ECE 385

Spring 2020

Experiment 3

A Logic Processor

Michael Faitz and Zohair Ahmed

Section ABJ: Friday 2:00-4:50

Yuming Wu and Lian Yu

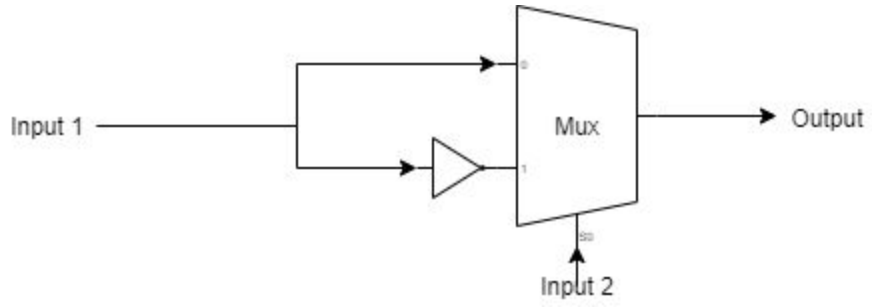
Introduction

Our circuit is meant to perform a variety of logical operations on two sets of data, which will then be stored back into the registers. The data inside the registers can be loaded by the user, and both the operations and locations for the logic outputs can be changed by switches. It can operate on 2 4-word shift registers, two bits at a time.

Prelab:

Q: 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.

A: The simplest two-input one-output circuit that can optionally invert a signal is a 2:1 multiplexer that takes one bit as a select bit, with one input being the signal and the other being the same signal with an inverter in between.



Q: Explain how a modular design such as that presented above improves testability and cuts down development time.

A: Modular design allows for each section of the circuit to be tested separately. It also allows different sections to be built at the same time, allowing for faster build speed. During this lab, we used this to test our loading, routing, and computing separately. While our only issue was using a faulty switch for a clock instead of the function generator, this allowed us to be confident as to which part of our circuit worked and which didn't.

