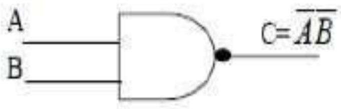
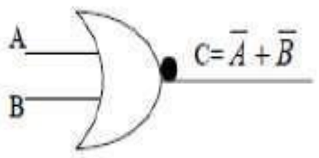
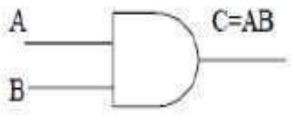
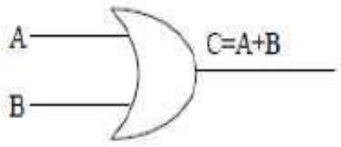
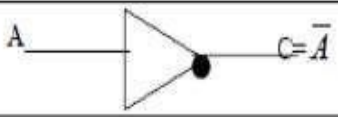
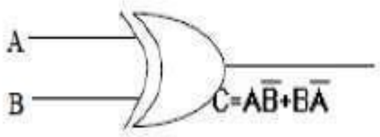
PRACTICAL COMPONENT OF IPCC

Sl.N O	Experiments Simulation packages preferred: Multisim, Modelsim, PSpice or any other relevant
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 Behavioural 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 multiplexer 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.

A logic gate performs a logical operation on one or more logic inputs and produces a single logic output. The logic normally performed is Boolean logic and is most commonly found in digital circuits.

Truth table with symbols

S.NO	GATE	SYMBOL	INPUTS		OUTPUT
			A	B	C
1.	NAND IC 7400		0	0	1
			0	1	1
			1	0	1
			1	1	0
2.	NOR IC 7402		0	0	1
			0	1	0
			1	0	0
			1	1	0
3.	AND IC 7408		0	0	0
			0	1	0
			1	0	0
			1	1	1
4.	OR IC 7432		0	0	0
			0	1	1
			1	0	1
			1	1	1
5.	NOT IC 7404		1	-	0
			0	-	1
6.	EX-OR IC 7486		0	0	0
			0	1	1
			1	0	1
			1	1	0

Verilog is a hardware description language (HDL) that is used to describe digital systems and circuits in the form of code. It was developed by Gateway Design Automation in the mid-1980s and later acquired by Cadence Design Systems.

Verilog is widely used for design and verification of digital and mixed-signal systems, including both application-specific integrated circuits (ASICs) and field-programmable gate arrays (FPGAs). It supports a range of levels of abstraction, from structural to behavioral, and is used for both simulation-based design and synthesis-based design.

The language is used to describe digital circuits hierarchically, starting with the most basic elements such as logic gates and flip-flops and building up to more complex functional blocks and systems. It also supports a range of modeling techniques, including gate-level, RTL-level, and behavioral-level modeling.

A Verilog program can be simulated in **Xilinx** Project Navigator. Xilinx ISE (Integrated Synthesis Environment) is a discontinued software tool from Xilinx for synthesis and analysis of HDL designs, which primarily targets development of embedded firmware for Xilinx FPGA and CPLD integrated circuit (IC) product families.

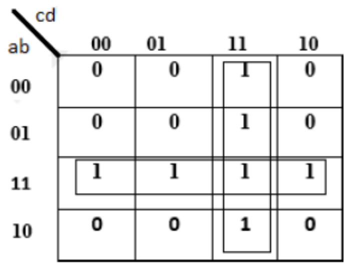
VERILOG CODE

```
module allgate ( a, b, yand,yor,ynot,ynand,ynor,yxor,yxnor );
input a,b;
output yand, yor, ynot, ynand, ynor, yxor, yxnor;

assign yand = a&b;            // ANDOperation
assign yor = a |b;            // OROperation
assign ynot = ~a;             // NOTOperation
assign ynand = ~(a&b);        // NANDOperation
assign ynor = ~(a|b);         //NOROperation
assign yxor = a^b;            //XOROperation
assign yxnor=~(a^b);          //XNOROperation
endmodule                     // END of themodule
```

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

Design:

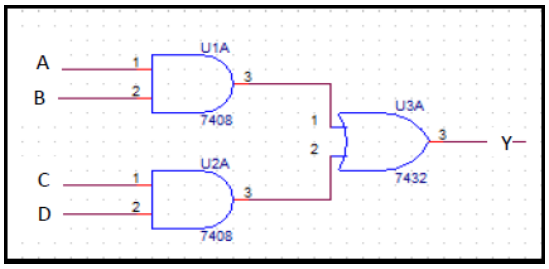
Consider the function f(a,b,c,d) $=\sum m\ (3,7,11,12,13,14,15)$



ab \ cd	00	01	11	10
00	0	0	1	0
01	0	0	1	0
11	1	1	1	1
10	0	0	1	0

Y = ab + cd

Circuit Diagram

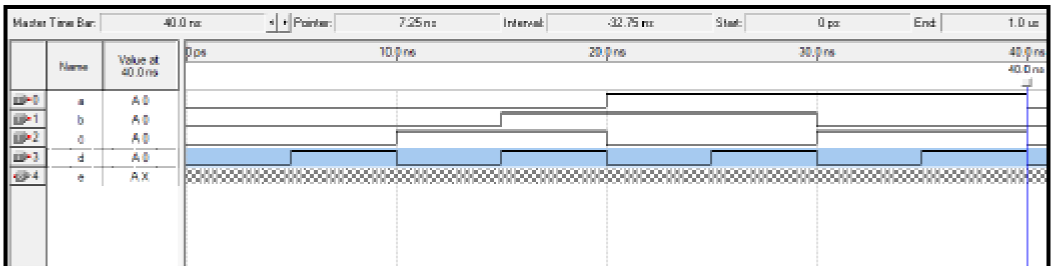


Truth Table

A	B	C	D	Y
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Verilog Code

```
module f(a,b,c,d,e);
        Input a,b,c,d;
        Output e;
        Assign e=a&b|c&d;
endmodule
```


Master Time Bar: 40.0 ns Pointer: 7.25 ns Interval: -32.75 ns Start: 0 ps End: 1.0 us

Name	Value at 40.0 ns
a	A 0
b	A 0
c	A 0
d	A 0
e	A X

Output

The four variable logic expression is simplified using K-map and simplified the same using basic gates.

