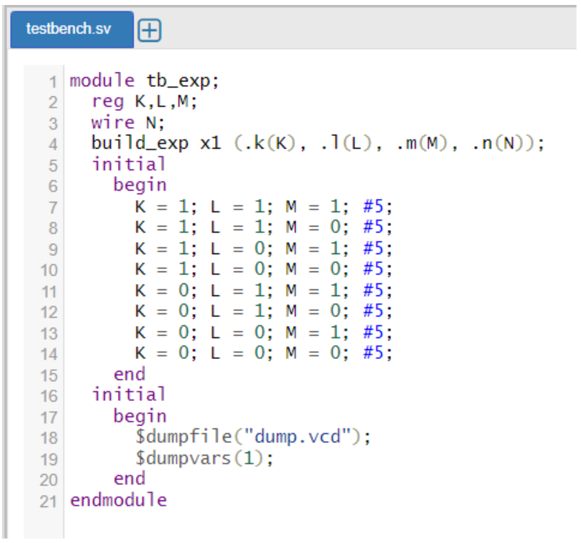
23/08/2022 21BCP359

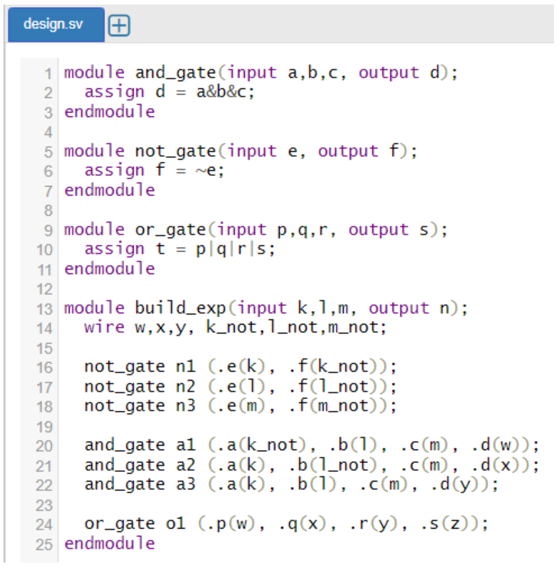
Lab 3: Implementation of Structural & Behavioral Verilog Code

