LAB 3

THE ARITHMETIC LOGIC UNIT

In Lab 3, you design an Arithmetic Logic Unit that supports six different operations. In the process, you gain even more practice with concepts like: hierarchical design, binary numbers, and hexadecimal numbers. You will also learn how to handle multi-bit values in Logisim Evolution.

This document describes what you need to prepare and demonstrate for Lab 3. Section 3.3 briefly describes multi-bit values in Logisim Evolution. Section 3.4 describes the tasks you must complete *before* your lab session. Section 3.5 describes the tasks you complete *during* your lab session. The next section describes lab logistics in more detail.

3.1 Logistics

Even though you work in pairs during your lab session, you are assessed individually on your Lab Preparation ("pre-lab") and Lab Demonstration ("demo"). All pre-lab exercises are submitted electronically before your lab (see the course website for exact due dates, times, and the submission process). So, **before** each lab, you must read through this document and complete all the pre-lab exercises. During the lab, use your pre-lab designs to help you complete all the required in-lab actions. The more care you put into your pre-lab designs, the faster you will complete your lab.

Before beginning the pre-lab, read Section 3.3. The Lab Preparation must be completed individually and submitted online by the due date. Follow the steps in Section 3.4 for the pre-lab. Remember to **download the starter files**.

You must upload *every required file* for your pre-lab submission to be complete. But you do *not* need to include images that are not on the list of required files (even if those images are in your lab report). If you have questions about the submission process, please ask ahead of time. The required files for Lab 3's pre-lab (Section 3.4) are:

- Your lab report: `lab3_report.tex`, `lab3_report.pdf` (as generated from the tex file)
- Your digital designs: `lab3.circ`

The Lab Demonstration must be completed during the lab session that you are enrolled in. During a lab demonstration, your TA may ask you to: go through parts of your pre-lab, run and simulate your designs in Logisim, and answer questions related to the lab. You may not receive outside help (e.g., from your partner) when asked a question.

3.2 Marking Scheme

Each lab is worth 4% of your final grade, where you will be graded out of 4 marks for this lab, as follows.

- Prelab: 1 mark
- Part I (in-lab): 0.5 mark
- Part II (in-lab): 1.5 mark
- Part III (in-lab): 1 mark

3.3 Multi-bit Values

In earlier labs, each wire, input, and output represented a single bit. But in digital logic, it is often necessary to work with numbers that are multiple bits. For example, the input to a circuit may be 32 bits. Drawing 32 different inputs, one for each bit, is both tedious and error-prone. In this section, you learn how to design circuits with components that can work with more than one bit at a time.

3.3.1 Bundles and Splitters

In Logisim Evolution, we can components in our schematic to have a different *bit width*. Before starting the pre-lab, read the Logisim Evolution tutorial by navigating to `Help -> Tutorial`. Then, in the index on the left-hand side, navigate to `Additional features -> Creating bundles`. By the end of the tutorial, you should be able to connect inputs, outputs, and gates together that use multi-bit values.

Sometimes it is necessary to split up or combine multi-bit values into their individual bits. Logisim Evolution allows for both using something called a `Splitter`. After completing the above tutorial, you should also read the tutorial on `Additional features -> Splitters`.

3.3.2 Bit Extenders

When a value is represented with N bits but the output must be M bits, where $M > N$, then the N-bit value must be *extended* to M bits. A *zero extension* concatenates `0`s to the most significant bits of N. Zero-extending an N-bit value should not be done if the value is signed. Instead, a *sign extension* preserves the sign of N when extending it to M bits. To learn more, navigate to `Help -> Library Reference` in Logisim Evolution. Next, select `Wiring -> Bit Extender`.

3.3.3 Simulating Multi-bit Values

In this lab, you test circuits where the inputs or outputs are multi-bit values, and include the results of those tests in your lab report. Consider the inverter in Figure 3.1, which takes an input `A` (that is 2 bits) and produces an output `Y` (that is also 2 bits). In the Test Vector file, you must use a special synatx for variables that are more than 1 bit. Listing 3.1 shows how this is done for both the input and the output, where the number between the square brackets corresponds to the number of bits for that variable. The resulting simulation is shown in Figure 3.2, which is an example of the type of screenshot you should include in your reports when asked to.

