TONTADARYA COLLEGE OF ENGINEERING

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

Mundaragi Road, GADAG – 582101

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Digital Design & Computer Organization

LABORATORY COMPONENT

(Integrated with BCS302)

2023-24(2022 SCHEME)

FORMAT NO: QF-TCE-Lab-07

Prepared by

Mr. Adokshaja Kulkarni M.Tech

Asst. Professor

INSTITUTE VISION

To develop technically competent engineering graduates with ethical values

INSTITUTE MISSION

- To impart technical education through quality academics
- To promote career opportunities in collaboration with industries
- To encourage creativity, innovation and research
- To inculcate ethical values and social responsibilities

DEPARTMENT VISION

To accomplish recognition in computer science and engineering by producing technical professionals with ethical values

DEPARTMENT MISSION

We at the computer science and engineering are committed to provide quality education and state-of-art facility for the students in order to:

- Impart the programming skills and knowledge of computer science and engineering.
- Encourage in involving research, innovation and higher studies with ethical values.

Syllabus

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

PRACTICAL COMPONENT OF IPCC

1. Given a 4-variable logic expression, simplify it using appropriate technique and simulate the same using basic gates.
2. Design a 4 bit full adder and subtractor and simulate the same using basic gates.
3. Design Verilog HDL to implement simple circuits using structural, Data flow and 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.

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

COMPONENTS REQUIRED:

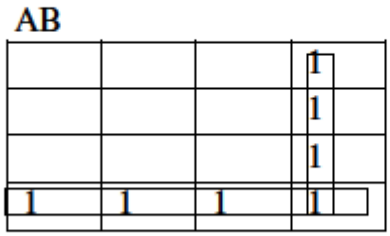
IC 7400, IC 7408, IC 7432, IC 7404, IC 7402, Connecting wires

THEORY:

Canonical Forms (Normal Forms): Any Boolean function can be written in disjunctivenormal form (sum of min-terms) or conjunctive normal form (product of maxterms). A Boolean function can be represented by a Karnaugh map in which each cell corresponds to a minterm. The cells are arranged in such a way that any two immediately adjacent cells correspond to two minterms of distance 1. There is more than one way to construct a map with this property.

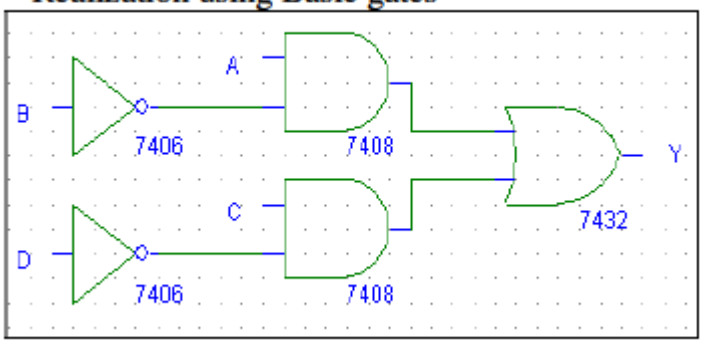
Realization of Boolean expression:

1) $Y = \bar{A}\bar{B}C\bar{D} + \bar{A}BC\bar{D} + ABC\bar{D} + A\bar{B}C\bar{D} + A\bar{B}\bar{C}\bar{D} + A\bar{B}\bar{C}D + A\bar{B}CD$



After simplifying using K-Map method we get $Y = A\bar{B} + C\bar{D}$

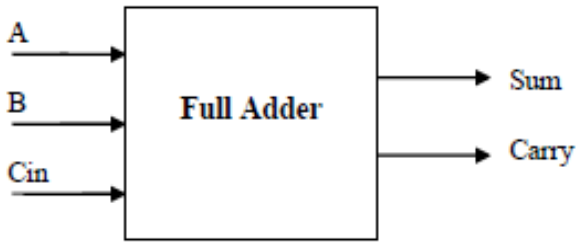
Realization using Basic gates



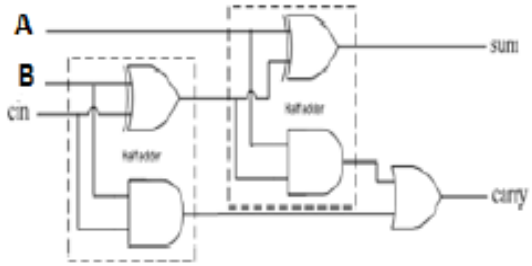
TRUTH TABLE

INPUTS				OUTPUT
A	B	C	D	Y
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	0

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



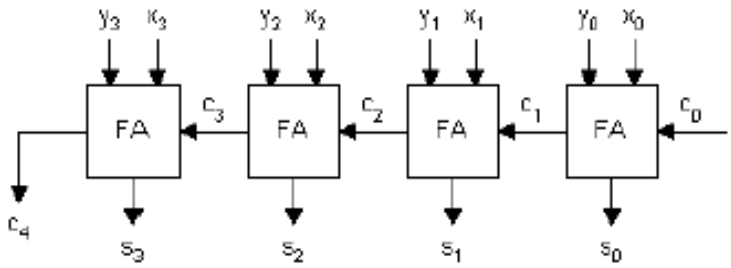
Full Adder using two Half Adders (for structural modeling)



Truth Table 1-bit Full adder

A	B	Cin	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

4-Bit Parallel Adder



Code: 1-bit Full Adder (Dataflow Modeling)

```
module fulladder(A,B,cin,Sum,Carry);
   input A;
   input B;
   input cin;
   output Sum;
   output Carry;
   assign Sum = A^B^cin;
   assign Carry = (A&B)|(A&cin)|(B&cin);
endmodule
```

Code: Half Adder, 1-bit Full Adder and 4-bit Ripple Carry Adder

For Half Adder

```
module half_adder( output Sum,Carry,
                   input A,B );
  xor(Sum,A,B);
  and(Carry,A,B);
endmodule
```

