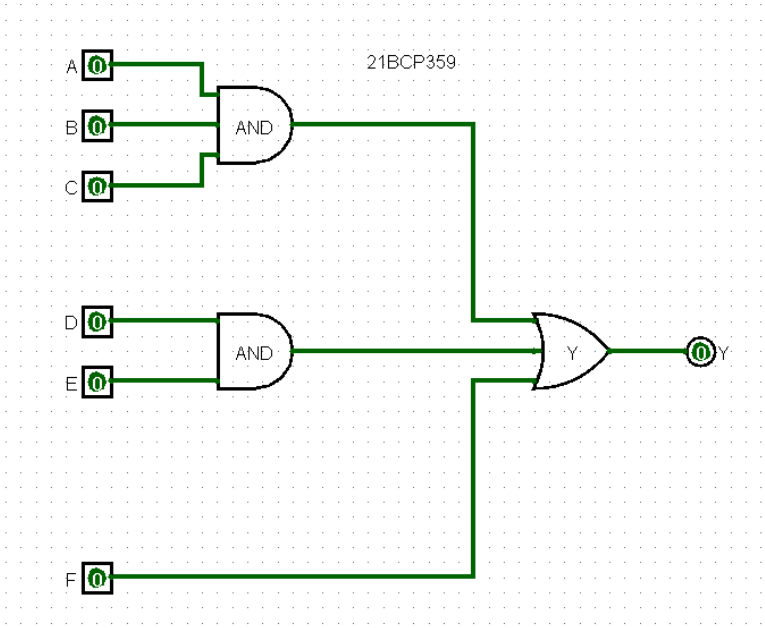
Lab 1: Basic Gate Operations

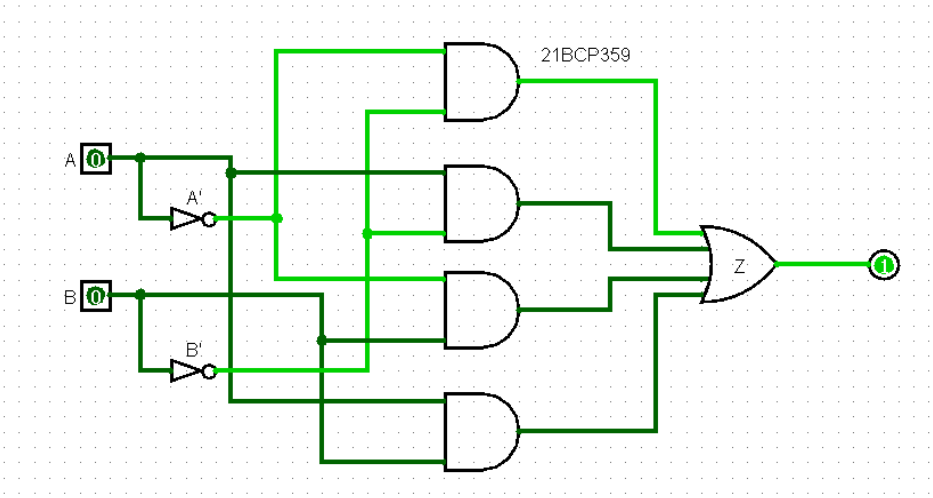
Question 1: Design the following logic functions using basic gates and test the output using truth table

a. Y = ABC + DE + F



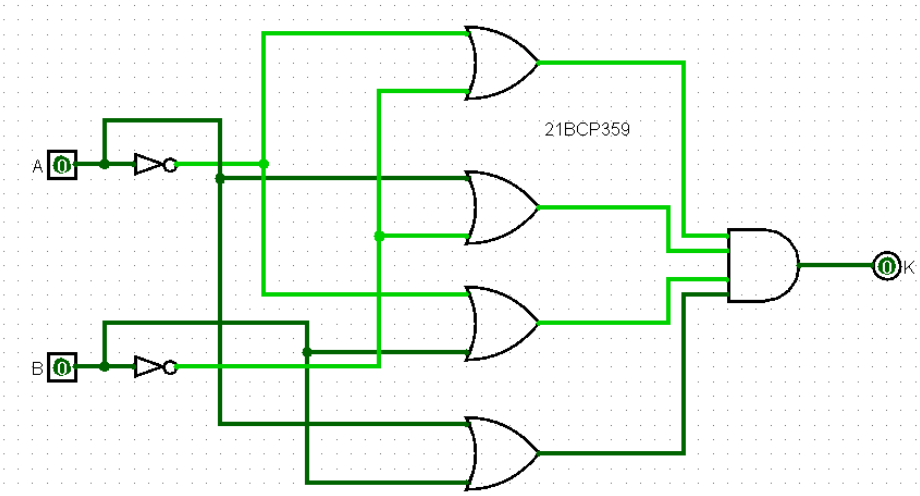
A	B	C	D	E	F	Y
0	0	0	0	0	0	0
0	0	0	0	0	1	1
0	0	0	0	1	0	0
0	0	0	0	1	1	1
0	0	0	1	0	0	0
0	0	0	1	0	1	1
0	0	0	1	1	0	1
0	0	0	1	1	1	1
0	0	1	0	0	0	0
0	0	1	0	0	1	1
0	0	1	0	1	0	0
0	0	1	0	1	1	1
0	0	1	1	0	0	0
0	0	1	1	0	1	1
0	0	1	1	1	0	1
0	0	1	1	1	1	1
0	1	0	0	0	0	0
0	1	0	0	0	1	1
0	1	0	0	1	0	0
0	1	0	0	1	1	1
0	1	0	1	0	0	0
0	1	0	1	0	1	1
0	1	0	1	1	0	1
0	1	0	1	1	1	1
0	1	1	0	0	0	0
0	1	1	0	0	1	1
0	1	1	0	1	0	0
0	1	1	0	1	1	1
0	1	1	1	0	0	0
0	1	1	1	0	1	1
0	1	1	1	1	0	1
0	1	1	1	1	1	1
1	0	0	0	0	0	0
1	0	0	0	0	1	1
1	0	0	0	1	0	0
1	0	0	0	1	1	1
1	0	0	1	0	0	0
1	0	0	1	0	1	1
1	0	0	1	1	0	1
1	0	0	1	1	1	1
1	0	1	0	0	0	0
1	0	1	0	0	1	1
1	0	1	0	1	0	0
1	0	1	0	1	1	1
1	0	1	1	0	0	0
1	0	1	1	0	1	1
1	0	1	1	1	0	1
1	0	1	1	1	1	1
1	1	0	0	0	0	0
1	1	0	0	0	1	1
1	1	0	0	1	0	0
1	1	0	0	1	1	1
1	1	0	1	0	0	0
1	1	0	1	0	1	1
1	1	0	1	1	0	1
1	1	0	1	1	1	1
1	1	1	0	0	0	1
1	1	1	0	0	1	1
1	1	1	0	1	0	1
1	1	1	0	1	1	1
1	1	1	1	0	0	1
1	1	1	1	0	1	1
1	1	1	1	1	0	1
1	1	1	1	1	1	1

b. Z = A'B' + AB' + A'B + AB



A	B	x
0	0	1
0	1	1
1	0	1
1	1	1

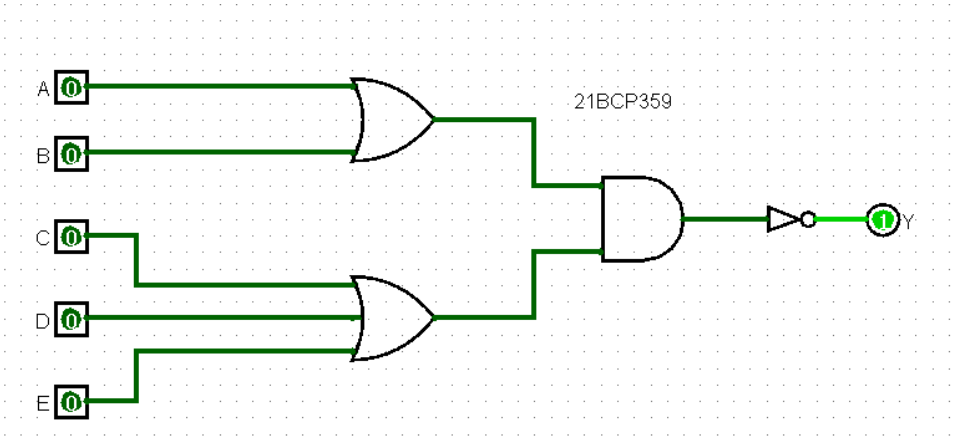
c. K = (A'+B') (A+B') (A'+B) (A+B)



A	B	K
0	0	0
0	1	0
1	0	0
1	1	0

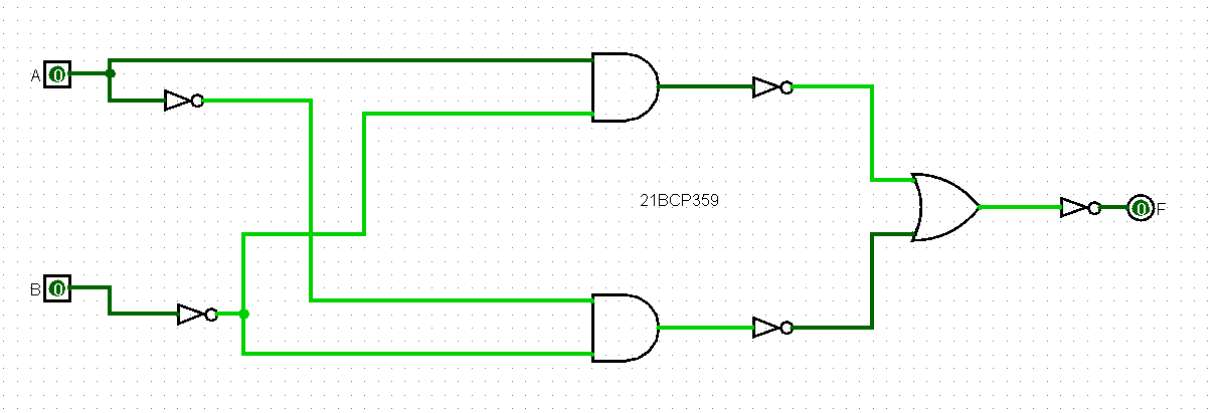
Question 2: Design the verify the functionality using existing library of basic gates in Logisim.

a. Y = ((A+B).(C+D+E))'



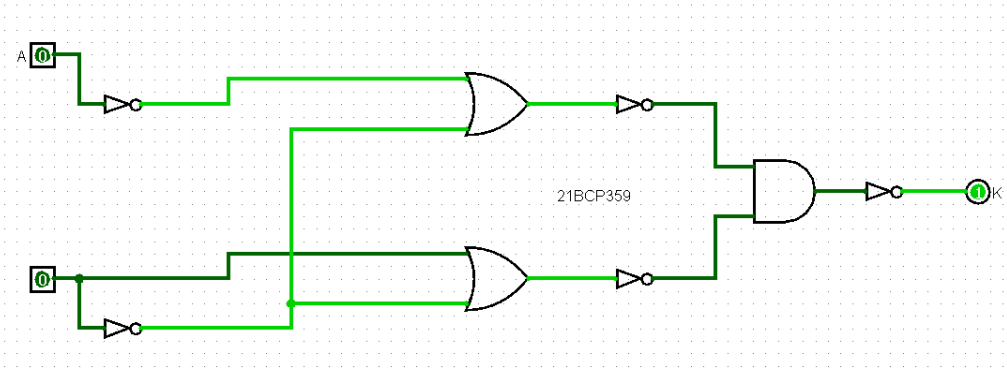
A	B	C	D	E	Y
0	0	0	0	0	1
0	0	0	0	1	1
0	0	0	1	0	1
0	0	0	1	1	1
0	0	1	0	0	1
0	0	1	0	1	1
0	0	1	1	0	1
0	0	1	1	1	1
0	1	0	0	0	1
0	1	0	0	1	0
0	1	0	1	0	0
0	1	0	1	1	0
0	1	1	0	0	0
0	1	1	0	1	0
0	1	1	1	0	0
0	1	1	1	1	0
1	0	0	0	0	1
1	0	0	0	1	0
1	0	0	1	0	0
1	0	0	1	1	0
1	0	1	0	0	0
1	0	1	0	1	0
1	0	1	1	0	0
1	0	1	1	1	0
1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

b. F = ((A.B')'+((A)'(B)')')'



A	B	F
0	0	0
0	1	0
1	0	0
1	1	0

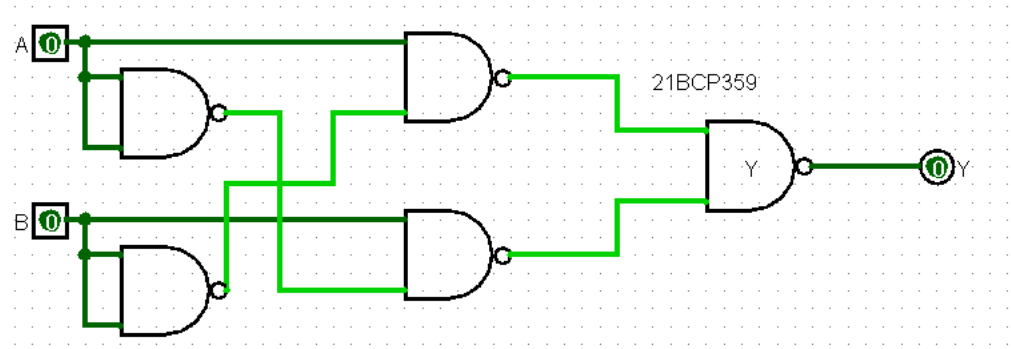
c. K = (((A)'+B)'. ((A)'+(B)')')'



A	a	K
0	0	1
0	1	1
1	0	1
1	1	1

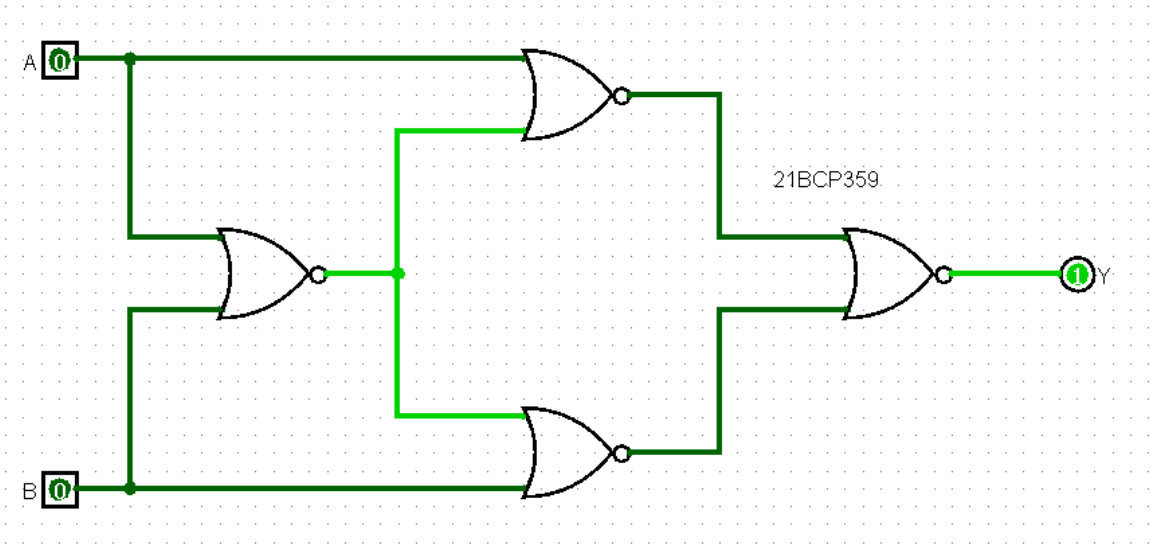
Question 3: For each of the following problem statement, design a circuit, describe a Truth table and verify the outcome using Logisim

a. Implement the functionality of XOR gate using only NAND gates.



