VISVESVARAYA TECHNOLOGICAL UNIVERSITY

JNANA SANGAMA,BELGAVI-590018,KARNATAK

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

LAB MANUAL

Course: Digital Designand Computer Organization

Course Code: BCS302

III-SEMESTER

Prepared by:**Pro. Neha Deshmukh**
Assistant Professor,
Department of Computer Science and Engineering VTUCPGS

ACADEMIC YEAR:2025-26

VISVESVARAYA TECHNOLOGICAL UNIVERSITY,BELAGAVI

B.E. in Computer Science and Engineering

Scheme of Teaching and Examinations 2022

Outcome-Based Education(OBE)and Choice Based Credit System(CBCS)

(Effective from the academic year 2025 - 26)

Sl. No	Experiments
1	Givena4-variable logic expression,simplify it using appropriate technique and simulate The same using basic gates.
2	Designa 4b it fulladder and subtractor and simulate the same using basic gates.
3	Design Verilog HDL to implement simple circuits using structural,Data flow and Behavi oural model.
4	Design Verilog HDL to implement Binary Adder-Subtractor–Halfand FullAdder,Half and Full Subtractor.
5	Design Verilog HDL to implement Decimal adder.
6	Design Verilog program to implement Different types of multiplexer like2:1,4:1and8:1.
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.

Procedure:

1. Open a Xilinx IDE,close old projects.
2. Create a new project through the project navigator icon.
3. AddVHDL/Verilognewsourcetotheprojectdependingontheuserrequirement.
4. Select the Verilog HDL mode.
5. Assign the inputs and outputs for the system to design.
6. And write the code as given below for the functionality.
7. Synthesize the code and correct the syntax errors if any.
8. Is imSimulator and Behavioral check syntax the Simulation behavioral model.
9. Observe the out puttiing wave form and verify it with the truth table.

EXPERIMENT NO–01

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

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

Objectives:

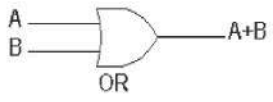
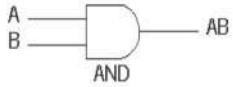
1. Simplify4-variablelogicexpressionforefficientdesign.
2. Simulate using basic gates for practical verification and optimization.

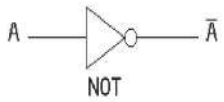
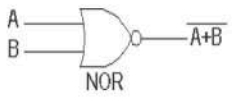
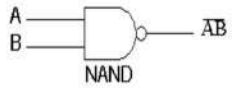
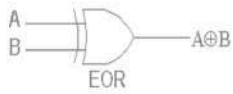
Theory:

SUMOF PRODUCT:

The short form of the sum of the productis SOP,and it is one kind of **Boolean algebra** expression. In this, the different product inputs are being added together. The product of inputs is Boolean logical AND whereas the sum or addition is Boolean logical OR. Before going to understand the concept of the sum of products

Expression of SOPF=AB+C(D+E)

Gate	Description	Truth Table			Logic Symbol
OR	The out put is active High if any one of the input is inactive high state,Mathematically, Q = A+B	A	B	Output	
		0	0	0	
		0	1	1	
		1	0	1	
		1	1	1	
AND	The out put is active High only if both the Inputs are inactive high state, Mathematically,	A	B	Output	
		0	0	0	
		0	1	0	
		1	0	0	
		1	1	1	

NOT	In this gate the output is opposite to the input state,Mathematically, Q=A	A 0 1		Output 1 0	
NOR	The output is active High only if both the Inputs are inactive low state, Mathematically, Q = (A+B)'	A 0 0 1 1	B 0 1 0 1	Output 1 0 0 0	
NAND	The out put is active High only if any one of the input is in active low state, Mathematically, Q=(A.B)'	A 0 0 1 1	B 0 1 0 1	Output Q 1 1 1 0	
XOR	The out put is active High only if any one of the input is in active High state, Mathematically, Q= A.B'+B.A'	A 0 0 1 1	B 0 1 0 1	Output 0 1 1 0	

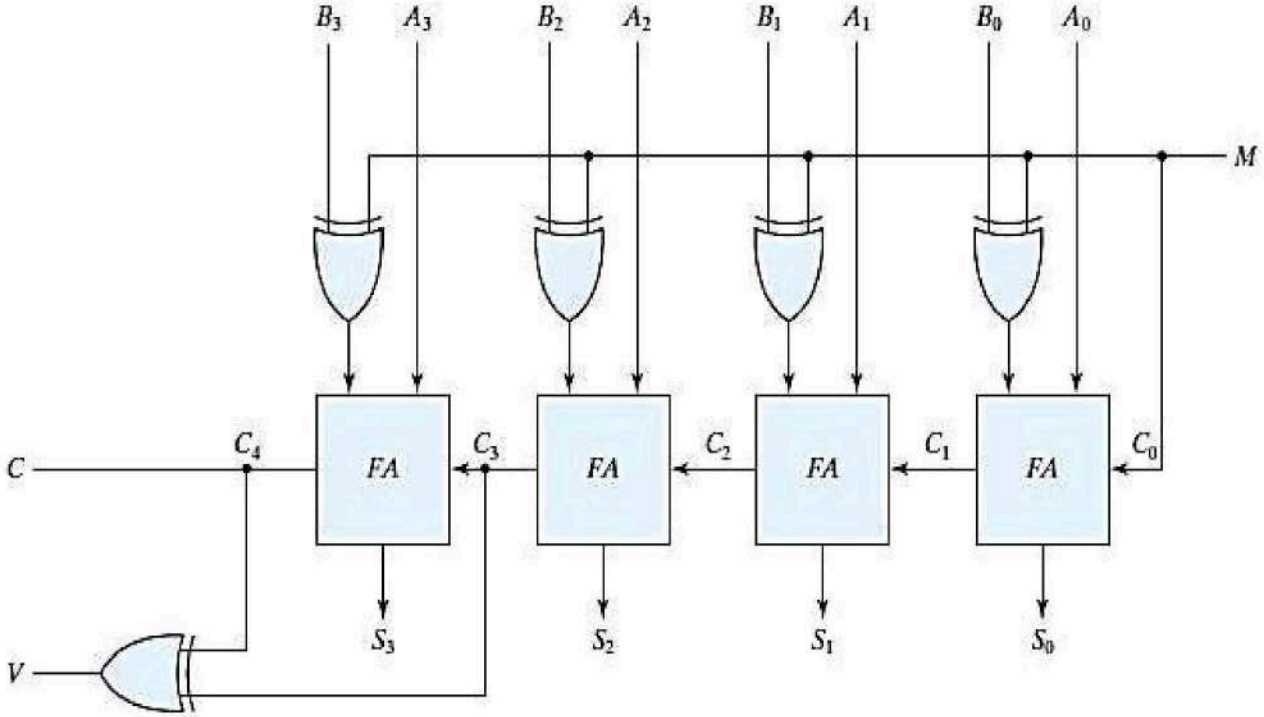
EXPERIMENT NO–02

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

Aim: To design a 4 bit full adder and subtractor and simulate the same using basic gates.

Objectives:

1. Designa4-bitfulladderandsubtractorcircuitinMultisimusingbasicgates.
2. Simulate the circuit to analyze and verify correct functionality,considering various input scenarios.



Theory:

The addition and subtraction operations can becom bined in to one circuit with one common binary adder by including an exclusive-OR gate with each full-adder.

EXPERIMENT NO–03

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

Aim: To design Verilog HDL to implement simple circuits using structural,Data flow and Behavioural model.

Objectives:

1. Implement Verilog for simple circuits using structural,data flow,and behavioral models.
2. Verify designs through simulation for functionality and performance assessment.

Theory:

As mentioned previously,the module is the basic building block for modeling hardware with the Verilog HDL. The logic of a module can be described in anyone (or a combination) of the following modeling styles:

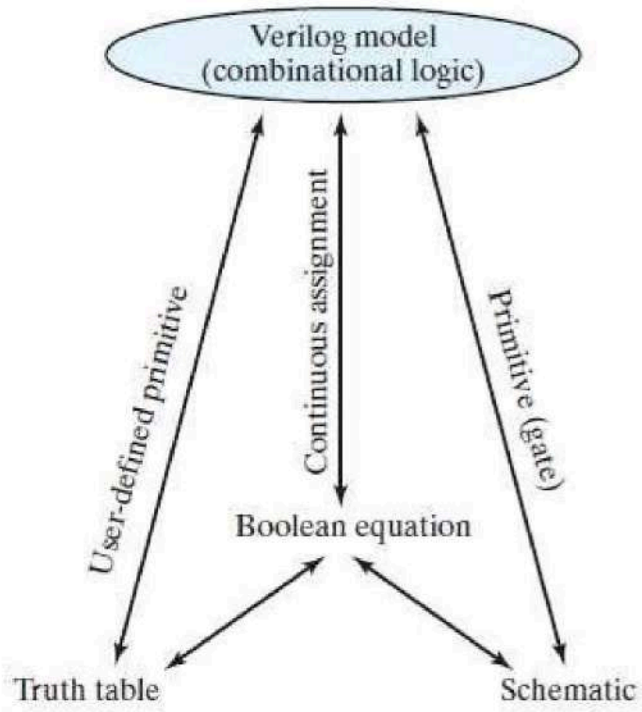


FIGURE3.1:RelationshipofVerilogconstructstotruthtables,Booleanequations,and schematics

- **Gate-level modeling** using instantiations of predefined and user-defined primitive gates.
  - Gate-level(structural) modeling describes a circuit by specifying its gates and how they are connected with each other.
- **Data flow modeling** using continuous assignment statements with the key word assign.

- Dataflow modeling is used mostly for describing the Boolean equations of combinational logic.

- **Behavioralmodeling**usingproceduralassignmentstatementswiththekeywordalways.
  - Behavioral modeling that is used to describe combinational and sequential circuits at a higher level of abstraction.

Combinational logic can be designed with truth tables,Booleane quations,and schematics;Verilog has a construct corresponding to each of these "classical" approaches to design: user-defined primitives, continuous sassignments,and primitives,asshown in Fig.3.1.There is one other modeling style,called switch-levelmodeling.It is sometimes used in the simulation of MOS transistor circuit models,but not in logic synthesis. We will not consider switch-level modeling.

