ECE385 Experiment #3

Eric Meyers, Ryan Helsdingen

Section ABG; TAs: Ben Delay, Shuo Liu

February 10th, 2016

emeyer7, helsdin2

I. Introduction

THE purpose of this lab is to design and construct a four-bit serial logic processor that performs a total of eight logical operations in a bit-wise fashion. There are two four-bit words stored in two shift registers, A and B. The operator may store any data of their choosing in these two shift registers by using the load/data-in switches and then execute any logical bitwise operation on these two words by using the function-select switches along with the routing switches.

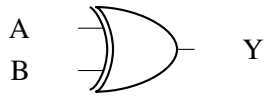
II. Pre-Lab

Part A) Describe the simplest (two-input one-output) circuit that can optionally invert a signal (i.e., one input determines if the output is equal to the other input or equal to the other input inverted). Sketch your circuit.

Answer: The logic needed is shown in the following truth table:

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

Taking B to be the ”select” line in this example, when the input (A) is driven high and our select line (B) is low, the output will be high. The reverse case is true when the input (A) is low and the select line (B) is high, the output will also be high. Therefore, depending on the select line, the input will be inverted. This can be implemented with a single XOR gate as shown below.



Part B) Explain how a modular design such as that presented above improves testability and cuts down development time. Propose an approach that could be used to troubleshoot the modular circuit above if it appeared to be completing the computation cycle correctly but was not giving the correct output. (Be specific.)

Answer: A modular design allows the operator to unit-test each individual module. Once each module is successfully tested, the entire system can be integrated into a whole-system and tested. A modular design ideally will cut down the debugging time needed on the system as a whole.

III. Description of Circuit

The logic processor was designed to contain two storage registers, and allow the operator to specify the data loaded into each register (A and B), as well as the operation performed between the two register, where the output is stored and The team designed a total of four modular sub-systems in this lab. The following sub-systems were designed and constructed:

1) Data Loader Unit
2) Computation Unit
3) Function-Select and Output Router Unit
4) Control Logic Finite State Machine (FSM) Unit

Each sub-system was tested individually and system integration tests were performed at the end upon successfully unit-testing each component. This way the team could eliminate errors and debug with efficiency.

A total of eight operations can be performed on this unit. AND, OR, XOR, NAND, NOR, and NXOR are the six primary logical operations that can be performed, along with the last two operations simply being loading

the specified registers with all 1s or 0s.

The Data Loader consists of a total of six switches. Two switches for loading either register A or B, and four switches to specify the contents to store in the particular register.

The Computation Unit performed the selected operation specified through the three function-select switches. The output of the operation is funnelled into the proper register as given by the operator.

Finally, the Control Logic/Finite State Machine Unit is a sequential circuit that essentially determines the correct number of cycles to perform a full logical operation. It relies on one input (execute switch) and has one output that determines if the registers should be shifting right, doing nothing, or loading.

In the end, the functionality of the circuit is as follows: when a user selects a four-bit word and loads it into the register of their choosing, they must then select the logical operation they would like to perform, select where they would like the output of this logical operation to go via the routing selection, and flip the execute switch. This will begin shifting the outputs of the shift register and perform bitwise logical operations of the requestion function. The machine will halt after exactly four operations and the register requested through the routing switches will contain the output of the logical operation.

IV. State Diagram

The FSM created in this lab is a Mealy State Machine and this was chosen due to the simplicity of the relationship between the input and output on the state diagram. (???????)

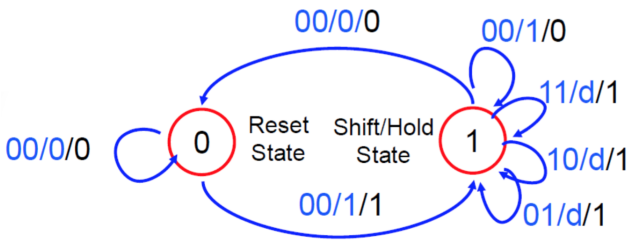


Fig. 1: Mealy FSM

V. Design

The first component designed in this lab was the FSM Sequential Logic Circuit. This is because the output of this state machine controls the actions on the shift register (i.e. this output controls the S1 and S0 ”control bits” , which determines whether the registers are shifting right, parallel loading, or doing nothing). This is important because it determines whether the machine is in a load/execute/do nothing state. These proper inputs to the control bits on the shift registers will be examined further on in this section.

