ECE 385

Fall 2019

Experiment #3

A Logic Processor

Xinglong Sun Churan He
ABD T 8AM
Mihir Iyer

Introduction (High level function of the circuit) Alex

a. High-level function the circuit performs

The circuit acts as a bit-serial logic operation processor. It's able to perform one of eight logical operations(AND,OR,XOR,1,NAND,NOR,XNOR,0) on 4-bit words serially and route the results of those operations in four different ways. Major components of our design include two 4-bit shift registers, several multiplexers, different kinds of logic gates to perform operations, and counter. The control unit of the circuit is essentially a finite state machine, controlling the shift and load of the registers.

b. Answers to prelab questions

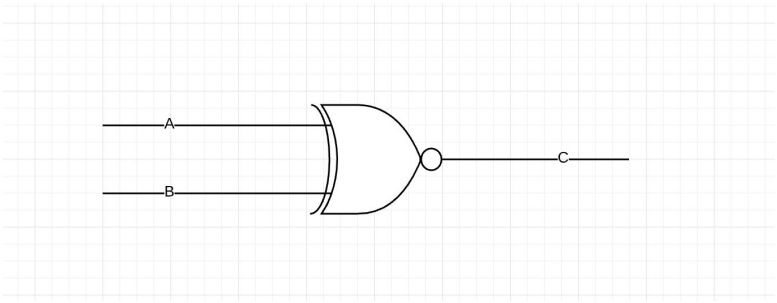
1. Simplest circuit that can optionally invert a signal.

Suppose A is our data signal, B is the signal to choose whether we want to invert A, and C is the output signal, the truth table can be shown as below:

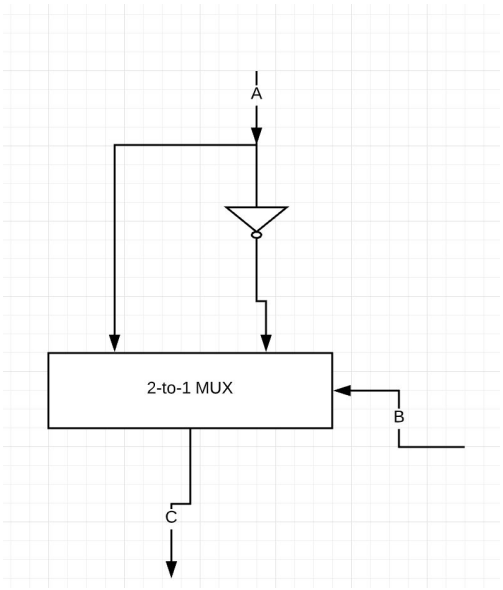
A	B	C
0	1	1
0	1	0
1	1	1
1	0	0

When B is equal to 0, A is inverted.

Without the use of a K-map, we can easily observe that $C = A'B'+AB$. More concisely, C = A XNOR B.



Though XNOR or XOR gate is the simplest way to achieve the optionally-inverting circuit, the circuit that is easier to think of is using a NOT gate along with a 2-to-1 MUX, as shown below:



This is also the circuit we using in our circuit. However, the first circuit(XNOR) described above is more superior and should have been used.

2. The concept of modular design is very important and should be carried out by digital designers while building and debugging the circuit. Since the entire circuit is generally very complex to analyze, it's crucial to figure out a way to logically break up the circuit into different units. While we building our circuit, we move to build the next unit of circuit only after we can ensure that the current unit we're building is free of any bugs. It greatly reduces our development time and reduces the occurrences of indiscoverable bugs.

Operation of the logic processor Alex

There are thirteen user input switches in our circuit. They are Reset, LoadA, LoadB, Execute, D3-D0, R1, R0, F2, F1, and F0. How to operate these switches is specified below.

a. Sequence of switches the user must flip to load data into the A and B registers

   The load of data of our registers is achieved by the parallel load capability of SN74LS194 shift-register chips. Moreover, Register A and Register B share the

same input data sources. The user first toggle the D3-D0 switches to the data values he/she wants to choose for Register A/B. Then, the user can toggle the Load A/ B to store the data into Register A/B. Note that at this time, user can still toggle the data input switches to change the values inside the register. We have LEDs that display the register values consistently for users' conveniences. Once the user finally determines the values he/she wants to load into Register A/B, user can toggle the Load A/B back to off. Then, repeating the same process described above by moving Load B/A will save the data values into Register B/A.

