Jaci Reichenberger, A20131719
Stephen Potter, A11341625
ECEN 4243: Computer Architecture
8 February 2021

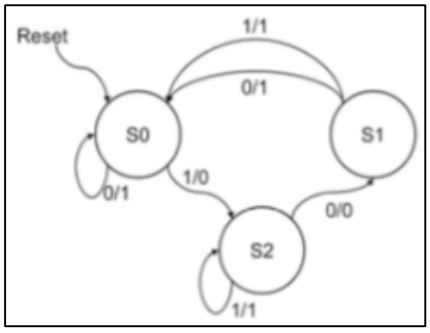
Lab 1: Revisiting Digital Logic and Verilog Simulation

Section 1: Introduction

The purpose of this lab is to introduce the Verilog tools that will be utilized in this class. Using code already provided, this lab introduced how to use ModelSim to view the output waveforms of an FSM. Students were able to read through the code and learn how to utilize a DO file in order to see the waveform inputs and outputs in the application. The second part of this lab introduced how to write code, a test bench, and modify a DO file in ModelSim in order to show both the input and the output waveforms for a register. The code for the register was created following guidelines given for this lab. The test bench code was written to test the various guidelines given, and the DO file was only slightly changed from the FSM's DO file. In sum, this lab introduced students to ModelSim and reminded students how to code in Verilog.

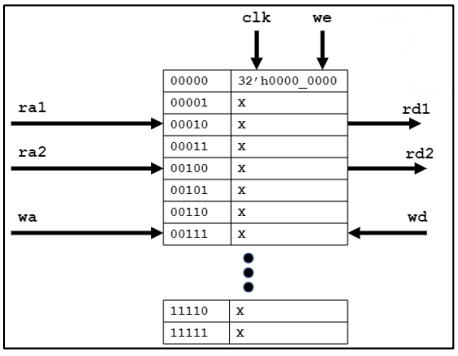
Section 2: Baseline Design

Part 1: FSM Simulation



The figure above is a basic Mealy Finite State Machine (FSM). The code was given for this part, and it walks through the various states of this diagram. This FSM works with inputs and outputs. Based on whether the input is a one or a zero will determine if this FSM moves on to the next state or not, and its output depends on the input as well. For example, in the above FSM, if the input is a zero, then it will stay in S0 with an output of one. If the input is a one, however, it will move from S0 to S2 and output a zero. All this is seen using the given testbench and the DO file that output the various waveforms for the various states of this FSM.

Part 2: Register File and Simulation



The second part of the lab was to create the code, testbench, and DO file for a register. A file was given that contained the variables needed to write the code in order implement the register. For the inputs, there are two 5-bit source register numbers (ra1, ra2), one 5-bit destination register number (wa3), one 32-bit wide date port for writes (wd3), one write enable signal (we3), a clock (clk), and two 32-bit register values (rd1, rd2) for the outputs. The register behaves as follows: writes take effect synchronously on the rising edge of the clock and when write enable is asserted high, the read port should output the value of the selected register combinatorically, the read port will update on the rising clock edge when reading and writing the same register, reading register zero always return the value zero, and writing register zero has no effect. The test bench will verify these conditions, and the DO file implements the test bench in order to prove the correct outcomes have occurred with the waveforms.

Section 3: Design

For Part One of this lab, an FSM design was provided in the form of a Mealy machine. The purpose was to study the design and understand how it was being implemented using Verilog code. This was also used to re-familiarize students to Verilog and to bring comfortability with using Verilog code, along with introducing students to the software that would be used throughout the semester. Looking at the code for the first part of the lab increased understanding about how the code flowed and connected, and observing how the output waveforms verified the results. Construction and implementation of the code for the FSM was observed. Also, the FSM testbench allowed for analyzing what results should be expected and allowed for further understanding of how all the parts work together. This helped in the second part of the lab when tasked with using a provided register file skeleton code to write the register file. The bare bones were provided, however students had to write their own logic to implement the guidelines found in the lab manual. The lab report stated that the register file must include, for input, two 5-bit source register read ports, one 5-bit destination register write port, one 32-bit data port for writes, a write enable signal, and a clock., as well as two 32-bit register values for reading outputs. This was all provided in the register file for students to observe and to glean the portion that was left to write in the register file. What was left to write in the register file was the logic to implement the tasks that were laid out in the lab manual. These guidelines were as follows:

- Writes should take effect synchronously on the rising edge of the clock and only when write enable is also asserted (active high).
- The register read port should output the value of the selected register combinatorially.
- When reading and writing the same register, the read port will update on the rising clock edge. The read port is still combinatorial.
- Reading register zero ($0) should always return (combinatorially) the value zero.
- Writing register zero has no effect.

The goal was to write the rest of the System Verilog code to implement these criteria into the provided register file to make a working register file. In the logic part of the register file, it was decided to implement an always combinational section under which it produced logic to check and read the above mentioned read ports. Then, as the lab manual stated, the write enable was implemented on the positive edge of the clock signal and instructed what registers to point to in order to write the data to the proper register. This completed the register file. Next, a testbench needed to be

created that allowed testing on the register file for both inputs and outputs. This part of the implementation will be discussed in the following section.

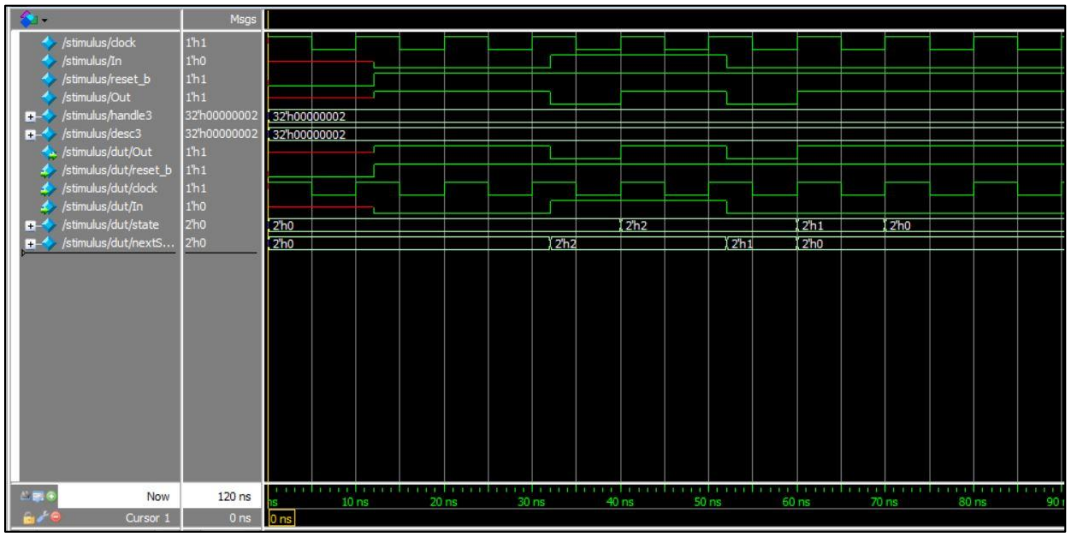
Section 4: Testing Strategy

The next step was creating a testbench for Part Two of the lab. Using the FSM testbench as a guide, the testbench for the register file was designed. After working through what was needed and what variables were required to change, only one section left needed changing from the FSM testbench. That section consisted of the main tests, in which values were asserted to the registers randomly to see if what went in came out as intended. It was ensured that the actions that were supposed to happen on the rising edge of the clock were happening when they were supposed to and with the correct values. It was asserted that the write enable is high at an arbitrary time, then proceeded to make sure that when a value was set to the write data register, the value observed in the simulation was correct. The other test cases were tested, such as when write enable was low, writing to the register is not allowed. Additionally, the time increments put into the code were checked to ensure that they made sense with the results and that everything behaved properly. This was the majority of the testing, which confirmed that the implementation was behaving as expected. It was decided to choose random defined numbers to set the registers to so that it would be easily identifiable to check the outputs for the simulation. The DO file was then created to allow ModelSim to simulate the new register files. This was done by adding the names of the files to the DO file, then running the simulation. This approach seemed most reasonable and simple to observe if the proper behavior was being produced. Below is a table showing the input and output behavior of the implementation that was obtained through simulation. As one can see when the values are asserted, the correct results are produced that was meant to be obtained through the code and testbench of the register file.

