//Gabriel Altman

//ECEN2350 Digital Logic

//April, 2018

module Project\_2\_TOP

(

MAX10\_CLK1\_50,

MAX10\_CLK2\_50,

LEDR,

HEX0,

HEX1,

HEX2,

HEX3,

HEX4,

HEX5,

KEY,

SW

);

input MAX10\_CLK1\_50; //50Mhz system clock 1

input MAX10\_CLK2\_50; //50Mhz system clock 2

input [1:0] KEY; //Push-Buttons

input [9:0] SW; //Switches

output [9:0] LEDR; //LED's (Surface mounted)

output [7:0] HEX0; //Seven segment display 0

output [7:0] HEX1; //Seven segment display 1

output [7:0] HEX2; //Seven segment display 2

output [7:0] HEX3; //Seven segment display 3

output [7:0] HEX4; //Seven segment display 4

output [7:0] HEX5; //Seven segment display 5

reg [1:0]CLEAR; //Boolean to clear the BCD Counter

wire [1:0]RESET; //Reset boolean

wire [3:0]BCD0; //BCD 0

wire [3:0]BCD1; //BCD 1

wire [3:0]BCD2; //BCD 2

wire [3:0]BCD3; //BCD 3

wire [9:0]divided\_Clock; //the 2kHz clock signal

wire [9:0]Slow\_Clock; //The 1Hz clock signal

wire [7:0]LFSR\_Value; //The random number generated by the LSFR

wire [1:0]downCount\_Complete; //boolean indicating the down\_Counter has completed its count

wire [7:0]count; //the value returned by the down\_Counter showing the current value

wire [3:0]display3; //Carries the BCD values from the MUX to the decoder

wire [3:0]display2; //Carries the BCD values from the MUX to the decoder

wire [3:0]display1; //Carries the BCD values from the MUX to the decoder

wire [3:0]display0; //Carries the BCD values from the MUX to the decoder

reg counter\_Enable; //boolean to enable the counter

reg downCounter\_Enable; //boolean to enable the down\_Counter

reg [15:0]highScore; //High score

reg [15:0]currentScore; //current score

reg [2:0]A; //Current state of Finite State Machine

reg downCount\_Boolean; //count complete=1 count running=0

reg displayState; //Controls the output of display\_MUX. 0 = display //current score, 1 = display High Score

assign RESET[0] = SW[0]; //Toggles the RESET boolean

assign HEX0[7] = 1'b1; //Turns OFF the decimal point for the 7-segment display

assign HEX1[7] = 1'b1; //Turns OFF the decimal point for the 7-segment display

assign HEX2[7] = 1'b1; //Turns OFF the decimal point for the 7-segment display

assign HEX3[7] = 1'b0; //Turns ON the decimal point for the 7-segment display

assign HEX4[7] = 1'b1; //Turns OFF the decimal point for the 7-segment display

assign HEX5[7] = 1'b1; //Turns OFF the decimal point for the 7-segment display

assign HEX4 = 8'b11111111; //Blanks digit 4 of the 7-segment display

assign HEX5 = 8'b11111111; //Blanks digit 5 of the 7-segment display

assign LEDR[9] = SW[0]; //indicator for the reset switch

assign LEDR[8] = downCounter\_Enable; //indicator for the downCount\_Complete boolean 1=count //complete

assign LEDR[7] = counter\_Enable; //indicator for the counter\_Enable boolean 1=count complete

assign LEDR[6] = displayState; //indicator for the displayState. 0 = display current //score, 1 = display High Score

assign LEDR[5] = downCount\_Complete; //indicator for the downCount\_Complete boolean 1=count //complete

assign LEDR[4] = downCount\_Complete; //indicator for the downCount\_Complete boolean 1=count //complete

assign LEDR[3] = downCount\_Complete; //indicator for the downCount\_Complete boolean 1=count //complete

assign LEDR[2:0] = A[2:0]; //TEMPORARY indicator to show current state of FSM

//assign LEDR[7:0] = LFSR\_Value[7:0]; //TEMPORARY indicator to show LFSR values to make a //histogram for the report

always@(A, KEY[1], KEY[0], SW[0], downCount\_Complete)

begin

if(SW[0]==1)begin //RESET ALL CONDITIONS

highScore <= 0;

