Lab 0
Revisiting Digital Logic and SystemVerilog Simulation

Thomas Kidd CWID: 20275230
Zach Wilson CWID: 20267002
Electrical and Computer Engineering
Oklahoma State University
Stillwater, Oklahoma - United states of America
thomas.kidd@okstate.edu & stealth.wilson@okstate.edu

1 Section 1: Introduction

Computers of all kinds use registers and even more, register files. A register file (RF) stores temporary variables. This is very important for a computer to help enable pipe-lining and other ways to speed up computation and instruction execution. Specifically it allows for the reading of data which gets stored in the RF, and for the writing of data which gets pulled from a location within the RF. In our laboratory and class we will be working with RISK-V's 32 bit ISA. Because of this our RF ends up having 32 registers x 32 bits. It is important for students to understand how a RF works, as its a key component in a computer and to help speed computation up. The simulations done to verify the DUT save time and money as they offer a quick method to test code via testbench.

Starting out with the given code, our baseline design was not effective. Our lack of understanding and having a hard time remembering SystemVerilog gave us a setback in the beginning. After some thought our baseline design ended up being as provided in the textbook of the DDCA Example 5.8. This allowed us to effectively complete the laboratory. Because the baseline design was effective enough, no alternative design came up.

Our program ran for a total of 12 cycles without any problems and any difficulty. Because this is a simulation, other than errors and warnings the program should run as expected without any trouble.

2 Section 2: Baseline Design

A RF, is simple a cabinet or collection of drawers that you can place items in and take items out. It allows for the storage of data and instructions and benefits the computer by helping with pipe-lining. The basic workings of the RF can be broken into writing and reading data. The reason why the baseline design given from the textbook works well is because of the simplicity it gives the computer. Simply having this module enables the reading of data from the RF at any moment. This is very useful for either regular operations, or for some emergency operation that must be executed at the next clock cycle.

2.1 Writing Data

WE can write data into the register file by having an input and a address to show where to store that data in the RF. The only caveat to this operation is that it must take place on the rising edge of the clock with an enable bit. Please see *Figure 2* for a visual explanation of the order of operations. In addition this operation of writing data must happen before any data is being read. This is because

if there is nothing in the register to be read, then nothing will happen. This is important because we don't want to write any data unintentionally into the RF. This could cause overwrites and unwanted changes causing errors and bottlenecks in the process.

2.2 Reading Data

Reading data can take place without the need for any clock or enable input and for two different registers at the same time. This is important because some instructions require two inputs at the same time, such as arithmetic. In addition, this data might need to be accessed immediately for the computer. To highlight the workings of this function we can explain it by using the drawer analogy. If we want to find a specific item, we need to know where to go. This is where we use addresses. Upon giving the address, the program immediately carries out the action of reading the data into the outputs of the RF. This can, yet again be seen in *Figure 3*. As the address for input is given, the data in the output also appears. Please see *Figure 4* for a visual explanation.

3 Section 3: Detailed Design

Going into detail, there are two main sections we can categorize our design. The DUT and the testbench. The DUT servers the purpose of what our code would do in a real hardware device, while the testbench allows us for testing our DUT.

3.1 The DUT

Within the DUT there are again two different sections we can further break down our design. Reading and writing data take place in different ways and serve different purposes. Writing data to the RF is a synchronous process while reading data from the RF is a combinatorially process that can take place at any time when we want. Please see *Figure 1* for a general overview of the entire process.

3.1.1 Writing data to the RF

As this is a synchronous process we have to ensure that the SystemVerilog code takes part in a always_ff block. This function also must be executed on the rising edge of the clock and with the enable bit high. This helps to avoid any overwriting of data. The execution of this section is simple but a few checks must be put in place. To be more specific we need to make sure that we are writing the data to the correct location. This is where the address variable helps us by pointing the data to the desired address in the RF. Please see *Figure 2* again for an explanation of the order of operations when it comes to writing data. Its important to understand that this process of writing data must comes before reading any data from the RF, for the exception of reading from the zeroth bit. This is because reading from the RF is impossible if no data has been given to it. This was our main issue when trying to test our code.

3.1.2 Reading data from the RF

This process is combinatorial, meaning that it can take place any time without the need for a clock signal or a enable signal. Again, this is crucial to the correct execution of the code. Sometimes instructions or data need to be fetched immediately in order to carry out the next instruction on the next clock cycle. As seen in *Figure 3*, the order of operations is simple. When we want to read

from the RF, we give an address that will then show the output where to get its data from. While a simple process, the fundamental idea of data retrieval, movement, and storage all go into this simple RF. The only additional nuance to this can also be seen in our code. We add a simple conditional statement to the combinatorial process. If the address bit is equal to 0 then the data output will also be equal to 0. This is a key component of the RF. Often times we must retrieve the number 0 to make sure our arithmetic operations carry out. For example, if we want to output 1, we can add 0 to 1. Because 0 is such a common number in arithmetic and other operations it is crucial that we make sure that we can retrieve 0 any time from the RF. This can clearly be seen by the red lines in the beginning of our simulation for rd1 and rd2 as showing in *Figure 3*.

