Sanjay Gounder

Professor Luis Garcia

ECE 3700

Lab Session – Friday Morning

Lab 1 Report

By Sanjay Gounder

Abstract – In this lab report, you will see my process for going through the lab and the various phases. To first provide context, I will explain the objective from my point of view. Knowing my opinion on the objective will showcase my approach for going through this lab. I plan to show off the design of the logic equations, any optimizations used, and the resulting schematics for building the circuit. I then progressed towards simulating the circuits and how it would look going from high and low states for each of the respective inputs (i_0 and i_1) and output functions (f_0 and f_1 – or minterms and maxterms). Finally, I connected the device properly, and soon, after some testing and tinkering, I managed to create a 2-bit computer where the input vectors mattered in determining the which pinheads would light up.

Lab Objective

As the lab indicated, the objective overall for this lab is to "learn the techniques of 'Schematic Entry', 'Verilog Design', 'Simulation' and 'Synthesis' in the context of digital logic design." It will also introduce me to a 2-bit computer.

Lab Approach

I first started with creating a schematic based off the desired inputs.

- When $I[1:0] = (i_1,i_0) = (0,0)$, the output $F[1:0]$ is a bit-wise AND of the inputs, i.e. $f_1 = a_1 \cdot b_1$; $f_0 = a_0 \cdot b_0$.
- When $I[1:0] = (i_1,i_0) = (0,1)$, the output $F[1:0]$ is a bit-wise OR of the inputs, i.e. $f_1 = a_1 + b_1$; $f_0 = a_0 + b_0$.
- When $I[1:0] = (i_1,i_0) = (1,0)$, the output $F[1:0]$ performs an *equality check* of the corresponding bits; i.e. $f_0 = 1$ if $a_0 = b_0$, otherwise $f_0 = 0$. Similarly, $f_1 = 1$ when $a_1 = b_1$, otherwise $f_1 = 0$.
- When $I[1:0] = (i_1,i_0) = (1,1)$, the output $F[1:0]$ perform the complement of $A[1:0]$; i.e. $f_0 = \overline{a_0}$ and $f_1 = \overline{a_1}$.

Fig. 1: The circuit functionality.

Based off these instructions for the circuit, and the fact that the values of the input control signals decide how the circuit functions, I determined we would simply implement the AND, OR, equality check (XOR), and NOT functions. I finally put an AND gate for the 4 minterms (m_0, m_1, m_2, m_3) as well as and OR gate for the 4 maxterms (M_0, M_1, M_2, M_3). The AND gate and the OR gate are then OR'd together overall, leading to two functional outputs, f_0 and f_1.

At this point, I would then go into Verilog software through generating a structural Verilog with the schematic I had designed.

```
13  // refer to the applicable agreement for further details.
14
15  // PROGRAM     "Quartus Prime"
16  // VERSION     "Version 18.1.0 Build 625 09/12/2018 SJ Lite Edition"
17  // CREATED     "Fri Feb  2 10:55:26 2024"
18
19  module Lab1(
20      A0,
21      A1,
22      B0,
23      B1,
24      I0,
25      I1,
26      f0,
27      f1
28  );
29
30
31  input wire  A0;
32  input wire  A1;
33  input wire  B0;
34  input wire  B1;
35  input wire  I0;
36  input wire  I1;
37  output wire f0;
38  output wire f1;
39
40  wire  SYNTHESIZED_WIRE_0 ;
41  wire  SYNTHESIZED_WIRE_1 ;
42  wire  SYNTHESIZED_WIRE_22 ;
43  wire  SYNTHESIZED_WIRE_3 ;
44  wire  SYNTHESIZED_WIRE_4 ;
45  wire  SYNTHESIZED_WIRE_23 ;
46  wire  SYNTHESIZED_WIRE_9 ;
47  wire  SYNTHESIZED_WIRE_12 ;
48  wire  SYNTHESIZED_WIRE_14 ;
49  wire  SYNTHESIZED_WIRE_15 ;
50  wire  SYNTHESIZED_WIRE_16 ;
51  wire  SYNTHESIZED_WIRE_17 ;
52  wire  SYNTHESIZED_WIRE_18 ;
53  wire  SYNTHESIZED_WIRE_19 ;
54  wire  SYNTHESIZED_WIRE_20 ;
55  wire  SYNTHESIZED_WIRE_21 ;
56
```

