|  |  |  |
| --- | --- | --- |
| **Name: Sriram R** | **SRN: PES1UG20CS435** | **Section: H** |
| **Date: 08-09-2021** | **Week Number: 1** |

|  |  |  |
| --- | --- | --- |
| **1** | Implement a 4 bit ripple carry adder using full adder modules. | |
|  | **Program:**  **d.v :**  // Module 4-bit ripple carry adder.  module invert(input wire i, output wire o1);  assign o1 = !i;  endmodule  module and2(input wire i0, i1, output wire o2);  assign o2 = i0 & i1;  endmodule  module or2(input wire i0, i1, output wire o3);  assign o3 = i0 | i1;  endmodule  module xor2(input wire i0, i1, output wire o4);  assign o4 = i0 ^ i1;  endmodule  module nand2(input wire i0, i1, output wire o5);  wire t;  and2 and2\_0 (i0, i1, t);  invert invert\_0 (t, o5);  endmodule  //Module full adder.  module fulladd(input wire a, b, cin, output wire sum, cout);  wire [4:0] t;  xor2 x0(a, b, t[0]);  xor2 x1(t[0], cin, sum);  and2 a0(a, b, t[1]);  and2 a1(a, cin, t[2]);  and2 a2(b, cin, t[3]);  or2 o0(t[1], t[2], t[4]);  or2 o1(t[3], t[4], cout);  endmodule  module fulladdR(input wire [3:0] a, b, input wire cin, output wire [3:0] sum, output wire cout);  // Instantiate full adder modules here.  wire [2:0] c;  fulladd f1(a[0], b[0], cin, sum[0], c[0]);  fulladd f2(a[1], b[1], c[0], sum[1], c[1]);  fulladd f3(a[2], b[2], c[1], sum[2], c[2]);  fulladd f4(a[3], b[3], c[2], sum[3], cout);  endmodule  **dtestBench.v :**  `timescale 1 ns / 100 ps  `define TESTVECS 8  module tb;  reg clk, reset;  reg [3:0] i0, i1; reg cin;  wire [3:0] o; wire cout;  reg [8:0] test\_vecs [0:(`TESTVECS-1)];  integer i;  initial begin $dumpfile("dump1.vcd");  $dumpvars(0,tb);  end  initial begin reset = 1'b1; #12.5 reset = 1'b0; end  initial clk = 1'b0; always #5 clk =~ clk;  initial begin  test\_vecs[0] = 9'b000000010;  test\_vecs[1] = 9'b000100010;  test\_vecs[2] = 9'b011100010;  test\_vecs[3] = 9'b000001110;  test\_vecs[4] = 9'b011001111;  test\_vecs[5] = 9'b001110011;  test\_vecs[6] = 9'b111100011;  test\_vecs[7] = 9'b011101110;  end  initial {i0, i1, cin, i} = 0;  fulladdR u0 (i0, i1, cin, o, cout);  initial begin  #6 for(i=0;i<`TESTVECS;i=i+1)  begin #10 {i0, i1, cin}=test\_vecs[i]; end  #100 $finish;  end  endmodule |
|  | **Output Screenshots:** | |
| 2 | Implement a 2:1 MUX. | |
|  | **Program :**  **MUX2to1.v :**  module mux2 (input wire i0, i1, j, output wire o);  assign o = (j==0)?i0:i1;  endmodule  **testBench2to1.v:**  module TB;  reg A,B,S;  wire X;  initial  begin  $dumpfile("dump2.vcd");  $dumpvars(0,TB);  end  mux2 newMUX(.i0(A), .i1(B), .j(S), .o(X));  initial  begin  S = 1'b0;  A = 1'b0;  B = 1'b0;  #5  A = 1'b0;  B = 1'b1;  #5  A = 1'b1;  B = 1'b0;  #5  A = 1'b1;  B = 1'b1;  #5  S = 1'b0;  A = 1'b0;  B = 1'b0;  #5  A = 1'b0;  B = 1'b1;  #5  A = 1'b1;  B = 1'b0;  #5  A = 1'b1;  B = 1'b1;  end  endmodule | |
|  | **Output Screenshots:** | |
| 3 | Implement a 4:1 MUX using 2:1 MUX modules. | |
|  | **Program :**  **MUX4to1.v :**  module mux2 (input wire i0, i1, j, output wire o);  assign o = (j==0)?i0:i1;  endmodule  module mux4 (input wire [0:3] i, input wire j1, j0, output wire o);  wire t0, t1;    mux2 m1(i[0], i[1], j0, t0);  mux2 m2(i[2], i[3], j0, t1);  mux2 m3(t0, t1, j1, o);  endmodule  **testBench4to1.v:**  module TB;  reg [0:3]ii;  reg s0;  reg s1;  wire yy;  initial  begin  $dumpfile("dump3.vcd");  $dumpvars(0, TB);  end  mux4 newMUX(.i(ii), .j0(s0),.j1(s1),.o(yy));  initial  begin  ii = 4'b0001;  s0=1'b0;  s1=1'b0;  #5  ii = 4'b1000;  #5  ii = 4'b0001;  s0=1'b1;  s1=1'b1;  #5  ii = 4'b0000;  s0=1'b1;  s1=1'b0;  end  endmodule | |
|  | **Output Screenshots:** | |