VISVESVARAYA TECHNOLOGICAL UNIVERSITY,
Jnana Sangama, Belagavi – 590018

KLS VISHWNATHRAO DESHPANDE INSTITUTE OF TECHNOLOGY,
Haliyal – 581 329, Uttara Kannada

LABORATORY MANUAL

Course Title	:	**Digital Design and Computer Organization**
Course Code	:	**BCS302**
Year / Semester	:	**2nd Year / 3rd Semester**
Academic Year	:	**Odd Semester of 2025 - 2026**
Course In-Charge	:	**Prof. Shree Gowri S. S.**

Department of Computer Science & Engineering

KLS Vishwanathrao Deshpande Institute of Technology

(Accredited by NAAC with "A" Grade)

(Approved by AICTE, New Delhi. Affiliated to VTU, Belagavi)

(Recognized Under Section 2(f) by UGC, New Delhi)

Udyog Vidya Nagar, Haliyal – 581329, Dist.: Uttara Kannada

Phone: 08284-220861, 220334, 221409, Fax: 08284-220813

Department of Computer Science and Engineering

College Vision and Mission Statements

Vision
To nurture talent & enrich society through excellence in technical education, research & innovation.
Mission
1. To augment innovative Pedagogy & kindle quest for interdisciplinary learning & to enhance conceptual understanding. **2. To build competence, professional ethics & develop entrepreneurial thinking.** **3. To strengthen Industry Institute Partnership & explore global collaborations.** **4. To inculcate culture of socially responsible citizenship.** **5. To focus on Holistic & Sustainable development.**

Department Vision and Mission Statements

Vision
To achieve excellence in technical education, research, innovation in Computer Science and Engineering by emphasizing on global trending technologies.
Mission
1. To train students with conceptual understanding through innovative pedagogies. 2. To imbibe professional, research and entrepreneurial Skills with commitment to the nation development at large. 3. To strengthen the industry institute Interaction. 4. To promote life–long learning with a sense of societal & ethical responsibilities

PEOs	
PEO 1:	To develop an ability to identify and analyze the requirements of Computer Science and Engineering in design and providing novel engineering solutions.
PEO 2:	To develop abilities to work in team on multidisciplinary projects with effective communication skills, ethical qualities and leadership roles.
PEO 3:	To develop abilities for successful Computer Science Engineer and achieve higher career goals.
PSOs	
PSO1	To develop the ability to model real-world problems using appropriate data structure and suitable algorithms in the area of Data Processing, System Engineering, and Networking for varying complexity.
PSO2	To develop an ability to use modern computer languages, environments and platforms in creating innovative career.

KLS Vishwanathrao Deshpande Institute of Technology

(Accredited by NAAC with "A" Grade)

(Approved by AICTE, New Delhi. Affiliated to VTU, Belagavi)

(Recognized Under Section 2(f) by UGC, New Delhi)

Udyog Vidya Nagar, Haliyal – 581329, Dist.: Uttara Kannada

Phone: 08284-220861, 220334, 221409, Fax: 08284-220813

Department of Computer Science and Engineering

Course Title	:	Digital Design and Computer Organization
Course Code	:	BCS302
Year / Semester	:	2nd Year / 3rd Semester
Academic Year	:	Odd Semester of 2025 – 2026

CLOs	:	1. To Understand the functionalities of binary logic system 2. To Understand Combinational and Sequential logic circuits
COs	:	**CO1:** To Apply K-map techniques to simplify various Boolean expression **CO2:** Design different types of combinational and sequential circuits with Verilog programs
CO / PO Mapping	:	**COs** | **CO1** | **CO2** **POs / PSOs** | **PO1** | **PO1, PO2, PO3**
Course In-Charge	:	Prof. Shree Gowri S. S.

CO-PO Mapping Matrix

CO	*PO1*	*PO2*	*PO3*	*PO4*	*PO5*	*PO6*	*PO7*	*PO8*	*PO9*	*PO10*	*PO11*	*PO12*	*PSO1*	*PSO2*
1	2													
2			1		1									

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

AIM: To simplify the given logic expression using suitable technique

Software Tool Used: Multisim

Objective: To understand to Simplifiy the logic expression and implement using basic gates

Theory: Logic Expression can be reduced using K-Map(Karnaugh Map). Types of K-Map are: 1. 2-variable K-Map
2. 3-Variable K-Map
3. 4-Variable K-Map

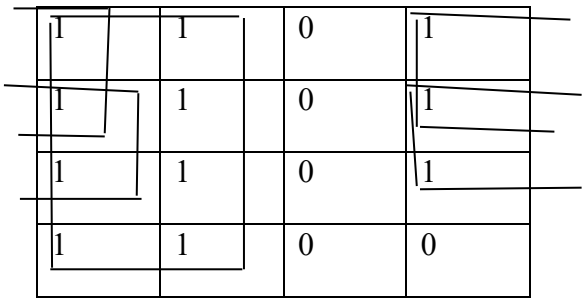
The given expression can be simplified by grouping 1's in K-Map by using Pair, Quad and Octet. Pair eliminates one variable and its complement. Quad eliminates two variable and its complement and Octet eliminated 3 variable and its complement.

Assume that the following 4-variable Boolean function is to be implemented
Y = F(A,B,C,D) = ∑ (0,1,2,4,5,6,8,9,12,13,14).