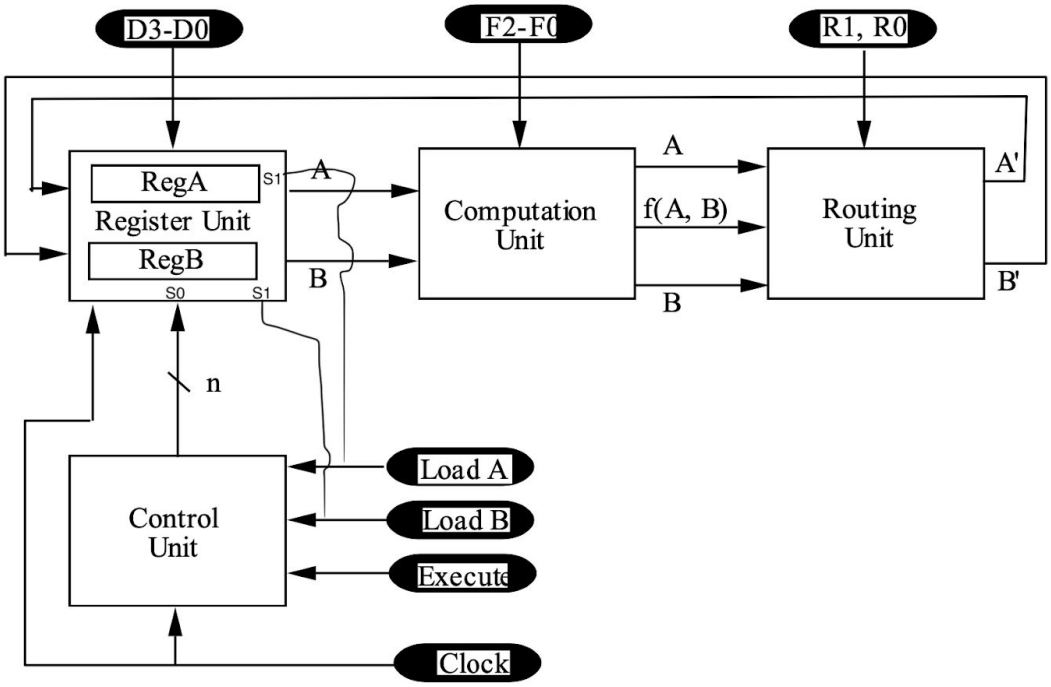
Operation of the Logic Processor

a. To load data into register A/B, the execute switch must be set to low and the shifting must halt. Then, the Load A/B switch must be set to high. Now the 4 data in bits D[3:0] can be set high or low to set each bit inside the register(s).
b. Before performing the computation and routing, set the F[2:0] switches to perform the desired operation, and then switches R[1:0] to route desired outputs to the desired registers. To initiate a computation and routing operation, make sure the Load A/B switches are set to low, then set execute to high. At this point, the processor will perform the operation on every 2-bit word in the register and route the output to the desired registers.

Written description, block diagram and state machine diagram of logic processor

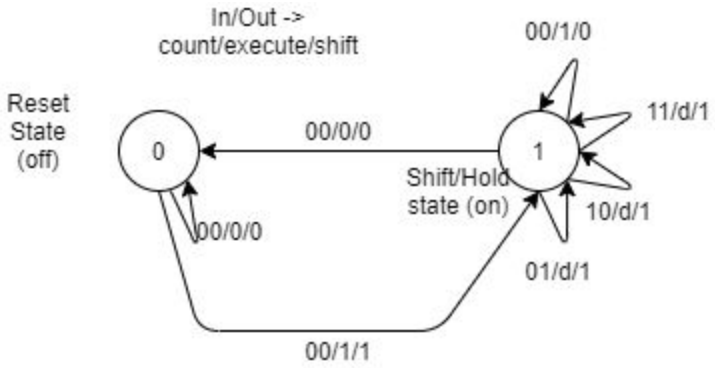
a. The routing unit is made of 2 4:1 MUXes and takes inputs from switches R1, R0. Using these switches, the 4:1 MUXes take the outputs from the shift registers and/or the computation unit and send them to registers A/B. The register unit takes outputs from the computation, directed by the routing unit into the leftmost bit. The register loads when LoadA/B is high. The register shifts when LoadA/B is low and shift is high. The register does nothing when shift and LoadA/B are low. The computation unit takes inputs from the rightmost bits of the shift register, picks an operation given by the switches F[2:0] and outputs that into the routing unit. The control unit uses a counter and many logic gates to create a shift bit output, which puts the rest of the circuit in operation. It is designed to perform 4 operations when the execute switch is set to high, and reset when execute is set to low again, regardless of its position during the 4 operations.

b. High Level Block Diagram



c. State Machine Diagram

i. We used the Mealy machine specified in the lecture slides.



Design steps taken and detailed circuit schematic diagram

- Written procedure of the design steps taken.
  i. Kmaps and Truth Tables

Execute	Q	C1	C0	shift	Q+	C1+	C0+
0	0	0	0	0	0	0	0
0	0	0	1	X	X	X	X
0	0	1	0	X	X	X	X
0	0	1	1	X	X	X	X
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	X	X	X	X
1	0	1	0	X	X	X	X
1	0	1	1	X	X	X	X
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

K-map for next-state bit

Count (C_1C_0) \ Execute/Current State	00	01	11	10
00	0	0	1	1
01	x	1	1	x
11	x	1	1	x
10	x	1	1	x

