Basaveshwara Veerashaiva Vidyavardhaka Sangha
AMRUTA INSTITUTE OF ENGINEERING AND MANAGEMENT SCIENCES
(Affiliated to Visvesvaraya Technological University, Belagavi & Approved by AICTE, New Delhi)
Bidadi, Ramanagara, Karnataka – 562109

Department of Artificial Intelligence and Machine Learning

Digital Design and Computer Organization

(Course code: BCS302)

LAB MANUAL

Kavya A D
Assistant Professor
Dept. of AI&ML,
AIEMS

2025 - 2026

Digital Design and Computer Organization	Semester 3
Course Code: BCS302	CIE Marks :50
Teaching Hours/Week (L:T:P: S) : 3:0:2:0	SEE Marks :50
Total Marks :100	Credits: 04
Total Hours of Pedagogy 40 hours Theory + 20 Hours of Practicals	
Exam Hours :3	Examination nature (SEE) Theory

PRACTICAL COMPONENT OF IPCC

1. Given a 4-variable logic expression, simplify it using appropriate technique and simulate the same using basic gates.
2. Design a 4-bit full adder and subtractor and simulate the same using basic gates.
3. Design Verilog HDL to implement simple circuits using structural, Data flow and Behavioral model.
4. Design Verilog HDL to implement Binary Adder-Subtractor – Half and Full Adder, Half and Full Subtractor.
5. Design Verilog HDL to implement Decimal adder.
6. Design Verilog program to implement Different types of multiplexers like 2:1, 4:1 and 8:1.
7. Design Verilog program to implement types of De-Multiplexer.
8. Design Verilog program for implementing various types of Flip-Flops such as SR, JK and D.

Course outcomes (Course Skill Set):

At the end of the course, the student will be able to:

CO1: Apply the K–Map techniques to simplify various Boolean expressions.

CO2: Design different types of combinational and sequential circuits along with Verilog programs.

CO3: Describe the fundamentals of machine instructions, addressing modes and Processor performance.

CO4: Explain the approaches involved in achieving communication between processor and I/O devices.

CO5: Analyze internal Organization of Memory and Impact of cache/Pipelining on Processor Performance.

Assessment Details (both CIE and SEE)

The weightage of Continuous Internal Evaluation (CIE) is 50% and for Semester End Exam (SEE) is 50%. The minimum passing mark for the CIE is 40% of the maximum marks (20 marks out of 50) and for the SEE minimum passing mark is 35% of the maximum marks (18 out of 50 marks). A student shall be deemed to have satisfied the academic requirements and earned the credits allotted to each subject/ course if the student secures a minimum of 40% (40 marks out of 100) in the sum total of the CIE (Continuous Internal Evaluation) and SEE (Semester End Examination) taken together.

- CIE for the theory component of the IPCC (maximum marks 50)
- IPCC means practical portion integrated with the theory of the course.

- CIE marks for the theory component are 25 marks and that for the practical component is 25 marks.
- 25 marks for the theory component are split into 15 marks for two Internal Assessment Tests (Two Tests, each of 15 Marks with 01-hour duration, are to be conducted) and 10 marks for other assessment methods mentioned in 22OB4.2. The first test at the end of 40-50% coverage of the syllabus and the second test after covering 85-90% of the syllabus.
- Scaled-down marks of the sum of two tests and other assessment methods will be CIE marks for the theory component of IPCC (that is for 25 marks).
- The student has to secure 40% of 25 marks to qualify in the CIE of the theory component of IPCC. CIE for the practical component of the IPCC
- 15 marks for the conduction of the experiment and preparation of laboratory record, and 10 marks for the test to be conducted after the completion of all the laboratory sessions.
- On completion of every experiment/program in the laboratory, the students shall be evaluated including viva-voce and marks shall be awarded on the same day.
- The CIE marks awarded in the case of the Practical component shall be based on the continuous evaluation of the laboratory report. Each experiment report can be evaluated for 10 marks. Marks of all experiments' write-ups are added and scaled down to 15 marks.
- The laboratory test (duration 02/03 hours) after completion of all the experiments shall be conducted for 50 marks and scaled down to 10 marks.
- Scaled-down marks of write-up evaluations and tests added will be CIE marks for the laboratory component of IPCC for 25 marks.
- The student has to secure 40% of 25 marks to qualify in the CIE of the practical component of the IPCC.

SEE for IPCC

Theory SEE will be conducted by University as per the scheduled timetable, with common question papers for the course (duration 03 hours)

1. The question paper will have ten questions. Each question is set for 20 marks.

2. There will be 2 questions from each module. Each of the two questions under a module (with a maximum of 3 sub-questions), should have a mix of topics under that module.

3. The students have to answer 5 full questions, selecting one full question from each module.

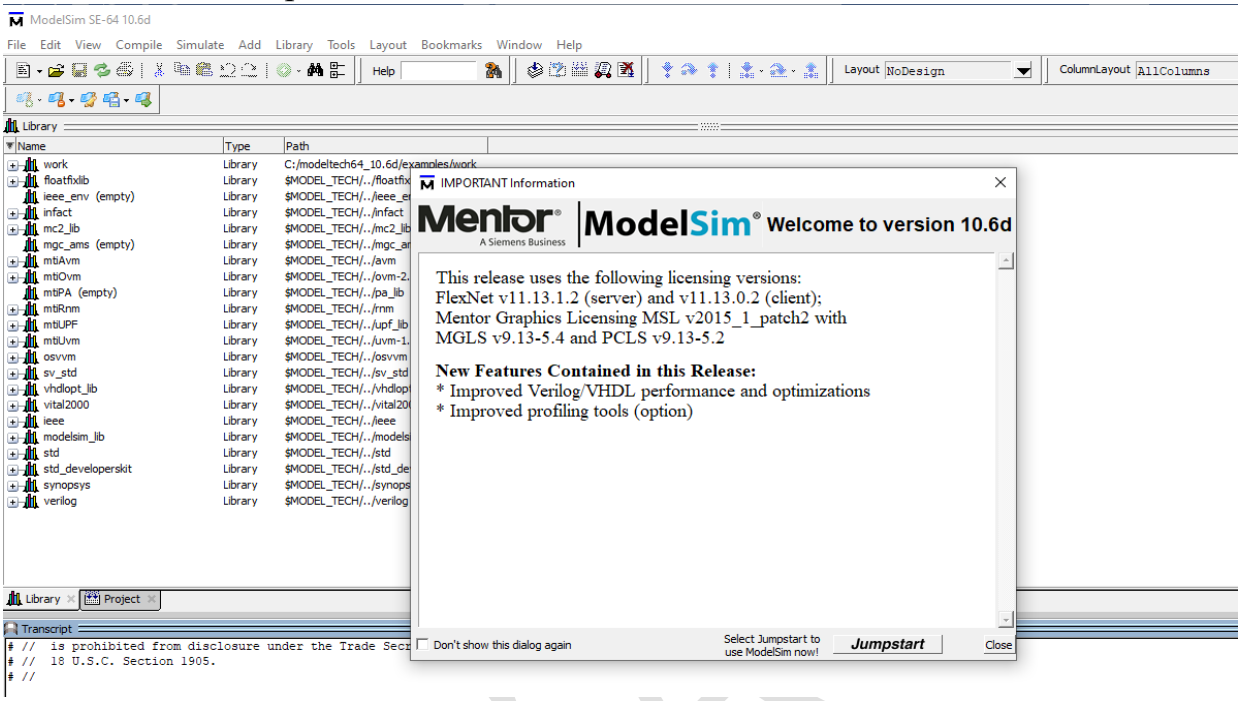
4. Marks scored by the student shall be proportionally scaled down to 50 Marks

The theory portion of the IPCC shall be for both CIE and SEE, whereas the practical portion will have a CIE component only. Questions mentioned in the SEE paper may include questions from the practical component.

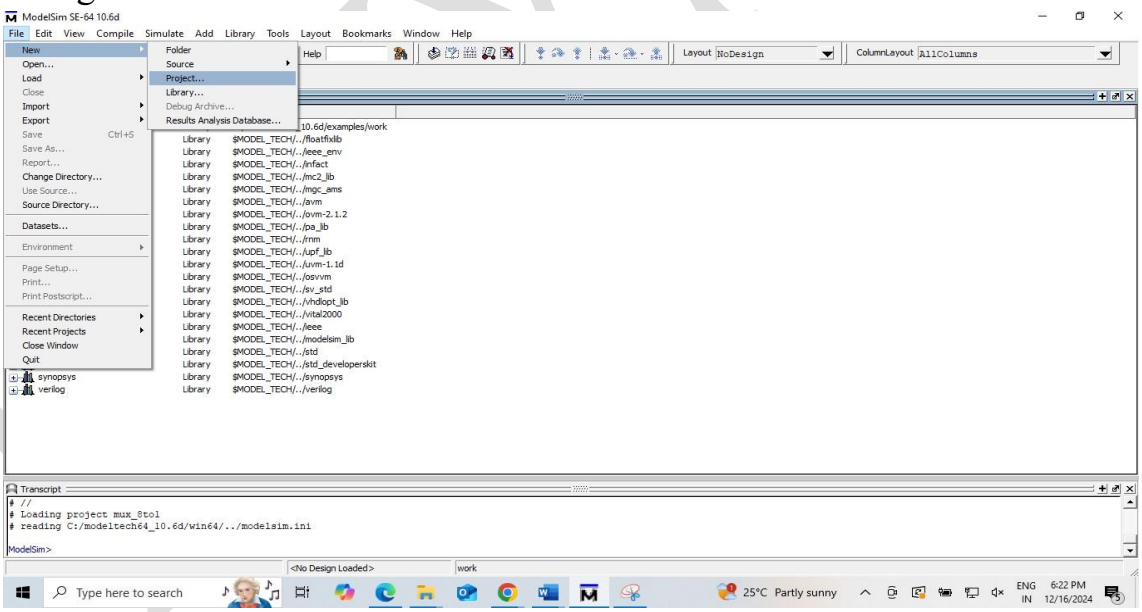
CONTENTS

Steps For Modelsim software:

1. Start ModelSim: Open ModelSim from the Start menu or a Windows shortcut icon



2. Create a project: Select File > New > Project or Create a Project from the Welcome dialog.



3. Add files: Click Add Existing File in the Add Items to Project dialog.
4. Compile: Right-click on a file, select Compile, and then click Compile All.
5. Run the simulation: Select Simulate > Start Simulation.