Truth Table

A	b	C	D	Y(output)
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

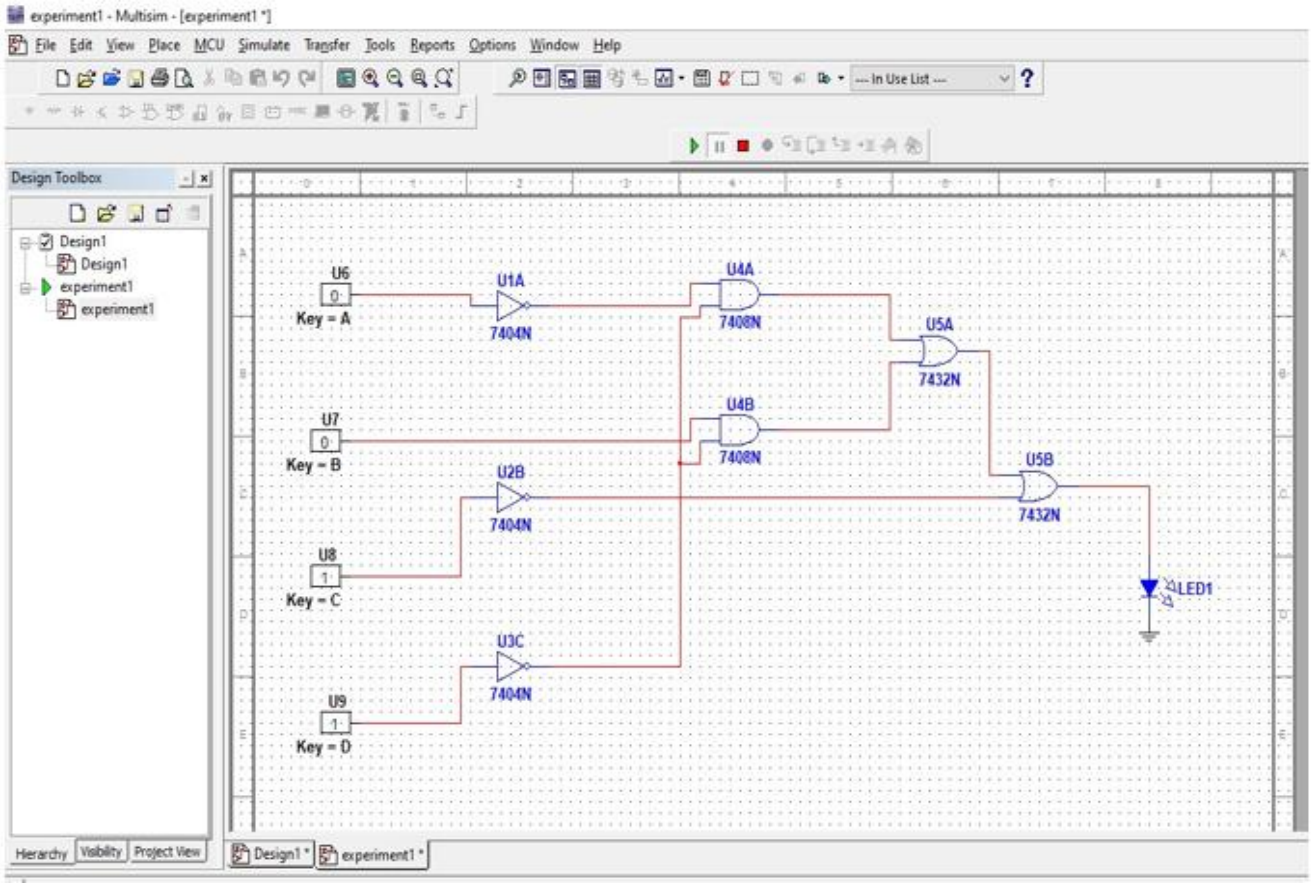
K-Map for Binary code expression

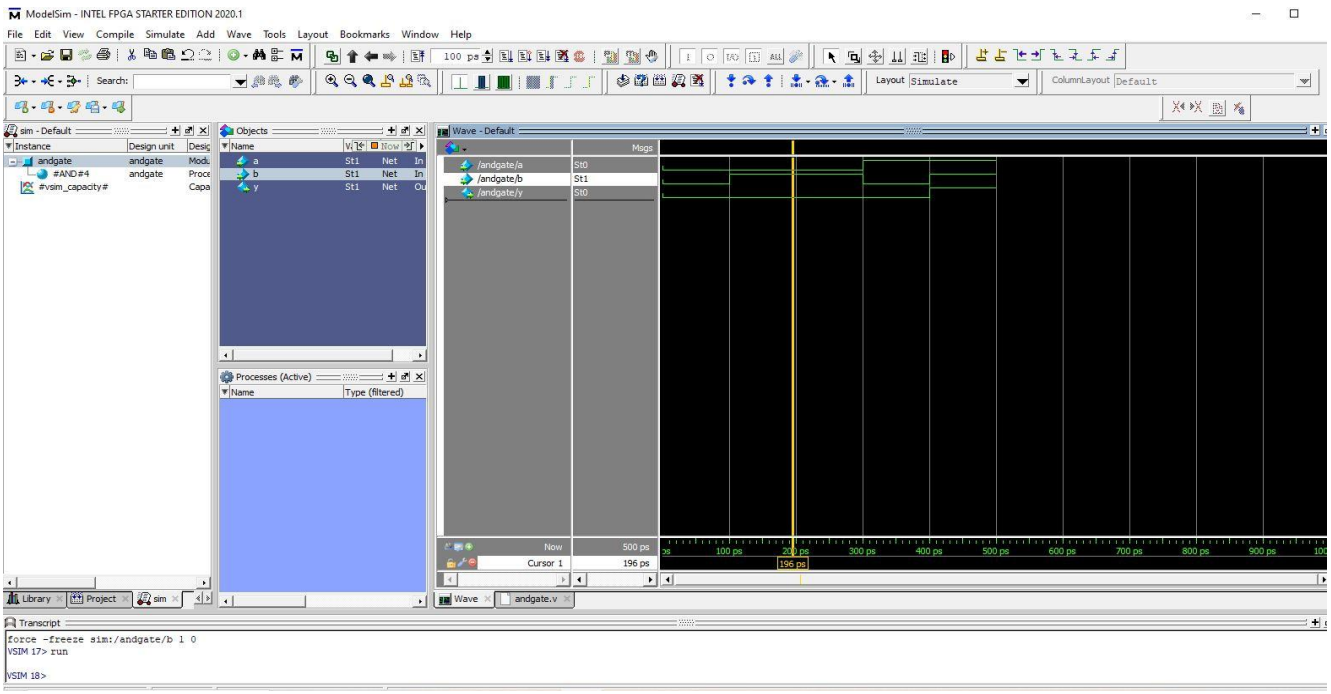


Logic Expression

Y=c'+a'd'+bd'

Implementation of Logic Circuit for logic expression Y=c'+a'd'+bd'





Conclusion: The Given 4 variable logic expression is simplified using K-Map technique and simulated the same using basic gates.

2. Design A 4 Bit Full Adder And Subtractor And Simulate The Same Using Basic Gates.

AIM: To design and construct full adder and full subtractor circuits and verify the truth table using logic gates.

Software Tool Used: Multisim

Objective: To Study and understand Adders and Subtractor

THEORY:

Adder circuit 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 ad Full adder.

• Half-Adder: A combinational 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.

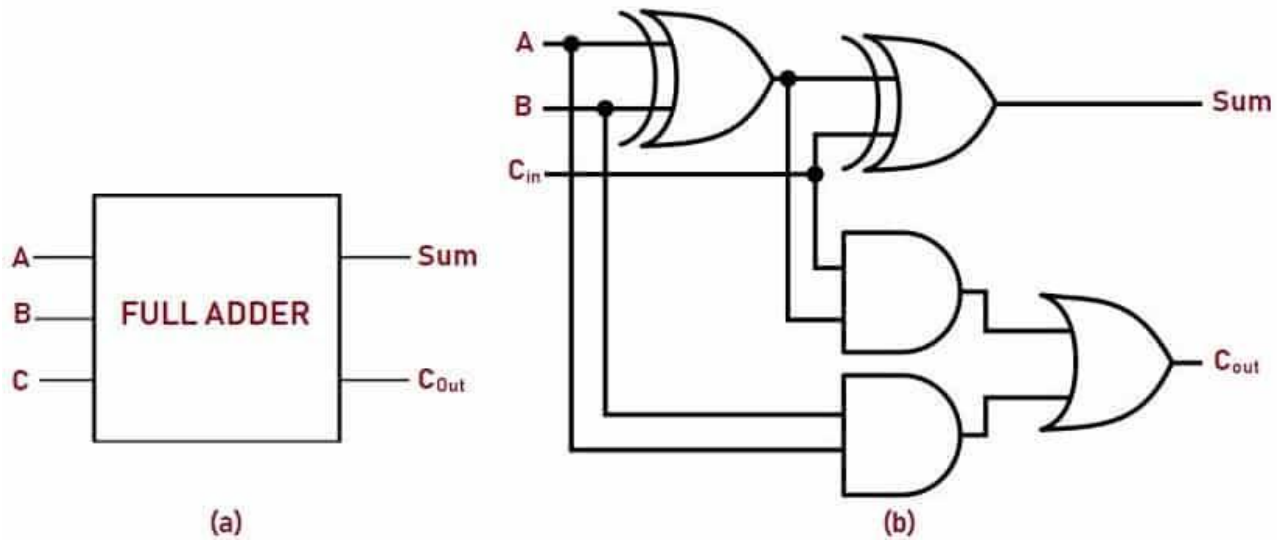
• Full-Adder: The half-adder does not take the carry bit from its previous stage into account. This carry bit from its previous stage is called carry-in bit. A combinational logic circuit that adds two data bits, A and B, and a carry-in bit, Cin , is called a full-adder.

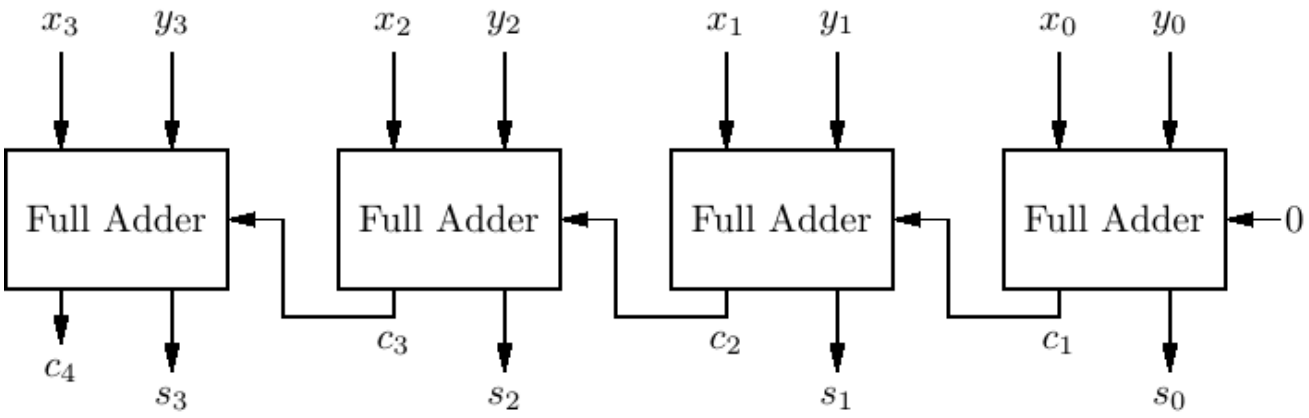
• Subtractor: Subtractor is the one which used to subtract two binary number(digit) and provides Difference and Borrow as a output.In digital electronics we have two types of subtractor. Half Subtractor and Full Subtractor.