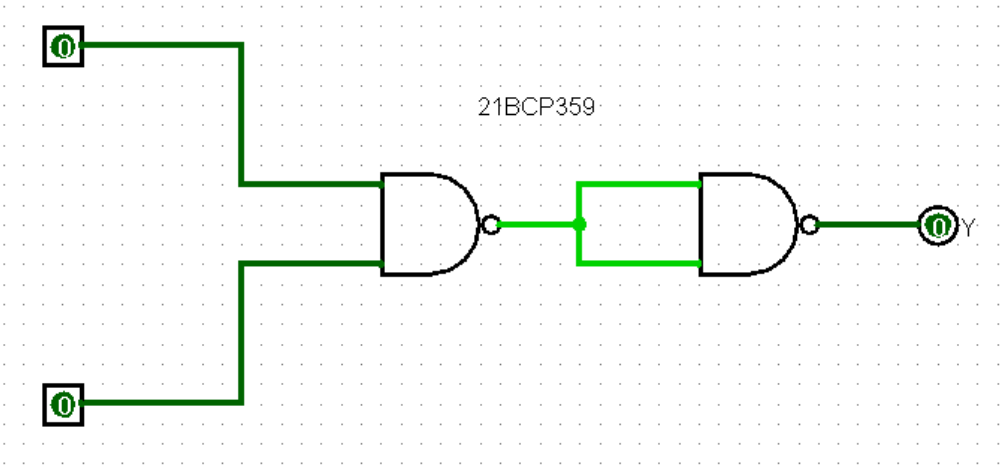
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

b. Implement the functionality of XNOR gate using only NOR gates.



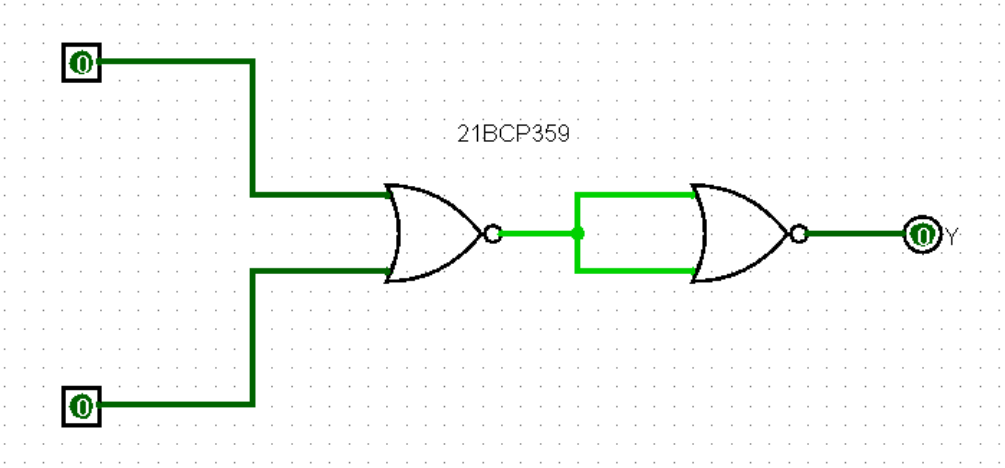
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

c. Implement the functionality of AND gate using only NAND gates.



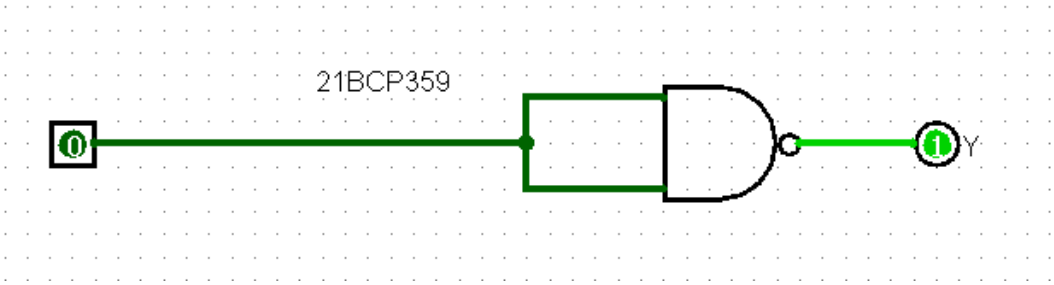
a	b	Y
0	0	0
0	1	0
1	0	0
1	1	1

d. Implement the functionality of OR gate using only NOR gates.



a	b	Y
0	0	0
0	1	1
1	0	1
1	1	1

e. Implement the functionality of NOT gate using only NAND gate.



a	Y
0	1
1	0

Lab 2: Introduction To Verilog

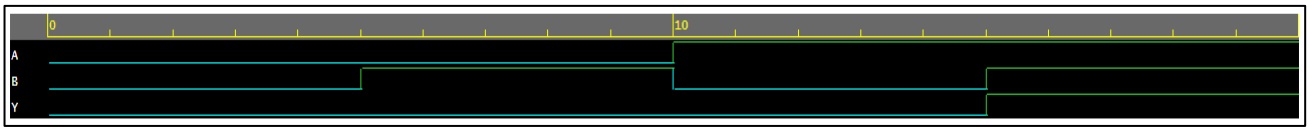
Question 1: Write a Verilog code to implement AND gate. Write the corresponding Testbench code for the verification of your Verilog code

testbench.sv

```
// Code your testbench here
// or browse Examples

module tb_and_gate;

  reg A,B;
  wire Y;
  and_gate a1 (.a(A), .b(B), .y(Y));

initial
  begin
        A = 0; B =0; #5;
        A = 0; B =1; #5;
        A = 1; B =0; #5;
        A = 1; B =1; #5;
  end

initial
  begin
    $dumpfile("dump.vcd");
    $dumpvars(1);
  end

endmodule
```

design.sv

```
// Code your design here
module and_gate(

  input a,b,
  output y);
  assign y = a & b;

endmodule
```



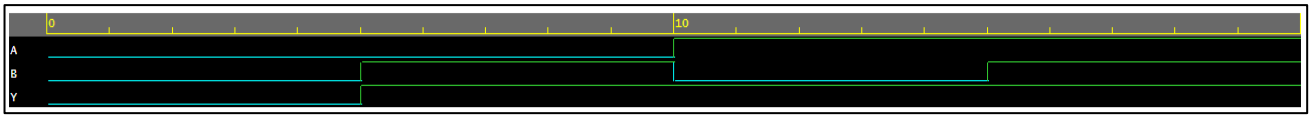
Question 2: Write a Verilog code to implement OR gate. Write the corresponding Testbench code for the verification of your Verilog code.

testbench.sv

```
// Code your testbench here
// or browse Examples

module tb_or_gate;

  reg A,B;
  wire Y;
  or_gate a1 (.a(A), .b(B), .y(Y));

initial
  begin
        A = 0; B =0; #5;
        A = 0; B =1; #5;
        A = 1; B =0; #5;
        A = 1; B =1; #5;
  end

initial
  begin
    $dumpfile("dump.vcd");
    $dumpvars(1);
  end

endmodule
```

design.sv

```
// Code your design here
module or_gate(

  input a,b,
  output y);
  assign y = a | b;

endmodule
```



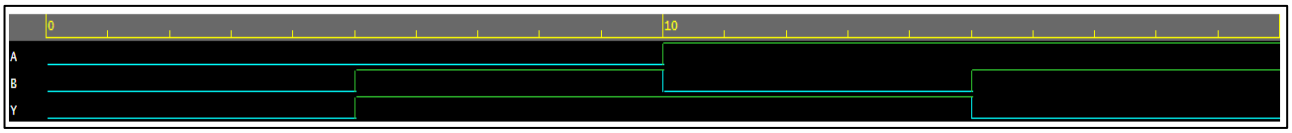
Question 3: Write a Verilog code to implement XOR gate. Write the corresponding Testbench code for the verification of your Verilog code.

testbench.sv

```
// Code your testbench here
// or browse Examples

module tb_xor_gate;

  reg A,B;
  wire Y;
  xor_gate a1 (.a(A), .b(B), .y(Y));

initial
  begin
        A = 0; B =0; #5;
        A = 0; B =1; #5;
        A = 1; B =0; #5;
        A = 1; B =1; #5;
  end

initial
  begin
    $dumpfile("dump.vcd");
    $dumpvars(1);
  end

endmodule
```

design.sv

```
// Code your design here
module xor_gate(

  input a,b,
  output y);
  assign y = ~a&b | a&~b;

endmodule
```



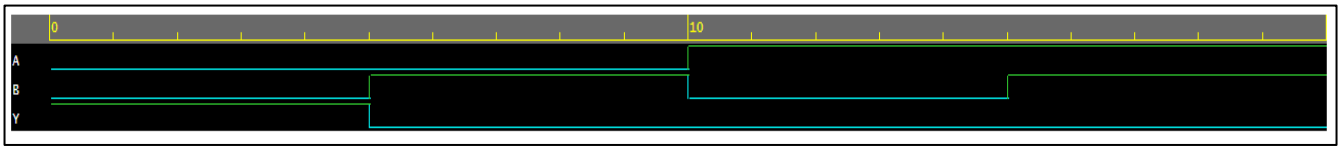
Question 4: Write a Verilog code to implement NOR gate. Write the corresponding Testbench code for the verification of your Verilog code.

testbench.sv

```
// Code your testbench here
// or browse Examples

module tb_nor_gate;

  reg A,B;
  wire Y;
  nor_gate a1 (.a(A), .b(B), .y(Y));

initial
  begin
        A = 0; B =0; #5;
        A = 0; B =1; #5;
        A = 1; B =0; #5;
        A = 1; B =1; #5;
  end

initial
  begin
    $dumpfile("dump.vcd");
    $dumpvars(1);
  end

endmodule
```

design.sv

```
// Code your design here
module nor_gate(

  input a,b,
  output y);
  assign y = ~(a | b);

endmodule
```



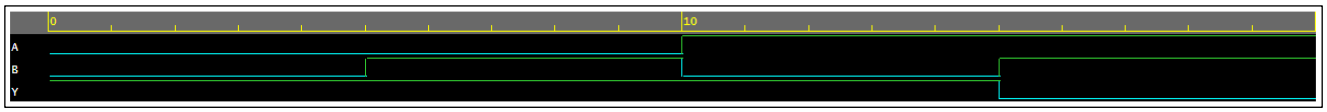
Question 5: Write a Verilog code to implement NAND gate. Write the corresponding Testbench code for the verification of your Verilog code.

testbench.sv

```
// Code your testbench here
// or browse Examples

module tb_nand_gate;

  reg A,B;
  wire Y;
  nand_gate a1 (.a(A), .b(B), .y(Y));

initial
  begin
        A = 0; B =0; #5;
        A = 0; B =1; #5;
        A = 1; B =0; #5;
        A = 1; B =1; #5;
  end

initial
  begin
    $dumpfile("dump.vcd");
    $dumpvars(1);
  end

endmodule
```

design.sv

```
// Code your design here
module nand_gate(

  input a,b,
  output y);
  assign y = ~(a & b);

endmodule
```



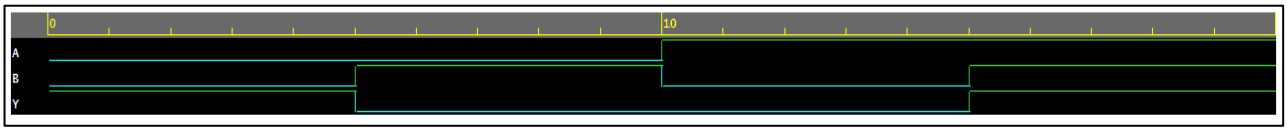
Question 6: Write a Verilog code to implement XNOR gate. Write the corresponding Testbench code for the verification of your Verilog code

testbench.sv

```
// Code your testbench here
// or browse Examples

module tb_xnor_gate;

  reg A,B;
  wire Y;
  xnor_gate a1 (.a(A), .b(B), .y(Y));

initial
  begin
        A = 0; B =0; #5;
        A = 0; B =1; #5;
        A = 1; B =0; #5;
        A = 1; B =1; #5;
  end

initial
  begin
    $dumpfile("dump.vcd");
    $dumpvars(1);
  end

endmodule
```

design.sv

```
// Code your design here
module xnor_gate(

  input a,b,
  output y);
  assign y = ~(~a&b | a&~b);

endmodule
```



Lab 3: Implementation of Structural & Behavioral Verilog Code

Question 1: Implement the following expression using the Verilog HDL. Moreover, Verify your circuit against the waveform.