4 Section 4: Testing Strategy

Our testing strategy for this lab revolved around using a testbench file. With this application we were able to test the functionalities of our DUT. Using Modelsim we were able to check different assigned values at different clock cycles as seen in *Figure 4* insuring each change resembles what is expected from the DUT. Specifically we tested each functionality of the DUT in the way a RF would work. Since the workings of the RF are quite simple, we didn't feel the need to extensively test it.

5 Section 5: Evaluation

As this is a software only lab, based on ideal simulations alone, the evaluation of the experiment comes down to is the RF working as desired. We were able to verify this by using our testbench described in the above section. As the enable reads high, we store wd3 at the address of wa3.Then we continue this process, adding another address location and storing that data as well. This can be seen in the first two clock cycles of *Figure 4* under wd3, wa3, and we3. Finally, to evaluate the correct function of our RF, we carry out the combinatorial logic. We give ra1 the address RF[2] and ra2 the address RF[3]. We can immediately see that rd1 changes when we give ra1 the address. The same goes for rd2 as this can be seen in the bottom half of *Figure 4*. The delay between the reading of data from RF is because in the testbench we give the delay. This descries the proper workings of the RF as its completing its intended purpose. In conclusion, the usage of a RF is simple, yet crucial to the workings of a computer. This will be demonstrated further in future labs where we can go more in detail on how our RF functions qualitatively and quantitatively with respect to the hardware implementation.

6 Figures,Tables, and other components

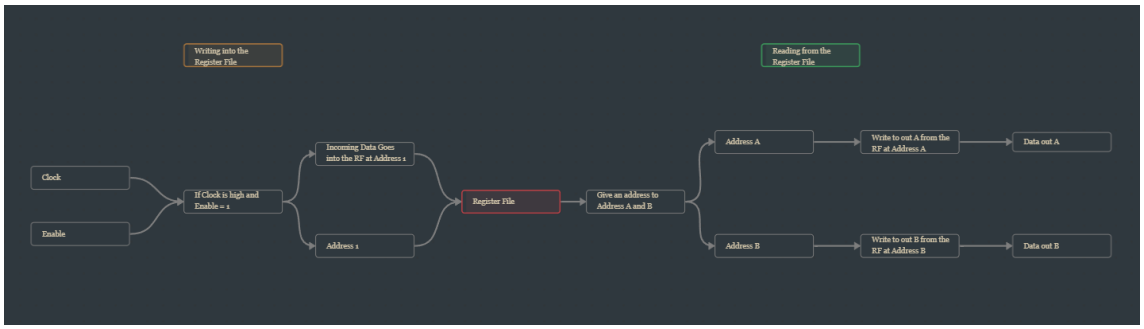

Figure 1: The order of operations when it comes to the Register File

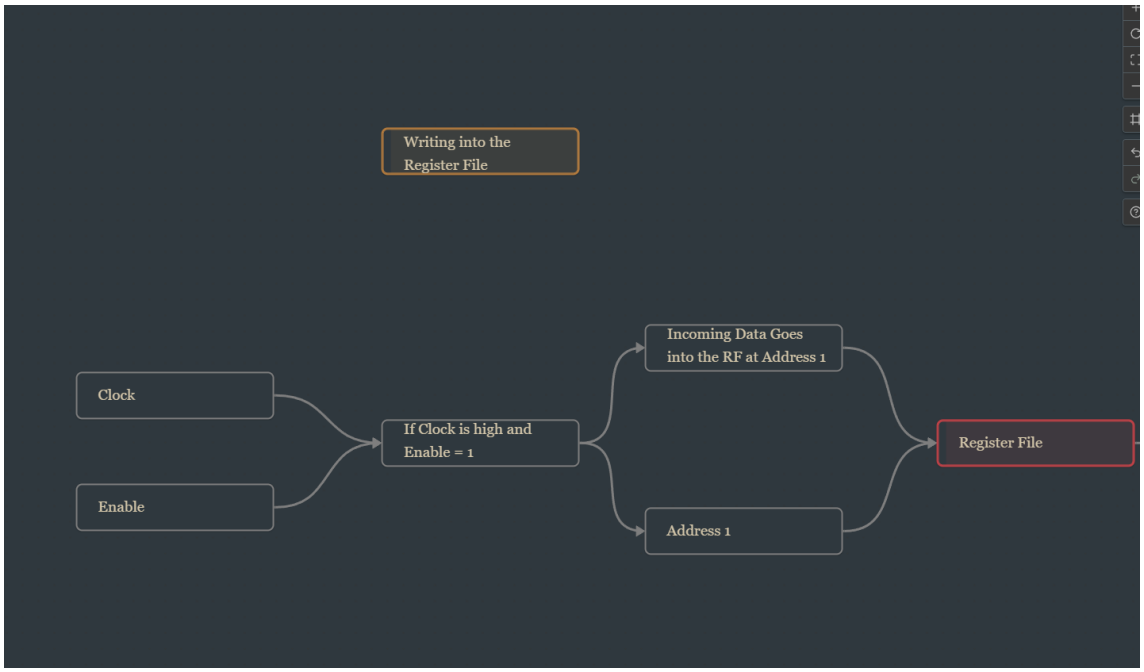

Figure 2: The order of operations when it comes to the writing of data in to the RF

