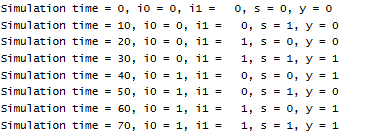
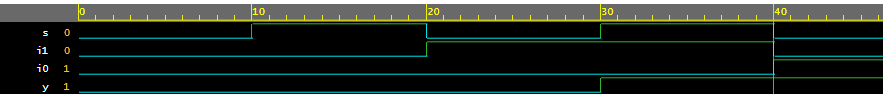
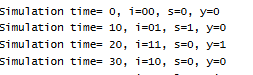
**P1). Gate level code for 2:1 MUX**

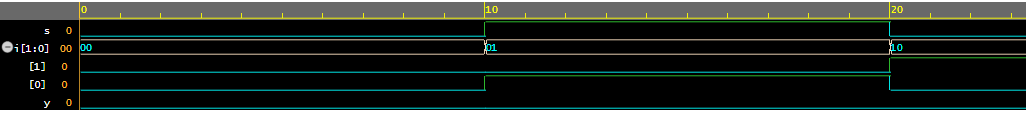
|  |  |
| --- | --- |
| ***Design code:***  module mux(i1, i0, s, y);  input i1, i0, s;  output y;  wire w1, w2, w3;  not n1 (w1, s);  and al (w2, i0, w1);  and a2 (w3, s, i1);  or o1 (y, w2, w3);  endmodule | ***Test bench code:***  module mux\_test;  reg i0, i1, s;  wire y;  mux21 dut(.i0(i0), .i1(i1), .s(s), .y(y));    initial begin  i0=0; i1=0; s=0;  #10 i0=0; i1=0; s=1;  #10 i0=0; i1=1; s=0;  #10 i0=0; i1=1; s=1;  #10 i0=1; i1=0; s=0;  #10 i0=1; i1=0; s=1;  #10 i0=1; i1=1; s=0;  #10 i0=1; i1=1; s=1;  end  initial begin  $monitor("Simulation time = %0t, i0 = %b, i1 = %b, s = %b, y = %b", $time, i0, i1, s, y);  end  initial begin  $dumpfile("dump.vcd");  $dumpvars(0, i0, i1, s, y);  end  endmodule |

**OUTPUT:**

**P2). Gate level code for 2:1 MUX using vector and concatenation**

|  |  |
| --- | --- |
| ***Design code:***  module mux(i, s, y);    input [1:0]i;  input s;  output y;  wire w[3:1];    not n1 (w[1], s);  and al (w[2], i[0], w[1]);  and a2 (w[3], s, i[1]);  or o1 (y, w[2], w[3]);  endmodule | ***Test bench code:***  module mux\_test;  reg [1:0]i;  reg s;  wire y;  mux dut(.i(i), .s(s), .y(y));  initial begin  {i, s} = 0;  #10 {i, s} = 3;  #10 {i, s} = 6;  #10 {i, s} = 12;  end  initial begin  $monitor("Simulation time= %0t, i=%b, s=%b, y=%b", $time, i, s, y);  end  initial begin  $dumpfile("dump.vcd");  $dumpvars(0, i, s, y);  end  endmodule |

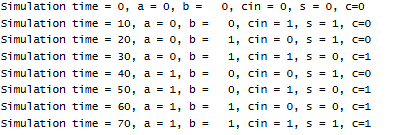
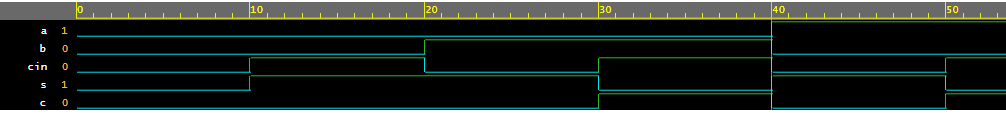
**OUTPUT:**

****

**P3).Gate level code for full adder**

|  |  |
| --- | --- |
| ***Design code:***  module full\_adder(a, b, cin, s, c);  input a, b, cin;  output s, c;  wire w1, w2, w3;  xor x1(s, a, b, cin);  and a1(w1, a, b);  and a2(w2, b, cin);  and a3(w3, a, cin);  or o1(c, w1, w2, w3);  endmodule | ***Test bench code:***  module full\_adder\_test;  reg a,b,cin;  wire s,c;  full\_adder dut(.a(a), .b(b), .cin(cin), .s(s), .c(c));  initial begin  a=0; b=0; cin=0;  #10 a=0; b=0; cin=1;  #10 a=0; b=1; cin=0;  #10 a=0; b=1; cin=1;  #10 a=1; b=0; cin=0;  #10 a=1; b=0; cin=1;  #10 a=1; b=1; cin=0;  #10 a=1; b=1; cin=1;  end  initial begin  $monitor("Simulation time = %0t, a = %b, b = %b, cin = %b, s = %b, c=%b", $time, a, b, cin, s, c);  end  initial begin  $dumpfile("dump.vcd");  $dumpvars(0, a, b, cin, s, c);  end  endmodule |

