Name_____

EET 1131 Lab #7
Arithmetic Circuits

Equipment and Components

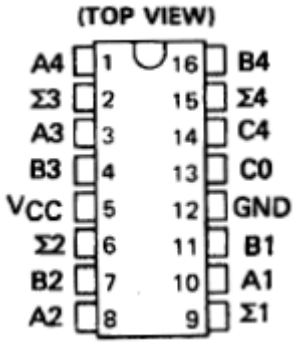
- Safety glasses
- ETS-7000 Digital-Analog Training System
- Integrated Circuits: 7483, 74181
- Quartus II software and Altera DE2-115 board
- Multisim simulation software

PART 1. 7483 Parallel Adder

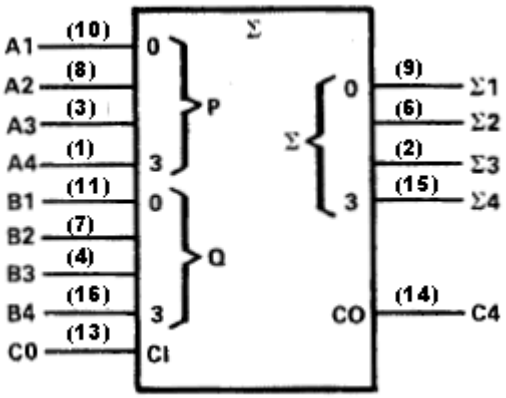
A chip's datasheet contains two diagrams showing the pin names and pin numbers. These two diagrams are organized differently. One of them, called the **pin diagram**, shows the pins in their actual positions on the physical DIP. The other one, called the **logic symbol**, usually shows input pins on the left and output pins on the right. Both diagrams are important and useful. The pin diagram is more useful when you're wiring the chip in a circuit. The logic symbol is more useful when you're thinking about how data flows through the chip.

Here are the pin diagram and logic symbol for a 7483 parallel adder.

7483 pin diagram



7483 logic symbol



1. Use the 7483's logic symbol to answer the following questions:

   How many **input** pins does a 7483 have? _____ How many **output** pins does a 7483 have? _____

2. Wire a 7483 chip on the breadboard. In addition to power and ground connections,
   - Make the chip's **C0** input constant LOW by connecting it to ground.
   - Connect the chip's **A** inputs to the trainer's data switches SW7 through SW4, with SW7 as the most significant bit (MSB) and SW4 the least significant bit (LSB).
   - Connect the chip's **B** inputs to the trainer's data switches SW3 through SW0, with SW3 as the MSB and SW0 the LSB.
   - Connect the chip's **Σ** outputs to LEDs 3 through 0, with LED 3 as the MSB and LED 0 as the LSB.
   - Connect the chip's **C4** output to LED 4.

3. 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**	**Σ4**	**Σ3**	**Σ2**	**Σ1**
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					

4. 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**	**Σ4**	**Σ3**	**Σ2**	**Σ1**

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

Safety glasses? _____ Circuit works? _____ DIPs inserted correctly? _____

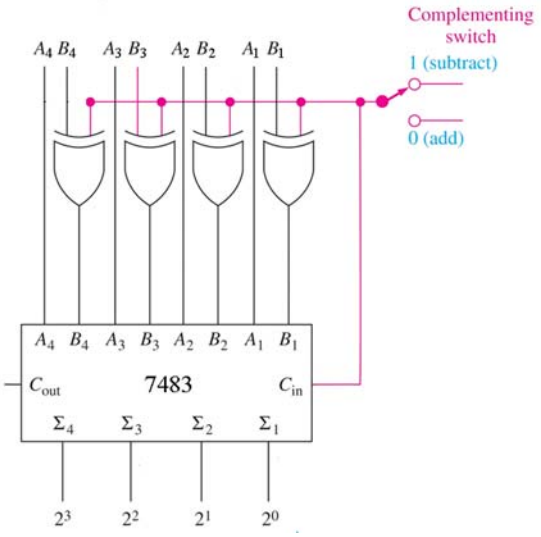
Using power/ground busses?_____ Wire colors? _____

Wire lengths? _____Wire ends trimmed? _____ DIPs accessible? _____

You're finished with this circuit and can take it apart.

