ECE272 Lab 6
Spring 2018

Final Project
Isak Foshay

May 14^{th}, 2018

1. **Introduction**

This lab aims to let students show their creativity and demonstrate what they have learned this term. I make a custom module that handles addition, subtraction, doubling, and halving. As well as only accepting one input per button press.

2. **Design**

	FPGA PIN	PULLMODE
buttonAdd	E3	UP
buttonDivTwo	B1	UP
buttonMinus	C1	UP
buttonMulTwo	F3	UP
buttonSetZero	D3	UP
reset_n	A4	DOWN
led[0]	G14	DOWN
led[1]	B16	DOWN
led[2]	D14	DOWN
led[3]	F14	DOWN
led[4]	D16	DOWN
led[5]	C15	DOWN
led[6]	E16	DOWN
state[0]	E7	DOWN
state[1]	E8	DOWN
state[2]	F9	DOWN

Table 1: Chosen Pins and Pull Modes

My LED pins and state pins are the same as my last labs in order to make things easier when rewiring between labs. For the button pins I just set them to pins near each other. The button pins were arbitrary yet near one another.

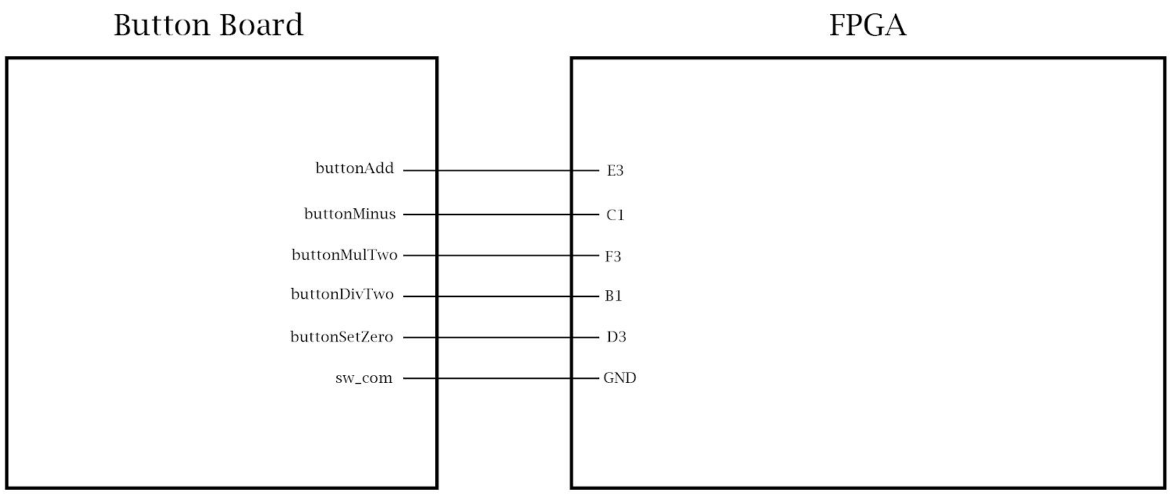

Figure 1a: Block Diagram for Hardware (Button Board)

I tried to align the buttons in a way that made sense. Add then subtract, multiply then divide. Then I put the reset at the end. It feels at least a bit familiar which is good enough when dealing with a row of button board buttons.

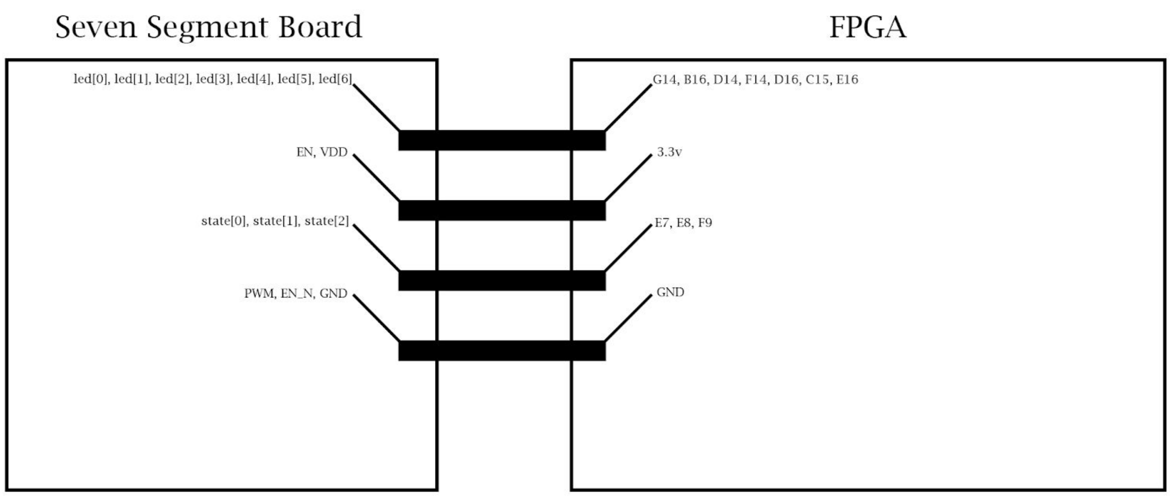

Figure 1b: Block Diagram for Hardware (Display)

This diagram shows the basic setup I used for the display for pretty much every lab. I kept things the same to make switching between labs easier.