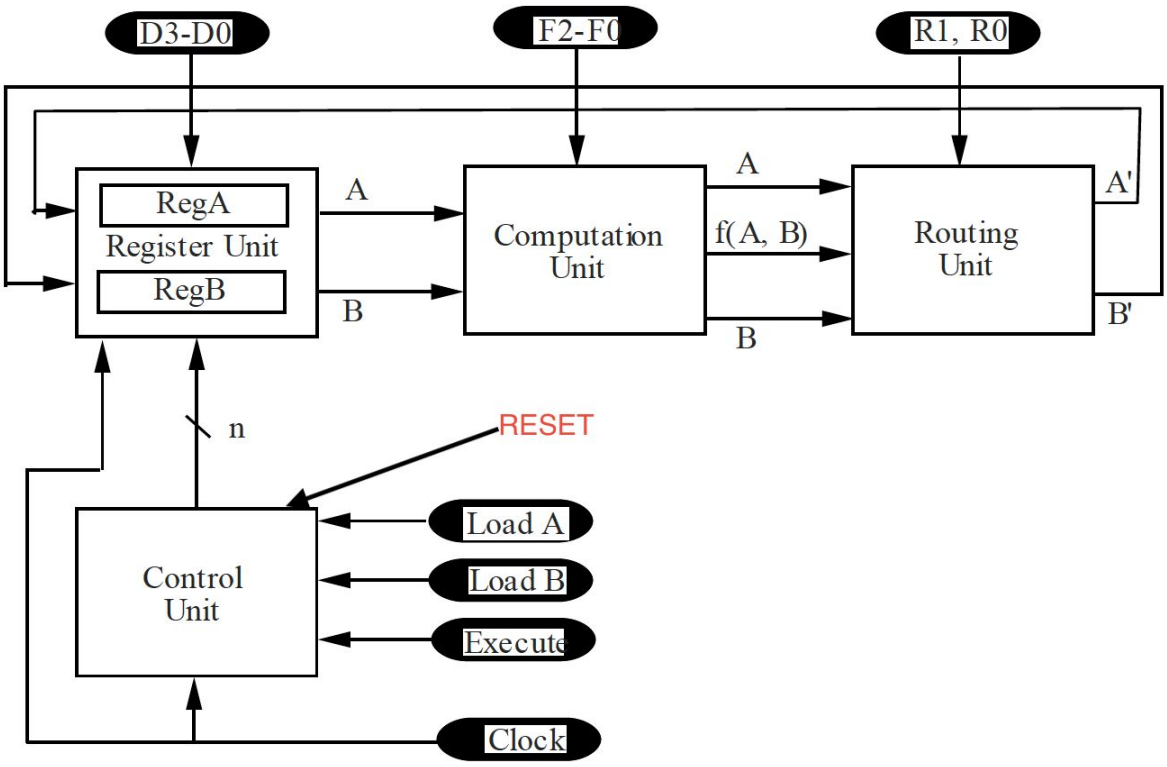
b. Sequence of switches the user must flip to initiate a computation and routing operation.

   Suppose that the user has already completed the part-a above (load the data). The first step the user should perform is to toggle the RESET switch. The RESET switch will reset the state of our circuit and prepare it for the new operations. The details of how we implement the RESET switch is described in details in the following sections. Then, the user can choose what logical operation he/she wants to perform and the routing option by moving the F2-F0(Function choose) and R1-R0(Routing choose) switches. Once the user toggle the Execute switch, the new values for Register A and Register B will be shown on the LEDs after four clock cycles. If the user happens to toggle the Execute switch back to off during the time of the operation, the circuit will still complete the current operation. After the logical operations finished, the circuit will be in a WAIT state waiting for the user to toggle the Execute switch back to off and start some new actions.