For Full Adder

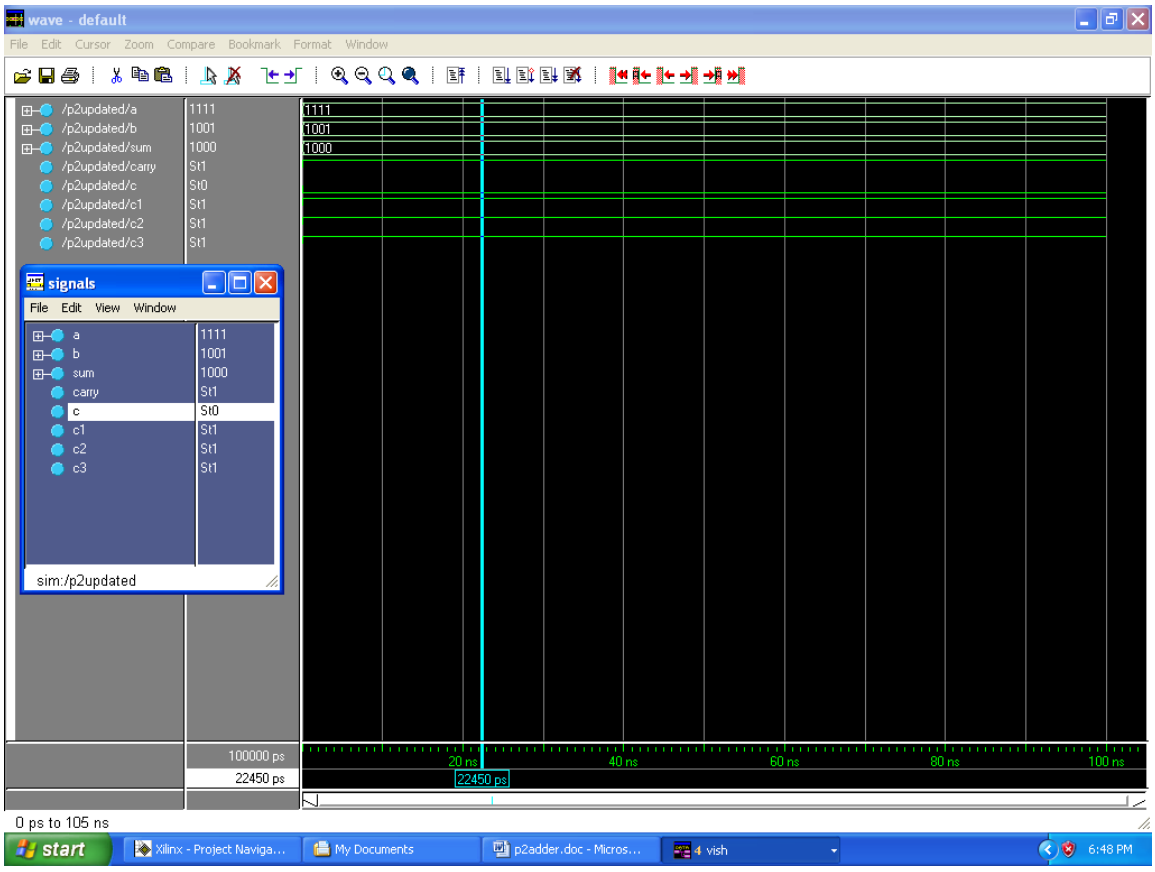
```
module full_adder( output Sum,Cout,
                   input A,B,Cin );
  wire s1,c1,c2;
  half_adder  ha1(s1,c1,A,B);
  half_adder  ha2(Sum,c2,s1,Cin);
  or or1(Cout,c1,c2);
endmodule
```

For 4-bit Ripple Carry Adder

```
module ripple_adder_4bit(output [3:0] Sum,
                         output Cout,
                         input [3:0] A,B,
                         input Cin );
 wire c1,c2,c3;
 full_adder FA1(Sum[0],c1,A[0],B[0],Cin),
            FA2(Sum[1],c2,A[1],B[1],c1),
            FA3(Sum[2],c3,A[2],B[2],c2),
            FA4(Sum[3],Cout,A[3],B[3],c3);

endmodule
```