PART 2. An Adder/Subtractor Circuit in Quartus II

Figure 7-23 in your textbook shows how to use two 4008 adder chips and a controlled inverter to build an 8-bit two's-complement adder/subtractor circuit. The 4008 chip in the figure is almost identical to the 7483 chip that you used above. To keep the wiring manageable, let's build (in Quartus II) a 4-bit version of this circuit, which looks like this:



1. In Quartus II, create a new project named **Lab7AdderSubtractor**.

2. Create a block diagram/schematic file that contains the circuit shown above.
   - Your design will have nine inputs (four for *A*, four for *B*, and one to choose between adding and subtracting).
   - It will have four outputs for ∑.

3. Perform analysis and synthesis, and then assign pin numbers as follows:
   - Use four of the DE2-115 board's slide switches for your four *A* inputs, in order (with the MSB on the left and the LSB on the right).
   - Use four more slide switches for your four *B* inputs, in order.
   - Use another slide switch for your add/subtract input.
   - Use four LEDs for your four outputs, in order.

4. Compile your design and download it to the chip.

5. To test your circuit, input several different values of your choice for *A* and *B*, and observe the results on the LEDs. The LEDs should display either *A*+*B* or *A*−*B*, depending on the position of the add/subtract switch.

6. When you're confident that your circuit is working, call me over to check it.

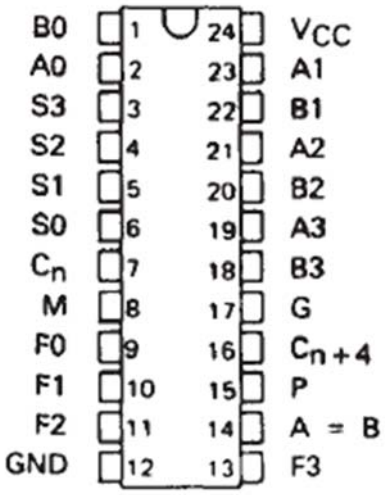
You're finished with this circuit and can take it apart.

PART 3. 74181 Arithmetic/Logic Unit

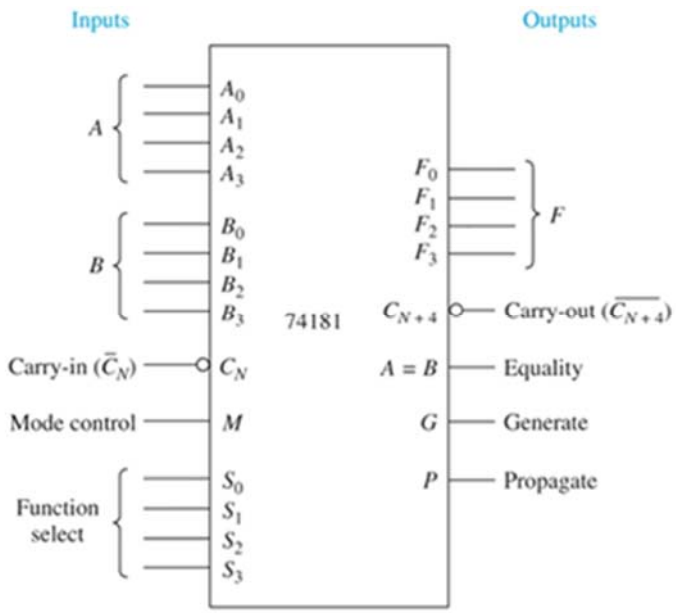
The 74181 chip is the forerunner of the math circuitry inside today's microprocessors. This arithmetic/logic unit is a medium-scale integration (MSI) chip that can perform sixteen arithmetic operations and sixteen logic operations on two 4-bit input numbers.

Shown below are the pin diagram and logic symbol for a 74181.

74181 pin diagram



74181 logic symbol



1. Use the 74181's logic symbol to answer the following questions:

   How many **input** pins does a 74181 have? _____

   How many **output** pins does a 74181 have? _____

2. Wire a 74181 chip on the breadboard. In addition to power and ground connections, make the following connections:
   - Connect the chip's **M** input to the trainer's switch SW7.
   - Connect the chip's **S3** through **S0** inputs to the trainer's switches SW3 through SW0 (in that order).
   - Hard wire a binary value of 0101_2 into **A3-A0** and a value of 0011_2 into **B3-B0**. (We're hardwiring these inputs because the trainer doesn't have enough switches to accommodate all of the inputs.)
   - Make the chip's **Carry-in** input constant HIGH by connecting it to V_{CC}.
   - Connect the chip's **F3-F0** outputs to the trainer's LEDs 3 through 0 (in that order).

