State University of New York at New Paltz
Division of Engineering Programs
Department of Electrical and Computer Engineering

EGC320 Digital Systems Design

Final Project: Game combo

Christopher Lepore

12/8/2020

Abstract

In this project a game was created using Verilog. The objective of the game is to flip switches 0 to 5 in the correct order given by the 7 segment displays before time runs out. There are 4 difficulties that can be controlled by switch 6 and 7, as the difficulty increases the number of switches that need to be flipped increase and the time given changes as well. In between rounds you can check your high score for each difficulty. Switch 8 and 9 are used to control which high score is shown.

Table of Contents

1. Introduction

The purpose of this project was to create a reaction game in Verilog where the player needs to flip the switches in the correct order before time runs out. In the game, the displayed a set of numbers on the 7-segment displays show the combination the player needs to match. Each 7-segment display has a correlating switch, for example HEX0 correlates with switch 0 (SW0). The number shown in that display tells the player when to flip the switch, so when HEX0 shows a 2 and HEX1 shows a 1, then switch 1 is flipped first and switch 0 is flipped second. Also, if the 7-segment display shows a 0 that means that switch is not part of the combination and should not be flipped. For example, if the display shows a combination of 0-0-0-3-1-2 that means the order the switches should be flipped is SW1 first, SW0 second, SW2 third, and switch 3 to 5 should not be flipped.

2. Design Procedure

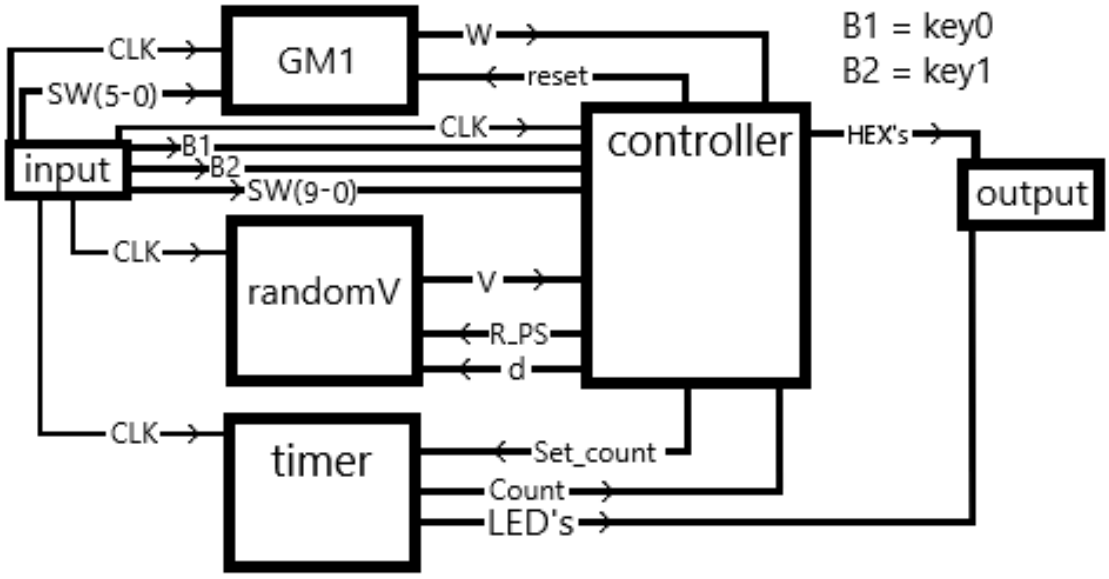

Figure 1 – game combo's block diagram

The overall design uses 3 modules to continuously produce the 3 necessary variables for the game which are an 18-bit vector “V”, a 1-bit variable “W”, and a 4-bit vector count. These modules are controlled by the controller through the wires – reset, d, R_PS, and set_count as shown in table 1 and 2. The controller then outputs 6 vectors from HEX0 to HEX5 to be shown on the “7 segment display”.

Table 1			
Sw7	Sw6	d	x
0	0	1	2
0	1	2	4
1	0	3	6
1	1	4	8

Table 2			
state	Set_count	R_PS	reset
S0	0	0	1
S1	x	1	0
S2	0	0	0
S3	0	0	0

Table 3		
SW9	SW8	High score
0	0	Highscore_1
0	1	Highscore_2
1	0	Highscore_3
1	1	Highscore_4