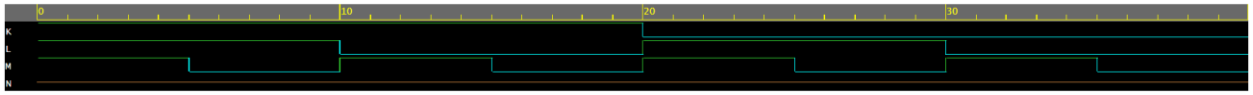
a. F(A, B, C)= A'BC + AB'C + ABC

testbench.sv

```
module tb_exp;
  reg K,L,M;
  wire N;
  build_exp x1 (.k(K), .l(L), .m(M), .n(N));
  initial
    begin
      K = 1; L = 1; M = 1; #5;
      K = 1; L = 1; M = 0; #5;
      K = 1; L = 0; M = 1; #5;
      K = 1; L = 0; M = 0; #5;
      K = 0; L = 1; M = 1; #5;
      K = 0; L = 1; M = 0; #5;
      K = 0; L = 0; M = 1; #5;
      K = 0; L = 0; M = 0; #5;
    end
  initial
    begin
      $dumpfile("dump.vcd");
      $dumpvars(1);
    end
endmodule
```

design.sv

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

module not_gate(input e, output f);
  assign f = ~e;
endmodule

module or_gate(input p,q,r, output s);
  assign t = p|q|r|s;
endmodule

module build_exp(input k,l,m, output n);
  wire w,x,y, k_not,l_not,m_not;

  not_gate n1 (.e(k), .f(k_not));
  not_gate n2 (.e(l), .f(l_not));
  not_gate n3 (.e(m), .f(m_not));

  and_gate a1 (.a(k_not), .b(l), .c(m), .d(w));
  and_gate a2 (.a(k), .b(l_not), .c(m), .d(x));
  and_gate a3 (.a(k), .b(l), .c(m), .d(y));

  or_gate o1 (.p(w), .q(x), .r(y), .s(z));
endmodule
```



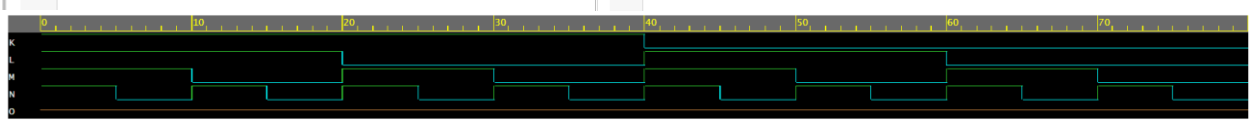
b. F(A, B, C, D)=ABCD' + A'BC D+ AB'CD' + ABC

testbench.sv

```
module tb_exp;
  reg K,L,M,N;
  wire O;
  build_exp x1 (.k(K), .l(L), .m(M), .n(N), .o(O));
  initial
    begin
      K = 1; L = 1; M = 1; N = 1; #5;
      K = 1; L = 1; M = 1; N = 0; #5;
      K = 1; L = 1; M = 0; N = 1; #5;
      K = 1; L = 1; M = 0; N = 0; #5;
      K = 1; L = 0; M = 1; N = 1; #5;
      K = 1; L = 0; M = 1; N = 0; #5;
      K = 1; L = 0; M = 0; N = 1; #5;
      K = 1; L = 0; M = 0; N = 0; #5;
      K = 0; L = 1; M = 1; N = 1; #5;
      K = 0; L = 1; M = 1; N = 0; #5;
      K = 0; L = 1; M = 0; N = 1; #5;
      K = 0; L = 1; M = 0; N = 0; #5;
      K = 0; L = 0; M = 1; N = 1; #5;
      K = 0; L = 0; M = 1; N = 0; #5;
      K = 0; L = 0; M = 0; N = 1; #5;
      K = 0; L = 0; M = 0; N = 0; #5;
    end
  initial
    begin
      $dumpfile("dump.vcd");
      $dumpvars(1);
    end
endmodule
```

design.sv

```
module and_gate1(input g,h,i, output j);
  assign j = g&h&i;
endmodule

module and_gate2(input a,b,c,d, output e);
  assign e = a&b&c&d;
endmodule

module not_gate(input e, output f);
  assign f = ~e;
endmodule

module or_gate(input p,q,r,s, output t);
  assign t = p|q|r|s;
endmodule

module build_exp(input k,l,m,n, output o);
  wire w,x,y,z,out, k_not,l_not,m_not,n_not;

  not_gate n1 (.e(k), .f(k_not));
  not_gate n2 (.e(l), .f(l_not));
  not_gate n3 (.e(m), .f(m_not));
  not_gate n4 (.e(n), .f(n_not));

  and_gate2 a1 (.a(k), .b(l), .c(m), .d(n_not), .e(w));
  and_gate2 a2 (.a(k), .b(l_not), .c(m), .d(n), .e(x));
  and_gate2 a3 (.a(k), .b(l_not), .c(m), .d(n_not), .e(y));
  and_gate1 a4 (.g(k), .h(l), .i(m), .j(z));

  or_gate o1 (.p(w), .q(x), .r(y), .s(z), .t(out));
endmodule
```



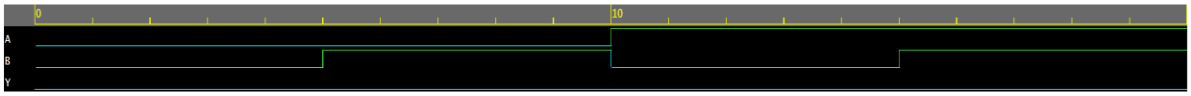
c. F(A, B) = (A'+B') (A+B') (A'+B) (A+B)

testbench.sv

```
// Code your testbench here
// or browse Examples
module tbandgate;
  reg A, B;
  wire Y;
  andgate a1 (.a(A),.b(B),.y(Y));
  initial
    begin
      A=0;B=0;#5;
      A=0;B=1;#5;
      A=1;B=0;#5;
      A=1;B=1;#5;
    end
  initial
    begin
      $dumpfile("dump.vcd");
      $dumpvars(1);
    end
endmodule
```

design.sv

```
// Code your design here
module andgate(
  input a,b,
  output y);
  assign y = (~a | ~b) & (a | ~b) & (~a | b) & (a | b);
endmodule
```



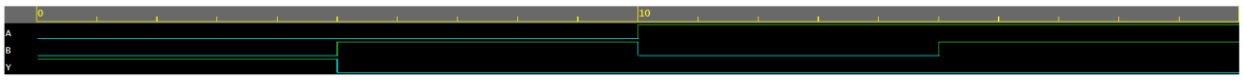
d. (A, B) = ((A.B')'+((A)'(B)')')'

testbench.sv

```
// Code your testbench here
// or browse Examples
module tbandgate;
  reg A, B;
  wire Y;
  andgate a1 (.a(A),.b(B),.y(Y));
  initial
    begin
      A=0;B=0;#5;
      A=0;B=1;#5;
      A=1;B=0;#5;
      A=1;B=1;#5;
    end
  initial
    begin
      $dumpfile("dump.vcd");
      $dumpvars(1);
    end
endmodule
```

design.sv

```
// Code your design here
module andgate(
  input a,b,
  output y);
  assign y = ~((a & ~b) | (~(~a & ~b)));
endmodule
```



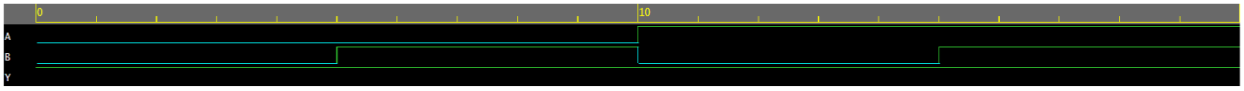
e. (A, B) = (((A)'+B)'. ((A)'+(B)')')'

testbench.sv

```
// Code your testbench here
// or browse Examples
module tbandgate;
  reg A, B;
  wire Y;
  andgate a1 (.a(A),.b(B),.y(Y));
  initial
    begin
      A=0;B=0;#5;
      A=0;B=1;#5;
      A=1;B=0;#5;
      A=1;B=1;#5;
    end
  initial
    begin
      $dumpfile("dump.vcd");
      $dumpvars(1);
    end
endmodule
```

design.sv

```
// Code your design here
module andgate(
  input a,b,
  output y);
  assign y = ~(~((~a | b)) & (~(~a | ~b)));
endmodule
```



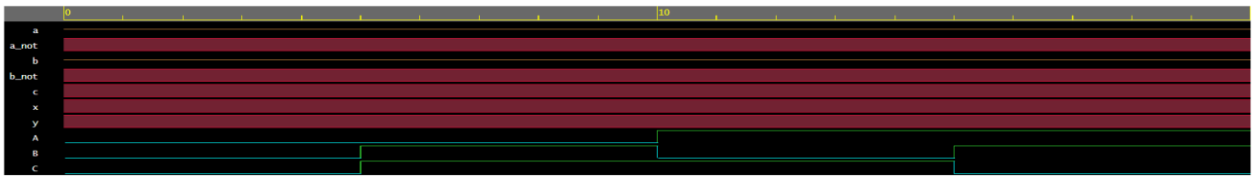
Question 2: Implement the XOR gate using Behavioral and Structural code of Verilog Hardware Description Language.

testbench.sv

```
module tb_xor_str;
  reg A,B;
  wire C;
  build_xor x1 (.m(A), .n(B), .o(C));
  initial
    begin
      A = 0; B =0; #5;
      A = 0; B =1; #5;
      A = 1; B =0; #5;
      A = 1; B =1; #5;
    end
  initial
    begin
      $dumpfile("dump.vcd");
      $dumpvars(1);
    end
endmodule
```

design.sv

```
// Behavioural Code
module not_gate (input e, output f);
  assign f = ~e;
endmodule
module and_gate (input a, b, output c);
  assign c = a & b;
endmodule
module or_gate (input p, q, output r);
  assign r = p | q;
endmodule
module build_xor (input m, n, output o);
  wire x, y, a_not, b_not;
  not_gate n1 (.e(m),.f(a_not));
  not_gate n2 (.e(n),.f(b_not));
  and_gate a1 (.a(a_not),.b(n),.c(x));
  and_gate a2 (.a(m),.b(b_not),.c(y));
  or_gate o1 (.p(x),.q(y),.r(o));
endmodule

// Structural Code
module Lab4_Build_XOR(a, b, c);
input a,b;
output c;
wire a_not, b_not;
wire x,y;
not(a_not,a);
not(b_not,b);
and(x,a_not,b);
and(y,a,b_not);
or(c,x,y);
endmodule
```



Question 3: Implement the following expression using the Verilog Hardware Description Language (HDL) (Structural Coding).

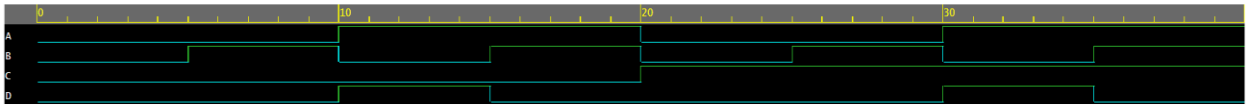
a. F (A, B, C) = (A'+B'+C')(A+B'+C')(A'+B+C')(A'+B')`

testbench.sv

```
// Code your testbench here
// or browse Examples
module tbxor;
  reg A, B, C;
  wire D;
  buildxor x1 (.m(A), .n(B), .o(C), .t(D));
  initial
    begin
      A = 0; B = 0; C = 0; #5;
      A = 0; B = 1; C = 0; #5;
      A = 1; B = 0; C = 0; #5;
      A = 1; B = 1; C = 0; #5;
      A = 0; B = 0; C = 1; #5;
      A = 0; B = 1; C = 1; #5;
      A = 1; B = 0; C = 1; #5;
      A = 1; B = 1; C = 1; #5;
    end
  initial
    begin
      $dumpfile("dump.vcd");
      $dumpvars(1);
    end
endmodule
```

design.sv

```
// Code your design here
module notgate (input e, output f);
  assign f = ~e;
endmodule

module andgate (input a, b, c, output d);
  assign d = a & b & c;
endmodule

module orgate (input p, q, output r);
  assign r = p | q;
endmodule

module buildxor (input m, n, o, output t);
  wire w, x, anot, bnot;
  notgate n1 (.e(n), .f(bnot));
  notgate n2 (.e(o), .f(cnot));
  andgate a1 (.a(m), .b(bnot), .c(o), .d(w));
  andgate a2 (.a(m), .b(bnot), .c(cnot), .d(x));
  orgate o1 (.p(x), .q(w), .r(t));
endmodule
```



Question 3: Implement the following expression using universal NAND and NOR gate. Write down Verilog Structural and Behavioral code for that expression.

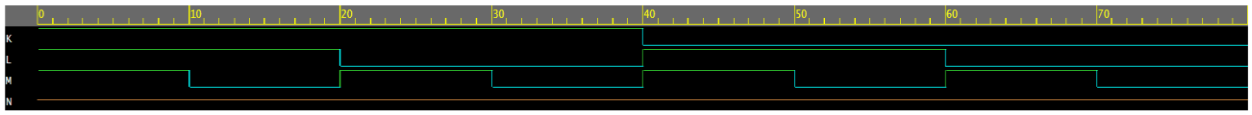
a. F (A, B, C) = (AB'C) + (AB'C')

testbench.sv

```
module tb_exp;
  reg K,L,M;
  wire N;
  build_exp x1 (.k(K), .l(L), .m(M), .n(N));
  initial
    begin
      K = 1; L = 1; M = 1; #5;
      K = 1; L = 1; M = 1; #5;
      K = 1; L = 1; M = 0; #5;
      K = 1; L = 1; M = 0; #5;
      K = 1; L = 0; M = 1; #5;
      K = 1; L = 0; M = 1; #5;
      K = 1; L = 0; M = 0; #5;
      K = 1; L = 0; M = 0; #5;
      K = 0; L = 1; M = 1; #5;
      K = 0; L = 1; M = 1; #5;
      K = 0; L = 1; M = 0; #5;
      K = 0; L = 1; M = 0; #5;
      K = 0; L = 0; M = 1; #5;
      K = 0; L = 0; M = 1; #5;
      K = 0; L = 0; M = 0; #5;
      K = 0; L = 0; M = 0; #5;
    end
  initial
    begin
      $dumpfile("dump.vcd");
      $dumpvars(1);
    end
endmodule
```

design.sv

```
module build_exp(k,l,m,n);
  input k,l,m;
  output n;
  wire w,x,y,z;
  wire k_not,l_not,m_not;

  not (k, k_not);
  not (l, l_not);
  not (m, m_not);

  or (k_not, l_not, m_not, w);
  or (k, l_not, m_not, x);
  or (k_not, l, m_not, y);
  or (k_not, l_not, l_not, z);

  not (z, z_not);

  and (w, x, y, z_not, out);

endmodule
```



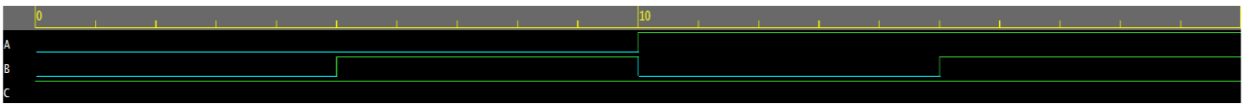
b. F (A, B) = A'B' + AB' + A'B + AB

testbench.sv

```
// Code your testbench here
// or browse Examples
module tbxor;
  reg A, B;
  wire C;
  buildxor x1 (.m(A), .n(B), .o(C));
  initial
    begin
      A = 0; B = 0; #5;
      A = 0; B = 1; #5;
      A = 1; B = 0; #5;
      A = 1; B = 1; #5;
    end
  initial
    begin
      $dumpfile("dump.vcd");
      $dumpvars(1);
    end
endmodule
```

design.sv

