Will Krska and Conor Naughton

EC551 Fall 2022

Lab 3

**Part 1: Finishing Existing Designs**

Our lab 2 required several corrections in order to make it fully functional. We did not have specific errors which we were running into during synthesis, unfortunately, most of our issues were occurring once we implemented the project on the board and were unable to observe the functionality we expected. We utilized a number of troubleshooting techniques in order to figure out what was going wrong including connecting flags to LEDs so we could track what was happening, displaying the UART signals as well as the program counter on the 7 segment as well as showing the entire contents of the register file and the first 12 locations of the memory over VGA based on a few switch values. The following describes some of the changes we needed to make in order to get our project functioning properly:

**UART Controller**

We simplified our UART controller to simply contain both the RX and TX UART controllers which pass the transmitted and received data through the UART protocol

**UART Arbiter**

We found that we needed a module which took the input from the UART and generated a sustained value in order for it to be used by other modules. This module waits for the UART valid flag to go high and then holds the data as an output until the UART valid flag goes low.

**Data Loader**

This module takes both the PS2 data from the USB connected keyboard as well as the user input over UART, translates the input as needed (two separate statements for PS/2 and UART input) which are then fed into a large FSM which identifies which processing mode has been selected (I, L, A or B) and which stage within the mode we are in (intermediate operand input or execution command). It also contains two submodules which implement the benchmark matrix multiplication operation as well as the ALU operations. The module outputs a number of flags which are used to identify when the mode has been selected, when instructions have been entered and when the ALU or benchmark operations have completed as well as the results of those operations or the instructions which have been entered.

**Character Stager**

This was the most significant change made to our project. This module takes as input the flags and data output from the data loader as well as the PS2 data (translated to ASCII format) and the UART data from the user input and pushes these values into a FIFO in the order that they should be written to the terminal display. Additionally, in the case that the operation performed was a benchmark or ALU, it formats the output and displays it properly. We also had to add some new FSMs in order to properly implement the display of text and data and generate new lines after data is displayed.

**Part 2: Design Optimizations**

**Pipelining**

Our originally designed datapath included a four-stage pipeline with the following stages: Instruction Fetch, Decode, Execution and Writeback. We looked for ways in which we could improve the pipelining of our datapath and did not see any obvious way in which it could be improved.

**Power Optimization**

In order to optimize the power consumption of our design, we looked to use clock gating to ensure that registers where the input data has not changed are not being clocked and thus wasting power. The largest power consumption in our design occurs in the datapath and so this is where implemented the clock gating. The following summary report produced from Vivado shows the power consumption of our design prior to implementing clock gating:

