# Part II

# FOUNDATIONS

Foundational 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.

# 2

# BOOLEAN LOGIC

## 2.1 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.

## 2.2 PROCEDURE

### 2.2.1 *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 2.1.

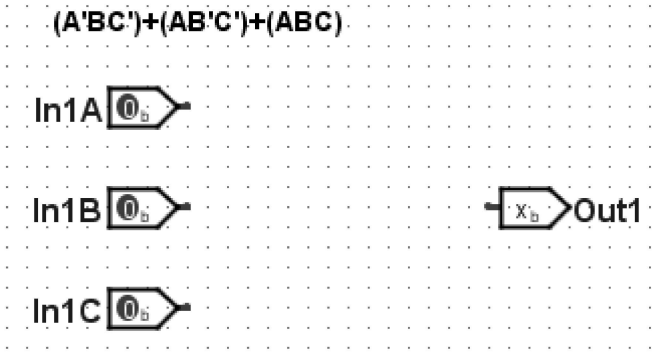



Figure 2.1: Equation 1 Inputs-Outputs

*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 2.2.

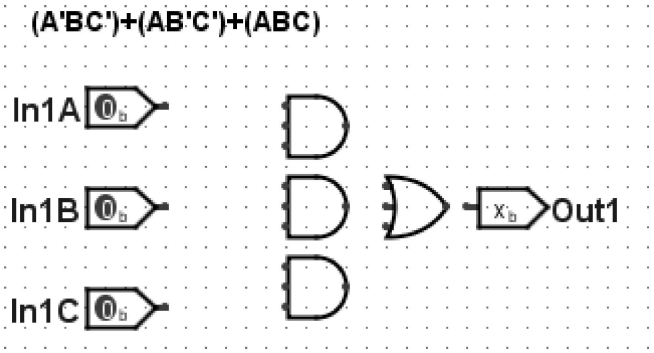



Figure 2.2: Equation 1 And-Or Gates

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 2.3.

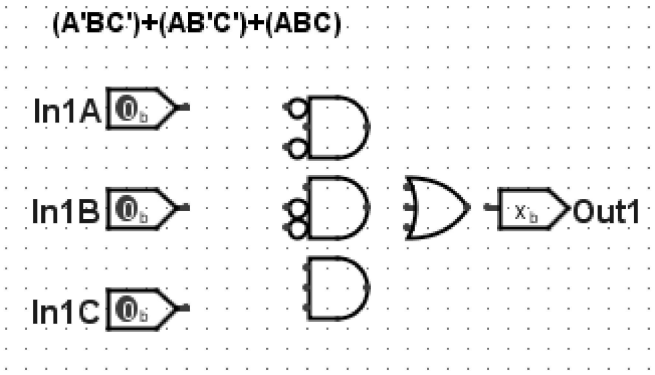



Figure 2.3: Equation 1 And Gate Inputs Set

Finally, connect all gates with wires, like Figure 2.4.

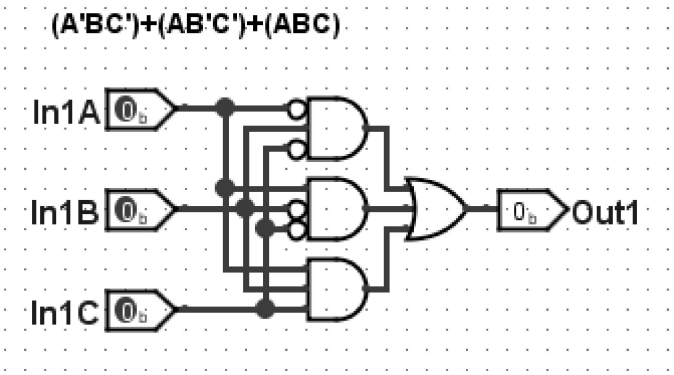



Figure 2.4: Equation 1 Circuit Completed

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 $(ABC)$.

### 2.2.2 *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.

### 2.2.3 *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 2.5. 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*.

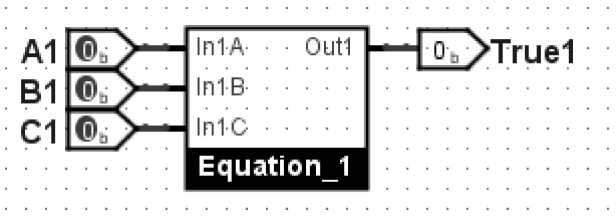



Figure 2.5: Main Circuit

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*.

