Name: Bryan Gonzalez ID# 1001443032
Date Submitted: 11/26/24 Lab Section # 004

CSE 2441 – Digital Logic Design I

Fall Semester 2024

Term Project – Eight-Bit, Two-Function Calculator

(100+ Points)

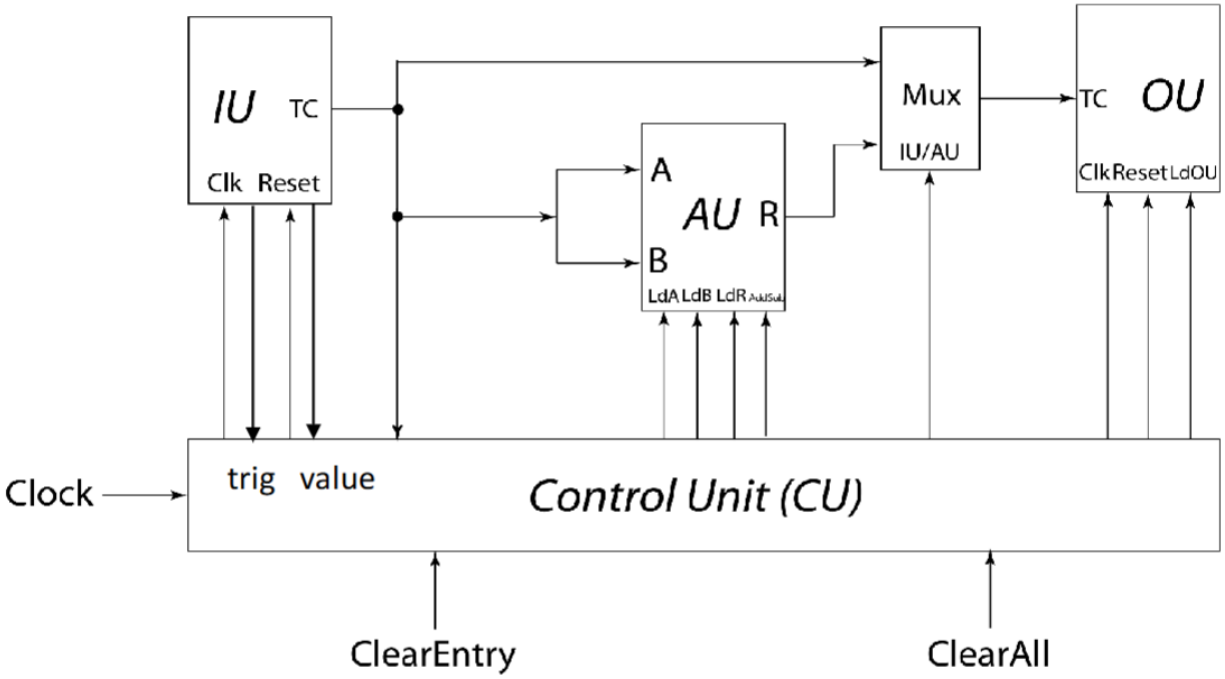
Project Demonstration Deadline – December 3, 2024, 6:00 pm

Project Report Deadline – December 3, 2024, 11:59 pm

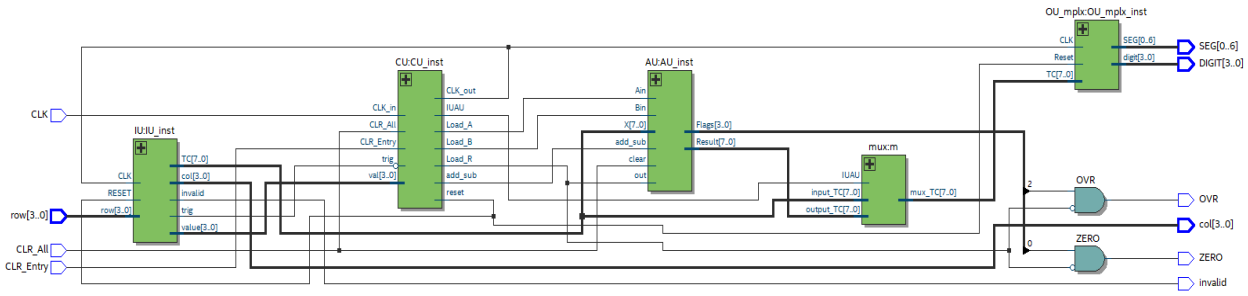
Early Submission Deadlines – November 26, 2024, 6:00 pm and 11:59 pm

