Lab 5

Alien Calculator

Group G

Chang, Philbert (013179257)

Hua, Russell (013184015)

Thai, Paul (014760252)

ECE 3300L.01

Professor Aly

07/17/2021

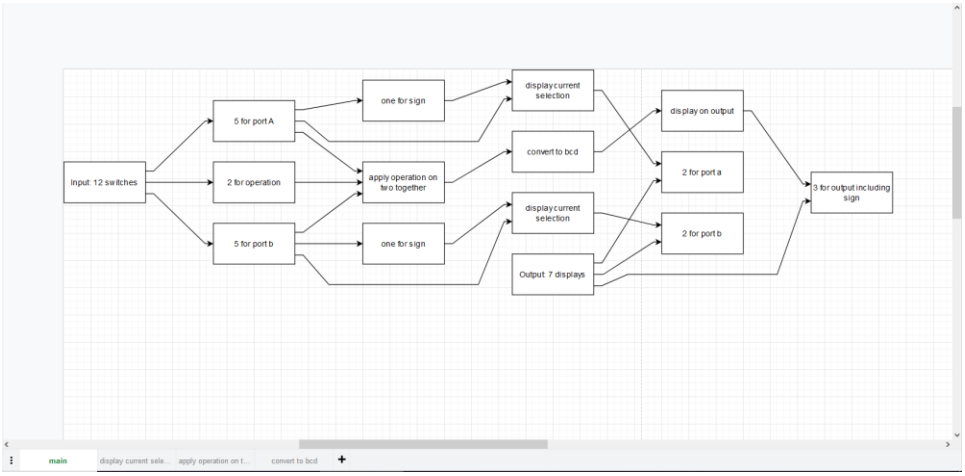
Problem

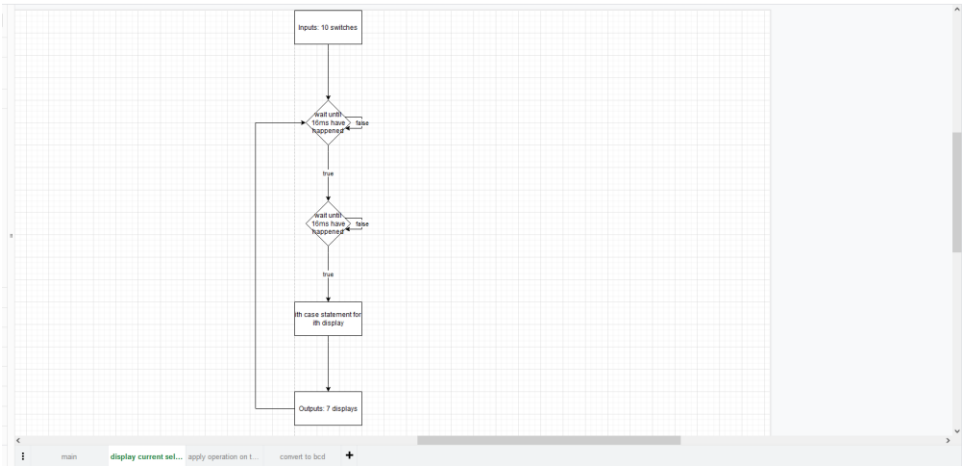
Create a calculator that adds, subtracts, or multiplies two signed values using two displays for the input values and three displays for the output values. Five switches are used for each signed value and two for selecting the desired operation.

