University of Colorado, Boulder

Electrical & Computer Engineering 2350

Digital Logic

FPGA Reaction Timer

by Connor Humiston

May 1, 2019

Introduction

Electronic devices operate at remarkably fast speeds, with the typical delay through a logic gate being less than 1 ns. This project aims to use a logic circuit to measure the speed of a much slower type of device—a person. A circuit will be designed and coded to measure the reaction time of a person to a specific event. This reaction timer will be implemented on the MAX 10 (10M50DAF484C7G) Field Programmable Gate Array (FPGA) utilizing Verilog Hardware Description Language (VHDL) and Quartus Prime. The circuit will illuminate an LED on the Terasic DE10-LITE board a random amount of time after a "start" button is pushed and released. The goal is for the player to switch on the "stop" switch that corresponds to a randomly lit LED as quickly as possible. The reaction time will be displayed in milliseconds on the board's seven-segment display after the "stop" switch is turned on. Furthermore, if the start or stop inputs triggered before the LED is lit, then the random timer is stopped, and everything is reset. An asynchronous reset switch will also reset the game to display the latest high score. The DE10-LITE board inputs and outputs are described in Figure 1 below. The Verilog modules are summarized in a block diagram that follows in Figure 2.

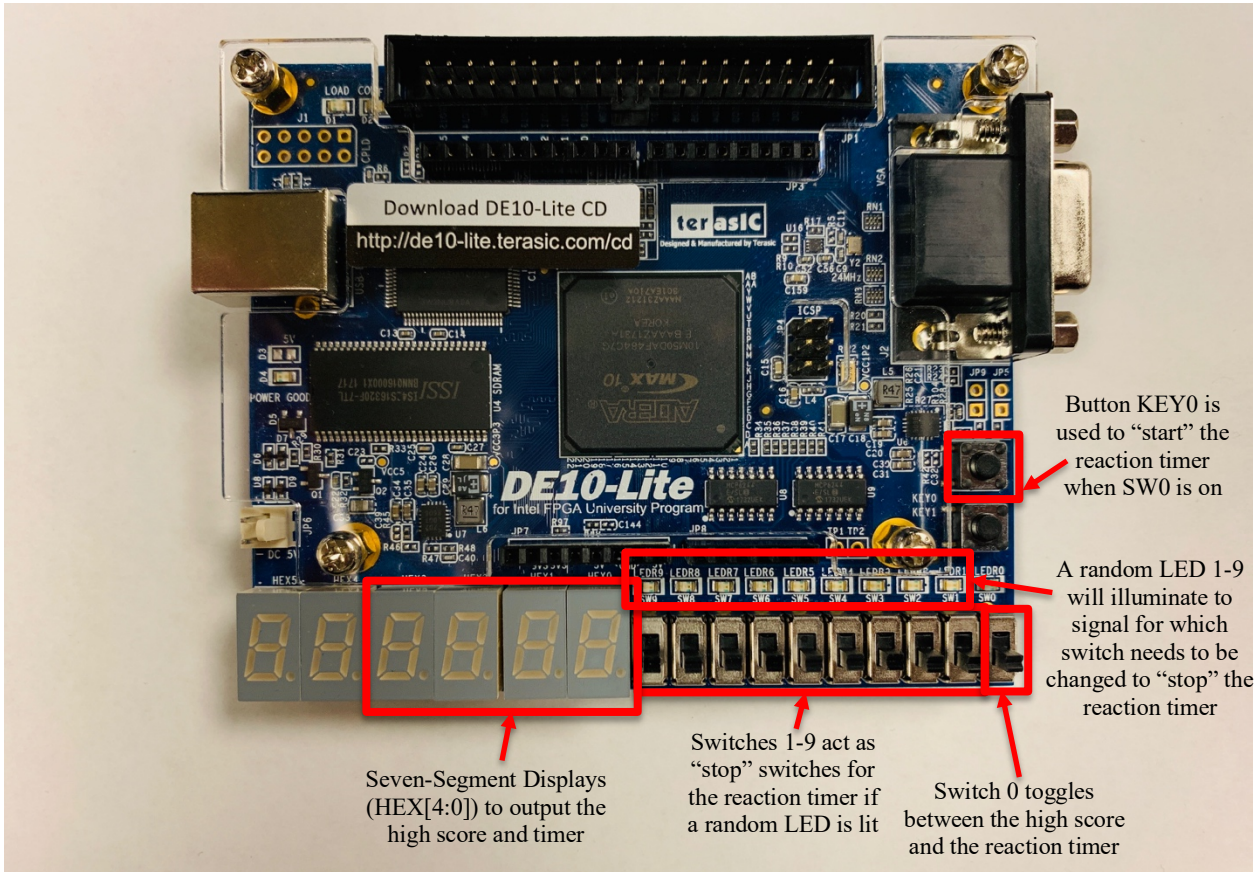

Figure 1: The DE10-Lite board with seven-segment displays, switches, LEDs, and button keys labeled