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

Design for Full Adder

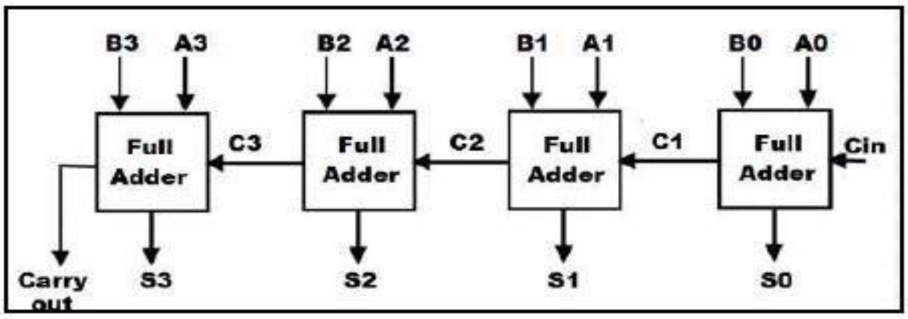
Truth Table

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

Sum = A'B'C + A'BC' + AB'C' + ABC

Cout = A'BC + AB'C + ABC' + ABC

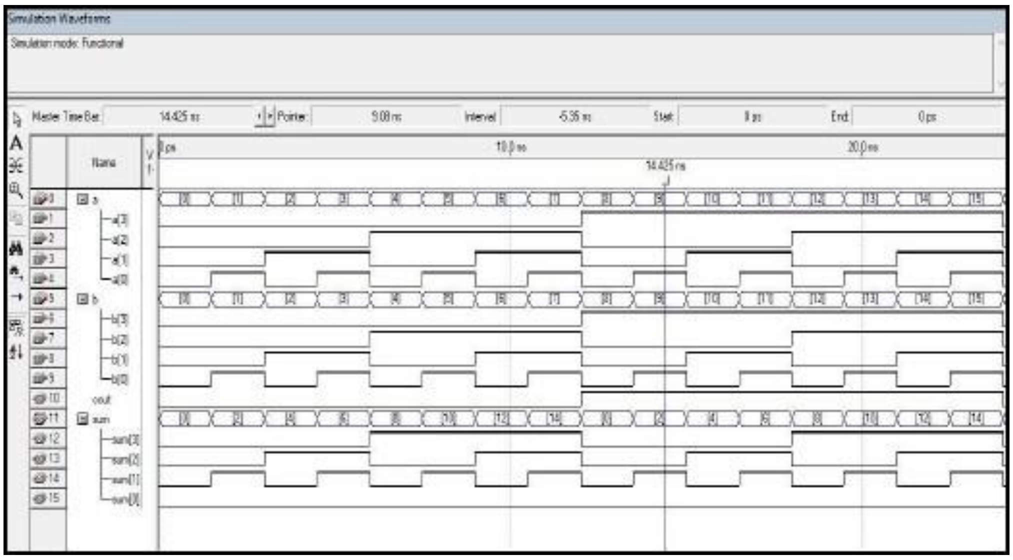
Circuit Diagram of 4-bit Full Adder



Verilog code

```
module fa(a,b,cin,sum,cout);
        input a,b,cin;
        output sum,cout;
        assign sum = ((!a & !b & cin) | (!a & b & !c) | (a & !b & !c) | (a & b & c));
        assign cout = ((!a & b & c) | (a & !b & c) | (a & b & !c) | (a & b & c));
endmodule

module fourbitadder(a,b,sum,c);
        input [3:0]a;
        input [3:0]b;
        output [3:0]sum;
        output c;
        wire c1,c2,c3;
        fa g0(a[0],b[0],0,sum[0],c1);
        fa g1(a[1],b[1],c1,sum[1],c2);
        fa g2(a[2],b[2],c2,sum[2],c3);
        fa g3(a[3],b[3],c3,sum[3],c);
endmodule
```

Output



Simulation Waveforms

Simulation mode: Functional

Master Time Bar: 14.425 ns Pointer: 9.08 ns Interval: -5.35 ns Start: 0 ps End: 0 ps

Design for Full Subtractor

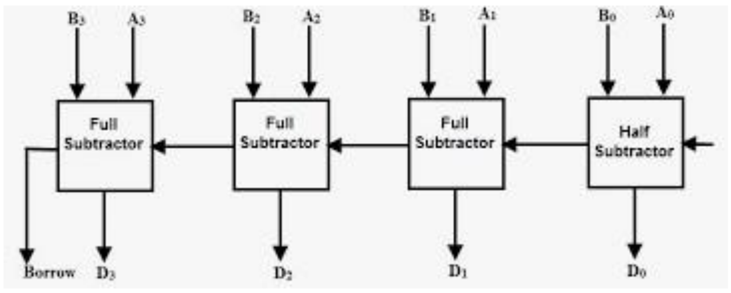
Truth Table

A	B	Cin	Diff	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

Sum = A’B’C + A’BC’ + AB’C’ + ABC

Cout = A’B’C + A’BC’ + A’BC + ABC

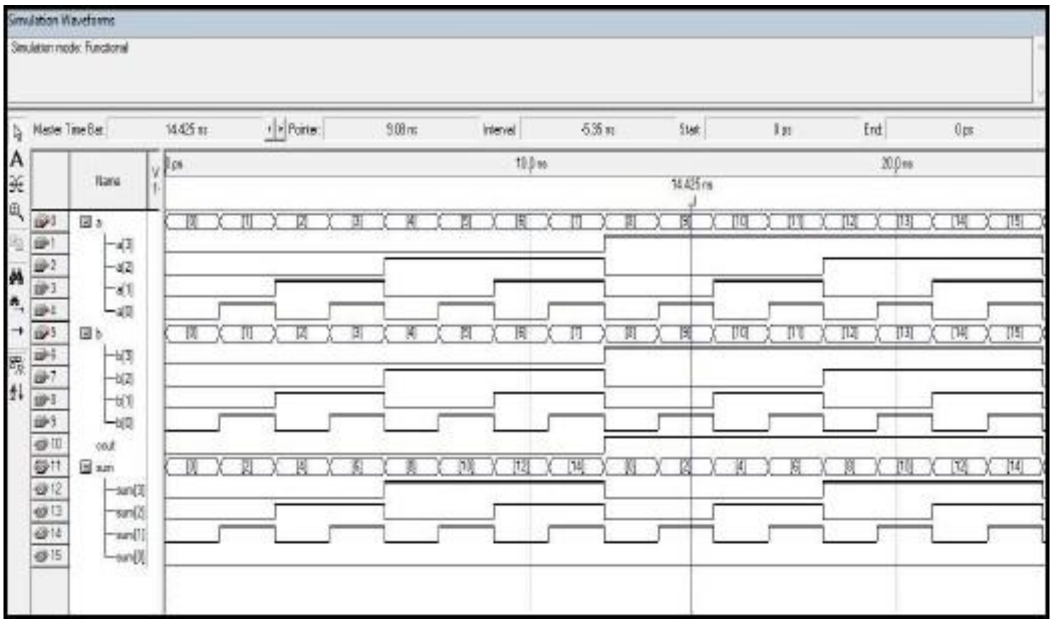
Circuit diagram of four bit full Subtractor