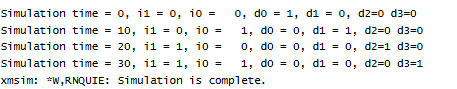
**OUTPUT:**

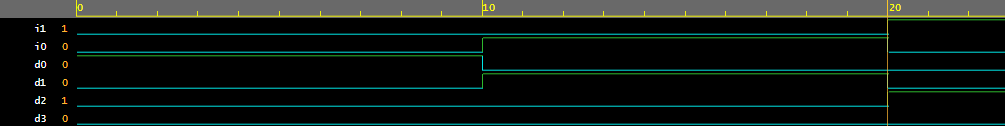
****

**P4). Gate level code for 2x4 decoder**

|  |  |
| --- | --- |
| ***Design code:***  module decoder(i1, i0, d0, d1, d2, d3);  input i1, i0;  output d0, d1, d2, d3;  wire w1, w2;  not n1(w1, i0);  not n2(w2, i1);  and a1(d0, w2, w1);  and a2(d1, w2, i0);  and a1(d2, i1, w1);  and a1(d3, i1, i0);  endmodule | ***Test bench code:***  module decoder\_test;  reg i1, i0;  wire d0, d1, d2,d3,d4;    decoder dut(.i1(i1), .i0(i0), .d0(d0), .d1(d1), .d2(d2), .d3(d3));    initial begin  i1=0; i0=0;  #10 i1=0; i0=1;  #10 i1=1; i0=0;  #10 i1=1; i0=1;  end    initial begin  $monitor("Simulation time = %0t, i1 = %b, i0 = %b, d0 = %b, d1 = %b, d2=%b d3=%b", $time, i1, i0, d0, d1, d2, d3);  end  initial begin  $dumpfile("dump.vcd");  $dumpvars(0, i0, i1, d0, d1, d2, d3);  end  endmodule |

**OUTPUT:**

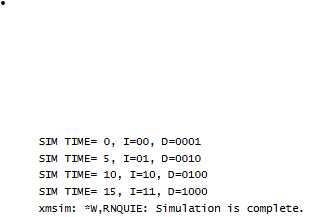
****

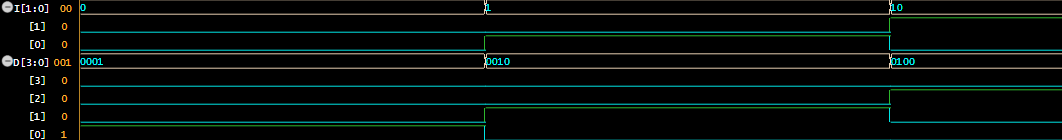
****

**P4). Data flow code for 2x4 decoder**

|  |  |
| --- | --- |
| ***Design code:***  module deco24(I, D);  input [1:0]I;  output [3:0]D;  assign D[0]=(~I[1])&(~I[0]);  assign D[1]=(~I[1])&(I[0]);  assign D[2]=(I[1])&(~I[0]);  assign D[3]=(I[1])&(I[0]);    endmodule | ***Test bench code:***  module deco24\_test;    reg [1:0]I;  wire [3:0]D;    deco24 dut(I, D);    initial begin  I=2'b00;  #5 I=2'b01;  #5 I=2'b10;  #5 I=2'b11;    end  initial begin  $monitor("SIM TIME= %0t, I=%b, D=%b ", $time, I, D);  end    initial begin  $dumpfile("dump.vcd");  $dumpvars(0,D,I);  end  endmodule |