Written description, block diagram and state machine diagram Alex

a. Written description & high-level block diagram

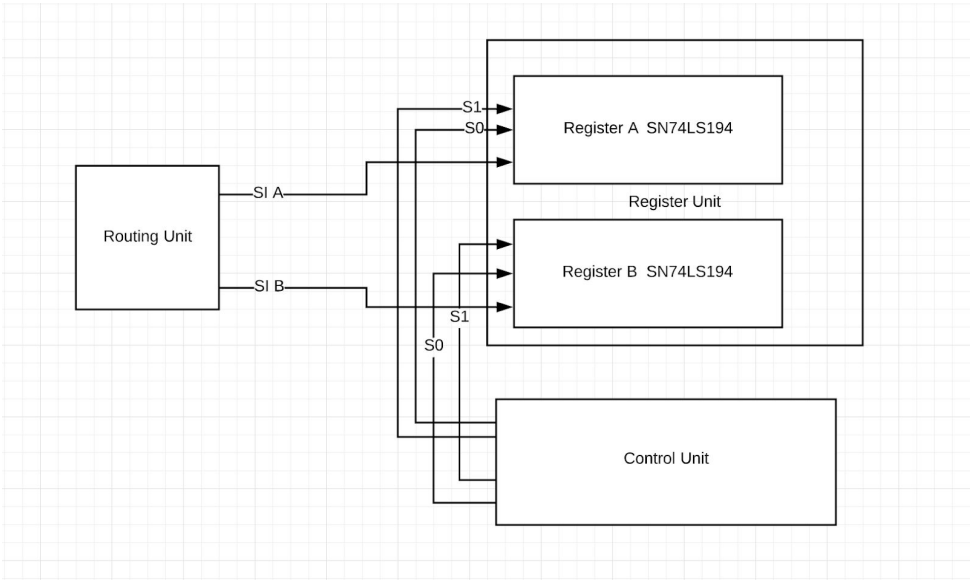
The circuit contains four high level blocks or modules : Register Unit, Computation Unit, Routing Unit, and Control Unit.



High-level Block Diagram of the entire circuit

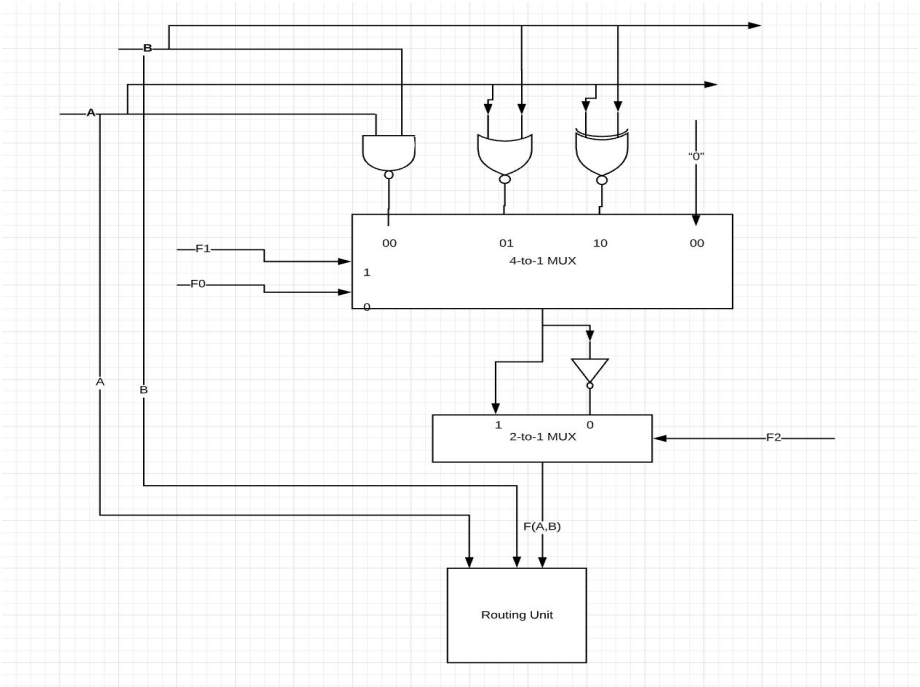
1. Register Unit:

The register unit is the main storage unit in our circuit and is made up of two 4-bit shift registers (SN74LS194). They will store the values used for logical operations and the results of the operations. There are four LEDs displaying the outputs of Register A and four LEDs displaying the outputs of Register B. The serial inputs of the registers are provided by the Routing Unit, and the control signals for the registers are provided by the Control Unit. In particular, shift signals S1 and S0 on the chip are controlled by the Control Unit, which determines when the register parallel loads, shifts right, or does nothing.