Structural-	**Data Flow-**	**BehaviouralModel-**

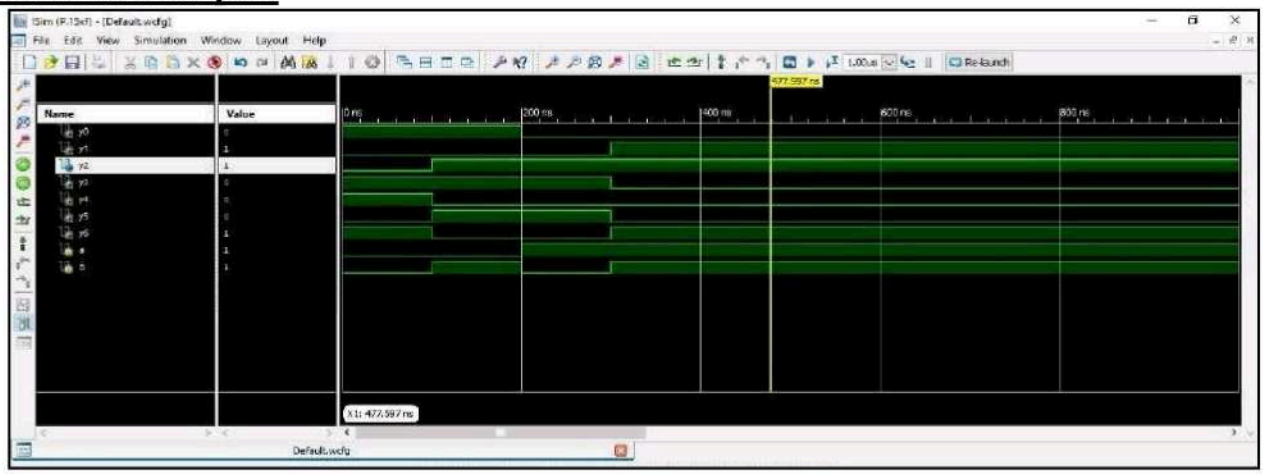
module logicg(a,
b,y0,y1,y2,y3,y4,y5,y6);
input a,b;
outputy0,y1,y2,y3,y4,y5,y6;
not g1(y0,a);
andg2(y1,a,b);
org3(y2,a,b);
nandg4(y3,a,b);
norg5(y4,a,b);
xorg6(y5,a,b);
xnorg7(y6,a,b);
endmodule
	module ldataflow(a,
	b,y0,y1,y2,y3,y4,y5,y6);
	input a,b;
	outputy0,y1,y2,y3,y4,y5,y6;
	assign y0=!a;
	assign y1=a&b;
	assign y2=a|b;
	assigny3=!(a&b);
	assign y4=!(a|b);
	assign y5=a^b;
	assign y6=!(a^b);
	endmodule
		module logicgB(a,
		b,y0,y1,y2,y3,y4,y5,y6);
		input a,b;
		outputy0,y1,y2,y3,y4,y5,y6;
		reg y0;
		regy1;
		regy2;
		regy3;
		regy4;
		regy5;
		regy6;
		always@(a,b) begin
		y0=!a; y1=a&b; y2=a|b;
		y3=!(a&b);
		y4=!(a|b); y5=a^b; y6=!(a^b);
		end endmodule

VerilogTestbenchcode:

Structural-	**Data Flow-**	**BehaviouralModel-**
Module logic g_tb; reg a; reg b; wirey0; wirey1; wirey2; wirey3; wirey4; wirey5; wirey6; logicguut(.a(a),.b(b), .y0(y0),.y1(y1), .y2(y2),.y3(y3), .y4(y4),.y5(y5), .y6(y6)); initialbegin	Moduleldataflow_tb; reg a; reg b; wirey0; wirey1; wirey2; wirey3; wirey4; wirey5; wirey6; ldataflowuut(.a(a),.b(b), .y0(y0),.y1(y1), .y2(y2),.y3(y3), .y4(y4),.y5(y5), .y6(y6)); initialbegin	Module logicgB_tb; reg a; reg b; wirey0; wirey1; wirey2; wirey3; wirey4; wirey5; wirey6; logicgBuut (.a(a),.b(b), .y0(y0),.y1(y1), .y2(y2),.y3(y3),.y4(y4), .y5(y5),.y6(y6)); initialbegin a=0;b=0;#100;

a=0;b=0;#100;
a=0;b=1;#100;
a=1;b=0;#100;
a=1;b=1;#100;
end endmodule
	a=0;b=0;#100;
	a=0;b=1;#100;
	a=1;b=0;#100;
	a=1;b=1;#100;
	end endmodule
		a=0;b=1;#100;
		a=1;b=0;#100;
		a=1;b=1;#100;
		end endmodule

SimulationOutput:



Results&conclusions: The Verilog code for simple circuits using structural, Data flow and Behavioral model are verified.

EXPERIMENT NO–04

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

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

Objectives:

1. Create Verilog HDL for Binary Adder-Subtractor,including Half and Full Adder,Half and Full Subtractor.
2. Validatefunctionalitythroughsimulationforaccuratearithmeticoperationsindigitalsystems.

Theory:

Adder is a combinational digital circuit that is used for adding two numbers. A typical adder circuit produces a sum bit (denoted by S) and a carry bit (denoted by C) as the output. Adder circuits are of two types: Half adder and Full adder.

Half-Adder:Acombinational logic circuit that performs the addition of two data bits,A and B,is called a half-adder. Addition will result in two output bits;one of which is the sum bit, S,and the other is the carry bit, C.

Full-Adder: The half-adder does not take the carry bit from its previous stage into account. This carry bit from its previous stage is called carry-in bit. A combinational logic circuit that adds two data bits, A and B, and a carry-in bit Cin, is called a full-adder.

Subtractor is a combinational digital circuit that is used for subtracting two numbers. A typical subtractor circuit produces a diff bit (denoted by D) and a borrow bit (denoted by B)as the output. Subtractor circuits are of two types: Half subtractor and Full subtractor.

Half Subtractor: Acombinational logic circuit that performs the subtraction of two data bits, A and B, is called a half-adder. Subtraction will result in two output bits;one of which is the diff bit, D, and the other is the borrow bit, B.

Full Subtractor: A combinational logic circuit that subtractor two data bits, A and B and a borrow in bit, is called full subtractor.

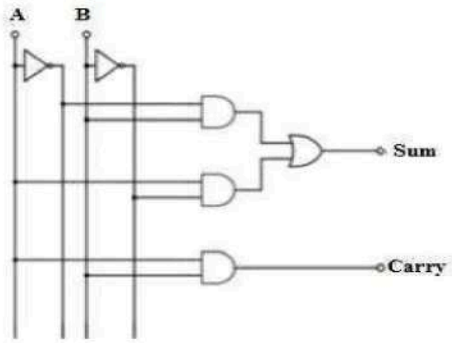
Using Basic Gates

Half Adder

Half Adder

Input		output	
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

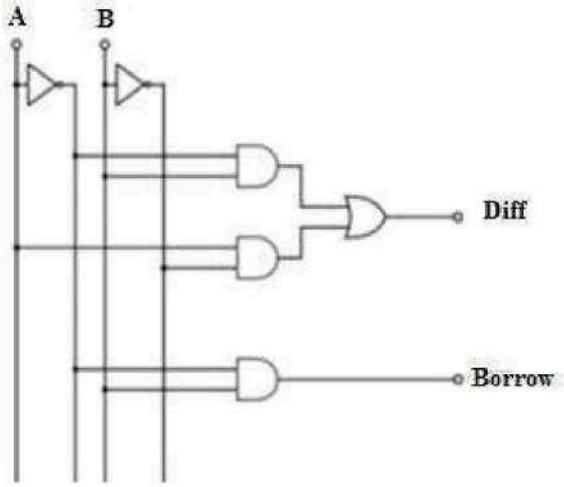
Sum= $\overline{A}B+A\overline{B}$
Carry=AB



HalfSubtractor

Input		output	
A	B	Difference	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Difference= $\overline{A}B+A\overline{B}$
Borrow= $\overline{A}B$

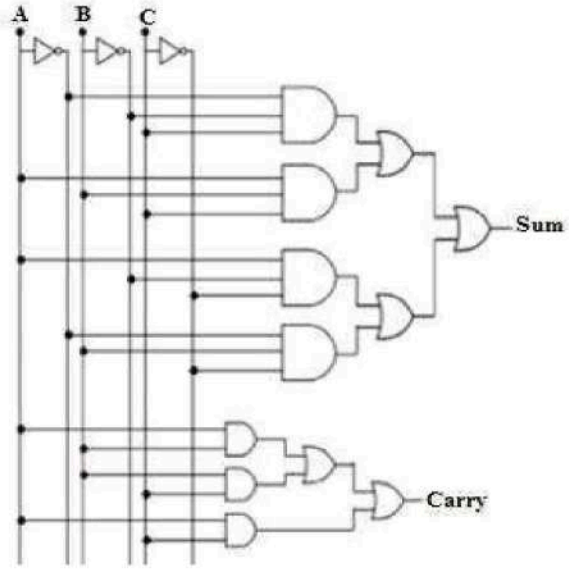


FullAdder

Full adder

Input			output	
A	B	C	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

Sum = $\overline{a}\,\overline{b}\,c + \overline{a}\,b\,\overline{c} + a\,\overline{b}\,\overline{c} + a\,b\,c$
Carry = $a \cdot b + a \cdot c + b \cdot c$

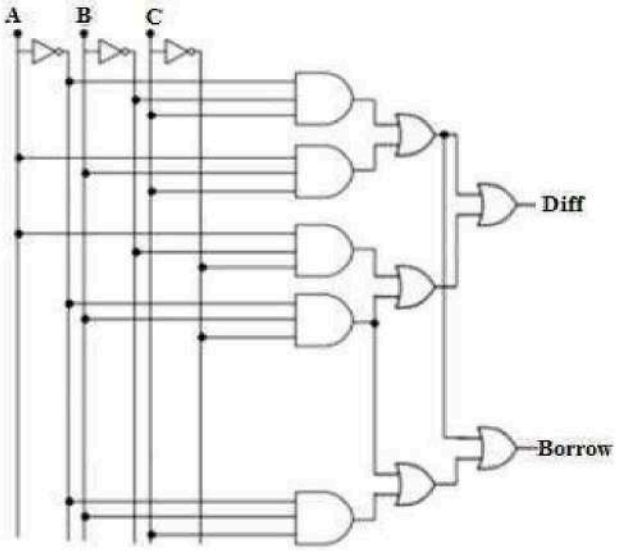


Full Subtractor

Full Subtractor

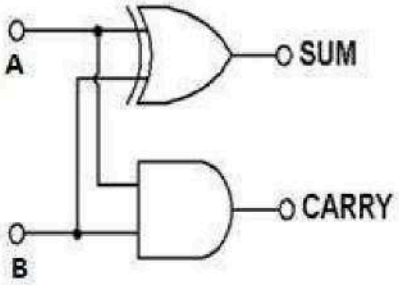
Input			output	
A	B	C	Difference	Borrow
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Difference = $\bar{a}\,\bar{b}\,c + \bar{a}\,b\,\bar{c} + a\,\bar{b}\,\bar{c} + a\,b\,c$
Borrow = $\bar{a}\,c + \bar{a}\,b + b\,c$



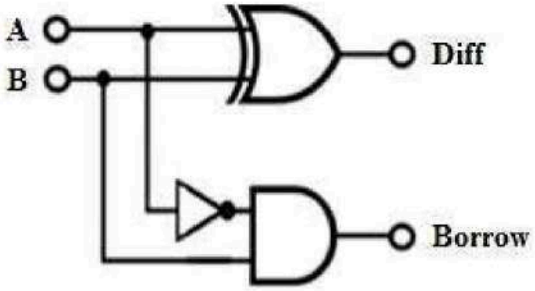
UsingEX-OR Gates

HalfAdder



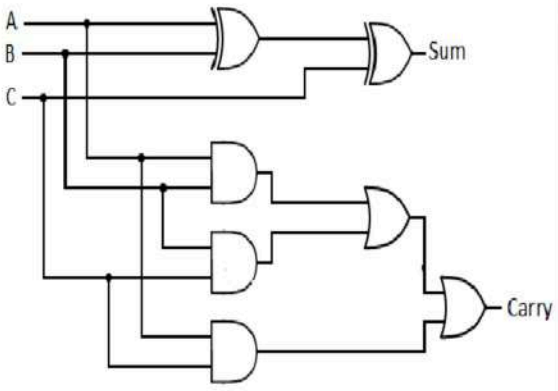
Sum=A⊕B
Carry=AB

HalfSubtractor



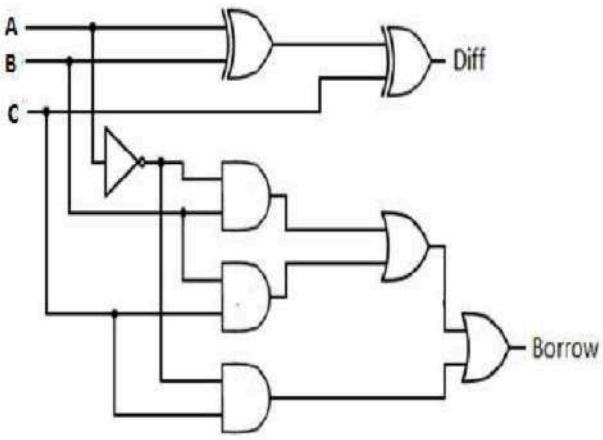
Diff=A⊕B
Borrow=A'B

Full Adder



Sum=A⊕B⊕C
Carry=AB+BC+AC

Full Subtractor



Diff=A⊕B⊕C
Borrow=A'B+BC+A'C

Verilogcode:

Half Adder-	HalfSubstractor-
Module half adder(a,b,sum,carry); Input a,b; Output sum,carry; Assign sum=a^b; Assign carry=a&b; endmodule	Module HS(a,b,diff,borrow); input a,b; Output diff,borrow; assign diff=a^b;assign borrow=!(a)&b; endmodule
Full Adder-	**Full Substractor-**
Module FA(a,b,c,sum,carry); Input a,b,c; Output sum,carry; Assigns um=a^b^c; Assign carry=(a&b)||(b&c)||(c&a); endmodule	Module FS(a,b,c,diff,borrow); Input a,b,c; Output diff,borrow; Assign diff=a^b^c; Assign borrow=((!a)&b)|(b&c)|((!a)&c); endmodule

VerilogTestbenchcode:

Half Adder-	Half Substractor-

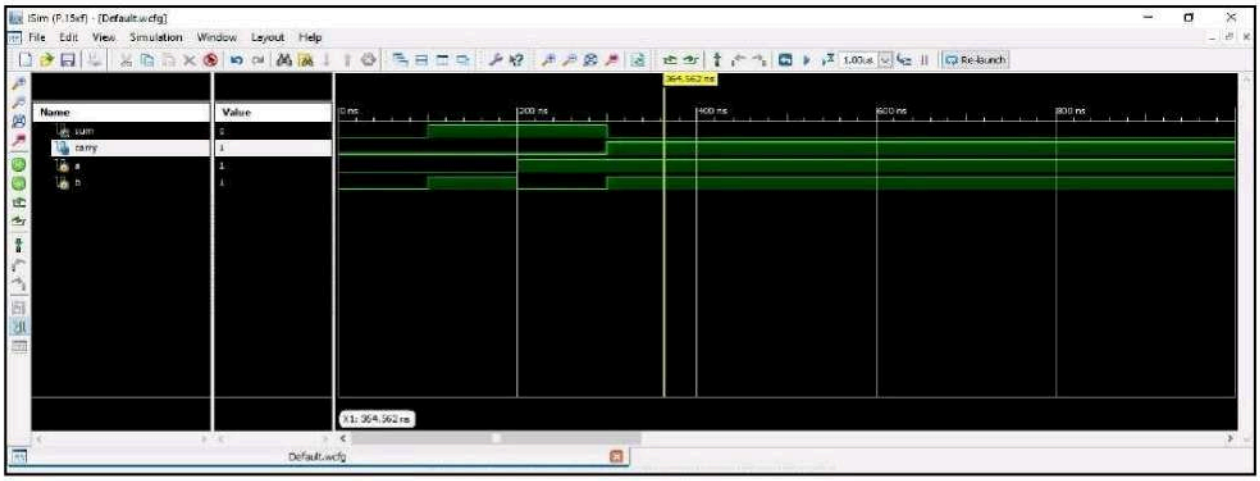
module halfadder_tb;
rega;
reg b;
wiresum;
wire carry;
halfadderuut(
.a(a),
.b(b), .sum(sum),
.carry(carry));
initial begin
a=0;b=0;#100;
a=0;b=1;#100;
a=1;b=0;#100;
a=1;b=1;#100;
end endmodule
	Module HS_tb;
	reg a;
	reg b;
	wire diff;
	wire borrow;
	HS uut (
	.a(a),
	.b(b),
	.diff(diff),
	.borrow(borrow));
	initial begin
	a=0;b=0; #100;
	a=0;b=1; #100;
	a=1;b=0; #100;
	a=1;b=1; #100;
	end endmodule

Full Adder-	**Full Substractor-**

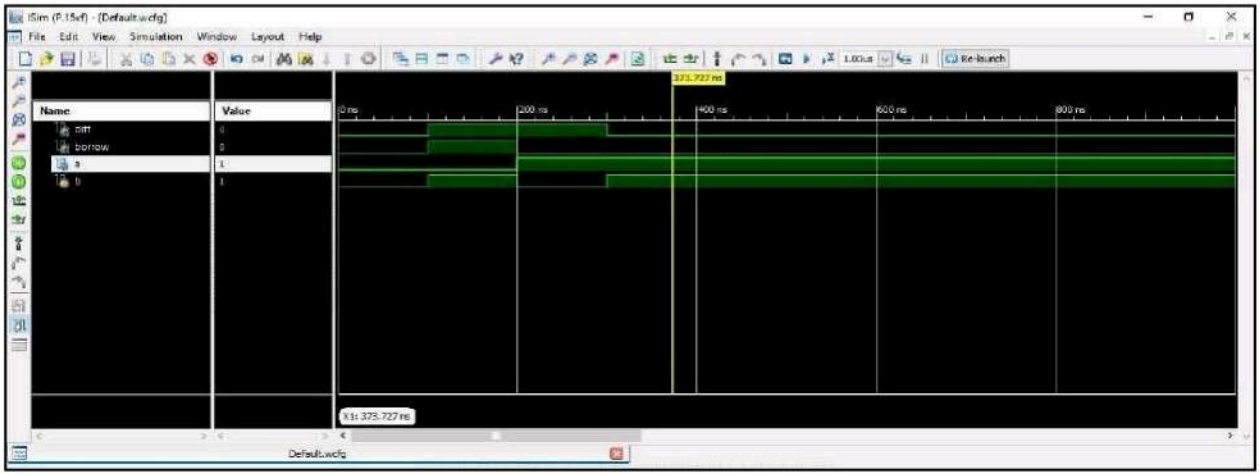
Module FA_tb;
Reg a;
Reg b;
Reg c;
Wire sum;
Wire carry;
FA uut(
.a(a),
.b(b),
.c(c),
.sum(sum),
.carry(carry));
Initial begin
a=0;b=0;c = 0;#100;
a=0;b=0;c = 1;#100;
a=0;b=1;c = 0;#100;
a=0;b=1;c = 1;#100;
a=1;b=0;c =0;#100;
a=1;b=0;c = 1;#100;
a=1;b= 1;c= 0;#100;
a=1;b=1;c = 1;#100;
end
endmodule
	Module FS_tb;
	Reg a;
	Reg b;
	Reg c;
	Wire diff;
	Wire borrow;
	FS uut(
	.a(a),
	.b(b),
	.c(c),
	.diff(diff),
	.borrow(borrow));
	Initial begin
	a=0;b=0;c = 0;#100;
	a=0;b=0;c = 1;#100;
	a=0;b=1;c = 0;#100;
	a=0;b=1;c = 1;#100;
	a=1;b=0;c = 0;#100;
	a=1;b=0;c = 1;#100;
	a=1;b=1;c = 0;#100;
	a=1;b=1;c = 1;#100;
	end
	endmodule

Simulation Output:

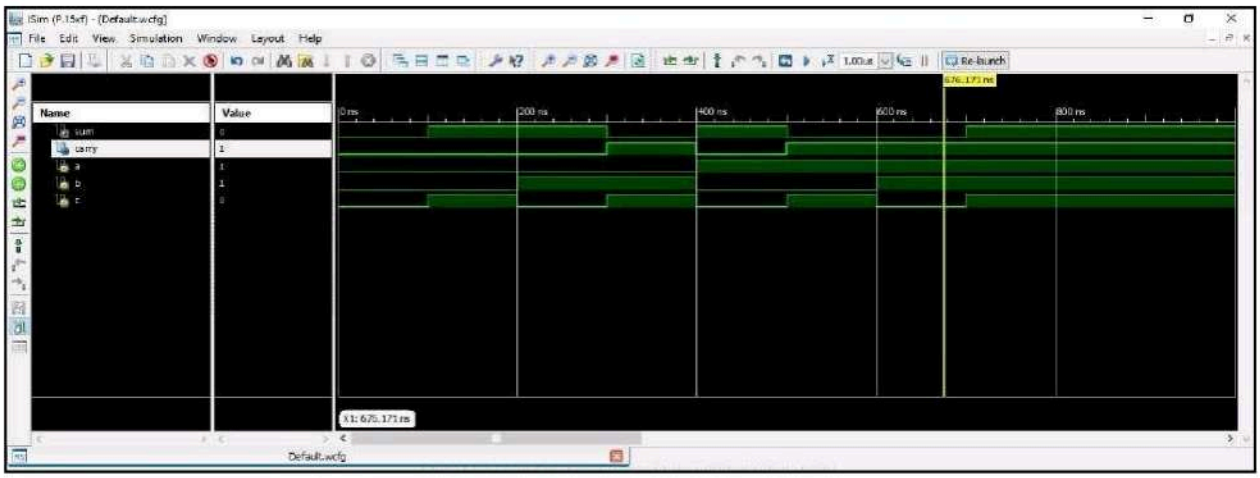
Half Adder-



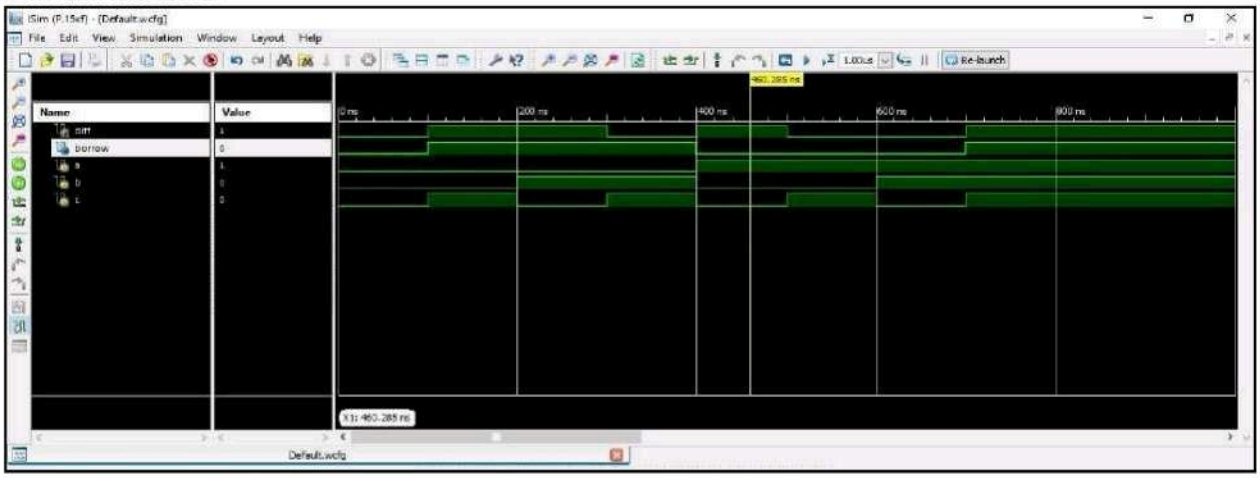
Half Substractor-



Full Adder-



Full Substractor



Results&conclusions: The Verilog code for adders and subtractors are verified.

EXPERIMENT NO–05

Design Verilog HDL to implement Decimal adder

Aim: To Design Verilog HDL to implement Decimal adder.

Theory: BCD or Binary code decimal is away of representing decimal digits in binary form. Generally 4 bits are used to represent values 0 to 9.

