CS 270 Homework 3, Part I (HW3)

Digital Circuits in Logisim (due Monday Feb 20, 5:00 pm)

Name _____ Date _____

Homework is to be done individually.
This assignment is intended to develop the skills of designing, implementing and testing digital circuits using Logisim. Part I covers combinational circuits. Part II deals with sequential circuits. The assignment includes Logisim circuits that are to be submitted using checkin (same deadline, Monday Feb 20, 5:00 pm). The written part is is due in class on Tuesday, Feb 21.

Problem 0 Please redo any problem that you missed in HW2. A correct solution will earn you back 90% of the original score. In order to get credit for this, print out a fresh copy of HW2, solve the problems and turn this in, together with your graded HW2 that was returned to you on Tue, Feb 14. Problem 0 must be submitted in class on Thursday, Feb 16.

Part I: Combinational Circuits

Problem 1 Design a 16-bit "fat inverter" in Logisim. It should take a 16-bit input, A and produce its bitwise complement (NOT). Test this circuit on at least 10 different values and save the circuit as a file, FatNOT.circ. Build a symbol for it so that it can later be used as a library/subcircuit in a more complex circuit. [5 pts]

Problem 2 Design a 16-bit "fat AND" in Logisim. It should take two 16-bit inputs, A and B produce its bitwise AND, C, i.e., $C_i = A_i \cdot B_i$, for $i = 0 \ldots 15$. Test this circuit on at least 10 different values and save the circuit as a file, FatAND.circ from Logisim. Like the previous case, build a symbol for it so that it can later be used as a subcircuit in a more complex circuit. [5 pts]

Problem 3 Design a 16-bit 2's-complement adder in Logisim. Work your way off the example in the book and as discussed in the lecture. Design your circuit in a modular fashion. First design, implement and test the standard three-input, two-output full

adder FA.circ, and save it as a subcircuit. Then using 16 instances of FA.circ, build the complete adder.circ. Make sure that your circuit is capable of adding 2's-complement integers, and make sure that your try all four combinations (about 5 each) of the signs of the two inputs. Also make sure that your circuit generates an overflow signal whenever the answer is "too large to fit." Make sure that your tests include cases where an overflow occurs on both positive and negative numbers. [10 pts]

Problem 4 Now using the previously designed circuits as modules, design test and implement the LC3 ALU. It should take a 2-bit signal (OP) and depending on its value, produce NOT(A), (A AND B) or (A + B) as its output, respectively, for OP = 00, 01 and 10. When OP is 11, the output should be all zeros. [10 pts]

Problem 5 Modify the previous circuit so that the ALU also implements subtraction: for the op-code 11, the output should now be B - A. DO NOT use the subtractor library element from Logisim. [10 pts]

Test Cases Sample test cases for this assignment are listed in Table 1. It shows (very few) sets of input and output cases for Problems 1-5.

Subcircuits The details of creating, using and editing subcircuits is given at: http://ozark.hendrix.edu/ burch/logisim/docs/2.7/en/html/guide/index.html. When we say "build a symbol," we ask you to edit appearance of your circuit. This will help you to use this edited circuit as a subcircuit for other bigger circuits. Please read the logisim user guide to do the problems 3 and 5 of this assignment. You can use the logisim MUX (from the predefined library) for problems 4 and 5 but you should not use the logisim subtractor in problem 5.

Part II: Sequential Circuits

Problem 6: Registers Use Logism to design, implement and test an 8-bit register. First, implement an RS latch as described in lecture (and Ch 3), and then extend it to a D-flip-flop. Save this as a module DFF.circ, and then use 8 of them to implement your register. It should take two additional control signals Ld (for Load, i.e., to set the contents of the register to the 8-bit input, A), and Rd (for read) which passes the contents to the output. Save this as a subcircuit Reg8.circ. [10 pts]

Problem No.	Input	Output
1	0xFFFF 0xAA11	0x0000 0x22EE
2	A=0xFFFF B=0xAAAA	C=0xAAAA
3	A=0xAAAA B=0xAAAA	C=0x5555
4	A=0xAAAA B=0xAAAA OP=00 OP=01 OP=10 OP=11	 C=0x5555 C=0xAAAA C=0x5554 C=0x0000
5	A=0xE7E7 B=0xAAAA OP=11	 C=0x3D3D

Table 1: **Some Sample Test Cases**

Problem 6: Memory Now design an 8-element memory (similar to but bigger than the one shown in Figs 3.21/3.22 of the text). Make use that you use the Reg8.circ module that you have defined in the previous problem. [10 pts]

Problem 8: Memory Extended Now modify this memory so that it can simultaneously write to one address, a1, and read from *another* address (that may or may not be the same). So there are two independent decoders, one for writing and one for reading. Save this as Memory.circ. [10 pts]

Finite State Machines

You are to design, implement and test in Logisim, a finite state machine with one input, I, and one output, O. The circuit should recognize a sequence 101101. Overlaps are allowed. For example, the input sequence (the commas are only for readability, and not part of the input)
0110,0101,1011,0100,1111 should produce
0000,0000,0010,0100,0000.

Problem 9 Draw the state diagram of the machine (hint: no more than 6 states are needed). [5 pts]

Problem 10 Choose an assignment/encoding of the states and write the truth tables for the output(s) and the next-state functions (you will need four separate tables, each with 4 inputs). From these truth tables, use Logisim to generate the circuits for each of the components. Note that logisim already has D-FFs as a predefined library element. THis is a master-slave FF, so you dont have to build one yourselves. [20 pts]

Problem 11 Hook them together and implement and test the entire finite state machine. You should use the sequence given above, as well as a number of other input sequences.. [5 pts]

Submission Instructions

You need to submit two things: a hard copy of the schematics of all your circuits, and a tar of all your .circ and README files using the checkin scripts. The README file should include a table like the one above listing a set of test cases with which you have tested each one of your circuits.