Abraham Carranza
822338381
COMPE 470L SECTION 01

Final Project Report—Simon Says

Table of Contents

Simon Says

As noted in my project proposal I will be recreating a version of Simon Says.

From Wikipedia, one player takes the role of "Simon" and issues instructions (usually physical actions such as "jump in the air" or "stick out your tongue") to the other players, which should be followed only when prefaced with the phrase "Simon says". Players are eliminated from the game by either following instructions that are not immediately preceded by the phrase, or by failing to follow an instruction which does include the phrase "Simon says"

In my previous report, I reached many roadblocks and was unsuccessful in creating a working simulation. This is not the case with this report. My idea has been realized.

My Simon Says game is composed of several Verilog files:

1. fsm_game.v
2. LFSR.v
3. led_decoder.v
4. clk_div.v
5. sevenSegDisplay.v
6. stim.v
7. tb_top.v (for simulation purposes)

fsm_game.v

We will begin with describing the *meat* of the program. In fsm_game.v we include make a parameterized module to change the speed of each mode and a parameter for bits—we use the NUM_BITS parameter for our LSFR.

Moving along, we declare several physical inputs essential to the functionality of the program such as btnC, btnU, etc. We declare several outputs to show our current state, the value for our LEDs, our switch for our LSFR, our value to be projected on our seven segment displays, and more. We then declare and initialize several registers; their names, values and descriptions are described in the source code provided.

We created incorporate two assign statements to asynchronously output the current state the user is in and the value to be outputted onto the seven segment displays.

Then we jump into the largest always block throughout the modules. In this always block we determine the state we are in, determine our clock dividers based on our difficulty chosen, show the results of the game played, and more. A more detailed description of the always block and its entirety is included in the source code. The source code for fsm_game.v can be found in APPENDIX A.

LFSR.v

Beginning in our parameterizable LSFR module we drive inputs essential for proper functionality, described in the source code. Our outputs, just as important, describe the value from the LSFR and when its pattern completes. The more bits used, the longer it takes for the pattern to repeat. Nevertheless, we create two registers, one to be used in the LFSR result from mechanic and another obtain the value from XNOR operations.

Then we asynchronously assign the values from the LSFR to the output and when the pattern repeats.

We enter our first always block, utilized at every positive clock edge. We create a condition if the flag from our previous module has been set to high then we have functionality in our LSFR; otherwise, we do not have functionality. The details if the condition is satisfied is noted in the source code.

In another always block for any condition, we have the same condition for functionality as before. If the flag has been set by the previous module, we have functionality in the LSFR; otherwise, we will not have functionality in the LSFR. The details of the XNOR operations are detailed in the source code for cases of 3 bits to 16 bits.

The source code for LFSR.v can be found in APPENDIX B.

led_decoder.v

The module in this Verilog file, led_decoder, previously took different signals from different modules to drive an input to the LEDs to prevent multi-driven pin errors. However, in this revision of the Final Project Report, we solved the issue and now only get inputs from one module, fsm_game.v. We declare and initialize a register that will hold the value spat out by our main module to output its value on the LEDs on the board.

The source code for decoder can be found in APPENDIX C.

clk_div.v

In our parameterized module we create a clock divider to obviously slow down our clock for a reasonable experience in the game. The details of the counters and parameters are described in the source code. For simulation purposes the always blocked contained in the module must be commented out, the third to last line also commented out, and the second to last line commented in for a proper simulation.

The source code for clk_div.v can be found in APPENDIX D.

sevenSegDisplay.v

Similar to our previous Verilog file, LFSR.v, the module in this Verilog file can have its functionality essentially stripped away depending on the state of the major FSM in fsm_game.v. In short, this module converts the value of the timer in the major FSM to be properly display on the four seven segment displays.

The source code for sevenSegDisplay.v can be found in APPENDIX E.

stim.v

In stim.v, our TOP module is contained. All our inputs and outputs from our board are driven into this module include with different bit length wires for our instantiations

The source code for stim.v can be found in APPENDIX F.

tb_top.v

In this Verilog file we create our test bench with all the details needed for its functionality and my previous testing are included in the source file.

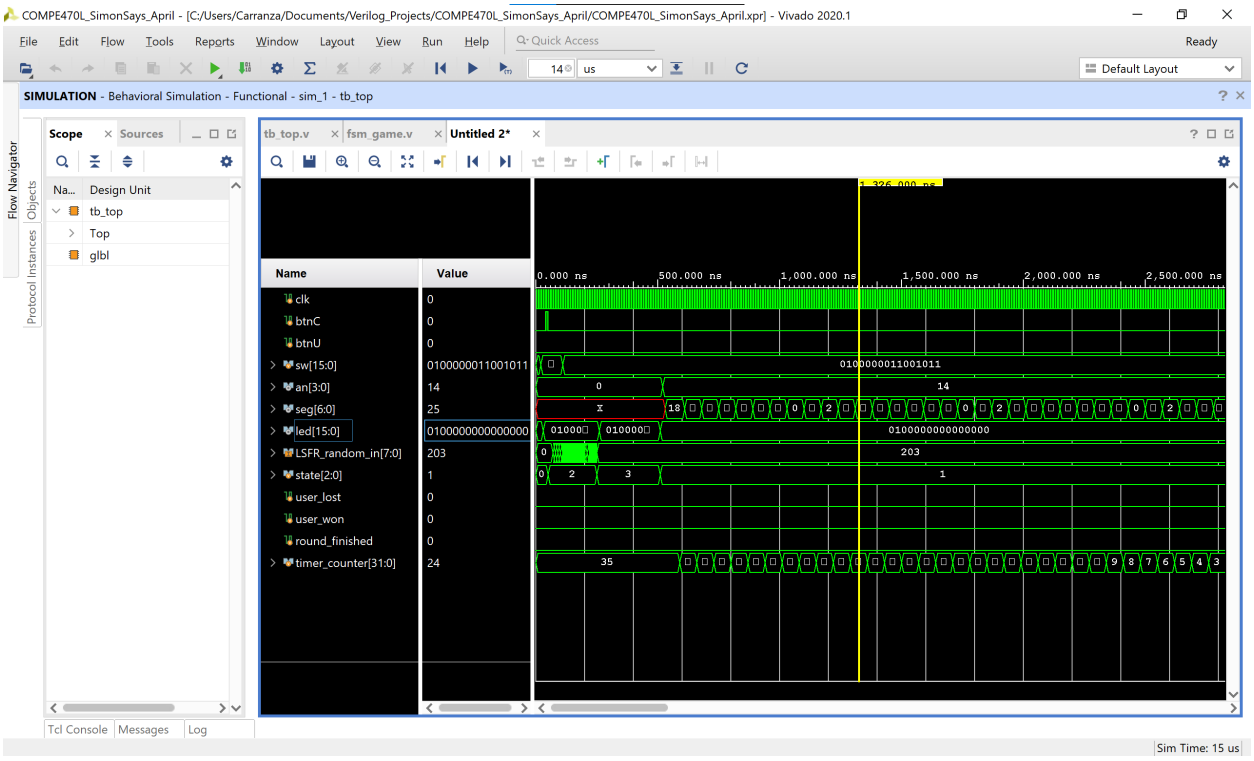
The source file can be found in APPENDIX G.