Figure 2: Set up Verilog File

Above is the setup which Verilog schematic designed. You see there are 6 input wires for a_0; a_1; b_0; b_1; input controllers, i_0 and i_1; and, followed by the output wires, f_0 and f_1. This sets up what actions will be run and how it will connect to the FPGA. At this point, I needed to actually create the testbench Verilog code as well as the actual functional Verilog file, with the definitions from above.

```
1  module Lab1TestBench();
2
3  reg I1t, I0t, A1t, A0t, B1t, B0t;
4  wire f1t;
5  wire f0t;
6
7  integer i, j, k, l, m, n;
8  //integer a0temp; integer a1temp; integer b0temp; integer b1temp; integer i0temp; integer i1temp;
9
10    Lab1Functional test(
11    .I1(I1t),
12    .I0(I0t),
13    .A1(A1t),
14    .A0(A0t),
15    .B1(B1t),
16    .B0(B0t),
17    .f1(f1t),
18    .f0(f0t)
19    );
20
21    initial begin
22       begin
23       for (i=0;i<2; i= i+1)
24          begin
25          for (j=0;j<2; j= j+1)
26             begin
27             for (k=0;k<2; k= k+1)
28                begin
29                for (l=0;l<2; l= l+1)
30                   begin
31                   for (m=0;m<2; m= m+1)
32                      begin
33                         for (n=0;n<2; n= n+1)
34                            begin
35                            A0t=i; A1t=j; B0t=k; B1t=l; I0t=m; I1t=n;
36                            #2;
37                            $display("A0:%b,A1:%b,B1:%b,I0:%b,I1:%b,f0:%b,f1:%b," ,A0t,A1t,B0t,B1t,I0t,I1t,f0t,f1t);
38                            end
39                      end
40                   end
41                end
42             end
43          end
44       end
45    end
46 endmodule
```

Figure 3: Testbench

Figure 3 is the testbench developed. Using the input wires and outputs defined from Figure 2, I wrote out this for-loop where I would do inner loops for all the inputs. I defined temporary registers, depicting these registers as such through putting the ending character as ‘t’ to identify them as temporary. I then ran it, and within the most inner for-loop, I would set the temp registers to their spot within the loop.

Simulating the results finally using this testbench, we see the following:

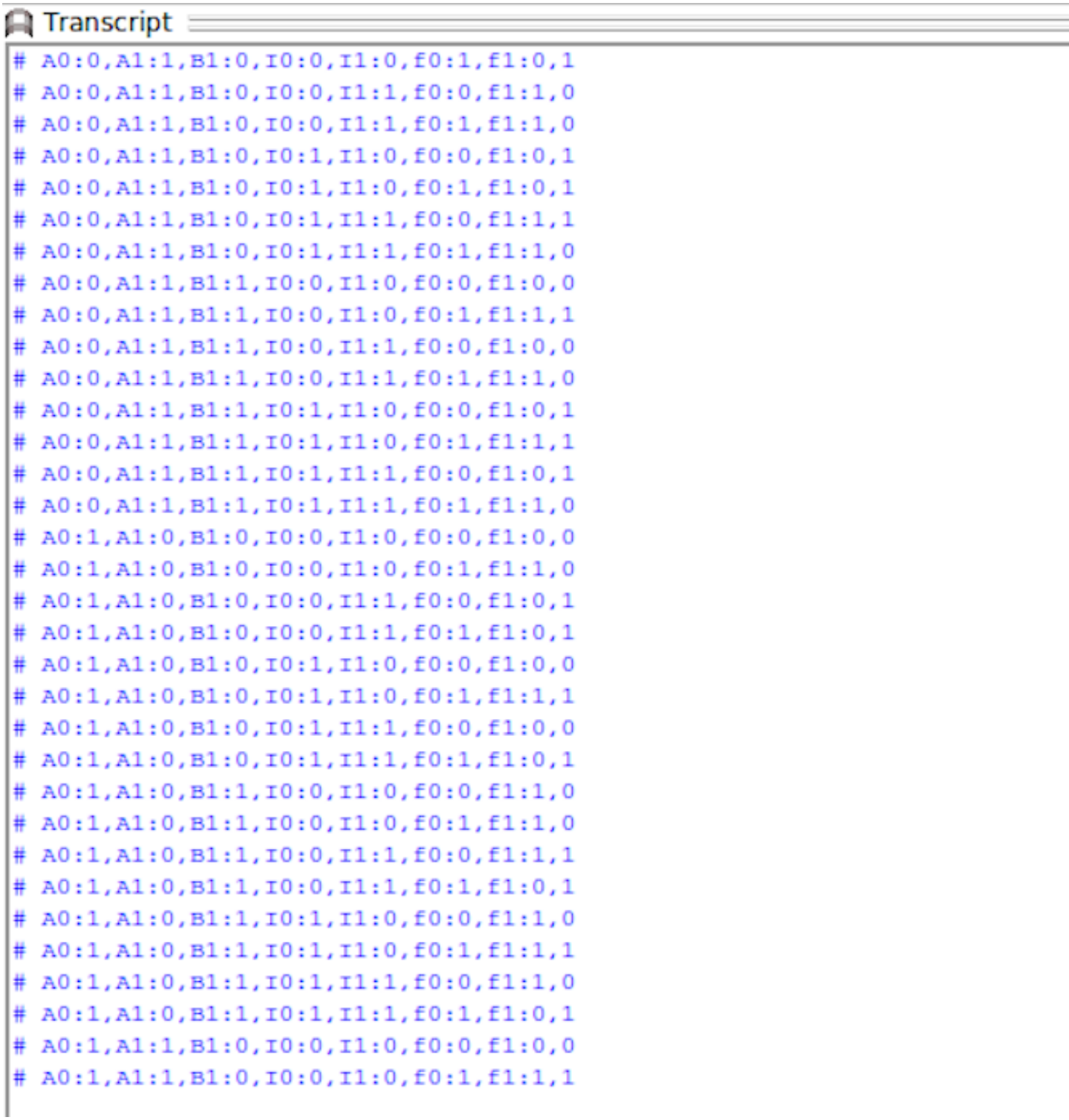

Figure 4: Simulation Register Results

Figure 4 shows the results and how the inputs are changing from high state to low state.

This next figure will show the waves generated, and how these inputs look graphed:

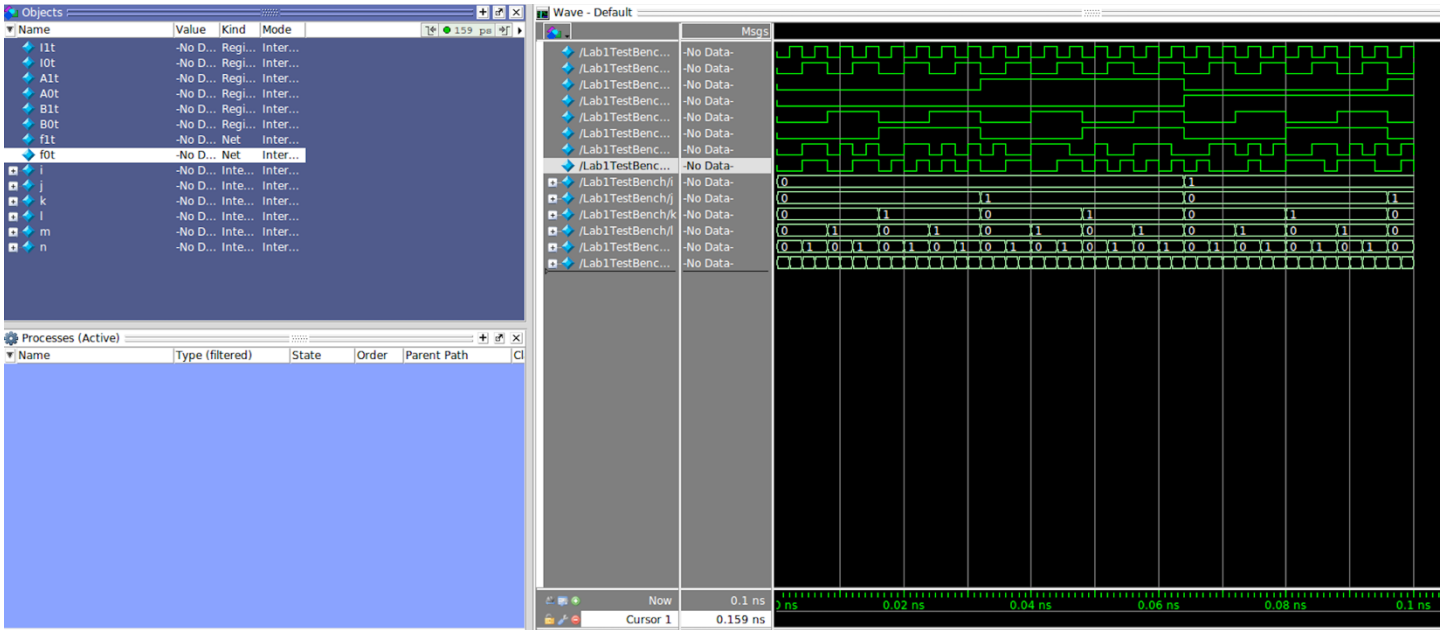

Figure 5: Simulation Waves

We see here the visualization of Figure 4, showcasing nice, smooth curves and doing exactly what we need.

Finally, here are pictures of the synthesis:

Figure 6: Picture showcasing the 2-bit computer working.

Figure 7: Another picture showcasing the 2-bit computer working with different inputs.

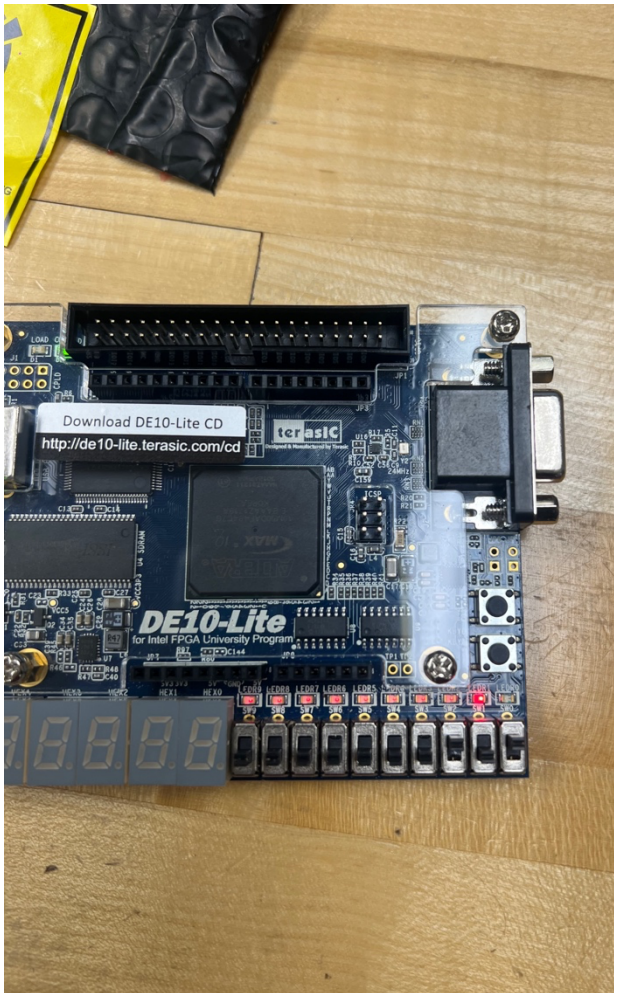

Figure 8: Another picture showcasing how. the 2-bit computer works with different inputs – maxterm here.

