Introduction to Computer Engineering (E114)

Lab 5: 32-bit ALU

Introduction

In this lab, you will build the 32-bit Arithmetic Logic Unit (ALU) that is described in your book in Section 4.5. Your ALU will make use of the Full Adder that you made in Lab 1, and will become an important part of the MIPS microprocessor that you will build in later labs.

Background

You should already be familiar with the ALU from Section 4.5 of your book. The design in this lab will make only a few modifications to the ALU schematics shown in Figures 4.17 and 4.19 (pages 238, 240).

Your ALU schematics will demonstrate the power and importance of hierarchical design. Your working 32-bit ALU will contain four levels of hierarchy. Each level is built using one or more schematic symbols from the previous level. You will also draw the schematics for the different levels in this order.

1. Full Adder (copied from lab 1)
2. 1 bit ALU
3. 4 bit ALU (with zero detection logic)
4. 32-bit ALU (with zero detection logic).

When you use this ALU inside your MIPS microprocessor design for later labs, you will have even more levels of hierarchy!

1. Schematics

The first schematic you will need to draw for this lab is for the 1-bit ALU, shown in Figure 4.17 in your book. In the figure, the box labeled with a plus sign is a full adder. Your 1-bit ALU will have inputs A, B, CarryIn, Less, Binvert, and Operation[1:0]; and outputs Result and CarryOut.

Before you can use your Full adder in your 1-bit ALU schematic, you will need to make it into a symbol and copy it from your Lab 1 project. Open the Xilinx Project Manager with your lab 1 project now. Then double-click on the file with the ".sch" extension to open your full adder schematic in the schematic editor. Create a symbol from the schematic (as you did in lab 3) by selecting 'Hierarchy•Create Macro Symbol from Current Sheet' from the menu. Name the symbol something like "FULLADD" and click OK. This adds the full adder as a symbol in your lab 1 symbol library.

Next, you need to create a new Project for this lab using File•New Project. Name the project something like "lab5_xx" (where xx are your initials). Once you have the new project opened in the Project Manager, select 'Tools•Utilities•Schematic Symbol Library Manager' from the menu.

In the Library Manager, select the "Libraries" tab to display the list of project libraries. Double-click on your library from project 1 (it should be named "lab1_xx"). The library manager should now list the contents of your project library, which should contain only the FULLADD symbol that you just created. Right-click on the FULLADD symbol and select "Copy" from the menu. At the bottom of the "Copy Object" window that appears, select your new lab 5 project (named "lab5_xx") and press OK.

You may want to edit pin placements of the symbol for the full adder (as you did in lab 3) by double-clicking on the symbol name FULLADD in the library manager. It will be most convenient if you place the CARRY_IN input at the top of the symbol and the CARRY_OUT output at the bottom, as on the full adder used in Figure 4.17.

The symbol for your full adder should now be accessible for use in your schematics for this lab. Open the Schematic Editor now and use your FULLADD symbol in drawing the schematic for your 1-bit ALU (Figure 4.17). To draw the 2 to 1 and 4 to 1 multiplexers shown in the figure, use the symbols named M2_1 and M4_1E. Note that the M4_1E has an enable input, which is not needed for your 1-bit ALU since this mux should always be enabled. Wire the enable input to high by attaching a VCC component that you may select from the library. You may find the GND component in the library to be useful later as a source of logic 0.

Although there are two slightly different 1-bit ALUs pictured in Figure 4.17, you actually only need to build one, since we will not be using overflow detection. Your 1-bit ALU should be like the one in the top of the figure, except with a "Set" output added as in the bottom of the figure.

Note that the OPERATION input is two bits wide, which requires the use of a bus. The separate bits of the OPERATION signal need to be wired to the selector inputs (S0, S1) of your 4 to 1 multiplexer (M4_1E). After drawing the wires form the selector inputs to the OPERATION bus, don't forget to add bus taps (as you learned in Lab 2).

When you have completed the schematic for your 1-bit ALU, make it into a symbol called ALU_1 for use in the next level of your hierarchical design.

It would be possible to build a 32-bit ALU now using 32 1-bit ALUs, as in Figure 4.19. However, this would require a lot of tedious wiring in your schematic! It is recommended that you instead build a 4-bit ALU using your 1-bit ALU, and then use eight 4-bit ALUs to build a 32-bit ALU. Your 4-bit ALU will have inputs A[3:0], B[3:0], CarryIn, Less, Binvert, and Operation[1:0]; and outputs Result[3:0], Zero, Set, and CarryOut.

To build your 4-bit ALU, you can piece together 1-bit ALUs just as in Figure 4.19. However, note that the LESS input of the first 1-bit ALU and the SET output of the last 1-bit ALU are special cases. To accommodate for the correct wiring of these signals in your 32-bit ALU, your 4-bit ALU needs to have a LESS input (which is connected to the LESS input of its least significant 1-bit ALU) and a SET output (which comes from the SET output of its most significant ALU). In your 32-bit ALU, the LESS input of all but the first ALU will be connected to ground, while the SET output of all but the last ALU will be disconnected, as shown in Figure 4.18 of P&H.

