CS2110 Spring 2013

Homework 3

Objectives

1. To understand digital logic
2. To use gates to perform various operations
3. To learn how to use sub-circuits

Overview

All computer processors have a very important component known as the Arithmetic Logic Unit (ALU). This component allows the computer to do, as the name suggests, arithmetic and logical operations. For this assignment, you're going to build an ALU of your own.

DO NOT USE TRANSISTORS!

1. Create a 1-bit full adder
2. Use your 1-bit full adder to create a 4-bit full adder
3. Use your 4-bit full adder and other components to construct a 4-bit ALU
4. Use your 4-bit ALU's to make a 16-bit ALU

Requirements

You may use anything from the Base and Wiring sections, basic gates (AND, OR, XOR, NOT, NAND, NOR, XNOR), multiplexers, and decoders. Use of anything not listed above will result in heavy deductions. **Your designs for the first three problems must each be a sub-circuit**.

More information on sub-circuits is given below

Use tunnels where necessary to make your designs more readable

Sub-circuit tutorial

As you build circuits that are more and more sophisticated, you will want to build smaller circuits that you can use multiple times within larger circuits. In Logisim, this is called a sub-circuit. Sub-circuits behave like classes in Object-Oriented languages. Any changes made in the design of a sub-circuit are automatically reflected wherever it is used. The direction of the IO pins in the sub-circuit correspond to their locations on the representation of the sub-circuit.

To create a sub-circuit:

1. Go to the “Project” menu and choose “Add Circuit…”
2. Name your sub-circuit

To use a sub-circuit:

1. Click the sub-circuit you want to use from the sidebar.
2. Place it in your design.

To set a sub-circuit as the main circuit:

1. Right-click the sub-circuit and choose "Set As Main Circuit".

Part 1: 1-bit Full Adder

The full adder has three **1-bit** inputs (A, B, and CarryIn), and two **1-bit** outputs (Answer and CarryOut). The full adder adds A+B+CarryIn and places the answer in Answer and the carry-out in CarryOut.

For example:
A = 0, B = 1, CarryIn = 0 then Answer = 1, CarryOut = 0
A = 1, B = 0, CarryIn = 1 then Answer = 0, CarryOut = 1
A = 1, B = 1, CarryIn = 1 then Answer = 1, CarryOut = 1

Hint: making a truth table of the inputs will help you

Make your 1-bit adder a sub-circuit. You will use it in part 3.

Part 2: 4-bit Full Adder

For this part of the assignment, you will daisy-chain together 4 of your **1-bit** full adders together in order to make a **4-bit** full adder.

This circuit should have two **4-bit** inputs (A and B) for the numbers you're adding, and one **1-bit** input for CarryIn. Note that the CarryIn can be used for more than just CarryIn, which will come in handy in the next two parts.

There should be one **4-bit** output for the answer and one **1-bit** output for CarryOut.

Make your 4-bit full adder a sub-circuit; you will use it in Part 4.

Part 3: 4-bit ALU

Now you will use your 4-bit adder to make a 1-bit ALU capable of the following operations:

1.	Addition	[A + B]
2.	Subtraction	[A – B]
3.	Increment	[A + 1]
4.	Negation	[-A]
5.	AND	[A & B]
6.	OR	[A | B]
7.	NOT	[~A]
8.	XOR	[A ^ B]

Notice that Increment, Negate, and NOT only operate on the A input

This ALU has two **4-bit** inputs for A and B, one **1-bit** CarryIn, and three **1-bit** inputs for S0, S1, and S2 (the selectors for the op-code of your ALU's functions)

This ALU should have one **4-bit** output for the answer, and one **1-bit** CarryOut.

You may assign the op-codes to the operations any way that you want as long as you implement every operation and each op-code only corresponds to one operation.

Add a label to your circuit that lists which operation each op-code corresponds to.

Part 4: 16-bit ALU

Daisy-chain 4 of your **4-bit** full ALUs with some other circuitry to create a **16-bit** ALU with the following operations (same as before):

1.	Addition	[A + B]
2.	Subtraction	[A – B]
3.	Increment	[A + 1]
4.	Negation	[-A]
5.	AND	[A & B]
6.	OR	[A | B]
7.	NOT	[~A]
8.	XOR	[A ^ B]

Notice that Increment, Negate, and NOT only operate on the A input

This ALU has two **16-bit** inputs for A and B and three **1-bit** inputs for S0, S1, and S2 (the selectors for the op-code of your ALU's functions)

This ALU should have one **16-bit** output for the answer.

You may assign the op-codes to the operations any way that you want as long as you implement every operation and each op-code only corresponds to one operation.

Add a label to your circuit that lists which operation each op-code corresponds to.

Also, set this sub-circuit as the main circuit.

Deliverables

Save the file as hw3.circ and turn it in through T-Square

Once again, your designs for the four problems must be contained in the same .circ file as subcircuits

You may also include a README file if there is anything you wish your grading TA to know about your designs. This would be a good place to discuss your choice of op-codes or other concerns.