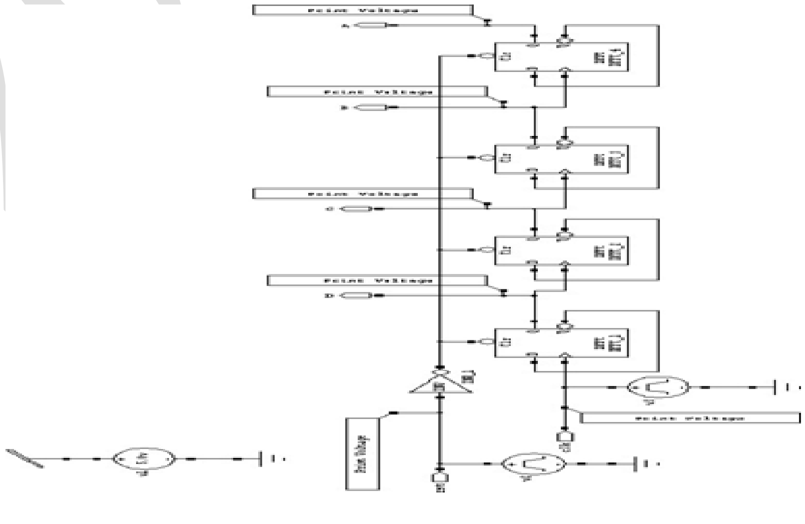
PRELAB QUESTIONS :

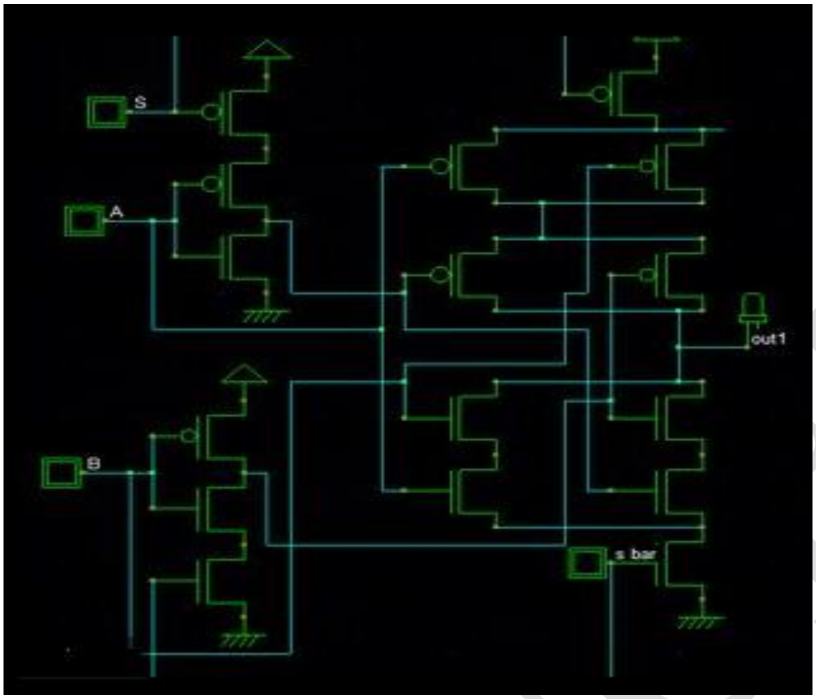
1. What is a 4-bit synchronous counter?
2. What is the difference between synchronous and asynchronous counters?
3. What type of flip-flop is typically used in a 4-bit counter?
4. How does a 4-bit synchronous counter work?
5. How is the counting sequence controlled in a synchronous counter?

THEORY: (COUNTER)

A counter that can change state in either direction, under the control of an up or down selector input, is known as an up/down counter. When the selector is in the up state, the counter increments its value. When the selector is in the down state, the counter decrements the count. Likewise the counter counts in both the directions continuously until attaining the end of the count. The count is initiated by the positive clock pulse. The counter counts from 0000 to 1111 for up count and 1111 to 0000 for down count.

LOGIC DIAGRAM:

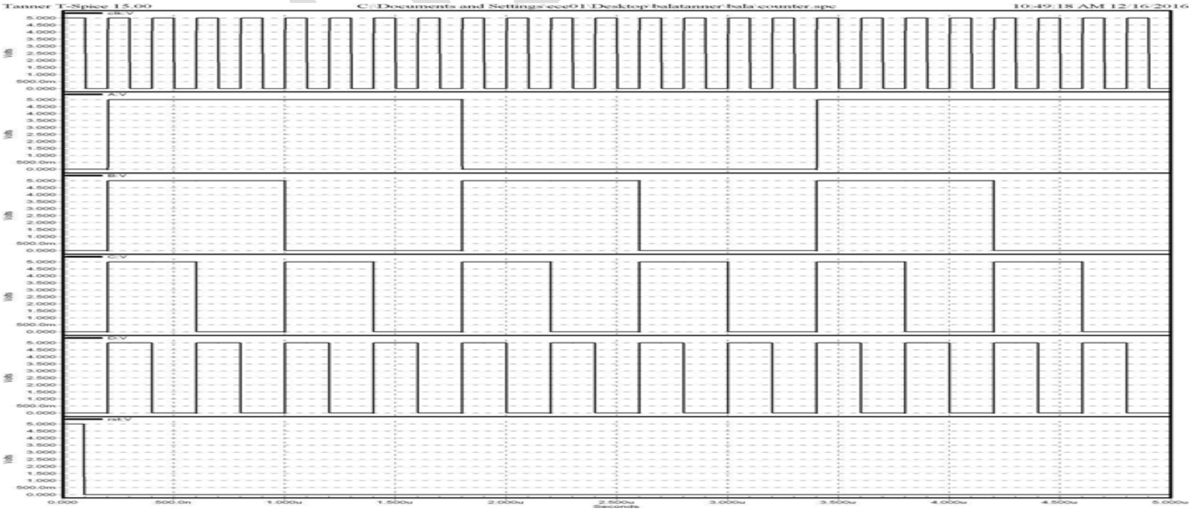




PROCEDURE:

1. Open microwind by double clicking on microwind.
2. Create new design by click on file-new
3. Select model file by click on file->select foundary and select the model file and save the file
4. Make Verilog file and click on generate
5. To run the simulation file
6. View the output waveform

OUTPUT:



POST LAB QUESTIONS :

1. What applications commonly use 4-bit synchronous counters?
2. What modifications would you make to convert a 4-bit counter to an n-bit counter?
3. Discuss the impact of fan-out on a synchronous counter's performance.
4. What is the significance of the propagation delay in a synchronous counter?
5. Describe how you would implement a counter with enable and disable features.

RESULT:

Thus the 4-bit synchronous counter schematic form was designed and simulated using Microwind.

EX.NO. 10. DESIGN AND SIMULATE CMOS INVERTER GENERATE LAYOUT

AIM :

To design, synthesize, simulate and implement the CMOS inverter in layout form using Microwind.

TOOLS REQUIRED :

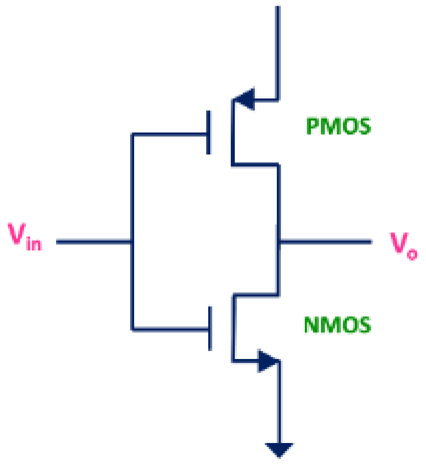
1. MICROWIND Tools
2. PC

PRELAB QUESTIONS :

1. What is a CMOS inverter?
2. How does a CMOS inverter work?
3. What are the regions of operation for the transistors in a CMOS inverter?
4. Why do we use both NMOS and PMOS transistors in a CMOS inverter?
5. What is the logic function of a CMOS inverter?

THEORY :

A **CMOS inverter** is a fundamental building block in digital circuits, used primarily for signal inversion. It employs complementary metal-oxide-semiconductor (CMOS) technology, utilizing both PMOS and NMOS transistors to achieve its functionality. PMOS Transistor is connected to the positive supply voltage (VDD) and is turned ON when the input is LOW (0). NMOS Transistor is connected to ground and is turned ON when the input is HIGH (1).



PROCEDURE:

1. Open microwind by double clicking on microwind.

2. Create new design by click on file-new

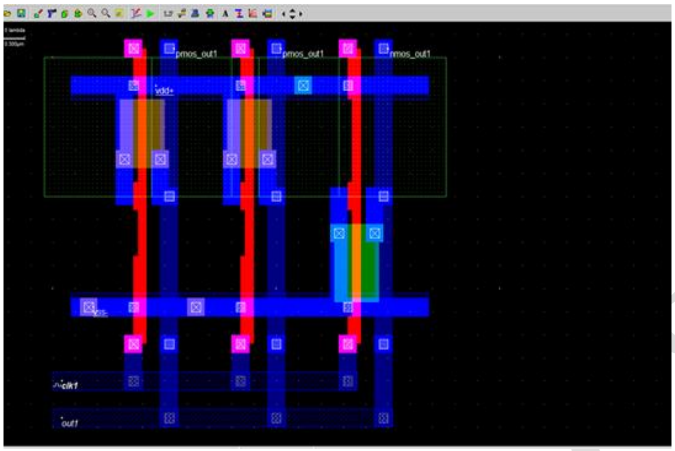
3. Select model file by click on file-> select foundary and select the model file and save the file

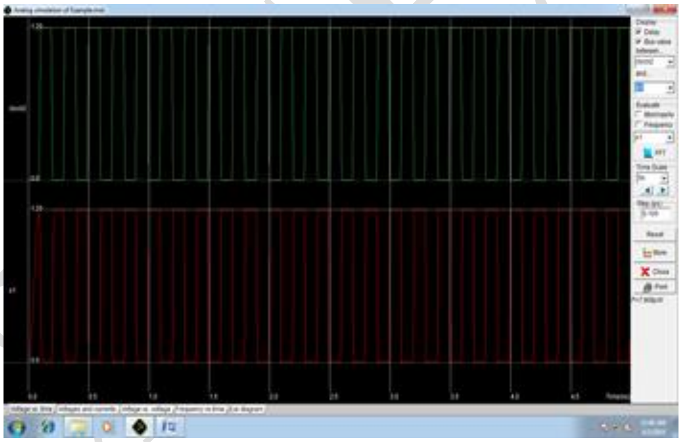
4. Make Verilog file and click on generate

5. To run the simulation file

6. View the output waveform

OUTPUT





POST LAB QUESTIONS :

1. What are the advantages of using CMOS technology?
2. Where are CMOS inverters typically used?
3. How do you determine the pull-up and pull-down strengths of the PMOS and NMOS transistors?
4. How do short-circuit currents occur in CMOS inverters?
5. In what applications would you use CMOS inverters?