Decimal Digit	BCD 8421
0	0000
1	0001
2	0010
3	0011
4	0100

Decimal Digit	BCD 8421
5	0101
6	0110
7	0111
8	1000
9	1001

ABCD adder,takes in two BCD numbers as inputs,and out puts a BCD digit along with a carry out put. If the sum of the BCD digits is less than or equal to 9,then we don't have a problem. But if its greater than9,wehave to convert it into BCD format.This is done by adding 6tothesumandtakingonlythe least significant 4 bits. The MSB bit is output as carry.

Consider the below BCD addition:

1001 +1000=10001

9+8=17

The output 17 ismorethan9.So weadd6to it.So we get,

17+6 = 23(inbinary23is 10111)

Now the least significant 4bits(which is"0111") represent the unitsdigit and the MSB(4thbit which is '1') bit represents the tens digit. Since the range of input is 0 to 9,the maximum output is 18. If you consider a carry it becomes 19. This means at the output side we need a 4 bit sum and a 1 bit carry to represent the most significant digit. These binary numbers are listed below

The condition for a correction and an output carry can be expressed by the Boolean Function

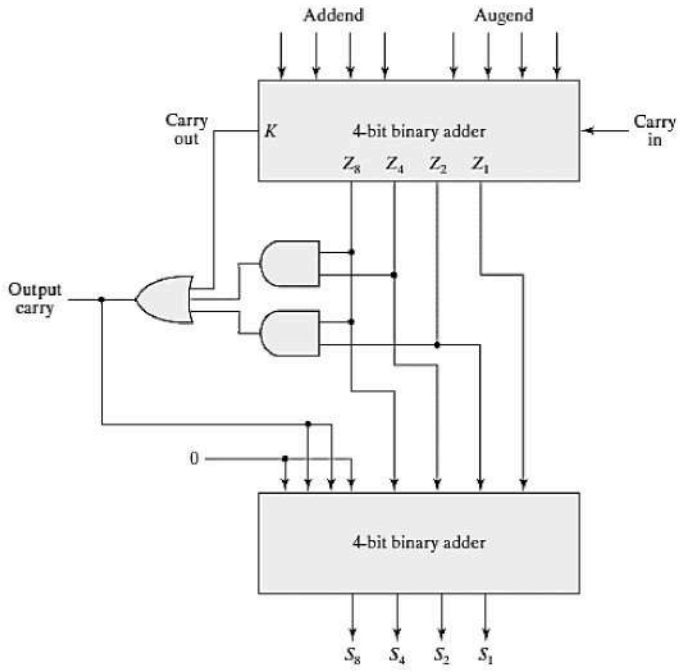
$$C = K + Z_8Z_4 + Z_8Z_2$$

When C = 1, it is necessary to add 0110 to the binary sum and provide an output carry for the next stage. A BCD adder that adds two BCD digits and produces a sum digit in BCD is shown in Figure below.

Derivation of BCD Adder

Binary Sum					BCD Sum					Decimal
K	Z_8	Z_4	Z_2	Z_1	C	S_8	S_4	S_2	S_1	
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	2
0	0	0	1	1	0	0	0	1	1	3
0	0	1	0	0	0	0	1	0	0	4
0	0	1	0	1	0	0	1	0	1	5
0	0	1	1	0	0	0	1	1	0	6
0	0	1	1	1	0	0	1	1	1	7
0	1	0	0	0	0	1	0	0	0	8
0	1	0	0	1	0	1	0	0	1	9
0	1	0	1	0	1	0	0	0	0	10
0	1	0	1	1	1	0	0	0	1	11
0	1	1	0	0	1	0	0	1	0	12
0	1	1	0	1	1	0	0	1	1	13
0	1	1	1	0	1	0	1	0	0	14
0	1	1	1	1	1	0	1	0	1	15
1	0	0	0	0	1	0	1	1	0	16
1	0	0	0	1	1	0	1	1	1	17
1	0	0	1	0	1	1	0	0	0	18
1	0	0	1	1	1	1	0	0	1	19

Logic Diagram:



For multiple digit addition ,you can connect the carry_out to the carry input of the next adder.A simple cascading network of these small adders is enough to realize the multiple digit BCD addition.

Procedure:

10. Open a Xilinx IDE,close old projects.
11. Create a new project through the project navigator icon.

12. AddVHDL/Verilognewsourcetotheprojectdependingontheuserrequirement.
13. Select the Verilog HDL mode.
14. Assign the inputs and outputs for the system to design.
15. And write the code as given below for the functionality.
16. Synthesize the code and correct the syntax errors if any.
17. Is im Simulator and Behavioral check syntax the Simulation behavioral model.
18. Observe the out puttiing wave form and verify it with the truth table.

Verilog code for BCD addition-Behavioral level:

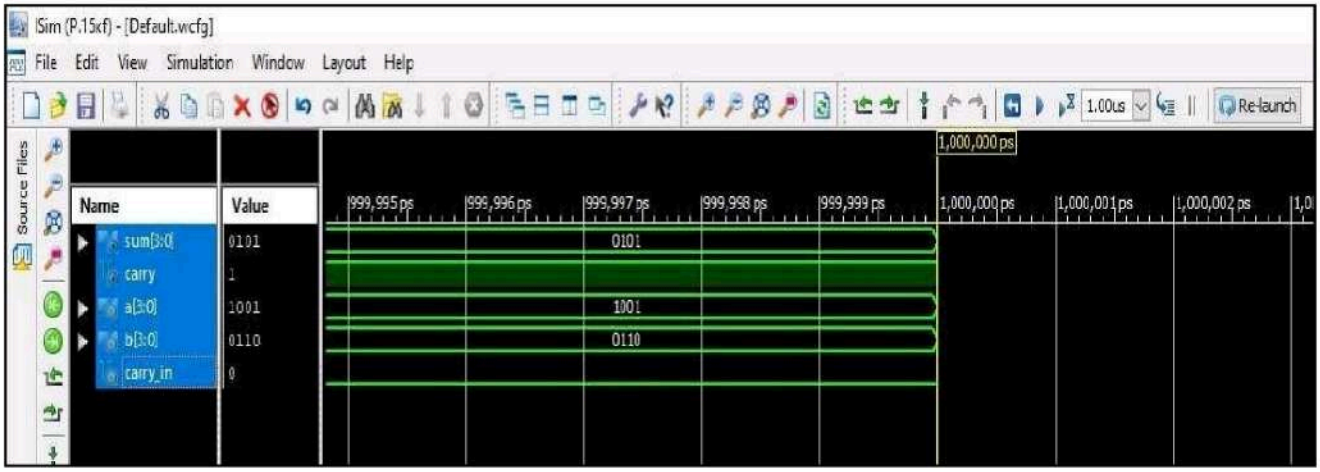
```
Module deciadder(a,b,carry_in,sum,carry);
                                    //declaretheinputsandoutputsofthemodulewiththeirsizes.
  input [3:0] a,b;
  input carry_in;
  output[3:0]sum;
  output carry;
                                          //Internalvariables
  reg[4:0]sum_temp;
  reg[3:0]sum;
  reg carry;
                                        //alwaysblockfordoingtheaddition
  always@(a,b,carry_in)
  begin

       sum_temp=a+b+carry_in;                          //addalltheinputs
   if(sum_temp> 9)
      begin
      sum_temp = sum_temp+6;                           //add6,ifresultismorethan9.
      carry = 1;//set the carry output
      sum=sum_temp[3:0];
      end
    else
      begin
      carry=0;
      sum=sum_temp[3:0];
     end
  end
endmodule
```

Verilog Test bench code for BCD 9adder:

```
Module deciadder_tb;
      reg[3:0]a;
      reg[3:0]b;
      reg carry_in;
      wire[3:0]sum;
```

```
        wirecarry;
deciadder uut(.a(a),.b(b),.carry_in(carry_in), .sum(sum),.carry(carry));
        initialbegin
                //InitializeInputs
                a=9;
                b=6;
                carry_in=0;
                #100;
        end
endmodule
```

Simulation wave form:



Results&conclusions: The Verilog code for Decimal adder is simulated and verified.

EXPERIMENTNO–06

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

Aim: Design Verilog program to implement Different typesofmultiplexerslike2:1,4:1and8:1.

Objectives:

1. DevelopVerilogprogramfor2:1,4:1,8:1multiplexer design.
2. Validatethroughsimulationforaccuracyandefficiencyincircuitimplementation.

Theory:

- Amultiplexer (data selector,abbreviatedasMUX)has a group of data inputs(2^n)anda group of control inputs (n) (also called as select inputs).
- The control inputs are used to select one of the data inputs and connect it to the output terminal.

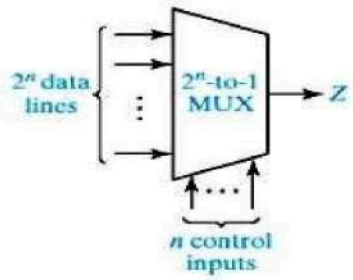


Figure 6.1:Generalblockdiagramof2^n:1multiplexer

- A general block diagramof2^n:1multiplexerisshowninthe fig.3.6.
- A2to1multiplexerrequires 1selectinput,4to1multiplexersrequire2selectinputsand8to 1multiplexersrequire3selectinputs.

2:1 Multiplexer

- A2to1multiplexerhas2datainputs, 1selectinputand1output.

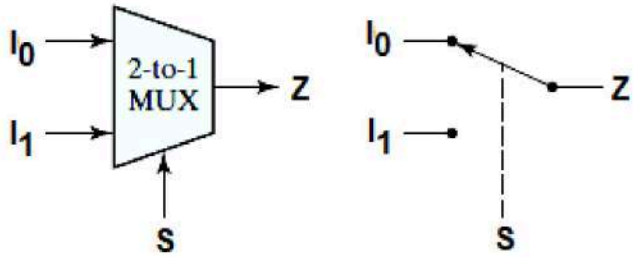


Figure6.2:2to1MultiplexerandSwitch Analog

- When the select (control) input A is 0,the switch is in the upper position and the MUX output is $Z = I_0$.
- When the select(control)inputAis1, thes witch is in the lower position and the MUX output Is $Z= I_1$.
- In other words ,a MUX acts like a switch that selects one of the data inputs(I_0 or I_1)and transmits it to the output.

Truth Table:

Select(S)	tput(Z)
0	I_0
1	I_1

- Thelogicequationforthe2-to-1MUXcanbewrittenas:

$$Z=\bar{S}I_0+SI_1$$

Logic Diagram:

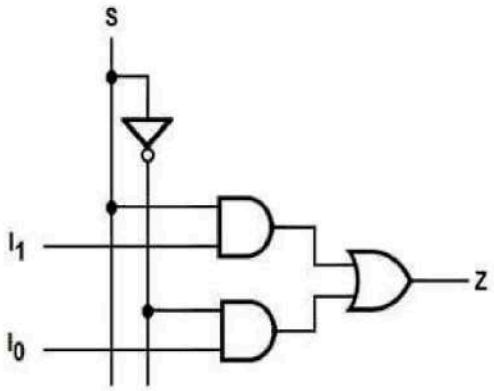


Figure 6.3:Logic diagram of2:1 Multiplexer

4:1Multiplexer

- A4to 1multiplexerhas4datainputs,2 selectinputsand1output.
- The 4 to 1 MUX acts like a four-position switch that transmits one of the four inputs to the output.