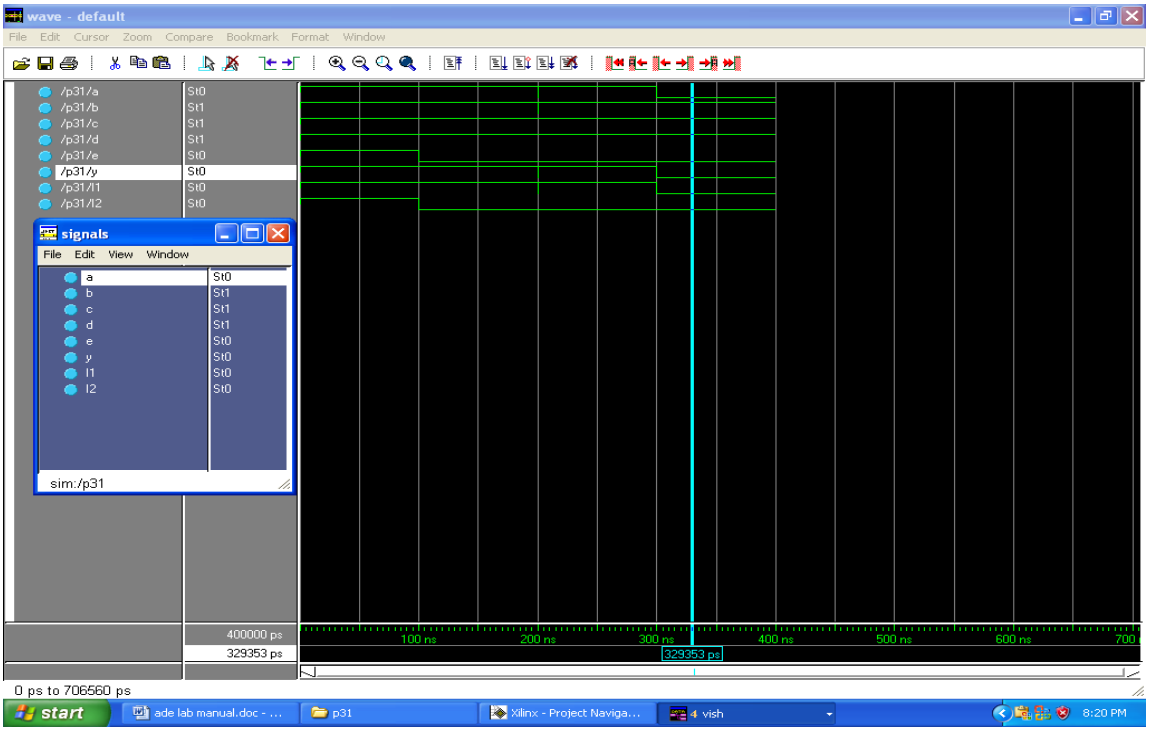
Sample Output



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

Structural Model

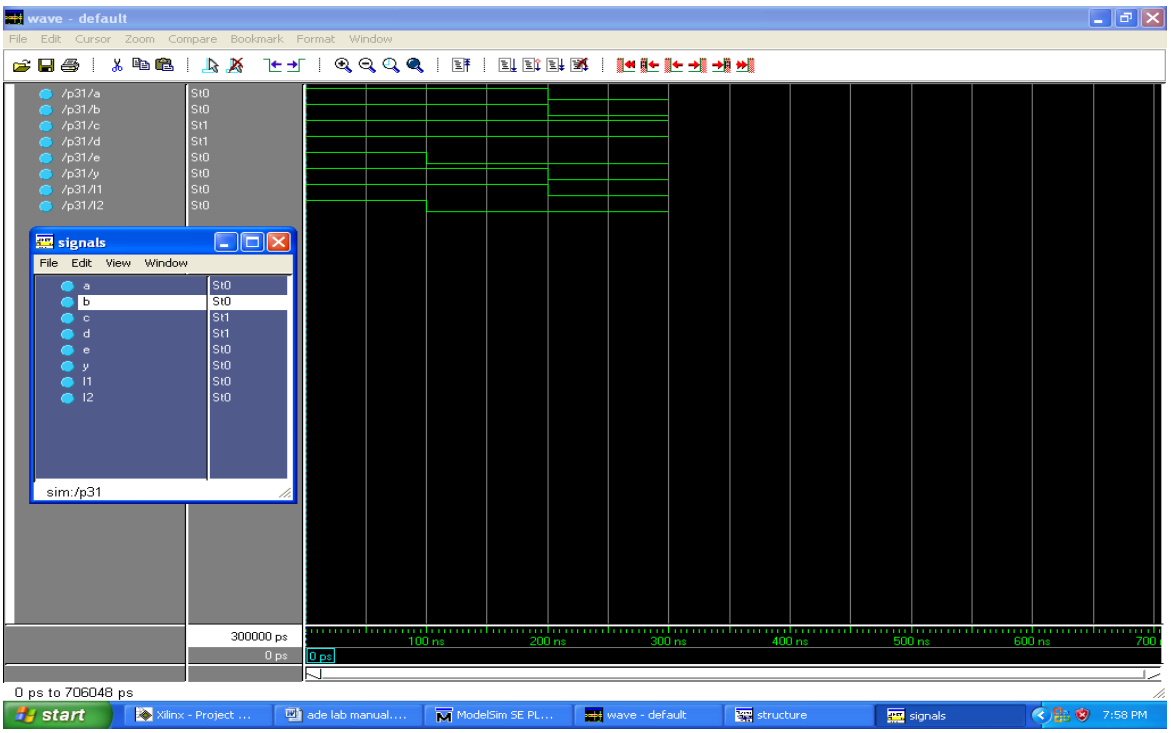
```
module p3structural(a,b,c,d,e,y);
    input a;
    input b;
    input c;
    input d;
    input e;
    output  y;
    wire Y1,Y2;
    and G1(Y1,a,b);
    and G2(Y2,c,d,e);
    or G3(Y,Y1,Y2);
    endmodule
```



Data Flow Model

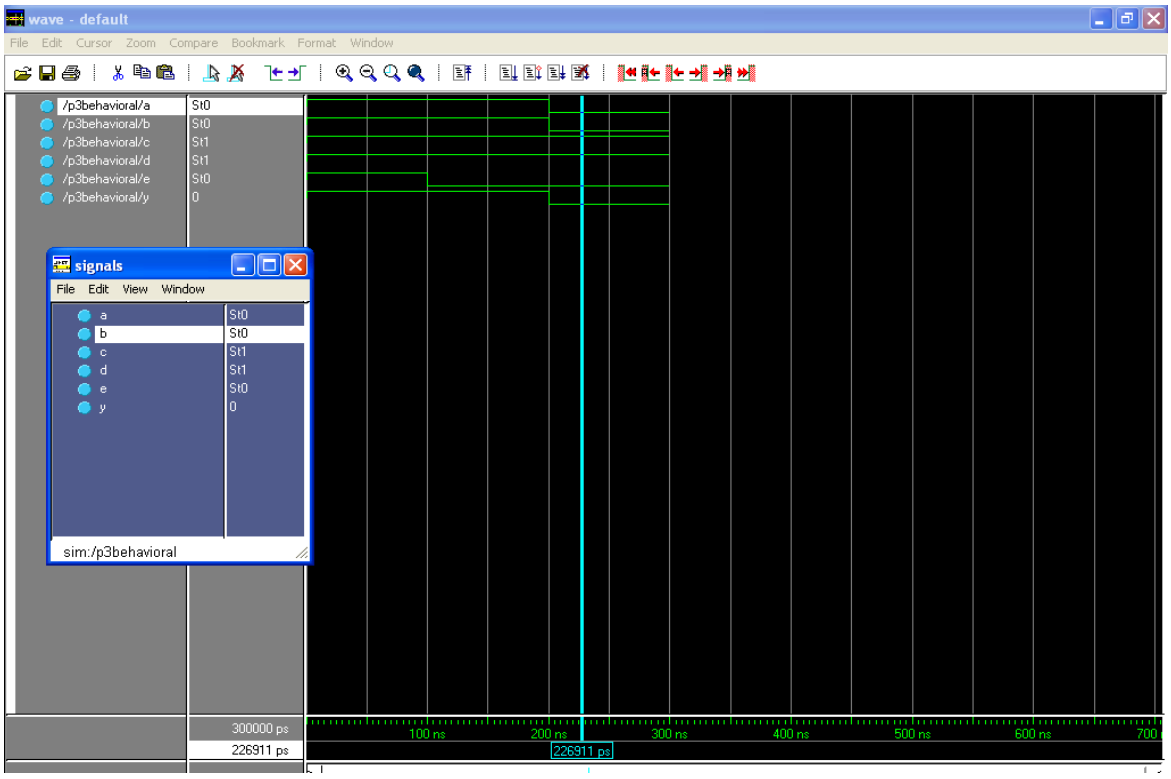
```
module p31(a,b,c,d,e,y);

  input a;
  input b;
  input c;
  input d;
  input e;
  output y;
  wireY1,Y2;
  assign Y1=a & b;
  assign Y2= c&d&e;
  assign y= Y1|Y2;
  endmodule
```



Behavioral Model

```
module p3behavioral(a,b,c,d,e,y);
    input a;
    input b;
     input c;
    input d;
    input e;
    output y;
    reg y;
    always @(a,b,c,d,e)
    begin
    y= (a & b) | (c & d &e);
    end
    endmodule
```