**OUTPUT:**

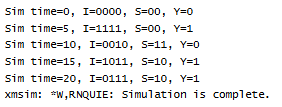
****

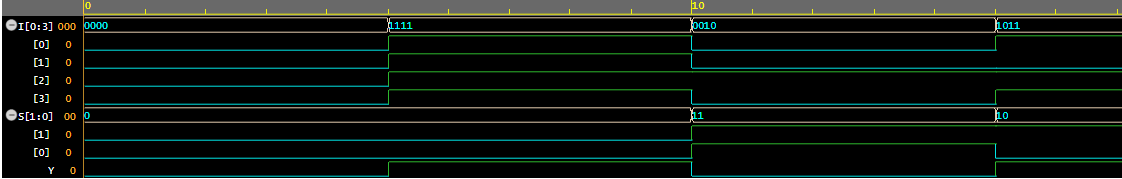
****

**P5). Data flow code for 4x1 mux using vector and concatenation**

|  |  |
| --- | --- |
| ***Design code:***  module mux41vector(I, S, Y);    input [0:3]I;  input [1:0]S;  output Y;    assign Y = ((~S[1])&(~S[0])&I[0])| ((~S[1])&(S[0])&I[1])|((S[1])&  (~S[0])&I[2])|((S[1])&(S[0])&I[3]);    endmodule | ***Test bench code:***  module mux41vector\_test;  reg [0:3]I;  reg [1:0]S;  wire Y;    mux41vector dut(I, S, Y);    initial begin  {I,S}=0;  #5 {I,S}=1;  #5 {I,S}=8;  #5 {I,S}=20;  #5 {I,S}=30;  end    initial begin  $monitor("Sim time=%0t, I=%b, S=%b, Y=%b", $time, I, S, Y);  end  initial begin  $dumpfile("dump.vcd");  $dumpvars(0, I, S, Y);  end  endmodule |

**OUTPUT:**

****

****

**P4). Gate level code for 3 bit binary to gray converter**

|  |  |
| --- | --- |
| ***Design code:***  module binary\_gray (B2, B1, BO, G2,GI,  GO)  input B2, Bl, B0;  output G2, G1, GO;  buf b1(G2, B2);  xor x1(G1, B2, B1);  xox x2(G0, B1, B0);  endmodule | ***Test bench code:*** |

**P5). Gate level code for 3-bit palindrome detector:**

|  |  |
| --- | --- |
| ***Design code:***  module palindrome (A2, A1, A0, Y);  input A2, A1, A0;  output Y;  xnor x1(Y, A2, A0);  endmodule | ***Testbench code:*** |

**P6). Gate level code for 4x2 Priority encoder**

|  |  |
| --- | --- |
| ***Design code:***  module priority\_enco (D3, D2, D1, D0, y1, y0);  input D3, D2, D1, D0;  output y1, y0;  wire W1,W2;  or o1 (y1, D2, D3);  not n1 (W1, D2);  and al (W2, W1, D1);  or o2 (y0, D3, w2);  endmodule | ***Testbench code:*** |

**P7). Gate level code for 2-bit n to n^2 converter**

|  |  |
| --- | --- |
| ***Design code:***  module nton2(n1, n0, y3, y2, y1,y0);  input n1, n0;  output y3, y2, y1, y0;  wire w1:  and a1(y3, n1, n0);  not n0(w1, n0);  and a2(y2, n1, w1);  buf b1(y0, n0);  endmodule | ***Testbench code:*** |

**P8). 2x1 mux using tristate buffer in gate level modelling**

|  |  |
| --- | --- |
| ***Design code:***  module nux21(I0, I1, S, Y);  inputI0, I1, S;  output y;  bufif0 b1(Y, I0, S);  bufif1 b1(Y, I1, S);  endmodule |  |

**P9). Gate level code for half adder**

|  |  |
| --- | --- |
| **Design code:**  module HA(A, B, S, C);  input A, B;  output S, C;  xor x1(S, A, B);  and a1(C, A, B);  endmodule; |  |

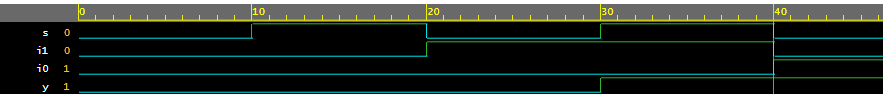
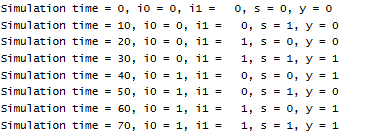
**P10). Data flow code for full adder**

|  |  |
| --- | --- |
| **Design code:**  module FA (A, B, C, S, Cout);  input A, B, C;  output S, Cout;  assign S = A^B^C;  assign Cout = (A&B)|(B&C)|(A&C);  endmodule | **Test bench code:**  module FA\_TB;  reg A, B, C;  wire S, Cout;  FA dut(.A(A), .B(B), .C(C), .S(S), .Cout(Cout));  initial begin  A=1' b0; B=1' b0; C=1’b0;  #10 A= 1' b0; B=1b’0; C= 1’b1;  #10 A= 1' b0; B=1b’1; C= 1’b0;  #10 A= 1' b0; B=1b’1; C= 1’b1;  #10 A= 1' b1; B=1b’0; C= 1’b0;  #10 A= 1' b1; B=1b’0; C= 1’b1;  #10 A= 1' b1; B=1b’1; C= 1’b0;  #10 A= 1' b1; B=1b’1; C= 1’b1;  end  initial begin  $monitor(“Simulation time = %0t, A=%b, B=%b, C=%b”, $time, A, B, C, S, Cout);  end  initial begin  $dumfile("dump.vcd");  $dumpvars (0, A, B, C, S. Cout);  end  endmodule- |