	With we3 set to 0 & wd3 set to 2				With we3 set to 1 & wd3 set to 7			
Registers	Input	Output	Input	Output	Input	Output	Input	Output
wa3	0	0	1	1	1	1	1	1
wd3	2	0	7	0	2	2	7	7
ra1	1	1	1	1	1	1	1	1
ra2	1	1	1	1	1	1	1	1
rd1	1	1	1	1	1	1	1	1
rd2	1	1	1	1	1	1	1	1

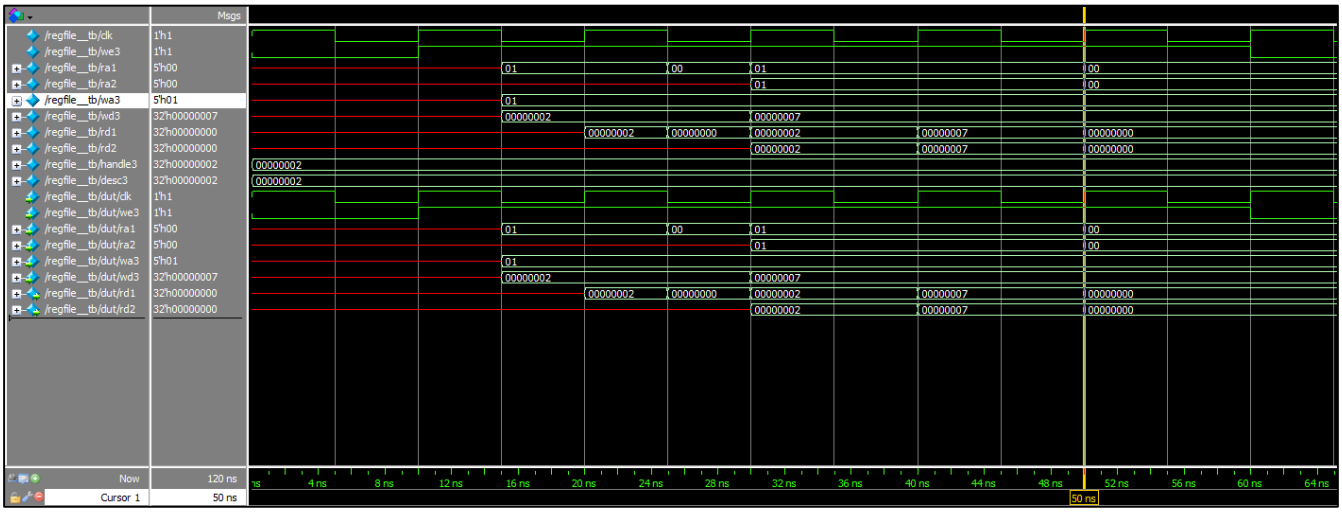
Section 5: Evaluation

Part 1: FSM Simulation Results



The results seen in the picture above prove that the given code matches the diagram of the FSM. Having the given code for the FSM and studying these results served as a reminder of how the testbench is able to prove the code for the FSM works as intended. In summary, Part 1 of this lab has improved the knowledge for using ModelSim for writing and testing code.

Part 2: Register File Simulation Results



The results seen above prove that the register is working as directed. The waveforms prove graphically that the testbench created proves that the code written works as intended. Writing the code for the register helped put into practice on writing in Verilog. Having the FSM code to reference helped in writing the testbench in order to test each case.

This lab reintroduced how to code in Verilog and how to navigate ModelSim. From analyzing the code for the FSM to writing the register file, this lab aided in learning how to work ModelSim. Learning how to design testbenches and create DO files helped in proving that the code for the FSM and the register works as expected. In summary, this lab was able to familiarize ModelSim for simulation and viewing the outputs and reteach the basics of writing in Verilog.