COS 116
The Computational Universe
Laboratory 8: Digital Logic II

In this lab you'll learn that, using only AND, OR, and NOT gates, you can build a circuit that can add two numbers.

If you get stuck at any point, feel free to discuss the problem with another student or a TA. However, you are not allowed to copy another student's answers.

Hand in your lab report at the beginning of lecture on Tuesday, April 10. Include responses to questions printed in bold. (Number them by Part and Step.)

This lab will be done entirely in Logisim. You can install Logisim by clicking this link:

http://www.cs.princeton.edu/courses/archive/spring07/cos116/lab7_files/logisim.exe

When prompted, save the file to the Desktop. After the download completes, double-click logisim.exe. (If you are warned that this program may be unsafe, click "Run" and "Ok" as needed.)

Part 1: Odd-Parity Circuit

The first step in creating an adder is building a three-input odd-parity circuit. The output of an odd-parity circuit is TRUE if and only if an odd number of its inputs are TRUE. The truth table for a three-input odd-parity circuit is:

A	B	C	Output
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

To be able to build a circuit from a truth table, you must first convert the truth table to a Boolean formula using a technique called sum-of-products. Sum-of-products is best explained by example. Suppose you had the following truth table:

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

The steps of the sum-of-products technique are:

i. Identify the combinations of input values (rows of the truth table) that make the output TRUE.

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

ii. For each combination of input values, create a Boolean formula that is TRUE if and only if the inputs have those particular values. In this case, there are two combinations of input values that make the output TRUE yielding two formulas:

$$\overline{A} \bullet B \text{ and } A \bullet \overline{B}$$

iii. Take the formulas that you created in step 2 and OR them together to form a single formula representing the truth table:

$$\overline{A} \bullet B + A \bullet \overline{B}$$

In English, this formula means that the output of the function is TRUE if either A is FALSE and B is TRUE or if A is TRUE and B is FALSE.

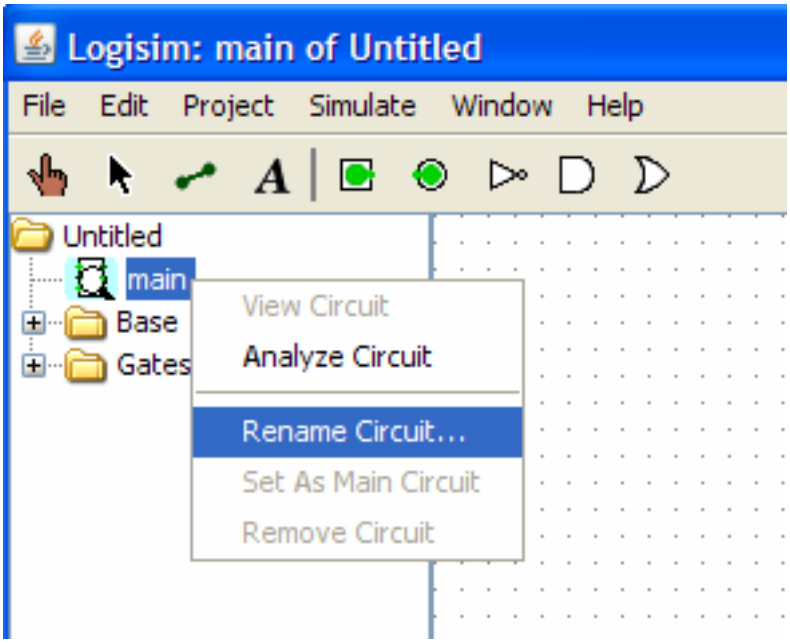
1. **Now that you know how to convert truth tables to Boolean formulas, derive a Boolean formula for the three-input odd-parity circuit.**

$$A \bullet \overline{B} \bullet \overline{C} + \overline{A} \bullet B \bullet \overline{C} + \overline{A} \bullet \overline{B} \bullet C + A \bullet B \bullet C$$

2. **Construct the three-input odd-parity circuit in Logisim. There should be three inputs and one output. For this lab, you may use AND and OR gates with more than two inputs.**

3. **Label the inputs ‘A’, ‘B’, and ‘C’ and the output ‘Out’. To label an input or output, switch to the arrow tool and then click on the input or output. A list of attributes for the object will appear in the lower left corner of the window. Edit the ‘Label’ attribute and press return when done.**