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

Half Adder

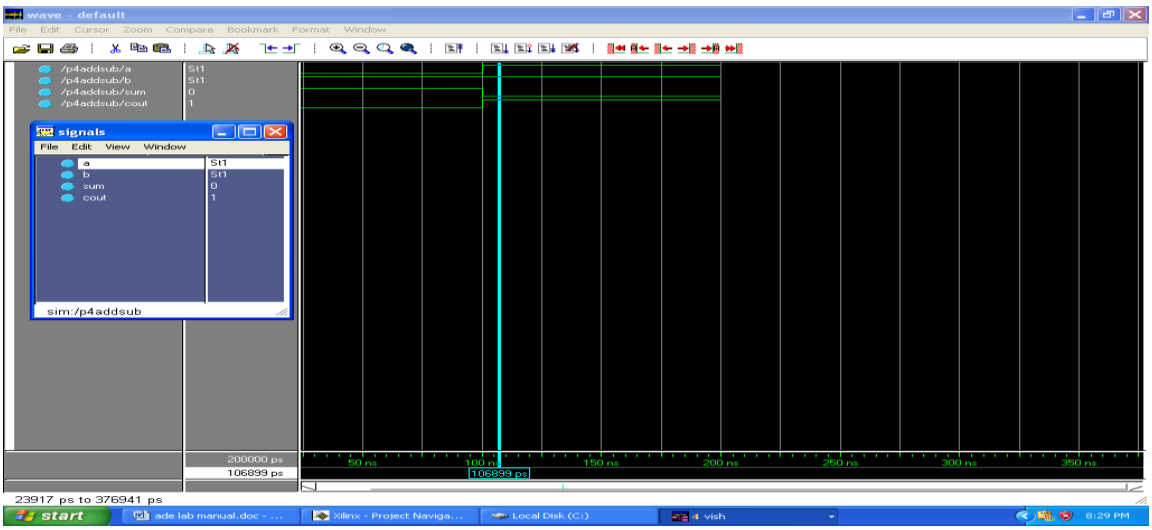
BOOLEAN EXPRESSIONS:

sum=A ⊕ B
cout=AB

TRUTH TABLE

INPUTS		OUTPUTS	
A	B	sum	cout
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

```
module p4addsub(a,b,sum,cout);
        input a;
        input b;
        output sum;
        output cout;
        reg sum, cout;
         always @(a,b)
        begin
        sum = a ^ b;
        cout = a & b;
        end
        endmodule
```



HALF SUBTRACTOR

BOOLEAN EXPRESSIONS:

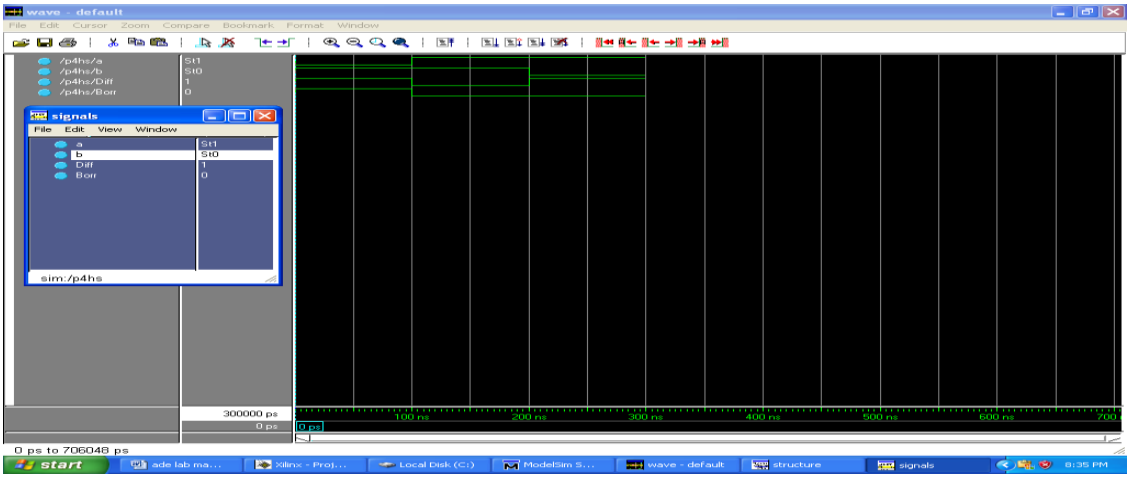
Diff = A ⊕ B

Borr = $\bar{A}$B

TRUTH TABLE

INPUTS		OUTPUTS	
A	B	Diff	Borr
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

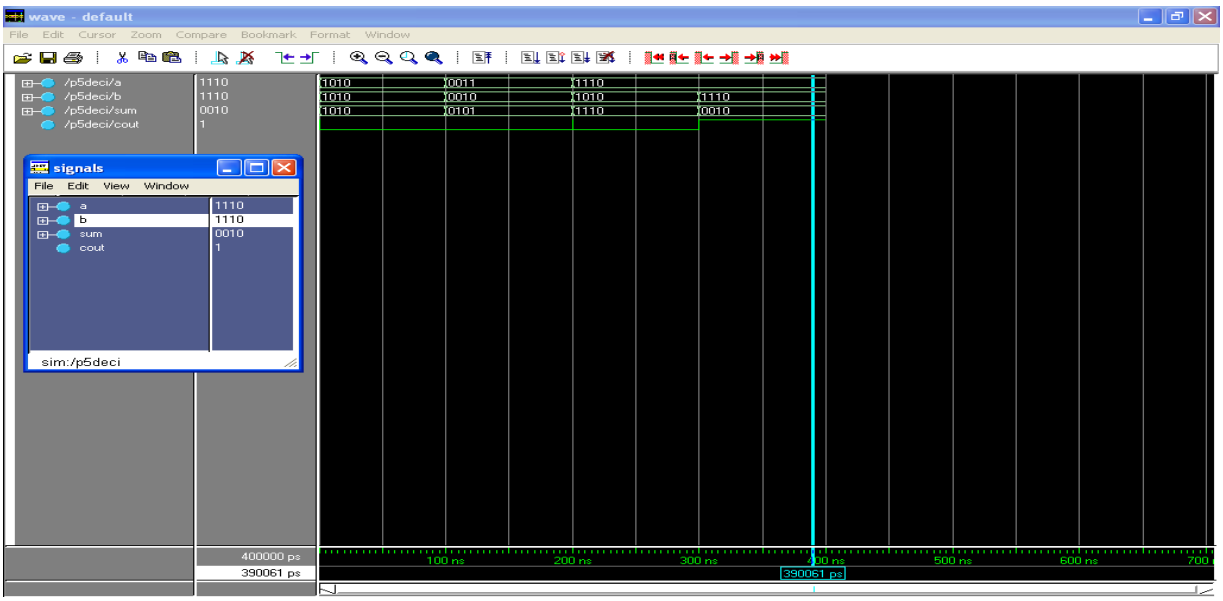
```
module p4hs(a,b,Diff,Borr);
        input a;
        input b;
        output Diff;
        output Borr;
        reg Diff, Borr;
        always @(a,b)
        begin
        Diff = a ^ b;
        Borr = (~ a) & b;
        end
        endmodule
```



5.Design Verilog HDL to implement Decimal adder.

