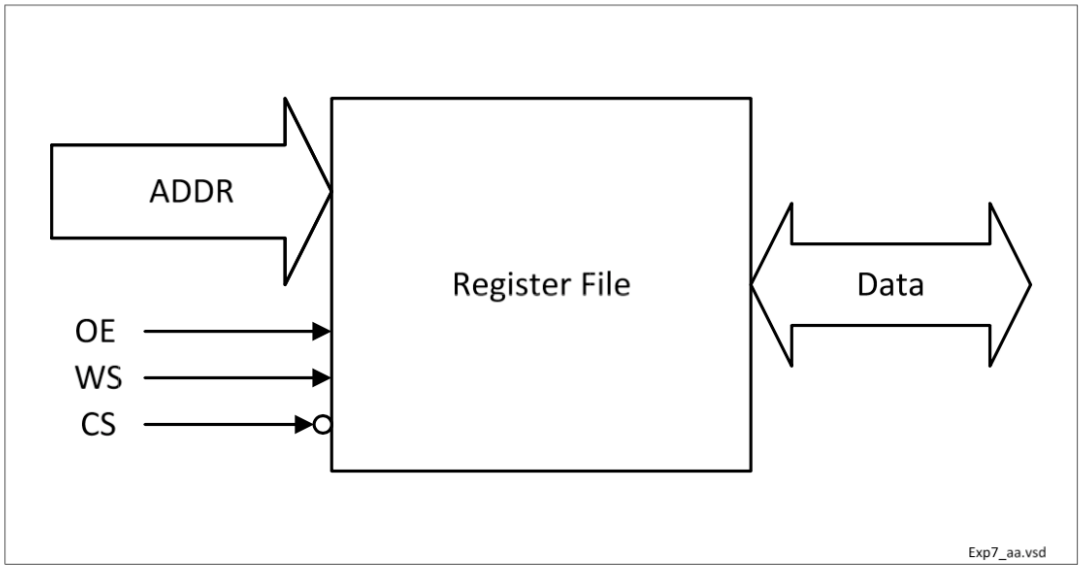
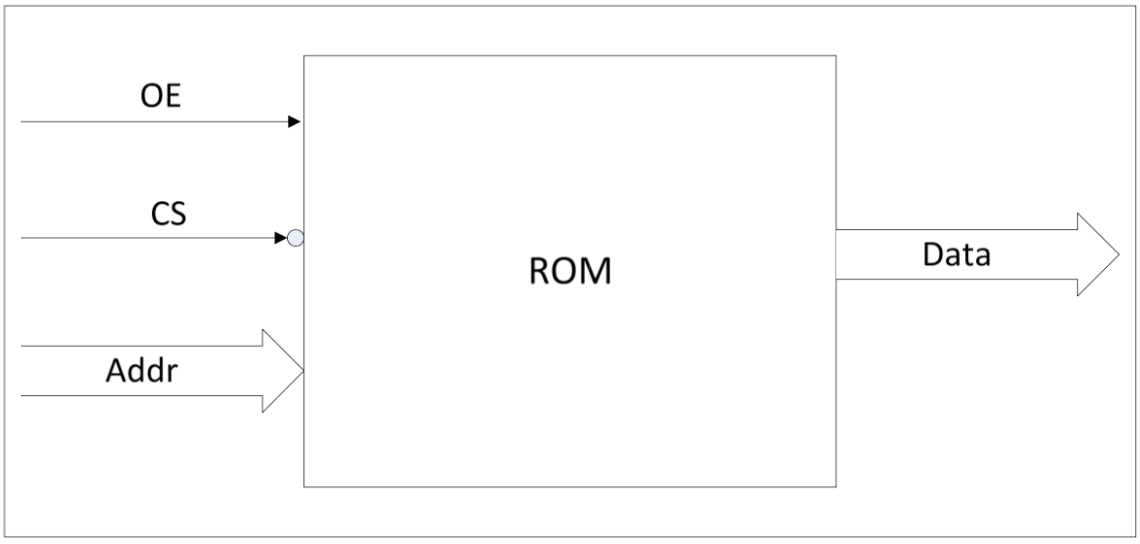
# ECE 526L Spring 2021

# Lab 7

## Experiment #7, Register File Models

A register file operates similarly to random access memory but is made from flipflops rather than DRAM or SRAM cells. A register file would consume far more power and take far more area than a true memory of the same capacity. However, register files have the advantages of operating fast, faster even than SRAM, and they can easily be coded in any arbitrary size (width and depth) in Verilog. In this lab, you will create a register file that will be used as a random access memory.


ADDR
Register File
Data
OE
WS
CS
Exp7_aa.vsd


OE
CS
ROM
Data
Addr

You will create a second model that has no write capability to be used as a Read Only Memory (ROM). Verilog system tasks $readmemb and/or $readmemh will be used to set values in the ROM model.