```
// Code your design here
module notgate (input e, output f);
  assign f = ~e;
endmodule

module andgate (input a, b, output c);
  assign c = a & b;
endmodule

module orgate (input p, q, r, s, output t);
  assign t = p | q | r | s;
endmodule

module buildxor (input m, n, output o);
  wire w, x, y, z, anot, bnot;
  notgate n1 (.e(m), .f(anot));
  notgate n2 (.e(n), .f(bnot));
  andgate a1 (.a(anot), .b(bnot), .c(x));
  andgate a2 (.a(m), .b(bnot), . c(y));
  andgate a3 (.a(anot), .b(n), .c(z));
  andgate a4 (.a(m), .b(n), .c(w));
  orgate o1 (.p(x),. q(y), .r(z), .s(w), .t(o));
endmodule
```



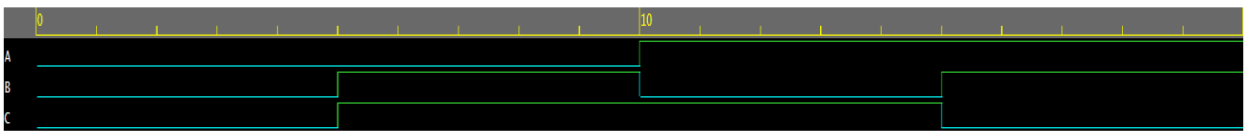
Question 4 Write down three modules in a single Verilog code to design AND, OR and NOT gate. Now use those modules to design the XOR gate. Use AND, OR and NOT gate as the instances to implement the XOR gate. Write the corresponding Testbench code for the verification of your XOR gate.

testbench.sv

```
module tb_xor_str;
  reg A,B;
  wire C;
  build_xor x1 (.m(A), .n(B), .o(C));
  initial
    begin
      A = 0; B = 0; #5;
      A = 0; B = 1; #5;
      A = 1; B = 0; #5;
      A = 1; B = 1; #5;
    end
  initial
    begin
      $dumpfile("dump.vcd");
      $dumpvars(1);
    end
endmodule
```

design.sv

```
module and_gate(input a, b, output c);
  assign c = a&b;
endmodule

module not_gate(input e, output f);
  assign f = ~e;
endmodule

module or_gate(input p,q, output r);
  assign r = p|q;
endmodule

module build_xor(input m, n, output o);
  wire x, y, a_not, b_not;
  not_gate n1 (.e(m),.f(a_not));
  not_gate n2 (.e(n),.f(b_not));
  and_gate a1 (.a(a_not),.b(n),.c(x));
  and_gate a2 (.a(m),.b(b_not),.c(y));
  or_gate o1 (.p(x),.q(y),.r(o));
endmodule
```



Question 5 Consider the following expression: Y = A'.B'.C' + A'.B.C' + A.B'.C' + A.B'.C

Design one three input 'And' gate module and one four input 'OR' gate module using Verilog. Instantiate those two modules to design the above-mentioned expression. Design the corresponding Testbench code for the verification purpose.

A	B	C	Y
F	F	F	T
F	F	T	F
F	T	F	T
F	T	T	F
T	F	F	T
T	F	T	T
T	T	F	F
T	T	T	F

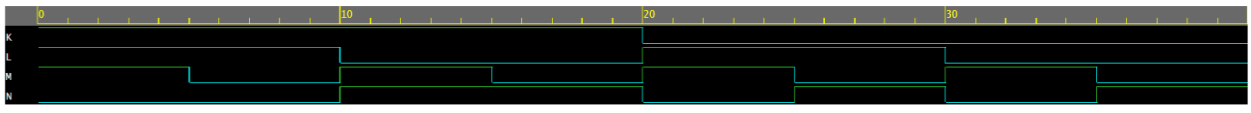
Reduced Exression: A'.B'.C'

testbench.sv

```
module tb_exp;
  reg K,L,M;
  wire N;
  build_exp x1 (.k(K), .l(L), .m(M), .n(N));
  initial
    begin
      K = 1; L = 1; M = 1; #5;
      K = 1; L = 1; M = 0; #5;
      K = 1; L = 0; M = 1; #5;
      K = 1; L = 0; M = 0; #5;
      K = 0; L = 1; M = 1; #5;
      K = 0; L = 1; M = 0; #5;
      K = 0; L = 0; M = 1; #5;
      K = 0; L = 0; M = 0; #5;
    end
  initial
    begin
      $dumpfile("dump.vcd");
      $dumpvars(1);
    end
endmodule
```

design.sv

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

module not_gate(input e, output f);
  assign f = ~e;
endmodule

module or_gate(input p,q,r,s, output t);
  assign t = p|q|r|s;
endmodule

module build_exp(input k,l,m, output n);
  wire w,x,y,z, k_not,l_not,m_not;

  not_gate n1 (.e(k), .f(k_not));
  not_gate n2 (.e(l), .f(l_not));
  not_gate n3 (.e(m), .f(m_not));

  and_gate a1 (.a(k_not), .b(l_not), .c(m_not), .d(w));
  and_gate a2 (.a(k_not), .b(l), .c(m_not), .d(x));
  and_gate a3 (.a(k), .b(l_not), .c(m_not), .d(y));
  and_gate a4 (.a(k), .b(l_not), .c(m), .d(z));

  or_gate o1 (.p(w), .q(x), .r(y), .s(z), .t(n));
endmodule
```



Lab 4: Design Half Adder & Full Adder Using Half Adder

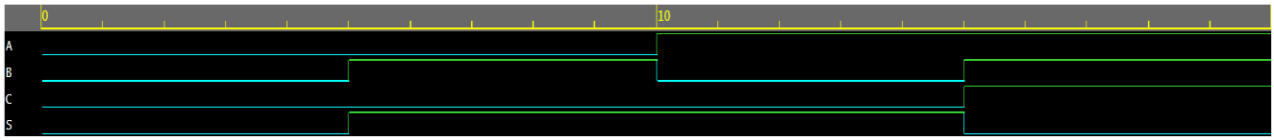
Question 1: Write Verilog code for half adder. Test using waveform

design.sv

```
module xorgate(
  input a,b,
  output sum, carry);
    assign sum = ~a & b || a & ~b;
    assign carry = a & b;
endmodule
```

testbench.sv

```
module tbxorgate;
  reg A, B;
  wire S, C;
  xorgate a1 (.a(A),.b(B),.sum(S),.carry(C));
  initial
    begin
      A=0;B=0;#5;
      A=0;B=1;#5;
      A=1;B=0;#5;
      A=1;B=1;#5;
    end
  initial
    begin
      $dumpfile("dump.vcd");
      $dumpvars(1);
    end
endmodule
```



Question 3. Write Verilog code for full adder. Test using waveform.

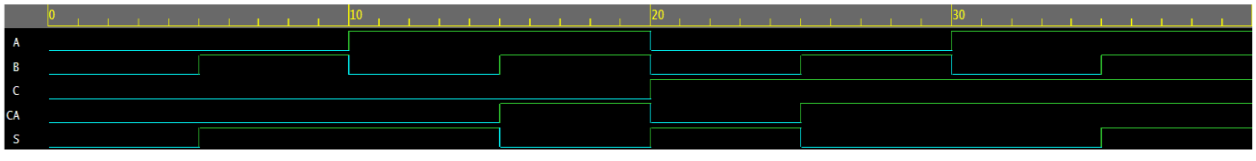
testbench.sv

```
module tbxorgate;
  reg A, B, C;
  wire S, CA;
  xorgate a1 (.a(A),.b(B),.c(C),.sum(S),.carry(CA));
  initial
    begin
      A=0;B=0;C=0;#5;
      A=0;B=1;C=0;#5;
      A=1;B=0;C=0;#5;
      A=1;B=1;C=0;#5;
      A=0;B=0;C=1;#5;
      A=0;B=1;C=1;#5;
      A=1;B=0;C=1;#5;
      A=1;B=1;C=1;#5;
    end
  initial
    begin
      $dumpfile("dump.vcd");
      $dumpvars(1);
    end
endmodule
```

design.sv

SV/Verilog Design

```
module xorgate(
  input a,b,c,
  output sum, carry);
  assign sum = (~(~a & b || a & ~b) & c) | ((~a & b || a & ~b) & ~c);
  assign carry = (a & b) | (b & c) | (c & a);
endmodule
```



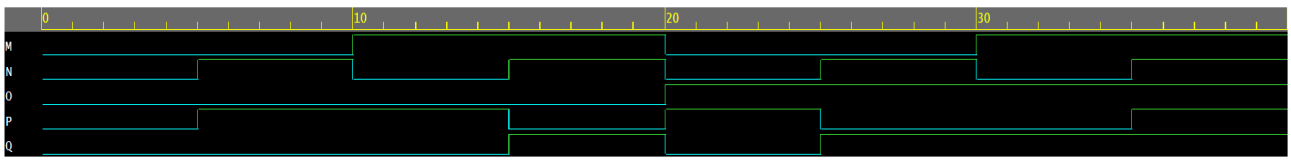
Question 4. Write Verilog code for full adder using module instantiation of half adder. Test using waveform.

design.sv

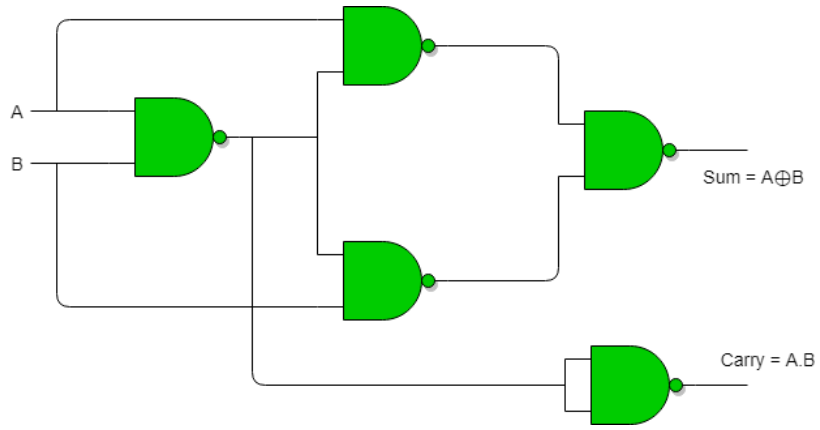
```
module halfadder (input a, b, output c, s);
  assign c = a & b;
  assign s = (~a & b) | (a & ~b);
endmodule

module orgate (input i, j, output k);
  assign k = i | j;
endmodule

module fulladder(input m, n, o, output p , q);
  wire sum1, carry1, carry2;
  halfadder h1 (.a(m),.b(n),.s(sum1));
  halfadder h2 (.a(m),.b(n),.c(carry1));
  halfadder h3 (.a(sum1),.b(o),.s(p));
  halfadder h4 (.a(sum1),.b(o),.c(carry2));
  orgate or1 (.i(carry1),.j(carry2),.k(q));

endmodule

```

testbench.sv

```
module fulladder_tb;
  reg M, N, O;
  wire P, Q;
  fulladder x1 (.m(M), .n(N), .o(O), .p(P), .q(Q));
  initial
    begin
      M = 0; N = 0; O = 0; #5;
      M = 0; N = 1; O = 0; #5;
      M = 1; N = 0; O = 0; #5;
      M = 1; N = 1; O = 0; #5;
      M = 0; N = 0; O = 1; #5;
      M = 0; N = 1; O = 1; #5;
      M = 1; N = 0; O = 1; #5;
      M = 1; N = 1; O = 1; #5;
    end
  initial
    begin
      $dumpfile("dump.vcd");
      $dumpvars(1);
    end
endmodule

```



Question 5. Write Verilog code for half adder using only NAND gate. Test using waveform

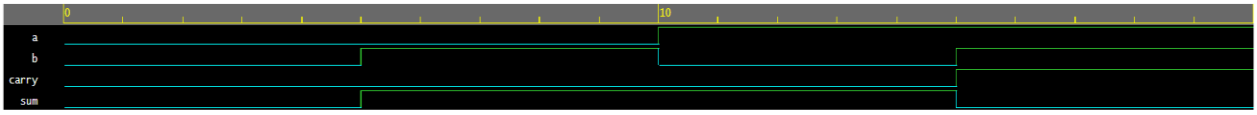


testbench.sv

```
module tb_ha_nand;
  reg a,b;
  wire sum,carry;

  ha_nand ha1(a,b,sum,carry);


  initial
    begin
      a = 0; b = 0; #5;
      a = 0; b = 1; #5;
      a = 1; b = 0; #5;
      a = 1; b = 1; #5;
    end
initial
  begin
    $dumpfile("dump.vcd");
    $dumpvars(1);
  end
endmodule

```

design.sv

```
module ha_nand(input a,b, output sum,carry);
  wire temp1,temp2,temp3assign ;
  assign temp1 = ~(a & b);

  assign temp2 = ~(a & temp1);
  assign temp3 = ~(b & temp1);

  assign sum = ~(temp2 & temp3);

  assign carry = ~(temp1 & temp1);
endmodule
```



LAB 5: Design Half Subtractor, Full Subtractor Using Half Subtractors

Question 1: Write a Verilog code to design a Half subtractor and test it using the Waveform

design.sv

```
module HalfSubstractor(a,b,difference,borrow);
  input a,b;
  output difference,borrow;
  xor(difference,a,b);
  assign borrow=(~a&b);
endmodule
```

testbench.sv

SV/Verilog Testbench

```
module TestModule;
  // Inputs
  reg a;
  reg b;

  // Outputs
  wire difference;
  wire borrow;

  HalfSubstractor uut (.a(a),.b(b), .difference(difference),
.borrow(borrow));
  initial begin
    // Initialize Inputs
    a = 0;
    b = 0;
    // Wait 100 ns for global reset to finish
    #100;
    a = 1;
    b = 0;
  end
  initial begin
    $dumpfile("dump.vcd");
    $dumpvars(1);
  end
endmodule
```

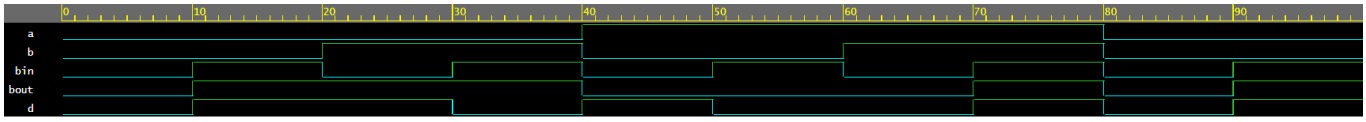
Question 2: Write a Verilog code to design a Full subtractor using Half subtractors and test it using the Waveform.

