CMPE 260 Laboratory Exercise 2
Register File

Glenn Vodra
Performed: February 2, 2022
Submitted: February 16, 2022

Lab Section: 3
Instructor Sutradhar
TA: Ian Chasse
Cole Johnson
Robbie Riviere
Jonathan Russo
Colin Vo

Lecture Section: 1
Professor 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.

Abstract

The purpose of this exercise was to design a two-read, one-write register file. This register file can be configured for the desired number of registers, as well as the desired bit depth, or size of each register. This register file also has a write enable that is active high. This register file is designed to write on the falling clock edge, and read asynchronously. The first register, register zero, cannot be written to. In addition to this design file, a self-checking test bench was also written. The test bench contained a record of twenty-six tests that tried to read and write data to every register in the design. This design was simulated using two tools, Vivado and Modelsim. Each of these tools produced the expected results and therefore it was concluded that the exercise was successful.

Design Methodology

There are two aspects of the register file that were considered for this design. The first of these was the register module. This was generated using a data type called mem_type. Mem_type is a custom type that consists of an array of arrays. The first array contains two to the power of (log_port_depth). Log_port_depth is an integer that defines the number of bits available to address the registers. To test this design, a value of three was used. Two to the power of three provides eight addressable registers. The second array, contained within each of the eight addressable registers is the amount of data stored in each register. For this exercise, a value of eight was used, which gives each of the eight registers, eight bits of data. An instance of this type was then created and initialized to all zero;

The second aspect of the register was the register file functionality. The read functionality happens asynchronously so the data of the register can be assigned outside of a process. This is done by taking the address provided by the Addr1 and Addr2 signals and finding their data contents from the two-dimensional array described above. For the write functionality, a process was created that contained the steps needed. The first of these is to wait for the falling edge of the clock signal. The following logic is then executed if it is the falling edge. First check the one-bit signal WE (write enable, active high), and second check the address that is being written. The address, Addr3, must not be zero, because the zero register must always return the value zero. If all of these conditions are met, then the value of WD (write data) is put into the array at the address specified by Addr3.

Results and Analysis

To ensure this design works, twenty-six test cases were created to test every register in the design and ensure the zero registers could not be overwritten. The first two simulations, the behavioral and post-implementation were run in Xilinx's Vivado, while the third, was run in software outside of Vivado; this being ModelSim. The following figure, figure one shows the simulation results of the behavioral simulation.

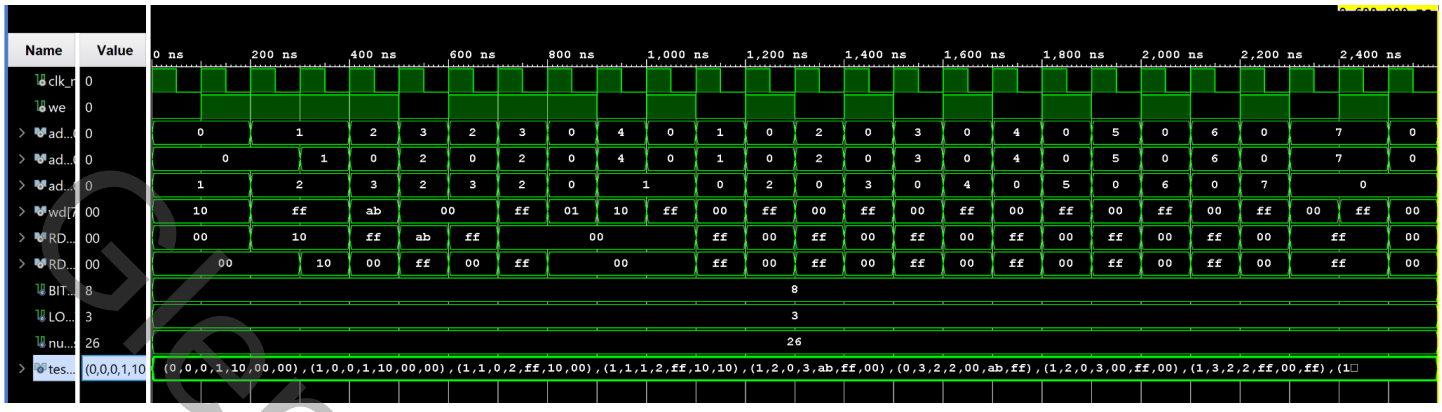

Figure 1: *Behavioral Simulation*

In order to ensure the proper functionality of this design. The contents of figure one can be examined closer. For each of the clock cycles a test is run on the design, with values being read asynchronously and written on the falling edge. The next figure, figure two, shows two valuable test cases that show both the proper functionality of the read/write, but also the immutability of the zero registers.