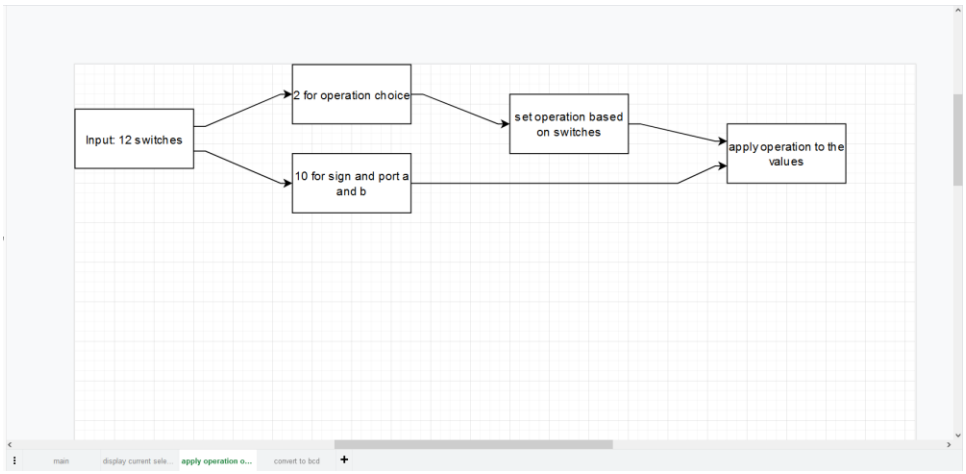
Code Detail

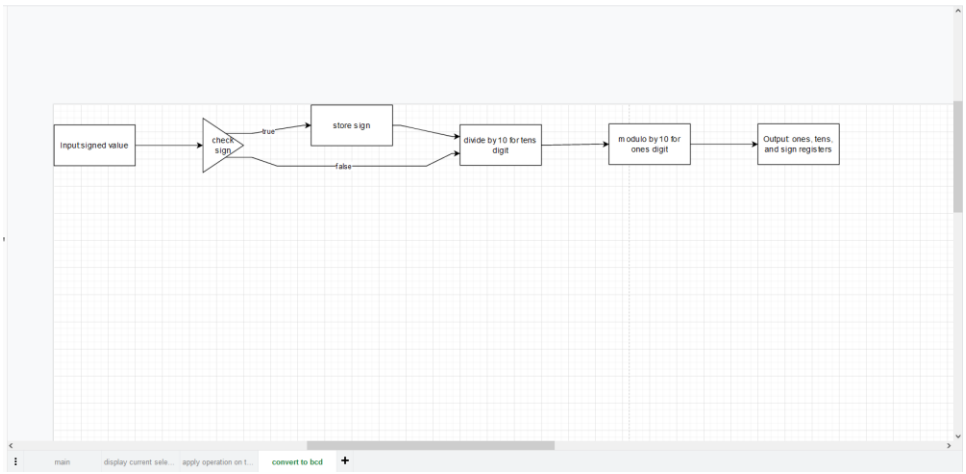
The main block of the code is written within an if statement that sequentially refreshes each seven-segment display. The input seven-segment displays are driven using a case statement that converts switch inputs to bcd and outputs it to the display. The calculations are handled outside of the refresh if statement, and the output is initialized to 0. In order to determine negative or positive output, an if statement is used that compares the output to 0.

Code Flowchart






Input: 12 switches
2 for operation choice
10 for sign and port a and b
set operation based on switches
apply operation to the values
main
display current sele...
apply operation o...
convert to bcd


Input signed value
check sign
true
false
store sign
divide by 10 for tens digit
modulo by 10 for ones digit
Output: ones, tens, and sign registers
main
display current sele...
apply operation on t...
convert to bcd

Results

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 07/15/2021 01:48:24 PM
// Design Name:
// Module Name: portA
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module portA #(parameter MAX_COUNT = (1666666)-1, MAX_INNER_COUNT = MAX_COUNT/32) (
            input [11:0] SW, // SW[3:0] is port a, SW[4] is port as, SW[8:5] is port b, SW[9] is port bs, swtich 10 and 11 are SW[14] and SW[15] repsectively (for arithemetic)
            input mclk,
            output reg [6:0] a_to_g,
            output reg [7:0] AN
            );
            reg [26:0] counter = 0;
            reg [26:0] innerCounter = 0;
            reg [2:0] displayFlag = 0;

            reg signed [4:0] temp1 = 0;
            reg signed [4:0] temp2 = 0;
            reg signed [4:0] sign1 = 0;
            reg signed [4:0] sign2 = 0;

            reg check=0;

            reg signed [8:0] out = 0;

            reg [3:0] hold_low = 0;
            reg [3:0] hold_high =0;

```

Figure 1: Verilog Code [P1]

