ECE C03
LAB#3
Design of an Arithmetic and Logic Unit

Spring 1998
Assigned: Tuesday, April 28, 1998
Due: Tuesday, May 12, 1998
at the BEGINNING of class

1 Introduction

The ALU is the combinational network that implements a function of its inputs based upon either logic or arithmetic operations. The ALU is the heart of the computer, as well as most digital subsystems. This lab involves understanding the design of the ALU, which performs a variety of arithmetic and logic operations. You will need to enter the design for a simple ALU block, and replicate it to build a complete ALU in Design Architect. Then, you will need to develop the force files for simulation using Quicksim.

As in previous labs, after typing *ecec03*, create a subdirectory called *lab3*.

****Important**: Throughout this series of labs, be sure to use the filenames recommended in the handouts. Certain components of the CPU expect specific filenames to be used, and use of other names could result in labs that don't work.

2 Advanced Capture and Simulation Techniques

The complexity of the CPU requires use of the following advanced features of design architect and Quicksim. You are encouraged to read more about them in the *Mentor Graphics User's Manual*.

2.1 Buses

To avoid showing many logically related nets (wires) on circuit designs, the Mentor system allows grouping of multiple signals into a bus. These buses appear as bold lines in logic circuits and their names imply the number of wires represented. For example, the output of the ALU is called ALU_OUT(3:0), implying that the bus is 4 bits wide with bit 3 the most significant bit. Values are forced onto buses in Quicksim command files and appear on simulation traces in hexadecimal.

Individual signals are "extracted" from a bus by components called rippers. These appear as short diagonal lines on a circuit with one end wide like a bus, the other narrow like a single net. The number near the ripper indicates the bit being extracted. For example, ALU_OUT(3), the ALU sign bit, is extracted by a ripper labeled 3.

3 The ALU

3.1 Functional Capabilities

The ALU can perform 12 arithmetic-logic operations on two inputs A(3:0) and B(3:0) to produce the output ALU_OUT(3:0) and maintain the status information bits V (overflow), S (sign), Z (result = zero), and C (carry). The operation performed is selected by the S0, S1, M, and C signals as in Table 1.

The basic logic block for one bit-slice of the ALU is shown in Figure 1. This can be replicated to form the entire 4-bit ALU. These single bit slices are then combined to form the four bit ALU as shown in Figure 2. The status bits are created from the output of the four bit ALU as shown in Figure 3. Remember to adjust the gate delays appropriately from the figure shown.

3.2 Subcomponents

The ALU is created from the following lower-level components:

- **Arithmetic Logic Unit**. Performs the seven arithmetic operations specified by FC(4:0) = 00000 to 00110.
- **Zero Detect**. Output is high when ALU_OUT(3:0) has zeros on all 4 input bits. This serves as input to the Z status bit.

3.3 Input Signals

The ALU has the following input signals:

- **A(3:0)** and **B(3:0)**. Two 4 bit buses on which the ALU will perform operations.
- **S0** and **S1**. A 2-bit Function Control bus that selects the desired operation.
- **M**. A periodic signal used to load information into the Status Register.
- **C**. Resets the F/F's that hold V, S, Z, and C to 0 when SysReset = 1.

3.4 Output Signals

The ALU has the following output signals:

- **ALU_OUT(3:0)**. The 4-bit output of the ALU.
- **V**, **S**, **Z**, and **C**. The status bits.

TABLE 1. ALU Operations

C	M	S(1:0)	Function	Comment
X	0	00	F = A	Input A transfered to output
X	0	01	F = not A	Complement of A transferred to output
X	0	10	F = A xor B	Compute XOR of A, B
X	0	11	F = A xnor B	Compute XNOR of A, B
0	1	00	F = A	Input A passed to output
0	1	01	F = not A	Complement of A passed to output
0	1	10	F = A + B	Sum of A and B
0	1	11	F = (not A) + B	Sum of B and complement of A
1	1	00	F = A + 1	Increment A
1	1	01	F = (not A) +1	Twos complement A
1	1	10	F = A + B + 1	Increment sum of A and B
1	1	11	F = (not A) + B + 1	B minus A

