Code

Project2_top.v

```
//=======================================================
//  This code is generated by Terasic System Builder
//=======================================================

module Project2_top(

	///////////// CLOCK //////////
	input 		          		ADC_CLK_10,
	input 		          		MAX10_CLK1_50,
	input 		          		MAX10_CLK2_50,

	///////////// SEG7 //////////
	output		     [7:0]		HEX0,
	output		     [7:0]		HEX1,
	output		     [7:0]		HEX2,
	output		     [7:0]		HEX3,
	output		     [7:0]		HEX4,
	output		     [7:0]		HEX5,

	///////////// KEY //////////
	input 		     [1:0]		KEY,

	///////////// LED //////////
	output		     [9:0]		LEDR,

	///////////// SW //////////
	input 		     [9:0]		SW
);

//defining states
parameter HI_SCORE = 3'b000;
parameter DELAYING = 3'b001;
parameter TIMING = 3'b010;
parameter DISPLAYING = 3'b011;
parameter GO_BUFFS = 3'b100; //last two digits are don't cares.
```

```
//=======================================================
//  REG/WIRE declarations
//=======================================================

	//state vars
	reg [2:0] curState;
	assign LEDR[9:7] = curState; //display state on LEDs
	reg [1:0] sub_state;

	//clocks
	wire clk_1kHz, clk;
	clock_div clk_ms(MAX10_CLK1_50, clk_1kHz);
	assign clk = MAX10_CLK1_50;

	//scores
	reg [3:0] hi_score_cs, hi_score_ds, hi_score_s;
	reg [3:0] score_cs, score_ds, score_s;

	reg point; //decimal point for timing

	//BCD times
	reg [3:0] time_hex0,time_hex1,time_hex2;


	////////////////BCD_counter//////////////////////
	wire RCO_ms,RCO_cs,RCO_ds, RCO_s; // carry out
	wire [3:0] count_ms;
	wire [3:0] count_cs;
	wire [3:0] count_ds;
	wire [3:0] count_s;
	wire timing;
	reg bcd_clear_;
	reg sevseg_clr;
	reg bcd_enable;

	BCD_counter time_ms(clk_1kHz, bcd_enable, bcd_clear_, 0, count_ms, RCO_ms);
	BCD_counter time_cs(RCO_ms, bcd_enable, bcd_clear_, 0, count_cs, RCO_cs);
	BCD_counter time_ds(RCO_cs, bcd_enable, bcd_clear_, 0, count_ds, RCO_ds);
	BCD_counter time_s(RCO_ds, bcd_enable, bcd_clear_, 0, count_s, RCO_s);

	//////////////////end BCD////////////////////
```

```
	//////////////////GO BUFFS//////////////////
	wire buff_clk;
	wire [3:0] a,b,c,d,e,f;
	wire [6:0] buff_hex0,buff_hex1,buff_hex2,buff_hex3,buff_hex4,buff_hex5;
	clock_div scroll_clk(MAX10_CLK1_50, buff_clk);
	defparam scroll_clk.n = 10000000;
	GOBUFFS scroll(buff_clk,a,b,c,d,e,f);
	//////////////////////end GO BUFFS/////////////////


	//////////////////////LFSR///////////////
	wire LFSR_clk;
	clock_div LFSR_shift(clk, LFSR_clk); //LFSR clk shifts the LFSR every cycle. FOr
randomness.
	defparam LFSR_shift.n = 20000;

	wire[11:0] LFSR_delay;
	LFSR produce_delay(LFSR_clk, LFSR_delay);//outputs random delays

	reg [11:0] LFSR_count;
	wire LFSR_ready;
	reg LFSR_en;
	//counts random delay out. en and count set in state-controlled alwyas block below
	counter LFSR_counter(clk_1kHz, LFSR_en, LFSR_count, LFSR_ready);
	//////////////////////end LFSR/////////////

	////////////////////////FINAL OUTS//////////////////
	reg [6:0] final_hex0,final_hex1,final_hex2,final_hex3,final_hex4,final_hex5;
	wire [6:0]
time_hex_out0,time_hex_out1,time_hex_out2,time_hex_out3,time_hex_out4,time_hex_out5;
	////////////////////////end finals//////////////////////

	//state transitions
	// button state control
  always @(negedge KEY[1],posedge KEY[0], posedge LFSR_ready)
  begin
		if(LFSR_ready)
				sub_state = 2;
		else
				begin
					if(!KEY[1])
						begin
```

```
                                        if(curState == TIMING)
                                                sub_state <= 3;
                                        else if ( ( curState == DISPLAYING ) && SW[5] )
                                                sub_state <= 0;
                                        else if (curState == DELAYING)
                                                sub_state <= 0;
                                        else if (curState == HI_SCORE)
                                                sub_state <= 0;
                                 end
                        else
                                begin
                                if (curState == DISPLAYING)
                                        sub_state <= 1;
                                else if(curState == HI_SCORE)
                                        sub_state <= 1;
                                end
                end
        end

 //State transitions (in concert with button state control)
always @(posedge clk)
begin
                if(SW[9])
                        begin
                                curState <= GO_BUFFS;
                        end
                else begin
                        if(LFSR_ready) begin
                                if(curState == DELAYING)
                                        curState <= TIMING;
                                end
                        else if(sub_state == 0)
                                curState <= HI_SCORE;
                        else if(sub_state == 1)
                                curState <= DELAYING;
                        else if(sub_state == 2)
                                curState <= TIMING;
                        else if(sub_state == 3)
                                curState <= DISPLAYING;
                end
end
```

```
        //state outputs/control signals
        always @(posedge clk)
         begin
                case(curState)
                 GO_BUFFS: begin
                                                 LFSR_en <= 0;