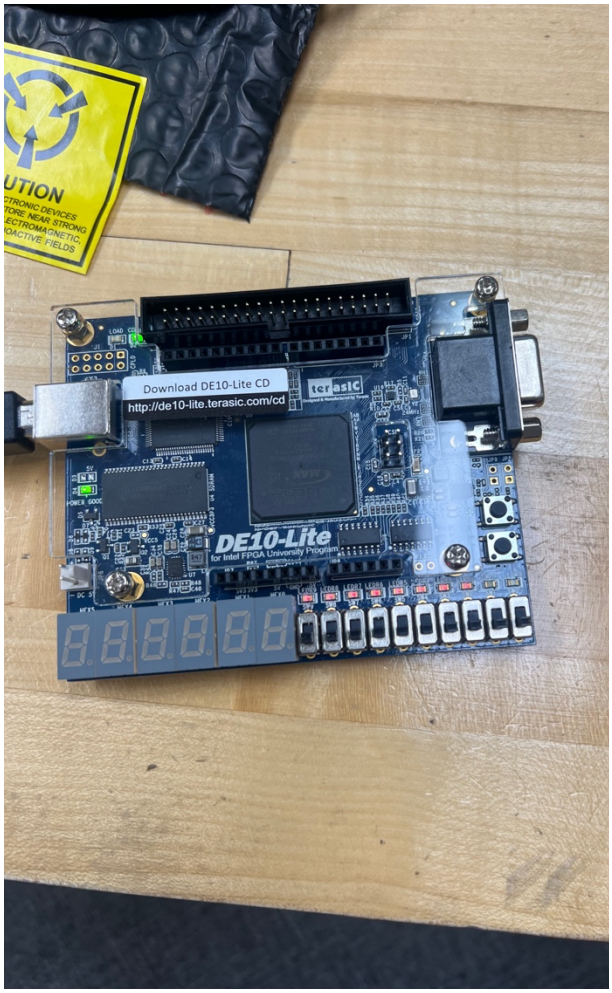

Figure 9: Another picture showcasing how. the 2-bit computer works with different inputs – maxterm here.

Pin-header layout shown here:

Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard
A0	Input	PIN_C12	7	B7_N0	PIN_C12	3.3-V LVTTL
A1	Input	PIN_D12	7	B7_N0	PIN_D12	3.3-V LVTTL
B0	Input	PIN_B12	7	B7_N0	PIN_B12	3.3-V LVTTL
B1	Input	PIN_A12	7	B7_N0	PIN_A12	3.3-V LVTTL
I0	Input	PIN_C11	7	B7_N0	PIN_C11	3.3-V LVTTL
I1	Input	PIN_C10	7	B7_N0	PIN_C10	3.3-V LVTTL
f0	Output	PIN_A9	7	B7_N0	PIN_A9	3.3-V LVTTL
f1	Output	PIN_A8	7	B7_N0	PIN_A8	3.3-V LVTTL
<<new node>>						

Figure 10: Pin-header layout

Lab Design

```
module Lab1Functional(A0, A1, B0, B1, I0, I1, f0, f1);

input A0, A1, B0, B1, I0, I1;
output f1, f0;

assign f0 = (~I0 & ~I1 & (A0 & B0)) | (I0 & ~I1 & (A0 | B0)) | (~I0 & I1 & ~(A0 ^ B0)) | (I0 & I1 & ~A0);
assign f1 = (~I0 & ~I1 & (A1 & B1)) | (I0 & ~I1 & (A1 | B1)) | (~I0 & I1 & ~(A1 ^ B1)) | (I0 & I1 & ~A1);

endmodule
```

Figure 11: Logic Equations (Boolean functions)

