Final Project Implementation Plan - a RISC-V Processor for ZYNQ

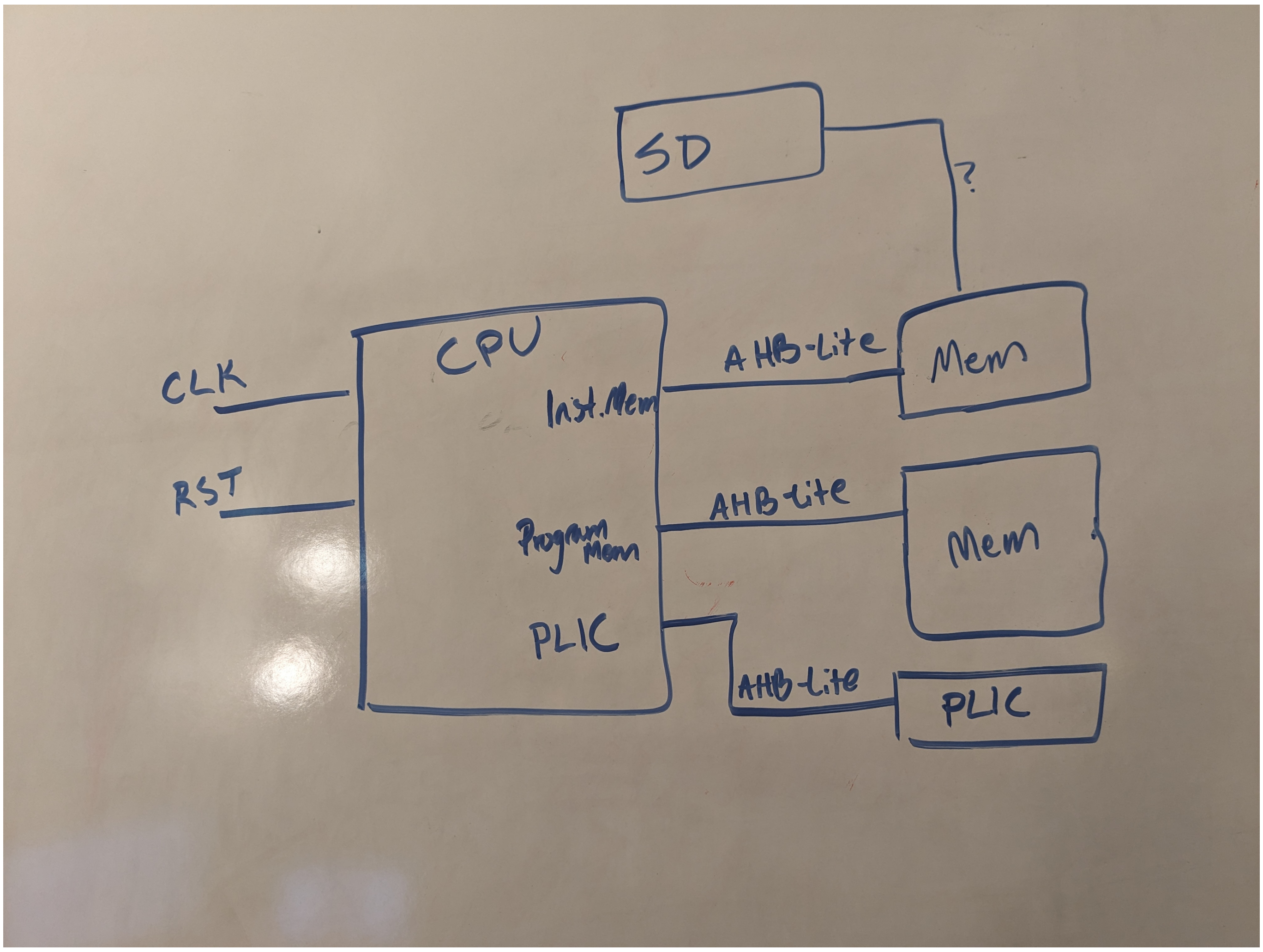
Taylore Todd & Emma Westerhoff

Status

We are starting with Verilog code generated from the ecen5593-startercode repository. We've been able to generate the code and include it in a Vivado project. We are building a small test bench suite leveraging iverilog and gtk-wave for the fetch and decode modules to ensure our understanding of the input/output signals needed for the AHB-Lite buses.

Plan

1. We will find IP blocks for AHB memory banks to act both as instruction memory and program memory. We'll be starting here: AHB-Lite Memory Datasheet. We will have a state machine in our top file, which will have two states- init and runtime. During init, a couple of hard-coded instructions will be loaded into instruciton memory. At runtime, the clock will be routed to the CPU and operation will begin. The only way to transition back to init will be the reset btn or a power cycle. We'll use an iverilog/gtk-wave test bench to confirm that the several instructions we hard-coded are performing as expected.



2. We will work on our memory initialization, specifically instruction memory initialization. We will need to leverage documentation from ARM Developer to do this. The init stage will be modified to copy instructions off of the SD card into instruction memory, although we aren't sure of which protocol we will leverage to do this. We will enforce a strict / hard-coded file naming convention to circumvent as much of the operating system as we can.
3. We will find an AHB-lite compatible RISC-V PLIC (Platform-Level Interrupt Controller). We'll need to modify the CODAL source code according to the ISA to process inputs from this controller. This will allow us to map IO devices (LED, switch, buttons) to memory and allow us to control them.
4. We will run some simple handwriten assembly (derived from a C program) on our system. The code asked about in this Forum Post will provide a good starting point. Our code will use an IRQ to flash some LEDs.
5. We will connect the UART to our processor. We'll write our own module since UART is so straightforward, and one of the learning goals of this project is familirization with writing Verilog.
6. We will add extended floating point capabilities to our processor. We will likely modify the design in Codasip and regenerate the RTL. This will give us some experience with the CODAL workflow and developing code in Codasip alongside integration with third-party IP blocks.
7. At this point, our design is done. We will document our system, include any performance metrics, and include future work.