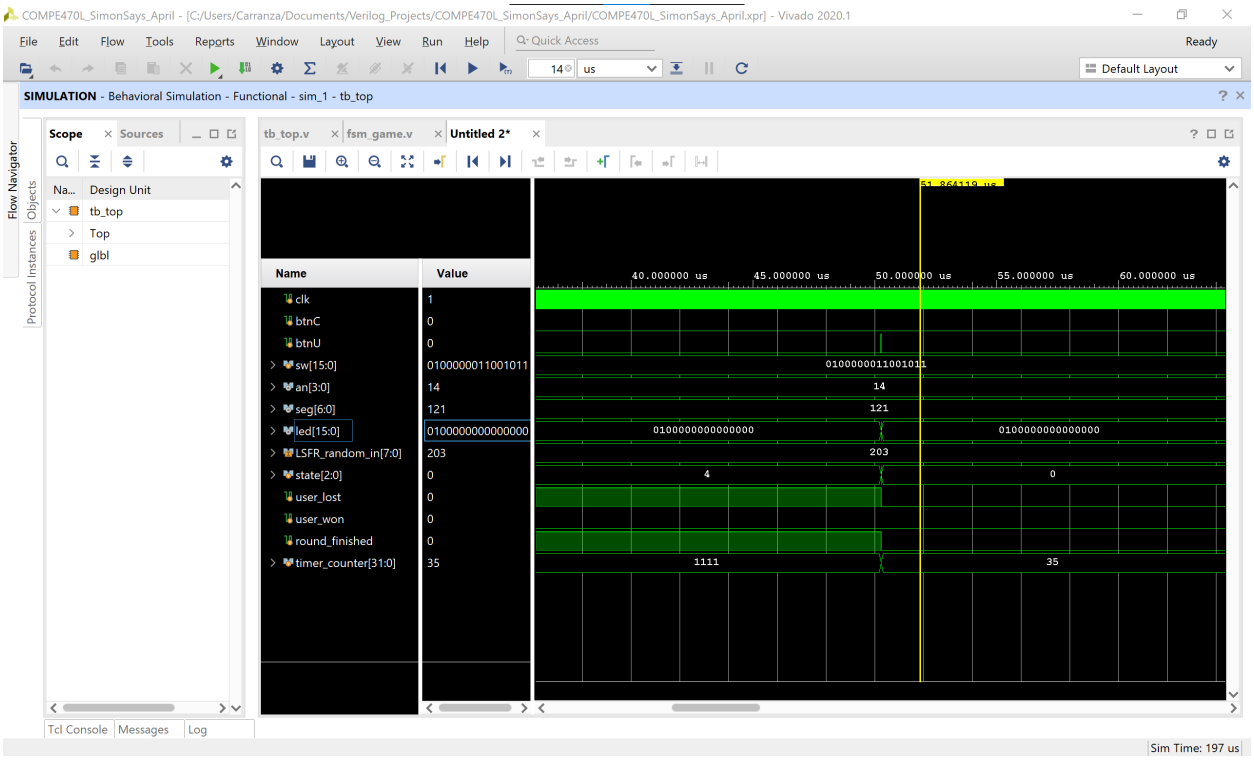
How to Play

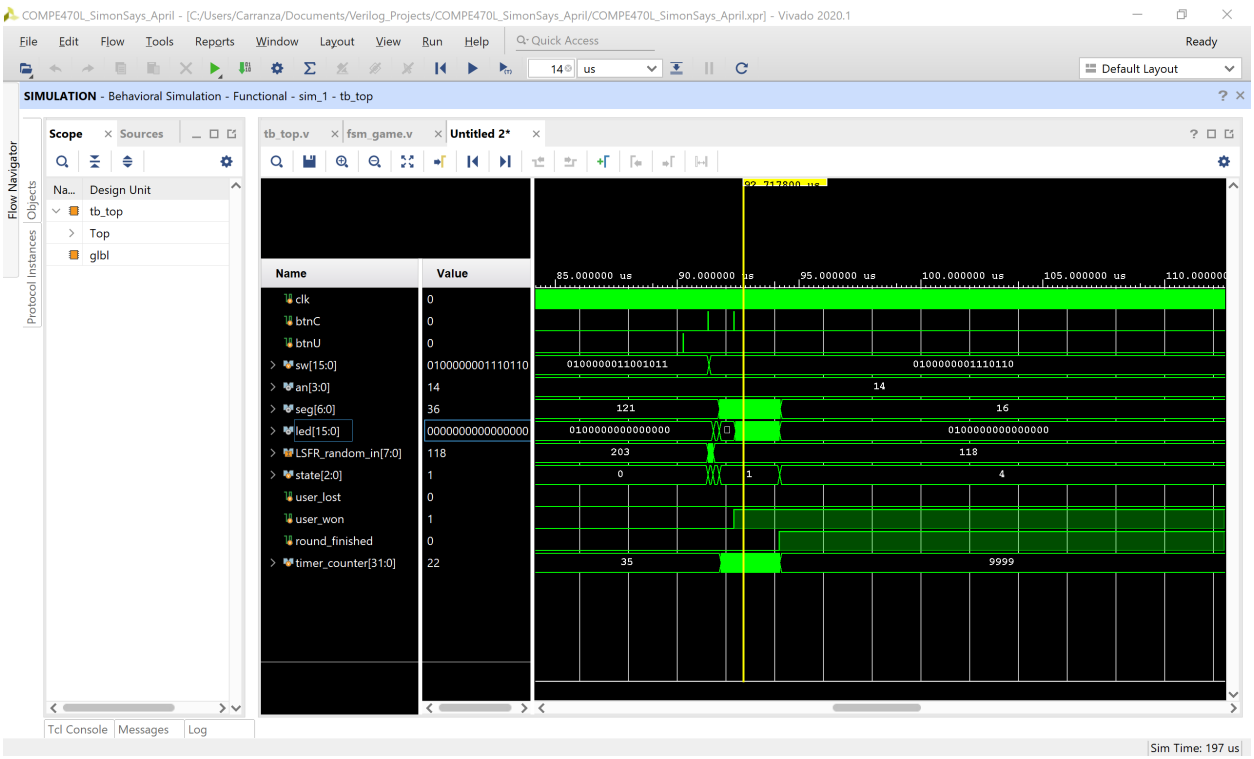
The game is basically played in a series of stages as follows:

A. At startup, the user is to choose the difficulty of their game by setting switches 15 and 14 between high and low. The modes are easy, medium, and hard or 1, 2, or 3, respectively.
    a. When a mode has been chosen it have been noted and we move onto B.
B. In the background our LSFR is at work and then we are outputted the binary value of our LSFR output using our LEDs
    a. After a value has been chosen, we move onto C.
C. The user is given a short amount of time to see said sequence. Once a certain amount of time has elapsed, we move onto D.
D. The LEDs are turned off. If the user is unable to remember the sequence of lit LEDs, they will lose the round; otherwise, they have a strong chance of winning.
    a. If the user wins or loses it is note. However, if the user wins, all the LEDs on the board flash on and off. Then we move onto E when the round is over
E. Depending on the results of the round the user will either see 9999 on the display or 1111. 9999 indicates a win and 1111 indicates a loss. When the user is ready to continue, they must press the btnU button, that will either reset and take them to change the difficulty or reset and start the round again with a new LSFR value.

Simulation Waveforms







Conclusion

Happily, everything works as intended in simulation and in hardware implementation. Everything I imagined this game could be, is. Some ideas I do have for future revisions is creating a score board that increments each time the user wins or loses a round and is displayed on the seven segment displays.

