ECE 385

EXPERIMENT #3

A Logic Processor

I. OBJECTIVE

In this experiment, you will design and build a bit-serial logic operation processor. The design will utilize two 4-bit shift registers, several multiplexers, and some type of counter. The circuit will be capable of calculating eight different functions and routing the results of those operations in four different ways. A finite state machine will be implemented to serve as the control unit of the circuit.

II. INTRODUCTION

In the experiment on storage, shift registers were used to store data by shifting a bit out of one end of the shift register and then shifting the same bit into the other end. However, when a write operation was performed, a new bit was shifted into the register. Taking this process one step further, we will modify, not one, but all the data bits in the register using a designated logical function. The circuit will provide the capability of bit-wise logical operations. We only want to perform the logical function once on each bit of the data, so we will also need to keep the data from circulating more than once through the circuit.

The operations that this circuit will perform are similar to the bit-wise logical functions provided in machine level programming. For example, your circuit will be able to perform a bit-wise AND of the two operands that are stored in the two registers (RegA and RegB) and be able to place the result in either of the registers, leaving the other register unchanged in the process. A bit-wise AND means that each bit of RegA is logically ANDed with the corresponding bit in RegB. So, if RegA = $a_3a_2a_1a_0$ and RegB = $b_3b_2b_1b_0$ then the destination register (RegA or RegB) will hold (a_3b_3), (a_2b_2), (a_1b_1), (a_0b_0) after the computation is complete. This is similar to the machine code instruction of the form:

```
AND R0, R1, R0        /*   R0 and R1 = > R0   */
```

The other functions are OR, XOR, NAND, NOR, XNOR, CLR, SET, and SWAP. The complete set of functions and their corresponding control inputs are tabulated below (Table 1).

The block diagram of the circuit you will design for this experiment is shown below (Fig. 1). It includes 1) a **register unit** that contains two 4-bit registers, which we will refer to as **RegA** and **RegB**, 2) a **computation unit** that executes the desired logical computation, 3) a **routing unit** that routes the signals back to the register unit after computation, and 4) a **control unit** that generates control inputs to the register unit.