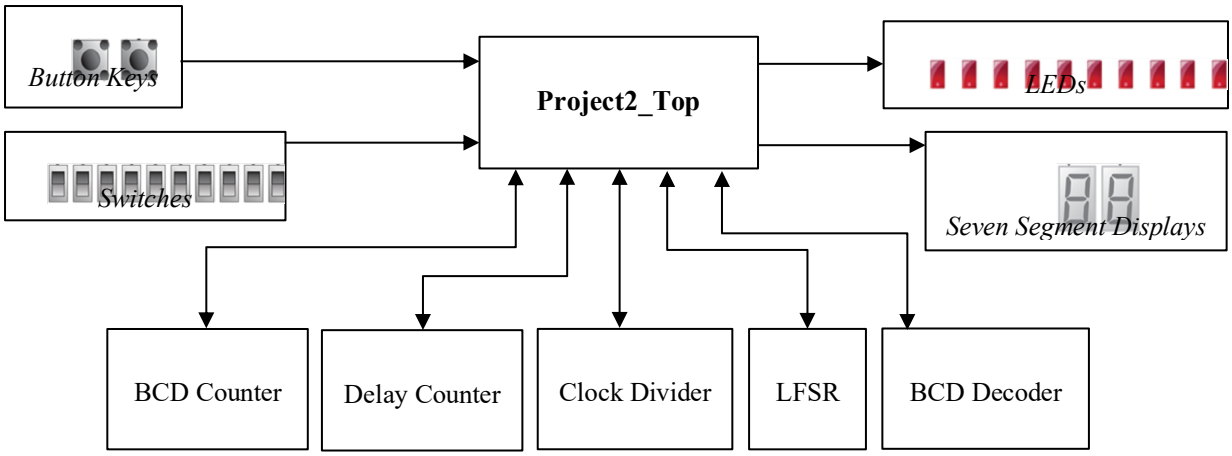

Figure 2: Block diagram (above) including key and switch inputs, project modules and functions, and LED and display outputs

The Finite State Machine

A finite state machine was chosen to implement the reaction timer concept due to its sequential nature. The state diagram illustrating the transitions between these states can be found in Figure 3 below. The FSM is a Mealy machine because its output values are determined by both its current state and the current inputs, more specifically the switches and buttons. The machine begins in the s_0 state where the reaction timer is not running, but the current high score is displayed. Then, if switch 1 is turned on, the state becomes s_1 where the reaction timer sits idle, but nothing is being displayed on either the HEXs or LEDs. If in state s_1 and the button KEY0 is pressed, the machine transitions to s_2 which acts as a random delay period before a random LED lights. Then, after that delay has been counted out by a down-counter, the state automatically switches to s_3 as a random LED illuminates. s_3 is the state in which the user must switch the corresponding switch under the randomly lit LED to stop the timer as quickly as possible. Once the correct switch as been switched, the machine moves to state s_4 which displays that score. If at any time switch 0 is turned off, state s_0 with the high score will be reset. An input functionality table and functional state table are shown in Figures 4 and 5.

