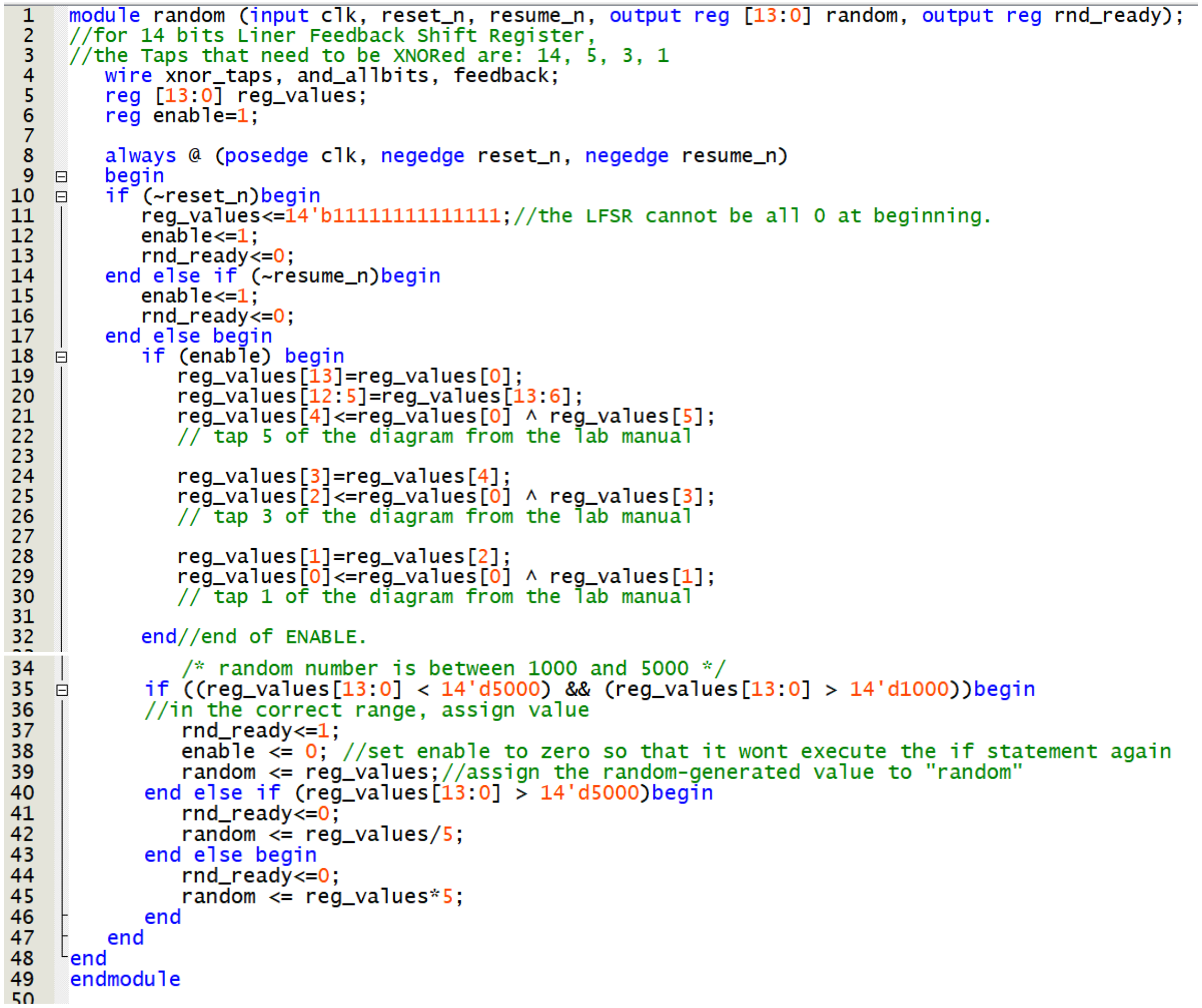
Lab 2 Report- Group 6

Zuoxiu Kang, Yimin Pang

Summary:

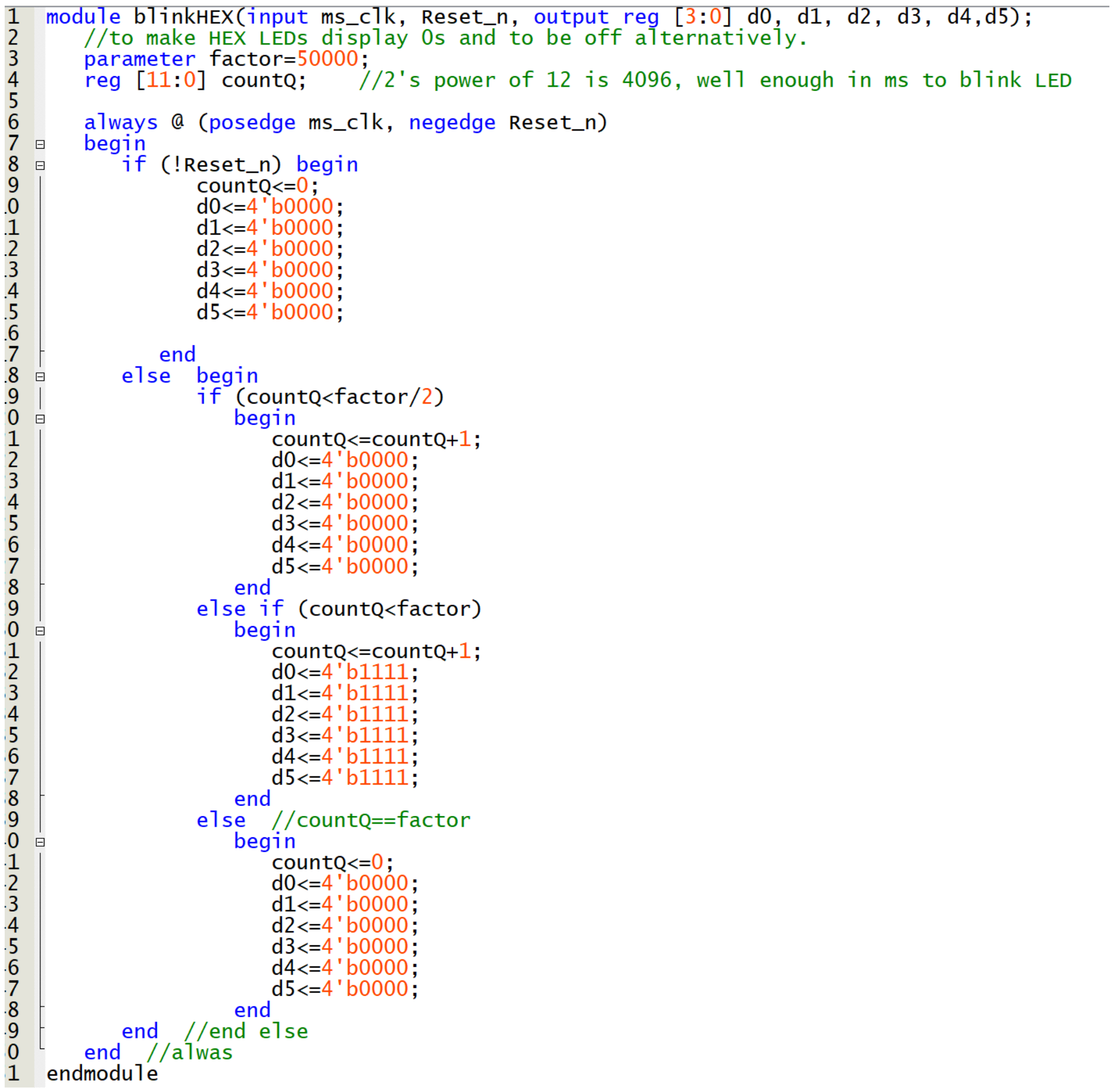
In this lab, we designed a device that allows two players to compete to test their reaction times. The user needs to press the button as soon as they see the HEX LED light up, the first person to press the button wins. If the user presses the button before the LED lights up, the display will display 111111 or 222222, and pressing the resume button will restart the game. To achieve this, we designed a clock divider, a counter, a hex-to-binary-coded-decimal converter, a seven-segment display handler, a random number generator (Galois LFSR), 4to1 mux and a top-level module that connects submodules. We coded the logic using Verilog Hardware Description Language (HDL) and implemented it on the FPGA on the DE1-SoC board.

