SANTA CLARA UNIVERSITY	ELEN /COEN 21L Fall 2021	atemeh Tehranipoor, Yu Zheng, Dat Tien Tran, Maria Kyrarini, Donald Chan

Laboratory #5: 4-bit Ripple-Carry Adder

For lab sections Monday-Friday October 25-29, 2021

I. OBJECTIVES

- Implement a hierarchical Verilog design of a 4-bit ripple carry adder
- Learn how to use busses for multi-bit inputs such as numbers and ASCII characters
- Use 7-segment displays to show inputs and outputs of the adder circuits,
- Develop and demonstrate strategic testing methods

PROBLEM STATEMENT

A logic circuit is needed to add multi-bit binary numbers.

A 2-level circuit that would add two four-bit numbers would have 9 inputs and five outputs. Although a 2-level SOP or POS circuit theoretically would be very fast, it has numerous drawbacks that make it impractical. The design would be very complex in terms of the number of logic gates. The number of inputs for each gate would challenge target technologies. Testing would be difficult. In addition, this approach can not be extended easily to add binary numbers with a higher number of bits.

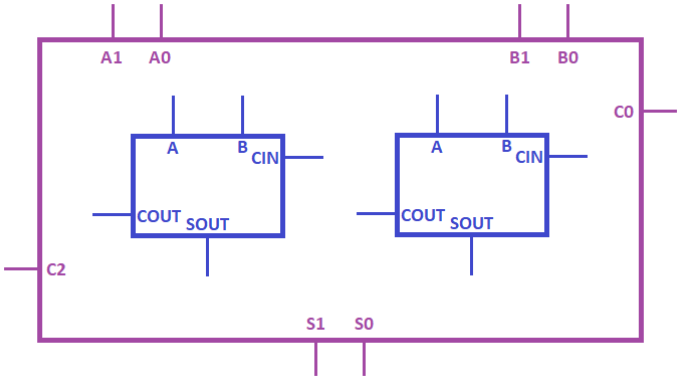
Hierarchical design methods can be used instead to make simpler multilevel implementations with far fewer logic gates and wired connections. Simple building blocks or components are designed and tested, and then these blocks are used in the design of a larger circuit much in the same way as simple logic AND and OR gates are used. The cost of this simplification is a longer delay between an input level change and a valid output level.

A full adder adds two one-bit numbers and also has a carry-in bit. The three-bit addition of the two inputs and the carry-in produces a sum output bit and a carry output bit. The three inputs are A and B and CIN, and the outputs SOUT and COUT

To add multi-bit input values, full adders can be connected in sequence with the carry-out of one full adder connected to the carry-in of the next full adder. This is called a ripple carry adder. It is similar to adding decimal numbers on paper, a column at a time, and adding the carry out from each column to the next column to the left. An *n*-bit adder would use *n* full adders to create an *n*-bit sum and a carry out.

II. PRE-LAB

1. Write the algebraic logic expressions for the two outputs, SOUT and COUT, of a full adder in terms of the three inputs A, B, and CIN.
2. Draw a logic gate schematic of a full adder.
   a. Clearly show the three inputs A, B, and CIN and the two outputs SOUT and COUT.
   b. Show all internal connections. Clearly label all inputs and outputs.
3. Draw the schematic of a two-bit ripple carry adder designed with full adders using the figure below.



   a. This is a hierarchical design, so there two instances of the symbol for the full adder. The inputs of the 2-bit adder are A1, A0, B1, B0, and C0. The outputs are S1, S0, and C2. Show all connections between the two-bit adder inputs and outputs and the full adder inputs and outputs. Show all internal connections between the two instances of the full adder components.

   b. How many logic gates are on the path from the inputs A1, A0, B1, B0, and C0 to the output C2?

