

*Department of Electrical and Computer Engineering  
University of Wisconsin - Madison*

## **ECE 554 Digital Engineering Laboratory**

# **THE PROJECT**

Version CRKs02-MHLf02

You are about to embark on what we hope will be one of the most interesting and challenging adventures in your academic program. A good deal of your time so far and quite a bit of it to come will be spent on building your background to face this challenge. Other than this, your assignment for the rest of the course is:

*Design a non-trivial computer with an original instruction set.*

As an aside, you might note that, for most of you, this is the first and probably the last time that you design and implement a computer including instruction set architecture (ISA). In fact, the proportion of computer architects designing real commercial computer systems at the ISA and other higher architectural levels is relatively small. Far more engineers work on lower level architecture and design tasks or the design of a variety of specialized digital systems. The reason that we do a computer design is that the variety of concepts involved and the techniques used span a range wider than that for many other types of digital systems. As a consequence, computer design serves you well as an educational foundation for design of a wide spectrum of systems at a variety of levels. Nevertheless, all of this belies the fact that you are not too likely to ever design a complete computer from the top down again. So we hope you will relish this unique opportunity and learn much from the experience.

You have an enormous amount of flexibility in terms of what to build. There are really only four formal requirements:

- It must be an *original* ISA. While it can certainly borrow from existing ISAs, the complete machine cannot be a direct copy of an existing product.
- It must be *non-trivial*. A ridiculously simple textbook machine from a logic design book is not acceptable.
- It must be *tractable*. You will have to complete it in the remaining part of the semester as a team project. You must avoid complex machines, as they will be impossible to finish in the allotted time. From other courses, some of you may have an idea of how long it takes to do a design, do thorough simulation, and debug the hardware and

software. If you don't, a good rule of thumb is that "everything takes at least twice as long as you expect."

- It must interface through the serial port with a terminal emulator (HyperTerminal) on the lab workstations (maybe).

The usual ECE 554 machine design is an 8–32 bit general-purpose computer having:

1. A simple core instruction set with the usual MOV, LD, ST, ADD, etc. opcodes and a small but adequate set of addressing modes.
2. A "gimmick" that makes the machine non-trivial and unusual.

Typical and not so typical gimmicks are:

- Multithreading
- Superscalar
- Pipeline with dynamic branch prediction
- Floating-point Add or Multiply (small numbers)
- Hardware multiply and divide
- Parallel processors
- Rich addressing modes
- Stack machine or stack cache
- Instruction and/or data cache
- Non-blocking cache (hit under miss)

Or, you could create a special-purpose processor:

- High-performance DSP (Digital Signal Processor)
- String processor
- BCD machine
- JAVA machine

Finally, the availability of various I/O interfaces on the XSV FPGA boards opens up additional possibilities:

- VGA display for fancy graphics
- USB interface to a USB peripheral (disk, mouse, etc.)
- Ethernet interface to network (implement simple web server or telnet/ftp)
- Audio codecs - perhaps play MIDI files? Or build an electronic instrument (e.g. vary pitch/volume/timbre based on USB mouse movements).
- Graphic equalizer display that performs FFT on audio input and displays frequency/amplitude bars on VGA display or onboard LED array.
- etc.

We will try to keep you from being too ambitious. We want you to be able to finish your machine.

You will be performing the following steps, more or less in this order:

1. **Form Teams.** An ideal team for this project has from five to eight members. This may or may not fit the populations of the individual lab sections. We may be looking for volunteers to switch sections to balance the teams.

Try to balance levels of experience and graduate/undergraduate status. Make sure you have at least one member who is a very good with VHDL or Verilog, one member who has had an operating systems course, one member who has a compilers course, and one who is a strong programmer.

2. **Choose a Team Leader.** This is **very important** — teams with a capable leader usually do better than teams with little or no leadership. The leader should know the ISA and architecture inside-out, but does not need to know the details of all subsystems, other than those in which the leader has direct design involvement. The leader coordinates communication between the team members of the interfaces between subsystems including their signal timing. The leader is responsible for scheduling and providing coordination of the team activities. The leader is not a manager who exercises control over the team and its members, but does have a role in resolving team conflicts and in insuring equitable participation of team members.

It is usually better to have a volunteer for team leader rather than to draft someone. It is also better to have only one leader per team. **Please let the instructors know if you are having problems selecting a leader, because if your team does not have a designated leader, you all lose one-half letter grade.**

3. **Define the Architecture.** Companies spend years defining an architecture if a new ISA is involved. Here you are to determine the ISA plus the "gimmick(s)" which are special architectural features. You have a few lab periods and the substantial time you are likely to find it necessary to meet off-line. Find a room with a white board or blackboard. 3444 Engineering or the lab is usually available. Spend the first hour or so *brainstorming* — sit in a circle and toss out ideas. Each person speaks in turn. If you don't have an idea, say "pass" and move on to the next person. Have one person who can write quickly and clearly write them down on the blackboard or use other techniques indicated in the team lecture. Do not stop to discuss any idea — get them out and on the table. Typical brainstorming results:

- "RISC!"
- "Microprogrammed! No, I hate microprogramming! It's too slow! No comments! This is a brainstorming session!"
- "Stack machine!"
- "Tomasulo algorithm!"
- "DSP machine!"
- etc.

