LOGISIM-EVOLUTION LAB MANUAL

GEORGE SELF

July 2019 – Edition 4.0

George Self: *Logisim-Evolution Lab Manual*

This work is licensed under a Creative Commons "CC0 1.0 Universal" license.

PREFACE

I have taught CIS 221, *Digital Logic*, for Cochise College since about 2003 and enjoy working with students on this topic. From the start, I wanted students to work with labs as part of our studies and actually design circuits to complement our theoretical instruction. As I evaluated circuit design software I had three criteria:

- **Open Educational Resource (OER)**. It is important to me that students use software that is available free of charge and is supported by the entire web community.
- **Platform**. While most of my students use a Windows-based system, some use Macintosh and it was important to me to use software that is available for both of those platforms. As a bonus, most OER software is also available for the Linux system, though I'm not aware of any of my students who are using Linux.
- **Simplicity**. I wanted to use software that was easy to master so students could spend their time understanding digital logic rather than learning the arcane structures of a simulation language.

I originally wrote a number of lab exercises using *Logisim*, but the creator of that software, Carl Burch, announced that he would quit developing it in 2014. Because it was published as an open source project, a group of Swiss institutes started with the *Logisim* software and developed a new version that integrated several new tools, like a chronogram, and released it under the name *Logisim-Evolution* .

It is my hope that students will find these labs instructive and the labs enhance their learning of digital logic. This lab manual is written with LaTeX and published under a Creative Commons Zero license with a goal that other instructors can modify it to meet their own needs. The source code can be found at my personal GITHUB page and I always welcome comments that will help me improve this manual.

—George Self

BRIEF CONTENTS

List of Figures x
List of Tables xi
Listings xii

I INTRODUCTION TO LOGISIM-EVOLUTION 1
1 INTRODUCTION TO LOGISIM-EVOLUTION 3

II THEORY 9
2 ADDER 11
3 ARITHMETIC OPERATIONS 19
4 LOGIC OPERATIONS 27
5 BOOLEAN LOGIC 29
6 PROGRAMMABLE LOGIC ARRAY 35

III PRACTICE 37
7 ARITHMETIC LOGIC UNIT (ALU) 39

IV SIMULATION 45

V APPENDIX 47
A TTL REFERENCE 49

CONTENTS

List of Figures x
List of Tables xi
Listings xii

I INTRODUCTION TO LOGISIM-EVOLUTION 1

1 INTRODUCTION TO LOGISIM-EVOLUTION 3
1.1 Purpose 3
1.2 Procedure 3
1.2.1 Installation 3
1.2.2 Beginner's Tutorial 3
1.2.3 Logisim-evolution Workspace 4
1.2.4 Simple Multiplexer 5
1.2.5 Identifying Information 8
1.3 Deliverable 8

II THEORY 9

2 ADDER 11
2.1 Purpose 11
2.2 Procedure 11
2.2.1 Half-Adder 11
2.2.2 Full Adder 12
2.2.3 Main Circuit 12
2.2.4 Automated Testing 13
2.3 Deliverable 17

3 ARITHMETIC OPERATIONS 19
3.1 Purpose 19
3.2 Procedure 19
3.3 Deliverable 26

4 LOGIC OPERATIONS 27
4.1 Purpose 27
4.2 Procedure 27
4.3 Deliverable 28

5 BOOLEAN LOGIC 29
5.1 Purpose 29
5.2 Procedure 29
5.2.1 Subcircuit: Equation 1 29
5.2.2 Subcircuit: Equation 2 31
5.2.3 Main Circuit 32
5.3 Deliverable 33

6 PROGRAMMABLE LOGIC ARRAY 35
6.1 Purpose 35
6.2 Procedure 35

6.3 Deliverable 35

III PRACTICE 37

7 ARITHMETIC LOGIC UNIT (ALU) 39
7.1 Purpose 39
7.2 Procedure 40
7.2.1 main 40
7.2.2 ALU 40
7.2.3 Arithmetic 41
7.2.4 Challenge 42
7.2.5 Testing the Circuit 43
7.3 Deliverable 43

IV SIMULATION 45

V APPENDIX 47

A TTL REFERENCE 49
A.1 7400: Quad 2-Input NAND Gate 49
A.2 7402: Quad 2-Input NOR Gate 50
A.3 7404: Hex Inverter 51
A.4 7408: Quad 2-Input AND Gate 52
A.5 7410: Triple 3-Input NAND Gate 53
A.6 7411: Triple 3-Input AND Gate 54
A.7 7413: Dual 4-Input NAND Gate (Schmitt-Trigger) 55
A.8 7414: Hex Inverter (Schmitt-Trigger) 56
A.9 7418: Dual 4-Input NAND Gate (Schmitt-Trigger Inputs) 57
A.10 7419: Hex Inverter (Schmitt-Trigger) 58
A.11 7420: Dual 4-Input NAND Gate 59
A.12 7421: Dual 4-Input AND Gate 60
A.13 7424: Quad 2-Input NAND Gate (Schmitt-Trigger) 61
A.14 7427: Triple 3-Input NOR Gate 62
A.15 7430: Single 8-Input NAND Gate 63
A.16 7432: Quad 2-Input OR Gate 64
A.17 7436: Quad 2-Input NOR Gate 65
A.18 7442: BCD to Decimal Decoder 66
A.19 7443: Excess-3 to Decimal Decoder 67
A.20 7444: Gray to Decimal Decoder 69
A.21 7447: BCD to 7-Segment Decoder 71
A.22 7451: Dual AND-OR-INVERT Gate 73
A.23 7454: Four Wide AND-OR-INVERT Gate 74
A.24 7458: Dual AND-OR Gate 75
A.25 7464: 4-2-3-2 AND-OR-INVERT Gate 76
A.26 7474: Dual D-Flipflops with Preset and Clear 77
A.27 7485: 4-Bit Magnitude Comparator 78
A.28 7486: Quad 2-Input XOR Gate 78
A.29 74125: Quad Bus Buffer, 3-State Gate 79

A.30 74165: 8-Bit Parallel-to-Serial Shift Register 80
A.31 74175: Quad D-Flipflops with Sync Reset 81
A.32 74266: Quad 2-Input XNOR Gate 81
A.33 74273: Octal D-Flipflop with Clear 82
A.34 74283: 4-Bit Binary Full Adder 83
A.35 74377: Octal D-Flipflop with Enable 84

LIST OF FIGURES

Figure 1.1 Logisim-evolution Initial Screen 4
Figure 1.2 Two AND Gates 5
Figure 1.3 AND Gate Properties 6
Figure 1.4 OR Gate Added to Circuit 6
Figure 1.5 Two NOT Gates Added to Circuit 7
Figure 1.6 Inputs and Output Added 7
Figure 1.7 Circuit Wiring Added 7
Figure 1.8 Simple multiplexer 8
Figure 2.1 Half-Adder 11
Figure 2.2 Full Adder 12
Figure 2.3 Full Adder 13
Figure 2.4 Test Vector Window 15
Figure 2.5 Test Completed 16
Figure 2.6 Test Failure 17
Figure 3.1 Placing the Arithmetic Components 20
Figure 3.2 Arithmetic Inputs and Outputs 22
Figure 3.3 Placing the Multiplexers 23
Figure 3.4 Wiring the Data Inputs and Outputs 24
Figure 3.5 Arithmetic Final Circuit 25
Figure 3.6 Arithmetic Main Circuit 25
Figure 4.1 The Main Logic Circuit 28
Figure 5.1 Equation 1 Inputs-Outputs 30
Figure 5.2 Equation 1 And-Or Gates 30
Figure 5.3 Equation 1 And Gate Inputs Set 31
Figure 5.4 Equation 1 Circuit Completed 31
Figure 5.5 Main Circuit 33
Figure 7.1 ALU main 40
Figure 7.2 ALU Subcircuit 41
Figure 7.3 Arithmetic Subcircuit 42
Figure 7.4 Logic Subcircuit 42
Figure A.1 Three Surface-Mounted Integrated Circuits 49
Figure A.2 7400: Single NAND Gate Circuit 49
Figure A.3 7402: Single NOR Gate Circuit 50
Figure A.4 7404: Single Inverter Circuit 51
Figure A.5 7408: Single AND Gate Circuit 52
Figure A.6 7410: Single 3-Input NAND Gate Circuit 53
Figure A.7 7411: Single 3-Input AND Gate Circuit 54
Figure A.8 7413: Single 4-Input NAND Gate Circuit 55
Figure A.9 7414: Single Inverter Circuit 56
Figure A.10 7418: Single 4-Input NAND Gate Circuit 57
Figure A.11 7419: Single Inverter Circuit 58

Figure A.12 7420: Single 4-Input NAND Gate Circuit 59
Figure A.13 7421: Single 4-Input AND Gate Circuit 60
Figure A.14 7424: Single NAND Gate Circuit 61
Figure A.15 7411: Single 3-Input NOR Gate Circuit 62
Figure A.16 7430: Single 8-Input NAND Gate 63
Figure A.17 7432: Single OR Gate Circuit 64
Figure A.18 7436: Single NOR Gate Circuit 65
Figure A.19 7442: BCD to Decimal Decoder 66
Figure A.20 7447: BCD to 7-Segment Decoder 71
Figure A.21 7451: Single AND-OR-INVERT Gate Circuit 73
Figure A.22 7454: Four Wide AND-OR-INVERT Gate Circuit 74
Figure A.23 7458: Dual AND-OR Gate Circuit 75
Figure A.24 7464: 4-2-3-2 AND-OR-INVERT Gate Circuit 76
Figure A.25 7486: Single XOR Gate Circuit 78
Figure A.26 74125: Single Buffer Circuit 79
Figure A.27 74266: Single XNOR Gate Circuit 81

LIST OF TABLES

Table 2.1 Test Vector For Full Adder 13
Table 7.1 Function Table for 74181 ALU 39
Table A.1 Pinout For 7400 50
Table A.2 Pinout For 7402 51
Table A.3 Pinout For 7404 52
Table A.4 Pinout For 7408 53
Table A.5 Pinout For 7410 54
Table A.6 Pinout For 7411 55
Table A.7 Pinout For 7413 56
Table A.8 Pinout For 7414 57
Table A.9 Pinout For 7418 58
Table A.10 Pinout For 7419 59
Table A.11 Pinout For 7420 60
Table A.12 Pinout For 7421 61
Table A.13 Pinout For 7424 62
Table A.14 Pinout For 7427 63
Table A.15 Pinout For 7430 64
Table A.16 Pinout For 7432 65
Table A.17 Pinout For 7436 66
Table A.18 Truth Table For The 7442 Circuit 67
Table A.19 Pinout For 7442 67
Table A.20 Truth Table For The 7443 Circuit 68
Table A.21 Pinout For 7443 69

Table A.22 Truth Table For The 7444 Circuit 70
Table A.23 Pinout For 7444 70
Table A.24 Truth Table For The 7447 Circuit 72
Table A.25 Pinout For 7447 73
Table A.26 Pinout For 7451 74
Table A.27 Pinout For 7454 75
Table A.28 Pinout For 7458 76
Table A.29 Pinout For 7464 77
Table A.30 Pinout For 7474 77
Table A.31 Pinout For 7485 78
Table A.32 Pinout For 7486 79
Table A.33 Pinout For 74125 80
Table A.34 Pinout For 74165 80
Table A.35 Pinout For 74175 81
Table A.36 Pinout For 74266 82
Table A.37 Pinout For 74273 83
Table A.38 Pinout For 74283 84
Table A.39 Pinout For 74377 85

LISTINGS

ACRONYMS

ALU Arithmetic Logic Unit

CPU Central Processing Unit

IC Integrated Circuit

OER Open Educational Resource

PLA Programmable Logic Array

TTL Transistor-Transistor Logic

Part I

INTRODUCTION TO LOGISIM-EVOLUTION

Logisim-Evolution is used to create and test simulations of digital circuits. This part of the lab manual includes only one lab designed to introduce *Logisim-Evolution* and teach the fundamentals of using this application.

1 INTRODUCTION TO LOGISIM-EVOLUTION

1.1 PURPOSE

This lab introduces the *Logisim-Evolution* logic simulator, which is used for all lab exercises in this manual.

1.2 PROCEDURE

1.2.1 *Installation*

Logisim-Evolution is a Java application, so a Java runtime environment will need to be installed before using the application. Many students who are taking a digital logic class already have a Java runtime on their computer and can skip this step, but those who do not will need to install the Java runtime. That process is not covered in this manual but information about installing the Java runtime environment is available at `http://www.oracle.com/technetwork/java/javase/downloads/index.html`. It can be confusing to know which version of Java to download but students working on the labs in this manual only need the runtime, called *JRE* on the website. Students who are also in programming classes will likely already have the runtime as part of the Java Developer's Kit (JDK). It can be tricky testing the Java installation since the Chrome, Firefox, and Edge browsers will not run Java apps, but students can open a command prompt and enter `java -version` to see what version of Java their computers are running, if any.

Logisim-Evolution (`https://github.com/reds-heig/logisim-evolution`) is available as a free download. Visit the website and about halfway down the page find a section named "Running logisim-evolution." Click the "here" link at the end of the first sentence in that section.

Since the *Logisim-Evolution* file is a Java application, it does not need to be installed like most software. To start *Logisim-Evolution* , double-click the *Logisim-Evolution* shortcut. That will start Java and then run the *Logisim-Evolution* application. Also, *Logisim-Evolution* will not need to be uninstalled when it is no longer needed since it is not actually installed, the *Logisim-Evolution* file can simply be deleted.

1.2.2 *Beginner's Tutorial*

Logisim-Evolution comes with a beginner's tutorial available in HELP -> TUTORIAL. That tutorial only takes a few minutes and introduces

students to the major components of the application. Students should complete that tutorial before starting this lab.

1.2.3 *Logisim-evolution Workspace*

Start *Logisim-Evolution* by double-clicking its icon. The initial *Logisim-Evolution* window will be similar to Figure 1.1.

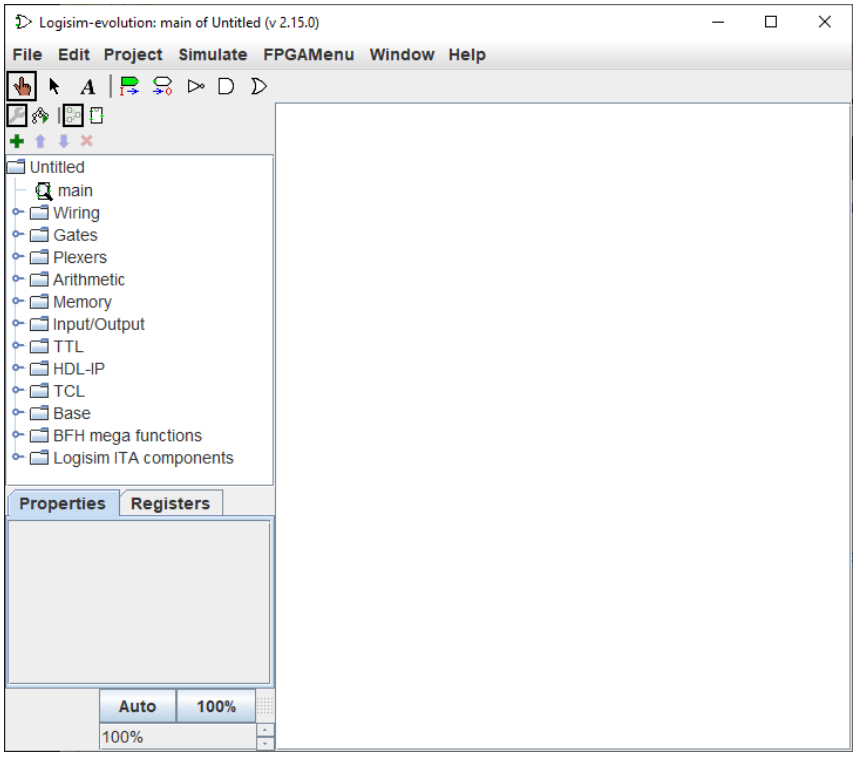

Figure 1.1: Logisim-evolution Initial Screen

The *Logisim-Evolution* space is divided into several areas. Along the top is a text menu that includes the types of selections found in most programs. For example, the "File" menu includes items like "Save" and "Exit." The "Edit" menu includes an "Undo" option that is useful. In later labs, the various options under "Project" and "Simulate" will be described and used. Items in the "FPGAMenu" are beyond the scope of this class and will not be used. Of particular importance at this point is "Library Reference" in the "Help" menu. It contains information about every logical device available in *Logisim-Evolution* and is very useful while using those components in new circuits.