APPENDIX A:

```
`timescale 1ns / 1ps
/**********************************************************************************
    Company:
    Engineer:

    Create Date: 03/15/2021 11:17:42 AM
    Design Name:
    Module Name: fsm_game
    Project Name:
    Target Devices:
    Tool Versions:
    Description:

    Dependencies:

    Revision:
    Revision 0.01 - File Created
    Additional Comments:
**********************************************************************************/



module fsm_game#(
    /****************************************
        The following parameters are used to
        increase or decrease the speed of the
        decrementing counter for the seven
        segment display. Higher is slower and
        lower is faster
    ****************************************/
    parameter easy_speed = 5,
    parameter normal_speed = 4,//42_659_840,
    parameter hard_speed = 3,
    parameter NUM_BITS = 0
) (
    input clk, // not system clock (slowed down)
    input btnC, // select
    input btnU, // reset button
    input [15:0] sw, // switches on board
    input [NUM_BITS-1:0] LSFR_random_in, // random value from lsfr
    output reg [1:0] game_difficulty, // output game difficulty
    output reg [15:0] fsm_led, // output what leds should be to a decoder
    output reg LSFR_start, // flag to enable/disable lsfr
    output [2:0] state_out, // outputs current state
    output [31:0] sevenSegDisplay_timer // output timer value to seven segment
);

    /****************************************
        The following local parameters are made
        for readability in cases or modes
    ****************************************/
    localparam choosing_diff = 0;
    localparam playing_game = 1;
    localparam getting_rand_sequence = 2;
    localparam simon_shows = 3;
    localparam show_results = 4;
    localparam reset = 5;
    localparam easy = 1;
    localparam normal = 2;
    localparam hard = 3;

    /****************************************
        The following registers will be used
        througout the module
    ****************************************/

    reg [2:0] state; // current state in fsm
```

```
    reg difficulty_chosen; // flag to confirm a mode has been chosen

    reg user_lost; // flag
    reg user_won; // flag
    reg round_finished; // flag

    reg [31:0] timer_counter;

    reg easy_decrement; // flag to decrement
    reg [31:0] easy_counter; // clock div for easy mode
    reg normal_decrement; // flag to decrement
    reg [31:0] normal_counter; // clock div for normal mode
    reg hard_decrement; // flag to decrement
    reg [31:0] hard_counter; // clock div for hard mode


    reg [NUM_BITS-1:0] led_sequence; // register to save lsfr input value

    reg sequence_on; // flag
    reg [31:0] sequence_on_counter; // counter to loop lsfr


    reg [31:0] show_led_delay_counter; // counter used to delay leds on ON state

    reg led_alternate; // flag to flag LEDs off and on
    reg [31:0] led_alternate_counter; // counter for flashing LEDs
    reg [15:0] Sixteen_LED_ON; // register to hold max and min 16-bit value

    /*****************************************
        The following initializes our former
        registers to default values
    *****************************************/
    initial begin
        state = choosing_diff;
        difficulty_chosen = 0;
        game_difficulty = 0;
        fsm_led <= 0;
        user_lost = 0;
        user_won = 0;
        round_finished = 0;
        timer_counter = 35;
        easy_decrement = 0;
        easy_counter = 0;
        normal_decrement = 0;
        normal_counter = 0;
        hard_decrement = 0;
        hard_counter = 0;
        led_sequence = 0;
        sequence_on = 0;
        sequence_on_counter = 0;
        LSFR_start = 0;
        show_led_delay_counter = 0;
        led_alternate = 0;
        led_alternate_counter = 0;
        Sixteen_LED_ON = 16'b1111_1111_1111_1111;
    end

    /*****************************************
        The following are asynchronously assigning
        values to output our current state and for
        our seven segment displays
    *****************************************/
    assign state_out = state;
    assign sevenSegDisplay_timer = timer_counter;

    always @(posedge clk ) begin
        /*****************************************
            The following conditions for xxx_decrement
```

```
            are used to decrement at a speed with
            respect to the difficulty chosen
        ****************************************/
        if (easy_decrement) begin
            easy_counter <= easy_counter + 1;
            if (easy_counter > easy_speed) begin
                easy_counter <= 0;
                timer_counter <= timer_counter - 1;
                if(timer_counter == 1) begin
                    easy_decrement = 0;
                    timer_counter = timer_counter - 1;
                end
            end
        end
        else if (normal_decrement) begin
            normal_counter <= normal_counter + 1;
            if (normal_counter > normal_speed) begin
                normal_counter <= 0;
                timer_counter <= timer_counter - 1;
                if(timer_counter == 1) begin
                    normal_decrement = 0;
                    timer_counter = timer_counter - 1;
                end
            end
        end
        else if (hard_decrement) begin
            hard_counter <= hard_counter + 1;
            if (hard_counter > hard_speed) begin
                hard_counter <= 0;
                timer_counter <= timer_counter - 1;
                if(timer_counter == 1) begin
                    hard_decrement = 0;
                    timer_counter = timer_counter - 1;
                end
            end
        end

        /****************************************
            For the following condition, when led_alternate
            is asserted the board will flash 16 LEDs on
            and off
        ****************************************/
        if(led_alternate) begin
            led_alternate_counter <= led_alternate_counter + 1;
            if(led_alternate_counter >= 7) begin
                Sixteen_LED_ON <= ~Sixteen_LED_ON;
                fsm_led <= Sixteen_LED_ON;
                led_alternate_counter <= 0;
            end
        end

        case (state)
            choosing_diff: begin
                /****************************************
                    Upon entering the first state, essentially our
                    default state, we assign the round_finished to
                    zero on the next clock cycle. If a difficulty
                    has yet to be chosen then we enter a case
                    block for our three modes. A mode is selected
                    by a button press and regardless of mode, the
                    two far left LEDs on the board (15 and 14) are
                    turned off and on with respect to switches 15
                    and 14
                ****************************************/
                round_finished <= 0;
                if (!difficulty_chosen) begin
                    fsm_led[15:14] <= sw[15:14];
                    case (sw[15:14])
                        easy: begin
```

```
                    if (btnC) begin
                        game_difficulty <= easy;
                        state <= getting_rand_sequence;
                        difficulty_chosen <= 1; // this is a flag
                    end
                end
                normal: begin
                    if (btnC) begin
                        game_difficulty <= normal;
                        state <= getting_rand_sequence;
                        difficulty_chosen <= 1;
                    end
                end
                hard: begin
                    if (btnC) begin
                        game_difficulty <= hard;
                        state <= getting_rand_sequence;
                        difficulty_chosen <= 1;
                    end
                end
            endcase
        end
    end
    getting_rand_sequence: begin
        /*****************************************
            In this state our LSFR runs until a counter
            is no longer satisfied.
        *****************************************/
        if (sequence_on_counter < 19) begin
            sequence_on_counter <= sequence_on_counter + 1;
            LSFR_start <= 1;
            state <= getting_rand_sequence;
        end
        else begin
            state <= simon_shows;
            LSFR_start <= 0;
        end
    end
    simon_shows: begin
        /*****************************************
            In this state we create a delay to show
            the user a sequence created by our lsfr
            and where bits 1 and 0 are leds on and off
            respectively
        *****************************************/
        if (show_led_delay_counter < 25) begin
            show_led_delay_counter <= show_led_delay_counter + 1;
            led_sequence <= LSFR_random_in;
            fsm_led[7:0] <= LSFR_random_in;
        end
        else begin
            state <= playing_game;
            show_led_delay_counter = 0;
            fsm_led[7:0] <= 0;
        end
    end
    playing_game: begin
        /*****************************************
            We have a condition where if the round has not
            finished we will play; otherwise, we will move onto
            the next state and show the results
        *****************************************/
        if (!round_finished) begin
            case (game_difficulty)
                /*****************************************
                    The following modes and their conditions
                    are fairly similar. They differ in how long
                    they remain active while playing. For example
                    in mode easy, if the timer counter condition
```

```
                    is satisfied then we decrement with respect
                    to the difficulty and if the user enters the
                    correct sequence that was previously lit along
                    with a button press. The user has won and
                    the LEDs will flash off and on; otherwise,
                    the user has lost.
                ****************************************/
                easy: begin
                    if (timer_counter > 0) begin
                        easy_decrement <= 1;
                        if (sw[7:0] == led_sequence[7:0]) begin
                            if (btnC) begin
                                user_won <= 1;
                                led_alternate <= 1;
                            end
                        end
                    end
                    else begin
                        led_alternate = 0;
                        fsm_led[15:14] = game_difficulty;
                        fsm_led[13:0] = 0;
                        round_finished <= 1;
                        if(!user_won) begin
                            user_lost <= 1;
                        end
                    end
                end
                normal: begin
                    if (timer_counter > 0) begin
                        normal_decrement <= 1;
                        if (sw[7:0] == led_sequence[7:0]) begin
                            if (btnC) begin
                                user_won <= 1;
                                led_alternate <= 1;
                            end
                        end
                    end
                    else begin
                        led_alternate = 0;
                        fsm_led[15:14] = game_difficulty;
                        fsm_led[13:0] = 0;
                        round_finished <= 1;
                        if(!user_won) begin
                            user_lost <= 1;
                        end
                    end
                end
                hard: begin
                    if (timer_counter > 0) begin
                        hard_decrement <= 1;
                        if (sw[7:0] == led_sequence[7:0]) begin
                            if (btnC) begin
                                user_won <= 1;
                                led_alternate <= 1;
                            end
                        end
                    end
                    else begin
                        led_alternate = 0;
                        fsm_led[15:14] = game_difficulty;
                        fsm_led[13:0] = 0;
                        round_finished <= 1;
                        if(!user_won) begin
                            user_lost <= 1;
                        end
                    end
                end
            endcase
        end
```

```
                else if (round_finished) begin
                    state <= show_results;
                end
            end
            /*****************************************
                In the following state, we allow the user
                to appreciate their win or loss by display
                9999 or 1111 respectively. They are able
                to proceed by pressing the reset button.
                If the user previously won, they will
                keep playing.
                If the user previously lost, they will
                enter the state to choose a difficulty
                Afterwards we move into the reset state
            *****************************************/
            show_results: begin
                if (user_won) begin
                    timer_counter <= 9999;
                    if (btnU) begin
                        state <= reset;
                    end
                end
                else if (user_lost) begin
                    timer_counter <= 1111;
                    if (btnU) begin
                        state <= reset;
                        difficulty_chosen <= 0;
                        fsm_led[10:0] <= 2;
                    end
                end
            end
            /*****************************************
                The following state is used to reset the values
                used throughout the module.
            *****************************************/
            reset: begin
                state <= getting_rand_sequence;
                timer_counter <= 35;
                sequence_on_counter <= 0;
                user_won <= 0;
                user_lost <= 0;
                round_finished <= 0;
                fsm_led[11:0] <= 0;
                easy_decrement <= 0;
                easy_counter <= 0;
                normal_decrement <= 0;
                normal_counter <= 0;
                hard_decrement <= 0;
                hard_counter <= 0;
                if (user_won) begin
                    state <= getting_rand_sequence;
                end
                else if(user_lost) begin
                    state <= choosing_diff;
                end
            end
        endcase
    end
endmodule
```