```
    always@(posedge mclk)
    // counts up to the start of refresh period of MAX_COUNT
        if (counter == MAX_COUNT) begin
            // true, start refreshing the displays, one by one with a delay of MAX_INNER_COUNT
            if (innerCounter == MAX_INNER_COUNT) begin
            /*
                if the left most display is only on, then the case statement
                is ending with the last case before the refresh so the inner
                counter should be reset to give time for the other cases
            */


                if (displayFlag < 7) begin
                    case (displayFlag)
                    0: begin
                        AN = 8'b11111110;
                        case(SW[3:0])
                            4'd0:begin a_to_g = 7'b0000001;
                                       templ = 4'd0; end
                            4'd1:begin a_to_g = 7'b1001111;
                                       templ = 4'd1; end
                            4'd2:begin a_to_g = 7'b0010010;
                                       templ = 4'd2; end
                            4'd3:begin a_to_g = 7'b0000110;
                                       templ = 4'd3; end
                            4'd4:begin a_to_g = 7'b1001100;
                                       templ = 4'd4; end
                            4'd5:begin a_to_g = 7'b0100100;
                                       templ = 4'd5; end
                            4'd6:begin a_to_g = 7'b0100000;
                                       templ = 4'd6; end
                            4'd7:begin a_to_g = 7'b0001111;
                                       templ = 4'd7; end
                            4'd8:begin a_to_g = 7'b0000000;
                                       templ = 4'd8; end
                            4'd9:begin a_to_g = 7'b0000100;
                                       templ = 4'd9; end
                            default: a_to_g = 7'b0000100;
                        endcase
                        end
                    1: begin
                        AN = 8'b11111101;
                        case(SW[4])
                            0:begin a_to_g = 7'b1001110;
                                    signl <= -1;
                                    end
                            1:begin a_to_g = 7'b0110010;
                                    signl <= 1;
                                    end
                        endcase
                        end
```

Figure 2: Verilog Code [P2]

```
97                  2: begin
98                      AN = 8'b11111011;
99                      case(SW[8:5])
100                         4'd0:begin a_to_g = 7'b0000001;
101                                    temp2 = 4'd0; end
102                         4'd1:begin a_to_g = 7'b1001111;
103                                    temp2 = 4'd1; end
104                         4'd2:begin a_to_g = 7'b0010010;
105                                    temp2 = 4'd2; end
106                         4'd3:begin a_to_g = 7'b0000110;
107                                    temp2 = 4'd3; end
108                         4'd4:begin a_to_g = 7'b1001100;
109                                    temp2 = 4'd4; end
110                         4'd5:begin a_to_g = 7'b0100100;
111                                    temp2 = 4'd5; end
112                         4'd6:begin a_to_g = 7'b0100000;
113                                    temp2 = 4'd6; end
114                         4'd7:begin a_to_g = 7'b0001111;
115                                    temp2 = 4'd7; end
116                         4'd8:begin a_to_g = 7'b0000000;
117                                    temp2 = 4'd8; end
118                         4'd9:begin a_to_g = 7'b0000100;
119                                    temp2 = 4'd9; end
120                         default: a_to_g = 7'b0000100;
121                     endcase
122                     end
123                 3: begin
124                     AN = 8'b11110111;
125                     case(SW[9])
126                         0:begin a_to_g = 7'b1001110;
127                                 sign2 <= -1;
128                                 end
129                         1:begin a_to_g = 7'b0110010;
130                                 sign2 <= 1;
131                                 end
132                     endcase
133                     end
134                 4: begin
135                     AN = 8'b11011111;
136                     case(hold_low)
137                         4'd0: a_to_g = 7'b0000001;
138                         4'd1: a_to_g = 7'b1001111;
139                         4'd2: a_to_g = 7'b0010010;
140                         4'd3: a_to_g = 7'b0000110;
141                         4'd4: a_to_g = 7'b1001100;
142                         4'd5: a_to_g = 7'b0100100;
143                         4'd6: a_to_g = 7'b0100000;
144                         4'd7: a_to_g = 7'b0001111;
145                         4'd8: a_to_g = 7'b0000000;
146                         4'd9: a_to_g = 7'b0000100;
147                         default: a_to_g = 7'b1111110;
```

Figure 3: Verilog Code [P3]

