

## Course Plans: CS 3710: Computer Design Laboratory

Instructor: Professor Ganesh Gopalakrishnan (“Ganesh”),  
Office: 3428 MEB (please come in or knock unless door says “busy”),  
Office hours: Tue 3.00 to 3.45pm, and Thu 5 to 5.45pm,  
Email: teach-cs3710@cs.utah.edu,  
Class webpage: [www.cs.utah.edu/classes/cs3710](http://www.cs.utah.edu/classes/cs3710)  
TA: Abhishek Ranjan,  
Office and office hours: TBD  
Email: teach-cs3710@cs.utah.edu

The objective of this course is to offer you a setting in which you can put into practice your existing digital design skills, perfect them, and integrate them while designing a significant digital system built around a computer. Specifically, you must design a computer all the way down to the level of gates, include enough interesting peripherals, and write an assembler. The lecture schedule of this course will offer us a common time and place to meet and discuss the progress of your designs. During the initial month, this will be in the form of lectures and labs that will examine how to evolve the specification of the microprocessor into a working piece of hardware. Specifically, I will lecture on

- the required functionality in your processors,
- the range of optional functionalities allowed,
- how to develop a reasonable execution unit for the processor,
- how to derive its control algorithm,
- how to integrate its peripherals, and
- how to write an assembler and other support tools for it.

While you will already know about many peripherals and tools, I will be explaining

- the tools and languages at your disposal,
- how some of the peripherals work, and how to self-study about others,
- how to document designs (including what notations are available to choose from, and what to choose in a given situation),
- how to simulate designs, and
- how to systematically test designs.

Labs will be conducted on a regular basis for the first month (the exact lab timings will be chosen during the first week of classes). Later labs will be run on an as-needed basis by the TA and the instructor.

**Reference Material:** There will be no required books. Here are the reference materials to be used. **Note: anything left in the lab** is only for your reference. You may borrow it for the purposes of xeroxing ONLY after leaving a sheet of paper in place of the borrowed material saying that you borrowed the item. You must also return the item(s) borrowed for xeroxing within an hour. Violations will cause inconvenience for others and hence will be dealt with punitively.

- [www.cs.utah.edu/classes/cs3710-elb](http://www.cs.utah.edu/classes/cs3710-elb) will be used heavily as many of the requirements are adopted from Prof. Brunvand’s course page (which this is)
- We will also adapt some of Prof. Kalla’s course material - I’ll put that online or leave copies in the lab

- I'll be xeroxing excerpts from Tredennick's "Microprocessor Logic Design" and leaving it in the lab. This will teach you the Hardware Flow-chart notation he used to design both the MC 68000 and the Micro IBM-370.
- Past projects: I hope to put some past CS 3710 projects (including their Verilog sources) online or in the labs. You may borrow good design features but never adopt something without thinking through the details as if you designed them.
- [www.xess.com](http://www.xess.com) and [www.xilinx.com](http://www.xilinx.com) must be thoroughly browsed. There are LOTS of helpful things there.

Section 1 talks about the general course plans. Section 2 talks about the primary focus, namely the processor including the CPU and its peripherals that you will be designing. Section 3 describes what each of you individually and your project group (to be discussed) should have achieved by the end of the semester, the grading method, and the points distribution. Section 4 provides an overview of how one can succeed in this course, including the knowledge one should seek to attain. Section 5 attempts to give a rough course time-table.

## 1 General Course Plans

You should be working in groups of size 2-4 (depends on the project, the class strength, etc.). You should attempt to form the groups by the noon of 8/27/04. In the afternoon of 8/27/04, you should come to the digital systems laboratory (DSL) at 1.00pm and checkout a kit, after going through a brief orientation. For those who cannot make it, a very limited number of slots will be available on the afternoon of 8/30 at 3.00 pm to come to the DSL and checkout kits. (These dates and timings are tentative.) Please plan on spending a few hours in the DSL on whichever day you come. You should send [teach-cs3710@cs](mailto:teach-cs3710@cs) an email including your project group name (tentative is okay), member names, the title of the project (tentative title is okay), and a few lines on what the project goals are (say 'don't know' if you don't know yet; you will be required to provide this information in another week's time).

By the class lecture time of 9/02/04, each project group must have a brief webpage describing their project, and containing the above requested information. The TA and myself will periodically be going through your project web-page. All your accomplishments, milestones, and recent updates must be clearly posted on the webpage, *as this will be the basis for our grading*, especially for weekly assignments. You may also post information on questions you are working on; this way, I may be able to provide hints on where to find answers.