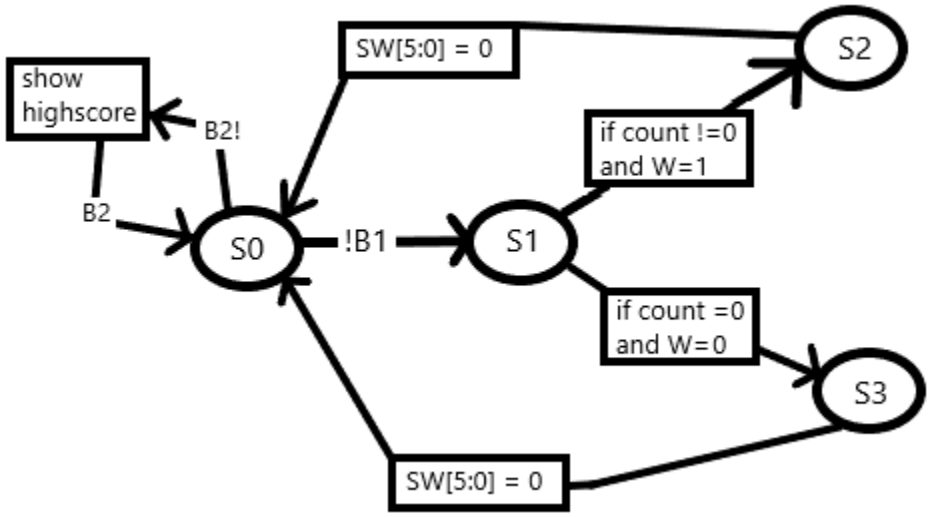

Figure 2 - controller state diagram

The controller module tracks the state of the game and determines the high scores for the player. As shown in Figure 2, S0 is the starting state for the program where the game is not being played. In state S0 the player can start the game by pressing key0, look at their high score by holding down key1 and set the difficulty the player wants to see using sw9 and sw8 as shown in Table 3, and they can set their difficulty using sw7 and sw6 if the round reads a value of 0 like in Figure 6. If sw7 and sw6 are changed in when the round is not 0 or the program is not in state S0 then there is no change until the player losses and the game goes back to round 0. Once the player loses the score is then

compared with the high score for the same difficulty, if the score is higher than the game replaces it. In state S1, the game is being played and the 7-segment display shows the 18-bit vector from the randomV module as shown in Figure 7, 13, 14, 15. If the wire W form the GM1 module becomes a logic 1 before the count from the timer module reaches 0 then the state becomes S2(win state), if not than the state becomes S3(lose state). Once in state S2 or S3 the program then waits for switches 0-5 to all be flipped to logic 0 and then it returns to state S0. In state S2 the 7-segment display will show dash lines at the top and in state S3 they will show dashed lines at the bottom.

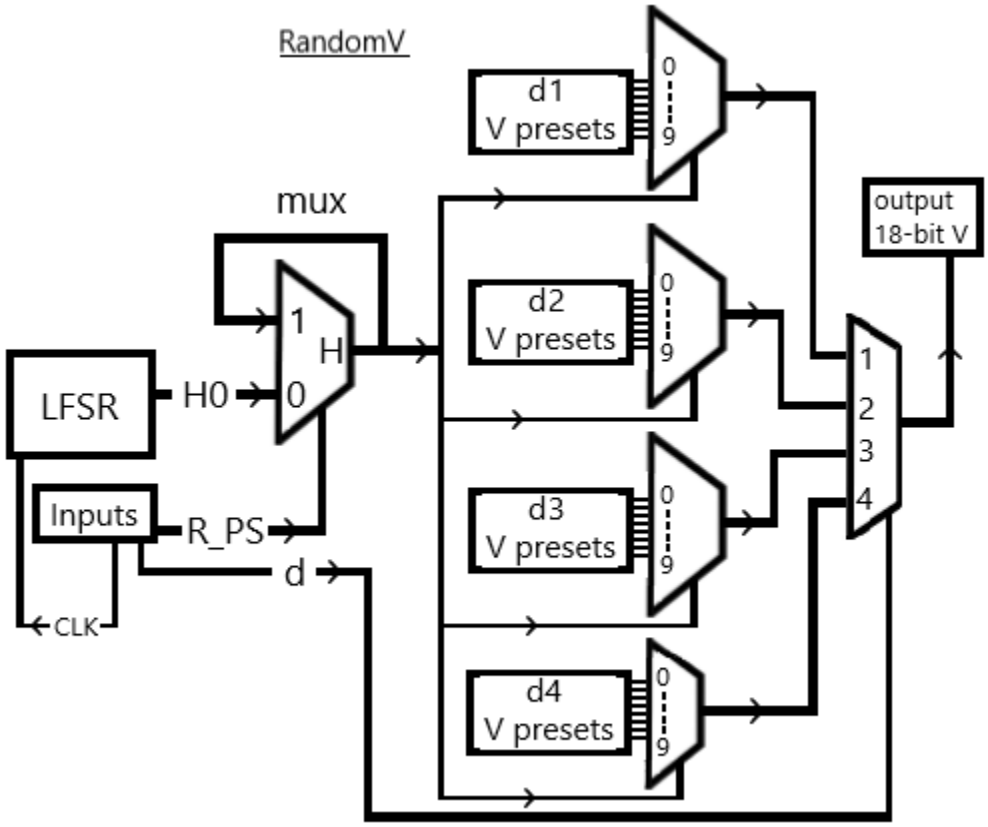

Figure 3 – RandomV module

The objective of the randomV module was to create a random 18-bit vector where every 3 bits represents a number to a total of 6 numbers. the module works by using a linear-feedback shift register (LFSR) that cycles a 4-bit register "H" to a number in between 0 and 9, this number constantly cycles until the controller sends a R_PS output of logic 1 (as the round is being played). then based off the difficulty "d" from the controller and the variable "H" a preset 18-bit vector is outputted.