Under the menu bar is the Toolbar, which is a row of eight buttons that are the most commonly used tools in *Logisim-Evolution* :

- **Pointing Finger**: Used to "poke" and change input values while the simulator is running.

- **Arrow**: Used to select components or wires in order to modify, move, or delete them.
- **A**: Activates the Text tool so text information can be added to the circuit.
- **Green Input Port**: Creates an input port for a circuit.
- **White Output Port**: Creates an output port for a circuit.
- **NOT Gate**: Creates a NOT gate.
- **AND Gate**: Creates an AND gate.
- **OR Gate**: Creates an OR gate.

The Explorer Pane is on the left side of the workspace and contains a folder list. The folders contain "libraries" of components organized in a logical manner. For example, the "Gates" folder contains various gates (AND, OR, XOR, etc.) that can be used in a circuit. The four icons across the top of the Explorer Pane are used for advanced operations and will be covered as they are needed.

The Properties panel on the lower left side of the screen is where the properties for any selected component can be read and set. For example, the number of inputs for an AND gate can be set to a specific number.

The drawing canvas is the largest part of the screen. It is where circuits are constructed and simulated.

1.2.4 *Simple Multiplexer*

Do not be concerned with the exact placement of components on the drawing canvas. They can be rearranged as the build progresses.

A multiplexer is used to select which of two or more inputs will be connected to a single output. For this lab, a simple two-input, one-bit multiplexer will be built. It is understood that students will not know the significance of a multiplexer at this point in the class, but the purpose of this lab is to use *Logisim-Evolution* to build a simple circuit and a multiplexer serves that purpose well.

Start by clicking the *And* button on the toolbar and placing two `AND` gates on the canvas. The canvas should resemble Figure 1.2

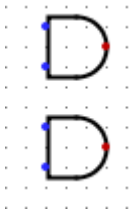

Figure 1.2: Two AND Gates

Click one of the AND gates to select it and observe the various properties available for that gate, as seen in Figure 1.3. The default values do not need to be changed for this circuit; however, all circuits in this manual use the "Narrow" gate size in order to make the circuit fit the screen better. The other properties will be explained as they are needed.

Properties | Registers

Selection: AND Gate

VHDL	Verilog
Facing	East
Data Bits	1
Gate Size	Narrow
Number Of Inputs	2
Output Value	0/1
Label	
Label Font	SansSerif Bold 16
Negate 1 (Top)	No
Negate 2 (Bottom)	No

Figure 1.3: AND Gate Properties

The outputs of the two `AND` gates need to be combined with an `OR` gate. Add an `OR` gate as illustrated in Figure 1.4.

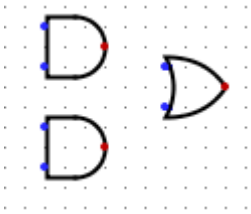

Figure 1.4: OR Gate Added to Circuit

The top input for the first `AND` gate needs two `NOT` gates (inverters) so the two `AND` gates can function as on/off switches. This is a rather common digital logic construct and when the circuit is complete it will become clear how the switching function works.

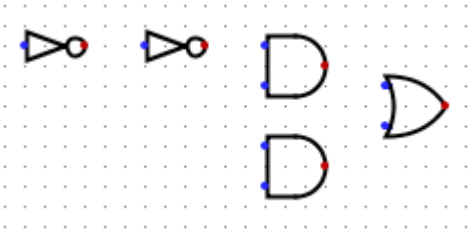

Figure 1.5: Two NOT Gates Added to Circuit

All inputs and outputs need to be added as in Figure 1.6. Note: inputs are square and outputs are round. The *Label* property for each input and output should be specified as in the figure. The pins are labeled according to their function in the circuit. Pin *Sel* carries a signal that selects which input to connect to the output, pins *In1* and *In2* are the two inputs, and pin *Out1* is the output. Note: output pins display a blue-colored X until they are actually wired to some device like the `OR` gate in the illustration.

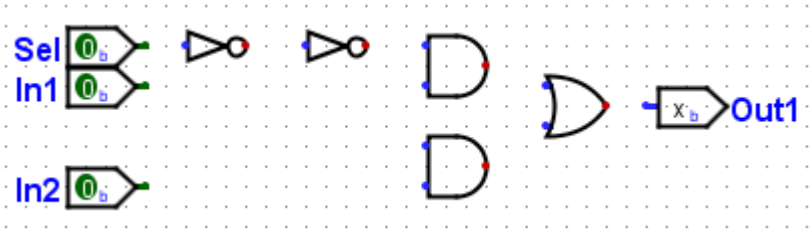

Figure 1.6: Inputs and Output Added

Finally, connect each device with a wire by clicking on the various ports and dragging a wire to the next port. To start the wire in the middle of the two NOT gates click the wire connecting those gates and drag downward. Wires will automatically "bend" one time but to get two bends, like between the output of an `AND` gate and the input of the `OR` gate, click-and-drag the wire from the output of the `AND` gate to a spot a short distance in front of that same gate, then release the mouse button and then immediately click again to start a new wire that will "bend" to the input of the `OR` gate. Only a little practice is needed to master this wiring technique.

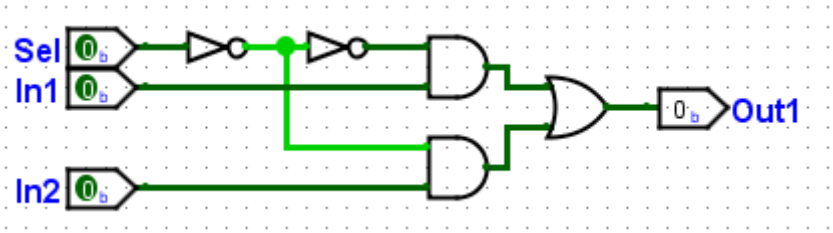

Figure 1.7: Circuit Wiring Added

To operate the circuit in a simulator, click the *Pointing Finger* and "poke" the various inputs. If it is working properly, when the *Sel* input is high then the value of *In2* should be transmitted to the output, but when *Sel* is low then the value of *In1* should be transmitted to the output. This circuit is used to select one of two inputs to be transmitted to the output.

1.2.5 *Identifying Information*

Before finishing, add standard identification information near the top left corner of the circuit using the text tool (the *A* button on the toolbar). That information should include the designer's name, the lab number and circuit name, and the date. Standard identification information for this lab would look like this:

```
George Self
Lab 01: 2-Way, 1-Bit multiplexer
February 13, 2018
```

The font properties in Figure 1.8 have been set to bold and a large size to make the text easier to read.

Note that *Logisim-evolution* will automatically center text in a new box, so text boxes will need to be aligned after they have been created. To align the text boxes, click the *Arrow* tool and use it to drag the boxes to their desired location. The completed circuit should look like Figure 1.8.

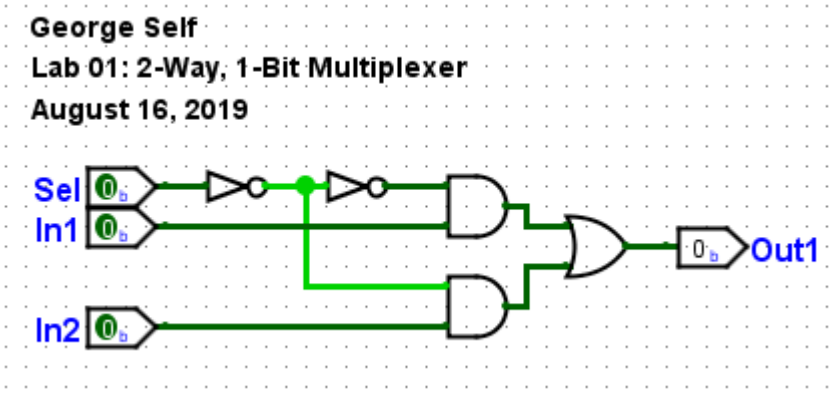

Figure 1.8: Simple multiplexer

1.3 DELIVERABLE

The purpose of this lab is to install and test the *Logisim-evolution* system and become comfortable creating a digital logic circuit.

To receive a grade for this lab, create the Simple Multiplexer as defined in this lab, be sure the standard identifying information is at the top left of the circuit, and then save the file with this name: `Lab01_Intro`. Submit that circuit file for grading.

Part II

THEORY

THEORY exercises are designed to provide practice with simple logic circuits in order to both develop skill with *Logisim-Evolution* and illustrate the foundations of digital logic theory.

2 ADDER

2.1 PURPOSE

This lab builds a full 1-bit adder, but the intent is to continue to familiarize students with *Logisim-Evolution* and how basic arithmetic functions can be completed using simple gate-level logic. Additionally, this lab develops an automated testing system that will be used to test future lab submissions.

2.2 PROCEDURE

2.2.1 *Half-Adder*

Open Logisim and start a new project. In *Logisim-Evolution* circuits can contain any number of sub-circuits. Subcircuits fill the same role in a physical circuit as a function or procedure fills in a software project. A new subcircuit can be added to a circuit by clicking PROJECT -> ADD CIRCUIT. Name the new circuit `Half_Adder`. Open the new subcircuit by double-clicking its name in the Explorer Pane.

Because this is a new subcircuit, the drawing canvas is blank. A half-adder is a circuit that will add two input bits. Because it is possible to add two bits and generate a carry, the half-adder must allow for a carry out bit. Figure 2.1 is the circuit diagram for a half-adder.

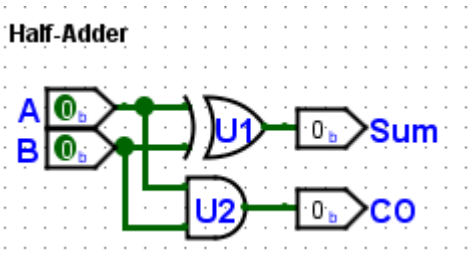

Figure 2.1: Half-Adder

This is fairly easy to build. Component `U1` is an `XOR` gate and `U2` is an `AND` gate. There are two inputs and two outputs and all of the devices should be connected as shown. To test the circuit, whenever input *A* and input *B* are different the *Sum* should be high and when input *A* and input *B* are the same the *CO* (Carry Out) bit should be high.

2.2.2 *Full Adder*

A full adder takes two input bits and adds them, like a half-adder, but it also includes the logic necessary to input or output a carry bit so it can be cascaded with other adders. Figure 2.2 is the logic diagram for a full adder.

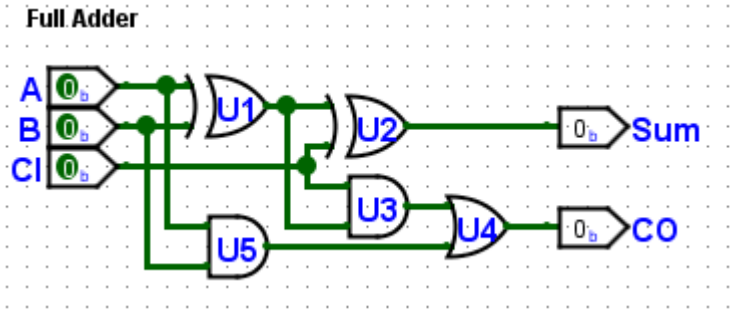

Figure 2.2: Full Adder

Create a new subcircuit named `Full_Adder` and wire all of the components as illustrated in Figure 2.2. Notice that `U1` and `U2` are `XOR` gates, `U3` and `U5` are `AND` gates, and `U4` is an `OR` gate. There are also three inputs and two outputs.

2.2.3 *Main Circuit*

In most *Logisim-Evolution* projects, the `main` circuit is used to provide a nice user interface for the project. Typically, various subcircuits are dropped onto the main circuit canvas and various inputs and outputs are wired to them. A user can then test the project without worrying about the details of the subcircuits.

For this project, open the `main` circuit by double-clicking its name in the Library panel. Click one time on the `Full_Adder` subcircuit then move the mouse over the drawing canvas. Notice that the mouse pointer has changed into a representation of the subcircuit. Drop that subcircuit anywhere on the drawing canvas and then wire the various inputs and outputs as shown in Figure 2.3.

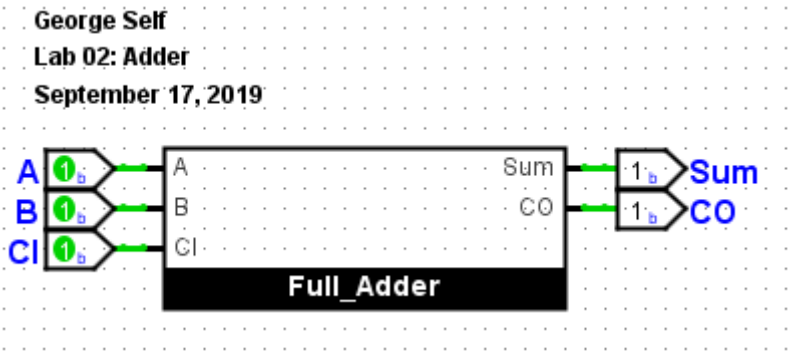

Figure 2.3: Full Adder

This circuit can be tested by using the *poke* tool and entering these values. After each line, check to see that the outputs are correct.

Inputs			Outputs	
A	B	CI	Sum	CO
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

Table 2.1: Test Vector For Full Adder

2.2.4 *Automated Testing*

While it is possible to use the *poke* tool and check the outputs for various input combinations, as digital logic circuits become more complex it is important to automate the testing process so no input combinations are overlooked. *Logisim-Evolution* includes a SIMULATE -> TEST VECTOR feature that is used for automating circuit testing.

The first step in using automatic testing is to create a *Test Vector* file. This is a simple *.txt* file that can be created in any text processor, like *Notepad*. The format for a test vector is fairly simple.

Do not use a word processor to create the Test Vector since that would add unneeded codes for things like fonts and margins.

- Every line is a single test of the circuit, except the first line.
- The first line defines the various inputs and outputs being tested.
- Any line that starts with a hash mark (#) is a comment and is ignored.

Following is the test vector file used to test the `main` circuit.

```
1  # Test vector for Adder
2  CI A B Sum CO
3  0 0 0 0 0
4  0 0 1 1 0
5  0 1 0 1 0
6  0 1 1 0 1
7  1 0 0 1 0
8  1 0 1 0 1
9  1 1 0 0 1
10 1 1 1 1 1
```

Following is an explanation for the *Test vector for Adder* file.

LINE 1 This is just the title of the file. Because this line starts with a hash (#) it is a comment and will be ignored by *Logisim-Evolution* .