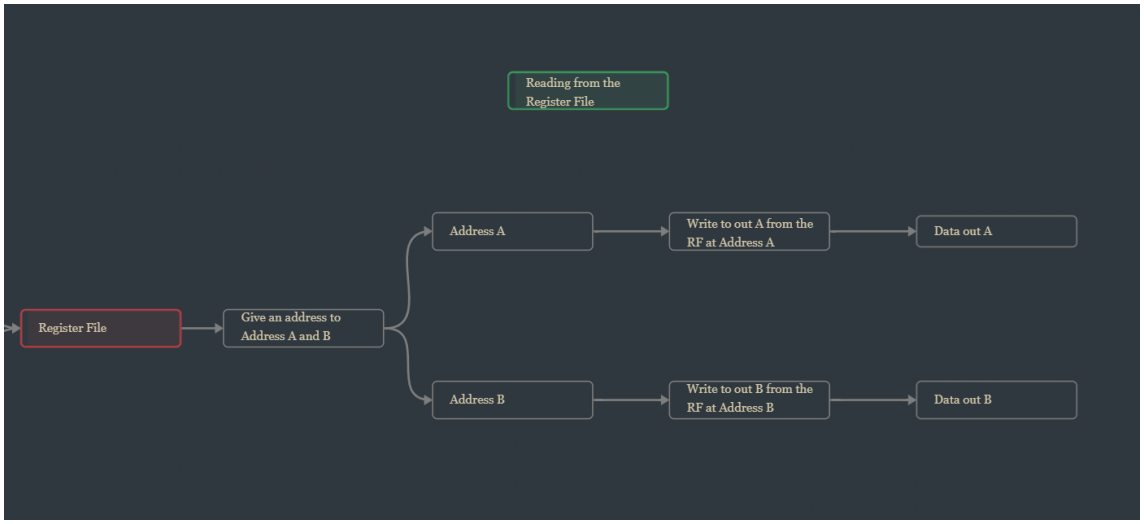

Figure 3: The order of operations when it comes to the reading of data from the RF

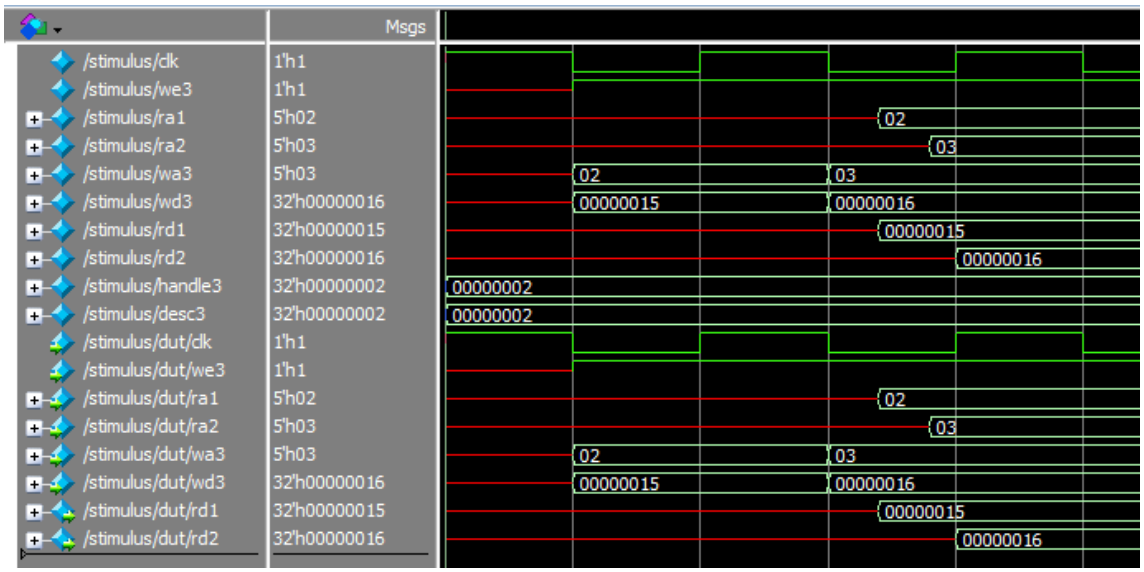

Figure 4: Nothing stored in the register as depicted by the red lines for rd1 and rd2