Lab 2
Adding a Memory Interface & Automatic Checker

Module ID : CX-301

Design Verification

Instructor : Dr. Abid Rafique

Version 1.2

Document History

The changes and versions of the document are outlined below:

Version	State / Changes	Date	Author
1.0	Initial Draft	Jan, 2024	Qamar Moavia
1.1	Modified with new exercises	feb, 2025	Qamar Moavia

Table of Contents

Objectives

By the end of this lab, students will be able to answer the following questions:

- What are the advantages of using interfaces in SystemVerilog?
- How do interfaces simplify communication between modules?
- How can arrays be helpful in writing self checking testbenches?

Tools

- SystemVerilog
- Synopsys VCS

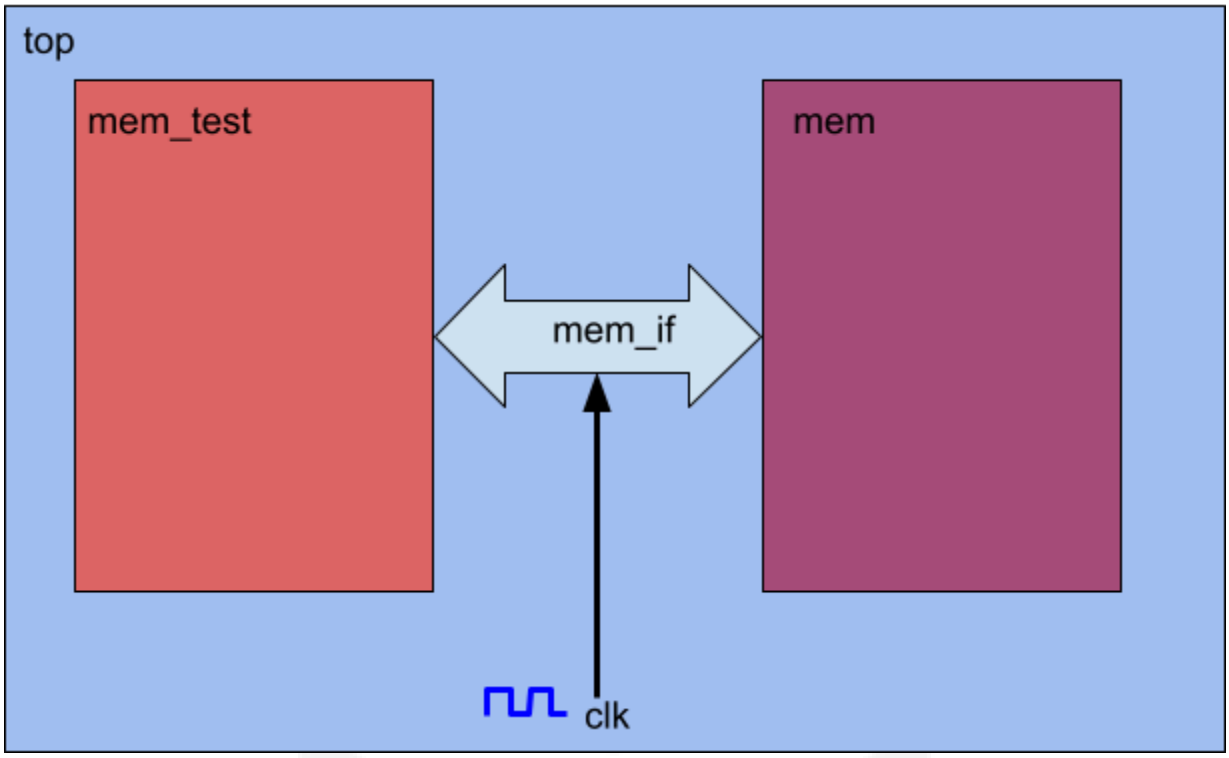
Instructions for Lab Tasks

For this lab, trainees must start from the **solution of Lab 1**. No additional files are provided.

The submission must follow the hierarchy below, with the folder named after the trainee (no spaces), and the file names exactly as listed below.

```
./trainee_name_lab2/
        ├── Task1/
        │    ├── mem.sv
        │    ├── mem_test.sv
        │    ├── top.sv
        │    ├── mem_interface.sv
        │    ├── // screenshots of outputs/waveforms
        ├── Task2/
        │    ├── mem.sv
        │    ├── mem_test.sv
        │    ├── top.sv
        │    ├── mem_interface.sv
        │    ├── // screenshots of outputs/waveforms
```

Along with that you also need to upload your solution on the github as well, and share the link



TASK 1 : Using a Memory Interface

Copy the solution of lab1/Task2 into the Lab2/Task1 directory.

Adding the Memory Interface

Working in the Lab2/Task1 Directory perform the following:

1. Define the Memory interface in a `mem_intf.sv` file and declarations for the addr, data_in, data_out, read and write signals.
2. Edit your Memory design and testbench by updating the port list with an interface port and referencing the interface signals via the interface port name.
3. Modify the top-level module to make an instantiation of the interface and connect this to the Memory design and testbench instances.
4. Rerun your memory test to check that the interface is working correctly. Implement the register using SystemVerilog or Verilog constructs.

Adding a clk input to Interface

5. Add a clock input port to your interface. Remove the clock ports from your Memory design and testbench modules. Update your interface instance to map the clk signal to the clock port. Rerun your memory test to check the interface port is working correctly.Capture a screenshot of the test output.

Adding modports

6. Add modports to the interface to define directional information for both the design and testbench. Reference the modports in your design and testbench port list. Rerun your memory test to check that the modports work correctly.

Adding Interface Methods

7. Redefine the write_mem() and read_mem() tasks as interface methods:
   a. Move the task declarations into the interface.
   b. Update the testbench to reference the tasks via the interface.
   c. Update your testbench modport to allow access to the interface methods via an import statement.
   d. Rerun your memory test.

TASK 2 : Using Arrays in Verification

Create another directory Task2 in the Lab2 directory.

1. Modify the "**Data = Random**" test case by writing random data to all addresses in the memory.
2. Use a loop to generate and write random data to each address one by one.
3. Keep track of the random data that was written to each address.
4. Once all data has been written, read back the data from each address.
5. Compare the read data with the data originally written to each address and verify that they match. This ensures the correctness of the memory operations.

Good Luck 🙂