RESULT:

Thus the CMOS inverter schematic form was designed and simulated using Microwind.

EX.NO. 11 DESIGN AND SIMULATE COMMON SOURCE , COMMON DRAIN , COMMON GATE GENERATE LAYOUT

AIM :

To design, synthesize, simulate and implement the CMOS inverter in layout form using Microwind.

TOOLS REQUIRED :

1. MICROWIND Tools
2. PC

PRELAB QUESTIONS:

1. What is a common source (CS) amplifier?
2. What are the key characteristics of a common source amplifier?
3. What is a common gate (CG) amplifier?
4. In what applications is a common gate amplifier commonly used?
5. What is the significance of the common drain amplifier in analog circuits?

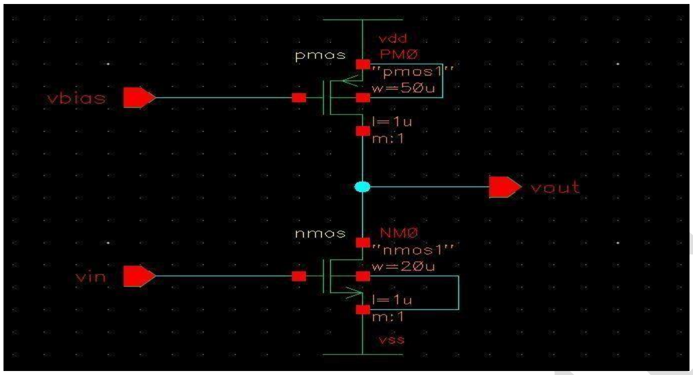
THEORY:

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, orcommon 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.

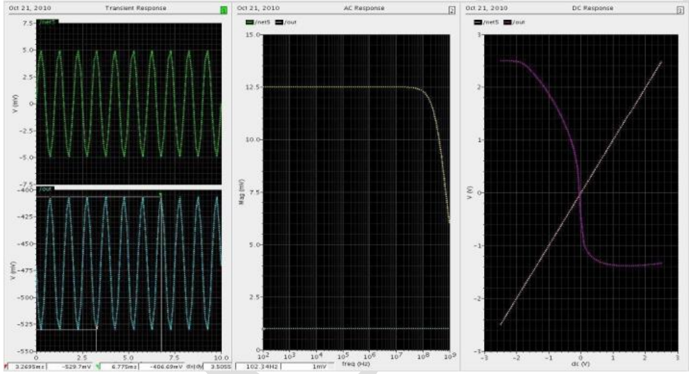
The analogous bipolar junction transistor circuit is the common-emitter amplifier. Common drain amplifier is a source follower or buffer amplifier circuit using a MOSFET. The output is simply equal to the input minus about 2.2V.

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. The CG amplifier in which the input signal is sensed at the source terminal and the output is produced at the drain terminal. The gate terminal is connected to V_B i.e. dc potential which will maintain the proper operating conditions.

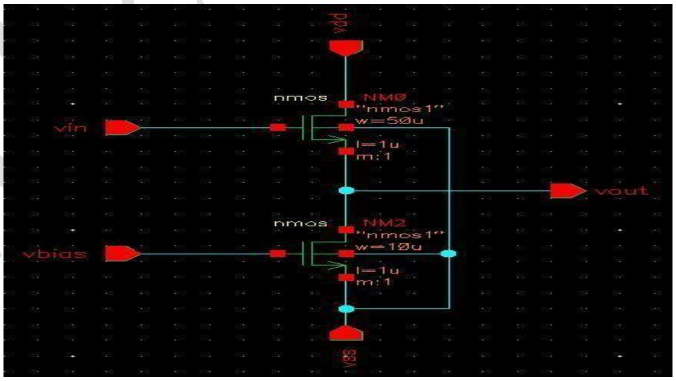
COMMON SOURCE AMPLIFIER:



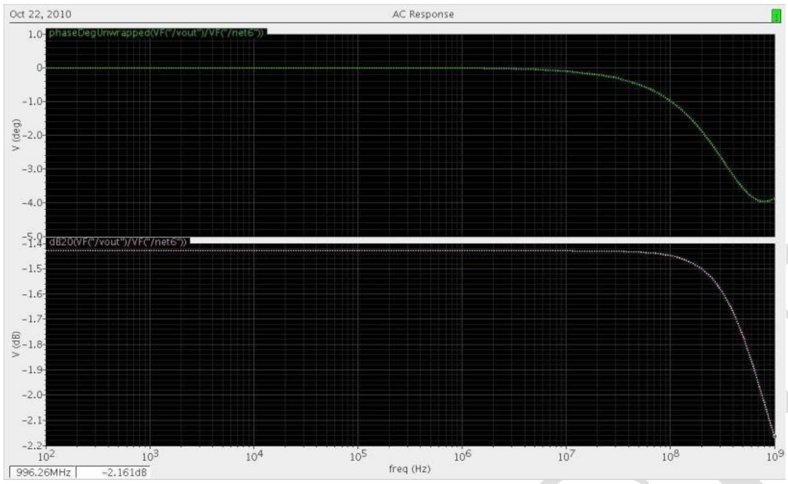
OUTPUT:



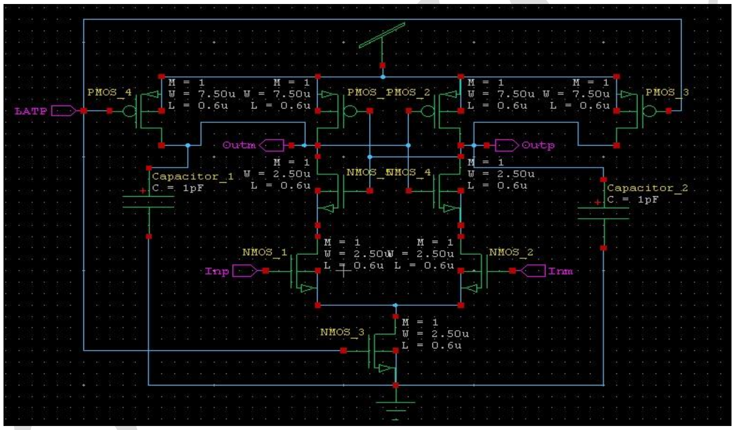
COMMON DRAIN AMPLIFIER:



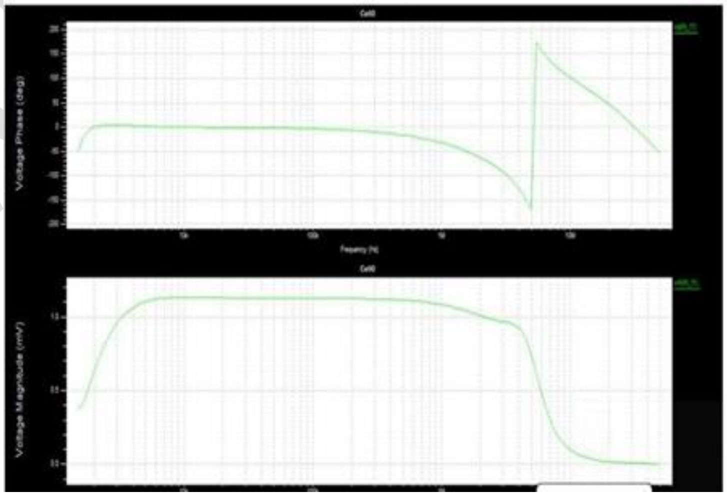
OUTPUT:



COMMON GATE AMPLIFIER:



OUTPUT:



PROCEDURE:

1. Open microwind by double clicking on microwind.
2. Create new design by click on file-new
3. Select model file by click on file->select foundary and select the model file and save the file
4. Make Verilog file and click on generate
5. To run the simulation file
6. View the output waveform

POST LAB QUESTIONS :

1. What is the primary use of a common drain amplifier in circuits?
2. How is a common gate amplifier configured in a simulation?
3. hat common issues might arise during the simulation of these amplifier configurations?
4. How does the output impedance of a common gate amplifier compare to that of a common source amplifier?
5. What is the main difference between a common gate amplifier and other configurations?

RESULT:

Thus the common souce common drain ,common gate schematic form was designed and simulated using microwind.

EX.NO. 12 DESIGN AND SIMULATE 5 TRNSISTOR DIFFERENTIAL AMPLIFIER AND GENERATE LAYOUT

AIM :

To design, synthesize, simulate and implement the 5 transistor differential amplifier in layout form using Microwind.

TOOLS REQUIRED :

1. MICROWIND Tools
2. PC

PRELAB QUESTIONS :

1. What is a differential amplifier?
2. What are the main applications of a differential amplifier?
3. Explain the difference between a differential amplifier and an operational amplifier (op-amp).
4. What is the differential gain of a differential amplifier?
5. What is common-mode rejection ratio (CMRR)?

THEORY :

The **5-transistor (5T) differential amplifier** is a specific configuration that enhances the basic differential amplifier design by utilizing additional transistors to improve performance characteristics such as gain, linearity, and common-mode rejection ratio (CMRR).

It typically consists of five transistors arranged to form the differential input stage and additional transistors for amplification and output.

The two main input transistors are responsible for the differential operation, while the remaining three transistors are used for improved gain and biasing.