A<=3'b000;

counter\_Enable <= 0;

downCounter\_Enable <= 0;

displayState <= 0;

highScore <= 16'b1001100110011001;

//highScore <= 16'b0000000000000000;

end

if(A == 3'b000 && SW[0] ==1'b0 && KEY[1] == 1'b0)begin

A <= 3'b001;

counter\_Enable <= 0;

downCounter\_Enable <=0;

displayState <= 1;

end

else if(A == 3'b001 && downCount\_Complete == 0 && KEY[0] == 1'b0)begin //

downCounter\_Enable <= 1'b1;

A <= 3'b010;

CLEAR[0]=1'b1; //Clears the hex display by resetting values to 0

displayState <= 0;

end

else if(A == 3'b010 && downCount\_Complete == 1)begin

CLEAR[0]=1'b0; //Clears the hex display by resetting values to 0

counter\_Enable <= 1;

downCounter\_Enable <=0;

A <= 3'b011;

displayState <= 0;

end

else if(A == 3'b011 && KEY[0] == 1'b0)begin

counter\_Enable <= 0;

downCounter\_Enable <=0;

A <= 3'b000;

displayState <= 0;

if(highScore[15:0] > {BCD3[3:0],BCD2[3:0],BCD1[3:0],BCD0[3:0]})begin

highScore[15:0] <= {BCD3[3:0],BCD2[3:0],BCD1[3:0],BCD0[3:0]};

end

end

end

//Instantiates an instance of display\_MUX.v

//This is the MUX which determines if the display will show the high score or the current score.

display\_MUX display\_MUX\_INST1(displayState, BCD3[3:0], BCD2[3:0], BCD1[3:0], BCD0[3:0], highScore[15:12],highScore[11:8],highScore[7:4], highScore[3:0], display3[3:0], display2[3:0], display1[3:0], display0[3:0]);

//Instantiates an instance of BCD\_decoder.v

//This is the decoder which translates BCD values into binary values that light the correct segments in the 7-segment display

BCD\_decoder decoder\_INST1(display0[3:0], display1[3:0], display2[3:0], display3[3:0], HEX0[6:0], HEX1[6:0], HEX2[6:0], HEX3[6:0]);

//Instantiates an instance of BCD\_counter.v

//This code generates the values which generate numbers 1-9 for the seven-segment displays

BCD\_counter BCD\_Count\_INST1(divided\_Clock, RESET, CLEAR, counter\_Enable, BCD3, BCD2, BCD1, BCD0);

//Instantiates an instance of Clock\_divider

//This clock divides the native 50MHz clock by 25,000 to yield a 2kHz clock

clock\_Divider clk\_Div\_INST1(MAX10\_CLK1\_50, divided\_Clock);

//Instantiates an instance of Clock\_divider

//This clock divides the native 50MHz clock by 250,000 to yield a 200Hz clock

clock\_Divider #(250000) Slow\_Clock\_INST1(MAX10\_CLK1\_50, Slow\_Clock);

//clock\_Divider #(10000000) Slow\_Clock\_INST1(MAX10\_CLK1\_50, Slow\_Clock);

//Instantiates an instance of LFSR

//Generates pseudo random numbers to set the delay between pressing the start button, and the start of the timer/start indicator light

LFSR LFSR\_INST1(Slow\_Clock, LFSR\_Value);

//instantiates an instance of down\_Counter

//This counts from 0 to the random number generated by LSFR. When count==LSFR\_Value, downCount\_Complete=1

down\_Counter down\_Counter\_INST1(Slow\_Clock, LFSR\_Value, RESET, downCounter\_Enable, count, downCount\_Complete);

endmodule

//Gabriel Altman

//ECEN2350 Digital Logic

//April, 2018

//Based on code from http://www.fpga4fun.com/Counters3.html

module LFSR(CLK1, LFSROutput);

