ECE 385

Fall 2019

Experiment #3

A Logic Processor

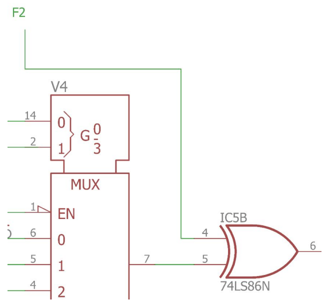
Eric Dong, Yifu Guo
ericd3, yifuguo3
Section AB3, Thursday 2pm
Gene Shiue

Introduction

In this lab, we built a 4 bit processor that is able to carry out bitwise operations on 2, four bit bit strings. The processor is able to do AND, NAND, OR, NOR, XOR, NXOR, clear to 0s and 1s. It operates on 2, four bit bit strings and stores the computed value, or not, into a register that the user chooses. Also the user is able to choose the type of computation executed. The circuit performs 1 single operation and waits until the next command for execution comes in.

Pre-lab Questions

1. We have concluded that the simplest circuit that allows optional inversion is by using a XNOR or a XOR gate. Since (for XOR) if the select input is 0, then the other value would be passed through, and when the select input is 1, then the other input pin would be inverted at the output.



2. With such a design mentioned above, it could help us greatly with modular design. In the case of lab 3, it allows us to only use a 4:1 MUX for doing the computation, and invert the signal as needed according to F2. For lab 3, we instead chose to use a XNOR gate. This allows to use just 3 types of gates(NAND, NOR, XNOR) as well as a LOW pin to do all the computations required for the lab. This way, we wouldn't have to use the other gates such as AND, OR, and XOR. This saves wiring time, designing time and also helps us debug since we wouldn't have to test as many gates when we debug.

Operation of the logic processor

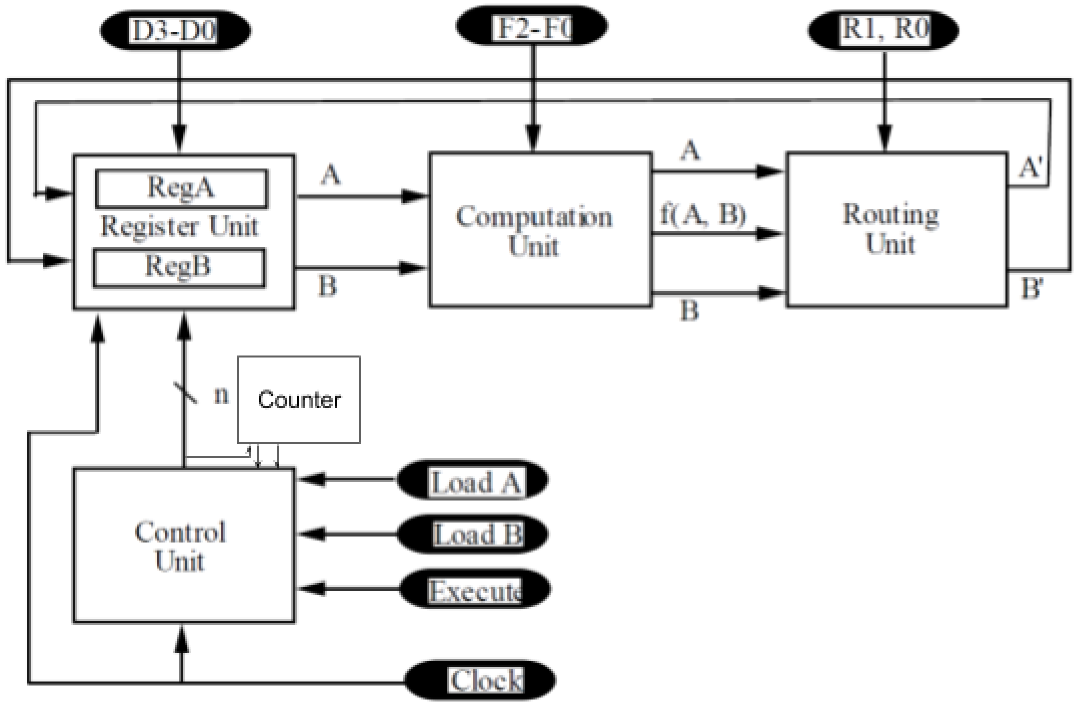
1. In order to load data into register A and B, we would first have to set the 4 data pins (DIN). After setting the DIN pins to the desired value, we can then turn on the load A or load B switches while making sure that the execute pin remains off. After setting one of the registers, we then turn off that load switch, set new values to the DIN switches and load the value into the other register.
2. In order to initiate a computation and routing operation, we would need to first determine the routing first, we can set R1 and R0 according to the table provided to set A' and B' accordingly. Next, we have to pick an operation, we can do so by setting the F2, F1, F0 switches according to the operations table provided as well. This determines the kind of

operation that would be carried out. Finally, we turn on the EXECUTE switch and as the clock progresses, A' and B' would change accordingly to what we chose.