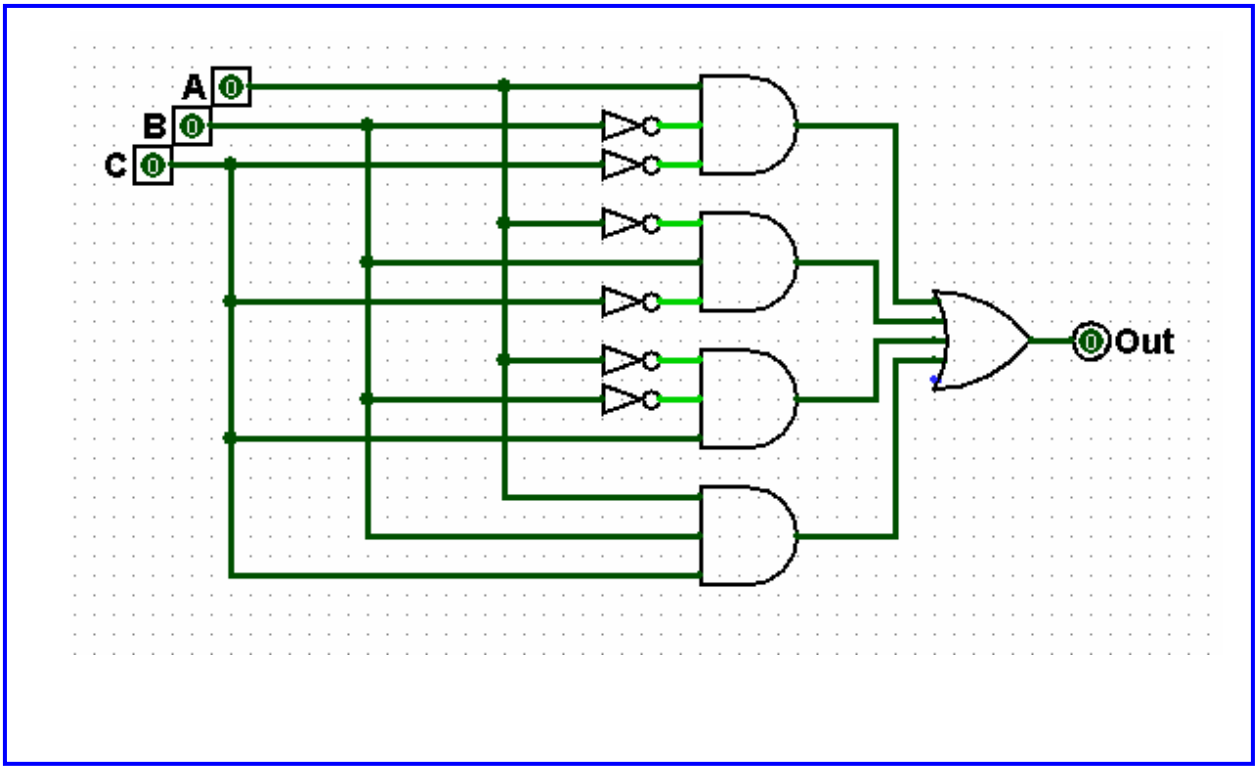
4. **Name the circuit. Since you’ll be using it later, you need to give the circuit a name. In the list on the left, right-click on the item labeled ‘main’ and select ‘Rename Circuit…’ Give the circuit the name ‘odd-parity’.**



5. **Verify that the circuit you built matches the truth table for a three-input odd-parity circuit.**

6. **Save the file by going to File > Save. Name the file ‘odd-parity.circ’.**

7. **Save a picture of your circuit to hand in with your lab report. There are a few ways to do this. Here is a simple method:**
    a. Click anywhere in the window of the Logisim program, and press Alt+PrintScreen. This takes a “snapshot” of the active window.
    b. Open a new Word document, and press Ctrl+V. This copies your snapshot into the document.



Part 2: 1-bit Adder

Computers perform addition using the same method that you learned in grade school – adding the two input numbers one column at a time starting on the right and carrying a one as needed. For example, to compute 7+6=13, a computer does:

	1	1	0	0	Carries
	0	1	1	1	Input A (7)
+	0	1	1	0	Input B (6)
	1	1	0	1	Sum (13)

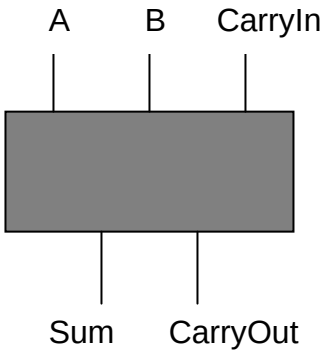
Consider one column of this addition:

	1	1	0	0	Carries
	0	1	1	1	Input A (7)
+	0	1	1	0	Input B (6)
	1	1	0	1	Sum (13)

You can view it as a circuit with three inputs

- A – one bit from the first input number
- B – one bit from the second input number
- CarryIn – a bit which is 1 if the we are carrying a one over from the previous column and two outputs
- Sum – one bit of the sum

- CarryOut – a bit which is 1 if we are carrying a one over to the next column



A circuit to compute a single column of an addition is called a 1-bit adder.