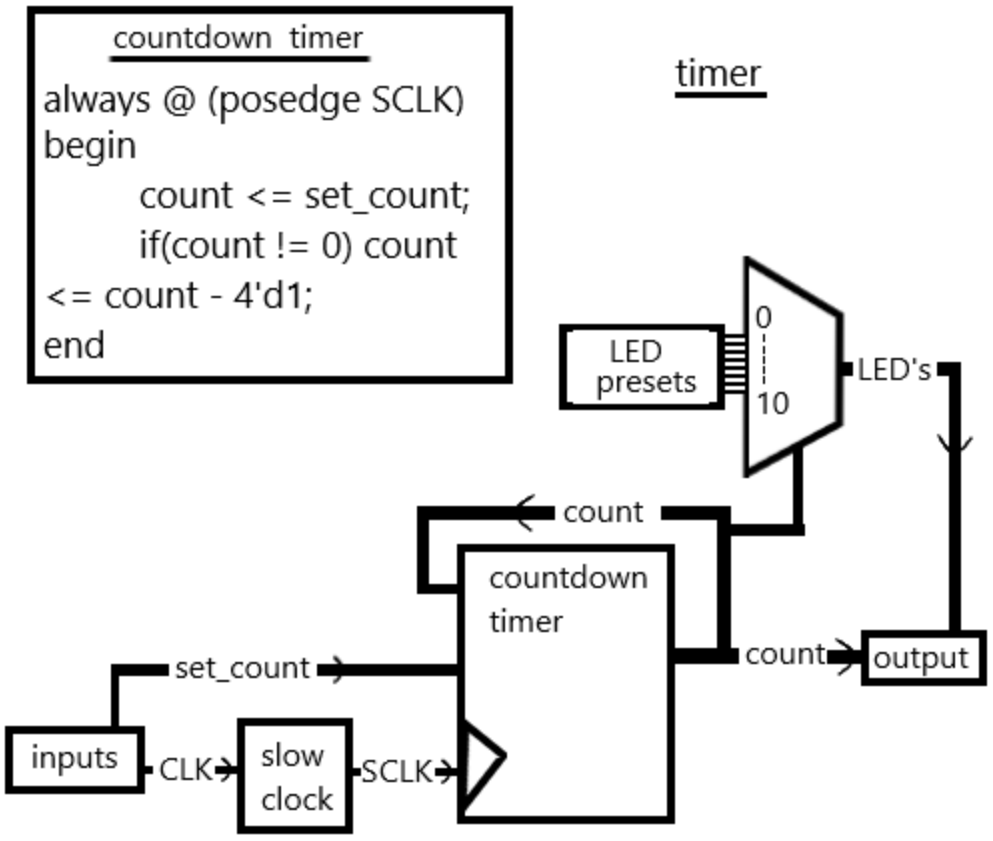

Figure 4 – timer module

The objective of the timer module was to create a 4-bit counter that counts down to 0 and control the LED's based off the counter. this works as follows- once the module is given a "set_count" of non-zero form the controller the counter register will decrease by 1 every slow clock pulse "SCLK" until the counter hits 0. while the counter is counting down the variable "set_count" as no effect. the count output is then put through a case statement to control the LED's.

