ECE 4750 Computer Architecture, Fall 2022

Lab Assignment Logistics

http://www.csl.cornell.edu/courses/ece5950
School of Electrical and Computer Engineering
Cornell University

revision: 2022-11-27-10-47

This document describes what students are expected to submit for the laboratory assignments and how their submissions will be evaluated at a high level. A handout is provided for each laboratory assignment that describes the motivation for the assignment and provides background on the baseline design, alternative design, testing strategy, and evaluation. Each laboratory assignment requires you to submit three parts: the lab milestone (i.e., one or more initial implementations along with the corresponding tests); the lab code (i.e., all implementations along with all tests); and the lab report (i.e., a document with several sections possibly including an introduction, testing strategy discussion, alternative design discussion, optimization discussion, and conclusion). Although each of these parts is graded separately, it is also useful to consider their relative importance to your final grade. The lab code is worth 25% of your final grade; the lab reports are worth 10% of your final grade; and the lab milestones are ungraded. In total, the labs are worth 35% of your final grade. For those students taking ECE 5740, the labs are worth 30% of your final grade.

Any computer architect can write code. *Good architects* can write code for multiple designs, use a testing strategy to verify these designs are correct, and evaluate the performance and space usage of these designs. *Great architects* can do all of these things, but can also explain their how their designs work at a high level, justify the specific implementation choices used in their designs, use an evidence-based approach to make a compelling argument that their code is correct, and both qualitatively and quantitatively compare and contrast different designs. Doing well on the lab code means you are making progress towards being a good architect. Doing well on the lab code *and* reports means you are making progress towards being a great architect. By the end of the semester, we hope every student will have evolved from simply being an architect to being a *great architect*.

1. Lab Code Release

Initial code for each lab will be released through GitHub, and students will be using GitHub for all development related to the lab. Every student will be working with in a group with one other student. Each group will have their own private repository. All of these repositories will be part of the `cornell-ece4750` GitHub organization, and all development must be done in these specific repositories. **You should never fork your remote repository! If you need to work in isolation then use a branch within your remote repository.** The course instructors will merge new code into the each of these remote repositories, and then students simply need to pull these updates.

2. Lab Code Quality

Code quality is one of the criteria used to assess the labs. Students should follow the coding conventions as illustrated in the tutorial on Verilog. Students should remove unnecessary comments that are provided by instructors in the code distributed to students. Students must not commit executable binaries or any other unnecessary files.

3. Lab Code Submission

The lab milestone and final code will be submitted via GitHub. If you are trying to submit your code at the last minute, then it is possible these last minute changes may not make it into your finalized submission. You should make sure your final code is pushed to GitHub well before the deadline.

You should browse the source on GitHub to confirm that the code in the remote repository is indeed the correct version. Make sure all new source files are added, committed, and pushed to GitHub. You should not commit the `build` directory or any generated content (e.g., unit test outputs, VCD dumps, `.pyc` files). Including generated content in your submission will impact the grade for the assignment. Here is how we will be testing your final code submission:

```
% mkdir -p ${HOME}/ece4750/submissions
% cd ${HOME}/ece4750/submissions
% git clone git@github.com:cornell-ece4750/lab-groupXX

% cd ${HOME}/ece4750/submissions/lab-groupXX
% mkdir -p sim/build
% cd sim/build
% pytest ../labname
# ... run the simulator ...
```

where `XX` is your group number and `labname` is either `lab1_imul`, `lab2_proc`, `lab3_mem`, or `lab4_mcore`. If, for any reasons, the above steps do not work, then the score for code functionality will be reduced. For example, students occasionally forget to commit new source files they have created in which case these new files will not be in the remote repository on GitHub.