This Verilog module, "DecimalAdder," takes two 4-bit decimal inputs A and B and produces a 4-bit sum (Sum) and a carry-out (CarryOut) output. The logic inside the "always" block performs decimal addition with carry propagation, and it also handles the case when the result is greater than 9. In such cases, it adds 6 to the result and updates the carry accordingly.

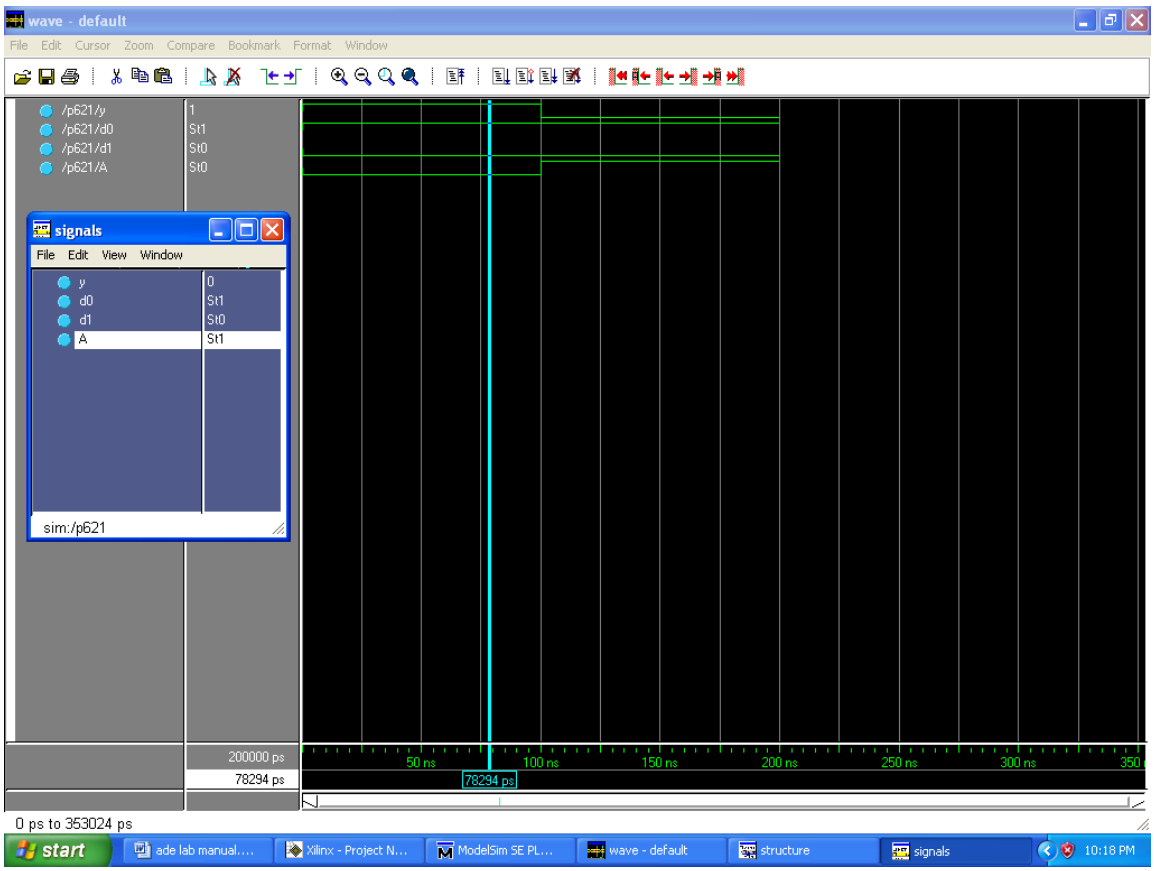
```
module p5deci(a,b,sum,cout);
    input [3:0] a;
    input [3:0] b;
    output [3:0] sum;
    output cout;
    reg [3:0] sum;
    reg cout;
    always@ (a,b)
    begin
    {cout,sum} = a+b;
    if(a>9 || b>9 || sum>9)
    begin
    {cout,sum} = sum+6;
    end
    end
    endmodule
```



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

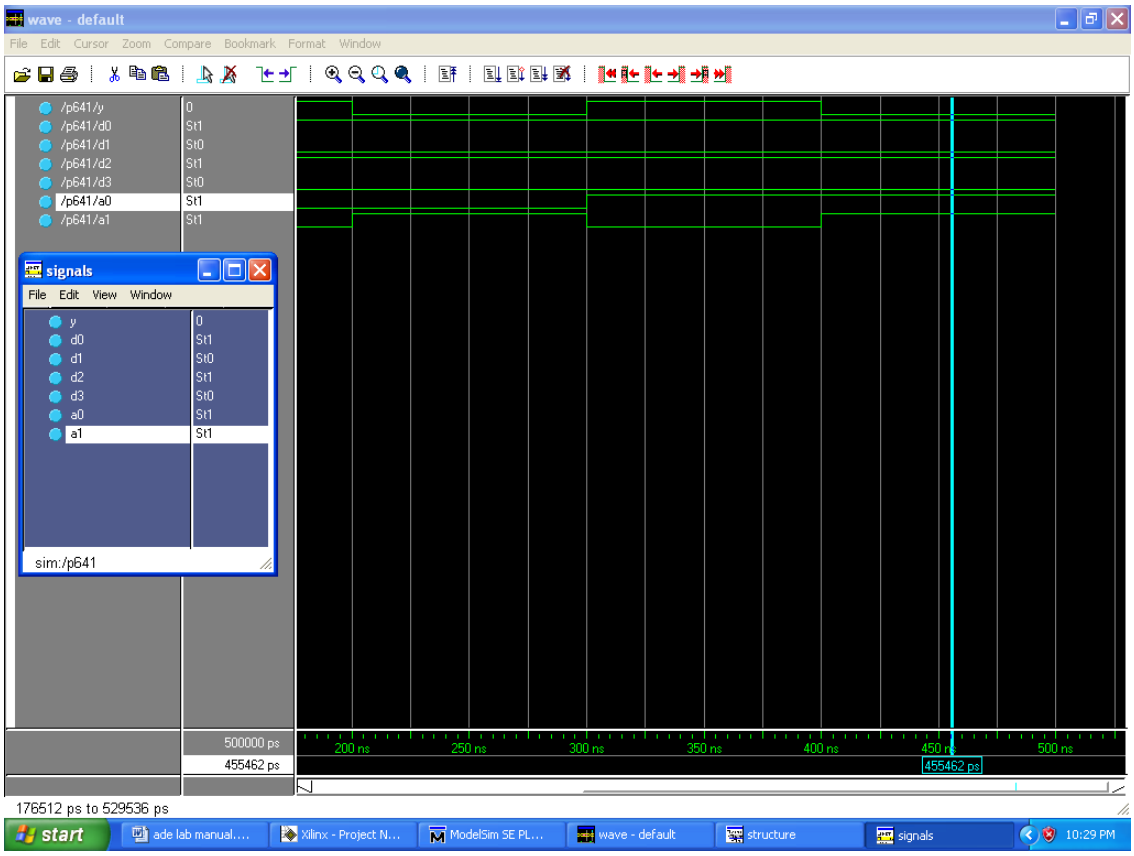
2:1 MUX

```
module p621(y,d0,d1,A);
 output y;
input d0;
 input d1; i
nput A;
reg y;
always @ (d0,d1,A)
begin
y=((~A & d0)|(A & d1));
end
endmodule
```