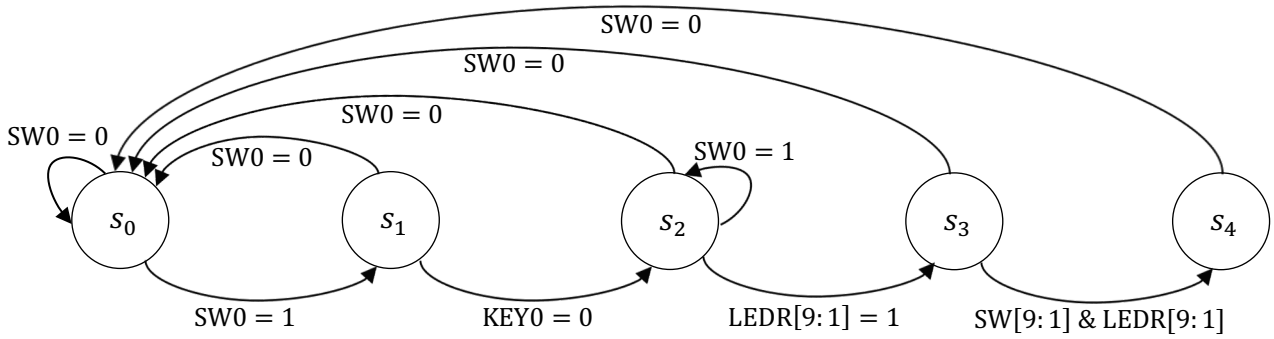

Figure 3: State diagram assuming button keys are high by default (some do not care states are ignored)

Input	Functionality
$\mathrm{SW}[0] = 0$	Reset/show high score
$\mathrm{SW}[0] = 1$	Begin/use reaction timer
$\mathrm{KEY}[0]$	Start Button
$\mathrm{SW}[9:1] = 1$	Reaction "stop" switches

Figure 4: Input Functionality Table

State	Functionality
s_0	Reset/primary state & high score viewer
s_1	Reaction timer switched on
s_2	Start button pressed and random amount of time elapses before LED lights
s_3	Random LED has illuminated, and reaction timer has started counting
s_4	Stop switch is switched on after LED is lit & time is revealed on displays

Figure 5: Functional State Table

Results

The Verilog code for the following findings can be found in Appendix A.

Project2_Top

As displayed by the block diagram in Figure 2 above, Project2_Top takes in the two buttons and ten switches as inputs, and utilizes the BCD Counter, Delay Down Counter, Clock Divider, LFSR, and BCD Decoder modules to create outputs for the ten LEDs and four seven-segment display units.

The Project2_Top is where the FSM can be found using a series of conditional statements that change every time the clock changes. The first always block determines the state based on the inputs. For example, if the state is 0 and SW0 is turned on, then the state register will be changed to 1 like the state diagram showed in Figure 3. The next always block changes whenever the state changes and assigns outputs based on that state. In the case statement, for instance, if the state is 3'b000, then the high score resister is output to the HEX displays, and the LEDs remain off. In addition to the HEX and LED determination in this always block, the counter and LFSR enables and reset are assigned depending on the state. The last always block we find in the top module functions as a decoder to take the LED selector register and convert it into a series of bits to turn on one LED at a time.

Clock_Divider

The clock divider is one of the most important modules as it keeps everything in sync. Both a 1kHz and 1Hz clock were created from the 10MHz clock on from the board. This was done by using a parameter to determine how many clock cycles from the 10MHz clock needed to be counted to divide that clock to 1kHz and 1Hz. For the case of the 1kHz, every 10MHz positive edge clock cycle was counted until the counter reached the 5,000-parameter value. 5,000 was used even though the clock was being divided by 10,000 because the new 1kHz clock cycle should be high for 5,000 cycles of that period and low for the other 5,000 cycles. The same thing was done for the 1Hz clock, except with a parameter of 5,000,000 because the 10MHz board clock needed a larger division.

BCD_Counter

The BCD Counter counts in decimal digits from 0 to 9999 to be output to the seven segment displays for the user to see their time. The counter counts in milliseconds because it is passed the 1000Hz clock that was created in the Clock_Divider module as described above. 1000kHz translates to 1/1000 of a second, which is simply 1 millisecond. Hence, the BCD Counter counts every one millisecond.

Down_Counter

This module simply counts down from the LFSR output value until it reaches 0 at which time the LED illuminates. Hence, the Down_Counter module was imperative in executing the random delay.

LFSR

The LFSR, or linear feedback shift register, was used to create a random number that could be utilized for counting the random delay before LED illumination and in choosing a random LED to be lit for a more difficult game. Two LFSRs were used. The first LFSR outputs a random number between 0 and 15 to be used in determining the delay. This output from 0 to 15 was then computed with modulus 7 to get a remainder output between 0 and 7. That number between 0 and 7 was then passed to the down counter for a random delay. This “random” delay (random is in parentheses because LFSRs actual follow a linear pattern) was tested 64 times, and the outputs of this test are shown in the histogram in Figure 6.

