Introduction to ASIPMeister

1 Week

Motivation and introduction

In this exercise, you will be introduced to *ASIP Meister* and our special directory structure. The overall workflow of the tools is given at the end; you do not need to deeply look into this. It is just an overview; we will learn gradually about this flow. First, to understand the directory structure, you have to read Chapter 2.4 from the Laboratory Script. Then you will create your first *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*. Therefore, you will have to read the Chapters 2.3 of the Laboratory Script. For every part, that starts like "a)", "b)" … you have to mail the answers and asked files to **sajjad.hussain@kit.edu** and use the topic "asipXX-Session2", with XX replaced by your group number.

Exercises

1. Preparing the Lab tools environment

1.1. ASIPmeister is only installed on i80pc57. It is preferred that you always SSH to this PC.

1.2. Use the public-key authentication system discussed in Chapter 2.2.1 of the Laboratory Script to avoid typing your password each time when logging into frequently.

1.3. To start, ASIPmeister, ModelSim & Xilinx ISE during the lab, you need to export the following variables each time, or you can add it in your "*/home/.bashrc.user*" to load automatically when you start shell terminal.

```
export ASIPS_LICENSE=29000@i80asip.ira.uka.de
export PATH=/AM/ASIPmeister/bin:$PATH
export ASIP_APDEV_SRCROOT=/home/asip00/epp/AM_tools ***
export PATH=/usr/java/jre1.6.0_45/bin:$PATH
export ASIPmeister_Home=/AM/ASIPmeister
export ASIPmeister_HOME=/AM/ASIPmeister
source /home/adm/modelsim_66d.setup
source /home/adm/xilinx_13.2_32bit.setup
```

1.4. Copy "*AM_tools*" from "*asip00/epp*" directory to your account, for example at your home folder. Export different environmental variables for ASIPmeister and ModelSim or put them in the ***bashrc.user*** and set "*AM_tools*" path from your home folder. It needs write permissions.

*** If the ASIPmeister gives error "could not the old work directory for binutils". Copy "AM_tools" to your home directory and set the path accordingly.

2. Preparing your project

2.1. Create a project directory for this session by copying the directory "*/home/asip00/epp/ASIP-MeisterProjects/TEMPLATE_PROJECT/*" and renaming it (e.g. *brownie*).

2.2. For each application (C or Assembly), you have to create a separate subdirectory in the "*Application*" directory (e.g. *LoopExample*).

NOTE: Never name the subdirectory same as the name of the application that you want to compile using "*Makefile*". This will result in problems for the Makefile script execution.

2.3. Copy the given assembly file from "/home/asip00/Sessions/Sessions/Session1/" into this application subdirectory i.e. *LoopExample*.
2.4. Copy a "*Makefile*" file from the "*TestPrint*" application subdirectory to each application subdirectory. This Makefile has been prepared to help you in performing different tasks during the Lab as discussed in Figure 2-3 in the Laboratory Script.
2.5. In an application directory, typing, "*make help*" on shell terminal will show usage of the "*Makefile*" and how different parameters can be passed.
2.6. Copy the provided *ASIPMeister* CPU file "*browstd32.pdb*" from "/home/asip00//Sessions/Session1/" into your project directory.
2.7. Set proper parameters and settings in "*env_settings*" as discussed in Figure 2-5 in the Laboratory Script. Specially the followings:

```
export PROJECT_NAME=brownie
export CPU_NAME=browstd32
export ASIPMEISTER_PROJECTS_DIR=${HOME}/ASIPMeisterProjects
export DLXSIM_DIR=/home/asip00/epp/dlxsimbr_Laboratory
```

2.8. Using above steps, for each session, you need to create a separate project if you have modified the CPU or you can have separate application subdirectories for the same CPU.

