CMPE 260 Laboratory Exercise 3

Instruction Fetch and Decode Stages

Aden Crimmins
Performed: March 3, 2021
Submitted: March 17, 2021

Lab Section: 2
Instructor: Richard Cliver
TA: Corey Sheridan
Justin Soler
Jake Michalski

Lecture Section: 2
Professor: Mr. Cliver

By submitting this report, you attest that you neither have given nor have received any assistance (including writing, collecting data, plotting figures, tables or graphs, or using previous student reports as a reference), and you further acknowledge that giving or receiving such assistance will result in a failing grade for this course.

Your Signature: *Aden Crimmins*

Abstract

The purpose of this exercise was to design both the Instruction Fetch and Instruction Decode stages of the MIPS architecture. It utilized a Register File and Control Unit for the decode stage in addition to its base functionality. The Instruction Fetch stage obtains the instructions for use in the decode stage which derives the commands for the execute step. Both components were tested using a test bench designed to check the functionality of the design for test instructions with changing inputs. This lab was successful in creating both of the Instruction stages as well as properly test it to ensure it functions as intended.

Design Methodology

The design was created using the Xilinx Vivado Design Suite and implemented on the Basys3 FPGA. The Instruction Memory was created to hold all the instructions that are used in the fetch and decode stages. It was created to be byte addressable, with a max capacity of 1024 bytes of instruction data. The output of the memory is in the word format so in order to obtain the instruction as a word, the value accessed at the input address is concatenated with the 3 bits that follow it.

The logic was created for the Instruction fetch stage which read instructions contained within the instruction memory starting at 0 and incrementing 4 each clock cycle, the output from the instruction memory are then immediately used as the Instruction output without further involvement from the fetch stage.

The Instruction Decode took in the Instruction, as well as register writing parameters and through the use of a register file and a control unit, decodes the instruction into the commands to be used in execute. The Control unit took in the Opcode and Function sections of the Instruction and from those values determined if it required reading or writing from the memory or writing to a register. It also determined which location the destination register was and if it would use an immediate. Using the inputs it also directly determined the ALU Opcode corresponding to the provided instruction. The Register File read the data from the provided registers for RD1 and RD2, while writing to registers on the falling edge of the clock when the enable was high. The Instruction Decode itself acted on the rising edge to assign the inputs to the internal components, with the component outputs directly tied to its own outputs.

Results and Analysis

The testbenches created for this lab tested a variety of cases of each of the input values for both the Instruction Fetch and Instruction Decode stages, as well as different operations to ensure full functionality of each stage and their components. Figure 1 shows the behavioral waveform for the Instruction Fetch stage

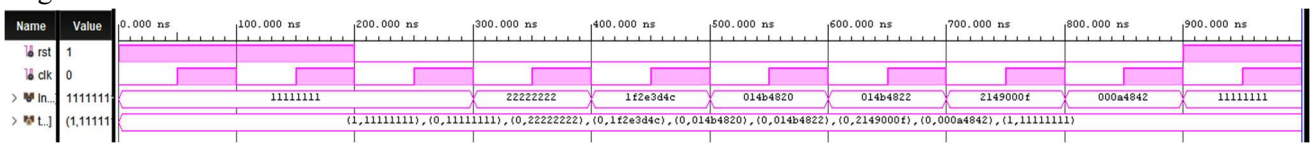

Figure 1: Instruction Fetch Behavioral Simulation

The simulation begins by testing the reset function with it being high on the first and second iteration, then low in the next iterations. This shows that the Instruction is not incrementing and is staying at the first value in the instruction memory. This is again validated in the last iteration as it is turned back on

and the output once again shows the value in the first address. On the falling edge fallowing the reset going low the register to be read is incremented and each iteration after that it is incremented again. This shows the correct functionality for the instruction fetch stage.
Figure 2 shows the Post-Synthesis Timing simulation of the same fetch stage using the same register values.

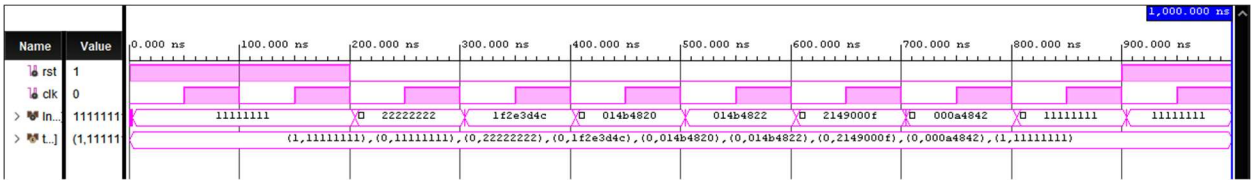

Figure 2: Instruction Fetch Synthesis Simulation

The Post-Synthesis timing simulation tests the gate level model of the code without accounting for any delays. This waveform shows the same results as the behavioral simulation which means that there arent any major design flaws that are visible within the synthesis simulation.
The next test was the Post-Implementation Timing Simulation shown in figure 3 which tests the same set of inputs as the behavioral simulation.