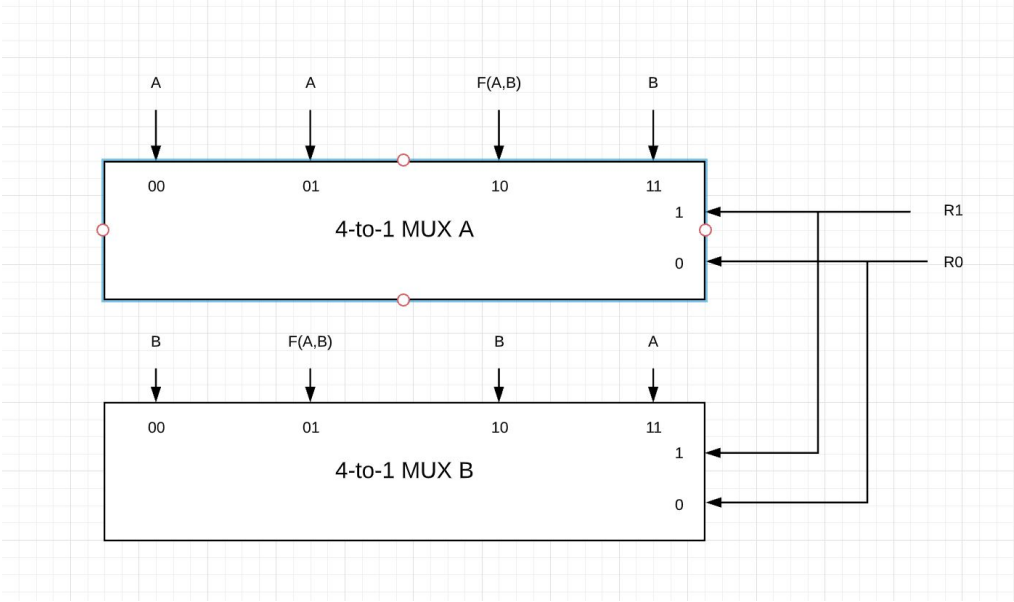
2. Computation Unit

The computation unit performs the actual 8 possible logical operations specified by the user. It's made up of various kinds of logical gates, a 4-to-1 Mux (SN74LS153), and a 2-to-1 Mux(SN74LS157).



3. Routing Unit

   The routing unit selects new A and new B, which will be fed into the serial inputs of Register A and Register B, from A, B, and F(A,B). Again, we use two 4-to-1 MUXes(SN74LS153) to achieve this.



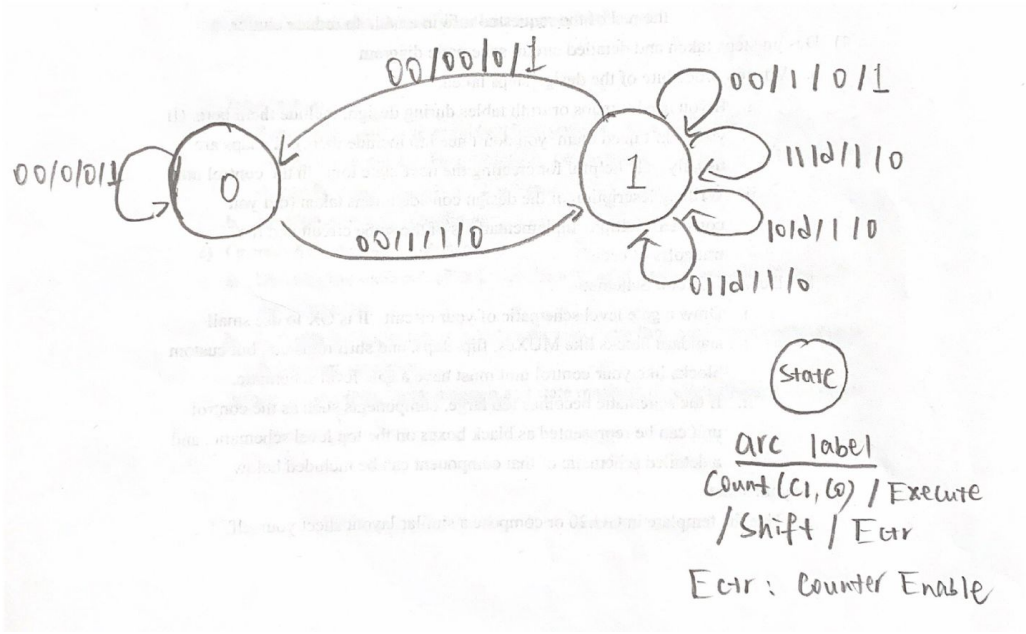
4. Control Unit

   The control unit accepts the following inputs : Load A, Load B, Execute, and the clock signal. The main component of the control unit is a finite state machine, which generates a shift signal indicating when the registers should do serial right shift. The shift signal is then fed into a simple combinational logic with Load A/B to control the S1 and S0 of Register A/B. It's made up of counter(SN74LS169), flip-flops(SN74LS74), and various kinds of logic gates for output and state transitions.

