|  |
| --- |
| **DIGITAL SYSTEM DESIGN LABORATORY** |
| **LAB 2** |

**IMPLEMENTATION OF COMBINATION LOGIC CIRCUIT (PART 2) USING VERILOG IN FPGA KIT**

### I. LAB OBJECTIVES

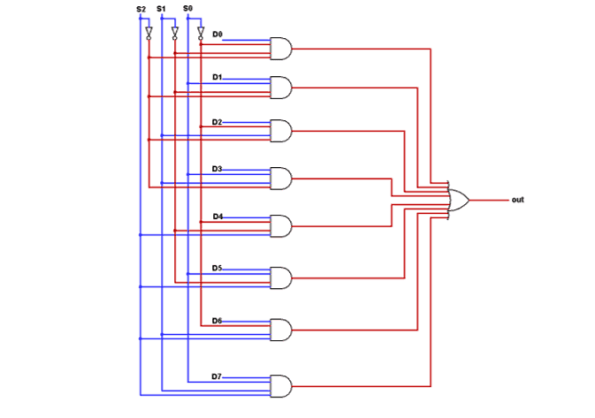
### This Lab experiments are intended to implement Combination Logic Circuits in Verilog. Students are require to write test bench to simulate the given example code and Top level module to implement these codes in DE2-115 FPGA Kit.

### II. LAB EXPERIMENT EXERCISES

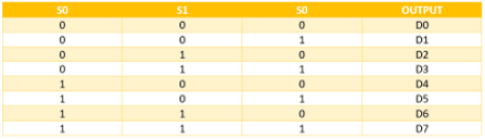
**AIM: WRITE VERILOG HDL CODES TO SIMULATE AND IMPLEMENT THE FOLLOWING DIGITAL SEQUENTIAL LOGIC CIRCUITS:**