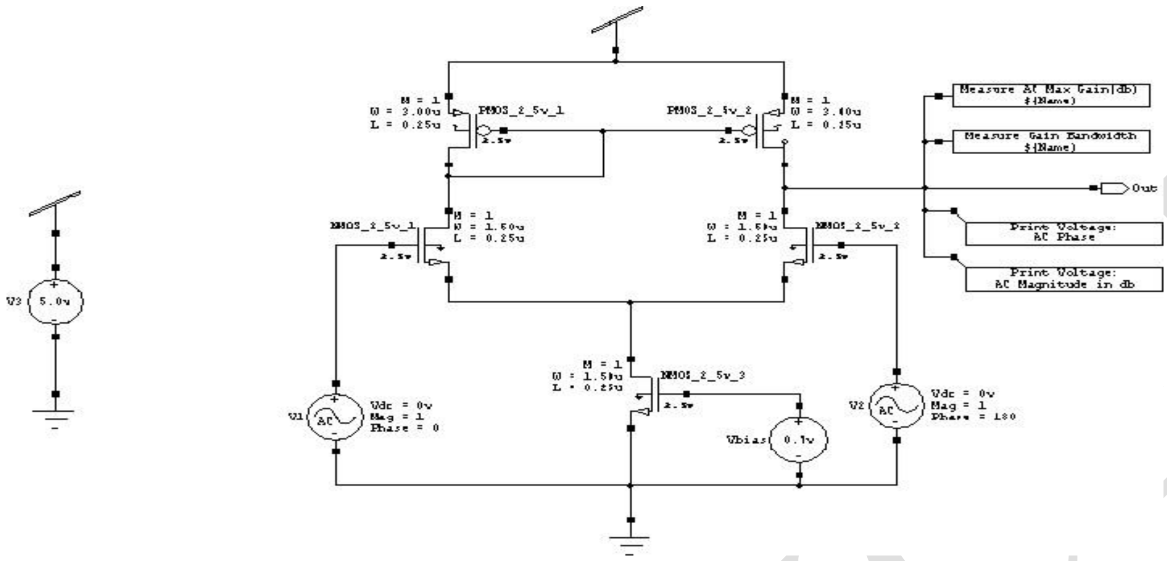
The basic configuration includes a differential pair (two transistors) for the inputs, often referred to as Q1Q_1Q1 and Q2Q_2Q2.

The output stage can include a current mirror configuration to improve the output impedance and gain. The additional transistors are often arranged to provide a better active load, leading to increased voltage gain.

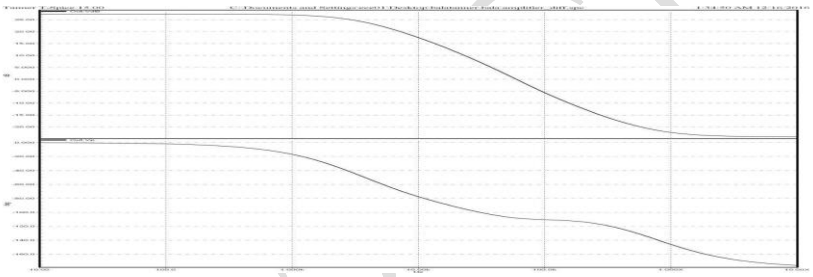
$$V_{out} = A_d \times (V_{in+} - V_{in-})$$

$$\text{CMRR} = \frac{A_d}{A_{cm}}$$

LOGIC DIAGRAM:



OUTPUT:



PROCEDURE:

1. Open microwind by double clicking on microwind.
2. Create new design by click on file-new
3. Select model file by click on file->select foundary and select the model file and save the file
4. Make Verilog file and click on generate
5. To run the simulation file
6. View the output waveform

POST LAB QUESTIONS:

1. What is an instrumentation amplifier, and how does it differ from a basic differential amplifier?
2. Discuss the significance of offset voltage in differential amplifiers.
3. How do you analyze the stability of a differential amplifier?
4. What methods would you use to test a differential amplifier in a lab setting?
5. Where would you typically use a differential amplifier in a circuit?

RESULT:

Thus the 5 transistor differential amplifier schematic form was designed and simulated using microwind.

VALUE ADDED EXPERIMENTS

EX.NO.1 DESIGN AND IMPLEMENTATION OF ENCODER USING HDL

SIMULATE IT USING XILINX /ALTERA SOFTWARE & IMPLEMENT BY XILINX ALTERA FPGA

AIM:

To design the 8x3 encoder using Verilog and simulate the design

TOOLS REQUIRED :

1. Xilinx ISE Design Suite 8.1i
2. PC

PRE-LAB QUESTIONS :

1. What is an encoder?
2. Why is an encoder used?
3. How many output lines does a 4-to-2 encoder have?
4. What is the logic equation for a 4-to-2 encoder?
5. What happens if all inputs of an encoder are 0?

THEORY:

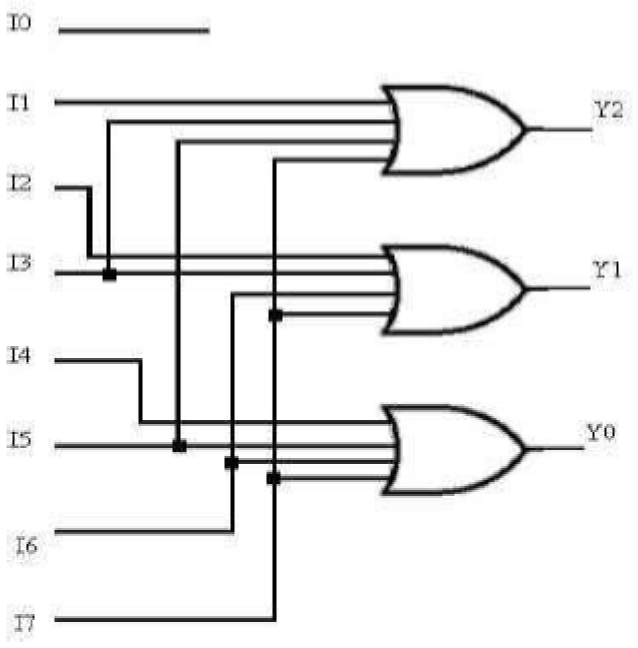
An encoder is a combinational logic circuit that essentially performs a “reverse” of decoder functions. An encoder has 2^N input lines and N output lines. In encoder the output lines generate the binary code corresponding to input value. An encoder accepts an active level on one of its inputs, representing digit, such as a decimal or octal digits, and converts it to a coded output such as BCD or binary. Encoders can also be devised to encode various symbols and alphabetic characters. The process of converting from familiar symbols or numbers to a coded format is called ***encoding***. An encoder has a number of input lines, only one of which input is activated at a given time and produces an N-bit output code, depending on which input is activated.