```
                        endcase
                        end
                    5: begin
                        AN = 8'b10111111;
                        case(hold_high)
                            4'd0: a_to_g = 7'b0000001;
                            4'd1: a_to_g = 7'b1001111;
                            4'd2: a_to_g = 7'b0010010;
                            4'd3: a_to_g = 7'b0000110;
                            4'd4: a_to_g = 7'b1001100;
                            4'd5: a_to_g = 7'b0100100;
                            4'd6: a_to_g = 7'b0100000;
                            4'd7: a_to_g = 7'b0001111;
                            4'd8: a_to_g = 7'b0000000;
                            4'd9: a_to_g = 7'b0000100;
                            default: a_to_g = 7'b1111110;
                        endcase
                        end
                    6: begin
                            AN = 8'b01111111;
                            case (check)
                                0: a_to_g = 7'b0110010;
                                1: a_to_g = 7'b1001110;
                            endcase

                            end
                        //end

                    endcase

                    displayFlag = displayFlag + 1;
                    innerCounter = 0;
                end

                else begin //if greater than 4, reset everything
                    displayFlag = 0;
                    innerCounter = 0;
                    counter = 0;
                    displayFlag = 0;
                    AN = 8'b11111111;
                end

            //Calculating

            if (SW[11]==0 & SW[10]==0) begin        //addition
                out = (temp1*sign1) + (temp2*sign2);
            end
            else if (SW[11]==0 & SW[10]==1) begin   //subtration
                out = (temp1*sign1) - (temp2*sign2);
            end
            else if (SW[11]==1 & SW[10]==0) begin   //multiplication
```

Figure 4: Verilog Code [P4]