Part 1 code: 1. Random



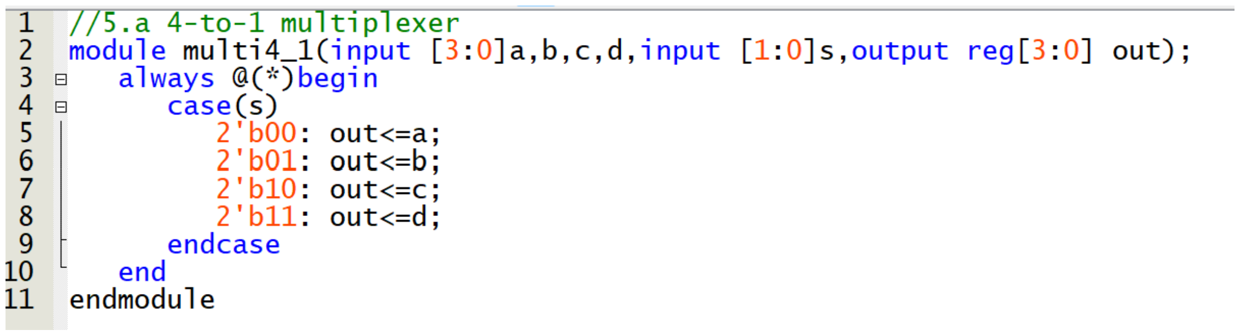
```verilog
module random (input clk, reset_n, resume_n, output reg [13:0] random, output reg rnd_ready);
//for 14 bits Liner Feedback Shift Register,
//the Taps that need to be XNORed are: 14, 5, 3, 1
   wire xnor_taps, and_allbits, feedback;
   reg [13:0] reg_values;
   reg enable=1;

   always @ (posedge clk, negedge reset_n, negedge resume_n)
   begin
   if (~reset_n)begin
      reg_values<=14'b11111111111111;//the LFSR cannot be all 0 at beginning.
      enable<=1;
      rnd_ready<=0;
   end else if (~resume_n)begin
      enable<=1;
      rnd_ready<=0;
   end else begin
      if (enable) begin
         reg_values[13]=reg_values[0];
         reg_values[12:5]=reg_values[13:6];
         reg_values[4]<=reg_values[0] ^ reg_values[5];
         // tap 5 of the diagram from the lab manual

         reg_values[3]=reg_values[4];
         reg_values[2]<=reg_values[0] ^ reg_values[3];
         // tap 3 of the diagram from the lab manual

         reg_values[1]=reg_values[2];
         reg_values[0]<=reg_values[0] ^ reg_values[1];
         // tap 1 of the diagram from the lab manual

      end//end of ENABLE.
         /* random number is between 1000 and 5000 */
      if ((reg_values[13:0] < 14'd5000) && (reg_values[13:0] > 14'd1000))begin
      //in the correct range, assign value
         rnd_ready<=1;
         enable <= 0; //set enable to zero so that it wont execute the if statement again
         random <= reg_values;//assign the random-generated value to "random"
      end else if (reg_values[13:0] > 14'd5000)begin
         rnd_ready<=0;
         random <= reg_values/5;
      end else begin
         rnd_ready<=0;
         random <= reg_values*5;
      end
   end
end
endmodule
```