The truth table for the FSM was given from the documentation in the Experiment 3 PDF. This is shown below in figure.

Exec. Switch (‘E’)	Q	C1	C0	Reg. Shift (‘S’)	Q’	C1’	C0’
0	0	0	0	0	0	0	0
0	0	0	1	d	d	d	d
0	0	1	0	d	d	d	d
0	0	1	1	d	d	d	d
0	1	0	0	0	0	0	0
0	1	0	1	1	1	1	0
0	1	1	0	1	1	1	1
0	1	1	1	1	1	0	0
1	0	0	0	1	1	0	1
1	0	0	1	d	d	d	d
1	0	1	0	d	d	d	d
1	0	1	1	d	d	d	d
1	1	0	0	0	1	0	0
1	1	0	1	1	1	1	0
1	1	1	0	1	1	1	1
1	1	1	1	1	1	0	0

Fig. 2: FSM Truth Table

Next, given this truth table, the team constructed a K-Map for each of the output bits - S, Q , C1 and C0. Since the only output is ”S”, and the remaining bits are only internal to the state machine, there must be three D-flip-flops to store the state of these internal bits. The K-Maps are shown along with the cooresponding S.O.P resultant equations.

EQ \ C1C0	00	01	11	10
00	0	x	x	x
01	0	1	1	1
11	0	1	1	1
10	1	x	x	x

FSM Sequential Circuit Output (S)

$$S = C_1 \vee C_0 \vee \neg E \wedge Q$$

EQ \ C1C0	00	01	11	10
00	0	x	x	x
01	0	1	1	1
11	1	1	1	1
10	1	x	x	x

State Internal Bit (Q)

$$Q = C_1 \vee C_0 \vee E$$

EQ \ C1C0	00	01	11	10
00	0	x	x	x
01	0	1	0	1
11	0	1	0	1
10	0	x	x	x

Counter Internal Bit (C1)

$$C_1 = C_1 \oplus C_0$$

EQ \ C1C0	00	01	11	10
00	0	x	x	x
01	0	0	0	1
11	0	0	0	1
10	1	x	x	x

Counter Internal Bit (C0

$$Q = (C_1 \wedge \neg C_0) \vee (\neg E \wedge Q)$$

Once these logic functions were derived, an AND-OR circuit was designed and then converted into a NAND-NOR circuit. This NAND-NOR circuit was optimized to use the least number of chips possible and as a result, only used a total of 5 TTL chips.

The next modular component that was designed was the computational and router unit. This was the circuit that performed a particular logical operation. The three primary functions (AND, OR, XOR) and a high signal is fed into a 4-to-1 MUX with the select lines connected to the F1 and F0 switches. This output of the 4-to-1 MUX is then fed into a comparator. The second line of this comparator is connected to the F2 switch (negated). This effectively acts as an XOR gate. This is so that we do not have to use extra logic gates on the second set of logical operations (NAND, NOR, NXOR, 0000), and instead use an optional inverter, which was demonstrated in the pre-lab.

This output is then fed to the proper inputs on the two 4-to-1 MUXs connected to the shift registers. The R1 and R0 control the routing selection, and the two MUXs on the top must be fed the right values in order to route the output correctly and perform the proper action. This output of the MUX then is fed into the left serial input on the shift register.

Next the shift register control bits needed to be determined to either be shifting right parallel loading or doing nothing. S1 and S0 must both be high if a parallel load is being performed on the shift register, so this means the LoadA switch can be wired directly to the A shift register S1 control bit, and the LoadB switch can be wired directly to the B shift register S1 control bit. The S0 must come from the Control Logic Circuit (the "S" output), and be fed into some combinational logic that is derived below:

S	LA	LB	S1A	S0A	S1B	S0B
0	0	0	0	0	0	0
0	0	1	0	0	1	1
0	1	0	1	1	0	0
0	1	1	1	1	1	1
1	0	0	0	1	0	1
1	0	1	0	0	1	1
1	1	0	1	1	0	0
1	1	1	1	1	1	1

This is referencing the fact that the control bits on each Register A and Register B are as follows:

S1	S0	Action
0	0	Do Nothing
0	1	Shift Right
1	0	Shift Left
1	1	Parallel Load

Next, from the above first truth table K-Maps can be formed, this is shown below with the cooresponding S.O.P functions along side them.