Verilog Program

```
module fs(a,b,c,d,bout);
        input a,b,c;
        output d,bout;
        assign d = ((!a&!b&c) | (!a&b&!c) | (a&!b&!c) | (a&b&c));
        assign bout = ((!a&!b&c) | (!a&b&!c) | (!a&b&c) | (a&b&c));
endmodule

module fourbitsub(a,bin,diff,bout);
        input [3:0]a;
        input [3:0]bin;
        output [3:0]diff;
        output bout;
        wire b1,b2,b3;

        fs g0(a[0],bin[3],0,diff[0].b1);
        fs g1(a[1],bin[2],b1,diff[1],b2);
        fs g2(a[2],bin[1],b2,diff[2],b3);
        fs g3(a[3],bin[0],b3,diff[3],bout);
endmodule
```

Output


Simulation Waveforms
Simulation mode: Functional
Master Time Bar: 14.425 ns Pointer: 9.08 ns Interval: -5.35 ns Start: 0 ps End: 0 ps

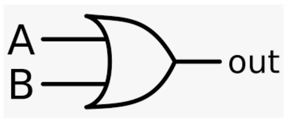
The four bit full adder and Subtractor is designed and simulated the same using basic gates.

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

Design of a simple OR Gate

Truth table

A	B	Y=A + B
0	0	0
0	1	1
1	0	1
1	1	1



Structural model

```
module structural(a,b,y);
        input a,b;
        output y;
        or g1(y,a,b);
endmodule
```

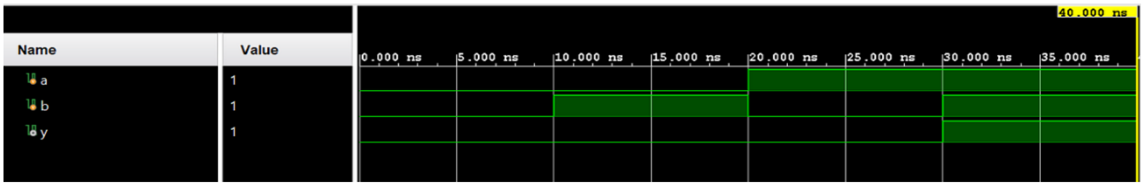
Dataflow model

```
module dataflow(a,b,y);
        input a,b;
        output y;
        assign y=a|b;
endmodule
```

Behavioural model

```
module beh(a,b,y);
        input  a,b;
        output  y;
        reg y;
        always @(a or b)
                begin
                        if((a==0)&&(b==0))
                                y=0;
                        else

                                y=1;
                end
endmodule
```

Output



The simple OR gate is implemented using Structural, Dataflow and Behavioral model.

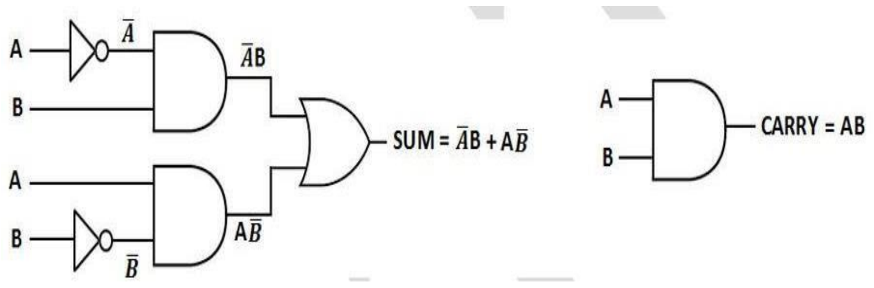
4. Design Verilog HDL to implement Binary Adder-Subtractor – Half and Full Adder, Half and Full Subtractor.

Design of Half Adder

INPUTS		OUTPUTS	
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$$S=\bar{A}B+A\bar{B}=A\oplus B$$
$$C=A\,B$$

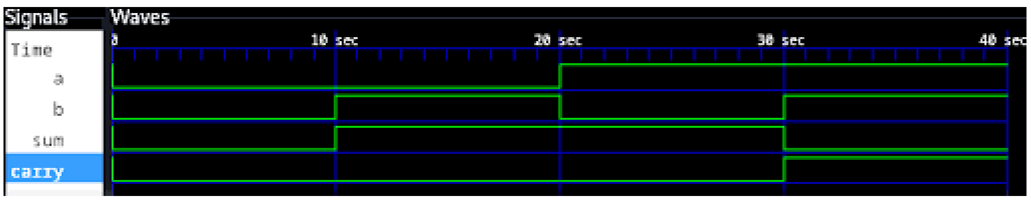
Circuit Diagram



Verilog Code

```
module ha(a,b,s,c);
            input a,b;
            output s,c;
            assign s=a^b;
            assign c=a&b;
endmodule
```