Testing the Logic Functions:

3. Use the function table in your textbook's Figure 7-27 to predict the outputs **F3-F0** for the inputs shown below if the chip is in logic mode (with input **M** set HIGH). Record your predictions in the "Predicted" column of the table.

4. In your breadboarded circuit, set the **M** bit HIGH to put the chip in logic mode. Go through all possible combinations of inputs **S3-S0** and record the chip's outputs in the "Measured" column of the table.

Instruction Select				**Outputs with M = 1 (Logic mode) and A = 0101_2 and B = 0011_2**							
				Predicted				**Measured**			
S_3	**S_2**	**S_1**	**S_0**	**F_3**	**F_2**	**F_1**	**F_0**	**F_3**	**F_2**	**F_1**	**F_0**
0	**0**	**0**	**0**								
0	**0**	**0**	**1**								
0	**0**	**1**	**0**								
0	**0**	**1**	**1**								
0	**1**	**0**	**0**								
0	**1**	**0**	**1**								
0	**1**	**1**	**0**								
0	**1**	**1**	**1**								
1	**0**	**0**	**0**								
1	**0**	**0**	**1**								
1	**0**	**1**	**0**								
1	**0**	**1**	**1**								
1	**1**	**0**	**0**								
1	**1**	**0**	**1**								
1	**1**	**1**	**0**								
1	**1**	**1**	**1**								

Circuit Check:

When you're confident that your table above is correct, and that you understand how the 74181 chip works in logic mode, call me over to check your circuit.

Safety glasses? _____ Circuit works? _____ DIPs inserted correctly? _____

Using power/ground busses?_____ Wire colors? _____

Wire lengths? _____Wire ends trimmed? _____ DIPs accessible? _____

Testing the Arithmetic Functions:

5. Use the function table in your textbook's Figure 7-27 to predict the outputs **F3-F0** for the inputs shown below if the chip is in arithmetic mode (with **M** set LOW). Record your predictions in the "Predicted" column of the table below.

6. On the breadboard, set the **M** bit LOW to put the chip in arithmetic mode. Go through all possible combinations of inputs **S3-S0** and record the chip's outputs in the "Measured" column of the table.

Instruction Select				**Outputs with M = 0 (Arithmetic mode) and A = 0101_2 and B = 0011_2**							
				Predicted				**Measured**			
S_3	**S_2**	**S_1**	**S_0**	**F_3**	**F_2**	**F_1**	**F_0**	**F_3**	**F_2**	**F_1**	**F_0**
0	**0**	**0**	**0**								
0	**0**	**0**	**1**								
0	**0**	**1**	**0**								
0	**0**	**1**	**1**								
0	**1**	**0**	**0**								
0	**1**	**0**	**1**								
0	**1**	**1**	**0**								
0	**1**	**1**	**1**								
1	**0**	**0**	**0**								
1	**0**	**0**	**1**								
1	**0**	**1**	**0**								
1	**0**	**1**	**1**								
1	**1**	**0**	**0**								
1	**1**	**0**	**1**								
1	**1**	**1**	**0**								
1	**1**	**1**	**1**								

Circuit Check:

When you're confident that your table above is correct, and that you understand how the 74181 chip works in arithmetic mode, call me over to check your circuit.

_____

You're finished with this circuit and can take it apart.

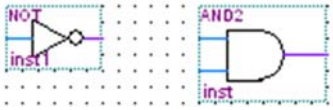
Part 4. Quartus II's Library of Parameterized Modules (LPM)

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

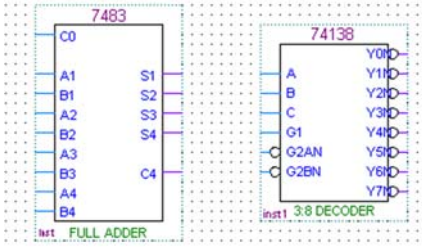
- By drawing a block diagram (or "schematic diagram"), which is saved in a .bdf file.
- By typing VHDL code, which is saved in a .vhd file.

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 II 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 II 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**.

(The following steps are a simplified version of Example 7−28 on page 299 of the textbook.)

1. Use Quartus II's New Project Wizard to create a new project called **lab7_lpm_add_sub**. Remember to use this same name for your project's working directory.