2. Blinking



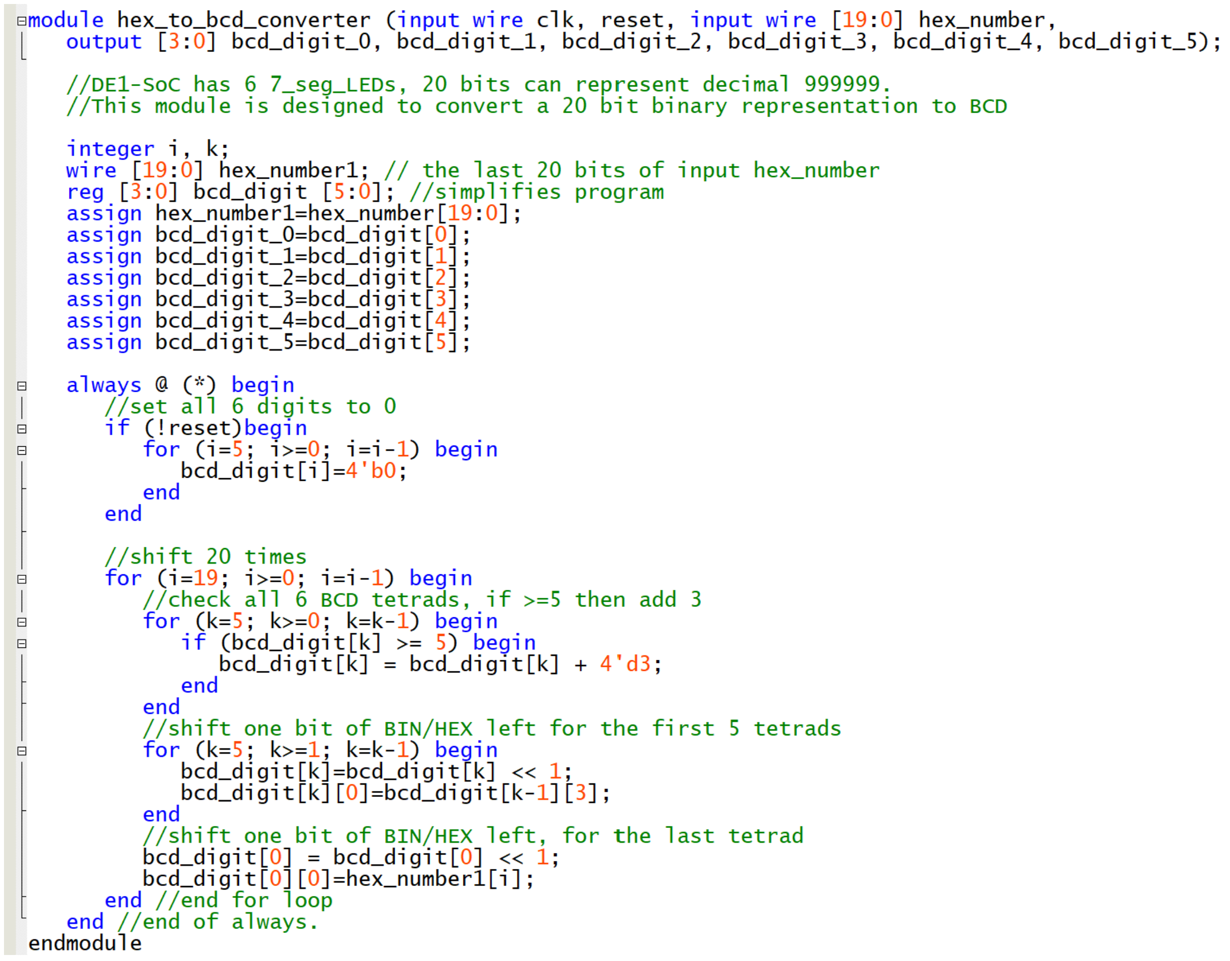
```verilog
module blinkHEX(input ms_clk, Reset_n, output reg [3:0] d0, d1, d2, d3, d4,d5);
   //to make HEX LEDs display 0s and to be off alternatively.
   parameter factor=50000;
   reg [11:0] countQ;     //2's power of 12 is 4096, well enough in ms to blink LED

   always @ (posedge ms_clk, negedge Reset_n)
   begin
      if (!Reset_n) begin
            countQ<=0;
            d0<=4'b0000;
            d1<=4'b0000;
            d2<=4'b0000;
            d3<=4'b0000;
            d4<=4'b0000;
            d5<=4'b0000;

         end
      else  begin
            if (countQ<factor/2)
               begin
                  countQ<=countQ+1;
                  d0<=4'b0000;
                  d1<=4'b0000;
                  d2<=4'b0000;
                  d3<=4'b0000;
                  d4<=4'b0000;
                  d5<=4'b0000;
               end
            else if (countQ<factor)
               begin
                  countQ<=countQ+1;
                  d0<=4'b1111;
                  d1<=4'b1111;
                  d2<=4'b1111;
                  d3<=4'b1111;
                  d4<=4'b1111;
                  d5<=4'b1111;
               end
            else  //countQ==factor
               begin
                  countQ<=0;
                  d0<=4'b0000;
                  d1<=4'b0000;
                  d2<=4'b0000;
                  d3<=4'b0000;
                  d4<=4'b0000;
                  d5<=4'b0000;
               end
      end  //end else
   end  //alwas
endmodule
```

3.Mux



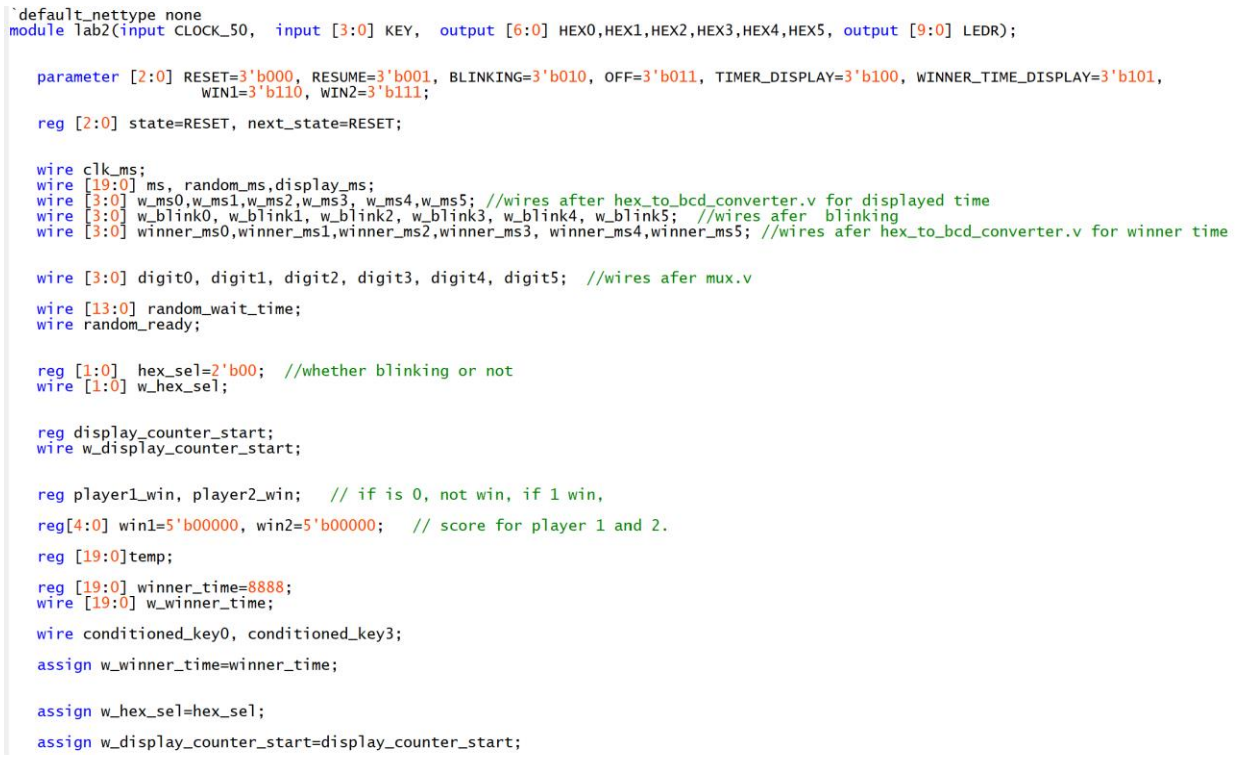
```verilog
//5.a 4-to-1 multiplexer
module multi4_1(input [3:0]a,b,c,d,input [1:0]s,output reg[3:0] out);
   always @(*)begin
      case(s)
         2'b00: out<=a;
         2'b01: out<=b;
         2'b10: out<=c;
         2'b11: out<=d;
      endcase
   end
endmodule
```

