Name_____

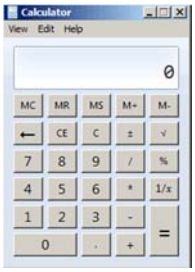
EGR 2131 Lab #6
Number Representation and Arithmetic Circuits

Equipment and Components

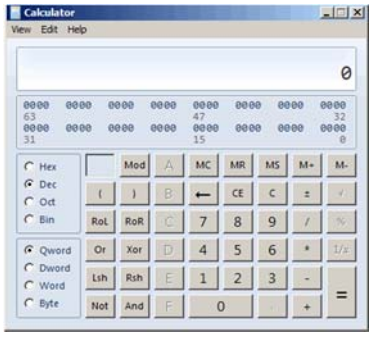
- Quartus software and Altera DE2-115 board

PART 1: Number Representation in Microsoft Calculator. First, let's use the Microsoft Calculator to do some conversions between decimal, binary, and hex. Any Windows computer should have the Calculator installed.

1. Go to the Windows Start menu, then select **All Programs > Accessories > Calculator**. This will start the Calculator program, which probably looks like this:



2. The view shown above is called the Standard view. To do conversions between different number bases, we need the Programmer view. Select **View > Programmer**. Also on the **View** menu, make sure that there is a checkmark next to **Digit Grouping**. The calculator should now look like this:



3. Notice that the **Dec** (short for **Decimal**) radio button is selected. To convert between different number bases, you simply select one of the other radio buttons: **Hex** for hexadecimal, or **Oct** for octal, or **Bin** for binary.

4. Either by typing on the computer's keyboard, or by using the mouse to click buttons on the calculator's front panel, enter the number **2165**.

5. Select the **Hex** radio button to convert 2165_{10} to hexadecimal, and record the value in the table below. Then select the **Bin** radio button convert the same number to binary, and record the value.

Decimal	Hexadecimal	Binary
2165_{10}		
	$3BE_{16}$	
		$1001\ 0110\ 1111_2$
1100_{10}		

6. In the same way, convert the other numbers listed above to the other bases. Record your results in the table. Using your knowledge of number bases, you should be able to check by hand that the three columns agree with each other.

PART 2: Online Hex Viewer Some earlier versions of Windows came with another useful type of program called a **hex viewer**. A hex viewer is a program that displays the contents of a disk file in ASCII code. Current versions of Windows don't include a hex viewer, but a Google search will turn up lots of hex viewers that you can download from the Web. Some hex viewers, including the one you'll use next, don't even require you to download anything: they work entirely in your Web browser.

In preparation for the following exercise, use your textbook's ASCII translation table (page 304) to fill in the following table. In each row, the first column contains a character from the message "ASCII rocks!" In the second column, you should fill in the ASCII code (in binary) for each character—you'll find that information in your book's ASCII table. In the third column, convert these binary codes to hex. I've filled in the first row for you. Make sure you understand that first row before you do the others.

Character	ASCII code in binary	ASCII code in hex
A	100 0001	41
S		
C		
I		
I		
r		
o		
c		
k		
s		
!		

To use the hex viewer, you'll need a file or two to work on, so first you'll use Microsoft Word to create two small files.

1. In Microsoft Word, create a new file that contains the message shown below in bold. Don't type anything else in the file.

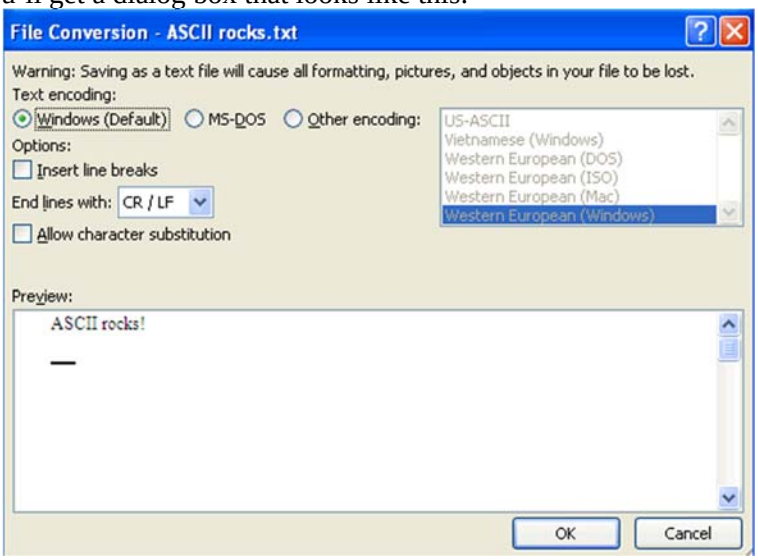
   ASCII rocks!