**P11). Behavioural code for 2x1 mux**

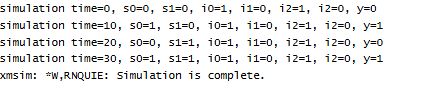
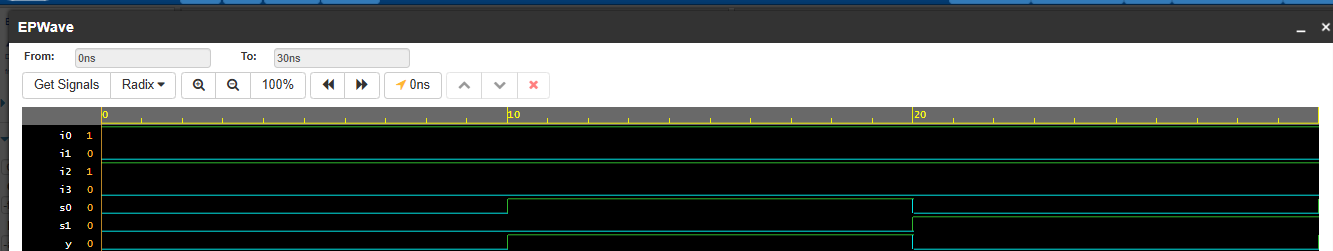
|  |  |
| --- | --- |
| **Design code:**  module mux21 (i0, i1, s, y);  input i0, i1, s;  output y;  assign y= ((~s)&i0) | (s&i1);  endmodule | **Test bench code:**  module mux21\_test;  reg i0, i1, s;  wire y;  mux21 dut(.i0(i0), .i1(i1), .s(s), .y(y));    initial begin  i0=0; i1=0; s=0;  #10 i0=0; i1=0; s=1;  #10 i0=0; i1=1; s=0;  #10 i0=0; i1=1; s=1;  #10 i0=1; i1=0; s=0;  #10 i0=1; i1=0; s=1;  #10 i0=1; i1=1; s=0;  #10 i0=1; i1=1; s=1;  end  initial begin  $monitor("Simulation time = %0t, i0 = %b, i1 = %b, s = %b, y = %b", $time, i0, i1, s, y);  end  initial begin  $dumpfile("dump.vcd");  $dumpvars(0, i0, i1, s, y);  end  endmodule |

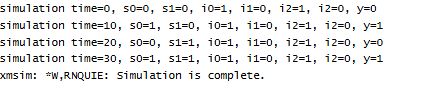
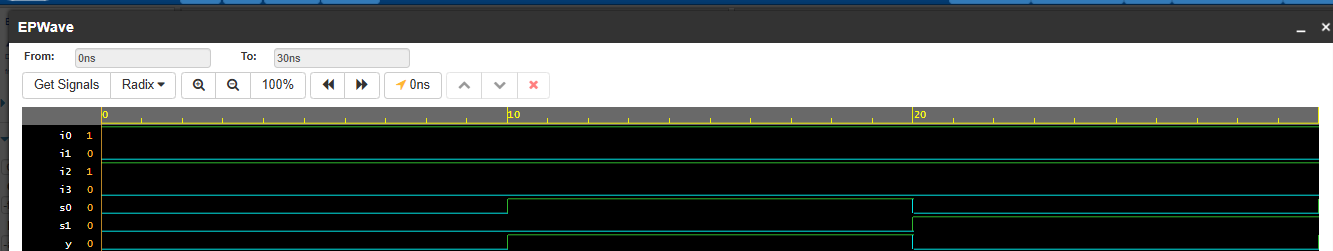
**OUTPUT:**

****

**P12).Gate level code for 4x1 mux**

|  |  |
| --- | --- |
| **Design code:**  module mux41(s0, s1, i0, i1, i2, i3, y);  input s0, s1;  input i0, i1, i2, i3;  output y;  wire x1, x2, w1, w2, w3, w4;    not n1(x1, s1);  not n2(x2, s0);  and a1(w1, x1, x2, i0);  and a2(w2, x1, s0, i1);  and a3(w3, s1, x2, i2);  and a4(w4, s1, s0, i3);  nor n3(y, w1, w2, w3, w4);    endmodule | **Test bench code:**    module mux41\_tb;  reg s0, s1;  reg i0, i1, i2, i3;  wire y;    mux41 dut(.s0(s0), .s1(s1), .i0(i0), .i1(i1), .i2(i2), .i3(i3), .y(y));    initial begin  i0=1; i1=0; i2=1; i3=0;  s1=0; s0=0;  #10 s1=0; s0=1;  #10 s1=1; s0=0;  #10 s1=1; s0=1;  end    initial begin  $monitor("simulation time=%0t, s0=%b, s1=%b, i0=%b, i1=%b, i2=%b, i2=%b, y=%b", $time, s0, s1, i0, i1, i2, i3, y);  end    initial begin  $dumpfile("dump.vcd");  $dumpvars(0, s0, s1,i0, i1, i2, i3, y);  end  endmodule |

