Digital Electronics & Computer Engineering (E85)

Lab 9: MIPS Processor

Introduction

In this lab, you will complete your model of the MIPS processor and run the Fibonacci program on it! This will involve putting together the ibox, the dbox, the ebox, and the cbox. You already built the ibox in Lab 8. The cbox is your controller from Lab 2. In this lab you will complete the ebox by incorporating your ALU from Lab 5 and your sgnext logic from Lab 8. Finally, you will design the dbox by yourself. You will need to refer to your solutions from Labs 2, 5, and 8 while assembling and testing your processor.

In Lab 8, you were responsible for correcting parts of Lab 2 and Lab 5 that may not have worked. If you did not do so, do it right now. You may refer to the published solutions or solutions from other students, but may not copy files from anyone else.

1. Getting Started

Please follow these directions very carefully. There have been innumerable problems with missing or duplicate macros caused by misunderstanding the procedures to copy your old files.

To get started, use File?Copy Project to copy lab9base from the g:\eng\e85\labs\lab9 directory to your new project lab9_xx. (If the \\igor\courses\ directory is mapped to a different letter than G, of course the path will change accordingly).

Open your Lab 2 project, lab2_xx, and create a macro from the control cell so you can use it in Lab 9. Do the same with your ALU_32 in Lab 5 and the ibox in Lab 8. Then reopen lab9_xx.

Now, use the File?Project Libraries command to add lab2_xx, lab5_xx, and lab8_xx to your lab9_xx. This will let you see the macros from the old labs and use them in your Lab 9.

2. ebox

Open the ebox schematic that came with your lab 9. It is missing the sign extension hardware and ALU. Add your ALU_32 and sgnext macros and wire them to the ebox. Make sure the parts really are connected by moving the parts around and seeing that the wires follow; it is possible for two wires to appear to touch, but not actually connect. A common problem is to wire the SRCA and SRCB busses to the ALU_32 backward; make sure they go to the correct inputs.

Take a look at the rest of the hardware in the ebox and make sure you understand how it relates to the hardware in your book. The register file is rather sneaky. Xilinx lacks 3-ported register files that can read two values and write a third in a single cycle. Therefore, the register is built out of a pair of dual-ported register files. The pair maintains duplicate copies of all the data. Data is written to both register files simultaneously. Independent values can be read from each register file. This trick works at the expense of twice as much hardware, so you can see that an FPGA with a 3-ported register file would have been more efficient for our task.

Testing the ebox is complex because it has so many inputs. Instead of testing it by itself, you will test it later as part of the entire processor. Unfortunately, this means that if you built it incorrectly, you will have to track the problem down from the midst of a rather large design, so double-check by hand that all your connections look right.

When you understand the ebox, create a macro named ebox from the schematic.

3. dbox

The last major module of your processor is the Data box containing the data memory. Open the dbox schematic. Use the LogiBLOX feature to create the data memory. It should be a SYNC_RAM type memory module with a bus width of 32 bits and a depth of 16 words. Name the module dmem. Attach the dmem to the terminals provided in the dbox schematic. Do an integrity test. You should get warnings about the top-level hierarchy and bus hierarchy connectors and that all the address bits except [5:2] are not loadless (because they aren’t used). Create a macro named dbox from the schematic.

4. Putting it all together

Finally, you'll integrate all of your modules to put together your single-cycle MIPS processor. Create a new schematic and save it as it mips.sch. The processor will have inputs `clk` and `reset` and no outputs.

Place your ibox, ebox, dbox, and control macros on the page. Wire them together, connecting clock and reset as appropriate. You will notice that the `jump` and `memread` outputs of the controller are not needed in this design because the ebox does not support j and the memory reads by default when not being written.