4. Using continuous assignment, write a Verilog module for a full adder. The module name should be *myfulladd*, and the inputs and outputs should be the same as in PreLab step 1 above.
5. Write a Verilog module for a two-bit ripple carry adder named *myadder2*. Use two instances of your *myfulladd* module from PreLab step 4 above. Note that each instance should have a unique instance name.
6. Plan a test procedure for your 2-bit adder. Assume that you have already fully tested the single full adder component. The 2-bit adder has 5 inputs, and exhaustive test would require 32 different sets of inputs. Instead of exhaustive testing, think about what could go wrong in connecting the full adders together and strategically design your test plan to verify these connections with fewer tests. For example:
   - If all inputs are set to 0, what should the outputs be? If any output is not correct, test it further by changing inputs that affect it.
   - For each input, if it is set to 1 and all other inputs are 0, what should the outputs be?
   - How would you set the inputs to test each individual internal carry connection? What output would you look at in this test?

   Write out the steps for your test plan.

Turn in the logic expressions, the schematic circuit diagrams, and Verilog source code for your prelab and include your test plan and answers to questions.

III. LABORATORY PROCEDURE

1. Create and test the Full-Adder:
    a. Create the Verilog full adder *myfulladd*.
    b. Simulate your full adder and verify that its outputs are correct for all possible input combinations.

2. Create a 4-bit Ripple Carry Adder:
    a. Use your fully tested full adder module to build a four-bit ripple carry adder, *myadder4*.
    b. The inputs of *myadder4* are C0 and the 4-bit numbers X and Y. The outputs are V, the arithmetic overflow, C4, the carry out of the most significant bit, and a 4-bit number, S.
    c. This adder will have 9 inputs and 6 outputs. Using vectored signals instead is more efficient and much easier to read and check. So define the inputs as X[3:0] and Y[3:0], and the sum output as S[3:0]. Note the following:
        i. The [3:0] specifies the order of the four bits and how they are indexed. For example, for X[3:0], the symbol X represents all four bits. X[3] is the individual bit of X that is the most significant bit and X[0] is the individual bit that is the least significant.
        ii. You do not need to modify your full adder building block to use the vectored inputs and outputs. You only need to specify the inputs and outputs of each full adder in terms of the individual elements of the vectors X, Y, and S.
        iii. Remember that in this Verilog style, you are describing the hardware and the order of the four instances of *myfulladd* does not matter. You are specifying the connections between the inputs and outputs of each of the components by the names of the internal wires, which are implicitly defined as wires by usage.
        iv. The internal carries could be specified as individual wires, or explicitly declared as a wire vector. There are only three internal carries, and defining them as a vector C[3:1] makes the Verilog more structured and easy to understand.
    d. This circuit will be adding 4-bit numbers and generating a 4-bit result. The value of these 4-bit numbers may be interpreted as either unsigned values or signed values in 2's complement representation. In either case, we know that the correct answer can't always be expressed as a 4-bit number, so we need to be able to detect an overflow when it occurs. For unsigned interpretation, the carry-out, C4, indicates an overflow for a result that is greater than 15. For signed 2's complement interpretation, write an assign statement to create the additional output V, the arithmetic overflow output, which will be 1 if addition of two positive numbers produces a negative result or addition of two negative numbers produces a positive result.
    e. Plan a test procedure for your four-bit adder by extending your PreLab test plan. Assuming that you have already fully tested a single full adder component, write out a test plan for how you think you should test the 4-bit adder. Note that the 4-bit adder

has 9 inputs, and exhaustive test would require 512 different sets of inputs. That is not going to be practical, so think about what could go wrong in connecting the full adders together and strategically design your test plan to verify these connections.

3. Assigning Inputs and Outputs:

   (i) Create a test circuit schematic for your four-bit ripple carry adder using its symbol and connecting its inputs and outputs as specified below.

   (ii) When assigning pins for the ripple-carry adder inputs, use switches for X[0] to X[3] and Y[0] to Y[3]. Also use a switch for C0, which is the first carry in. Connect the sum outputs, S[0] to S[3], to red LEDs. Make sure you assign pins so that you can see the MSB of S on the left and the LSB of S on the right. Connect the V output and the C4 output to green LEDs.

   (iii) You will also connect the inputs and outputs to the 7-segment displays as you did in Lab3. The Altera DE2-115 board has eight 7-Segment Displays as described in the tutorial. Choose one for X, one for Y, and one for S.

