CAD 5: Register File

EE 476 | University of Washington

Notes

1. **For this and all future projects**:
    - Physically connect all signals, including `VDD!` and `VSS!` (i.e., students should not use the "connect by name" feature). Interconnect is one of the most important aspects of physical design, so it is important that students understand the challenges and constraints related to signal routing. Simulation results obtained without interconnect tend to be meaningless.
    - Use 1.2V for `VDD!`
    - Design all circuits to be static CMOS with full-rail switching (unless discussed beforehand with the instructor).
    - Follow metal directions strictly. Odd-numbered metals (e.g., M3) run vertically, even-numbered metals (e.g., M2) run horizontally, and M1 is in any direction.

Setup

Use the same general directory structure as with CAD's 1-4. Review *Tutorial 1* for details.

Project Overview

The vast majority of processor architectures utilize at least one small, fast memory module called a *register file* (also called a *scratchpad* in some contexts). In this project, students will build a 16-bit, 13-entry register file with 1 write port and 2 read ports (Figure 1). There is also a Verilog model that is available for reference. Teams are free to choose their implementation (e.g., 16x13 master-slave flip-flops, 16x1 master latches coupled with 16x13 slave latches, 16x13 SRAM array, etc), so long as it is compatible with the functional specification.

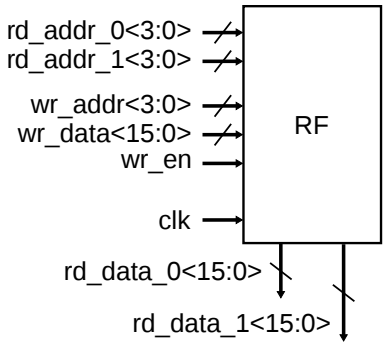

Figure 1: Register file interface

Given a write address and write enable triggered by a positive edge at cycle T, the updated register contents are available for read at cycle $T+1$ (Figure 2). Reads themselves are asynchronous, i.e., there are no registers for storing read addresses, and the latency between read-address updates and read outputs is purely combinational.

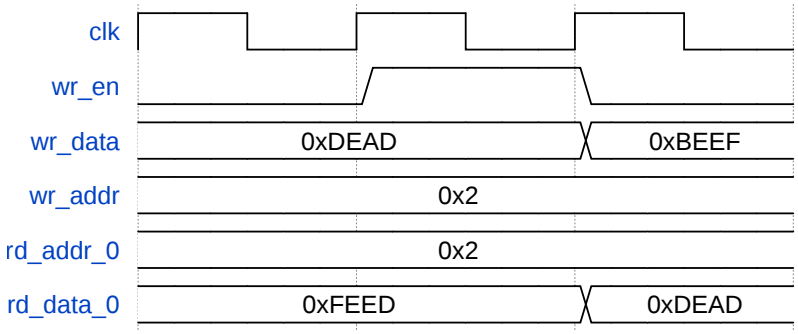

Figure 2: Register file read/write timing

Keep in mind that the read and write ports of the register file should ideally be width-matched to the other modules in the processor, such as in the datapath. For instance, if the bit-cells in the ALU (next CAD) are much larger than those in the register file, it will be very difficult to route and floorplan the module interfaces effectively (Figure 3). To this effect, students are required to use a bit-cell width of 5.0 um.

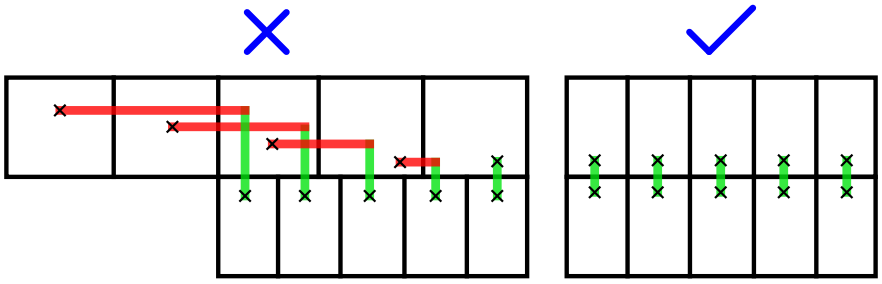

Figure 3: Having modules that are matched in bit-cell width/height improves routing and general layout quality by reducing routing congestion and interconnect length. The example above illustrates a simple case with two hypothetical non-width-matched (left) and width-matched (right) module interfaces (crosses denote pin connections).

1 Register File Schematic

a. Create a schematic for the register file in Figure 1, and name the cell `RF`. For sizing and testing, we should keep in mind that the register file outputs will be connected to long wires, with large fanout; hence, approximate the wiring and transistor capacitance with 40fF loads on all outputs. In addition, buffer all inputs with a schematic-level buffer composed of two inverters, both of size 0.4um/0.3um (Remember: inverter sizing is written as W_pmos/W_nmos). The measurements below should be taken with the 40fF loads and input buffers, but remember that *submitted netlists should always be unloaded and unbuffered.*

For verification, it is recommended that students use HSPICE vector files to simplify testing.

Note: it is very common for naive RF implementations to contain race conditions in the latches. This will show up as writes being executed on both the positive edge and the *negative* edge of the clock. Be sure to design with this in mind and verify through testing that your register file does not have these race conditions.

b. Collect the measurements in Table 1 for the `RF` schematic, using the loads and buffers described above. Use a clock period of **1ns** (this should be sufficient for even the slowest register files), and average the measurements over at least 5 clock periods. Note that the autograder uses these exact test parameters, so it is important for students to use the same parameters to minimize discrepancies.

Measurement	Description
`en_deliv_read`	Average energy delivered by the supply for read operations, assuming random read inputs, and stable write inputs with `wr_en`= 0
`en_deliv_write`	Average energy delivered by the supply for write operations, assuming random write inputs (except for `wr_en`, which is logic '1') and stable read inputs

Table 1

Delivery

a. The schematic netlist for `RF`

b. The measurements listed in Table 1

2 Register File Layout

a. Create the layout for `RF`, using `C+CC` for parasitics extraction. Make sure to leverage good design hierarchy, floorplan carefully, and route upper-level metals using tracks.

Note: Because the layout will contain lots of long traces, simulation results are likely to be notably different than in the schematic (especially, for example, the clock propagation). As such, it's a good idea to re-run all your schematic-level testing from Part 1 at the layout

level. A good layout is worthless if it does not perform correctly, and implementations that suffer from incorrect behavior *will* lose points.

b. Collect the same measurements from Part 1 in Table 1, but for the post-layout `RF`. Use the same test conditions (i.e., loading and input drive) as in Part 1.

Delivery

a. The netlist for the post-layout `RF` extracted with `C+CC`, and DRC/LVS reports

b. The measurements from Table 1, taken with the post-layout design

File Submission

There is no report for this project. Measurements should be submitted using the provided `specifications.json` file, and files should be placed in the specified locations (see the *CAD Submission and Grading* document for details).