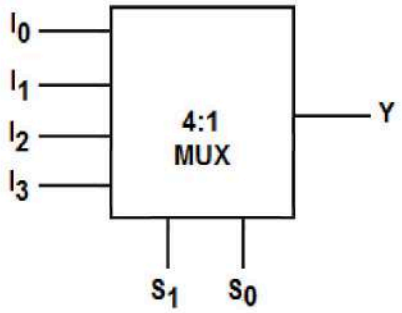


Figure6.4:4to1 Multiplexer

- Two select(control) inputs(S_1 and S_0)are needed to select one of the four inputs.If the control inputs are S_1S_0=00,theoutputisI_0;similarly, forthecontrolinputs01,10,and11giveoutputs of I_1, I_2, and I_3, respectively.

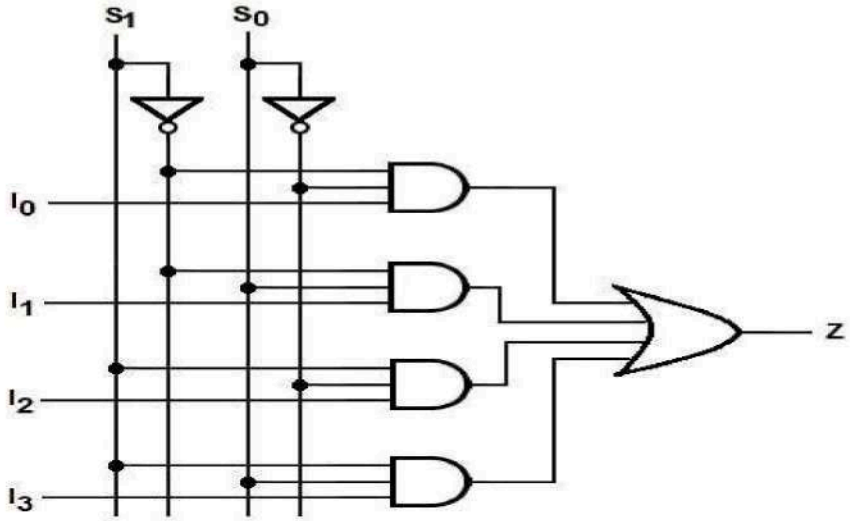
Truth Table:

elect(S_1)	Select(S_0)	Output(Z)
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

- Thelogicequationfor4-to-1multiplexer can be written as

$$Z=S_1S_0I_0+S_1S_0I_1+S_1S_0I_2+S_1S_0I_3$$

Logic Diagram:



8:1Multiplexer

- An8to1multiplexerhas8datainputs,3select inputsand1output.8to1MUXselectsoneof eight data inputs using three select inputs

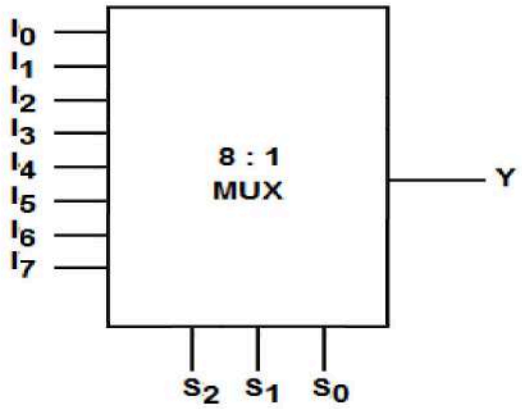


Figure6.5:8to1 Multiplexer

- The8-to-1MUXactslikeaneight-positionswitchthattransmitsoneoftheeight inputs to the output.

Truth Table:

elect(S_2)	Select(S_1)	Select(S_0)	Output(Z)
0	0	0	I_0
0	0	1	I_1
0	1	0	I_2
0	1	1	I_3
1	0	0	I_4
1	0	1	I_5
1	1	0	I_6
1	1	1	I_7

- The logic equation forthe8to1 MUX can be written as:

$Z=\bar{S}_2\bar{S}_1\bar{S}_0I_0+\bar{S}_2\bar{S}_1S_0I_1+\bar{S}_2S_1\bar{S}_0I_2+\bar{S}_2S_1S_0I_3+S_2\bar{S}_1\bar{S}_0I_4+S_2\bar{S}_1S_0I_5+S_2S_1\bar{S}_0I_6 + S_2S_1S_0I_7$

Verilog code:

2:1MUX	4:1MUX	8:1MUX

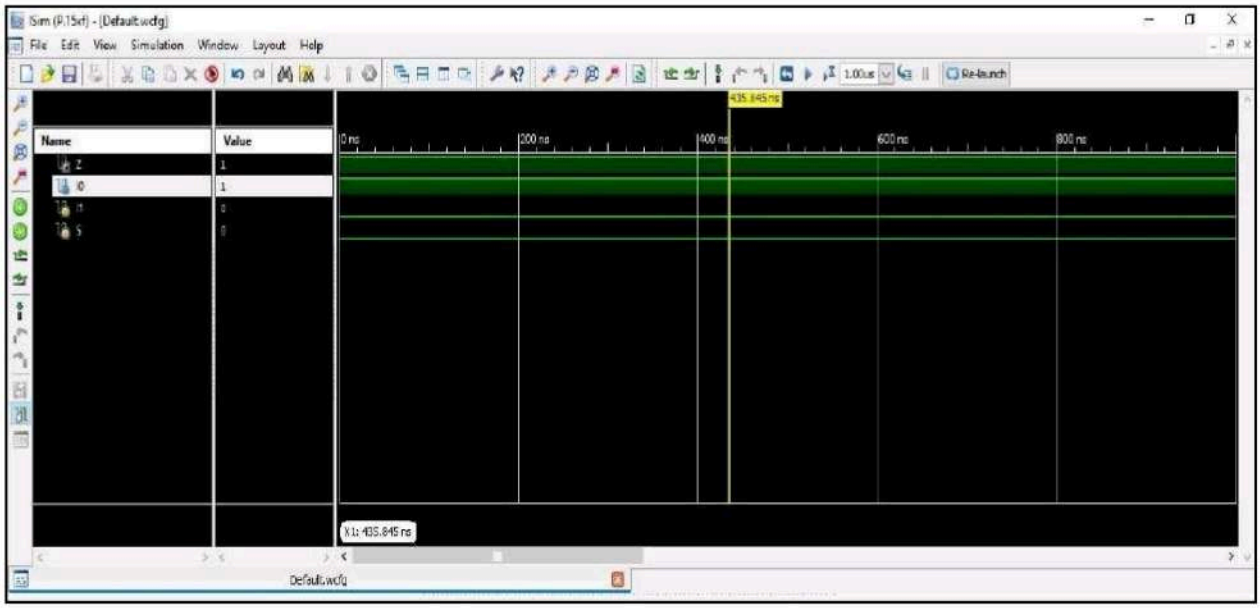
Module two_to_one_MUX (I0,I1,S,Y);
input I0;
input I1;
input S;
output Y;
wireS_bar,w1,w2;
not g1(S_bar,S);
and(w1,I0,S_bar);
and (w2,I1,S);
or(Y,w1,w2);
endmodule
	Module four_to_one_MUX (I1,I2,I3,I4,S1,S2,Y);
	input I1,I2,I3,I4;
	inputS1,S2;
	output Y;
	
	assign Y=((!S1)&(!S2)&I1)|((!S1)&(S2)&I2)|(S1&(!S2)&I3)|(S1&S2&I4);
	endmodule
		module eight_to_one_MUX (I1,I2,I3,I4,I5,I6,I7,I8,S1,S2,S3,Y);
		input I1,I2,I3,I4,I5,I6,I7,I8;
		inputS1,S2,S3;
		output Y;
		
		assignY=((!S1)&(!S2)&(!S3)&I1)|((!S1)&(!S2)&S3&I2)|((!S1)&S2&(!S3)&I3)|((!S1)&S2&S3&I4)|(S1&(!S2)&(!S3)&I5)|(S1&(!S2)&S3&I6)|(S1&S2&(!S3)&I7)|(S1&S2&S3&I8);
		endmodule

Verilog Test bench code:

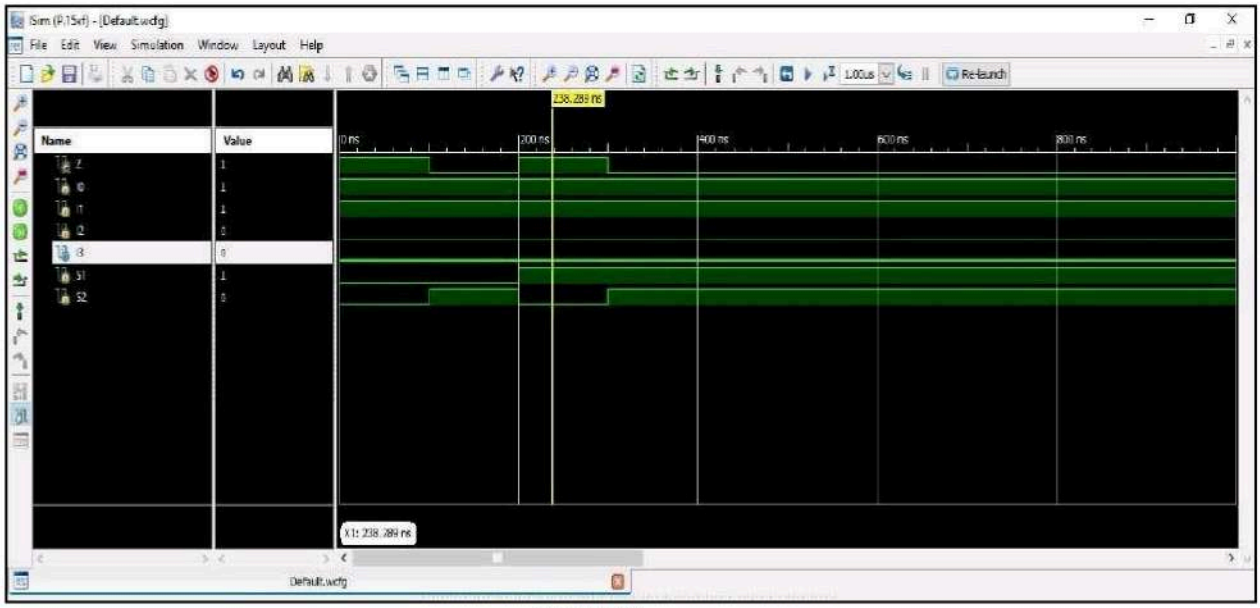
2:1MUX	4:1MUX	8:1MUX
moduletwo_to_one_MUX_tb; reg I0; reg I1; reg S; wireY; two_to_one_MUXuut(.I0(I0), .I1(I1), .S(S), .Y(Y)); Initial begin I0 = 1; I1 = 0; S =0; #100; end endmodule	modulefour_to_one_MUX_tb; reg I1; reg I2; reg I3; reg I4; regS1; regS2; wire Y; four_to_one_MUX uut(.I1(I1), .I2(I2), .I3(I3), .I4(I4), .S1(S1), .S2(S2), .Y(Y)); Initial begin I1=0;I2=0;I3=0;I4=1; S1 = 0;S2 = 1; #100; end endmodule	moduleeight_to_one_MUX1; reg I1; regI2; regI3; regI4; regI5; regI6; regI7; reg I8; reg S1; reg S2; reg S3; wire Y; eight_to_one_MUX uut(.I1(I1),.I2(I2),.I3(I3),.I4(I4), .I5(I5),.I6(I6),.I7(I7),.I8(I8), .S1(S1),.S2(S2),.S3(S3), .Y(Y)); Initial begin I1=1;I2=1;I3=1;I4=1;I5= 1;I6 = 0;I7 = 1;I8 = 1; S1=1;S2=0;S3=1; #100; end endmodule

Simulation Output:

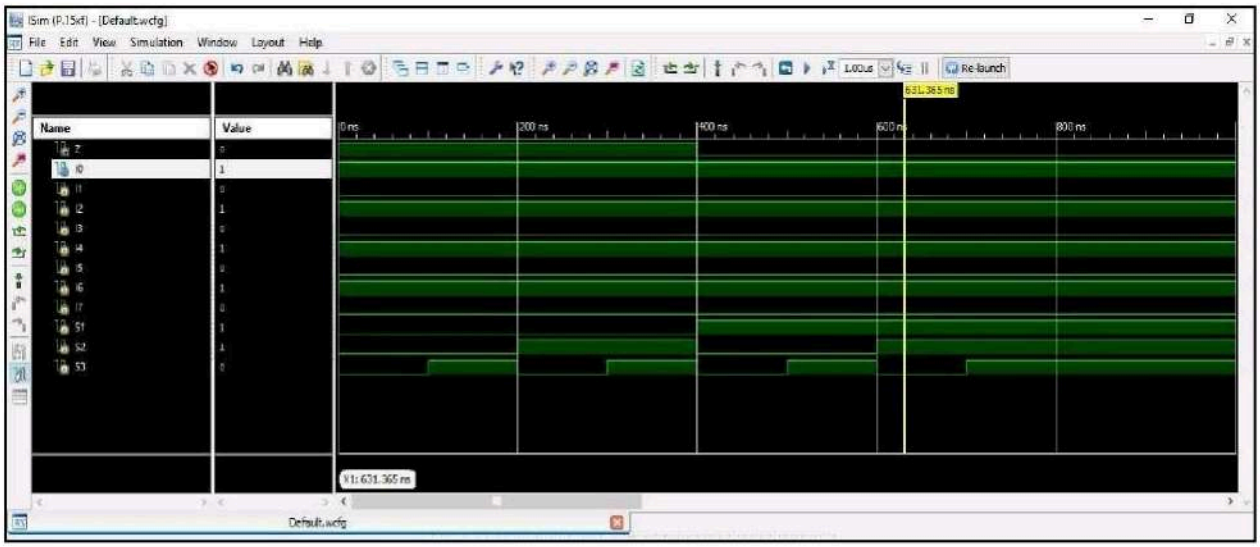
2:1 MUX



4:1-MUX



8:1 MUX



Result & Conclusion:-

The Verilog code for Different types of multiplexer s like 2:1,4:1 and 8:1 is simulated and verified.

EXPERIMENT NO–07

Design Verilog program to implement types of De-Multiplexer.

To Design Verilog program to implement Different types of De-multiplexer like1:2,1:4 and 1:8.

Objectives:

1. ImplementVerilogprogramfor1:2,1:4,1:8Demultiplexer designs.
2. Verify functionality through simulation for accuracy and performance assessment.

Theory:

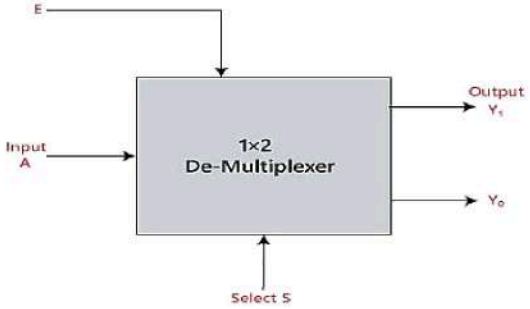
A De-multiplexer is a combinational circuit that has only1 input line and 2^Noutput lines. Simply, the multiplexer is a single-input and multi-output combinational circuit. The information is received from the single input lines and directed to the output line. On the basis of the values of the selection lines, the input will be connected to one of these outputs. De-multiplexer is opposite to the multiplexer.

Unlike encoder and decoder,there are n selection online sand2^noutputs.So,there is a total of 2^npossible combinations of inputs. De-multiplexer is also treated as **De-mux**.

There are various types of De-multiplexer which are as follows:

1×2 De-multiplexer: In the 1 to 2 De-multiplexer, there are only two outputs, i.e., Y_0, and Y_1, 1 selection onlines,i.e.,S_0,and single input,i.e.,A .Based on the selection value,the input will be connected to one of the outputs. The block diagram and the truth table of the 1×2 multiplexer are given below.

Block Diagram:



Truth Table:

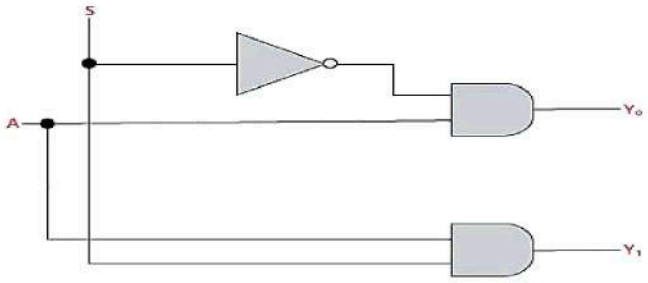
INPUTS	Output	
S_0	Y_1	Y_0
0	0	A
1	A	0

The logical expression of the term Y is as follows:

$$Y_0=S_0'.A$$

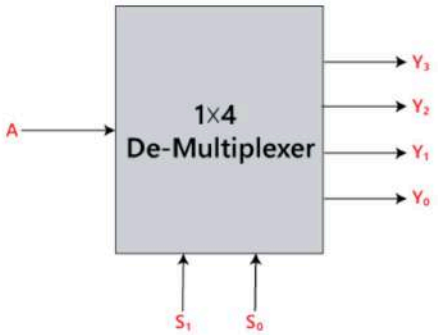
$$Y_1=S_0.A$$

Logical circuit of the above expressions is given below:



1×4 De-multiplexer: In 1 to 4 De-multiplexer, there are total of four outputs, i.e., Y_0, Y_1, Y_2, and Y_3,2selection lines, i.e.,$S_0$and $S_1$and single input,i.e.,A. Based on the combination of input swhich are present at the selection lines S_0 and S_1, the input be connected to one of the outputs. The block diagram and the truth table of the 1×4 multiplexer is given below.

Block Diagram:



Truth Table:

INPUTS		Output			
S_1	S_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	A
0	1	0	0	A	0
1	0	0	A	0	0
1	1	A	0	0	0

The logical expression of the term Y is as follows:

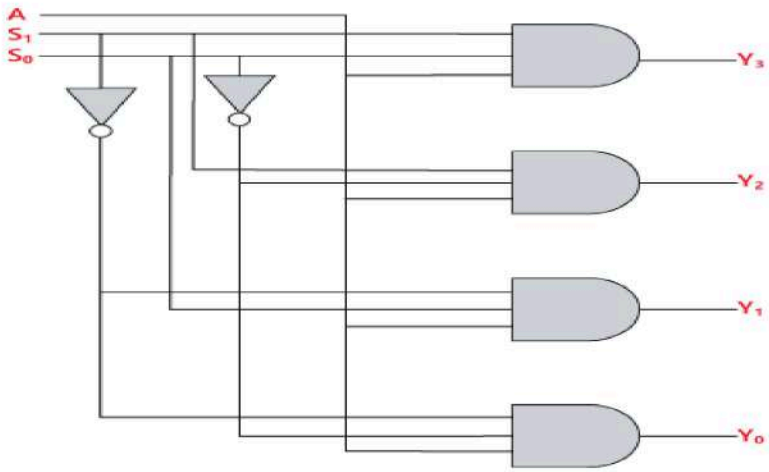
$Y_0=S_1'S_0'A$

$y_1=S_1' S_0A$

$y_2=S_1S_0' A$

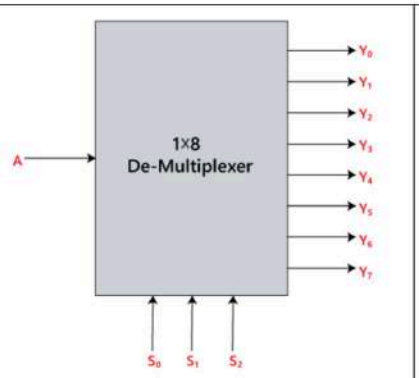
$y_3=S_1 S_0 A$

Logical circuit of the above expressions is given below:



1×8 De-multiplexer: In 1 to 8 De-multiplexer, there are total of eight outputs, i.e., Y_0, Y_1, Y_2, Y_3, Y_4, Y_5, Y_6, and Y_7, 3 selection lines, i.e., S_0, $S_1$and $S_2$and single input, i.e., A. On the basis of the combination of inputs which are present at the selection lines S^0, S^1 and S_2, the input will be connected to one of these outputs. The block diagram and the truth table of the 1×8 de-multiplexer is given below.

Block Diagram:	**Truth Table:**



INPUTS			Output							
S_2	S_1	S_0	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0	0	0	A
0	0	1	0	0	0	0	0	0	A	0
0	1	0	0	0	0	0	0	A	0	0
0	1	1	0	0	0	0	A	0	0	0
1	0	0	0	0	0	A	0	0	0	0
1	0	1	0	0	A	0	0	0	0	0
1	1	0	0	A	0	0	0	0	0	0
1	1	1	A	0	0	0	0	0	0	0

The logical expression of the term Y is as follows:

$Y_0=S_0'.S_1'.S_2'.A$

$Y_1=S_0.S_1'.S_2'.A$

$Y_2=S_0'.S_1.S_2'.A$

$Y_3=S_0.S_1.S_2'.A$

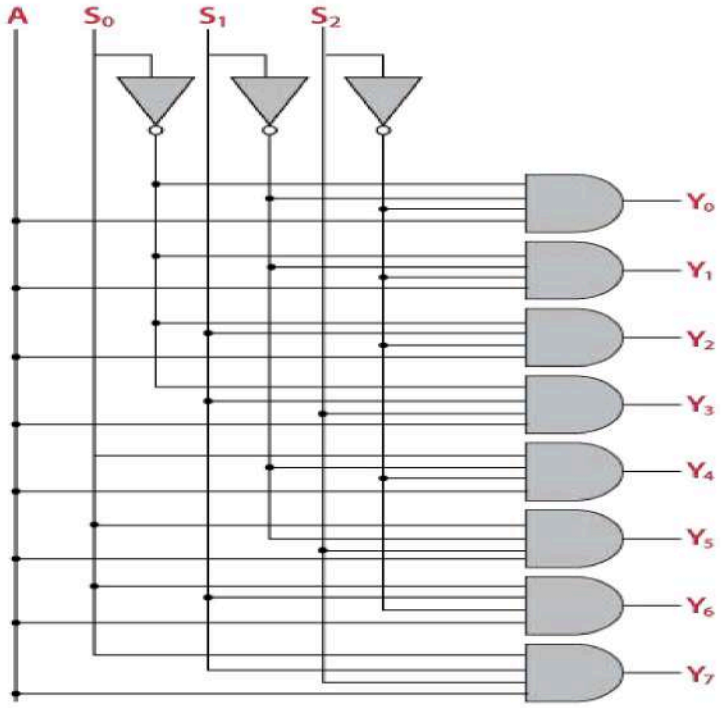
$Y_4=S_0'.S_1'.S_2\ A$

$Y_5=S_0.S_1'.S_2\ A$

$Y_6=S_0'.S_1.S_2\ A$

$Y_7=S_0.S_1.S_3.A$

Logical circuit of the above expressions is given below:



Verilog code:

1:2DEMUX	1:4DEMUX	1:8DEMUX

moduledemux12_BM(S0,A,Y);
inputA,S0;
wireA,S0;
output[1:0]Y;
reg [1:0]Y;
always@(AorS0)
begin
case(S0)
1'b0:Y={A,1'b0};
default:Y={1'b0,A};
endcase
end
endmodule
	module
	one_four_demux_BM(S,A,Y);
	input A;
	wire A;
	input[1:0]S;
	wire[1:0]S;
	output[3:0]Y;
	reg[3:0]Y;
	
	always@(AorS)begin
	
	case(S)
	1'b00:Y={A,3'b000};
	1'b01:Y={1'b0,A,2'b00};
	1'b10:Y={2'b00,A,1'b0};
	default:Y={3'b000,A};
	endcase
	end
	endmodule
		module
		one_eight_demux_BM(S,A,Y);
		input A;
		wire A;
		input[2:0]S;
		wire[2:0]S;
		output[7:0]Y;
		reg[7:0]Y;
		
		always@(AorS)
		begin
		
		case(S)
		0:Y={A,7'b0000000};
		1:Y={1'b0,A,6'b000000};
		2:Y={2'b00,A,5'b00000};
		3:Y={3'b000,A,4'b0000};
		4:Y={4'b0000,A,3'b000};
		5:Y={5'b00000,A,2'b00};
		6:Y={6'b000000,A,1'b0};
		default:Y={7'b0000000,A};
		endcase
		end
		endmodule

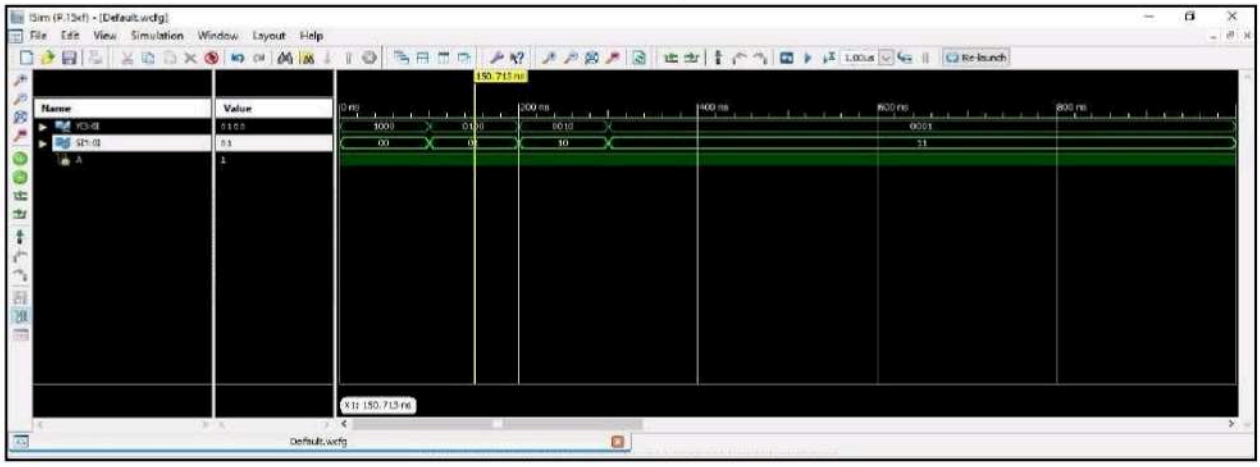
Verilog Test bench code:

1:2DEMUX	1:4DEMUX	1:8DEMUX
moduledemux12_BM_tb; reg S0; reg A; wire [1:0] Y; demux12_BMuut (.S0(S0), .A(A), .Y(Y)); Initial begin S0=0;A=1; #100; S0=1;A=1; #100; end endmodule	module one_four_demux_BMtb; reg[1:0]S; reg A; wire [3:0] Y; one_four_demux_BMuut(.S(S), .A(A), .Y(Y)); Initial begin S=0;A=1; #100; S=1;A=1; #100; S=2;A=1; #100; S=3;A=1; #100; end endmodule	module one_eight_demux_BMtb; reg[2:0]S; reg A; wire [7:0] Y; one_eight_demux_BMuut(.S(S), .A(A), .Y(Y)); Initial begin S=0;A=1; #100; S=1;A=1; #100; S=2;A=1; #100; S=3;A=1; #100; S=4;A=1; #100; S=5;A=1; #100; S=6;A=1; #100; S=7;A=1; #100; end endmodule

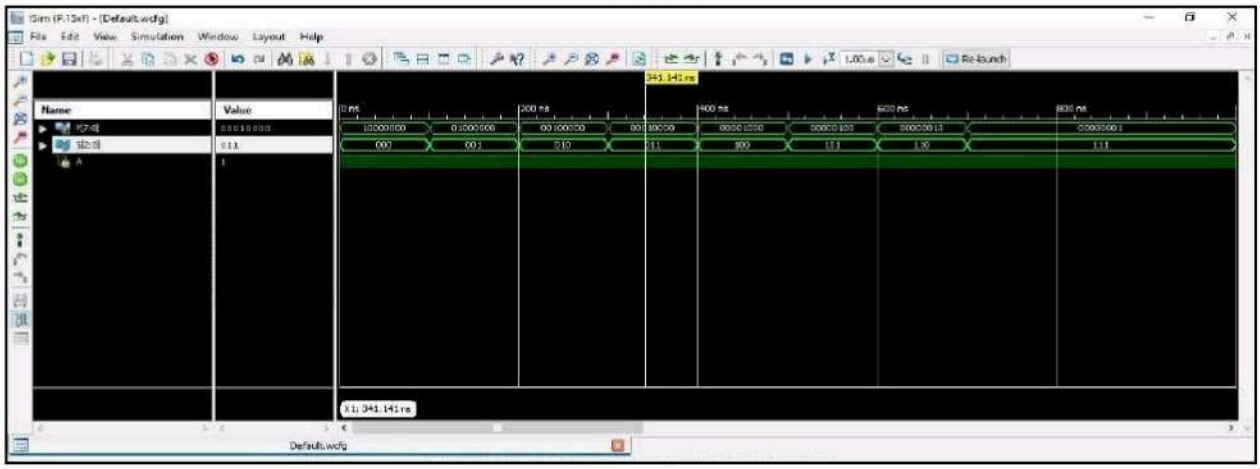
Simulation Output:

1:2 DEMUX

1:4 DEMUX



1:8 DEMUX



Results & conclusions:

The Verilog code for Different types of De-multiplexer like1:2, 1:4 and 1:8 is simulated and verified.

EXPERIMENTNO–08

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

Aim: To design and simulate Verilog modules for three different types of flip-flops(SR,JK,and D) using behavioral modeling. The program should demonstrate the functionality and behavior of these flip-flops under various input conditions.

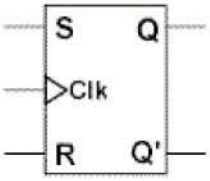
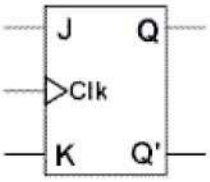
Objectives:

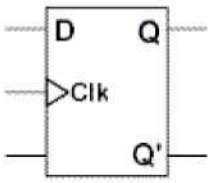
- Implement Verilog modules for SR,JK,and D flip-flops using behavioral modeling.Develop a testbench to simulate the behavior of the implemented flip-flops.
- Applydifferentinputconditionstotheflip-flopsinthetestbenchtoobservetheir responses.
- Verify that the flip-flops exhibit the epected sequential logic behavior,such as state changes and toggling.

Theory:

Flip-flops are synchronous bitable devices. The term synchronous means the output changes state only when the clock input is triggered. That is, changes in the output occur in synchronization with the clock. A flip-flop circuit has two outputs, one for the normal value and one for the complement value of the stored bit.Since memory element sinsequential circuits are usually flip-flops,it is worth summarizing the behavior of various flip-flop types before proceeding further. Flip-flops can be divided into four basic types:SR,JK,D and T.They differint the number of inputs and in there sponse invoked by different value of input signals. The four types of flip-flops are defined in the Table 8.1. Eachoftheseflip-flopscanbeuniquelydescribedbyitsgraphicalsymbol, its character is tictable, its characteristic equation or excitation table. All flip-flops have output signals Q and Q'.

Table8.1:Flip-flopsandtheirproperties

Flip-Flop Name	Flip-Flop Symbol	Characteristic Table	Characteristic Equation	Excitation Table
SR		S R Q(next) 0 0 Q 0 1 0 1 0 1 1 1 ?	Q(next)= S + R'Q SR = 0	Q Q(next) S R 0 0 0 X 0 1 1 0 1 0 0 1 1 1 X 0
JK		J K Q(next) 0 0 Q 0 1 0 1 0 1 1 1 Q'	Q(next)= JQ' +K'Q	Q Q(next) J K 0 0 0 X 0 1 1 X 1 0 X 1 1 1 X 0

D		D | Q(next) 0 | 0 1 | 1	Q(next)= D	Q | Q(next) | D 0 | 0 | 0 0 | 1 | 1 1 | 0 | 0 1 | 1 | 1		