In a lab course such as this, we will be flexible in terms of your milestones. This is because what you set out to develop may not exactly be feasible or may take more time than expected. The idea is to allow you the freedom to explore and find out *good, reliable answers* even if it takes a few extra days. However, the onus is on you to record *briefly and concisely* the efforts being made to seek these answers. Otherwise I'll have to assume that you are not working hard enough on your projects! Also, for the specific assignments given during the first month, there will be hard deadlines, as these assignments will not explore uncharted territory. I shall also instruct the TA to record what he saw during your lab checkoff. On Friday afternoon every week, I intend to browse everyone's webpage and determine your weekly progress, make comments on your work, answer questions or address issues you may have raised in your webpage. I shall also go through the TA's record of the checkoffs. If you miss posting your updates by Friday noon, you will not earn the points for those accomplishments. Make sure I can read the webpage (i.e. the requisite read-access permissions are in place).

You must also sign-up in the email lists `cs3710@cs` (only for messages that warrant to be broadcast) and `teach-cs3710@cs` (messages for me and the TA). Your name also will be installed in the card database of this class to permit you 24/7 DSL lab access.

## 2 Project Requirements

It is a course requirement that you (meaning “each project group”) design a central processing unit that supports an instruction set to be given to you. Specifically, it will be a small subset of the instructions of an embedded processor called CR-16. This common requirement will help us structure the course and help provide a set of core issues that we can discuss and thus reinforce our learning<sup>1</sup>.

Also it is a requirement that all the datapath elements must be hand-designed and entered through schematic capture. Not only will this ensure that your design will be compact, and hence fit on a Spartan-2 – you will also understand digital design much better, without an HDL occluding your understanding. You may design controllers and “random logic” through synthesis.

You will be required to extend the core instruction set to include additional instructions. There is no hard and fast rule on what these extensions should be; here are some thoughts:

- New instructions that perform special-purpose arithmetic functions, such as multiplication, new kinds of bit-vector operations (e.g., *blit*), or inner product (used during matrix multiplication).
- New instructions, or new conventions with respect to existing instructions (e.g., special semantics when certain special addresses or registers are employed) to facilitate the handling of I/O devices. The XSA/XST board combination has the following peripherals that you can use in creative ways:
  - A stereo codec
  - A PS/2 interface
  - A VGA interface
  - A/D and D/A converters
  - (many more; check the documentation)

In addition, you can procure new peripherals (e.g., a solid-state accelerometer) and interface them to your computer.

- New items of software, such as a good simulator for your computer, an in-depth study of the SDRAM controller, a more flexible SDRAM controller that allows you to use the memory chip more efficiently, etc.
- Innovative designs to components of the computer (e.g., better arbiters).
- Adding interrupts to your computer, including perhaps priority interrupts, and system calls.

The real trick is to get the main (required subsystems) working, thoroughly test and document them, and then only take up the proposed extensions.

Last but not least, I’ll assume that you are taking this course because you truly want to be a successful computer engineer. This course is perhaps one of your best opportunities to achieve that result. You must start early on specific tasks; you must give each task the time it deserves; if in the end you are stuck, you must approach us for help asap.

---

<sup>1</sup>Question: what is the least number of instructions one must have in a microprocessor, and on what basis should one choose this set of instructions?

## 3 Deliverables

The final course deliverables must be a natural culmination of all the partial accomplishments you have made throughout. In other words, do not plan on there being a final “crunch mode push” to finish-off things. Even if you pull-off some wonder like that, you would have lost out on weekly progress points if you don’t make uniform progress throughout. In addition, *each member* of a project must be proficient in all aspects of the design. Thus, if one person is doing primarily hardware, he/she must be able to answer questions pertaining to software. The best way to gain this level of knowledge is to *force your project-group member to present it on a regular basis* to you. This is called variously (code review sessions, structured walk-throughs, etc.). By this, you would not only have gained insight into what the other person in your group is doing, you would help him/her articulate their thoughts better on the webpage, and also help them find answers to questions they are seeking. So in your course webpage, do indicate who is taking the lead role on what, and ask him/her to create their web entry after the brainstorming session.

### 3.1 Grading

**30%** of the points will be for weekly progress updates and TA’s checkoff points. You should send a *pass-phrase* to the TA by 9/02/04. Your course points and grades can be looked-up against this pass-phrase. These points will be divided up between the instructor and the TA (both will be making weekly assessments).

**40%** of the points will be for the final project report and how well your project finally worked. Again the split between the final report and the final project sign-off is yet to be decided. It is likely that this is 50-50.

**30%** of the points will be for a final exam, part of which will be a “take home.” The final exam will assess how much you have learnt from this course. You may be asked to design something during the exam - choose a suitable architecture, describe the software you will procure or use, and the kind of tools you will use. The creative part of this exam will be to give you stymied design situations and ask you to solve the associated problems to get the design moving forward again. The take-home will most likely be a particular task to be realized by programming your processor.

There is no late policy, meaning I will not accept anything late. Special occasions needing adjustment of submission dates must be negotiated early enough with me.