This project is performed with the DE10-Lite and extension boards

Calculator top-level functional diagram



Calculator top-level functional diagram (Quartus RTL)



List of added features

- Multiplexed Output Unit
- Blank Leading Zeros

Multiplexed Output Unit and Blank Leading Zeros Verilog

```
module OU_mplx(
        input CLK, Reset,
        input [7:0] TC,
        output [3:0] digit,
        output [0:6] SEG
);

        wire [7:0] SM;
        wire [3:0] sign, HUNDREDS, TENS, ONES;
        reg HUNDREDS_Blank, TENS_Blank;

        wire [1:0] SEL;
        wire [3:0] val;

        assign sign[0] = 1;
        assign sign[1] = ~TC[7];
        assign sign[2] = 1;
        assign sign[3] = 1;

        twos2signed t2s_inst(.A(TC), .B(SM));

        binary2bcd b2b_inst(
                .A(SM),
                .ONES(ONES),
                .TENS(TENS),
                .HUNDREDS(HUNDREDS)
        );

        Controller cont_inst(
                .CLK(CLK),
                .Reset(Reset),
                .SEL(SEL),
                .CAT(digit)
        );

        four2one f2o(
                .SEL(SEL),
                .digit0(ONES),
                .digit1({TENS_Blank ^ TENS[3],
                        TENS_Blank ^ TENS[2],
                        TENS_Blank ^ TENS[1],
                        TENS_Blank ^ TENS[0]}),
                .digit2({HUNDREDS_Blank ^ HUNDREDS[3],
                        HUNDREDS_Blank ^HUNDREDS[2],
                        HUNDREDS_Blank ^ HUNDREDS[1],
                        HUNDREDS_Blank ^ HUNDREDS[0]}),
```

```
		.digit3(sign),
		.val(val)
	);

	//Checking for Leading zeroes
	always @ (TENS, HUNDREDS)
		if(HUNDREDS > 0)
			 begin HUNDREDS_Blank=0; TENS_Blank = 0; end
		else if(TENS > 0)
			begin HUNDREDS_Blank=1; TENS_Blank = 0; end
		else
			begin HUNDREDS_Blank=1; TENS_Blank = 1; end


	bcd2seven_high bcd2sev_inst(
		val[3],val[2], val[1], val[0],
		SEG[0],
		SEG[1],
		SEG[2],
		SEG[3],
		SEG[4],
		SEG[5],
		SEG[6]
	);

endmodule

module four2one(
	input [1:0] SEL,
	input [3:0] digit0,
	input [3:0] digit1,
	input [3:0] digit2,
	input [3:0] digit3,
	output reg [3:0] val
);
	always @ (SEL)
		case ({SEL})
			2'b00: val = digit0;
			2'b01: val = digit1;
			2'b10: val = digit2;
			2'b11: val = digit3;
		endcase
endmodule

module Controller(
	input CLK, Reset,
	output [1:0] SEL,
	output [3:0] CAT
```

