Exercise 4: Register File

This exercise guides you through the development of a register file.

Task 1. The implementation of a register file requires additional numerical functionality. These functions will be provided using a package. Add a package (`misc.vhd`) to your register-file-project and implement a $\log_2$ function called `log2`. Your implementation should satisfy the following requirements.

- Input $x \in \mathbb{N}_0$: a positive value
- Output $y \in \mathbb{N}$: a natural value
- Function: $y = \log_2 x$

Task 2. Create a VHDL description of a register file (entity name `register_file`). The following files give you support in your endavour:

- `register_file_tb.vhd` (Simulation Source)

Your design should satisfy the following description.

- There are n registers.
- Each register has a width of b bit.
- Two registers can be read simultaneously.
- Additionally, one register at a time can be written to.
- All registers share the same clock.

Use the following inputs:

- `addr_a`, `addr_b`, `addr_c` for addresses
- `data_a`, `data_b`, `data_c` for data ports
- `w_en` for a write enable
- `clk` for the clock signal

Finally, verify the design using the provided testbench. If everything works correctly, you see fibonacci numbers written to and read from the registers.

Task 3. Recall the lecture, there are different components of the FPGA the register file can be mapped to. Set your synthesis settings to `-mode out_of_context` and synthezise and implement your design. What is the register file mapped to in your implementation?

Task 4. We want to measure the critical path of the register file. Therefore, we wrap its inputs and outputs in registers. Remember, the critical path is the longest combinational path between two registers. By adding registers, we ensure that all combinational paths of our design are terminated by registers. For this, include `register_file_timing_wrapper.vhd` in your design and ensure it is selected as the top entity. Then run synthesis and implementation. What is the critical path, and what is the maximal frequency you can run your register file at?

Bonus Task 1. Program the FPGA and with the provided wrapper entity and your register file to see it working in action. For this, remove the `out_of_context` setting. Then, add the follwing sources to the project.

- `register_file_wrapper.vhd` (Design Source)
- `register_file_constraints.xcd` (Constraint)

If necessary, adjust the entity instantiation in the wrapper to fit your design. The wrapper implements a simple state machine which allows you to write values or the sum of two registers to a register. You can find an overview over the implemented states in Figure 1. Additional information is given in Table 1.

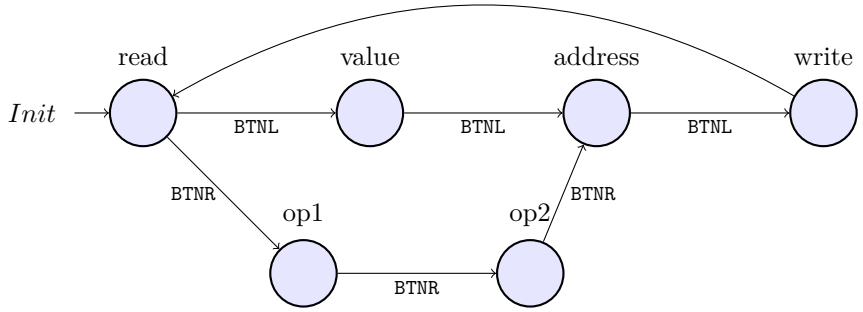

Figure 1: Incomplete overview over the state machine implemented in `register_file_wrapper.vhd`.

Table 1: Description of states

State	Description
read	Switches set address, LEDs show the value stored at register(address).
value	Set the value to be stored using switches.
address	Set the address of the register to be written to using switches.
op1	Set the address of the first operand of the sum-operation.
op2	Set the address of the second operand of the sum-operation.
write	The designated value is written to the register(address). The next state is automatically read.