2. Create a new block diagram/schematic file, and save it as **lab7_lpm_add_sub.bdf**.

3. 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. Make sure that **Launch MegaWizard Plug-In** is checked, and press the **OK** button.

4. 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**.

5. 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**.

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

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

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

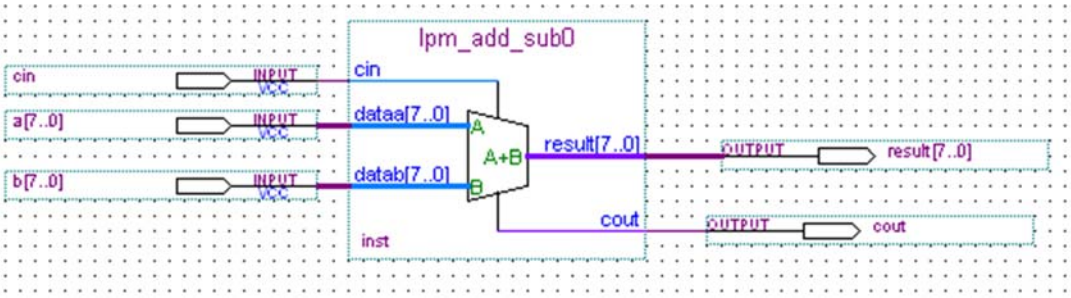
9. Page 7 displays the simulation libraries that will be used. Click **Next**.

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

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

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

13. 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**.



14. Notice two features of this diagram that you saw in Lab 6. 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).

15. 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. Recall that a node line carries a single bit, while a bus line carries many bits.

16. 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.

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

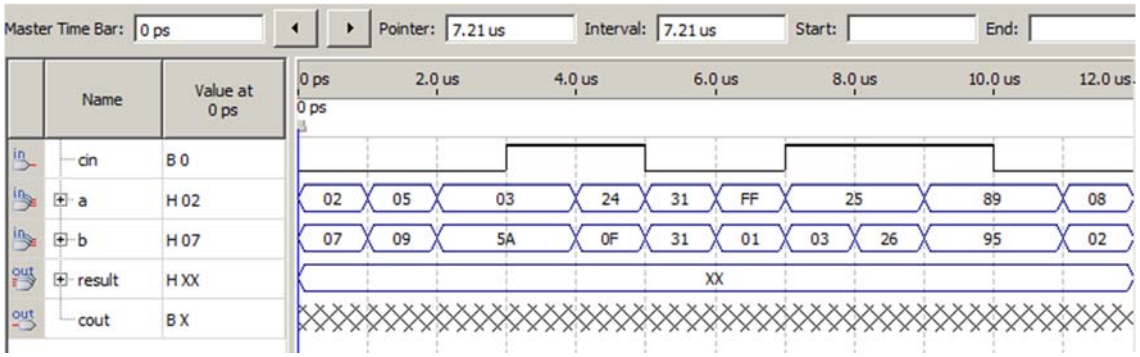
18. Create a new vector waveform file, and save it as **lab7_lpm_add_sub.vwf**.

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

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

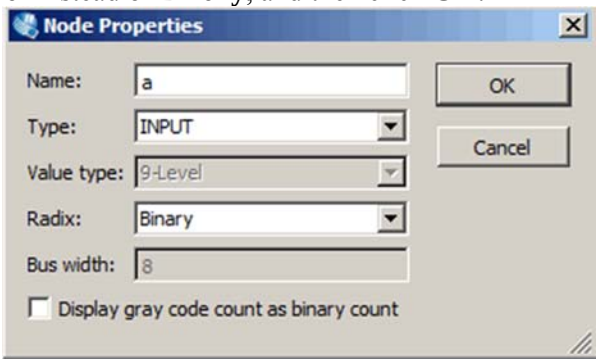
21. Double-click in the **Name** column of your waveform file and add the name of each of your inputs and outputs. You don't need to include the bit numbers as part of the names. For example, you can just type **a** instead of **a[7..0]**.