Program 1

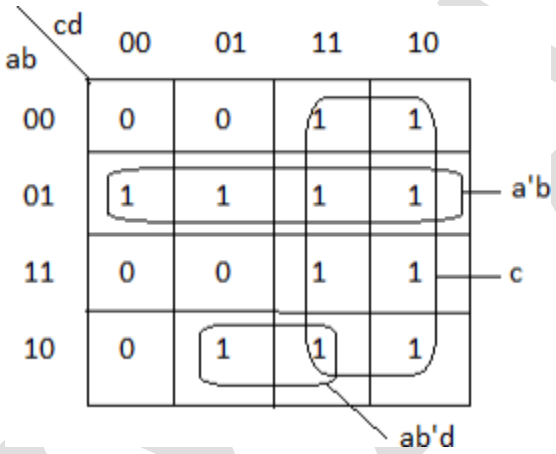
Given a 4-variable logic expression, simplify it using appropriate technique and simulate the same using basic gates.

F (a, b, c, d) = Σm(2,3,4,5,6,7,9,10,11,14,15)

Truth Table:

a	b	c	d	Y
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1

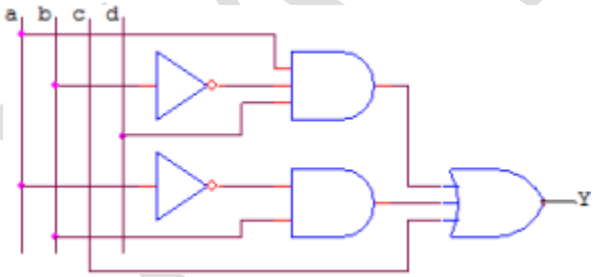
K-Map Simplication:



Boolean Function:

Y= c + a`b + ab`d

Logic Diagram:



Verilog Code (Data flow modelling):

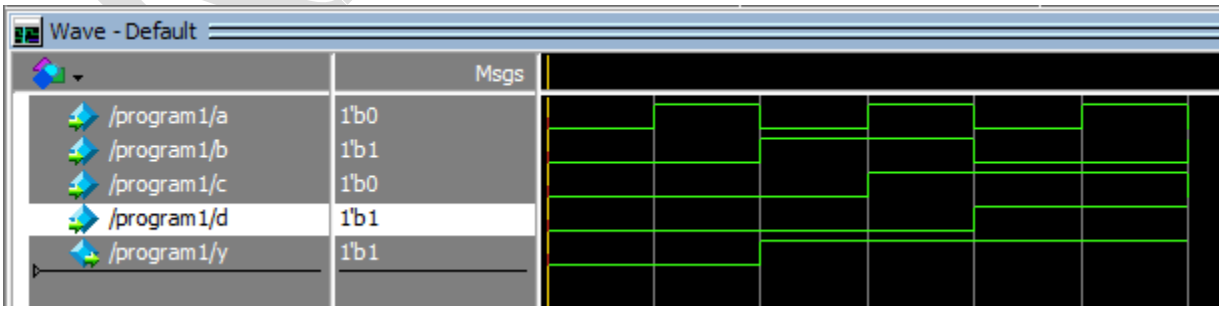
```
module program1(y,a,b,c,d);
input a,b,c,d;
output y;
assign y=(c|(~a&b)|(a&~b&d));
endmodule
```

Test Bench:

```
module program1_tb;
reg a,b,c,d;
wire y;
program1 u1(y,a,b,c,d);
initial
    begin
    a=1'b0;b=1'b0;c=1'b0;d=1'b0;#100
    a=1'b0;b=1'b0;c=1'b1;d=1'b0;#100
    a=1'b1;b=1'b1;c=1'b0;d=1'b0;#100
    a=1'b1;b=1'b1;c=1'b1;d=1'b1;
    end
endmodule
```

Result:

Simulated Waveform:



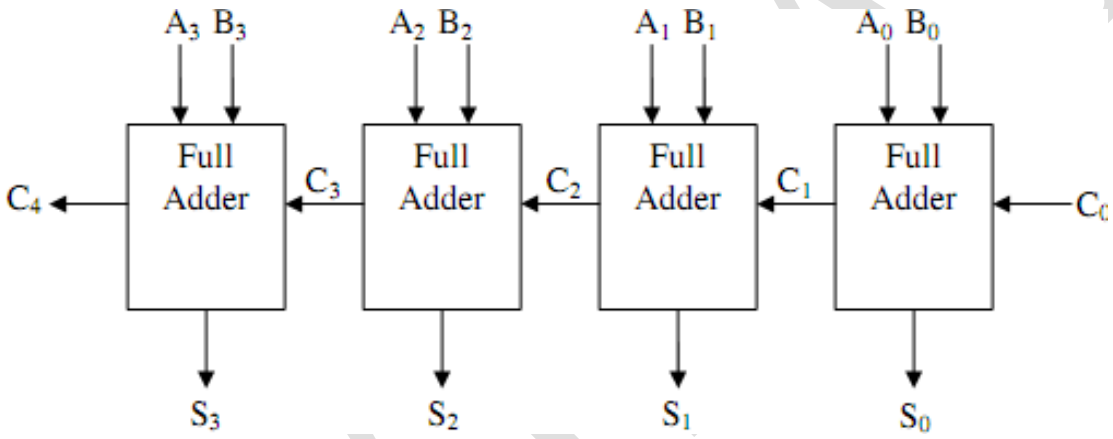
Program 2:

Design a 4-bit full adder and subtractor and simulate the same usingbasic gates.

4-bit full adder/parallel adder

A 4-bit parallel adder is a digital circuit that adds two 4-bit binary numbers. It consists of multiple full adders connected in series, where each full adder handles the addition of corresponding bits along with the carry from the previous bit.

Block Diagram:

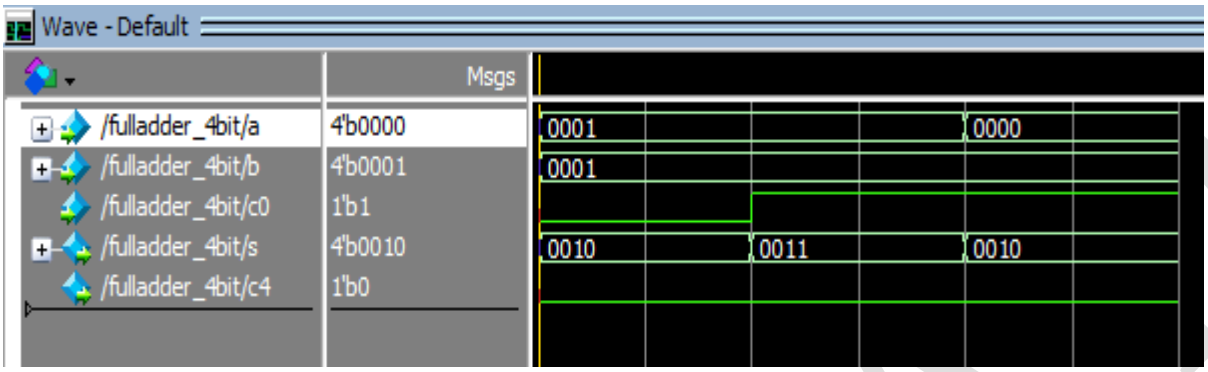


Verilog code:

```
module fulladder_4bit(s,c4,a,b,c0);
input [3:0]a;
input [3:0]b;
input c0;
output [3:0]s;
output c4;
wire c1,c2,c3;
full_adder fa0(s[0],c1,a[0],b[0],c0);
full_adder fa1(s[1],c2,a[1],b[1],c1);
full_adder fa2(s[2],c3,a[2],b[2],c2);
full_adder fa3(s[3],c4,a[3],b[3],c3);
endmodule
```

Result:

Simulated Waveform:



Program 3:

Design Verilog HDL to implement simple circuits using structural, Data flow and Behavioral model.