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

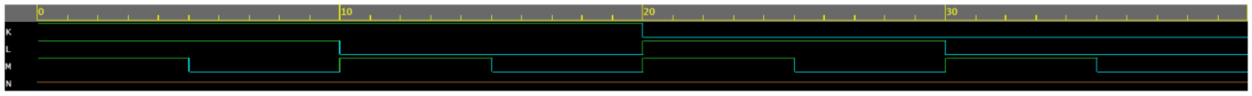
a. F(A, B, C)= A’BC + AB’C + ABC


testbench.sv

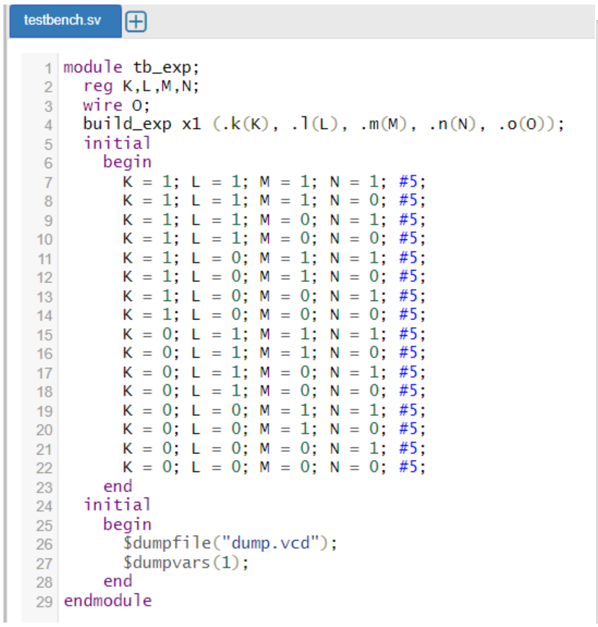
```
module tb_exp;
  reg K,L,M;
  wire N;
  build_exp x1 (.k(K), .l(L), .m(M), .n(N));
  initial
    begin
      K = 1; L = 1; M = 1; #5;
      K = 1; L = 1; M = 0; #5;
      K = 1; L = 0; M = 1; #5;
      K = 1; L = 0; M = 0; #5;
      K = 0; L = 1; M = 1; #5;
      K = 0; L = 1; M = 0; #5;
      K = 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

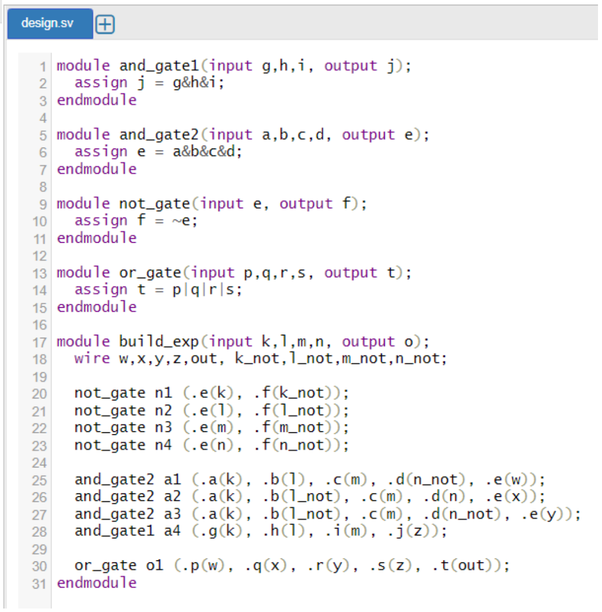
```
module and_gate(input a,b,c, output d);
  assign d = a&b&c;
endmodule

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
```



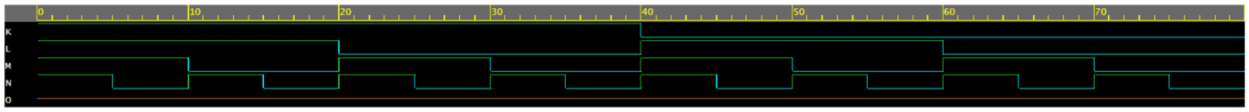
b. F(A, B, C, D)=ABCD’ + A’BC D+ AB’CD’ + ABC


testbench.sv

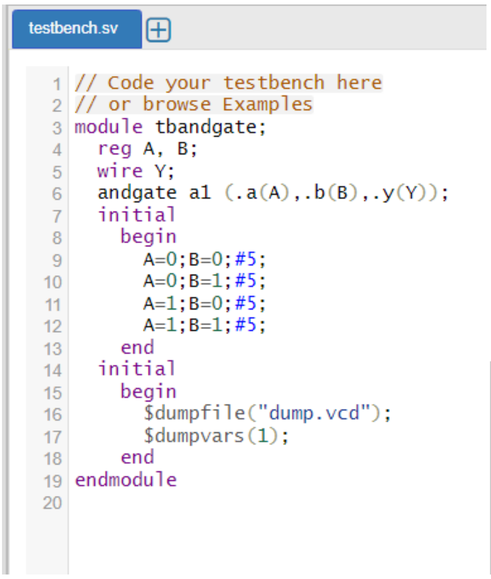
```
module tb_exp;
  reg K,L,M,N;
  wire O;
  build_exp x1 (.k(K), .l(L), .m(M), .n(N), .o(O));
  initial
    begin
      K = 1; L = 1; M = 1; N = 1; #5;
      K = 1; L = 1; M = 1; N = 0; #5;
      K = 1; L = 1; M = 0; N = 1; #5;
      K = 1; L = 1; M = 0; N = 0; #5;
      K = 1; L = 0; M = 1; N = 1; #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
```

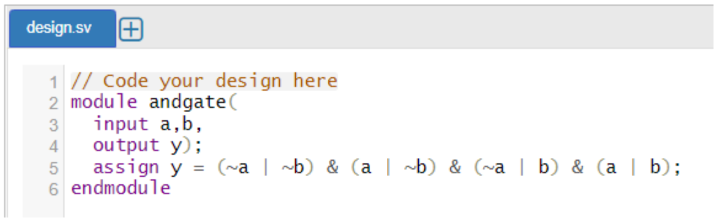


c. F(A, B) = (A’+B’) (A+B’) (A’+B) (A+B)



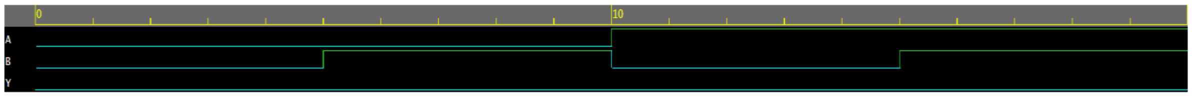
testbench.sv

