Midterm Project
Hardware Design - ECE 311

December 10, 2019

Partners: Jinhan Zhang
Kevin Jiang

Abstract

The lab aims to study MIPS and ARM processors and learn the fundamental architecture, functional blocks, performance, and applications of them. Then, we designed a 32-bit MIPS processor that implemented the required functional blocks on Vivado and run simulation on testbench and test it the Zedborad. Finally, we analyze the FPGA utilization of our design and include pipeline to our design to increase the throughput of the processor.

Introduction

So far in the class, we have programmed our FPGA board to function as many different computer components, such as logic gates, flip-flops, multiplexers, ALU, RAM, registers, and so on. These components can all be combined to make one processor unit. Two processor architectures are especially suited for programming on an FPGA board: MIPS and ARM architectures. In this project, we research on these two computer architectures, and in part 2, we implement a MIPS Single-Cycle processor on our FPGA board.

Procedure

Part 1

RISC Architecture

MIPS and Arm are both reduced instruction set computer (RISC) architectures. The RISC instruction set makes the common case fast by including only simple, commonly used instructions. The number of instructions is kept small so that the hardware required to decode the instruction and its operands can be simple, small, and fast. More elaborate operations that are less common are performed using sequences of multiple simple instructions.

MIPS Architecture

MIPS instruction sets includes R, I, and J type as shown in [1]Figure 1.
R type instruction takes in three registers as the source register that stores the input of the instruction, the target register that stores the second input, and the destination register that stores the output of the instruction. op and funct specify the operation of the instruction. For R type format, op is 0 and the operation is defined by funct. shamt is 0 for R type instruction. Figure 2 shows an example of R format instruction and the corresponding assembly code. I format takes two register operands and one immediate operand. The rs and imm field are used as input source and rt stores the destination register or another source sometimes. The operation is determined by opcode only. Noted that the imm stores 16-bit immediate fields but are used in 32-bit operations and there will be a sign extension of the immediate. Figure 3 shows an example of I format instruction. J format. This format is used only with jump instructions. J-type instructions begin with a 6-bit opcode. The remaining bits are used to specify a 26-bit address, addr. An example of J format instruction is shown in Figure 4.

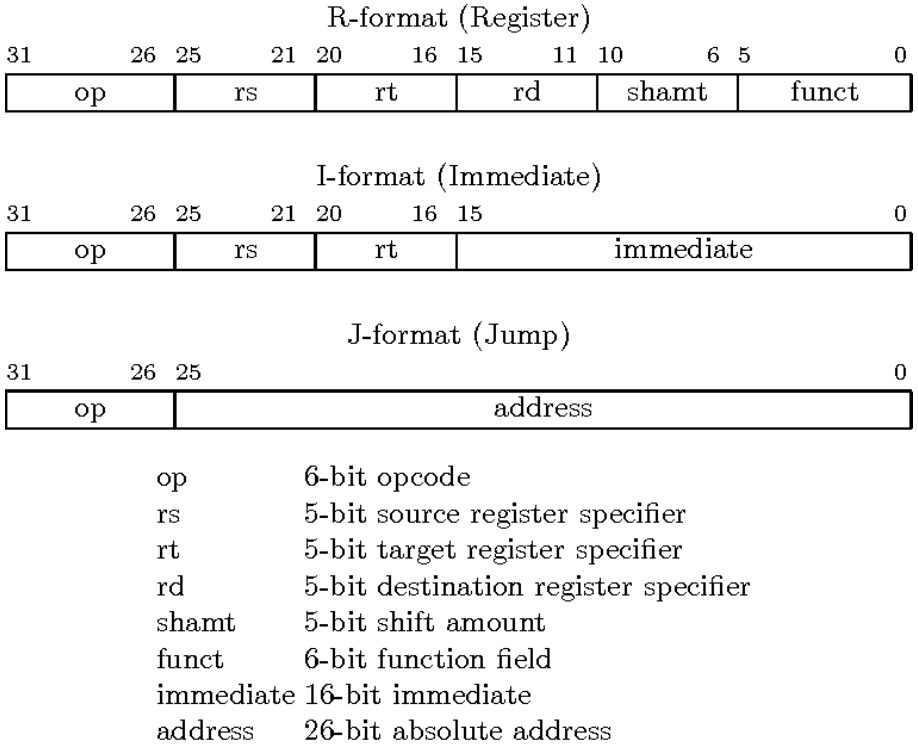

Figure 1: MIPS Instruction Types

[1]Figure 1 source: https://www.semanticscholar.org/paper/MIPS-code-compression-Hirvola/900487134b68c5180441b14d7d86f4f4d1887d13