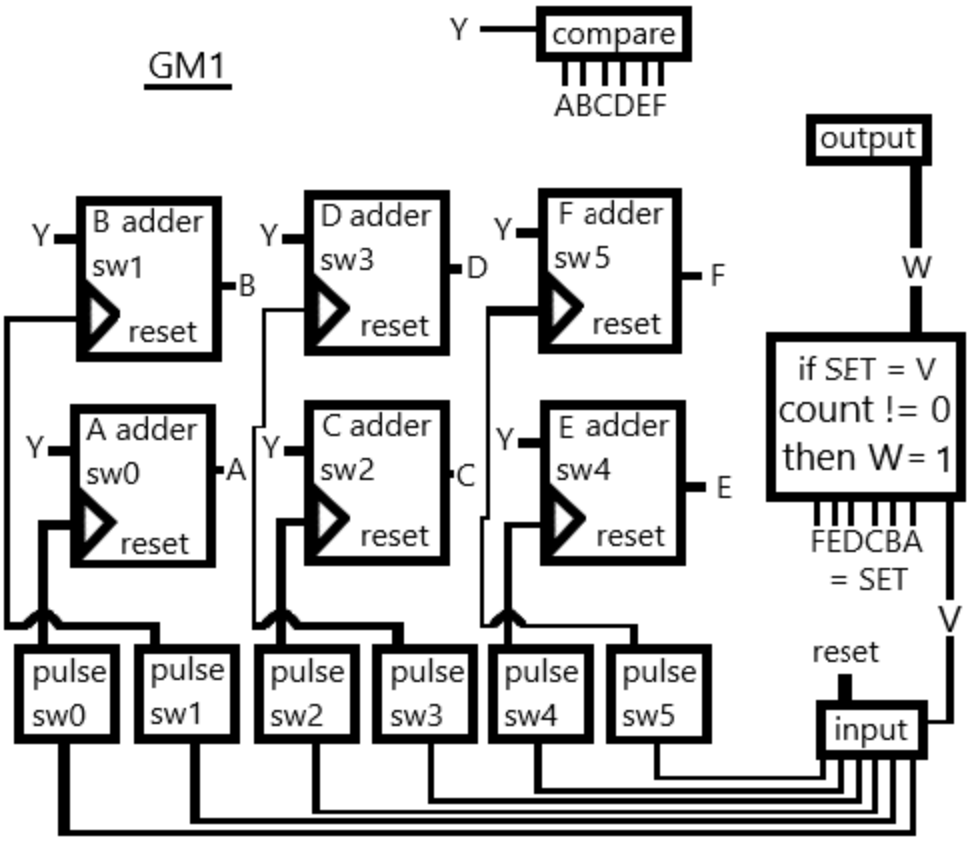

Figure 5 – GM1 module

The objective of the GM1 module was to collect the inputted combination of the switches the player inputted and then test if it matches the combination from the randomV module. the player's combination was recorded through outputting a pulse every time a switch is flipped to logic 1, then every time there is a pulse the correlating register is assigned a value of "Y" increment by 1. the register "Y" is derived from comparing each of the 6 registers (A, B, C, D, E, F) and assigning the highest value to "Y". once the player matches the combination the wire "S" will be assigned to a logic 1 through the code - "assign SET = {F, E, D, C, B, A}; assign S = (SET [17:0] == V[17:0]);" where "V" is the vector created by the randomV module. then once "S" goes from a logic 0 to 1 it checks if the count has reached 0, if not than the output register "W" is set to 1 and the player won the round. after each round the controller outputs a logic 1 for reset which sets registers A, B, C, D, E, F, and W to a value of 0 so that the next round can be played.

3. Verification

Figure 6 – state S0

Figure 7 – state S1, difficulty 1

Figure 8 – win state

Figure 9 – state S0, round increase

Figure 10 – high score, difficulty 1 before round loss

Figure 11 – loss state

Figure 12 – high score, difficulty 1 after round loss

Figure 13 – state S1, difficulty 2

Figure 14 – state S1, difficulty 3

Figure 15 – state S1, difficulty 4

4. Conclusions

This project created a game meant to test the reaction time of the player by flip 6 switches in the order given by the 7-segment display before the timer runs out. The program will keep track of the players score until the player loses then if they beat their high score then it will be replaced. The main problem faced was finding a way to record the players combination by incrementing by 1 each time a switch is flipped and assign it to its correlating register without affecting the other registers. This was solved in the end by generating 1 clock pulse every time a switch is flipped to logic 1 and then increment by 1 based on the pulse. Another issue was finding the correct design for the controller to reset the game variables so the next round can be played and so each state is reached when it is supposed to. This was solved through trial and error to see which design will produce the correct results. Once these problems were fixed the result of the project was a fully functioning game that tested the players reaction and able to run an endless number of rounds and keep track of the players high scores.