22. 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:



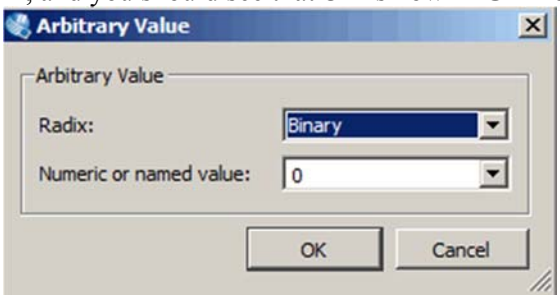
23. In this picture, the values of **a** and **b** are shown as hex numbers. For example, in the first column I've set **a** to hexadecimal 02 (which is 0000 0010 in binary), and I've set **b** to hex 07 (which is 0000 0111 in binary). How do I know that these are hexadecimal numbers? Because of the **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**.

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

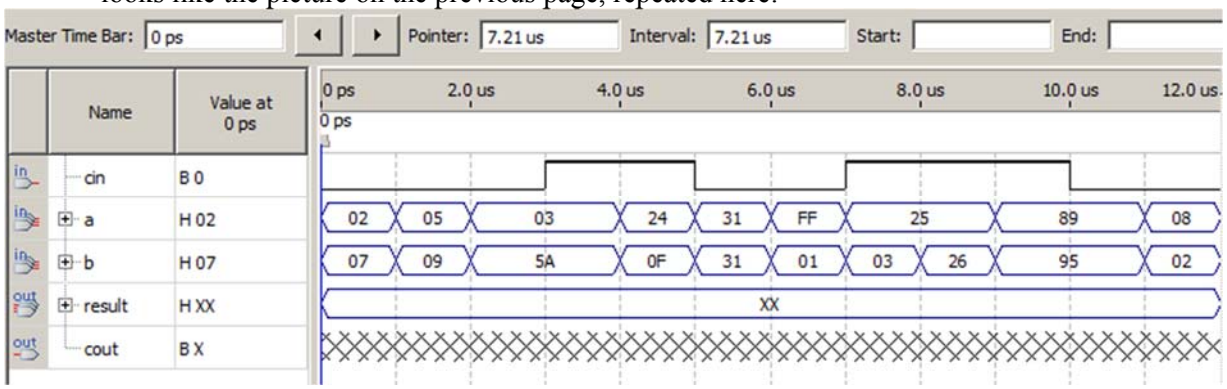


25. Next, how do you set the waveforms to specific values, as I've done in the picture above? For example, how did I make **cin** go high from 3 µs to 5 µs? Here's how. In your own waveform file you should have a constant LOW for **cin**. Use your mouse to select the portion of cin's waveform from 3 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.

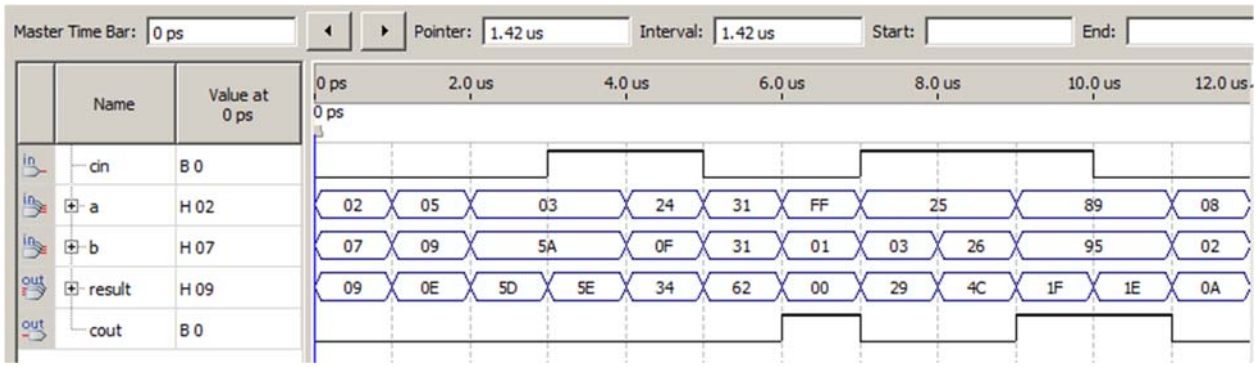
26. The dialog box shown below will open. Change the **Numeric or named value** to 1. Then click **OK**, and you should see that **cin** is now HIGH from 3 µs to 5 µs.



27. 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:



28. Save your vector waveform file and simulate your design. Your simulation results should look like this:



29. When you're satisfied that your results match mine, and you understand why these results are correct, call me over to check your work.