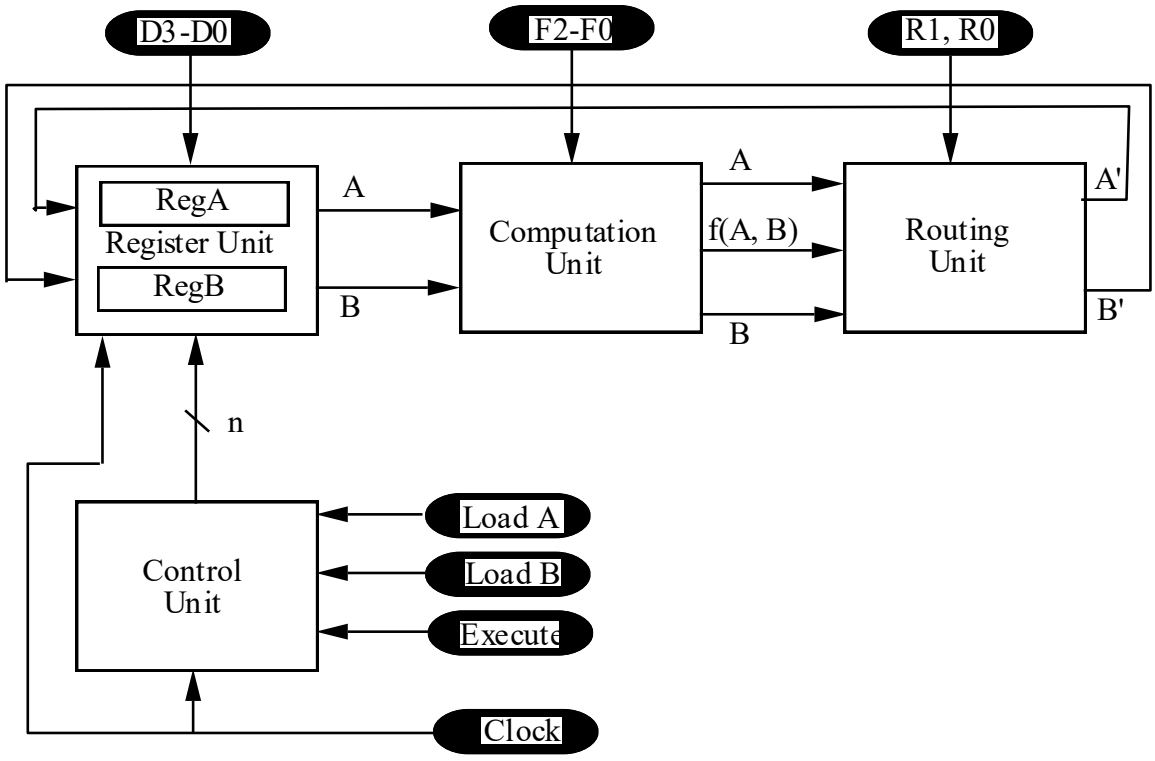

Figure 1: Block Diagram

Register Unit

The register unit will be made up of two 4-bit shift registers (7495A or 74LS194A) that will hold the values of Register A (RegA) and Register B (RegB). The contents of these registers should be displayed so that the contents before and after execution can be inspected. The control of these registers will be provided from the Control Unit while the serial input will be provided from the routing unit.

Computation Unit

The computation unit will accept as inputs the contents of RegA and RegB, and the function selection inputs F2, F1, F0. The unit will output the logical function f(A, B) specified by <F2, F1, F0> and will also output the A and B inputs unchanged. The three outputs will be fed to the Routing Unit.

Routing Unit

The routing unit will accept the A, B, and f(A, B) inputs and, based on the routing selection inputs R1, R0, will determine which signals to feed to the A' (new A) and B' (new B) outputs.

TABLE 1: Functions

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

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

Control Unit

The control unit will accept the following inputs: Load A, Load B, Execute, and the clock signal. The Load A and Load B inputs will perform parallel loads from the data input switches (D3-D0) into the A and B registers. Execute tells the control unit that the select switches and the register contents are ready for execution and that the control unit should begin the computation cycle. The control unit then shifts the register unit the required number of times and halts until the next execution is requested. Obviously,

some type of mechanism to keep track of the shifts will be required. The clock input should be taken from the function generator so as to make the computation cycle appear to be instantaneous while also leaving the debugging capacity of single stepping.

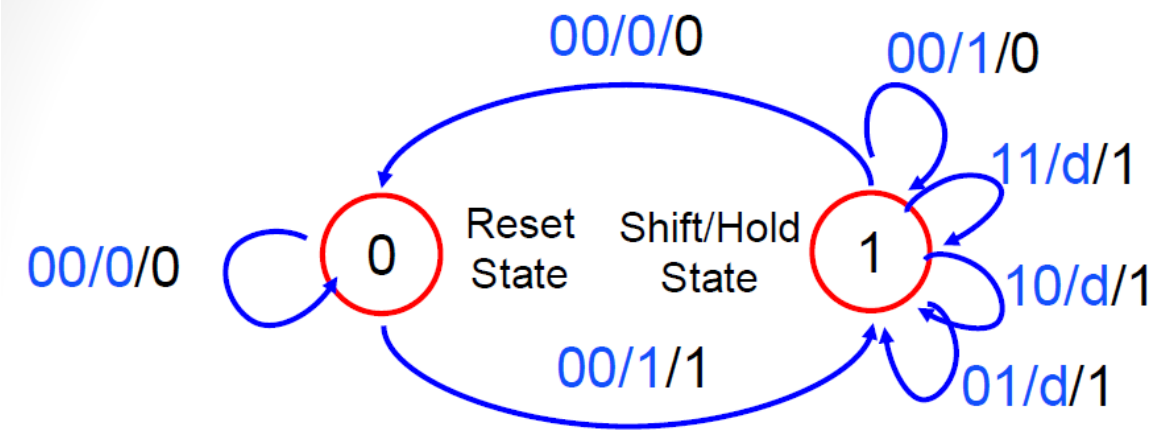

Figure 2: State Diagram

To accomplish this, a finite state machine is devised to control the operation of the register unit. There are two common state machine types: the Moore machine and the Mealy machine. During operation, both of these machines take in a set of inputs, transitions through a finite number of states, and output the relevant controls. The biggest difference between the two state machines is that the outputs of the Moore machine depend solely on the current state, each serving a specific output configuration, while the outputs of the Mealy machine depends on a combination of the current state and the current inputs. From this point of view, it is apparent that the Mealy machine will be able to achieve the same level of control by using fewer states than what's required by the Moore machine, which also makes the circuit implementation a little bit easier.

Table 1 shows an example of the Mealy machine for our control unit. The inputs of the Mealy machine are the 'Execute' switch, a single-bit state representation 'Q', and two-bit count 'C1C0'. The 'Execute' switch dictates when the circuit should initiate the computation cycle. The single-bit state 'Q' split the Mealy machine into two states that serves distinct purposes – one is the reset/rest state, and the other one is the shift/halt state. And the count bits 'C1C0' are used to keep track of the number of shifts in the shift/halt state. The outputs of the mealy machine are the output signal 'Reg. Shift' ('S') which goes to the register unit, the next state 'Q', and the next count 'C1C0'. Notice that

the state descriptions are not very precise here, and this is the characteristics of the Mealy machine. Unlike the Moore machine where every state is directly linked to an operation and thus the purpose is clearly defined, the Mealy machine groups similar operations into a single state, then uses a combination of the current state and the current inputs to perform an operation.

