Exercise 1: Testing the Waters

Overview

In this exercise, we will build a 4 input multiplexer from a 2 input multiplexer. This should be fairly straightforward from a design standpoint and is meant to get you familiar with a basic subset of Verilog as well as make sure you have a functional simulation environment.

Mechanical Goals

By the end of going through this exercise you should:

- Have a working simulation environment
- Be comfortable defining a module
- Be comfortable instantiating a module
- Be comfortable declaring wires and connecting modules with wires

Conceptual Goals

As this assignment is mostly mechanical, there's not a specific conceptual goal. The primary goal of this assignment, which should be in the back of your mind in general is that **Verilog describes hardware**. Even though it can feel at some times like you are just writing code, you should remember that this is not software and at the end of the day, you should have a rough idea of how the Verilog you write is going to translate into hardware structures.

Verilog Resources

There are many guides on (System)Verilog syntax that explain how to write Verilog. However, not every feature that these guides introduce are compatible with all tools and some features are not synthesizable (i.e. they will not work when you run the code through ASIC/FPGA tools to produce actual hardware). Rather than try and reproduce the guides, here are some links to guides that I would recommend:

- ChipVerify is a good place to start for overall syntax and figuring out what you can do with the language, but don't skip the other guides.
- This Github page describes the specific structures you should be using to write synthesizable (System)Verilog
- This is fairly strict, but comprehensive style guide used by an open-source RISC-V core. I don't follow it strictly, but I like a lot of it.

Steps

Step 0: Install Verilator and GTKwave
Follow the instructions on the Verilator install page. I recommend installing from source rather than using your distribution's package manager. These exercises were tested using version 4.210, so try to install at least that version or newer.

GTKWave installed from the package manager as of Ubuntu 20.04 works. The version in the package manager is 3.3.103. Your mileage may vary depending on your package manager.

Step 1: Define the module interface
Write the definition of the module interface for the 4 input mux. This should look a lot like the 2 input mux interface.

Step 2: Fill in the module logic
Using just the 2 input multiplexer module and wires (you can use wire indexing), write the logic for the 4 input mux module. You shouldn't write any logic operations yourself, just instantiate modules and connect them.

Step 3: Build the Verilog design
Just run `make build` in the `exercise_1` directory.

Step 4: Run the Verilog design
Just run `make run` in the `exercise_1` directory. If everything passes, you should get the output:

```
-- RUN ---------------------
obj_dir/Vmux_sim_top +trace
[1] Tracing to logs/vlt_dump.vcd...

[1] Model running...

[1] data_sel=0 data_out=1
[2] data_sel=1 data_out=2
[3] data_sel=2 data_out=3
[4] data_sel=3 data_out=4
[5] data_sel=3 data_out=8

-- DONE --------------------
To see waveforms, open vlt_dump.vcd in a waveform viewer
```

In order to debug, you can view the waveform generated by the tests in `logs/vlt_dump.vcd`. You can also add more tests into `sim_main.cpp` where indicated by copying the tests in there and modifying where appropriate.