Instruction Architectures and Compilers Lab : Autumn

You already have some experience with digital design for FPGAs through DECA lab, where the main focus was on block design with bits of Verilog. Those of you who chose the CPU project in 1st year will have more experience, but there should be no problems for those who did other projects.

The main differences in this course are related to:

- Complexity: developing a full MIPS is inherently a more complex task than doing a small custom ISA.
- Specifications: you must adhere to a reasonably complex and precise ISA spec, rather than making your own choices according to convenience.
- Language: we are using pure Verilog with no block diagrams or vendor primitives.
- Testing: you need not only develop the CPU, but also deliver a test framework.

This lab has a number of intended outcomes, some directly related to the module, and some related to your wider development as an engineer:

- Learning Verilog : These labs are intended to get you up to speed on Verilog in specific, and creating synthesisable digital logic in general. The approach taken is more to show you lots of examples of Verilog, rather than to ask you to write Verilog from scratch, and to learn by example and investigation.

- Understanding computer architecture : the labs should reinforce some of the principles of the lectures, such as computer arithmetic and area-time tradeoffs.

- Working with tools : In your academic and eventual career you are likely to need to use lots of different software tools, and also will have to work out how to mix and match them through scripting.

- Testing : Functional correctness is a key part of any engineering deliverable, and one of the most basic ways of ensuring correctness is through testing - often testing takes up as much engineering time as developing. Here we look at testing of digital circuits, but the idea is also to develop a more general intuition for testing, whether it is hardware, software, or something else.

- Team work : Most engineering is done in teams, which requires the ability to actually organise and work within groups. In this lab you can develop organisational principles within your team that you can take through to the CPU coursework.

- Self-guided learning : while the labs have structure and suggested exercises, it is also up to each group and each student to decide exactly how to approach things.

There are many suggested activities which are worth trying, either individually or by sharing them out in the group. It is highly recommended that you talk to each other occasionally while doing the lab, and share experience. There are also a number of suggested "thinking" points. You should genuinely try to think about them, and potentially discuss them as a group. There is not necessarily one answer, or it may be subjective, but they are worth thinking about.

There is no assessment of what you *did* in these labs, the only thing that matters is what you can do after learning the topics in lab. The mid-term assessment requires you to perform similar tasks to those in the lab, and are not intended to be any harder. The main difference is that the mid-term assessment is time-limited to try to assess what you have already learnt *before* the assessment, just like a lab oral is looking back on what you already learnt.

Structure

The lab is organised into six sections:

- 0-tools This describes how to get hold of the Verilog simulator we'll use, and some basic simulation tasks.
- 1-verilog-syntax This is a short intro to the parts of SystemVerilog needed to build the CPU. This section is mainly reading.
- 2-verilog-errors Provides 7 very simple examples of different types of error, so that you can see what happens, fix them, and think about how to stop them happening in your code.
- 3-functional-units Provides slightly more complex working examples of functional units, described using a number of styles and architectures. These are mainly for you to run, read, and understand.
- 4-verilog-MU0 An example of a working synthesisable MU0 in Verilog, including test-benches and testing infrastructure. The purpose here is to give an example of a complete system, and to show all the other stuff that is needed beyond just the Verilog for the CPU.
- 5-assessment Describes how the oral assessment will work.

Approach

In total there are ~9000 words in the instructions, which is about 30 minutes for most readers. So it is suggested that you individually read the whole thing through once first, then come back and do the more technical aspects.

A suggested path is to do earlier sections individually, then come together as a group for the later sections, but it is up to you.

- Performed Individually : if you get stuck on a part, then just leave it and then talk about it in your group
  - 0-tools : 30-60 minutes (depending on internet connection)
  - 1-verilog-syntax : 30 minutes of reading
  - 2-verilog-errors : 30-60 minutes (depending on how well you understood it). This is mainly just to fix the errors. The thinking can be done more in the group.
- Performed as a group or pairs : it's worth reading the later sections first, but you'll likely get more value from discussion
  - 3-functional units : 30 minutes of checking everyone understands, then 30-60 minutes of breaking and fixing
  - 4-verilog-MU0 : worth discussing as a group once to try to work out how it works, the possibly try modifying it - for example, to add a new instruction, or extend the test-bench.

Remember that each individual doesn't have to do everything or know everything. You are trying develop knowledge and learn skills so that you can deliver a MIPS CPU as a group.

Overall about 4-6 hours of individual and 4-6 hours of group work should be about right, including time spent during the mid-term week as a group preparing for the mid-term. So you could organise it as:

- 11th-22nd : Individual reading and practical work (4-6 hours)
- 25th-29th : Group discussions and learning (2-3 hours)
- 1st-6th : Group revision and mid-term preparation (2-3 hours)

Obviously these are just estimates - some people will be faster, some slower (either due to ability, or decisions about how to manage time.

On errors

I (dt10) freely admit that I had never really used Verilog before June 2020 (though I have written many thousands of lines of VHDL), and learnt the language purely to develop this coursework. Based on my research, reading, and experiments, the information and approaches presented here seem to present the best balance between writing nice code and writing correct code. Some "best practises" are inherently subjective, so there will often be differences between what I suggest and what others suggest (and even between what I suggest and actually do in the example code). However, if there is anything which is factually wrong, then let me know - I'm perfectly happy with being corrected.