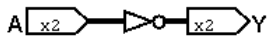

Figure 3.1: Contrived Circuit

```
A[2]  Y[2]
00    11
01    10
10    01
11    00
```

Listing 3.1: Test Vector

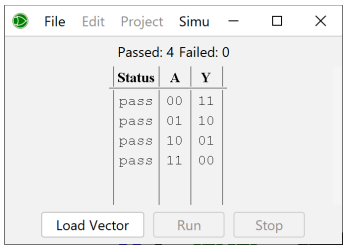

Figure 3.2: Test Outcomes

3.4 Lab Preparation

The pre-lab for Lab 3 consists of two parts. In Part I, you implement and test a four-bit ripple-carry adder. In Part II, you design and implement an Arithmetic Logic Unit (ALU).

3.4.1 Part I

In this part, you construct a full adder and then apply modularity and regularity to implement a four-bit ripple-carry adder. Both adders were discussed in lecture and are covered in the textbook. You may **not** use the built-in `Arithmetic -> Adder` element in Logisim Evolution for Part I. After constructing the adders in Logisim Evolution, you ensure they are correct with test vectors.

Perform the following steps:

1. In a file called `lab3.circ` create a subcircuit called `full_adder` that implements the behaviour of a Full Adder. The subcircuit should have (1-bit) inputs `A, B, Cin` and (1-bit) outputs `S, Cout`.

2. In the same `lab3.circ` file, create a subcircuit called `ripple_carry4` that implements the behaviour of a 4-bit Ripple Carry Adder. The subcircuit should have one 1-bit input `Cin` and two 4-bit inputs `A, B`. The outputs should be one 1-bit output `Cout` and one 4-bit output `S`. You must re-use your Full Adder subcircuit here. No gates are necessary, but you will need to use more than one `Splitter` in Logisim Evolution.

   Export the subcircuit schematic as an image and include it in your report.

3. Simulate your 4-bit Ripple Carry Adder using test vectors with *intelligently* chosen values for the inputs `A, B, Cin`. Take a screenshot of your test vector results and include it in your report.

   Intelligently chosen means that your test cases should not exhaustively try every input combination (in this case, that would be 2^9 cases). Instead, find the particular *corner cases* that exercise key aspects of the circuit. For example, a pattern that shows the carry signals are working. Or how the circuit behaves with maximum and minimum values for A and B.

   Be prepared to justify and explain your test cases during the demonstration.

3.4.2 Part II

In this part, you design an Arithmetic Logic Unit (ALU). The ALU has three inputs: two 4-bit data inputs `A, B` (unsigned), and a multi-bit input `op` (i.e., operation). It also has one 8-bit output: `ALUout` (unsigned). A summary of the operations supported by the ALU is shown in Table 3.1. Each operation

should first be designed as a subcircuit with the name `op#`, where # corresponds to the Operation column in Table 3.1.

Table 3.1: ALU Operations

Operation	Subcircuit Name	Input(s)	Behaviour
0	op0	`A[4]`	The output, ALUout, equals A + 1. The + indicates summation, not a logical OR. This operation must use the adder circuit from Section 3.4.1.
1	op1	`A[4], B[4]`	The output, ALUout, equals A + B. The + indicates summation, not a logical OR. This operation must use the adder circuit from Section 3.4.1.
2	op2	`A[4], B[4]`	The output, ALUout, equals A + B. The + indicates summation, not a logical OR. This operation must use the adder from Logisim Evolution found under `Arithmetic`.
3	op3	`A[4], B[4]`	The output, ALUout, equals a bitwise A XOR B in the least significant four bits and a bitwise A OR B in the most significant four bits.
4	op4	`A[4], B[4]`	The output, ALUout, equals 1 (i.e., `00000001`) if any of the bits in A or B are high and 0 if all the bits are low. This is also known as a "reduction OR operation"
5	op5	`A[4], B[4]`	The output, ALUout, equals A in the most significant four bits and B in the least significant four bits.