Give each wire a name so you will be able to view it during simulation. You will need to connect pieces of busses to each other; for example, the ibox puts out a bus Instruction[31:0], but you only want to connect Instruction[25:0] to the ebox and bits [31:26] and [3:0] of the instruction to the cbox. You can read about busses in more detail by choosing Help?Foundation Help Contents, then looking at the Schematic Editor / Using Busses page. You can also look at examples in the ebox. Convince yourself that your bus connections are correct; errors here would be tricky to track in simulation.

When you do an integrity test, the only warnings you should get are that `clk` and `reset` are hierarchy connectors on the top level schematic and that `jump` and `memread` are not connected. You are ready to simulate your processor.

Ignore warnings that some of the macros, specifically those designed in earlier labs, are not updated. In the simulator, add the following signals. To add the IBOX/PC signal, in the Component Selection dialog, click on the IBOX in the rightmost panel. The left panel will display the wires within the IBOX, where you can select PC. Finally, click on the Root entry in the rightmost panel to go back to the top level of your design where you will find the remaining signals.

```
CLK
RESET

IBOX/PC
INSTRUCTION

EBOX/SRCA
EBOX/SRCB
ALURESULT

ALUCONTROL
ALUSRC
BRANCH
MEMTOREG
MEMWRITE
REGDST
REGWRITE
ZERO
```

To determine if the machine is working properly, you will need to know what values you should expect on each signal. Complete Table 2 (attached on the back) showing key signals during the first 24 cycles. Since most digits are 0, only fill in the first nibble (4 bits) of SRCA, SRCB, and ALUResult and the first byte of the PC.

In Lab 8, you had entered the Fibonacci code into the instruction memory. Now you can simulate the entire single cycle microprocessor executing the Fibonacci program! Assign the B0 stimulus to CLK and the r key to Reset. Initially set Reset high. Simulate for 10 ns. Then set Reset low and simulate for the 10 ns at a time to walk through running the program until a total of 540 ns has elapsed.

Check that the machine operates properly on the first 24 cycles. If you are fortunate it will not work and you will get the opportunity to hone your debugging skills! See the end of the lab manual for hints. Once it does work, check that it computes the Fibonacci number 0000000D at time 490 ns and concludes in an infinite loop at PC=24. If you find that SRCA, SRCB, or ALUResult do not match your expectations, check the control signals to verify that they are operating properly. If the PC is wrong, check that the previous branch operated properly and received the right control signals. Since the ebox is the only major unit that you have not already tested, it is a potential source of bugs.

Turn in simulation waveforms showing your processor working for the first 24 cycles (first 235 ns). You can use the File?Page Setup command to limit the time that is printed. Congratulations! You've designed and debugged your first microprocessor!

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. A printout of Table 2 showing your expected instruction trace. Does your processor operation match your expectations? Does it output 0000000D at time 490 ns? If you can't get the processor to work, explain the likely causes of your problems.
3. A printout of your schematics from:

   ?? EBOX
   ?? DBOX
   ?? MIPS
4. Simulation waveforms of MIPS (235 ns). Make sure all values are readable.

If All Else Fails

If you think you've done everything right and your lab does not simulate properly, check for the following common problems:

?? Make sure that the project manager lists the following libraries as attached to your project (indicated by small cylinders in the Files tab of the project manager). Check that each has the following macros. You should not have any other libraries attached; if you have duplicate macros in different libraries, Xilinx may use the wrong one that you did not want:

?? lab9_xx: dbox, dmem, ebox, regarray, regfile

?? lab2_xx: control

?? lab5_xx: alu_1, alu_4, alu_32, fulladd

?? lab8_xx: adder_32, constant4, flopr_32, ibox, imem, mux2_32, mux2_5, sgnext

?? spartan

?? simprims

If you quit Xilinx, then reopen your lab9_xx and get a message that Automatic Loading of Libraries is disabled or that schematics are not found in the project libraries, you will have to manually let Xilinx know where all your libraries are. Do this by opening lab2_xx, lab5_xx, and lab8_xx, then reopening lab9_xx. Now you can use the File * Project Libraries command to reattach those three labs. If you don't do this, you'll find that macros from those labs are grayed out in lab 9 and you will get messages about missing symbols when you try to simulate. This can lead to warnings about sourceless and loadless signals in the simulator. Do NOT try to solve the problem by manually recreating the macros; you'll probably end up with multiple copies of macros in your libraries confusing Xilinx.