**1) Multiplexer 8 to 1**

**Logic circuit**



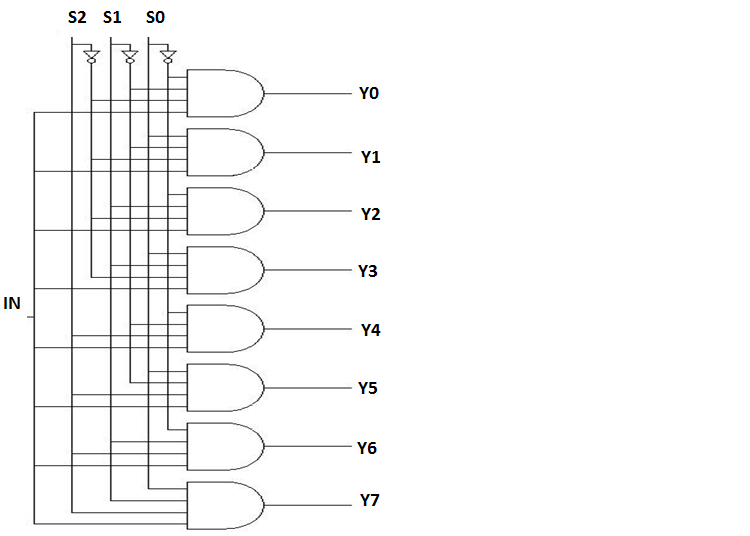
**Truth table**



|  |  |  |
| --- | --- | --- |
| **Top module** | | |
| module lab2\_ex1(SW,LEDG,LEDR);  input[17:0]SW;  output[7:0] LEDG;  output[17:0] LEDR;    assign LEDR = SW;    //m81\_GL(.d(SW[11:4]),.s(SW[2:0]),.out(LEDG[7]));  //m81\_DF(.d(SW[11:4]),.s(SW[2:0]),.out(LEDG[7]));  m81\_BEH(.d(SW[11:4]),.s(SW[2:0]),.out(LEDG[7]));  endmodule | | |
| **Structural** | **Dataflow** | **Behavior** |
| module m81\_GL(d,s,out);  input [7:0] d;  input [2:0] s;  output out;  wire [10:0] t;  not(t[0], s[0]);  not(t[1], s[1]);  not(t[2], s[2]);  and(t[3], d[0], t[0], t[1], t[2]), (t[4], d[1],s[0], t[1], t[2]);  and(t[5], d[2], t[0], s[1], t[2]), (t[6], d[3], s[0], s[1], t[2]);  and(t[7], d[4], t[0], t[1], s[2]), (t[8], d[5], s[0], t[1], s[2]);  and(t[9], d[6], t[0], s[1], s[2]), (t[10], d[7], s[0], s[1], s[2]);  or(out, t[3], t[4], t[5], t[6], t[7],t[8], t[9], t[10]);  endmodule | module m81\_DF(d,s,out);  input [7:0] d;  input [2:0] s;  wire [2:0] n\_s;  output out;  assign n\_s[0]=~s[0];  assign n\_s[1]=~s[1];  assign n\_s[2]=~s[2];  assign out = (d[0] & n\_s[2] & n\_s[1] & n\_s[0]) | (d[1] & n\_s[2] & n\_s[1] & s[0]) | (d[2] & n\_s[2] & s[1] & n\_s[0]) + (d[3] & n\_s[2] & s[1] & s[0]) + (d[4] & s[2] & n\_s[1] & n\_s[0]) + (d[5] & s[2] & n\_s[1] & s[0]) + (d[6] & s[2] & s[1] & n\_s[0]) + (d[7] & s[2] & s[1] & s[0]);  endmodule | module m81\_BEH(d,s,out);  input [7:0] d;  input [2:0] s;  output reg out;  always@(\*) begin  case(s)  3'b000: out=d[0];  3'b001: out=d[1];  3'b010: out=d[2];  3'b011: out=d[3];  3'b100: out=d[4];  3'b101: out=d[5];  3'b110: out=d[6];  3'b111: out=d[7];  default: out=1'b0;  endcase  end  endmodule |
| **Testbench** | | |
| module tb\_m81;  reg [7:0] d;  reg [2:0] s;  wire out;  m81\_BEH uut (  .d(d),  .s(s),  .out(out)  );  initial begin  d = 8'b00000000;  s = 3'b000;  d = 8'b10101010;  #10 s = 3'b000;  #10 s = 3'b001;  #10 s = 3'b010;  #10 s = 3'b011;  #10 s = 3'b100;  #10 s = 3'b101;  #10 s = 3'b110;  #10 s = 3'b111;  #10 $finish;  end  initial begin  $monitor("Time = %t | d = %b | s = %b | out = %b", $time, d, s, out);  end  endmodule | | |
|  | | |
|  | | |

2) 1:8 Demultiplexer

**Logic circuit**



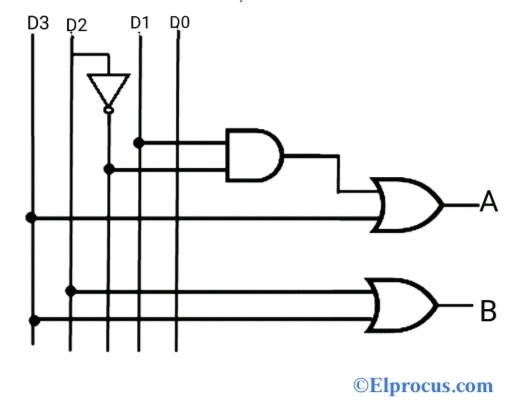
**Truth table**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **S2** | **S1** | **S0** | **Y7** | **Y6** | **Y5** | **Y4** | **Y3** | **Y2** | **Y1** | **Y0** |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | IN |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | IN | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | IN | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | IN | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | IN | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | IN | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | IN | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | IN | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

