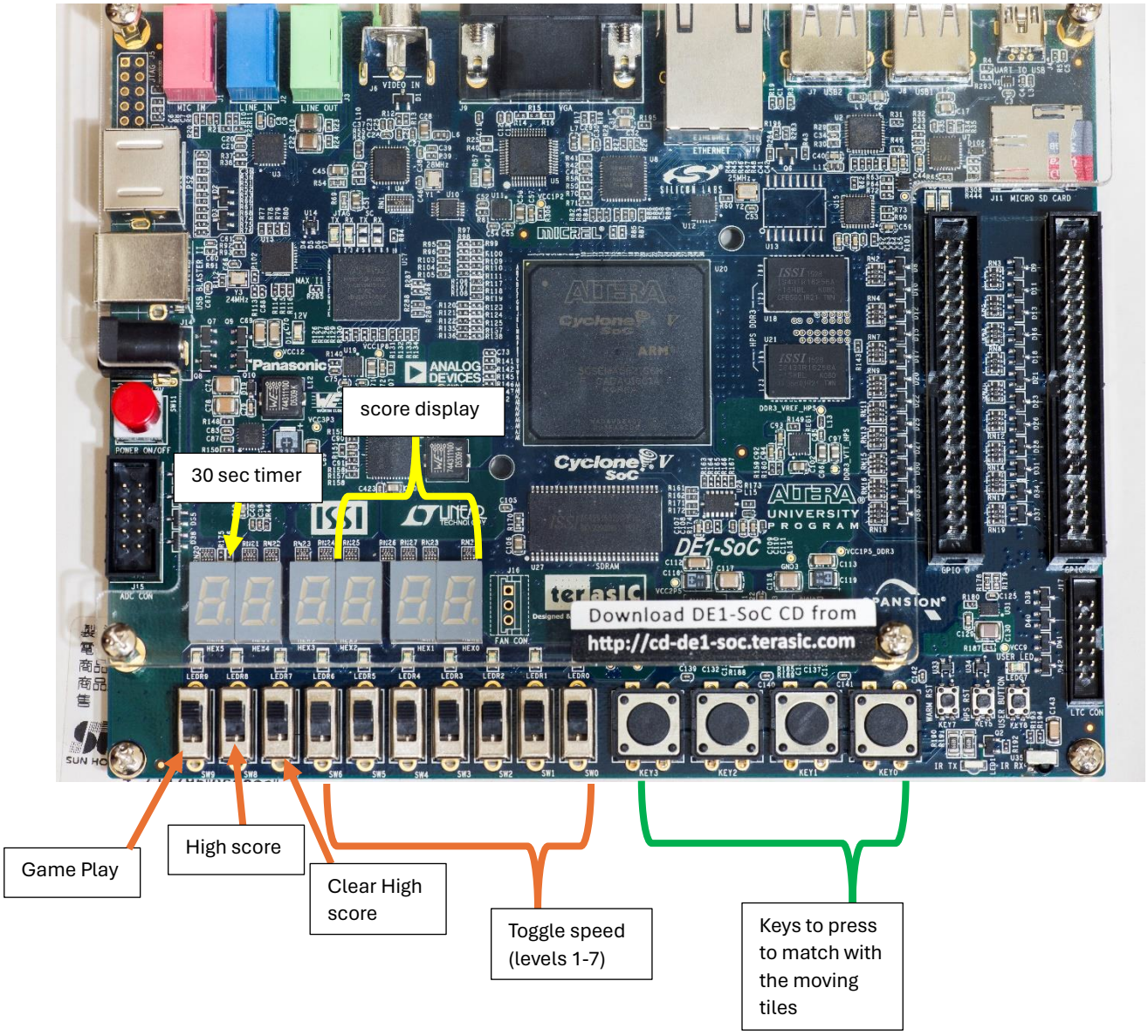
DANCING WITH YOUR THUMBS – ANKITH TUNUGUNTLA

EE – 271 LAB #8

The game:

Dancing with your thumbs involves using your thumbs to hit different keys on the DE1SoC to perfectly time the tiles moving upward. You earn points with each key pressed in sync with the moving tiles under a time limit of 30 seconds.