3. Results

I thought my project was cute. It worked and was able to add, divide by two, multiply by two, subtract, and set to zero. It also implemented a switch like nature for the buttons so that it would only send one action per press.

4. Experiment Notes

This lab took a while because at first I wanted to make a mouse that auto clicked but when I took apart the mouse it showed that a mouse didn't work how I thought it worked. So I had to scrap my first idea so I went with this instead. I feel it demonstrates a wide array of things we learned in lab.

<u>Study Questions:</u>

1. What was the toughest aspect of ECE 272? What should be changed or added to the ECE 272 manual to make this course better?
The data sheet goes nowhere, need further instructions on soldering / why we need to solder, more resources for system verilog.

2. What would you like to explore further about Lattice Diamond or Digital Logic Design?
There are lots of things I would like to do with lattice diamond, none of which should be detailed in a lab report. There are lots of bugs and glitches that make things confusing and frustrating.

3. What section of ECE 272 did you dislike the most? Why?
That I found none of this intuitive, therefore making everything take forever to do. My least favorite part would have to be the lack of system verilog resources and lack of the data sheet. Seriously if you click the link on the website it goes nowhere.

4. What was your favorite section of ECE 272? Why?
I feel like exercising my problem solving skills for dozens of hours for every lab made me a better problem solver. The constant headaches were a downside to this though. Believe it or not though, I bought an arduino set so that I could tinker more during the summer in order to get a better grasp on how things work.