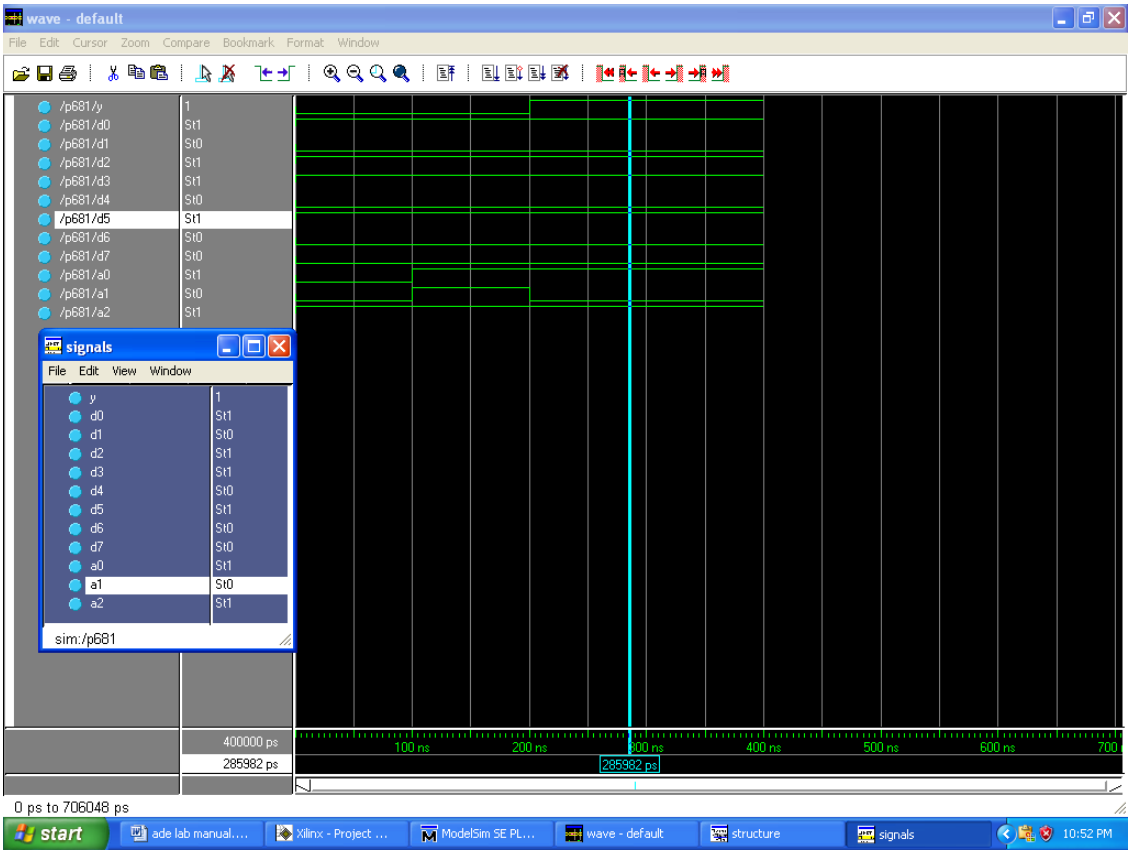
4:1 MUX

```
module p641(y,d0,d1,d2,d3,a0,a1);
 output y;
 input d0;
  input d1;
  input d2;
 input d3;
  input a0;
 input a1;
 reg y;
 always @ (d0,d1,d2,d3,a0,a1)
  begin
 y= (~a0 & ~a1 & d0) | (~a0 & a1 & d1) | (a0 & ~a1 & d2) | (a0 & a1 & d3);
 end
 endmodule
```