```
// 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
```

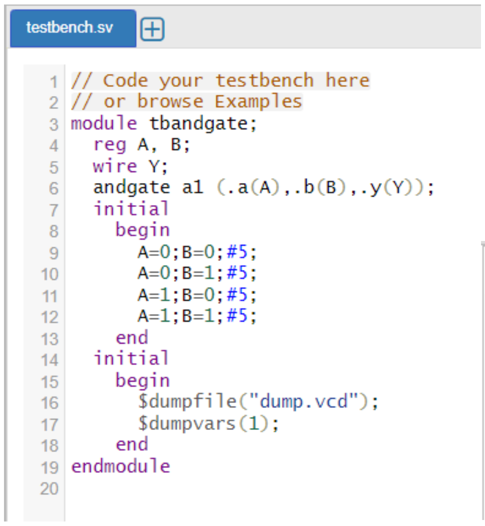


design.sv

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

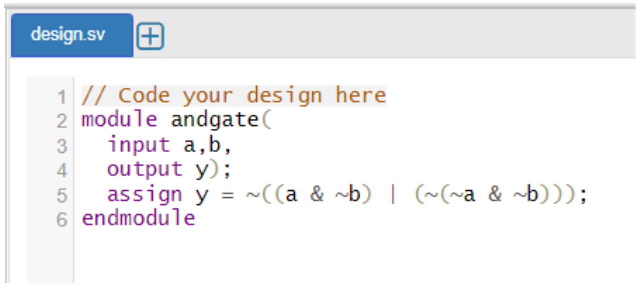


d. (A, B) = ((A.B’)’+((A)’(B)’)’)’



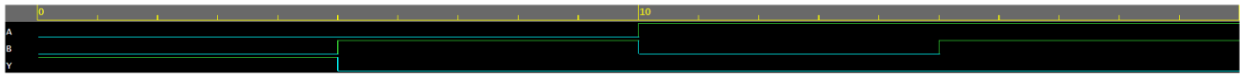
testbench.sv

```
// 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
```

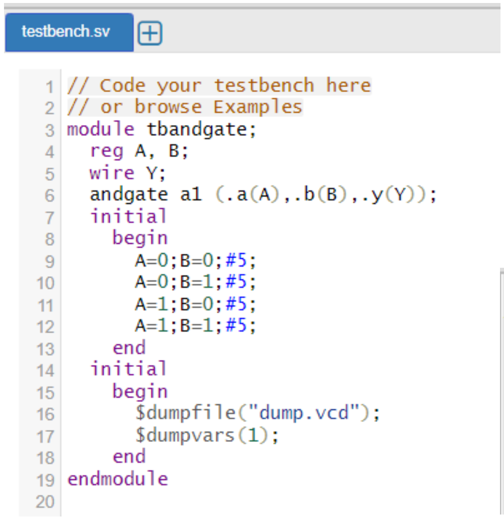


design.sv

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

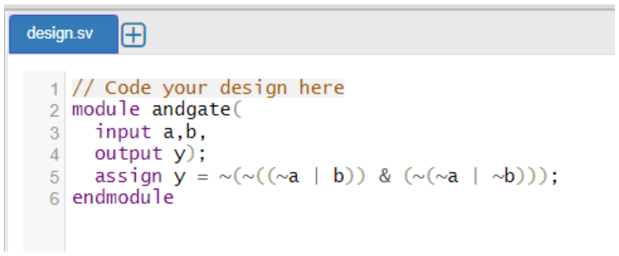


e. (A, B) = (((A)’+B)’. ((A)’+(B)’)’)’



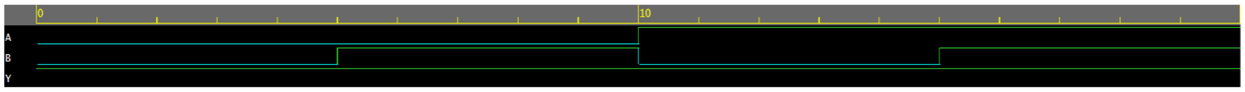
testbench.sv