Appendix

```
module game_combo (SW, B1, B2, CLK, LED, HEX0, HEX1, HEX2, HEX3, HEX4,
HEX5);
input [9:0] SW;
input B1, CLK, B2;
output [0:6] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
output [9:0] LED;
wire [17:0] V;
wire [3:0] count, set_count, d;
wire W, sp, R_PS, reset;

randomV vector (R_PS, d, CLK, V);
timer t (set_count, CLK, count, LED);                    //main code
GM1 G (reset, SW[5:0], count, CLK, V, W);
controller CC (V, B1, B2, W, CLK, count, SW[9:0], set_count, R_PS, reset, d, HEX0,
HEX1, HEX2, HEX3, HEX4, HEX5);

endmodule
```

```
module controller(V, B1, B2, W, CLK, count, SW, set_count, R_PS, reset, d, HEX0,
HEX1, HEX2, HEX3, HEX4, HEX5);
input [17:0] V;
input [9:0] SW;
input [3:0] count;
input B1, B2, W, CLK;
output reg R_PS, reset;
output reg [3:0] set_count, d;
output reg [0:6] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
parameter [2:0] S0 = 3'd0, S1 = 3'd1, S2 = 3'd2, S3 = 3'd3;
reg cp = 0, ON_c, off_c = 0, slow_clk, SCLK, sp = 0, ON_s, off_s = 0, S = 0, T = 0;
reg [3:0] Hscore_1 = 4'd0, Hscore_2 = 4'd0, Hscore_3 = 4'd0, Hscore_4 = 4'd0, read, x,
score = 4'd0;
reg [1:0] PS = S0, NS;
integer count_c = 0;
integer count_s = 0;

always @ (score, PS, SW[7], SW[6])
begin
        if(score == 4'd0 & PS == S0 & SW[7] == 0 & SW[6] == 0) begin x = 4'd2; d =
4'd1; end                   //difficulty - 1
        else if(score == 4'd0 & PS == S0 & SW[7] == 0 & SW[6] == 1) begin x = 4'd4; d
= 4'd2; end      //difficulty - 2             time
        else if(score == 4'd0 & PS == S0 & SW[7] == 1 & SW[6] == 0) begin x = 4'd6; d
= 4'd3; end //difficulty - 3                 d determines size of the random vector
        else if(score == 4'd0 & PS == S0 & SW[7] == 1 & SW[6] == 1) begin x = 4'd8; d
= 4'd4; end //difficulty - 4                 x determines the time
end

always @ (posedge CLK)
if (count != 0)  off_c = 0;                                     //pulse for count
else
begin
        if (off_c == 0)  ON_c = 1;
        else ON_c = 0;

        if ((ON_c == 1) &&  (count_c <5))
        begin
         count_c = count_c +1;
         cp = 1;
        end
         else if ( (count_c == 5) && (ON_c == 1))
         begin
          count_c = 0;
          cp = 0;
          off_c = 1;
```

```
        end
end


always @ (posedge CLK)
if (S == 0)  off_s = 0;                          //pulse for score(S)
else
begin
        if (off_s == 0)  ON_s = 1;
        else ON_s = 0;

        if ((ON_s == 1) &&  (count_s <5))
        begin
         count_s = count_s +1;
         sp = 1;
        end
         else if ( (count_s == 5) && (ON_s == 1))
         begin
          count_s = 0;
          sp = 0;
          off_s = 1;
         end
end


always @ (posedge sp)                  //if your score is higher than the highscore it
replaces it
begin
        if(T & d == 1 & Hscore_1 <= score) Hscore_1 = score;
        else if(T & d == 2 & Hscore_2 <= score) Hscore_2 = score;
        else if(T & d == 3 & Hscore_3 <= score) Hscore_3 = score;
        else if(T & d == 4 & Hscore_4 <= score) Hscore_4 = score;
        else if(!T) score = score + 4'd1;
        if(T) score = 4'd0;
end

always @ (SW[9], SW[8], Hscore_1, Hscore_2, Hscore_3, Hscore_4)
        //determines which highscore to read
begin
        if(SW[9] == 0 & SW[8] == 0) read = Hscore_1;
        else if(SW[9] == 0 & SW[8] == 1) read = Hscore_2;
        else if(SW[9] == 1 & SW[8] == 0) read = Hscore_3;
        else if(SW[9] == 1 & SW[8] == 1) read = Hscore_4;
end
```

```
always @ (V, PS, cp, SW[9:0], B1, W, score, B2, read, x, count)
begin
	case(PS)
	S0: begin
                                                        //state - game is not being played
                        set_count = 0;
                        R_PS = 0;
                        reset = 1;
                        S = 0;
                        T = 0;

                        if(B2)
                        begin
                                HEX0 = 7'h7E; HEX1 = 7'h7E; HEX2 = 7'h7E; HEX3 =
7'h7E; HEX4 = 7'h7E;
                                case (score) // HEX5
                                0: HEX5 = 7'h01;
                                1: HEX5 = 7'h4F;
                                2: HEX5 = 7'h12;
                                3: HEX5 = 7'h06;
                                4: HEX5 = 7'h4C;
                                5: HEX5 = 7'h24;
                                6: HEX5 = 7'h20;
                                7: HEX5 = 7'h0F;
                                8: HEX5 = 7'h00;
                                9: HEX5 = 7'h04;
                                default: HEX5 = 7'h7F;
                                endcase