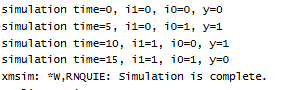
****

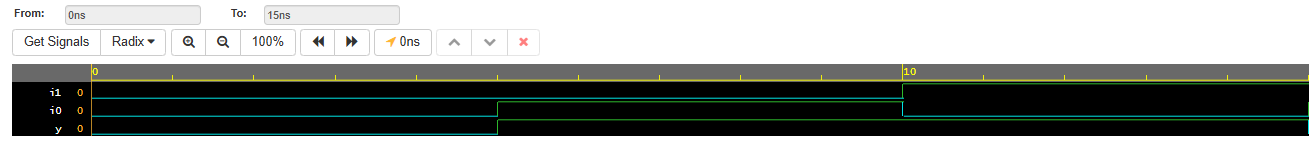
**P13).Behavioura] level code for 4x1 mux**

|  |  |
| --- | --- |
| **Design code:**  module mux41(s0, s1, i0, i1, i2, i3, y);  input s0, s1;  input i0, i1, i2, i3;  output y;    assign y=((~s1)&(~s0)&i0)|((~s1)&(s0)&i1)|((s1)&(~s0)&i2)|((s1)&(s0)&i3);    endmodule | **Test bench code:**    module mux41\_tb;  reg s0, s1;  reg i0, i1, i2, i3;  wire y;    mux41 dut(.s0(s0), .s1(s1), .i0(i0), .i1(i1), .i2(i2), .i3(i3), .y(y));    initial begin  i0=1; i1=0; i2=1; i3=0;  s1=0; s0=0;  #10 s1=0; s0=1;  #10 s1=1; s0=0;  #10 s1=1; s0=1;  end    initial begin  $monitor("simulation time=%0t, s0=%b, s1=%b, i0=%b, i1=%b, i2=%b, i2=%b, y=%b", $time, s0, s1, i0, i1, i2, i3, y);  end    initial begin  $dumpfile("dump.vcd");  $dumpvars(0, s0, s1,i0, i1, i2, i3, y);  end  endmodule |

**P14). Gate level code for xor using nand**

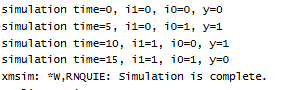
|  |  |
| --- | --- |
| **Design code:**  module xornand(i1, i0, y);  input i1, i0;  output y;  wire w1, w2, w3;    nand n1(w1, i1,i0);  nand n2(w2, i1, w1);  nand n3(w3, i0, w1);  nand n4(y, w2, w3);    endmodule | **Test bench code:**    module xornand\_tb;  reg i1, i0;  wire y;    xornand dut(.i1(i1), .i0(i0), .y(y));    initial begin  i1=0; i0=0;  #5 i1=0; i0=1;  #5 i1=1; i0=0;  #5 i1=1; i0=1;  end    initial begin  $monitor("simulation time=%0t, i1=%b, i0=%b, y=%b", $time, i1, i0, y);  end    initial begin  $dumpfile("dump.vcd");  $dumpvars(0, i0, i1, y);  end  endmodule |