Controls:



Gameplay:

Turn on SW9 to start the game where the tiles shall move at a default speed. You can turn on SW8 to view the high score at any time. SW7 clears the high score and resets it to zero. When the game starts, a thirty second timer counts down beyond which the game will not count the score any further.

You can use:

- SW[6:0] – to turn on different levels of speed. The game will run at a particular speed when a particular switch is on. If no switch in this range is switched on, then the game runs at the default speed.
- KEY[3:0] – to press when a tile moves to the top of the respective bank. Each key corresponds to a 16x4 section on the LED array. Key 3 corresponds to the first section, Key 2 corresponds to the second section, and so on and so forth.

Scoring:

For every tile you match correctly with the appropriate key, you gain +2 points. Matching incorrectly by a row with a key press shall gain you +1 points. Matching incorrectly by not pressing a key when the tile is at the top or pressing incorrectly will earn you a deduction of 2 points.

Extra features absent from the spec:

- Timer – 30 second timer that counts to 0.
- High score and the option to clear it

Market and Usability Analysis:

- Usability and ergonomics – I have used the entire LED array split into four different parts which involves tiles in alternating colors of red and green so the user can see the difference between the banks of tiles. The keys when pressed turn the top row orange to indicate the key being pressed. The option to change speeds, see the scores on the hexes add to the ergonomics of the entire design.
- Suitability for goals – I have made a game for a project, so the system is indeed fun and the game play is logical. The addition of a high score and a time limit induces a competitive nature within the game.
- Cost and resource utilization – The resource utilization is high due to several modules, LFSRs, LED modules, etc.

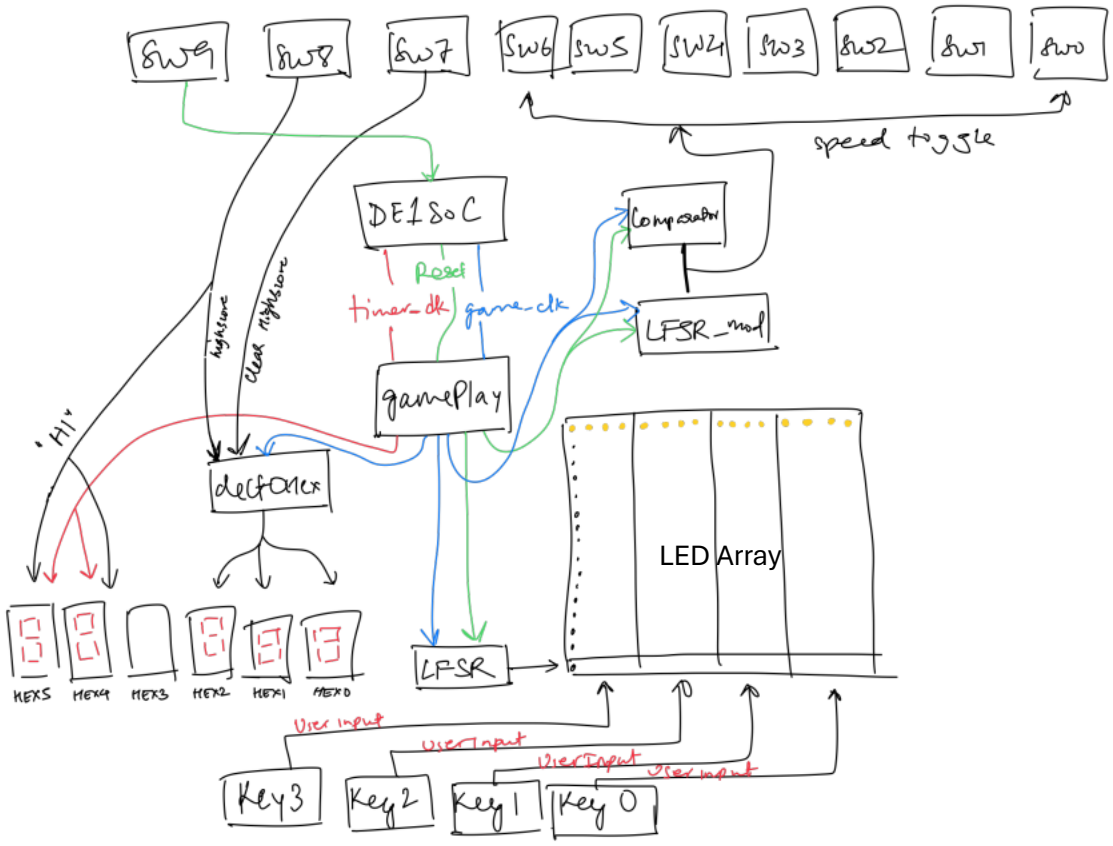
y Node	Combinational ALUTs	Dedicated Logic R
	748 (26)	147 (0)
er|	28 (28)	4 (4)