• Half Subtractor :Half Subtractor is used for subtracting one single bit binary digit from another single bit binary digit.

• Full Subtractor : A logic Circuit Which is used for Subtracting Three Single bit Binary digit is known as Full Subtractor.

Block Diagram of 1-bit full adder and fuull subtractor





$$s = x + y$$
$$CF = c_4$$
$$OF = c_3 \oplus c_4$$

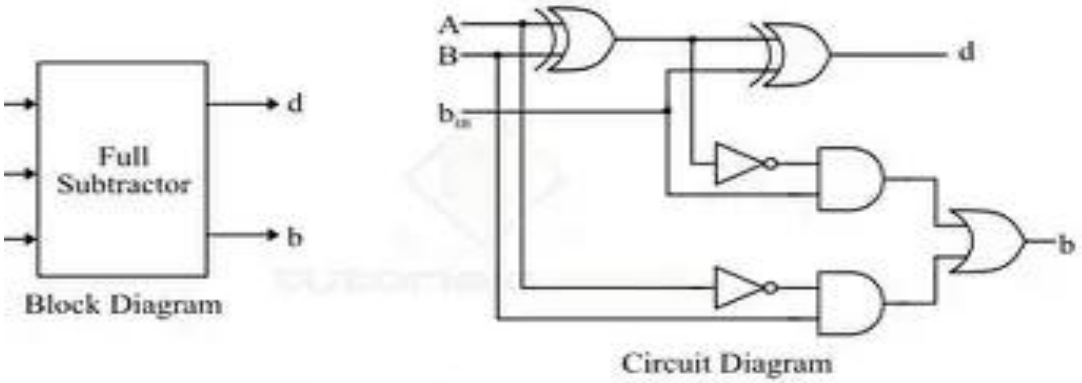
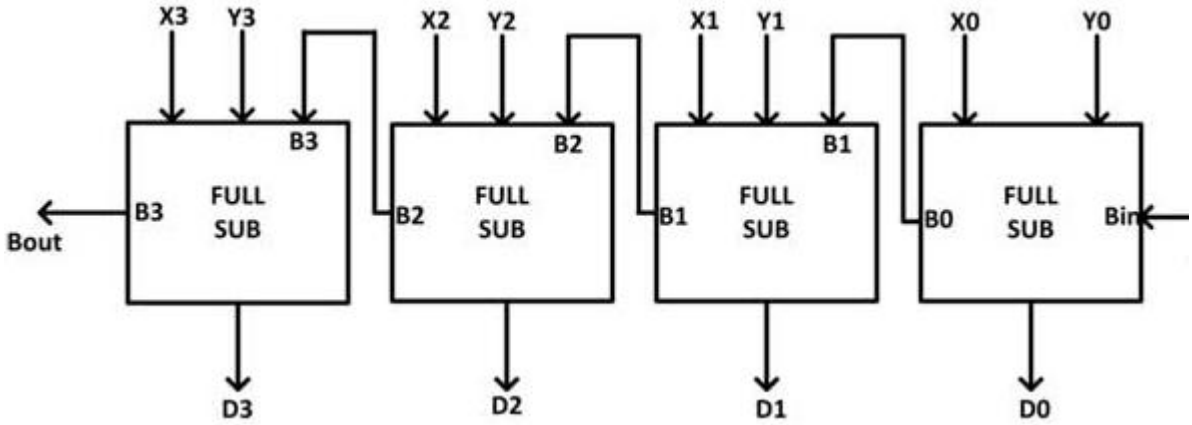
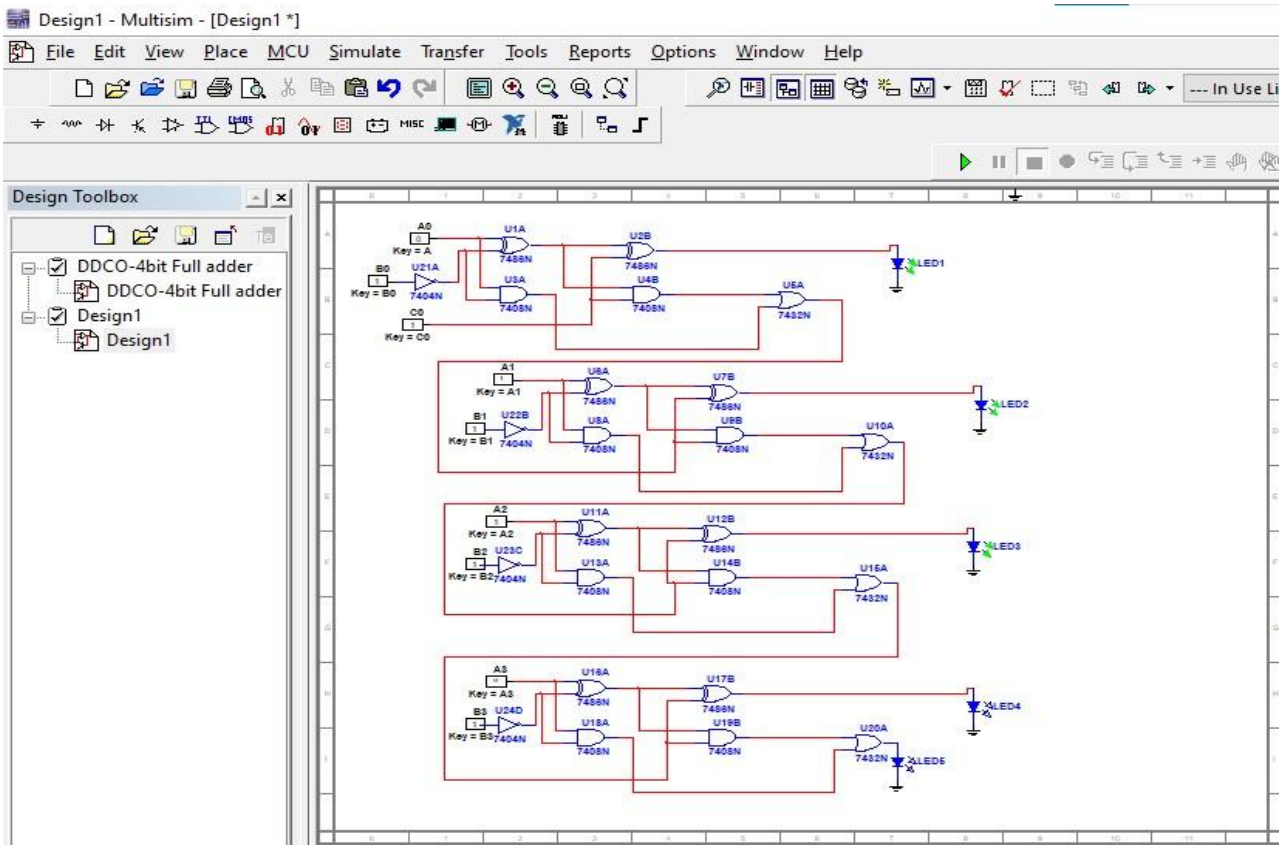
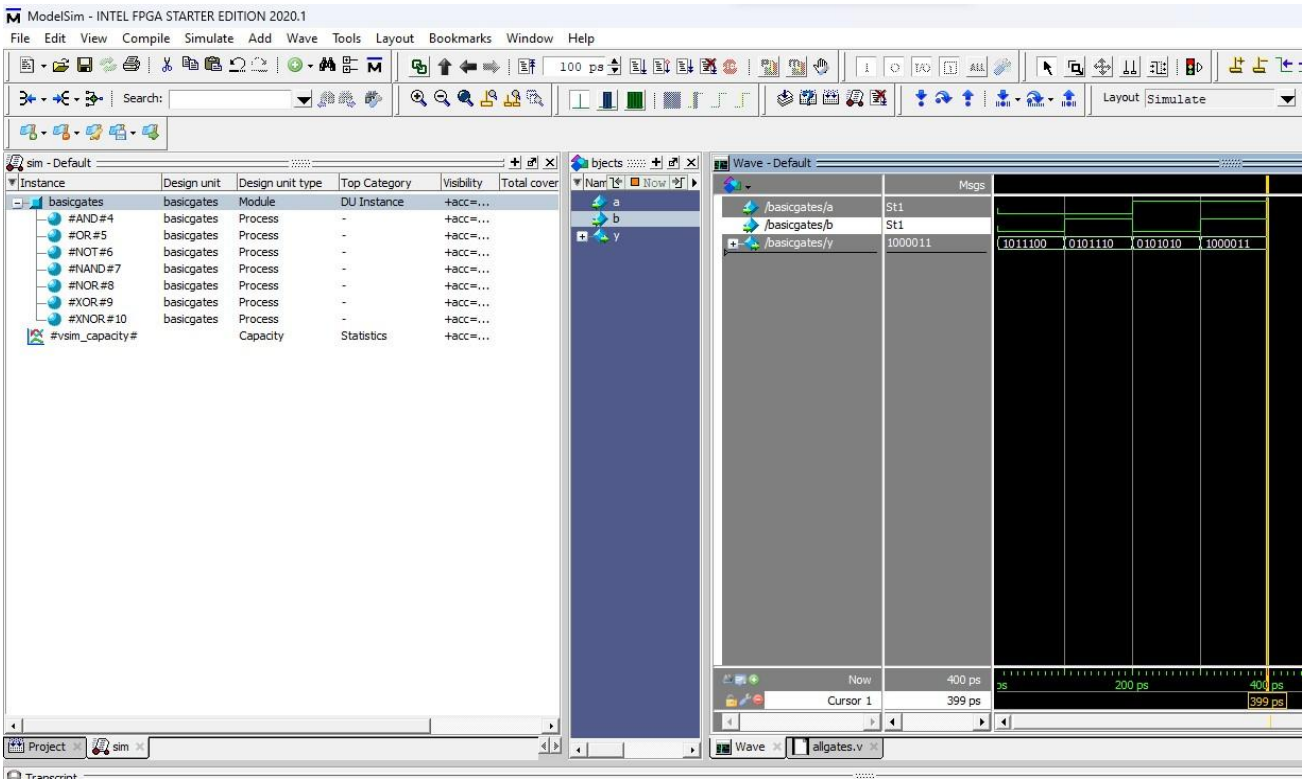