4.BCD

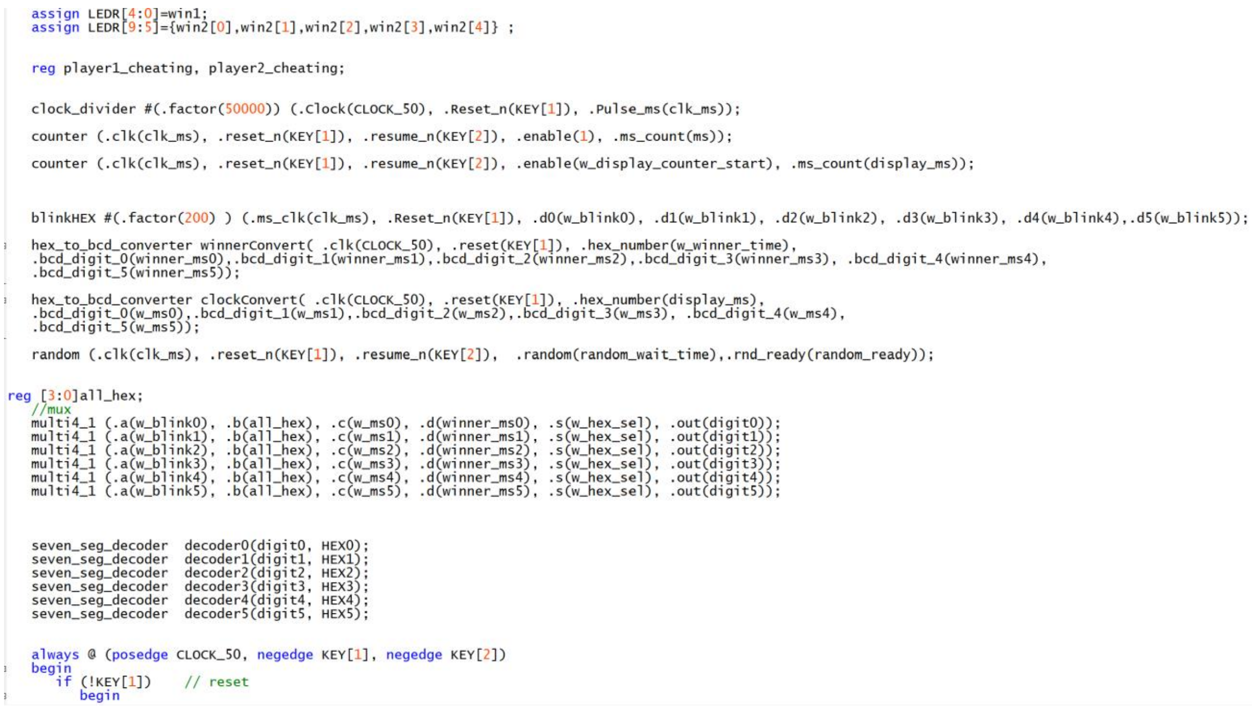


```
module hex_to_bcd_converter (input wire clk, reset, input wire [19:0] hex_number,
   output [3:0] bcd_digit_0, bcd_digit_1, bcd_digit_2, bcd_digit_3, bcd_digit_4, bcd_digit_5);

   //DE1-SoC has 6 7_seg_LEDs, 20 bits can represent decimal 999999.
   //This module is designed to convert a 20 bit binary representation to BCD

   integer i, k;
   wire [19:0] hex_number1; // the last 20 bits of input hex_number
   reg [3:0] bcd_digit [5:0]; //simplifies program
   assign hex_number1=hex_number[19:0];
   assign bcd_digit_0=bcd_digit[0];
   assign bcd_digit_1=bcd_digit[1];
   assign bcd_digit_2=bcd_digit[2];
   assign bcd_digit_3=bcd_digit[3];
   assign bcd_digit_4=bcd_digit[4];
   assign bcd_digit_5=bcd_digit[5];

   always @ (*) begin
      //set all 6 digits to 0
      if (!reset)begin
         for (i=5; i>=0; i=i-1) begin
            bcd_digit[i]=4'b0;
         end
      end

      //shift 20 times
      for (i=19; i>=0; i=i-1) begin
         //check all 6 BCD tetrads, if >=5 then add 3
         for (k=5; k>=0; k=k-1) begin
            if (bcd_digit[k] >= 5) begin
               bcd_digit[k] = bcd_digit[k] + 4'd3;
            end
         end
         //shift one bit of BIN/HEX left for the first 5 tetrads
         for (k=5; k>=1; k=k-1) begin
            bcd_digit[k]=bcd_digit[k] << 1;
            bcd_digit[k][0]=bcd_digit[k-1][3];
         end
         //shift one bit of BIN/HEX left, for the last tetrad
         bcd_digit[0] = bcd_digit[0] << 1;
         bcd_digit[0][0]=hex_number1[i];
      end //end for loop
   end //end of always.
endmodule
```

