ECE 385

Fall 2021
Experiment #2

A Logic Processor

Wei Jiahao, Qian Liyang
3190112144, 3190110719
D231
TA: Chenhao Wang

1 Introduction

In this lab, we design a circuit that can deal with each bit of the input according to the function signals, $F_2F_1F_0$, and store the result in different registers, R_0R_1, according to the register signal. For example, for input A and B, if $F_2F_1F_0 = 000$ and $R_0R_1 = 01$, then the circuit will process $A \wedge B$ bit-wisely and store the result in register B. The circuit consists of 4 parts, *Register Unit*, *Computation Unit*, *Control Unit* and *Routing Unit*. This circuit operates on 4-bit data because the *Register Unit*, which is exactly the right-shift memory in lab2, can only store 4 bit. We use the ***FSM*** idea to design the *Control Unit* so that it can work very well.

2 Answers to pre-lab questions

2.1 simplest circuit to optionally invert a signal

The XOR gate circuit, with two inputs at the input PIN, IN_1IN_0 of an XOR gate and the output to the output PIN, OUT_0. According to the truth-table below,

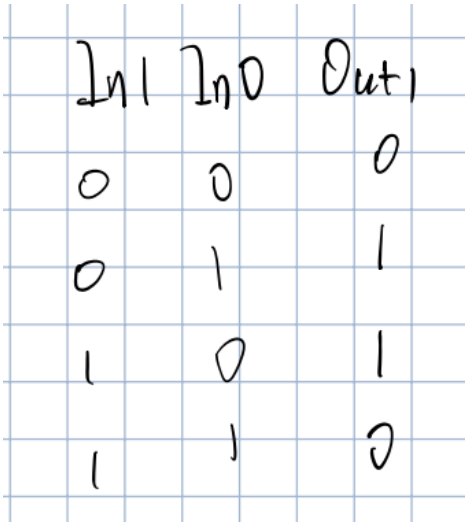

Figure 1: Xor-truth-table

When IN_1 is 0, the output is the original value of IN_0. When IN_1 is 1, the output is the inverted value of IN_0.

2.2 Explanation of simplicity and the approach to troubleshoot

The XOR circuit simplify the design because instinctively, when we consider a function to be optional, we usually tend to design the circuit generating the expected output and the circuit generating the inverted one and then use a 2-to-1 mux to choose, and we need to respectively test these two circuits to make sure they work well. However, when attempting the XOR gate, we only need to test the compliance between IN_0 and OUT_0.

The troubleshooting process is, we need to see (whether the OUT_0 is consistent with the IN_0 when IN_1 is low) $\wedge$ (whether the OUT_0 is the inverted one of IN_0 when IN_1 is high).

2.3 Design, document and build the circuit described in Part II.

Since we finish designing and testing the circuit by *Quartus*, there is not a physical circuit. However, we have simulated the output of our circuit by the form of waveform file. We will upload our *Quarus* project so that everyone is able to run our circuit and see the result. In the *Quartus* project waveform file, the time when *execute* switch is flipped will be shown in the file with the corresponding input signals and monitors for register A and B . The result of our circuit will also be represented by the electrical level signals in waveform file.

3 Operation of the logic processor

3.1 load data into A and B registers:

We need to set Execute to 0 and waiting for 4 clock periods so that the FSM can be in **Reset State**. The ***DIN*** inputs has been connected to the 2 parallel load interface of 2 right-shift memory units. And the Load A, Load B input signals are respectively connected to the operation pins to activate the parallel loading operation.

Then we flip Load A to 1 to load data into A register. And then flip Load B to 1 to load data into B register after the loading data A operation finishing.

3.2 initiate a computation and routing operation:

Firstly, we need to determine what kind of bit-wise operation we want by confirming the value of $F_2F_1F_0$. And we also need to determine the value of routing signals, R_1R_0 to make sure the place to store the result or not.

After setting these two input signals, we need to flip execute signal to high to start computation.

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

The register unit has six 1-bit data inputs. It functions as the storage of the two 4-bit words, and consists of two 4-bit shift register 74194. It has two 1-bit outputs which are fed into the computation unit. It can either load data from the outputs of routing unit, or from the 4 1-bit parallel load inputs D3-D0. The two selection inputs A and B will decide which place (word A or B) the 4 1-bit parallel inputs will be loaded into.

The computation unit takes two 1-bit data input at a time and output both the two 1-bit inputs and the computed result. The function used in computation depends on the selection inputs F2-F0. The outputs are fed into the routing unit.

The routing unit takes three 1-bit data inputs and output two bits, which is fed back to the register unit. The outputs are chosen between the three inputs. The choice depends on the selection inputs R1 and R0.