LINE 2 This line lists all of the inputs and outputs in the circuit under test. In this case, there are three inputs, *CI*, *A*, and *B*, along with two outputs, *Sum* and *CO*. *Logisim-Evolution* is able to determine whether the pin is an input or output from its properties.

LINE 3 This line contains the first test for the circuit. This line specifies that *Logisim-Evolution* make *CI*, *A*, and *B* equal to zero and then check to be certain that *Sum* and *CO* are also zero.

OTHER LINES All other lines set the three input bits and specify the expected response in the output bits.

To start a test, click SIMULATE -> TEST VECTOR. The window illustrated in Figure 2.4 opens.

Figure 2.4: Test Vector Window

Click the *Load Vector* button at the bottom of the window and load the test vector file. The test will automatically start and *Logisim-Evolution* will report the results, like in Figure 2.5.

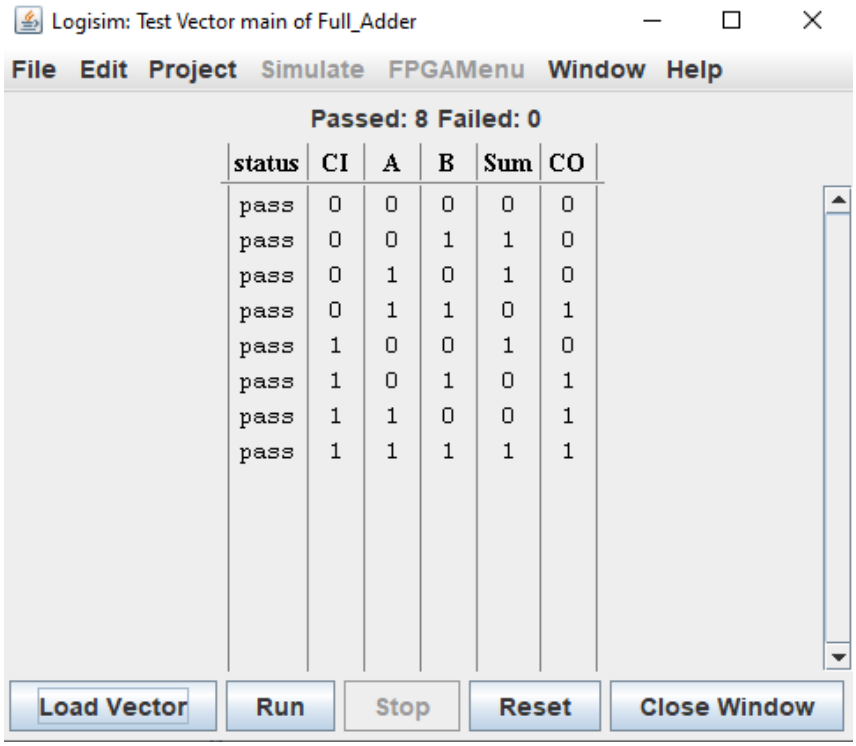

Figure 2.5: Test Completed

An error was intentionally added to the test vector file to generate a failed test.

The test indicates all 8 lines passed and zero failed so it could be reasonably concluded that the circuit is functioning properly. Figure 2.6 illustrates a failed test. The circuit designer would then need to troubleshoot to determine what went wrong with the circuit.

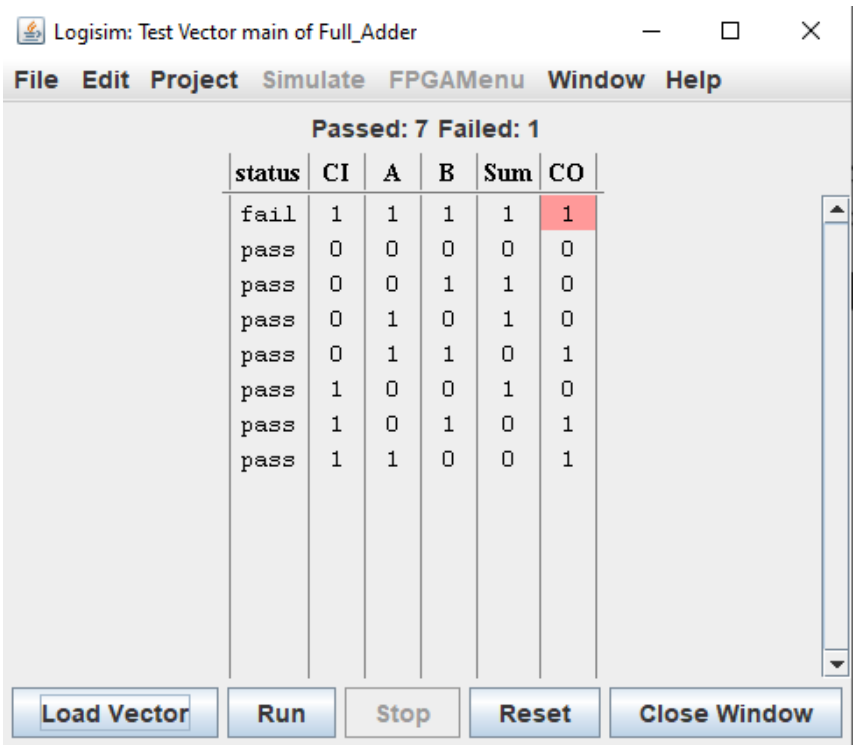

Figure 2.6: Test Failure

Note: the test vector files for all labs are made available to students so they can check their work prior to submitting them.

2.3 DELIVERABLE

To receive a grade for this lab, build both the half-adder and full adder and then add the full adder to the `main`. It is important to ensure the input and output pin names are the same as in the lab instructions since a test vector will be used to check the circuit. Be sure the standard identifying information is at the top left of the `main` circuit, similar to:

```
George Self
Lab 02: Adder
September 17, 2019
```

Save the file with this name: *`Lab02_Adder`* and submit that file for grading.

3 ARITHMETIC OPERATIONS

3.1 PURPOSE

This lab develops an arithmetic unit that includes eight different arithmetic functions using *Logisim-Evolution* library items. This device will eventually be used as part of the Arithmetic Logic Unit (ALU) in Lab 7. This device will have two inputs, labeled A and B, and will output the following calculations.

1. -1
2. $A - 1$
3. $A + B$
4. $A - B$
5. $AB - 1$
6. $AB' - 1$
7. $A + A$
8. $A + 1$

3.2 PROCEDURE

Start a new *Logisim-Evolution* project and create a subcircuit named `arithmetic` that will eventually contain the entire arithmetic unit. Begin the build by adding eight devices in the subcircuit as in Figure 3.1. *Notes: exact device placement is not important at this point since they can be repositioned as necessary; however, they should be in the correct order on the subcircuit. Also, all of the devices, inputs, and outputs need to be set for eight data bits in the properties panel.*

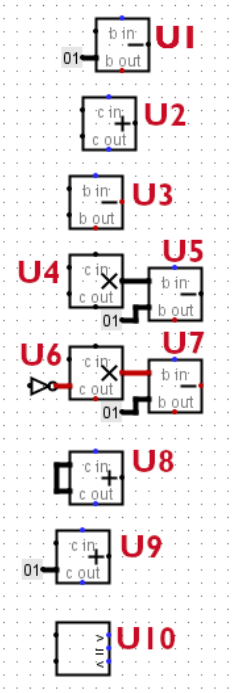

Figure 3.1: Placing the Arithmetic Components

Here are the devices in Figure 3.1. *Note: the device numbers were added to the illustration as an aid for the following discussion; they will not be present in the Logisim-Evolution subcircuit.*

- U1-Subtractor (*Arithmetic* library). This device subtracts the bottom input from the top input on its east side and sends the result to the output on its west side.
- Constant (*Wiring* library). Subtractor *U1* is intended to supply the *A-1* output and the "01" on its bottom input is a constant used in this calculation. It should be placed near the bottom input for subtractor *U1* and its properties set for Facing: East, Data Bits: 8, Value: 0x1 (this means hexadecimal 1).
- U2-Adder (*Arithmetic* library). This device adds the top and bottom inputs on its east side and sends the result to the output on its west side.
- U3-Subtractor (*Arithmetic* library). This device subtracts the bottom input from the top input on its east side and sends the result to the output on its west side.
- U4-Multiplier (*Arithmetic* library). This device multiplies the top and bottom inputs on its east side and sends the result to the output on its west side.
- U5-Subtractor (*Arithmetic* library). This device is connected to multiplier U4 output and is intended to subtract one from its product.

- Constant (*Wiring* library). Because subtractor U5 is designed to subtract one from the product of U4, the "01" on its bottom input is a constant. It should be placed near the bottom input and its properties set for Facing: East, Data Bits: 8, Value: 0x1 (this means hexadecimal 1).
- U6-Multiplier (*Arithmetic* library). This device multiplies the top and bottom inputs on its east side and sends the result to the output on its west side.
- Not Gate (*Gates* library). This is placed near the bottom input of multiplier U6 in order to negate that input.
- U7-Subtractor (*Arithmetic* library). This device is connected to multiplier U6 output and is intended to subtract one from its product.
- Constant (*Wiring* library). Because subtractor U7 is designed to subtract one from the product of U6, the "01" on its bottom input is a constant. It should be placed near the bottom input and its properties set for Facing: East, Data Bits: 8, Value: 0x1 (this means hexadecimal 1).
- U8-Adder (*Arithmetic* library). This device is designed to output the value of $A + A$ so the two inputs on its east side are tied together. The result is sent to the output on its west side.
- U9-Adder (*Arithmetic* library). This device adds the top and bottom inputs on its east side and sends the result to the output on its west side.
- Constant (*Wiring* library). Adder *U9* is intended to supply the *A+1* output and the "01" on its bottom input is a constant. It should be placed near the bottom input for adder *U9* and its properties set for Facing: East, Data Bits: 8, Value: 0x1 (this means hexadecimal 1).
- U10-Comparator (*Arithmetic* library). A comparator compares the two inputs on its east side. If the top input is greater than the bottom input then output ">" will go high. If the top and bottom inputs are equal then output "=" will go high. If the top input is less than the bottom input then output "<" will go high.

The next step is to place all of the inputs and outputs. Figure 3.2 shows where those items should go and the label for each item. Note: exact placement is not important since they can be repositioned. The *Radix* property for each of the inputs and outputs should be set to "Hexadecimal." The *Data Bits* property should be set as follows.

- CI: 1
- A: 8
- B: 8
- ArOut: 8
- CO: 1
- Cmp: 1

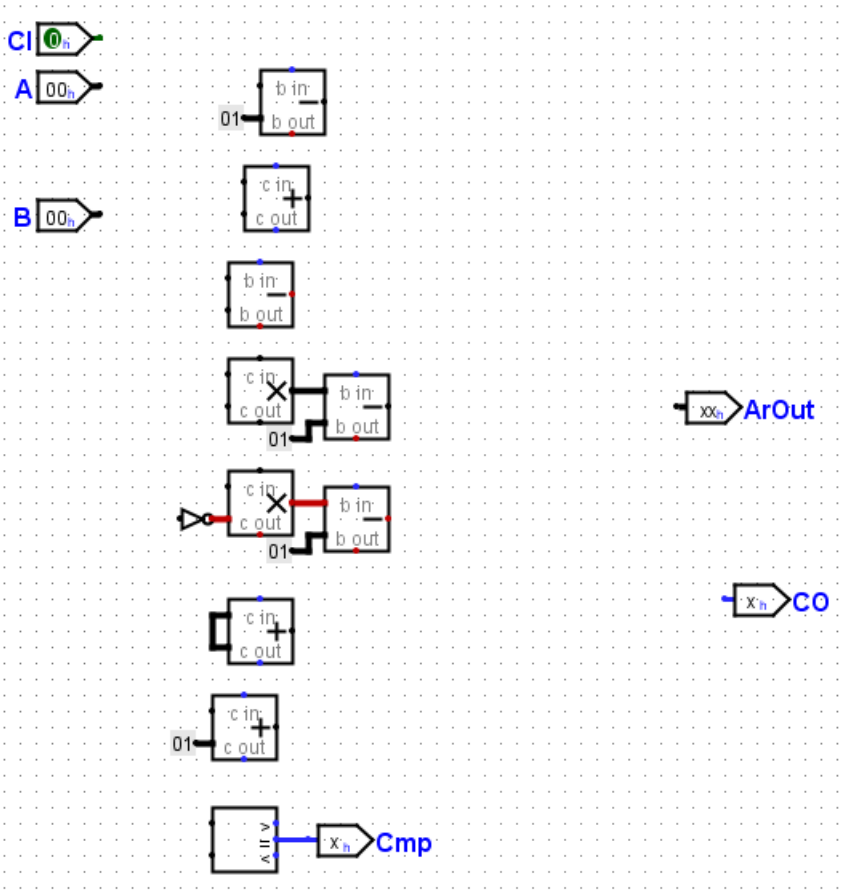

Figure 3.2: Arithmetic Inputs and Outputs

This circuit uses a Multiplexer (*Plexers* library) to select which device's output to connect to *ArOut*. A multiplexer is a digital logic workhorse that is found in many circuits, including Central Processing Units (CPUs). It is designed to switch a selected input to the output while ignoring all other input ports. In Figure 3.3, two multiplexers have been placed in the circuit. The top multiplexer will select one of eight inputs coming from the eight arithmetic devices to connect to *ArOut*. The bottom multiplexer will connect the carry out signal from the selected arithmetic device to the *CO* output. Also notice that the bottom multiplexer has a constant zero wired to input port zero.

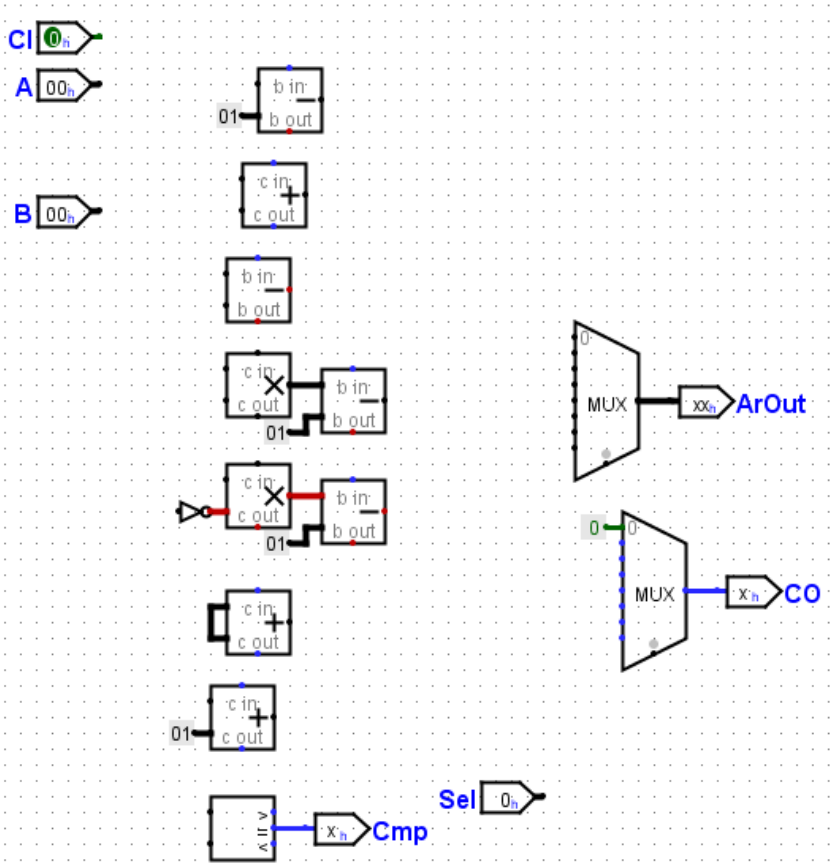

Figure 3.3: Placing the Multiplexers

The next step is to wire inputs *A* and *B* to each of the devices, as shown in Figure 3.4. Then the outputs of each device is wired to an appropriate port in the top multiplexer. This is not particularly challenging, but be careful to avoid crossed wires.