testbench.sv

```
module top;
  reg a, b, bin;
  wire d, bout;

  full_subtractor instantiation(.A(a), .B(b), .Bin(bin), .D(d), .Bout(bout));

  initial
    begin
      $dumpfile("xyz.vcd");
      $dumpvars;
      a=0;
      b=0;
      bin=0;
      #100 $finish;
    end
  always #40 a=~a;
  always #20 b=~b;
  always #10 bin=~bin;
  always @(a or b or bin)
    $monitor("At TIME(in ns)=%t, A=%d B=%d Bin=%d D = %d Bout = %d", $time, a, b, bin, d, bout);
  endmodule
```

design.sv

```
module or_gate(a0, b0, c0);
  input a0, b0;
  output c0;
  assign c0 = a0 | b0;
endmodule

module xor_gate(a1, b1, c1);
  input a1, b1;
  output c1;
  assign c1 = a1 ^ b1;
endmodule

module and_gate(a2, b2, c2);
  input a2, b2;
  output c2;
  assign c2 = a2 & b2;
endmodule

module not_gate(a3, b3);
  input a3;
  output b3;
  assign b3 = ~ a3;
endmodule

module half_subtractor(a4, b4, c4, d4);
  input a4, b4;
  output c4, d4;
  wire x;
  xor_gate u1(a4, b4, c4);
  and_gate u2(x, b4, d4);
  not_gate u3(a4, x);
endmodule

module full_subtractor(A, B, Bin, D, Bout);
  input A, B, Bin;
  output D, Bout;
  wire p, q, r;
  half_subtractor u4(A, B, p, q);
  half_subtractor u5(p, Bin, D, r);
  or_gate u6(q, r, Bout);
endmodule
```



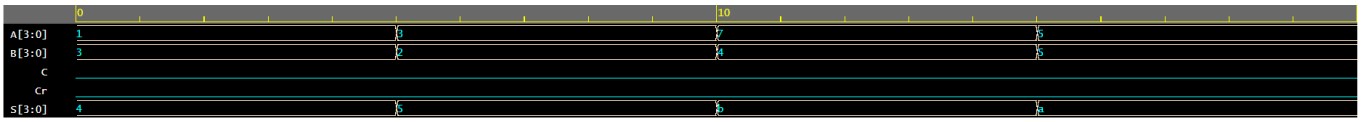
Question 3: Write the Verilog code (either structural or behavioural) for a 4-bit ripple carry adder. The block diagram of a 4-bit ripple carry adder has been given in Figure 1 for your reference.

design.sv

```
module FA (a,b,c,s,cr);
  input a,b,c;
  output s,cr;
  wire s1,c1,c2;

  xor(s1, a, b);
  xor(s, s1, c);
  and(c1, a, b);
  and(c2, s1, c);
  or(cr, c1, c2);
endmodule

module RCA(x,y,z,sum,cout);
  input [3:0]x,y;
  input z;
  output [3:0]sum;
  output cout;
  wire w1,w2,w3;

  FA f0(x[0], y[0], z, sum[0], w1);
  FA f1(x[1], y[1], w1, sum[1], w2);
  FA f2(x[2], y[2], w2, sum[2], w3);
  FA f3(x[3], y[3], w3, sum[3], cout);
endmodule
```

testbench.sv

```
module tb_RCA;
  reg [3:0]A,B;
  reg C;
  wire [3:0]S;
  wire Cr;

  RCA rca1(.x(A), .y(B), .z(C), .sum(S), .cout(Cr));
  initial
  begin
    A = 1; B = 3; C = 0; #5;
    A = 3; B = 2; C = 0; #5;
    A = 7; B = 4; C = 0; #5;
    A = 5; B = 5; C = 0; #5;
  end
  initial begin
    $dumpfile("dump.vcd");
    $dumpvars(1);
  end
endmodule
```



Question 4: Write the Verilog code to construct a 4-bit adder-subtractor using. The logic diagram of a 4-bit adder-subtractor using a ripple carry adder has been given in Figure 2 for your reference.

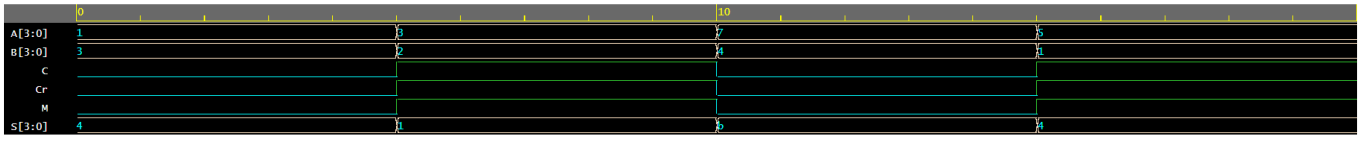
design.sv

```
module FA (a,b,c,s,cr);
  input a,b,c;
  output s,cr;
  wire s1,c1,c2;

  xor(s1, a, b);
  xor(s, s1, c);
  and(c1, a, b);
  and(c2, s1, c);
  or(cr, c1, c2);
endmodule

module FBAS(x,y,c0,m,sum,cout);
  input [3:0]x,y;
  input c0,m;
  output [3:0]sum;
  output cout;

  FA f0(x[0], y[0]^m, c0, sum[0], c1);
  FA f1(x[1], y[1]^m, c1, sum[1], c2);
  FA f2(x[2], y[2]^m, c2, sum[2], c3);
  FA f3(x[3], y[3]^m, c3, sum[3], cout);
endmodule
```

testbench.sv

SV/Verilog Testbench

```
module tb_FBAS;
  reg [3:0]A,B;
  reg C,M;
  wire [3:0]S;
  wire Cr;

  FBAS fbas1(.x(A), .y(B), .c0(C), .m(M), .sum(S), .cout(Cr));
  initial
  begin
    A = 1; B = 3; C = 0; M = 0; #5; // Adder
    A = 3; B = 2; C = 1; M = 1; #5; // Subtractor
    A = 7; B = 4; C = 0; M = 0; #5; // Adder
    A = 5; B = 1; C = 1; M = 1; #5; // Subtractor
  end
  initial begin
    $dumpfile("dump.vcd");
    $dumpvars(1);
  end
endmodule
```



Question 5: Write the Verilog code for a 4-bit carry look-ahead adder (CLA). The block diagram of the same has been given in Figure 3.

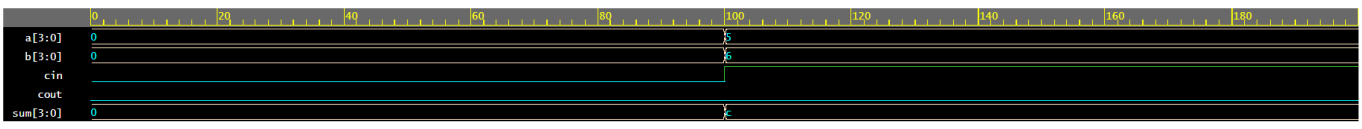
design.sv

SV/Verilog Design

```
module CLA_Adder(a,b,cin,sum,cout);
  input[3:0] a,b;
  input cin;
  output [3:0] sum;
  output cout;
  wire p0,p1,p2,p3,g0,g1,g2,g3,c1,c2,c3,c4;

  assign p0=(a[0]^b[0]),
    p1=(a[1]^b[1]),
    p2=(a[2]^b[2]),
    p3=(a[3]^b[3]);

  assign g0=(a[0]&b[0]),
    g1=(a[1] & b[1]),
    g2=(a[2] & b[2]),
    g3=(a[3] & b[3]);

  assign c0=cin,
    c1=g0|(p0 & cin),
    c2=g1|(p1 & g0)|(p1 & p0 & cin),
    c3=g2|(p2 & g1)|(p2 & p1 & g0)|(p1 & p1 & p0 & cin),
    c4=g3|(p3 & g2)|(p3 & p2 & g1)|(p3 & p2 & p1 & g0)|(p3 & p2 & p1 & p0 & cin);

  assign sum[0]=p0^c0,
    sum[1]=p1^c1,
    sum[2]=p2^c2,
    sum[3]=p3^c3;
  assign cout=c4;
endmodule
```

testbench.sv

```
module TestModule;
  // Inputs
  reg [3:0] a;
  reg [3:0] b;
  reg cin;

  // Outputs
  wire [3:0] sum;
  wire cout;

  // Instantiate the Unit Under Test
  CLA_Adder uut (
    .a(a),
    .b(b),
    .cin(cin),
    .sum(sum),
    .cout(cout)
  );
  initial begin
    // Initialize Inputs
    a = 0;
    b = 0;
    cin = 0;
    // Wait 100 ns for global reset to finish
    #100;
    a = 5;
    b = 6;
    cin = 1;
    // Wait 100 ns for global reset to finish
    #100;
  end
  initial begin
    $dumpfile("dump.vcd");
    $dumpvars(1);
  end
endmodule
```



LAB 6: Designing of Multiplexer, Demultiplexer, Decoder, Encoder

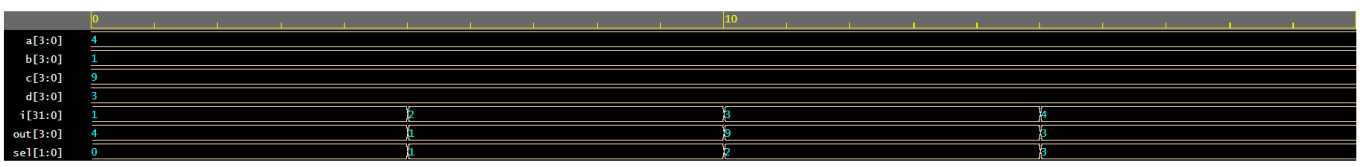
Question 1: Write a Verilog code to design a 4:1 Multiplexer and verify the same.

design.sv

```
module mux_4to1_case ( input [3:0] a,
                       input [3:0] b,
                       input [3:0] c,
                       input [3:0] d,
                       input [1:0] sel,
                       output reg [3:0] out);
   always @ (a or b or c or d or sel) begin
      case (sel)
         2'b00 : out <= a;
         2'b01 : out <= b;
         2'b10 : out <= c;
         2'b11 : out <= d;
      endcase
   end
endmodule
```

testbench.sv

```
module tb_4to1_mux;

   reg [3:0] a;
   reg [3:0] b;
   reg [3:0] c;
   reg [3:0] d;
   wire [3:0] out;
   reg [1:0] sel;
   integer i;

   mux_4to1_case  mux0 (   .a (a),
                           .b (b),
                           .c (c),
                           .d (d),
                           .sel (sel),
                           .out (out));

   initial
     begin
       $monitor ("[%0t] sel=0x%0h a=0x%0h b=0x%0h c=0x%0h d=0x%0h
out=0x%0h", $time, sel, a, b, c, d, out);
       sel <= 0;
      a <= $random;
      b <= $random;
      c <= $random;
      d <= $random;
       $dumpfile("dump.vcd");
      $dumpvars(1);
      for (i = 1; i < 4; i=i+1) begin
         #5 sel <= i;
      end
      #5 $finish;
    end
endmodule
```



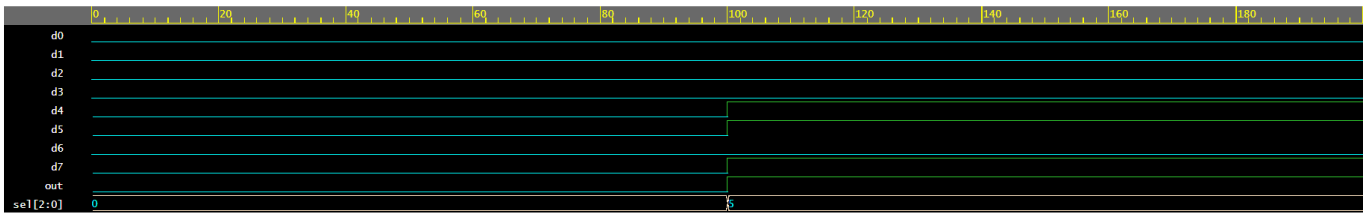
Question 2: Write a Verilog code to design an 8:1 Multiplexer and verify the same.

design.sv

```
module Mulitplexer(d0,d1,d2,d3,d4,d5,d6,d7,sel,out);
  input d0,d1,d2,d3,d4,d5,d6,d7;
  input [2:0] sel;
  output reg out;
  always@(sel)
    begin
      case(sel)
        3'b000:out=d0;
        3'b001:out=d1;
        3'b010:out=d2;
        3'b011:out=d3;
        3'b100:out=d4;
        3'b101:out=d5;
        3'b110:out=d6;
        3'b111:out=d7;
      endcase
    end
endmodule
```

testbench.sv

```
module TestModule;

reg d0, d1, d2, d3, d4, d5, d6, d7;
reg [2:0] sel;
wire out;

Mulitplexer uut(.d0(d0),.d1(d1),.d2(d2),.d3(d3),.d4(d4),.d5(d5),.d6(d6),.d7(d7),.sel(sel),.out(out));

initial
  begin
    d0 = 0;
    d1 = 0;
    d2 = 0;
    d3 = 0;

    d4 = 0;
    d5 = 0;
    d6 = 0;
    d7 = 0;
    sel = 0;
    #100;
    d0 = 0;
    d1 = 0;
    d2 = 0;
    d3 = 0;
    d4 = 1;
    d5 = 1;
    d6 = 0;
    d7 = 1;
    sel = 5;
    #100;
  end
  initial
    begin
      $dumpfile("dump.vcd");
      $dumpvars(1);
    end
endmodule
```



Question 3: Write a Verilog code to design a 1:4 Demultiplexer and verify the same.

design.sv

```
module Demultiplexer1to4case (output reg [3:0] Y,
                              input [1:0] A,
                              input din);
  always @(Y, A) begin
    case (A)
        2'b00 : begin Y[0] = din; Y[3:1] = 0; end
        2'b01 : begin Y[1] = din; Y[0] = 0;   end
        2'b10 : begin Y[2] = din; Y[1:0] = 0; end
        2'b11 : begin Y[3] = din; Y[2:0] = 0; end
    endcase
end
endmodule
```

testbench.sv

```
module Demultiplexer1to4case_tb;
wire [3:0] Y;
reg [1:0] A;
reg din;
Demultiplexer1to4case I0 (Y, A, din);
initial begin
    din = 1;
    A = 2'b00;
#1  A = 2'b01;
#1  A = 2'b10;
#1  A = 2'b11;
end
initial begin
    $monitor("%t| Din = %d| A[1] = %d| A[0] = %d| Y[0] = %d| Y[1] = %d| Y[2] = %d| Y[3] = %d",
              $time, din, A[1], A[0], Y[0], Y[1], Y[2], Y[3]);
    $dumpfile("dump.vcd");
      $dumpvars(1);
end
endmodule
```

Question 4: Write a Verilog code to design a 1:8 Demultiplexer and verify the same.