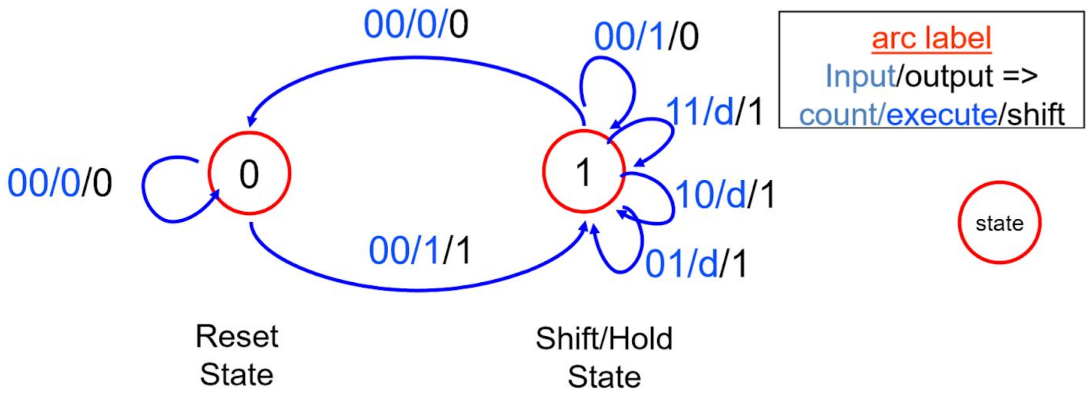
Written description, block diagram and state machine diagram of logic processor

1. Written Description
    a. Register Unit
    The register unit consists of 2 shift registers that takes in a shift signal from the control unit which tells the unit whether to shift or not. One of the shift registers holds the value of A and the other register holds the value of B. We can load custom values separately to each of these shift register units, and when execute is on as well as the shift signal, the shift register shifts data out serially to the computation unit.
    b. Computation Unit
    The computation unit consists of various gates to do the different bitwise computations needed by the user. It takes in two values from A and B registers and then the user selects the type of operation to be performed. The unit also has a 4:1 MUX as well as an XNOR gate to implement the 8 different choices of operation.
    c. Routing Unit
    The routing unit takes in 3 different values, A, B, and F, the answer output from the computation unit. Then, it is also connected to the shift registers to shift in the values that the user wants. The unit consists of a 4:1 MUX that has 2 user inputs which allows the user to pick what to store into shift register A and B. The user has the ability to either store A, B or F into either of the two registers. The choices are AB, AF, FB, BA respectively.
    d. Control Unit
    The control unit is a finite state machine that controls whether the shift register should keep on shifting or not. The control unit takes in 3 inputs, load A, load B, as well as execute. When load A or load B is 1, the respective shift register is loaded with the value of the DIN switches. Then when execute is 1, the control unit allows 1 computation to happen and stops when 4 bits have been shifted in and out of the shift register. The control unit is also connected to a counter which tells when control unit when to stop. We decided to use a 74ls169 to simplify the design as well as reduce the debugging time required.

2. Block Diagram



3. State Machine Diagram
    a. For this lab assignment, we decided to use the Mealy machine.



    b.

Design steps taken and detailed circuit schematic diagram

1. K Map

S	EQ 00	EQ 01	EQ 11	EQ 10
C1C0 00	0	0	0	1
C1C0 01	X	1	1	X
C1C0 11	X	1	1	X
C1C0 10	X	1	1	X

S = C0+C1+EQ’