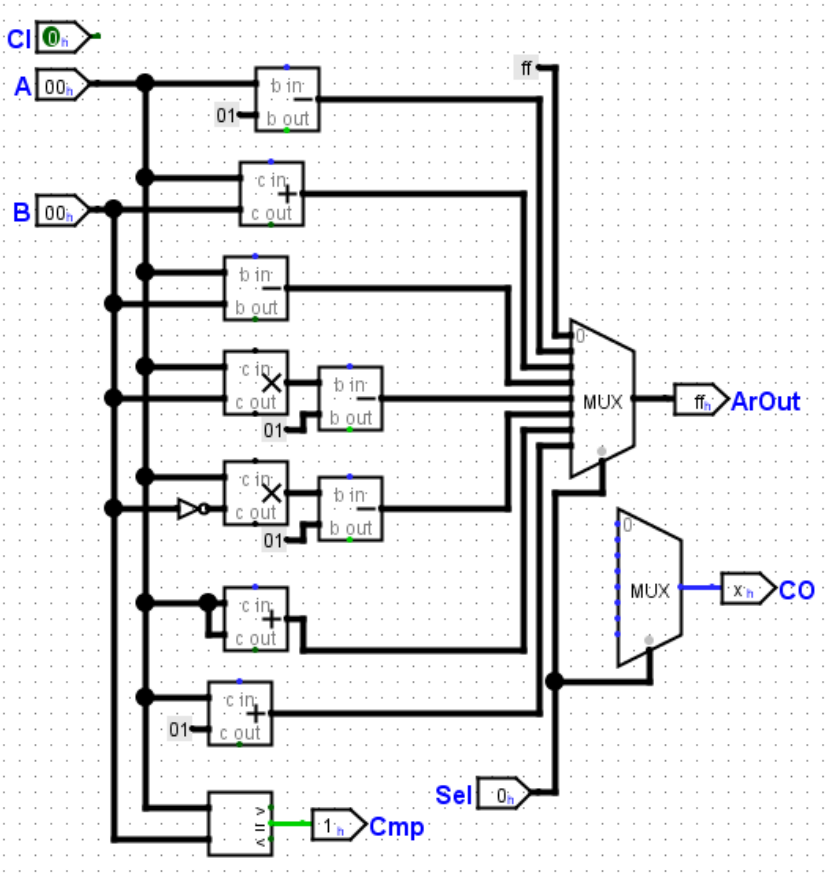

Figure 3.4: Wiring the Data Inputs and Outputs

Figure 3.4 shows the completed subcircuit with wires connecting *CI* to each device and then the devices wired to the appropriate port on the bottom multiplexer.

Notice that input zero on the top multiplexer is wired to a constant *ff*. That is the two's complement of negative one so that port will always transmit negative one, as in the specification sheet.

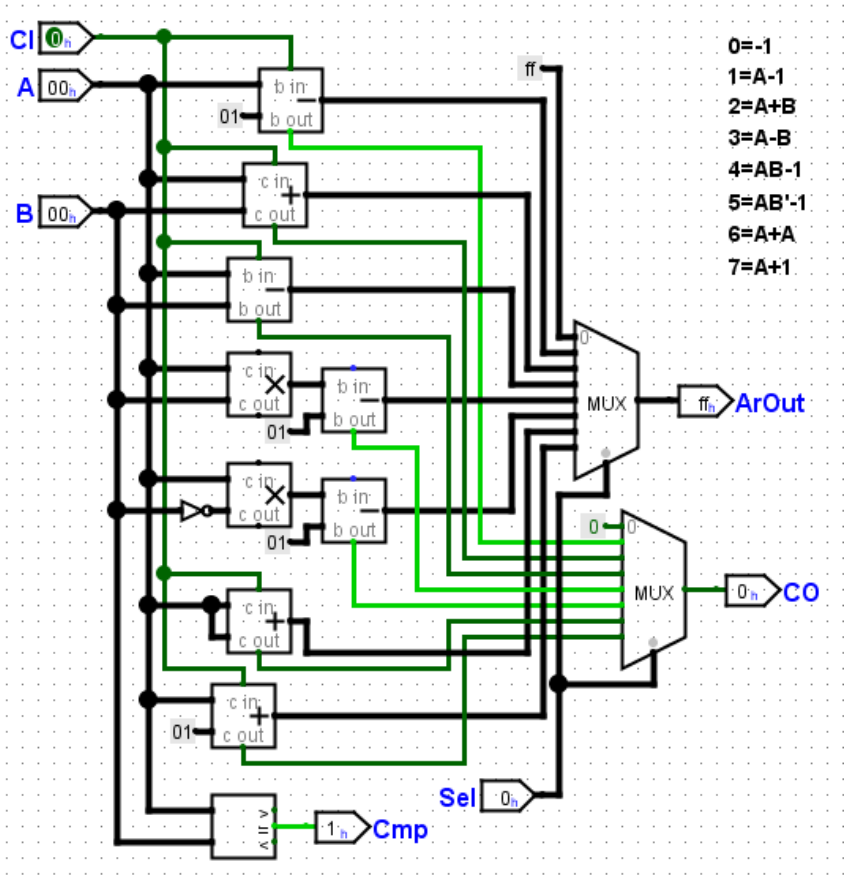

Figure 3.5: Arithmetic Final Circuit

Finally, the `main` circuit is completed by dropping the `arithmetic` subcircuit on the canvas and wiring an appropriate input or output port to each of the ports on the subcircuit. Figure 3.6 illustrates the main circuit.

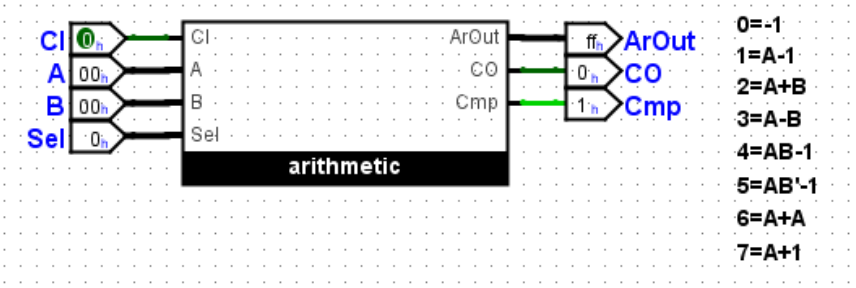

Figure 3.6: Arithmetic Main Circuit

This arithmetic device can be tested by entering numbers for the various inputs and checking to see if the output is correct. For example, if input A was set to 3 and input B were set to 4, then if *Sel* is set to 2, *ArOut* should be seven $(3 + 4)$. A test vector file has been provided for this lab so students can use that file to exercise all of the various settings in the circuit.

3.3 DELIVERABLE

To receive a grade for this lab, complete the circuit. Be sure the standard identifying information is at the top left of the `main` circuit, similar to:

```
George Self
Lab 03: Arithmetic Operations
September 17, 2019
```

Save the file with this name: *`Lab03_Arithmetic`* and submit that file for grading.

4 LOGIC OPERATIONS

4.1 PURPOSE

This lab develops a logic unit that includes eight different logic functions using *Logisim-Evolution* library items. This device will eventually be used as part of the Arithmetic Logic Unit (ALU) in Lab 7. This device will have two inputs, labeled *A* and *B*, and will output the following logic values.

1. AB
2. $(AB)'$
3. $A + B$
4. $(A + B)'$
5. AXorB
6. AB'
7. $A + B'$
8. A'

4.2 PROCEDURE

This circuit is very similar to the arithmetic circuit developed in Lab 3. However, the logic circuit is much simpler than the arithmetic circuit since there is not carry in/out bit and no comparator output.

To complete the lab, create a subcircuit named `logic`. Place appropriate devices from the *Gates* library in the `logic` subcircuit. Connect each of the devices to inputs *A* and *B* and then wire the outputs from each device to *LoOut* through a multiplexer. The exact design of the `logic` subcircuit is left to the student.

Drop the `logic` subcircuit on the `main` circuit and wire the various inputs and outputs, as shown in Figure 4.1.

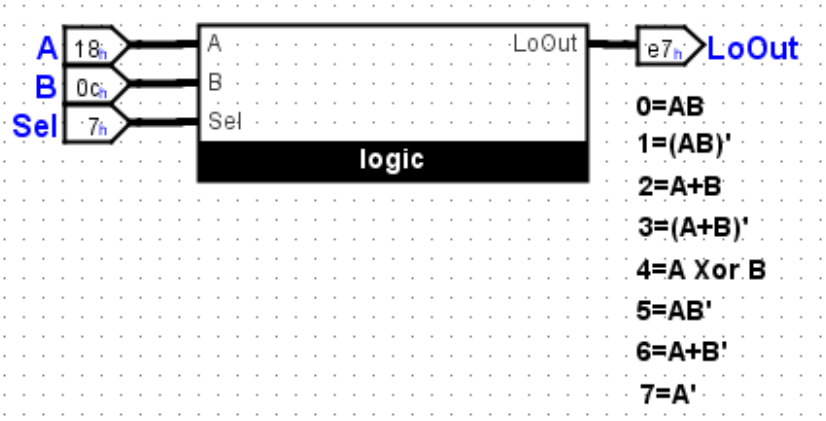

Figure 4.1: The Main Logic Circuit

The circuit can be tested by using the *poke* tool and entering various inputs and then checking to see that the output is correct. A test vector files is also provided with the lab and that can be used to ensure the subcircuit is correct.

4.3 DELIVERABLE

To receive a grade for this lab, complete the circuit. Be sure the standard identifying information is at the top left of the `main` circuit, similar to:

```
George Self
Lab 04: Logic Operations
September 17, 2019
```

Save the file with this name: *`Lab04_Logic`* and submit that file for grading.

5 BOOLEAN LOGIC

5.1 PURPOSE

This lab has three goals:

- Design circuits when given a Boolean expression.
- Create subcircuits.
- Create and exercise a test of the subcircuits.

Logisim-Evolution permits designers to work with a main circuit and any number of subcircuits. Students who have studied programming languages are familiar with "functions" or "classes" that can be designed and built one time and then reused many times whenever they are needed. *Logisim-Evolution* permits that same type of modular design by using subcircuits.

The *Logisim-Evolution* starter for this lab includes a `main` circuit and one subcircuit, named `Equation_1`. The starter subcircuit is used to practice creating a circuit from a Boolean expression and then a new subcircuit is added and a second Boolean expression is used to build that circuit.

5.2 PROCEDURE

5.2.1 *Subcircuit: Equation 1*

A magnifying glass icon is used to indicate which circuit is active on the drawing canvas.

The starter circuit includes a subcircuit named `Equation_1`. Double-click that circuit in the Explorer Pane to activate it. The drawing canvas for this subcircuit is mostly blank except for a Boolean expression: $(A'BC') + (AB'C') + (ABC)$. Before starting to design a circuit, it is helpful to take a minute to analyze the expression.

- There are only three variables used in the entire expression: *A*, *B*, and *C*. Therefore, there would be three inputs into the circuit.
- There are three groups of variables and within each group the variables are joined with an `AND`. Therefore, the circuit must include three `AND` gates with three inputs for each gate.
- The three groups of variables are joined with an `OR`. Therefore, the circuit must include an `OR` gate with three inputs.

- While the expression does not name an output variable, it is reasonable to assume that the circuit would output a logic 1 or 0. Therefore, a one-bit output variable must be specified.

Do not be concerned with the exact placement of components on the drawing canvas. They can be rearranged as the build progresses.

Start by placing three inputs and an output on the drawing canvas. Inputs are indicated by a green icon with *I->* on the tool bar above the drawing canvas. Click that tool and place three input pins named *In1A*, *In1B*, and *In1C* —that means "Input for Equation One, variable A" and so forth.

Outputs are indicated by a white icon with *->O* found on the tool bar above the drawing canvas. Click that tool and place an output named *Out1*. The circuit should look like Figure 5.1.

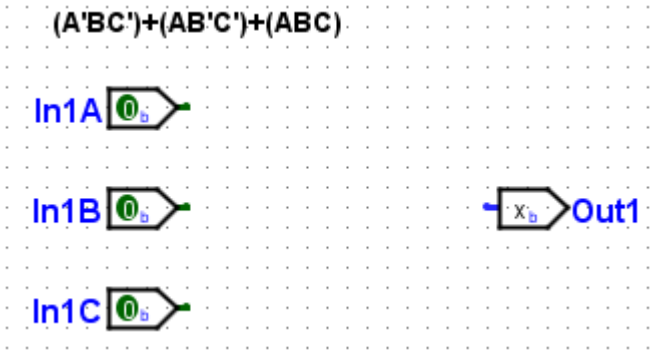

Figure 5.1: Equation 1 Inputs-Outputs

The gates in this manual are all "narrow" size. The size does not change the gate behavior but makes it easier to wire the complex circuits in later labs.

Next, the gates should be added. The `AND` gate tool can be found on the tool bar. Click that tool and place three `AND` gates on the circuit. Click each gate and in its properties panel set the *Number of Inputs* to 3.

The `OR` gate tool can be found on the tool bar. Click that tool and place one `OR` gate on the circuit. Click that gate and in its properties panel set the *Number of Inputs* to 3.

The circuit should look like Figure 5.2.

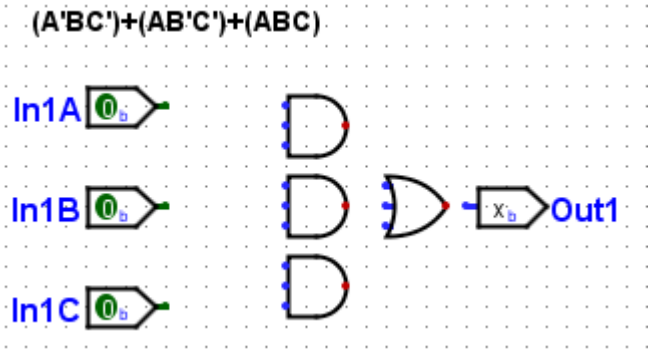

Figure 5.2: Equation 1 And-Or Gates

Next, the inputs for the AND gates should be set to match the Boolean expression. The top AND gate will match the first group of inputs, $(A'BC')$, so inputs A and C should be negated. To negate those two inputs, click the AND gate and in the properties panel set the *Negate* item for the top and bottom input to "Yes." When that is done, the two inputs on the AND gate should include a small "negate" circle.

In the same way, the middle and bottom input for the second AND gate should also be negated. The circuit should look like Figure 5.3.

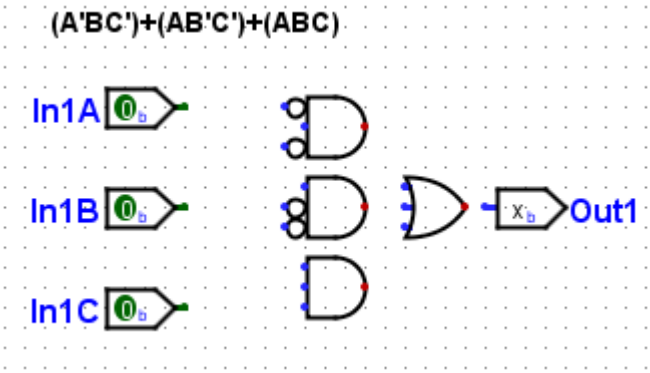

Figure 5.3: Equation 1 And Gate Inputs Set

Finally, connect all gates with wires, like Figure 5.4.

