COMP 273

Assignment #2

Due: Sunday February 18, 2018 at 23:30 on myCourses

Basic Von Neumann Designs

How to organize your answers for this question

- Please answer these questions on your own and submit your own unique solution to myCourses.
- Before starting to solve this assignment, make sure that you can locate the *answer-folder* in the Assignment's folder. You must modify the starter projects in that folder to answer this assignment.

***QUESTION 1**: Full 4-Bit Adder-Subtractor (12 Points)*

Instructions

- For solving this question, open the Assignment2 folder that you downloaded form MyCourses. In this folder, you can find **logisim-evolution.jar** file. For solving this question, you must use this software! Update your java and run it.
- You should design your circuit using the provided template circuit file (Four_Bits_Add_Sub.circ).This file is located in *answer-folder* and you can open it with Logisim-evolution software.

In class, we saw a circuit diagram for a 1-bit full adder. In this question you will build a simple 4-bit adder-subtractor that implements two different functions (addition and subtraction). You must build your circuit in the **Logisim-Evolution**[1] using only the basic gates provided in the built-in library, specifically, AND, OR, NOT, XOR, XNOR, and flip-flop. You may set the properties on these gates such as changing the negation of inputs, the number of inputs, or the number of data bits (e.g, you may use 3-input gates). We provided a starter project for you to help you organize your solutions. You must implement your solution in that starter project by following rules that are mentioned in the following.
You will also need to use wiring such as splitters to organize your implementation. To complete the objectives of this assignment, you must organize your solution into sub-circuits using the names and labels specified below (leave the main circuit empty).

1) You will need to also edit the appearances of some sub-circuits to better organize your solution. But, be careful not to change the sub-circuits' names, and input/output labels in the starter project file!
2) You are free to create additional sub-circuits with custom appearances as you see it and then use them in the starter sub-circuits. The sub-circuits for the main objectives, and in some cases the inputs and outputs, are already set up for you in a provided starter logisim-evolution project file called "Four_Bits_Add_Sub.circ." Make sure that you are filling the starter project and its corresponding sub-circuits with correct circuitries. Do not leave them empty and design other sub-circuits to answer this question!

(A) Warm up (2 points):

Implement a one bit full adder in the "Add_1Bit" sub-circuit that takes A, B and Cin as single-bit inputs and produces the Sum and carry-out (Cout) functions.
Note that the sub-circuit appearance was created for you in the starter code. It is also important to note that the text labels in the appearance are short form from the official input and output names defined in the circuit layout! Specifically, Sum is labeled with S for short.

(B) Build a 4-bit adder/subtractor (10 points):

To do so, you require to fill out the starter sub-circuit ("Add_Sub_4Bits") with proper circuitries as it is mentioned below:

- You already have implemented a 1-bit adder in sub-circuits "Add_1Bit". Implement a 4-bit adder-subtractor in "Add_Sub_4Bits" by using four instances of "Add_1Bit" and complementary circuitries. (6 points)
  Note that there is a control input signal, "**Add_Sub**", that whenever it is asserted to '1', this circuit ("Add_Sub_4Bits") should perform 4-bit addition; otherwise when '0' it should perform 4-bit subtraction by using 2's complement methodology. Then, your circuit should show the correct result on the output "R".
  Note that the inputs are represented in 2's complement.
  For example, if A = +1 = 0001, B = -7 = 1001, then R = -6 = 1010, Overflow = 0, Zero = 0 if we perform addition operation.
- Also, your implementation must be able to handle overflow[2] and zero[3]. (4 points)

[1] **DO NOT USE "LOGISIM" VERSION FOR SOLVING THIS ASSIGNMENT! We provided the software (Logisim-evolution) in the assignment's folder. First update your *java*, then easily run it.**

[2] When overflow happens in the operation (add/sub), "Overflow" output signal, in your circuit, should be asserted to '1'; otherwise '0'.

[3] When the result ("R") is all zeros, "0000", "Zero" output signal should be asserted to '1'; otherwise '0'.

***Mandatory;* to organize your solution, for this question, do this:**

1) Save the project file that you have completed, "Four_Bits_Add_Sub.circ", in the *answer-folder*.
   Note: DO NOT change the project file's name ("Four_Bits_Add_Sub.circ")!

QUESTION 2: 16 Nibble RAM (13 Points)

Instructions

- For solving this question, open the Assignment2 folder that you downloaded from MyCourses. In this folder, you can find *logisim-evolution.jar* file. For solving this question, you **must** use this software! Update your java and run it.
- You should design your circuit in the provided template circuit file (RAM_16_Nibbles_Read_Write.circ).This file is located in *answer-folder* and you can open it with Logisim-evolution software.

