18-500 - ECE Design Experience
Team Name: Game Boi
Team Members: Tess Chan (theresec), Adolfo Victoria (avictori), Pratyusha Duvvuri (dpratyus)

Statement of Work

Our refocused project will be similar to our original project with the only change being that we will no longer be supporting audio functionality. With remote access only, we will no longer have access to the tools we would need to get the audio circuitry working. To be able to realistically continue on with the project, we needed to remove the audio portion to accommodate the situation. We are able to continue with the remainder of the project because each member has a board, so testing and debugging can still be evenly distributed amongst members.

With the removal of audio, the main division of labor change that we will need to make is Tess will focus her time working on the CPU, rather than balancing her time between the APU and CPU. This will allow Adolfo and Pratyusha to focus on their individual components and give the team more time at the end to work on debugging integration.

The emulator community has spent countless man-hours working on getting the timings for instructions and execution of the Game Boy. Thus to test our FPGA portion, we will be using the test ROMs that they have come up with to test that the different timings work. These test suites are mooneyes-gb tests and blarggs instruction tests, which are very comprehensive and consist mainly of unit tests. Our plan to use these won't change since they target specific parts of the subsystem (i.e. only the CPU or PPU), and after integration we can do more of these that rely on timings between the PPU and the CPU. We will scale back on our goal to match VerilogBoy because of the additional testing and integration issues that will come since we are all separated. We will just try our best to get as many tests to pass and get at least a simple game that does not use too many timing specific graphical tricks (like Tetris) running.

Our SoC component is unique relative to other emulators. Therefore, we have to write our own unit tests to verify the communication between the FPGA and SoC. Since the SoC will house the persistent memory, it will hold our game state and will need to transfer the game bits depending on the game that the user wants to play. To check this, once we transfer the bits, we will sum all of the stored bits in both the FPGA and persistent memory. If the sums are the same, we can reasonably assume that none of the information was corrupted. Additionally, we will attempt to read in a game to the FPGA, then the FPGA will read the information back to the SoC untouched. If the bits all match, then we have proven that two way communication between the FPGA and SoC are correct. To test our game switching, we will run the above test consecutively,

but use a different game in each iteration. This will prove that the SoC can switch between memory areas and the FPGA does not have any stuck bits.

To test the controller driver, we will map each button of the controller to an LED. The LEDs that correspond to the pressed buttons must be on until the buttons are released. This will prove that we are able to receive the right signals from one or more buttons of the controller at any time. Additionally, it will prove that we are able to convert the input signal from the controller into a signal that the FPGA can interpret.

We plan to use these test suites and our developed unit tests as validation that each individual component works and the issue lies in integration if our project does not work.

If we are able to successfully integrate all the components and have a running project, we will keep our performance requirements as stated in the design document. We will test input lag by using a high speed camera to measure the input delay, measuring the time it takes from a button press to a change registering on the screen. To verify that we are cycle accurate, we will trace our bus and VerilogBoy's bus while running the same input commands, and compare the logs.