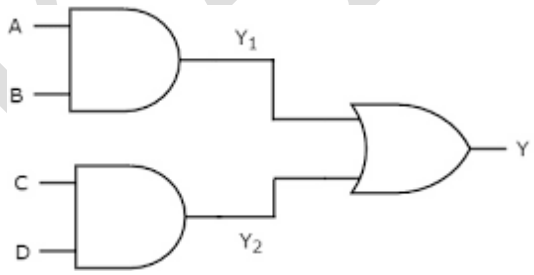
Truth Table:

INPUTS				OUTPUTS		
A	**b**	**c**	**d**	**ab**	**cd**	**Y=ab+cd**
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	1	0	0	0	0
0	0	1	1	0	1	1
0	1	0	0	0	0	0
0	1	0	1	0	0	0
0	1	1	0	0	0	0
0	1	1	1	0	1	1
1	0	0	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	0	0	0
1	0	1	1	0	1	1
1	1	0	0	1	0	1
1	1	0	1	1	0	1
1	1	1	0	1	0	1
1	1	1	1	1	1	1

Logic expression/Boolean Function:

y(a,b,c,d)=∑(3,7,11,12,13,14,15)= ab+cd

Logic circuit:



Verilog code:

Dataflow modelling :

```
module simple_circuit( y,a,b,c,d);
input a,b,c,d;
output y;
assign y=(a&b)|(c&d);
endmodule
```

Behavioral Modelling

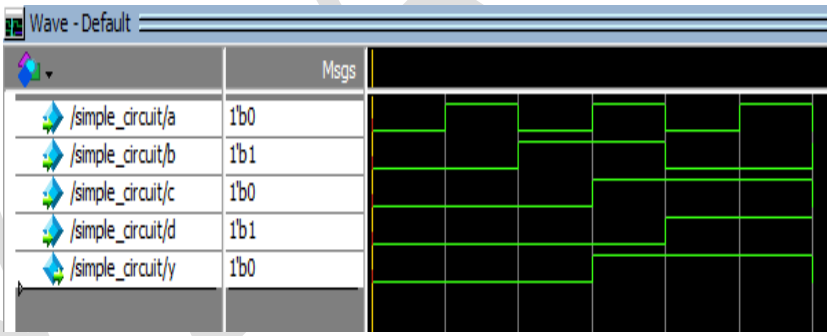
```
module simple_circuit( y,a,b,c,d);
input a,b,c,d;
output y;
reg y;
always@(a,b,c,d)
    begin
          y=(a&b)|(c&d);
     end
endmodule
```

Structural Modelling:

```
module simple_circuit( y,a,b,c,d);
input a,b,c,d;
output y;
wire y1,y2;
and G1(y1,a,b);
and G2(y2,c,d);
or G3(y,y1,y2);
endmodule
```

Result:

Simulated Waveform:



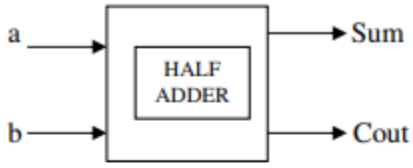
Program 4:

Design Verilog HDL to implement Binary Adder-Subtractor – Half and Full Adder, Half and Full Subtractor.

1. Half adder:

A half adder is a digital circuit that adds two single-bit binary numbers and produces two outputs: the sum and the carry.

BLOCK DIAGRAM:

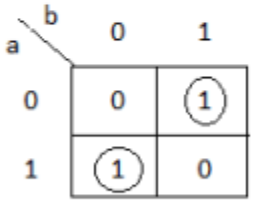


TRUTH TABLE :

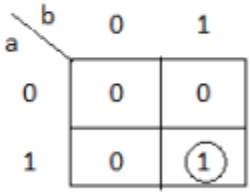
Input		Output	
a	b	Sum	Cout
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

K –MAPS:

Sum:



Cout:

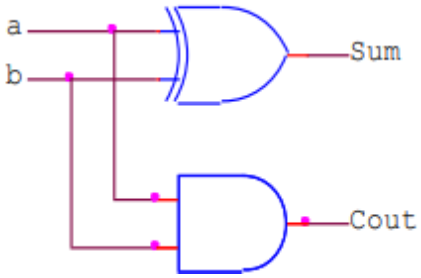


EQUATIONS:

Sum= a ⊕ b

Cout= a.b

LOGIC CIRCUIT:



Verilog code (Data flow modelling):

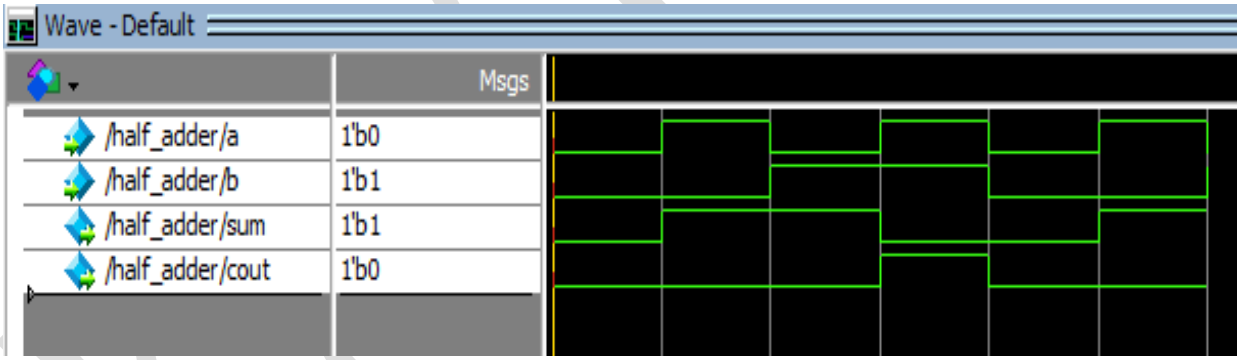
```
module half_adder (sum, cout, a, b);
input a, b;
output sum, cout;
assign sum=a^b;
assign cout=a&b;
endmodule
```

Test Bench

```
module halfadder_tb;
reg a,b;
wire sum,cout;
half_adder u1(sum,cout,a,b);
begin
a=1'b0;b=1'b0;#100
a=1'b0;b=1'b1;#100
a=1'b1;b=1'b0;#100
a=1'b1;b=1'b1;
end
endmodule
```

Result:

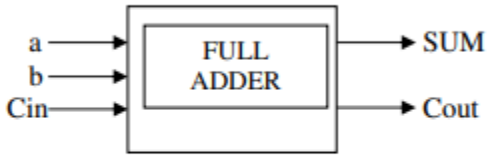
Simulated Waveform:



2. Full adder:

Full adder is an arithmetic combinational logic circuit that performs addition of three single bits. It contains three inputs (a, b, Cin) and produces two outputs (Sum and Cout) where, Cin is Carry In and Cout is Carry Out.

BLOCK DIAGRAM:



TRUTH TABLE:

Input			Output	
a	b	Cin	Sum	Cout
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

K –MAPS:

Sum:

a \ b Cin	00	01	11	10
0	0	(1)	0	(1)
1	(1)	0	(1)	0

Cout:

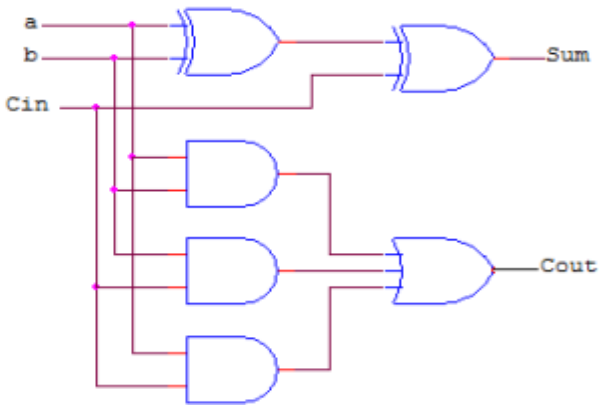
a \ b Cin	00	01	11	10
0	0	0	1	0
1	0	1	1	1

EQUATIONS:

$$Sum = a \oplus b \oplus Cin$$

$$Cout = a\,b + b\,Cin + a\,Cin$$

LOGIC CIRCUIT:



Verilog code:

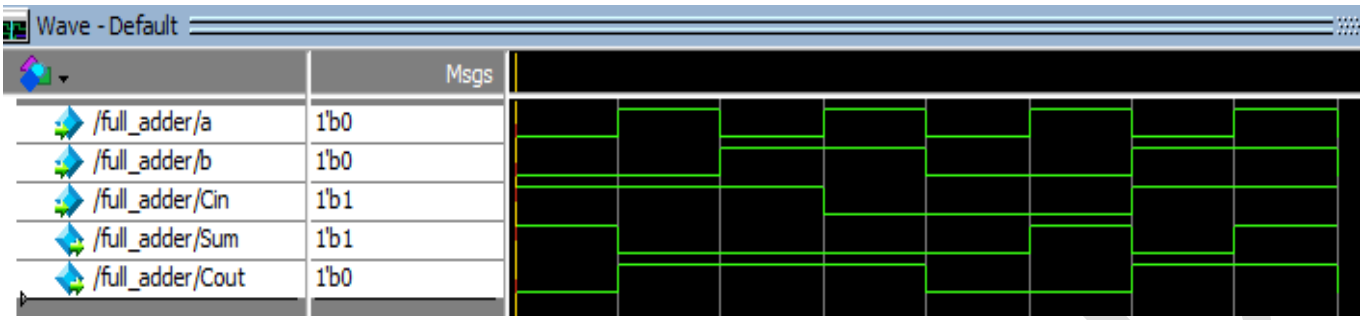
```
module full_adder(Sum,Cout, a,b,Cin);
input a,b,Cin;
output Sum,Cout;
assign Sum= a ^b^Cin;
assign Cout = (a&b) | (b&Cin) | (a&Cin);
endmodule
```