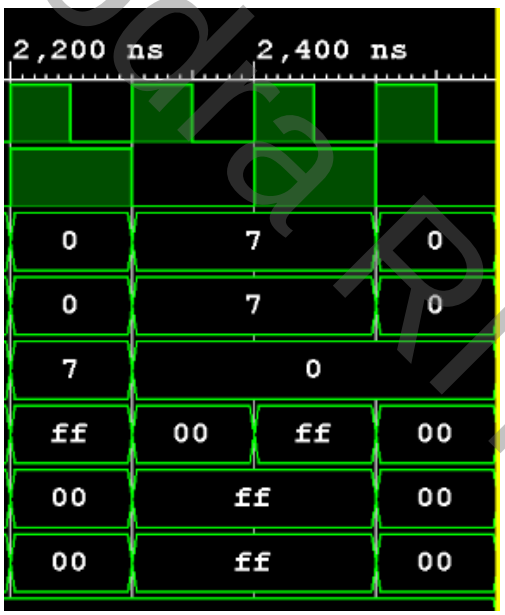

Figure 2: *Two Example Test Cases*

The first thing to note, in figure two, is the second signal, the one below the clock signal. This is the write enable. In this test, registers 0 and 0 are read from, which both return zero results, seen as the last two signals. In addition, the write enable is high, and the value “0xFF” is being written to register seven. In the following period, this write is confirmed by reading from register seven, through both ports, and seeing both return “0xFF”. In the next clock period, the write enable goes high. Register seven is read again, which is not important. During this time, the test bench attempts to write the value, “0xFF” to register zero. The last clock period, in figure two, shows register zero being read, after the attempted

write. As shown in the figure, the output shows “0x00” and “0x00”, meaning no data was written to that location.

Once the behavioral simulation was run, a Post-Implementation simulation was run. Figure three, below, shows the timing for this simulation using the same test cases found in figure one.

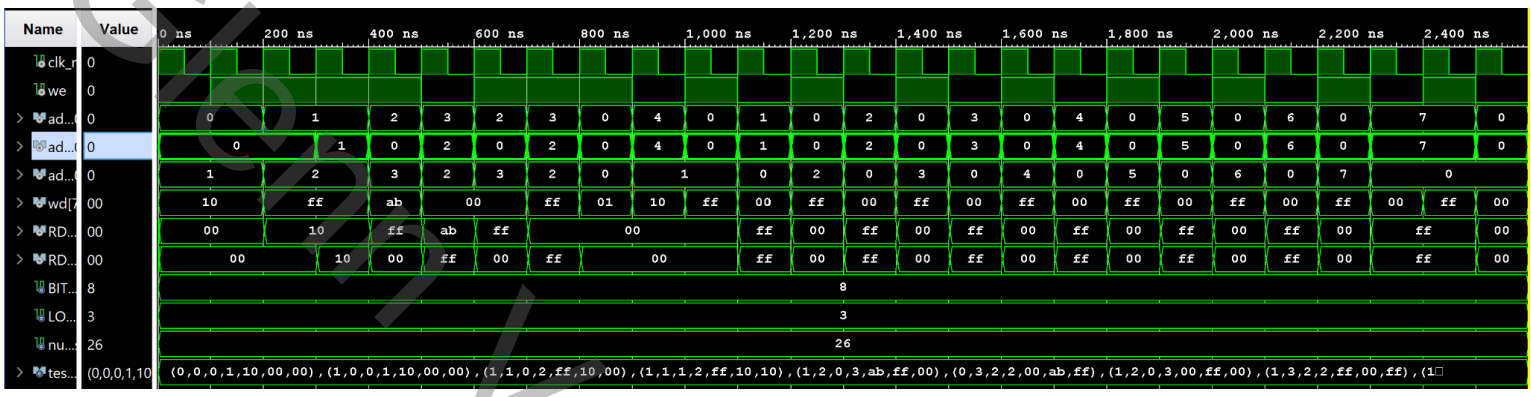

Figure 3: *Post-Implementation Simulation*

This simulation matches the waveform produced by the behavioral simulation, which is shown in figure one, above. The only difference, which is expected, is the propagation delay found as the signals are read. The register file design was also tested in Modelsim. Modelsim also produced results that were consistent with simulations run in Vivado, however, an examination of a test from this simulation will also be considered.