Output



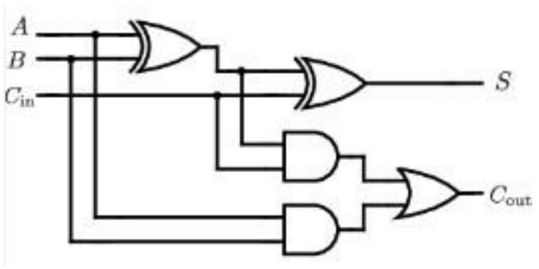
Design of Full Adder

INPUTS			OUTPUTS	
A	B	Cin	S	C
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

$$S = A \oplus B \oplus Cin$$

$$C = Cin\,(A \oplus B) + AB$$

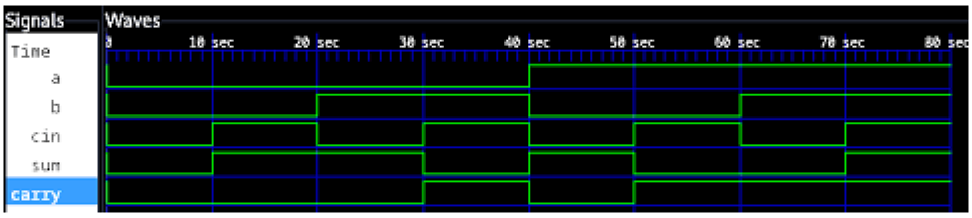
Circuit Diagram



Verilog Code

```
module fa(a,b,cin,s,cout);
            input a,b,cin;
            output s,cout;
            assign s = a^b^cin;
            assign cout=(c&(a^b))|(a&b);
endmodule
```

Output



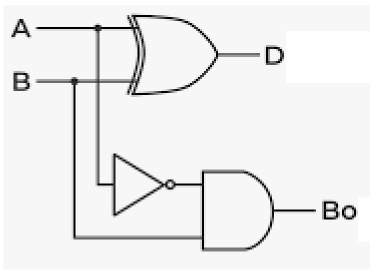
Design of Half Subtractor

A	B	D	B_O
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

$$D = \overline{A}.B + A.\overline{B}$$

$$B_o = \overline{A}.B$$

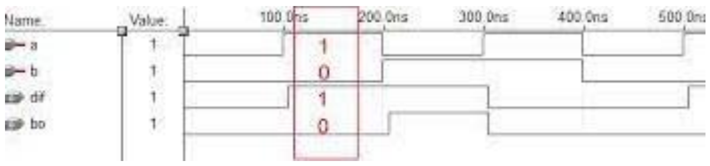
Circuit diagram



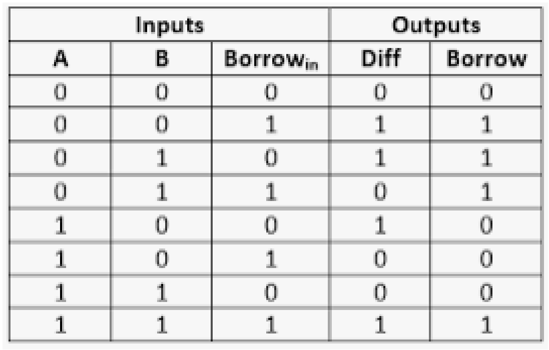
Verilog code

```
module hs(a,b,d,bout)
            input a,b;
            output d,bout;
            assign d=a^b;
            assign bout=!a&b;
endmodule
```

Output



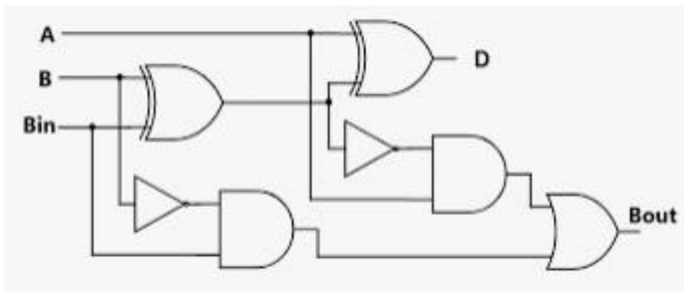
Design of full Subtractor



Inputs			Outputs	
A	B	Borrow_{in}	Diff	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

$$d = A \oplus B \oplus C$$
$$b = C(A \odot B) + \overline{A}B$$

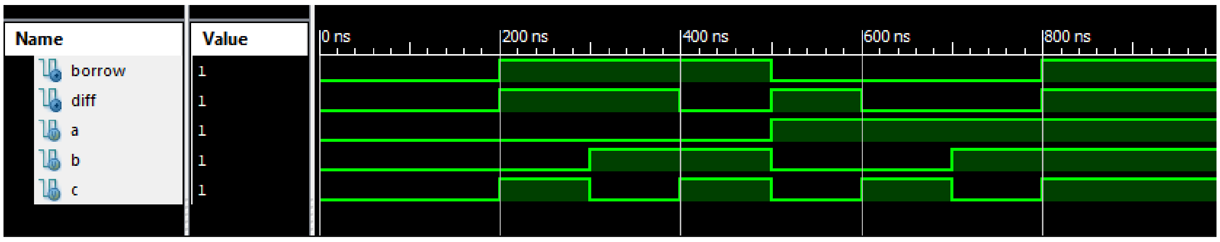
Circuit diagram



Verilog Code

```
module fs(a,b,bin,d,bout);
            input a,b,bin;
            output d,bout;
            assign d=a^b^bin;
            assign bout=(!a&(b^bin))|(b&bin);
endmodule
```

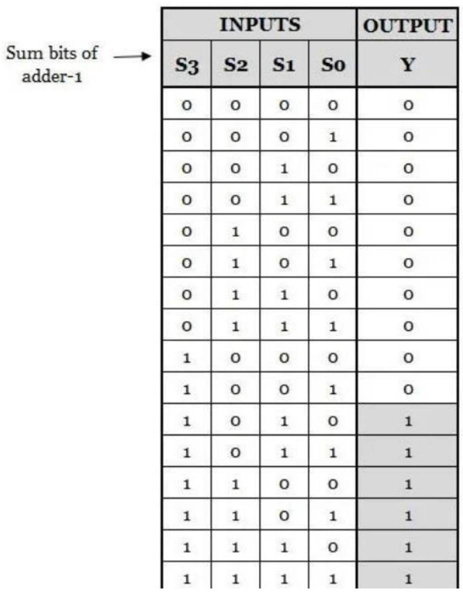
Output



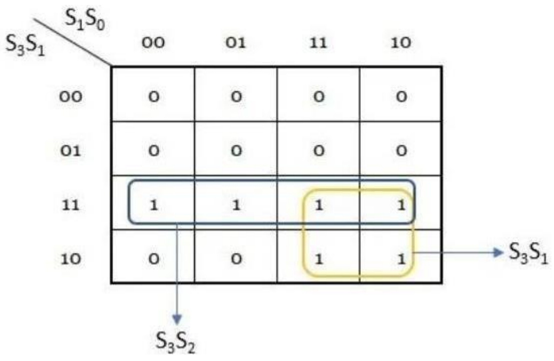
The truth table of half adder, full adder, half Subtractor, and full Subtractor are verified.