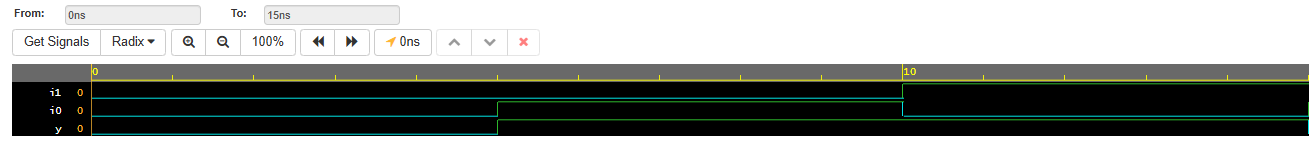
****

****

**P15). Behavioural level code for xor using nand**

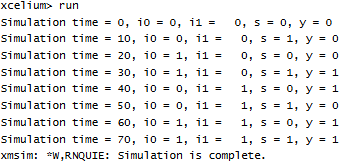
|  |  |
| --- | --- |
| **Design code:**  module xornand(i1, i0, y);  input i1, i0;  output y;  wire w1, w2, w3;    assign w1 = ~(i1 & i0);  assign w2 = ~(i1 & w1);  assign w3 = ~(i0 & w1);  assign y = ~(w2 & w3);    endmodule | **Test bench code:**    module xornand\_tb;  reg i1, i0;  wire y;    xornand dut(.i1(i1), .i0(i0), .y(y));    initial begin  i1=0; i0=0;  #5 i1=0; i0=1;  #5 i1=1; i0=0;  #5 i1=1; i0=1;  end    initial begin  $monitor("simulation time=%0t, i1=%b, i0=%b, y=%b", $time, i1, i0, y);  end    initial begin  $dumpfile("dump.vcd");  $dumpvars(0, i0, i1, y);  end  endmodule |

****

****

**P16). Behavioural code for 2x1 mux using nand**

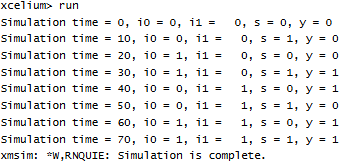
|  |  |
| --- | --- |
| **Design code:**  module muxnand(i1, i0, s, y);  input i1, i0, s;  output y;  wire w1, w2, w3;    assign w1 = ~(i0 & s);  assign w2 = ~(s&s);  assign w3 = ~(i1 & w2);  assign y = ~(w1 & w3);    endmodule | **Test bench code:**  module muxnand\_test;  reg i1, i0, s;  wire y;  muxnand dut(.i1(i1), .i0(i0), .s(s), .y(y));    initial begin  i1=0; i0=0; s=0;  #10 i1=0; i0=0; s=1;  #10 i1=0; i0=1; s=0;  #10 i1=0; i0=1; s=1;  #10 i1=1; i0=0; s=0;  #10 i1=1; i0=0; s=1;  #10 i1=1; i0=1; s=0;  #10 i1=1; i0=1; s=1;  end  initial begin  $monitor("Simulation time = %0t, i0 = %b, i1 = %b, s = %b, y = %b", $time, i0, i1, s, y);  end  initial begin  $dumpfile("dump.vcd");  $dumpvars(0, i0, i1, s, y);  end  endmodule |

****

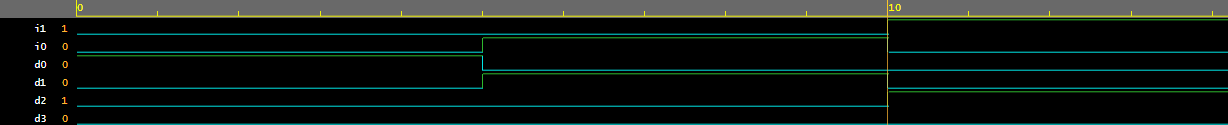
****

**P17). Gate level code for 2x1 mux using nand**

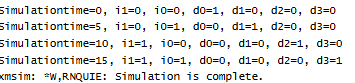
|  |  |
| --- | --- |
| **Design code:**  module mux21nand(i1, i0, s, y);  input i1, i0,s;  output y;  wire w1,w2,w3;    nand n1(w1,i0,s);  nand n2(w2,s);  nand n3(w3, i1);  nand n4(y,w1,w3);    endmodule | **Test bench code:**  module muxnand\_test;  reg i1, i0, s;  wire y;  muxnand dut(.i1(i1), .i0(i0), .s(s), .y(y));    initial begin  i1=0; i0=0; s=0;  #10 i1=0; i0=0; s=1;  #10 i1=0; i0=1; s=0;  #10 i1=0; i0=1; s=1;  #10 i1=1; i0=0; s=0;  #10 i1=1; i0=0; s=1;  #10 i1=1; i0=1; s=0;  #10 i1=1; i0=1; s=1;  end  initial begin  $monitor("Simulation time = %0t, i0 = %b, i1 = %b, s = %b, y = %b", $time, i0, i1, s, y);  end  initial begin  $dumpfile("dump.vcd");  $dumpvars(0, i0, i1, s, y);  end  endmodule |

****

****

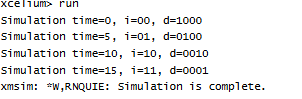
**P18). Gate level code for 2x4 decoder**

