EEEE-220-Lab8

1) From mycourses download on your desktop this pdf file. This lab consists of part 0 (0 points), and part 2 **or** 3 (ALL points as described at the end of the document).
2) **Objective**: The objective of this lab exercise is to capture and verify the functionality of your fmlRISC data path design. You can use either VHDL or Verilog. From hereon until the end of these labs, you can choose to use either HDLs or a combination thereof.
3) **Part 0:** Familiarize yourself with dxpRISC implemented in VHDL and Verilog.
4) Download from mycourses the project archive files **dxp_DP_vhdl.qar** and **dxp_DP_v.qar**. Open these in two separate folders called **dxp_DP_vhdl** and **dxp_DP_v**, which you can choose to create in your 220 projects folder.
5) Synthesize and testbench both projects. If you haven't decided which language to use for your implementation, this is the time to do it. If you decide to use VHDL, carry on to part 1. Otherwise, if you have decided to use Verilog, continue with part 2.
6) Part 0 carries no points!
7) **Part 1**: fmlRISC – DP – VHDL Capture and Verification.
8) Create a new project and project folder called fml_DP_vhdl.
9) Copy/paste into this folder all components that you have already developed for your data path design. Use explorer or any other file manager application.
10) From the project dxp_DP_vhdl copy/paste in your fml_DP_vhdl project directory any components you need to adapt for your design. You should at the least copy dxp_DP_vhdl.vhd and dxp_DP_vhdl_tb.vhd. Change all file and component names from dxp to fml.
11) There no new VHDL constructs used. The only new component is the tri-state buffer, in which for EN=0 we assign the value high-impedance or z. Remember: z is one fo the values a STD_LOGIC signal can get/have. We will discuss it during the next lecture.
12) Now go to Assignments > Settings > Files, and add all these files to your project, except for the testbench. Any new files you create are added to the project if this option is checked when you first save the file.
13) Update your fml_package.vhd with the declaration of all components.
14) Next, open your fml_DP_vhdl.vhd and create the structure of your data path. As you can see, the I/O-Ps address decoding logic I created in a separate component, while the address arithmetic block I created explicitly within this high level entity. It is an arbitrary choice, i.e. you can create the AAB as a separate block.
15) While you are creating fml_DP_vhdl.vhd, you should "Analyze and Synthesize" often, so that you catch syntax errors early on. Once you don't get errors for fml_DP_vhdl.vhd and any other sub-components, proceed to the testbench.
16) Add now fml_DP_vhdl_tb.vhd to the project. As in the high-level entity, make the necessary changes to all signal names.

17) The testbench has to generate test vectors for all machine cycles necessary to execute the program captured in dxp_rom0_vhdl.mif. Open the later using Notepad++.
18) Modify its content to match your instruction set. I recommend you keep the radices of the address and data values in hexadecimal. It is easier to figure out the jump address for JMPC and JMPU.
19) Run the program manually and determine all relevant values after each machine cycle. Include these in your report. For example:
    a. At the end of an IF the values of IR and PC are relevant and can be predicted. These are the values against which you will assert in the testbench.
    b. Similarly, if register A is loaded during the current machine cycle, its value is relevant.
20) My testbench has 57 test vectors. Yours will have on average 30, because your manipulation instruction cycles are shorter.
21) In my testbench I have not completed the assertion of all relevant values at the end of every machine cycle (test vector). **You have to complete this in yours.**
22) As you can see, the testbench uses a new data type called a record to define the structure of a test vector. This is a composite data type and is then used in an array of constants (like a table) to capture the values of all stimuli and expected outputs.
23) I used named association in the declaration of each vector, because it enhances readability. Although, it is possible to use positional association, this compiler will give an error in this case. Thus, use the former. Also, if the array has only one element, you'll get an error too.
24) **Part 2**: fmlRISC – DP – Verilog Capture and Verification.
25) Create a new project and project folder called fml_DP_v.
26) Copy/paste into this folder all components that you have already developed for your data path design. Use explorer or any other file manager application.
27) From the project dxp_DP_v copy/paste in your fml_DP_vhdl project directory any components you need to adapt for your design. You should at the least copy dxp_DP_v.vhd and dxp_DP_v_tb.vhd. Change all file and component names from dxp to fml.
28) The only new Verilog construct not covered in class yet you'll find used in the tri-state buffer code. It is a ternary operator. We will discuss it during the next lecture.
29) Now go to Assignments > Settings > Files, and add all these files to your project, except for the testbench. Any new files you create are added to the project, if the option is checked, when you first save the file.
30) Next, open your fml_DP_v.v and create the structure of your data path. As you can see, the I/O-Ps address decoding logic I created in a separate component, while the address

arithmetic block I created explicitly within this high level entity. It is an arbitrary choice, i.e. you can create the AAB as a separate block.

31) While you are creating fml_DP_v.v, you should "Analyze and Synthesize" often, so that you catch syntax errors early on. Once you don't get errors for fml_DP_v.v and any other sub-components, proceed to the testbench.
32) Add now fml_DP_v_tb.v to the project. As in the high-level entity, make the necessary changes to all signal names.
33) The testbench has to generate test vectors for all machine cycles necessary to execute the program captured in dxp_rom0_vhdl.mif. Open the later using Notepad++.
34) Modify its content to match your instruction set. I recommend you keep the radices of the address and data values in hexadecimal. It is easier to figure out the jump address for JMPC and JMPU.
35) Run the program manually and determine all relevant values after each machine cycle. Include these in your report. For example:
    a. At the end of an IF the values of IR and PC are relevant and can be predicted. These are the values against which you will assert in the testbench.
    b. Similarly, if register A is loaded during the current machine cycle, its value is relevant.
36) My testbench has 57 test vectors. Yours will have on average 30, because your manipulation instruction cycles are shorter.
37) In my testbench I have not completed the assertion of all relevant values at the end of every machine cycle (test vector). **You have to complete this in yours.**
38) As you can see, when the ***apply_test_vector*** task is called, input values are passed to the task. Some of these are real input values, while others are expected output values. The latter have to match 100%, otherwise the check outcome will be false, and a wrong value message is reported. You'll have to replace the "x", i.e. don't care or unknown values, by the correct, expected values.
39) The due date and time for the completion of lab8 will be offset for ALL lab sections by 7 days, the number of additional days it took me to complete the above designs and testbenches.
40) The due dates and times for all subsequent labs remain unchanged. This means that the due dates of lab8 and lab 9 will overlap!
41) **Grading**:
    a. 20 points for the following ten instruction cycles: ADD, SUB, INC, DEC, NOT, AND, OR, SHR, SWAP, CPY; 2 points per instruction cycle.
    b. 18 points for the remaining six instructions. 3 points per instruction cycle.
    c. 2 points for working I/O-Ps address decoding logic.

42) Show your complete design to the TA, preferably during her/his OH, i.e. before the due date and time. Within the first hour of next week's lab she/he won't have time to check all 16-22 designs.
43) Write your report and upload it along with your archived data path project in the dropbox on mycourses, as described in the lab policy.
44) Even if you don't complete your entire design before the due date and time, you should still upload whatever you have up to that point on mycourses!
45) This concludes this week's lab.