```
// 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
```

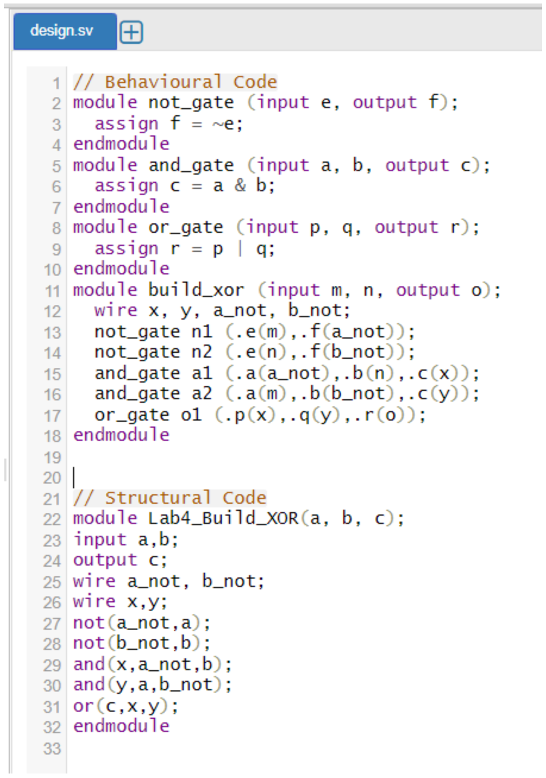


design.sv

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

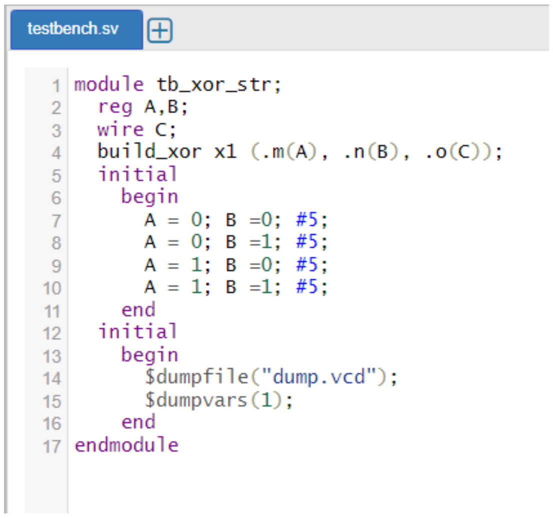


Question 2: Implement the XOR gate using Behavioral and Structural code of Verilog Hardware Description Language.



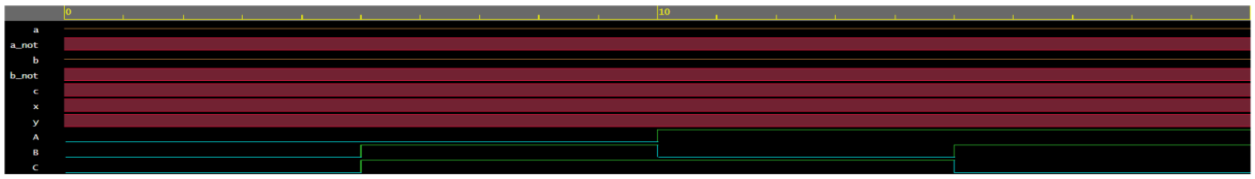
design.sv

```
// 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
```



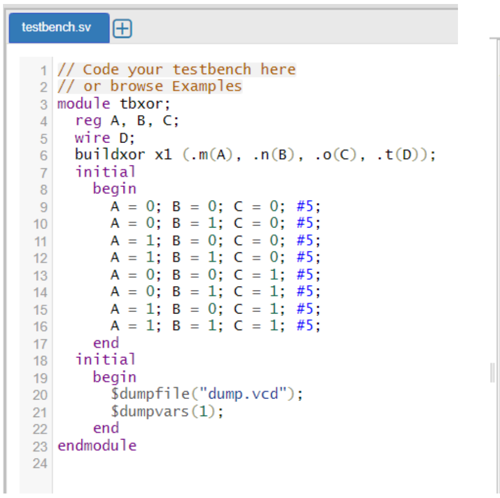
testbench.sv