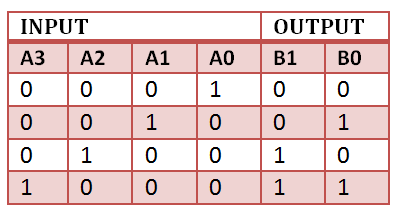
|  |  |  |
| --- | --- | --- |
| **Top module** | | |
| module lab2\_ex2(SW,LEDG,LEDR);  input[17:0]SW;  output[7:0] LEDG;  output[17:0] LEDR;    assign LEDR = SW;    //demux18\_GL(.a(SW[4]),.s(SW[2:0]),.y(LEDG[7:0]));  //demux18\_DF(.a(SW[4]),.s(SW[2:0]),.y(LEDG[7:0]));  demux18\_BH(.a(SW[4]),.s(SW[2:0]),.y(LEDG[7:0]));  endmodule | | |
| **Structural** | **Dataflow** | **Behavior** |
| module demul18\_GL(a,s,y);  input a;  input [2:0]s;  output[7:0]y;    wire [2:0] n\_s;  not (n\_s[0], s[0]);  not (n\_s[1], s[1]);  not (n\_s[2], s[2]);  and (y[0], a, n\_s[2], n\_s[1], n\_s[0]);  and (y[1], a, n\_s[2], n\_s[1], s[0]);  and (y[2], a, n\_s[2], s[1], n\_s[0]);  and (y[3], a, n\_s[2], s[1], s[0]);  and (y[4], a, s[2], n\_s[1], n\_s[0]);  and (y[5], a, s[2], n\_s[1], s[0]);  and (y[6], a, s[2], s[1], n\_s[0]);  and (y[7], a, s[2], s[1], s[0]);  endmodule | module Demultiplexer\_1\_to\_8\_assign(output [7:0] Y, input [2:0] A, input din);  assign Y[0] = din & (~A[0]) & (~A[1]) & (~A[2]);  assign Y[1] = din & (~A[1]) & A[0] & (~A[2]);  assign Y[2] = din & A[1] & (~A[0]) & (~A[2]);  assign Y[3] = din & A[1] & A[0] & (~A[2]);  assign Y[4] = din & (~A[0]) & (~A[1]) & (A[2]);  assign Y[5] = din & (~A[1]) & A[0] & (A[2]);  assign Y[6] = din & A[1] & (~A[0]) & (A[2]);  assign Y[7] = din & A[1] & A[0] & (A[2]);  endmodule | module demux18\_BH(a,s,y);  input a;  input [2:0]s;  output reg [7:0]y;    always @(\*) begin  y[0] = (s == 3'b000) ? a : 1'b0;  y[1] = (s == 3'b001) ? a : 1'b0;  y[2] = (s == 3'b010) ? a : 1'b0;  y[3] = (s == 3'b011) ? a : 1'b0;  y[4] = (s == 3'b100) ? a : 1'b0;  y[5] = (s == 3'b101) ? a : 1'b0;  y[6] = (s == 3'b110) ? a : 1'b0;  y[7] = (s == 3'b111) ? a : 1'b0;  end  endmodule |
| Testbench | | |
| module demux\_18\_tb;  wire [7:0] y;  reg a;  reg [2:0] s;  demul18\_GL dut (a, s, y);  initial begin  a = 1'b0;  s = 3'b000;  #100 a = 1'b1;  #800 $finish;  end  initial begin  $monitor("Time = %0t | a = %b | s = %b | y = %b", $time, a, s, y);  end  always #100 s[0] = ~s[0];  always #200 s[1] = ~s[1];  always #400 s[2] = ~s[2];  endmodule | | |
|  | | |
|  | | |

