Homework #3 – Digital Logic Design

Due date: see course website

Directions:

- For short-answer questions, submit your answers in PDF format as a file called <NetID>-hw3.pdf. **Word documents will not be accepted**.
    - Please type your solutions. If hand-written material must be included, ensure it is photographed or scanned at high quality and oriented properly so it appears right-side-up on screen.
    - Please include your name on submitted work.
- For Logisim questions:
    - **Circuits will be tested using an automated system, so you must name the input/output pins exactly as described, and submit using the specified filename!**
    - **You may only use the basic gates (NOT, AND, OR, NAND, NOR, XOR), D flip-flops, multiplexers, splitters, tunnels, and clocks. Everything else you must construct from these.**
- A Logisim circuit self-tester has been provided. It works much the same as previous self-test tools; you just need to have your .circ files in the directory with the tester. The tester is known to work in the Duke Linux environment, but may possibly work elsewhere. There are a few things that need to be done for the tester to work correctly:
    - Name the files and label the pins as per the directions given. The self tester will NOT WORK with different names or labels.
    - For the FSM question, use the clock available in logisim to run the DFFs.
    - Additionally, to run the self tester you will have to place the logisim files in the same folder as the python script, the jar file and the folder labelled tests.
    - You can use the command ./hw3test.py in the following manner:

```
./hw3test.py <arguments>
The following arguments can be used with that command:
- ALL: Runs all the tests
- CLEAN: Removes all saved test outputs
- circuit1a: Runs tests for circuit1a.circ
- circuit1c: Runs tests for circuit1c.circ
- adder: Runs tests for adder.circ
- fsm: Runs tests for fsm.circ
```

    - Lastly, remember that the tests cases provided are not exhaustive so testing more cases manually would be recommended.
- **You must do all work individually, and you must submit your work electronically via Sakai**.
    - All submitted circuits will be tested for suspicious similarities to other circuits, and the test will uncover cheating, even if it is “hidden.”

Q1. Boolean Algebra

(a) [5 points] Write a truth table for the following function: Output=(A+B)·C +((A·B)+(C·B))

(b) [10] Use Logisim to implement and test the circuit from (a). Name this file circuit1a.circ. Your circuit must have the following pins:

Label	Type	Bit(s)
A	input	1
B	input	1
C	input	1
out	output	1

(c) [5 points] Write a sum-of-products Boolean function for both outputs in the following truth table and then minimize them using Boolean logic, de Morgan's laws, etc. (You should use only AND, OR, and NOT gates.) You do NOT have to have a perfectly optimal circuit, but you must show some optimizations.

A	B	C	out1	out2
0	0	0	0	1
0	0	1	0	0
0	1	0	1	1
0	1	1	1	0
1	0	0	1	0
1	0	1	1	0
1	1	0	1	0
1	1	1	0	1

(d) [10] Use Logisim to implement and test the circuit from (c). Name this file circuit1c.circ. Your circuit must have the following pins:

Label	Type	Bit(s)
A	input	1
B	input	1
C	input	1
out1	output	1
out2	output	1

Q2. Adder/Subtractor Design

[30] Use Logisim to build and test a 16-bit ripple-carry adder/subtractor. You must first create a 1-bit full adder that you then use as a module in the 16-bit adder. The unit should perform `A+B` if the `sub` input is zero, or `A−B` if the `sub` input is 1. The circuit should also output an overflow signal (`ovf`) indicating if there was a signed overflow.

Name the file adder.circ. Your circuit must have the following pins:

Label	Type	Bit(s)
A	input	16
B	input	16
sub	input	1
result	output	16
ovf	output	1

Note: To split about the 16-bit inputs and to combine the individual outputs of the one-bit adders together, use Splitters.

Q3. Finite State Machines

Design the following finite state machine (FSM). It has two 1-bit inputs (`in1` and `in2`) and one 1-bit output (`out`). The output bit should be equal to one if, on both of the last two cycles, `in1` and `in2` weren't equal to each other; otherwise, `out` should equal zero. For full credit, you must use the systematic design methodology we covered in class:

(a) [10] Draw a state transition diagram, where each state has a unique "name" that is a string of bits (e.g., states 00, 01, and 11) as well as the associated value for `out`. Label all of the arcs between transitions with the inputs that cause those transitions.

(b) [10] Draw a truth table for the state transition diagram. The inputs are `in1`, `in2`, and the current state bits. The outputs are `out` and the next state bits.

(c) [30] Use Logisim to implement and test this circuit. Name this file fsm.circ. Your circuit must have the following pins:

Label	Type	Bit(s)
in1	input	1
in2	input	1
out	output	1

Additionally, to keep this problem from becoming either trivial or troublesome, please adhere to the following restrictions:

- Do not "pre-process" the inputs in some way, such as by XORing the inputs together to get a single "are they different right now" input; that's lame.
- Implement your FSM as a "Moore" machine, meaning that the output should depend *exclusively* on the current state. In other words, your output should be written on the state nodes in the state transition diagram rather than on the edges. When writing the truth table for this, the `out` column should just be based on the current state columns.
- Run a "Clock" component to all the clock inputs in the DFFs.