3. Using ASIP Meister

3.1. In your project directory, start *ASIPMeister* for the given basis CPU: "`ASIPmeister browstd32.pdb &`". Moreover, do not forget to start *ASIP Meister* in your project directory, because it will create the "*meister*" subdirectory to generate different VHDL files and GNU tools, where it is started. The "*meister*" subdirectory is expected to be in your current project directory.
3.2. Read the *ASIP Meister* "*User Manual*" and "*Tutorial*" to get used to the GUI. You can find the related files in the "*/home/asip00/Documents*" directory. 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.
3.3. 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*", "*MicroOp Description*", "*HDL Generation*", "*C Definition*" and "*Compiler Generation*" so have a detailed look at them.
3.4. Go through all the steps one-by-one as mentioned in the tutorial and generate VHDL files and GNU Tools. VHDL files will later be used in ModelSim for hardware simulation, while GNU tools will be used to compile, assemble, and link your application code. In this step, recommended settings are "*VHDL*" and "*sim. model and syn. model*".
3.5. Make sure that AM_tools path is set properly in your "bashrc.user" and is permissible.
3.6. Make sure that you complete both/two steps i.e. "*Input Description Generation*" and "*GNU Tools Generation*".
3.7. GNU Tool generation may take 10-15minutes. After GNU Tool generation, you will see compiler, assemble and linker according to your instruction sets. See the directory in ${ASIPMEISTER_PROJECTS_DIR}/${PROJECT_NAME}/meister/${CPU_NAME}. swgen/bin.

a) How many pipeline-stages this CPU has? Normally, CPU has fetch, decode, execute, memory and write-back stages, how these stages are mapped into brownie 4 stage CPU? For CPU related details look at the Brownie32-Std datasheet at /home/asip00/epp/Documents.

Resource Declarations:

b) See different hardware resources used in the CPU. Select ALU in the "Instance" list. What do you understand from the contents of "Function Set" tab? What is listed there?

c) How many does read/write ports GPR has?

d) CPU uses full forwarding in pipeline. How many forwarding units are used? How many intermediate register values can we forward now?

e) What should we need to change in GPR and Forwarding Units if we need an instruction, which writes two operands like quotient and remainder (DIV rd1, rd2, rs1, rs2) to be returned?

Storage Spec:

f) What does GPR0-7 are used for?

Instruction Definition:

g) Why does this CPU have **SP** instruction format? What instructions are covered by this **SP** format? Can we map these instructions with some other format and remove SP format from the CPU?

Micro Op. Description:

h) What does ForwardDataFromWB() and ForwardDataFromEXE() are doing? Which hardware resources are being used here?

i) What does GPRRead(src1) macro perform? Why FWO.forward() is used here?

j) What does "*alu_flag*" mean in ALUExec() macro? What does individual bits mean?

VHDL Generation:

k) What does different bits in "*alu_flag*" stand for? You can take help from the generated VHDL for understanding "*alu_flag*" in fhm_alu_w32.vhd.

4. Simulating with dlxsim

4.1. Now you have a CPU that is able to execute the given example code *6_for.s*. This code has implemented the following part:

```
for (i=0; i<10; i++) {
  A[i] = B[i] + 5 + C;
}
```

4.2. In this session, we only simulate the assembly code with *dlxsim*. You should have a copy for the "*Makefile*" in your "*LoopExample*" subdirectory.

4.3. Then, go to the "*LoopExample*" subdirectory inside your Applications directory and execute "*make sim*".

4.4. When "*make sim*" is finished, a new subdirectory called "*BUILD_SIM*" containing some important files is created in your current directory. There is a special "***.dlxsim***" file used for dlxsim simulation and there are "*TestData.IM*" and "*TestData.DM*" files used for ModelSim simulation. *TestData.IM* and *TestData.DM* are instruction and data memory image files respectively.

4.5. Simulate "*.dlxsim*" file with dlxsim by typing: "*make dlxsim*" with default settings or "*make dlxsim DLXSIM_PARAM="-fBUILD_SIM/LoopExample.dlxsim -da0 –pf1*". The parameter pf1 indicates the full forwarding in CPU pipeline, for details see brownie32-std datasheet.

4.6. Inside dlxsim terminal, you can use "go" to execute whole program, "step" to execute one instruction at a time, and "stats" to see the statistics.

4.7. The command “make sim” adds some start-up and ending code to your program to generate a “.dlxsim” file which is then executed by “make dlxsim”. You can directly run your assembly program only using dlxsim like DLXSIM_PATH/dlxsim –fAssembly.s –da0 –pf1

a) What is meant by the following lines and values in the dlxsim:

Biggest used address for Text Section (word aligned): 0xdc
Biggest used address for Data Section (word aligned): 0x130

b) In Dlxsim, you can use "*get start_address #of_of_instructions*" to see the 32-bit binary values of each instructions and from this, you can also extract opcodes. As “get _A 10d” will show 10 values for array A in decimal. What does “get 0 10” gives you? What are these values?

c) Can you see the data _A, _B and _C variables in TestData.DM? What does the first 32-bit word indicate? This large value is a stack pointer value, used while you call many subroutines one by one. Why this value is so large?

d) How many cycles are required to execute this program?