APPENDIX B:

```
`timescale 1ns / 1ps
/**********************************************************************************
    Company:
    Engineer:
```

```
    Create Date: 03/15/2021 11:17:42 AM
    Design Name:
    Module Name: LSFR
    Project Name:
    Target Devices:
    Tool Versions:
    Description:

    Dependencies:

    Revision:
    Revision 0.01 - File Created
    Additional Comments:
**********************************************************************************/


module LFSR #(
    parameter NUM_BITS = 8
) (
    input clk,
    input LSFR_start, // will start and stop functionality of module
    input LSFR_enable,
    input seed,
    input [NUM_BITS-1:0] seed_data,
    output [NUM_BITS-1:0] LSFR_random_out,
    output LFSR_Done //
);

    reg [NUM_BITS:1] r_LFSR;
    reg r_XNOR;

    initial begin
        r_LFSR = 0;
    end

    assign LSFR_random_out = r_LFSR[NUM_BITS:1];
    assign LFSR_Done = (r_LFSR[NUM_BITS:1] == seed_data) ? 1'b1 : 1'b0;

    always @(posedge clk) begin
        if (LSFR_start) begin
            if (LSFR_enable == 1'b1) begin
                if (seed == 1'b1) begin
                    r_LFSR <= seed_data;
                end
                else begin
                    r_LFSR <= {r_LFSR[NUM_BITS-1:1], r_XNOR};
                end
            end
        end
    end

    always @(*) begin
        if (LSFR_start) begin
            case (NUM_BITS)
                3: begin
                    r_XNOR = r_LFSR[3] ^~ r_LFSR[2];
                end
                4: begin
                    r_XNOR = r_LFSR[4] ^~ r_LFSR[3];
                end
                5: begin
                    r_XNOR = r_LFSR[5] ^~ r_LFSR[3];
                end
                6: begin
                    r_XNOR = r_LFSR[6] ^~ r_LFSR[5];
                end
                7: begin
```

```
                    r_XNOR = r_LFSR[7] ^~ r_LFSR[6];
                end
                8: begin
                    r_XNOR = r_LFSR[8] ^~ r_LFSR[6] ^~ r_LFSR[5] ^~ r_LFSR[4];
                end
                9: begin
                    r_XNOR = r_LFSR[9] ^~ r_LFSR[5];
                end
                10: begin
                    r_XNOR = r_LFSR[10] ^~ r_LFSR[7];
                end
                11: begin
                    r_XNOR = r_LFSR[11] ^~ r_LFSR[9];
                end
                12: begin
                    r_XNOR = r_LFSR[12] ^~ r_LFSR[6] ^~ r_LFSR[4] ^~ r_LFSR[1];
                end
                13: begin
                    r_XNOR = r_LFSR[13] ^~ r_LFSR[4] ^~ r_LFSR[3] ^~ r_LFSR[1];
                end
                14: begin
                    r_XNOR = r_LFSR[14] ^~ r_LFSR[5] ^~ r_LFSR[3] ^~ r_LFSR[1];
                end
                15: begin
                    r_XNOR = r_LFSR[15] ^~ r_LFSR[14];
                end
                16: begin
                    r_XNOR = r_LFSR[16] ^~ r_LFSR[15] ^~ r_LFSR[13] ^~ r_LFSR[4];
                end
            endcase
        end
    end
endmodule
```

APPENDIX C:

```
`timescale 1ns / 1ps
/**********************************************************************************
    Company:
    Engineer:

    Create Date: 04/10/2021 01:01:41 PM
    Design Name:
    Module Name: led_decoder
    Project Name:
    Target Devices:
    Tool Versions:
    Description:

    Dependencies:

    Revision:
    Revision 0.01 - File Created
    Additional Comments:

**********************************************************************************/

/****************************************
    The following module is made to output
    on the board's LEDs with respect to the
    state of the game. For example if the user
    is choosing a difficulty, then the LEDs on
    the board will only be driven from the
    module choose_difficulty. The same occurs
    when after a difficulty has been chosen
****************************************/
```

```
module led_decoder (
    //input clk,
    input state_choose_difficulty,
    input state_post_difficulty,
    input [15:0 ] led_from_fsm,
    input [15:0] leds_from_choose,
    input [15:0] leds_from_post,
    output reg [15:0] led
);

    reg [15:0] reg_led_bruh;
    initial begin
        reg_led_bruh = 0;
    end
    always @(*) begin
        led <= led_from_fsm;
    end

endmodule
```

APPENDIX D:

```
`timescale 1ns / 1ps
/**********************************************************************************
    Company:
    Engineer:

    Create Date: 03/15/2021 11:17:42 AM
    Design Name:
    Module Name: clk_div
    Project Name:
    Target Devices:
    Tool Versions:
    Description:

    Dependencies:

    Revision:
    Revision 0.01 - File Created
    Additional Comments:
**********************************************************************************/



/*
The following module is our clock divider
we will be output "clock" signals that will
operate at variable rates
*/

module clk_div #(
    parameter speed = 42_659_840
) (
    input clk,
    output new_clk // slower clock
);

    reg new_clk_reg;
    reg [31:0] clk_counter;

    initial begin
        new_clk_reg = 0;
        clk_counter = 0;
    end

    always @(posedge clk ) begin
```

```
        clk_counter <= clk_counter + 1;
        if (clk_counter >= speed/27) begin
            clk_counter <= 0;
            new_clk_reg <= ~new_clk_reg;
        end
    end

    assign new_clk = new_clk_reg;
    //assign new_clk = clk;
endmodule
```

APPENDIX E:

```
`timescale 1ns / 1ps
/**********************************************************************************
    Company:
    Engineer:

    Create Date: 03/15/2021 11:17:42 AM
    Design Name:
    Module Name: sevenSegDisplay
    Project Name:
    Target Devices:
    Tool Versions:
    Description:

    Dependencies:

    Revision:
    Revision 0.01 - File Created
    Additional Comments:

**********************************************************************************/


module sevenSegDisplay (
    input clk,
    input [2:0] state,
    input [31:0] visual_counter,
    output [3:0] an,
    output [6:0]seg
);

    localparam choosing_diff = 0;
    localparam playing_game = 1;
    localparam getting_rand_sequence = 2;
    localparam simon_shows = 3;
    localparam show_results = 4;
    localparam reset = 5;



    reg [15:0] ssd_counter;
    reg [3:0] ssd_display;
    reg [3:0] ssd_display_value;
    reg [6:0] ssd_display_out;

    initial begin
        ssd_counter = 0;
        ssd_display = 0;
    end

    always @(posedge clk ) begin
        case (state)
            playing_game,
```

```
            show_results: begin
                ssd_counter <= ssd_counter + 1;
                if (ssd_counter < 13000*1) begin
                    ssd_display <= 4'b1110;
                    ssd_display_value <= (((visual_counter%1000)%100)%10);
                end
                else if (ssd_counter < 13000*2) begin
                    ssd_display <= 4'b1101;
                    ssd_display_value <= (((visual_counter%1000)%100)/10);
                end
                else if (ssd_counter < 13000*3) begin
                    ssd_display <= 4'b1011;
                    ssd_display_value <= (((visual_counter%1000)/100));
                end
                else if (ssd_counter < 13000*4) begin
                    ssd_display <= 4'b0111;
                    ssd_display_value <= (((visual_counter/1000)));
                end
                else begin
                    ssd_counter <= 0;
                end
            end

        endcase
    end

// gfedcba format and active low
    always @(posedge clk ) begin
        case (ssd_display_value)
            0:ssd_display_out <= 7'b1000000;////0000

            1:ssd_display_out <= 7'b1111001;////0001
            2:ssd_display_out <= 7'b0100100;////0010
            3:ssd_display_out <= 7'b0110000;////0011
            4:ssd_display_out <= 7'b0011001;////0100
            5:ssd_display_out <= 7'b0010010;////0101
            6:ssd_display_out <= 7'b0000010;////0110
            7:ssd_display_out <= 7'b1111000;////0111
            8:ssd_display_out <= 7'b0000000;////1000
            9:ssd_display_out <= 7'b0010000;////1001            default:
        endcase
    end

    assign seg = ssd_display_out;
    assign an = ssd_display;
endmodule
```