### 3.2 Lecture and Lab Schedule

I will be lecturing from 8/26/04 through 9/30/04. On 9/30/04, I will be taking an overseas trip lasting slightly over 2 weeks. We will set-up ways to get the projects moving forwards during this period. The plan is to put you all on ‘auto-pilot’ by 9/30/04 so that you can pretty much be working uninterrupted to get the basic goals of this course achieved. If, by 10/19/04 (when I shall resume my lectures) you have a simple processor working, the essential peripherals working, and the basic software working, you will then be spending a really fun time lasting till December extending your design, studying the extensions, and deriving a lot of first-hand experience with new design approaches. It is okay to fail in some of these attempts or make mid-course corrections in these extensions, as will be required in the real world anyhow.

## 4 Specific Items of Study

The following is a non-exhaustive list of things we shall examine to varying degrees:

- *Basics*: Recap everything from the pre-requisite courses. Also think about
  - Finite automata, Pushdown automata, Turing machines, or alternatively
  - Zero-stack machines, 1-stack machines, 2-stack machines
  - 3-counter machines and 2-counter machines
  - “Real toy” machines:
    - . Move machine (Sutherland)
    - . Ultimate RISC and Minimal CISC (Jones)
    - . Ultimate Ultimate RISC (Griffin)
- *ISE usage*:
  - Download and configure ISE Webpack and ModelSim from xilinx (available in our labs)
  - Schematic editor, control generator, testbench creator, writing HDL test-benches,
  - Synthesis, including options
  - Reading synthesis and map reports, and pad reports
  - Examining floor-plans, and seeing effect of UCF on it
  - Where various modules (e.g. common files) show up in module hierarchy
  - Link to Modelsim, licensing, slow-down of simulation
  - Synthesizing the SDRAM controller
  - Simulation using available libraries (e.g., Unisim, and Block RAM implementation)
  - Unisim libraries
    - . What do they contain?
    - . How to rebuild the library and when?
    - . Care using the library
  - Downloading designs, and uploading memory images
  - Incremental testing using a simulator and on the real hardware
  - Debugging
- *HDL usage*:
  - Describe a hardware design in Verilog (the `pet` circuit)
  - Describe the same design in VHDL, and describe how the porting was done
  - Discuss how the VHDL design was ported to employ the SDRAM memory
  - Which SDRAM controller and why
  - Store duration values into the `block RAM`, making the same type of interface to the block RAM as with the SDRAM. (This will allow you to simulate your design using the simulator; we don’t have a simulation model for SDRAMs but do have one for block RAMs.)
  - Incorporate handshake signals to interface with the SDRAM
  - Write routines to write memory image files. Upload them and see how you can affect the music being played.
  - A taste of Confluence
    - . Demo the Confluence language that can generate Verilog, VHDL, C simulation models, and NuSMV formal verification models

- NuSMV for formal verification: an overview
  - Estimate sizes of blocks designed using schematics vs. behavioral synthesis
- *SDRAM usage + testing:*
  - Go through controller options
  - How to create an interface that makes block RAMs look like the SDRAM (for simulation)
  - Use non-pipelined (Verilog) version?
- *Block RAMs:*
  - What type of block RAMs on chip?
  - What primitives in ISE library?
  - How to synthesize onto one?
  - How to map logic onto block RAMs?
- *CLBs:*
  - What are they?
  - How related to slices?
  - How to use them as RAMs?
- *Onboard FLASH:*
  - Programming the FLASH
  - Quirks; when a FLASH can get wiped out (what “s” switch does that, plus others)
- *Upload, download, formats:*
  - Memory corruption after upload (avoidable?)
  - Formats (“S record,” etc.)
- *XSA and XST boards:*
  - Care and power up/down
  - Handling to ensure long life
  - Hot spots on the board (how to avoid a burn)
  - Speaker interfacing
  - Pin numbering
- *UART:*
  - Learn about a simple UART. Look at its code in Verilog. Also, obtain a schematic design.
  - Learn to incorporate a simple UART into your design. Be able to affect the frequency and duration values via the UART (for PET).
- *VGA controller:*
  - Arbitrating with CPU
  - Learn about a simple VGA controller. Look at its code in VHDL. Also, obtain a schematic design.
- *PS2:*
  - Read and understand VHDL spec