```
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
```

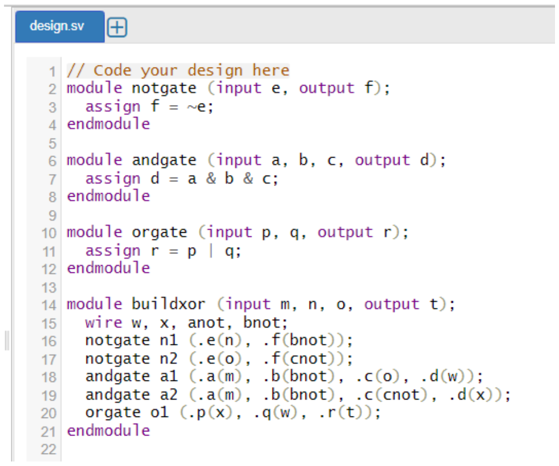


Question 3: Implement the following expression using the Verilog Hardware Description Language (HDL) (Structural Coding).

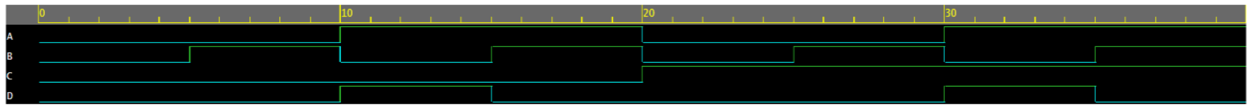
a. F (A, B, C) = (A'+B'+C')(A+B'+C')(A'+B+C')(A'+B')`



```
// 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
```

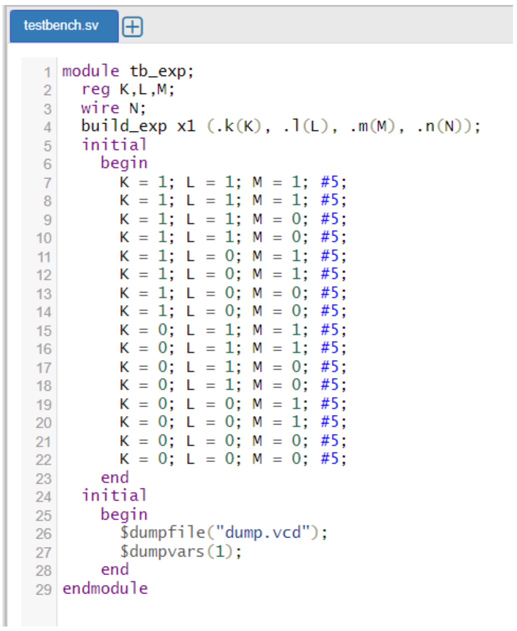


```
// Code your design here
module notgate (input e, output f);
  assign f = ~e;
endmodule

