CoE 113 Machine Exercise 2

Automated Testing and Register File

1 Introduction

This machine exercise is composed of two parts. For the first part, you will be introduced to the use of memory in automated testing. Familiarizing yourself with this test methodology will make it easier for you to test processor designs in the future. For the second part, you will be creating a register file for the MIPS ISA, which will then be used in your implementation of the whole MIPS processor on the next machine exercises.

2 Automated Testing using Memory

In the context of testing, memory models may be used within Verilog testbench code. This memory is considered part of the testbench and not of the design that we wish to verify; thus, the memory model is usually not synthesized.

2.1 Displaying Memory Contents in the Testbench

1. Untar the *CoE113_ME2.tar* file uploaded on the class webpage and examine the files *mem.v* and *tb_mem.v*. The top-level testbench module is contained in *tb_mem.v* while the memory model, which is instantiated in *tb_mem.v*, is contained in the file *mem.v*.

2. The memory model has one read and one write port, addressable by a 32-bit address (byte-addressable). Data is read or written with a word size of 4 bytes. Writes are performed by asserting the active-high `wr_en` signal.

3. Initial contents of the memory (i.e., at the start of the simulation) can be found in the text file *mem.txt*, as specified by the use of the `$readmemh` directive in the memory model. This directive is executed at runtime (not compile time), so the contents of the memory may be changed without the need for recompilation of the Verilog file.

4. Compile and run the testbench (*tb mem.v*), making sure that you include all dependencies. For the initial code, the only dependency is *mem.v*. The text file referenced by the `$readmemh` directive is not a dependency because it is not a Verilog file. Observe the output.

5. Modify the testbench such that it outputs all five words contained in memory as specified by *mem.txt*. **Show the simulation result and your modified code to your instructor**.

2.2 Test Environment

Automated testing is achieved by using memory to generate test inputs for the system. Figure 1 shows an example block diagram of a finite state machine (FSM), which serves as the unit under test (UUT), interfaced to a memory model for testing purposes. A particular set of inputs is associated with an address, which is provided by the testbench or, in this case, the UUT itself.

1. Examine the file *fsm.v* from the tarball extracted in the previous subsection.

2. Modify the testbench again to implement the test environment shown in Figure 1.

3. The `fsm` block also contains an active-high output, `done`. Use this signal to determine whether the automated test has completed. Modify the testbench such that it prints (to the console) the memory contents at the start of the automated test (as with the previous section). Also print the contents of the memory after the automated test has completed. The automated test is complete when the output `done` of the `fsm` block is asserted. Show the simulation result and your modified code to your instructor.

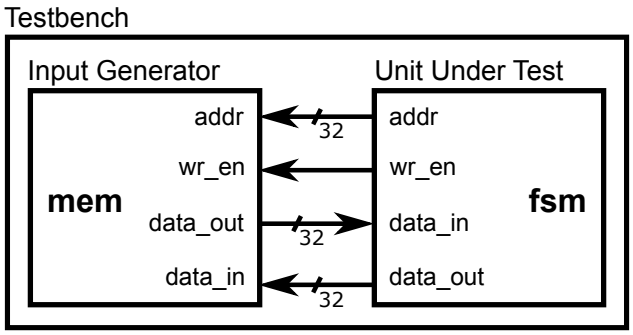

Figure 1: Testbench block diagram.

3 Register File

Implement a register file that contains 32 32-bit registers. Assume that the first register (address 0) contains the constant 0 and cannot be written (i.e., when a write is performed at this register, nothing happens). The following subsections discuss details on the implementation.

3.1 Ports

The register file has two read ports and a single write port. Each port is associated with a 5-bit address and a 32-bit data. Table 1 lists all ports and their names, along with other pertinent information. The reset signal, `nrst`, is a global active-low asynchronous reset; that is, when `nrst` is low, the register file is reset. When the register file is reset, all contents of the register file are set to 0.

Table 1: Register file ports.

Port name	Direction	Width	Description
`clk`	in	1	global clock
`nrst`	in	1	active-low reset
`rd_addrA`	in	5	read port A address
`rd_addrB`	in	5	read port B address
`rd_dataA`	out	32	read port A data
`rd_dataB`	out	32	read port B data
`wr_en`	in	1	active high write port enable
`wr_addr`	in	5	write port address
`wr_data`	in	32	write port data

3.2 Read Operation

Reading is done asynchronously by providing the read address. Contents of the register addressed by the provided read address are asynchronously (no need to wait for the next clock rising edge) forwarded to the corresponding port.

3.3 Write Operation

Writing is done in a synchronous manner. Data to be written is set in the write data port, along with the destination register's address. The write enable signal is then asserted in the same clock cycle as the address and data are set. Data is successfully written to the destination register at the next rising edge of the global clock signal. When the write enable signal is deasserted, contents of the register file are not modified.

3.4 Synthesis

You are required to synthesize your design using the generic 90nm standard cell library provided in class. Name your RTL file as *rf.v*, your synthesized (mapped) file as *rf_mapped.v*, and your SDF file for the synthesized (mapped) file as *rf_mapped.sdf*.

4 Submission and Deliverables

Submissions will be done through UVLê. Deliverables for this exercise include: (1) modified testbench from part 1, (2) source code for the RTL model of the register file, (3) source code for the synthesized (mapped) model of the register file, and (4) SDF file for the synthesized (mapped) model of the register file. Place all required files in a single archived *zip*, *tar*, or *tar.gz* file. Make sure that all the files within the archive are located in a flat structure, with no subdirectories/folders containing the required files. The filename format for the archive should be *CoE113_ME2_<Surname>_<Initial_of_given_name>.<zip*, *tar*, or *tar.gz>*. For example, *CoE113_ME2_Agaton_M.zip* or *CoE113_ME2_Galapon_F.tar.gz*. Appropriate demerits will be given to those who fail to follow instructions.

The grading of this machine exercise is as follows:

- Part I: Automated Testing
  - 20% - Modified testbench (for this part, you are required to show the simulation result to your instructor on the laboratory class)
- Part II: Register File
  - 40% - RTL model (points will be distributed equally among the 32 registers for further partial points)
  - 40% - Synthesized model (points will be distributed equally among the 32 registers for further partial points)

The deadline of the submission of this machine exercise is on February 3, 5, and 7 for the Monday, Wednesday, and Friday classes, respectively.