- *CR-16 ISA:*
  - The ISA of CR-16
  - Overall block diagram, and tasks involved in developing CR-16
  - Memory subsystem of CR-16
  - Attaching peripherals to CR-16
  - Sharing the SDRAM - memory arbitration
  - Interrupt subsystem (or programmed I/O?)
  - Detailed structural design of CR-16, and area estimation
  - Various control organizations. Control derivation: HW Flowchart
  - Writing an assembler
  - Writing an architectural simulator
  - Simulation of control and datapaths
  - Block level design of execution units
    - . Register file
    - . ALU
  - Ensuring electrical integrity
  - Testing subsystems + instructions piecemeal
  - Memory maps
- *Project planning, Critical paths, PERT and Gantt charts:*
- *Time management, division of labor:*
- *Doing the bare minimum in all aspects, making it work end to end:*
- *Where to learn about past projects and how much is legit:*
- *Brainstorming:*
- *Presentations:*
- *Maintaining Ongoing Documentation:*
- *More on development methods:*
  - Formal specification of requirements
  - Specification of design
  - Formal and semi-formal verification
  - Writing test-benches
  - Testing the hardware systematically by going through well-chosen scenarios
  - Creating a specification document
  - Use of NuSMV to formally verify designs

## 5 Course Timetable (rough)

The course will consist of several labs done in August and September, followed by a period in October when you get a rudimentary processor up and running. Following that, you will spend November extending your processor, developing the software infrastructure, and preparing for the final demo in December. I'll plan on having some labs where you will demonstrate your basic ability to carry out design sub-tasks. Once the rudimentary processor runs in October, we will have the luxury to examine modern topics such as systems on chips, formal verification, the use of functional languages such as Confluence, other systems and tools such as Esterel Studio and Scade, etc. Here is the schedule of labs planned. Most are modeled after Prof. Kalla's labs except we will require the use of schematics for execution units. Since Prof. Kalla's webpage is not available, I list his lab sequence here to give you an idea of his choices. My choices are listed in Section 6. Lab-1: Familiarization with a computer. Lab-2: ALU and Register files. Lab-3: Control Design. Lab-4: SDRAM controller. Lab-5: Full processor working.

## 6 Tasks till September 23, 2004

1. Everything that's said to be "due" must be checked off by the TA.
2. Some lectures may meet in the lab (I'll post this on the class webpage or is noted below; be sure to check there)

**8/26:** Course intro. The following work is assigned:

- Go through the "warmup exercise" posted on the class webpage. I'll go through this example on 8/31 in class. Download all those files into the ISE Webpack tool and build the circuit all the way to bit-maps. Download into the ISE board and play a song or two. Examine the memory map of the system and upload data through the GXLOAD tool.
- Read cs3710-elb/handouts/ISA.pdf which defines the baseline CRA-16 ISA you must implement
- Read cs3710-elb/cr16/cr16a-prog-ref.pdf which defines the full CRA-16 processor and fully understand the baseline instruction semantics well
- Write a simulator for the baseline CR-16 (due **9/7**). The simulator must read a file containing machine instructions (entered either via mnemonic strings or 0s and 1s; I don't care but you define what's best suited and tell me), and simulates those instructions. For concreteness, you can add a "ORG" (origin) headers that tell where each block of code begins. Assume that each line contains a machine instruction. An example might be:

```
.org 00100010
0000 0010 0101 0011    ; comment: add Rsrc and Rdst
0101 0010 0101 0011    ; add immediate Rdest and immediate value
0100 0000 1100 0010    ; conditional jump on condition 'eq'

.org 00110110
0000 0010 0101 0011    ; comment: add Rsrc and Rdst
0101 0010 0101 0011    ; add immediate Rdest and immediate value
...
```

**Note:** You are welcome to write a tiny assembler now itself. Later it will be required. If you embark on this path, consult the assembler hints contained in Prof. Kalla's Lab 5 handout. In any case, all I want from you is a simulator.

- The simulator must be functional enough to simulate the following programs:
  - . Sort the contents of R0, R1, and R2 in ascending order (contents assumed to be 2's complement numbers).
  - . Factorial of the number in R0; result in R1.

8/27: DSL Lab Kit checkout beginning 1pm for those who've had this kit before

8/30: DSL Lab Kit checkout beginning 3pm for those who've *not* had this kit before

**8/31:** Lecture will be held in the lab.

**9/2:** Lecture in class. Will go through baseline CR-16.

**9/7:** Lectures will be held in the lab. Checkoff for your simulator and the PET circuit in VHDL happens this week in the lab.

**9/9:** Lectures will be held in the lab. Checkoff + discussions in the lab

The following lectures may be in the lab or class; we'll see how the "lab lectures" work out. I'll also clarify all the following tasks by 9/9.

**9/14:** Task to be achieved by now: Make Block RAM work like SDRAM (to be clarified)

**9/16:** Task to be achieved by now: Design and test ALU

**9/21:** Task to be achieved by now: Design and test Register File

**9/23:** Task to be achieved by now: Rudimentary computer working with SDRAM, ALU, and Register File

**9/28:**

**9/30:** Task to be achieved by now: Above rudimentary computer also interacting with one peripheral device (UART and/or VGA controller)