APPENDIX F:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 03/15/2021 11:08:22 AM
// Design Name:
// Module Name: stim
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
```

```
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////



/*
The following module is out top module for our
simon says application
*/

module Top #(
    parameter NUM_BITS = 8
) (
    input clk,
    input btnC, // select
    input btnU, // to be determined
    //input btnR, // will be used to reset difficulty
    input [15:0] sw, // switches input
    output [3:0] an, // anodes
    output [6:0] seg, // seven segment display
    output [15:0] led // LEDs on the board
);

    // wire in_clk;

    // clk_div inst2 (
    //     .clk(clk),
    //     .new_clk(in_clk)
    // );
    wire [1:0] diff_out_jumper;
    wire diff_chosen_jumper;
    wire pre_diff_busy_jumper;
    wire post_diff_busy_jumper;
    wire [15:0] choose_led_jumper;
    wire [15:0] post_led_jumper;

    wire [1:0] game_difficulty_jumper;
    wire [15:0] fsm_led_jumper;
    wire [NUM_BITS-1:0] LSFR_random_jumper;
    wire LSFR_start_jumper;
    wire slower_clk_jumper;
    wire [2:0] state_out_jumper;
    wire [31:0] sevenSegDisplay_timer_jumper;

/*
    choose_difficulty inst1 (
        // .clk(clk),
        // .btnC(btnC),
        // .btnR(btnR),
        // .sw(sw),
        // //.led(led),
        // .difficulty_out(diff_out_jumper),
        // .difficulty_chosen(diff_chosen_jumper),
        // .pre_diff_busy(pre_diff_busy_jumper)
        .clk(clk), // stupid fast
        .btnC(btnC), // select
        .btnR(btnR), // will be used to reset difficulty
        .sw(sw), // switches on board
        .difficulty_out(diff_out_jumper),
        .difficulty_chosen(diff_chosen_jumper), // flag
        .pre_diff_busy(pre_diff_busy_jumper), //
        .choose_led(choose_led_jumper)
    );

    post_choose inst2 (
        .clk(clk), // fast AF
        .btnC(btnC), // select
```

```
        .btnU(btnU), // to be determined
        .sw(sw),
        .difficulty_chosen_in(diff_chosen_jumper),
        .difficulty_in(diff_out_jumper),
        // output [3:0] an,
        // output [6:0] seg,
        .post_diff_busy(post_diff_busy_jumper), //
        .post_led(post_led_jumper)
    );

*/
    fsm_game #(.NUM_BITS(NUM_BITS)) inst4(
        .clk(slower_clk_jumper), // stupid fast
        .btnC(btnC), // select
        .btnU(btnU), // reset
        //.btnR(btnR), // will be used to reset difficulty
        .sw(sw), // switches on board
        .LSFR_random_in(LSFR_random_jumper),
        //input post_diff_busy_in,
        .game_difficulty(game_difficulty_jumper),
        //output reg difficulty_chosen, // flag
        //output reg pre_diff_busy, //
        .fsm_led(fsm_led_jumper),
        .LSFR_start(LSFR_start_jumper),
        .state_out(state_out_jumper),
        .sevenSegDisplay_timer(sevenSegDisplay_timer_jumper)
    );


    LFSR #(.NUM_BITS(NUM_BITS)) inst5(
        .clk(clk),
        .LSFR_start(LSFR_start_jumper),
        .LSFR_enable(1'b1),
        .seed(1'b0),
        .seed_data({NUM_BITS{1'b0}}), // Replication
        .LSFR_random_out(LSFR_random_jumper),
        .LFSR_Done(w_LFSR_Done)
    );

    led_decoder inst3 (
        //.clk(clk),
        .state_choose_difficulty(pre_diff_busy_jumper),
        .state_post_difficulty(post_diff_busy_jumper),
        .led_from_fsm(fsm_led_jumper),
        .leds_from_choose(choose_led_jumper),
        .leds_from_post(post_led_jumper),
        .led(led)
    );

    clk_div inst2 (
        .clk(clk),
        .new_clk(slower_clk_jumper)
    );

    sevenSegDisplay inst6 (
        .clk(clk),
        .state(state_out_jumper),
        .visual_counter(sevenSegDisplay_timer_jumper),
        .an(an),
        .seg(seg)
    );
endmodule
```

APPENDIX G:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
```

```
// Company:
// Engineer:
//
// Create Date: 04/09/2021 04:04:10 PM
// Design Name:
// Module Name: tb_top
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module tb_top;

    reg clk, btnC, btnU, btnR;
    reg [15:0] sw;
    wire [3:0] an;
    wire [6:0] seg;
    wire [15:0] led;

    initial begin
        clk = 1;
        btnC = 0;
        btnU = 0;
        btnR = 0;
        sw = 0;
    end

    always begin
        #5 clk = ~clk;
    end

    Top inst11 (
        .clk(clk),
        .btnC(btnC),
        .btnU(btnU),
        .sw(sw),
        .led(led),
        .seg(seg),
        .an(an)
    );

    initial begin
        #1;
        #(10*1)
        #10;
        sw = sw | (1 << 14);
        #10;
        #10;
        btnC = 1;
        #10;
        btnC = 0;
        #10;
        #10;
        #10;
        #10;
        #10;
        #10;
        sw =  sw | 8'b1100_1011;
        #10;
```

```
        #10;
        #10;
        #(1000*10);
        //btnC = 1;
        #10;
        btnC = 0;
        #10;
        #10;
        #10;
        #10;
        #10;
        #10;
        #10;
        #10;
        //btnC = 1;
        #10;
        btnC = 0;
        #(10*4000);
        btnU = 1;
        #10;
        btnU = 0;
        #(10*4000);
        btnU = 1;
        #10;
        btnU = 0;
        #(10*100);
        btnC= 1;
        #10;
        btnC = 0;
        #10;
        #10;
        #10;
        sw[7:0] = 7'b01110110;
        #10;
        #10;
        #(10*100);
        btnC= 1;
        #10;
        btnC = 0;
        #10;

    end
endmodule
```

```
`timescale 1ns / 1ps
/**********************************************************************************
    Company:
    Engineer:

    Create Date: 03/15/2021 11:17:42 AM
    Design Name:
    Module Name: fsm_game
    Project Name:
    Target Devices:
    Tool Versions:
    Description:

    Dependencies:

    Revision:
    Revision 0.01 - File Created
    Additional Comments:
**********************************************************************************/


module fsm_game#(
    /****************************************
        The following parameters are used to
        increase or decrease the speed of the
        decrementing counter for the seven
        segment display. Higher is slower and
        lower is faster
    ****************************************/
    parameter easy_speed = 5,
    parameter normal_speed = 4,//42_659_840,
    parameter hard_speed = 3,
    parameter NUM_BITS = 0
) (
    input clk, // not system clock (slowed down)
    input btnC, // select
    input btnU, // reset button
    input [15:0] sw, // switches on board
    input [NUM_BITS-1:0] LSFR_random_in, // random value from lsfr
    output reg [1:0] game_difficulty, // output game difficulty
    output reg [15:0] fsm_led, // output what leds should be to a decoder
    output reg LSFR_start, // flag to enable/disable lsfr
    output [2:0] state_out, // outputs current state
    output [31:0] sevenSegDisplay_timer // output timer value to seven segment
);

    /****************************************
        The following local parameters are made
        for readability in cases or modes
    ****************************************/
    localparam choosing_diff = 0;
    localparam playing_game = 1;
    localparam getting_rand_sequence = 2;
    localparam simon_shows = 3;
    localparam show_results = 4;
    localparam reset = 5;
    localparam easy = 1;
    localparam normal = 2;
    localparam hard = 3;