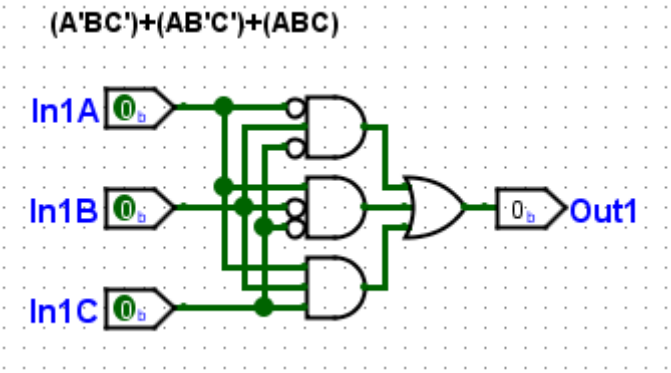

Figure 5.4: Equation 1 Circuit Completed

Test the circuit by selecting the *poke* tool in the tool bar (it looks like a pointing finger) and setting various combinations of 1 and 0 on the three inputs. The output pin should go high only when the inputs are set to $(A'BC')$, $(AB'C')$, or (ABC).

5.2.2 *Subcircuit: Equation 2*

A new subcircuit can be added to a circuit by clicking PROJECT -> ADD CIRCUIT. Name the new circuit `Equation_2`. Open the new subcircuit by double-clicking its name in the Explorer Pane.

Because this is a new subcircuit, the drawing canvas is blank. To start this subcircuit, write the equation for the circuit near the top of the drawing canvas by clicking the "A" button on the Toolbar and then clicking near the top of the drawing canvas and typing the following:

$$(A'B'CD') + (A'BCD) + (AB'CD') + (ABCD')$$

It will save time to take a few minutes and analyze the expression.

- There are only four variables used in the entire expression: *A*, *B*, *C*, and *D*. Therefore, there would be four inputs into the circuit.
- There are four groups of variables and within each group the variables are joined with an `AND`. Therefore, the circuit must include four `AND` gates with four inputs for each gate.
- The four groups of variables are joined with an `OR`. Therefore, the circuit must include an `OR` gate with four inputs.
- While the expression does not name an output variable, it is reasonable to assume that the circuit would output a logic 1 or 0. Therefore, a one-bit output variable must be specified.

Design the subcircuit using these names for the inputs: *In2A*, *In2B*, *In2C*, and *In2D*. Also include an output named *Out2*. Set the `AND` gates so the their inputs are negated properly and then wire the entire subcircuit. Finally, test the circuit to ensure the output goes high only when the four specified combinations of inputs are present.

5.2.3 *Main Circuit*

Make the `main` circuit active by double-clicking its name in the Explorer Panel. Click once on the `Equation_1` circuit and the cursor will change into an image of that circuit as it will appear on the drawing canvas. Click on the drawing canvas to drop that subcircuit. The circuit can later be moved by clicking it and dragging it to a new location. Wire the three inputs and output as shown in Figure 5.5. Notice that the input/output pins do not need to be named the same as in the subcircuit; for example, the output for `Equation_1` is labeled *Out1* but it is connected to an output pin labeled *True1*.

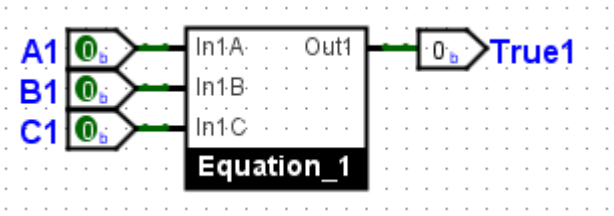

Figure 5.5: Main Circuit

Add the `Equation_2` circuit in the same way and wire four inputs and one output to that circuit. The inputs should be labeled *A2*, *B2*, *C2*, and *D2* and the output labeled *True2*.

5.3 DELIVERABLE

To receive a grade for this lab, complete the `main` circuit and both subcircuits. Be sure the standard identifying information is at the top left of the `main` circuit, similar to:

It is important to name all inputs and outputs as specified in the lab since they are checked with a Test Vector file that depends on those names.

```
George Self
Lab 05: Boolean Equations
February 18, 2018
```

Save the file with this name: *`Lab05_Bool`* and submit that file for grading.

6 PROGRAMMABLE LOGIC ARRAY

6.1 PURPOSE

This lab explores using a Programmable Logic Array (PLA) to simplify circuits that are designed for Boolean operations.

6.2 PROCEDURE

Lorem

6.3 DELIVERABLE

To receive a grade for this lab, complete the circuit. Be sure the standard identifying information is at the top left of the `main` circuit, similar to:

```
George Self
Lab 06: PLA
September 17, 2019
```

Save the file with this name: *Lab06_PLA* and submit that file for grading.

Part III

PRACTICE

Practice exercises are designed to familiarize students with many aspects of both combinational and sequential digital logic circuits. This section develops devices as varied as counters, encoders, and read-only memory. It also includes a rather complex

7

ARITHMETIC LOGIC UNIT (ALU)

7.1 PURPOSE

In this lab you will build an Arithmetic Logic Unit (ALU). An ALU is an important digital logic device used to perform all sorts of arithmetic and logic functions in a circuit. The commercial 74181 ALU has two four-bit data inputs along with a one-bit mode (M) and a four-bit select input. Depending on those settings, the device will complete one of the functions listed in Table 7.1.

Select	Logic (M=1)	Arithmetic (M=0)
0000	A'	A
0001	(A + B)'	A + B
0010	A'B	A + B'
0011	Logical 0	minus 1 (2's Comp)
0100	(AB)'	A + AB'
0101	B'	(A + B) plus AB'
0110	A XOR B	A minus B minus 1
0111	AB'	AB' minus 1
1000	A' + B	A plus AB
1001	(A XOR B)'	A plus B
1010	B	(A + B') plus AB
1011	AB	AB minus 1
1100	Logical 1	A plus A
1101	A + B'	(A + B) plus A
1110	A + B	(A + B') plus A
1111	A	A minus 1

Table 7.1: Function Table for 74181 ALU

Notes: in the "Arithmetic" column, the + sign indicates logic *OR* while the words *plus* and *minus* indicate arithmetic add and subtract operations. The value of *A plus A* is the same as shifting the bits left to the next most significant position.

The ALU built in this lab is not as complex as a 74181 Integrated Circuit (IC), however it demonstrates the basic functions of an ALU.

7.2 PROCEDURE

This is a rather complex circuit so several completed subcircuits are provided.

Load the ALU starter circuit in *Logisim-evolution*. That starter circuit already has the `main`, `ALU`, and `Arithmetic` subcircuits completed.

7.2.1 *main*

The `main` circuit does nothing more than provide a human-friendly interface for the rest of the ALU. That interface include two four-bit inputs (labeled *InA* and *InB*), a three-bit select, a one-bit mode, a carry-in and carry-out bit (so the ALU could be chained to another to create an eight-bit device), a *compare* output (TRUE if the two inputs are equal), and a four-bit output (labeled *ALUOut*). In operation, numbers are entered on *InA* and *InB*, the mode and select are set, and then the result is read on *ALUOut*.

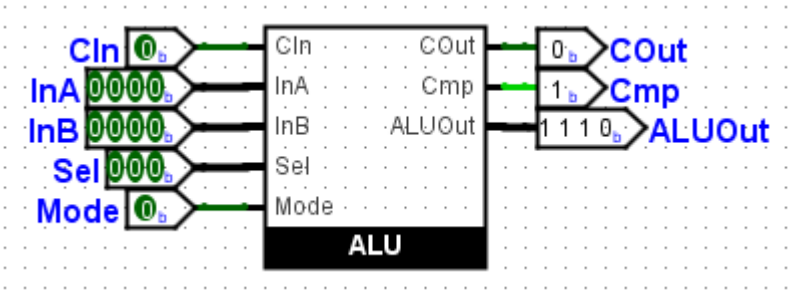

Figure 7.1: ALU main

7.2.2 *ALU*

The `ALU` subcircuit contains the logic that routes *InA*, *InB*, and *Sel* to two other subcircuits, `Arithmetic` or `Logic`. It then uses a multiplexer to route the output of one of those subcircuits to an output port depending on the setting of the *Mode* bit. Note that the inputs are sent to both subcircuits but only the output specified by the *Mode* is returned to the user. This type of logic is also used in the `Arithmetic` circuit.

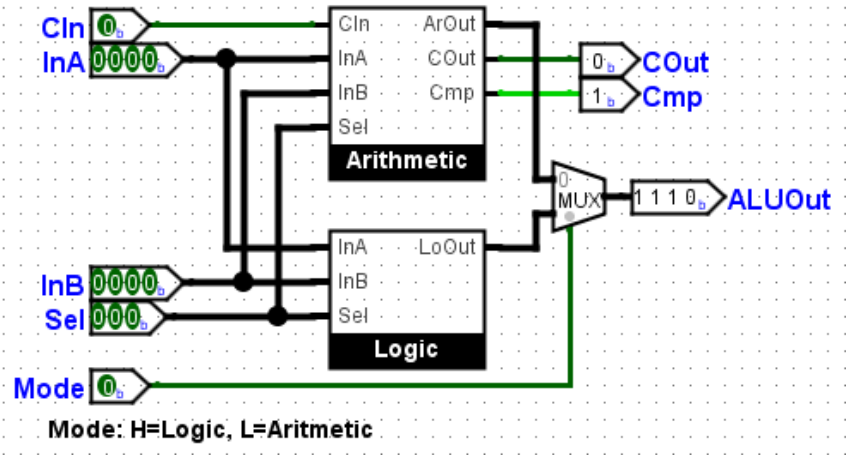

Figure 7.2: ALU Subcircuit

7.2.3 *Arithmetic*

This subcircuit contains numerous devices from the *Arithmetic* library and they are all wired appropriately for whatever operation is selected. The concept for this subcircuit is rather simple but routing the wiring to all of the devices is challenging.

Notice that two multiplexers are necessary since the circuit provides two different outputs. The top multiplexer routes the four-bit solution and the bottom multiplexer routes the carry-out bit. The *compare* output is always active since it is comparing the input signals and does not rely on the function that is selected.

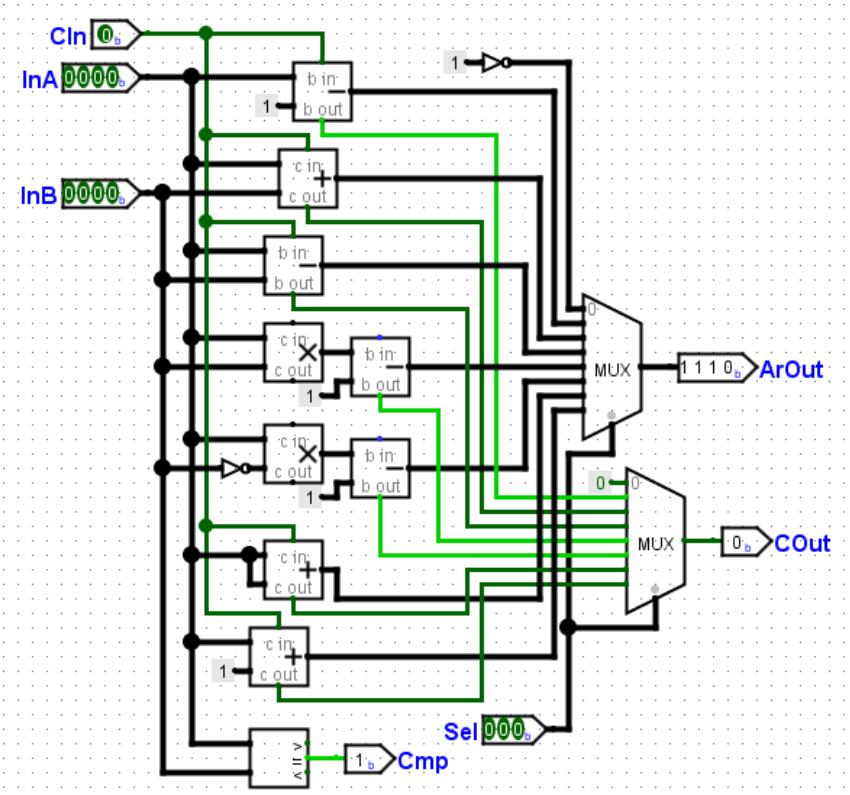

Figure 7.3: Arithmetic Subcircuit

7.2.4 *Challenge*

In the starter circuit, the `Logic` subcircuit is only a shell with three inputs and one output.

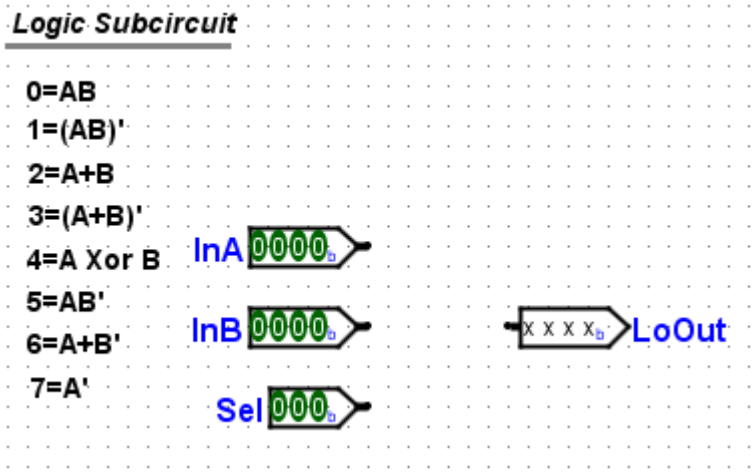

Figure 7.4: Logic Subcircuit

Complete that subcircuit by adding the necessary logic gates and wiring, similar to the `Arithmetic` subcircuit. This subcircuit is much

simpler than the `Arithmetic` subcircuit since there are no carry-in, carry-out, or compare bits. When completed, the subcircuit only needs eight logic gates and a multiplexer added to the starter.

7.2.5 *Testing the Circuit*

The `ALU` should be tested by entering several values on *InA* and *InB* and then select all possible arithmetic and logic operations. The outputs for each check should be accurate.

7.3 DELIVERABLE

To receive a grade for this lab, complete the Challenge. Be sure the standard identifying information is at the top left of the *main* circuit, similar to this:

```
George Self
Lab 08: ALU
February 18, 2018
```

Save the file with this name: *`Lab08_ALU`* and submit that file for grading.

Part IV

SIMULATION

SIMULATION is the most complex topic covered in this lab manual. Included in this manual are a vending machine simulator, a simple processor, designed to teach the foundations of a Central Processing Unit, and an elevator simulator, designed to be a capstone project.

Part V

APPENDIX

A

TTL REFERENCE

Logisim-Evolution includes a number of Transistor-Transistor Logic (TTL) ICs. These are pre-packaged digital logic circuits that perform specific, well-defined functions. There are, literally, hundreds of TTL ICs available for purchase from electronics warehouses but *Logisim-Evolution* includes only 35 of the most commonly-used devices. Figure A.1 shows three surface-mounted ICs on a circuit board.