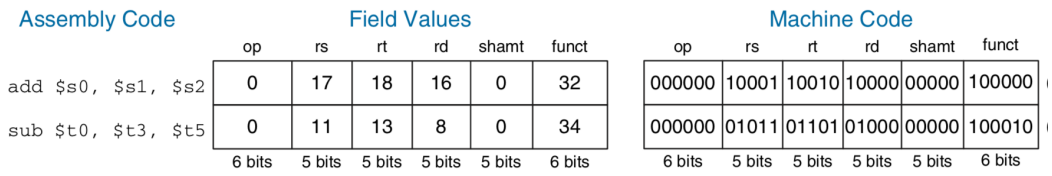

Figure 2: R Format Example

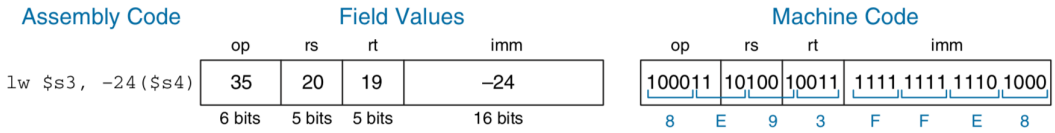

Figure 3: I Format Example

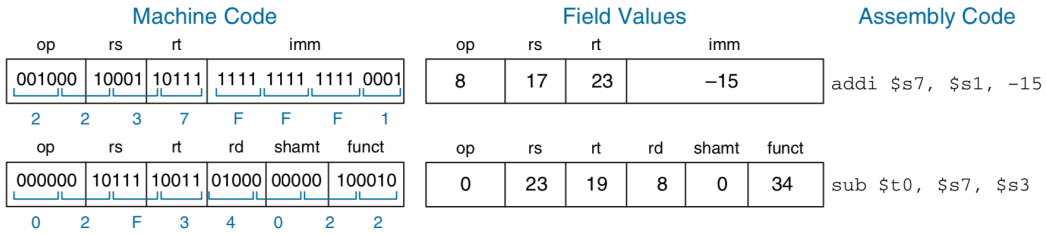

Figure 4: J Format Example

ARM Architecture

The ARM architecture processor is a 32bit reduced instruction set computer (RISC) micro-controller. There are many different versions of ARM architecture. ARM Cortex implement Thumb-2 instructions set and v7 implements Neon, HW divide and Thumb-2 instrucition sets.

Part 2

Figure 5 shows the high-level design schematics for our MIPS processor. The "imem" module is the Instruction Memory, where the data from the intruction file is read to. The "dmem" module is the Data Memory, and all "lw" and "sw" instructions write/read to/from it. The "mips" module contains the rest of the component of a MIPS processor covered in the previous section, such as the Program Counter, Register File, ALU, Sign Extendor, and MUX's.

Simulation Results

Figure 6 shows the behavioral simulation of the 2-1 multiplexer. The simulation results match what you would expect from a typical 2 bit wide 2 to 1 multiplexer.