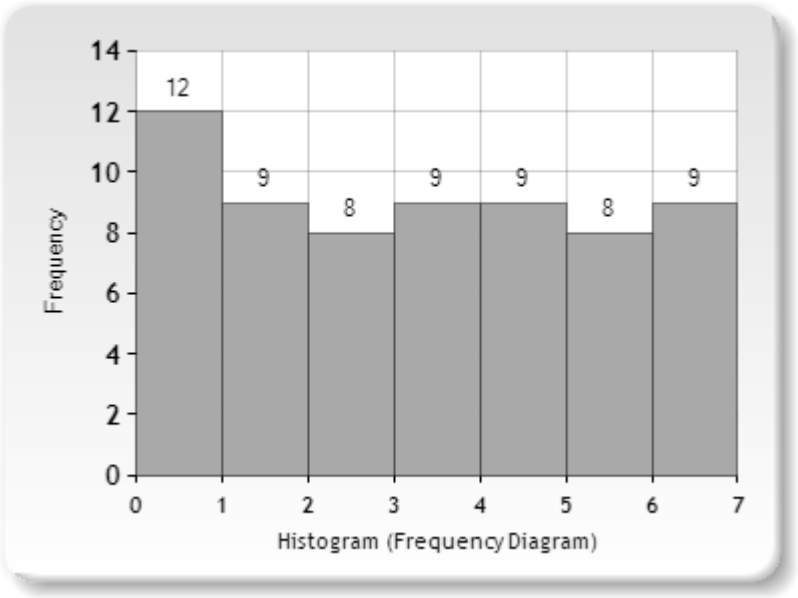

Figure 6: Histogram of the LFSR delay outputs from 0 to 7 in seconds

Hence, although the delay is not completely random nor perfectly distributed as some outputs appear more than others, this minute difference in frequency will be unnoticeable by a human user. The second LFSR was used to determine a random LED. This LFSR outputs a value from 0 to 7 as well, but was then added to 2 in order to span 2-9. The range from 2-9 was desired because these are the LEDs above switches 2-9 that are used to stop the reaction timer. Hence, the LEDs are illuminated at random. Furthermore, since the LSFR is always running, that number changes, and different LEDs light up over time, making the reaction timer that much harder.

BCD_Decoder

This module takes in binary inputs and outputs hexadecimal proper 8 bit binary to light up the LEDs on the seven-segment display. It is important to note that the LEDs on the seven segment displays are high by default.

Conclusion

The purpose of this project was to create a reaction timer using an FPGA and Verilog code in Quartus Prime. This goal was met aside from one disagreement. That disagreement is between the random delay after the start button has been pressed, and the display of the high score. For an unknown reason, when the random number from the LSFR was capable of executing a random delay, the high score function did not work. Similarly, when the high score was storing that score

in a register after comparing it to the current reaction time, there was no delay at all before an LED had been illuminated. After countless hours in office hours, a compromise was reached, and the final version of code choses the high score function over the delay. With this being the only issue, everything else worked according to plan. The states change on the proper inputs and the seven segment displays display the correct outputs when they are supposed to. Furthermore, an extra bit of randomness was added by illuminating a random LED that corresponds to the switch that needs to be “stopped” to stop the timer. To go further, the LED that lights up changes every second (because it is on the 1Hz clock cycle), so if the user misses the first LED switch, they must follow it to the second, adding more challenge to the game. In conclusion, although creation of the modules themselves was relatively easy, getting the modules to work together in sync became the most difficult part of the project. In conclusion, although I learned a lot, if I were to do it again, I realize not that it would have worked out better if I had started by putting more though into the FSM before jumping straight into Verilog code.