Note that a **bitwise** operation (see op3 in Table 3.1) treats each bit separately, where each bit in the output is the result of performing the bitwise operation on the corresponding bits in the input. So a bitwise OR would set the first digit in the output to the OR of the first digits in the two inputs, and so on for the remaining digits. Also note that when performing logical operations (AND, OR, XOR, etc) on multi-bit input values, the **reduction** operation (see op4 in Table 3.1) treats the input value as "false" if the multi-bit value is 0 and "true" otherwise. The output is a single true or false value, represented as a multibit 1 (i.e., `00000001`) or 0.

Perform the following steps **for each operation** described in Table 3.1:

1. In the same `lab3.circ` file, create a subcircuit that implements the behaviour as described in Table 3.1. **Every operation must have one 8-bit output** called `OPout` (inputs may vary, see Table 3.1). You may need to use a `Splitter` and/or `Bit Extender` in Logisim Evolution.

   Export the subcircuit schematic as an image and include it in your report.

Perform the following steps **for only operations 3, 4, and 5** described in Table 3.1:

2. Simulate your subcircuit using test vectors with *intelligently* chosen values for the input. Take a screenshot of your test vector results and include it in your report.

   Be prepared to justify and explain your test cases during the demonstration.

Once you have designed a subcircuit for each operation, perform the following steps:

3. In the same `lab3.circ` file, create a subcircuit called `ALU`. The ALU has three inputs: two unsigned 4-bit data inputs `A, B`, and a multi-bit input `op` (i.e., operation). It also has one unsigned 8-bit output: `ALUout`. The ALU should re-use the subcircuits you created for each operation. *Hint: Use a multiplexer to change the output, ALUout, based on the operation, op.*

Export the subcircuit schematic as an image and include it in your report.

3.5 Lab Demonstration

The lab demonstration consists of three parts. The first two parts correspond to Parts I and II from the pre-lab. In Part III, you combine what you have accomplished in this lab with your designs from Lab 2. When demonstrating a part to your TA, be ready to answer questions **individually**.

3.5.1 Part I

Test your two subcircuits in Logisim Evolution with the `Poke` tool. Test your subcircuits with test vectors. Discuss with your partner why your tests are "intelligently" selected. **Make sure you are ready to, if asked, explain your testing methodology.** When you are done, demonstrate your working design to your TA.

3.5.2 Part II

Test all your subcircuits in Logisim Evolution with the `Poke` tool. Test all your subcircuits with test vectors. Discuss with your partner why your tests are "intelligently" selected. **Make sure you are ready to, if asked, explain your testing methodology.** When you are done, demonstrate your working design to your TA.

3.5.3 Part III

In this part, you combine what you have completed this lab with your decoder from Lab 2. Perform the following steps:

1. With `lab3.circ` open, use `File -> Merge` and select your `lab2_part2.circ` from last lab. This should import all the subcircuits from `lab2_part2.circ`, one of which should be the `HEX_DECODER` subcircuit. When prompted, you can safely choose to or choose not to import the `main` circuit from `lab2_part2.circ`. If you do choose to import it, note that you will be re-doing `main` in a later step.

2. Modify your `HEX_DECODER` subcircuit so that it takes only one 4-bit input, `D`. You should *not* modify any other subcircuits to do this. You should also *not* change the outputs to the `HEX_DECODER` subcircuit.

3. In the top-level (i.e., main) circuit of `lab3.circ`, create a circuit that has two 4-bit inputs `A, B` and a multi-bit input for `op`.

4. Connect each input to its own `HEX_DECODER`.

5. Also connect each input to one `ALU`.

6. Connect the output of the `ALU` to two `HEX_DECODER`s, one for the most significant bits and the other for the least significant bits.

7. Connect all four `HEX_DECODER`s to their own 7-segment displays. Label the displays `HEXA, HEXB` corresponding to inputs `A, B`. Label the displays `HEX_ALU1, HEX_ALU0` corresponding to the most and least siginficant 4 bits of `ALUout`, respectively.

Test your top-level circuit in Logisim Evolution with the `Poke` tool. Note the hexadecimal values appearing on the 7-segment displays. Familiarize yourself with the operations of the ALU in terms of hexadecimal values. When you are done, demonstrate your working design to your TA.