2. **Question:** How many characters are contained in the text that you typed? (Characters include letters, digits, spaces, and punctuation marks.)

   _____

3. Save your file as a Word document on your computer's desktop, with filename "Word.docx."

4. Save the file again, using Word's **Save As** option, but this time save it as a Plain Text file named "Text.txt." You'll get a dialog box that looks like this:



   Notice that Word is warning you that you'll lose some information if you proceed. That's okay, so click the OK button.

5. Still in Microsoft Word, open your original file (Word.docx) so that you have Word.docx and Text.txt open at the same time in two different windows. When you compare them in Word, the two files should look almost the same.

6. Close Microsoft Word. You should see your two files on the desktop. By right-clicking each file and selecting **Properties** from the shortcut menu, find the size of the each file in bytes. Record these sizes below. (I want you to record the Size, not the Size on Disk.)

   Size of Text.txt = _____bytes Size of Word.docx = _____bytes

7. **Question:** How does the number of bytes in Text.txt compare to the number of characters you typed in the file?

   _____

8. **Question:** How does the number of bytes in Word.docx compare to the number of characters you typed in the file?

   _____

9. You should have found that, even though the two files look the same when viewed in Word, one of the files is much larger than the other. That's because when you save a file as a Word document, Word automatically saves a lot of information in addition to the text you typed. (For example, it saves font sizes, font colors, the widths of your page margins, and so on.) But when you save it as a Plain Text file, Word only saves the text that you typed (plus a little more).

10. To examine the actual contents of your two files, let's look at the files in a hex viewer. In a Web browser, go to **webhex.net**. You'll see the following welcome screen:

11. Click the **Browse…** button, navigate to your desktop, and select Text.txt. Then click **Upload**. You should see this:

12. The information here is divided into two halves. The left half shows the ASCII code (in hex) for the contents of your file. The right half shows the contents of your file in normal text. The blue numbers in the top line simply count the ASCII codes (in the left half) or the characters (in the right half). Using the table that you built a couple of pages ago, verify that the ASCII code is correct for each character.

13. At the end of the file are two ASCII codes ($0D_{16}$ and $0A_{16}$) that don't stand for anything that you typed. Let's use the textbook's ASCII translation table to see what these codes stand for. First, we need to translate the codes into binary:

    Convert $0D_{16}$ to binary: _____ Convert $0A_{16}$ to binary: _____

14. **Question:** According to the ASCII translation table in your textbook and the definitions below the table, what is the definition of ASCII $0D_{16}$?

    _____

15. **Question:** According to the textbook's translation table and the definitions below the table, what is the definition of ASCII $0A_{16}$?

    _____

16. **Question:** Why did Microsoft Word automatically add $0D_{16}$ and $0A_{16}$ at the end of your file, even though you didn't type it? To understand why, read the first two or three screens of Wikipedia's article on **Newline**. After reading it, briefly explain in two or three complete sentences why it makes sense to end the file with these ASCII codes.

17. Finally, go back to webhex.net's home page and upload your other file (Word.docx). Recall that this file is a lot larger than Text.txt, so you'll see much more information on the screen. Most of it will be gibberish, but let's see if you can make sense of any of it. Click **Next offsets>>** to scroll through the entire contents of the file. You'll have to click it about 30 times. As you scroll through the file, look at the right half of the screen to see whether our message ("ASCII rocks!") ever shows up in readable form. Does it ever show up? (Yes or no?)