- Public Health, safety and welfare – the system poses no risk or injury and is a low-voltage game. The game is a positive harmless recreational activity and appeals to all ethnics, cultures and age groups and does not require any prior knowledge other than the instructions given in this manual.

Time taken:

3 days approx. for development of idea/schematic, coding, simulations and debugging

Block diagram:



```
module gamePlay(
   input logic clk,        // Clock[22]
   input logic timer_clk,   // Clock[25]
   input logic reset,
   input logic highScoreDisplay,
   input logic clearHighScore,
   input logic [6:0] speedSW,
   input logic key3,
   input logic key2,
   input logic key1,
   input logic key0,
   output logic [15:0][15:0] RedPixels,
   output logic [15:0][15:0] GrnPixels,
   output logic [6:0] hex2,
   output logic [6:0] hex1,
   output logic [6:0] hex0,
   output logic [6:0] hex5,
   output logic [6:0] hex4,
   output logic [6:0] hex3
);

   assign hex3 = 7'b1111111; // HEX 3 NOT USED

   // LFSR FOR TILE PATTERN
   logic [15:0] lfsr_pattern;
   LFSR lfsr_inst (.Out(lfsr_pattern), .clk(clk), .rst(reset));

   logic [8:0] score; // game score
   logic [8:0] highScore; // all-time highscore initialized to 0
```

```
  logic [6:0] hex2_2, hex1_2, hex0_2; // hexes for score
  logic [6:0] hex2_1, hex1_1, hex0_1; // hexes for high score

  logic [15:0][15:0] nextRedPixels, nextGrnPixels; // updated red and green pixels

  // Timer variables
  logic [4:0] timer;
  logic [6:0] hex5_disp, hex4_disp; // hexes for timer

  decToHex timer_to_hex (.score({4'b0, timer}), .hex2(), .hex1(hex5_disp), .hex0(hex4_disp)); // conversion of timer to hex display

  // Score to hex conversion
  decToHex dh1 (.score(score), .hex2(hex2_1), .hex1(hex1_1), .hex0(hex0_1)); // conversion of score to hex display
  decToHex dh2 (.score(highScore), .hex2(hex2_2), .hex1(hex1_2), .hex0(hex0_2)); // conversion of high score to hex display

  logic gameOver;

  logic [9:0] lfsr_output; // LFSR FOR SPEED TOGGLE
  logic speed; // speed signal

  LFSR_mod l (.Clock(clk), .Reset(reset), .Out(lfsr_output));
  comparator comp1 (.a({3'b0, speedSW[6:0]}), .b(lfsr_output), .out(speed));

  logic [10:0] counter;

  // Timer decrement logic
```

```
always_ff @(posedge timer_clk or posedge reset) begin
  if (reset) begin
    timer <= 30;
  end else begin

    if (timer != 0) begin
      timer <= timer - 1;
    end
  end
end

// Main logic with clk for key responsiveness and tile generation
always_ff @(posedge clk or posedge reset) begin
  if (reset) begin
    nextRedPixels <= '{default: 16'b0};
    nextGrnPixels <= '{default: 16'b0};
    RedPixels <= '{default: 16'b0};
    GrnPixels <= '{default: 16'b0};
    score <= 0;
  end else begin
    // Shift pixels up
    for (int i = 1; i < 16; i++) begin
      nextRedPixels[i-1] = RedPixels[i];
      nextGrnPixels[i-1] = GrnPixels[i];
    end

                                  // Update RedPixels and GrnPixels
    RedPixels <= nextRedPixels;
    GrnPixels <= nextGrnPixels;
```

```
                                    // Key handling
if (~key3) begin
   GrnPixels[0][15:12] <= 4'b1111;
   RedPixels[0][15:12] <= 4'b1111;
end
if (~key2) begin
   GrnPixels[0][11:8] <= 4'b1111;
   RedPixels[0][11:8] <= 4'b1111;
end
if (~key1) begin
   GrnPixels[0][7:4] <= 4'b1111;
   RedPixels[0][7:4] <= 4'b1111;
end
if (~key0) begin