Finally, you will need to add some zero detection logic to your 4-bit ALU. This logic must provide an output that can be used in the zero detection logic of the 32 bit ALU that you will build, which must also have a ZERO output that is asserted when all bits of the RESULT are zero. It is up to you to design the zero detection logic in both the 4-bit and 32-bit ALUs.

Draw the schematic for your 4-bit ALU now and convert it into a symbol named "ALU_4."

Now all you have to do is piece together 8 of your 4-bit ALUs to make a 32-bit ALU. Your 32-bit ALU will have inputs A[31:0], B[31:0], Bnegate, and Operation[1:0]; and outputs Result[31:0], and Zero. Note that this set of inputs differs slightly from Figure 4.21 in the book.

This time, you will need to use 32-bit buses for the A, B, and RESULT signals. To connect these inputs to the 4-bit ALUs, draw connecting busses, then label the connecting busses with the appropriate bits by double-clicking on the connecting busses and entering the appropriate bits (e.g. A[7:4] is an input to the second 4-bit ALU). . Note that in the 32-bit ALU, the CARRY_IN and B_INVERT inputs are combined into a single B_NEGATE input. This is because both of those inputs are always the same values at the same time anyway. You can simply connect the B_NEGATE input to both the CARRY_IN of your first 4-bit ALU and all of the B_INVERT inputs of all of the 4-bit ALUs. Also, don't forget to add the zero detection logic to assert the ZERO signal when all 32 bits of RESULT are zeros.

2. Simulation and Testing

Once you have completed your 32-bit ALU, you are ready to test it in the simulator. We would like to apply the following set of test vectors to verify the basic functionality. For each test, the Operation[1:0], Bnegate, A[31:0], and B[31:0] inputs are given in hexadecimal. The expected Result[31:0] is also given. Complete the missing entries of the test vector table.

Test	Operation	Bnegate	A	B	Expected Result
ADD 0+0	2	0	00000000	00000000	00000000
ADD 0+(-1)	2	0	00000000	FFFFFFFF	FFFFFFFF
ADD 1+(-1)	2	0	00000001	FFFFFFFF	00000000
ADD FF+1	2	0	000000FF	00000001	
SUB 0-0	2	1	00000000	00000000	00000000
SUB 0-(-1)			00000000	FFFFFFFF	00000001
SUB 1-1			00000001		
SUB 100-1			00000100		
SLT 0,0	3	1	00000000	00000000	00000000
SLT 0,1			00000000		00000001
SLT 0,-1			00000000		
SLT 1,0			00000001		
SLT –1,0			FFFFFFFF		
AND FFFFFFFF, FFFFFFFF			FFFFFFFF		
AND FFFFFFFF, 12345678			FFFFFFFF	12345678	12345678
AND 12345678, 87654321			12345678		
AND 00000000, FFFFFFFF			00000000		
OR FFFFFFFF, FFFFFFFF			FFFFFFFF		
OR 12345678, 87654321			12345678		
OR 00000000, FFFFFFFF			00000000		
OR 00000000, 00000000			00000000		

Before simulating, check in the Project Manager that only the 32-bit ALU schematic is in the project. If the project includes other schematics such as the four bit ALU, they will all be simulated simultaneously, leading to very strange behavior in the simulator because the names of some signals on the 4 and 32 bit ALU are identical. Instead, the smaller blocks should only appear as macros called by the bigger blocks.

Fire up the simulator and add the input and output signals. Set the simulation time step to 210 ns and the waveform scale to 500 ps/div. Create formula inputs F0, F1, F2, and F3, and assign them to Operation[1:0], BNegate, A[31:0], and B[31:0], respectively, just as you did in lab 2. The formula for F2 should be:

[00000000]20[00000001]10[000000Ff]10[00000000]20[00000001]10[00000100]10[00000000]30[00000001]10[ffffffff]10[ffffffff]20[12345678]10[00000000]10[ffffffff]10[12345678]10[00000000]20

Press the Simulation Step button to run the simulation and check that the test vectors are what you intended and that Result and Zero outputs agree with your expectations.

If the outputs of your ALU are incorrect in any of the cases, you may find it most efficient to go back and test the lower levels in your schematic hierarchy, rather than trying to debug the entire 32-bit ALU at once. You already tested your full adder in lab 1, so you could test the 1-bit ALU separately to make sure it works, and then work your way back up to the 32-bit ALU until it works correctly.

To rerun a simulation after changing your schematic or input vectors, use the Device•Power On/Reset command, then press simulation step again. One tip for debugging: if the simulator gives a question mark as the value of an output, it usually means that somewhere in your schematics there is a disconnected wire. Be sure that the M4_1E enable input is tied to high, and that all of the LESS inputs except for one are tied low.

What to Turn In

Please turn in each of the following items:

1. Please indicate how many hours you spent on this lab. This will not affect your grade, but will be helpful for calibrating the workload for next semester's labs.
2. Turn in a completed table of test vectors.
3. Printouts of each of your schematics. Be sure they are legible; try not to use schematic pages larger than B-sized.
4. Printouts of your test waveforms.
5. EXTRA CREDIT: Modify your ALU to use a fast carry-lookahead adder instead of the slow ripple carry, as described in your book starting on page 241. Test the new ALU to make sure you did not accidentally introduce any bugs. Turn in printouts of your revised schematics.