5. lab2



```verilog
`default_nettype none
module lab2(input CLOCK_50,  input [3:0] KEY,  output [6:0] HEX0,HEX1,HEX2,HEX3,HEX4,HEX5, output [9:0] LEDR);


   parameter [2:0] RESET=3'b000, RESUME=3'b001, BLINKING=3'b010, OFF=3'b011, TIMER_DISPLAY=3'b100, WINNER_TIME_DISPLAY=3'b101,
                   WIN1=3'b110, WIN2=3'b111;

   reg [2:0] state=RESET, next_state=RESET;


   wire clk_ms;
   wire [19:0] ms, random_ms,display_ms;
   wire [3:0] w_ms0,w_ms1,w_ms2,w_ms3, w_ms4,w_ms5; //wires after hex_to_bcd_converter.v for displayed time
   wire [3:0] w_blink0, w_blink1, w_blink2, w_blink3, w_blink4, w_blink5;  //wires afer  blinking
   wire [3:0] winner_ms0,winner_ms1,winner_ms2,winner_ms3, winner_ms4,winner_ms5; //wires afer hex_to_bcd_converter.v for winner time


   wire [3:0] digit0, digit1, digit2, digit3, digit4, digit5;  //wires afer mux.v

   wire [13:0] random_wait_time;
   wire random_ready;


   reg [1:0]  hex_sel=2'b00;  //whether blinking or not
   wire [1:0] w_hex_sel;


   reg display_counter_start;
   wire w_display_counter_start;


   reg player1_win, player2_win;   // if is 0, not win, if 1 win,

   reg[4:0] win1=5'b00000, win2=5'b00000;   // score for player 1 and 2.

   reg [19:0]temp;

   reg [19:0] winner_time=8888;
   wire [19:0] w_winner_time;

   wire conditioned_key0, conditioned_key3;

   assign w_winner_time=winner_time;


   assign w_hex_sel=hex_sel;

   assign w_display_counter_start=display_counter_start;
```



```verilog
   assign LEDR[4:0]=win1;
   assign LEDR[9:5]={win2[0],win2[1],win2[2],win2[3],win2[4]} ;


   reg player1_cheating, player2_cheating;


   clock_divider #(.factor(50000)) (.Clock(CLOCK_50), .Reset_n(KEY[1]), .Pulse_ms(clk_ms));

   counter (.clk(clk_ms), .reset_n(KEY[1]), .resume_n(KEY[2]), .enable(1), .ms_count(ms));

   counter (.clk(clk_ms), .reset_n(KEY[1]), .resume_n(KEY[2]), .enable(w_display_counter_start), .ms_count(display_ms));


   blinkHEX #(.factor(200) ) (.ms_clk(clk_ms), .Reset_n(KEY[1]), .d0(w_blink0), .d1(w_blink1), .d2(w_blink2), .d3(w_blink3), .d4(w_blink4),.d5(w_blink5));

   hex_to_bcd_converter winnerConvert( .clk(CLOCK_50), .reset(KEY[1]), .hex_number(w_winner_time),
   .bcd_digit_0(winner_ms0),.bcd_digit_1(winner_ms1),.bcd_digit_2(winner_ms2),.bcd_digit_3(winner_ms3), .bcd_digit_4(winner_ms4),
   .bcd_digit_5(winner_ms5));

   hex_to_bcd_converter clockConvert( .clk(CLOCK_50), .reset(KEY[1]), .hex_number(display_ms),
   .bcd_digit_0(w_ms0),.bcd_digit_1(w_ms1),.bcd_digit_2(w_ms2),.bcd_digit_3(w_ms3), .bcd_digit_4(w_ms4),
   .bcd_digit_5(w_ms5));

   random (.clk(clk_ms), .reset_n(KEY[1]), .resume_n(KEY[2]),  .random(random_wait_time),.rnd_ready(random_ready));


reg [3:0]all_hex;
   //mux
   multi4_1 (.a(w_blink0), .b(all_hex), .c(w_ms0), .d(winner_ms0), .s(w_hex_sel), .out(digit0));
   multi4_1 (.a(w_blink1), .b(all_hex), .c(w_ms1), .d(winner_ms1), .s(w_hex_sel), .out(digit1));
   multi4_1 (.a(w_blink2), .b(all_hex), .c(w_ms2), .d(winner_ms2), .s(w_hex_sel), .out(digit2));
   multi4_1 (.a(w_blink3), .b(all_hex), .c(w_ms3), .d(winner_ms3), .s(w_hex_sel), .out(digit3));
   multi4_1 (.a(w_blink4), .b(all_hex), .c(w_ms4), .d(winner_ms4), .s(w_hex_sel), .out(digit4));
   multi4_1 (.a(w_blink5), .b(all_hex), .c(w_ms5), .d(winner_ms5), .s(w_hex_sel), .out(digit5));



   seven_seg_decoder  decoder0(digit0, HEX0);
   seven_seg_decoder  decoder1(digit1, HEX1);
   seven_seg_decoder  decoder2(digit2, HEX2);
   seven_seg_decoder  decoder3(digit3, HEX3);
   seven_seg_decoder  decoder4(digit4, HEX4);
   seven_seg_decoder  decoder5(digit5, HEX5);


   always @ (posedge CLOCK_50, negedge KEY[1], negedge KEY[2])
   begin
      if (!KEY[1])    // reset
         begin
         state<=RESET;
      end
   else if (!KEY[2])   //start/resume
      begin
         state<=RESUME;
      end

   else
      begin
         state<=next_state;
      end
end


always @(posedge CLOCK_50, negedge KEY[1] )    //for solving the inferred latch problem caused by win1 and win2.
begin
   if (!KEY[1]) begin
      win1<=5'b00000;
      win2<=5'b00000;
   end
   else if (player1_win==1)
         win1<=(win1<<1) | 5'b00001;
   else if (player2_win==1)
         win2<=(win2<<1) | 5'b00001;


end


always @ (*)
begin
   next_state=state;   //default
   player1_win=0;
   player2_win=0;


   case (state)
      RESET:
         begin
            display_counter_start=0;
            winner_time=0;

            hex_sel=2'b00;
            next_state=BLINKING;
         end

      RESUME:
         begin
            //hex_sel=2'b00;
            display_counter_start=0;

            winner_time=0;

            hex_sel=2'b00;
            next_state=BLINKING;
         end

      BLINKING:
         begin
            hex_sel=2'b00;


            if (ms>=5000)        //blink for  about 5 second
               begin
                  hex_sel=2'b01;
                  next_state=OFF;
               end
            else
               next_state=BLINKING;

         end

      OFF:
         begin
            all_hex = 4'b1111;
            hex_sel=2'b01;

            if (ms>(7000+random_wait_time)) begin     //(7-5) seconds + random seconds)
               display_counter_start=0;
               next_state=TIMER_DISPLAY;

            end

            //if press key0 and key3 at the same time, both cheat,display 888888
            if (~KEY[0] && ~KEY[3])
            begin
               player1_cheating = 1'b1;
               player2_cheating = 1'b1;
               all_hex = 4'b1000;
               winner_time=888888;
               next_state=WINNER_TIME_DISPLAY;
               end else begin
               player1_cheating = 1'b0;
               player2_cheating = 1'b0;
               end

            //if key0 pressed, player1 cheat, display 111111
            if(~KEY[0])
               begin
                  player1_cheating = 1'b1;
                  all_hex = 4'b0001;
                  winner_time=111111;
```