### 2.2.4 *Testing the Circuit*

One way to test this circuit is to use the *poke* tool and click various input combinations for both subcircuits. If the subcircuits are correct then the output will only go high when the correct combination is set on the inputs. However, 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 `Equation_1` subcircuit.

```
# Test vector for Lab 2
# Equation 1
A1 B1 C1 True1
0   0  0     0
0   0  1     0
0   1  0     1
0   1  1     0
1   0  0     1
1   0  1     0
1   1  0     0
1   1  1     1
```

Following is an explanation for the *Test vector for Lab 2* 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 is another descriptor line and is ignored by *Logisim-Evolution* .

LINE 3 This line lists all of the inputs and outputs in the circuit under test. In this case, there are three inputs, *A1*, *B1*, and *C1*, along with one output, *True1*. *Logisim-Evolution* is able to determine whether the pin is an input or output from its properties. NOTE: each of the inputs and outputs in this circuit are single bits. If an input or output has more than one bit then that must be specified on this line. For example, if *True1* was actually a four-bit output then that pin would be listed as *True1[4]*.

LINE 4 This line contains the first test for the circuit. This line specifies that *Logisim-Evolution* make *A1*, *B1*, and *C1* equal to zero and then check to be certain that *True1* is also zero.

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

The test vector for Equation 2 would look like this:

```
1  # Test vector for Lab 2
2  # Equation 2
3  A2 B2 C2 D2 True2
4  0   0  0  0     0
5  0   0  0  1     0
6  0   0  1  0     1
7  0   0  1  1     0
8  0   1  0  0     0
9  0   1  0  1     0
10 0   1  1  0     0
11 0   1  1  1     1
12 1   0  0  0     0
13 1   0  0  1     0
14 1   0  1  0     1
15 1   0  1  1     0
16 1   1  0  0     0
17 1   1  0  1     0
18 1   1  1  0     1
19 1   1  1  1     0
```

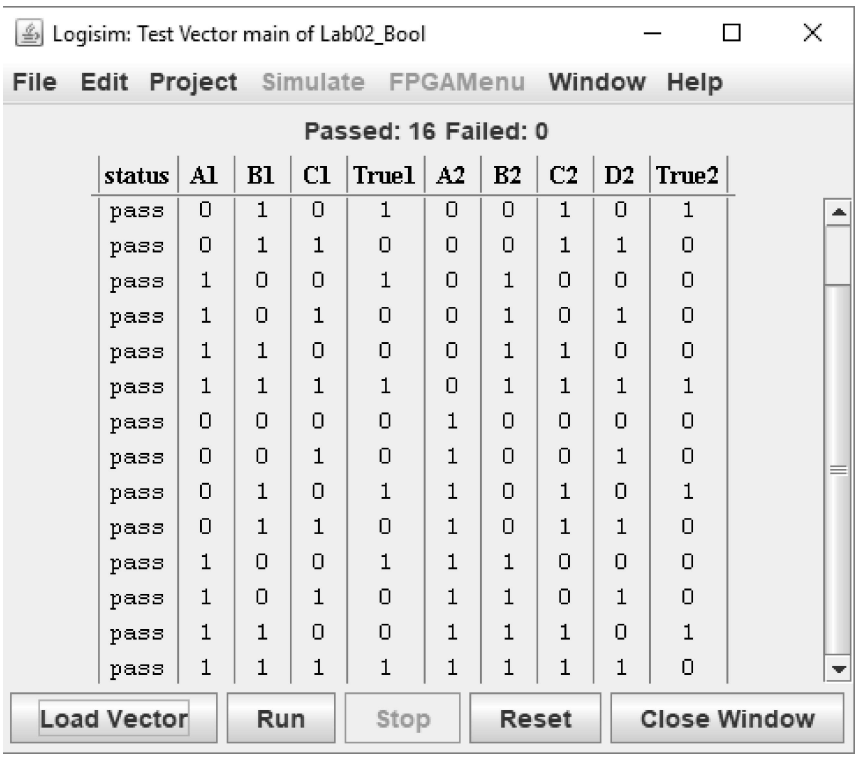
In practice, a circuit designer would usually not create two different test vectors but would, instead, create just one file to test all parts of the circuit. Combining the *Equation 1* test and the *Equation 2* test is not quite as easy as appending one after the other since all input and output pins for both circuits must be specified at the top of the file. Following is the test vector for a circuit that combines *Equation 1* and *Equation 2*. Notice that all input and output pins are defined on line three then each line beginning with line four tests both of the equation circuits. Because only eight tests are needed to fully exercise *Equation 1* but 16 are needed for Equation 2, the *Equation 1* tests are repeated starting on Line 12.

```
1  # Test vector for Lab 2
2  # Equation 1   - Equation 2
3  A1 B1 C1 True1   A2 B2 C2 D2 True2
4  0   0  0     0    0  0  0  0     0
5  0   0  1     0    0  0  0  1     0
6  0   1  0     1    0  0  1  0     1
7  0   1  1     0    0  0  1  1     0
8  1   0  0     1    0  1  0  0     0
9  1   0  1     0    0  1  0  1     0
10 1   1  0     0    0  1  1  0     0
11 1   1  1     1    0  1  1  1     1
12 0   0  0     0    1  0  0  0     0
13 0   0  1     0    1  0  0  1     0
14 0   1  0     1    1  0  1  0     1
15 0   1  1     0    1  0  1  1     0
16 1   0  0     1    1  1  0  0     0
17 1   0  1     0    1  1  0  1     0
18 1   1  0     0    1  1  1  0     1
19 1   1  1     1    1  1  1  1     0
```