```
198             else if (SW[11]==1 & SW[10]==0) begin   //multiplication
199                 out = (temp1*sign1) * (temp2*sign2);
200             end
201
202
203             if (out <= 0) begin
204                 // set to positive and set check (sign)
205                 check = 1;
206                 hold_high = (out*-1) / 10;                  //set the high and lower bits
207                 hold_low = (out*-1) % 10;
208             end
209             else if(out >= 0) begin
210                 check = 0;
211                 hold_high = out / 10;                   //set the high and lower bits
212                 hold_low = out % 10;
213             end
214
215
216
217
218         end
219             //if not max inner_counter, increment
220                 else begin
221                     innerCounter = innerCounter + 1;
222                 end
223         end //end of inner counter
224             //if not max counter, increment
225         else begin
226                 counter = counter + 1;
227     end//end of counter
228
229
230
231 endmodule
232
```

Figure 5: Verilog Code [P5]

Hierarchy

Name	Slice LUTs (63400)	Slice Registers (126800)	Slice (15850)	LUT as Logic (63400)	Bonded IOB (210)	BUFGCTRL (32)
portA	1063	103	330	1063	28	1

Figure 6: Report Utilization

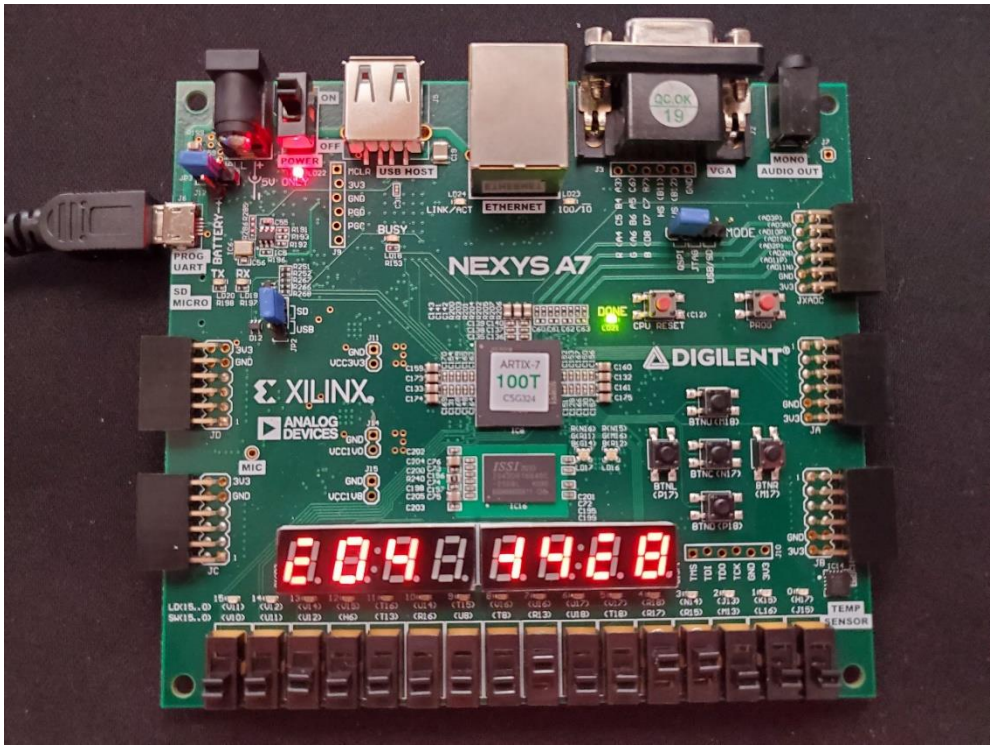


Figure 7: FPGA Adding Positive and Negative Number