    _____

18. Here are two lessons to be learned from this exercise:
    a. When you save a file as Plain Text, the computer doesn't save much more than what you actually typed. But when you save it as a Word document, the computer saves a lot of formatting information, in addition to the text that you typed.
    b. When you save a file as Plain Text, the computer saves your text using the standard ASCII code. But when you save it as a Word document, the computer saves your text using a different code that the programmers at Microsoft decided to use instead of ASCII.

19. On your flash drive, locate a Word document from another class you've taken. Upload the Word document to webhex.net. You probably won't be able to make any sense out of what you see displayed. Then open the document in Word and save it as a Plain Text file. Upload that Plain Text file to webhex.net. You should be able to read the text in your file. (But no images, Web links, or formatting information in your document will be carried over into the Plain Text file.)

Part 3. 7483 Adder in Quartus

The 7483 is a 4-bit parallel adder, which means that it can add two 4-bit numbers to each other.

1. Use Quartus's New Project Wizard to create a new project called **lab6Adder7483**. Remember to use this same name for your project's working directory.

2. Create a new block diagram/schematic file, and save it as **lab6Adder7483**.

3. Place a **7483** symbol on your workspace, along with a **GND** symbol, eight **input** symbols, and five **output** symbols.

4. Connect the **GND** symbol to the 7483's **C0** terminal. By doing this you are forcing the adder's carry-in input to be 0 at all times.

5. Connect an **input** symbol to each of the 7483's remaining input terminals, and connect an **output** symbol to each of the 7483's output terminals.

6. Perform analysis and synthesis. Then assign input pin numbers so that you can use the leftmost four switches on the DE2-115 board to enter the value of A and the rightmost four switches to enter the value of B. Note that A4 is A's most significant bit and A1 is A's least significant bit. Also assign output pin numbers so that the adder's result will be displayed on the five rightmost green LEDs. Note that C4 is the result's most significant bit and S1 is the result's least significant bit.

7. Compile your design and download it to the DE2-115 board.

8. Test the following combinations. Fill in the table's Output columns, and verify that the five-bit output is the sum of the two four-bit inputs.

Inputs								Outputs				
A4	**A3**	**A2**	**A1**	**B4**	**B3**	**B2**	**B1**	**C4**	**S4**	**S3**	**S2**	**S1**
0	0	0	0	0	0	0	0					
0	1	0	1	1	0	0	0					
0	1	1	0	0	1	1	0					
0	0	0	1	1	1	1	1					
0	1	1	1	1	0	1	0					
0	0	0	1	0	0	0	1					
0	0	1	0	0	0	1	0					
1	1	1	0	0	1	0	1					

9. Now use the chip to add the following pairs of decimal numbers. Fill in all input and output columns in the table:
   - A = 7, B = 0
   - A = 4, B = 11
   - A = 8, B = 10
   - A = 12, B = 15

Inputs								Outputs				
A4	**A3**	**A2**	**A1**	**B4**	**B3**	**B2**	**B1**	**C4**	**S4**	**S3**	**S2**	**S1**

10. When you're finished, ask me to check your work.

Part 4. An Adder from Quartus's Library of Parameterized Modules (LPM)

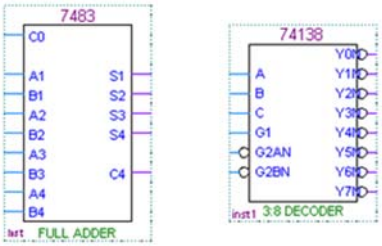
As you've learned, there are two general ways to enter a design in Quartus:

- By drawing a block diagram (or "schematic diagram"), which is saved in a .bdf file.
- By typing VHDL code, which is saved in a .vhd file.

Also, when using the block diagram/schematic method, you have a couple of choices:

- You can place and connect individual gates, which have names like **NOT** and **AND2**:

- You can place and connect simulated versions of 74XX chips, such as a 7483 adder or a 74138 decoder. (We'll study decoders next week.) Quartus calls these objects "macro-functions." They have names like **7483** and **74138:**



Next you'll learn about a third kind of object you can place on a block diagram: adders, decoders, and other components that are customized to have exactly the features you want, even if there's no 74XX chip with these features. These customizable components are stored in a collection that Quartus calls the Library of Parameterized Modules (or LPM), and most of them have names beginning with **lpm**, such as **lpm_add_sub** or **lpm_decode**.