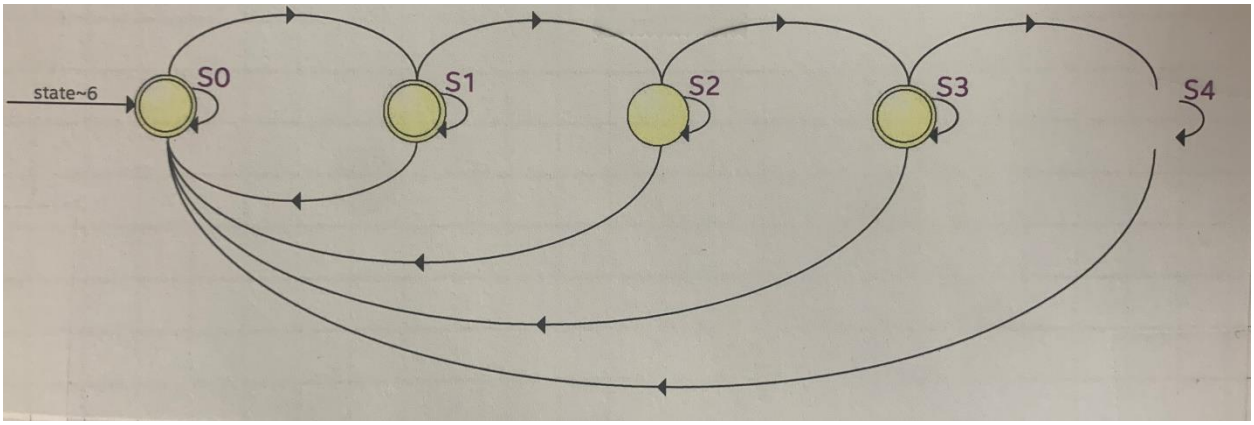
```
);
	wire slowclock;

	clock_div #(.DIV(100000)) keypad_clock_divider(
		.clk(CLK),
		.clk_out(slow_clock)
	);

	FSM fsm_inst(.slow_clock(slow_clock),.reset(Reset),.SEL(SEL),.CAT(CAT));
endmodule

module FSM(
	input slow_clock, reset,
	output reg [1:0] SEL,
	output reg [3:0] CAT
);
	reg [1:0] state, nextstate;
	always @ (negedge slow_clock, negedge reset)
		if (reset == 0) state <= 2'b0; else state <= nextstate;
	always @ (state)
		case ({state})
			2'b00: begin nextstate = 2'b01; SEL = 2'b00; CAT = 4'b1000; end
			2'b01: begin nextstate = 2'b10; SEL = 2'b01; CAT = 4'b0100; end
			2'b10: begin nextstate = 2'b11; SEL = 2'b10; CAT = 4'b0010; end
			2'b11: begin nextstate = 2'b00; SEL = 2'b11; CAT = 4'b0001; end
		endcase
endmodule
```