8:1 MUX

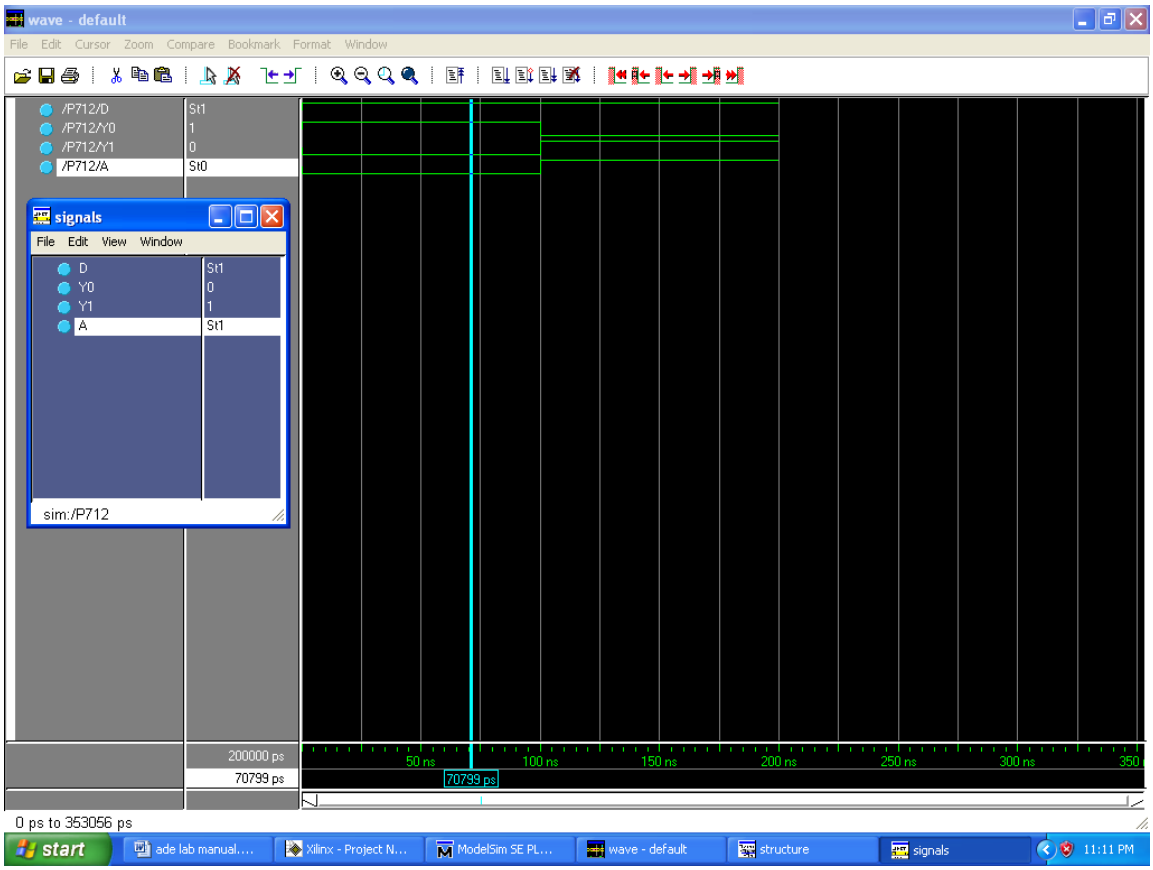
```
module p681(y,d0,d1,d2,d3,d4,d5,d6,d7,a0,a1,a2);
    output y;
    input d0;
    input d1;
     input d2;
     input d3;
     input d4;
    input d5;
    input d6;
     input d7;
    input a0;
    input a1;
     input a2;
    reg y;
    always @ (d0,d1,d2,d3,d4,d5,d6,d7,a0,a1,a2)
    begin
    y= (~a0 & ~a1 & ~a2 & d0 ) |
    (~a0 & ~a1 & a2 & d1) | (~a0 & a1 & a2 & d2)|( ~a0 & a1 & a2 & d3)|
    (a0 & ~a1 & ~a2 & d4) |
     (a0 & ~a1 & a2 & d5)| (a0 & a1 & ~a2 & d6)| (a0 & a1 & a2 & d7) ;
    end
    endmodule
```


wave - default
File Edit Cursor Zoom Compare Bookmark Format Window
/p681/y
/p681/d0
/p681/d1
/p681/d2
/p681/d3
/p681/d4
/p681/d5
/p681/d6
/p681/d7
/p681/a0
/p681/a1
/p681/a2
signals
File Edit View Window
y
d0
d1
d2
d3
d4
d5
d6
d7
a0
a1
a2
sim:/p681
400000 ps
285982 ps
100 ns
200 ns
300 ns
400 ns
500 ns
600 ns
285982 ps
0 ps to 706048 ps
start
ade lab manual....
Xilinx - Project ...
ModelSim SE PL...
wave - default
structure
signals
10:52 PM

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

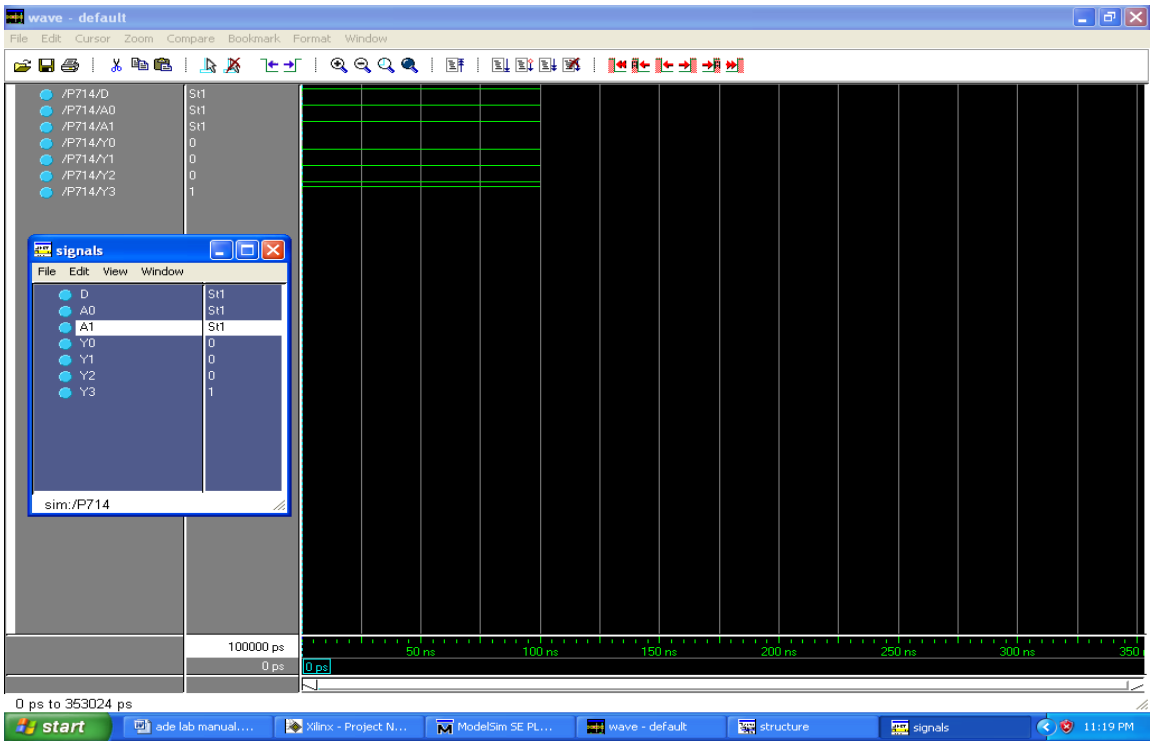
1:2 DEMUX

```
module P712(D,Y0,Y1,A);
        input D;
        output Y0;
        output Y1;
        input A;
        reg Y0,Y1;
        always @ (A,D)
        begin
        Y0=(~A & D); Y1=(A & D);
        end
        endmodule
```



