Course Syllabus

ENEE 646: Digital Computer Design, Fall 2020
Prof. Bruce Jacob

Basic Information

Time & Place

Lecture: TuTh 2:00–3:15 pm, IRB-1116

Professor

Bruce L. Jacob: AVW-1333, blj@umd.edu
Office hours: open-door policy

Teaching Assistant

Xiaomin Wu, xiaomwu@terpmail.umd.edu

Class Home Page

http://www.ece.umd.edu/courses/enee646

Class Schedule

This is a weekly schedule of my hours, including class time and scheduled office hours, but also including other things that make me unavailable. It is subject to change.

	MON	TUE	WED	THU	FRI
9–9:30					
9:30–10					
10–10:30					
10:30–11					
11–1:30					
11:30–12					
12–12:30					
12:30–1					
1–1:30					
1:30–2					Weekly meetings with graduate students
2–2:30					
2:30–3		**ENEE 646** Lecture IRB-1116		**ENEE 646** Lecture IRB-1116	
3–3:30					
3:30–4					
4–4:30					
4:30–5					

Course Overview

This course covers the architecture and design of microprocessors, memory hierarchies, and system-level software. It is intended to give you a solid understanding of how digital computers are implemented today. We will cover concepts such as pipelines, caching, superscalar execution, out-of-order execution, very-long-instruction-word architectures (VLIW), precise interrupts, and low-level operating system mechanisms.

You will learn these concepts by not only reading about them but by building them; you will partially design and "build" simple computers at various levels of sophistication. You will build a simple machine, then a pipelined machine, and then you will have the opportunity to improve on a design reflecting current thinking in high-performance microprocessors. The term "building" in this course will mean implementing your design using the Verilog *hardware description language (HDL)*.

Building in Verilog (or in any HDL, for that matter) is interesting for several reasons. First, you must be infinitely more precise in your design than if you built the model in a high-level language such as C. This is good because it forces you to understand all the finer points of your design and the ramifications of your choices—if you are not thorough, it will not work. Second, the performance/power/die-area results that you obtain are infinitely more believable than had you built the model in a high-level language such as C. It is good to learn this because, historically, lots of money has been spent on ideas that looked good on paper but whose implementations fell far short of projected measurements. Lastly, an HDL implementation is just steps away from actual silicon.

Prerequisites

Students must have knowledge of digital logic design and computer organization. You should understand digital design concepts such as multiplexers, gates, boolean algebra, finite-state machines, and flip-flops. You should understand fundamental computer organization: what the program counter is, what a register file is for, how busses are used, what happens in the hardware to effect instruction execution, etc. It would help if you also understand and are reasonably fluent in programming C or Perl, because Verilog is very C-like (as is Perl).

Course Material

The required text for the course:

> *Memory Systems: Cache, DRAM, Disk*, by Jacob Ng, & Wang.

Due to the tremendous advancements in processor design over the past 3–4 decades, the processor is no longer the system bottleneck that it perhaps once was. It has not, in fact, been a significant bottleneck for most applications for over twenty years; rather, the memory system has. *Memory Systems* is one of the few texts that treat the topic in significant detail. It also has more than enough information on processor pipeline design to serve as a general, introductory architecture text. 0A highly recommended complementary text:

> *Computer Architecture: A Quantitative Approach, 4th Ed.*, by Hennessy and Patterson.

This is a widely used textbook that describes in reasonable detail most of today's thinking in high-performance architecture. It is a good, solid book and is valuable as a reference for later on.

Because it is often worthwhile to go to the horse's mouth for information, when discussing some high-performance designs, we will go to the original descriptions of those designs, written by the designers. For example, I will hand out copies of papers describing Tomasulo's hardware algorithm for out-of-order execution, Smith and Pleszkun's reorder buffer for precise interrupts, and Sohi & Vajapeyam's register update unit for providing both out-of-order execution and precise interrupts.

For learning Verilog, there is a brief handout on the course website, and I recommend the following very, very highly:

> *Verilog Styles for Synthesis of Digital Systems*, by Smith & Franzon.

Class Projects

Four projects will be assigned during the term, each of which will require a substantial time commitment on your part. You will find the work load in this course to be extremely heavy.