**3) 4-to-2 bit Encoder**

**Logic circuit**



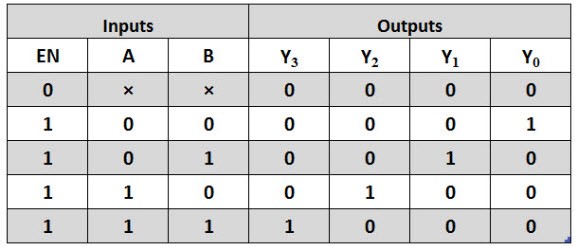
**Truth table**



|  |  |  |
| --- | --- | --- |
| **Top module** | | |
| module lab2\_ex3(SW,LEDG,LEDR);  input[17:0]SW;  output[7:0] LEDG;  output[17:0] LEDR;    assign LEDR = SW;    //encoder42\_GL(.a(SW[3:0]),.y(LEDG[7:6]));  //encoder42\_DF(.a(SW[3:0]),.y(LEDG[7:6]));  encoder42\_BH(.a(SW[3:0]),.y(LEDG[7:6]));  endmodule | | |
| **Structural** | **Dataflow** | **Behavior** |
| module encoder42\_GL(a,y);  input [3:0]a;  output [1:0]y;  wire n\_a2,w;    not (n\_a2, a[2]);  and (w, n\_a2,a[1]);  or (y[0], w, a[3]);  or (y[1], a[2], a[3]);  endmodule | module encoder42\_DF(a,y);  input [3:0]a;  output [1:0]y;    assign y[0] = a[1] | a[3];  assign y[1] = a[2] | a[3];  endmodule | module encoder42\_BH(a,y);  input [3:0]a;  output reg [1:0]y;  always @(\*) begin  case (1'b1)  a[3]: y = 2'b11;  a[2]: y = 2'b10;  a[1]: y = 2'b01;  a[0]: y = 2'b00;  default: y = 2'b00;  endcase  end  endmodule |
| **Testbench** | | |
| module tb\_encoder42;  reg [3:0]a;  wire [1:0]y;    encoder42\_GL dut(a,y);    initial begin  a=4'b0000;    $monitor("Time: %0t,a = %b,y = %b",$time,a,y);  #10 a=4'b0001;  #10 a=4'b0010;  #10 a=4'b0100;  #10 a=4'b1000;    $finish;  end  endmodule | | |
|  | | |
|  | | |