Test Bench:

```
module fulladder_tb ;
reg a,b,Cin;
wire Sum,Cout;
full_adder u1(Sum,Cout, a,b,Cin);
initial
begin
    a=1'b0;b=1'b0;Cin=1'b0;#100
    a=1'b0;b=1'b0;Cin=1'b1;#100
    a=1'b0;b=1'b1;Cin=1'b0;#100
    a=1'b0;b=1'b1;Cin=1'b1;#100
    a=1'b1;b=1'b0;Cin=1'b0;#100
    a=1'b1;b=1'b0;Cin=1'b1;#100
    a=1'b1;b=1'b1;Cin=1'b0;#100
    a=1'b1;b=1'b1;Cin=1'b1;

end

endmodule
```

Result:

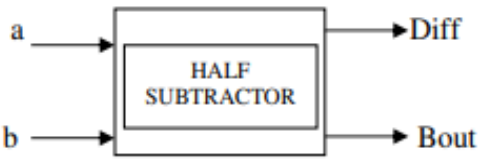
Simulated Waveform:



3. Half Subtractor:

Half subtractor is a combinational logic circuit designed to perform the subtraction of two single bits. It contains two inputs (a and b) and produces two outputs (Difference and Borrow-out).

BLOCK DIAGRAM:

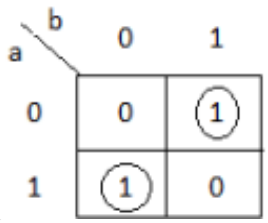


TRUTH TABLE:

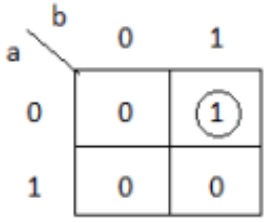
Input		Output	
a	b	Diff	Bout
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

K –MAPS:

Diff:



Bout:

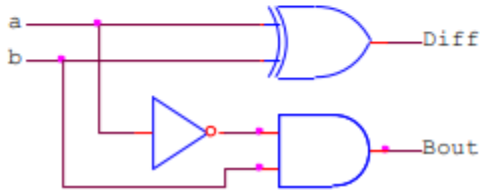


EQUATIONS:

Diff = a ⊕ b

Bout= a`b

LOGIC DIAGRAM:

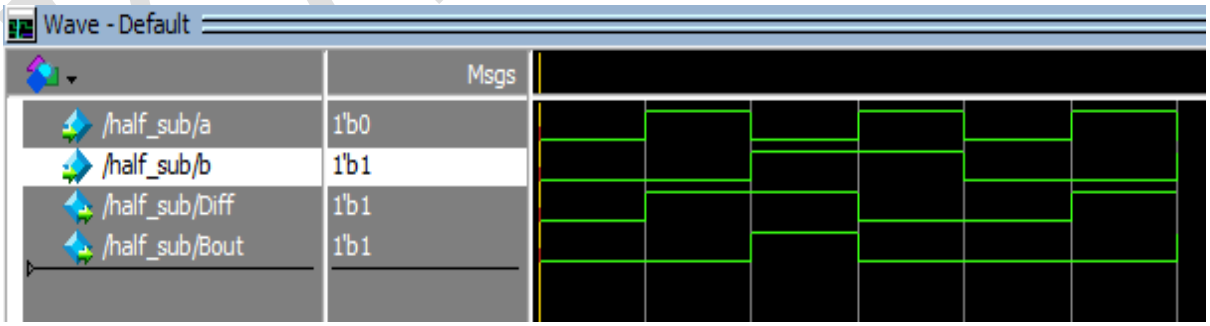


Verilog Code:

```
module half_sub(a,b, Diff,Bout);
input a,b;
output Diff,Bout;
assign Diff= a ^ b;
assign Bout = ~a & b;
endmodule
```

Result:

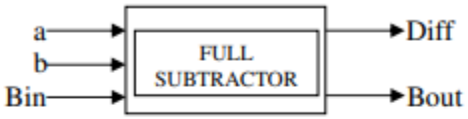
Simulated Waveform:



4. Full Subtractor

Full subtractor is a Combinational logic circuit designed to perform subtraction of three single bits. It contains three inputs(a, b, Bin) and produces two outputs (Diff, Bout) where Bin is Borrow-In and Bout is Borrow-Out.

BLOCK DIAGRAM:



TRUTH TABLE :

Input			Output	
a	b	Bin	Diff	Bout
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

K –MAP:

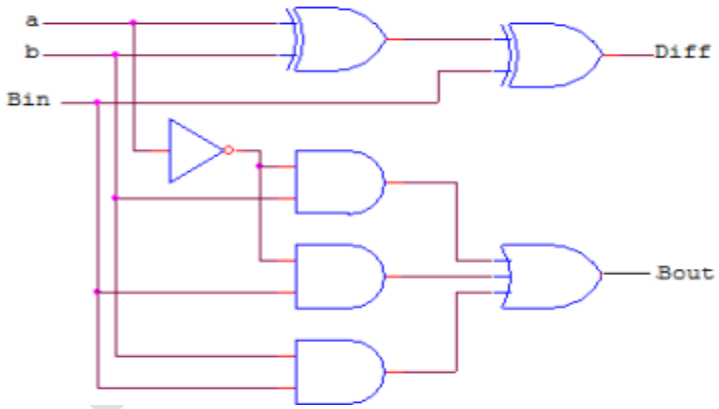
Diff:

a \ b Bin	00	01	11	10
0	0	(1)	0	(1)
1	(1)	0	(1)	0

Bout:

a \ b Bin	00	01	11	10
0	0	1	1	1
1	0	0	1	0

LOGIC CIRCUIT:



EQUATIONS:

Diff = a ⊕ b ⊕ Bin

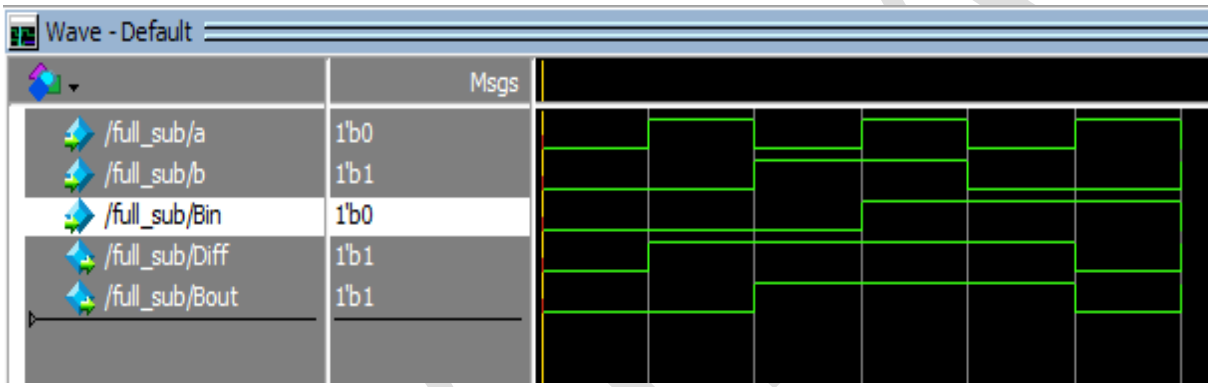
Bout = a` b +a` Bin + b Bin

Verilog Code:

```
module full_sub(a,b,Bin, Diff,Bout);
input a,b,Bin;
output Diff, Bout;
assign Diff= a ^b ^Bin;
assign Bout = (~a & b) | (~a & Bin) | (b & Bin)
endmodule
```

Result:

Simulated Waveform:



Program 5:

Design Verilog HDL to implement Decimal adder.

BCD numbers:

BCD stands for binary coded decimal. It is used to perform the addition of BCD numbers. A BCD digit can have any of ten possible four-bit representations. Suppose, we have two 4-bit numbers A and B. The value of A and B can vary from 0(0000 in binary) to 9(1001 in binary) because we are considering decimal numbers.

A BCD adder is a circuit for the addition of two binary-coded decimal numbers. BCD is another format used in representing numbers where each digit will be represented using a 4-bit binary code.

When adding BCD numbers, if the sum of two BCD digits is greater than 9, the result is greater than 1001 in binary and hence is not valid in BCD. A correction needs to be performed by adding 0110 (6 in BCD) to the sum to get the correct BCD result.