After an hour of brainstorming, or when the ideas have died out, stop brainstorming and consider the possibility of a multivote on the ideas you have accumulated. Argue about what you would like to implement. Cross out (do not erase yet) ideas as they lose in the multivote. At this point, the basic ISA should be shaping up quite well.

At the second major architecture meeting, firm up the architecture. Define the visible registers in detail. Define the instruction set in detail, including instruction format(s). Define opcodes, addresses modes, and data types. Define the "gimmick". Make sure everything fits together. Sketch out the basic organization and data paths, though details are premature at this point. Plan the interrupts, if any, and include the I/O subsystem. If any part is of questionable complexity, sketch out how to implement it so you have an idea as to whether it is tractable or not.

4. **Architecture Review.** Your team will present your architecture to the guidance and evaluation (G&E) team (the professor and the instructors). Other teams are not invited, so you can present proprietary information. This presentation should cover the architectural registers, instruction format(s), addressing modes, opcodes, data types, "gimmicks", and the basic data flow. This presentation should be 45 minutes, which including the discussion for the G&E team will stretch to one and one-half to a bit over two hours. The purpose of the review is to ensure that you are on the right

track, i.e., the architecture meets the project requirements, and that you have not “bitten off more than you can chew.” Please consider the G&E team comments as constructive advice, though many will sound like pretty vicious criticism.

This presentation should be lead by the team leader and involve other team members as appropriate. **All team members are expected to be present for a review!**

5. **ISA Report.** A written draft of your architecture’s *Principles of Operation*, a manual describing the ISA for an assembly language programmer is to be submitted to the G&E team one week after Architecture Review. The ISA report should contain a description of the following:

- A description of all visible registers and memory,
- A description of the instruction formats,
- A verbal description of each instruction, including side-effects such a setting status bits, if any,
- An HDL-like description of each instruction,
- a discussion of the I/O (and interrupt) interface,
- a description of any “gimmick” at the top level such as specialized I/O, and
- anything else such as simple code examples or a memory map that you believe will clarify understanding of your computer.

6. **Define the Microarchitecture.** After the architectural review, you will define the detailed architecture exclusive of the ISA which is complete. Rather than use the clumsy term “Detailed Architecture,” we will use the term “Microarchitecture.” This involves breaking the machine up into a small number of *subsystems*, each of which will be designed by a single team member. Define the subsystem interfaces extremely carefully. If you do a good job, each member will be able to design his or her subsystem(s) separately with minimum interaction needed with other members. If you do a poor job, a great deal of logic will have to be redesigned, wasting a great deal of valuable time that you do not have. *A mistake made now results in an all-nighter near the end of the semester.*

Make sure the function of each subsystem is specified very carefully. Make sure the interfaces, including signal names, directions, polarities, and timing are absolutely clear and *written down*. Consider carefully the partitioning, realizing that the best partition may not necessarily follow subsystem boundaries. Estimate the FPGA utilization to be sure that the design will fit into the available hardware resources. If you are not sure how complex a subsystem will be, you may want to sketch out its detailed logic design. Remember, redesign is *very* expensive.

7. **Microarchitecture Review.** You present your microarchitecture to the G&E team. The team leader presents the overall microarchitecture, and each team member will present his or her subsystem(s). In addition, describe any changes to the ISA or “gimmicks” made since the architecture review. This is similar in structure to the architecture review.
8. **Progress Reviews.** There are several Progress Reviews listed on the course outline. As their name suggests, these are reviews to assess your team progress toward the project goal. These reviews will be conducted as 30-minute long meetings between your team and at least two members of the G&E team. At these meetings, your team is expected to report on your project progress. If you get behind schedule, you will be expected to present realistic proposals for getting back on track. The progress meetings also provide a forum for you to ask questions of the G&E of an administrative or technical nature. Please note that in addition to these meetings, however, it is important that you keep your section instructor well aware of major technical decisions so that these can also be evaluated or discussed as necessary by the G&E team.

At an early progress review, we will ask you for a detailed project schedule and will then use your schedule to gauge your progress. Microsoft Project on the workstations in the lab can be used for generating a schedule. You can also use other methods such as a spreadsheet. The schedule is somewhat dynamic as tasks are allocated to specific team members, etc. Detailed allocation should ordinarily be done at least a week before a task begins. The schedule when fully developed should include:

- A daily time grid on the horizontal axis with remaining items on the vertical axis
- Major project tasks,
- Project subtasks,
- Starting and ending times for tasks and subtasks,
- The team member or members assigned to tasks/subtasks,
- Indication of level of completion of tasks/subtasks

The schedule should be updated on a regular basis and available for all progress meetings.