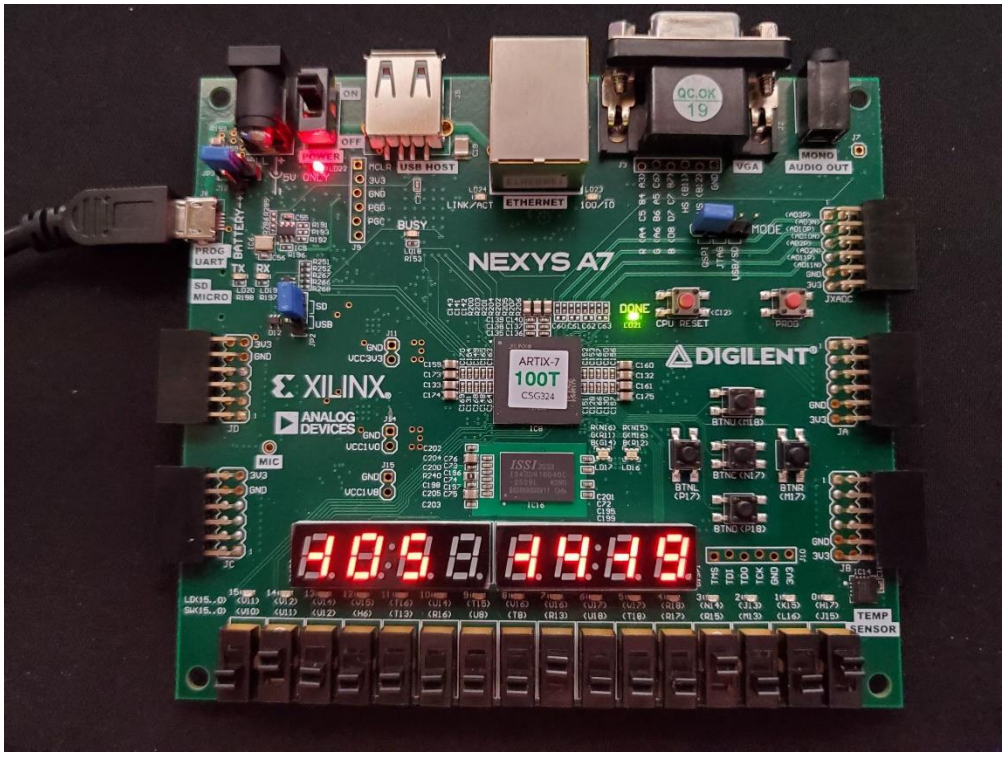


Figure 8: FPGA Subtracting Two Negative Numbers

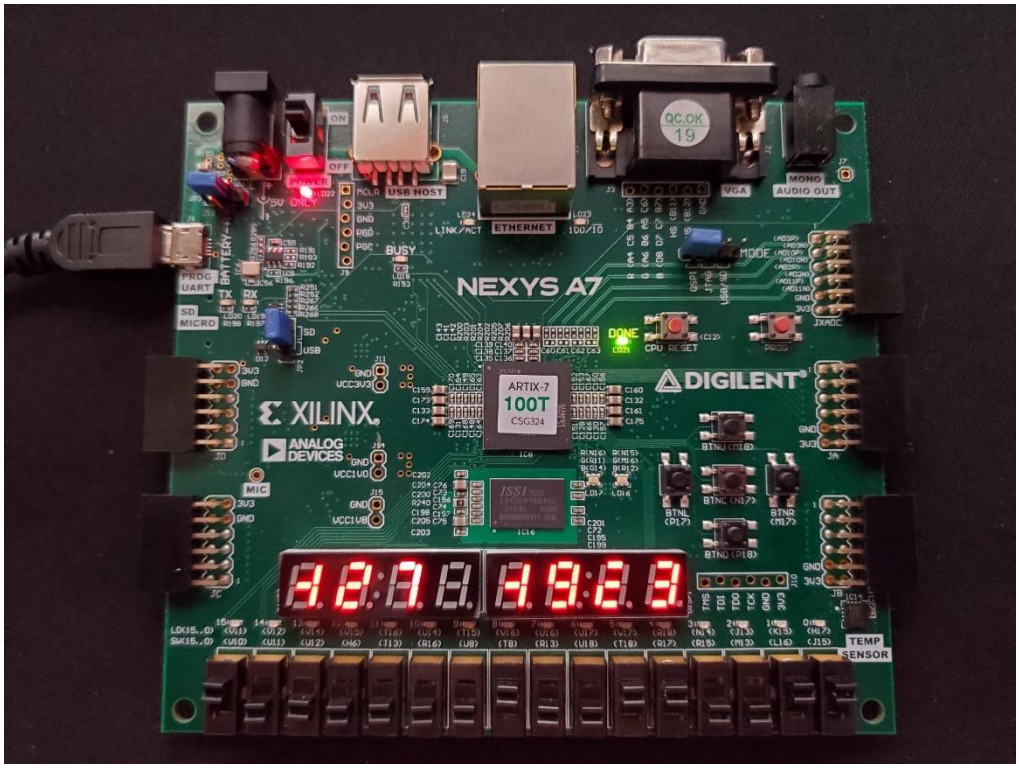


Figure 9: FPGA Multiplying Two Negative Numbers

Discussion

There was a small issue where splitting our ones and tens digits in order to display them on the seven-segment caused our refresh loop to slow down. This caused a small bit of flickering on the seven-segment displays. However, this was fixed by doubling the refresh rate of our seven-segment displays. The number of LUTs used is a little bit concerning but could be fixed with some code optimizations. This could be in the form of figuring out a way to create a module whose only purpose is to convert to BCD from binary. Overall, the calculator works and all the major bugs were fixed.

Conclusion

This lab proved to be a worth adversary. It finally worked in the end when we encased the display of the high and low values in an if statement. The clock and the display took the longest to figure out, and then it was the sign values. Overall, we learned a lot of things that we could do and couldn't do, but our main takeaway was creating a counter that would refresh so that we could display all the seven-segment display.

Work Distribution

Russell: Code, testbench, debugging, video and report

Philbert: Code, video, debugging, and lab report

Paul: Code, video, debugging, and lab report.