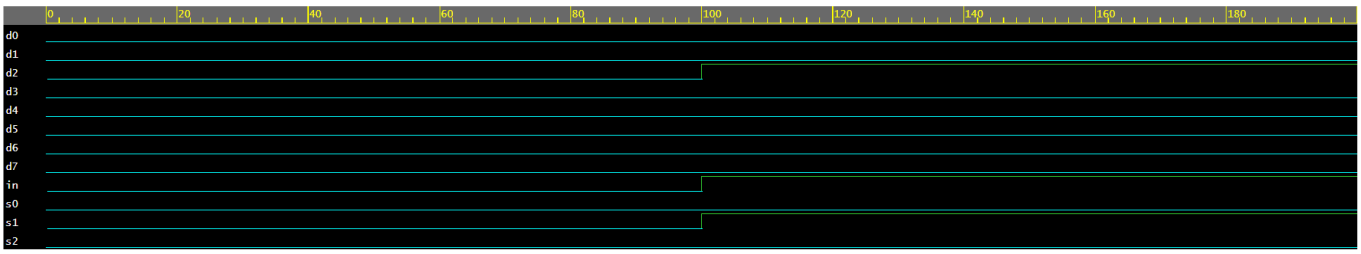
design.sv

```
module Demultiplexer(in,s0,s1,s2,d0,d1,d2,d3,d4,d5,d6,d7);
  input in,s0,s1,s2;
  output d0,d1,d2,d3,d4,d5,d6,d7;
  assign d0=(in & ~s2 & ~s1 &~s0),
    d1=(in & ~s2 & ~s1 &s0),
    d2=(in & ~s2 & s1 &~s0),
    d3=(in & ~s2 & s1 &s0),
    d4=(in & s2 & ~s1 &~s0),
    d5=(in & s2 & ~s1 &s0),
    d6=(in & s2 & s1 &~s0),
    d7=(in & s2 & s1 &s0);
endmodule
```

testbench.sv

SV/Verilog Testbench

```
module TestModule;
  reg in;
  reg s0;
  reg s1;
  reg s2;

  wire d0;
  wire d1;
  wire d2;
  wire d3;
  wire d4;
  wire d5;
  wire d6;
  wire d7;

  Demultiplexer d(.in(in), .s0(s0), .s1(s1), .s2(s2), .d0(d0), .d1(d1),
.d2(d2), .d3(d3), .d4(d4), .d5(d5), .d6(d6), .d7(d7));
  initial
    begin
      in = 0;
      s0 = 0;
      s1 = 0;
      s2 = 0;
      #100;
      in = 1;
      s0 = 0;
      s1 = 1;
      s2 = 0;
      #100;
    end
  initial
    begin
      $dumpfile("dump.vcd");
      $dumpvars(1);
    end
  endmodule
```



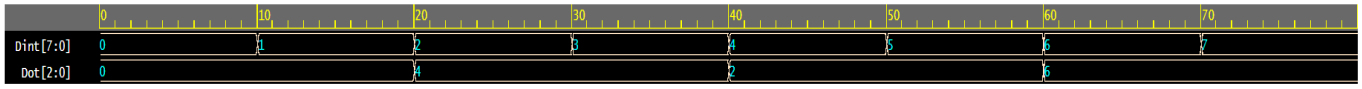
Question 5: Write a Verilog code to design a 3:8 Decoder and verify the same.

design.sv

```
module encoder(Do, Din);
  input [7:0]Din;
  output [2:0]Do;

  or(Do[0], Din[4], Din[5], Din[6], Din[7]);
  or(Do[1], Din[2], Din[3], Din[6], Din[7]);
  or(Do[2], Din[1], Din[3], Din[5], Din[7]);
endmodule

```

testbench.sv

```
module encoder_tb_v;
  reg [7:0] Dint;
  wire [2:0] Dot;

    encoder uut(.Do(Dot),.Din(Dint));
    initial
      begin
        Dint = 8'b00000000; #10;
        Dint = 8'b00000001; #10;
        Dint = 8'b00000010; #10;
        Dint = 8'b00000011; #10;
        Dint = 8'b00000100; #10;
        Dint = 8'b00000101; #10;
        Dint = 8'b00000110; #10;
        Dint = 8'b00000111; #10;
    end
  initial
    begin
      $dumpfile("dump.vcd");
      $dumpvars(1);
    end
endmodule
```



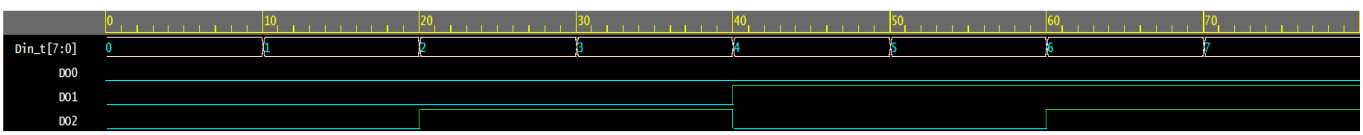
Question 6: Write a Verilog code to design a 8:3 Encoder and verify the same.

design.sv

```
module encoder83(din, do0, do1, do2);
  input [7:0]din;
  output do0, do1, do2;
  or(do0[0], din[4], din[5], din[6], din[7]);
  or(do1[1], din[2], din[3], din[6], din[7]);
  or(do2[2], din[1], din[3], din[5], din[7]);
endmodule
```

testbench.sv

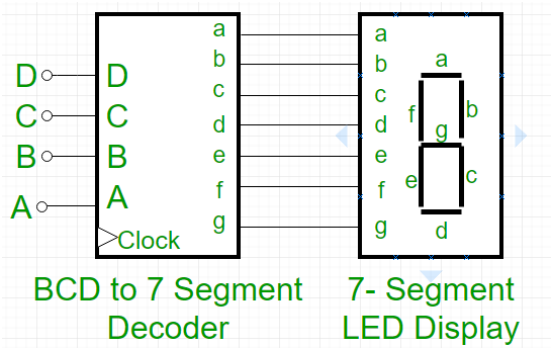
```
module encoder83_tb;
  reg [7:0] Din_t;
  wire DO0, DO1, DO2;
  encoder83 e1(.din(Din_t), .do0(DO0), .do1(DO1), .do2(DO2));
    initial
      begin
        Din_t = 8'b00000000; #10;
        Din_t = 8'b00000001; #10;
        Din_t = 8'b00000010; #10;
        Din_t = 8'b00000011; #10;
        Din_t = 8'b00000100; #10;
        Din_t = 8'b00000101; #10;
        Din_t = 8'b00000110; #10;
        Din_t = 8'b00000111; #10;
    end
  initial
    begin
      $dumpfile("dump.vcd");
      $dumpvars(1);
    end
endmodule
```



LAB 7: Application of Decoder, Recap of Logisim

Question 1: Write a Verilog code to implement BCD to seven segment display Decoder. Prepare the Truth Table and circuits and verify the same using Test Bench.

Decimal Number	INPUT Lines				OUTPUT Lines						
	A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	1	1

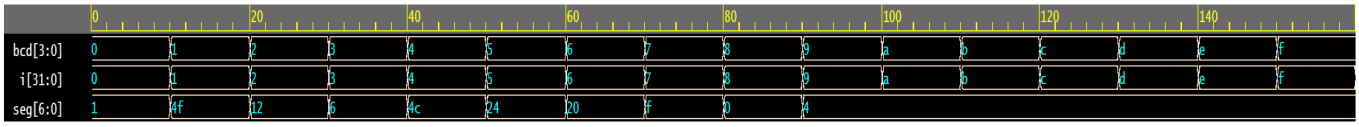


testbench.sv

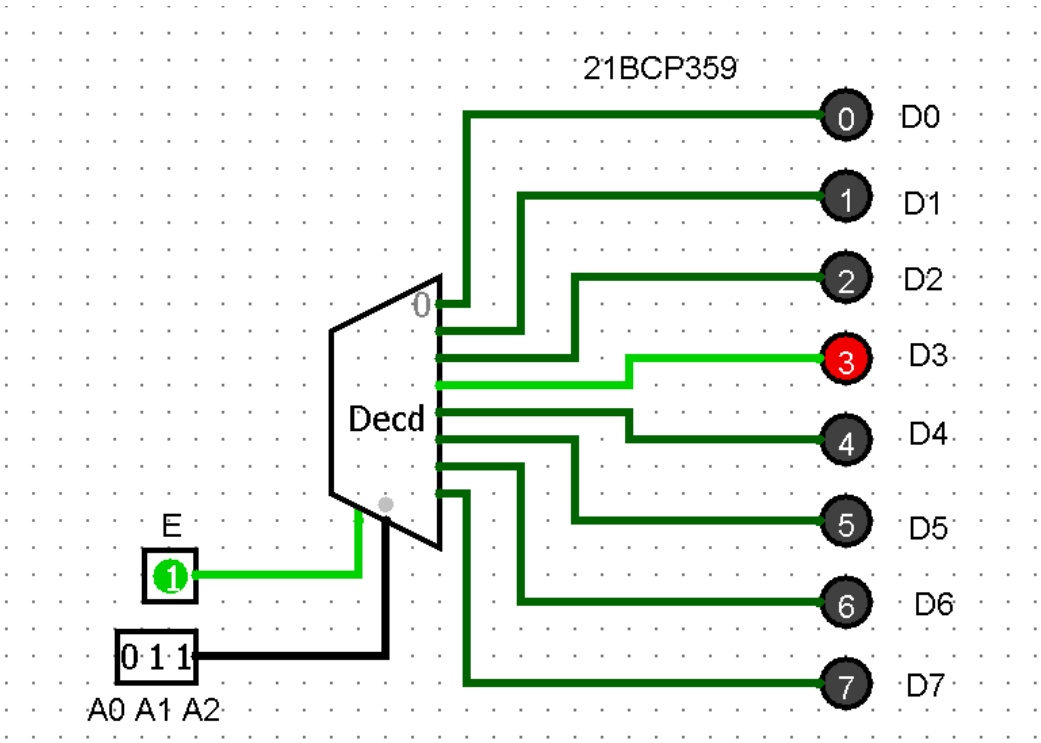
```
module tb_segment7;

    reg [3:0] bcd;
    wire [6:0] seg;
    integer i;

    segment7 uut (
        .bcd(bcd),
        .seg(seg)
    );

    initial begin
        for(i = 0;i < 16;i = i+1)
        begin
            bcd = i;
            #10;
        end
    end
  initial
    begin
      $dumpfile("dump.vcd");
      $dumpvars(1);
    end

endmodule
```

design.sv

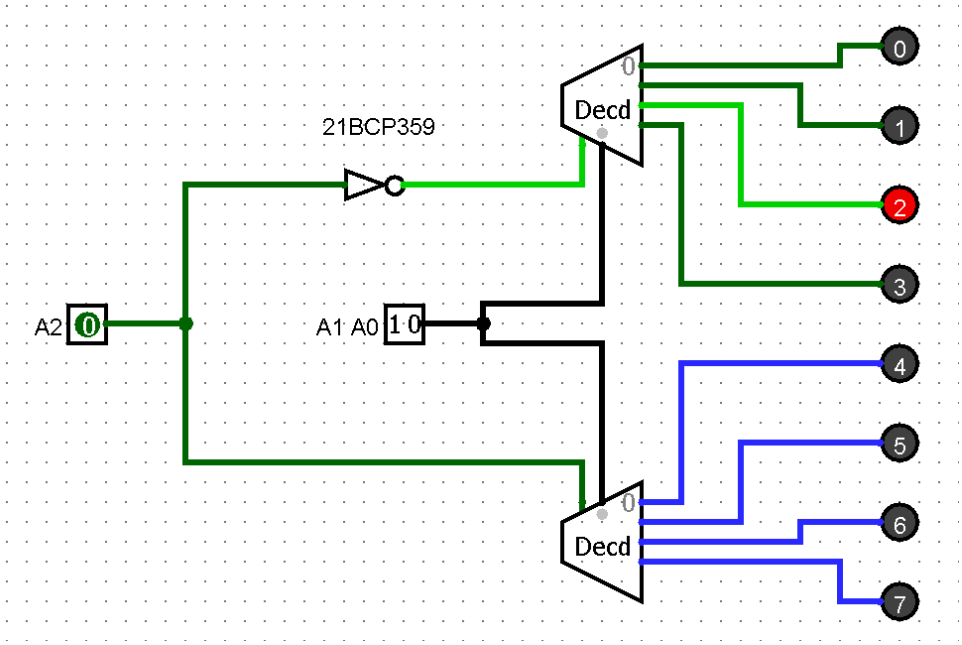
```
module segment7(
     bcd,
     seg
    );

     input [3:0] bcd;
     output [6:0] seg;
     reg [6:0] seg;

    always @(bcd)
    begin
        case (bcd)
            0 : seg = 7'b0000001;
            1 : seg = 7'b1001111;
            2 : seg = 7'b0010010;
            3 : seg = 7'b0000110;
            4 : seg = 7'b1001100;
            5 : seg = 7'b0100100;
            6 : seg = 7'b0100000;
            7 : seg = 7'b0001111;
            8 : seg = 7'b0000000;
            9 : seg = 7'b0000100;
        endcase
    end

endmodule
```



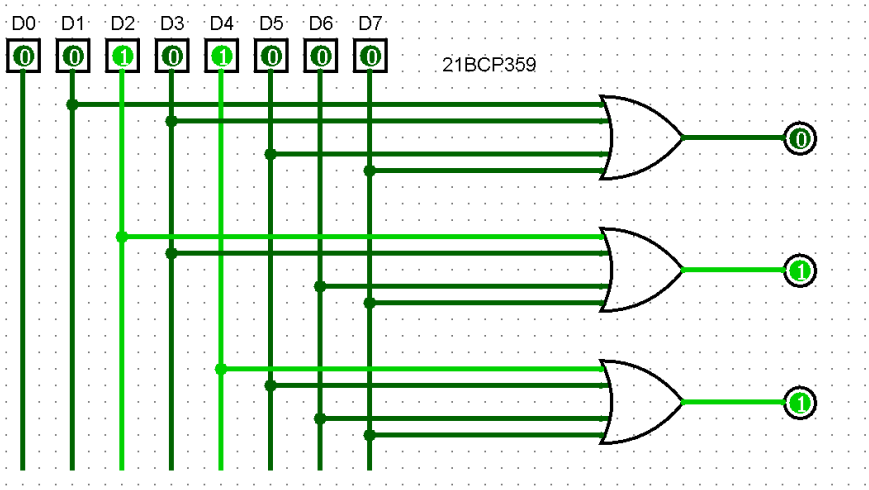
Question 2: Design a 3:8 Decoder using Logisim.



