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

Assignment #7 **Due: March 4, 2022**

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

Laboratory (20 pts)

Generic N-bit ALU

In last week's lab, you created a module that performs the function of 3-bit addition. In practice though, it will be desirable to create adders and similar units that have a variable data width. This can be relatively easy accomplished through the use of parameters, which we will explore in this lab.

First we will extend the adder designed last week to implement an Arithmetic Logic Unit (ALU) which is capable of implementing subtraction, XOR, and XNOR operations in addition to addition. We will then parameterize this module so it works for an arbitrary input width, and then test the design.

Step 1: Designing the ALU Slice

As mentioned, the ALU will be able to perform 4 different types of operations: addition, subtraction, XOR, and XNOR. We will use a 2-bit signal, "f", to select between these inputs as shown in Table 1.

f[1]	f[0]	Output Function
0	0	S = A ^ B
0	1	S = A ~^ B
1	0	S = A + B
1	1	S = A - B

Table 1: ALU Function Select

We will start the design by first extending the adder slice module to implement this new functionality.

Create a new module called alu_slice which has the same ports as the rc_adder_slice module with the addition of a new 2-bit input called "f". Inside the module body, instantiate a copy of rc_adder_slice and then add some additional control logic to implement the function selection behavior described in Table 1
.
As a hint, you will only need to add two additional gates worth of logic to the alu_slice module. Start by working out how to implement the XOR function (recall that s = a ^ b ^ c_in in the adder slice). Once you have determined how to remove the "c_in" term from the function for "s", then attempt to implement the XNOR function (is there a way express XNOR in terms of just XOR and complements?). Once you have implemented these two functions, additional and subtraction should also work in your design.

Step 2: N-bit ALU Design

After completing the slice design, create another new module called gen_alu. This will implement a generic N-bit version of the ALU. It should have the following ports and parameters:

- one parameter ("N") which defines the width of the ALU
- two N-bit input ports ("a" and "b") which hold the data to be added
- one 2-bit input port ("f") for the function select
- one N-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

Similar to the 3-bit adder we designed last week, the body of this module should include N (the parameter) instances of the alu_slice module. Any other code you require to perform the connections between the instances should also scale with the parameter N.

Step 3: Testbench Design

Once all the gen_alu module is completed, create a testbench to ensure that your design works. This design is complex enough that it will not realistically be possible to test every possible combination of inputs. For this assignment you should test N=4 and N=8. You may test additional cases if you wish, but testing these two will earn full credit.

For N=4 test the following combinations of inputs:

F	A	B
00	0000	0000
00	1111	1111
00	1010	0101
00	0101	1010
01	0000	0000
01	1111	1111
01	1010	0101
01	0101	1010
10	0011	0011
10	0110	0110
10	1100	1100
10	1010	0101
11	0011	0011
11	0110	0110
11	1100	1100
11	1010	0101

And for N=8 test these combinations of inputs:

F	A	B
00	00000000	00000000
00	11111111	11111111
00	10101010	01010101
00	01010101	10101010
01	00000000	00000000
01	11111111	11111111
01	10101010	01010101
01	01010101	10101010
10	00000011	00000011
10	00000110	00000110
10	00001100	00001100
10	00011000	00011000
11	00110000	00110000
11	01100000	01100000
11	11000000	11000000
11	11111111	10101010

You will probably find it easiest to generate these inputs using a file to store the test vectors and reading them into an unpacked array using the $readmemb or $readmemh system tasks as discussed in the recitation slides.

Step 4: 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.