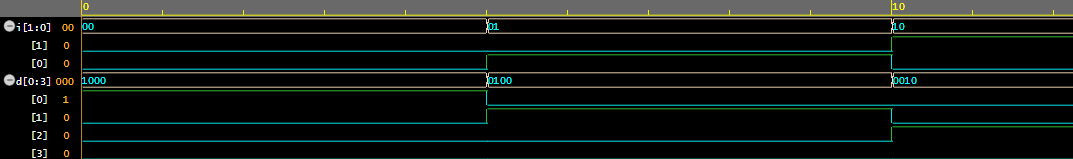
|  |  |
| --- | --- |
| **Design code:**  module decoder24(i1, i0, d0, d1, d2, d3);  input i1, i0;  output d0, d1, d2, d3;  wire w1, w2;  not n1(w1, i0);  not n2(w2, i1);  and a1(d0, w2, w1);  and a2(d1, w2, i0);  and a3(d2, i1, w1);  and a4(d3, i1, i0);  endmodule | **Test bench code:**  module decoder24\_test;  reg i1, i0;  wire d0, d1, d2, d3;    decoder24 dut(.i1(i1), .i0(i0), .d0(d0), .d1(d1), .d2(d2), .d3(d3));    initial begin  i1=0; i0=0;  #5 i1=0; i0=1;  #5 i1=1; i0=0;  #5 i1=1; i0=1;  end    initial begin  $monitor("Simulationtime=%0t, i1=%b, i0=%b, d0=%b, d1=%b, d2=%b, d3=%b", $time, i1, i0, d0,d1, d2,d3);  end    initial begin  $dumpfile("dump.vcd");  $dumpvars(0, i1, i0, d0,d1,d2,d3);  end  endmodule |

****

**P19). Behvioural code for 2x4 decoder**

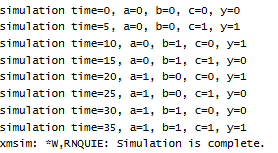
|  |  |
| --- | --- |
| **Design code:**  module decoder24(i, d);  input [1:0]i;  output [0:3]d;    assign d[0]= (~i[1]) & (~i[0]);  assign d[1]= (~i[1]) & (i[0]);  assign d[2]= (i[1]) & (~i[0]);  assign d[3]= (i[1]) & (i[0]);    endmodule | **Test bench code:**  // Code your testbench here  // or browse Examples  module decoder24\_test;  reg [1:0]i;  wire [0:3]d;    decoder24 dut(.i(i), .d(d));    initial begin  i=2'b00;  #5 i=2'b01;  #5 i=2'b10;  #5 i=2'b11;  end    initial begin  $monitor("Simulation time=%0t, i=%b, d=%b", $time, i, d);  end    initial begin  $dumpfile("dump.vcd");  $dumpvars(0, i, d);  end  endmodule |

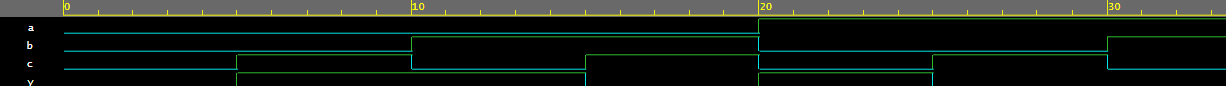
****

****

**P20). Gate level Even parity detector using 2x1 mux**

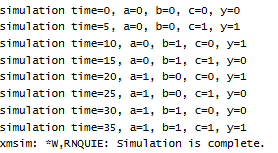
|  |  |
| --- | --- |
| **Design code:**  module EPUMUX(a,b,c,y);  input a,b,c;  output y;  wire w1, w2, w3;  wire [1:0]i;    xor x1(i[1], b, c);  xnor x2(i[0], b, c);  not n1(w1, a);  and a1(w2,w1,i[1]);  and a2(w3, a, i[0]);  or o1(y, w2, w3);    endmodule | **Test bench code:**  // Code your testbench here  // or browse Examples  module EPUMUX\_test;  reg a,b,c;  wire y;    EPUMUX dut(a,b,c,y);    initial begin  a=0; b=0; c=0;  #5 c=1;  #5 b=1; c=0;  #5 c=1;  #5 a=1; b=0; c=0;  #5 c=1;  #5 b=1; c=0;  #5 c=1;  end    initial begin  $monitor("simulation time=%0t, a=%b, b=%b, c=%b, y=%b", $time, a, b, c,y);  end    initial begin  $dumpfile("dump.vcd");  $dumpvars(0, a, b, c, y);  end  endmodule |

