SANTA CLARA UNIVERSITY

Fatemeh Tehranipoor, James Lewis, and Tokunbo Ogunfunmi **ELEN/COEN 21L**

Laboratory #5: 4-bit Ripple-Carry Adder

For lab sections Monday-Friday Oct. 26 – Oct. 30, 2020

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

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.

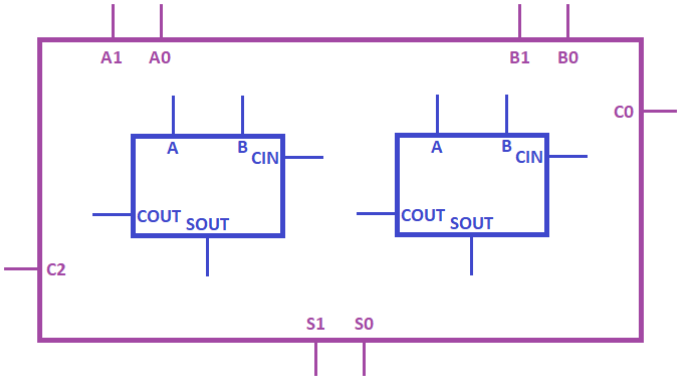
Hierarchical design methods can be used instead to make simpler multilevel implementations with far fewer logic gates and wired connections. Simple building blocks or components are designed and tested, and then these blocks are used in the design of a larger circuit much in the same way as simple logic 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. Testing your design:

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					

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.
- Did the initial testing of your circuit go smoothly or did you encounter incorrect results? If the latter, describe how you determined what was wrong.
- 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?
- Your TA will make available to you a waveform showing an adder circuit that is not behaving correctly; there is some incorrect wiring inside the adder.
  - Identify exactly where it's mis-behaving, i.e., where it showing incorrect results.
  - Make a guess (an hypothesis) about what might be wrong, which would explain the incorrect behavior.
  - Identify another test (i.e., a set of input stimulus) that might help prove or dis-prove your hypothesis.