We will be using GitHub Actions to grade the code functionality for the labs. So in addition to verifying that a clean clone works on the `ecelinux` machines, you should also verify that all of the tests you expect to pass are passing on GitHub Actions by visiting the GitHub Actions page for your repository:

- `https://github.com/cornell-ece4750/lab-groupXX/actions`

where `XX` is your group number. If your code is failing tests on GitHub Actions, then the score for code functionality will be reduced. Keep in mind that in the final few hours before the deadline, the GitHub Actions work queue can easily fill up. You should always make sure your tests are passing on the `ecelinux` machines and not rely solely on GitHub Actions to verify which tests are passing and failing.

4. Lab Code Revision

After the deadline for submitting the final code, the course instructors will branch your submission and create a pull request on GitHub. The instructors will then commit the instructor tests and evaluation program into this pull request which will trigger a GitHub Actions build. This will enable the instructors and the students to immediately see how their submission does on both the student tests and the instructor tests. The course instructors will provide feedback on the submission including comments on code quality. If the student's lab fails some of the instructor tests, then the students are free to fix bugs and commit these changes as part of the pull request. The students are encouraged to add comments to the pull request indicating what they had to change to pass the instructor tests, and why the student tests did not catch this bug. Students can also commit changes to their code to improve their code quality. **Revised code will be considered in assessing the final code function-**

ality and code quality scores! Also keep in mind that later labs use the designs from earlier labs, so revising your code will improve your chances of success in later labs.

5. Lab Report Submission

In addition to the actual code, we also require students to submit a lab report. The report offers an opportunity for students to convey the high-level implementation approach, motivation for specific design decisions, and quantitative and qualitative evaluation of different designs. We would argue that the ability to convey this information via a technical report is just as important, or potentially even more important, than simply writing code.

The lab report should be written assuming the reader is familiar with the lecture material and has read the lab handout. You might need to paraphrase some of the content in the handout in your own words to demonstrate understanding. Details about the actual code should be in the code comments. The report should focus on the high-level implementation and evaluation aspects of the assignment. All reports should include a title and the name(s) and NetID(s) of the student(s) which worked on the assignment at the top of the first page. Do not put this information on a separate title page. The report should be written using a serif font (e.g., Times, Palatino), be single spaced, use margins in the range of 0.5–1 in, use a 10 pt font size. All figures must be legible. Avoid scanning hand-written figures and do not use a digital camera to capture a hand-written figure. Do not just use a screen capture of code. Definitely do not include screen captures that have white text on a black background. This is not an appropriate way to include code in a technical document. Cut and paste the code into your report as text and format it appropriately. Clearly mark each section with a *numbered* section header. Your report should not look like an outline. It should look like a report with paragraphs and prose. Avoid subsections unless there is a very compelling reason to include them. **Although there is no page limit, most high-quality reports require around four pages of actual text with another one or two pages of figures, plots, and tables.** Note you should interleave your figures, plots, and tables in the main body text where appropriate and not place all of your figures, plots, and tables at the end.

Each report should include an introduction and conclusion, along with a section on the alternative design, testing strategy, evaluation, and work distribution. The final lab report can also include an optional section on optimizations.

- **Introduction (1 paragraph maximum) –** All reports must start with an introduction. Students must summarize the purpose of the lab. Why are we doing this assignment? How does it connect to the lecture material? There are often many purposes. Think critically about how the assignments fits into the other labs. Students can paraphrase from the handout as necessary. Students must include a sentence or two that describes at a very high-level their baseline and alternative design. The introduction should be brief but still provide a good summary of the lab.
- **Alternative Design –** Students must describe their alternative design and their implementation. Students must include a datapath diagram or a block diagram and possibly an FSM diagram for the alternative design. Consider a paragraph that provides an overview of your design, before doing a deep dive into the details of one or two interesting aspects of the design. Think critically about what are the key items to mention in order for the reader to understand how the alternative design works. Examples are usually great to include here to illustrate how the alternative design works. Students are encouraged to describe how their design incorporates specific design patterns and principles discussed in lecture and the discussion section, but be specific. Simply saying the design exhibits modularity, hierarchy, encapsulation, regularity, and/or extensibility is not sufficient; be specific and explain *how* the design exhibits

these design principles. Do not include detailed information about Verilog signals or code; your lab report should be at a higher level. If you include waveforms or line traces then you must annotate them so that the reader can understand what they mean. This section will likely be about one page. **Remember that you must provide a balanced discussion between what you implemented *and* why you chose that implementation.**