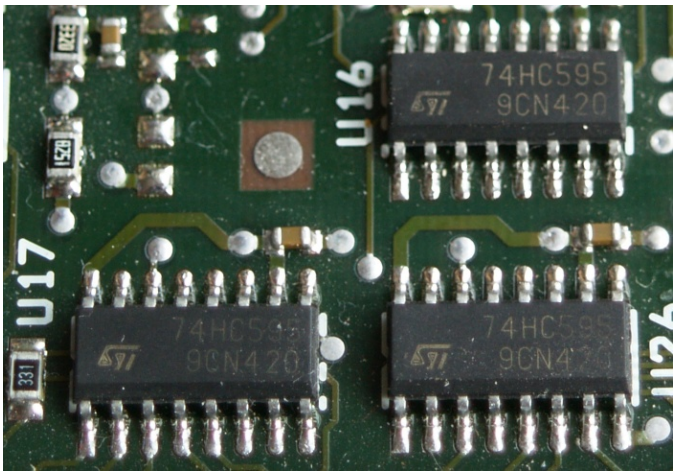

Figure A.1: Three Surface-Mounted Integrated Circuits

A.1 7400: QUAD 2-INPUT NAND GATE

This device contains four independent 2-input NAND gates. Figure A.2 is a logic diagram of one of the four circuits.

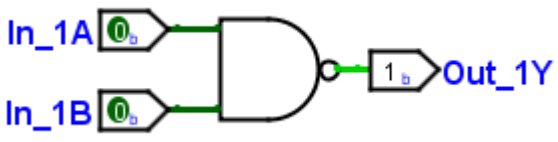

Figure A.2: 7400: Single NAND Gate Circuit

The 7400 device in *Logisim-Evolution* uses the wiring connections indicated in Table A.1.

Logisim Label	Function
Input: 1	In 1A
Input: 2	In 1B
Output: 3	Out 1Y
Input: 4	In 2A
Input: 5	In 2B
Output: 6	Out 2Y
Output: 8	Out 3Y
Input: 9	In 3A
Input: 10	In 3B
Output: 11	Out 4Y
Input: 12	In 4A
Input: 13	In 4B

Table A.1: Pinout For 7400

A.2 7402: QUAD 2-INPUT NOR GATE

This device contains four independent 2-input NOR gates. Figure A.3 is a logic diagram of one of the four circuits.

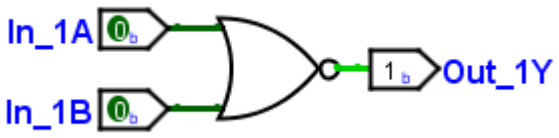

Figure A.3: 7402: Single NOR Gate Circuit

The 7402 device in *Logisim-Evolution* uses the wiring connections indicated in Table A.2.

Logisim Label	Function
Input: 1	In 1A
Input: 2	In 1B
Output: 3	Out 1Y
Input: 4	In 2A
Input: 5	In 2B
Output: 6	Out 2Y
Output: 8	Out 3Y
Input: 9	In 3A
Input: 10	In 3B
Output: 11	Out 4Y
Input: 12	In 4A
Input: 13	In 4B

Table A.2: Pinout For 7402

A.3 7404: HEX INVERTER

This device contains six independent inverters. Figure A.4 is a logic diagram of one of the six circuits.

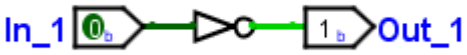

Figure A.4: 7404: Single Inverter Circuit

The 7404 device in *Logisim-Evolution* uses the wiring connections indicated in Table A.3.

Logisim Label	Function
Input: 1	In 1
Output: 2	Out 1
Input: 3	In 2
Output: 4	Out 2
Input: 5	In 3
Output: 6	Out 3
Output: 8	Out 4
Input: 9	In 4
Output: 10	Out 5
Input: 11	In 5
Output: 12	Out 6
Input: 13	In 6

Table A.3: Pinout For 7404

A.4 7408: QUAD 2-INPUT AND GATE

This device contains four independent 2-input AND gates. Figure A.5 is a logic diagram of one of the four circuits.

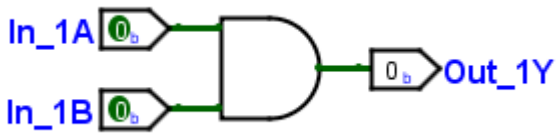

Figure A.5: 7408: Single AND Gate Circuit

The 7408 device in *Logisim-Evolution* uses the wiring connections indicated in Table A.4.

Logisim Label	Function
Input: 1	In 1A
Input: 2	In 1B
Output: 3	Out 1Y
Input: 4	In 2A
Input: 5	In 2B
Output: 6	Out 2Y
Output: 8	Out 3Y
Input: 9	In 3A
Input: 10	In 3B
Output: 11	Out 4Y
Input: 12	In 4A
Input: 13	In 4B

Table A.4: Pinout For 7408

A.5 7410: TRIPLE 3-INPUT NAND GATE

This device contains three independent 3-input NAND gates. Figure A.6 is a logic diagram of one of the three circuits.

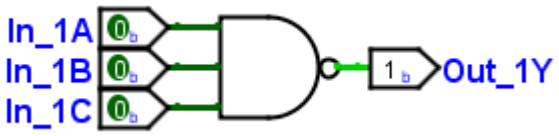

Figure A.6: 7410: Single 3-Input NAND Gate Circuit

The 7410 device in *Logisim-Evolution* uses the wiring connections indicated in Table A.5.

Logisim Label	Function
Input: 1	In 1A
Input: 2	In 1B
Input: 3	In 2A
Input: 4	In 2B
Input: 5	In 2C
Output: 6	Out 2Y
Output: 8	Out 3Y
Input: 9	In 3A
Input: 10	In 3B
Input: 11	In 3C
Output: 12	Out 1Y
Input: 13	In 1C

Table A.5: Pinout For 7410

A.6 7411: TRIPLE 3-INPUT AND GATE

This device contains three independent 3-input AND gates. Figure A.7 is a logic diagram of one of the three circuits.

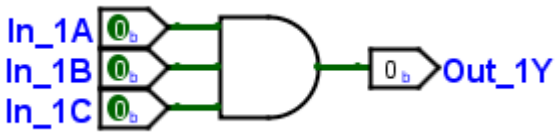

Figure A.7: 7411: Single 3-Input AND Gate Circuit

The 7411 device in *Logisim-Evolution* uses the wiring connections indicated in Table A.6.

Logisim Label	Function
Input: 1	In 1A
Input: 2	In 1B
Input: 3	In 2A
Input: 4	In 2B
Input: 5	In 2C
Output: 6	Out 2Y
Output: 8	Out 3Y
Input: 9	In 3A
Input: 10	In 3B
Input: 11	In 3C
Output: 12	Out 1Y
Input: 13	In 1C

Table A.6: Pinout For 7411

A.7 7413: DUAL 4-INPUT NAND GATE (SCHMITT-TRIGGER)

This device contains two independent 4-input NAND gates. Schmitt-triggers are a special type of device that are used to filter out spurious noise on a circuit. They are designed to change from low-to-high or high-to-low only when the input voltage reaches a preset level but not if the voltage randomly fluctuates without crossing the set-points. This device is essentially the same as the 7418. Figure A.8 is a logic diagram of one of the two circuits.

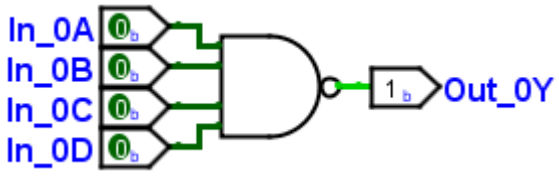

Figure A.8: 7413: Single 4-Input NAND Gate Circuit

The 7413 device in *Logisim-Evolution* uses the wiring connections indicated in Table A.7.

Logisim Label	Function
Input: 1	In A0
Input: 2	In B0
Pin 3: NC	Not Connected
Input: 4	In C0
Input: 5	In D0
Output: 6	Out Y0
Output: 8	Out Y1
Input: 9	In D1
Input: 10	In C1
Pin 11: NC	Not Connected
Input: 12	In B1
Input: 13	In A1

Table A.7: Pinout For 7413

A.8 7414: HEX INVERTER (SCHMITT-TRIGGER)

This device contains six independent inverters. Schmitt-triggers are a special type of device that are used to filter out spurious noise on a circuit. They are designed to change from low-to-high or high-to-low only when the input voltage reaches a preset level but not if the voltage randomly fluctuates without crossing the set-points. This device is essentially the same as the 7419. Figure A.9 is a logic diagram of one of the six circuits.

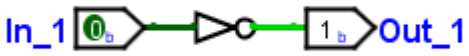

Figure A.9: 7414: Single Inverter Circuit

The 7414 device in *Logisim-Evolution* uses the wiring connections indicated in Table A.8.

Logisim Label	Function
Input: 1	In 1
Output: 2	Out 1
Input: 3	In 2
Output: 4	Out 2
Input: 5	In 3
Output: 6	Out 3
Output: 8	Out 4
Input: 9	In 4
Output: 10	Out 5
Input: 11	In 5
Output: 12	Out 6
Input: 13	In 6

Table A.8: Pinout For 7414

A.9 7418: DUAL 4-INPUT NAND GATE (SCHMITT-TRIGGER INPUTS)

This device contains two independent 4-input NAND gates. Schmitt-triggers are a special type of device that are used to filter out spurious noise on a circuit. They are designed to change from low-to-high or high-to-low only when the input voltage reaches a preset level but not if the voltage randomly fluctuates without crossing the set-points. This device is essentially the same as the 7413. Figure A.10 is a logic diagram of one of the two circuits.

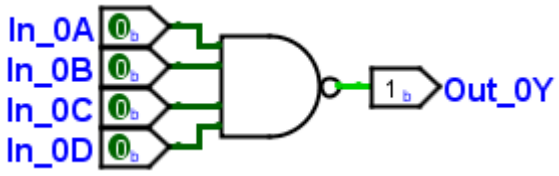

Figure A.10: 7418: Single 4-Input NAND Gate Circuit

The 7418 device in *Logisim-Evolution* uses the wiring connections indicated in Table A.9.

Logisim Label	Function
Input: 1	In A0
Input: 2	In B0
Pin 3 NC	Not Connected
Input: 4	In C0
Input: 5	In D0
Output: 6	Out Y0
Output: 8	Out Y1
Input: 9	In D1
Input: 10	In C1
Pin 11 NC	Not Connected
Input: 12	In B1
Input: 13	In A1

Table A.9: Pinout For 7418

A.10 7419: HEX INVERTER (SCHMITT-TRIGGER)

This device contains six independent inverters. Schmitt-triggers are a special type of device that are used to filter out spurious noise on a circuit. They are designed to change from low-to-high or high-to-low only when the input voltage reaches a preset level but not if the voltage randomly fluctuates without crossing the set-points. This device is essentially the same as the 7414. Figure A.11 is a logic diagram of one of the six circuits.

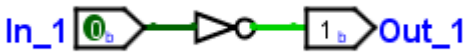

Figure A.11: 7419: Single Inverter Circuit

The 7419 device in *Logisim-Evolution* uses the wiring connections indicated in Table A.10.

Logisim Label	Function
Input: 1	In 1
Output: 2	Out 1
Input: 3	In 2
Output: 4	Out 2
Input: 5	In 3
Output: 6	Out 3
Output: 8	Out 4
Input: 9	In 4
Output: 10	Out 5
Input: 11	In 5
Output: 12	Out 6
Input: 13	In 6

Table A.10: Pinout For 7419

A.11 7420: DUAL 4-INPUT NAND GATE

This device contains two independent 4-input NAND gates. Figure A.12 is a logic diagram of one of the two circuits.

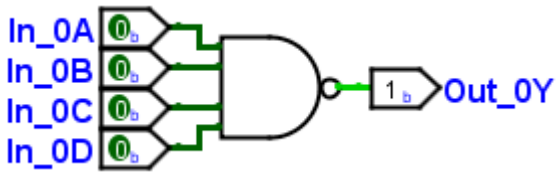

Figure A.12: 7420: Single 4-Input NAND Gate Circuit

The 7420 device in *Logisim-Evolution* uses the wiring connections indicated in Table A.11.

Logisim Label	Function
Input: 1	In A0
Input: 2	In B0
Pin 3 NC	Not Connected
Input: 4	In C0
Input: 5	In D0
Output: 6	Out Y0
Output: 8	Out Y1
Input: 9	In D1
Input: 10	In C1
Pin 11 NC	Not Connected
Input: 12	In B1
Input: 13	In A1

Table A.11: Pinout For 7420

A.12 7421: DUAL 4-INPUT AND GATE

This device contains two independent 4-input AND gates. Figure A.13 is a logic diagram of one of the two circuits.

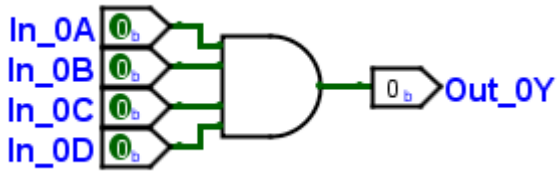

Figure A.13: 7421: Single 4-Input AND Gate Circuit

The 7421 device in *Logisim-Evolution* uses the wiring connections indicated in Table A.12.

Logisim Label	Function
Input: 1	In A0
Input: 2	In B0
Pin 3 NC	Not Connected
Input: 4	In C0
Input: 5	In D0
Output: 6	Out Y0
Output: 8	Out Y1
Input: 9	In D1
Input: 10	In C1
Pin 11 NC	Not Connected
Input: 12	In B1
Input: 13	In A1

Table A.12: Pinout For 7421

A.13 7424: QUAD 2-INPUT NAND GATE (SCHMITT-TRIGGER)

This device contains four independent 2-input NAND gates. Schmitt-triggers are a special type of device that are used to filter out spurious noise on a circuit. They are designed to change from low-to-high or high-to-low only when the input voltage reaches a preset level but not if the voltage randomly fluctuates without crossing the set-points. This device is essentially the same as the 7400. Figure A.14 is a logic diagram of one of the four circuits.

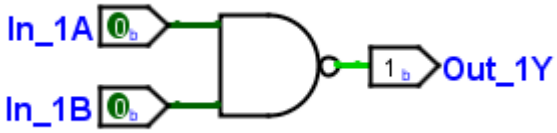

Figure A.14: 7424: Single NAND Gate Circuit

The 7424 device in *Logisim-Evolution* uses the wiring connections indicated in Table A.13.

Logisim Label	Function
Input: 1	In 1A
Input: 2	In 1B
Output: 3	Out 1Y
Input: 4	In 2A
Input: 5	In 2B
Output: 6	Out 2Y
Output: 8	Out 3Y
Input: 9	In 3A
Input: 10	In 3B
Output: 11	Out 4Y
Input: 12	In 4A
Input: 13	In 4B

Table A.13: Pinout For 7424

A.14 7427: TRIPLE 3-INPUT NOR GATE

This device contains three independent 3-input NOR gates. Figure A.15 is a logic diagram of one of the three circuits.

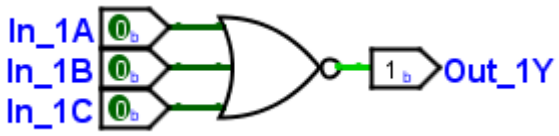

Figure A.15: 7411: Single 3-Input NOR Gate Circuit

The 7427 device in *Logisim-Evolution* uses the wiring connections indicated in Table A.14.