Top-level Verilog code

```
module Calculator(
        input CLK, CLR_Entry, CLR_All,
        input [3:0] row,
        output [3:0] col,
        output invalid,
        output OVR, ZERO,
        output [0:6] SEG,
        output [3:0] DIGIT
);

        wire CLK_IU;
        wire [3:0] val;
        wire [3:0] flags;
        wire [7:0] input_TC, output_TC, mux_TC;
        wire reset, trig, Load_A, Load_B, Load_R, add_sub, IUAU;

        CU CU_inst
        (
                .trig(~trig),                   // input  trig_sig
                .val(val),                      // input [3:0] val_sig
                .CLK_in(CLK),
                .CLK_out(CLK_IU),               // input  CLK_sig
                .CLR_Entry(CLR_Entry) ,         // input  CLR_Entry_sig
                .CLR_All(CLR_All) ,             // input  CLR_All_sig
                .reset(reset) ,                 // output  reset_sig
                .Load_A(Load_A) ,               // output  Load_A_sig
                .Load_B(Load_B) ,               // output  Load_B_sig
                .Load_R(Load_R) ,               // output  Load_R_sig
                .add_sub(add_sub) ,             // output  add_sub_sig
                .IUAU(IUAU)                     // output  IUAU_sig
        );


        IU IU_inst
        (
                .CLK(CLK_IU) ,                  // input  CLK_sig
                .RESET(reset) ,                 // input  RESET_sig
                .row(row) ,                     // input [3:0] row_sig
                .col(col) ,                     // output [3:0] col_sig
                .TC(input_TC) ,                 // output [7:0] TC_sig
                .invalid(invalid),              // output  invalid_sig
                .trig(trig),
                .value(val)
        );
```

```
	OU_mplx OU_mplx_inst
	(
		.CLK(CLK_IU) ,			// input  CLK_sig
		.Reset(reset) ,			// input  Reset_sig
		.TC(mux_TC) ,			// input [7:0] TC_sig
		.digit(DIGIT) ,			// output [3:0] digit_sig
		.SEG(SEG)			// output [0:6] SEG_sig
	);

	mux m
	(
		.IUAU(IUAU),
		.input_TC(input_TC),
		.output_TC(output_TC),
		.mux_TC(mux_TC)
	);

	AU AU_inst
	(
		.X(input_TC) ,			// input [7:0] x_sig
		.Ain(Load_A) ,			// input  ain_sig
		.Bin(Load_B) ,			// input  bin_sig
		.out(Load_R) ,			// input  out_sig
		.clear(CLR_All) ,		// input  clear_sig
		.add_sub(add_sub) ,		// input  add_sub_sig
		.Result(output_TC) ,		// output [7:0] result_sig
		.Flags(flags)			// output [3:0] flags_sig
		//flag[3] = CARRY OUT
		//flag[2] = OVERFLOW
		//flag[1] = NEGATIVE
		//flag[0] = ZERO
	);
	assign ZERO = flags[0] & ~Load_R;
	assign OVR  = flags[2] & ~Load_R;

endmodule

module mux(
	input IUAU,
	input [7:0] input_TC, output_TC,
	output reg [7:0] mux_TC
);

	always @ (IUAU)
		if(IUAU == 0)  mux_TC = input_TC;
		else			mux_TC = output_TC;
endmodule
```

Control Unit State diagram



Control Unit Verilog code

```
module CU(
        input trig,
        input [3:0] val,
        input CLK_in,
        input CLR_Entry, CLR_All,
        output CLK_out,
        output reg reset,
        output reg Load_A, Load_B, Load_R,
        output reg add_sub,
        output reg IUAU
);
        assign CLK_out = CLK_in;

        reg [2:0] state, nextstate;
        parameter S0 = 3'b000, S1 = 3'b001, S2 = 3'b010, S3 = 3'b011, S4=3'b100;

        always @ (negedge trig, negedge CLR_All)
        begin
                if (CLR_All == 0) state <= S0;
                //else if (CLR_Entry == 0) state <= S0;
                else state <= nextstate;

                case (state)
                        S0:     begin if (trig) nextstate <= S1; else nextstate <= S0; end
                        S1:     begin
                                                if (trig)
                                                        begin
                                                                if (val == 10) begin
                                                                        nextstate <= S2;
                                                                        add_sub = 0; end
                                                                else if(val == 11) begin
                                                                        nextstate <= S2;
                                                                        add_sub = 1; end
```