Appendix A

Top Module

```
// Connor Humiston
// ECEN2350
// University of Colorado, Boulder
// Project 2

module Project2_Top(
	output [7:0] HEX0,
	output [7:0] HEX1,
	output [7:0] HEX2,
	output [7:0] HEX3,
	//output [7:0] HEX4,
	//output [7:0] HEX5,
	output [9:0] LEDR,
	input  [9:0] SW,
	input  KEY0,
	input  KEY1,
	input  CLK_50MHZ,
	input CLK_10MHZ
);
	wire clk_1000Hz;
	wire clk_1Hz;
	reg[2:0] state = 0;
	wire [3:0] BCD3, BCD2, BCD1, BCD0;
	reg [3:0] display3out, display2out, display1out, display0out;
	reg [15:0] highscore = 16'b1001100110011001;
	wire [3:0] LFSRout;
	//wire [15:0] LFSRout;
	reg [9:1] leds;
	reg [3:0] ledselect;
	wire [3:0] downCounterOut;
	reg counterReset;
	reg counterEnable;
	reg LFSRenable;
	reg downCounterEnable;
	wire [12:0] LFSRout2;

	//Clock Dividers into 1000Hz (ms) and 1Hz (s)
	Clock_Divider1000 divider1000(CLK_10MHZ, 0, clk_1000Hz);
	Clock_Divider1	  divider1(CLK_10MHZ, 0, clk_1Hz);

	//Output to the displays
	BCD_Decoder segment0(display0out[3:0], HEX0[7:0]);
	BCD_Decoder segment1(display1out[3:0], HEX1[7:0]);
```

```
	BCD_Decoder segment2(display2out[3:0], HEX2[7:0]);
	BCD_Decoder segment3(display3out[3:0], HEX3[7:0]);
	//BCD_Decoder segment4((LFSRout2[12:0] % 7) + 2, HEX4[7:0]);
	//BCD_Decoder segment5((LFSRout[3:0] % 7), HEX5[7:0]);
	//BCD_Decoder segment5(state[2:0], HEX5[7:0]);

	//Counter                                                    reset should be 0 and enable
1 to count
	BCD_Counter counter1(clk_1000Hz, counterReset, counterEnable, BCD0[3:0],
BCD1[3:0], BCD2[3:0], BCD3[3:0]);

	//Random number generator
	//LFSR random(clk_1000Hz, LFSRenable, 0, 16'b0, LFSRout);
	LFSR random(clk_1Hz, LFSRenable, LFSRout[3:0]);
	//LFSR7 random2(2, 0, clk_1Hz, LFSRout2[3:0]);
	LFSRl random2(LFSRout2, clk_1Hz, 0);
	//LFSR random(1, LFSRenable, clk_1Hz, LFSRout);

	//Down counter for delay
	Down_Counter delay1(clk_1Hz, downCounterEnable, (LFSRout[3:0] % 7),
downCounterOut);

	//State determination from inputs
	always @(clk_1000Hz)
	begin
		if(state[2:0] == 0) //State 0
			if(SW[0] == 1)
				state[2:0] = 1;
		if(state[2:0] == 1) //State 1
			if(~KEY0)
				state[2:0] = 2;
		if(state[2:0] == 2) //State 2
			if(downCounterOut == 0)
				state[2:0] = 3;
		//if(state[2:0] == 3) //State 3
		//	if(SW[9:1] & LEDR[9:1])
		//		state[2:0] = 4;

		if(SW[0] == 0)		//Reset
			state[2:0] = 0;
	end

	//State machine outputs
	always @(state)
	begin
		case(state)
```

```
3'b000://0: reset and highscore
begin
                display3out[3:0] = highscore[3:0];
                display2out[3:0] = highscore[7:4];
                display1out[3:0] = highscore[11:8];
                display0out[3:0] = highscore[15:12];
                ledselect[3:0] = 4'b0000;

                counterReset = 1; //clears the counter
                counterEnable = 0; //counter not enabled

                LFSRenable = 1; //enabling the LFSR to run in the
background and select a random number

                downCounterEnable = 0;
end

3'b001://1: enters game before start button pressed
begin
                display3out[3:0] = 0;
                display2out[3:0] = 0;
                display1out[3:0] = 0;
                display0out[3:0] = 0;
                ledselect[3:0] = 4'b0000;

                counterReset = 0;
                counterEnable = 0;

                LFSRenable = 1;

                downCounterEnable = 0;
end

3'b010://2: start button pressed & random time elapses
begin
                display3out[3:0] = 0;
                display2out[3:0] = 0;
                display1out[3:0] = 0;
                display0out[3:0] = 0;
                ledselect [3:0] = 4'b0000; //the LED select is
assigned to the random LED passed from LSFR

                counterReset = 0;
                counterEnable = 0;
```

```
                                        LFSRenable = 0; //keeps the previous value
generated in previous states

                                        downCounterEnable = 1; //the down counter begins
and the state will change when its output equals 0
                end

                3'b011://3: random LED lights and counter begins
                begin
                                        display3out[3:0] = BCD3[3:0];
                                        display2out[3:0] = BCD2[3:0];
                                        display1out[3:0] = BCD1[3:0];
                                        display0out[3:0] = BCD0[3:0];
                                        //ledselect [3:0] = (LFSRout2[3:0] % 7) + 2; //this is
a random number from 2 to 9 for LEDs
                                        ledselect [3:0] = (LFSRout2 % 7) + 2;
                                        //ledselect [3:0] = a + 2; //this is a random number
from 2 to 9 for LEDs

                                        counterReset = 0;
                                        counterEnable = 1;

                                        LFSRenable = 0;

                                        downCounterEnable = 0;

                                        if(SW[9:1] & LEDR[9:1])
                                        begin
                                                counterEnable = 0;
                                                if(highscore > {BCD0, BCD1, BCD2,
BCD3})
                                                begin

                                                        if({BCD0, BCD1, BCD2, BCD3} !=
0)
                                                                highscore[15:0] = {BCD0,
BCD1, BCD2, BCD3};
                                                end
                                        end
                end

                3'b100://4: stop switch switched to end counter
                begin
                                        display3out[3:0] = BCD3[3:0]; //keeps score
                                        display2out[3:0] = BCD2[3:0];
                                        display1out[3:0] = BCD1[3:0];
```

```
                                        display0out[3:0] = BCD0[3:0];
                                        //ledselect [3:0] = (LFSRout % 9) + 2;

                                        counterReset = 0;
                                        counterEnable = 0; //keeps the counter number the
same

                                        LFSRenable = 0;

                                        downCounterEnable = 0;

                                        //high score compared
                                        if(highscore > {BCD0, BCD1, BCD2, BCD3})
                                        begin
                                                if({BCD0, BCD1, BCD2, BCD3} != 0)
                                                        highscore[15:0] = {BCD0, BCD1,
BCD2, BCD3};
                                        end
                        end
                        /*
                        default:        //everything off by default
                        begin
                                        display3out[3:0] = 0;
                                        display2out[3:0] = 0;
                                        display1out[3:0] = 0;
                                        display0out[3:0] = 0;
                                        ledselect[3:0] = 4'b0000;
                                        counterReset = 0;
                                        counterEnable = 0;
                                        LFSRenable = 1;
                                        downCounterEnable = 0;
                        end
                        */
                endcase
        end

        //LED decoder
        always @(ledselect)
        begin
                case(ledselect)
                        0: leds[9:1] = 9'b000000000;
                        1: leds[9:1] = 9'b000000001;
                        2: leds[9:1] = 9'b000000010;
                        3: leds[9:1] = 9'b000000100;
                        4: leds[9:1] = 9'b000001000;
                        5: leds[9:1] = 9'b000010000;
```

```
                        6: leds[9:1] = 9'b000100000;
                        7: leds[9:1] = 9'b001000000;
                        8: leds[9:1] = 9'b010000000;
                        9: leds[9:1] = 9'b100000000;
                        default: leds[9:1] = 9'b000000000;
                endcase
        end

        //Assigning LEDs to desired output
        assign LEDR[9:1] = leds[9:1];

endmodule
```

