ECEN 651: Microprogrammed Control of Digital Systems
Department of Electrical and Computer Engineering
Texas A&M University

Prof. Mi Lu
TA: Ye Wang

Laboratory Exercise #3

Storage Elements in Verilog

Objective

The purpose of lab this week is to familiarize you with a couple different types of storage elements that you will use in your MIPS microarchitecture. In this lab, you will describe those storage elements in Verilog and test them using provided testbenches.

Background

- The register file provides high speed storage of data within the MIPS processor and is at the heart of any Reduced Instruction Set Computer (RISC). Figure 1 depicts the register file that we will create for our MIPS processor. In the case of RISC, the instructions directly address the register file, instructing the machine where within the register file to pull the operands from and where to store the result. For this reason, RISC is often referred to as a register-to-register machine. A register file is typically constructed from flip-flops or, in the case of a larger processor or an FPGA implementation, SRAM cells and uses multiplexers and decoders to provide a means for reading and writing. Register files are typically architected for a reduced access latency and concurrent reads and writes, which implies that they are small (on the order of 8 to 128 registers deep). Figure 1 shows the MIPS register file to be 32 registers deep and 32-bits wide. Also shown is the port interface specifically for the MIPS architecture. Notice that there are two read ports and one write port (i.e. BusA, BusB and BusW) and the appropriate address buses, Ra, Rb, and Rw, correspond to the rs, rt, and rd fields in the arithmetic

and logic instructions within the MIPS ISA. For our implementation of MIPS, the register file supports asynchronous reads and synchronous writes. RegWr is Figure 1 is an active high write enable signal.

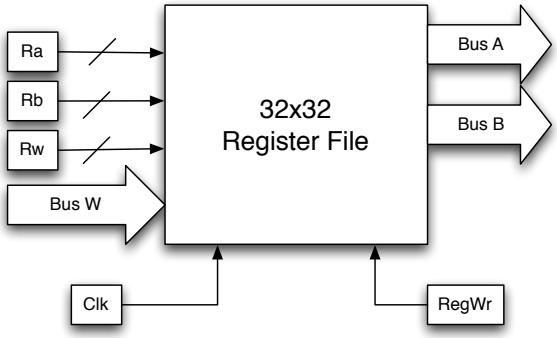

Figure 1: The port interface for the MIPS Register File

- The data memory provides the microprocessor with a means of storing more long term data. The data memory, in comparison to the register file, is typically much larger in size (i.e. in the kB to MB range) but does not provide as much parallel access, meaning it has less data ports. Figure 2 shows the data memory module that you will describe in Verilog for the MIPS processor implementation. For our implementation, the data memory will provide synchronous read and synchronous write capabilities and include both a read and write enable signal. Access to the memory requires only a single clock cycle, which simplifies the overall design of our microprocessor, but limits the size of memory we can instantiate due to timing constraints. Also note that the storage element in Figure 2 includes only one address port, which means data cannot be written while being read and only one data element can be retrieved at a time.

Procedure

1. Design and simulate a 32-by-32 register file.

   (a) Using Figure 1 as a reference, describe a register file in behavioral Verilog to the following specifications:

   - The register file is 32 registers deep by 32-bits wide.

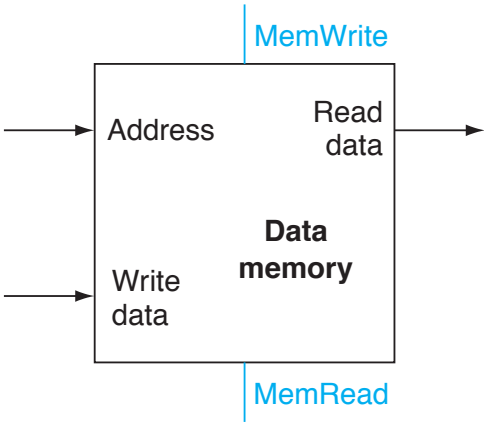

Figure 2: The Data Memory for the MIPS processor

- It has two 32-bit wide asynchronous read ports, Bus A and Bus B.
- The read ports are word addressable (address 32-bits at a time) by Ra and Rb.
  Note: Since there are 32 words to be addressed, each address bus needs to be 5-bits wide!
- The 0th register must be **wired** to the value 0.
- Registers 1 through 31 must be synchronously writable (triggered on the negative edge of the clock) when the RegWr signal is high.
  Note: Writes to register 0 should be ignored!
- The write port is word addressable by Rw.
- The Verilog for the register **must** be synthesizable.
- Use the following port interface:

```
RegisterFile(BusA, BusB, BusW, RA, RB, RW, RegWr, Clk) ;
```

(b) Use the code provided on the lab website, `http://ece.tamu.edu/~atarghe1/ECEN651lab/`, to test your behavioral model of the register file. Demonstrate your progress to the TA.

(c) In Xilinx, synthesize your register file with the following device properties:

Set the device properties to the following:
Device Family: Virtex5
Device: XC5VLX110T
Package: ff1136
Speed Grade: -1

(d) Ensure your design incurs no warnings or errors and demonstrate your progress to the TA.
Note: Do not attempt to synthesize the testbench.

2. Design and simulate a **simple** data memory module.

(a) Using Figure 2 as a reference, describe a data memory module in behavioral Verilog to the following specifications:

- The data memory must hold 256 bytes. We will expand this later on.
- Reads and writes must be synchronous on the positive and negative clock edge respectively.
- The memory module must use the read and write enable signals, i.e. MemRead and MemWrite. In the case of a read, the read data bus should retain its previous value when the MemRead signal is low.
- The memory has one address bus shared for both reading and writing and is word addressable.
  Note: The data memory within the MIPS processor is actually byte addressable, but we will fix this later.
- The read and write buses are each 32-bits wide.
- The Verilog for the memory module **must** be synthesizable.
- Use the following port interface:

```
DataMemory(ReadData, Address, WriteData,
           MemoryRead, MemoryWrite, Clock);
```

(b) Use the code provided on the lab website to test your behavioral model of the memory module. Demonstrate your progress to the TA.

(c) In Xilinx, synthesize your memory module with the aforementioned device properties.

(d) Ensure your design incurs no warnings or errors and demonstrate your progress to the TA. *Note: Do not attempt to synthesize the testbench.*

1 Deliverables

1. Submit a lab report that captures your efforts in lab.

2. Include all Verilog source files with appropriate comments.

   Note: Code submitted without adequate comments will not be graded!

3. Be sure to include all material requested in lab.

4. Answer the following review questions:

   (a) Suppose we could make the data memory module dual-ported (i.e. two separately addressed read ports). Could one design a microarchitecture which uses the dual-ported memory module in order to eliminate the register file? If so, would this be a good design? Explain your answer.

   (b) What is the purpose of the data memory's MemRead, (i.e. the read enable signal)? Is it functionally necessary? What advantages might it provide? Why do we not implement a similar signal for the register file?

   (c) Elaborate on the term *synthesizable*. What sort of constructs in Verilog are not synthesizable?