```

```
61      /*****************************************
62          The following registers will be used
63          througout the module
64      *****************************************/
65
66      reg [2:0] state; // current state in fsm
67      reg difficulty_chosen; // flag to confirm a mode has been chosen
68
69      reg user_lost; // flag
70      reg user_won; // flag
71      reg round_finished; // flag
72
73      reg [31:0] timer_counter;
74
75      reg easy_decrement; // flag to decrement
76      reg [31:0] easy_counter; // clock div for easy mode
77      reg normal_decrement; // flag to decrement
78      reg [31:0] normal_counter; // clock div for normal mode
79      reg hard_decrement; // flag to decrement
80      reg [31:0] hard_counter; // clock div for hard mode
81
82
83      reg [NUM_BITS-1:0] led_sequence; // register to save lsfr input value
84
85      reg sequence_on; // flag
86      reg [31:0] sequence_on_counter; // counter to loop lsfr
87
88
89      reg [31:0] show_led_delay_counter; // counter used to delay leds on ON state
90
91      reg led_alternate; // flag to flag LEDs off and on
92      reg [31:0] led_alternate_counter; // counter for flashing LEDs
93      reg [15:0] Sixteen_LED_ON; // register to hold max and min 16-bit value
94
95      /*****************************************
96          The following initializes our former
97          registers to default values
98      *****************************************/
99      initial begin
100         state = choosing_diff;
101         difficulty_chosen = 0;
102         game_difficulty = 0;
103         fsm_led <= 0;
104         user_lost = 0;
105         user_won = 0;
106         round_finished = 0;
107         timer_counter = 35;
108         easy_decrement = 0;
109         easy_counter = 0;
110         normal_decrement = 0;
111         normal_counter = 0;
112         hard_decrement = 0;
113         hard_counter = 0;
114         led_sequence = 0;
115         sequence_on = 0;
116         sequence_on_counter = 0;
117         LSFR_start = 0;
118         show_led_delay_counter = 0;
119         led_alternate = 0;
120         led_alternate_counter = 0;
```

```
121         Sixteen_LED_ON = 16'b1111_1111_1111_1111;
122     end
123 
124     /****************************************
125         The following are asynchronously assigning
126         values to output our current state and for
127         our seven segment displays
128     ****************************************/
129     assign state_out = state;
130     assign sevenSegDisplay_timer = timer_counter;
131 
132     always @(posedge clk ) begin
133         /****************************************
134             The following conditions for xxx_decrement
135             are used to decrement at a speed with
136             respect to the difficulty chosen
137         ****************************************/
138         if (easy_decrement) begin
139             easy_counter <= easy_counter + 1;
140             if (easy_counter > easy_speed) begin
141                 easy_counter <= 0;
142                 timer_counter <= timer_counter - 1;
143                 if(timer_counter == 1) begin
144                     easy_decrement = 0;
145                     timer_counter = timer_counter - 1;
146                 end
147             end
148         end
149         else if (normal_decrement) begin
150             normal_counter <= normal_counter + 1;
151             if (normal_counter > normal_speed) begin
152                 normal_counter <= 0;
153                 timer_counter <= timer_counter - 1;
154                 if(timer_counter == 1) begin
155                     normal_decrement = 0;
156                     timer_counter = timer_counter - 1;
157                 end
158             end
159         end
160         else if (hard_decrement) begin
161             hard_counter <= hard_counter + 1;
162             if (hard_counter > hard_speed) begin
163                 hard_counter <= 0;
164                 timer_counter <= timer_counter - 1;
165                 if(timer_counter == 1) begin
166                     hard_decrement = 0;
167                     timer_counter = timer_counter - 1;
168                 end
169             end
170         end
171 
172         /****************************************
173             For the following condition, when led_alternate
174             is asserted the board will flash 16 LEDs on
175             and off
176         ****************************************/
177         if(led_alternate) begin
178             led_alternate_counter <= led_alternate_counter + 1;
179             if(led_alternate_counter >= 7) begin
180                 Sixteen_LED_ON <= ~Sixteen_LED_ON;
```

```
181                 fsm_led <= Sixteen_LED_ON;
182                 led_alternate_counter <= 0;
183             end
184         end
185 
186         case (state)
187             choosing_diff: begin
188                 /*****************************************
189                     Upon entering the first state, essentially our
190                     default state, we assign the round_finished to
191                     zero on the next clock cycle. If a difficulty
192                     has yet to be chosen then we enter a case
193                     block for our three modes. A mode is selected
194                     by a button press and regardless of mode, the
195                     two far left LEDs on the board (15 and 14) are
196                     turned off and on with respect to switches 15
197                     and 14
198                 *****************************************/
199                 round_finished <= 0;
200                 if (!difficulty_chosen) begin
201                     fsm_led[15:14] <= sw[15:14];
202                     case (sw[15:14])
203                         easy: begin
204                             if (btnC) begin
205                                 game_difficulty <= easy;
206                                 state <= getting_rand_sequence;
207                                 difficulty_chosen <= 1; // this is a flag
208                             end
209                         end
210                         normal: begin
211                             if (btnC) begin
212                                 game_difficulty <= normal;
213                                 state <= getting_rand_sequence;
214                                 difficulty_chosen <= 1;
215                             end
216                         end
217                         hard: begin
218                             if (btnC) begin
219                                 game_difficulty <= hard;
220                                 state <= getting_rand_sequence;
221                                 difficulty_chosen <= 1;
222                             end
223                         end
224                     endcase
225                 end
226             end
227             getting_rand_sequence: begin
228                 /*****************************************
229                     In this state our LSFR runs until a counter
230                     is no longer satisfied.
231                 *****************************************/
232                 if (sequence_on_counter < 19) begin
233                     sequence_on_counter <= sequence_on_counter + 1;
234                     LSFR_start <= 1;
235                     state <= getting_rand_sequence;
236                 end
237                 else begin
238                     state <= simon_shows;
239                     LSFR_start <= 0;
240                 end
```

```
241             end
242             simon_shows: begin
243                 /*****************************************
244                     In this state we create a delay to show
245                     the user a sequence created by our lsfr
246                     and where bits 1 and 0 are leds on and off
247                     respectively
248                 *****************************************/
249                 if (show_led_delay_counter < 25) begin
250                     show_led_delay_counter <= show_led_delay_counter + 1;
251                     led_sequence <= LSFR_random_in;
252                     fsm_led[7:0] <= LSFR_random_in;
253                 end
254                 else begin
255                     state <= playing_game;
256                     show_led_delay_counter = 0;
257                     fsm_led[7:0] <= 0;
258                 end
259             end
260             playing_game: begin
261                 /*****************************************
262                     We have a condition where if the round has not
263                     finished we will play; otherwise, we will move onto
264                     the next state and show the results
265                 *****************************************/
266                 if (!round_finished) begin
267                     case (game_difficulty)
268                         /*****************************************
269                             The following modes and their conditions
270                             are fairly similar. They differ in how long
271                             they remain active while playing. For example
272                             in mode easy, if the timer counter condition
273                             is satisfied then we decrement with respect
274                             to the difficulty and if the user enters the
275                             correct sequence that was previously lit along
276                             with a button press. The user has won and
277                             the LEDs will flash off and on; otherwise,
278                             the user has lost.
279                         *****************************************/
280                         easy: begin
281                             if (timer_counter > 0) begin
282                                 easy_decrement <= 1;
283                                 if (sw[7:0] == led_sequence[7:0]) begin
284                                     if (btnC) begin
285                                         user_won <= 1;
286                                         led_alternate <= 1;
287                                     end
288                                 end
289                             end
290                             else begin
291                                 led_alternate = 0;
292                                 fsm_led[15:14] = game_difficulty;
293                                 fsm_led[13:0] = 0;
294                                 round_finished <= 1;
295                                 if(!user_won) begin
296                                     user_lost <= 1;
297                                 end
298                             end
299                         end
300                         normal: begin
```

```
                        if (timer_counter > 0) begin
                            normal_decrement <= 1;
                            if (sw[7:0] == led_sequence[7:0]) begin
                                if (btnC) begin
                                    user_won <= 1;
                                    led_alternate <= 1;
                                end
                            end
                        end
                        else begin
                            led_alternate = 0;
                            fsm_led[15:14] = game_difficulty;
                            fsm_led[13:0] = 0;
                            round_finished <= 1;
                            if(!user_won) begin
                                user_lost <= 1;
                            end
                        end
                    end
                    hard: begin
                        if (timer_counter > 0) begin
                            hard_decrement <= 1;
                            if (sw[7:0] == led_sequence[7:0]) begin
                                if (btnC) begin
                                    user_won <= 1;
                                    led_alternate <= 1;
                                end
                            end
                        end
                        else begin
                            led_alternate = 0;
                            fsm_led[15:14] = game_difficulty;
                            fsm_led[13:0] = 0;
                            round_finished <= 1;
                            if(!user_won) begin
                                user_lost <= 1;
                            end
                        end
                    end
                endcase
            end
            else if (round_finished) begin
                state <= show_results;
            end
        end
        /*****************************************
            In the following state, we allow the user
            to appreciate their win or loss by display
            9999 or 1111 respectively. They are able
            to proceed by pressing the reset button.
            If the user previously won, they will
            keep playing.
            If the user previously lost, they will
            enter the state to choose a difficulty
            Afterwards we move into the reset state
        *****************************************/
        show_results: begin
            if (user_won) begin
                timer_counter <= 9999;
                if (btnU) begin
```

```
361                         state <= reset;
362                     end
363                 end
364                 else if (user_lost) begin
365                     timer_counter <= 1111;
366                     if (btnU) begin
367                         state <= reset;
368                         difficulty_chosen <= 0;
369                         fsm_led[10:0] <= 2;
370                     end
371                 end
372             end
373             /*****************************************
374                 The following state is used to reset the values
375                 used throughout the module.
376             *****************************************/
377             reset: begin
378                 state <= getting_rand_sequence;
379                 timer_counter <= 35;
380                 sequence_on_counter <= 0;
381                 user_won <= 0;
382                 user_lost <= 0;
383                 round_finished <= 0;
384                 fsm_led[11:0] <= 0;
385                 easy_decrement <= 0;
386                 easy_counter <= 0;
387                 normal_decrement <= 0;
388                 normal_counter <= 0;
389                 hard_decrement <= 0;
390                 hard_counter <= 0;
391                 if (user_won) begin
392                     state <= getting_rand_sequence;
393                 end
394                 else if(user_lost) begin
395                     state <= choosing_diff;
396                 end
397             end
398         endcase
399     end
400 endmodule
401
```

```
`timescale 1ns / 1ps
/**********************************************************************************
    Company:
    Engineer:

    Create Date: 03/15/2021 11:17:42 AM
    Design Name:
    Module Name: LSFR
    Project Name:
    Target Devices:
    Tool Versions:
    Description:

    Dependencies:

    Revision:
    Revision 0.01 - File Created
    Additional Comments:
**********************************************************************************/


module LFSR #(
    parameter NUM_BITS = 8
) (
    input clk,
    input LSFR_start, // will start and stop functionality of module
    input LSFR_enable,
    input seed,
    input [NUM_BITS-1:0] seed_data,
    output [NUM_BITS-1:0] LSFR_random_out,
    output LFSR_Done //
);

    reg [NUM_BITS:1] r_LFSR;
    reg r_XNOR;

    initial begin
        r_LFSR = 0;
    end

    assign LSFR_random_out = r_LFSR[NUM_BITS:1];
    assign LFSR_Done = (r_LFSR[NUM_BITS:1] == seed_data) ? 1'b1 : 1'b0;

    always @(posedge clk) begin
        if (LSFR_start) begin
            if (LSFR_enable == 1'b1) begin
                if (seed == 1'b1) begin
                    r_LFSR <= seed_data;
                end
                else begin
                    r_LFSR <= {r_LFSR[NUM_BITS-1:1], r_XNOR};
                end
            end
        end
    end

    always @(*) begin
        if (LSFR_start) begin
            case (NUM_BITS)
                3: begin
```

```
                    r_XNOR = r_LFSR[3] ^~ r_LFSR[2];
                end
                4: begin
                    r_XNOR = r_LFSR[4] ^~ r_LFSR[3];
                end
                5: begin
                    r_XNOR = r_LFSR[5] ^~ r_LFSR[3];
                end
                6: begin
                    r_XNOR = r_LFSR[6] ^~ r_LFSR[5];
                end
                7: begin
                    r_XNOR = r_LFSR[7] ^~ r_LFSR[6];
                end
                8: begin
                    r_XNOR = r_LFSR[8] ^~ r_LFSR[6] ^~ r_LFSR[5] ^~ r_LFSR[4];
                end
                9: begin
                    r_XNOR = r_LFSR[9] ^~ r_LFSR[5];
                end
                10: begin
                    r_XNOR = r_LFSR[10] ^~ r_LFSR[7];
                end
                11: begin
                    r_XNOR = r_LFSR[11] ^~ r_LFSR[9];
                end
                12: begin
                    r_XNOR = r_LFSR[12] ^~ r_LFSR[6] ^~ r_LFSR[4] ^~ r_LFSR[1];
                end
                13: begin
                    r_XNOR = r_LFSR[13] ^~ r_LFSR[4] ^~ r_LFSR[3] ^~ r_LFSR[1];
                end
                14: begin
                    r_XNOR = r_LFSR[14] ^~ r_LFSR[5] ^~ r_LFSR[3] ^~ r_LFSR[1];
                end
                15: begin
                    r_XNOR = r_LFSR[15] ^~ r_LFSR[14];
                end
                16: begin
                    r_XNOR = r_LFSR[16] ^~ r_LFSR[15] ^~ r_LFSR[13] ^~ r_LFSR[4];
                end
            endcase
        end
    end
endmodule