S \ LA/LB	00	01	11	10
0	0	0	1	1
1	0	0	1	1

$$S_1 : A = LoadA$$

S \ LA/LB	00	01	11	10
0	0	0	1	1
1	1	0	1	1

$$S_0 : A = \neg LoadA \vee (\neg LoadB \wedge S)$$

S \ LA/LB	00	01	11	10
0	0	1	1	0
1	0	1	1	0

$$S_1 : B = LoadB$$

S \ LA/LB	00	01	11	10
0	0	1	1	0
1	1	1	1	0

$$S_0 : B = \neg LoadB \vee (\neg LoadA \wedge S)$$

Once these equations were formed, an AND-OR circuit was designed, then from this a NAND-NAND circuit containing only four logic gates was created. This serves as the control of our shift registers and will feed to the proper place upon constructing.

All in all, the total number of chips used was 18 TTL chips. This chip count could have been reduced if better optimization was performed, however, since the team possessed the required chips, none was needed. The chips used were as follows:

- Control Logic Unit
  - 74LS00 - 2-input NAND (2)
  - 74LS02 - 2-input NOR
  - 74LS04 - Hex Inverter
  - 74LS27 - 3-input NOR
  - 74LS107 - J-K Flip Flop (2)
- Computation and Router Unit
  - 74LS00 - 2-input NAND
  - 74LS85 Comparator
  - 74LS153 4-to-1 Multiplexer
  - 74LS194 - Universal Shift Register (2)
- Loader Unit
  - 74LS02 - 2-input NOR (3)
  - 74LS04 - Hex Inverter

VI. Block Diagram

Please refer to Figure 3 in "Section XI: Figures" of this document to view the Block Diagram created in this lab.

VII. Circuit/Logic Diagrams

As stated before, to allow for easier construction and debugging, this system was broken down into modular components. All circuit diagrams are shown in "Section XI: Figures" of this document.

The FSM Control Logic is shown in Figure 4. The Loader is shown in Figure 5 and the Computational/Router Unit is shown in Figure 6.

VIII. Component Layout Sheet

Please refer to Figure 7 in "Section XI: Figures" of this document to view the Component Layout Sheet created in this lab.

IX. Documentation from Experiment

Lab 3 began with a design phase and quickly turned into debug, adjust design accordingly and a tedious loop back to debug. The issues derived from a variety of errors including logic, design, and wiring among others.

Debugging was broken down into sections beginning with the Register Unit. Initially, the shift registers were parallel loading into their respective registers when that register was high, as they should. But if only one load switch was high, then that register received its input while the other register was pushed to all zeroes. The error was in our logic controlling when to parallel load DA3-DA0 and DB3-DB0 and when to tell the shift registers to simply do nothing. Each input originally only had the load switch ANDed to the data input but the problem was the two switch bits for the shift registers were tied together. Both registers were completing the same action which needed to be fixed. The solution was to make a large truth table with inputs

LoadA, LoadB, and S and outputs SA1, SA0, SB1, and SB0. The logic can be seen in the Loader Logic Diagram minus the inverter and NOR gates. Although the team did not remove them from our circuit, the inverter and NOR gates in the Loader Logic Diagram are unnecessary. They are redundant logic. These gates were intended to select which registers should receive data upon switching the proper load gate, but the logic leading up to the switch bits on the shift registers does all the work for us. they decide when to parallel in and when to do nothing. Eliminating the redundancy would have taken away 8 NOR gates (2 chips) and 4 inverters (1 chip) from our board.

Debugging moved on to the Computation Unit and Routing Unit. The Computation Unit had small misplaced wire issues, but for the most part was quickly functional for the team. For the Routing Unit, the 4-1 mux gave the team trouble, particularly with lining up the inputs correctly. At one point the pin numbers were associated with the wrong input bits. For example, pin 10 rather than pin 13 was thought to pass through with an R1R0 of 11. Also confusion over which pin of the mux was R1 and R0 created some unnecessary debugging.

At this point everything but the control unit worked. The circuit would properly load in data, run through the functions, and route them properly, but the registers would constantly shift right. The S bit was coming from the control logic was not changing value. After debugging, it turned out to be some missing connections on the breadboard. The next issue the team had was a clock issue. The circuit would clock three times instead of four on about 75% of the test cases. The error was not obvious at first, but it soon became clear it the fault was the team's flip-flop selection in the control unit. The SN74LS107A is a negative edge triggered JK flip-flop. The timing of the negative-edge was not consistent and the clock needed to be inverted to conclude debugging.