                                                 point <= 1; //off
                                                 sevseg_clr <= 0;
                                                end
                 HI_SCORE: begin
                                                 LFSR_en <= 0;
                                                 if (hi_score_s == 0 && hi_score_ds == 0 &&
hi_score_cs == 0) begin
                                                        hi_score_s <= 9;
                                                        hi_score_ds <= 9;
                                                        hi_score_cs <= 9;
                                                 end
                                                 time_hex0 <= hi_score_cs;
                                                 time_hex1 <= hi_score_ds;
                                                 time_hex2 <= hi_score_s;
                                                 point <= 0;
                                                 sevseg_clr <= 1;
                                                end
                 TIMING: begin
                                                bcd_clear_ <= 1;
                                                bcd_enable <= 1;
                                                LFSR_en <= 0;
                                                time_hex0 <= count_cs;
                                                time_hex1 <= count_ds;
                                                time_hex2 <= count_s;
                                                point <= 0;
                                                sevseg_clr <= 1;
                                        end
                 DISPLAYING: begin
                                                bcd_enable <= 0;
                                           LFSR_en <= 0;
                                                score_s <= count_s;
                                                score_ds <= count_ds;
                                                score_cs <= count_cs;
                                                time_hex0 <= score_cs;
                                                time_hex1 <= score_ds;
                                                time_hex2 <= score_s;
                                                point <= 0;
```

```
                                                    sevseg_clr <= 1;
                                                    if ({count_s,count_ds,count_cs} <
{hi_score_s, hi_score_ds,hi_score_cs}) begin
                                                            hi_score_s <= score_s;
                                                            hi_score_ds <= score_ds;
                                                            hi_score_cs <= score_cs;
                                                    end
//                                                  bcd_clear_ <= 0;//reset counter
                                                end
                    DELAYING: begin
                                                bcd_clear_ <= 0;//reset counter
                                                LFSR_count <= LFSR_delay;
                                                LFSR_en <= 1;
                                                time_hex0 <= 0;
                                                time_hex1 <= 0;
                                                time_hex2 <= 0;
                                                point <= 0;
                                                sevseg_clr <= 1;
                                               end
                endcase
            end


  //Timer SevSeg
  SevenSeg mS(time_hex0, time_hex_out0,0);
  SevenSeg cS(time_hex1, time_hex_out1,0);
  SevenSeg dS(time_hex2, time_hex_out2,0);
        SevenSeg off3(0, time_hex_out3,sevseg_clr);
        SevenSeg off4(0, time_hex_out4,sevseg_clr);
        SevenSeg off5(0, time_hex_out5,sevseg_clr);

        //Hi

        //GOBUFFS SevSeg
        Sko_SevSeg out0(a, buff_hex0);
        Sko_SevSeg out1(b, buff_hex1);
        Sko_SevSeg out2(c, buff_hex2);
        Sko_SevSeg out3(d, buff_hex3);
        Sko_SevSeg out4(e, buff_hex4);
        Sko_SevSeg out5(f, buff_hex5);

  assign HEX0[7] = 1;
  assign HEX1[7] = 1;
```

```
  assign HEX2[7] = point;
  assign HEX3[7] = 1;
        assign HEX4[7] = 1;
        assign HEX5[7] = 1;

        //finalize HEX outs
        always @(posedge clk)
        begin
                if (curState == GO_BUFFS) begin
                        final_hex0 <= buff_hex0;
                        final_hex1 <= buff_hex1;
                        final_hex2 <= buff_hex2;
                        final_hex3 <= buff_hex3;
                        final_hex4 <= buff_hex4;
                        final_hex5 <= buff_hex5;
                        test_led <= 0;
                end
                else begin
                        test_led <= 1;
                        final_hex0 <= time_hex_out0;
                        final_hex1 <= time_hex_out1;
                        final_hex2 <= time_hex_out2;
                        final_hex3 <= time_hex_out3;
                        final_hex4 <= time_hex_out4;
                        final_hex5 <= time_hex_out5;
                end
        end
        assign HEX0[6:0] = final_hex0;
        assign HEX1[6:0] = final_hex1;
        assign HEX2[6:0] = final_hex2;
        assign HEX3[6:0] = final_hex3;
        assign HEX4[6:0] = final_hex4;
        assign HEX5[6:0] = final_hex5;

        reg test_led;
        assign LEDR[0] = test_led;

endmodule // Project2_top
```