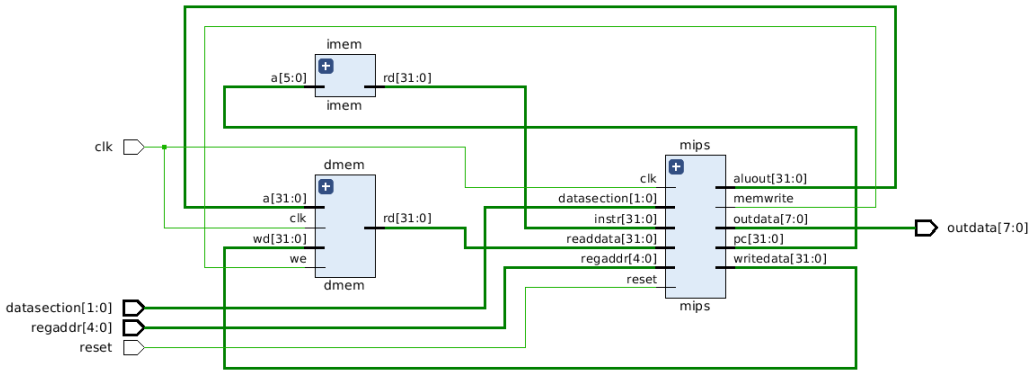

Figure 5: Design Schematics of Single-Cycle MIPS Processor

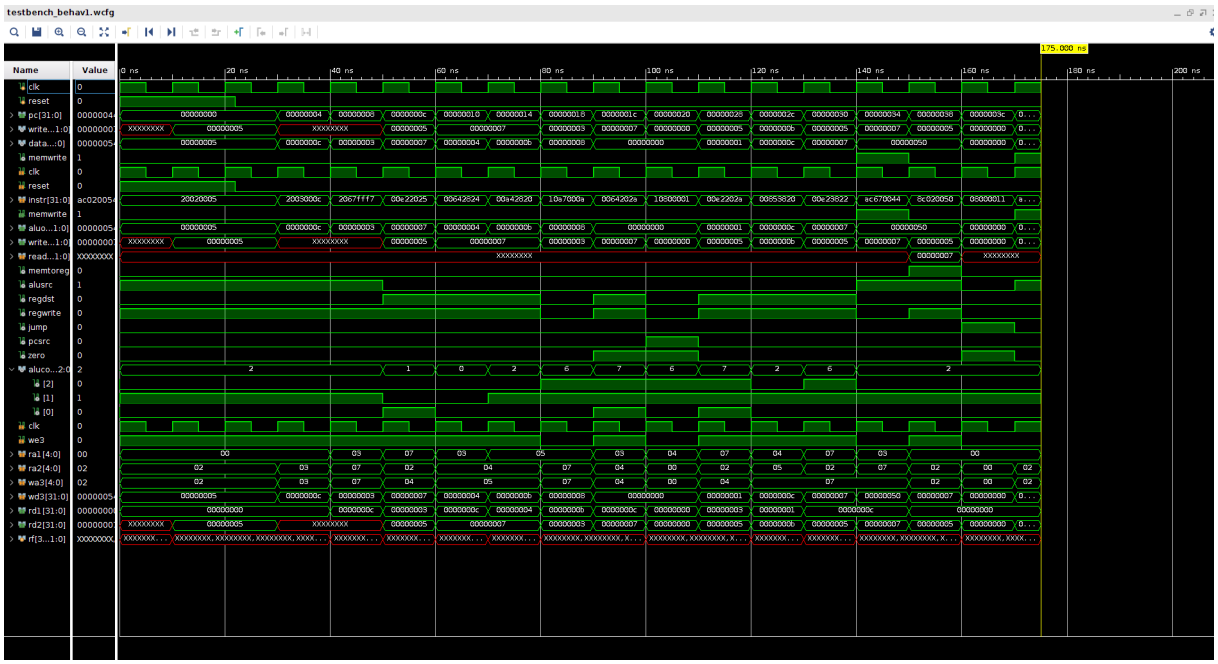

Figure 6: Behavioral Simulation of Single-Cycle MIPS Processor

Synthesis Results

Figure 7 shows the synthesized schematic of the design.

Implementation Results

Project Summary is shown in Figure 8, and the implemented schematic is shown in Figure 9.
LUT: 0.81% of LUT is utilized.
LUTRAM: 0.60% of LUT is utilized.
FF: 0.01% of FF is utilized.
IO: 8.50% of IO is utilized.
BUFG: 3.13% of BUFG is utilized.