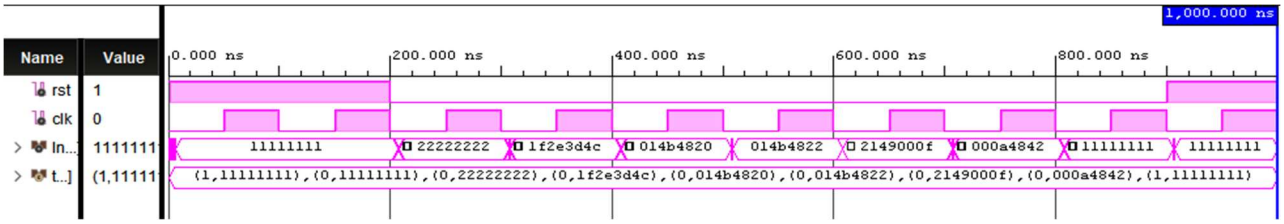

Figure 3: Instruction Fetch Implementation Simulation

The Post-Implementation timing simulation tested the implemented model as it would behave on the actual hardware. This includes the delays resulting from each gate in the model which can be seen whenever there is a change in the inputs as there is delay between that change and the resulting output. This delay is relatively small as there is relatively few gates that the logic has to move through due to the simple nature of the Instruction Fetch stage.

The Decode stage has its own testbench to confirm its' functionality and the behavioral simulation is shown in Figure 4.

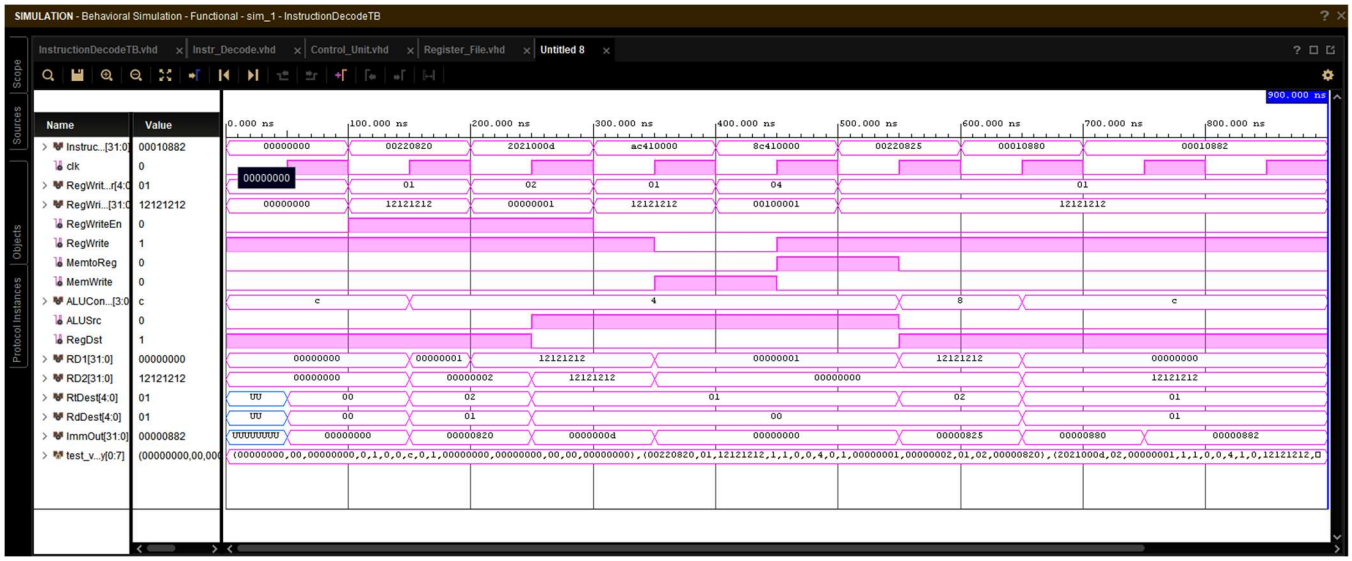

Figure 4: Instruction Decode Behavioral Simulation

The simulation begins by testing the function of the decode with an empty instruction. This shows that the outputs for the function are valid for this extraneous test. The ALUcontrol has the correct value of “1100” for a 0-value operator, each of the registers are reading zeroes as the 0 register always returns a value of 0. Both of the Rt and Rd destinations are 0 and the immediate is calculated to 0.These all align with the expected values from the test bench. for each of the following instructions different components are tested to make sure they work. Register 1 is written to with a value of “12121212” in the second iteration and that can be seen on the falling edge as that is the clock determined time for the registers to be written to. The second iteration also reads registers 1 and 2 which have values of 1 and 2 initialized at the start of the code. The immediate value is calculated correctly using the last four digits of the hex instruction input, with a sign extended function to make them 32 bits. In the next iterations the ALUControl is tested to make sure it gets the correct output for the provided input, as well as MemtoReg and MemWrite values for the SW and LW functions. Each of the outputs from the simulation matched those of the testbench. This shows the correct functionality for the instruction fetch stage.

Figure 5 shows the Post-Synthesis Timing simulation of the same decode stage using the same instruction values.

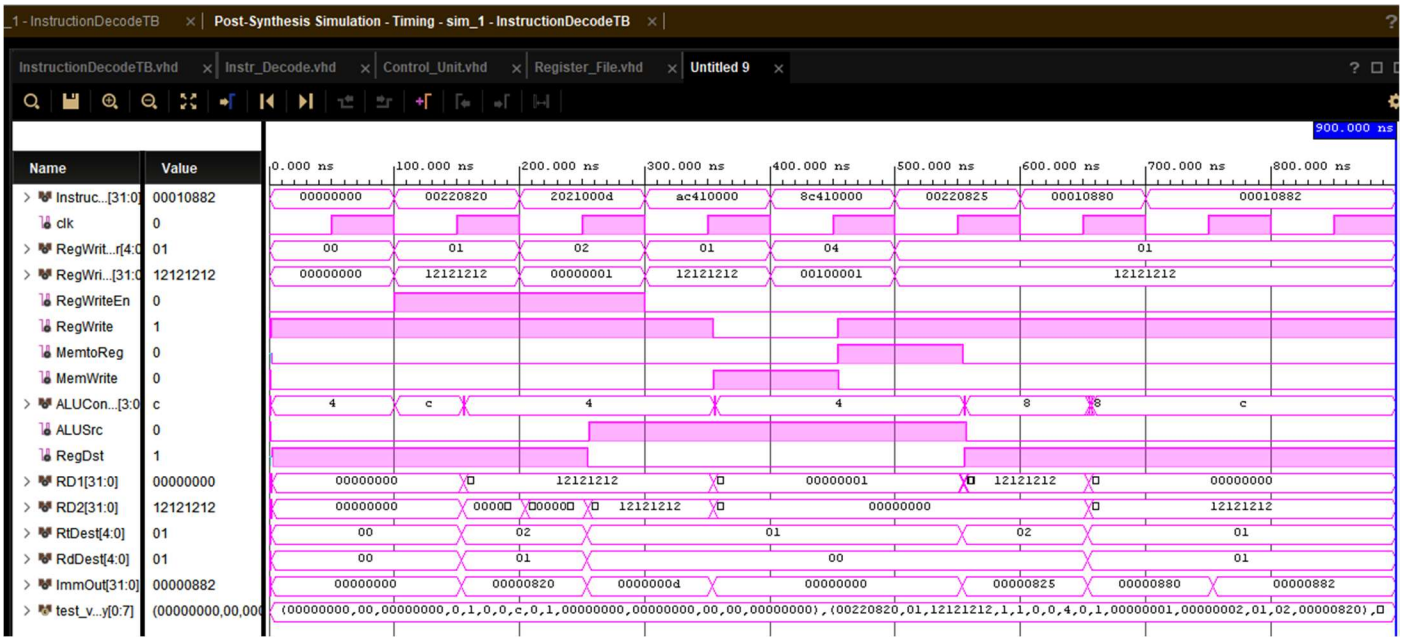

Figure 5: Instruction Decode Synthesis Simulation

The Post-Synthesis timing simulation tests the gate level model of the code without accounting for any delays. This waveform shows the same results as the behavioral simulation for the decode stage which means that there arent any major design flaws that are visible within the synthesis simulation.
The next test was the Post-Implementation Timing Simulation shown in figure 3 which tests the same set of inputs as the behavioral simulation.