The control unit manipulates the register unit using its 1-bit output. When the input EXECUTE goes from low to high, the control unit will manipulate the register unit to shift four times to perform a desired computation.

We used a Mealy machine. The binary flip-flop value is the number inside the red circle.

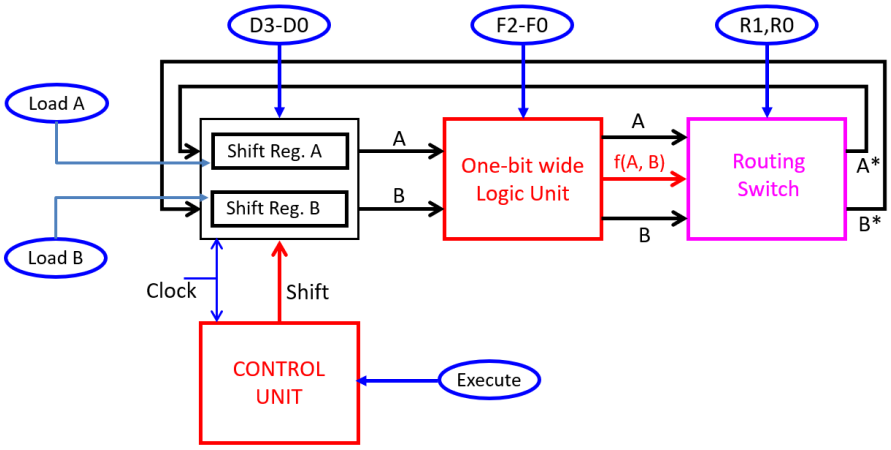

Figure 2: High-Level Block Diagram

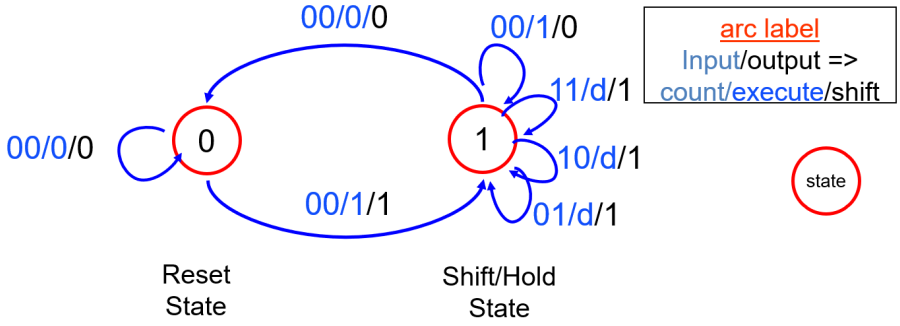

Figure 3: State Machine Diagram

5 Design steps taken and detailed circuit schematic diagram

The design step is to map the state machine diagram into a simulation circuit. To achieve this, we draw a k-maps basing on the truth table given in the lecture. K-maps helped us create the next-state logic in the control unit.

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

Figure 4: Truth Table

In the truth table, S indicates whether to shift the register, where 0 for not to shift and 1 for shift. Q+ indicates the next state value. P' is used to control the counter either to stop or continue counting.

EQ

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

S = EQ' + C0 + C1

	00	01	11	10
00	0	0	1	1
01	x	1	1	x
11	x	1	1	x
10	x	1	1	x

Q+ = E + C0 + C1

	00	01	11	10
00	1	1	1	0
01	x	0	0	x
11	x	0	0	x
10	x	0	0	x

P' = (E' + Q)C0'C1'

Figure 5: K-maps

The S+ can not be directly used to control the register unit. It must be combined with other logic gates to be consistent with the data sheet of 74194.

The following figures are our detailed circuit schematics.