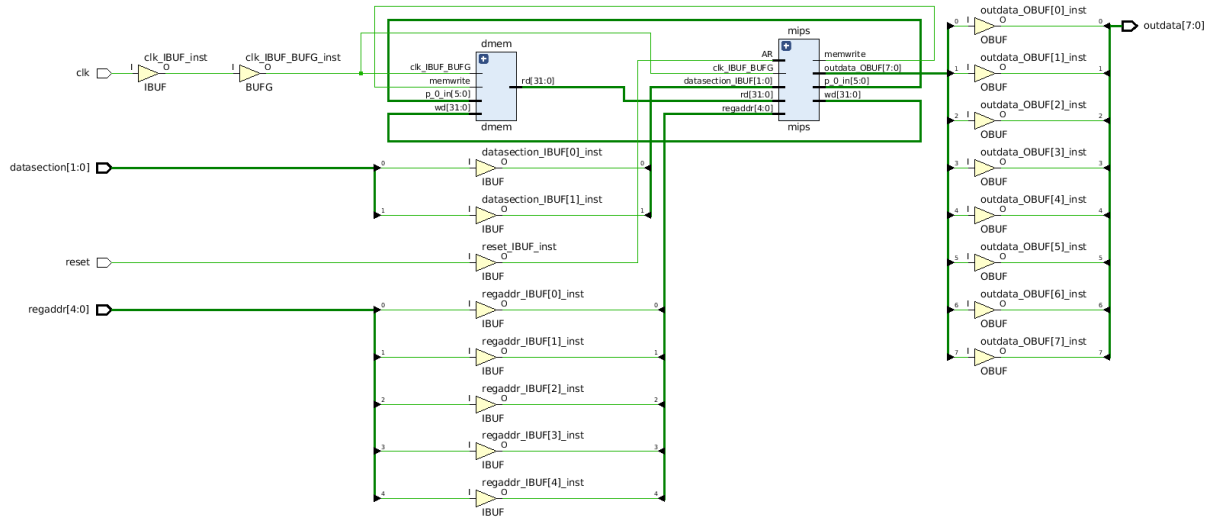

Figure 7: Synthesized Schematics of Single-Cycle MIPS Processor

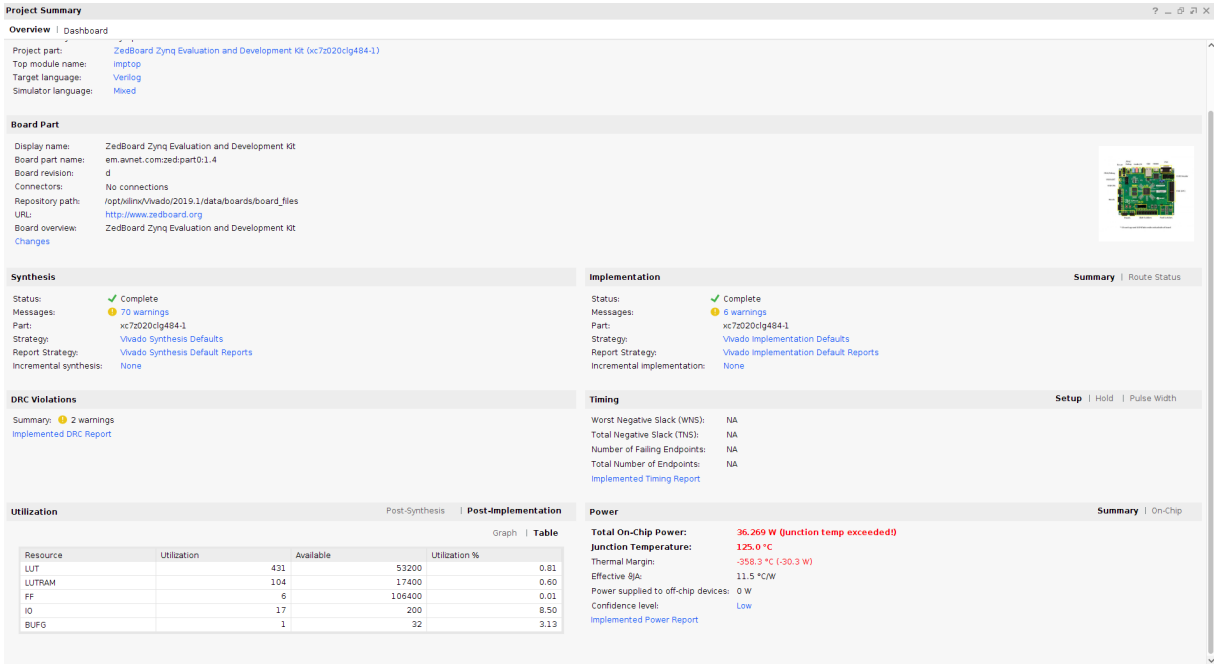

Figure 8: Project Summary of Single-Cycle MIPS Processor

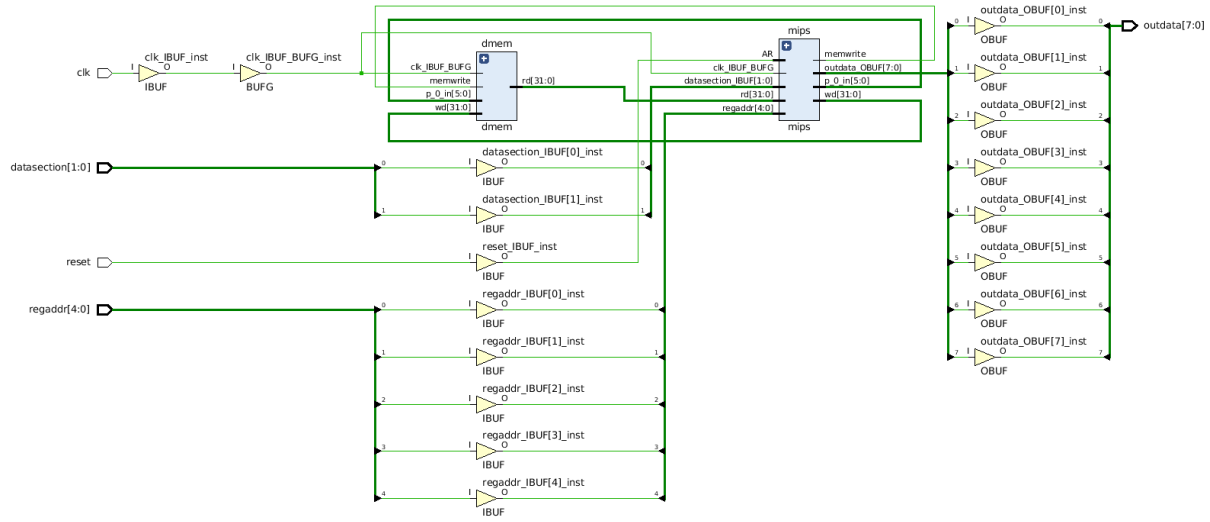

Figure 9: Implemented Schematics of Single-Cycle MIPS Processor

Summary and Discussion

Experiment	Simulation	Theoretical
The project was to build a MIPS processor using Verilog, and that was a success. We were able to generate a program the FPGA board as a MIPS processor, and by manually controlling the clock of the processor, we were able to execute a full set of instructions given by the textbook.	The simulation was a success. It took us a couple tries to succeed because some of our modules, like the ALU were not written correctly. By looking at the states in the incorrect simulation, we were able to quickly debug our code.	From this project, we learned about the architecture of MIPS processors, and we learned how to implement one using Verilog and a FPGA board. We learned about the essential components that evert MIPS processor should have and we feel confident that we can use what we have learned so far to continue to improve this processor.

Overall Conclusions

Over the course of this project, we applied everything we learned in the labs and lots of research to build a Single-Cycle MIPS Processor. We learned about the details of a MIPS architecture, such as how data is represented, how instructions allocate bits for different purposes, and the overall logical flow of data in different instructions. Although most of the module codes for the Single-Cycle MIPS processor were provided by the textbook, we implemented our own ALU module and integrated it into the MIPS processor. We also added code to the modules to allow us to observe the data in each of the 32 registers by flipping switches on the FPGA board. Although this processor is very simple and not the most efficient, resarching and learning about it gives us a good foundation for us to start making better processors.