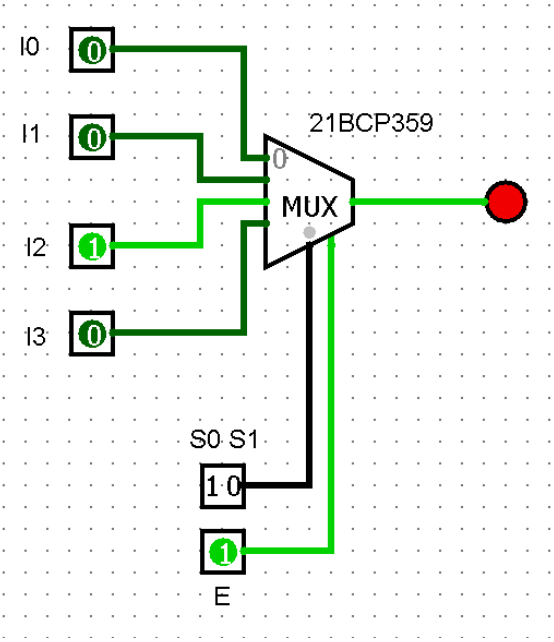
Question 3. Design a 3:8 Decoder using two 2:4 Decoder using Logisim.



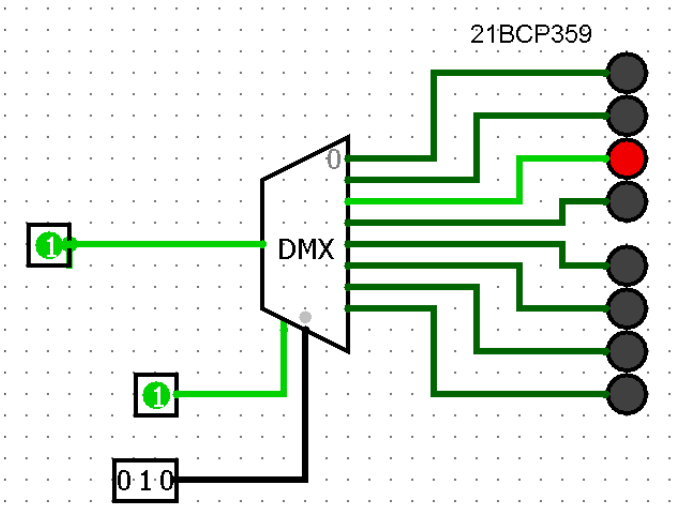
Question 4. Design an 8:3 Priority Encoder using Logisim.



Question 5. Design a 4:1 Multiplexer using Logisim.

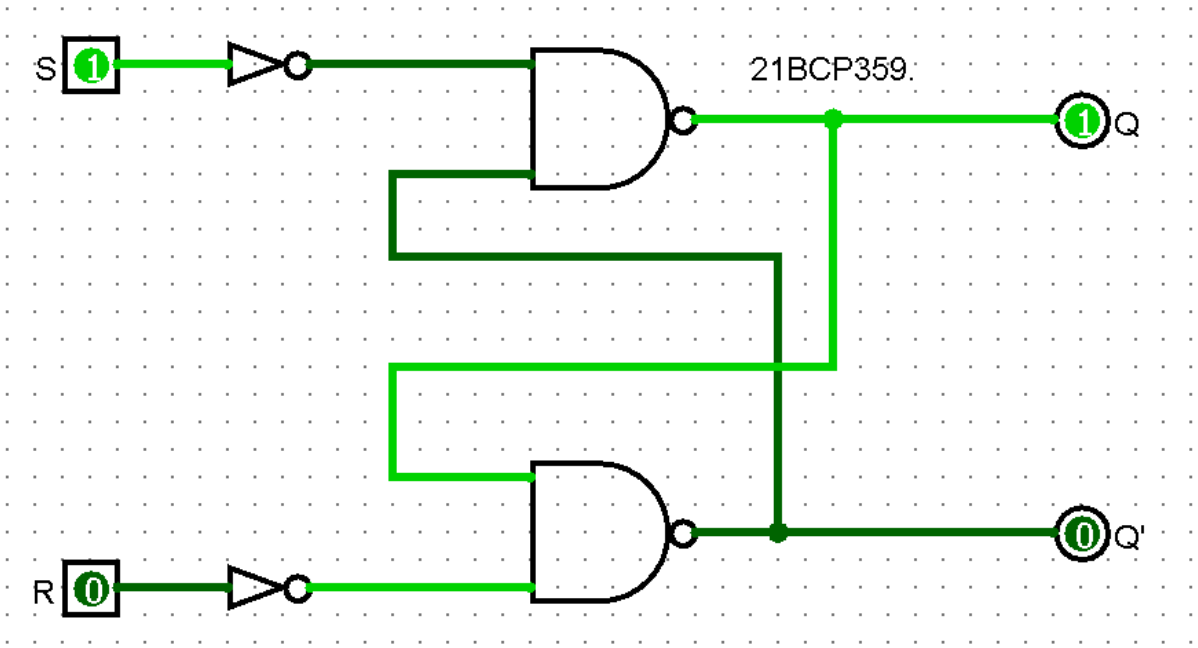


Question 6. Design a 1:8 Demultiplexer using Logisim



LAB 8: Sequential Circuits: Latch And SR Flip Flop

Question 1: Design an SR flip-flop using Logisim.



Question 2: Write a module for NOR gate and develop a structural Verilog code for the S-R latch using the NOR gate module. Validate the code via a suitable Testbench code.

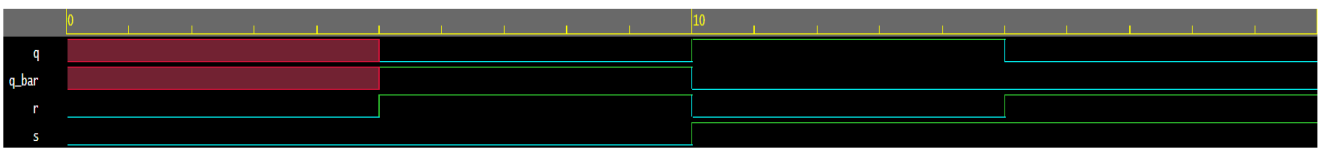
Expression: $Q_{n+1} = E.D + E'.Q_n$

testbench.sv

```
1  module nor_latch_tb;
2    reg s,r; //input
3    wire q,q_bar; //output
4    nor_latch x(.s(s),.r(r),.q(q),.q_bar(q_bar));
5    initial
6      begin
7        s<=1'b0; r<=1'b0; #5;
8        s<=1'b0; r<=1'b1; #5;
9        s<=1'b1; r<=1'b0; #5;
10       s<=1'b1; r<=1'b1; #5;
11       end
12   initial
13     begin
14       $dumpfile("dump.vcd");
15       $dumpvars(1);
16     end
17 endmodule
```

design.sv

```
1  // SR LATCH
2  module nor_latch(input s,r, output q,q_bar);
3    nor(q,r,q_bar);
4    nor(q_bar,q,s);
5  endmodule
```



Question 3: Write a Verilog module for NAND gate and utilize it to develop a structural Verilog module. Validate it using suitable Test bench.

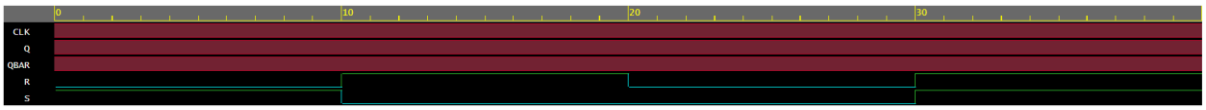
Expression: $Q_n+1 = S + Q_nR'$

testbench.sv

```
module SRflipfloptest;
  reg S, R, CLK;
  wire Q, QBAR;
  SRflipflop f1 (.s(S), .r(R), .clk(CLK), .q(Q), .qbar(QBAR));
  initial
    begin
      S = 1; R = 0; #10;
      S = 0; R = 1; #10;
      S = 0; R = 0; #10;
      S = 1; R = 1; #10;
    end
  initial
    begin
      $dumpfile("dump.vcd");
      $dumpvars(1);
    end
endmodule
```

design.sv

```
module SRflipflop(q, qbar, s, r, clk);

input s, r, clk;
output q, qbar;

assign q = clk ? (s + ((~r) & q)) : q;
assign qbar = ~q;
endmodule
```



Q 4. Develop a similar behavioral code and test bench for S-R flip flop using if-else condition as per question 3.

testbench.sv

```
// 21BCP294 -Ayush Thakor
// SR -Flip flop behavioral code
module tb_srff_ifelse;
reg S,R, CLK;
wire Q, QBAR;
srff_ifelse dut(S,R,CLK,Q,QBAR);
initial
begin
S= 0; R= 0; CLK=0; #1;
$display ("CLK = %b, S= %b, R = %b , Q = %b , QBAR = %b", CLK,S,R, Q, QBAR);
S= 0; R= 1; CLK=1; #1;
$display ("CLK = %b, S= %b, R = %b , Q = %b , QBAR = %b", CLK,S,R, Q, QBAR);
S= 0; R= 0; CLK=0; #1;
$display ("CLK = %b, S= %b, R = %b , Q = %b , QBAR = %b", CLK,S,R, Q, QBAR);
S= 1; R= 0; CLK=1; #1;
$display ("CLK = %b, S= %b, R = %b , Q = %b , QBAR = %b", CLK,S,R, Q, QBAR);
S= 0; R= 0; CLK=0; #1;
$display ("CLK = %b, S= %b, R = %b , Q = %b , QBAR = %b", CLK,S,R, Q, QBAR);
S= 1; R= 1; CLK=1; #1;
$display ("CLK = %b, S= %b, R = %b , Q = %b , QBAR = %b", CLK,S,R, Q, QBAR);
$finish;
end
initial
begin
$dumpfile("dump.vcd");
$dumpvars (1);
end
```

design.sv

```
// 21BCP294 -Ayush Thakor
module srff_ifelse(S, R, CLK, Q, QBAR);
input S, R, CLK;
output reg Q, QBAR;
always@(CLK)
begin
if(S==1 & R==0)
begin
Q = 1;
QBAR = 0;
end
else if(S==0 & R==1)
begin
Q = 0;
QBAR = 1;
end
else if(S==0 & R==0)
begin
Q = Q;
QBAR = QBAR;
end
else if(S==1 & R==1)
begin
Q = 1;
QBAR = 1;
end
end
endmodule
```


```
CLK = 0, S= 0, R = 0 , Q = x , QBAR = x
CLK = 1, S= 0, R = 1 , Q = 0 , QBAR = 1
CLK = 0, S= 0, R = 0 , Q = 0 , QBAR = 1
CLK = 1, S= 1, R = 0 , Q = 1 , QBAR = 0
CLK = 0, S= 0, R = 0 , Q = 1 , QBAR = 0
CLK = 1, S= 1, R = 1 , Q = 1 , QBAR = 1
```

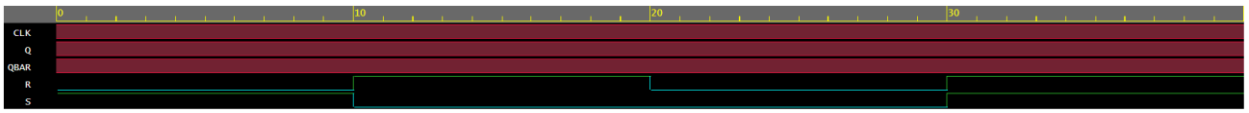
Question 5: Develop a similar behavioural code and test bench for S-R flip flop using if-else condition as per question 3.

testbench.sv

```
module SRflipfloptest;
  reg S, R, CLK;
  wire Q, QBAR;
  SRflipflop f1 (.s(S), .r(R), .clk(CLK), .q(Q), .qbar(QBAR));
  initial
    begin
      S = 1; R = 0; #10;
      S = 0; R = 1; #10;
      S = 0; R = 0; #10;
      S = 1; R = 1; #10;
    end
  initial
    begin
      $dumpfile("dump.vcd");
      $dumpvars(1);
    end
endmodule
```

design.sv

```
module SRflipflop(s, r, clk, q , qbar);
  input clk, s, r;
  output reg q, qbar;
  always@(posedge clk)
    begin
      if(s == 1)
        begin
          q <= 1;
          qbar <= 0;
        end
      else if(r == 1)
        begin
          q <= 0;
          qbar <= 1;
        end
      else if(s == 0 & r == 0)
        begin
          q <= q;
          qbar = qbar;
        end
    end
endmodule
```



LAB 9: Flip Flops (JK, D, T)

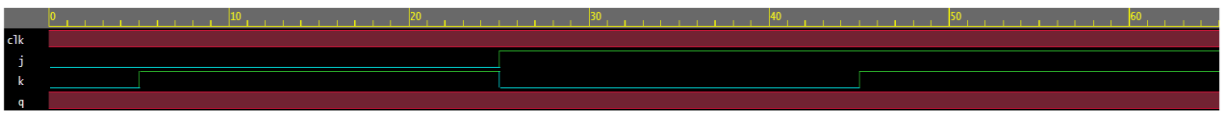
Question 1: Write a Verilog code to implement J-K flip flop and validate the code via a suitable Test bench code.

design.sv

```
module jk_ff ( input j, input k, input clk, output q);
  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
```

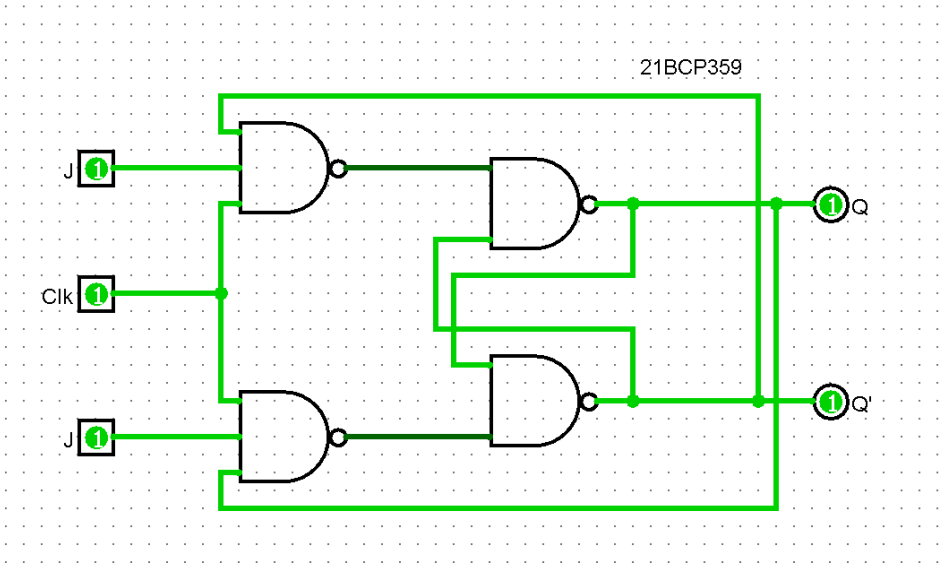
testbench.sv

```
module tb_jk;
  reg j;
  reg k;
  reg clk;
  always #5 clk = ~clk;
  jk_ff jk0 ( .j(j), .k(k), .clk(clk), .q(q));

  initial
    begin
      j <= 0;
      k <= 0; #5

      j <= 0;
      k <= 1; #20

      j <= 1;
      k <= 0; #20

      j <= 1;
      k <= 1; #20
      $finish;
    end
  initial
    begin
      $dumpfile("dump.vcd");
      $dumpvars(1);
    end

  initial
    $monitor ("j=%0d k=%0d q=%0d", j, k, q);
endmodule
```




```
[2022-11-14 09:32:17 EST] iverilog '-Wall' design.sv testbench.sv  && unbuffer vvp a.out
testbench.sv:6: warning: implicit definition of wire logic tb_jk.q.
VCD info: dumpfile dump.vcd opened for output.
j=0 k=0 q=x
j=0 k=1 q=x
j=1 k=0 q=x
j=1 k=1 q=x
Finding VCD file...
./dump.vcd
[2022-11-14 09:32:17 EST] Opening EPWave...
Done
```

Question 2: Design a J-K flip flop in Logisim and validate the circuit.



