CS447 Lab #7 (Week 8)

Introduction

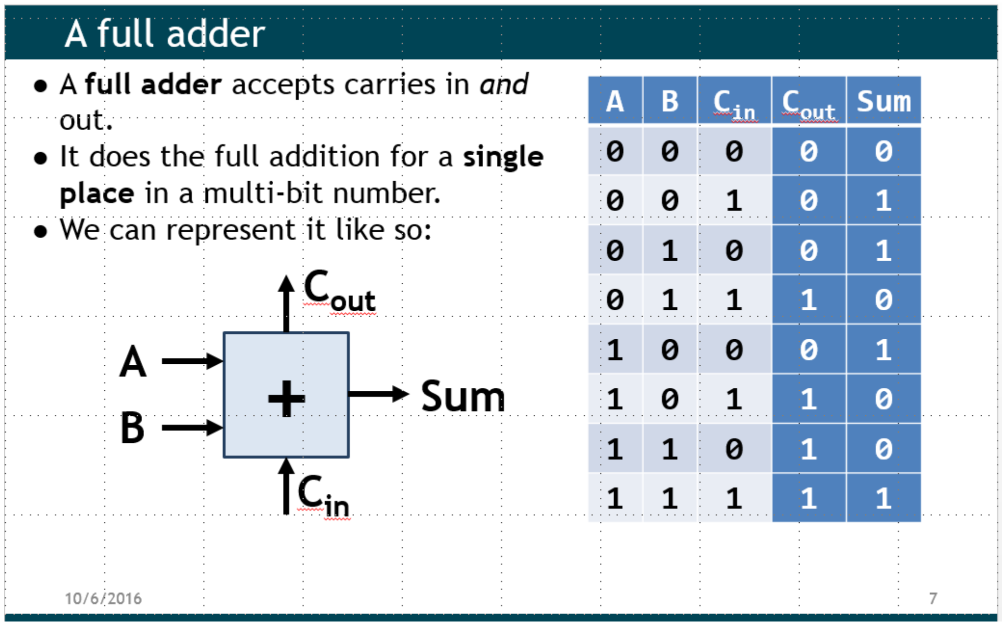
For this lab, you will use Logisim to implement the truth table relationships for the 1-bit adder. Please see Lab #6 for information on how to access Logisim.

Note: This Lab must be submitted online by 11:59 pm on Monday, October 24th. Please see the instructions on the final page.

Objective: Use logic gates to implement a 1-bit adder

Review of the 1-bit adder

The 1-bit adder was discussed in the **Addition and Subtraction** slides (see 10/6 on the course website). A relevant slide is pasted below:



A full adder

- A **full adder** accepts carries in *and* out.
- It does the full addition for a **single place** in a multi-bit number.
- We can represent it like so:

C_{out}
A
B
+
Sum
C_{in}

A	B	C_{in}	C_{out}	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

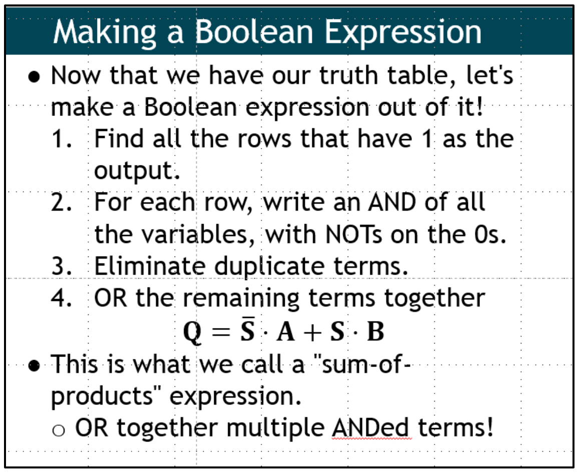
10/6/2016 7

Recall that the one-bit adder is set up so that it can be chained together with a series of other one-bit adders, so that a multi-bit addition can be performed. This implies that the one-bit adder is responsible for adding the two bits within its assigned column (**A** and **B**) to the bit carried in from the most recently computed column (**C_{in}**). The least significant bit (bit 0) of the result is output as the **Sum**, and bit 1 (set to 1 if there is a carry) is output as **C_{out}**.

The **truth table** in the slide lists all of the possible **A, B,** and **C_{in}** inputs that the adder might receive, and the **C_{out}** and **Sum** columns indicate the values of the bits that should be output for each of the inputs.

Transforming a truth table into a Boolean expression

The goal of this lab is to create a 1-bit adder that will rely on nothing more than logic gates, pins, and wires. Prior to constructing this adder in Logisim, it will be helpful to determine Boolean expressions that capture the mappings listed in the truth table. Although you are welcome to use any method you wish to find this expression, you may find it useful to refer to a simple method recently discussed in class (see the slides from 10/20, and below left; the adder table is re-pasted on the right as a reference).

