Lab 5: 32-bit ALU

Introduction

In this lab, you will build the 32-bit Arithmetic Logic Unit (ALU) that is described in *Computer Organization and Design* in Section B.5 of Appendix B on the CD. 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 B.5. The design in this lab will make only a few modifications to the ALU schematics shown in Figures B.5.10 (page B-33) and B.5.12 (page B-35).

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!

Schematics

The first schematic you will need to draw for this lab is for the 1-bit ALU, shown in Figure B.5.10 of Appendix B on the CD. 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, AINVERT, 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 copy it from your Lab 1 project and make it into a symbol. First you can 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 Navigator, create a new schematic file using Project→New Source and name it "fulladd". In the Schematic Editor, instead of re-drawing the schematic of the full adder, you can use File→Open to revisit your file from Lab 1. Choose Edit→Select All, click the right mouse and choose copy. Then switch back to the *fulladd.sch* file, click right mouse and choose paste. You now get a copy of the schematic of the full adder created in Lab 1. Save the schematic file and create a symbol for it (as you did in Lab 3) by selecting Create Schematic Symbol in the Design Entry Utilities in the process window. This adds fulladd as a symbol in your lab5 symbol library. The other way to reuse the full adder that you have built in Lab 1 is simply by choosing Project→Add Source to locate the *lab1_xx.sch* file and adding it to the project. You will similarly create a symbol out of this schematic file, but the name of the symbol will be lab1_*xx* instead of fulladd.

Next, create a new schematic for the 1-bit ALU using the fulladd symbol or lab1_*xx* symbol. Again, you can edit the symbol itself. It will be most convenient if you place the CARRYIN input at the top of the symbol and the CARRYOUT output at the bottom, as on the full adder used in Figure B.5.10. In addition to the full adder, draw the 2-to-1 and 4-to-1 multiplexers shown in the figure, using the symbols named M2_1 and M4_1E under the category Mux. Note that M4_1E has an enable input, which is not needed for your 1-bit ALU, since this multiplexer should always be enabled. Wire the enable input to high by attaching a VCC component to it. VCC can be found in the library under the category General. 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 B.5.10, 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. Create the OPERATION bus as you learned in Lab 2. The separate bits of the OPERATION signal need to be wired to the selector inputs (S0 and S1) of your 4-to-1 multiplexer (M4_1E). Also remember to name the individual wires of the bus to be OPERATION(0) and OPERATION(1).

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 B.5.12. However, this would require a lot of tedious wiring in your schematic! Instead it is recommended that you 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, AINVERT, BINVERT and OPERATION(1:0); and outputs RESULT(3:0), ZERO, SET and CARRYOUT. For A(3:0), B(3:0) and RESULT(3:0), you need to properly create the buses as you did in Lab 2, wire each input bit properly to the 1-bit ALU and name each wire of the bus. For the OPERATION(1:0) bus, there is no need to create a bus because both wires of the input bus are sent to all four of the 1-bit ALUs.

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 (which is in the symbol library under the category General), while the SET output of all but the last ALU will be disconnected, as shown in Figure B.5.11 of Appendix B.

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.

When you are done with the schematic for your 4-bit ALU, convert it into a symbol named ALU_4."

Now all you have to do is piece together eight of your 4-bit ALUs to make a 32-bit ALU. Your 32-bit ALU will have inputs A(31:0), B(31:0), AINVERT, BNEGATE and OPERATION(1:0); and outputs RESULT(31:0) and ZERO. Note that this set of inputs differs slightly from Figure B.5.14 in the book. A common mistake is to inadvertently use an input bus rather than an output bus for RESULT.

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, you need to create new buses with width of 32, and bus taps properly connected to the 4-bit bus inputs of each ALU_4. You need to label the connecting busses as before, but now named with the appropriate range of bits (e.g., A(7:4) is an input to the second 4-bit ALU). Note that in the 32-bit ALU, the CARRYIN and BINVERT inputs are combined into a single BNEGATE input. This is because both of those inputs are always the same values at the same time anyway. You can simply connect the BNEGATE input to both the CARRYIN of your first 4-bit ALU and all of the BINVERT 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.

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), AINVERT, 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 in Figure 1. Note that the entries are specified in hexadecimal to reduce the amount of writing.

Now set up a testbench waveform to enter all the input vectors for A(31:0), B(31:0), OPERATION(1:0), AINVERT and BNEGATE. Make sure you first choose Display in Hex, then click the beginning of each blue cell. You can then enter the input vectors. Save the waveform when it is done.

Now launch Simulation Behavioral Verilog Model in the source process window and check the output vectors RESULT and ZERO. If the results don't agree with your expectations, check the inputs to see if there is any typographic error in the initial waveform setup. If you still have trouble seeing the correct results, go back to the schematic files and check the layout. You may find it most efficient to first 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

Test	OPERATION	AINVERT	BNEGATE	A	B	Expected RESULT
ADD 0+0	2	0	0	00000000	00000000	00000000
ADD 0+(-1)	2	0	0	00000000	FFFFFFFF	FFFFFFFF
ADD 1+(-1)	2	0	0	00000001	FFFFFFFF	00000000
ADD FF+1	2	0	0	000000FF	00000001	
SUB 0-0	2	0	1	00000000	00000000	00000000
SUB 0-(-1)				00000000	FFFFFFFF	00000001
SUB 1-1				00000001		
SUB 100-1				00000100		
SLT 0,0	3	0	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		

FIGURE 1 Test vectors to verify functionality.

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.

After changing your schematic or input vectors, re-run a simulation and repeat the process until you are successful. 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.

Extra Credit

Modify your ALU to use a fast carry-lookahead adder instead of the slow ripple carry, as described in *Computer Organization and Design* in Section B.6 of Appendix B on the CD. Test the new ALU to make sure you did not accidentally introduce any bugs.