//input KEY0;

input CLK1;

//output [7:0] LFSROutput;

output reg [7:0] LFSROutput = 8'b11111111;

wire FEEDBACK = LFSROutput[7];

always @(posedge CLK1)

begin

//LFSROutput[0] <= 1 //Seed the shift register

LFSROutput[0] <= FEEDBACK;

LFSROutput[1] <= LFSROutput[0];

LFSROutput[2] <= LFSROutput[1];

LFSROutput[3] <= LFSROutput[2] ^ FEEDBACK;

LFSROutput[4] <= LFSROutput[3] ^ FEEDBACK;

LFSROutput[5] <= LFSROutput[4] ^ FEEDBACK;

LFSROutput[6] <= LFSROutput[5] ^ FEEDBACK;

LFSROutput[7] <= LFSROutput[6];

end

endmodule

//Gabriel Altman

//ECEN2350 Digital Logic

//April, 2018

module down\_Counter(clock, random\_Number, clear, ENABLE, count, downCount\_Complete);

input clock;

input clear;

input ENABLE;

input [7:0]random\_Number;

output reg [7:0]count;

output reg [7:0]downCount\_Complete;

//reg current\_count[7:0] = random\_Number[7:0];

always@(posedge clock)

begin

if (clear)

begin

downCount\_Complete <= 1'b0;

count <= 8'b00000000;

end

else if(ENABLE)

count <= count + 1;

//current\_count <= current\_count - 1;

//count = current\_count;

if (count == random\_Number)

begin

downCount\_Complete <= 1'b1;

//count <= count - 1;

//count = random\_Number;

//ENABLE <= 0;

end

else

downCount\_Complete <= 1'b0;

end

endmodule

//Gabriel Altman

//ECEN2350 Digital Logic

//April, 2018

module display\_MUX(displayState, BCD3, BCD2, BCD1, BCD0, HS3, HS2, HS1, HS0, Display3, Display2, Display1, Display0);

input displayState;

input [3:0]BCD0;

input [3:0]BCD1;

input [3:0]BCD2;

input [3:0]BCD3;

input [3:0]HS0;

input [3:0]HS1;

input [3:0]HS2;

input [3:0]HS3;

output reg[6:0]Display0;

output reg[6:0]Display1;

output reg[6:0]Display2;

output reg[6:0]Display3;

always@(displayState)begin

if(displayState == 1'b0)begin

Display3 = BCD3;

Display2 = BCD2;

Display1 = BCD1;

Display0 = BCD0;

end

else if(displayState == 1'b1)begin

Display3 = HS3;

Display2 = HS2;

Display1 = HS1;

Display0 = HS0;

end

else begin

Display3 = BCD3;

Display2 = BCD2;

Display1 = BCD1;

Display0 = BCD0;

end

end

endmodule

//Gabriel Altman

//ECEN2350 Digital Logic

//April, 2018

module clock\_Divider(clock, divided\_Clock, clock\_Count );

parameter q = 25000;

input clock;

//output divided\_Clock;

output reg divided\_Clock;

output reg [25:0]clock\_Count;

//reg div;

//reg [25:0]clock\_Count;

//wire [25:0]clock\_Count;

always@(posedge clock)

begin

clock\_Count <= clock\_Count + 1;

if (clock\_Count == q)

begin

clock\_Count <= 0;