Figure 1 - Full Subtractor



Implementation of 4-bit adder, subtractor using 1-bit full Adder:





Conclusion: 4-bit Full adder and full subtractor circuits are simulated using multisim

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

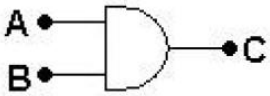
Aim: To design Verilog HDL to implement simple circuits using structural, Data flow and Behavioral model.

Objective: To design the basic gates in verilog using xilinx tool and verify it on ISE simulator.

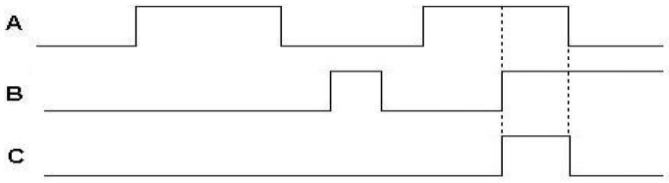
Tool Required: Model Sim

AND GATE:

Symbol and Truth table and Waveform of AND gate



A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

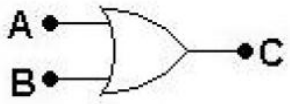


Design Description:

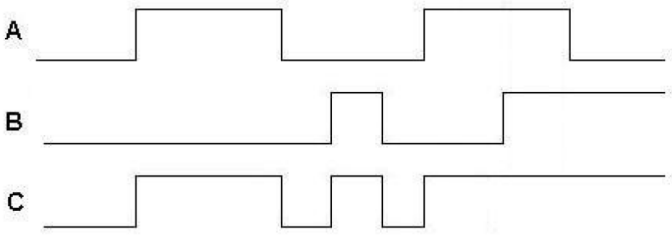
The output of an AND gate is only equal to 1 if both inputs (A AND B in this case) are equal to 1. Otherwise the output is equal to 0. The above picture shows a two input AND gate, but an AND gate can have many inputs. In any case, no matter how many inputs it has, the output is only equal to 1 if all the inputs are equal to 1, otherwise the output is 0. The equation of an AND gate is: **C = A & B**

OR GATE:

Symbol and Truth table and Waveform of OR gate



A	B	C
0	0	0
0	1	1
1	0	1
1	1	1



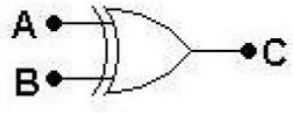
Design Description:

The output of an OR gate is equal to 1 if either input (A OR B in this case) is equal to one. If neither input is equal to 1, the output is equal to zero. Again, the above picture shows a two input OR gate, but an OR gate can have as many inputs as you like. The output will be equal to 1 if any of the inputs is equal to 1.

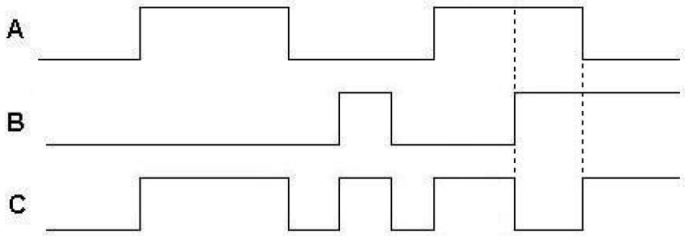
The equation of an OR gate is: **C = A + B**

EX-OR GATE:

Symbol and Truth table and waveform of XOR gate:



A	B	C
0	0	0
0	1	1
1	0	1
1	1	0



Design Description:

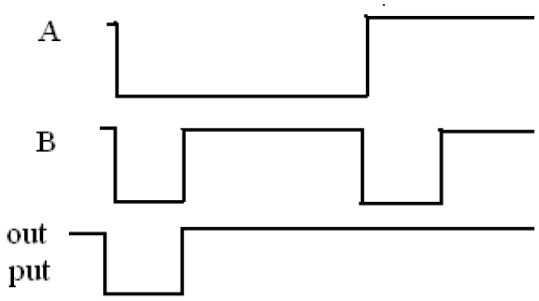
The output of an XOR gate is equal to 1 if either input (A or B in this case) is equal to one, but equal to zero if both inputs are equal to zero or if both inputs are equal to 1. This is the difference between an OR gate and an XOR gate, an OR gates output will equal 1 if both inputs are equal to 1.

The equation OF an XOR gate is: **C = A ^ B**

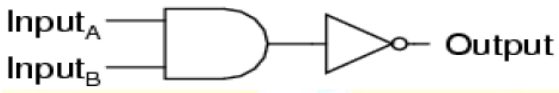
NAND GATE:

Symbol and Truth table and waveform of NAND gate:

A	B	Output
0	0	1
0	1	1
1	0	1
1	1	0



Equivalent gate circuit



Design Description:

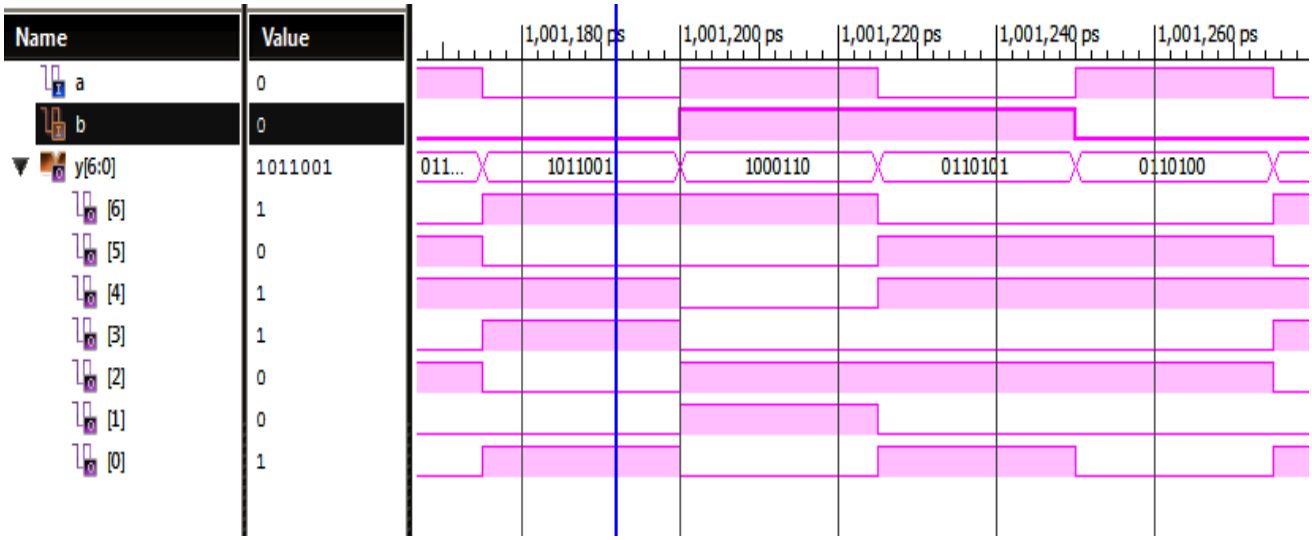
A variation on the idea of the AND gate is called the NAND gate. The word "NAND" is a verbal contraction of the words NOT and AND. Essentially, a NAND gate behaves the same as an AND gate with a NOT (inverter) gate connected to the output terminal. To symbolize this output signal inversion, the NAND gate symbol has a bubble on the output line. The truth table for a NAND gate is as one might expect, exactly opposite as that of an AND gate:As with AND gates, NAND gates are made with more than two inputs. In such cases, thesame general principle applies: the output will be "low" (0) if and only if all inputs are "high" (1). If any input is "low" (0), the output will go "high" (1).

