**Project Description**

* My project is titled “Python2RTL” which is an application which can convert basic python code to RTL hardware. Given a string of commands in python syntax, “Python2RTL” will generate the RTL hardware needed to execute these commands.

**Competitive Analysis**

* Currently, there are several industry applications out there that allow a user to code in C or C++ and then converts the user code into RTL hardware. Products like these include Xilinx Vivado HLS (High Level Synthesis) and Cadence Genus which have extremely complicated compilers working in the background to transform complex high level programs into hardware designs for chips. My project will be similar to these tools as they will also transform python to RTL. However, I will limit the complexity of the python code typed in initially to simple, numerical operations on variables for my MVP. Beyond my MVP, I hope to include function declarations, conditional statements, and loops into my project’s functionality.

**Structural Plan**

* My project will be organized into various files where each file contains its own class. As I will be using object oriented programming to define the RTL structures, I plan on using classes to define an RTL structure and then instantiating these classes to generate the RTL hardware itself. All of these files which will constitute the logical and algorithmic part of the project will be organized into a folder I will title “Python2RTL Logic”. Since I plan on implementing a GUI with tkinter to allow the user type in his python code into a simple IDE and then display the hardware it generates, I will have a separate folder I will title “Python2RTL GUI”. This folder will contain the GUI python code in its own file and will have other files with graphics of the RTL structures such as the gates and higher level blocks

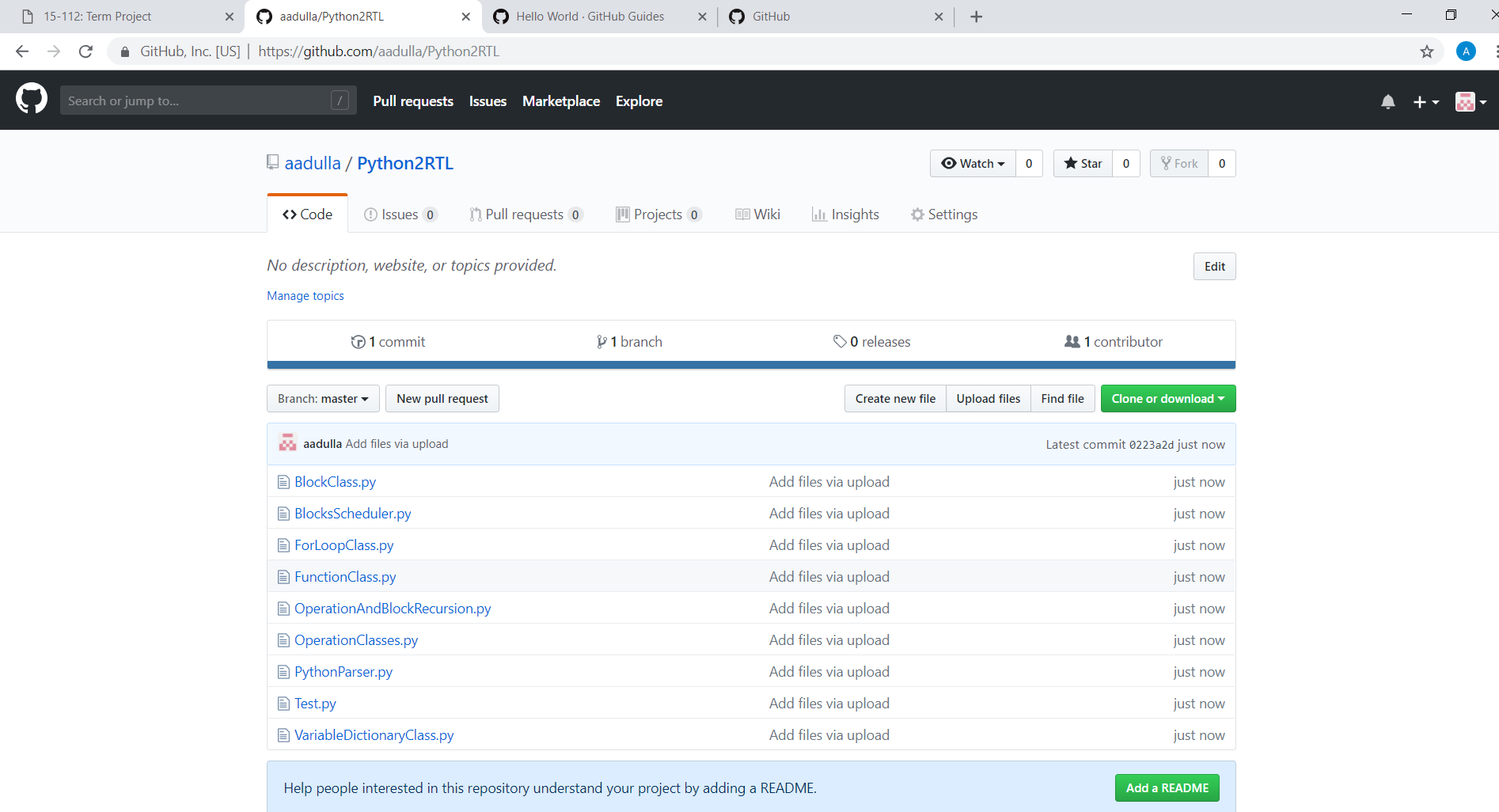
**Algorithmic Plan**

* The hardest part of this project will be the shift in paradigm between sequential execution to parallel execution. When code is written in any high level language such as python, the code is executed sequentially line by line. However, in hardware, operations can be done both concurrently and sequentially which allows programs to be run more efficiently than if they were run in software. To approach this problem of transforming sequential python code to a sequential and parallel execution of operations, I will use recursion to step through all the code the user types in and map what each operation depends on. If Operation A depends on Operation B, then Operation A needs be executed sequentially after Operation B. However, if Operation A does not depend on Operation B, then Operation A can be executed in parallel with Operation B. With this idea, I will generate blocks for each operation which consists of three main attributes: input 1 dependency, input 2 dependency, and output. Using these attributes, I can see what operations are dependent/not dependent on each other and will then generate a CDFG (control data flow graph) mapping the operations in the most optimized order.

**Timeline Plan**

* 11/20: Finish scheduling compiler and be able to do basic computation
* 11/28: Finish basic GUI which allows user to type in python code and displays RTL hardware
* 12/6: Extend functionality to work with conditional statements, loops, and function declarations

**Version Control Plan**



* I will be using GitHub to back up my code. Each time I plan on modifying or adding on to my existing code, I will first test it on with my local files in Pyzo. If I am confident in my new code, I will then make a new branch with my changed code, and will then merge it into the master branch.

**Module List**

* No modules being used

**TP2 Update**

* I will be restructuring the GUI to instead have pop up windows instead of tabs
* I will be implementing different levels of abstraction for the hardware components (i.e. 16 bit adder🡪full adder🡪RTL gates) rather than just going directly to RTL gates. The application will display the highest level of abstraction first, and the user can click on the individual blocks to open up the next lower layer of abstraction
* I do not think that I will have enough time to implement function declarations, loops, and conditional statements. Instead, I will focus on the arithmetic operations and the user experience of the GUI

Note: to test out the application, need to run Master.py