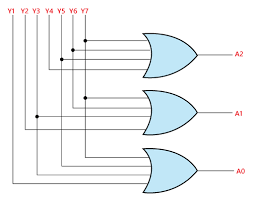
**4) 2-to-4 Binary Decoders**

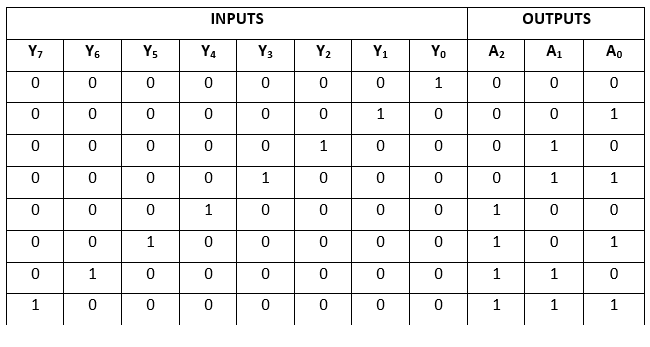




|  |  |  |
| --- | --- | --- |
| **Top module** | | |
| module lab2\_ex4(SW,LEDG,LEDR);  input[17:0]SW;  output[7:0] LEDG;  output[17:0] LEDR;    assign LEDR = SW;    //decoder24\_BH(.e(SW[2]),.x(SW[1:0]),.y(LEDG[7:4]));  //decoder24\_DF(.e(SW[2]),.x(SW[1:0]),.y(LEDG[7:4]));  decoder24\_GL(.e(SW[2]),.x(SW[1:0]),.y(LEDG[7:4]));  endmodule | | |
| **Structural** | **Dataflow** | **Behavior** |
| module decoder24\_GL(e, x, y);  input e;  input [1:0] x;  output [3:0] y;    wire [1:0] nx;    not g1(nx[0], x[0]);  not g2(nx[1], x[1]);    and g3(y[0], nx[0], nx[1], e);  and g4(y[1], x[0], nx[1], e);  and g5(y[2], nx[0], x[1], e);  and g6(y[3], x[0], x[1], e);  endmodule | module decoder24\_DF(e, x, y);  input e;  input [1:0] x;  output [3:0] y;  assign y[0] = ~x[0] & ~x[1] & e;  assign y[1] = x[0] & ~x[1] & e;  assign y[2] = ~x[0] & x[1] & e;  assign y[3] = x[0] & x[1] & e;  endmodule | module decoder24\_BH(e, x, y);  input e;  input [1:0] x;  output reg [3:0] y;  always @(\*) begin  if (e) begin  case (x)  2'b00: y <= 4'b0001;  2'b01: y <= 4'b0010;  2'b10: y <= 4'b0100;  2'b11: y <= 4'b1000;  default: y <= 4'b0000;  endcase  end else begin  y <= 4'b0000;  end  end  endmodule |
| **Testbench** | | |
| module decoder24\_testbench;  reg e;  reg [1:0] x;  wire [3:0] y\_GL, y\_DF, y\_BEH;  decoder24\_GL gate\_level\_decoder (.e(e), .x(x), .y(y\_GL));  decoder24\_DF dataflow\_decoder (.e(e), .x(x), .y(y\_DF));  decoder24\_BH behavioral\_decoder (.e(e), .x(x), .y(y\_BEH));  initial begin  $monitor("Time = %0d | e = %b | x = %b | GL = %b | DF = %b | BEH = %b",  $time, e, x, y\_GL, y\_DF, y\_BEH);  e = 1; x = 2'b00; #10;  e = 1; x = 2'b01; #10;  e = 1; x = 2'b10; #10;  e = 1; x = 2'b11; #10;  e = 0; x = 2'b00; #10;  e = 0; x = 2'b11; #10;  $finish;  end  endmodule | | |
