# **Project 1: 3-bit Adder**

## CS 200 • 20 Points Total Due Friday, February 3, 2017

#### Objectives

- Design a 3-bit adder using **only** AND, OR, and NOT gates.
- Implement your designs using the Logisim software.
- Practice Boolean Algebra, truth tables, Karnaugh Maps, and logic diagrams.

#### Overview

In this project we will design a hardware circuit to accomplish a specific task. After determining a general approach, we'll start with a truth table that reflects the desired behavior, obtain a minimized Boolean Function that describes that functionality, and finally build and simulate the circuit using the Logisim digital logic simulator.

#### Logisim

Download Logisim from here: <u>http://ozark.hendrix.edu/~burch/logisim/</u>

It's very easy to learn just by playing around with it. So play around with it! Here's some tips:

- Wires are considered to be just a single (1-bit) wire by default. You can gather a bunch of 1-bit wires into a multi-bit *bundle* by placing a "backwards" splitter, or separate a bundle into 1-bit wires again with a splitter facing the other way. Note that the "Bit Width In" on the splitter refers to the bundle size in bits regardless of whether wires are being bundled or unbundled. Some parts (like the Decoder) only work with a bundled input.
- Your inputs to a circuit can be "pins" or "constants". Input pins can have their value changed by poking at them with the Poke tool, whereas constants have to be changed via their attributes menu. Both pins and constants can be single-bit or multi-bit. Pins can be set as input or output pins, or you can just use the predefined icons on the top toolbar.
- You can add multiple circuits to your project. Double-click on each circuit to view and edit it. You can then use that circuit as an abstract block or subcircuit by viewing the circuit you wish to place the subcircuit into, single-clicking on another circuit, moving the cursor into the drawing window, and clicking to stamp the circuit block down. When you do this, each Input Pin shows up as a blue dot on the left side and each Output Pin shows up as a blue dot on the right side. Using this technique you could for example easily define a 1-bit full-adder and chain three of them together as blocks to make a 3-bit full adder,

#### Requirements

The task we want to accomplish is to create a circuit that can add two 3-bit numbers together. We'll refer to the first number as "X" and the second number as "Y". Since each is composed of 3 bits, we can also refer to  $X_2$ ,  $X_1$ , and  $X_0$  as the bits of X, with  $X_0$  being the least significant bit of X.

Rather than try and design a circuit with 64 combinations of 6 inputs and 4 outputs, we can create a 1-bit "full" adder that takes three bits of input (an X bit, a Y bit, and a Carry-in bit) and produces two bits of output (a Sum bit and a Carry-out bit). Then, chain together three of these full adders by hooking the Carry-out of an earlier stage into the Carry-in of the next stage. Each adder would get a different pair of bits from X and Y. You would need two Boolean functions describing each of the output bits for a single adder. The functions would come from a truth table of 8 rows (3 inputs =  $2^3$  rows).

Do the following:

- Draw a truth table showing all your combinations of inputs (X,Y,Cin = 8 combinations) and having columns for the desired outputs (Sum,Cout)
- Create a non-minimized Boolean function for each output column in your truth table, e.g. Cout(X,Y,Cin) = XY + XCin + ...
- Use the Karnaugh Map technique to minimize each of your functions. Clearly show both the K-Map process and your resulting, minimized functions. Use <u>only</u> AND, OR, and NOT functionality (You are not allowed to use exclusive-or, NAND, etc.).
- In Logisim, wire together the gates corresponding to each of your functions. Test your circuit to be sure it adds two bits and a carry bit correctly.
- Create three copies of the bit adder and show how each stage feeds into the next. Have the four outputs (the three sum bits and the final carry out) go to LED lights that represent the binary sum of your two numbers. Try different input values and make sure the outputs are correct.
- You may use gates with any number of inputs.

### **Project Report**

The final step of this assignment is to create a report consisting of a cover page, an overview of the project, sample output, and the source code. See Assignment Policies on either the class website or Bb Learn.

Since this is your first project, here are some hints for the sections of the report:

For the overview, **restate** the purpose (don't just copy my overview) and then the approach section can consist of the steps you took to solve the problem. Include your truth table(s) and K-map(s) along with anything else you think might be helpful for me to see how you arrived at your solution. Don't be skimpy; a good or reasonable approach can get you partial points even if your final solution didn't quite work correctly.

For the sample output, show me your circuit. You can do a screen shot but be sure to show all the components if your circuit consists of sub-circuits. And also **attach your circuit file** to your submission so I can test your circuit if necessary. Be sure to also show some testing that shows you tried it out. I don't need to see every possible combination of inputs, just some reasonable tests.

The conclusions are your own to tell me what you thought about the project. I'm always happy to hear that you learned something but you can also use it to explain why things didn't work out or even tell me how you think I could improve the project. A good conclusion, like a good approach, can net you some partial credit even if your solution didn't quite work. But, avoid generalities unless you back them up. An example:

BAD: "I really liked this project; I learned a lot."

So? You've told me nothing. What did you like? What did you learn?

BETTER: "This project was good because I finally saw how K-maps get translated to circuits."

Ok, I'm glad you learned something. But what do I do with what you told me?

GOOD: "I had a hard time figuring out the truth tables; the inputs and outputs weren't clear to me. Once a friend showed me how they worked, I figured out how to translate them to a K-map and then to a circuit. Best of all, I could make the circuit work and it was really cool to see it add numbers and get the right answer. I could keep stringing adders together to add any size numbers…"

Now, I have a clear idea of what caused problems (maybe I should talk more about how to create a truth table from scratch), and also what you figured out plus even something you figured out that was beyond the scope of the project. (By the way, don't copy my GOOD example; it's just an example and probably won't describe your experience at all.)

Hopefully, that gives you some ideas. Don't give me less than a paragraph per section (and a paragraph **is not** a single long sentence) but you don't need to write a long report – unless you have a lot to say, in which case I'm perfectly willing to read it.