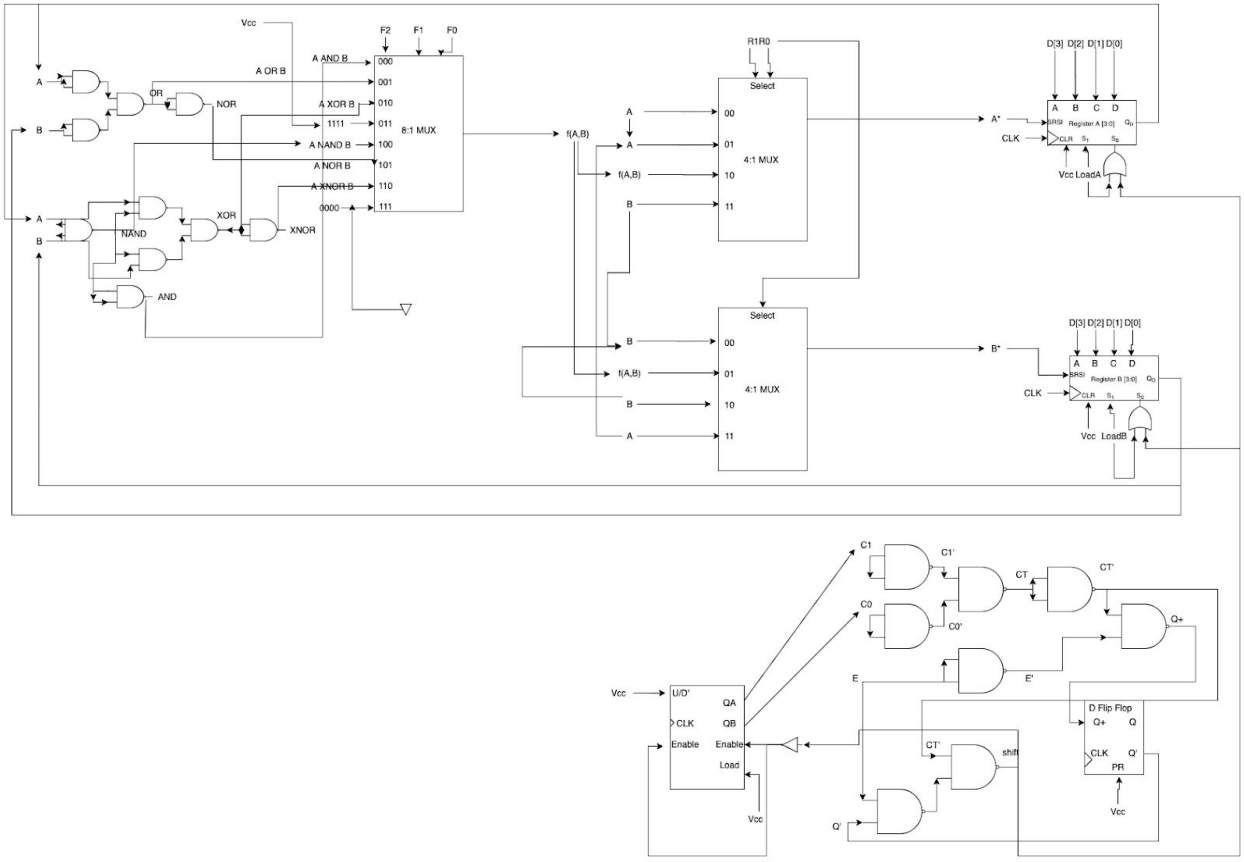
K-map for shift bit

Count (C_1C_0) \ Execute/State	00	01	11	10
00	0	0	0	1
01	x	1	1	x
11	x	1	1	x
10	x	1	1	x