   GrnPixels[0][3:0] <= 4'b1111;
   RedPixels[0][3:0] <= 4'b1111;
end



// Assign new random pattern to the first row
nextRedPixels[15][15:12] <= {4{lfsr_pattern[0]}};
nextGrnPixels[15][11:8] <= {4{lfsr_pattern[4]}};
nextRedPixels[15][7:4] <= {4{lfsr_pattern[8]}};
nextGrnPixels[15][3:0] <= {4{lfsr_pattern[12]}};



// Light hit detection
if ((RedPixels[0][15:12] == 4'b1111 & ~key3) ||
```

```
        (GrnPixels[0][11:8] == 4'b1111 & ~key2) ||
        (RedPixels[0][7:4] == 4'b1111 & ~key1) ||
        (GrnPixels[0][3:0] == 4'b1111 & ~key0)) begin
        score <= score + 2;
                                        //highScore <= highScore + 2;
      end else if ((RedPixels[1][15:12] == 4'b1111 & RedPixels[0][15:12] == 4'b0000 & ~key3) ||
            (GrnPixels[1][11:8] == 4'b1111 & GrnPixels[0][11:8] == 4'b0000 & ~key2) ||
            (RedPixels[1][7:4] == 4'b1111 & RedPixels[0][7:4] == 4'b0000 & ~key1) ||
            (GrnPixels[1][3:0] == 4'b1111 & GrnPixels[0][3:0] == 4'b0000 & ~key0)) begin
        score <= score + 1;
                                        //highScore <= highScore + 1;
      end else if (((RedPixels[1][15:12] == 4'b0000 & RedPixels[0][15:12] == 4'b0000 & ~key3) ||
            (GrnPixels[1][11:8] == 4'b0000 & RedPixels[0][11:8] == 4'b0000 & ~key2) ||
            (RedPixels[1][7:4] == 4'b0000 & RedPixels[0][7:4] == 4'b0000 & ~key1) ||
            (GrnPixels[1][3:0] == 4'b0000 & RedPixels[0][3:0] == 4'b0000 & ~key0)) & score >= 2)
begin
        score <= score - 2;
                                        //highScore <= highScore - 2;
      end

      if (clearHighScore & ~highScoreDisplay & ~reset) begin
        highScore <= 0;
      end else if (score > highScore) begin
        highScore <= score;
      end
    end
  end

  always_comb begin
```

```
    if (highScoreDisplay) begin
      // Display high score
      hex5 = 7'b0001001; // 'H'
      hex4 = 7'b1001111; // 'I'
      hex2 = hex2_2;
      hex1 = hex1_2;
      hex0 = hex0_2;
    end else begin
      // Normal game play display
      hex5 = hex5_disp;
      hex4 = hex4_disp;
      hex2 = hex2_1;
      hex1 = hex1_1;
      hex0 = hex0_1;
    end
  end

endmodule

module gamePlay_testbench();

  // Inputs
  reg clk;
  reg timer_clk;
  reg reset;
  reg highScoreDisplay;
  reg clearHighScore;
  reg [6:0] speedSW;
  reg key3;
```

```
reg key2;
reg key1;
reg key0;

// Outputs
wire [15:0][15:0] RedPixels;
wire [15:0][15:0] GrnPixels;
wire [6:0] hex2;
wire [6:0] hex1;
wire [6:0] hex0;
wire [6:0] hex5;
wire [6:0] hex4;
wire [6:0] hex3;


gamePlay dut (
  .clk(clk),
  .timer_clk(timer_clk),
  .reset(reset),
  .highScoreDisplay(highScoreDisplay),
  .clearHighScore(clearHighScore),
  .speedSW(speedSW),
  .key3(key3),
  .key2(key2),
  .key1(key1),
  .key0(key0),
  .RedPixels(RedPixels),
  .GrnPixels(GrnPixels),
  .hex2(hex2),
```

```
    .hex1(hex1),
    .hex0(hex0),
    .hex5(hex5),
    .hex4(hex4),
    .hex3(hex3)
);

// Clock generation
initial begin
    clk = 0;
    forever #5 clk = ~clk; // 100 MHz clock
end

initial begin
    timer_clk = 0;
    forever #50 timer_clk = ~timer_clk; // Slower clock for timer
end

// Initial setup and stimulus
initial begin
    // Initialize Inputs
    reset = 1;
    highScoreDisplay = 0;
    clearHighScore = 0;
    speedSW = 7'b0;
    key3 = 1;
    key2 = 1;
    key1 = 1;
    key0 = 1;
```

```
// Wait for global reset to finish