BCD Counter

```
// Connor Humiston
// ECEN2350
// University of Colorado, Boulder
// Project 2

//This module counts in milliseconds for output to the 7seg display
module BCD_Counter(clock, reset, enable, BCD0, BCD1, BCD2, BCD3);
        input clock, reset, enable;
        output reg [3:0] BCD0, BCD1, BCD2, BCD3;

        always @(posedge clock, posedge reset)//, posedge enable)
        begin
                if (reset)
                begin
                        BCD0 <= 4'b0000;
                        BCD1 <= 4'b0000;
                        BCD2 <= 4'b0000;
                        BCD3 <= 4'b0000;
                end

                else if (enable)
                begin
                        BCD0 <= BCD0 + 1;
                        if (BCD0 == 4'b1001)
                        begin
                                BCD0 <=  0;
                                BCD1 <= BCD1 + 1'b1;
                                if (BCD1 == 4'b1001)
                                begin
                                        BCD1 <= 0;
```

```
                                        BCD2 <= BCD2 + 1'b1;
                                        if (BCD2 == 4'b1001)
                                        begin
                                                BCD2 <= 0;
                                                BCD3 <= BCD3 + 1;
                                        end
                                end
                        end
                end

                else if(!enable)
                begin
                        BCD0 <= BCD0;
                        BCD1 <= BCD1;
                        BCD2 <= BCD2;
                        BCD3 <= BCD3;
                end

        end
endmodule
//BCD counter module
```