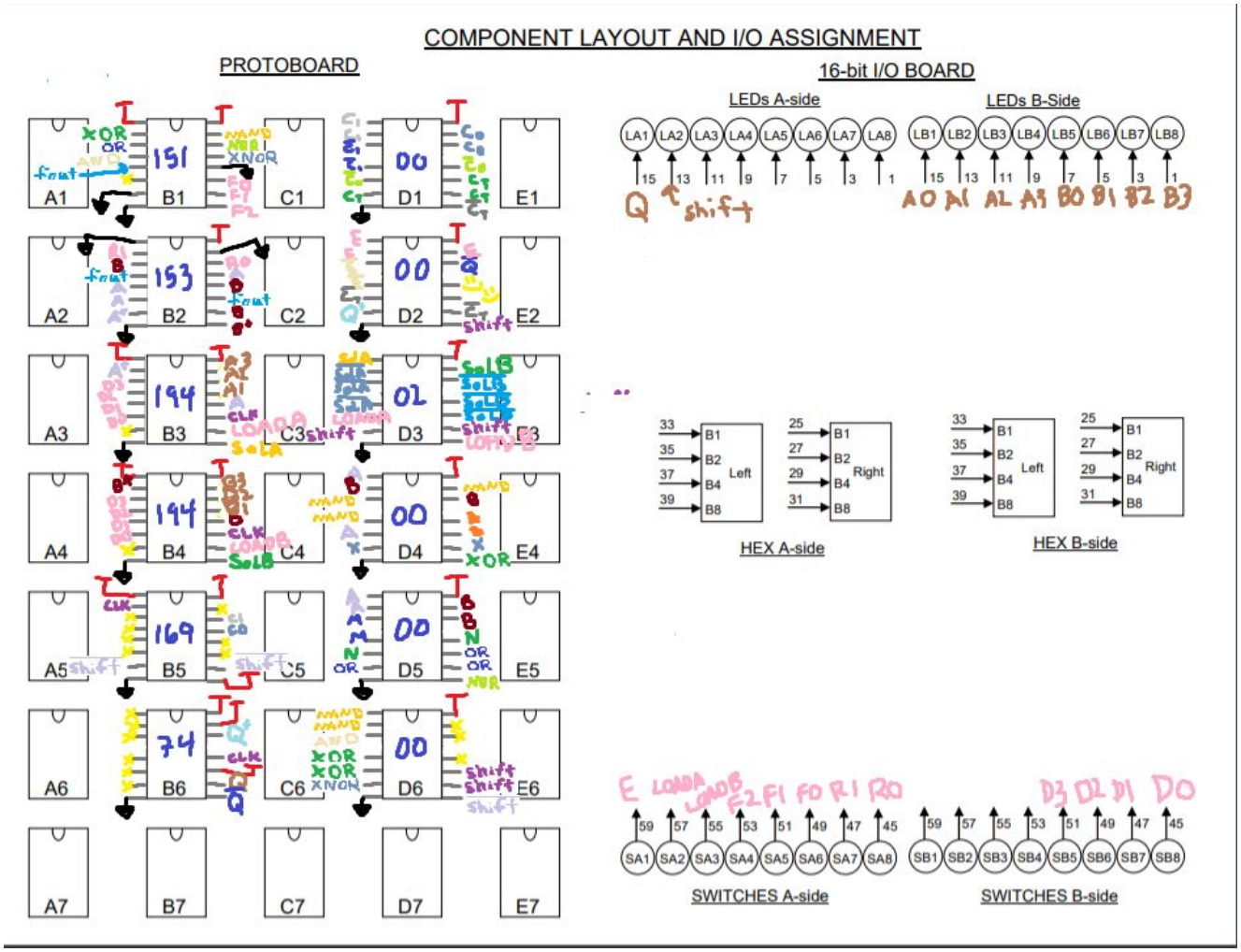
   ii. We decided to avoid gating the clock in place of using the inhibit pins on the relevant chips to halt the circuit. This made it easy to avoid state and shifting errors, and only took a few chips worth of gates. We decided to build a mealy machine because it was easier to build and we could use the states as inputs, as opposed to the more complex circuit implementation of the moore machine.

- Detailed Circuit Schematic
   i. We used the high level circuit diagram given in the lab instructions, and altered in our high level one. Below are the control unit, routing unit, and computation unit.

Combined Circuit Diagram



Layout Sheet



Bugs encountered and corrective measures

The most significant bug we ran into was trying to use the switchbox to manually control the clock. We thought there were issues with our routing switch but it turns out the switchbox did not provide voltage consistently. When we used the function generator we found that everything worked. Other than that, a lot of wires came loose while trying to connect other components.

Conclusion

This lab, during the design process, was very challenging. We thought we had a working design more than 3 times before realizing some circuit-breaking issue, and when we came up with our final design (which coincided with the lecture notes) it was satisfying to run a successful test. The actual testing part was frustrating. The modular design made testing easier but was still frustrating due to the switchbox being inconsistent when used for the clock. We got different errors doing the same tests and it wasn't until the TA suggested using the function generator that we realized that our circuit worked. This took an hour to achieve, and was not a rewarding time.

Post-Lab:

Q: Discuss the design process of your state machine, what are the tradeoffs of a Mealy machine vs a Moore machine?
A: The Mealy machine is more complex and requires more thought to build and implement, but the Moore machine requires more components and wires, which makes it trickier to debug. Since the logic and states of the Mealy machine were provided in the lecture slides we decided to go with that one as the remainder of its use would be significantly more trivial.