```

```
`timescale 1ns / 1ps
/**********************************************************************************
    Company:
    Engineer:

    Create Date: 04/10/2021 01:01:41 PM
    Design Name:
    Module Name: led_decoder
    Project Name:
    Target Devices:
    Tool Versions:
    Description:

    Dependencies:

    Revision:
    Revision 0.01 - File Created
    Additional Comments:

**********************************************************************************/

/****************************************
    The following module is made to output
    on the board's LEDs with respect to the
    state of the game. For example if the user
    is choosing a difficulty, then the LEDs on
    the board will only be driven from the
    module choose_difficulty. The same occurs
    when after a difficulty has been chosen
****************************************/

module led_decoder (
    //input clk,
    input state_choose_difficulty,
    input state_post_difficulty,
    input [15:0 ] led_from_fsm,
    input [15:0] leds_from_choose,
    input [15:0] leds_from_post,
    output reg [15:0] led
);

    reg [15:0] reg_led_bruh;
    initial begin
        reg_led_bruh = 0;
    end
    always @(*) begin
        led <= led_from_fsm;
    end

endmodule
```

```
`timescale 1ns / 1ps
/**********************************************************************************
    Company:
    Engineer:

    Create Date: 03/15/2021 11:17:42 AM
    Design Name:
    Module Name: clk_div
    Project Name:
    Target Devices:
    Tool Versions:
    Description:

    Dependencies:

    Revision:
    Revision 0.01 - File Created
    Additional Comments:
**********************************************************************************/



/*
The following module is our clock divider
we will be output "clock" signals that will
operate at variable rates
*/

module clk_div #(
    parameter speed = 42_659_840
) (
    input clk,
    output new_clk // slower clock
);

    reg new_clk_reg;
    reg [31:0] clk_counter;

    initial begin
        new_clk_reg = 0;
        clk_counter = 0;
    end

    always @(posedge clk ) begin
        clk_counter <= clk_counter + 1;
        if (clk_counter >= speed/27) begin
            clk_counter <= 0;
            new_clk_reg <= ~new_clk_reg;
        end
    end

    assign new_clk = new_clk_reg;
    //assign new_clk = clk;
endmodule