Decimal Digit	BCD code 8 4 2 1
0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
4	0 1 0 0
5	0 1 0 1
6	0 1 1 0
7	0 1 1 1
8	1 0 0 0
9	1 0 0 1

1. **Input:**
   - $A = 6$ (BCD: 0110_2)
   - $B = 5$ (BCD: 0101_2)
2. **Binary Addition:**

$$0110_2 + 0101_2 = 1011_2$$

The sum is 11 in decimal, which is greater than 9.

3. **Correction:** Add 6 to the result:

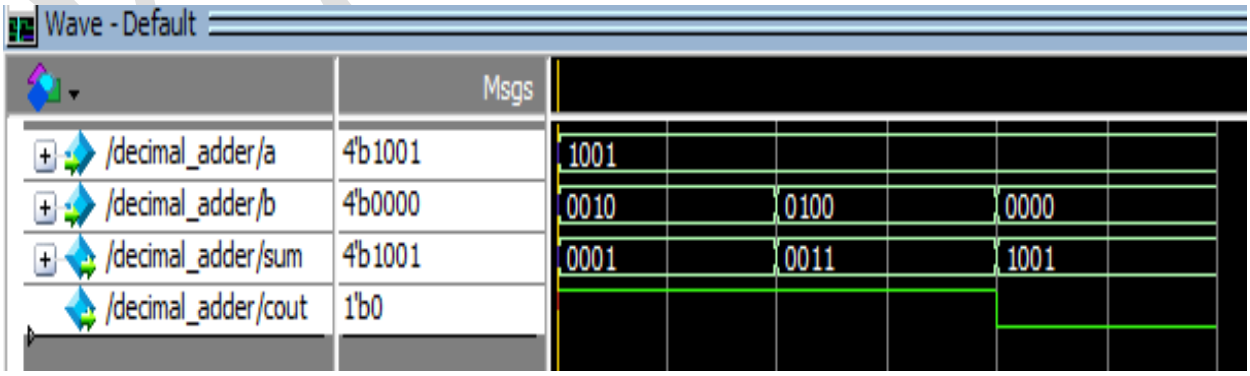
$$1011_2 + 0110_2 = 10001_2$$

Verilog code:

```
module p5deci(a,b,sum,cout);
input [3:0] a;
input [3:0] b;
output [3:0] sum;
output cout;
reg [3:0] sum;
reg cout;
always@ (a,b)
begin
    {cout,sum} = a+b;
    if(a>9 || b>9 || sum>9)
    begin
    {cout,sum} = sum+6;
end
end
endmodule
```

Result:

Simulated Waveform:

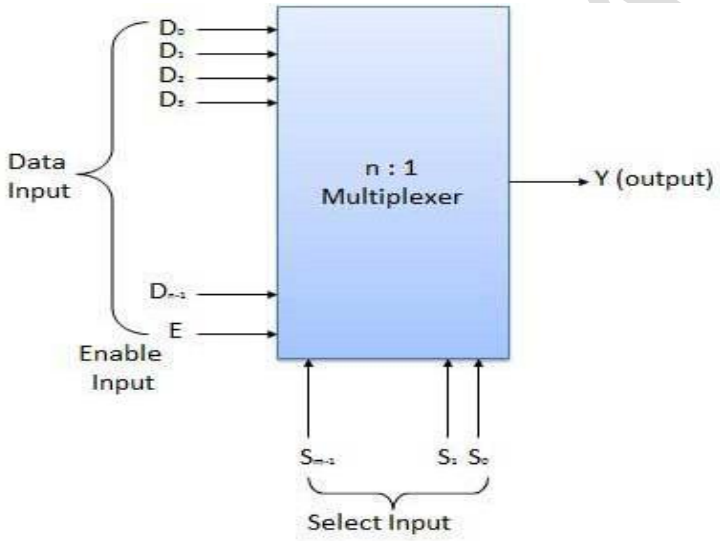


Program 6

Design Verilog program to implement Different types of multiplexer like 2:1, 4:1 and 8:1.

Multiplexer:

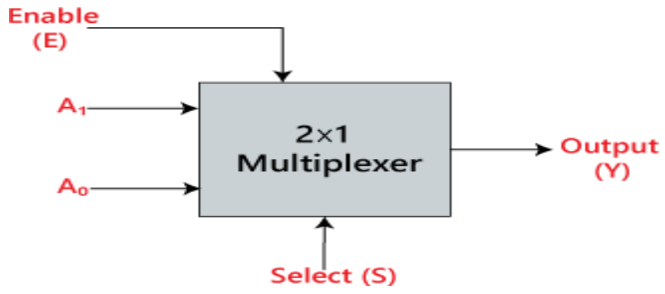
A multiplexer is a combinational circuit that has 2n input lines and a single output line. Simply, the multiplexer is a multi-input and single-output combinational circuit. The binary information is received from the input lines and directed to the output line. On the basis of the values of the selection lines, one of these data inputs will be connected to the output.



2×1 Multiplexer:

In 2×1 multiplexer, there are only two inputs, i.e., A0 and A1, 1 selection line, i.e., S0 and single outputs, i.e., Y. On the basis of the combination of inputs which are present at the selection line S0, one of these 2 inputs will be connected to the output. The block diagram and the truth table of the 2×1 multiplexer are given below.

Block Diagram:



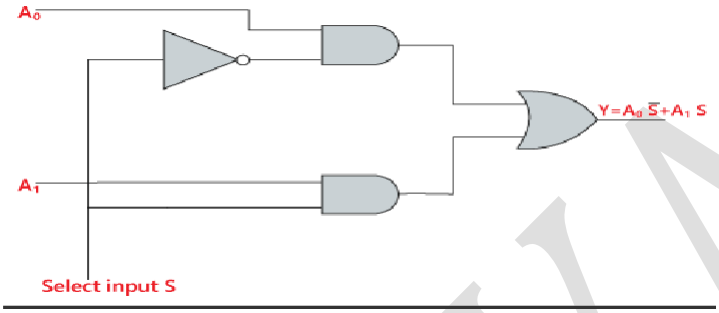
Truth Table:

INPUTS	Output
S_0	Y
0	A_0
1	A_1

The logical expression of the term Y is as follows:

$Y=S_0'.A_0 + S_0. A_1$

Logical circuit:

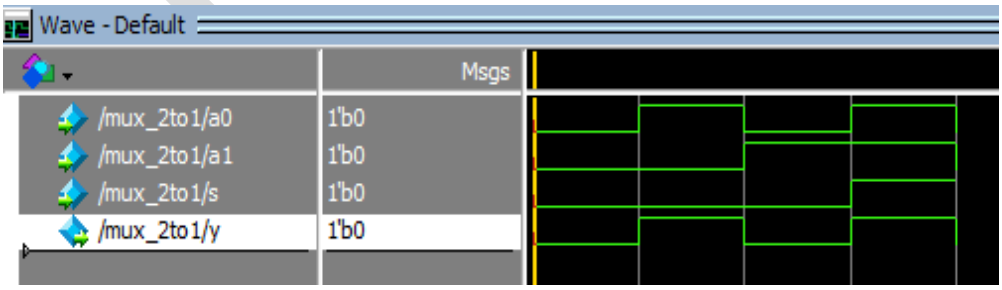


Verilog Code:

```
module mux_2to1(y,a0,a1,s);
input a0,a1,s;
output y;
reg y;
always @ (a0,a1,s)
begin
      y=((~s & a0)|(s & a1));
end
endmodule
```

Result:

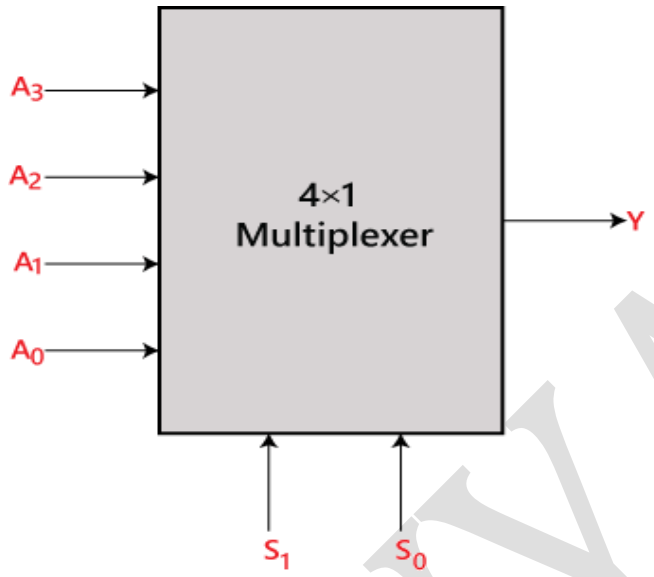
Simulated Waveform:



4×1 Multiplexer:

In the 4×1 multiplexer, there is a total of four inputs, i.e., A0, A1, A2, and A3, 2 selection lines, i.e., S0 and S1 and single output, i.e., Y. On the basis of the combination of inputs that are present at the selection lines S0 and S1, one of these 4 inputs are connected to the output. The block diagram and the truth table of the 4×1 multiplexer are given below.

