3TB4 Prelab 2 Report

Lin Fu	ful10	400234794
Keyin Liang	liangk10	400236736

1. Generate random number between 1000 to 5000 in Verilog.

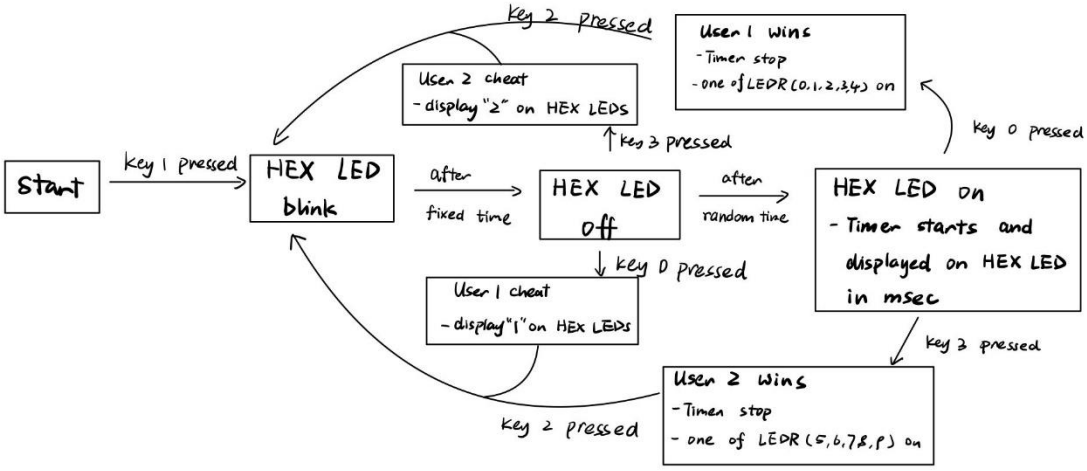
```
module random (input clk, reset_n, resume_n, output reg [13:0] random,
        output reg rnd_ready);
        //for 14 bits Liner Feedback Shift Register,
        //the Taps that need to be XNORed are: 14, 5, 3, 1
        wire xnor_taps, and_allbits, feedback;
        reg [13:0] reg_values;
    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

                                /* number is between 1000 and 5000 */

                                if (reg_values >= 1000 && reg_values <= 5000) begin
                                        random <= reg_values;
                                        rnd_ready <= 1'b1;
                                        enable <= 1'b0;
                                end

                        end //end of ENABLE.
                end
        end
endmodule
```

2. FSM diagram



3. Verilog code for FSM

```
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
                                hex_sel=2'b00;
                                display_counter_start=0;
                            winner_time=0;

                            next_state=BLINKING;
                    end
           RESUME:
                        begin
                                hex_sel=2'b00;
                                display_counter_start=0;
                                winner_time=0;

                                next_state=BLINKING;
                        end
           BLINKING:
                        begin
                                hex_sel=2'b00;

                                if (ms>=2000)        //blink for about 2 second
                                        begin
                                                hex_sel=2'b01;
                                                next_state=OFF;
                                        end
                                else
                                        next_state=BLINKING;
                        end
           OFF:
                        begin
                                hex_sel=2'b01;

                                if (ms>(5000+random_wait_time)) begin     //(7-5) seconds + random seconds)
                                        next_state=TIMER_DISPLAY;
```

```
                    end

                    if (!KEY[0]) begin //P1 cheat
                            cheat1 = 1'b1;
                            winner_time = 111111;
                            next_state=WINNER_TIME_DISPLAY;
                    end
                    else begin
                            cheat1 = 1'b0;
                    end

                    if (!KEY[3]) begin //P2 cheat
                            cheat2 = 1'b1;
                            winner_time = 222222;
                            next_state=WINNER_TIME_DISPLAY;
                    end
                    else begin
                            cheat2 = 1'b0;
                    end

            end
        TIMER_DISPLAY:
            begin
                    display_counter_start=1;
                    hex_sel=2'b10;

                    if (!KEY[0]) begin //P1 win
                            player1_win = 1;
                            winner_time = display_ms;
                            next_state=WINNER_TIME_DISPLAY;
                        end
                        else if (!KEY[3]) begin //P2 win
                                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
```

4. Module for “blinking”

```
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=200;

	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;
					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
```

5. Additional modules

Clock divider

```
module clock_divider (input Clock, Reset_n, output reg Pulse_ms);

	parameter factor=10; //50000;   // 32'h000061a7;

	reg [31:0] countQ;


	always @ (posedge Clock, negedge Reset_n)
	begin
		if (!Reset_n) begin
				countQ<=0;
				Pulse_ms<=0;
			end
		else
			begin
				if (countQ<factor/2)
					begin
						countQ<=countQ+1;
						Pulse_ms<=0;
					end
				else if (countQ<factor)
					begin
						countQ<=countQ+1;
						Pulse_ms<=1;
					end
				else    //countQ==factor
					begin
						countQ<=0;
						Pulse_ms<=0;
					end

			end
```

Counter

```
module counter(input clk, input reset_n, resume_n, enable,  output reg [19:0] ms_count);
	//since DE1-SoC board has only 6 HEX, for six digits of decimal , 20 bits on binary is enough.
	//therefore declare ms_count as 20bits

	always @ (posedge clk, negedge reset_n, negedge resume_n)

	begin


			if (!reset_n ||  !resume_n) begin
				ms_count<=   20'h00000;
			end
			else begin
				if (enable)
					ms_count<=ms_count+1;

			end

  end // End of Block COUNTER



endmodule
```

Hexadecimal to binary-coded decimal translator

```
module hex_to_bcd_converter (input reset_n, 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_n)begin
                        for (i=5; i>=0; i=i-1) begin
                                bcd_digit[i]=4'b0;
                        end
                end
                for (i=5; i>=0; i=i-1) begin
                        bcd_digit[i]=4'b0;
                end
                //shift 20 times
                for (i=19; i>=0; i=i-1) begin
                        //bcd_digit[0] = bcd_digit[0] + hex_number1[i];
                        //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
                //bcd_digit[0] = bcd_digit[0] + hex_number1[0];
        end //end of always.
endmodule
```