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
```

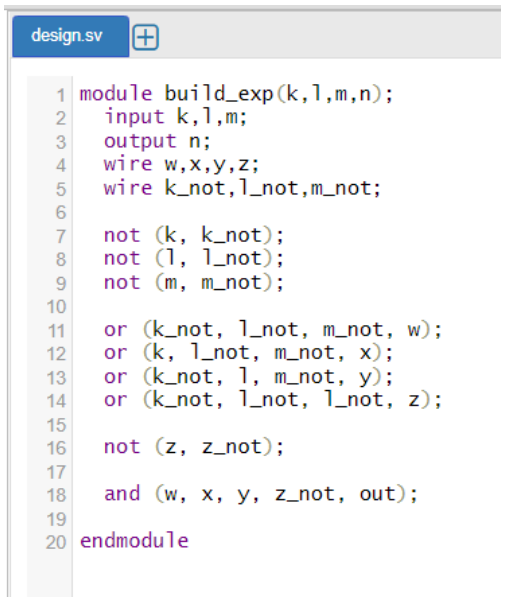


Question 3: Implement the following expression using universal NAND and NOR gate. Write down Verilog Structural and Behavioral code for that expression.

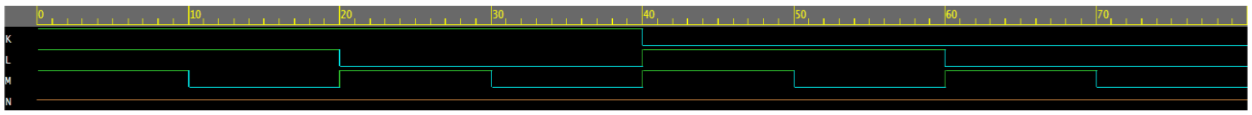
a. F (A, B, C) = (AB'C) + (AB'C')



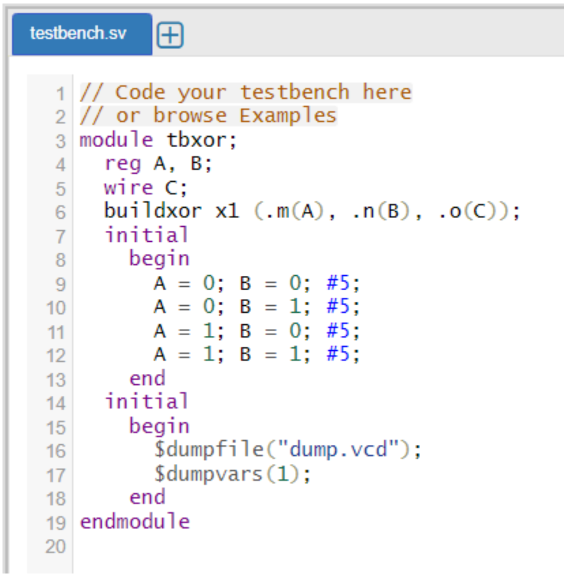
```
module tb_exp;
  reg K,L,M;
  wire N;
  build_exp x1 (.k(K), .l(L), .m(M), .n(N));
  initial
    begin
      K = 1; L = 1; M = 1; #5;
      K = 1; L = 1; M = 1; #5;
      K = 1; L = 1; M = 0; #5;
      K = 1; L = 1; M = 0; #5;
      K = 1; L = 0; M = 1; #5;
      K = 1; L = 0; M = 1; #5;
      K = 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
```



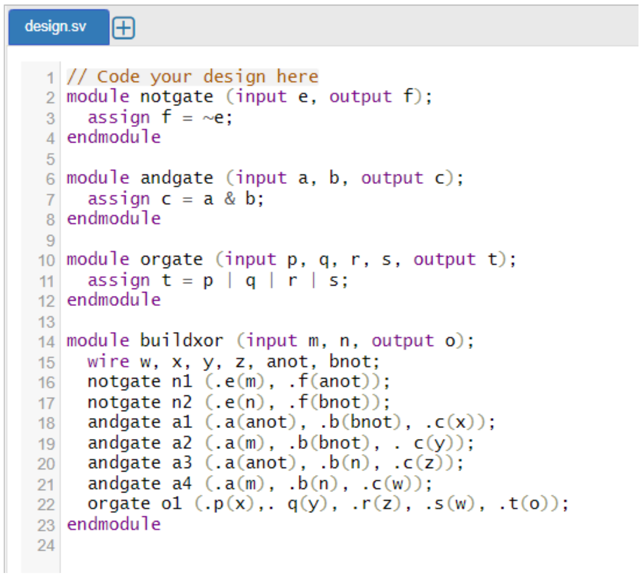
```
module build_exp(k,l,m,n);
  input k,l,m;
  output n;
  wire w,x,y,z;
  wire k_not,l_not,m_not;

  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
```



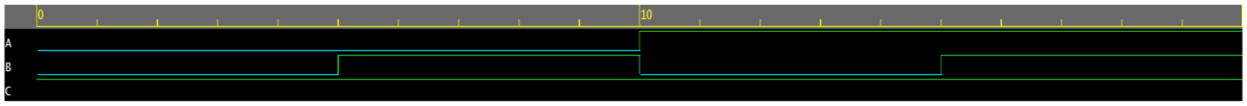
b. F (A, B) = A’B’ + AB’ + A’B + AB



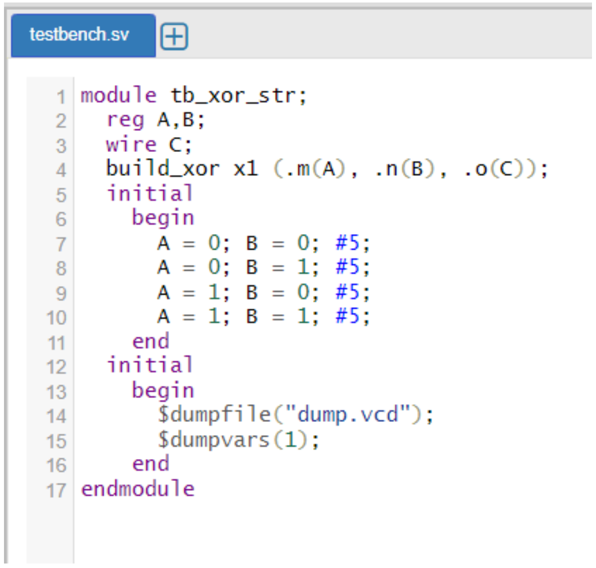
```
// Code your testbench here
// or browse Examples
module tbxor;
  reg A, B;
  wire C;
  buildxor x1 (.m(A), .n(B), .o(C));
  initial
    begin
      A = 0; B = 0; #5;
      A = 0; B = 1; #5;
      A = 1; B = 0; #5;
      A = 1; B = 1; #5;
    end
  initial
    begin
      $dumpfile("dump.vcd");
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