Clock	J	K	Q_{n+1}	State
0	x	x	Q_n	
1	0	0	Q_n	Hold
1	0	1	0	Reset
1	1	1	1	Set
1	1	1	$\overline{Q}_n$	Toggle

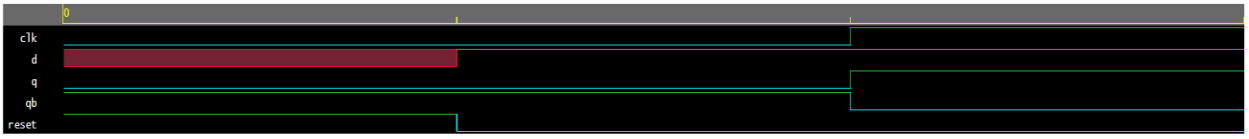
Question 3: Write a Verilog code to implement D flip flop and validate the code via a suitable Test bench code.

design.sv

```
// Design
// D flip-flop
module dff (clk, reset,
  d, q, qb);
  input      clk;
  input      reset;
  input      d;
  output     q;
  output     qb;

  reg        q;

  assign qb = ~q;

  always @(posedge clk or posedge reset)
  begin
    if (reset) begin
      // Asynchronous reset when reset goes high
      q <= 1'b0;
    end else begin
      // Assign D to Q on positive clock edge
      q <= d;
    end
  end
endmodule
```

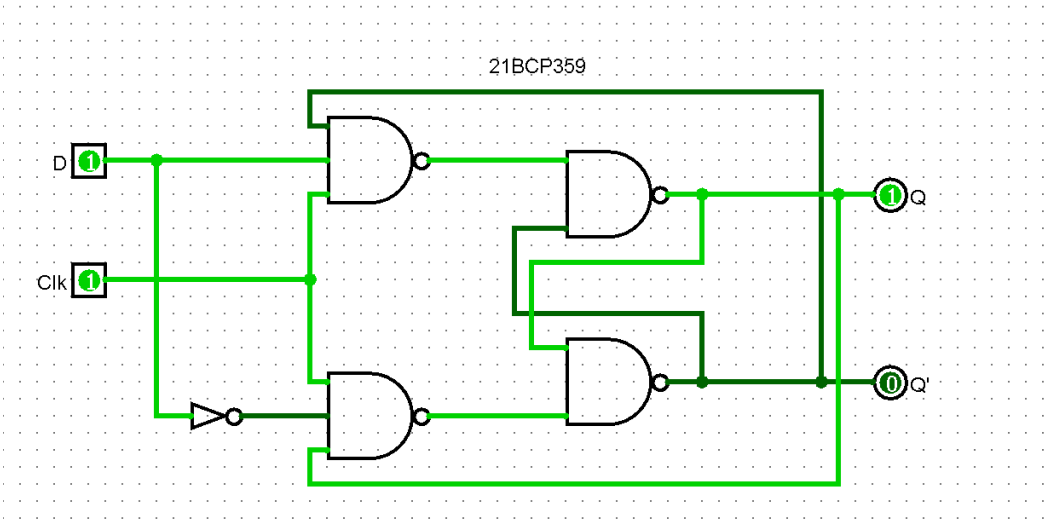
testbench.sv

```
// Testbench
module test;

  reg clk;
  reg reset;
  reg d;
  wire q;
  wire qb;

  // Instantiate design under test
  dff DFF(.clk(clk), .reset(reset),
          .d(d), .q(q), .qb(qb));

  initial begin
    // Dump waves
    $dumpfile("dump.vcd");
    $dumpvars(1);

    $display("Reset flop.");
    clk = 0;
    reset = 1;
    d = 1'bx;
    display;

    $display("Release reset.");
    d = 1;
    reset = 0;
    display;

    $display("Toggle clk.");
    clk = 1;
    display;
  end

  task display;
    #1 $display("d:%0h, q:%0h, qb:%0h",
      d, q, qb);
  endtask

endmodule
```




```
[2022-11-14 09:37:27 EST] iverilog '-Wall' design.sv testbench.sv  && unbuffer vvp a.out
VCD info: dumpfile dump.vcd opened for output.
Reset flop.
d:x, q:0, qb:1
Release reset.
d:1, q:0, qb:1
Toggle clk.
d:1, q:1, qb:0
Finding VCD file...
./dump.vcd
[2022-11-14 09:37:28 EST] Opening EPWave...
Done
```

Question 4: Design a D flip flop in Logisim and validate the circuit.



S	R	Q	Q`
0	0	0	1
0	1	0	1
1	0	1	0
1	1	∞	∞

Question 5: Write a Verilog code to implement T flip flop and validate the code via a suitable Test bench code.

testbench.sv

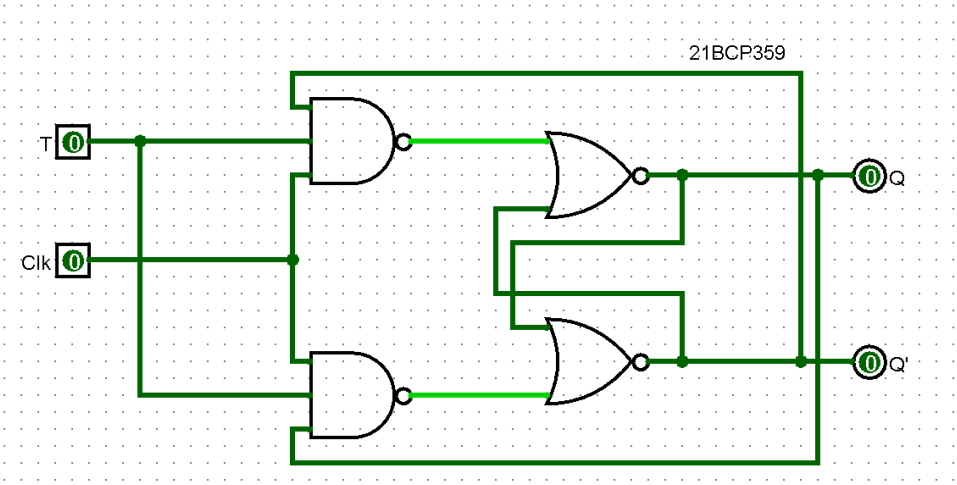
```
module tb;
  reg clk;
  reg rstn;
  reg t;

  tff u0 (  .clk(clk),
            .rstn(rstn),
            .t(t),
           .q(q));

  always #5 clk = ~clk;

  initial
    begin
    {rstn, clk, t} <= 0;

    $monitor ("T=%0t rstn=%0b t=%0d q=%0d", $time, rstn, t, q);
    repeat(2) @(posedge clk);
    rstn <= 1;
    end
  initial
    begin
      $dumpfile("dump.vcd");
      $dumpvars(1);
  end
endmodule
```

design.sv

```
module tff ( input clk, input rstn, input t, output reg q);

  always @ (posedge clk) begin
    if (!rstn)
      q <= 0;
    else
        if (t)
            q <= ~q;
        else
            q <= q;
  end
endmodule
```

```
[2022-11-15 00:51:05 EST] iverilog '-Wall' design.sv testbench.sv  && unbuffer vvp a.out
testbench.sv:9: warning: implicit definition of wire logic tb.q.
VCD info: dumpfile dump.vcd opened for output.
T=0 rstn=0 t=0 q=x
T=5 rstn=0 t=0 q=0
T=15 rstn=1 t=0 q=0
```

Question 6: Design a T flip flop in Logisim and validate the circuit.



Truth table			
CLK	T	Q_{next}	Comment
Rising edge	0	Q	Hold state
Falling edge	0	Q	Hold state
Rising edge	1	$\overline{Q}$	Toggle
Falling edge	1	Q	No change

Q_{next} - "after the clock transition" output
Q - the current output

Lab 10: Flip Flops Conversion

Question 1: Modify the S-R flip flop created in last lab into JK flip flop. Verify the functionality of J-K flip flop using suitable Testbench.

Conversion Table

J-K Inputs		Outputs		S-R Inputs	
J	K	Qp	Qp+1	S	R
0	0	0	0	0	X
0	0	1	1	X	0
0	1	0	0	0	X
0	1	1	0	0	1
1	0	0	1	1	0
1	0	1	1	X	0
1	1	0	1	1	0
1	1	1	0	0	1

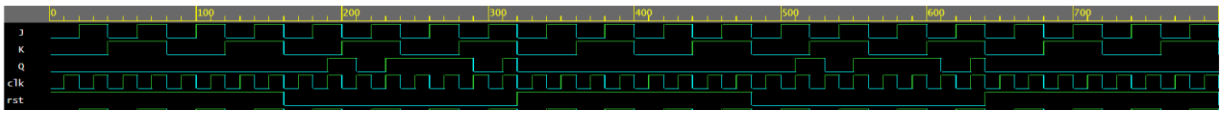
Conversion Expression: S = J'Q and R = KQ

testbench.sv

```
module FLOP();
  reg clk,J,K,rst;
  wire Q;

  flop f_1(.clk(clk),.J(J),.K(K),.rst(rst),.Q(Q));

initial begin
  clk=0;
  J=0;
  K=0;
  rst=1;
end

  initial forever #10 clk=~clk;
  initial forever #20 J=~J;
  initial forever #40 K=~K;
  initial forever #160 rst=~rst;

  initial begin
    #800 $finish;
  end

  initial begin
    $dumpfile("flop.vcd");
    $dumpvars;
  end
endmodule
```

design.sv

```
module flop(input clk,J,K,rst, output Q);
  reg ff1,ff2;

  always @(rst) begin
    ff1 <= 1'b0;
    ff2 <= 1'b0;
  end

  always @(posedge clk) begin
    if (!rst) begin
      if (J==1 && K==0) begin
        ff1 <= 1'b1;
      end
      else if (J==0 && K==1) begin
        ff1 <= 1'b0;
      end
      else if (J==1 && K==1) begin
        ff1 <=~ff1;
      end
      else begin
        ff1 <=ff1;
      end
    end
  end
  assign Q=ff1;
endmodule
```



Question 2: Develop the behavioural Verilog module of D flip-flop, converting it from a J-K flip flop. You may utilize if-else statements to develop your Verilog code. Verify the functionality via a suitable test bench code.

Conversion Table

D Input	Outputs		J-K Inputs	
	Qp	Qp+1	J	K
0	0	0	0	X
0	1	0	X	1
1	0	1	1	X
1	1	0	X	0

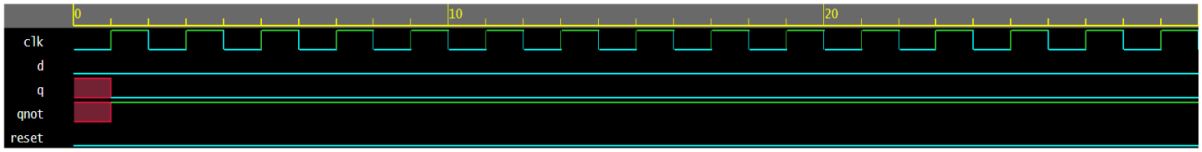
Conversion Expression: J = D and K = D'

testbench.sv

```
module tb_jkff

  reg clk=0;
  rreg d=0;
  reg reset=1;
  wire q, qnot;

  jkff dut(reset, clk, d, q, qnot);

  initial
    begin
      $dumpfile("dump.vcd);
      $dumpvars(1);
      d = 1'b0;
      reset 1'b0;
      #5
      #25
      $finish;
    end
  always #1 clk = ~clk;
endmodule
```

design.sv

```
module jkff (input reset, input clk, input
d, output reg q, output qnot);
  wire j,k;
  assign qnot = ~q;
  assign j=d;
  assign k = ~d;
  always @(posedge clk)
    if(reset) q<=1'b0; else
      case({d})
        2'b0 : q<=1'b0;
        2'b1 : q<=1'b1;
      endcase
endmodule
```



Question 3: Develop a behavioural Verilog module for JK flip flop using case statements. Modify it to act like a T flip flop. Validate the modification via a suitable test bench.

Conversion Expression: J = T and K = T

Conversion Table

T Input	Outputs Qp	Outputs Qp+1	J-K Inputs J	J-K Inputs K
0	0	0	0	X
0	1	1	X	0
1	0	1	1	X
1	1	0	X	1

testbench.sv

```
module TFLOP();
  reg clk,T,rst;
  wire Q;

  TFF tff_1(.clk(clk),.T(T),.rst(rst),.Q(Q));

  initial begin
    clk=0;
    T=0;
    rst=1;
  end

  initial forever #10 clk=~clk;
  initial forever #20 T=~T;
  initial forever #160 rst=~rst;

  initial begin
    #800 $finish;
  end

  initial begin
    $dumpfile("TFLOP.vcd");
    $dumpvars;
  end
endmodule
```

design.sv

```
module flop(input clk,J,K,rst, output Q);
  reg ff1,ff2;

  always @(rst) begin
    ff1 <= 1'b0;
    ff2 <= 1'b0;
  end

  always @(posedge clk) begin
    if (!rst) begin
      if (J==1 && K==0) begin
        ff1 <= 1'b1;
      end
      else if (J==0 && K==1) begin
        ff1 <= 1'b0;
      end
      else if (J==1 && K==1) begin
        ff1 <=~ff1;
      end
      else begin
        ff1 <=ff1;
      end
    end
  end
  assign Q=ff1;
endmodule

module TFF(input T,clk,rst,output Q);

  flop TFF_1(.J(T),.K(T),.clk(Clk),.rst(rst),.Q(Q));
endmodule
```