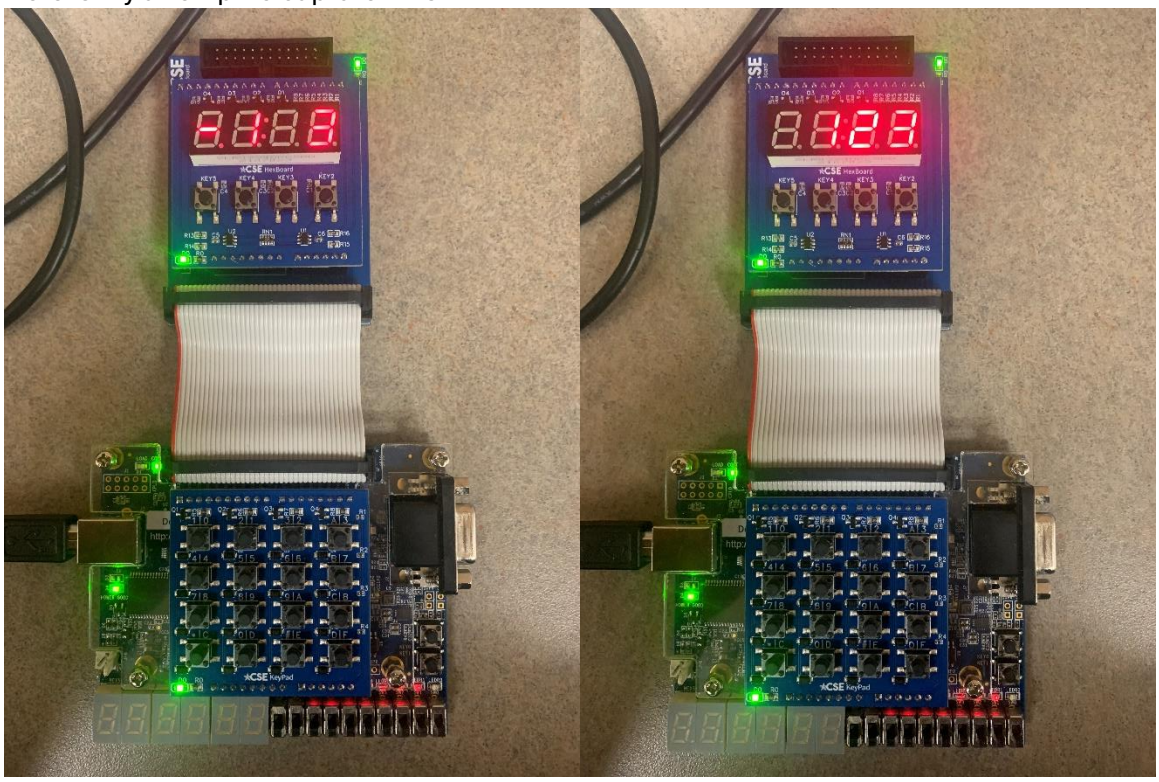
```
                                                    else nextstate <= S1;
                                                end
                                            else nextstate <= S1;
                                        end
                        S2:     begin if (trig)  nextstate <= S3; else nextstate <= S2; end
                        S3:     begin if (trig && val == 15) nextstate <= S4; else nextstate <=
S3; end
                        S4:     nextstate <= S4;
                endcase
        end

        always @ (state)
                case(state)
                        S0:     begin reset = 0   & CLR_All & CLR_Entry; Load_A = 1'b1;
                                Load_B = 1'b1; Load_R = 1'b1; IUAU = 0;    end
                        S1:     begin reset = 1'b1 & CLR_All & CLR_Entry; Load_A = 0;
                                Load_B = 1'b1; Load_R = 1'b1; IUAU = 0;    end
                        S2:     begin reset = trig & CLR_All & CLR_Entry; Load_A = 1'b1;
                                Load_B = 1'b1; Load_R = 1'b1; IUAU = 0;    end
                        S3:     begin reset = 1'b1 & CLR_All & CLR_Entry; Load_A = 1'b1;
                                Load_B = 0;    Load_R = 1'b1; IUAU = 0;    end
                        S4:     begin reset = 1'b1 & CLR_All & CLR_Entry; Load_A = 1'b1;
                                Load_B = 1'b1; Load_R = 0;    IUAU = 1'b1; end
                endcase
endmodule
```

