Lab 3 — 32 Bit Register File

Objectives

For this lab, I am creating a register file for a processor which uses RISC-V ISA. The register file will have 32 registers with 32-bit width.

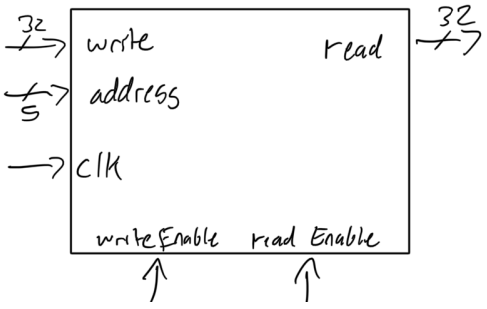
Introduction

To create this register file, I will also be designing a register that takes 32-bit width input data and gives an output of 32-bit width data, a decoder that takes a 5-bit input and decodes it to a 32-bit output, and a multiplexer that takes 32 input signals with 32-bit width and outputs one at 32-bit width. Each of these modules will be instantiated one or multiple times (in the case of the register) in order to create a functioning register file.

Methodology

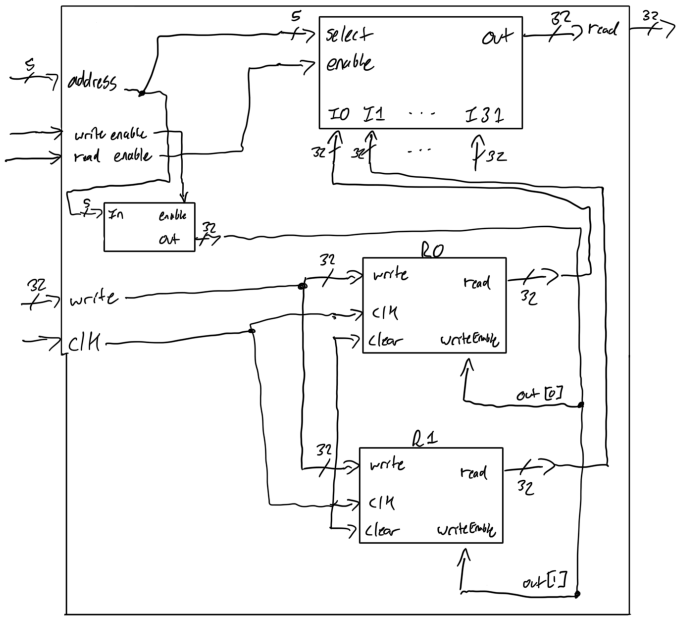
The following is the level -0 block diagram of my register file. It will have a write input with 32 bit width, a 5 bit input to select a register by address, a 32 bit read output to read data from, and a clk input. Additionally, a writeEnable input will write to the selected address when high, and a readEnable will read from the selected address when high.

Register File -0



Making up my register file are several instantiated modules which can be seen in the level -1 block diagram. These include a 5 to 32 bit decoder with it's outputs connected to the writeEnable input of each register. When an address is input, the writeEnable input on the registerFile will enable the decoder and the register associated with the address will have its writeEnable pulled high. This causes the write input to the register file to be written only to one register, the one corresponding to the input address. Each individual register's output is connected to one input of the multiplexer. When the register files readEnable is high, it pulls the enable input of the multiplexer high. Then, the address, connected to select on the multiplexer, connects the read output of the registerFile to the read output of the register corresponding to the input address.

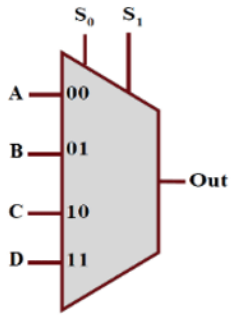
Register File -1



My decoder follows the pattern in this truth table from Javatpoint. When enable is low, all outputs are low. When enable is high, each unique input corresponds to a single unique output going high. No two outputs are ever high at the same time.

Enable	INPUTS			Outputs							
E	A_2	A_1	A_0	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	x	x	x	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	1	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0

My multiplexer behaves like this truth table from ResearchGate. For each select input, exactly one of the 32 bit inputs is connected to the single 32 bit output.