                        end

                        else if(!B2) begin
                                HEX5 = 7'h42; HEX4 = 7'h7E; HEX2 = 7'h7E; HEX1 =
7'h7E; HEX0 = 7'h7E;
                                case (read) //HEX3 = "high score";
                                0: HEX3 = 7'h01;
                                1: HEX3 = 7'h4F;
                                2: HEX3 = 7'h12;
                                3: HEX3 = 7'h06;
                                4: HEX3 = 7'h4C;
                                5: HEX3 = 7'h24;
                                6: HEX3 = 7'h20;
                                7: HEX3 = 7'h0F;
                                8: HEX3 = 7'h00;
                                9: HEX3 = 7'h04;
                                default: HEX3 = 7'h7F;
                                endcase
```

```
                end

                if(!B1 & !W) NS = S1; else NS = S0;
        end
S1: begin
                        //game starts

                set_count = x; //determined by difficulty
                R_PS = 1;
                reset = 0;

                case (V[2:0]) //HEX0
                0: HEX0 = 7'h01;
                1: HEX0 = 7'h4F;
                2: HEX0 = 7'h12;
                3: HEX0 = 7'h06;
                4: HEX0 = 7'h4C;
                5: HEX0 = 7'h24;
                6: HEX0 = 7'h20;
                7: HEX0 = 7'h0F;
                default: HEX0 = 7'h7F;
                endcase

                case (V[5:3])   //HEX1
                0: HEX1 = 7'h01;
                1: HEX1 = 7'h4F;
                2: HEX1 = 7'h12;
                3: HEX1 = 7'h06;
                4: HEX1 = 7'h4C;
                5: HEX1 = 7'h24;
                6: HEX1 = 7'h20;
                7: HEX1 = 7'h0F;
                default: HEX1 = 7'h7F;
                endcase

                case (V[8:6])   //HEX2
                0: HEX2 = 7'h01;
                1: HEX2 = 7'h4F;
                2: HEX2 = 7'h12;
                3: HEX2 = 7'h06;
                4: HEX2 = 7'h4C;
                5: HEX2 = 7'h24;
                6: HEX2 = 7'h20;
                7: HEX2 = 7'h0F;
                default: HEX2 = 7'h7F;
                endcase
```

```
                case (V[11:9]) //HEX3
                0: HEX3 = 7'h01;
                1: HEX3 = 7'h4F;
                2: HEX3 = 7'h12;
                3: HEX3 = 7'h06;
                4: HEX3 = 7'h4C;
                5: HEX3 = 7'h24;
                6: HEX3 = 7'h20;
                7: HEX3 = 7'h0F;
                default: HEX3 = 7'h7F;
                endcase

                case (V[14:12])         //HEX4
                0: HEX4 = 7'h01;
                1: HEX4 = 7'h4F;
                2: HEX4 = 7'h12;
                3: HEX4 = 7'h06;
                4: HEX4 = 7'h4C;
                5: HEX4 = 7'h24;
                6: HEX4 = 7'h20;
                7: HEX4 = 7'h0F;
                default: HEX4 = 7'h7F;
                endcase

                case (V[17:15])         //HEX5
                0: HEX5 = 7'h01;
                1: HEX5 = 7'h4F;
                2: HEX5 = 7'h12;
                3: HEX5 = 7'h06;
                4: HEX5 = 7'h4C;
                5: HEX5 = 7'h24;
                6: HEX5 = 7'h20;
                7: HEX5 = 7'h0F;
                default: HEX5 = 7'h7F;
                endcase

                S = 0;
                T = 0;
                if(W) NS = S2;
                if(cp) NS = S3;
        end
S2: begin
//WIN state
                set_count = 0;
                R_PS = 0;
```

```
                        reset = 0;
                        HEX0 = 7'h3F; HEX1 = 7'h3F; HEX2 = 7'h3F; HEX3 = 7'h3F;
HEX4 = 7'h3F; HEX5 = 7'h3F;
                        S = 1;
                        T = 0;
                        if(SW[5:0] == 5'd0 & count == 0) NS = S0;
                end
        S3: begin
        //LOSE state
                        set_count = 0;
                        R_PS = 0;
                        reset = 0;
                        HEX0 = 7'h77; HEX1 = 7'h77; HEX2 = 7'h77; HEX3 = 7'h77;
HEX4 = 7'h77; HEX5 = 7'h77;
                        S = 1;
                        T = 1;
                        if(SW[5:0] == 5'd0 & count == 0) NS = S0;
                end
        default: NS = S0;
        endcase
end

always @ (posedge CLK)                  //changes the state every clock pulse
begin
        PS <= NS;
end
endmodule


module randomV (R_PS, d, CLK, V);
input R_PS, CLK;
input [3:0] d;
output reg [17:0] V;

reg [3:0] H;
reg [3:0] H0;

reg sclk;
reg [4:0] A = 5'b10000;
reg [3:0] B;
integer count =0;
wire S;

assign S = R_PS;