X. Conclusion

The team was able to fulfill all demo points in lab 3 successfully creating the bit-serial logic operation processor. All eight functions, four routing locations, the finite state machine control, and register unit successfully came together in the end. While there were minor challenges along the way, lab 3 was not a difficult lab upon going through the logic and design. However handling the TTL was a challenge. There is a finite number of ways to get to a working design, but there are infinite ways to create an error. For this lab, the tiniest error such as a wire misplaced by one hole on the breadboard could completely change the functionality of the circuit.

The team's final design was a tangle of wires and too many chips. 18 chips were used in total and that number certainly could be optimized. As discussed earlier, the loader logic diagram was not optimized. The NOR gates and four of the hex inverters were redundant logic. Also the team used a comparator to invert the function given a HIGH on the F2 input. This function could have been implemented by an XOR rather than an entire chip.

XI. FIGURES

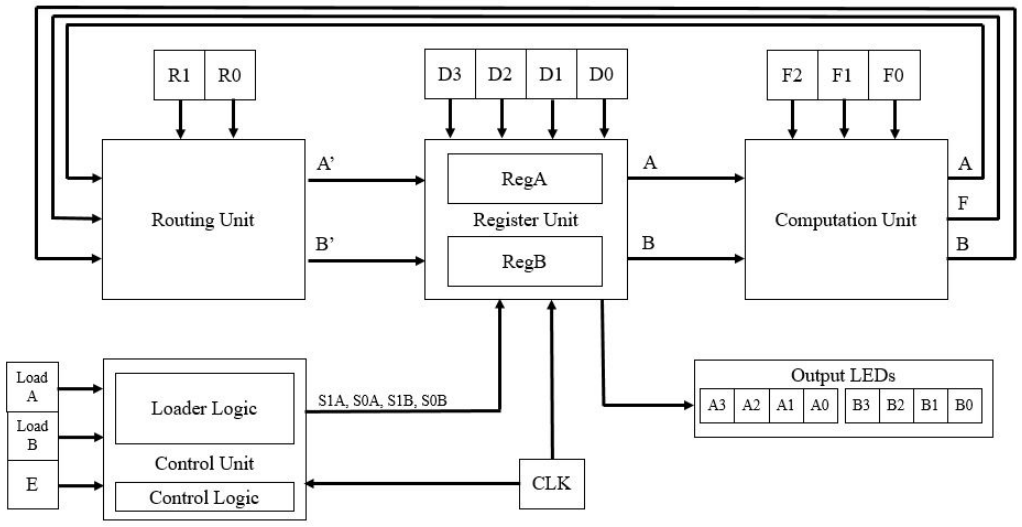


Fig. 3: Block Diagram

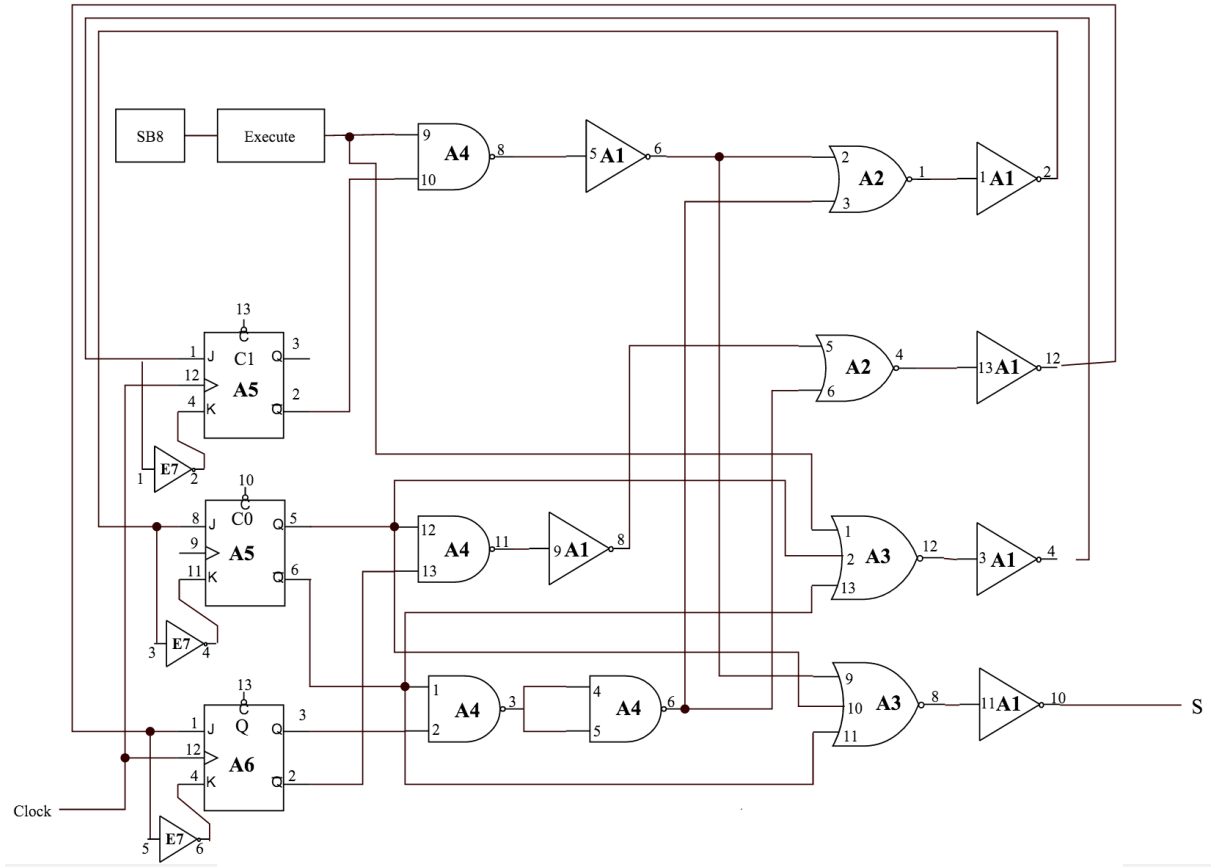


Fig. 4: Control Logic Diagram

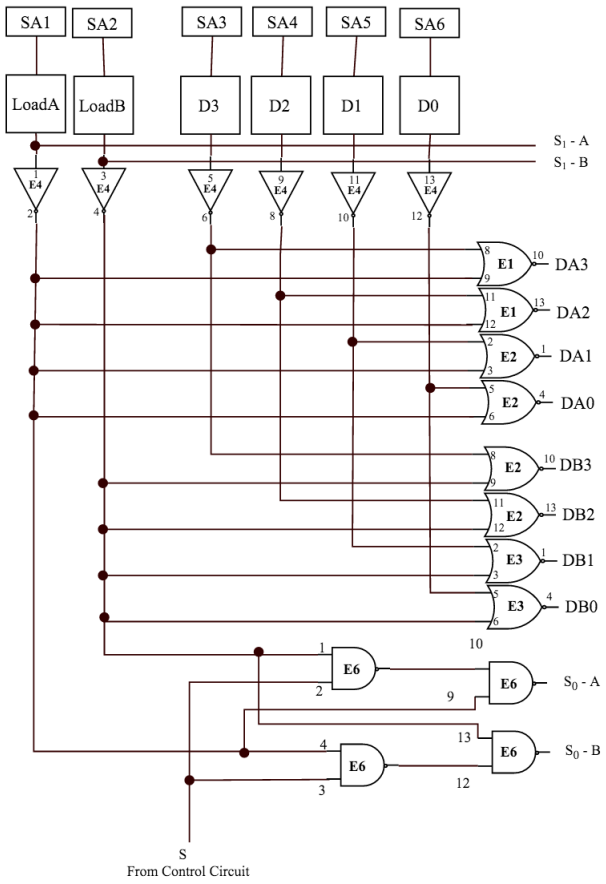


Fig. 5: Loader Logic Diagram

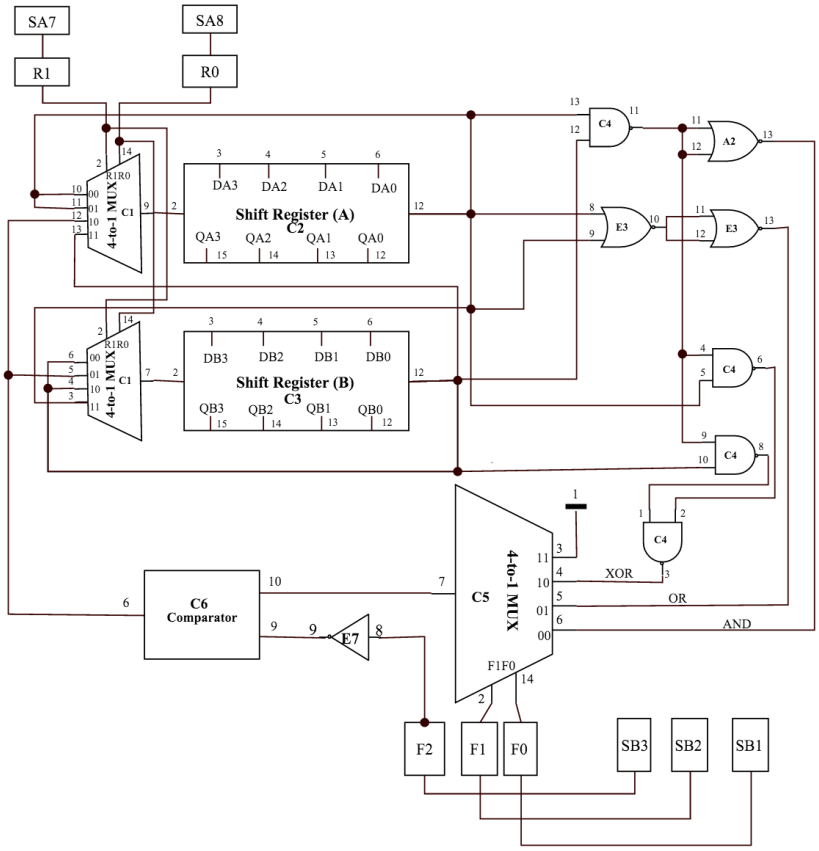


Fig. 6: Computational/Router Logic Diagram

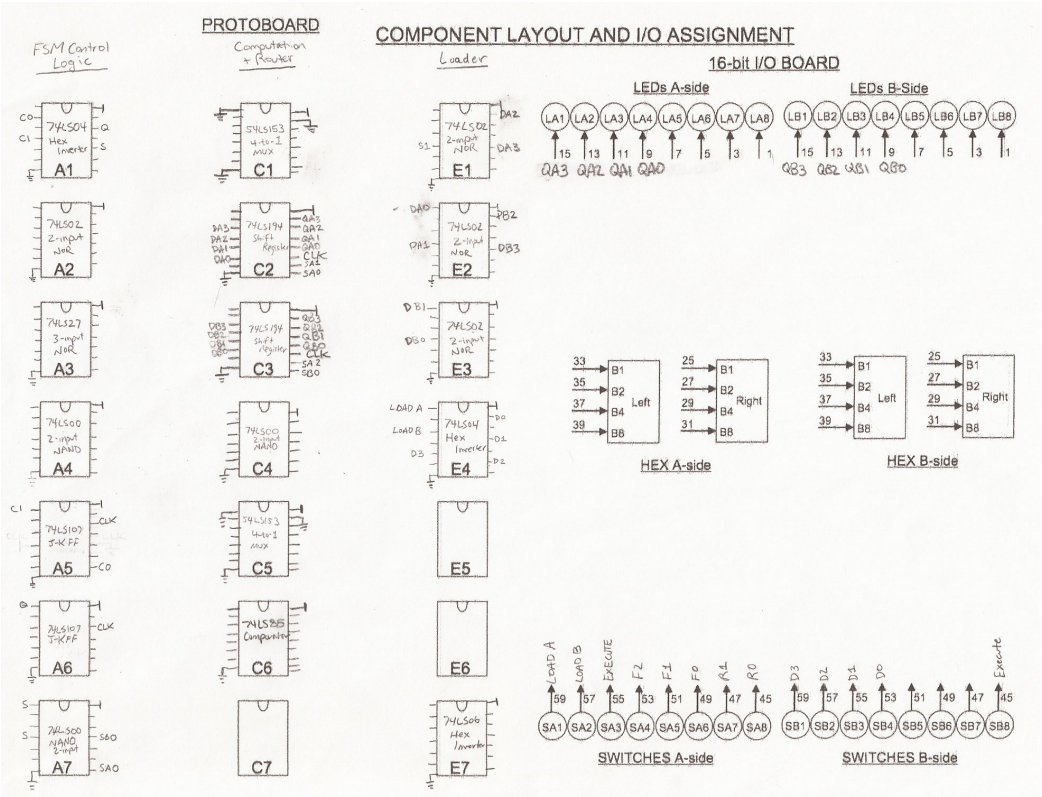


Fig. 7: Component Layout Diagram