Lab Schematic

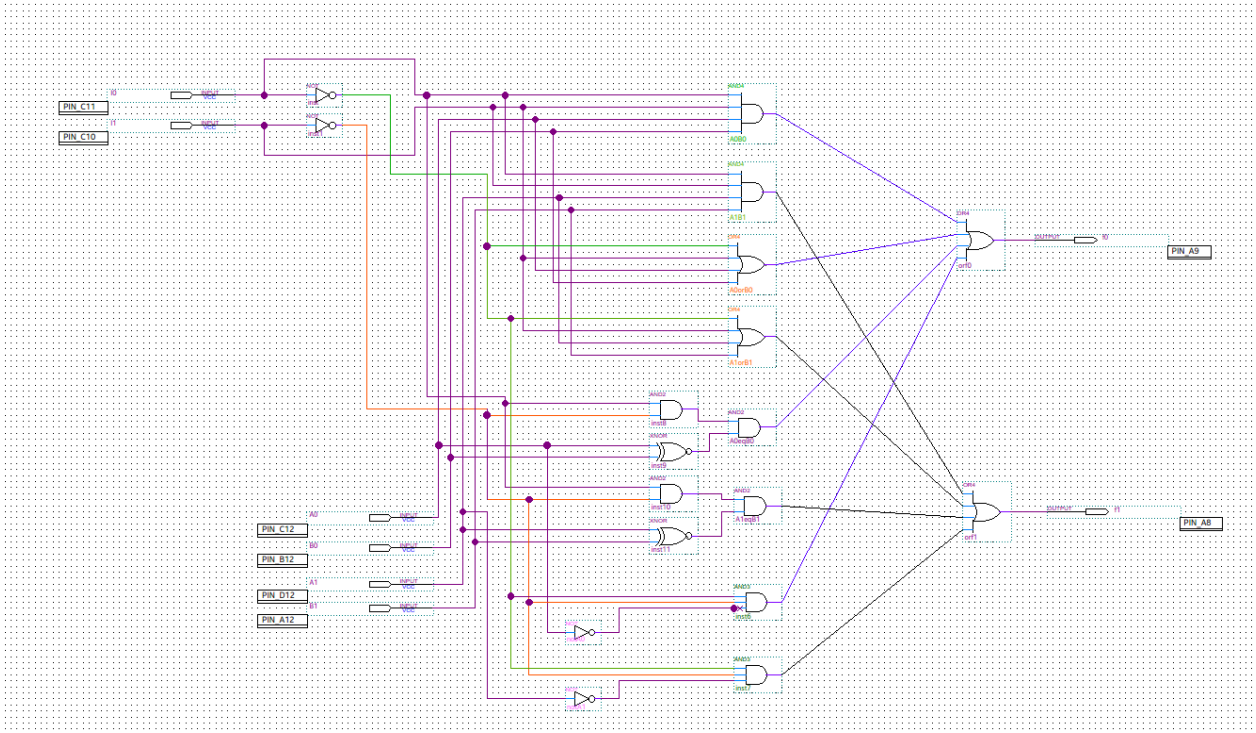

Figure 12: Lab Schematic

Lab Simulation and Synthesis Results

Flow Summary

<<Filter>>

Flow Status	Successful - Fri Feb 2 14:48:12 2024
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	Lab1_2
Top-level Entity Name	Lab1Functional
Family	MAX 10
Device	10M50DAF484C7G
Timing Models	Final
Total logic elements	3 / 49,760 (< 1 %)
Total registers	0
Total pins	8 / 360 (2 %)
Total virtual pins	0
Total memory bits	0 / 1,677,312 (0 %)
Embedded Multiplier 9-bit elements	0 / 288 (0 %)
Total PLLs	0 / 4 (0 %)
UFM blocks	0 / 1 (0 %)
ADC blocks	0 / 2 (0 %)

Figure 13: Synthesis and Simulation Results

From these results, we see the types of logic elements really used, AND/OR/XOR logic, really. This is verified through the logic equations from Figure 12.

Conclusion

Overall, this report has taken us through the method and assembly of creating a schematic, simulating and synthesizing the schematic with the appropriate functional aspects, and then taking this schematic to life. I really learned a lot and came to feel more proficient in Verilog, creating schematics, and navigating future digital circuit designs.