1:4 DEMUX

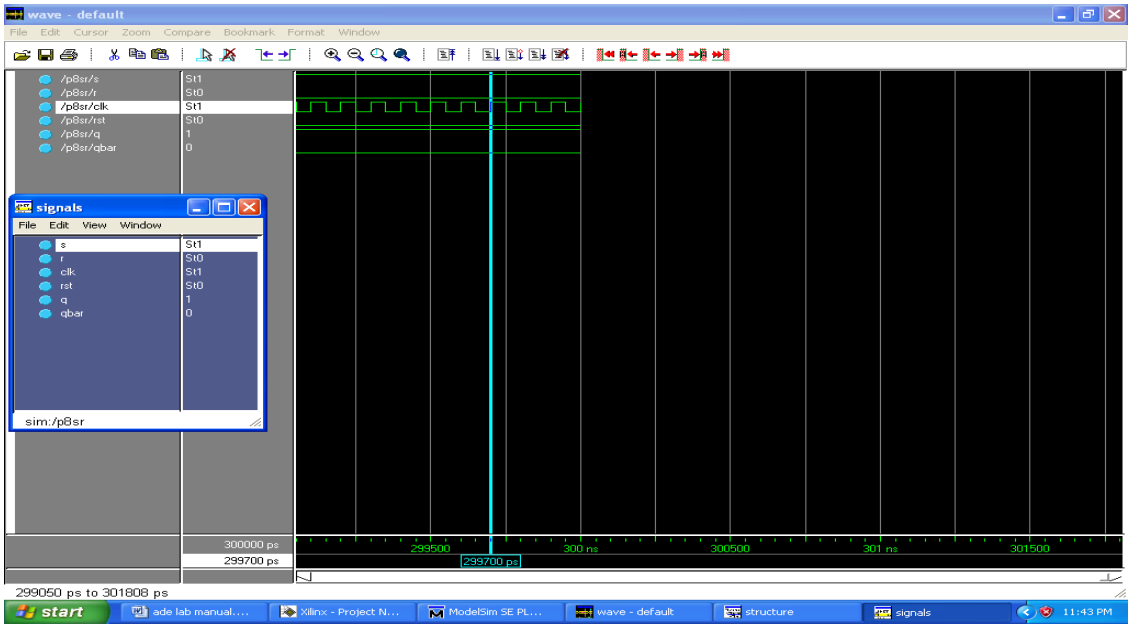
```
module P714(D,A0,A1,Y0,Y1,Y2,Y3);
    input D;
    input A0;
    input A1;
    output Y0;
    output Y1;
    output Y2;
    output Y3;
    reg Y0,Y1,Y2,Y3;
    always @ (A0,A1,D)
     begin
    Y0=(~A0 & ~A1 & D);
    Y1=(~A0 & A1 & D);
    Y2=(A0 & ~A1 & D);
    Y3=(A0 & A1 & D);
    end
    endmodule
```



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

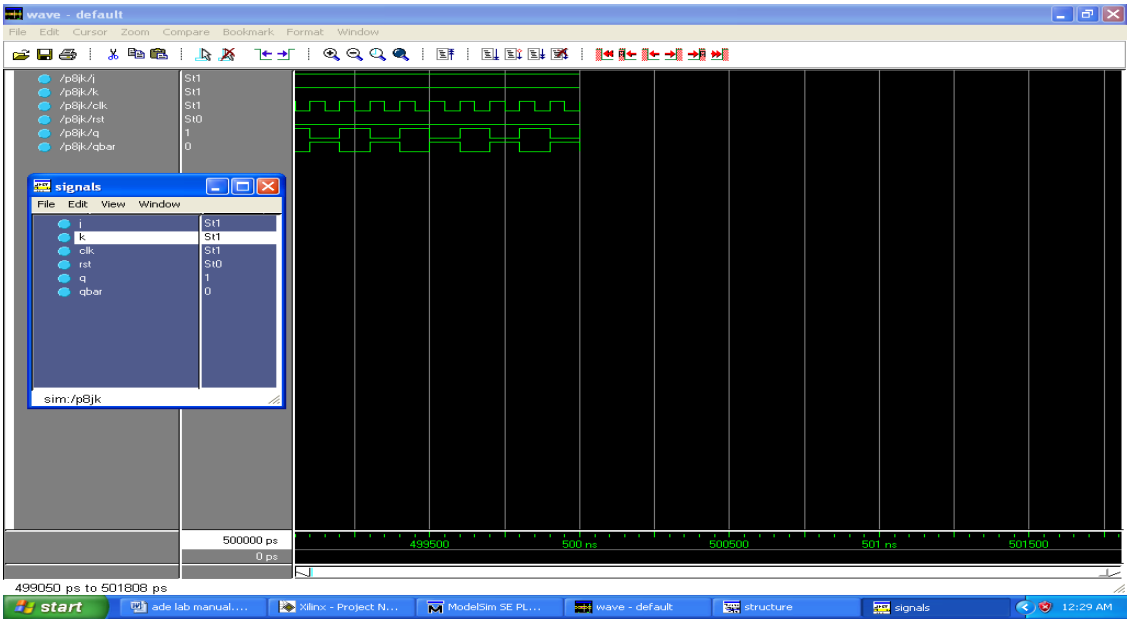
SR FLIP FLOP

```
module p8sr(s,r,clk,rst,q,qbar);
    input s;
    input r;
     input clk;
     input rst;
    output q;
    output qbar;
    reg q,qbar;
    always@(posedge clk)
    begin
    if(rst)
    q<=1'b0;
    else if (s==1'b0 && r==1'b0) q<=q;
    else if (s==1'b0 && r==1'b1) q<=1'b0;
    else if (s==1'b1 && r==1'b0) q<=1'b1;
    else if (s==1'b1 && r==1'b1) q<=1'bx;
    assign qbar=~q;
    end
    endmodule
```



JK FLIP FLOP

```
module p8jk(j,k,clk,rst,q,qbar);
        input j;
        input k;
        input clk;
        input rst;
        output q;
        output qbar;
        reg q,qbar;
        always@(posedge clk)
        begin
        if(rst) q<=1'b0;
        else if (j==1'b0 && k==1'b0) q<=q;
        else if (j==1'b0 && k==1'b1) q<=1'b0;
        else if (j==1'b1 && k==1'b0) q<=1'b1;
        else if (j==1'b1 && k==1'b1) q<= ~q;
        assign qbar = ~q;
        end
        endmodule
```



D FLIP FLOP

```
module p8dff(d,clk,q);
    input d;
    input clk;
    output q;
    reg q;
    always @(posedge clk)
    begin
    q<=d;
    end
    endmodule
```