Block Diagram:



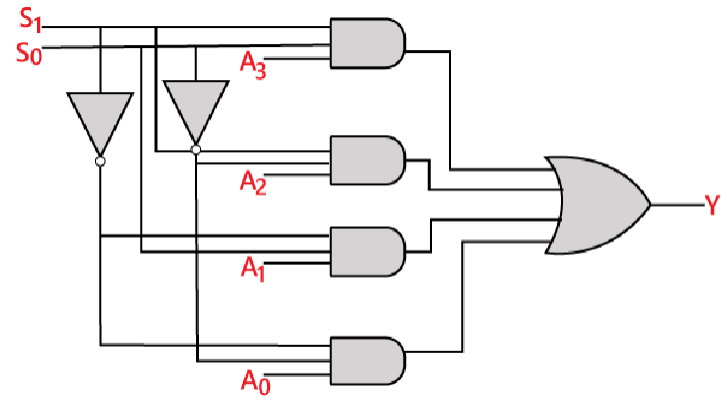
Truth Table:

INPUTS		Output
S_1	S_0	Y
0	0	A_0
0	1	A_1
1	0	A_2
1	1	A_3

The logical expression of the term Y is as follows:

Y=S1' S0' A0+S1' S0 A1+S1 S0' A2+S1 S0 A3

Logical circuit:

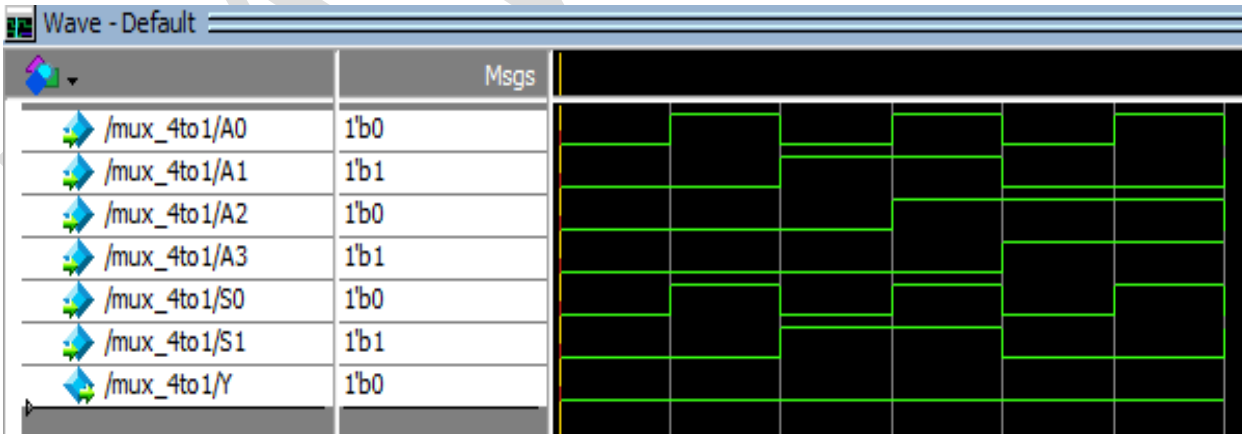


Verilog code:

```
module mux_4to1(Y,A0,A1,A2,A3,S0,S1);
input A0,A1,A2,A3,S0,S1;
output Y;
reg Y;
always @ (A0,A1,A2,A3,S0,S1)
begin
      Y=(~S1&~ S0& A0)|(~S1& S0& A1)|(S1&~ S0& A2)|(S1&S0&A3)
end
endmodule
```

Result:

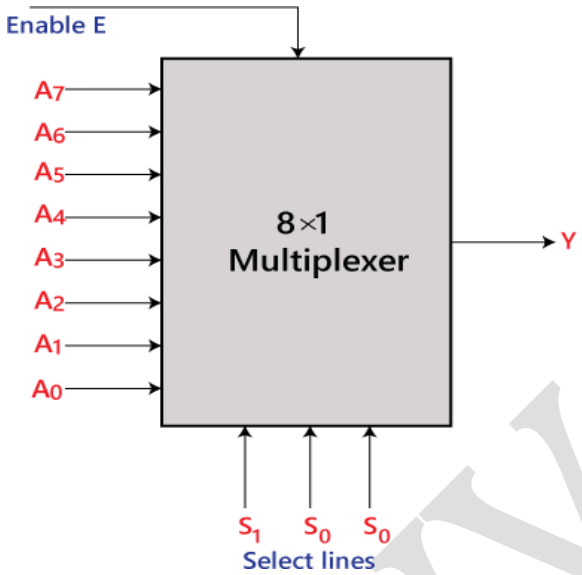
Simulated Waveform:



8 to 1 Multiplexer:

In the 8 to 1 multiplexer, there are total eight inputs, i.e., A0, A1, A2, A3, A4, A5, A6, and A7, 3 selection lines, i.e., S0, S1and S2 and single output, i.e., Y. On the basis of the combination of inputs that are present at the selection lines S0, S1, and S2, one of these 8 inputs are connected to the output. The block diagram and the truth table of the 8×1 multiplexer are given below.

Block Diagram:



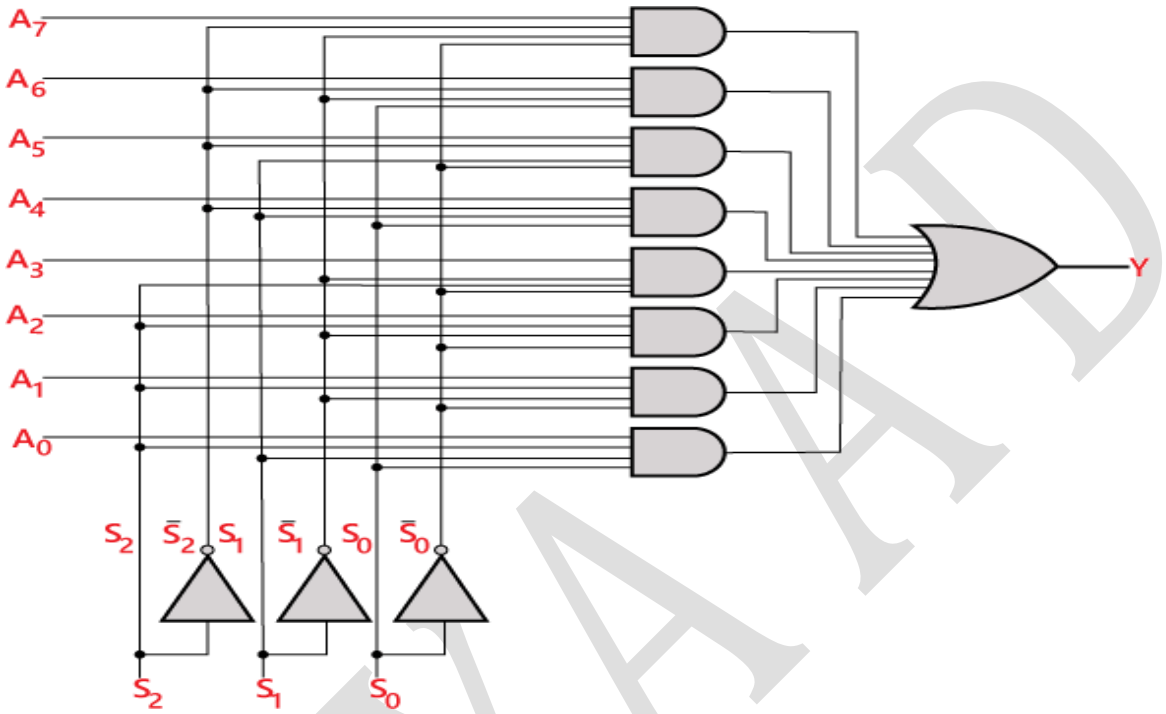
Truth Table:

INPUTS			Output
S_2	S_1	S_0	Y
0	0	0	A_0
0	0	1	A_1
0	1	0	A_2
0	1	1	A_3
1	0	0	A_4
1	0	1	A_5
1	1	0	A_6
1	1	1	A_7

The logical expression of the term Y is as follows:

Y=S0'.S1'.S2'.A0+S0.S1'.S2'.A1+S0'.S1.S2'.A2+S0.S1.S2'.A3+S0'.S1'.S2A4+S0.S1'.S2A5+S0'.S1.S2.A6+S0.S1.S3.A7

Logical circuit:

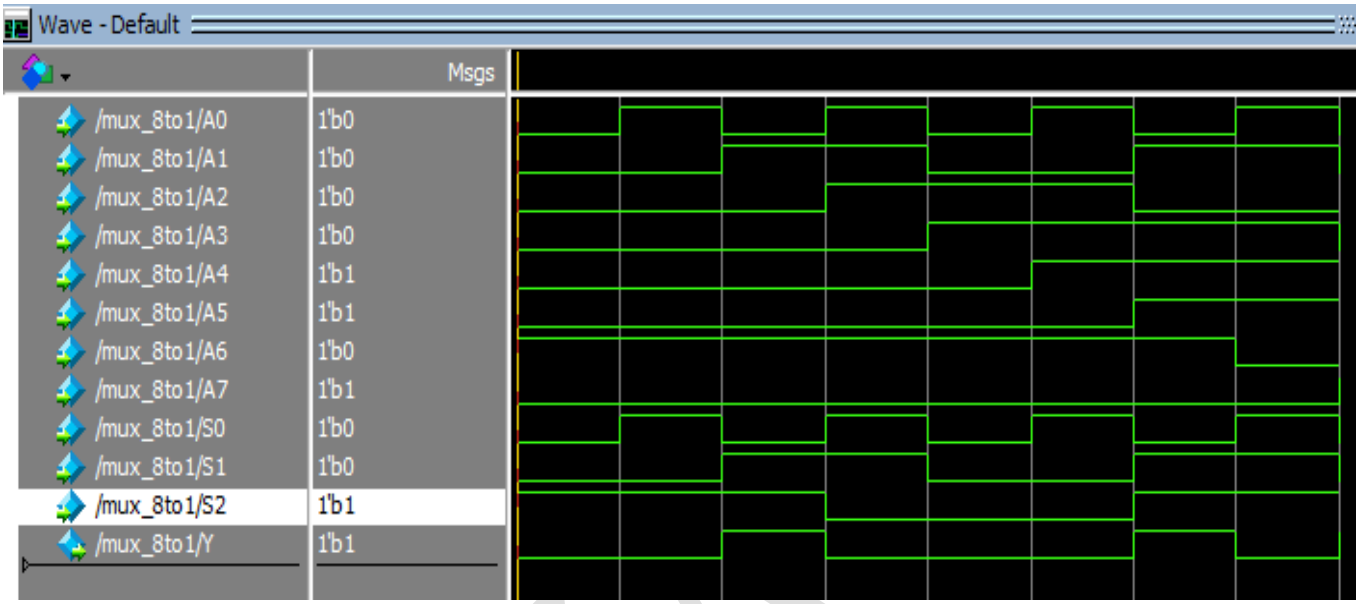


Verilog code:

```
module mux_8to1(Y,A0,A1,A2,A3,A4,A5,A6,A7,S0,S1,S2);
input A0,A1,A2,A3,A4,A5,A6,A7,S0,S1,S2;
output Y;
reg Y;
always @ (A0,A1,A2,A3,A4,A5,A6,A7,S0,S1,S2)
begin
      Y= (~S2&~S1&~ S0& A0)| (~S2&~S1& S0& A1)|( ~S2&S1&~ S0& A2)|(
~S2&S1&S0&A3)|
         (S2&~S1&~ S0& A4)| (S2&~S1& S0& A5)| (S2&S1&~ S0& A6)| (~S2&~S1&~ S0&
A0);
end
endmodule
```

Result:

Simulated Waveform:



Program 7:

Design Verilog program to implement types of De-Multiplexer.

De-multiplexer (DEMUX):

A Demultiplexer is a combinational logic circuit that accepts a single input and distributes it over several output lines. Demultiplexer is also termed as DEMUX in short. As Demultiplexer is used to transmit the same data to different destinations, hence it is also known as data distributor.

Block Diagram:

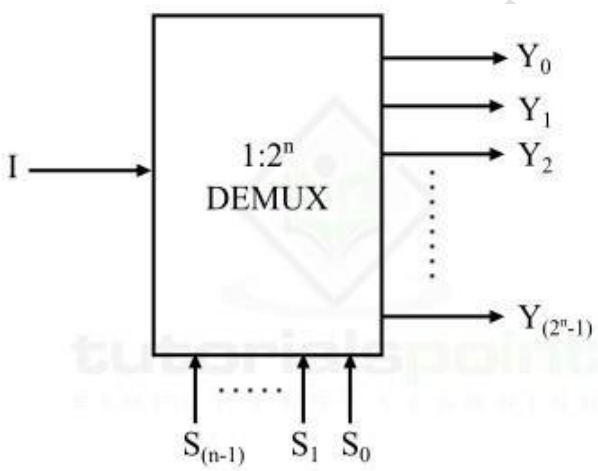


Figure 1 - Demultiplexer

1×2 Demultiplexer:

Block Diagram:

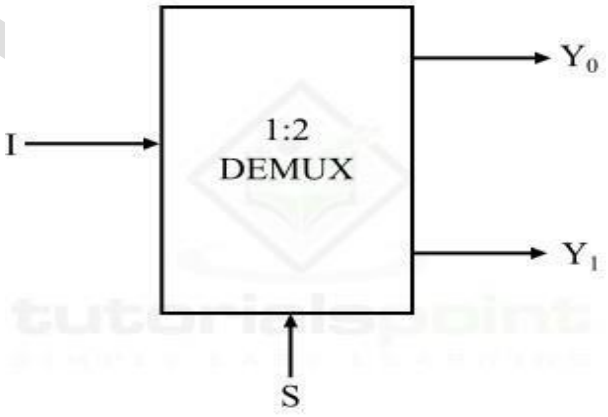


Figure 2 - 1:2 Demultiplexer

Truth table:

Select line	outputs	
S	Y1	Y0
0	0	I
1	I	0

Boolean function:

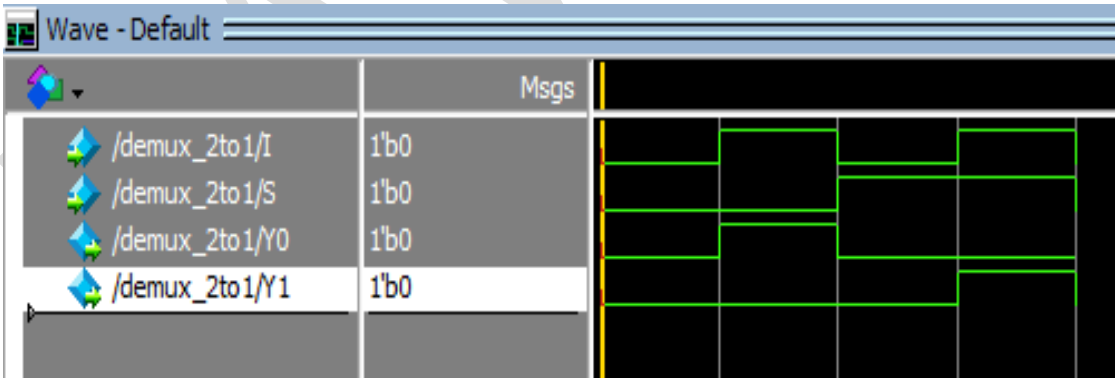
Y0=S'I and Y1=SI

Verilog Code:

```
module demux_2to1(Y0,Y1,S,I);
input I,S;
output Y0,Y1;
reg Y0,Y1;
always @ (S,I)
begin
Y0=(~S& I);
Y1=(S&I);
end
endmodule
```

Result:

Simulated Waveform:



1×4 Demultiplexer:

Block Diagram:

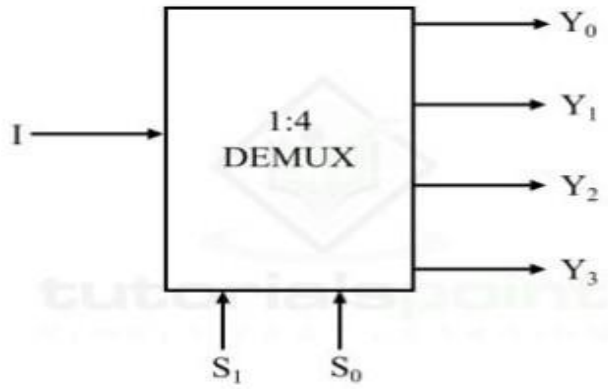


Figure 3 - 1:4 Demultiplexer

Truth table:

Select line		outputs			
S1	S0	Y3	Y2	Y1	Y0
0	0	0	0	0	I
0	1	0	0	I	0
1	0	0	I	0	0
1	1	I	0	0	0

Boolean function:

Y0 = S1'S0'I Y1= S1'S0I Y2= S1S0'I Y3= S1S0I

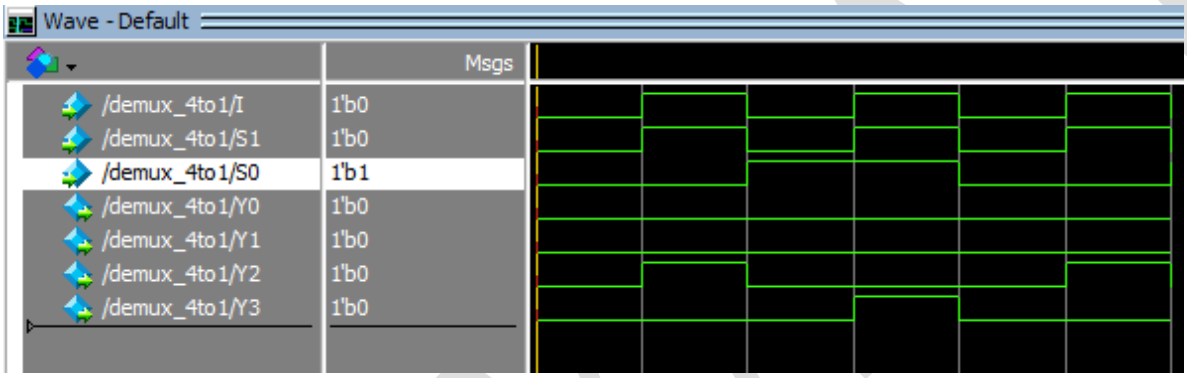
Verilog Code:

```
module demux_4to1(Y0,Y1,Y2,Y3,S1,S0,I);
input I,S1,S0;
output Y0,Y1,Y2,Y3;
reg Y0,Y1,Y2,Y3;
always @ (S1,S0,I)
```

```
begin
Y0=(~S1&~S0& I);
Y1=(~S1&S0& I);
Y2=(S1&~S0& I);
Y0=(S1&S0& I);
end
endmodule
```

Result:

Simulated Waveform:



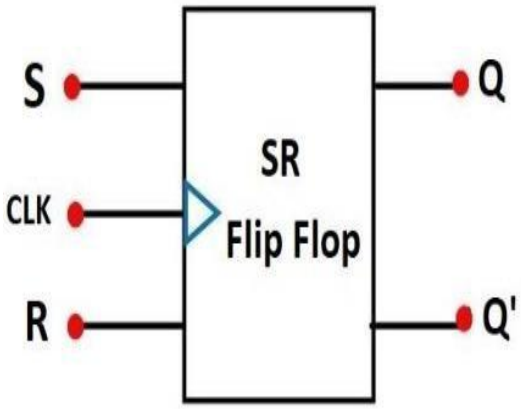
Program 8:

Design Verilog program for implementing various types of Flip-Flops such as SR, JK and D.