11. Use Quartus's New Project Wizard to create a new project called **lab6LpmAdder**. Remember to use this same name for your project's working directory.

12. Create a new block diagram/schematic file, and save it as **lab6LpmAdder.bdf**.

13. Right-click in the blank workspace and select **Insert > Symbol**. For the name, type **lpm_add_sub**. This will bring up the symbol for an LPM_ADD_SUB component. Press the **OK** button.

14. The MegaWizard Plug-In Manager will open, and from its title bar you can see that it's starting you on page 2c of the Wizard. Select the **VHDL** radio button, and make sure that the output file name listed below is located inside your project directory. Then click **Next**.

15. On page 3 of the Wizard, we'll use the default settings (**Match project/default** is checked, data-bus width is **8** bits, and **Addition only)**. Click **Next**.

16. On page 4 of the Wizard, we'll use the default settings (**No, both values vary** and **Unsigned**). Click **Next**.

17. On page 5 of the Wizard, select **Create a carry input** and **Create a carry output.** Then click **Next**.

18. On page 6, we'll use the default setting (**No** pipelining). Click **Next**.

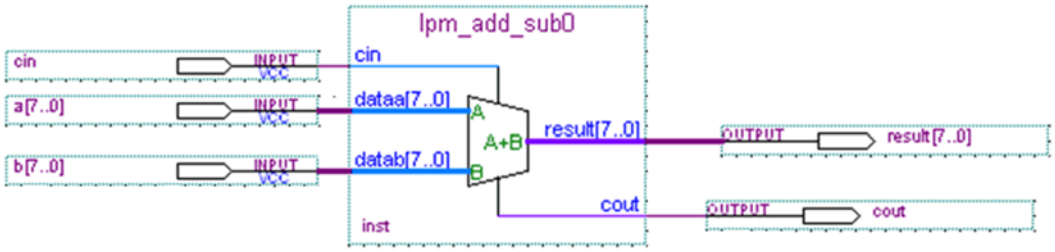
19. Page 7 displays the simulation libraries that will be used. Click **Next**.

20. Page 8 displays the output files that will be created. Click **Finish**.

21. A new dialog box will ask you about adding a Quartus IP file to your project. Click **Yes.**

22. Place your custom-designed adder on the workspace.

23. Place three inputs and two outputs. Connect them to your adder and give them the names shown in the picture below: **cin**, **a[7..0]**, **b[7..0]**, **result[7..0]**, and **cout**.



24. Notice two features of this diagram. First, some of the wires are thicker than normal. Second, some of the names end in [7..0]. Both of these features indicate that the wires in question are busses, which carry many bits at a time instead of a single bit. All three of our busses in this design are 8-bit busses. That's why the names show a range of numbers from 7 (most-significant bit) to 0 (least-significant).

25. Right-click on one of your thick purple wires, and notice that **Bus Line** is selected. On the other hand, right-click on one of your thin purple wires, and notice that **Node Line** is selected. In Quartus terminology, a node line carries a single bit, while a bus line carries many bits.

26. The inputs and outputs in this design serve the following functions:
    a[7..0]: the first 8-bit number that we're adding.
    b[7..0]: the second 8-bit number that we're adding.
    cin: the carry-in bit.
    result[7..0]: the result of our addition.
    cout: the carry-out bit.

27. Save your bdf file and perform analysis and synthesis. You should get no errors.

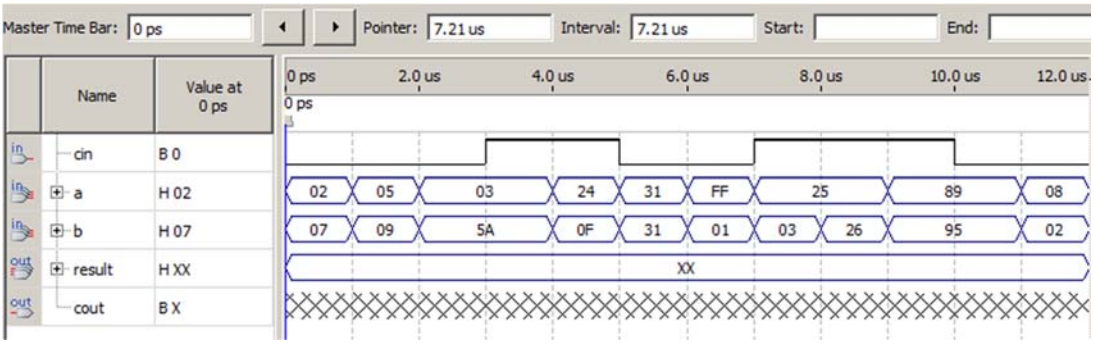
28. Create a new vector waveform file, and save it as **lab6LpmAdder.vwf**.