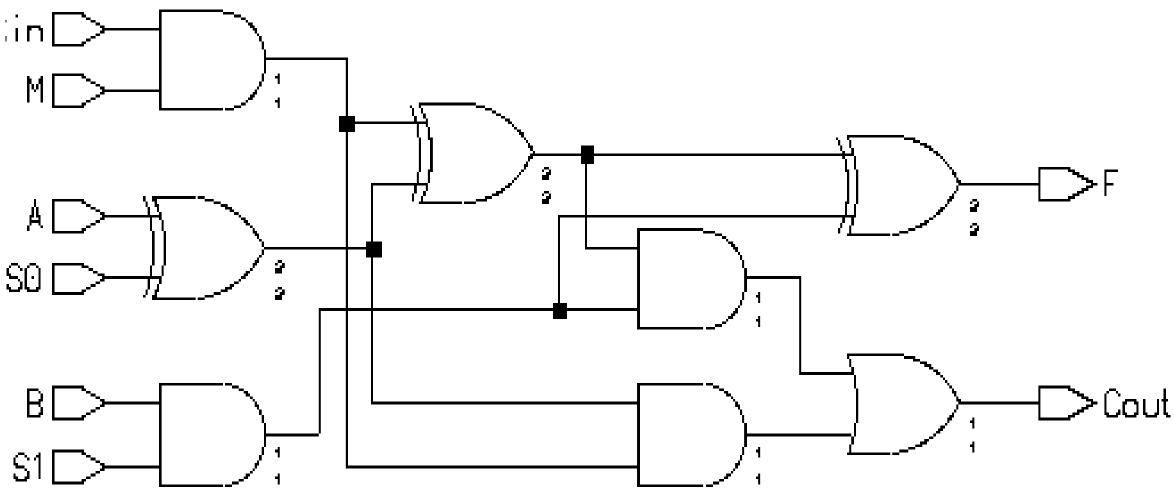

Figure 1 - Single Bit Slice

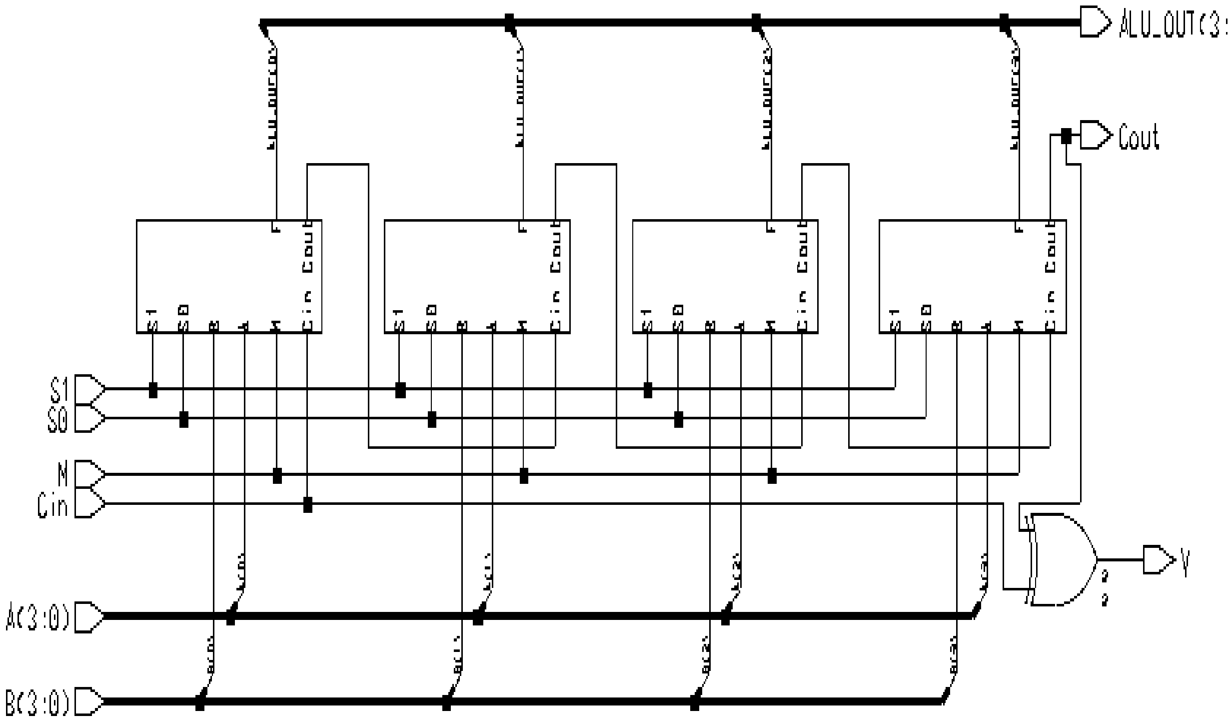

Figure 2 - Four ALU bit slices

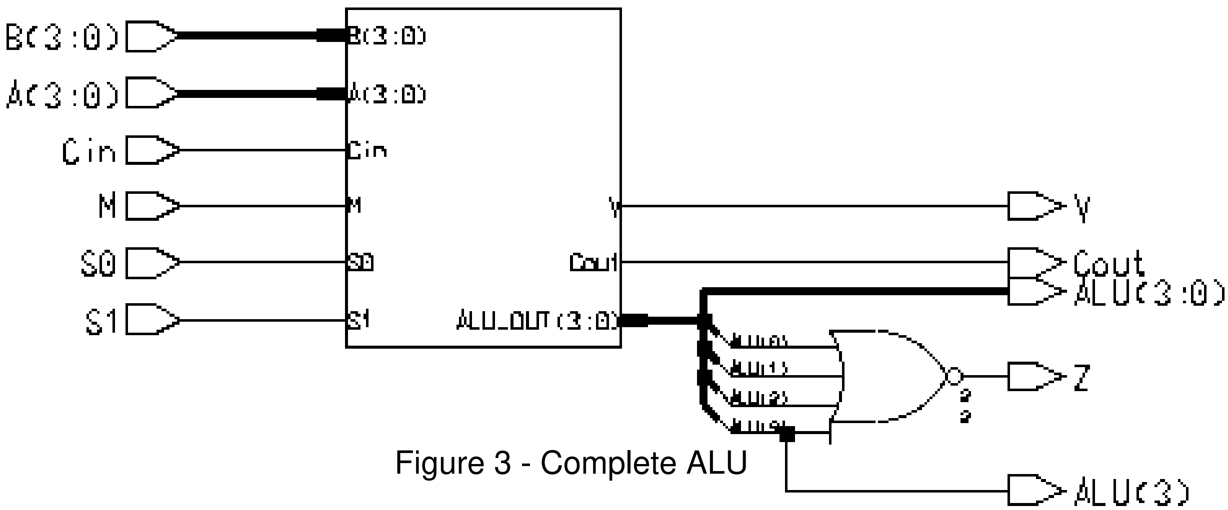

Figure 3 - Complete ALU

4 Simulation

Your assignment is to simulate the ALU using a given circuit and partially given Quicksim command file. In your *lab3* directory, copy the command file to your account by typing the following command at the Unix prompt:

```
cp /vol/MGC/ecec03/alu_test .
```

Don't forget the "." at the end of the command.

Complete Table 4 by hand, assuming A(3:0)=c_{16} and B(3:0)=4_{16}. Then use your favorite editor to edit the command file to make the ALU perform the computations listed in Table 4. Specifically, you must change fifteen sets of *'s to the appropriate values. Everything else has been given to you. For example, if you were to perform the operation Increment A at time t=60 you would find in the file:

```
// Increment A at t=60
$force(″/S1″, ″**″, 60, 70, @fixed, @absolute, @norepeat);
$force(″/***″, ″0″, 60, 70, @fixed, @absolute, @norepeat);
$force(″/***″, ″1″, 60, 70, @fixed, @absolute, @norepeat);
$force(″/***″, ″1″, 70, 80, @fixed, @absolute, @norepeat);
```

You would change the first set of two *'s to 0, since to perform Increment A according to Table 1 you must set S1=0. Since the increment instruction also has S0=0, the second set of *'s is set to S0. This is an arinthmetic operation so M=1, and the carry must also be set to 1 to perform the increment, so the last two sets of *'s are set to M and Cin respectively. The code after making your changes would look as follows:

```
// Increment A at t=60
$force(″/S1″, ″0", 60, 70, @fixed, @absolute, @norepeat);
$force(″/S0″, ″0″, 60, 70, @fixed, @absolute, @norepeat);
$force(″/M″, ″1″, 70, 80, @fixed, @absolute, @norepeat);
$force(″/Cin″, ″1″, 60, 70, @fixed, @absolute, @norepeat);
```

Each set of three *'s may be replaced with S1, S0, M, or Cin to enable the appropriate status bits. Complete Table 4 showing the expected results of each computation.

Enter Quicksim by clicking on your complete alu, and starting quicksim on it. Execute your command file by typing:

```
dofile  lab3/alu_test
```

as in previous labs. The circuit is simulated for 60 time units since the command `run 60` is included in the command file. Since there are more traces than can appear in the trace window at its default size, grab the top edge of the window and extend it up until all of the traces fit. Verify that the output is consistent with the predictions you made in Table 4, then generate a printed trace.

TURN IN the following items for a grade for Lab #3:

√ Your Quicksim trace.

√ A printout of your file alu_test.

TABLE 4. ALU Test Operations

time (ns.)	Function	Expected ALU_OUT(3:0) (in hex)	V	S	Z	C
0	F=A		0	0	0	0
10	F=(not A) plus B					
20	F= A XOR B					
30	F = (not A) plus B plus 1					
40	F = not A					
50	F = A plus B					