Next Session: Assembly Programs
Readings for the next session: Laboratory Chapters 8.1, 8.2, 8.3
BrownieSTD32 Datasheet Introduction and Section 1.1 - 1.3
2.2.3. Instruction Set Quick Reference
4. Memory Access
Appendix A. Delayed Load

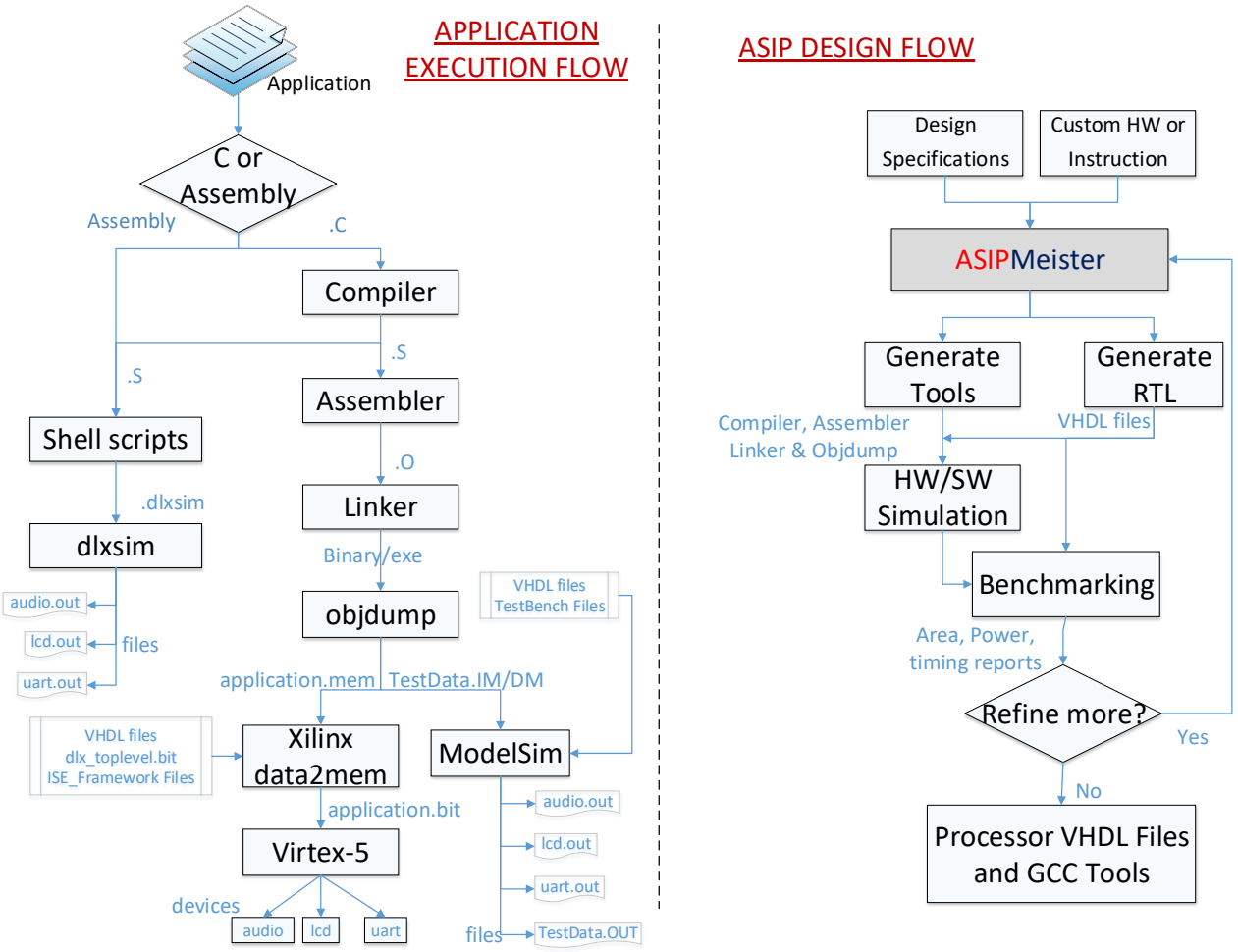

Figure 1: An overview of our Lab Tools