The equation of an XOR gate is: **C = ~(A &B)**

Behavioral Modeling	**Data Flow Modeling**	**Gate Level/Structural Modeling**

module allgatesverilog(a,b,y);
input a;
input b;
output reg[6:0] y;
always@(a,b)
begin
y[0]=~a;
y[1]=a&b;
y[2]=a|b;
y[3]=~(a&b);
y[4]=~(a|b);
y[5]=a^b;
y[6]=~(a^b);
end
endmodule
	module allgatesverilog(a,b,y);
	input a;
	input b;
	output [6:0] y;
	assign y[0]=~a;
	assign y[1]=a&b;
	assign y[2]=a|b;
	assign y[3]=~(a&b);
	assign y[4]=~(a|b);
	assign y[5]=a^b;
	assign y[6]=~(a^b);
	endmodule
		module allgatesverilog(a,b,y);
		input a;
		input b;
		output [6:0] y;
		not n0(y[0],a);
		and n1(y[1],a,b);
		or n2(y[2],a,b);
		nand n3(y[3],a,b);
		nor n4(y[4],a,b);
		xor n5(y[5],a,b);
		xnor n6(y[6],a,b);
		endmodule

OUTPUT:



Conclusion: Simple Logic Circuits are designed using Verilog HDL and Implemented using Model Sim.

4. Design Verilogg HDL to implement Binary Adder-Subtractor

Aim: To Design Verilogg HDL to implement Binary Adder-Subtractor half, full adder and half, full Subtractor.

Tool Required : Model Sim

Objective: To verify the above adders and subtractor using verilog.

Verilog Code for Half Adder:

```
module fadd(s,c,a,b);
input a,b;
output reg s,c;
always@(a,b)
begin
s=a^b;
c=a&b;
end
endmodule
```

Verilog Code for Full Adder:

BEHAVIORAL MODELING USING case statement

```
module fadd(sum,cout,a,b,cin);
input a,b,cin;
output regsum,cout;
always@(a,b,cin)
begin
case({a,b,cin})
3'b000:begin sum=1'b0;cout=1'b0;end
3'b001:begin sum=1'b1;cout=1'b0;end
3'b010:begin sum=1'b1;cout=1'b0;end
3'b011:begin sum=1'b0;cout=1'b1;end
3'b100:begin sum=1'b1;cout=1'b0;end
3'b101:begin sum=1'b0;cout=1'b1;end
3'b110:begin sum=1'b0;cout=1'b1;end
3'b111:begin sum=1'b1;cout=1'b1;end
endcase
end
endmodule
```

BEHAVIORAL MODELING

```
module
fadd(sum,cout,a,b,cin);
input a,b,cin;
output regsum,cout;
always@(a,b,cin)
begin
sum=a^b^cin;
cout=(a&b)|(b&cin)|(cin&a);
end
endmodule
```

GATE LEVEL MODELING

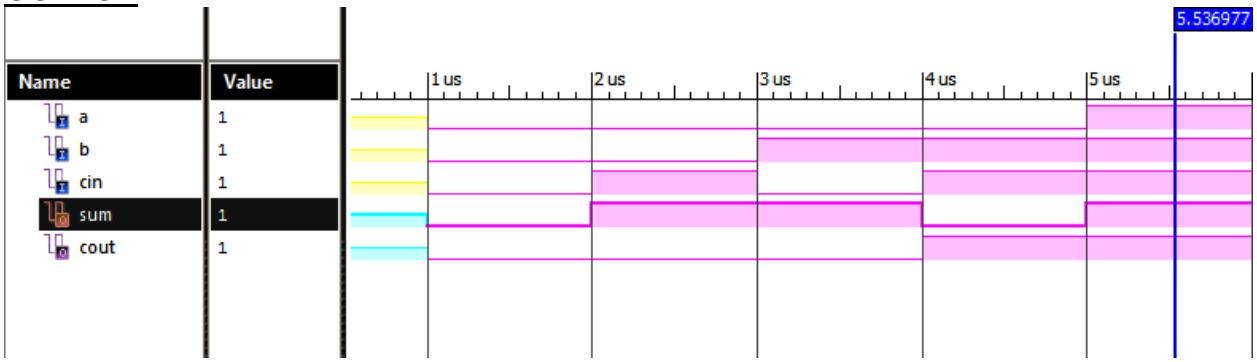
```
module fadd( sum,cout,a,b,cin);
input a,b,cin;
output sum,cout;
wire c1,c2,s1;
half_adder h1(s1,c1,a,b);
half_adder h2(sum,c2,s1,cin);
or o1(cout,c1,c2);
endmodule

//HA PROGRAM
module half_adder(s,c,x,y);
input x,y;
output s,c;
assign s=x^y;
assign c=x&y;
endmodule
```

DATAFLOW MODELING

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

OUTPUT



Verilog Code for Half Subtractor:

```
module fadd(d,b,a,b);
input a,b;
output reg d,b;
always@(a,b)
begin
d=a^b;
b=(~a)&b;
end
endmodule
```

Verilog Code for Full Subtractor:

```
module fadd(diff,borrow,a,b,bin);
input a,b,cin;
output reg diff,borrow;
always@(a,b,bin)
begin
diff = a^b^bin;
borrow =(~a& bin)|(~a&b)|(a& bin);
end
endmodule
```

Conclusion: 4-bit Adders and Subtractor are designed using Verilog HDL and Implemented using Model Sim.

5. Design Verilog HDL to implement Decimal Adder

Aim: To design Verilog HDL to implement Decimal Adder

Tool Required: Model Sim

Objective: To understand Decimal Adder

Theory: Computers understand binary number system while humans are used to arithmetic operations in decimal number systems. To enhance Computer-Human relationship in this perspective, arithmetic operations are performed by the computer in a *binary coded decimal*, (BCD) form.
In this article I'll analyse and design a BCD adder which requires a minimum of nine inputs and five outputs, four bits are require to code the aguend and the addend making eight bits, and the circuit input carry makes the nine inputs. The four bit sum output and the output carry represents the five outputs.

Verilog Code for Decimal Adder

```
module BCD_Adder ( A,B,C);
input [3:0] A, B, C;
 output [3:0] sum,;
output C_out;
 sum = A + B + C;
always@(A,B,C)
begin
if (sum > 9)
sum=sum+6;
else
sum=sum;
end
endmodule
```

Conclusion: Decimal Adder is designed using Verilog HDL and Implemented using Model Sim.