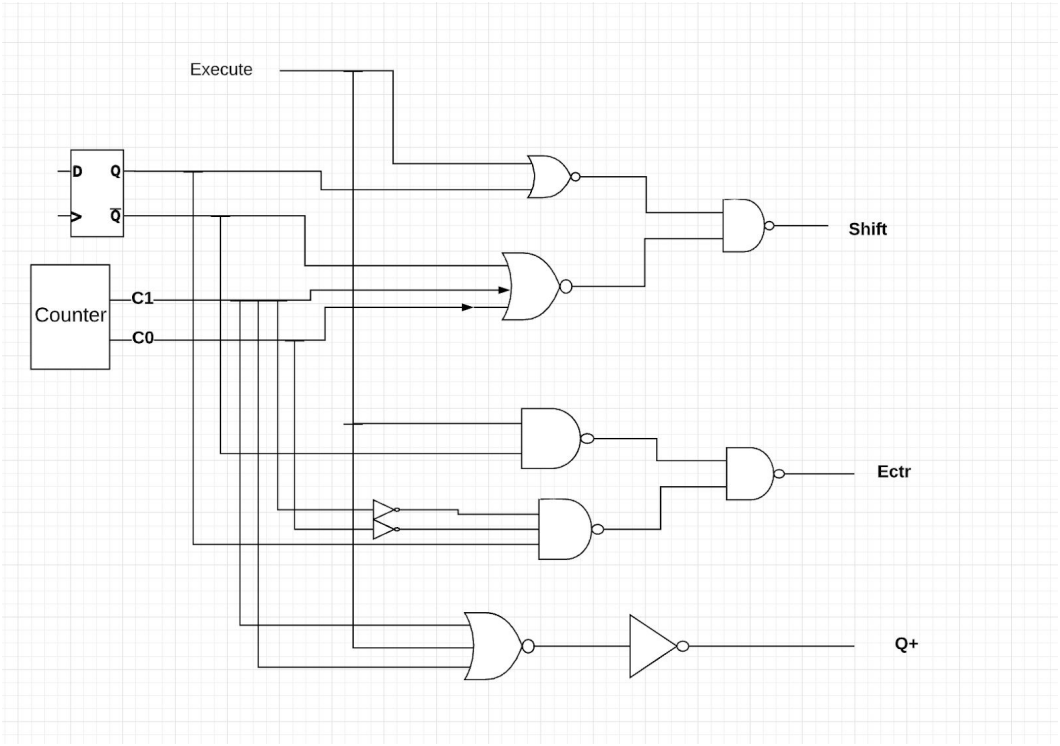
   State Machine:

   We use a Mealy State Machine with two states. Why we weigh Mealy State Machine over Moore State Machine is explained in detail in the Design Consideration section. The two states of the FSM are identified as State 0 and State 1. The inputs of the FSM include the counter values (C1, C0) and the Execute signal. They determine the outputs of the state machine, which are the Shift signal and the Counter Enable signal. The Counter Enable signal is used to control when the counter counts up or keeps its current values. Particularly, the

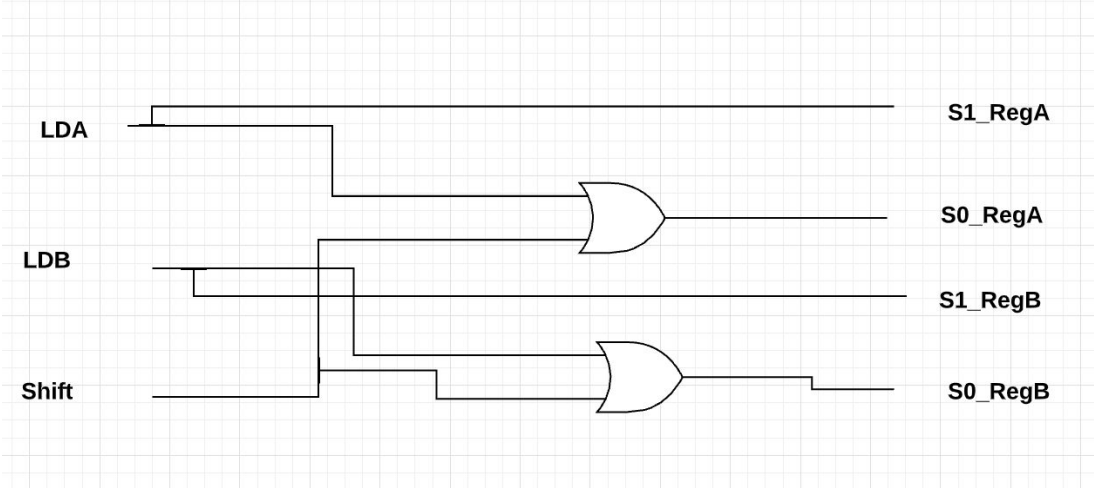
Counter Enable connects to the Enable P and Enable T Pins on the SN74LS169 chip. A state machine diagram is shown as follows:



The gate-level circuit is as follows:



However, not only the shift signal controls the shift of the register. Load A/B could also control the shift of the registers. Therefore, simple combinational logic needs to be devised. By analysis, the circuit is as follows:



Design steps taken.

We start the project from building the units without the use of clock signal, which are the Computation Unit and Routing Unit. Given the truth table of the Computation as follows:

Function Selection Inputs			Computation Unit Output
F2	**F1**	**F0**	**f(A, B)**
0	0	0	A AND B
0	0	1	A OR B
0	1	0	A XOR B
0	1	1	1111
1	0	0	A NAND B
1	0	1	A NOR B
1	1	0	A XNOR B
1	1	1	0000

,the use of MUXes comes very intuitively. We choose to build a half part of the logic (NAND, NOR, XNOR, 0000) and use the MSB of F (F2) to optionally invert the signal.

The optional inverting is achieved by a 2-to-1 MUX. However, we have noticed that a XNOR gate is simpler to build and should be considered as a superior replacement.

After we test the Computation Unit, we move to build the Routing Unit. Given the truth table of the Routing Unit as follows:

Routing Selection		**Router Output**	
R1	**R0**	**A'**	**B'**
0	0	A	B
0	1	A	F
1	0	F	B
1	1	B	A

,we also use MUXes to achieve the routing selections.

After we ensure that the Routing Unit works normally with the Computation Unit, we move to design and build the Control Unit, which is the hardest part of this lab.

We choose to use FSM as the main component of our control unit. Again, why we weigh Mealy State Machine over the Moore State Machine is explained in the following Design Consideration section.

In order to transform a state machine into a physical circuit, we need to build a state transition table, which is shown as follows:

Execute	Q	C1	C0	Shift	Q+	Ectr
0	0	0	0	0	0	1
0	0	0	1	d	d	d
0	0	1	0	d	d	d
0	0	1	1	d	d	d
0	1	0	0	0	0	1

0	1	0	1	1	1	0
0	1	1	0	1	1	0
0	1	1	1	1	1	0
1	0	0	0	1	1	0
1	0	0	1	d	d	d
1	0	1	0	d	d	d
1	0	1	1	d	d	d
1	1	0	0	0	1	1
1	1	0	1	1	1	0
1	1	1	0	1	1	0
1	1	1	1	1	1	0

Kmaps are used are to derive boolean algebraic expressions of output and next state:

“S”

	C1C0				
EQ		**00**	**01**	**11**	**10**
	00	**0**	**d**	**d**	**d**
	01	**0**	**1**	**1**	**1**
	11	**0**	**1**	**1**	**1**
	10	**1**	**d**	**d**	**d**

S = (E + Q) * (Q’ + C1 + C0)

Q+

	C1C0

EQ		00	01	11	10
	00	0	d	d	d
	01	0	1	1	1
	11	1	1	1	1
	10	1	d	d	d

Q+ = E + C1 + C0

ECtr

	C1C0				
EQ		**00**	**01**	**11**	**10**
	00	**1**	**d**	**d**	**d**
	01	**1**	**0**	**0**	**0**
	11	**1**	**0**	**0**	**0**
	10	**0**	**d**	**d**	**d**

ECtr = E'Q' + C1'C0'Q

As described in the above sections, simple logic of shift and Load A/B needs to be devised here to determine S1 and S0 of the registers. We build a simple truth table to analyze:

S	Load A/B	S1	S0
0	0	0	0
0	1	1	1
1	1	d	d
1	0	0	1

We assume that Shift and Load A/B are never simultaneously open, therefore we use don't-cares to simplify the logic.

