ECE 272 LAB 5
Spring 2018

Basic combinational logic and the MachX03
Jonathan Carney

May 24th, 2018
Grading TA: Lyubo Gankov

Introduction

In this lab our task was to design and implement a voltmeter onto our FPGA. We were given ADC chips as well as SV files for a clock state machine and an ADC module to filter the input from our ADC chip. We then designed a system to take these values and convert them into an output to a seven segment LED display.

Design

The circuit was designed entirely in system verilog. In addition to the given SV files, the ones I created and modified are as follows.

Top module

```
module LED_top_module(
    /*************************/
    /* Set inputs and outputs */
    /* to the whole FPGA here */
    /*************************/
    input logic reset_n, //be sure to set this input to PullUp, or connect the pin to 3.3V
    input logic miso_i, //for adc chip

    output logic decimal,
    output logic mosi_o,//for adc chip
    output logic sck_o,
    output logic reset_o,
    output logic [6:0] display,//seven seg display
    output logic [2:0] state//seven seg display digit select
```

```
);
        /*******************************/
        /* Set internal variables here */
        /*******************************/
        logic clk;                  //used for the oscillator's 2.08 MHz clock
        logic clk_slow; //used for slowed down, 5 Hz clock
        logic [3:0]one;//ones place
        logic [3:0]ten;
        logic [3:0]hun;
        logic [3:0]tho;
        logic [3:0]num;
        logic [15:0]databus;//data from adc to parser

        /***********************/
        /* Define modules here */
        /***********************/
        //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));

    adc adc_1(
        .reset(reset_n),//inputs
        .clk_2Mhz(clk),
        .MISO(miso_i),
        .raw_data(databus),//outputs
        .MOSI(mosi_o),
        .adc_reset(reset_o),
        .SCK(sck_o));

    Lab5Parser parser_1(//yet to be re written for lab 5
        .data_i(databus),
        .one_o(one),
        .ten_o(ten),
        .hun_o(hun),
        .tho_o(tho));

    MUXL4 mux_1(
        .one_i(one),
        .ten_i(ten),
        .hun_i(hun),
        .tho_i(tho),
        .sel_i(state),
        .num_o(num),
        .dec(decimal));

    DecoderL4 decoder_1(
        .ss_o(display),
        .num_i(num));
    /***********************************************/
    /* Add modules for:                                                */
    /* Parser              Determines the 1000's, 100's,   */
    /*                             10's and 1's place of the number*/
    /* Multiplexer   Determines which parser output  */
    /*                             to pass to the decoder                  */
    /* Decoder             Convert 4-bit binary to 7-seg   */
    /*                             output for numbers 0-9                  */
    /***********************************************/
endmodule
```

Parser

```
module Lab5Parser(//take input from button board and parse into outputs for each digit of value
```

```
	input logic [15:0] data_i,//input from adc to be parsed
	output logic [3:0] one_o,//output for ones place
	output logic [3:0] ten_o,
	output logic [3:0] hun_o,
	output logic [3:0] tho_o
		);
		logic [20:0]c;
			assign c = (data_i*5000/65535);
			assign one_o = c%10;//isolate the ones place
			assign ten_o = (c/10)%10;
			assign hun_o = (c/100)%10;
			assign tho_o = (c/1000)%10;

endmodule
```