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

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

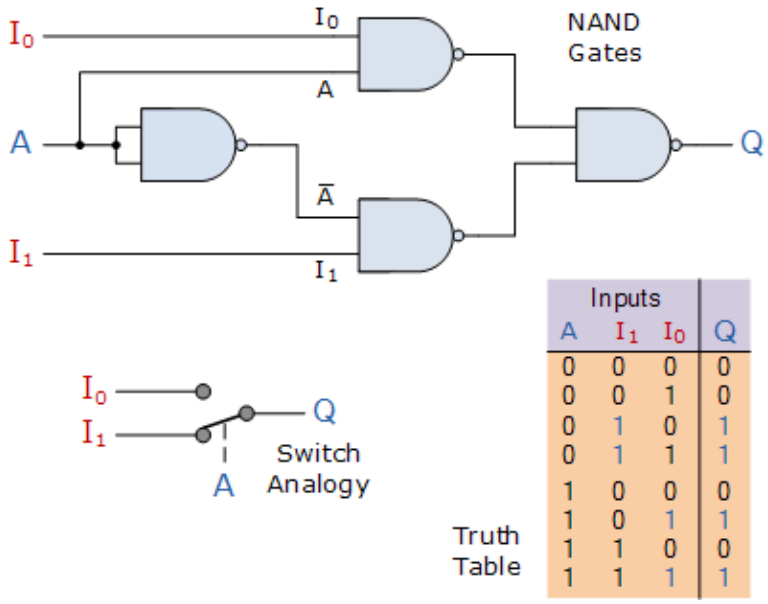
Tool Required : Model Sim

Objective: To verify the above mentioned combinational logics using verilog.

Theory: Multiplexing is the generic term used to describe the operation of sending one or more analogue or digital signals over a common transmission line at different times or speeds and as such, the device we use to do just that is called a **Multiplexer**.The *multiplexer*, shortened to "MUX" or "MPX", is a combinational logic circuit designed to switch one of several input lines through to a single common output line by the application of a control signal. Multiplexers operate like very fast acting multiple position rotary switches connecting or controlling multiple input lines called "channels" one at a time to the output.

Multiplexers, or MUX's, can be either digital circuits made from high speed logic gates used to switch digital or binary data or they can be analogue types using transistors, MOSFET's or relays to switch one of the voltage or current inputs through to a single output.

2-input Multiplexer Design



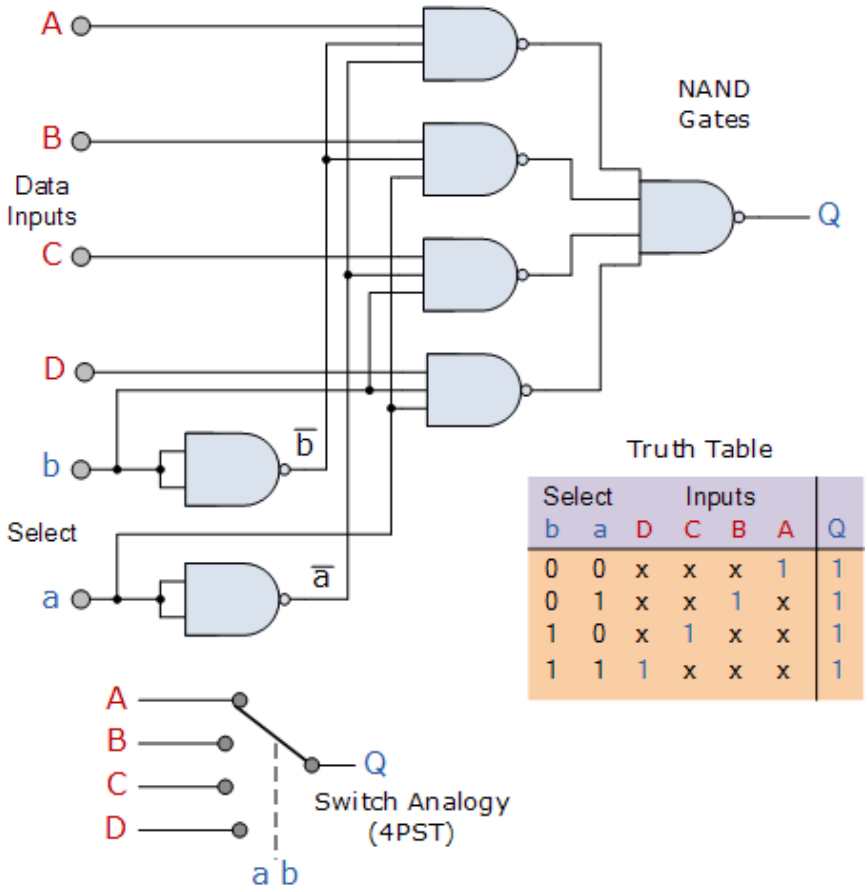
Inputs			
A	I_1	I_0	Q
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

VERILOG CODE FOR 2TO 1 MULTIPLEXER

```
module mux2_1(d,s,out);
input d0,d1;
input s;
output out;
always@(d,s)
begin
case(s)
```

```
1'b0:out=d0;
1'b1:out=d1;
endcase
end
endmodule
```

4-to-1 Multiplexer



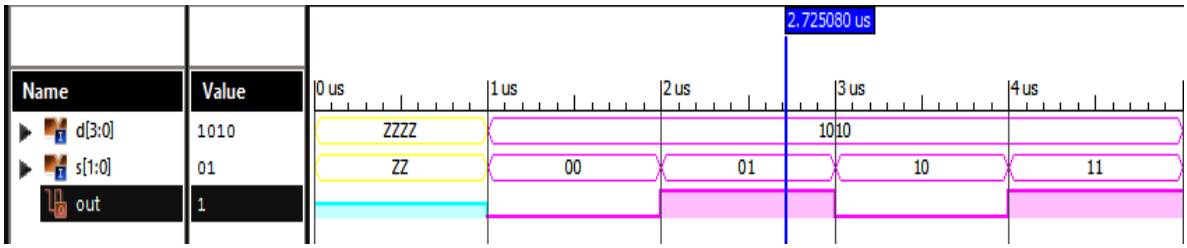
VERILOG CODE FOR 4 TO 1 MULTIPLEXER

BEHAVIORAL MODELING	DATA FLOW MODELING	GATE LEVEL MODELING

module mux4_1(d,s,out);
input[3:0]d;
input[1:0]s;
output reg out;
always@(d,s)
begin
case(s)
2'b00:out=d[0];
2'b01:out=d[1];
2'b10:out=d[2];
	module mux4_1(d,s,out);
	input[3:0]d;
	input[1:0]s;
	output out;
	assign out=(~s[1] & ~s[0] & d[0])|(~s[1] & s[0] & d[1])|(s[1] & ~s[0] & d[2])|(s[1] & s[0] & d[3]);
	endmodule
		module mux4_1(d,s,out);
		input[3:0]d;
		input[1:0]s;
		output out;
		wire s1n,s0n;
		wire [3:0]y;
		
		not(s1n,s[1]);
		not(s0n,s[0]);
		and(y[0],d[0],s1n,s0n);

default:out=d[3];
endcase
end
endmodule
		and(y[1],d[1],s1n,s[0]);
		and(y[2],d[2],s[1],s0n);
		and(y[3],d[3],s[1],s[0]);
		or(out,y[0],y[1],y[2],y[3]);
		endmodule

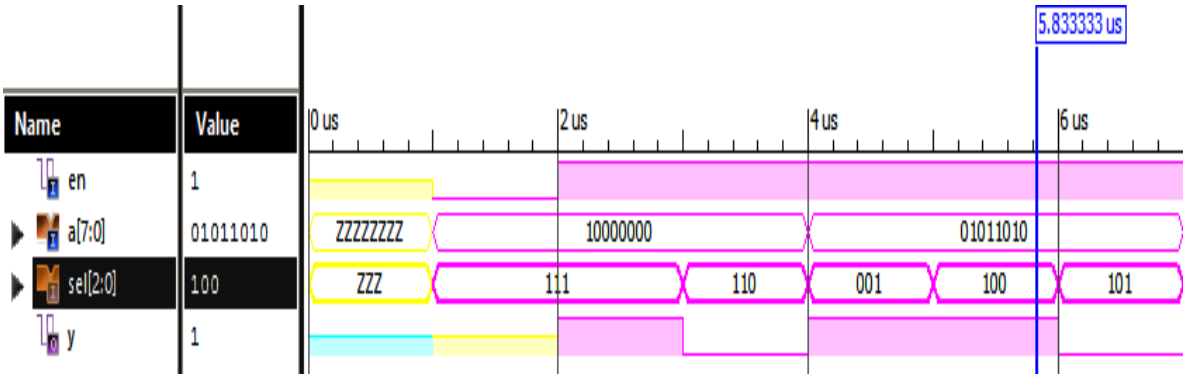
OUTPUT



BEHAVIORAL MODELING using nested if-else	BEHAVIORAL MODELING using case statement

module mux8_1(a,sel,en,y);
input en;
input[7:0]a;
input[2:0]sel;
output reg y;
always@(a,sel,en)
begin
if(en==1)
begin
if(sel==0)
y=a[0];
else if(sel==1) y=a[1];
else if(sel==2) y=a[2];
else if(sel==3) y=a[3];
else if(sel==4) y=a[4];
else if(sel==5) y=a[5];
else if(sel==6) y=a[6];
else if(sel==7) y=a[7];
else y=1'bx;
end
else
y=1'bz;
end
endmodule
	module mux8_1(en,a,sel,y);
	input en;
	input[7:0]a;
	output reg y;
	input[2:0]sel;
	always@(a,sel,en)
	begin
	if(en==1)
	case(sel)
	3'd0:y=a[0];
	3'd1:y=a[1];
	3'd2:y=a[2];
	3'd3:y=a[3];
	3'd4:y=a[4];
	3'd5:y=a[5];
	3'd6:y=a[6];
	3'd7:y=a[7];
	default:y=1'bx;
	endcase
	else y=1'bz;
	end
	endmodule

OUTPUT



Conclusion: Types of Mux is designed using Verilog HDL and Implemented using Model Sim.

