CS2110 Fall 2010

Homework 3

This assignment is due by:

Day: Tuesday, September 14th, 2010
Time: 11:55 PM

Objective: Understand basic operation of an arithmetic and logic unit, or ALU.

Short Instructions: Complete all problems and turn them in.

NOTICE Banned Components NOTICE

You may only use basic gates for this assignment, that is, any component in the Plexers, Arithmetic, or Memory folders you may not use for this assignment!

Problem 1

Create a 1-bit full adder. This full adder will take in three inputs A and B and a CarryIn and will have two outputs CarryOut and Answer. The full adder adds A + B + CarryIn and places the answer of this addition on the Answer wire and if there was an overflow (a Carry Out) asserts the CarryOut wire.

For example:

A = 1 B = 1 and CarryIn = 1 then the Answer will be 1 with a CarryOut of 1.

A = 0 B = 1 and CarryIn = 1 then the Answer will be 0 with a CarryOut of 1.

To get started form a truth table for the full adder and then use your truth table to implement your full adder.

Make your full adder into a subcircuit you will be using this in problem 2.

Problem 2

Bit Slice Definition:

> A technique for constructing a processor from modules, each of which processes one bit-field or "slice" of an operand. Bit slice processors usually consist of an ALU of 1, 2, 4 or 8 bits and control lines including carry or overflow signals usually internal to the CPU. For example, two 4-bit ALUs could be arranged side by side, with control lines between them, to form an 8-bit ALU.

- The Free On-line Dictionary of Computing, http://www.foldoc.org, Editor Denis Howe

Create a 1-bit bit slice ALU that will handle: Addition, Subtraction, Increment (i.e. A + 1 / A++), AND, OR, XOR, and COMPLEMENT (NOT). If you want an additional challenge, add the NAND operation to your design *by adding a single XOR gate (and additional wires, of course) to your design.* The NAND functionality is not a requirement - if you attempt it, though, you should only add this single gate to a design that would be a sensible, efficient, and functioning design to fulfill the other requirements. Another challenge if you wish is to only use one full adder to make your 1-bit ALU you will need to get the full adder to act like an adder for one case, a subtracter in another and an incrementer in another; however you should not have to modify the full adder you created in part I to make it do this all that is needed is a few additional gates.

Make this 1-bit bit slice ALU into a single component (subcircuit).

The ALU we want you to create should have inputs for: A, B, Carry-in, S0, S1, and S2. S0, S1, and S2 are your control lines- they tell the ALU, through patterns called opcodes, what operation it should perform. **Please make it clear in your circuit diagram what opcodes correspond to which operations.** The ALU should have outputs for V (the final answer) and a Carry-out. The final answer is always sent to V. For the NOT and INCREMENT operations, the B input should be completely ignored.

Design Goal:
The intention of this one bit ALU will be to daisy-chain them together to form a multi-bit ALU, so your design and positioning of input and output pins should reflect this.

HINT for "Subtract" and "Increment", you may treat "Subtract" as "Add A to complement of B" and "Increment" as "Add A to zero", but for both of those operations set the carry-in for the first bit to one. It is okay to set the carry-in bit outside of the ALU subcomponent, but you **must** do this logically rather than by hand **(i.e., there should not be a separate switch you flip to set the carry-in bit when you're using the operations that need it).** When you chain your ALUs together for the next problem, you can then just put four of the simpler units together and have some simple logic to control the carry-in bit of the first ALU, depending on what operation is taking place. (If you do figure out a trick to make a full-functioning ALU that is happy with always receiving a zero for the first carry-in, you are welcome to, but be warned that this requires a good deal of thought).

Problem 3

Using the 1-bit bit slice ALU component from Problem 1, make a 4-bit ALU.

Then create two LEDs one which will detect overflow when working with 2's complement binary and another to detect overflow when working with unsigned binary.

Some form of output from your ALU is a requirement (e.g. hex display, output pins).

Please test your 4-bit ALU and make your final circuit for this part clearly labeled so the TA can easily test it!

Turn-in Details

Save both problems under the same .circ file (Otherwise you will have to import your ALU as a library for problem 2).

You may include a README file with your submission if there is any information that your TA needs to know about your designs. (This would be a good place to discuss your choice of opcodes or other concerns.) **Also include the opcodes on your circuit diagram and clearly label them.**

We will ask that you demo your ALU to a TA during specific times. This will allow for more fair grading (so you can show us your exact implementation, etc.). More information on this and the sign-up schedule will be posted to the T-Square Wiki. We will post a T-Square announcement and announce in lab or lecture when this has occurred. ***You must show up for your demo in order to receive credit for this assignment.***

Appendix 1: SUBCIRCUIT HOWTO

Here is a brief subcircuit tutorial:

1. First go to the 'Project' menu and select 'Add Circuit'.
2. Name the circuit and press 'OK'
3. Create your subcircuit using Input pins for Input and Output pins for Output.
4. That is all

If you would like to use circuits from past assignments:

1 Go to the 'Project's menu and select 'Load Library'
2 Select 'Logisim Library'
3 Load the .circ file containing the circuit you want to use.
4 Any circuits from the imported library will be located in a folder with the same name as the library file.