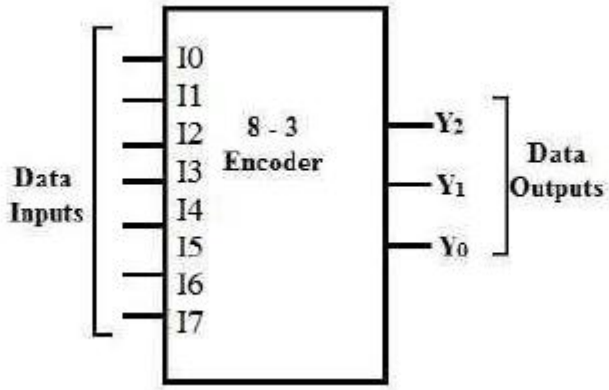
For an 8-to-3 binary encoder with inputs I0-I7 the logic expressions of the outputs Y0-Y2 are:

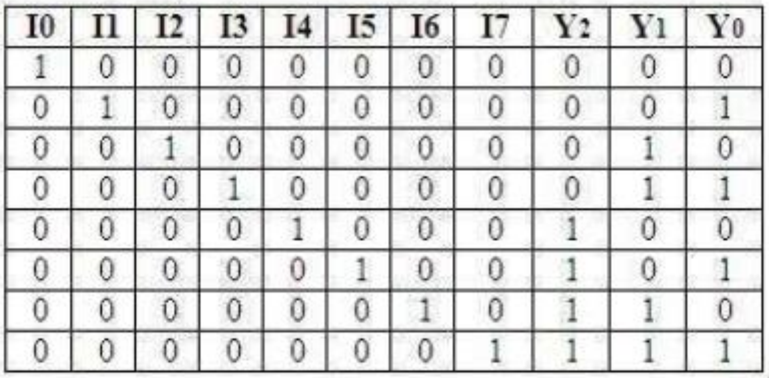
$$Y0 = I1 + I3 + I5 + I7$$

$$Y1= I2 + I3 + I6 + I7$$

LOGIC DIAGRAM :







I0	I1	I2	I3	I4	I5	I6	I7	Y_2	Y_1	Y_0
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

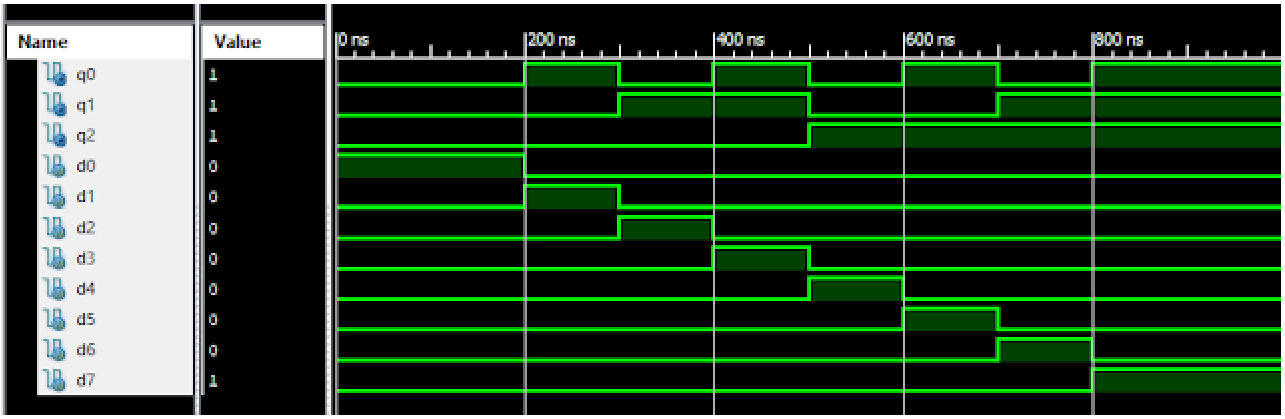
PROGRAM:

```
module encoder8to3 (
    input  [7:0] in,    // 8 input lines
    output reg [2:0] out // 3-bit output );
always @(*) begin
    case (1'b1) // prioritize the first high input from MSB to LSB
        in[7]: out = 3'b111;
        in[6]: out = 3'b110;
        in[5]: out = 3'b101;
        in[4]: out = 3'b100;
        in[3]: out = 3'b011;
        in[2]: out = 3'b010;
        in[1]: out = 3'b001;
        in[0]: out = 3'b000;
        default: out = 3'bxxx; // undefined when no input is high
    endcase
end
```

PROCEDURE:

1. Click on the Xilinx ISE Design Suite 12.1or Xilinx Project navigator icon on the desktop ofPC.
2. Write the Verilog code by choosing HDL as top level source module.
3. Check syntax, view RTL schematic and note the device utilization summary by double
4. clicking on the synthesis in the process window.
5. Perform the functional simulation using Xilinx ISE simulator.
6. The output wave form can be observed in model sim.