7. Design Verilog program to implement types of De-Multiplexer.

Aim: To design Verilog program to implement types of De-Multiplexer.

Tool Required: Model Sim

Objective: To understand demultiplexer and its types

Theory: Demultiplexer is one to many. The various types of demultiplexer are 1to 2 demux, 1-to-4 demux,1-to-8 demux and so on.

Verilog code for 1-to-2 Demux

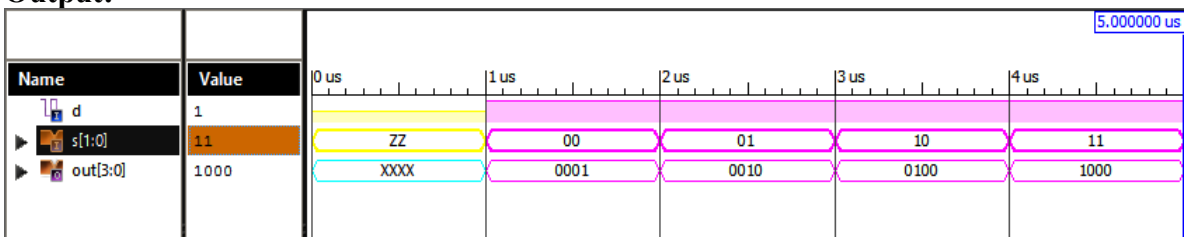
```
module OneToTwoDemux (
  input wire D,
  input wire Sel,
  output reg Y0,
  output reg Y1
  );

  always @* begin
    Y0 = (Sel == 1'b0) ? D : 1'b0;
    Y1 = (Sel == 1'b1) ? D : 1'b0;
  end
endmodule
```

Verilog code for 1-to-4 Demux

```
module demux1_4(d,s,out);
input d;
input[1:0]s;
output [3:0]out;
wire s1n,s0n;
not(s1n,s[1]);
not(s0n,s[0]);
and(out[0],d,s1n,s0n);
and(out[1],d,s1n,s[0]);
and(out[2],d,s[1],s0n);
and(out[3],d,s[1],s[0]);
endmodule
```

Output:



Conclusion: Types of Demux is designed using Verilog HDL and Implemented using Model Sim.

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

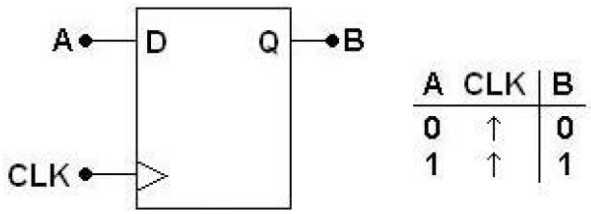
Aim: To design Verilog program to implement various types of Flip-Flops such as SR, JK and D.

Tool Required: Model Sim

Objective: To understand Flip-Flops and its types

Theory: D FLIPFLOP

Symbol and truth table



A	CLK	B
0	↑	0
1	↑	1

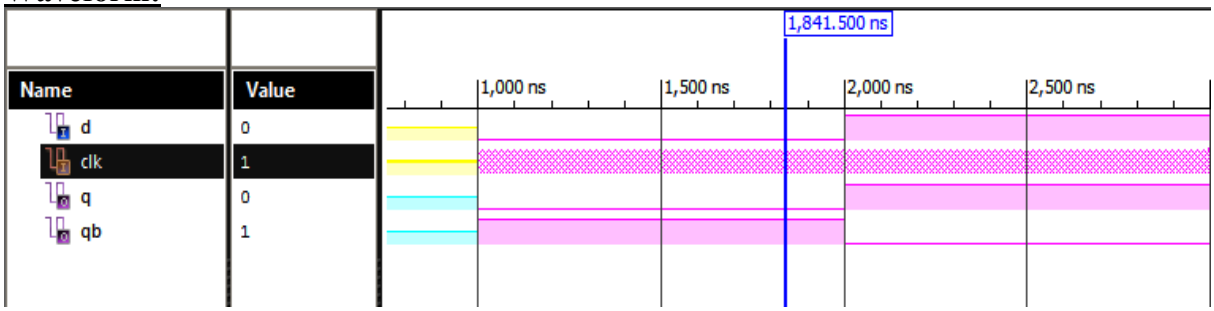
Design Description:

D Flip Flop has two inputs, the clock and the D input, and one output, Q. In the picture D is connected to the node A, and Q is connected to the node B, so these are essentially names of the same thing. As can be seen in the truth table, the output is equal to the input on the rising edge of the clock. If there is no rising clock edge, the output will remain in its current state.

BEHAVIORAL MODELING	DATA FLOW MODELING

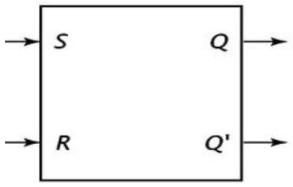
module d_ff(d,clk,q,qb);
input d,clk;
output reg q,qb;
always@(posedge clk)
begin
case(d)
1'b0:q=0;
1'b1:q=1;
endcase
qb=~q;
end
endmodule
	module d_ff(d,clk,q,qb);
	input d,clk;
	inout q,qb;
	wire x,y,db;
	not i1(db,d);
	nand n1(x,d,clk);
	nand n2(y,db,clk);
	nand n3(q,x,qb);
	nand n4(qb,y,q);
	endmodule

Waveform:



SR FLIPFLOP:

Symbol and truth table



S	R	Q
0	0	Q_0
0	1	0
1	0	1
1	1	Undefined

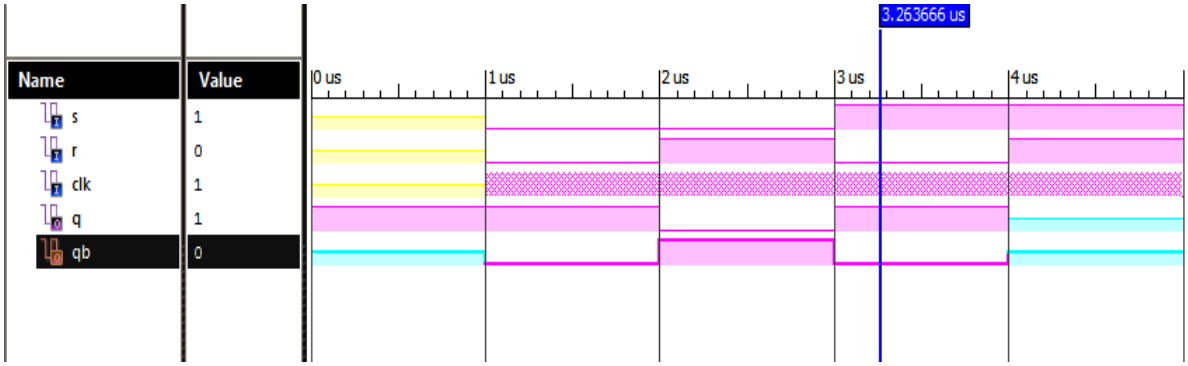
Design Description:

An SR Flip Flop is an arrangement of logic gates that maintains a stable output even after the inputs are turned off. This simple flip flop circuit has a set input (S) and a reset input (R). The set input causes the output of 0 (top output) and 1 (bottom output). The reset input causes the opposite to happen (top = 1, bottom =0).