Use parameters for both width and depth in your models. Where specific sizes are given in the description below, make those the default values for your parameters.

1. For the RAM, create a Verilog model of a register file with the following specifications:

a. The memory has an eight bit bi-directional data bus.
b. The memory has a five bit address bus.
c. The *output enable (oe)* signal is active high.
d. Chip select (*cs*) is active low.
e. The value on the data bus is written to the address on the address bus on the rising edge of the write strobe (*ws)* if the output is not enabled.
f. The contents of the currently specified address are placed on the data bus when *oe* is high.
g. Block read: as long as *oe* remains high, the contents of the newly specified address are placed on the data bus.
h. The data bus returns to the high impedance state after *oe* goes low.
i. The chip select signal, *cs*, must be low to read or write. If it's high, *oe* and
j. *ws* are ignored and the data bus remains in the high impedance state.

Assume the address bus is stable before or at the assertion of *oe* or *ws* and remains stable for at least the access time of the memory. Assume *cs* will never be changed during a read or write cycle.

The ROM will be similar but it can only be read. Therefor it does not need a write strobe or any way to write new data to it. This implies that the data port is an output, not a bidirectional port.

2. Create a Verilog testbench for your register file. It must do the following:

A. Write to and read from every memory location. Use sequential numbering to write 0 to address 0, 1 to address 1, etc.

B. Demonstrate both an individual and a block read.

C. Verify both enabled and disabled states

D. Demonstrate that the data bus is in the high impedance state at some point.

E. Output a Walking Ones or similar test to demonstrate that all output bits are capable of operating independently. A Walking Ones pattern for four bits is shown below.

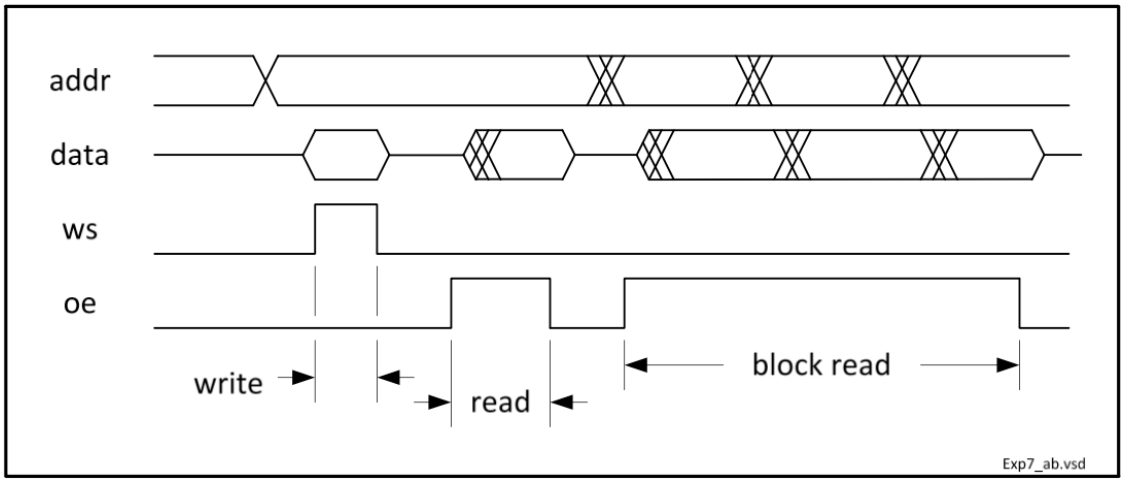
Address 0: 0001

Address 1: 0010

Address 2: 0100

Address 3: 1000

Note: some of these requirements can be combined in a single test.



3. Create a second Verilog testbench that does the following:

A. initializes the ROM to the following hex values using the **$readmemh** system task:

addresses 5'h04 to 5'h0F: 58 ED B7 34 C9 8F A0 9B 65 11 03 4C

addresses 5'h10 to 5'h17: DA 7E F2 26 86 95 FD B1

addresses 5'h1C to 5'h1E: 12 AF 33

B. demonstrates the memory has been successfully initialized.

C. demonstrates that unspecified locations remain undefined.

D. reads the ROM, scrambles each byte, and writes the new byte into RAM. The scrambling algorithm is to change the bit order of each byte from [7654 3210] to [0716 2534]. *e.g.* 8'hDA becomes 8'h73.

E. demonstrates the memory bytes have been successfully scrambled.

Use a task to print the contents of all memory locations. Waveform printouts must be detailed enough to verify bus timing and content of address and data buses (zoom in until numbers are readable). Include a printout of your **$readmemh** data file.

Lab report question: How many edge constructs do you use in your models? Why that is the best answer for this design?