Lab 2

Creating Interface and TB for a Cache Memory Design

Due date

Sep 10, 2024 12:59 PM CST

Table of content

Academic Integrity

The following actions are strictly prohibited and violate the honor code. The minimum penalty for plagiarism is a grade of zero and a report to the Aggie honor system office.

- Sharing your solutions with a classmate.
- Uploading assignments to external websites or tutoring websites
- Copying solutions from external websites or tutoring websites
- Copying code from a classmate or unauthorized sources and submitting it as your

Introduction

In this lab, you will learn how to create an interface and a testbench for a 5-bit wide cache memory. The cache memory has a 32-address space, four cache entries and exhibits behaviors like cache hits, misses, and writebacks. You will create a testbench to verify the cache's operation.

Design Under Test

Description

The DUT is a 5-bit wide cache memory with an address range of 0 to 31. It is synchronous with asynchronous reset, which means that it only operates on the rising edge of the clock signal.

Data can be written into the cache memory on the rising edge of the clock signal only when the wr_en signal is high. Data from the cache memory can be placed onto the data_out bus on the rising edge of the clock signal only when the rd_en signal is high. Read and write requests cannot be placed simultaneously.

The cache has four entries (0-3). Each entry corresponds to a set of 8 addresses. For example, entry 0 in the cache corresponds to the address range 0-7, entry 1 corresponds to the address range 8-15, and so on.

A cache hit occurs when the address of a read or write request is already stored in the cache. In this case, the request is served from the cache. A cache miss occurs when the address of a read or write request is not stored in the cache. In this case, the request is served from the main memory. If a cache miss occurs, the stored cache entry is written back to the main memory, and the read or write request is stored in the cache entry.

Summary

- The cache memory is 5 bits wide, and the address range is 0 to 31
- The cache access is synchronous with asynchronous reset
- Write data_in into the cache-memory on the posedge of clk only when wr_en=1

- Place cache-memory data onto data_out bus on the posedge of clk only when rd_en=1
- Read and Write requests cannot be placed simultaneously
- The cache has four entries (0-3); there is an entry corresponding to a set of 8 addresses, with a total of 32 addresses. i.e., entry 0 in cache corresponds to address range 0-7, entry 1: 8-15, and likewise
- Cache Hit - For a read/write request, if the cache entry address is the same as the requested address, the read/write request is served from the cache
- Cache Miss - For a read/write request if the cache entry address is different than the requested address, the read/write request is served from the main memory
- Cache writeback - For a cache miss, the stored cache entry is written back to main memory, and the read/write request is stored in the cache entry

This table summarizes the operation of the DUT:

Operation	Condition	Action
Write	wr_en high	Write data_in to cache memory
Read	rd_en high	Place cache data onto data_out bus
Cache hit	Address of request is in cache	Serve request from cache
Cache miss	Address of request is not in cache	Serve request from main memory, write back old cache entry, and store new request in cache

Environment Setup

1. Log in to the Linux server. Use a secure shell to remote login to hera.ece.tamu.edu

Start SSH with X11 Forwarding enabled (-X). X11 forwarding allows you to run graphical applications such as Cadence tools from the remote server and have their graphical user interfaces (GUIs) displayed on your local machine.

```
ssh -Y <netid>@olympus.ece.tamu.edu
load-csce-616
```

2. Accept the assignment's repository on GitHub Classroom: https://classroom.github.com/a/rkUO5exu.

3. Clone your lab repository on the Linux server.

4. cd in the lab directory

```
cd lab-2
```

5. Change the directory to the design testbench

```
cd work/
```

6. Read the files in the tb/ directory. All the required signals in the interface are defined. Define the mod-ports for tb & mem. Include the tasks mem_read and mem_write in the tb modports, as we will use these tasks to create stimulus.

7. cmbus is an instance of cache_mem_intf with the tb mod-port settings. i.e. the cmbus.mem_read and cmbus.mem_write tasks are available. Create your stimulus to find the bug in the design. Ensure thet the entire memory is initialized to zero after reset. (hint: You can use $urandom_range(i,j) to generate random addresses and data for your stimulus.)

8. Store the value after each write_request in a separate array, and on every read_request, compare it with the value in the array. Print any mismatches and debug.

9. Source setup bash from sim directory; then compile and simulate the design.

```
source ../../setupX.bash
xrun -f run.f
```

10. Debug if any error in compilation.

11. There is a bug in the design, which you are expected to find with your own stimulus.

12. Describe the bug and your strategy for finding the bug in the file report.md

Deliverables

Commit and push all your changes to your remote repository.

Your repository must include the following:

1. design directory
2. sim directory
3. tb directory containing the updated testbench files
4. bug report file (updated report.md)

Important note: To get full credit, you must upload all the required files and directories and strictly name your files according to the requirements.