CPS499: Independent Study

Subject of study: Reworking the major project of CPS310: Microprocessor architecture.

Student: Joel Sampson ([JSamp701@students.bju.edu](mailto:JSamp701@students.bju.edu))

Table of Contents

[Abstract 3](#__RefHeading___Toc210_140408578)

[Background 3](#__RefHeading___Toc212_140408578)

[Proposal 3](#__RefHeading___Toc214_140408578)

[Milestones 4](#__RefHeading___Toc216_140408578)

[Results 4](#__RefHeading___Toc218_140408578)

[Future Work 5](#__RefHeading___Toc220_140408578)

# Abstract

The current project for CPS310 suffers from a frustrating dispersal of focus. My project is an attempt to assist in Mr. Jueckstock's revamping of the aforementioned project, to develop some tools to aid with the new project, and to establish the ease or feasibility of developing the new project. To that extent, I have aided in the refining of the project specifications, developed a tool to assist the grading of the projects, and attempted to implement a working version of the project.

# Background

The problem my project was aiding in solving is the transformation of the major project of CPS310 from that of a full stack (that is, front- and back- end) simulator to more of a library-based back-end system. In its current state, the original CPS310 project suffers from a lack of focus. One day, the student could be discovering how to implement various ARM assembly instructions, but the next day the same student could be struggling against a frustrating UI system. Though variety sometimes can relieve monotony, the jarring transition from back-end work to front-end work left many students of my year frustrated. In order to get good grades on some of the earlier sections, students, by necessity, would spend long hours tinkering with GUIs, when the major focus of the course itself was developing a large scale simulator for th ARM language.

To rectify this encumbrance, Mr. Jueckstock brought forward the idea of transitioning the simulator (known as ARMSim) from a front and back end development to that of a library-based simulator sitting exclusively in the backend. Instead of students simply implementing a series of written specifications, with this new system students could gain practice on multiple fronts, such as working with native languages for a large scale project as well as implementing a library interface, gaining valuable experience with a task that will directly impact each student in the future. With these ideas in mind, my tasks were three fold.

# Proposal

* Goal: The next group of students to develop the CPS310 project will do so using the new library / API scheme. To this extent, the interface must be complete, reasonable, and extendable. Complete: no core features of a simulator (stepping, memory inspection, register inspection, tracing, etc), should be impossible to implement using the specified interface. Reasonable: the interface should make ready sense as to its intended implementation, through a combination of comments and logical design decisions. Extendable: the API, though it needs to be feature complete and reasonable to envision and implement, must also be easily added, such that additional features can be added without impeding the functionality of other features unnecessarily.
* Goal: The next teacher of CPS310 will not have to develop the entirety of the tools for the course. Because of the new implementation details (that of a linked-library rather than a full-stack application), the numerous old tools for CPS310 will not work correctly in their current state. Therefore, new applications for interfacing with the CPS310 projects need to be written. These applications should accomplish at least one of these three tasks: aiding the teacher in grading the course / assisting the students, allowing students to better visualize their suddenly more obtuse system, and to allow enterprising students to further their discoveries from the CPS310 project.
* Goal: The CPS310 API ARMSim needs to have a working reference simulator by the start of the next session. This implementation needs to be feature-complete and well-documented, serving as an aid both to students and to the teacher.

# Milestones

My project ambitiously started out with numerous milestones, many of which proved unnecessary / infeasible. Currently, the project has / had two major milestones:

1. The completion of the CommandLineRunner application, which fulfills the 2nd goal (the ancillary applications).
2. The completion of the Reference Simulator, in fulfillment of the 3rd goal.

Initially , there were plans for at least one other major milestone, but, after some discussion approximately ¾ of the way through the semester, that milestone was dropped or changed.

My project proceeded in fits in spurts. In the initial few meetings, enthusiasm was high (leading to the ambitious number of milestones). A working prototype of the CommandLineRunner was completed after a modicum of effort and time. Around the latter three-fifths of the semester, however, progress slowed down as other concerns grew. After an unfortunate hard-drive incident near the end of the semester, the Reference Simulator was hastily re-implemented.

# Results

I will split my results into three categories:

* Ancillary Applications: The CommandLineRunner is reasonably feature complete. Using it, the teacher can effectively run a student's ARMSim project. It isn't terribly “idiot-proof”, but it works well for its intended purpose.
  + Part of the project was split off: along with the CmdLineRunner itself is (what can be best defined as) a wrapper around a simulator instance. Combining code from Mr. Jueckstock as well as a lot of ctypes work, this class is currently bare-bones, but is ready for extension to further give grasp on a library instance.
* Reference Implementation: The Reference Implementation is currently in a buggy state. Due to time constraints, development was still ongoing near the end of term. However, I expect, with the addition of one to three days, of effort, the simulator would be a fully-operational A-Level implementation.
* API Refinment: Throughout the other two tasks, my constant interworking with the API led me to several questions / concerns / improvements that I discussed with Mr. Jueckstock. The state of API itself is feature-complete, somewhat-reasonable, and very extensible.

Any perceived, possible improvements to these results will be listed in the “Future Work” Section

Documentation: The code documentation systems are set up in the repository, but, many annotations need to be made. There is much written documentation of meetings in the repository, however.

Tests:

* The CmdLineRunner successfully runs and handles logging for a simulator instance.
* The Reference Implementation unsuccessfully implements the API due to its current, buggy state.
* The API has gone through numerous changes and discussion meetings, leading to a slew of ares to further refine it.

# Future Work

Ancillary Applications

* The CommandLineRunner still has several, possible improvments, such as
  + Adding a scripted input system.
  + Threading support
  + Multi-project support
* The LibWrapper class can be extended so that
  + it gives access to more of the API's functionality. Currently, it allows the host program to run the simulator and to reset it. With only a little extra effort, the wrapper can be made to accommodate the host of features the API provides (i.e., most of the work was in the setup).
* A GDB stub implementation can be created such that a GDB client can connect to the simulator and perform a subset of debugging commands upon it. (with the aforementioned extending of the wrapper class, this GDB stub would be somewhat like a wrapper around the wrapper).
  + This would allow both commmand-line and graphical debugging (quite easily, in fact)

API Refinement (these are somewhat smaller)

* Combine the “set\_entry\_point” function into some other function. As it stands right now, it feels a little extraneous / burdensome to have to remember to set the entry point. A good place would probably be the “as\_create\_simulator” function.
* Rework the MMIO so that it accepts both byte and word sized requests. Currently, the IO feels a little clunky, because the only size of data you can request from an external source is that of a word.
* Better define what varying log and trace levels are. This could simply be done by adding comments to the ARMSIM\_LOGLEVEL and ARMSIM\_TRACELEVEL enums.
* Finally, perhaps the “as\_level” function could be reworked a bit. I rather like the idea of it being like the major and minor numbers listed earlier in the API.