always @ (posedge CLK)                                          //slow clock
```

```
if (count < 50000000) count = count +1;
else begin
        count = 0;
        sclk = ~sclk;
end

always @ (posedge sclk)                                          //LFSR
begin
        A <= {A[2]^A[0],A[4:1]};
        if (A[4:1] > 4'b1001) B <= A[4:1] - 10;
        else
        begin
                B <= A[4:1];
                case (B)
                0: H0 = 4'd0;
                1: H0 = 4'd1;
                2: H0 = 4'd2;
                3: H0 = 4'd3;
                4: H0 = 4'd4;
                5: H0 = 4'd5;
                6: H0 = 4'd6;
                7: H0 = 4'd7;
                8: H0 = 4'd8;
                9: H0 = 4'd9;
                default: H0 = 4'dx;
                endcase
        end

        //////////////
        if(!S) H = H0; // if true the vector cycles else it stays the same
        ////////////

        if (d == 4'd1) //size = 3 d = 1
        begin
                case(H)
                0: V = 18'o000123;
                1: V = 18'o000132;
                2: V = 18'o000213;
                3: V = 18'o000231;
                4: V = 18'o000321;
                5: V = 18'o000312;

                6: V = 18'o000312;
                7: V = 18'o000132;
                8: V = 18'o000213;
                9: V = 18'o000231;
```

```
        default: V = 18'o000123;
        endcase
end
else if (d == 4'd2)  //size = 4 d = 2
begin
        case(H)
        0: V = 18'o003421;
        1: V = 18'o001234;
        2: V = 18'o001342;
        3: V = 18'o001432;
        4: V = 18'o002413;
        5: V = 18'o002143;
        6: V = 18'o003124;
        7: V = 18'o003214;
        8: V = 18'o004321;
        9: V = 18'o004213;
        default: V = 18'o001234;
        endcase
end
else if (d == 4'd3) //size = 5 d = 3
begin
        case(H)
        0: V = 18'o012345;
        1: V = 18'o014532;
        2: V = 18'o025143;
        3: V = 18'o021354;
        4: V = 18'o032145;
        5: V = 18'o035241;
        6: V = 18'o042513;
        7: V = 18'o043125;
        8: V = 18'o054312;
        9: V = 18'o051243;
        default: V = 18'o012345;
        endcase
end
else if (d == 4'd4) //size = 6 d = 4
begin
        case(H)
        0: V = 18'o124365;
        1: V = 18'o231546;
        2: V = 18'o254631;
        3: V = 18'o342156;
        4: V = 18'o364521;
        5: V = 18'o465312;
        6: V = 18'o412536;
        7: V = 18'o541236;
```

```
                8: V = 18'o654321;
                9: V = 18'o621345;
                default: V = 18'o621345;
                endcase
        end

end

endmodule



module timer(set_count, CLK, count, LED);
input [3:0] set_count;
input CLK;
output reg [9:0] LED;
output reg [3:0] count = 4'd0;
integer J = 0;
reg SCLK;


always @ (posedge CLK)                          //slow clock
        if(J < 25000000) J = J+1;
else
begin
        J = 0;
        SCLK = ~SCLK;
end

always @ (posedge SCLK)                         //timer count down
begin
        count <= set_count;
        if(count != 0) count <= count - 4'd1;
end

always @ (count)                                //outputs single for LED's
begin
        case(count)
        10: LED = 10'h3FF;
        9: LED = 10'h1FF;
        8: LED = 10'h0FF;
        7: LED = 10'h07F;
        6: LED = 10'h03F;
        5: LED = 10'h01F;
        4: LED = 10'h00F;
        3: LED = 10'h007;
```

```
        2: LED = 10'h003;
        1: LED = 10'h001;
        0: LED = 10'h000;
        default: LED = 10'h111;
        endcase
end
endmodule


module GM1 (reset, SW, count, CLK, V, W);
input [5:0] SW;
input [17:0] V;
input [3:0] count;
input CLK, reset;
output reg W = 0;
reg ap, bp, cp, dp, ep, fp, sp;
reg [2:0] Y;
reg [2:0] A=0, B=0, C=0, D=0, E=0, F=0;
wire [17:0] SET;
wire S;

reg slow_clk;
integer j =0;

integer count_a = 0, count_b = 0, count_c = 0, count_d = 0, count_e = 0, count_f = 0,
count_s = 0;
reg  ON_a, ON_b, ON_c, ON_d, ON_e, ON_f, ON_s;

                //pulse values
reg off_a = 0, off_b = 0, off_c = 0, off_d = 0, off_e = 0, off_f = 0, off_s = 0;


always @ (posedge CLK)                                          //slow clock
if (j < 2800000) j = j+1;
else begin
j = 0;
slow_clk = ~ slow_clk;
end

always @ ( posedge slow_clk)
if (SW[0] == 0)  off_a = 0;                             //pulse for sw0
else
begin
        if (off_a == 0)  ON_a = 1;
        else ON_a = 0;
```

```
        if ((ON_a == 1) &&  (count_a <5))
        begin
         count_a = count_a +1;
         ap = 1;
        end
         else if ( (count_a == 5) && (ON_a == 1))
         begin
          count_a = 0;
          ap = 0;
          off_a = 1;