#100;

reset = 0;

// Simulate key presses
#100;
key3 = 0; #20; key3 = 1; // Press and release key3
#100;
key2 = 0; #20; key2 = 1; // Press and release key2
#100;
key1 = 0; #20; key1 = 1; // Press and release key1
#100;
key0 = 0; #20; key0 = 1; // Press and release key0

// Simulate high score display
#200;
highScoreDisplay = 1;
#100;
highScoreDisplay = 0;

// Simulate clear high score
#200;
clearHighScore = 1;
#100;
clearHighScore = 0;

// Simulate speed switch changes
```

```
    #200;
    speedSW = 7'b0111111;
    #100;
    speedSW = 7'b1011111;
    #100;
    speedSW = 7'b1101111;
    #100;
    speedSW = 7'b1110111;
    #100;
    speedSW = 7'b1111011;
    #100;
    speedSW = 7'b1111101;
    #100;
    speedSW = 7'b1111110;

    // Finish simulation
    #500;
    $finish;
  end

  // Monitor outputs
  initial begin
    $monitor("Time: %0t, score: %d, highScore: %d, timer: %d, hex0: %b, hex1: %b, hex2: %b,
hex4: %b, hex5: %b",
      $time, dut.score, dut.highScore, dut.timer, hex0, hex1, hex2, hex4, hex5);
  end

endmodule
```

```
module DE1_SoC (
   output logic [6:0]  HEX0, HEX1, HEX2, HEX3, HEX4, HEX5,
   output logic [9:0]  LEDR,
   input  logic [3:0]  KEY,
   input  logic [9:0]  SW,
   output logic [35:0] GPIO_1,
   input  logic CLOCK_50
);

   // Turn off HEX displays




   // Clock divider
   logic [31:0] Clock;
   logic SYSTEM_CLOCK;
   logic game_clk;
         logic timer_clk;
   clock_divider divider (.clock(CLOCK_50), .divided_clocks(Clock));
   assign SYSTEM_CLOCK = Clock[14]; // 1526 Hz clock signal
         assign timer_clk = Clock[25];



   // Speed control logic
   logic [4:0] speed;

   always_comb begin
      if (SW[6]) begin
```

```
      speed = 5'd19;
    end else if (SW[5]) begin
      speed = 5'd20;
    end else if (SW[4]) begin
      speed = 5'd21;
    end else if (SW[3]) begin
      speed = 5'd22;
    end else if (SW[2]) begin
      speed = 5'd23;
    end else if (SW[1]) begin
      speed = 5'd24;
    end else if (SW[0]) begin
      speed = 5'd25;
    end else begin
      speed = 5'd22; // Default speed if no switch is turned on
    end
  end

        assign game_clk = Clock[speed];


// LED board driver
logic [15:0][15:0] RedPixels; // 16 x 16 array representing red LEDs
logic [15:0][15:0] GrnPixels; // 16 x 16 array representing green LEDs
logic reset; // reset signal

assign reset = ~SW[9];

// LED Driver instantiation
```

```
  LEDDriver Driver (
    .GPIO_1(GPIO_1),
    .CLK(SYSTEM_CLOCK),
    .RST(reset),
    .EnableCount(1'b1),
    .RedPixels(RedPixels),
    .GrnPixels(GrnPixels)
  );


            //gamePlay g (.clk(game_clk), .reset, .highScoreDisplay(SW[8]),
.clearHighScore(SW[7]), .key3(KEY[3]), .key2(KEY[2]), .key1(KEY[1]), .key0(KEY[0]), .RedPixels,
.GrnPixels, .hex2(HEX2), .hex1(HEX1), .hex0(HEX0), .hex5(HEX5), .hex4(HEX4), .hex3(HEX3));


            gamePlay g (.clk(game_clk), .timer_clk, .reset, .highScoreDisplay(SW[8]),
.clearHighScore(SW[7]), .speedSW(SW[6:0]), .key3(KEY[3]), .key2(KEY[2]), .key1(KEY[1]),
.key0(KEY[0]),.RedPixels, .GrnPixels, .hex2(HEX2), .hex1(HEX1), .hex0(HEX0), .hex5(HEX5),
.hex4(HEX4), .hex3(HEX3));

endmodule
```

```
module LFSR_mod(Clock, Reset, Out);

input logic Clock, Reset;
output logic [9:0] Out;

logic xnor_out;
assign xnor_out = (Out[3] ~^ Out[0]);

logic LFSR;
        always_ff @(posedge Clock) begin
                if (Reset)
      Out <= 10'b0;