Appendix

```
module clock_counter(
        input logic clk_i,              //often, "tags" are added to variables to denote what they do for the user
        input logic reset_n,        //here, 'i' is used for input and 'o' for the output, while 'n' specifies
                                                        //an active low signal ("not")
        output logic clk_o
                );

                logic [13:0] count;         //register stores the counter value so that it can be modified
                                                //on a clock edge. Register size needs to store as large of a
                                                //number as the counter reaches. Here, 2^(13+1) = 16,384.

                always_ff @ (posedge clk_i, negedge reset_n)
                        begin
                                count <= count + 1;             //at every positive edge, the counter is increased by 1
                                if(!reset_n) //If reset_n gets pulled to ground (active low), reset count to 0
                                        begin
                                                clk_o <= 0;
                                                count <= 0;
                                        end
                                else
                                        if(count >= 5000) //Flips the slow clock every 10000 clock cycles
                                                begin
                                                        clk_o <= ~clk_o;            //Flip slow clock
                                                        count <= 0;                         //Reset the counter
                                                end
                        end
endmodule
```

```
module sevenseg(
        input logic [3:0] data,
        output logic [6:0] segments );
        always @(*)
                case( data ) // 7'bABCDEFG
                0: segments = 7'b1000000;
                1: segments = 7'b1111001;
                2: segments = 7'b0100100;
                3: segments = 7'b0110000;
                4: segments = 7'b0011001;
                5: segments = 7'b0010010;
                6: segments = 7'b0000010;
                7: segments = 7'b1111000;
                8: segments = 7'b0000000;
                9: segments = 7'b0011000;
                default:segments = 7'b1111111;
        endcase
endmodule
```

```
module LED_top_module(

        input logic reset_n, //be sure to set this input to PullUp, or connect the pin to 3.3V
        input logic buttonAdd,
        input logic buttonMinus,
        input logic buttonMultTwo,
        input logic buttonDivTwo,
        input logic buttonSetZero,
        output logic [2:0] state,
        output logic [6:0] led
        );

                logic clk;                      //used for the oscillator's 2.08 MHz clock
                logic clk_slow;                 //used for slowed down, 5 Hz clock
                logic [12:0] count = 0;
                logic [3:0] digitA; // ten thousand
                logic [3:0] digitB; // thousand
                logic [3:0] digitC; // ten
                logic [3:0] digitD; // one
                logic [3:0] muxtodec;


                counter counter (
                .buttonAdd(buttonAdd),
                .buttonMinus(buttonMinus),
                .buttonMultTwo(buttonMultTwo),
```

```
			.buttonDivTwo(buttonDivTwo),
			.buttonSetZero(buttonSetZero),
			.clk(clk_slow),
			.newCount(count)
			);


			parser parse (
			.val(count),
			.tenThous(digitA),
			.thous(digitB),
			.ten(digitC),
			.one(digitD)
			);

			mux2 mix (
			.a(digitA),
			.b(digitB),
			.c(digitC),
			.d(digitD),
			.s(state),
			.y(muxtodec));

			sevenseg decodr (
			.data(muxtodec),
			.segments(led)
			);

			//This is an instance of a special, built in module that accesses our chip's oscillator
			OSCH #("2.08") osc_int (	//"2.08" specifies the operating frequency, 2.08 MHz.
										//Other clock frequencies can be found in the MachX02's
documentation
				.STDBY(1'b0),				//Specifies active state
				.OSC(clk),				//Outputs clock signal to 'clk' net
				.SEDSTDBY());				//Leaves SEDSTDBY pin unconnected


			//This module is instantiated from another file, 'Clock_Counter.sv'
			//It will take an input clock, slow it down based on parameters set inside of the module, and
			//output the new clock. Reset functionality is also built-in
			clock_counter counter_1(
				.clk_i(clk),
				.reset_n(reset_n),
				.clk_o(clk_slow));

			//This module is instantiated from another file, 'State_Machine.sv'
			//It contains a Moore state machine that will take a clock and reset, and output a state
			state_machine FSM_1(
				.clk_i(clk_slow),
				.reset_n(reset_n),
				.state(state)
				);



endmodule
```

```
module mux2 (			input [3:0] a,
		input [3:0] b,
		input [3:0] c,
		input [3:0] d,
		input [2:0] s,
		output logic [3:0] y);

  always @ (a or b or c or d or s)
    case (s)
      3'b100 : y = a;
      3'b011 : y = b;
      3'b001 : y = c;
			3'b000 : y = d;
			default:y = a;
		endcase
endmodule
```

```
module parser (
			input logic [12:0] val,
			output logic [3:0] tenThous,
			output logic [3:0] thous,
			output logic [3:0] ten,
			output logic [3:0] one);
```

```
	//assign a = (inp - (inp%1000))/1000;
	//assign b = ((inp%1000)-(inp%100))/100;
	//assign c = ((inp%100)-(inp%10))/10;
	//assign d = (inp%10);
	assign tenThous = (val/1000)%10;
	assign thous = (val/100)%10;
	assign ten = (val/10)%10;
	assign one = (val % 10);
endmodule
```

```
module state_machine( //example of a Moore type state machine
	input logic clk_i,
	input logic reset_n,

	output logic [2:0] state //The state outputted by this state machine
		);

		//next state register
		logic [2:0] state_n;

		//each possible value of the state register is given a unique name for easier use later
		parameter S0 = 3'b000; //First digit
		parameter S1 = 3'b001; //Second digit
		parameter S2 = 3'b011; //Third digit
		parameter S3 = 3'b100; //Fourth digit

		//asynchronous reset will set the state to the start, S0, otherwise, the state is changed
		//on the positive edge of the clock signal
		always_ff @ (posedge clk_i, negedge reset_n)
			begin
				if(!reset_n)
					state = S0;
				else
					state = state_n;

			end

		//this section defines what the next state should be for each possible state. in this
		//implementation, it simply rotates through each state automatically
		always_ff @ (*)
			begin
				case(state)
					S0: state_n = S1;
					S1: state_n = S2;
					S2: state_n = S3;
					S3:	state_n = S0;

					default: state_n = S0;
				endcase
			end
endmodule
```

```
module counter (
					input logic buttonAdd,
					input logic buttonMinus,
					input logic buttonMultTwo,
					input logic buttonDivTwo,
					input logic buttonSetZero,
					input logic clk,
					output logic [12:0] newCount
				);
			logic flipAdd = 0;
			logic flipMinus = 0;
			logic flipMult = 0;
			logic flipDiv = 0;
			logic flipZero = 0;
	always @(posedge clk)
		begin
				if ((buttonAdd==0) && (flipAdd==0))
					begin
					newCount <= newCount + 1;
					flipAdd = 1;
					end
				else if ((buttonAdd==1) && (flipAdd ==1))
					begin
						flipAdd = 0;
					end
				else if ((buttonMinus==0) && (flipMinus==0))
					begin
					newCount <= newCount - 1;
					flipMinus = 1;
```

```
			end
		else if ((buttonMinus==1) && (flipMinus ==1))
			begin
				flipMinus = 0;
			end
		else if ((buttonMultTwo==0) && (flipMult==0))
			begin
			newCount <= newCount * 2;
			flipMult = 1;
			end
		else if ((buttonMultTwo==1) && (flipMult ==1))
			begin
				flipMult = 0;
			end
		else if ((buttonDivTwo==0) && (flipDiv==0))
			begin
			newCount <= newCount / 2;
			flipDiv = 1;
			end
		else if ((buttonDivTwo==1) && (flipDiv ==1))
			begin
				flipDiv = 0;
			end
		else if ((buttonSetZero==0) && (flipZero==0))
			begin
			newCount <= 0;
			flipZero = 1;
			end
		else if ((buttonSetZero==1) && (flipZero ==1))
			begin
				flipZero = 0;
			end

	end

endmodule
```