SIMULATION OUTPUT :



POST LAB QUESTIONS:

1. What IC did you use for the encoder experiment?
2. Can an encoder have a ‘don’t care’ condition?
3. What are the limitations of using basic encoders in real systems?
4. Why is the number of output lines in an encoder less than the number of input lines?
5. What is the general formula for the number of output lines in an encoder?

RESULT:

Verilog code for the design of Encoder circuit verification is written, the waveform   is   observed and the code is synthesized with the technological library and is verified.

EX.NO.2 DESIGN AND IMPLEMENTATION OF DECODER USING HDL.

SIMULATE IT USING XILINX /ALTERA SOFTWARE & IMPLEMENT BY XILINX ALTERA FPGA

AIM:

To Design, Implement and simulate the 3:8 Decoder.

TOOLS REQUIRED :

1. Xilinx ISE Design Suite 8.1i
2. PC

PRE-LAB QUESTIONS :

1. What is a decoder?
2. What is the function of a decoder?
3. What is the use of the Enable pin in decoders?
4. What is the general formula for number of outputs in a decoder?
5. Why is only one output active in a decoder at a time?

,

THEORY :

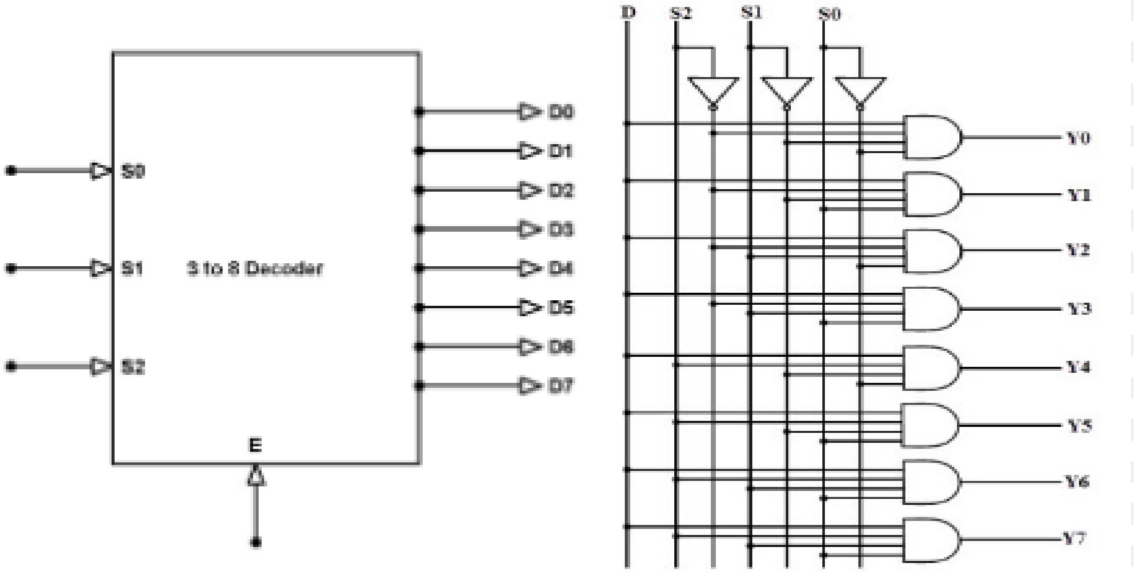
In digital electronics, a decoder can take the form of a multiple-input, multiple-output logic circuit that converts coded inputs into coded outputs, where the input and output codes are different e.g. n-to-2n , binary-coded decimal decoders. Decoding is necessary in applications such as data multiplexing, 7 segment display and memory address decoding.

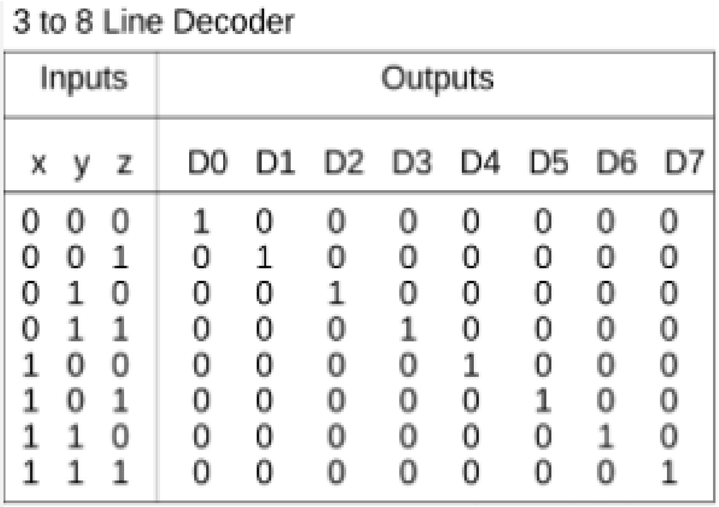
The example decoder circuit would be an AND gate because the output of an AND gate is "High" (1) only when all its inputs are "High". Such output is called as "active High output". If instead of AND gate, the NAND gate is connected the output will be "Low" (0) only when all its inputs are "High". Such output is called as "active low output".