Logic diagram

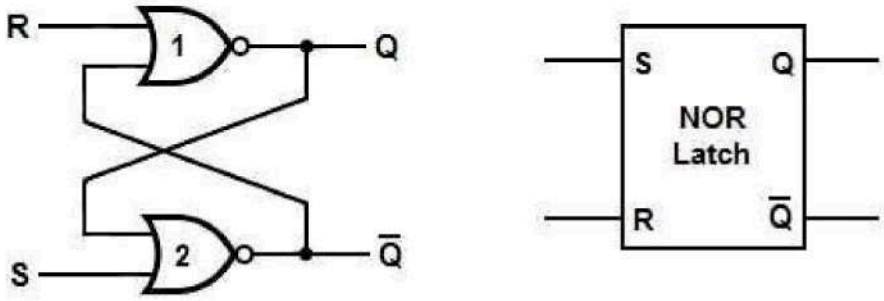


Figure8.1:SR(a) NOR Gate latch(b) Symbol

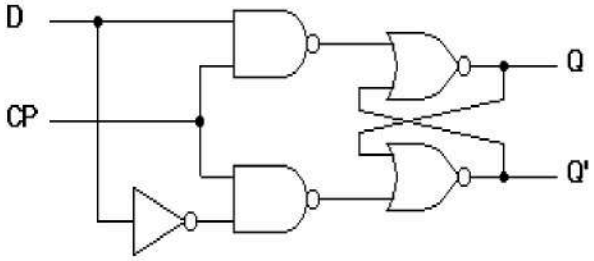


Figure8.2:D-FlipFlop

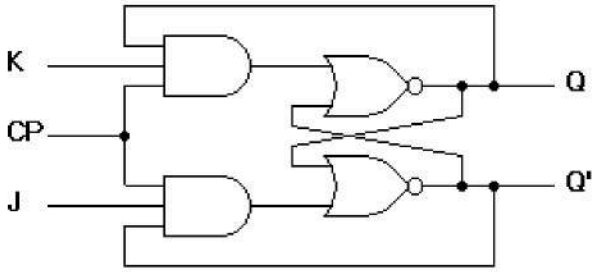


Figure8.3:JKFlipFlop

Graphs

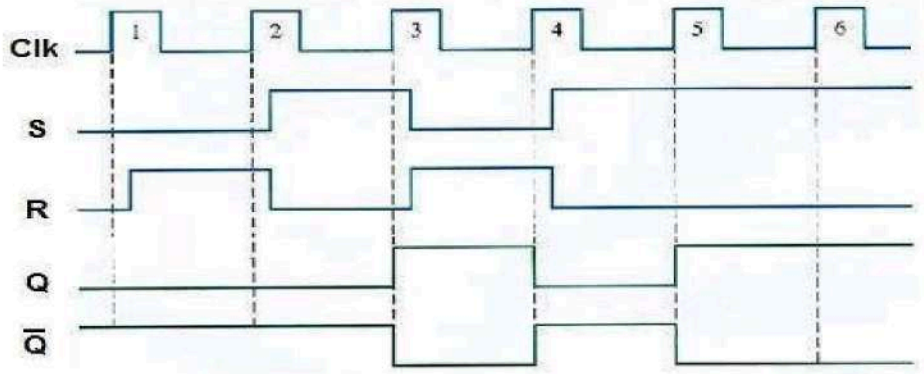


Figure8.4:Timing diagram of SR Flip-Flop

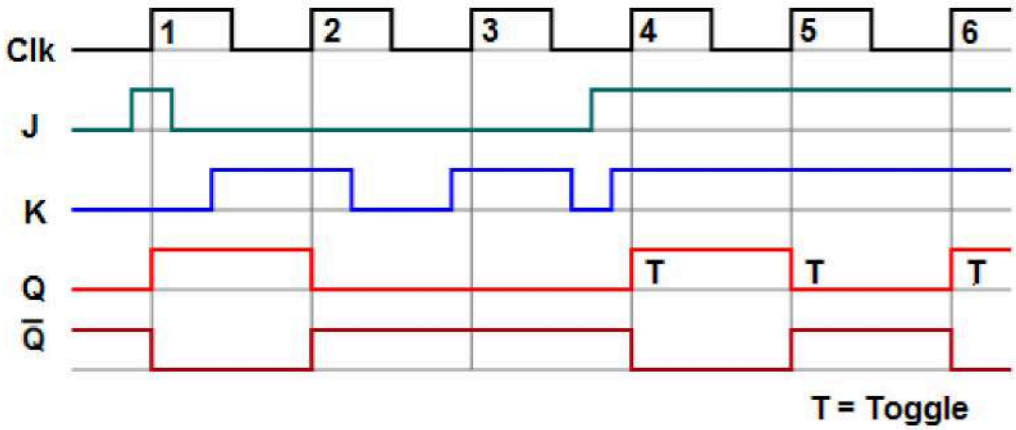


Figure8.5: Timing diagram of JK Flip-Flop

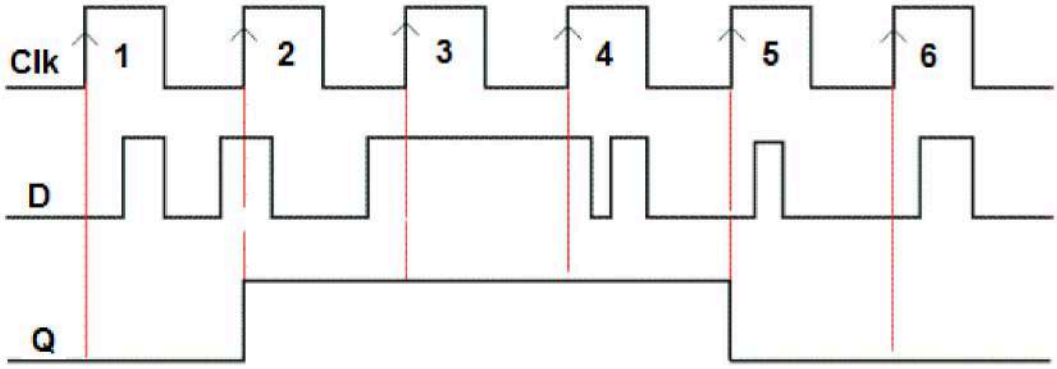


Figure8.6: Timing diagram of D Flip-Flop

Verilog code:

1.SR Flip-Flop	2.JK FlipFlop	3.D FlipFlop

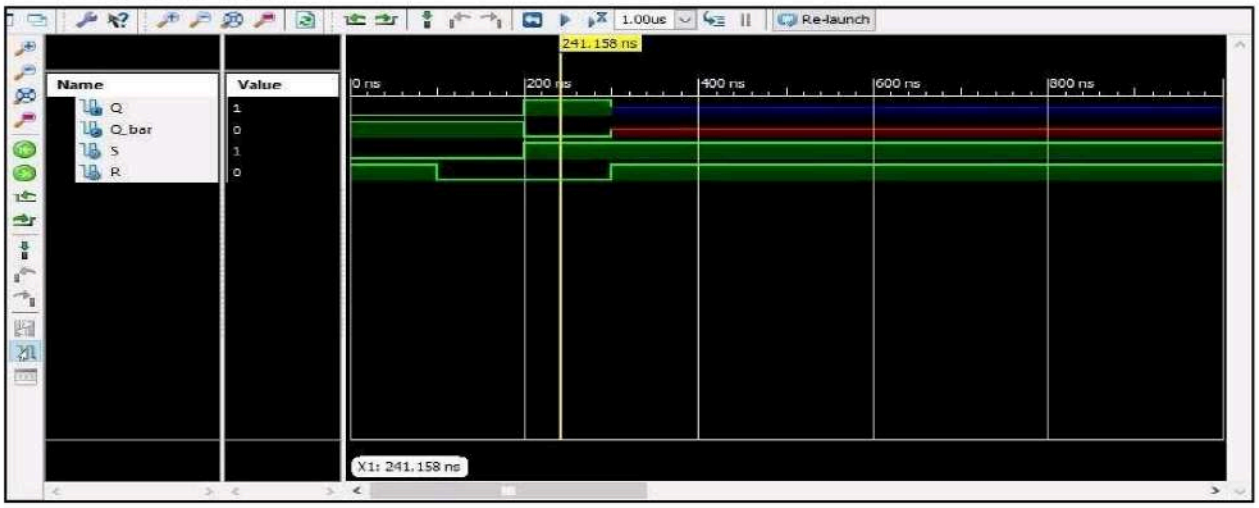
module srlax(
input wireS,
input wireR,
input output reg Q,
output wireQ_bar);
always@(S,R)begin if
(S && ~R)
Q <=1'b1;
elseif(~S&& R)
Q <=1'b0;
elseif(S==0&&R==0)
Q <= Q;
elseif(S==1&&R==1)
Q <=1'bz;
End
assignQ_bar=~Q;
endmodule
	moduleJK_FF(
	input wire j,
	Input wirek,
	Output regQ,
	Output wireQ_bar);
	
	always@(j,k)begin
	if(j&&~k)
	Q <=1'b1;
	elseif(~j&&k)
	Q <=1'b0;
	elseif(j==0&& k==0)
	Q <=Q;
	elseif(j==1 && k==1)
	Q <=!Q;
	End
	assignQ_bar=~Q;
	endmodule
		moduleD_FF(
		Input wireD,
		Output regQ,
		Output wireQ_bar
		);
		
		always@(D)begin
		if(D==0)
		Q <=0;
		else
		Q <=1;
		end
		
		Assign Q_bar=~Q;
		endmodule

Verilog Test bench code:

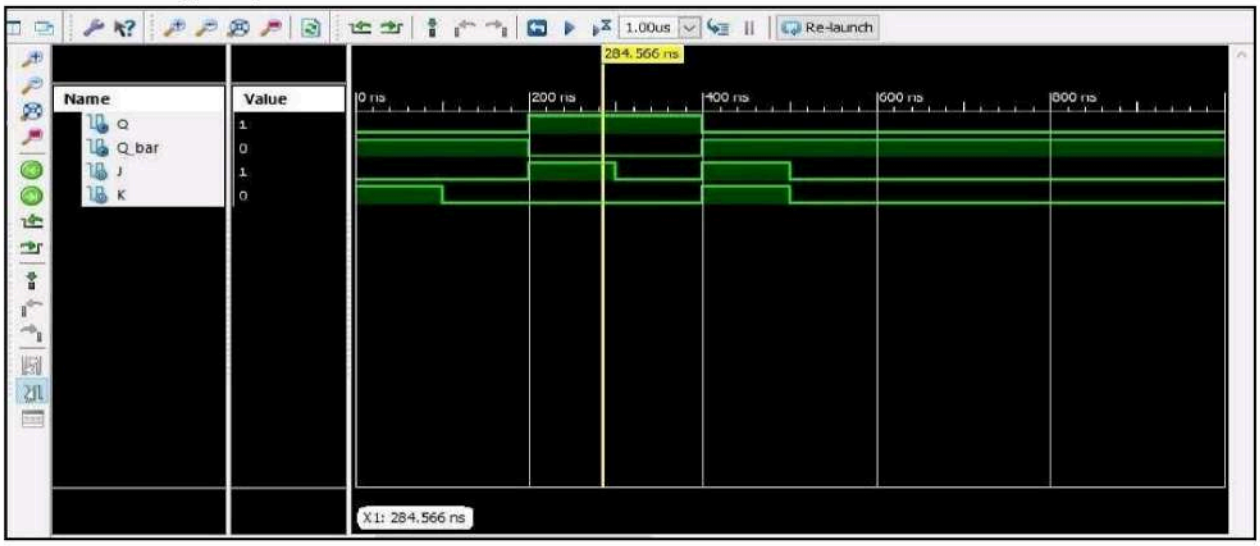
1.SRFlip-Flop:	2.JK FlipFlop	3.DFlipFlop
Module srlax_tb; reg S; regR; wire Q; wireQ_bar; srlax uut (.S(S),.R(R), .Q(Q),.Q_bar(Q_bar)); initial begin S=0;R =0; #100; S=1;R =0; #100; S=0;R =1; #100; S=1;R =1; #100; end endmodule	Module JK_FF_tb; reg J; reg K; wireQ; wireQ_bar; JK_FFuut (.J(J),.K(K), .Q(Q),.Q_bar(Q_bar)); initial begin J =0;K=0;#100; J=1;K=0; #100; J =0;K=1;#100; J=1;K=1; #100; end endmodule	Module D_FF_tb; reg D; wireQ; wireQ_bar; D_FF uut (.D(D), .Q(Q),.Q_bar(Q_bar)); initial begin D=0; #100; D=1; #100; end endmodule

Simulation Output:

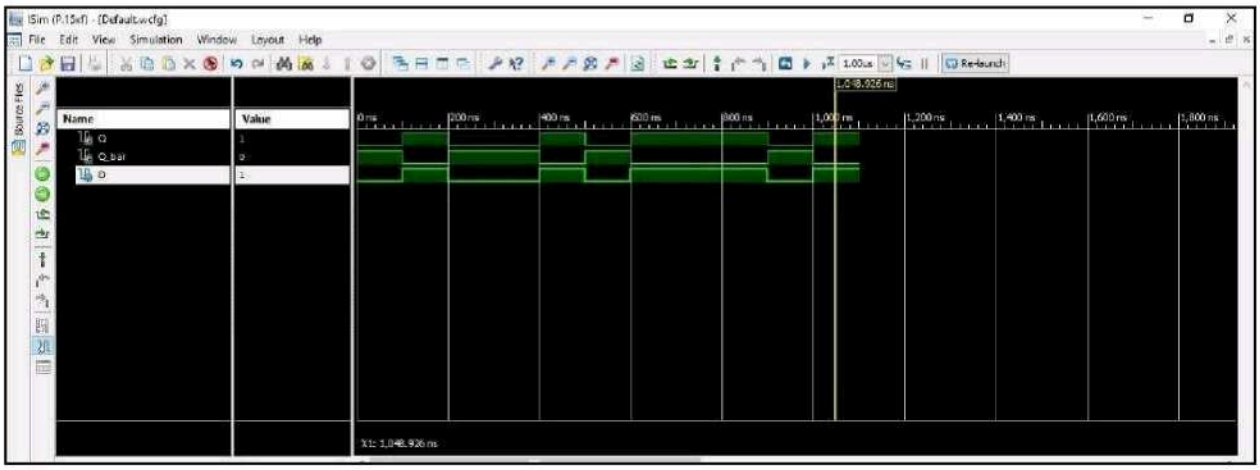
1. SR Flip-Flop



2. JKFlip Flop



3. D Flip Flop:-



Results & conclusions:

The Verilog code for Different types various types of Flip-Flops such as SR, JK and D.is simulated and verified