By direct observation, we get:

S1 (Reg A) = Load A S0 (Reg A) = Load A + Shift

S1 (Reg B) = Load B S0 (Reg B) = Load B + Shift

Design considerations taken

1. Why we use Mealy State Machine instead of Moore State Machine

The outputs of the Mealy State Machine depend on both the current state and the inputs; however the outputs of the Moore State Machine depend solely on the current state. In Mealy State Machine design, counter's output and the execute signal control the shift output in the Shift/Hold State. Instead, if we use Moore State Machine, each state can only have a specific output, thus the Shift/Hold State need to be split into two states. From this point of view, using the Mealy State Machine in this case can save one state, which in turn could also simplify transition logics and improve system overall reliability.

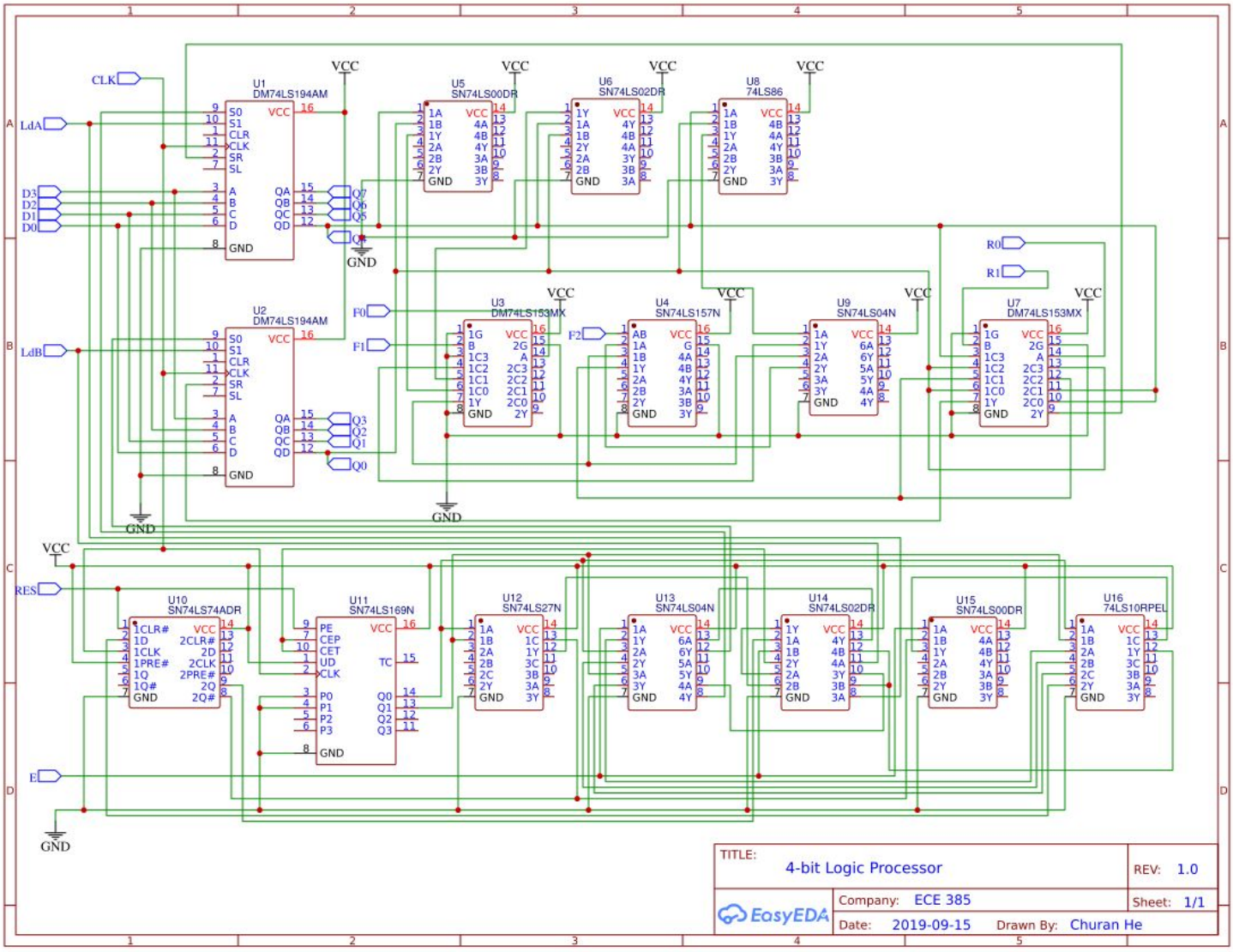
2. Why we use a RESET switch

A reset switch can ensure the initial value of the flip-flop that stores the current state is 0, and the initial value of the counter is 00. This switch is important because floating values within these two components will cause the Mealy State Machine working inappropriately, like undefined input for the current state or entering the Shift/Hold State and execute right after power on. The RESET switch can always make sure our machine would strictly follow our design and start at the initial state.