A slightly more complex decoder would be the n-to-2n type binary decoders. These types of decoders are combinational circuits that convert binary information from 'n' coded inputs to a maximum of 2n unique outputs. In case the 'n' bit coded information has unused bit combinations, the decoder may have less than 2n outputs. 2-to-4 decoder, 3-to-8 decoder or 4-to-16 decoder are other examples.

The input to a decoder is parallel binary number and it is used to detect the presence of a particular binary number at the input. The output indicates presence or absence of specific number at the decoder input.
Let us suppose that a logic network has 2 inputs A and B. They will give rise to 4 states A, A', B, B' . The truth table for this decoder is shown below:

LOGIC DIAGRAM :





3 to 8 Line Decoder

Inputs			Outputs							
x	y	z	D0	D1	D2	D3	D4	D5	D6	D7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

PROGRAM :

```
module decoder3_to_8(
    input x,
    input y,
    input z,
    output d0,
    output d1,
    output d2,
    output d3,
    output d4,
    output d5,
    output d6,
    output d7
    );
assign d0 = xn & yn & zn;
assign d1 = xn & yn & z;
assign d2 = xn & y & zn;
assign d3 = xn & y & z;
assign d4 = x & yn & z;
assign d5 = x & yn & z;
assign d6 = x & y & zn;
assign d7 = x & y & z;
assign xn = ~ x;
assign yn = ~ y;
assign zn = ~ z;
endmodule
```

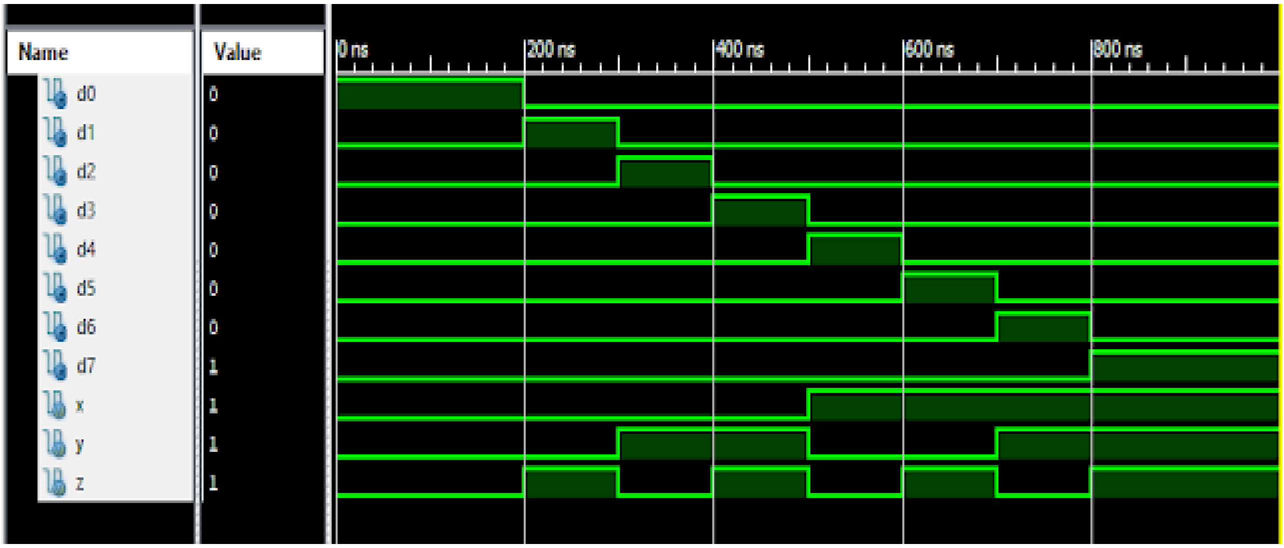
PROCEDURE:

1. Click on the Xilinx ISE Design Suite 12.1or Xilinx Project navigator icon on the desktop ofPC.
2. Write the Verilog code by choosing HDL as top level source module.
3. Check syntax, view RTL schematic and note the device utilization summary by double
4. clicking on the synthesis in the process window.

5. Perform the functional simulation using Xilinx ISE simulator.

6. The output wave form can be observed in model sim.

SIMULATION OUTPUT :



Name	Value
d0	0
d1	0
d2	0
d3	0
d4	0
d5	0
d6	0
d7	1
x	1
y	1
z	1

POST LAB QUESTIONS:

1. What are the advantages of active-low output decoders in real circuits?
2. How would you modify a decoder to make it programmable?
3. What happens to the output of a 3-to-8 decoder if the enable pin is not active?
4. What is the minimum number of input lines required for a decoder with 32 outputs?
5. What’s the impact of fan-out on decoder performance?

RESULT:

Verilog code for the design of decoder circuit verification is written, the waveform is observed and the code is synthesized with the technological library and is verified.