BCD_counter.v

```
//after class on Monday
```

```
module BCD_counter(clk,enable,clear_,data, count,RCO);
	input clk, enable, clear_;
	input [3:0] data;
	output reg RCO;
	output reg [3:0] count;

	wire load_;
	assign load_ = ~(count[3] & count[0]);

	always @(posedge clk, negedge clear_)
	begin
		if (!clear_)
			count <= 0;
		else if(enable)
		begin
			if(!load_)
			begin
				count <= data; //data will be passed as 0 for this module
				RCO <= 1;
			end
			else
			begin
				count <= count + 1;
				RCO <= 0;
			end
		end
	end

endmodule


	//TODO: produce reliable reset signal
module counter(clk, enable, count_to, done);
  parameter k = 12;

  input clk, enable;
  input [k-1:0] count_to;
  output reg done;

  reg [k-1:0]  count = 0;


  always @(posedge clk)
```

```
    begin
        if(enable) begin
            count <= count + 1;
            if(count == count_to)
                done <= 1;
        end
        else begin
                done <= 0;
                count <= 0;
                end
  end
endmodule // counter
```

Clock_divider.v

```
module clock_div(clk, new_clk);
  parameter n = 25000; //for 1kHz clock_out 50M/25k/2 = 1k
  input clk;
  output reg new_clk;

  reg [25:0] count;// Gives a lowest clock freq of ~1/2 Hz


  always @(posedge clk)
  begin
    if (count == n)
        begin
            count = 0;
            new_clk <= ~new_clk;
        end
        else
            count <= count + 1;
  end

      endmodule
```

LFSR.v

```
//check this out: https://en.wikipedia.org/wiki/Linear-feedback_shift_register
// Try to find a 12-bit Fib LFSR...
// would be good.
//The 16-bit Fibonnaci LFSR can go through all 65535 states. Good one.
// BUT, 65 seconds is probably toooo long.

module LFSR(enable, LF_shift_reg);
  input enable;
  output reg [11:0] LF_shift_reg;
  reg               in_shift;

  always @ (posedge enable)
    begin
                  if(!LF_shift_reg)
                          LF_shift_reg <= 12'b101101110110;
                          else begin
            in_shift <= LF_shift_reg[11] ^ LF_shift_reg[10];
            LF_shift_reg <= {LF_shift_reg[10:0],in_shift};
                    end
    end
endmodule // LFSR
```

GO_BUFFS.v

```
module GOBUFFS(clk, a,b,c,d,e,f);
  input clk;
  output reg [3:0] a; //[GO BUFFS  ]
          output reg [3:0] b;
          output reg [3:0] c;
          output reg [3:0] d;
          output reg [3:0] e;
          output reg [3:0] f;
   initial
           begin
                   a = 5;
                   b = 4;
                   c = 3;
                   d = 2;
```

```
            e = 1;
            f = 0;
        end

  always @(posedge clk)
    begin
            a <= a+1;
            b <= b+1;
            c <= c+1;
            d <= d+1;
            e <= e+1;
            f <= f+1;
            end

endmodule // GOBUFFS

module Sko_SevSeg(input [3:0] char, output reg[6:0] HEX_out);

  parameter G = 0, O = 1, SPACE = 2, B = 3, U = 4, F1 = 5, F2 = 6, S = 7, END1 = 8, END2 =
9;

        always @(char)
        begin
            case(char)
                G:          HEX_out <= 7'b1000010;
                O:      HEX_out <= 7'b1000000;
                SPACE: HEX_out <= 7'b1111111;
                B:      HEX_out <= 7'b0000011;
                U:      HEX_out <= 7'b1000001;
                F1:     HEX_out <= 7'b0001110;
                F2:     HEX_out <= 7'b0001110;
                S:      HEX_out <= 7'b0010010;
                END1:  HEX_out <= 7'b1111111;
                END2:  HEX_out <= 7'b1111111;
                default: HEX_out <= 7'b1111111;
            endcase

        end

        endmodule
```

SevenSegment.v

```
module SevenSeg (x, leds,off);
	input [3:0] x;
	input off;
	output reg [6:0] leds;

	always @(x,off)
	begin
		if(off)
			leds = 7'b1111111;
		else
		begin
			case(x)
				0: leds <= 7'b1000000;
				1: leds <= 7'b1111001;
				2: leds <= 7'b0100100;
				3: leds <= 7'b0110000;
				4: leds <= 7'b0011001;
				5: leds <= 7'b0010010;
				6: leds <= 7'b0000010;
				7: leds <= 7'b1111000;
				8: leds <= 7'b0000000;
				9: leds <= 7'b0010000;
//				default: leds<= 7'b1111111;

				///////HEX VALUES /////
			  10: leds <= 7'b0001000; //A
			  11: leds <= 7'b0000011; //B
			  12: leds <= 7'b1000110; //C
			  13: leds <= 7'b0100001; //D
			  14: leds <= 7'b0000110; //E
			  15: leds <= 7'b0001110; //F
			endcase
		end
	end

endmodule
```