   else
      Out <= {xnor_out, Out[9:1]};
        end
endmodule

module LFSR_mod_testbench();
  logic clk, Reset;
  logic [9:0] Out;
  LFSR_mod l(.Clock(clk), .Reset, .Out);

         parameter CLOCK_PERIOD=100;
   initial begin
    clk <= 0;
   forever #(CLOCK_PERIOD/2) clk <= ~clk; // Forever toggle the clock
   end
```

```
  // Set up the inputs to the design. Each line is a clock cycle.
  initial begin
                        repeat(1); @(posedge clk);
                        Reset <= 1; repeat(1) @(posedge clk); // Always reset FSMs at start
                        Reset <= 0; repeat(100) @(posedge clk);
                        $stop; // End the simulation.
  end
endmodule
```

```
module LFSR(Out, clk, rst);



                input logic clk, rst;
                output logic [15:0] Out;

                logic xnor_out;
                assign xnor_out = (Out[0] ~^ Out[1] ~^ Out[3] ~^ Out[12]);

                        always_ff @(posedge clk) begin
                                if (rst)
                                                Out <= 16'b0;
                                else
                                                Out <= {xnor_out, Out[15:1]};
                        end
endmodule

module LFSR_testbench();
  logic clk, Reset;
  logic [15:0] Out;
  LFSR l(.clk, .rst(Reset), .Out);

          parameter CLOCK_PERIOD=100;
   initial begin
    clk <= 0;
   forever #(CLOCK_PERIOD/2) clk <= ~clk; // Forever toggle the clock
   end

  // Set up the inputs to the design. Each line is a clock cycle.
```

```
  initial begin
                        repeat(1); @(posedge clk);
                        Reset <= 1; repeat(1) @(posedge clk); // Always reset FSMs at start
                        Reset <= 0; repeat(100) @(posedge clk);
                        $stop; // End the simulation.
  end
endmodule
```

```
module decToHex (score, hex2, hex1, hex0);

input [8:0] score; //score as input
output logic [6:0] hex2, hex1, hex0;

logic [31:0] hund, ten, unit;



always_comb begin
hund = score /100; //find hex2 value

ten = (score /10)%10; //find hex1 value

unit = score % 10; //find hex0 value

end

always_comb begin
    if (hund == 1) begin
      hex2 = 7'b1111001;
    end
    else if (hund == 2) begin
      hex2 = 7'b0100100;
    end
                else if (hund == 3) begin
      hex2 = 7'b0110000;
    end
                else if (hund == 4) begin
      hex2 = 7'b0011001;
```

```
    end
                else if (hund == 5) begin
      hex2 = 7'b0010010;
    end
                else if (hund == 6) begin
      hex2 = 7'b0000010;
    end
                else if (hund == 7) begin
      hex2 = 7'b1111000;
    end
                else if (hund == 8) begin
      hex2 = 7'b0000000;
    end
                else if (hund == 9) begin
      hex2 = 7'b0010000;
    end
                else if (hund == 0) begin
      hex2 = 7'b1000000;
    end
                else begin
                  hex2 =  7'b1;
                end
  end

always_comb begin
    if (ten == 1) begin
      hex1 = 7'b1111001;
    end
    else if (ten == 2) begin
```

```
  hex1 = 7'b0100100;
end
                else if (ten == 3) begin
  hex1 = 7'b0110000;
end
                else if (ten == 4) begin
  hex1 = 7'b0011001;
end
                else if (ten == 5) begin
  hex1 = 7'b0010010;
end
                else if (ten == 6) begin
  hex1 = 7'b0000010;
end