Clock Divider

```
// Connor Humiston
// ECEN2350
// University of Colorado, Boulder
// Project 2

//1000Hz clock divider
module Clock_Divider1000(clock, reset, clk_out_1000Hz);
        input clock, reset;
        output clk_out_1000Hz;

        parameter WIDTH = 24; // Width of the register required
        parameter N = 5_000; //divides 10Mhz by 10,000

        reg [WIDTH-1:0] r_reg;
        wire [WIDTH-1:0] r_nxt;
        reg clk_track;

        always @(posedge clock or posedge reset)
        begin
                if (reset)
```

```
                begin
                        r_reg <= 0;
                        clk_track <= 1'b0;
                end

                else if (r_nxt == N)
          begin
                        r_reg <= 0;
                        clk_track <= ~clk_track;
          end

                else
                        r_reg <= r_nxt;
        end

        assign r_nxt = r_reg + 1;
        assign clk_out_1000Hz = clk_track;
endmodule


//1Hz clock divider
module Clock_Divider1(clock, reset, clk_out_1Hz);
        input clock, reset;
        output clk_out_1Hz;

        parameter WIDTH = 24; // Width of the register required
        parameter N = 5_000_000; //divides 10Mhz by 10,000,000

        reg [WIDTH-1:0] r_reg;
        wire [WIDTH-1:0] r_nxt;
        reg clk_track;

        always @(posedge clock or posedge reset)
        begin
                if (reset)
                begin
                        r_reg <= 0;
                        clk_track <= 1'b0;
                end

                else if (r_nxt == N)
          begin
                        r_reg <= 0;
                        clk_track <= ~clk_track;
          end
```

```
                else
                        r_reg <= r_nxt;
        end

        assign r_nxt = r_reg + 1;
        assign clk_out_1Hz = clk_track;
endmodule
```

LFSR

```
// Connor Humiston
// ECEN2350
// University of Colorado, Boulder
// Project 2

//generates a random number between 0 and 7
module LFSR7(randomIn, reset, clock, out);
        input [0:2] randomIn;
        input reset, clock;
        output reg [0:2] out;

        always @(posedge clock)
                if(reset)
                        out <= randomIn;
                else
                        out <= {out[2], out[0] ^ out[2], out[1]};
endmodule


//generates a random number between 0 and 15
module LFSR(clock, enable, out);
        input clock, enable;
        output reg [3:0] out;
        wire feedback;

        assign feedback = ~(out[3] ^ out[2]);

        always @(posedge clock)
        begin
                if(enable)
                        out = {out[3:0], feedback};
                else
                        out = 4'b0;
```

```
        end
endmodule


module LFSR1 (out, clk, rst);

  output reg [3:0] out;
  input clk, rst;

  wire feedback;

  assign feedback = ~(out[3] ^ out[2]);

always @(posedge clk, posedge rst)
  begin
    if (rst)
      out = 4'b0;
    else
      out = {out[2:0],feedback};
  end
endmodule
```

Down Counter/Delay

```
// Connor Humiston
// ECEN2350
// University of Colorado, Boulder
// Project 2

module Down_Counter(clk, enable, LFSRout, downout);
        input clk, enable;
        input [3:0] LFSRout;
        output reg [3:0] downout;

        always @ (posedge clk)
        begin
                if(enable)
                        downout <= downout - 1'b1;
                else
                        downout <= LFSRout;
        end
endmodule
//down counter that takes random value from lfsr
```

BCD Decoder

```
// Connor Humiston
// ECEN2350
// University of Colorado, Boulder
// Project 2

//this is the bcd to seven segment display decoder
module BCD_Decoder(bcd, leds);
	input [3:0] bcd;
	output reg [7:0] leds;

	always @(bcd)
		case (bcd)		//abcdefg
			0: leds = 8'b11000000;
			1: leds = 8'b11111001;
			2: leds = 8'b10100100;
			3: leds = 8'b10110000;
			4: leds = 8'b10011001;
			5: leds = 8'b10010010;
			6: leds = 8'b10000010;
			7: leds = 8'b11111000;
			8: leds = 8'b10000000;
			9: leds = 8'b10010000;
			default: leds = 8'b11000000;
		endcase
endmodule
```