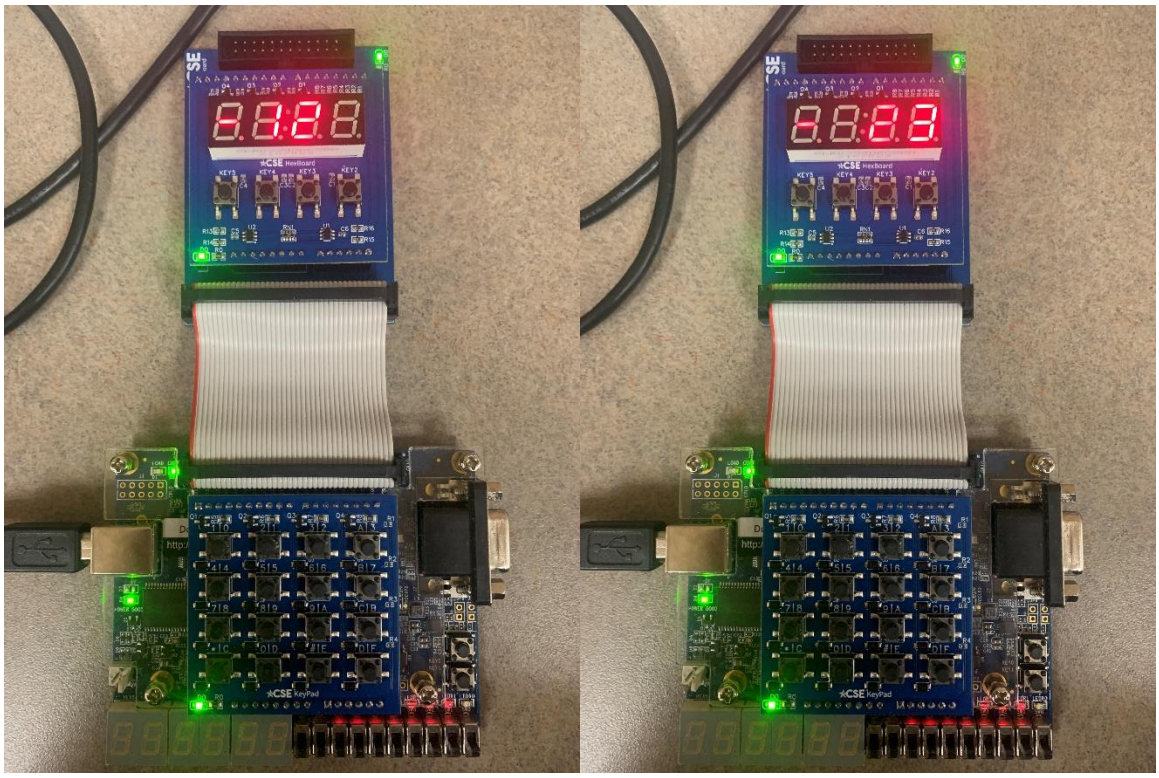
Test demo. Must include specified requirements and added features.

- Testing normal calcuation
    a. 34+75
    b. 34-75
    c. -34+75
    d. -34-74
    e. 127+9
    f. 127-9
    g. 9+9
    h. 9-9
    i. 86-125
    j. 86+125
- Testing mutiplexed display
- Testing blank leading zeroes
    a. Single digit
    b. Two digits
    c. Negative single digit
    d. Negative two digit

I was going to submit pictures to show my tests for the report but because of how the multiplexed display works, the camera on my phone can only capture two or three digits displaying at one time.

Here is my attempt to capture -123





It may be inconvinient but cool to see the how the multiplexed display is working. We get to see the FSM selecting cycling across all the digits, displaying “one at a time”. (Note our camera does capure the digit before and after the selected digit to display.)

The TA, Andrew did see the multiplex display working properly and blank leading zeroes.

Table of results

A	Op	B	Result	Flag
34	+	75	109	---
34	-	75	-41	---
-34	+	75	41	---
-34	-	75	-109	---
127	+	9	-120	OVR
127	-	9	118	---
9	+	9	18	---
9	-	9	0	ZERO
86	-	125	-39	---
86	+	125	-45	OVR

Description of unresolved problems if any

I don't feel like I have any unresolved problems. I am satisfied with my term project and got all the extra credit features to work.

Lessons learned

I came into the course knowing a little about digital logic already and now I feel like I really have a grasp of how it works and some possible applications of it in the real world such as a calculator. We I hadn't had any experience with was sequential circuits. I learned how sequential circuits can be used to create memory in the form of registers or to hold states. I feel like I can apply what I learned in Computer Theory like DFA's and I can apply that to designing secuential circuits. Also working with bits and different number systems, I feel a lot more comfortable working with binary, twos compliment, hexadecimal and conversions to decimal. Lastly, I learned about verilog and we can use to program an FGPA.