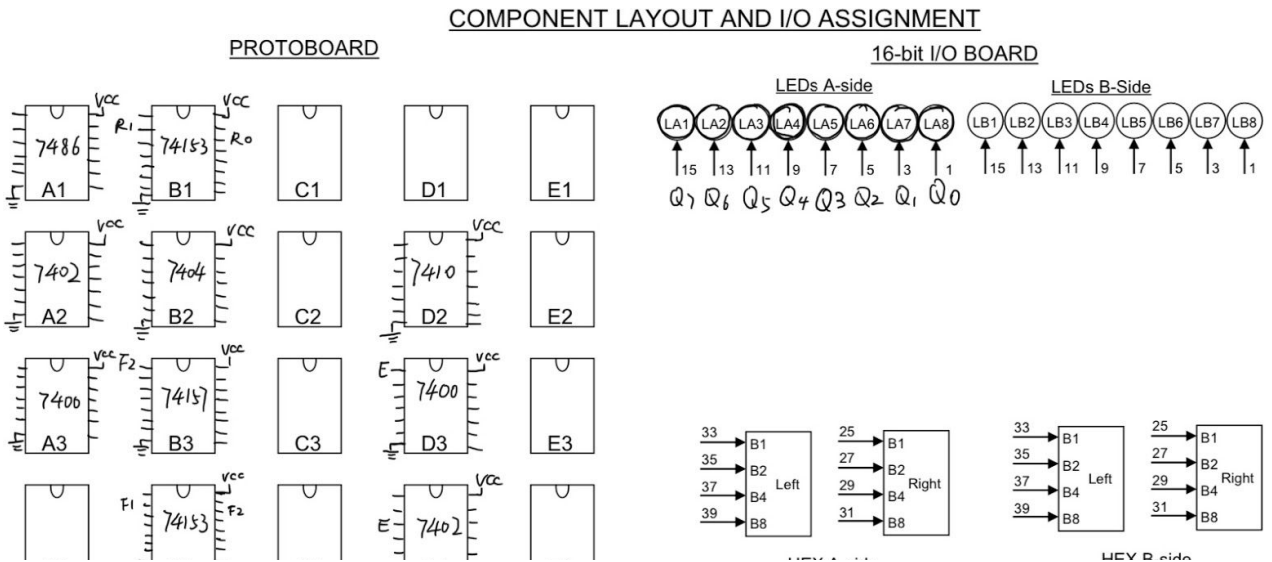
3. Why we use counter instead of two flip-flops

Another option instead of using a counter inside our state machine is to treat C1, C0 as two additional states and build our custom counters. However, this option uses more gates, and our option could effectively break the state machine into logical parts for both building and debugging.

Detailed Circuit Schematic



Component Layout sheet



Bugs encountered Alex

1. Wrong setting of the function generator

   A very strange bug we encountered during the lab session is that our circuit worked perfectly when we used the debounced switch to mimic the clock signal but failed when we used the function generator to produce the clock signal. It took us a long time to realize that we didn't set an offset for the function generator. Therefore, with amplitude as 5V and offset as 0V, the square wave signal actually goes from -2.5V to 2.5 V rather than 0V to 5V as expected. After we add a 2.5 V offset to it, the circuit is back to normal. A very interesting thing is that we also didn't set the correct offset for Lab2 while using the function generator, however, the Lab2 circuit performs normally. After a discussion with TA and online research, we found that lots of the IC chips are of very complex design, and some of them even include tiny capacitors to charge other electronics. In Lab2, our circuit contains relatively fewer components, therefore a 2.5V is enough to raise the clock signal. However, in Lab3, our circuit contains lots of parts and chips, a 2.5V is no longer enough to charge the capacitor while circuit performs at a high frequency.

Conclusion

We successfully accomplished the goals for this lab. We got more familiar with the mealy state machine and how to use FSM as the main component of our control unit. We also got more familiar with modular design as we built and tested our circuit unit by unit. With experiences and lessons obtained from previous lab sessions, we finished the design and the building of this lab very quickly.