?? Be sure you only have one .sch schematic file in your library files list. Xilinx tries to simulate the union of all the .sch files in the library. If you have multiple schematics with signals having the same names, you'll get errors regarding shorted or "totem pole" outputs and you may discover bus conflicts indicated with x's in the simulator.

?? Watch the project manager window for warning and error messages. It is normal to get warnings about macros not being updated and about hierarchy connectors (e.g. CLK and RESET) at the top level of the schematic. You should also expect warnings that JUMP and MEMREAD are unconnected output pins because you don't use them for anything. If you get other warnings, track down why they happened.

?? You may notice some busses reading very funny results, as if the bits were reversed. This is a strange part of the Foundation tools dealing with busses; it sometimes reads the bits in the wrong order. Click on the misbehaving bus in the Waveform Viewer and choose Signal?Bus?Change Direction. Do NOT rewire the bus in the opposite order in your schematics.

?? A common bug is connecting the ALU backward. Your ALU macro may have the two data inputs shown in either order. Be sure they are connected to the correct sources.

?? This is the lab where you REALLY learn to debug a circuit. Learn to use the simulator to your best advantage to display waveforms. On the right panel is a hierarchical display of all the blocks in your design. If you click on a block, the left panel will display all the internal signals within the block and the center panel will display all the subblocks within the block. If you double-click on a subblock, the right panel will change to display all the interface signals (inputs and outputs) of that subblock.

When you find an incorrect output of a block, add all the inputs of the block to the simulation window and resimulate. Check the inputs. If they are correct, you know the problem is inside the block. If one is incorrect, trace it back to the previous block that generated it and repeat this process. Never assume a block is correct just because it simulated correctly in a previous lab. Add the waveforms at the input and output of the block to the simulator and see for yourself that they do what you expect. Often, the block is correct, but the inputs are hooked up incorrectly. Other times, your block had a bug that you did not catch during earlier testing.

?? If a RAM receives an invalid input such as x or z, Xilinx will spit out many warnings about it. Don't panic. Just display all the inputs and outputs of the RAM cell and find which one is incorrect. Then trace your logic backward to determine why the x or z was produced. For example, one sneaky bug was to accidentally create a RAM with a depth of 32 words rather than 16. This meant it had 5 address inputs, but was only attached to the four address bits ALUResult[5:2]. Displaying ALUResult doesn't reveal the problem because ALUResult is correct. But displaying the ADDR input of the RAM shows that there are five bits instead of the expected four and that one of them is floating (z).

?? If you have a block that looks good but always produces floating outputs (z's), check if the schematic has an input terminal instead of an output terminal on the offending signal. If so, you will have to not only change the terminal in the schematic, but also edit the symbol and change the terminal direction to output in the symbol.

Cycle	Reset	PC	Instruction	SrcA	SrcB	ALUResult
1	1	00	addi $t0, $0, 8 20080008	0	8	8
2	0	00	addi $t0, $0, 8 20080008	0	8	8
3	0	04	addi $t1, $0, -1 2009ffff	0	F	F
4	0	08	addi $t2, $0, 1 200a0001	0	1	1
5	0	0C	beq $t0, $0, done 11000005	8	0	8
6	0	10		F	1	0
7	0					
8	0					
9	0					
10	0					
11	0					
12	0					
13	0					
14	0					
15	0					
16	0					
17	0					
18	0					
19	0					
20	0					
21	0					
22	0					
23	0					
24	0	10	add $t3, $t1, $t2 012a5820	1	1	2

Table 2: Expected Instruction Trace