In this question you will build a simple 16 Nibbles RAM that implements two different functions (read and write). You must build your circuit in the **Logisim-Evolution**[4] using only the basic gates provided in the built-in library, specifically, AND, OR, NOT, XOR, XNOR, flip-flops (D, SR, JK, etc.). You may set the properties on these gates such as changing the negation of inputs, the number of inputs, or the number of data bits. We provided a starter project for you to help you organize your solutions. You must implement your solution in that starter project by following rules that are mentioned in the following.
You will also need to use wiring such as splitters to organize your implementation. To complete the objectives of this assignment, you must organize your solution into sub-circuits using the names and labels specified below (leave the main circuit empty).

1) You will need to also edit the appearances of some sub-circuits to better organize your solution. But, be careful not to change the sub-circuits' names, and input/output labels in the starter project file!

2) You are free to create additional sub-circuits with custom appearances as you see it and then use them in the starter sub-circuits. The sub-circuits for the main objectives, and in some cases the inputs and outputs, are already set up for you in a provided starter logisim-evolution project file called "Four_Bits_Add_Sub.circ." Make sure that you are filling the starter project and its corresponding sub-circuits with correct circuitries. Do not leave them empty and design other sub-circuits to answer this question!

3) Be careful not to use complicated built-in modules from Logisim-evolution library (such as Mux, Decoder, Registers, Counters, RAM, etc). If you want to have any of them you should implement them by yourself (preferably in some sub-circuits).

(A) Build a 4bit register (nibble) (3 points):

You should implement one nibble (4 bits) using flip-flops. The corresponding starter sub-circuit (*nibble*) is prepared for you with proper input /output ports. Fill it with proper circuitries.
Your circuit should perform read and write operations in one clock cycle. In other words, if we want to write (Read_Write = 0, and En =1), the value of Data should be saved in the flip-flops. Also, this

[4] **DO NOT USE "LOGISIM" VERSION FOR SOLVING THIS ASSIGNMENT! We provided the software (Logisim-evolution) in the assignment's folder that is uploaded on MyCourse. First update your *java*, and then easily run it. It is like LOGISIM you do not need to learn anything new to work with Logisim-evolution.**

value should be appeared in the Nibble_Out within one clock cycle. Otherwise, if En =0, no write operation should be performed. In other words, the last value of Nibble should be presented on the Nibble_out. However, if Read_Write = 1 the value of the Nibble should be observed on the Nibble_Out no matter what the value of En is.
You may add a clk source in the nibble circuit to check/test its functionality. Do not forget to remove it. Nibble should receive its clk's signal from the higher module (16-nibble RAM).

(B) Build a 16-nibble RAM (10 points):

To do so, you should use the nibble that you synthesized in the previous part to make your 16-nibble RAM. You require to fill out the starter sub-circuit ("RAM_16_Nibbles") with proper circuitries as it is mentioned below:

- Your circuit should perform read and write operations in one clock cycle. In other words, if we want to read (Read_Write =1), in the next Clk the value of the nibble that address bits indicates should be observed in the RAM_Out.
- Your RAM circuit should read a 4-bit data from input and set it as the value of the nibble that is defined by the address bits. For example, if Data = 1000 and Address = 0011, then in the next Clk cycle the value of 4^{th} nibble of the RAM should become 1000, and also this value should be displayed on the output.
- Note that each module should not have its own clock source. The circuit should have just one single Clk source that is located in RAM_16_Nibbles module. If the nibble module requires a clk signal to work properly, pass the clk signal from RAM_16_Nibbles to Nibble sub-circuit as an input for Nibble sub-circuit.

WHAT TO HAND IN FOR THIS ASSIGNMENT

Everything should be handed in electronically on myCourses. Each student is to submit his or her own unique solution to these questions on myCourses. **Do not forget that you should submit a single zip file!**

1) Zip your *answer-folder*, rename it to your student ID number. For example, 260763964.zip
2) Submit this single compressed file on myCourse in the box called Assignment#2 in MyCourses.
3) Make sure that you submit a single file (yourStudentID.zip), not many files.

HOW IT WILL BE GRADED

- This assignment is worth a total of 25 points
- The breakdown is stated on each question
- Each question is graded proportionally. In other words, if your question is 50% correct you get 50% of the marks.
- For the first two days after the due date, we deduct 15% per day. After that we will dedicate zero to the late assignments.
- Any deviation from the instruction above (submission procedure, circuit design, etc.) will cause deduction in your mark.