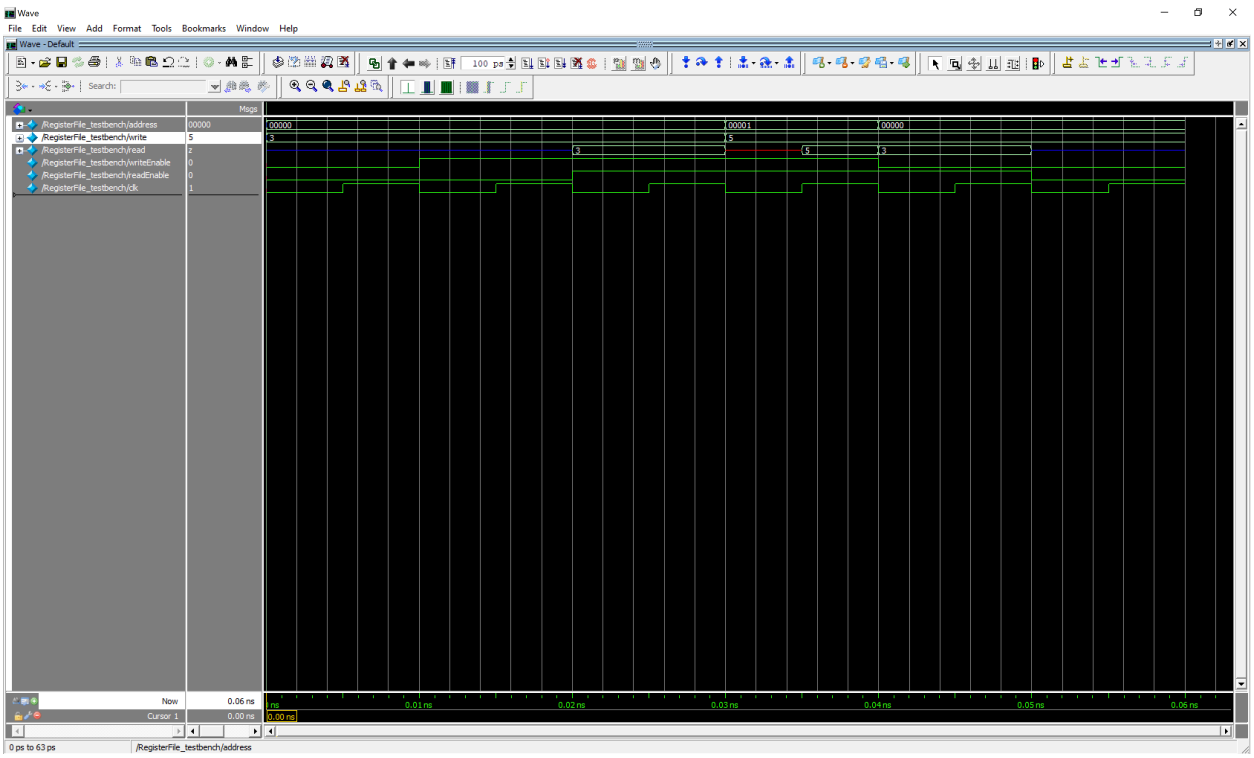
S_1	S_0	A	B	C	D	Out
0	0	0	x	x	x	0
0	0	1	x	x	x	1
0	1	x	0	x	x	0
0	1	x	1	x	x	1
1	0	x	x	0	x	0
1	0	x	x	1	x	1
1	1	x	x	x	0	0
1	1	x	x	x	1	1

Result and analysis

When tested in ModelSim, everything on the register file functions as expected. When writeEnable is low, nothing can be written to the file. Once it comes high, the write input is saved to the register at the input address. When read enable is high, the read output of the file gives the correct values.

All code, diagrams, and simulation results can be found as files on GitHub: https://github.com/Eth7an/Lab-3.git

A video explanation of the top level design and ModelSim simulation can be found here: https://youtu.be/3u-Fuy-8Cyo



Discussion and Conclusion

To conclude, using custom register, decoder, and multiplexer modules, I have built a fully working and tested 32 bit register file with 32 registers. The functionality of all inputs and outputs has been tested and verified, and this module is complete and ready to be instantiated in higher level projects.