Q+	EQ 00	EQ 01	EQ 11	EQ 10
C1C0 00	0	0	1	1
C1C0 01	X	1	1	X
C1C0 11	X	1	1	X
C1C0 10	X	1	1	X

Q+ = C0 + C1 + E

C1+	EQ 00	EQ 01	EQ 11	EQ 10
C1C0 00	0	0	0	0
C1C0 01	X	1	1	X
C1C0 11	X	0	0	X
C1C0 10	X	1	1	X

C1+ = C1’C0 +C1C0’ = C1 XOR C0

C0+	EQ 00	EQ 01	EQ 11	EQ 10
C1C0 00	0	0	0	1
C1C0 01	X	0	0	X
C1C0 11	X	0	0	X
C1C0 10	X	1	1	X

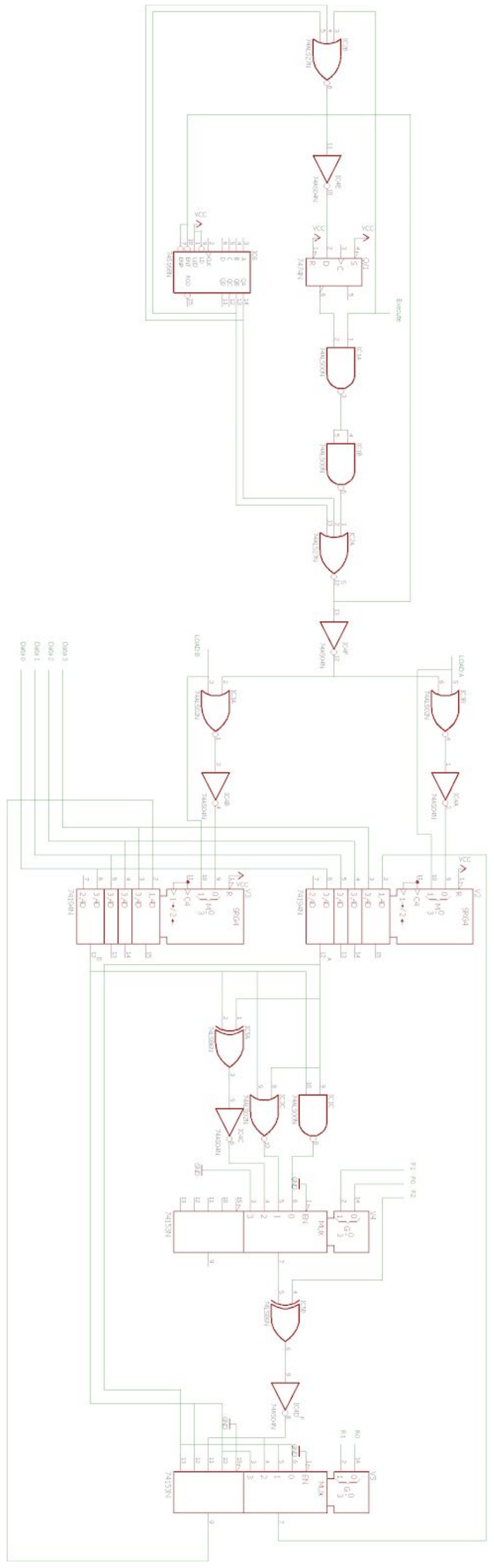
C0+ = C1C0’ + EQ’