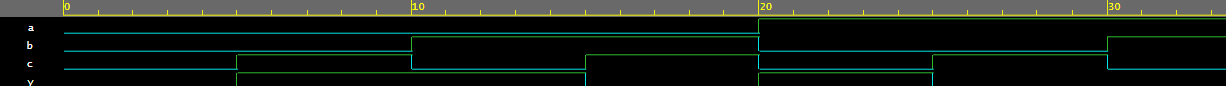
****

****

**P21). Behavioural Even parity detector using 2x1 mux**

|  |  |
| --- | --- |
| **Design code:**  module EPUMUX(a,b,c,y);  input a,b,c;  output y;  assign y= ((~a)&(b^c))+((a)&(~(b^c)));    endmodule | **Test bench code:**  // Code your testbench here  // or browse Examples  module EPUMUX\_test;  reg a,b,c;  wire y;    EPUMUX dut(a,b,c,y);    initial begin  a=0; b=0; c=0;  #5 c=1;  #5 b=1; c=0;  #5 c=1;  #5 a=1; b=0; c=0;  #5 c=1;  #5 b=1; c=0;  #5 c=1;  end    initial begin  $monitor("simulation time=%0t, a=%b, b=%b, c=%b, y=%b", $time, a, b, c,y);  end    initial begin  $dumpfile("dump.vcd");  $dumpvars(0, a, b, c, y);  end  endmodule |

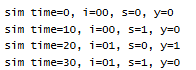
****

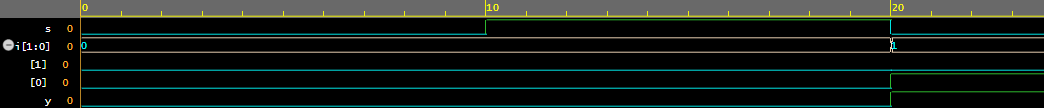
****

**P21). Code for 2x1 mux using ternary operator**

|  |  |
| --- | --- |
| **Design code:**  module mux21 (i, s, y);  input [1:0]i;  input s;  output y;    assign y= (s==1'b0) ? i[0] : i[1];    endmodule | **Test bench code:**  module mux21\_test;  reg [1:0]i;  reg s;  wire y;    mux21 dut(.i(i), .s(s), .y(y));    initial begin  {i, s} = 0;  #10 {i, s} = 1;  #10 {i, s} = 2;  #10 {i, s} = 3;  end  initial begin  $monitor("sim time=%0t, i=%b, s=%b, y=%b", $time, i, s, y);  end    initial begin  $dumpfile("dump.vcd");  $dumpvars(0, i, s, y);  end  endmodule |

**OUTPUT:**

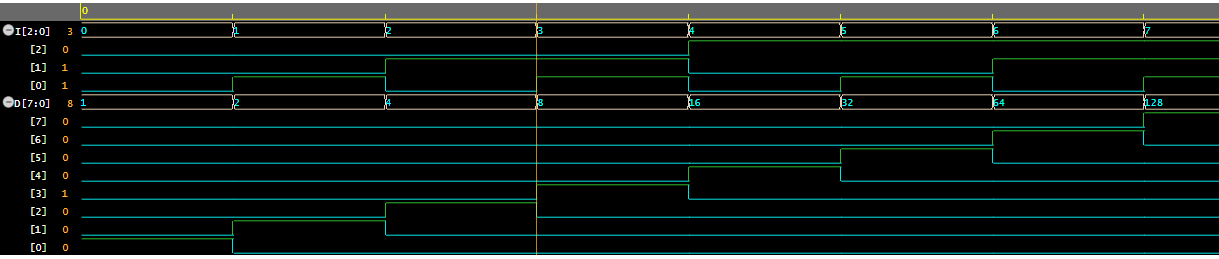
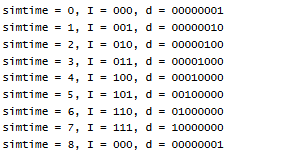
****

****

**P21). Code for 3x8 using 1x2 and 2x4 decoder**

|  |  |
| --- | --- |
| **Design code:**  **//**Code for 1x2 decoder  module DEC12(I,D);  input I;  output[1:0]D;  assign D[0]=(~I);  assign D[1]=(I);  endmodule  //Code for 2x4 decoder  module DEC24(e,I,D);  input[1:0]I;  input e;  output[3:0]D;  assign D[0]=(e&(~I[1])&(~I[0]));  assign D[1]=(e&(~I[1])&(I[0]));  assign D[2]=(e&(I[1])&(~I[0]));  assign D[3]=(e&(I[1])&(I[0]));  endmodule  // Code for 3x8 decoder  `include "DEC12.v"  `include "DEC2X4.v"  module DEC38(I,D);  input[2:0]I;  output[7:0]D;  wire [1:0]W;  DEC12 d1(I[2],W);  DEC24 d2(W[0],I[1:0],D[3:0]);  DEC24 d3(W[1],I[1:0],D[7:4]);  endmodule | **Test bench code:**  module DEC38\_tst;  reg[2:0]I;  wire[7:0]D;  DEC38 dut(I,D);  initial begin  for(integer i=0;i<=2\*\*3;i=i+1)  begin  {I}=i;  #1;  end  end  initial begin  $dumpfile("dump.vcd");  $dumpvars(1,DEC38\_tst);  End  endmodule |

**OUTPUT:**

****