Multiplexor

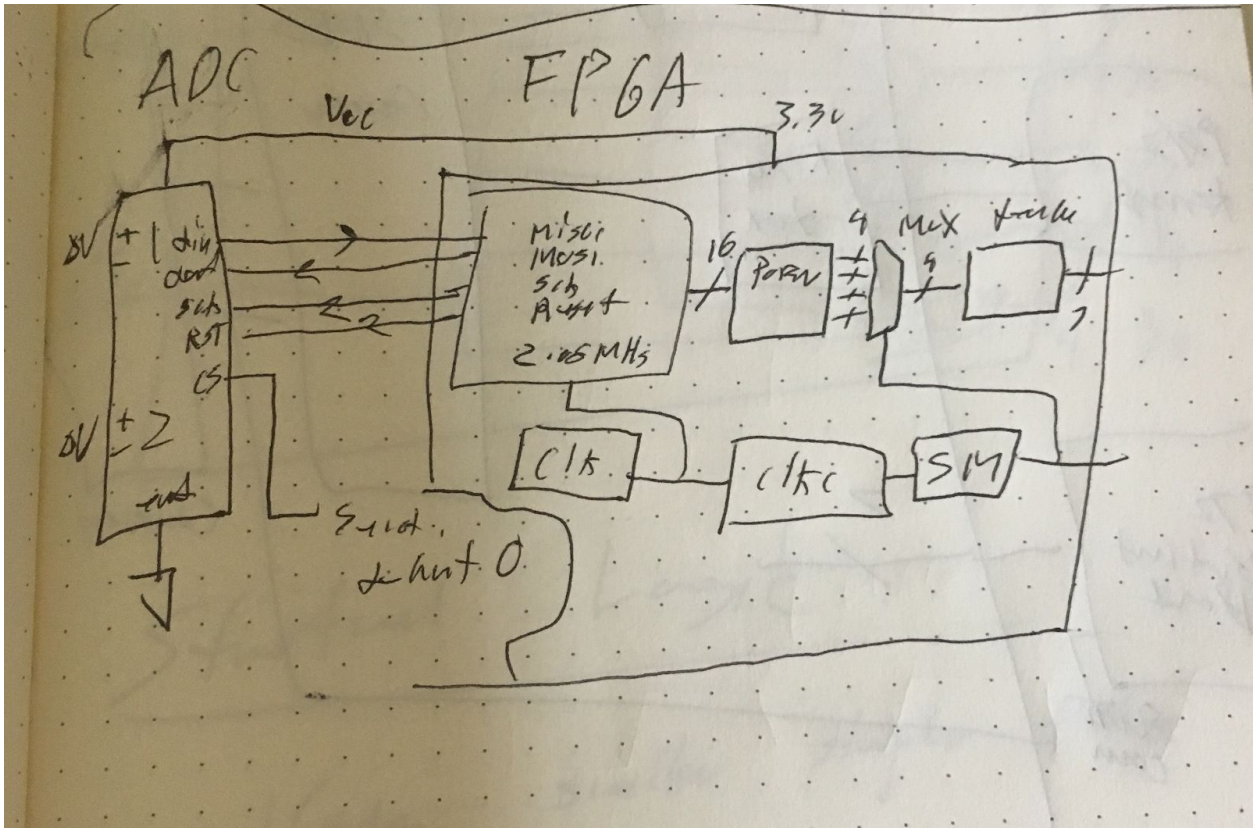
```
module MUXL4(
	input logic [3:0] one_i,//input for ones place
	input logic [3:0] ten_i,
	input logic [3:0] hun_i,
	input logic [3:0] tho_i,
	input logic [2:0] sel_i,

	output logic dec;
	output logic [3:0] num_o
		);
			always_comb
				case(sel_i)//sel for the mux
					0:	num_o = one_i;//set output to 1's place for case 0
					1:	num_o = ten_i;
					3:	num_o = hun_i;
					4:	num_o = tho_i;
					default:	num_o = 4'b0000;
			endcase
			always_comb
				case(sel_i)
					4:	dec = 0;
					default dec = 1;
			endcase
endmodule
```

Decoder

```
module DecoderL4(
```

```
        input logic [3:0] num_i,//input from mux
        output logic [6:0] ss_o//output to seven segment display
                );
                        always_comb//determining binary output based upon input cases
                                case(num_i)
                                        0:      ss_o = 7'b0000001;//set segments to display 0
                                        1:      ss_o = 7'b1001111;
                                        2:      ss_o = 7'b0010010;
                                        3:      ss_o = 7'b0000110;
                                        4:      ss_o = 7'b1001100;
                                        5:      ss_o = 7'b0100100;
                                        6:      ss_o = 7'b0100000;
                                        7:      ss_o = 7'b0001111;
                                        8:      ss_o = 7'b0000000;
                                        9:      ss_o = 7'b0001100;
                                        default: ss_o = 7'b1111111;
                        endcase
endmodule
```

Photo of block diagram describing this Lab



Results

After sufficient error checking the circuit behaved as it was intended to. However the FPGA I initially utilized lost power to it's pins and had to be replaced with another FPGA, additionally, I had to use a second ADC as the first one was non functional. I got to practice using an Oscilloscope to diagnose these problems during the implementation phase of this lab.

Lab notes

This lab was very hard for me. The design was easy, however the implementation was extremely difficult due to multiple hardware failures. It was cool to use an O-scope, but it came after two hours of frustration trying to figure out what was going wrong.

Study Questions

1. The ADC code takes a varying signal that does not conform to binary expression, and approximates it into a value that can be computed by the circuit. This value is then sent to the parser in the form of a 16 bit binary number.