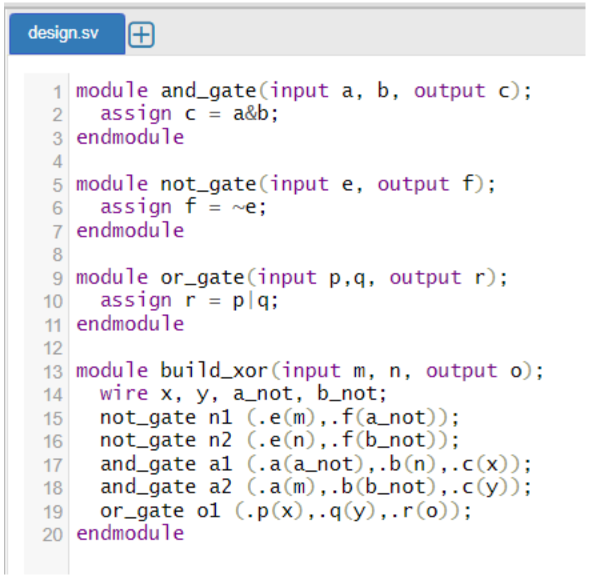
```
// Code your design here
module notgate (input e, output f);
  assign f = ~e;
endmodule

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
```



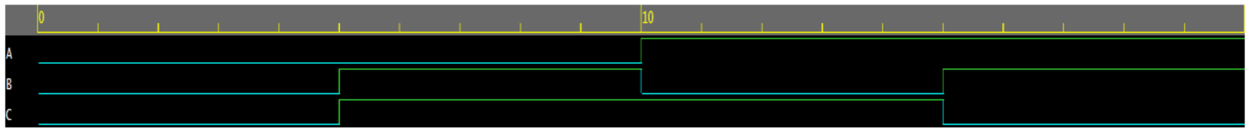
Question 4 Write down three modules in a single Verilog code to design AND, OR and NOT gate. Now use those modules to design the XOR gate. Use AND, OR and NOT gate as the instances to implement the XOR gate. Write the corresponding Testbench code for the verification of your XOR gate.



```
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
```

