CASE WESTERN RESERVE UNIVERSITY
Case School of Engineering
Department of Electrical, Computer and Systems Engineering
ECSE 281. Logic Design and Computer Organization (4)

Assignment #6 **Due: February 24, 2022**

PLEASE SUBMIT YOUR WORK AS A SINGLE pdf DOCUMENT ON CANVAS.

Part 1: Problems (30 pts)

Please show your work.

1) (6 pts) Simplify the following functions using Karnaugh maps:

   $F = \sum_{w,x,y,z}(1,2,3,5,7,9,10)$

   $F = \sum_{w,x,y,z}(0,1,2,8,9,10)$

2) (6 pts) Simplify the following functions using Karnaugh maps:

   $F = \sum_{x,y,z}(1,2,3,4,5)$

   $F = \sum_{x,y,z}(0,1,4,5,6)$

3) (6 pts) Simplify the following function using Karnaugh maps:

   $F = \sum_{w,x,y,z}(5,6,7,10,11,13,14,15)$

   and draw the circuit using only three two-input and one three-input NAND gates.

4) (6 pts) Draw the circuit for the following function using only two 2-input and one 3-input NOR gates.

   The complements of the inputs are also available.

   F = a·b + a'·b' + b'·c

   (Hint: First, find a simplified product-of-sums expression.)

5) (6 pts) Design a three-input logic circuit that will produce a 1-output when there are more ones in the input combination than the zeros. For example, 001 will produce a 0-output whereas 110 will produce a 1-output. First write the truth table for this problem and then find the minimal sum using a Karnaugh map. Draw the circuit using only NAND gates. The complements of the inputs are available.

Part 2: Laboratory (20 pts)

In this assignment, we will design and test the logic of a full adder in SystemVerilog. A gate-level diagram for one possible implementation of a 1-bit full adder is shown below in Figure 1. We will also learn how to handle data that is more than 1-bit wide by using the 1-bit full adder to implement a 3-bit adder in a ripple-carry topology, like shown in Figure 2.

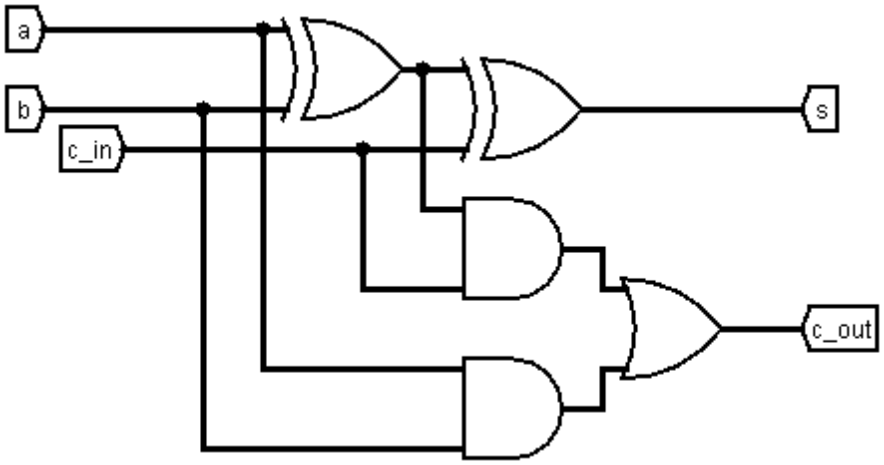

Figure 1: Gate-level implementation of an adder

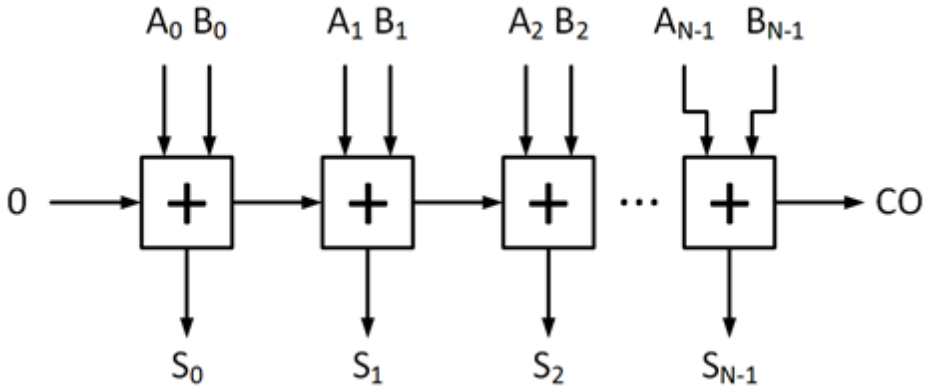

Figure 2: Ripple-carry adder topology

Step 1: Create a new Modelsim project

Create a new project and name it “abc123_eecs281_lab3”, where abc123 is your Case ID.

Step 2: Create the adder module

Create a new SystemVerilog file and name it “rc_adder_slice.sv” and in it create a module that implements the circuit in Figure 1. You may use whatever coding style you prefer, though you will probably find it simplest to implement the circuit as a series of dataflow assignments (*i.e.* assign statements).

Your module should have inputs (of the logic type) named “a”, “b”, and “c_in”, and outputs named “s” and “c_out”. You can either write two complete expressions for s and c_out in terms of the primary inputs, or you can create intermediate terms for the output of some or all of the gates.

Step 3: Create a multi-bit adder module

Create a new file called rc_adder4.sv and in it implement a 3-bit adder module. The module should have the following ports

- two 3-bit input ports (“a” and “b”) which hold the data to be added
- one 3-bit output port (“s”) which holds the sum of the two inputs
- one 1-bit output port (“co”) which holds the carry out from the most significant bit

The body of the module should include 3 instances of the rc_adder_slice module to perform the addition operation. Please note that you must instance your modules using array instantiation. **You will not receive credit for this part of the assignment if your module contains 3 separate instances of rc_adder_slice**.

Step 4: Create a Testbench for the adder

Create a second new SystemVerilog file and name it “testbench.sv”. The header of the Testbench should be the same as the one created for the previous lab, just add logic variables for all of the inputs and outputs of the adder module and instantiate and connect a copy of the adder module.

Like in the previous assignment, you should test all possible combinations of inputs for the adder. You may use whatever method you prefer to generate the input stimulus. Also create a truth table for the adder using $display and $monitor statements.

Step 5: Simulate the adder module

Once the adder and testbench modules are complete, compile the code and start a simulation with the testbench as the top-level module. Run the simulation to completion and confirm that the console output and the waveform agree with the truth table for binary addition (refer to table 2-3 on page 32 of your textbook).

Step 6: Deliverables

To turn in your lab code, submit all your code as part of your homework pdf file. You can copy and paste the code into a word processor and then add it to your homework pdf. You also need to submit the console output and the waveform. You can take a screenshot and add it to your homework file.