//div <= !div;

divided\_Clock <= !divided\_Clock;

end

end

//assign divided\_Clock = div;

endmodule

//Gabriel Altman

//ECEN2350 Digital Logic

//April, 2018

module BCD\_decoder(BCDIn0, BCDIn1, BCDIn2, BCDIn3, HEX0, HEX1, HEX2, HEX3);

input [3:0]BCDIn0;

input [3:0]BCDIn1;

input [3:0]BCDIn2;

input [3:0]BCDIn3;

output reg[6:0]HEX0;

output reg[6:0]HEX1;

output reg[6:0]HEX2;

output reg[6:0]HEX3;

always @(BCDIn0, BCDIn1, BCDIn2, BCDIn3)

begin

case (BCDIn0)

4'b0000: HEX0 = 7'b1000000; //0

4'b0001: HEX0 = 7'b1111001; //1

4'b0010: HEX0 = 7'b0100100; //2

4'b0011: HEX0 = 7'b0110000; //3

4'b0100: HEX0 = 7'b0011001; //4

4'b0101: HEX0 = 7'b0010010; //5

4'b0110: HEX0 = 7'b0000010; //6

4'b0111: HEX0 = 7'b1111000; //7

4'b1000: HEX0 = 7'b0000000; //8

4'b1001: HEX0 = 7'b0010000; //9

endcase

case (BCDIn1)

4'b0000: HEX1 = 7'b1000000; //0

4'b0001: HEX1 = 7'b1111001; //1

4'b0010: HEX1 = 7'b0100100; //2

4'b0011: HEX1 = 7'b0110000; //3

4'b0100: HEX1 = 7'b0011001; //4

4'b0101: HEX1 = 7'b0010010; //5

4'b0110: HEX1 = 7'b0000010; //6

4'b0111: HEX1 = 7'b1111000; //7

4'b1000: HEX1 = 7'b0000000; //8

4'b1001: HEX1 = 7'b0010000; //9

endcase

case (BCDIn2)

4'b0000: HEX2 = 7'b1000000; //0

4'b0001: HEX2 = 7'b1111001; //1

4'b0010: HEX2 = 7'b0100100; //2

4'b0011: HEX2 = 7'b0110000; //3

4'b0100: HEX2 = 7'b0011001; //4

4'b0101: HEX2 = 7'b0010010; //5

4'b0110: HEX2 = 7'b0000010; //6

4'b0111: HEX2 = 7'b1111000; //7

4'b1000: HEX2 = 7'b0000000; //8

4'b1001: HEX2 = 7'b0010000; //9

endcase

case (BCDIn3)

4'b0000: HEX3 = 7'b1000000; //0

4'b0001: HEX3 = 7'b1111001; //1

4'b0010: HEX3 = 7'b0100100; //2

4'b0011: HEX3 = 7'b0110000; //3

4'b0100: HEX3 = 7'b0011001; //4

4'b0101: HEX3 = 7'b0010010; //5

4'b0110: HEX3 = 7'b0000010; //6

4'b0111: HEX3 = 7'b1111000; //7

4'b1000: HEX3 = 7'b0000000; //8

4'b1001: HEX3 = 7'b0010000; //9

endcase

end

endmodule

//Gabriel Altman

//ECEN2350 Digital Logic

//April, 2018

//Based on code from the textbook

module BCD\_counter (Clock, Reset, Clear, ENABLE, BCD3, BCD2, BCD1, BCD0);

input Clock, Reset, Clear, ENABLE;

output reg [3:0] BCD3, BCD2, BCD1, BCD0;

always @(posedge Clock)

begin

if (Clear || Reset)

begin

BCD3 <= 0;

BCD2 <= 0;

BCD1 <= 0;

BCD0 <= 0;

end

else if (ENABLE)

if (BCD0 == 4'b1001)

begin

BCD0 <= 0;

if (BCD1 == 4'b1001)

begin

BCD1 <= 0;

if (BCD2 == 4'b1001)

begin

BCD2 <= 0;

if (BCD3 ==4'b1001)

begin

BCD3 <= 0;

end

else

BCD3 <= BCD3 +1;

end

else

BCD2 <= BCD2 +1;

end

else

BCD1 <= BCD1 + 1;

end

else

BCD0 <= BCD0 + 1;

end

endmodule