```

```
`timescale 1ns / 1ps
/**********************************************************************************
    Company:
    Engineer:

    Create Date: 03/15/2021 11:17:42 AM
    Design Name:
    Module Name: sevenSegDisplay
    Project Name:
    Target Devices:
    Tool Versions:
    Description:

    Dependencies:

    Revision:
    Revision 0.01 - File Created
    Additional Comments:

**********************************************************************************/


module sevenSegDisplay (
    input clk,
    input [2:0] state,
    input [31:0] visual_counter,
    output [3:0] an,
    output [6:0]seg
);

    localparam choosing_diff = 0;
    localparam playing_game = 1;
    localparam getting_rand_sequence = 2;
    localparam simon_shows = 3;
    localparam show_results = 4;
    localparam reset = 5;



    reg [15:0] ssd_counter;
    reg [3:0] ssd_display;
    reg [3:0] ssd_display_value;
    reg [6:0] ssd_display_out;

    initial begin
        ssd_counter = 0;
        ssd_display = 0;
    end

    always @(posedge clk ) begin
        case (state)
            playing_game,
            show_results: begin
                ssd_counter <= ssd_counter + 1;
                if (ssd_counter < 13000*1) begin
                    ssd_display <= 4'b1110;
                    ssd_display_value <= (((visual_counter%1000)%100)%10);
                end
                else if (ssd_counter < 13000*2) begin
                    ssd_display <= 4'b1101;
```

```
                    ssd_display_value <= (((visual_counter%1000)%100)/10);
                end
                else if (ssd_counter < 13000*3) begin
                    ssd_display <= 4'b1011;
                    ssd_display_value <= (((visual_counter%1000)/100));
                end
                else if (ssd_counter < 13000*4) begin
                    ssd_display <= 4'b0111;
                    ssd_display_value <= (((visual_counter/1000)));
                end
                else begin
                    ssd_counter <= 0;
                end
            end

        endcase
    end

// gfedcba format and active low
    always @(posedge clk ) begin
        case (ssd_display_value)
            0:ssd_display_out <= 7'b1000000;////0000                                __

      1:ssd_display_out <= 7'b1111001;////0001                       f/    /b
      2:ssd_display_out <= 7'b0100100;////0010                         g__
      3:ssd_display_out <= 7'b0110000;////0011                      e /   /c
      4:ssd_display_out <= 7'b0011001;////0100                        __
      5:ssd_display_out <= 7'b0010010;////0101
          d
      6:ssd_display_out <= 7'b0000010;////0110
      7:ssd_display_out <= 7'b1111000;////0111
      8:ssd_display_out <= 7'b0000000;////1000
      9:ssd_display_out <= 7'b0010000;////1001           default:
        endcase
    end

    assign seg = ssd_display_out;
    assign an = ssd_display;
endmodule

```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 03/15/2021 11:08:22 AM
// Design Name:
// Module Name: stim
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////



/*
The following module is out top module for our
simon says application
*/

module Top #(
    parameter NUM_BITS = 8
) (
    input clk,
    input btnC, // select
    input btnU, // to be determined
    //input btnR, // will be used to reset difficulty
    input [15:0] sw, // switches input
    output [3:0] an, // anodes
    output [6:0] seg, // seven segment display
    output [15:0] led // LEDs on the board
);

    // wire in_clk;

    // clk_div inst2 (
    //     .clk(clk),
    //     .new_clk(in_clk)
    // );
    wire [1:0] diff_out_jumper;
    wire diff_chosen_jumper;
    wire pre_diff_busy_jumper;
    wire post_diff_busy_jumper;
    wire [15:0] choose_led_jumper;
    wire [15:0] post_led_jumper;

    wire [1:0] game_difficulty_jumper;
    wire [15:0] fsm_led_jumper;
    wire [NUM_BITS-1:0] LSFR_random_jumper;
    wire LSFR_start_jumper;
    wire slower_clk_jumper;
    wire [2:0] state_out_jumper;
```

```
61      wire [31:0] sevenSegDisplay_timer_jumper;
62
63  /*
64      choose_difficulty inst1 (
65          // .clk(clk),
66          // .btnC(btnC),
67          // .btnR(btnR),
68          // .sw(sw),
69          // //.led(led),
70          // .difficulty_out(diff_out_jumper),
71          // .difficulty_chosen(diff_chosen_jumper),
72          // .pre_diff_busy(pre_diff_busy_jumper)
73          .clk(clk), // stupid fast
74          .btnC(btnC), // select
75          .btnR(btnR), // will be used to reset difficulty
76          .sw(sw), // switches on board
77          .difficulty_out(diff_out_jumper),
78          .difficulty_chosen(diff_chosen_jumper), // flag
79          .pre_diff_busy(pre_diff_busy_jumper), //
80          .choose_led(choose_led_jumper)
81      );
82
83      post_choose inst2 (
84          .clk(clk), // fast AF
85          .btnC(btnC), // select
86          .btnU(btnU), // to be determined
87          .sw(sw),
88          .difficulty_chosen_in(diff_chosen_jumper),
89          .difficulty_in(diff_out_jumper),
90          // output [3:0] an,
91          // output [6:0] seg,
92          .post_diff_busy(post_diff_busy_jumper), //
93          .post_led(post_led_jumper)
94      );
95
96  */
97      fsm_game #(.NUM_BITS(NUM_BITS)) inst4(
98          .clk(slower_clk_jumper), // stupid fast
99          .btnC(btnC), // select
100         .btnU(btnU), // reset
101         //.btnR(btnR), // will be used to reset difficulty
102         .sw(sw), // switches on board
103         .LSFR_random_in(LSFR_random_jumper),
104         //input post_diff_busy_in,
105         .game_difficulty(game_difficulty_jumper),
106         //output reg difficulty_chosen, // flag
107         //output reg pre_diff_busy, //
108         .fsm_led(fsm_led_jumper),
109         .LSFR_start(LSFR_start_jumper),
110         .state_out(state_out_jumper),
111         .sevenSegDisplay_timer(sevenSegDisplay_timer_jumper)
112     );
113
114
115     LFSR #(.NUM_BITS(NUM_BITS)) inst5(
116         .clk(clk),
117         .LSFR_start(LSFR_start_jumper),
118         .LSFR_enable(1'b1),
119         .seed(1'b0),
120         .seed_data({NUM_BITS{1'b0}}), // Replication
```

```
121         .LSFR_random_out(LSFR_random_jumper),
122         .LFSR_Done(w_LFSR_Done)
123     );
124 
125     led_decoder inst3 (
126         //.clk(clk),
127         .state_choose_difficulty(pre_diff_busy_jumper),
128         .state_post_difficulty(post_diff_busy_jumper),
129         .led_from_fsm(fsm_led_jumper),
130         .leds_from_choose(choose_led_jumper),
131         .leds_from_post(post_led_jumper),
132         .led(led)
133     );
134 
135     clk_div inst2 (
136         .clk(clk),
137         .new_clk(slower_clk_jumper)
138     );
139 
140     sevenSegDisplay inst6 (
141         .clk(clk),
142         .state(state_out_jumper),
143         .visual_counter(sevenSegDisplay_timer_jumper),
144         .an(an),
145         .seg(seg)
146     );
147 endmodule
148 /*
149     // /*
150     // The following commented out code is from alarm clock
151     // please ignore it
152     // */
153     // module stim(
154     //     input clk,
155     //     input btnC,
156     //     input btnU,
157     //     input [15:0] sw,
158     //     output [3:0] an,
159     //     output [6:0] seg,
160     //     output [15:0] led
161     //     );
162 
163     //     wire in_clk;
164 
165     //     clk_div inst2 (
166     //         .clk(clk),
167     //         .new_clk(in_clk)
168     //     );
169 
170     //     alarm inst1 (
171     //         .clk(in_clk),
172     //         .fast_clk(clk),
173     //         .btnC(btnC),
174     //         .btnU(btnU),
175     //         .sw(sw),
176     //         .an(an),
177     //         .seg(seg),
178     //         .led(led)
179     //     );
180 
```

```
181     // endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/09/2021 04:04:10 PM
// Design Name:
// Module Name: tb_top
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module tb_top;

    reg clk, btnC, btnU, btnR;
    reg [15:0] sw;
    wire [3:0] an;
    wire [6:0] seg;
    wire [15:0] led;

    initial begin
        clk = 1;
        btnC = 0;
        btnU = 0;
        btnR = 0;
        sw = 0;
    end

    always begin
        #5 clk = ~clk;
    end

    Top inst11 (
        .clk(clk),
        .btnC(btnC),
        .btnU(btnU),
        .sw(sw),
        .led(led),
        .seg(seg),
        .an(an)
    );

    initial begin
        #1;
        #(10*1)
        #10;
        sw = sw | (1 << 14);
        #10;
        #10;
        btnC = 1;
```

```
        #10;
        btnC = 0;
        #10;
        #10;
        #10;
        #10;
        #10;
        #10;
        sw =  sw | 8'b1100_1011;
        #10;
        #10;
        #10;
        #(1000*10);
        //btnC = 1;
        #10;
        btnC = 0;
        #10;
        #10;
        #10;
        #10;
        #10;
        #10;
        #10;
        #10;
        //btnC = 1;
        #10;
        btnC = 0;
        #(10*4000);
        btnU = 1;
        #10;
        btnU = 0;
        #(10*4000);
        btnU = 1;
        #10;
        btnU = 0;
        #(10*100);
        btnC= 1;
        #10;
        btnC = 0;
        #10;
        #10;
        #10;
        sw[7:0] = 7'b01110110;
        #10;
        #10;
        #(10*100);
        btnC= 1;
        #10;
        btnC = 0;
        #10;

    end
endmodule

```