29. Using **Edit > Set End Time…**, set your end time to 12 µs.

30. Using **Edit > Grid Size…**, set your grid size to 1 µs.

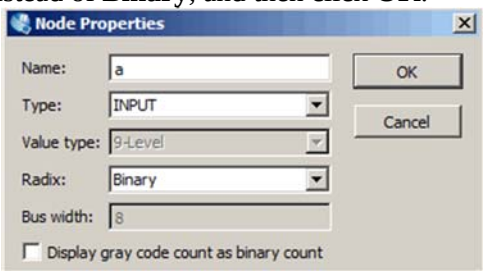
31. Double-click in the **Name** column of your waveform file and add the name of each of your inputs and outputs. You don't need to include the bit numbers as part of the names. For example, you can just type **a** instead of **a[7..0]**.

32. Now we want to set up the input waveforms to test the adder by giving it various combinations of numbers. **After you complete the next few steps**, your vector waveform file will look like this:



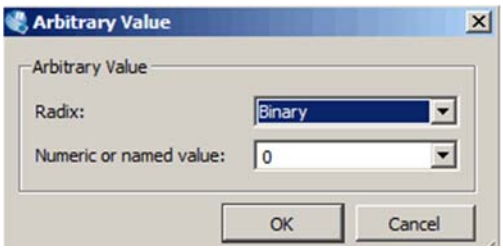
33. In this picture, the values of **a** and **b** are shown as hex numbers. For example, in the first column I've set **a** to hexadecimal 02 (which is 0000 0010 in binary), and I've set **b** to hex 07 (which is 0000 0111 in binary). How do I know that these are hexadecimal numbers? Because of the **H** in the column right after the names. You have a choice of displaying values in hex, or in binary, or in some other form. In your own waveform file, you'll probably see **B**s instead of **H**s, which means that your values are displayed in binary. To change this, right-click a name (such as **a**) and select **Properties**.

34. The dialog box shown below will open. In the **Radix** drop-down box, select **Hexadecimal** instead of **Binary**, and then click **OK**.

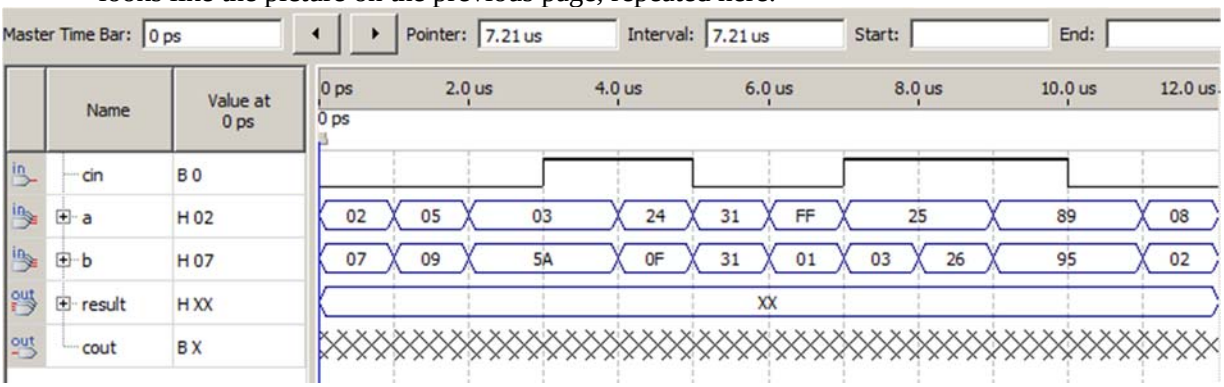


35. Next, how do you set the waveforms to specific values, as I've done in the picture above? For example, how did I make **cin** go high from 3 μs to 5 μs? Here's how. In your own waveform file you should have a constant LOW for **cin**. Use your mouse to select the portion of cin's waveform from 3 μs to 5 μs. Then do one of the following:
    - Select **Edit > Value > Arbitrary Value…** in the menus.
    - Or press the toolbar button that looks like this:  .
    - Or press **Ctrl+Alt+B** on the keyboard.