1. **Fill in the truth table for a 1-bit adder and include it in your lab report:**

A	B	CarryIn	Sum	CarryOut
0	0	0	**0**	**0**
0	0	1	**1**	**0**
0	1	0	**1**	**0**
0	1	1	**0**	**1**
1	0	0	**1**	**0**
1	0	1	**0**	**1**
1	1	0	**0**	**1**
1	1	1	**1**	**1**

It turns out that the circuits that you've built so far in this lab and the previous one – majority and odd-parity – are enough to implement a 1-bit adder. Examine the truth table for a 1-bit adder.

2. **Which of the circuits that you've already built (majority or odd-parity) can be used to compute the 'Sum' output of the adder and which can be used to compute the 'CarryOut' output?**

   For reference, the truth table of the three-input majority circuit is:

A	B	C	Output
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0

1	0	1	1
1	1	0	1
1	1	1	1

Odd-Parity can be used to compute Sum, and Majority can be used to compute Carry-Out.

3. **Now you'll build a 1-bit adder. Download our pre-built majority circuit by clicking this link:**

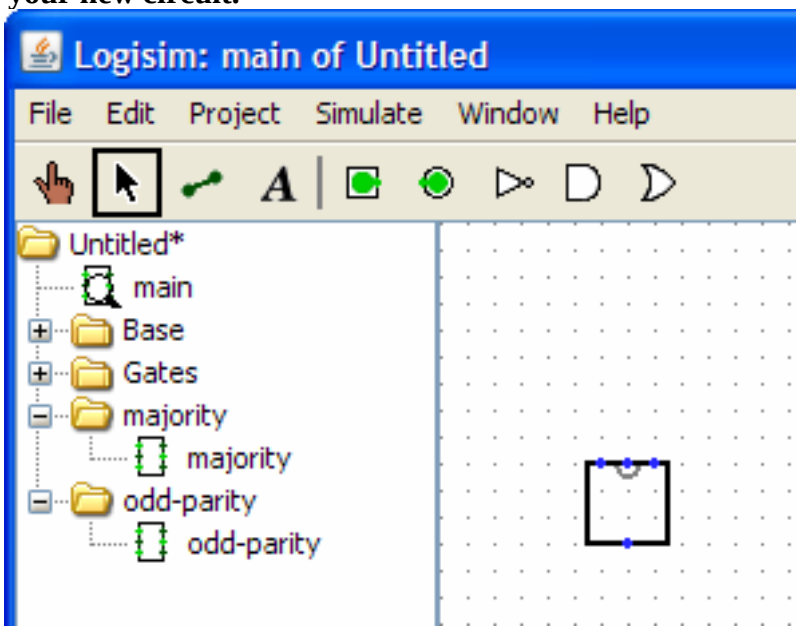
http://www.cs.princeton.edu/courses/archive/spring07/cos116/lab8_files/majority.circ

When prompted, save the file to a location that you can remember. If you are not prompted to save the file and instead see text in the Web browser window, select File > Save As… in the Web browser.

4. **Create a new circuit file in Logisim by going to File > New.**

5. **Import our majority subcircuit and your odd-parity subcircuit into the new circuit file. To import a circuit into the file, go to Project > Load Library > Logisim Library… and then navigate to file containing the subcircuit that you want to import. The files that you'll need to import should be named 'majority.circ' and 'odd-parity.circ'.**

6. **Once the two subcircuits have been imported, you should see two new folders named 'majority' and 'odd-parity' in the column on the left. Click on the '+' signs next to them to see the circuits themselves. You can click on these subcircuits to select them and then click anywhere in the dotted field to place them in your new circuit.**



Unfortunately, when you place one of the subcircuits in the dotted field, it just shows up as an unlabeled box. However, there are little blue dots on the box indicating the locations of the subcircuit's input and output connectors. The input connectors are on the side that has the gray semicircle. Also, if you hold the mouse pointer over a subcircuit, a label with the name of the subcircuit appears, and if you hold the pointer over one of the subcircuit's inputs or outputs, a label with the name of the input or output appears.