|  | | |
|  | | |

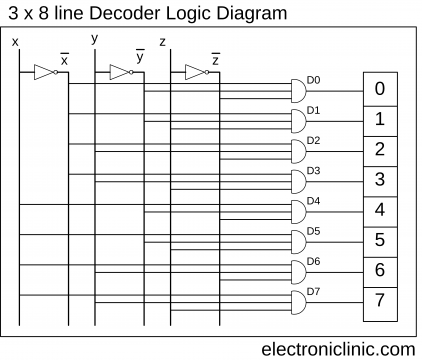
**5) 8:3 Encoder**

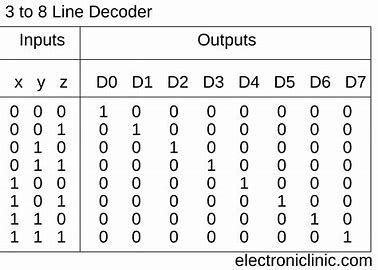




|  |  |  |
| --- | --- | --- |
| **Top module** | | |
| module lab2\_ex5(SW,LEDG,LEDR);  input[17:0]SW;  output[7:0] LEDG;  output[17:0] LEDR;    assign LEDR = SW;    //encoder83\_DF(.a(SW[7:0]),.y(LEDG[7:5]));  //encoder83\_BH(.a(SW[7:0]),.y(LEDG[7:5]));  encoder83\_GL(.a(SW[7:0]),.y(LEDG[7:5]));    endmodule | | |
| **Structural** | **Dataflow** | **Behavior** |
| module encoder83\_GL(a,y);  input [7:0] a;  output [2:0] y;    or g1(y[0],a[1],a[3],a[5],a[7]);  or g2(y[1],a[2],a[3],a[6],a[7]);  or g3(y[2],a[4],a[5],a[6],a[7]);  endmodule | module encoder83\_DF(a, y);  input [7:0] a;  output [2:0] y;  assign y[0] = a[1] | a[3] | a[5] | a[7];  assign y[1] = a[2] | a[3] | a[6] | a[7];  assign y[2] = a[4] | a[5] | a[6] | a[7];  endmodule | module encoder83\_BH(a, y);  input [7:0] a;  output reg [2:0] y;  always @(\*) begin  if (a[7]) y <= 3'b111;  else if (a[6]) y <= 3'b110;  else if (a[5]) y <= 3'b101;  else if (a[4]) y <= 3'b100;  else if (a[3]) y <= 3'b011;  else if (a[2]) y <= 3'b010;  else if (a[1]) y <= 3'b001;  else y <= 3'b000;  end  endmodule |
| **Testbench** | | |
| module encoder83\_testbench;  reg [7:0] a;  wire [2:0] y\_GL, y\_DF, y\_BEH;  encoder83\_GL gl\_encoder (.a(a), .y(y\_GL));  encoder83\_DF df\_encoder (.a(a), .y(y\_DF));  encoder83\_BH beh\_encoder (.a(a), .y(y\_BEH));  initial begin  $monitor("Time = %0d | a = %b | GL = %b | DF = %b | BEH = %b",  $time, a, y\_GL, y\_DF, y\_BEH);  a = 8'b00000000; #10;  a = 8'b10000000; #10;  a = 8'b01000000; #10;  a = 8'b00100000; #10;  a = 8'b00010000; #10;  a = 8'b00001000; #10;  a = 8'b00000100; #10;  a = 8'b00000010; #10;  a = 8'b00000001; #10;  $finish;  end  endmodule | | |
|  | | |
|  | | |