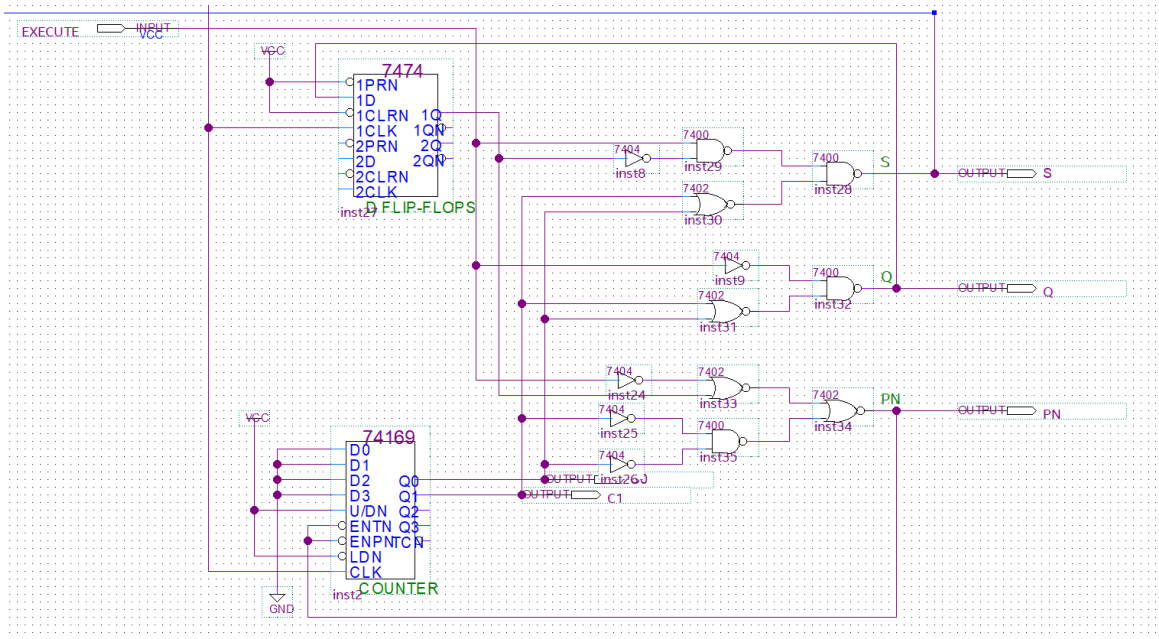

Figure 6: Circuits for State Machine

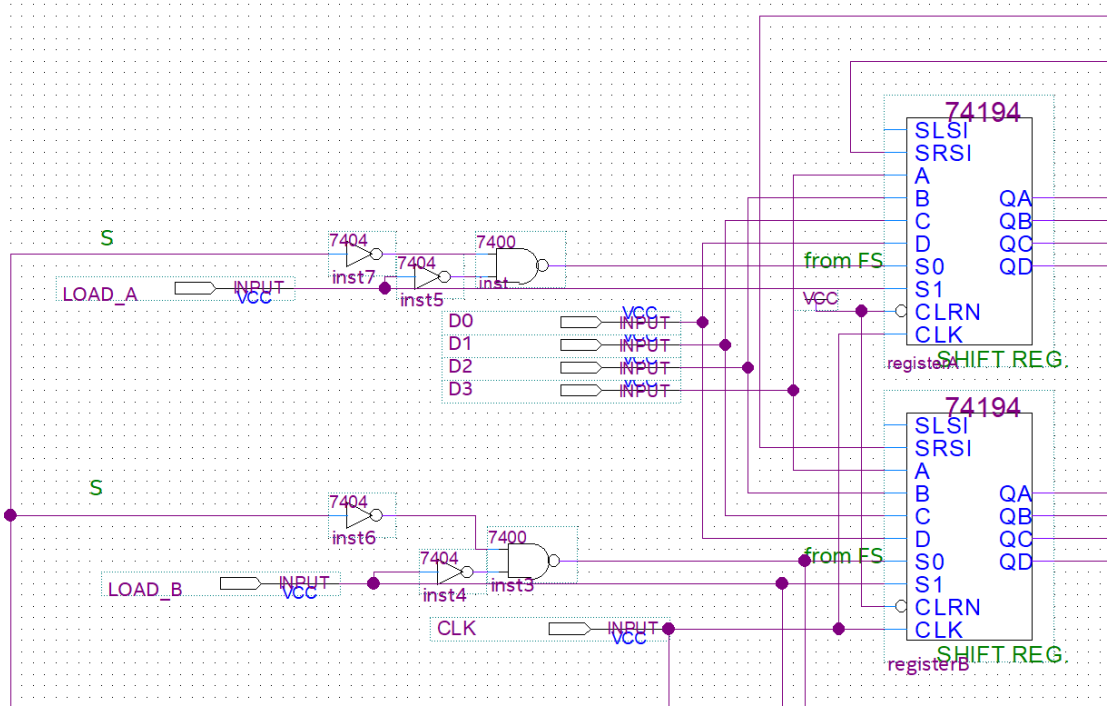

Figure 7: Control Logic for S and Load

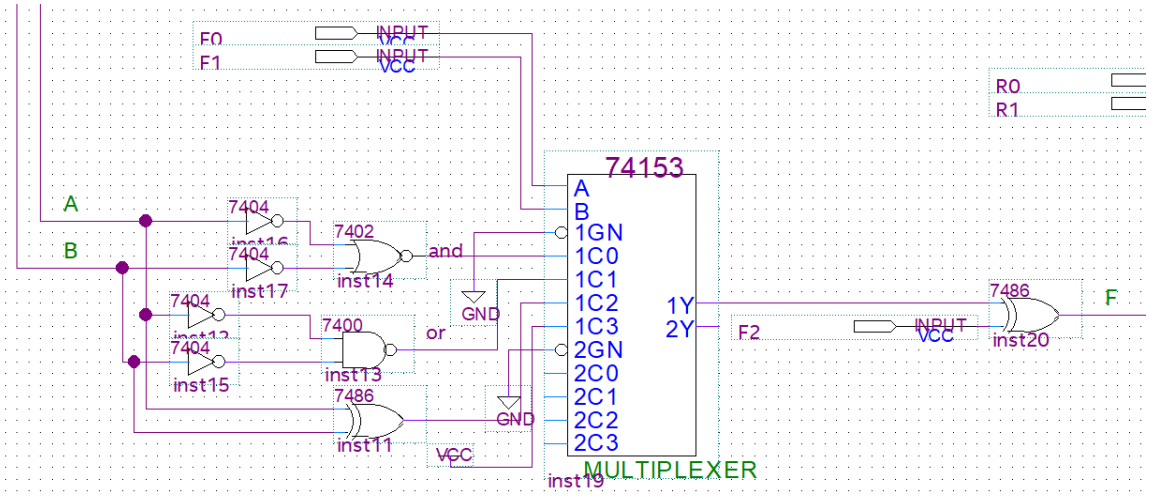

Figure 8: Computation Unit

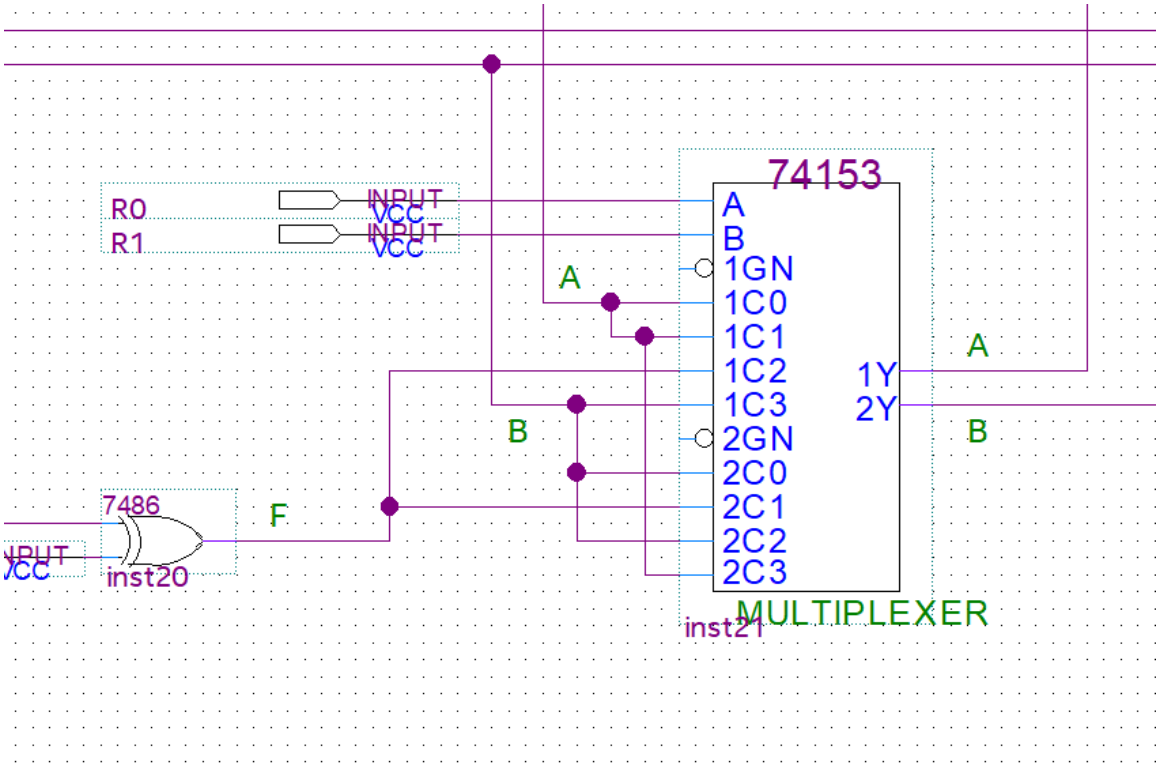

Figure 9: Routing Unit

6 Description of all bugs encountered and corrective measures taken

The bug happens when we try to do a optimization on the circuit. It is, we reverse the output sequence of the 2 4bits right-shift memory we forget to flip the sequence of input signal, which caused a problem that the $DIN[0]$ will appear at $A[3]$.

7 Conclusion

This lab is impressive because this is the first time we build a circuit which performs like a FSM as we designed on the lecture. There are lots of model which can fit in the FSM model in daily life. Therefore, this lab greatly enlarge our ability range to solve real questions.