         end
        end

always @ ( posedge slow_clk)
if (SW[1] == 0)  off_b = 0;                              //pulse for sw1
else
begin
        if (off_b == 0)  ON_b = 1;
        else ON_b = 0;

        if ((ON_b == 1) &&  (count_b <5))
        begin
         count_b = count_b +1;
         bp = 1;
        end
         else if ( (count_b == 5) && (ON_b == 1))
         begin
          count_b = 0;
          bp = 0;
          off_b = 1;
         end
end


always @ ( posedge slow_clk)                             //pulse for sw2
if (SW[2] == 0)  off_c = 0;
else
begin
        if (off_c == 0)  ON_c = 1;
        else ON_c = 0;

        if ((ON_c == 1) &&  (count_c <5))
        begin
         count_c = count_c +1;
         cp = 1;
        end
```

```
          else if ( (count_c == 5) && (ON_c == 1))
          begin
           count_c = 0;
           cp = 0;
           off_c = 1;
          end
end


always @ ( posedge slow_clk)                          //pulse for sw3
if (SW[3] == 0)  off_d = 0;
else
begin
          if (off_d == 0)  ON_d = 1;
          else ON_d = 0;

          if ((ON_d == 1) &&  (count_d <5))
          begin
           count_d = count_d +1;
           dp = 1;
          end
           else if ( (count_d == 5) && (ON_d == 1))
           begin
            count_d = 0;
            dp = 0;
            off_d = 1;
           end
end


always @ ( posedge slow_clk)                          //pulse for sw4
if (SW[4] == 0)  off_e = 0;
else
begin
          if (off_e == 0)  ON_e = 1;
          else ON_e = 0;

          if ((ON_e == 1) &&  (count_e <5))
          begin
           count_e = count_e +1;
           ep = 1;
          end
           else if ( (count_e == 5) && (ON_e == 1))
           begin
            count_e = 0;
            ep = 0;
```

```
            off_e = 1;
           end
end


always @ ( posedge slow_clk)                                    //pulse for sw5
if (SW[5] == 0)  off_f = 0;
else
begin
          if (off_f == 0)  ON_f = 1;
          else ON_f = 0;

          if ((ON_f == 1) &&  (count_f <5))
          begin
           count_f = count_f +1;
           fp = 1;
          end
           else if ( (count_f == 5) && (ON_f == 1))
           begin
            count_f = 0;
            fp = 0;
            off_f = 1;
           end
end


always @ (A, B, C, D, E, F)
begin
          if(A < F & B < F & C < F & D < F & E < F) Y = F;
          else if(A < E & B < E & C < E & D < E & F < E) Y = E;
          else if(A < D & B < D & C < D & E < D & F < D) Y = D;
          else if(A < C & B < C & E < C & D < C & F < C) Y = C;              //compare
values and outputs highest value
          else if(A < B & E < B & C < B & D < B & F < B) Y = B;
          else if(E < A & B < A & C < A & D < A & F < A) Y = A;
          else Y = A;
end


always @ (posedge ap or posedge reset)
begin
          if(reset) A = 0; else A = Y + 3'b001;                                    //pulse
adder sw0
end

always @ (posedge bp or posedge reset)
```

```
begin
        if(reset) B = 0; else B = Y + 3'b001;                                   //pulse
adder sw1
end

always @ (posedge cp or posedge reset)
begin
        if(reset) C = 0; else C = Y + 3'b001;                                   //pulse
adder sw2
end

always @ (posedge dp or posedge reset)
begin
        if(reset) D = 0; else D = Y + 3'b001;                                   //pulse
adder sw3
end

always @ (posedge ep or posedge reset)
begin
        if(reset) E = 0; else E = Y + 3'b001;                                   //pulse
adder sw4
end

always @ (posedge fp or posedge reset)
begin
        if(reset) F = 0; else F = Y + 3'b001;                                   //pulse
adder sw5
end

assign SET = {F,E,D,C,B,A};

assign S = (SET[17:0] == V[17:0]);                                  //test who wins



always @ ( posedge CLK or negedge S)
if (S == 0)  off_s = 0;                                                         //pulse for S
else
                //if a person matches the combination- 1 single is outputed
begin
        if (off_s == 0)  ON_s = 1;
        else ON_s = 0;

        if ((ON_s == 1) &&  (count_s <5))
        begin
         count_s = count_s +1;
```

```
            sp = 1;
           end
            else if ( (count_s == 5) && (ON_s == 1))
            begin
             count_s = 0;
             sp = 0;
             off_s = 1;
            end
           end

always @ (posedge sp or posedge reset)
begin
        if(reset) W = 0;
        else if(count != 0) W = 1;                          //pulse S checker
end
        // checks if the person won before time ran out

endmodule
```