36. The dialog box shown below will open. Change the **Numeric or named value** to 1. Then click **OK**, and you should see that **cin** is now HIGH from 3 μs to 5 μs.



37. In the same way, set the values of all of your inputs so that your vector waveform file looks like the picture on the previous page, repeated here:



38. Save your vector waveform file and simulate your design. Your simulation results should show that initially, when a = 2 and b = 7, the result = 9, which makes sense. Check the rest of the result values to make sure they're correct. (You can use the Microsoft Calculator from Part 1 of this lab to quickly perform hex additions.) When you're satisfied that your values of **result** and **cout** are correct, call me over to check your work.

Part 5. Designing Arithmetic Circuits Using VHDL

In this section you will enter, test, and modify VHDL code from the textbook. As you'll see, VHDL often gives you different ways to do the same thing. This can be frustrating if you'd rather just be told the "one best way" to do it. But in the long run, there are advantages to being flexible and knowing different possible approaches to the same problem.

1. Use Quartus's New Project Wizard to create a new project called **lab6FourBitAdder**. Remember to use this same name for your project's working directory.

2. After reading pages 283-284 in the textbook so that you understand the code, type the code from Figure 5.22 into a new VHDL file named **fulladd.vhd**, and type the code from Figure 5.23 into a new VHDL file named **adder4.vhd**.

3. At this point our design contains two files, one of which defines an entity named **fulladd**, and the other of which defines an entity named **adder4**. Since adder4 contains several copies of fulladd, we call adder4 the "top-level entity" of this design. If you try to perform analysis and synthesis, you'll get an error because Quartus will be looking for a top-level entity whose name is the same as our project's name, lab6FourBitAdder. To fix this, click the **Assignments** menu and then click **Settings…**. In the resulting dialog box, click the Category named **General**, and

you'll see a text box that lets you tell Quartus the name of the design's top-level entity. In this box, type **adder4**, and then click the OK button.

4. Perform analysis and synthesis; then assign input and output pin numbers so that you can use:
   - The four leftmost slide switches on the DE2-115 board to input the four *x* bits, with x3 on the left and x0 on the right
   - The four rightmost slide switches to input the four *y* bits, with y3 on the left and y0 on the right
   - The leftmost push-button to input the value of Cin. ***Note:*** *Each push-button produces a value of 0 when you press it and a value of 1 when you do not press it.*
   - The five leftmost red LEDs to display the value of the result, with Cout on the left followed by the four *s* bits with s3 on the left and s0 on the right.

5. Compile your design and download it to the chip on the DE2-115 board.

6. Test your circuit by using the switches and pushbutton to enter various values of *x*, *y*, and Cin, making sure that the LEDs display the correct result. Show me your working circuit.

_____

7. Earlier in the chapter, Figures 5.4(c) and 5.5(b) gave two different ways of implementing a full adder using gates. In the code from Figure 5.22 that you typed into **fulladd.vhd**, do the simple assignment statements for *s* and *Cout* match the schematic diagram from Figure 5.4 or the one from Figure 5.5? (Circle your answer below.)

   Figure 5.4 Figure 5.5

   To make sure that the two schematic diagrams are equivalent, let's rewrite the code to use the other diagram. In the space below, write the VHDL code that matches the other diagram's implementation of *s* and *Cout*:

   s <= _____

   Cout <= _____

8. Modify the code in **fulladd.vhd** to use the equations you just wrote down instead of the original ones. Then re-compile your design and download it to the chip on the DE2-115 board.

