Fall 2018 / EC311
Lab 1
Due Dates:
Section B1 - October 8, 2018
Section B2 - October 5, 2018
Section B3 - October 9, 2018
Section B4 - October 4, 2018
Section B5 - October 3, 2018

1. Goals

- Introduction to Verilog modular code design.
- Introduction to structural Verilog.
- Designing simple combinational circuits.

2. Overview

In this lab, you will use structural Verilog to design a 4-bit binary adder-subtractor.

3. Objective

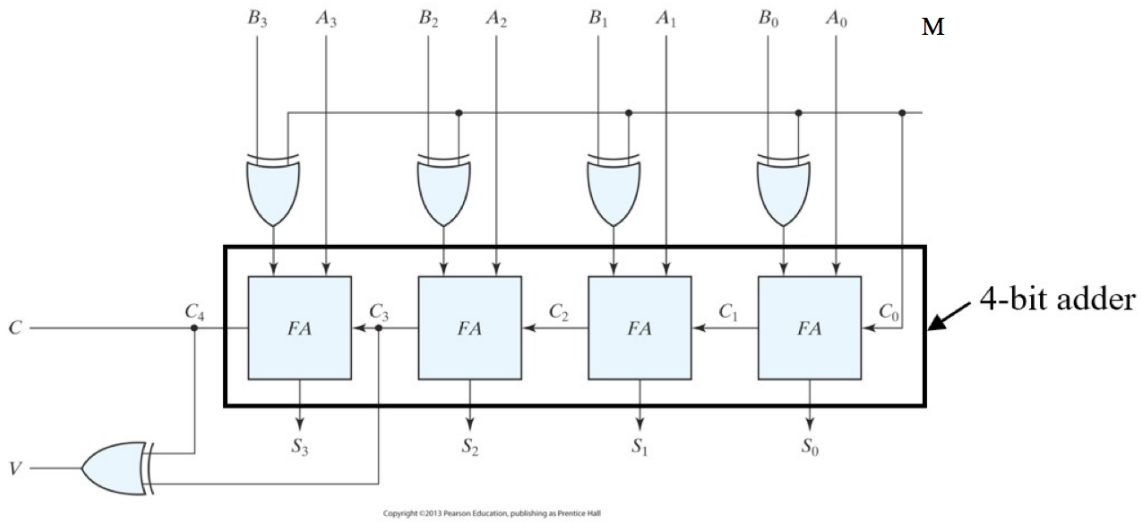
Write the HDL gate level (structural) hierarchical description for 4-bit binary adder-subtractor.

3.1. I/O Specifications

The adder-subtractor must have 4-bit inputs A and B (signed radix), and 1-bit input M as control bit. The circuit must output 4-bit S, representing the the result of the selected operation. The circuit must also output the 1-bit signal C for carry (or borrow) out and a 1-bit signal V for overflow. When M is 0, S is result of A plus B; when M is 1, S is the result of A minus B. Overflow bit V indicates whether 4-bit signed radix is enough to represent the correct answer S. V=1 means 4-bit signed radix does not have enough bits to accurately represent S. V=0 means 4 bits are enough, and that S correctly represents output of the operation.

3.2. Methodology

1. Design a 1-bit half adder circuit and write it in structural code. Simulate your design with a test bench to ensure it is correct.
2. Design a 1-bit full adder by instantiating the half adder one or more times. Simulate your design with a test bench to ensure it is correct.
3. Make a 4-bit adder by instantiating the 1-bit full adder one or more times. Simulate your design with a test bench to ensure it is correct.
4. Make a 4-bit adder-subtractor following the figure below (taken from page 142 of the Mano, Ciletti textbook). Note here vector B will need to have a controlled complementor proceeding its input to the adder bits. (xor primitive). Simulate your design with a test bench to ensure it is correct.



4. Deliverables

You need to show your design hierarchy (under Design → Implementation window), your Verilog code, and top module simulation of the half adder, full adder and 4-bit adder-subtractor to TA. You don't have to implement your code on the FPGA board for this lab.

Your design has to abide by the following requirements. Failure to follow the instructions will result in a grade reduction.

1. Use multi-bit vector/bus/array (just different ways of naming) instead of 1-bit signals for A, B, and S.

2. Follow the input/output specifications.

3. For each test bench, you need to have complete test cases. This means you need to cover all the possible combinations of input values to prove your logic is right.

4. Be hierarchical (consisting of other sub modules), use separate file for each module. Test each module. Demonstrate that the half adder and the full adder bit work as standalone units. Your 4-bit adder will fail if the submodules don't work. It's generally good engineering practice to test submodules before going to the next level. You must turn in a test bench for each module.

You must turn in a lab report describing what you did in this lab, how you did it and what happened when you did it. The lab report should be broken up into these sections: Objective, Methodology, Observation (can include screenshots timing diagrams, schematics, etc.) and Conclusion. Your report should contain enough detail such that any other engineer can replicate your experiments after reading your lab report. Please include all the required details in your lab report.

You must submit two PDFs to TurnItIn. The first PDF is your lab report. The second PDF is all of your Verilog code. You will have to copy and paste your Verilog code into an editor that can output PDF documents.

5. Testing Tips

In previous schematic labs, you have practiced serial assignment of test cases (or stimulus), such as below.

```
initial begin
        A = 0; B = 0;       // set AB = 00 at time 0
        #10 A = 0; B = 1;   // set AB = 01 at time 10
        #10 A = 1; B = 0;   // set AB = 10 at time 20
                            // (10 after previous time stamp)
        #10 A = 1; B = 1;   // set AB = 11 at time 30 (10 after 20)
end
```

There is another way to assign stimulus in parallel by using 'fork – join'. The equivalent code as previous example is below. Here all of the #(delays) are referenced to time=0.

```
initial fork
        A = 0; B = 0;       // set AB = 00 at time 0
        #10 A = 0; B = 1;   // set AB = 01 at time 10
        #20 A = 1; B = 0;   // set AB = 10 at time 20
        #30 A = 1; B = 1;   // set AB = 11 at time 30
join
```

Given that in our adder-subtractor design there are 9 bits of input in total, the total number of test cases will be $2^9 = 512$. It is unreasonable to manually assign all these stimulus. You can follow the code example below and then formulate your own test bench for the 4-bit adder-subtractor design.

```
initial begin
        A = 0; B = 0; // assign initial values
end
always begin
        #10    {A,B} = {A,B} + 1'b1;
end
```

Here, {} means concatenation. You are adding the 1 bit (1') binary number (b) '1'to {A,B} every 10 ns. When all bits become 1's, the value will overflow and start over at all 0's again. This will continue forever, unless you specify an ending time using $finish or $stop as seen below.

```
initial begin
        A = 0; B = 0;
        #10 A = 0; B = 1;
        #10 A = 1; B = 0;
        #10 A = 1; B = 1;
        #10 $stop;
end
```

You can follow the directions in Lab A and the Schematic Capture Tutorial to generate waveforms for observe the outputs of your simulations. After you generate the waveforms, you can use the simulation tool to display the multi-bit binary signals as signed decimal numbers. This makes your verification easier. To change the number system a signal is displayed with:

1. Select the signals and right click on them.

2. Select Radix $\rightarrow$ Signed Decimal

6. Due date

Lab1 is due on the following dates:
Section B1 - October 8, 2018
Section B2 - October 5, 2018
Section B3 - October 9, 2018
Section B4 - October 4, 2018
Section B5 - October 3, 2018