7. **Build a 1-bit adder circuit and label the inputs and outputs. There should be three inputs labeled 'A', 'B', and 'CarryIn', and two outputs labeled 'Sum' and 'CarryOut'. You should only need one majority subcircuit and one odd-parity subcircuit and no other gates.**

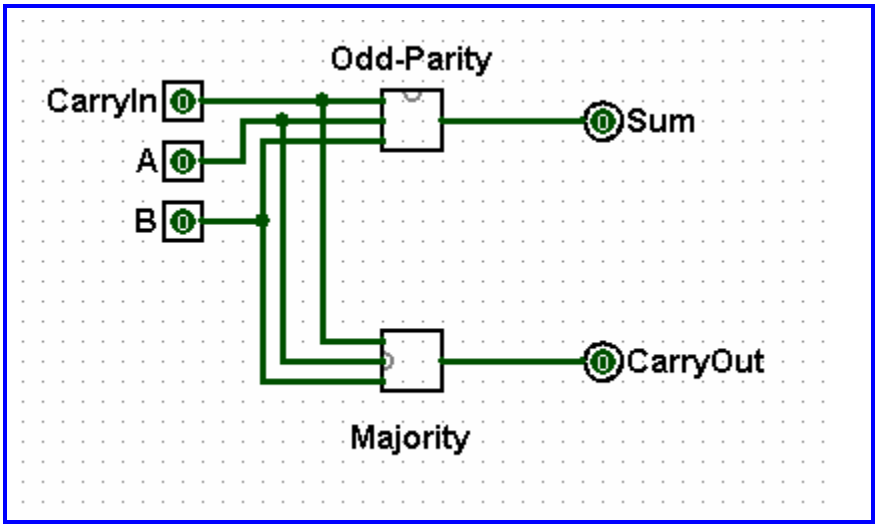
8. **Verify that the circuit you built matches the truth table that you completed in Part 2 Step 1.**

9. **Place some text next to each of the subcircuits (majority and odd-parity) that identifies the subcircuit so that we can tell which subcircuit is which. To place text, select the 'A' tool and then click on the location in the dotted field where you want the text to appear.**

10. **Name the circuit '1bit-adder' using the procedure described in Part 1 Step 4.**

11. **Save the circuit file with the name '1bit-adder.circ'.**

12. **Save a picture of your circuit to hand in with your lab report.**



Part 3: 4-bit Adder

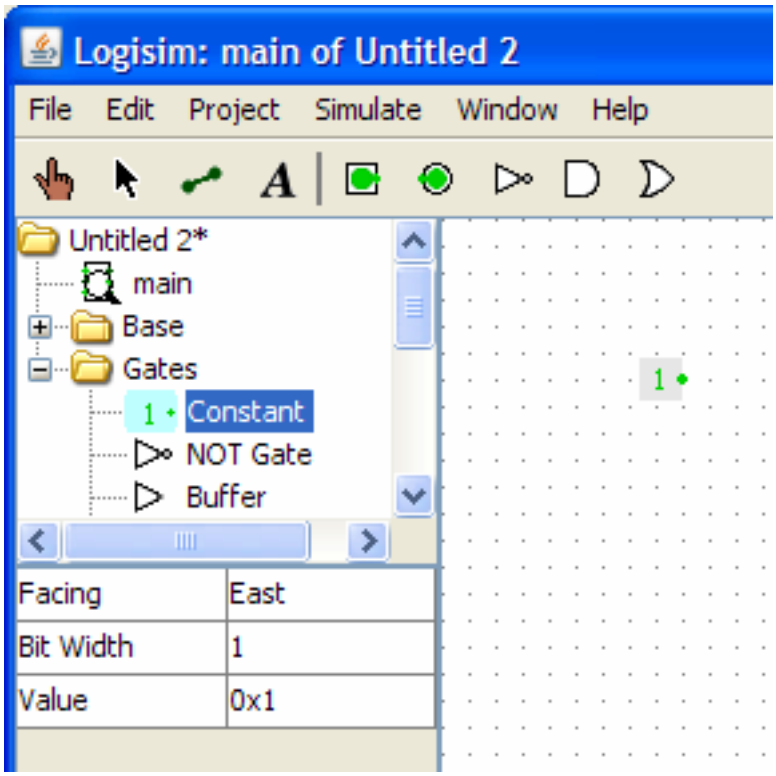
Now that you've built a circuit that can process one column of an addition – a 1-bit adder – you can now combine several 1-bit adders to make an adder that can add two 4-bit numbers.