5. Design Verilog HDL to implement Decimal adder.

Design


Sum bits of adder-1

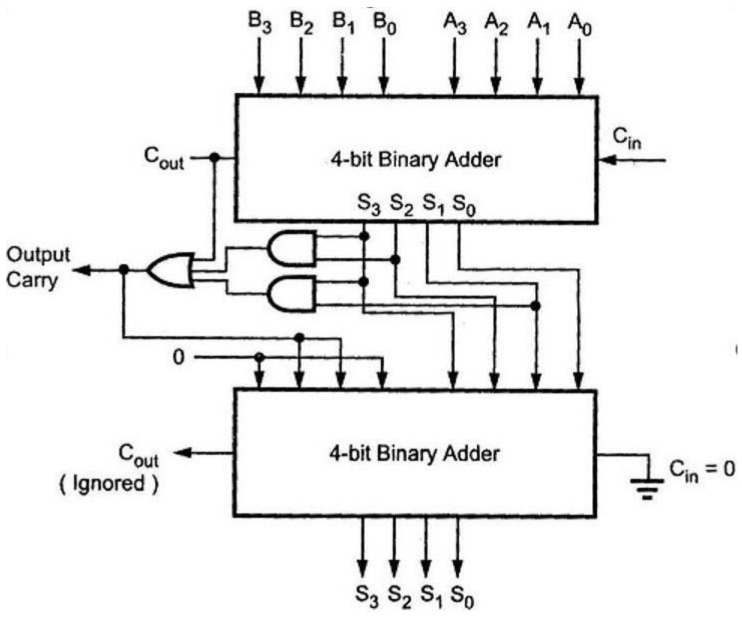
INPUTS				OUTPUT
S3	S2	S1	S0	Y
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1



S_3S_1 \ S_1S_0	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	1	1
10	0	0	1	1

$$\mathbf{Y = S_3S_2 + S_3S_1}$$

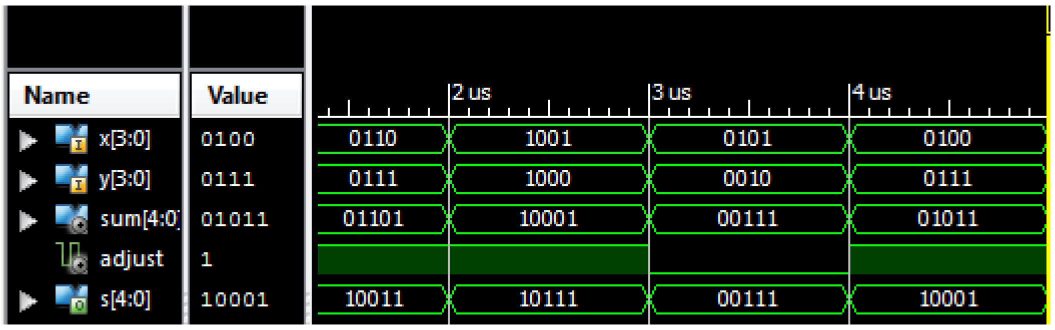
Circuit diagram



Verilog code

```
module bcd(a,b,carry_in,sum,carry);

//declare the inputs and outputs of the module with their sizes.
          input [3:0] a,b;

          input carry_in;
          output [3:0] sum;
          output carry;

//Internal variables

          reg [4:0] sum_temp;

          reg [3:0] sum; reg carry;

//always block for doing the addition
          always @(a,b,carry_in)
          begin

                    sum_temp = a+b+carry_in; //add all the inputs
                    if(sum_temp > 9)

                              begin

                                        sum_temp = sum_temp+6; //add 6, if result is more than 9.
                                        carry = 1; //set the carry output

                                        sum = sum_temp[3:0];
                              end
                    else
                              begin

                                        carry = 0;
                              end
          end
                                        sum = sum_temp[3:0];
endmodule
```

Output

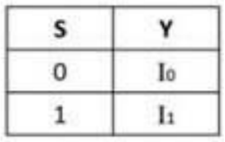


```
Name        Value   |       |2 us     |3 us     |4 us
x[3:0]      0100    0110    1001      0101      0100
y[3:0]      0111    0111    1000      0010      0111
sum[4:0]    01011   01101   10001     00111     01011
adjust      1
s[4:0]      10001   10011   10111     00111     10001
```

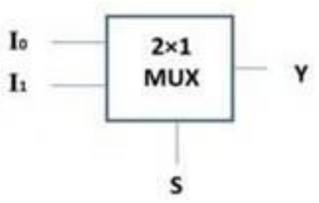
Decimal Adder is implemented using Verilog code and simulated and its functioning correctly for all possible values.

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

Design of 2:1 Multiplexer



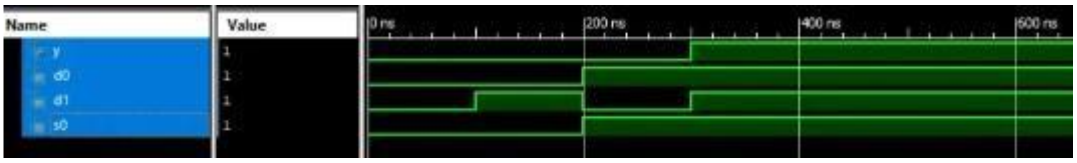
S	Y
0	I_0
1	I_1



Verilog code

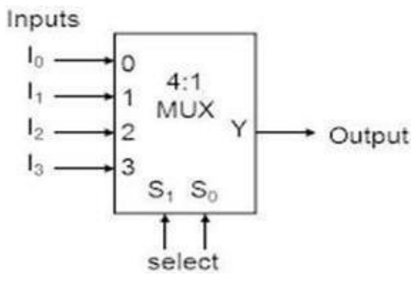
```
module mux2_1( I, sel, y);
        input [1:0] I;
        input sel;
        output y;
        reg y;
        always@ (sel , I)
        begin
                case (sel)
                        1'b0: y = I [0];
                        1'b1: y = I [1];
                endcase
        end
endmodule
```

