CS270 Homework 3 (Due 11/17/2016 at 11:59 PM, No Late Submissions)
Word-Level Circuits in Logisim

Goal

Designing "word-level combinational circuits" in Logisim.

The Assignment

You are going to design and implement two word-level combinational circuits in this assignment. The first one is a ***12-bit decrementer*** circuit similar to the incrementer you saw in R11. The other is an ***11-bit left-normalizer.***

Note: Logisim provides a number of libraries that may implement some of what we ask, but for this assignment, you are not allowed to use any elements from the **Plexers** or **Arithmetic** libraries. Rather, you should design every sub-circuit with elements only from the **Gates** and **Wiring** libraries. You are expected to use combinational logic only (no latches, flip flops, or state machines).

Problem I: 12-bit Decrementer

This circuit takes a 12-bit 2's complement number *A* as input, and produces *A - 1*. Like the R11 incrementer, you need to do a stepwise design. There are two ways to think of the algorithm and both lead to equivalent designs.

1. *First understand and choose the algorithm that your circuit is to implement.*

   Option I: in order to subtract B from A in 2's complement, we could use the grade school algorithm: work our way from the least significant bit (LSB) to the left towards the most significant bit (MSB). At each step, if the digit/bit being subtracted is small enough, then there is no need to borrow. Otherwise, we need to borrow from the left. Even though we borrow from the left, the "intent to borrow" is a signal that goes from right to left (in the grade school algorithm, you couldn't have figured it out otherwise).

   Option II: to subtract B from A, we can negate B in 2's complement, and *add to* (rather than subtract from) A, allowing us to reuse previous adder designs.

   You may pick either option. You have to now specialize the algorithm to the case where B is a constant, rather than an arbitrary input. Work out three examples before you start.

2. *Decompose the algorithm into a set of functions for each bit and write their truth tables.*

   For both options, the input at any bit position is an input bit A and a "carry/borrow" bit C_{in}, from the right. The results at that position are an output bit S and a "carry/borrow" bit C_{out} sent to the left. So, our first step is to write a truth table that specifies these two outputs as a function of the two inputs. Since there are two inputs and two outputs, your truth table should have 4 rows and 4 columns.

3. *Now implement and test it in **Logisim**.*

    a. Start from the following skeleton file:

    https://www.cs.colostate.edu/~cs270/.Fall16/assignments/H3dec/src/decrementer.circ

    b. This assignment will be auto-graded. Don't change the name of the **main** circuit. Also, don't change the input and output sections (the only exception is that you may toggle the input pins for testing). Preliminary testing will perform some sanity tests, but it will not check that you got the right answers.

    c. We want your design to be modular. So, first, implement a sub-circuit called **Decr1Bit** that deals with a single bit position.

    d. In the **main** circuit, build your logic by making 12 copies of the **Decr1Bit** sub-circuit and connecting them appropriately. The input to your circuit should come from the splitter in the input section. The output of your circuit should go to the splitter in the output section. Don't add any additional input or output pins in the **main** circuit.

    e. Finally, note that the bit-0 sub-circuit can be simplified in both options. Design another sub-circuit called **DecrBit0** and use that instead of **Decr1Bit** for the LSB.

Problem II: 11-bit Left Normalizer.

This circuit takes an 11-bit 2's complement number *A* and shifts it to the left until the leftmost 1 in *A* becomes the MSB.

1. Start from the following skeleton file:

    https://www.cs.colostate.edu/~cs270/.Fall16/assignments/H3nrm/src/normalizer.circ

2. This assignment will be auto-graded. Don't change the name of the **main** circuit. Also, don't change the input and output sections (the only exception is that you may toggle the input pins for testing). Preliminary testing will perform some sanity tests, but it will not check that you got the right answers.

3. Design a sub-circuit called **LSH1** that takes an 11-bit input X and shifts it left by 1 bit. The least significant bit of the output is 0. The output Y needs to be 11 bits wide.

    [Hint: it does not need any gates, just splitters from the **Wiring** library]

4. Next, design and build a sub-circuit called **Mux11**. It should have two 11-bit inputs A and B and a single select bit S. Depending on the select signal, its 11-bit output X should be either A or B.

    [Hint: experiment with the splitters and the *Data Bits* property of the AND/OR gates. If you use them correctly, your multiplexer design will be very compact]

5. Next, design and build a sub-circuit called **CLSH1** (for *Conditional LSH1*). It has a single 11-bit input X and an 11-bit output Y. The output depends on the MSB of X:

    a. If MSB(X) is 1, the output is X unchanged.

    b. If MSB(X) is 0, the output is LSH1(X).

    [Hint: you will only need splitters and one copy of each of the previous sub-circuits]

6. Finally, build the normalizer in the **main** circuit. It will take the 11-bit input that comes from the splitter in the input section, and it will produce an 11-bit output that goes to the splitter in the output section. Don't add any additional input or output pins in the **main** circuit.

    The output should be the result of left-shifting the input until the leftmost 1 becomes the MSB. If the input is all zeroes (no leftmost 1), the output should be 0. You should use only the previous sub-circuits, and possibly other elements from the **Gates** and **Wiring** libraries.

When finished, submit decrementer.circ to the H3dec box in the Checkin section. Also, submit normalizer.circ to the H3nrm box in the Checkin section.