ASIP Meister and ModelSim

Motivation and introduction

In this exercise, you will be introduced to *ASIP Meister*, *ModelSim* and our special directory structure, which makes it easier to combine *ASIP Meister*, *ModelSim*, *dlxsim* and the later introduced tools. First, you have to read Chapter 2.4 from the Laboratory Script to understand the directory structure. Read this chapter extremely carefully, as this one is very important for every task you have to perform during the laboratory. Then you will create your first own *ASIP Meister* project and you will go through the *ASIP Meister* "*User Manual*" and "*Tutorial*" to get used to *ASIP Meister*. Afterwards you will simulate a given Assembly Code with *dlxsim* and with *ModelSim*. Therefore, you will have to read the Chapters 2.3 (for preparing the simulation files for *dlxsim*) and 5 (for using *ModelSim*) of the Laboratory Script. For every part, that starts like "a)", "b)" … you have to mail the answers and asked files with a CC to your group members to **asip00@ira.uka.de** and use the topic "asipXX-Session2", with XX replaced by your group number.

Readings for the next session: Chapters 3, Chapter 8 Introduction

Exercises

1) Preparing the Project

1. After you have read Chapter 2 of the Laboratory Script, you can start creating your own project directory structure in your home directory. Create a copy of the *TEMPLATE_PROJECT* directory from "*/home/asip00/ASIPMeisterProjects*" for your first project. Name your project directory "*dlx_LaboratoryBasis*" and adjust the "*env_settings*" accordingly.
2. Copy the provided *ASIP Meister* file "*dlx_LaboratoryWithoutAdd.pdb*" from "/home/asip00/Session02/" into your project directory and rename it as "*dlx_LaboratoryBasis.pdb*".
3. Create a directory for the example application in your "*Applications*" directory, name this directory "*LoopExample*" and copy the given assembly file from "/home/asip00/Session02/" into this directory.
4. Copy the "*Makefile*" to your application directory from "*Application/TestPrint/*". Now your first project directory is completed and you can start using it during the following exercises.

2) Using ASIP Meister

5. In your project directory, start *ASIP Meister* for the given basis CPU: "`ASIPmeister dlx_LaboratoryBasis.pdb &`". Moreover, do not forget to start *ASIP Meister* in your project directory, because it will create the "*meister*" subdirectory to generate different files where it is started and the "*meister*" subdirectory is expected to be in your current project directory.
6. Read the *ASIP Meister* "*User Manual*" and "*Tutorial*" to get used to the GUI. You can find the related files in the "*/Software/epp/ASIPmeister/*share" directory and in the "*/Software/epp/ASIPmeister/tutorial*" subdirectory respectively. Read systematically through both files simultaneously and play with the specific parts of the GUI for which you are currently reading the user manual and tutorial.
7. If you anyhow configure something wrong, then just reload the original file. The most important parts for the later work are the "*Resource Declaration*", "*Instruction Definition*", "*Behavior*

Description" and "*MicroOp Description*", so have a detailed look at them.

a) What is the difference between the *MicroOp* implementation for *jr* and *jalr*. What is happening in hardware and when is it happening. Why has *link* to be global?

b) What is the difference between the *MicroOp* implementation for *subi* and *subui*. Why this difference is only needed for the immediate instructions?

3) Adding an Instruction

8. The *add* commands (*add*, *addu*, *addi*, *addui*) have been removed from the provided CPU. Go ahead and insert them into the provided CPU. You will have to work with the *Instruction Definition*, the *Behaviour*- and *MicroOp*-Description. Use the following *opcode/func*-values: *add*:000000/00000100000, *addu*:000000/00000100001, *addi*:001000/-, *addui*: 001001/-.
9. If you are unsure about any detail, then have a look at the corresponding subtract "*sub*" commands. **However, if you only copy-and-paste everything without learning anything, then you will pay for this in later sessions** where you will not have a template to copy-and-paste from. Generate the hardware and the software description files for the modified CPU. Make sure, that you do not have any mistakes in the "*add*" instructions, as this will cause problems in the later sessions.

4) Simulating with dlxsim

10. Use the public-key authentication system described as the last point in Chapter 2.2.1 of the Laboratory Script to avoid typing your password multiple times when executing the Makefile.
11. Now you have a CPU that is able to execute the given example code *6_for.s*. This code is similar to the code you created in the last session for exercise 6, but this version also takes care of removing the data-dependencies for the DLX pipeline with *NOP* commands. First, simulate the assembly code with *dlxsim*. Therefore, you should have a copy for the "*Makefile*" in your "*LoopExample*" subdirectory.
12. Then, go to the "*LoopExample*" subdirectory inside your Applications directory and execute "*make sim*".
13. When "*make sim*" is finished, a new subdirectory called "*BUILD_SIM*" containing some important files is created in your current directory, see Figure 2-2. There is a special "*.dlxsim*" file used for dlxsim simulation and there are "*TestData.IM*" and "*TestData.DM*" files used for ModelSim simulation. Simulate "*.dlxsim*" file with dlxsim by typing: "*make dlxsim*" or "*make dlxsim DLXSIM_PARAM="-fBUILD_SIM/LoopExample.dlxsim -da0 –pd4*"

a) How many *NOP's* are executed as compared to the total number of instructions and as compared to the total number of executed cycles (in percentage)?

b) Remove the *NOP* after the "*add %r6, %r6, %r7*" instruction and simulate the program again by typing "*make sim*" and then "*make dlxsim*". Understand the warning, that *dlxsim* will print and have a look to the result array in *dlxsim*. What is the difference as compared to the correct result and why has it been changed in such a way?

c) The real CPU does not have a *NOP* instruction. Into which instruction is a *NOP* translated and what parameters (register and immediate values) does this instruction get as input? Have a look into the "*TestData.IM*" to find out which *opcode* is used to implement a *NOP*.

5) Simulating with *ModelSim*

14. Now we simulate the assembly program together with the real CPU VHDL files. Read Chapter 5 in the Laboratory Script, and create a ModelSim project in your "*ModelSim*"

directory (see Figure 2-3) for your CPU and simulate the program.

15. Compare the created "*TestData.OUT*" with the results from dlxsim. Try to understand the program flow by looking at the "*TestData.IM"* and the *register write* part from *ModelSim waveforms*.

    a) Write a short description what is happening in the first data memory access. Mention all the data bus signals and the final register write back. In the upper part of the *ModelSim waveform*, you can see a clock counter. Use this value to describe when something is happening. Sometimes things happen in the middle of a clock cycle (i.e. falling edge); mention this, too.