9. Test your circuit as you did above, making sure that it produces correct results. Show me your working circuit.

_____

Next you'll learn about the common practice of placing component declarations in their own separate files.

10. Read the textbook's section named "Alternative Style of Code" on pages 285-286. Then type the code from Figure 5.24 into a new VHDL file named **fulladd_package.vhd**. Also revise the file named **adder4.vhd** so that it looks like Figure 5.25.

11. Re-compile your design and download it to the chip on the DE2-115 board. Test your circuit as you did above, making sure that it still works correctly. Show me your working circuit.

_____

For multi-bit signals that contain many bits, it's not very convenient to have to define each bit as its own separate signal, such as x0, x1, x2, and so on. Let's learn a better way to do this.

12. Read the book's Section 5.5.3 on pages 286-287. Then revise the file named **adder4.vhd** so that it now looks like Figure 5.26.

13. Perform analysis and synthesis. Then, since the names of some signals have changed, you'll need to reassign some of the input and output pins numbers. Then compile your design and download it to the chip on the DE2-115 board. Test your circuit as you did above, making sure that it still works correctly. Show me your working circuit.

_____

14. Next, revise your code so that it implements an 8-bit adder instead of a 4-bit adder. You'll need to figure out which file(s) to revise and what changes to make.

15. Before downloading to the DE2-115 board, reassign input and output pin numbers so that you can use:
- The eight leftmost slide switches for *x*, with x(7) on the left and x(0) on the right.
- The eight rightmost slide switches for *y*, with y(7) on the left and y(0) on the right.
- The leftmost push-button to input the value of Cin.
- The nine leftmost red LEDs to display the value of the result, with Cout on the left followed by the eight *s* bits with s(7) on the left and s(0) on the right.

16. Re-compile your design and download it to the chip on the DE2-115 board. Test your circuit as you did above, making sure that it still works correctly. Show me your working circuit.

_____

17. Close your Quartus project before you go on to the next circuit.

Now let's take a completely different approach to adder circuits by taking advantage of VHDL's built-in support for arithmetic operators such as +.

18. Use Quartus's New Project Wizard to create a new project called **lab6SixteenBitAdder**. Remember to use this same name for your project's working directory.

19. Read the first two paragraphs in the book's Section 5.5.4, starting on page 287 and ending at the bottom of page 288. Then type the code from Figure 5.27 into a new VHDL file named **adder16.vhd**.

20. As you learned to do above, use the menus in Quartus to change the project's top-level entity to **adder16**. Then perform analysis and synthesis.

21. Let's simulate this design instead of downloading it to the chip. Following steps similar to the ones that you followed earlier in this lab for the LPM adder, create a new vector waveform file named **lab6SixteenBitAdder.vwf**. Set this file's end time to 12 µs and its grid size to 1 µs. Then, in the **Name** column of the file, add the names of your design's input and output signals (**X**, **Y**, and **S**). We want the values of these signals to be displayed in hex, not binary, so take the necessary steps to make this happen.

22. Rather than specifying certain values for X, let's tell Quartus to generate random values. To do this, click **X** in the **Name** column. This should highlight the entire line for X with a blue highlight. Then do one of the following:
    - Select **Edit > Value > Random Values…** in the menus.
    - Or press the toolbar button that looks like this:  .
    - Or press **Ctrl+Alt+R** on the keyboard.

23. A dialog box will open to let you choose how frequently the random values change. Accept the default choice by clicking **OK**.

24. Repeat the previous two steps for Y so that it too is assigned random values.

25. Save your vector waveform file and simulate your design. Check the simulation results values to make sure they're correct. (Use the Microsoft Calculator from Part 1 of this lab.) When you're satisfied that your values of **S** are correct, call me over to check your work.

_____

26. Read the book's page 289, which describes a more few features we'd like our adder to have. Then modify the code in **adder16.vhd** to look like Figure 5.28.

27. Perform analysis and synthesis, and then modify **lab6SixteenBitAdder.vwf** so that it includes the new input signal (Cin) and output signals (Cout and Overflow). Tell Quartus to assign random values to Cin.

28. Save your vector waveform file and simulate your design. When you're satisfied that your values of **S**, **Cout**, and **Overflow** are correct, call me over to check your work.

_____