To produce a state machine, you should follow the actual sequential operation of the circuit, where the state and the counts starts from (QC1C0='0000'), and the 'Execute' switch is held low ('E'='0'). As long as the 'Execute' switch remains low, the circuit stay in a rest state, where the register unit stays put and the next state and count also stay unchanged (SQ'C1'C0'='0000'). However, at the immediate clock edge after the 'Execute' switch is flipped up (EQC1C0='1000'), the state machine moves to the shift/halt state, and sends out the signal to shift the registers and begins to increment the counter (SQ'C1'C0'='1101'). The state machine in total should then carry out three additional shifts regardless of the condition of the 'Execute' switch. After the four shifts, the state machine will state in (SQ'C1'C0'='0100') if the 'Execute' switch remains high, or transitions back to (SQ'C1'C0'='0000') if the 'Execute' switch drops low. This completes one full cycle of bit-serial logic operation as the state machine comes back to where it had started, and awaits for another full cycle of operation when the 'Execute' switch is flipped up again.

To transform the state machine into a physical circuit, we will next build a state transition table. Follow through the entire operation cycle to fill out as much of the state transition table as possible. You will then notice that not all combinations are valid. For example, if we are in the reset/rest state, it is not possible for our counts to hold any value other than '00', and therefore we will never encounter, for example, (EQC1C0='0011') during our circuit operation. If the combination is not valid, a 'don't care' ('d') should be placed in the outputs for easier implementation of the circuit (remember that it is ok to circle the K-map minterm over the 'don't cares' without affecting the functionality of the circuit).

TABLE 1: Control unit state transition table using the Mealy state machine

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

After the state transition table is made, we then proceed on producing the K-maps and circling the minterms. Since the transitioning of each of the four state machine components (S, Q, C1, C0) is dictated by four inputs (E, Q, C1, C0), you will need to convert the table to four K-maps and obtain the resulting circuits for each of the four components. Notice that while (Q, C1, C0) are internal components of the state machine, the only actual output 'S' will be used to control the shifting of the two registers. To adapt 'S' and 'LoadA', 'LoadB' to the two registers for the shift/load/halt operations, some kind of simple combinational logic will need to be devised.

Demo Points Breakdown:

1.0 point: Show correct loading of the A and B registers

1.0 point: Show the computation cycle is of the right length

1.0 point: Demonstrate the four routing operations

1.0 point: Demonstrate the eight function operations

1.0 point: Show the computation cycle completes even if the EXECUTE switch is returned to the low position mid-computation.

III. PRE-LAB

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.

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.)

C. Design, document and build the circuit described in Part II. Your circuit should be able to perform correctly all of the functions listed. You may use either 7495A or 74LS194A chips for your shift registers. You will want to study each of the chips carefully before deciding on one or the other. Be sure to make your design as efficient as possible (there is more than one way to design this circuit).

A square wave from the Pulse Generator should be used as the basic system clock. Load A, Load B, Execute, D3-D0, R1, R0, and F2-F0 should be inputs from the switches. The control unit must be designed to perform the desired function once and only once each time the execute switch is flipped on. Results of the operation should be obtained even if the execute switch is flipped off in the middle of the computation cycle. You may only assume that the execute switch will remain high for at least one full clock period. Display the contents of Register A and

Register B on LEDs. You may also want to include an LED that indicates when the computation cycle is complete for debugging purposes.

D. Work with your partner to wire-up the circuit.

IV. LAB

Follow the Lab 3 demo information on the course website.

V. POST-LAB

Document changes to your design and correct your Pre-Lab write-up, explaining any difficulties you had in debugging your circuit.

VI. REPORT

In your lab report, should hand in the following:

- An introduction;
- Answers to parts A and B of the pre-lab;
- Written description of the operation of your circuit;
- State diagram for your controller;
- Design steps taken for all circuits. This includes but not limited to brain storms, state machines, truth tables and K-maps made to arrive at the final circuit design;
- Block diagram;
- Circuit diagram;
- One (1) component layout sheet, with the package layout of all circuits (DO NOT draw the interconnections! Refer to GG.20 for the proper documentation);
- Documentation of any design changes during the lab;
- A conclusion regarding what worked and what didn't, with explanations of any possible causes and the potential remedies.