- Project 1: Verilog implementation of a simple VLIW CPU core
- Project 2: Reslistic Verilog implementation of a pipelined VLIW CPU core
- Project 3: Dealing with the realities of the memory system: Quantification and mitigation/optimization [group project]
- Project 4: Multicore cache coherence [group project]

The most common reason for not doing well on projects is not starting them early enough. You are given plenty of time to complete each project. However, if you wait until the last minute to start, you **will not** be able to finish. Plan to do some work on a project every day. Also plan to have it finished at least 1 week ahead of the due date—many unexpected problems arise during the debugging phase. The computing sites can become quite crowded as deadlines approach, making it difficult to get a computer. **Plan** for these things to happen. Your lack of starting early is **not** an excuse for turning in your project late, even if unfortunate situations arise such as computer crashes.

There are many sources of help on which you can draw. Simple questions can be submitted to the professor and fellow classmates via email (**use the email list given on page 1**). These will typically be answered within the day, often more quickly during working hours. Keep in mind, however, that many types of questions cannot be answered without seeing your project. If you have detailed questions, your best option is to speak to the professor in person during office hours. Bring along a listing of your project and the output from a run if available. Students are also encouraged to help one another. *One of the best ways for you to make sure that you understand a concept is to explain it to someone else.* Keep in mind, however, that you should not expect anyone else to do any part of your project for you. The project that you turn in must be your own.

Projects are due at 5:00 pm on the due date. We will allow a grace period and accept projects until 11:59 pm. Sometimes unexpected events make it difficult to get a project in on time. For this reason, each person will have a total of 3 free late days to be used for projects throughout the semester. **These late days should only be used to deal with unexpected problems such as computer crashes, illness, or submission problems.** They should not be used simply to start later on a project or because you are having difficulty completing the project. Projects received after the due date (assuming that you have no late days left) will receive a zero, even if it is just one second late. I advise you to save at least one or two late days for the last project. Weekend days are counted in exactly the same way as weekdays (e.g. if the project deadline is Friday and you turn it in Sunday, that's two days late). You will be submitting your projects electronically via email attachment.

The projects will be graded primarily for correctness: doing all the required tasks, simulating at the correct hardware level, and giving correct results.

Exams

You are expected to take both the midterm and final exams at the scheduled times. Unless a (documented) medical or personal emergency is involved in your missing an exam, you will receive a zero for that exam. If you anticipate conflicts with the exam time, you must come talk to the instructor about it at least **1 month** before the exam date. The exam dates are given at the beginning of the term so that you can avoid scheduling job interviews or other commitments on exam days. Outside commitments are not considered a valid reason for missing an exam. Exams will be closed book, closed notes.

Grading Policy

Final grades will be based on the total of points earned on the projects and exams. The tentative point breakdown is as follows:

- Projects: 50%
- Midterm Exam: 25%
- Final Exam: 25%

Incompletes will generally not be given. According to university policy, doing poorly in a course is not a valid reason for an incomplete. If you are having problems in the course, your best bet is to come talk to the instructor as soon as you are aware of it.

Tentative Lecture Schedule

Week of	Subject	Readings	Projects
Aug 31	Intro: overview of course	Overview	P1 out
Sep 7	Intro continued: overview of topics and the Verilog language	Ch. 27	
Sep 14	What is important: performance/power/cost	Ch. 28	
Sep 21	Processor implementation: pipelines	Ch. 31.1	P1 due, P2 out
Sep 28	Processor implementation: branch prediction & performance	Ch. 31.1	
Oct 5	Instruction-level parallelism: origins & today	Handouts	
Oct 12	Instruction-level parallelism: software mechanisms	Handouts	
Oct 19	**Review & Midterm** (in class)		P2 due, P3 out
Oct 26	Operating systems: virtual memory	Ch. 31.2	
Nov 2	Operating systems: processes & protection	Handouts	
Nov 9	Memory systems: Caches	Chs. 1 & 2	
Nov 16	Memory systems: DRAM devices & systems	Ch. 7	P3 due, P4 out
Nov 23	Memory systems: New technologies		
Nov 30	Advanced Topics	Handouts	
Dec 7	**Final Review**		P4 due
Exams	**Final Exam** (Saturday, Dec 19, 10:30–12:30 pm, in classroom)		

Special Needs

If you have a documented disability that requires special needs, please see me as soon as possible, and certainly no later than the third week of classes.