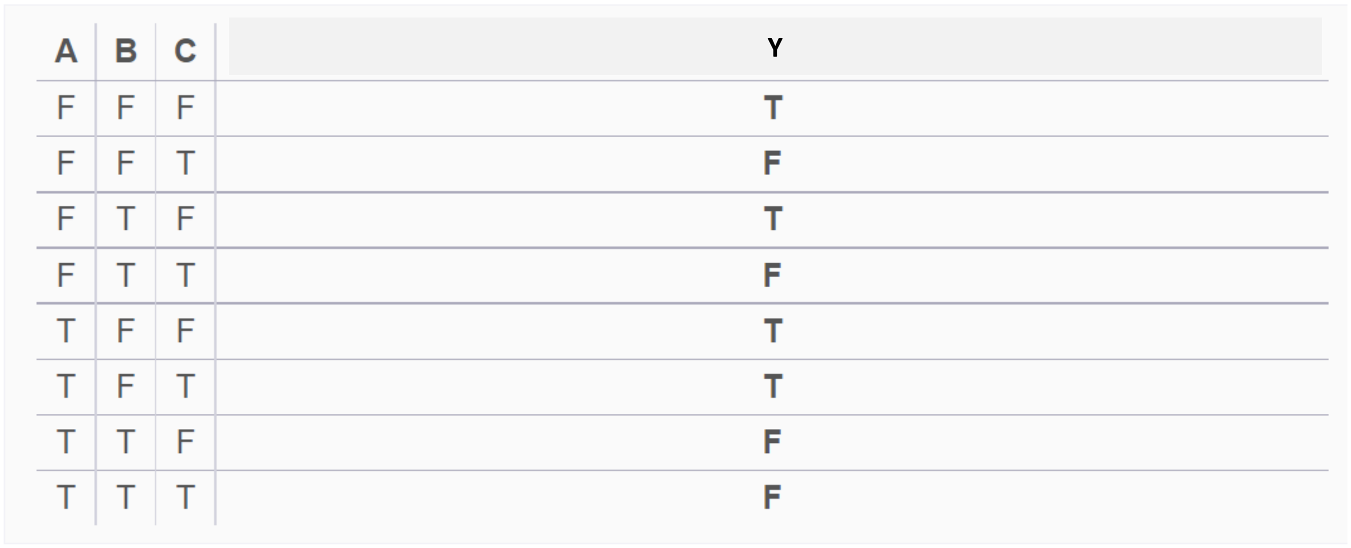


```
module and_gate(input a, b, output c);
  assign c = a&b;
endmodule

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
```



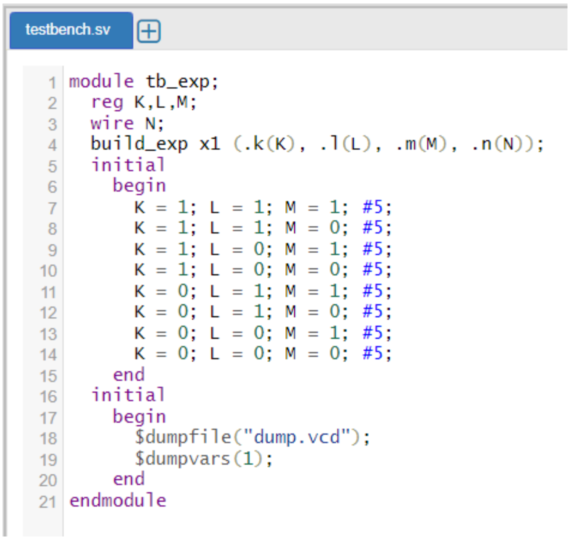
Question 5 Consider the following expression: Y = A'.B'.C' + A'.B.C' + A.B'.C' + A.B'.C

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.



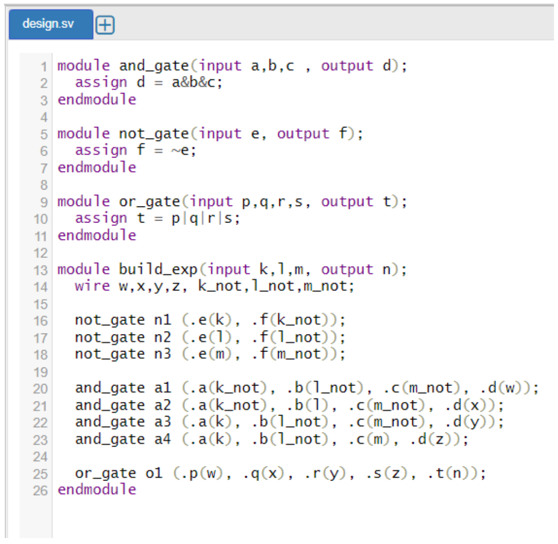
A	B	C	Y
F	F	F	T
F	F	T	F
F	T	F	T
F	T	T	F
T	F	F	T
T	F	T	T
T	T	F	F
T	T	T	F

Reduced Exression: A'.B'.C'



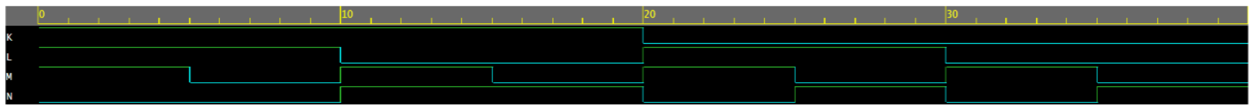
testbench.sv

```
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
```


0
10
20
30
K
L
M
N