   (iv) Add three instances of your seven-segment display controller to your schematic for the display of X, Y, and S. Connect each 4-bit number to the 4-bit input of one of the display controllers. Connect each of the 7-bit outputs to a display as in Lab 3.

4. Programming the FPGA:

   Download your design and test it operationally following your test plan. Observe the results of specific additions on the 7-segment display. Fill in the table below with the results that you observe and do additional testing as needed to verify correct operation. When you are sure that it operates correctly, demonstrate it to your lab assistant.

Inputs				**Outputs Observed**			
Hexadecimal 4-bit				**Sum 4-bit**		**Overflow Bits**	
X	Y	C0		S		V	C4
0	0	0					
0	0	1					
1	0	0					
0	1	0					
2	0	0					
4	3	0					
4	4	0					
9	6	0					
9	7	0					
C	A	0					
E	A	0					
E	A	1					

Take a picture of the working FPGA showing the input switches and the 7-segment display.

5. Testing Challenge:

Your testing up to this point may have been uneventful if everything was correctly connected in your first implementation. This section is meant to ensure that you gain some experience and insight into both developing good tests and debugging when you get unexpected results.

You and your partner will debug your circuit by operational testing after you lab assistant has modified it so that there is an error. Note that this exercise is not about how quickly you get through it, or even that you're successful. It's about thinking through the process of testing a design in a manner that is both efficient and effective so that you can confidently design more complex digital systems.

While the lab partners are away from the bench, the lab assistant will make a change in the four-bit adder design that will cause some errors in operation. (No changes will be made to the full adder module.) These changes will be saved and the monitor will not be showing the design. When you return to your bench, compile and download the modified design to the FPGA.

Your job is to test the circuit implemented with a design or wiring error and, based on your testing observations, see if you can figure out what specifically is wrong with it. When you come back to the bench, start following your test plan as you did before. Take notes on what inputs you test, what you observe, and how your observation is used to determine the next test. At some point in your testing, you should come across a result that is incorrect. (If you don't, then see the next paragraph.) When you do see an incorrect output, follow these steps:

a. Think about the result you're observing compared to the result you were expecting.
b. Think about how the circuit is supposed to be implemented.
c. Try to come up with some hypothesis for what might be wrong. If you can't, that's OK. Just note this result and move on to the next test case in your test plan.
d. If you did come up with a hypothesis, think about what test might make sense to run next in order to prove or disprove your hypothesis. You should look at what tests you have already run, because those results may be relevant to your thinking. Note that it's OK for this next test to not be a test in your pre-defined test plan.
e. Do this next test and note the result. If you think you have enough information to identify what's wrong with the circuit, tell the lab assistant what you believe is wrong. If you feel like you need more info, repeat these steps as many times as necessary.

If you did not come across any incorrect results, your pre-defined test plan was not effective enough. Take this time to think about what additional tests you should add to your test plan and then run those tests. Try to think systematically rather than taking shots in the dark. Once you come across an incorrect result, then follow the steps in the previous paragraph to try to identify what is wrong with the circuit.

IV. REPORT

For your lab report, include the schematics, Verilog, and simulation waveforms of all the components you designed. In addition, include answers to the following questions.

- Find one pair of X and Y inputs (with C0=0) that would result in the following:
  - C4 = 0 and V = 0
  - C4 = 0 and V = 1
  - C4 = 1 and V = 0
  - C4 = 1 and V = 1
- If you had to make an 8-bit adder, show how would you do it using only instances of the 4-bit module you have built in this lab? Specifically show how you would create the C8 output and the V output.
- Discussion of testing procedures and results.
  - Initial implementation: Did the initial testing of your circuit (i.e., before Section 5) go smoothly or did you encounter incorrect results? If the latter, describe how you determined what was wrong.
  - Testing Challenge: In Section 5 you tested a circuit to detect an error. Summarize what the steps you followed to identify the problem. Discuss whether you could have followed a shorter set of tests to identify the problem.
- If the circuit were completely correct except that X[1] and Y[1] were interchanged in a full adder input, would you be able to detect this based on observing outputs during testing? Why or why not?