2. Truth table

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

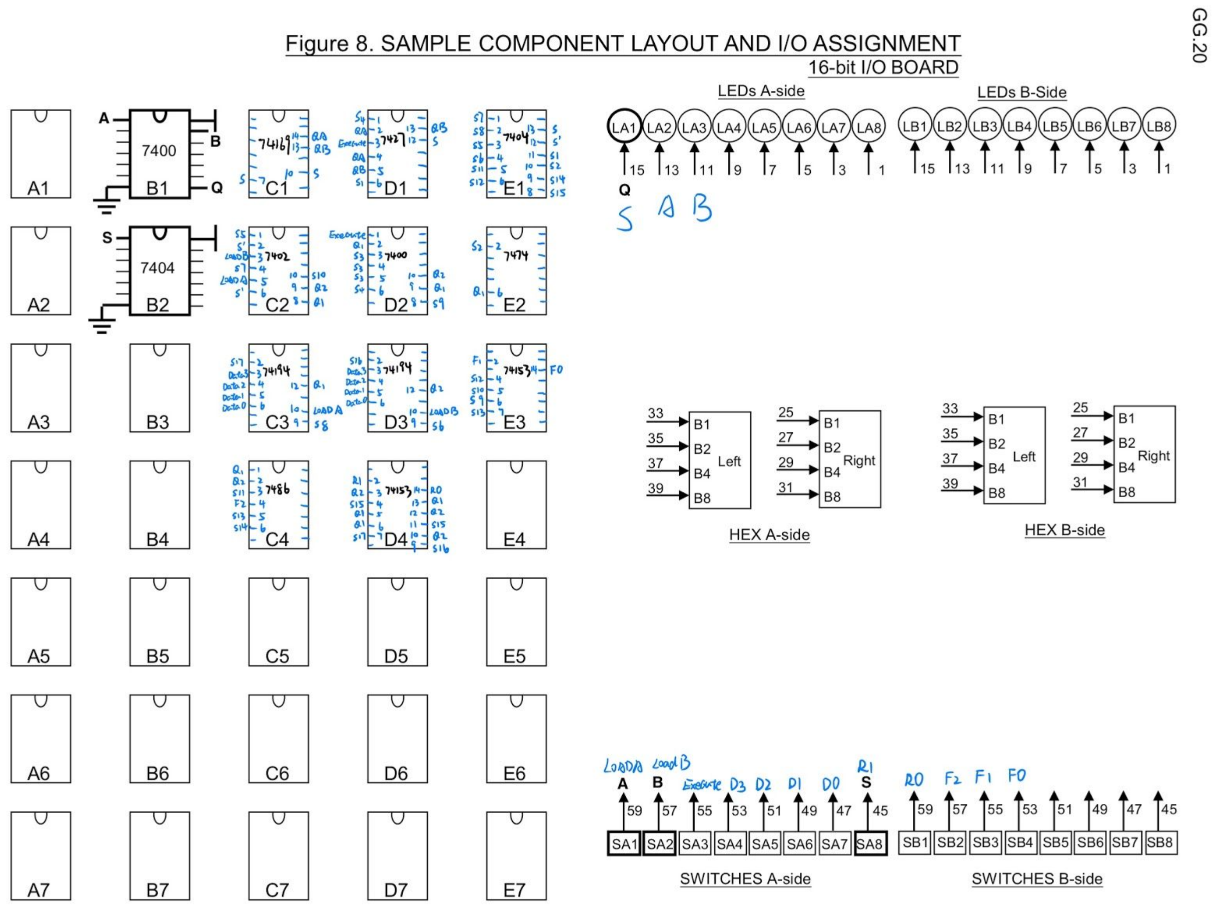
3. During the designing process, there were many options that we considered, and we decided to go with the one we have right now. For the counter, we had the option to either have an addition 2 flip flops or to just use a 169 counter instead. If we use the 169 counter, the circuit for the finite state machine would be much simpler but we would have to think of a way to incorporate the timer with the FSM. Ultimately we decided to go with the 169 since it would allow us to save a lot of wiring time as well as potential debugging time.
   Another design decision that we had to make was choosing to either use an XNOR gate or an XOR gate to optionally invert the signal that comes out of the computation unit. By using an XNOR gate, we would have to use an extra inverter but the input to the 4:1 MUX would only require NAND, NOR gates, instead of 2 additional inverters had we gone with the design with an XOR gate to optionally invert the signal.

4. Detailed Schematic



Layout sheet

1.



Description of all bugs encountered, and corrective measures taken

1. During the writing process, we took a step by step approach to debugging: whenever we finished building a component, we would debug it by giving it test inputs and also monitor the output so there weren't too many bugs that we encountered with our design. One issue that we ran into was that our 74ls27 3 input NOR gate was faulty, when we tried to give the 3 inputs all 0's the output remained a 0 instead of a 1. We were really confused so we decided to get another from the lab room and replaced it, which worked on first try. So the issue was that the chip was faulty.

Conclusion

1. In this lab, we learned how to design a 4 bit serial processor which does different bitwise calculations. We used shift registers and stored the answer of the computation into either one of the shift registers based on our liking. This lab taught us how a basic serial

processor works as well as how to simplify the design if we see an opportunity such as using a 4:1 MUX instead of an 8:1 MUX.

Post Lab Questions

1. For debugging the lab, we didn't run into too many problems, we just had a faulty 74ls27 chip that gave the wrong value given any input. We simply swapped that chip out and the circuit was functioning correctly. The modular design was helpful in simplifying down the wiring needed for the computation unit. Since the wiring was simpler, it was easier to test that part of the circuit as well. We only had to make sure that 4 of the bitwise operations were functionally correctly and the modular design was right to ensure that the entire computation unit was going to work fine.
2. For a Mealy machine, there are only two states for this lab that we are doing, it is very simple to understand at least what each state means: shift or don't shift. The down side of a Mealy machine is that the wiring of the machine is very complex, it required more components, which is the exact opposite of the Moore machine. The Moore machine needs more states, for example, Moore machine needs a state for each counter increment whereas the Mealy only needs 2 states for shift or no shift. The Moore machine does however allow simpler wiring and component design.