BEHAVIORAL MODELING	GATE-LEVEL MODELING

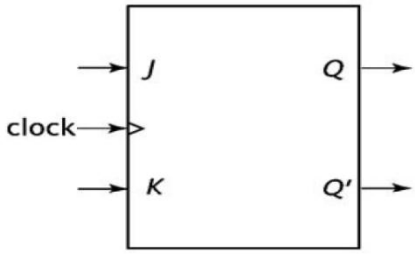
module sr_ff(s,r,clk,q,qb);
input s,r,clk;
output q,qb;
reg q,qb;
initial
begin
q=1;
end
always@(posedge clk)
begin
case({s,r})
2'b00:q=q;
2'b01:q=0;
2'b10:q=1;
2'b11:q=1'bx;
endcase
qb=~q;
end
endmodule
	module sr_ffg(s,r,clk,q,qb);
	input s,r,clk;
	inout q,qb;
	wire x,y;
	nand n1(x,s,clk);
	nand n2(y,r,clk);
	nand n3(q,x,qb);
	nand n4(qb,y,q);
	endmodule

Waveform:



JK FLIP FLOP:

Symbol and truth table:



J	K	clock	Q
X	X	not↑	Q_0
0	0	↑	Q_0
0	1	↑	0
1	0	↑	1
1	1	↑	Q_0'

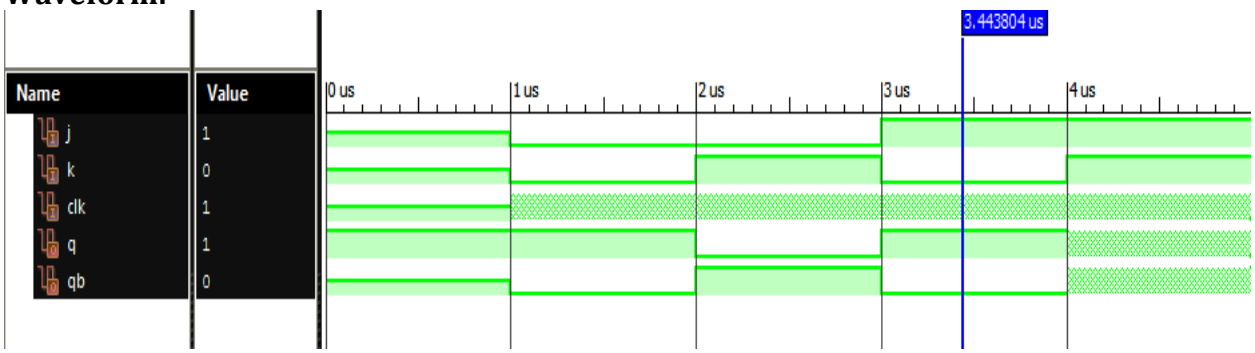
Design Description:

The J-K flip-flop is perhaps the most widely used type of flip-flop. Its function is identical to that of the S-R flip flop in the SET, RESET and HOLD conditions of operation. The difference is that the J-K flip-flop does not have any invalid states. The logic symbol for the J-K flip-flop is presented in Figure above and its corresponding truth table is listed in Table . Notice that for J=1 and K=1 the output toggles, that is to say that the output at time *t* is complemented at time *t+1*.

BEHAVIORAL MODELING

```
module jk_ff(j,k,clk,q,qb);
input j,k,clk;
output reg q,qb;
initial
begin
q=1;
end
always@(posedge clk)
begin
case({j,k})
2'b00:q=q;
2'b01:q=0;
2'b10:q=1;
2'b11:q=~q;
endcase
qb=~q;
end
endmodule
```

Waveform:



Conclusion: Design of Flipflops(D,T,SR,JK) in xilinx tool and is verified according to the truth

Virtual LAB

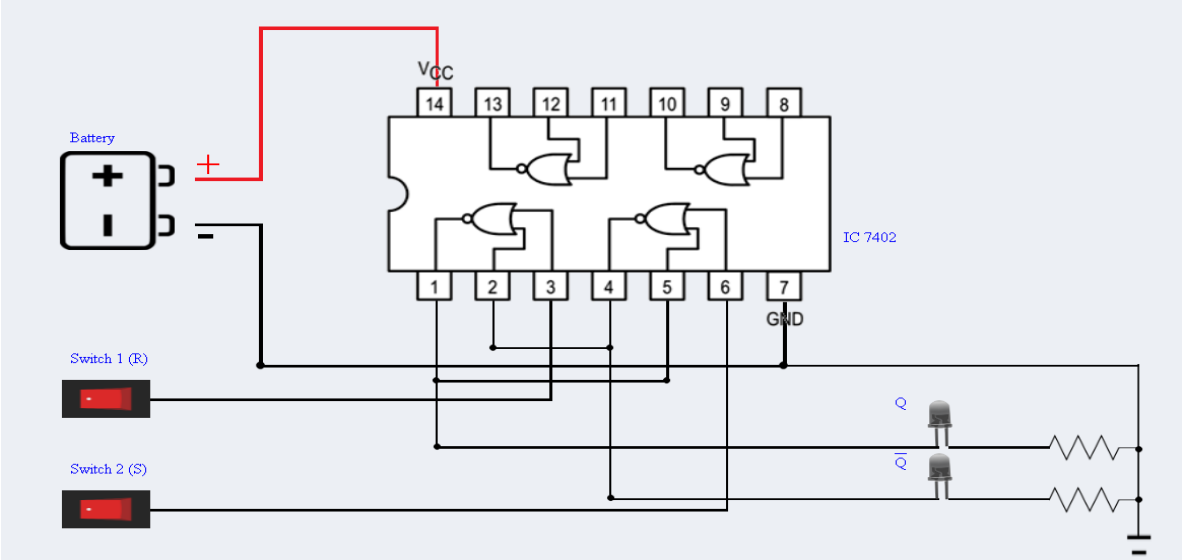
Construction of NOR gate latch and verification of its operation

Aim: To verify the truth table and timing diagram of NOR gate latch using NOR gate IC and analyse the circuit of NOR gate latch with the help of LEDs display.

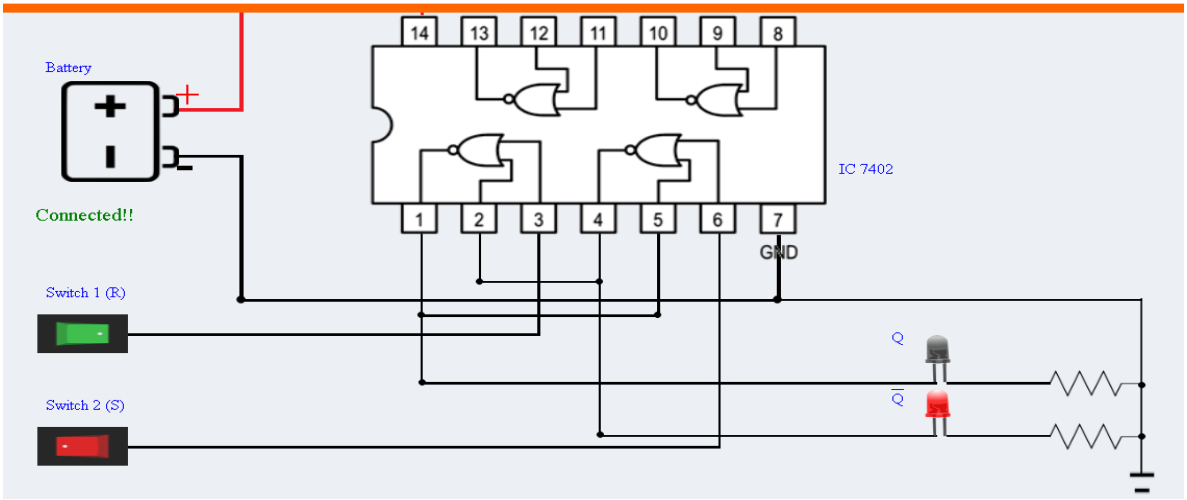
Tools Required: Virtual Simulator Platform support by MHRD.

Objective: To learn NOR gate latch operation

Logic Circuit Diagram:



Snapshot: Logic Operation: Switch 1=0, Switch 2=0



Snapshot: Logic Operation: Switch 1=1, Switch 2=0

Conclusion: To understand working NOR Gate using Virtual Lab