- **Testing Strategy –** Students must describe the testing framework provided for testing your design. Students must describe the overall testing strategy (e.g., ad-hoc vs. assertion testing, directed vs. random testing, black-box vs. white-box testing, value vs. delay testing, unit vs. integration testing, and usage of reference models). Simply saying the group used unit testing is not sufficient; be specific and explain *why* you used a specific testing strategy. Students must explain at a high-level the kind of directed tests cases they implemented and why they used these test cases. Consider including a table with a test case summary, or some kind of quantitative summary of the number of test cases that are passing. Students are trying to provide a compelling, evidence-based argument that there design is functionally correct. We recommend students start this section with a short paragraph that provides an overview of your *strategy* for testing (so how all of the testing fits together). Then you might have one paragraph for each kind of testing. Each paragraph starts with the "why" (why that kind of testing) and then goes on to the "what" (what did you actually test using that kind of testing). Then you can end with a paragraph that pulls it all together and tries to make a compelling case for why you believe your design is functionally correct. Do not include the actual test code itself; your lab report should be at a higher level. If you include waveforms line traces then you must annotate them so that the reader can understand what they mean (i.e., what corner case does the waveform or line trace illustrate?). For lab 4, you will also need to discuss your *software* testing strategy in addition to your *hardware* testing strategy. This section will likely be about one page. **Remember to provide a balanced discussion between how you tested your design *and* why you chose that testing strategy and test cases.**

- **Optimizations –** For lab 4, students can include a dedicated section to discuss any software or hardware optimizations they experimented with. We suggest dedicating a pragraph to each optimization. Start by motivating what overhead the optimization is focusing on, then discuss how the optimization can mitigate this overhead, and finally discuss the details of how the optimization was implemented in your implementations. Students should discuss optimizations even if the optimization did not actually improve performance. Do not quantitatively evaluate your optimizations in this section. This section may just be a single paragraph, or could be several paragraphs. **This section is optional. Students can still earn full credit without this section.**

- **Evaluation –** Students must report their simulation results using an appropriate mix of text, tables, and plots. Do not simply include the raw data. A plot is almost always helpful. You must include some kind of analysis of the results: Why is one design better or worse than another? Can you predict how the results might change for other designs or parameters? What can we learn from these results? Students must include some kind of qualitative analysis of the impact of the alternative design on cycle time (i.e., clock frequency), area, and energy. Simply saying one design uses more area or energy is not sufficient; be specific and explain *why* one design might use more area or energy. We recommend students start with the performance analysis and then have three (possibly short?) paragraphs: one each on area, energy, and cycle time. This is where your qualitative analysis comes in on these important metrics. This section will probably be one of the longer (and most important) sections. **Remember to provide a balanced discussion between what the results are *and* what those results mean.**

- **Conclusion (1 paragraph maximum) –** All reports end with a conclusion. Students must include a brief qualitative *and* quantitative overview of the evaluation results (Which implementation performed best? By how much? On which inputs?). Students must include some high-level conclusions they can draw from their qualitative and quantitative evaluation. Do not over-generalize. Can you predict how the results might change for other inputs? What can we learn from these results? Which design should we use in the future? If it depends, explain why it depends.
- **Work Distribution (1 paragraph maximum) –** Students must include a one paragraph description of which student did what work. It is perfectly fine if one students does more work; the key is to transparent and honest about the work distribution across students.