Logisim Label	Function
Input: 1	In 1A
Input: 2	In 1B
Input: 3	In 2A
Input: 4	In 2B
Input: 5	In 2C
Output: 6	Out 2Y
Output: 8	Out 3Y
Input: 9	In 3A
Input: 10	In 3B
Input: 11	In 3C
Output: 12	Out 1Y
Input: 13	In 1C

Table A.14: Pinout For 7427

A.15 7430: SINGLE 8-INPUT NAND GATE

This device contains a single 8-input NAND gate. The logic for this gate is $Y = \overline{A \cdot B \cdot C \cdot D \cdot E \cdot F \cdot G \cdot H}$. Figure A.16 is a logic diagram of the circuit.

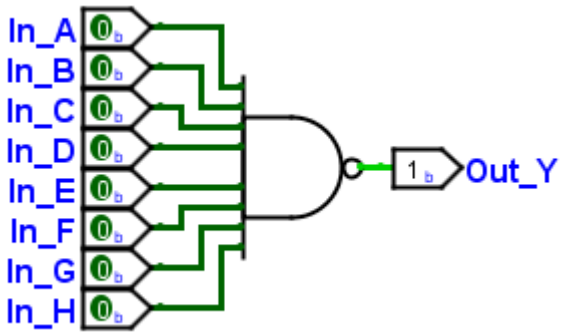

Figure A.16: 7430: Single 8-Input NAND Gate

The 7430 device in *Logisim-Evolution* uses the wiring connections indicated in Table A.15.

Logisim Label	Function
Input: 1	In A
Input: 2	In B
Input: 3	In C
Input: 4	In D
Input: 5	In E
Input: 6	In F
Output: 8	Out Y
Pin 9: NC	Not Connected
Pin 10: NC	Not Connected
Input: 11	In G
Input: 12	In H
Pin 13: NC	Not Connected

Table A.15: Pinout For 7430

A.16 7432: QUAD 2-INPUT OR GATE

This device contains four independent 2-input OR gates. Figure A.17 is a logic diagram of one of the four circuits.

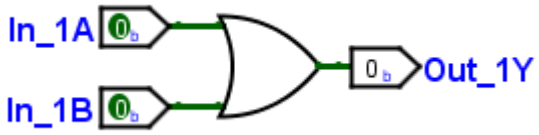

Figure A.17: 7432: Single OR Gate Circuit

The 7432 device in *Logisim-Evolution* uses the wiring connections indicated in Table A.16.

Logisim Label	Function
Input: 1	In 1A
Input: 2	In 1B
Output: 3	Out 1Y
Input: 4	In 2A
Input: 5	In 2B
Output: 6	Out 2Y
Output: 8	Out 3Y
Input: 9	In 3A
Input: 10	In 3B
Output: 11	Out 4Y
Input: 12	In 4A
Input: 13	In 4B

Table A.16: Pinout For 7432

A.17 7436: QUAD 2-INPUT NOR GATE

This device contains four independent 2-input NOR gates. This device is essentially the same as the 7402. Figure A.18 is a logic diagram of one of the four circuits.

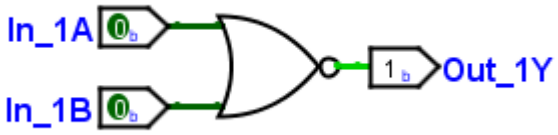

Figure A.18: 7436: Single NOR Gate Circuit

The 7436 device in *Logisim-Evolution* uses the wiring connections indicated in Table A.17.

Logisim Label	Function
Input: 1	In 1A
Input: 2	In 1B
Output: 3	Out 1Y
Input: 4	In 2A
Input: 5	In 2B
Output: 6	Out 2Y
Output: 8	Out 3Y
Input: 9	In 3A
Input: 10	In 3B
Output: 11	Out 4Y
Input: 12	In 4A
Input: 13	In 4B

Table A.17: Pinout For 7436

A.18 7442: BCD TO DECIMAL DECODER

This device takes a BDC input and deactivates a single line corresponding to the input number. It is often called a "One-Of-Ten" decoder. As an example, if 0111_{BCD} is input then line 7-of-10 will go low while all other outputs will remain high. Figure A.19 illustrates a 7442 IC in a very simple circuit.

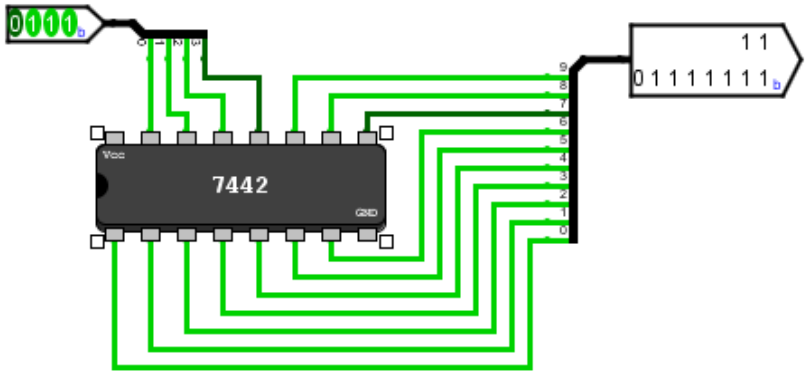

Figure A.19: 7442: BCD to Decimal Decoder

Table A.18 is the truth table for this device. Any BCD input greater than 1001 is ignored and all outputs will be high for those inputs.

Inputs				Output									
A	B	C	D	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	1	1	1	1	1	1	1	1	1
0	0	0	1	1	0	1	1	1	1	1	1	1	1
0	0	1	0	1	1	0	1	1	1	1	1	1	1
0	0	1	1	1	1	1	0	1	1	1	1	1	1
0	1	0	0	1	1	1	1	0	1	1	1	1	1
0	1	0	1	1	1	1	1	1	0	1	1	1	1
0	1	1	0	1	1	1	1	1	1	0	1	1	1
0	1	1	1	1	1	1	1	1	1	1	0	1	1
1	0	0	0	1	1	1	1	1	1	1	1	0	1
1	0	0	1	1	1	1	1	1	1	1	1	1	0

Table A.18: Truth Table For The 7442 Circuit

The 7442 device in *Logisim-Evolution* uses the wiring connections indicated in Table A.19.

Logisim Label	Function
Output 1: O0	Out 0
Output 2: O1	Out 1
Output 3: O2	Out 2
Output 4: O3	Out 3
Output 5: O4	Out 4
Output 6: O5	Out 5
Output 7: O6	Out 6
Output 8: O7	Out 7
Output 10: O8	Out 8
Output 11: O9	Out 9
Input 12: D	In D
Input 13: C	In C
Input 14: B	In B
Input 15: A	In A

Table A.19: Pinout For 7442

A.19 7443: EXCESS-3 TO DECIMAL DECODER

This device takes an Excess-3 input and deactivates a single line corresponding to the input number. It is often called a "One-Of-Ten"

decoder. As an example, if 0011_{Ex3} is input then line 0-of-10 will go low while all other outputs will remain high. This is wired in exactly the same way as the 7442 IC illustrated in Figure A.19.

Table A.20 is the truth table for this device. Any input numbers other than those found in the truth table are ignored and all outputs will be high for those inputs.

Inputs				Output									
A	B	C	D	0	1	2	3	4	5	6	7	8	9
0	0	1	1	0	1	1	1	1	1	1	1	1	1
0	1	0	0	1	0	1	1	1	1	1	1	1	1
0	1	0	1	1	1	0	1	1	1	1	1	1	1
0	1	1	0	1	1	1	0	1	1	1	1	1	1
0	1	1	1	1	1	1	1	0	1	1	1	1	1
1	0	0	0	1	1	1	1	1	0	1	1	1	1
1	0	0	1	1	1	1	1	1	1	0	1	1	1
1	0	1	0	1	1	1	1	1	1	1	0	1	1
1	0	1	1	1	1	1	1	1	1	1	1	0	1
1	1	0	0	1	1	1	1	1	1	1	1	1	0

Table A.20: Truth Table For The 7443 Circuit

The 7443 device in *Logisim-Evolution* uses the wiring connections indicated in Table A.21.

Logisim Label	Function
Output 1: O0	Out 0
Output 2: O1	Out 1
Output 3: O2	Out 2
Output 4: O3	Out 3
Output 5: O4	Out 4
Output 6: O5	Out 5
Output 7: O6	Out 6
Output 8: O7	Out 7
Output 10: O8	Out 8
Output 11: O9	Out 9
Input 12: D	In D
Input 13: C	In C
Input 14: B	In B
Input 15: A	In A

Table A.21: Pinout For 7443

A.20 7444: GRAY TO DECIMAL DECODER

This device takes a Gray Excess Code, which is a combination of Gray and Excess-3 Codes, input and deactivates a single line corresponding to the input number. It is often called a "One-Of-Ten" decoder. As an example, if 1100_{GrayEx3} is input then line 5-of-10 will go low while all other outputs will remain high. This is wired in exactly the same way as the 7442 IC illustrated in Figure A.19.

Table A.22 is the truth table for this device. Any input numbers other than those found in the truth table are ignored and all outputs will be high for those inputs.

Inputs				Output									
A	B	C	D	0	1	2	3	4	5	6	7	8	9
0	0	1	0	0	1	1	1	1	1	1	1	1	1
0	1	1	0	1	0	1	1	1	1	1	1	1	1
0	1	1	1	1	1	0	1	1	1	1	1	1	1
0	1	0	1	1	1	1	0	1	1	1	1	1	1
0	1	0	0	1	1	1	1	0	1	1	1	1	1
1	1	0	0	1	1	1	1	1	0	1	1	1	1
1	1	0	1	1	1	1	1	1	1	0	1	1	1
1	1	1	1	1	1	1	1	1	1	1	0	1	1
1	1	1	0	1	1	1	1	1	1	1	1	0	1
1	0	1	0	1	1	1	1	1	1	1	1	1	0

Table A.22: Truth Table For The 7444 Circuit

The 7443 device in *Logisim-Evolution* uses the wiring connections indicated in Table A.23.

Logisim Label	Function
Output 1: O0	Out 0
Output 2: O1	Out 1
Output 3: O2	Out 2
Output 4: O3	Out 3
Output 5: O4	Out 4
Output 6: O5	Out 5
Output 7: O6	Out 6
Output 8: O7	Out 7
Output 10: O8	Out 8
Output 11: O9	Out 9
Input 12: D	In D
Input 13: C	In C
Input 14: B	In B
Input 15: A	In A

Table A.23: Pinout For 7444

A.21 7447: BCD TO 7-SEGMENT DECODER

This device takes a BCD Code input and activates a combination of outputs such that a 7-segment display will correctly indicate the input number. Figure A.20 illustrates a 7447 IC in a very simple circuit.

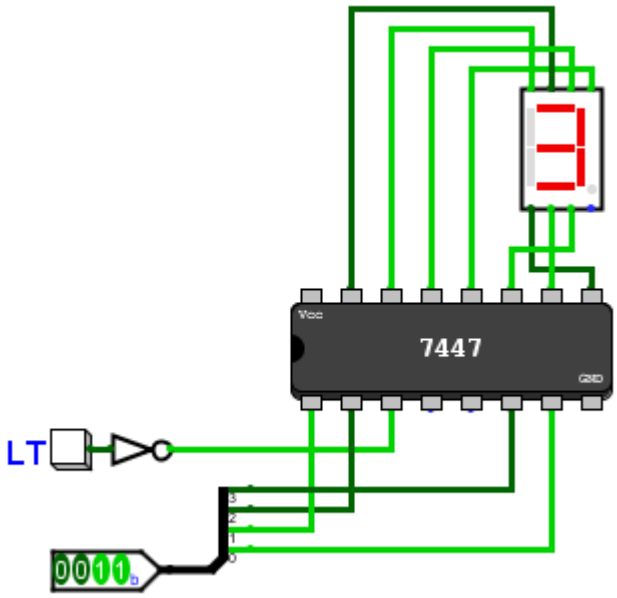

Figure A.20: 7447: BCD to 7-Segment Decoder

Table A.24 is the truth table for this device.

Inputs				Output						
A	**B**	**C**	**D**	**a**	**b**	**c**	**d**	**e**	**f**	**g**
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	0	0	1	1
1	0	1	0	1	1	1	0	1	1	1
1	0	1	1	0	0	1	1	1	1	1
1	1	0	0	1	0	0	1	1	1	0
1	1	0	1	0	1	1	1	1	0	1
1	1	1	0	1	0	0	1	1	1	1
1	1	1	1	1	0	0	0	1	1	1

Table A.24: Truth Table For The 7447 Circuit

The 7447 device in *Logisim-Evolution* uses the wiring connections indicated in Table A.25.

Logisim Label	Function
Input 1: B	B
Input 2: C	C
Input 3: LT	LT
Input 4: BI	BI
Input 5: RBI	RBI
Input 6: D	D
Input 7: A	A
Output 8: e	e
Output 10: d	d
Output 11: c	c
Output 12: b	b
Output 13: a	a
Output 14: g	g
Output 15: f	f

Table A.25: Pinout For 7447

A.22 7451: DUAL AND-OR-INVERT GATE

This device contains two independent AND-OR-INVERT gates. Figure A.21 is a logic diagram of one of the two circuits.

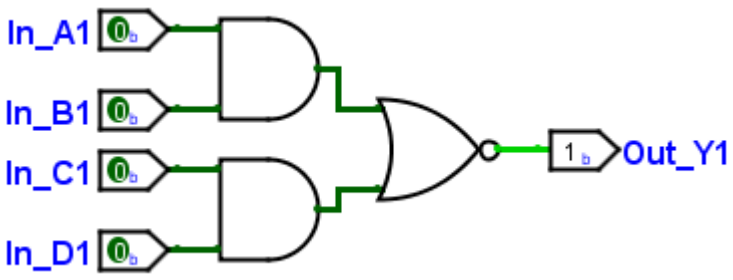

Figure A.21: 7451: Single AND-OR-INVERT Gate Circuit

The 7451 device in *Logisim-Evolution* uses the wiring connections indicated in Table A.26.

Logisim Label	Function
Input 1: A1	In A1
Input 2: A2	In A2
Input 3: B2	In B2
Input 4: C2	In C2
Input 5: D2	In D2
Output 6: Y2	Out Y2
Output 8: Y1	Out Y1
Input 9: C1	In C1
Input 10: D1	In D1
Pin 11: NC	Not Connected
Pin 12: NC	Not Connected
Input 13: B1	In B1

Table A.26: Pinout For 7451

A.23 7454: FOUR WIDE AND-OR-INVERT GATE

This device contains a single four-wide AND-OR-INVERT gate. Figure A.22 is a logic diagram of the circuit.

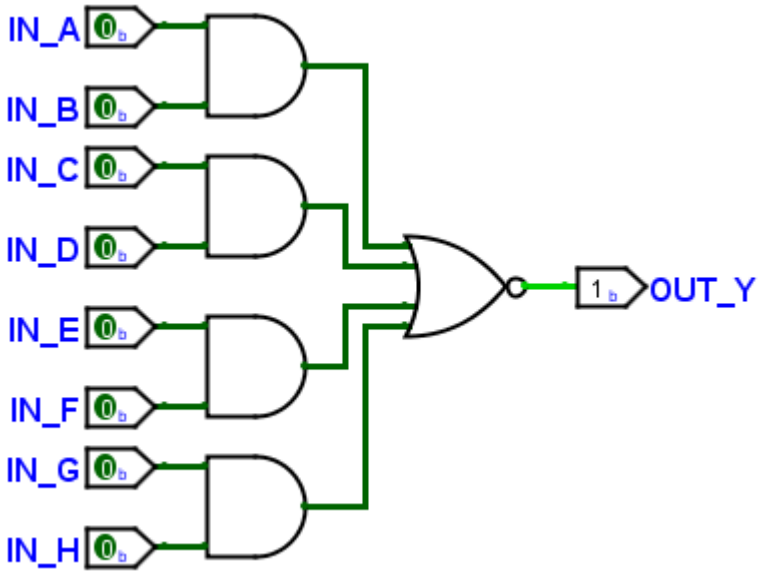

Figure A.22: 7454: Four Wide AND-OR-INVERT Gate Circuit

The 7454 device in *Logisim-Evolution* uses the wiring connections indicated in Table A.27.