1. **Create a new circuit file in Logisim by going to File > New.**

2. **Import your 1-bit adder subcircuit using the procedure described in Part 2 Steps 5 and 6.**

3. **Build a 4-bit adder. There should be 8 inputs (4 bits for each input number) and 4 outputs (the bits of the sum). You should only need four 1-bit adder subcircuits.**

4. **To build the 4-bit adder, you will need one other gate that will allow you to force a wire to always carry a particular value (0 or 1). To use this gate, click on the '+' next to the 'Gates' folder in the column on the left to view the list of available gates, then select 'Constant', and then click in the dotted field to place the gate. To change the value of a 'Constant' gate, switch to the pointer tool, click on the gate, and change the gate's 'Value' attribute.**



5. **Label the inputs A0-A3 and B0-B3 and the outputs S0-S3 where A0, B0, and S0 are the least significant bits of the input and output numbers.**

6. **Try your adder out and verify that it can add** 7+6 **correctly. Also, try out some other input values and verify that your circuit adds them correctly as well.**

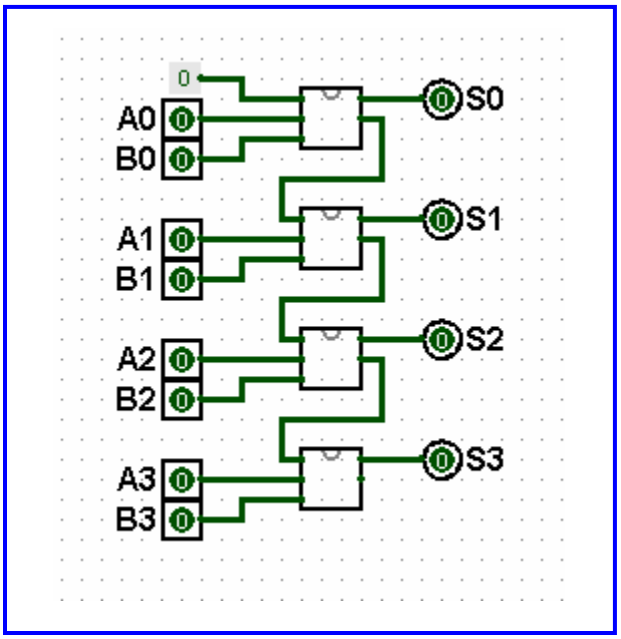
7. **Try adding** 8+8**. What is the result? Is that result correct? If it's not correct, why isn't it?**

   The result was not correct. This is because 8+8 = 16, and 16 needs 5 bits to represent, and the adder only has four output bits.

8. **Name the circuit '4bit-adder' using the procedure described in Part 1 Step 4.**

9. **Save the circuit file with the name '4bit-adder.circ'.**

10. **Save a picture of your circuit to hand in with your lab report.**



Clearly, the procedure that you used to build a 4bit adder could be used to build an adder of any size. Modern computers have adders that are 32, 64, or 128 bits wide.

Further Questions to Answer in Your Report

1. **In your 4-bit adder, the CarryOut wire of the last 1-bit adder (the one connected to A3 and B3) doesn't affect any of the bits of the sum. When is the value of that wire 1? What does a value of 1 on that wire mean?**

That wire is 1 when the sum of the A and B requires at least 5 bits to represent, and the 5th bit of the answer is 1.

2. **For any path through a circuit from a particular input to a particular output, you can measure its depth by counting the number of AND, OR, and NOT gates standing between the input and the output. The depth of an entire circuit is equal to the depth of the deepest path through that circuit. It is useful to measure the depth of a circuit because the deeper a circuit is; the longer it takes for a signal to travel through the circuit.**

   What is the depth of your 4-bit adder circuit? Justify your answer. (You may need to open up majority.circ in Logisim in order to determine its depth.)

   The deepest path through the 4-bit adder crossed all four 1-bit adders (from the carry-in of the first adder to the carry-out of the last adder). In each 1-bit adder, the deepest path was through the odd-parity circuit, which had a depth of three (the deepest path through the odd-parity circuit crossed a NOT gate, an AND gate, and an OR gate). So the depth of the 4-bit adder was 12.

3. **You saw how it's possible to derive a Boolean formula corresponding to a truth table using the sum-of-products technique. Does the sum-of-products procedure always produce the shortest possible Boolean formula corresponding to a given truth table? If you think it does, explain why. If you think it does not, give an example of a truth table for which the sum-of-products technique does not produce the shortest possible formula.**

   It does not always produce the shortest formula. Here is a counterexample:

A	B	Output
0	0	0
0	1	0
1	0	1
1	1	1

   The sum-of-products technique yields the Boolean formula $A \bullet \overline{B} + A \bullet B$**, which can be simplified to just** A**.**