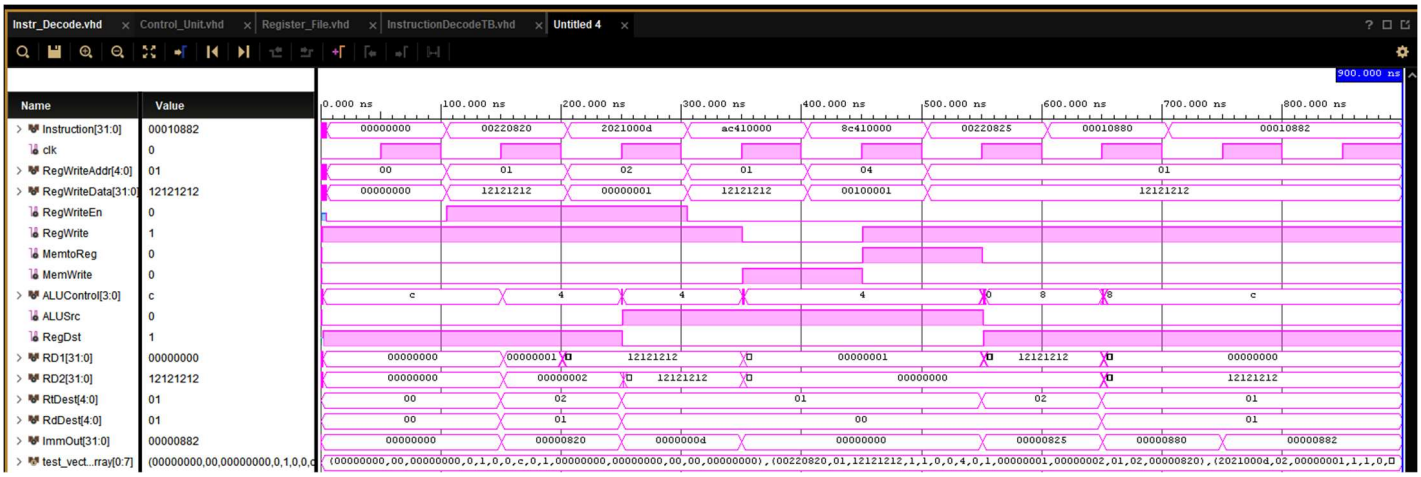

Figure 6: Instruction Decode Implementation Simulation

The Post-Implementation timing simulation tested the implemented model of the decode stage as it would behave on the actual hardware. This includes the delays resulting from each gate in the model which can be seen whenever there is a change in the inputs as there is delay between that change and the resulting output.

The synthesis and implementation of the project was done without any errors as is shown in Figure 7.

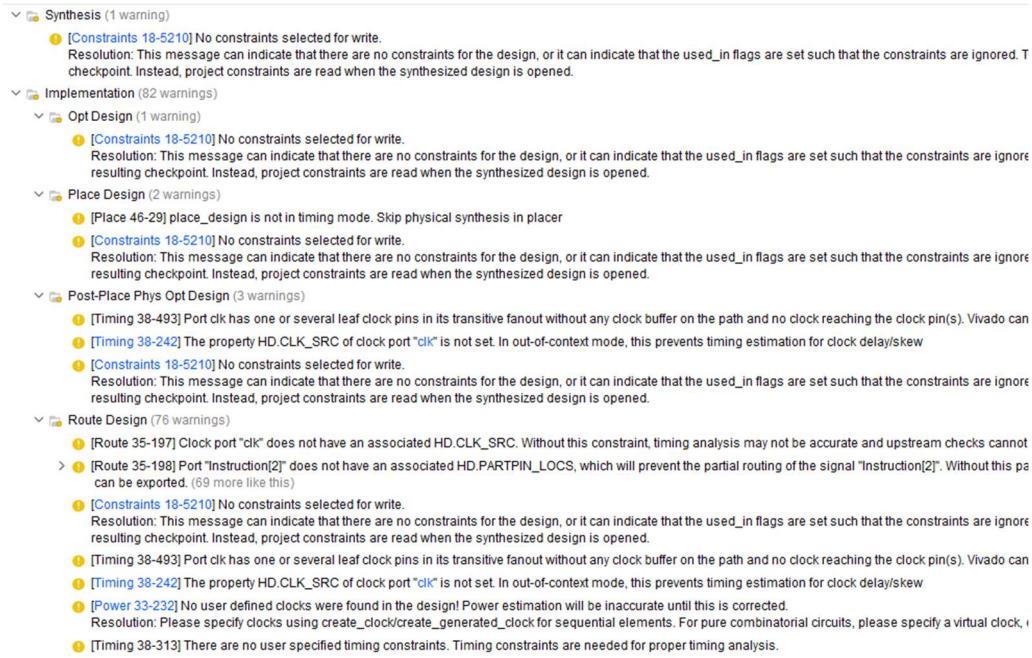

Figure 7: Synthesis and Implementation Warnings

The only warnings that were generated during the processes were timing and constraint warnings that do not hold importance or raise concern on the design of these components.

Conclusion

The Fetch and Decode stages created were implemented successfully to complete the functionality to obtain instructions, then break them down into their functional components. Each of these operations within the stages were tested in the Xilinx architecture through the Behavioral, Post-synthesis and Post-Implementation simulations with the test benches designed for both stages. The simulations showed that both stages were designed correctly both in the Fetch stage and Decode Stage. There were minor issues when obtaining the Implementation waveforms, but after closer inspection a latch was removed and the timing for the test bench were changed, then it compiled correctly and there were no other major issues.