9. **Logic Design and Simulation.** After the microarchitecture review, start designing logic! Each member should design his or her subsystems. It is a good idea to have two members very familiar with every subsystem. Each subsystem should be simulated extensively to ensure that it works and run through synthesis as a separate non-downloadable FPGA part.

10. **System Integration.** Once all the subsystems have been simulated, bring them together into a single model and synthesize, implement and timing simulate it. This should be a very exciting moment. If you have done a careful job of defining interfaces, the entire computer simulation should work! If it doesn't, find out why! It is always faster to find bugs using simulation than to build it, since it is so much easier to observe many nodes.

*A special word of caution:* There is a major strategic question to be answered here; how much time should be spent on system level simulation? Due to the limited time available, if you simulate too much you may have no time to do the hardware; but, if your hardware fails, the simulation may be the only thing you have to show for your efforts! This issue is particularly important to smaller teams; larger teams should have no problem doing both. Keep this in mind as the last couple of weeks loom large!

11. **Software Development.** Once the ISA and architecture has been completed, you can begin software development of diagnostic and demonstration software for the final machine.
12. **Test and Debug.** Since you simulated the logic to death, and have diagnostics and demo software working by now, your design is quite likely to work at this point. Hmm. Of course it won't, so you will need to debug it! You have failed to get good "first silicon"! The most likely causes of failure at this point are:
  - (a) **Architecture, subsystem or logic errors** – These just happened to slip through your perfect procedures. In particular, check for bad logic design practices and timing problems. We **told** you not to gate clocks and to avoid asynchronous operation except for power up and pushbutton clear!
  - (b) **Compile problems** – Make sure the Xilinx tools are not deleting circuitry or giving incompletely routed designs by examining reports.
  - (c) **Download problems** – Scope the power supplies on the board: are VCC and GND really the values expected all the time, or are they noisy? Check the clocks — Are they present clean or are they ringing? Download a simple design and test it to see if download is working.
  - (d) **Bad chips** – We hope not; all the chips were perfect when we installed them. You have to understand that 95% of the problems in this course aren't due to bad chips although this seems to be popular assumption. If you really think a chip is bad, try to isolate the problem and solve it if possible by changing your design and reprogramming. And do report the problem to us. We don't like false alarms; a \$1600 loss is painful!

13. **You Did It!** Congratulations! You have accomplished a very difficult project and can be very satisfied with your *team*! This should build your confidence at being able to accomplish a major task as a member of a team. If you, however, didn't do it, you should still be satisfied that you probably came very close (needed one or two more days) and learned a lot along the way including some of the pitfalls to avoid on your first real team project.
14. **Demonstration.** The demonstration should be planned to occupy about 1.5 hours duration including time for questions and on-line evaluation of the system. You should prepare an overview of the architecture and microarchitecture that can be presented in about 20 minutes. You should have some interesting software running on your system. Some software should thoroughly exercise the instruction set. Other software should add some flair to your demonstration (character graphics, interactive, etc.). This system demonstration portion should be targeted to last about 25 minutes. Any presentation on the software itself should be at a very high level, not detail. We will be asking you a number of questions during the demonstration. In particular, we will be interested in the empirically evaluated maximum clock rate for your machine and a discussion of performance enhancing features of the microarchitecture and design. The latter features are more important to us than raw speed.
15. **Final Report.** You are to prepare a Project Final Report including the following:
  - a) Overview: Summarizes the architecture, microarchitecture, implementation and software for the project.
  - b) Principles of Operation of the final machine: This describes the ISA from an assembly/machine language programmer's point of view. It should include the architectural registers, the complete instruction set, memory and I/O description, and interrupts. Instructions should be defined with register transfers and all status setting information specified.
  - c) Machine organization: block diagram(s), definitions of control, and pipeline stages (if appropriate); a register transfer language description would be helpful but is not required. Include necessary explanations for easy understanding.
  - d) Detailed design for each subsystem: ALU, control, memory interface, etc. Include schematics, ASM charts, VHDL, tables, etc. Include timing diagrams if helpful to describe behavior. Again include explanations necessary for easy understanding.
  - e) Simulations: Provide selected subsystem simulation results, carefully annotated for clarity and thorough full system simulation results.
  - f) Implementation: Provide a copy of all implementation reports for the final implementation run; print 2 pages/page if possible.

- g) Software: Describe software at a functional level; give listings of source code and explanation as necessary.
- f) Report on the individual contributions of each team member to the project. Each individual should have about a one-quarter page verbal description. The collective descriptions are to be agreed upon by consensus by the team, yielding a consistent correct representation. In addition, a table, representing the team consensus is to be provided listing 1) all tasks and major subtasks as rows and 2) the team members as columns and giving a percent contribution to each task/subtask by each team member. All rows must total to 100 percent.