To start a test, click Simulate -> Test Vector. The window illustrated in Figure 2.6 opens.


Figure 2.6: 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 2.7.




| status | A1 | B1 | C1 | True1 | A2 | B2 | C2 | D2 | True2 |
|---|---|---|---|---|---|---|---|---|---|
| pass | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| pass | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 |
| pass | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |
| pass | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| pass | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| pass | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| pass | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| pass | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| pass | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 |
| pass | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| pass | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 |
| pass | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 |
| pass | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 |
| pass | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |

Figure 2.7: Test Completed

The test indicates all 16 lines passed and zero failed so it could be reasonably concluded that the circuits are functioning properly. Figure 2.8 illustrates a failed test. The circuit designer would then need to troubleshoot to determine what went wrong with the circuit.

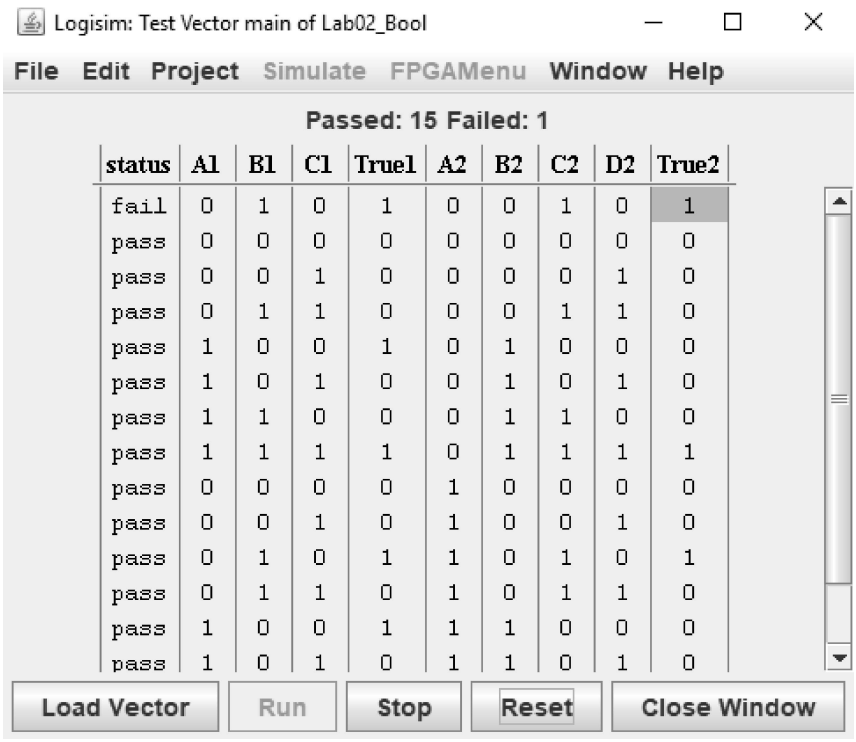



Figure 2.8: Test Failure

## 2.3 DELIVERABLE

*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.*

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:

```
George Self
Lab 02: Boolean Equations
February 18, 2018
```

Save the file with this name: *`Lab02_Bool`* and submit that file for grading.