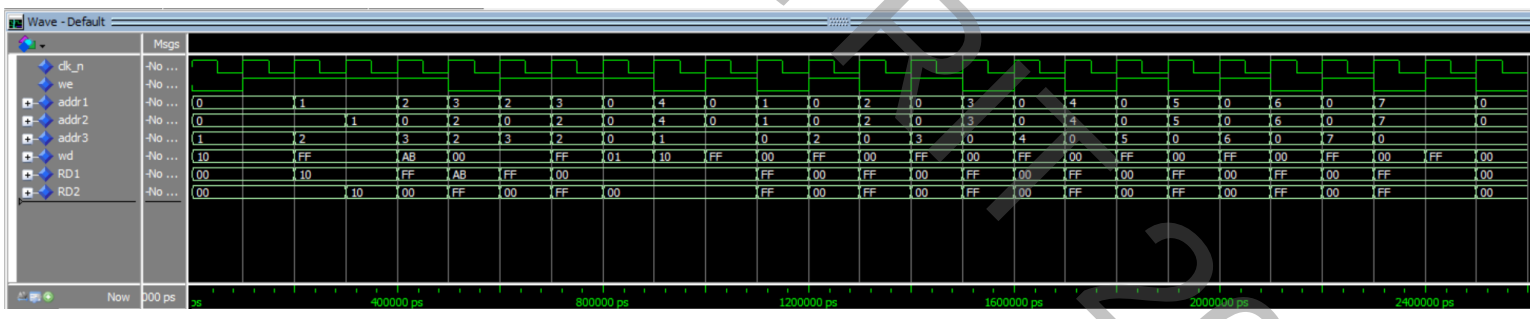

Figure 4: *Modelsim Simulation*

Note, that instead of using EDA playground, Modelsim was used. This was communicated to be an acceptable method of simulation for this exercise. To ensure the results are consistent with those from Vivado, a snippet of figure four is used.

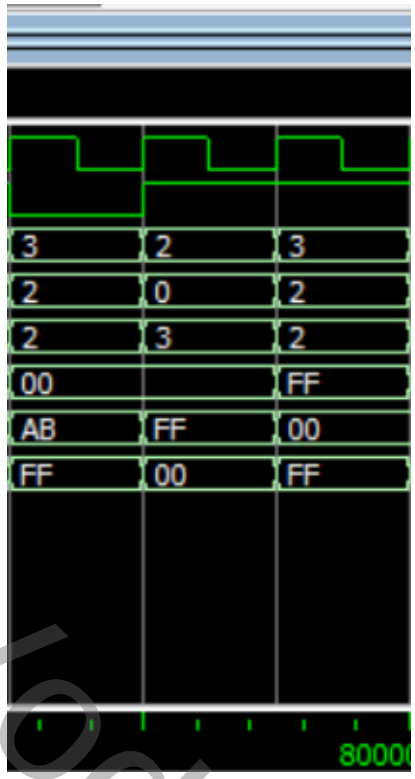

Figure 5: *Modelsim Example*

Figure five shows a different test, than the one shown in figure two, however, it produces consistent behavior for the operation of the register file. Note, the signals displayed are listed in the same order. In this test case, the write enable functionality is tested by trying to write the value “0x00” to register two when the write enable is low. In addition, the value stored in register three is shown as AB in the first period. In the next period, the value of register two is read. Register two still holds “0xFF”, which is the value from a previous write operation, and not the write that was attempted when the write functionality was not enabled. Also during this clock period write enable is set high and the value “0x00” is written to register three. In the next period, the value of two is checked again, and it is still “0xFF”. The value of register three however was changed, as the output is now “0x00”.

Conclusion

This exercise was successful because a register file design was successfully created, and tested using multiple forms of software simulators. Registers or even broader, addressable memory are very important for a computer system; they allow data to be stored and used for operations. Ensuring that data is being accessed from the correct location and read/stored at the correct time is very important as a processor is developed over multiple exercises. As this design is expanded, it is also important that this register design can be quickly changed to work with different data sizes and a different number of data storage locations (addressable registers in this case). As demonstrated in this exercise, the read-and-write operations work as intended, no data can be stored in register zero, and the write enable works, disabling and enabling write operations.