Flip-Flops:

A flip-flop is an electronic circuit that can store single-bit binary data either logic 0 or logic 1. Basically, a flip flop is a Bistable multivibrator that changes its output depending on the input. Flip Flops are of two types edge triggered, and level triggered. State of an Edge triggered flip flop changes during the positive or negative edge of a clock cycle. Whereas in the case of level-triggered flip flop output can change during high or low clock duration. Level triggered flip flops are called latch, they are transparent because when they are enabled their output becomes the same as the input.

S R Flip-Flop:



Truth Table:

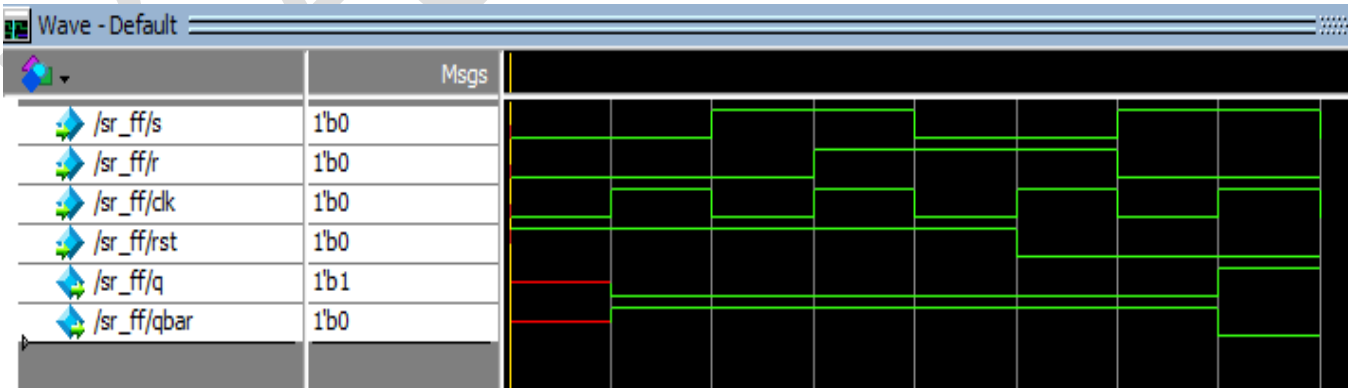
S	R	Q	Q'	
1	0	1	0	
0	0	1	0	After S = 1 and R = 0
0	1	0	1	
0	0	0	1	After S = 0 and R = 1
1	1	0	0	Invalid

Verilog Code:

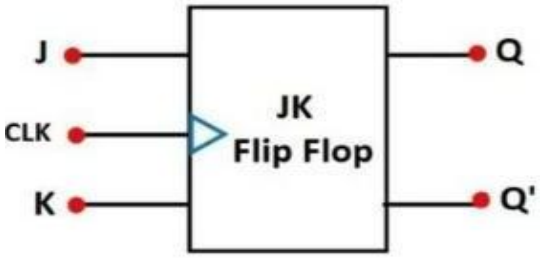
```
module sr_ff(q,qbar,s,r,clk,rst);
input s,r,clk,rst;
output q,qbar;
reg q,qbar;
always@(posedge clk)
begin
if(rst)
q<=1'b0;
else if (s==1'b0 && r==1'b0) q<=q;
else if (s==1'b0 && r==1'b1) q<=1'b0;
else if (s==1'b1 && r==1'b0) q<=1'b1;
else if (s==1'b1 && r==1'b1) q<=1'bx;
assign qbar=~q;
end
endmodule
```

Result:

Simulated Waveform:



JK flip-flop:



Truth table:

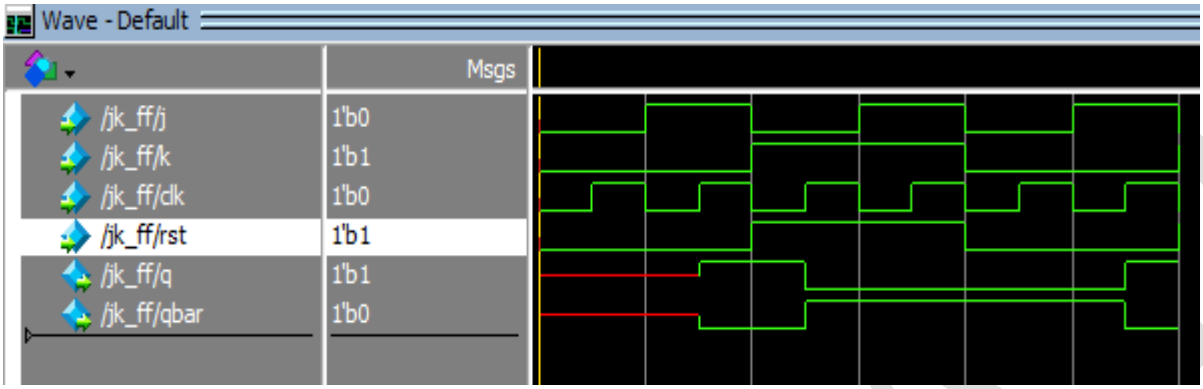
Clk	J	K	Q	Q'	State
1	0	0	Q	Q'	No change in state
1	0	1	0	1	Resets Q to 0
1	1	0	1	0	Sets Q to 1
1	1	1	-	-	Toggles

Verilog Code:

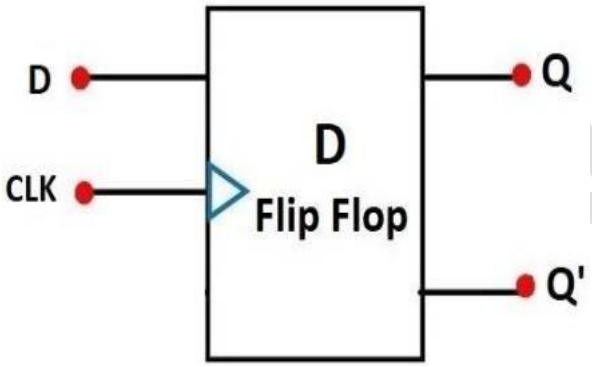
```
module jk_ff(q,qbar ,j,k,clk,rst);
input j,k,clk,rst;
output q,qbar;
reg q,qbar;
always@(posedge clk)
begin
if(rst)
q<=1'b0;
else if (j==1'b0 && k==1'b0) q<=q;
else if (j==1'b0 && k==1'b1) q<=1'b0;
else if (j==1'b1 && k==1'b0) q<=1'b1;
else if (j==1'b1 && k==1'b1) q<= ~q;
assign qbar = ~q;
end

endmodule
```

Result:

Simulated Waveform:



D Flip-Flop:



Truth Table:

D	CLK	Q	Q`
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

Verilog Code:

```
module d_ff(q,d,clk);
input d,clk;
output q;
reg q;
always @(posedge clk)
begin
q<=d;
end
endmodule
```

Result:

Simulated Waveform:

VIVA QUESTIONS

1. Define a logic gate.
2. What are basic gates?
3. Why NAND and NOR gates are called as universal gates?
4. State De morgans theorem
5. Give examples for SOP and POS
6. Explain how transistor can be used as NOT gate
7. Realize logic gates using NAND and NOR gates only
8. List the applications of EX-OR and EX~NOR gates
9. What is a half adder?
10. What is a full adder?
11. Differentiate between combinational and sequential circuits. Give examples
12. Give the applications of combinational and sequential circuits
13. Define flip flop
14. What is an excitation table?
15. What is race around condition?
16. How do you eliminate race around condition?
17. What is minterm and max term?
18. Define multiplexer/ data selector
19. What is a demultiplexer?
20. Give the applications of mux and demux
21. What is a encoder and decoder?
22. Compare mux and encoder
23. Compare demux and decoder
24. What is a priority encoder?
25. What are counters? Give their applications.