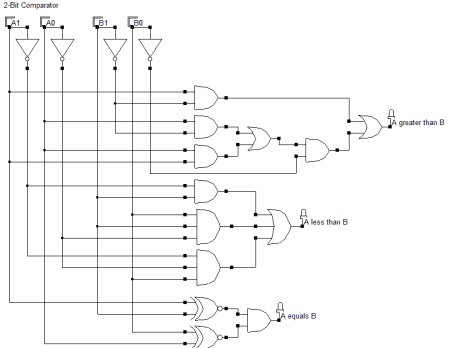
**6) 3:8 Decoder**





|  |  |  |
| --- | --- | --- |
| **Top module** | | |
| module lab2\_ex6(SW,LEDG,LEDR);  input[17:0]SW;  output[7:0] LEDG;  output[17:0] LEDR;    assign LEDR = SW;    dec38\_GL(.a(SW[2:0]),.d(LEDG[7:0]));  //dec38\_DF(.a(SW[2:0]),.d(LEDG[7:0]));  //dec38\_BH(.a(SW[2:0]),.d(LEDG[7:0]));    endmodule | | |
| **Structural** | **Dataflow** | **Behavior** |
| module dec38\_GL(a,d);  input [2:0] a;  output [7:0] d;  wire [2:0] n\_a;  not g1(n\_a[0],a[0]);  not g2(n\_a[1],a[1]);  not g3(n\_a[2],a[2]);    and g4(d[0], n\_a[2], n\_a[1], n\_a[0]);  and g5(d[1], n\_a[2], n\_a[1], a[0]);  and g6(d[2], n\_a[2], a[1], n\_a[0]);  and g7(d[3], n\_a[2], a[1], a[0]);  and g8(d[4], a[2], n\_a[1], n\_a[0]);  and g9(d[5], a[2], n\_a[1], a[0]);  and g10(d[6], a[2], a[1], n\_a[0]);  and g11(d[7], a[2], a[1], a[0]);  endmodule | module dec38\_DF(a, d);  input [2:0] a;  output [7:0] d;  assign d[0] = ~a[2] & ~a[1] & ~a[0];  assign d[1] = ~a[2] & ~a[1] & a[0];  assign d[2] = ~a[2] & a[1] & ~a[0];  assign d[3] = ~a[2] & a[1] & a[0];  assign d[4] = a[2] & ~a[1] & ~a[0];  assign d[5] = a[2] & ~a[1] & a[0];  assign d[6] = a[2] & a[1] & ~a[0];  assign d[7] = a[2] & a[1] & a[0];  endmodule | module dec38\_BH(a,d);  input [2:0] a;  output reg [7:0] d;  always @( a )  begin  d=8'd0;  case (a)  3'b000: d[0]=1'b1;  3'b001: d[1]=1'b1;  3'b010: d[2]=1'b1;  3'b011: d[3]=1'b1;  3'b100: d[4]=1'b1;  3'b101: d[5]=1'b1;  3'b110: d[6]=1'b1;  3'b111: d[7]=1'b1;  default: d=8'd0;  endcase  end  endmodule |
| **Testbench** | | |
| module dec38\_testbench;  reg [2:0] a;  wire [7:0] d\_GL, d\_DF, d\_BEH;  dec38\_GL gl\_decoder (.a(a), .d(d\_GL));  dec38\_DF df\_decoder (.a(a), .d(d\_DF));  dec38\_BH beh\_decoder (.a(a), .d(d\_BEH));  initial begin  $monitor("Time = %0d | a = %b | GL = %b | DF = %b | BEH = %b",  $time, a, d\_GL, d\_DF, d\_BEH);  a = 3'b000; #10;  a = 3'b001; #10;  a = 3'b010; #10;  a = 3'b011; #10;  a = 3'b100; #10;  a = 3'b101; #10;  a = 3'b110; #10;  a = 3'b111; #10;  $finish;  end  endmodule | | |
|  | | |
|  | | |