It is also always great to include extra material to help demonstrate your understanding. For example, you could include line traces and reference them in the alternative design to illustrate a key feature of your design, or reference them in the testing strategy section to illustrate a subtle bug or a kind of testing, or reference them in your evaluation to illustrate *why* a specific input pattern performs the way it does. If you include line traces you must annotate them. Label the columns and maybe even draw on them to show what is going You could include a particularly clever test case and reference it in the testing strategy section. You could include a pen-and-paper example to illustrate how your baseline design or alternative design works. Also be sure to highlight "extra" work you did in your design, testing, or evaluation. If you tried two different alternative designs discuss them in the alternative design section and make sure to use them to create a richer comparative analysis in the evaluation section. If you used a new kind of testing technique (e.g., randomly generating different mixed instruction sequences) then make sure you highlight that in the testing strategy. If you added an interesting new evaluation input, make sure you highlight that in your evaluation section. There are many creative things you can do to set your report apart!

Many students initially struggle with the idea of preparing the lab report. In previous courses, students often simply describe their code at a low level in a lab report. In this course, we are challenging students to prepare reports that better demonstrate the student's understanding of the course content. Before starting to write the report, we encourage students to prepare a detailed outline. The outline should include one section for each of the sections that will eventually make up the report. Under each section, there should be one bullet for each paragraph the student is planning to include in that section. This bullet should describe the topic of the paragraph. Under each bullet there should be several sub-bullets, one for each topic to be discussed in that paragraph. The outline should also explicitly include references to the figures, tables, and plots the student plans to include in the report. This is called a *structured approach* to technical writing. Students are strongly discouraged from "just starting to write". Just like we should always plan our approach before starting to write our programs, we should plan our approach before writing the report. Students are encouraged to review their outline with the course staff several days before the deadline.

6. Lab Grading Scheme

The lab milestone is ungraded and is purely a way for students to receive early feedback from the course instructors. The lab code is assessed using four criteria weighted as follows:

- Lab Code: Baseline Functionality 30%
- Lab Code: Alternative Functionality 30%
- Lab Code: Verification Quality 30%
- Lab Code: Code Quality 10%

As discussed in the syllabus, each criteria/subcriteria is scored on a scale from 0 (nothing) to 4.25 (exceptional work). The functionality of the designs is assessed based on the number of test cases that pass in both the student and instructor test suites in combination with the severity of any errors. The verification quality is assessed for labs 1–3 based on the judgment of the instructor in terms of how well the students' test cases actually test the design. The optimization quality is assessed for lab 4 based on holistic quantitative analysis of the performance of the software and hardware on the sorting benchmark. The code quality is based on: how well the code follows the course coding guidelines; inclusion of comments that clearly document the structure, interfaces, and implementation of all modules; following the naming convention and build system structure appropriately; decomposing complicated monolithic expressions into smaller sub-expressions to increase readability; cleanly separating combinational and sequential logic; using local parameters for constants; organizing the code logically to match the dataflow in the design. Overall, good code quality means little work is necessary to figure out how the code works and how we might improve or maintain the design.

The lab report is weighted as follows:

- Lab Report: Introduction 10%
- Lab Report: Alternative Design 25%
- Lab Report: Testing Strategy or Optimizations 25%
- Lab Report: Evaluation 25%
- Lab Report: Conclusion 10%
- Lab Report: Writing Quality 5%

Again, each criteria/subcriteria is scored on a scale from 0 (nothing) to 4.25 (exceptional work). A detailed rubric will be provided with the PA final report grade to explain how each section was assessed.

7. GitHub and Academic Integrity Violations

Students are explicitly prohibited from sharing their code with anyone that is not within their group or on the course staff. This includes making public forks or duplicating this repository on a different repository hosting service. Students are also explicitly prohibited from manipulating the Git history or changing any of the tags that are created by the course staff. The course staff maintain a copy of all repositories, so we will easily discover if a student manipulates a repository in some inappropriate way. Normal users will never have an issue, but advanced users have been warned.

Sharing code, manipulating the Git history, or changing staff tags will be considered a violation of the Code of Academic Integrity. A primary hearing will be held, and if found guilty, students will face a serious penalty on their grade for this course. More information about the Code of Academic Integrity can be found here:

- `http://theuniversityfaculty.cornell.edu/academic-integrity`