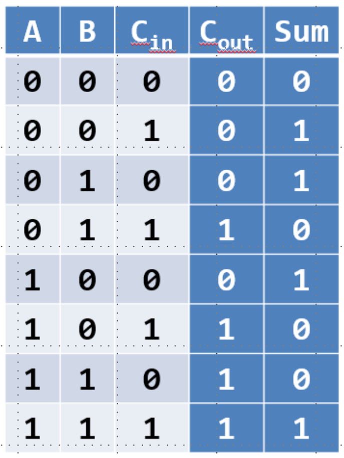


Making a Boolean Expression

- Now that we have our truth table, let's make a Boolean expression out of it!
  1. Find all the rows that have 1 as the output.
  2. For each row, write an AND of all the variables, with NOTs on the 0s.
  3. Eliminate duplicate terms.
  4. OR the remaining terms together

$$Q = \overline{S} \cdot A + S \cdot B$$

- This is what we call a "sum-of-products" expression.
  - OR together multiple ANDed terms!



A	B	C_{in}	C_{out}	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Note that the procedure described in the slide must be applied for each of the two output columns separately. In other words, we must find a Boolean expression to map **A, B,** and **C**$_{in}$ to **C**$_{out}$, and another Boolean expression to map **A, B,** and **C**$_{in}$ to **Sum**.

As an example of how Steps 1-2 work, consider the application of this procedure to the fourth row of the **A, B, C**$_{in}$ → **C**$_{out}$ mapping (**0, 1, 1** → **1**). On this row, **C**$_{out}$ **= 1**, so according to Step #1 of the procedure, we need to proceed with Step #2 for this row. For Step #2, we should create an AND of the input variables, with NOT operations applied to any input variable set to **0**:

$$\mathbf{AND(\bar{A}, B, C_{in})}$$

Since an AND gate only outputs a 1 when all of its inputs = 1, we can confirm that this expression will perform the operation specified by the fourth row for C_{out}: That is, it will output a 1 when **A = 0, B = 1**, and **C = 1,** and a 0 otherwise. Similar AND expressions will need to be calculated for every other row in which you find a 1 for **C**$_{out}$ or for **Sum**.

Once you have found all the individual AND expressions, you can then create OR expressions (again, one per output column) that will chain the ANDs together. For example, for C_{out}, you will ultimately submit four AND expressions as the input to an OR gate:

$$\mathbf{OR[AND(\bar{A}, B, C_{in}), \ldots, \ldots, \ldots]}$$

- where the dots stand for the three other AND expressions you will find. Since OR outputs a 1 if and only if ≥ 1 of its inputs = 1, such an OR expression will suffice to implement the input → output mappings for any given output column.

Once you've created the two OR expressions (or found some logically equivalent expressions using your own preferred method), you will be ready to create the Logisim circuit.

Transforming the Boolean expressions into a Logisim circuit

To construct the Logisim adder, you can proceed through the steps listed below. The basics of getting started will also be demonstrated during recitation.

1. Create 1-bit input pins for **A, B,** and **C_{in},** and 1-bit output pins for **C_{out}** and **Sum**.
2. Assemble **AND gates** corresponding to each of the AND expressions that you generated. In theory, you could create eight AND gates (corresponding to the four 1s in the **C_{out}** column and the four 1s in the **Sum** column), but you can save some work by repurposing the AND expression for the final row for both **C_{out}** and **Sum**. To set up each of the AND gates:
    a. The **AND icon** will allow you to create a new AND gate (see **AND Gate** in the **Gates** folder, or the corresponding symbol in the toolbar).
        i. In the Attribute Panel, set **Number of Inputs** to the number of inputs in your AND expression.
        ii. As with the other Logisim tools, you may optionally set other properties (e.g., **Facing, Gate Size, Label**).
    b. For input variables that do not need to be negated, connect a wire leading from the corresponding input pin to one of the input points (blue dots) on the **AND gate**.
    c. For input variables that do need to be negated, the wire will first need to go through a **NOT gate**:
        i. Create a NOT gate using the icon from either the **Gates** folder or toolbar.
        ii. Ensure that **Data Bits** is set to 1.
        iii. Hook up a wire from the **appropriate input pin** to **the input point** of the **NOT gate** (again, a blue dot).
        iv. Use a wire to connect the **output** of the **NOT gate** (red dot) to an **input** point of the **AND** gate.

3. Once the AND gates are implemented, you can set up the two **OR gates** needed to output **C_{out}** and **Sum**. For each **OR** gate:
    a. Create the OR gates using the **OR gate** icon from either the **Gates** folder or the toolbar.
    b. Set **Number of Inputs** to the number of **AND** expressions contained within your **OR** expression.
    c. Connect the **outputs** from the relevant **AND** gates (red dots) to the **input** points of the **OR** gate (blue dots).

d. The **output** of the **OR** gate should be connected to the **output pin** corresponding to the value that the gate is intended to compute.

Once your circuit is ready, you should be able to test it by using the **Poke** tool on the **A, B,** and **C_{in}** inputs, and comparing your outputs to those in the truth table on page 2. Optional: As an alternative reference source, you may want to compare your output results to those produced by the **Adder** tool provided by Logisim (see the **Arithmetic** folder).

Submission:

If you finish during recitation, please notify the TA, so that your credit for this assignment can be confirmed right away.

Regardless of whether you finish during recitation or not, you must submit your work online. Save the file as **YourPittUserName_lab7.circ** and submit the file via the appropriate link in Courseweb (see Course Documents/Week 8 Lab). The lab must be submitted by **Monday, October 24th**, by **11:59 pm**.