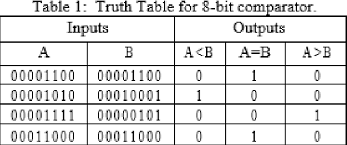
**7) 4 Bit Comparator**



|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **A3B3** | **A2B2** | **A1B1** | **A0B0** | **A>B** | **A<B** | **A=B** |
| A3>B3 | x | x | x | 1 | 0 | 0 |
| A3<B3 | x | x | x | 0 | 1 | 0 |
| A3=B3 | A2>B2 | x | x | 1 | 0 | 0 |
| A3=B3 | A2<B2 | x | x | 0 | 1 | 0 |
| A3=B3 | A2=B2 | A1>B1 | x | 1 | 0 | 0 |
| A3=B3 | A2=B2 | A1<B1 | x | 0 | 1 | 0 |
| A3=B3 | A2=B2 | A1=B1 | A0>B0 | 1 | 0 | 0 |
| A3=B3 | A2=B2 | A1=B1 | A0<B0 | 0 | 1 | 0 |
| A3=B3 | A2=B2 | A1=B1 | A0=B0 | 0 | 0 | 1 |

|  |  |  |
| --- | --- | --- |
| **Top module** | | |
| module lab2\_ex7(SW,LEDG,LEDR);  input[17:0]SW;  output[7:0] LEDG;  output[17:0] LEDR;    assign LEDR = SW;    comp4bit\_GL(.a(SW[7:4]),.b(SW[3:0]),.d(LEDG[7:5]));  //comp4bit\_DF(.a(SW[7:4]),.b(SW[3:0]),.d(LEDG[7:5]));  //comp4bit\_BH(.a(SW[7:4]),.b(SW[3:0]),.d(LEDG[7:5]));  endmodule | | |
| **Structural** | **Dataflow** | **Behavior** |
| module comp4bit\_GL(a,b,d);  input [3:0]a;  input [3:0]b;  output [2:0]d;  wire [7:0]y;    wire [3:0] n\_a;  wire [3:0] n\_b;  wire [3:0] w;    not g1\_0(n\_a[0], a[0]);  not g1\_1(n\_a[1], a[1]);  not g1\_2(n\_a[2], a[2]);  not g1\_3(n\_a[3], a[3]);  not g2\_0(n\_b[0], b[0]);  not g2\_1(n\_b[1], b[1]);  not g2\_2(n\_b[2], b[2]);  not g2\_3(n\_b[3], b[3]);    xnor g3(w[3],a[3],b[3]);  xnor g4(w[2],a[2],b[2]);  xnor g5(w[1],a[1],b[1]);  xnor g6(w[0],a[0],b[0]);    and g7(d[0],w[0],w[1],w[2],w[3]);  and g8(y[0],n\_a[3],b[3]);  and g9(y[1],n\_a[2],b[2],w[3]);  and g10(y[2],n\_a[1],b[1],w[3],w[2]);  and g11(y[3],n\_a[0],b[0],w[3],w[2],w[1]);    and g12(y[4],a[3],n\_b[3]);  and g13(y[5],a[2],n\_b[2],w[3]);  and g14(y[6],a[1],n\_b[1],w[3],w[2]);  and g15(y[7],a[0],n\_b[0],w[3],w[2],w[1]);    or g16(d[1],y[0],y[1],y[2],y[3]);  or g17(d[2],y[4],y[5],y[6],y[7]);  endmodule | module comp4bit\_DF (a, b, d);  input [3:0] a;  input [3:0] b;  output [2:0] d;  assign d[0] = ~(a[3] ^ b[3]) & ~(a[2] ^ b[2]) & ~(a[1] ^ b[1]) & ~(a[0] ^ b[0]);  assign d[1] = (a[3] & ~b[3]) | (a[2] & ~b[2] & ~(a[3] ^ b[3])) | (a[1] & ~b[1] & ~(a[3] ^ b[3]) & ~(a[2] ^ b[2])) | (a[0] & ~b[0] & ~(a[3] ^ b[3]) & ~(a[2] ^ b[2]) & ~(a[1] ^ b[1]));  assign d[2] = (~a[3] & b[3]) | (~a[2] & b[2] & ~(a[3] ^ b[3])) | (~a[1] & b[1] & ~(a[3] ^ b[3]) & ~(a[2] ^ b[2])) | (~a[0] & b[0] & ~(a[3] ^ b[3]) & ~(a[2] ^ b[2]) & ~(a[1] ^ b[1]));  endmodule | module comp4bit\_BH(a, b, d);  input [3:0] a;  input [3:0] b;  output reg [2:0] d;  always @(a or b) begin  d = 3'b000;  if (a == b)  d[0] = 1;  else if (a > b)  d[1] = 1;  else  d[2] = 1;  end  endmodule |
| **Testbench** | | |
| module comp4bit\_testbench;  reg [3:0] a;  reg [3:0] b;  wire [2:0] d\_GL, d\_DF, d\_BEH;  comp4bit\_GL gl\_comparator (.a(a), .b(b), .d(d\_GL));  comp4bit\_DF df\_comparator (.a(a), .b(b), .d(d\_DF));  comp4bit\_BH beh\_comparator (.a(a), .b(b), .d(d\_BEH));  initial begin  $monitor("Time = %0d | a = %b | b = %b | GL = %b | DF = %b | BEH = %b",  $time, a, b, d\_GL, d\_DF, d\_BEH);  a = 4'b0000; b = 4'b0000; #10;  a = 4'b1000; b = 4'b0000; #10;  a = 4'b0111; b = 4'b1000; #10;  a = 4'b1100; b = 4'b1100; #10;  a = 4'b1010; b = 4'b0111; #10;  $finish;  end  endmodule | | |
|  | | |
|  | | |

**8) 8 Bit& N bit Comparator**



|  |
| --- |
| **Answer code** |
| module lab2\_ex8(  input [17:0] SW,  output [17:0] LEDR,  output [7:0] LEDG  );  assign LEDR = SW;  parameter N = 8;  compNbit DUT (  .A(SW[N-1:0]),  .B(SW[N\*2-1:N]),  .eq(LEDG[0]),  .gt(LEDG[1]),  .lt(LEDG[2])  );  defparam DUT.N = N;  endmodule  module compNbit #(  parameter N = 8  )(  input [N-1:0] A,  input [N-1:0] B,  output eq,  output gt,  output lt  );  assign eq = (A == B);  assign gt = (A > B);  assign lt = (A < B);  endmodule |
| **Testbench** |
| module compNbit\_testbench;  reg [7:0] SW;  wire [7:0] LEDR;  wire [2:0] LEDG;  compNbit DUT (  .SW(SW),  .LEDR(LEDR),  .LEDG(LEDG)  );  initial begin  $monitor("Time = %0d | SW = %b | LEDR = %b | LEDG = %b", $time, SW, LEDR, LEDG);  SW = 8'b00000000; #10;  SW = 8'b10000000; #10;  SW = 8'b11111111; #10;  SW = 8'b01111111; #10;  SW = 8'b01010101; #10;  $finish;  end  endmodule |
|  |
| ss |