Lab 4

Registers

In this lab you will use registers in a couple of applications. The first is a register file and the second is an accumulator. You will test both applications on the FPGA board.

Part 1 (Building a register file)

Processors contain an array of registers called a register file. They are usually implemented using RAM. However, in this part, you will build a register file using regular registers with flip-flops.

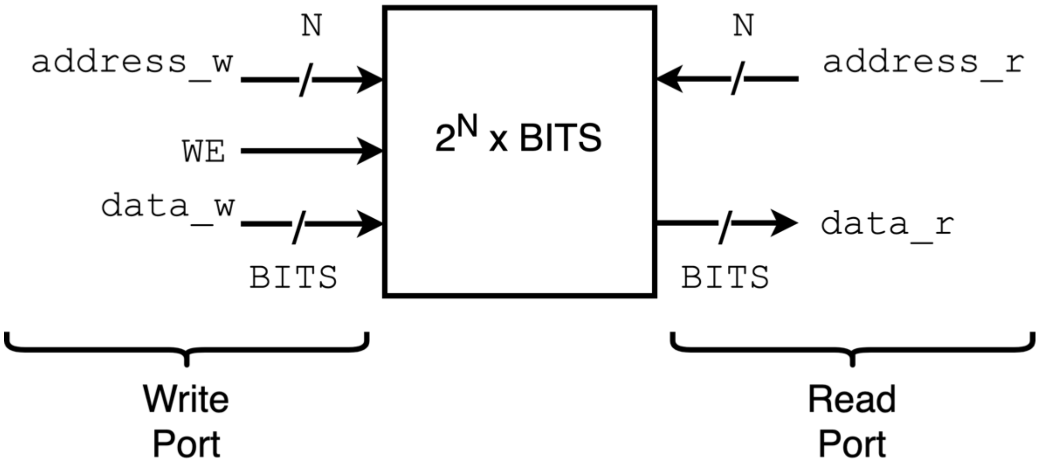

Figure 1: Register File

Figure 1 shows a register file that contains 2^N registers (rows) and each row is of size BITS (columns). This particular implementation utilizes two ports, a write port and a read port. The dual ports allow a user to access the register file for reading and writing simultaneously.

You can write a new word into the register file by placing it at `data_w` and specifing the write address on `address_w`, asserting `WE` will then invoke the write operation. A similar process can be used for reading a word from the register file, place an address on `address_r` and `data_r` will have the word stored at that address.

Generic Register File

1. Build a generic register file (call it `reg_file`) that takes two parameters `N` and `BITS`.
2. The generic register file should utilize 2^N registers and each of the registers should be of size BITS.
3. The write port can be implemented using an $N\text{x}2^N$ binary decoder.

4. The read port can be implemented using an Nx2^N binary decoder and a 2^N Tri-State Buffers.
5. Use the following provided code:
    a. `simple_register_load.v` for the base registers
    b. `decoder_generic.v` whenever you need a decoder
    c. You need to figure out how a tri-state buffer works. *Hint*: the following implements a tri-state buffer of input a and output b
    `assign b = (enable) ? a : 1'bz;`
6. Include a schematic diagram of your design with your submission

Register File Application

Verify the functionality of the register file by implementing it on the FPGA board:

1. Place your implementation in a new file called `reg_file_application.v`
2. Instantiate a version of the register file that contains 128 rows with 4 bits each.
3. SW3 ← SW0 should be connected to `data_w`
4. SW10 ← SW4 should be used to specify `address_w` or `address_r`. There isn't enough switches on the board to separate the addresses, so you will have to demultiplex them
5. SW15 should be used to specify which address is being specified on SW10 ←SW4. For example if SW15 is 1'b1 then the adderss at SW10←SW4 is the read address
6. The down push button should be connected to `WE`
7. The `data_r` should be displayed on one of the seven segment digits. Use the provided `hex2sseg.v`

<u>Question:</u> How many Flip-Flops are being used on the board? Look at the utilizing report to figure out the answer.

Part 2 (Building an Accumulator)

Many processors contain a special function register called an accumulator (you might have seen it called a working register). The register stores the result of the last operation and might also be used in following operations.

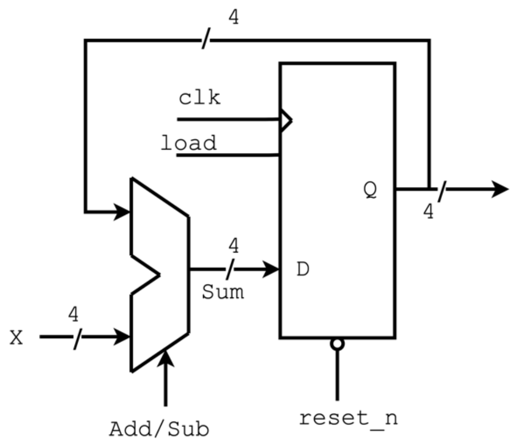

Figure 2: 4-bit accumulator

Figure 2 shows a diagram of a simple accumulator that can store and accumulate the result from an adder/subtractor. Build this accumulator and test it on the FPGA board using the following specifications:

- User a 4-bit register
- Use a 4-bit adder/subtractor. You can use the provided `adder_subtractor.v`
- SW3→SW0 should be connected to input `X`
- SW15 should be connected to the `Add/Sub` input
- `load` should be connected to the down push button
- `reset_n` should be connected to the reset button
- You should display the value stored in the accumulator on one of the seven segment digits

Submission check list:

[ ] All Verilog code you generated or modified
[ ] All testbenches written
[ ] Embed all screenshot of your testbench output in your README.md
[ ] Embed all block diagram generated in your README.md
[ ] Short videos demonstrating each of the parts you implemented on the FPGA