Output



Design of 4:1 Multiplexer

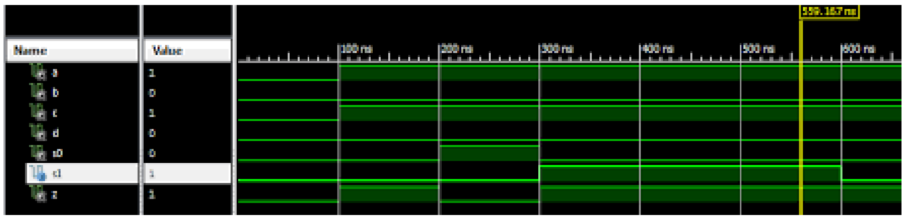
S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3



Verilog code

```
module mux_4_1 (I, sel, y);
        input [3:0] I;
        input [1:0] sel;
        output y;
        reg y;
        always@ (sel, I)
                begin
                        case (sel)
                                2'b00: y = I [0];
                                2'b01: y = I [1];
                                2'b10: y = I [2];
                                default: y = I [3];
                        endcase
                end
endmodule
```

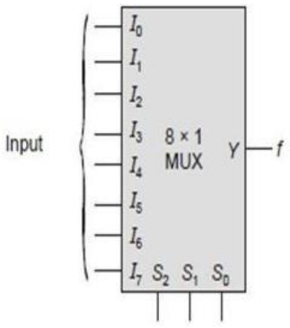
Output



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

Design of 8:1 Multiplexer

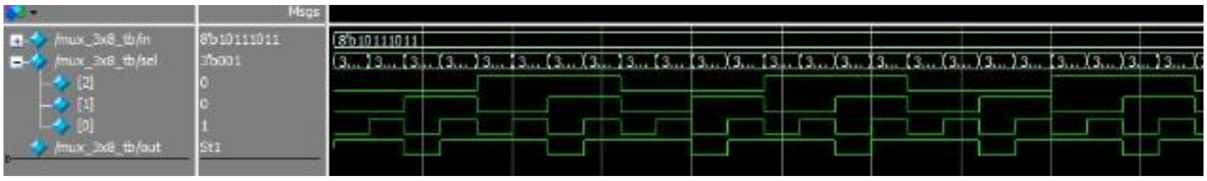
S_2	S_1	S_0	Y
0	0	0	I0
0	0	1	I1
0	1	0	I2
0	1	1	I3
1	0	0	I4
1	0	1	I5
1	1	0	I6
1	1	1	I7



Verilog code

```
module mux_8_1 (I, sel, y);
        input [7:0] I;
        input [2:0] sel;
        output y;
        reg y;
        always@ (sel, I )
                begin
                        case (sel)
                                3'b000: y = I [0];
                                3'b001: y = I [1];
                                3'b010: y = I [2];
                                3'b011: y = I [3];
                                3'b100: y = I [4];
                                3'b101: y = I [5];
                                3'b110: y = I [6];
                                default: y = I [7];
                        endcase
                end
endmodule
```

Output

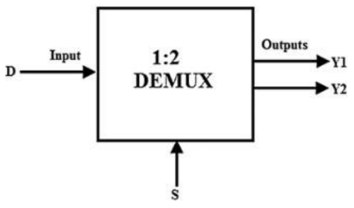


```
/mux_3x8_tb/in     8'b10111011
/mux_3x8_tb/sel    3'b001
    [2]            0
    [1]            0
    [0]            1
/mux_3x8_tb/out    St1
```

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

Design of 1:2 Demultiplexer

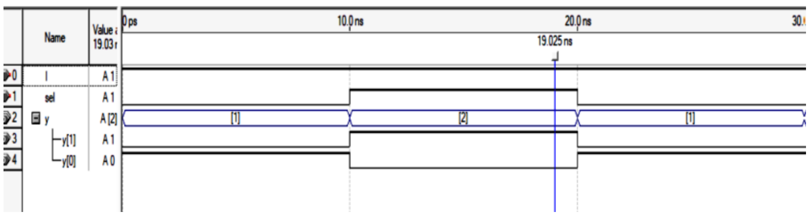
Select	Input	Outputs	
S	D	Y_2	Y_1
0	0	0	0
0	1	0	1
1	0	0	0
1	1	1	0



Verilog Code

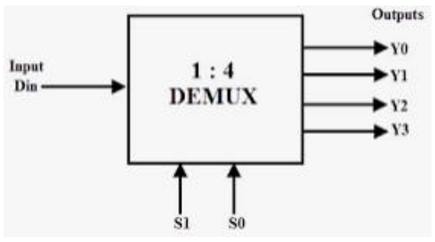
```
module de_mux_1_2 (I, sel, y);
          input I;
          input sel;
          output reg [1:0] y;
          always@ (sel, I)
                    begin
                              case (sel)
                                        1'b0: begin y [0] = I; y [1] = 1'b0; end
                                        1'b1: begin y [0] = 1'b0 ; y [1] = I; end
                              endcase

endmodule          end
```

Output



Design of 1:4 Demultiplexer

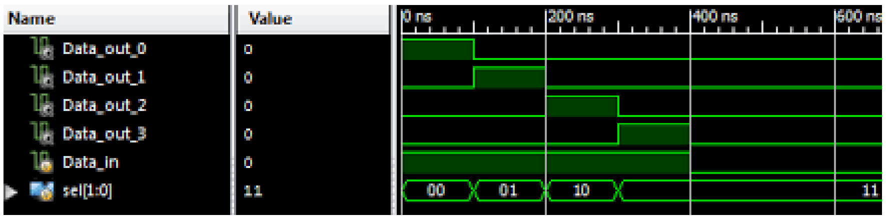
sel[0]	sel[1]	y_0	y_1	y_2	y_3
0	0	i	0	0	0
0	1	0	i	0	0
1	0	0	0	i	0
1	1	0	0	0	i