                else if (ten == 7) begin
  hex1 = 7'b1111000;
end
                else if (ten == 8) begin
  hex1 = 7'b0000000;
end
                else if (ten == 9) begin
  hex1 = 7'b0010000;
end
                else if (ten == 0) begin
  hex1 = 7'b1000000;
end
                else begin
                  hex1 =  7'b1;
                end
```

```
  end


always_comb begin
    if (unit == 1) begin
      hex0 = 7'b1111001;
    end
    else if (unit == 2) begin
      hex0 = 7'b0100100;
    end
                else if (unit == 3) begin
      hex0 = 7'b0110000;
    end
                else if (unit == 4) begin
      hex0 = 7'b0011001;
    end
                else if (unit == 5) begin
      hex0 = 7'b0010010;
    end
                else if (unit == 6) begin
      hex0 = 7'b0000010;
    end
                else if (unit == 7) begin
      hex0 = 7'b1111000;
    end
                else if (unit == 8) begin
      hex0 = 7'b0000000;
    end
                else if (unit == 9) begin
      hex0 = 7'b0010000;
```

```
    end
                else if (unit == 0) begin
      hex0 = 7'b1000000;
    end
                else begin
                  hex0 = 7'b1;
                end
  end

endmodule

module decToHex_testbench();

  // Inputs
  reg [8:0] score;

  // Outputs
  wire [6:0] hex2;
  wire [6:0] hex1;
  wire [6:0] hex0;

  decToHex dut (
    .score(score),
    .hex2(hex2),
    .hex1(hex1),
    .hex0(hex0)
  );

  // Display values
```

```
  initial begin
    $monitor("Time=%0d score=%d -> hex2=%b hex1=%b hex0=%b", $time, score, hex2, hex1, hex0);
  end

  // Stimulus
  initial begin
    // Initialize Inputs
    score = 0;

    // Test cases
    #10 score = 9;
    #10 score = 10;
    #10 score = 21;
    #10 score = 32;
    #10 score = 43;
    #10 score = 54;
    #10 score = 65;
    #10 score = 76;
    #10 score = 87;
    #10 score = 98;
    #10 score = 99;
    #10 score = 100;
    #10 score = 111;
    #10 score = 123;
    #10 score = 134;
    #10 score = 145;
    #10 score = 156;
    #10 score = 167;
```

```
        #10 score = 178;
        #10 score = 189;
        #10 score = 200;
        #10 score = 210;
        #10 score = 255;
        #10 score = 300;
        #10 score = 345;
        #10 score = 400;
        #10 score = 450;
        #10 score = 500;
        #10 score = 555;
        #10 score = 600;
        #10 score = 650;
        #10 score = 700;
        #10 score = 750;
        #10 score = 800;
        #10 score = 850;
        #10 score = 900;
        #10 score = 950;
        #10 score = 999;

        // End simulation
        #10 $finish;
    end
endmodule
```

```
module comparator(a, b, out);

input logic [9:0]a;
input logic [9:0]b;
output logic out;

always_comb begin
      out = 0;
      for (int i = 9; i >= 0; i = i - 1) begin
        if (a[i] & ~b[i]) begin
           out = 1;
           break;
        end
        if (~a[i] & b[i]) begin
           out = 0;
           break;
        end
                                  end
   end

endmodule

module comparator_testbench();

logic [9:0]a;
logic [9:0]b;
logic out;

comparator dut(.a, .b, .out);
```

```
initial begin
a <= 10'b1111100000; //992
b <= 10'b1100101001; //809
#10;

a <= 10'b1100101001; //809
b <= 10'b1111100000; //992
#10;

a <= 10'b1111100000; //992
b <= 10'b1111100000; //992
#10;

a <= 10'b0; //0
b <= 10'b1; //1
#10;
$stop; // End the simulation.
end
endmodule
```