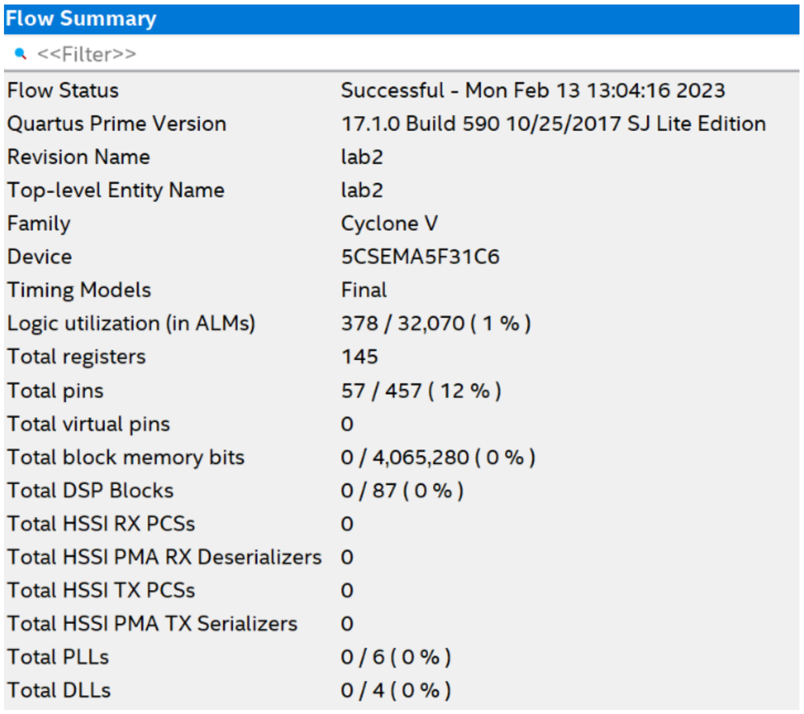
```
                next_state=WINNER_TIME_DISPLAY;
             end
             else begin
             player1_cheating = 1'b0;
             end

          //if key3 pressed, player2 cheat, display 222222
          if(~KEY[3])
             begin
                player1_cheating = 1'b1;
                all_hex = 4'b0010;
                winner_time=222222;
                next_state=WINNER_TIME_DISPLAY;
             end
             else begin
             player1_cheating = 1'b0;
             end

       end

    TIMER_DISPLAY:
       begin
          all_hex = 4'b0000;
          display_counter_start=1;
          hex_sel=2'b10;

          //if press key0 and key3 at the same time, no one wins
          if (~KEY[0] && ~KEY[3])
          begin
             player1_win=0;
             player2_win=0;
             winner_time =1'b0;
             next_state=RESUME;
          end

          //if key0 pressed, player1 win
          else if (~KEY[0])
          begin
             display_counter_start=0;
             player1_win=1;
             winner_time = display_ms;
             next_state=WINNER_TIME_DISPLAY;
          end
          //if key3 pressed, player2 win
          else if(~KEY[3])
          begin
             display_counter_start=0;
             player2_win=1;
             winner_time = display_ms;
             next_state=WINNER_TIME_DISPLAY;
          end


       end

    WINNER_TIME_DISPLAY:
       begin
          hex_sel=2'b11;
          winner_time=winner_time;
       end


    default:
       begin
          next_state=RESET;
       end
  endcase


 end

endmodule
```