Verilog Code

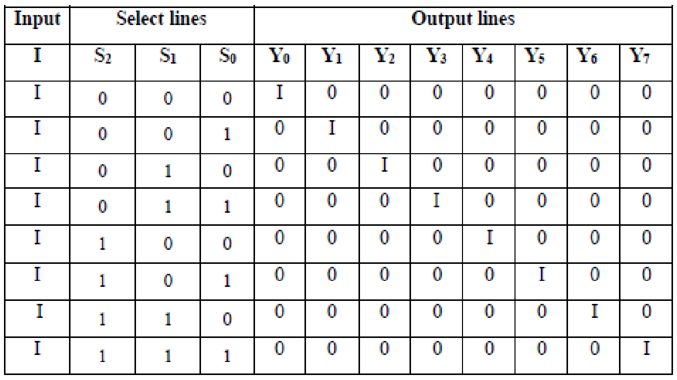
```
module de_mux_1_4 (I, sel, y);
        input I;
        input [1:0] sel;
        output reg [3:0] y;
        always@ (sel, I)
                begin case (sel)
                        2'b00: begin y [0] = I; y [1] = 0; y [2] = 0; y [3] = 0; end
                        2'b01: begin y [0] = 0; y [1] = I; y [2] = 0; y [3] = 0; end
                        2'b10: begin y [0] = 0; y [1] = 0; y [2] = I; y [3] = 0; end
                        2'b11: begin y [0] = 0; y [1] = 0; y [2] = 0; y [3] = I; end
                     endcase
                end
endmodule
```

Output

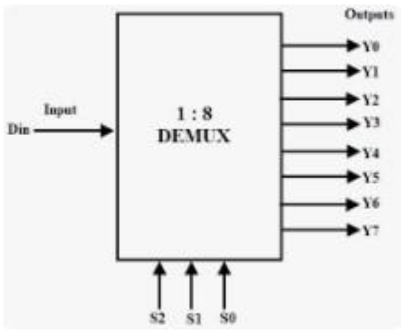


```
Name          Value
Data_out_0    0
Data_out_1    0
Data_out_2    0
Data_out_3    0
Data_in       0
sel[1:0]      11
```

Design of 1:8 Demultiplexer



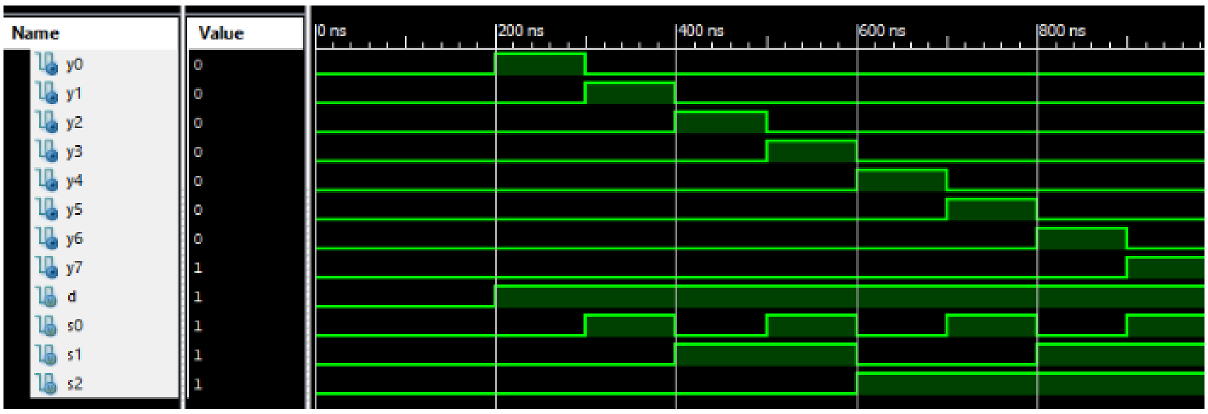
Input	Select lines			Output lines							
I	S_2	S_1	S_0	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7
I	0	0	0	I	0	0	0	0	0	0	0
I	0	0	1	0	I	0	0	0	0	0	0
I	0	1	0	0	0	I	0	0	0	0	0
I	0	1	1	0	0	0	I	0	0	0	0
I	1	0	0	0	0	0	0	I	0	0	0
I	1	0	1	0	0	0	0	0	I	0	0
I	1	1	0	0	0	0	0	0	0	I	0
I	1	1	1	0	0	0	0	0	0	0	I



Verilog code

```
module de_mux_1_8 (I, sel, y);
        input I;
        input [2:0] sel;
        output reg [7:0] y;
        always@(sel ,I)
                case (sel)
                 3'b000:begin y[0] = I; y[1] = 0 ;y[2] = 0;y[3] = 0 ;y[4] = 0;y[5] = 0;y[6] = 0;y[7] = 0;end
                 3'b001:begin y[0] = 0; y[1] = I; y[2] = 0;y[3] = 0;y[4] = 0;y[5] = 0;y[6] = 0;y[7] = 0;end
                 3'b010:begin y[0] = 0; y[1] = 0;y[2] = I; y[3] = 0;y[4] = 0; y[5] = 0;y[6] = 0;y[7] = 0;end
                 3'b011:begin y[0] = 0;y [1] = 0; y[2] = 0;y[3] = I ;y[4]= 0;y[5] = 0 ;y[6] = 0;y[7] = 0;end
                 3'b100:begin y[0] = 0; y[1] = 0;y[2] = 0;y[3] = 0;y[4] = I; y[5] = 0;y[6] = 0;y[7] = 0; end
                 3'b101:begin y[0] = 0; y[1] = 0;y[2] = 0;y[3] = 0;y[4] = 0;y[5] = I; y[6] = 0;y[7] = 0; end
                 3'b110:begin y[0] = 0; y[1] = 0;y[2] = 0;y[3] = 0;y[4] = 0;y[5] = 0;y[6] = I; y[7] = 0; end
                 3'b111:begin y[0] = 0; y[1] = 0;y[2] = 0;y[3] = 0;y[4] = 0;y[5] = 0;y[6] = 0;y[7] = I; end
                 default: y = 8'bxxxxxxxx;
                endcase
endmodule
```

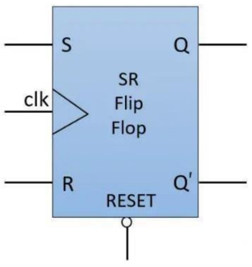
Output



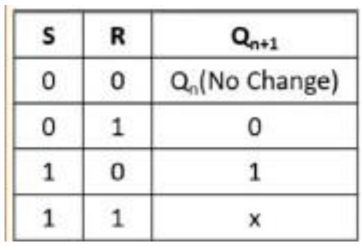
The truth table of different types of de-mux 2:1, 4:1 and 8:1 are verified and simulated.