Logisim Label	Function
Input 1: A	In A
Input 2: C	In C
Input 3: D	In D
Input 4: E	In E
Input 5: F	In F
Pin 6: NC	Not Connected
Output 8: Y	Out Y
Input 9: G	In G
Input 10: H	In H
Pin 11: NC	Not Connected
Pin 12: NC	Not Connected
Input 13: B	In B

Table A.27: Pinout For 7454

A.24 7458: DUAL AND-OR GATE

This device contains a two AND-OR gates. One has three-input AND gates and the other has two-input AND gates. Figure A.23 is a logic diagram of the circuit.

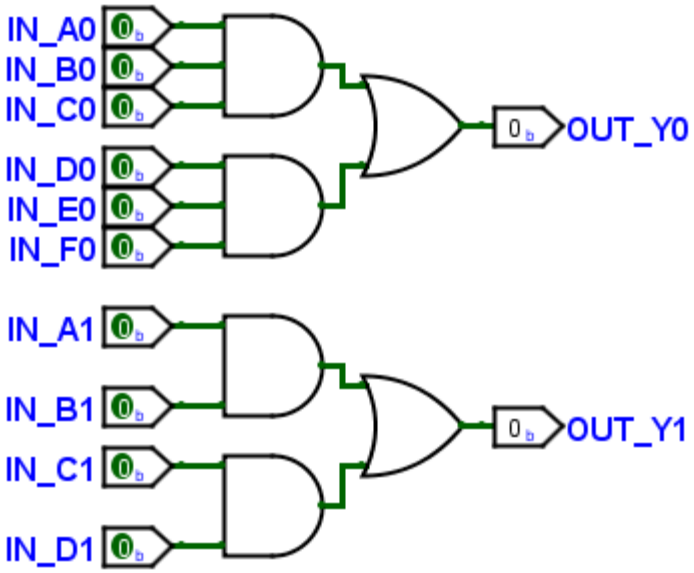

Figure A.23: 7458: Dual AND-OR Gate Circuit

The 7458 device in *Logisim-Evolution* uses the wiring connections indicated in Table A.28.

Logisim Label	Function
Input 1: A0	In A0
Input 2: A1	In A1
Input 3: B1	In B1
Input 4: C1	In C1
Input 5: D1	In D1
Output 6: Y1	Out Y1
Output 8: Y0	Out Y0
Input 9: D0	In D0
Input 10: E0	In E0
Input 11: F0	In F0
Input 12: B0	In B0
Input 13: C0	In C0

Table A.28: Pinout For 7458

A.25 7464: 4-2-3-2 AND-OR-INVERT GATE

This device contains four AND gates of different input sizes that feed a NOR gate. Figure A.24 is a logic diagram of the circuit.

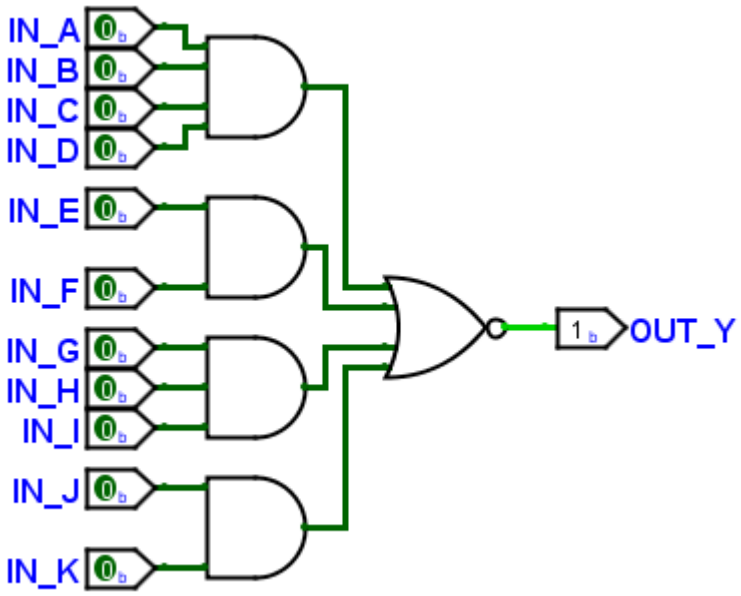

Figure A.24: 7464: 4-2-3-2 AND-OR-INVERT Gate Circuit

The 7464 device in *Logisim-Evolution* uses the wiring connections indicated in Table A.29.

Logisim Label	Function
Input 1: A	In A
Input 2: E	In E
Input 3: F	In F
Input 4: G	In G
Input 5: H	In H
Input 6: I	In I
Output 8: Y	Out Y
Input 9: J	In J
Input 10: K	In K
Input 11: B	In B
Input 12: C	In C
Input 13: D	In D

Table A.29: Pinout For 7464

A.26 7474: DUAL D-FLIPFLOPS WITH PRESET AND CLEAR

This device contains two D-Flipflops, each with its own preset and clear. The 7474 device in *Logisim-Evolution* uses the wiring connections indicated in Table A.30.

Logisim Label	Function
Input 1: nCLR1	On low, clear FF1
Input 2: D1	FF1 data input
Input 3: CLK1	FF1 clock
Input 4: nPRE1	On low, set FF1
Output 5: Q1	FF1 Q-out
Output 6: nQ1	FF1 Q-not-out
Output 8: nQ2	FF2 Q-not-out
Output 9: Q2	FF2 Q-out
Input 10: nPRE2	On low, set FF2
Input 11: CLK2	FF2 clock
Input 12: D2	FF2 data input
Input 13: nCLR2	On low, clear FF2

Table A.30: Pinout For 7474

A.27 7485: 4-BIT MAGNITUDE COMPARATOR

This device compares two 4-bit numbers and outputs one of three values: $A > B$, $A = B$, or $A < B$. It is also designed to be cascaded by including an input port for each of the three values. The 7485 device in *Logisim-Evolution* uses the wiring connections indicated in Table A.31.

Logisim Label	Function
Input 1: B3	Bit B3
Input 2: A<B	Value from prior stage
Input 3: A=B	Value from prior stage
Input 4: A>B	Value from prior stage
Output 5: A>B	High if A>B
Output 6: A=B	High if A=B
Output 7: A<B	High if A<B
Input 9: B0	Bit B0
Input 10: A0	Bit A0
Input 11: B1	Bit B1
Input 12: A1	Bit A1
Input 13: A2	Bit A2
Input 14: B2	Bit B2
Input 15: A3	Bit A3

Table A.31: Pinout For 7485

A.28 7486: QUAD 2-INPUT XOR GATE

This device contains four independent 2-input XOR gates. Figure A.25 is a logic diagram of one of the four circuits.

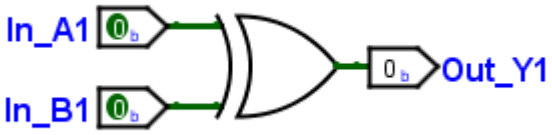

Figure A.25: 7486: Single XOR Gate Circuit

The 7486 device in *Logisim-Evolution* uses the wiring connections indicated in Table A.32.

Logisim Label	Function
Input: 1	In 1A
Input: 2	In 1B
Output: 3	Out 1Y
Input: 4	In 2A
Input: 5	In 2B
Output: 6	Out 2Y
Output: 8	Out 3Y
Input: 9	In 3A
Input: 10	In 3B
Output: 11	Out 4Y
Input: 12	In 4A
Input: 13	In 4B

Table A.32: Pinout For 7486

A.29 74125: QUAD BUS BUFFER, 3-STATE GATE

This device contains four independent buffers. When each is enabled with a low on the enable line then the input is passed to the output, when not enabled then the output floats. Figure A.26 is a logic diagram of one of the four circuits.

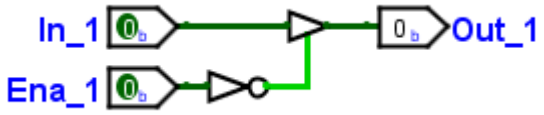

Figure A.26: 74125: Single Buffer Circuit

The 74125 device in *Logisim-Evolution* uses the wiring connections indicated in Table A.33.

Logisim Label	Function
Input: 1	nEna 1
Input: 2	In 1
Output: 3	Out 1
Input: 4	nEna 2
Input: 5	In 2
Output: 6	Out 2
Output: 8	Out 3
Input: 9	In 3
Input: 10	nEna 3
Output: 11	Out 4
Input: 12	In 4
Input: 13	nEna 4

Table A.33: Pinout For 74125

A.30 74165: 8-BIT PARALLEL-TO-SERIAL SHIFT REGISTER

This device can accept data in either parallel or serial form and shift it out in serial form. The 74165 device in *Logisim-Evolution* uses the wiring connections indicated in Table A.34.

Logisim Label	Function
Input 1: Shift/Load	Load when low, shift when high
Input 2: Clock	Clock
Input 3: P4	Input bit 4
Input 4: P5	Input bit 5
Input 5: P6	Input bit 6
Input 6: P7	Input bit 7
Output 7: Q7n	Complement of serial out
Output 9: Q7	Serial out
Input 10: Serial Input	Serial data in
Input 11: P0	Input bit 0
Input 12: P1	Input bit 1
Input 13: P2	Input bit 2
Input 14: P3	Input bit 3
Input 15: Clock Inhibit	Clock inhibit

Table A.34: Pinout For 74165

A.31 74175: QUAD D-FLIPFLOPS WITH SYNC RESET

This device contains four D-Flipflops with a single clock and master reset. The 74175 device in *Logisim-Evolution* uses the wiring connections indicated in Table A.35.

Logisim Label	Function
Input 1: nCLR	On low, clear all FF
Output 2: Q1	FF1 Q-out
Output 3: nQ1	FF1 Q-not-out
Input 4: D1	FF1 data input
Input 5: D2	FF2 data input
Output 6: nQ2	FF2 Q-not-out
Output 7: Q2	FF2 Q-out
Input 9: CLK	Clock for all FF
Output 10: Q3	FF3 Q-out
Output 11: nQ3	FF3 Q-not-out
Input 12: D3	FF3 data input
Input 13: D4	FF4 data input
Output 14: nQ4	FF4 Q-not-out
Output 15: Q4	FF4 Q-out

Table A.35: Pinout For 74175

A.32 74266: QUAD 2-INPUT XNOR GATE

This device contains four independent 2-input XNOR gates. Figure A.27 is a logic diagram of one of the four circuits.

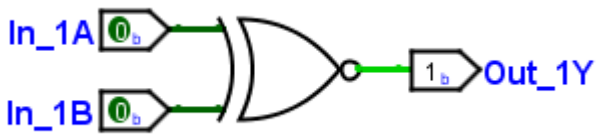

Figure A.27: 74266: Single XNOR Gate Circuit

The 74266 device in *Logisim-Evolution* uses the wiring connections indicated in Table A.36.

Logisim Label	Function
Input: 1	In 1A
Input: 2	In 1B
Output: 3	Out 1Y
Input: 4	In 2A
Input: 5	In 2B
Output: 6	Out 2Y
Output: 8	Out 3Y
Input: 9	In 3A
Input: 10	In 3B
Output: 11	Out 4Y
Input: 12	In 4A
Input: 13	In 4B

Table A.36: Pinout For 74266

A.33 74273: OCTAL D-FLIPFLOP WITH CLEAR

This device contains a single 8-bit D-Flipflop with a single clock and master clear. The 74273 device in *Logisim-Evolution* uses the wiring connections indicated in Table A.37.

Logisim Label	Function
Input 1: nCLR	On low, clear the FF
Output 2: Q1	data bit 1 output
Input 3: D1	data bit 1 input
Input 4: D2	data bit 2 input
Output 5: Q2	data bit 2 output
Output 6: Q3	data bit 3 output
Input 7: D3	data bit 3 input
Input 8: D4	data bit 4 input
Output 9: Q4	data bit 4 output
Input 11: CLK	Clock
Output 12: Q5	data bit 5 output
Input 13: D5	data bit 5 input
Input 14: D6	data bit 6 input
Output 15: Q6	data bit 6 output
Output 16: Q7	data bit 7 output
Input 17: D7	data bit 7 input
Input 18: D8	data bit 8 input
Output 19: Q8	data bit 8 output

Table A.37: Pinout For 74273

A.34 74283: 4-BIT BINARY FULL ADDER

This device contains a 4-bit adder with carry-in and carry-out bits. The 74283 device in *Logisim-Evolution* uses the wiring connections indicated in Table A.38.

Logisim Label	Function
Output 1: ∑2	Sum, bit 2
Input 2: B2	Operand B, bit 2
Input 3: A2	Operand A, bit 2
Output 4: ∑1	Sum, bit 1
Input 5: A1	Operand A, bit 1
Input 6: B1	Operand B, bit 1
Input 7: CIN	Carry in bit
Output 9: C4	Carry out bit
Output 10: ∑4	Sum, bit 4
Input 11: B4	Operand B, bit 4
Input 12: A4	Operand A, bit 4
Output 13: ∑3	Sum, bit 3
Input 14: A3	Operand A, bit 3
Input 15: B3	Operand B, bit 3

Table A.38: Pinout For 74283

A.35 74377: OCTAL D-FLIPFLOP WITH ENABLE

This device contains a single 8-bit D-Flipflop with a single clock and enable. The 74377 device in *Logisim-Evolution* uses the wiring connections indicated in Table A.39.

Logisim Label	Function
Input 1: nCLKen	On low, enable the clock
Output 2: Q1	data bit 1 output
Input 3: D1	data bit 1 input
Input 4: D2	data bit 2 input
Output 5: Q2	data bit 2 output
Output 6: Q3	data bit 3 output
Input 7: D3	data bit 3 input
Input 8: D4	data bit 4 input
Output 9: Q4	data bit 4 output
Input 11: CLK	Clock
Output 12: Q5	data bit 5 output
Input 13: D5	data bit 5 input
Input 14: D6	data bit 6 input
Output 15: Q6	data bit 6 output
Output 16: Q7	data bit 7 output
Input 17: D7	data bit 7 input
Input 18: D8	data bit 8 input
Output 19: Q8	data bit 8 output

Table A.39: Pinout For 74377

COLOPHON

This book was typeset using the typographical look-and-feel `classicthesis` developed by André Miede. The style was inspired by Robert Bringhurst's seminal book on typography "*The Elements of Typographic Style*". `classicthesis` is available for both LaTeX and LyX:

https://bitbucket.org/amiede/classicthesis/

Happy users of `classicthesis` usually send a real postcard to the author, a collection of postcards received so far is featured here:

http://postcards.miede.de/

Final Version as of September 18, 2019 (`classicthesis` Edition 4.0).

Hermann Zapf's *Palatino* and *Euler* type faces (Type 1 PostScript fonts *URW Palladio L* and *FPL*) are used. The "typewriter" text is typeset in *Bera Mono*, originally developed by Bitstream, Inc. as "Bitstream Vera". (Type 1 PostScript fonts were made available by Malte Rosenau and Ulrich Dirr.)