Part2:

1. Type of LFSR used in this project is Galois LFSR, which is 14 bits long and taps on bits 14, 5, 3 and 1. We choose Galois LFSR because our project only needs simple hardware complexity and not long sequence length, while Fibonacci might be too much for our project as Fibonacci more suitable for complex hardware needs and long sequence length. The specific taps(14, 5, 3 ,1) is picked since it is the max length sequence for a 16-bit LFSR and it has low complexity with good performance.
2. LFSR does not generate random numbers directly, instead, it gives pseudo random sequences of 0s and 1s. LFSR is a shift register whose input bit is the feedback of some subset of its contents by using XOR or NXOR. With choosing appropriate subsets and initial seeds, numbers are generated and would eventually repeat.
   If the LFSR is too short,
   - Advantage: less hardware needed due to low complexity and could generate output faster as short LFSR would have high clock speed
   - Disadvantages: Due to low complexity, the results could be seen as looping through limited numbers which could be predicted.

   If the LFSR is long,

   - Advantage: with long repeating cycle, more random numbers could be generated.
   - Disadvantage: more complexity.
3. Compilation report
   - Total number of logic elements used: 378
   - Total number of registers: 145
   - Total number of pins: 57
   - Max number of logic elements that can fit on the FPGA that we used: 32070



Flow Summary

<<Filter>>

Flow Status	Successful - Mon Feb 13 13:04:16 2023
Quartus Prime Version	17.1.0 Build 590 10/25/2017 SJ Lite Edition
Revision Name	lab2
Top-level Entity Name	lab2
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	378 / 32,070 (1 %)
Total registers	145
Total pins	57 / 457 (12 %)
Total virtual pins	0
Total block memory bits	0 / 4,065,280 (0 %)
Total DSP Blocks	0 / 87 (0 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0 / 6 (0 %)
Total DLLs	0 / 4 (0 %)