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

Design of SR Flip Flop



SR Flip Flop

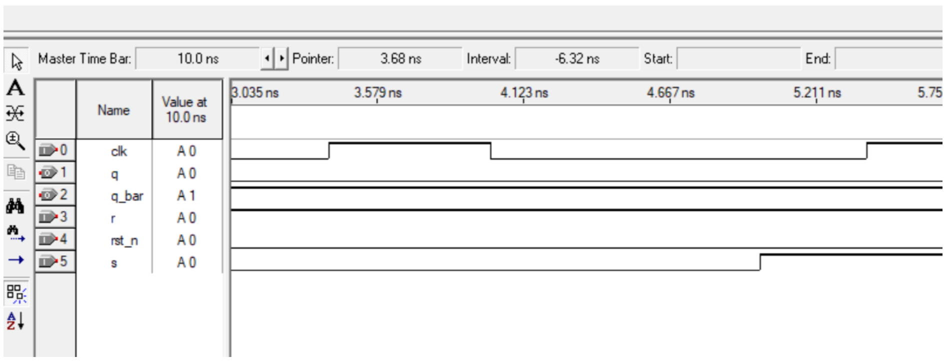


S	R	Q_{n+1}
0	0	Q_n(No Change)
0	1	0
1	0	1
1	1	x

Verilog Code

```
module sr(input clk, rst_n, input s,r, output reg q, output q_bar);
//always@(posedge clk or negedge rst_n) // for asynchronous reset
always@(posedge clk) begin // for synchronous reset
        if(!rst_n) q <= 0;
        else begin
                case({s,r})
                  2'b00: q <= q; // No change
                  2'b01: q <= 1'b0; // reset
                  2'b10: q <= 1'b1; // set
                  2'b11: q <= 1'bx; // Invalid inputs
                endcase
        end
        end
assign q_bar = ~q;
endmodule
```

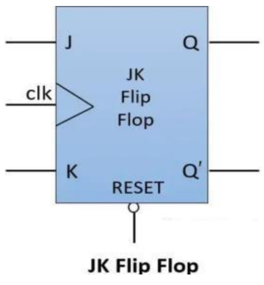
Output



Master Time Bar: 10.0 ns Pointer: 3.68 ns Interval: -6.32 ns Start: End:

	Name	Value at 10.0 ns
0	clk	A 0
1	q	A 0
2	q_bar	A 1
3	r	A 0
4	rst_n	A 0
5	s	A 0

Design of JK Flip Flop

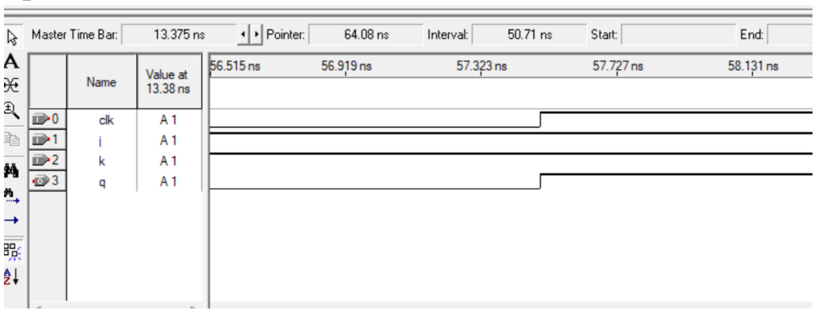


J	K	Q_{n+1}
0	0	Q_n(No Change)
0	1	0
1	0	1
1	1	$\overline{Q_n}$(Toggles)

Verilog code

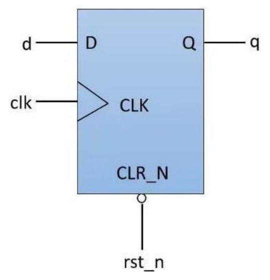
```
module jk( input j, input k, input clk, output reg q);
always @ (posedge clk)

          case ({j,k})

                    2'b00 : q <= q;

                    2'b01 : q <= 0;

                    2'b10 : q <= 1;

                    2'b11 : q <= ~q;

                    endcase

endmodule
```

Output



```
Master Time Bar: 13.375 ns   Pointer: 64.08 ns   Interval: 50.71 ns   Start:   End:

      Name   Value at    56.515 ns   56.919 ns   57.323 ns   57.727 ns   58.131 ns
             13.38 ns
  0   clk    A 1
  1   j      A 1
  2   k      A 1
  3   q      A 1
```

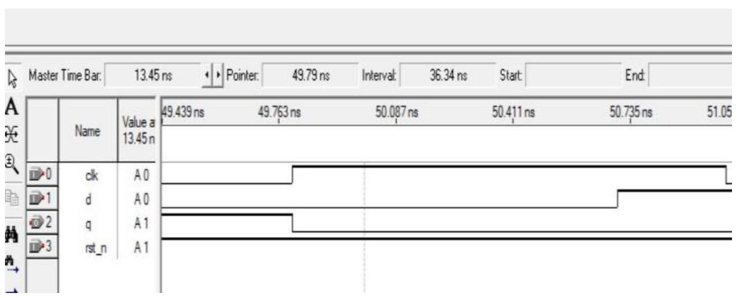
Design of D- Flipflop



Verilog code:

```
module dd (

input clk, rst_n, input d,
output reg q

);

always@(posedge clk or negedge rst_n)

begin

        if(!rst_n) q <= 0;

        else q <= d;

        end

endmodule
```

Output



```
Master Time Bar: 13.45 ns   Pointer: 49.79 ns   Interval: 36.34 ns   Start:   End:

     Name    Value a 13.45 n   49.439 ns  49.763 ns  50.087 ns  50.411 ns  50.735 ns  51.05
0    clk     A 0
1    d       A 0
2    q       A 1
3    rst_n   A 1
```