SANTA CLARA UNIVERSITY	ELEN 127 Fall 2018	Jim Lewis

Laboratory #6: Cache/memory model

For lab section on November 2, 2018

I. OBJECTIVES

We're going to take last week's arbitration scheme and add a datapath to go along with it.

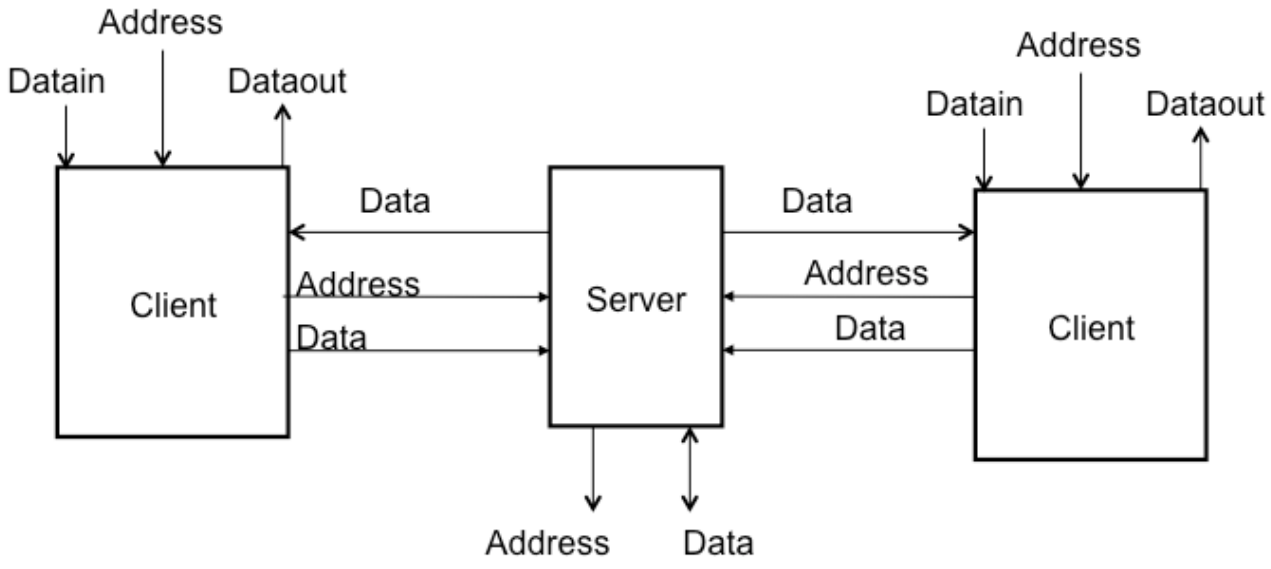
PROBLEM STATEMENT

We need to take our two clients and one server and evolve them into something we can integrate with a basic processor pipeline design. Specifically, we're going to turn our two clients into very simple cache/buffering units, and we're going to turn the server into a very simple memory controller. We already have two interfaces on our client module (which we'll be extending). But we'll also need to a second interface to our server unit, which interact with a memory module.

Looking at the client/cache units first, we need to extend the interface with the server to include three 8-bit buses. One of the buses will be an address bus and the other two will be data buses, one for each direction. The other interface of the client, which we will be connecting to a processor, will need to have the same: one address bus and two data buses.

Looking at the server/memory controller, the interface to the clients will need 6 buses to match the three buses from each client. And the interface to the memory module will need to have one address bus and one bi-directional data bus. You can use the memory module we looked at in class as a baseline for how to define the interface to memory.

The block diagram below shows the additional interfaces. You will need to maintain the same control signals from the previous lab, even though they are not shown on the diagram.



PRE-LAB

(i) Define the Verilog module of the client to account for the interface extensions described in the problem statement. This should be an extension of the client module you created in the previous lab.

(ii) Define the Verilog module of the server to account for the interface extensions described in the problem statement. Again, this should be an extension of the server module you created in the previous lab.

(iii) Define a read buffering strategy for your clients. The requests from the processor will always be for just a single 8-bit quantity (i.e., a byte). But, for read requests, you are required to always convert this into at least a 2-byte request to the memory controller/server. You can choose to make it a 4-byte request as well, or maybe have some policy where sometimes it is 2 bytes and sometimes it is 4.

Given that you will be requesting multiple bytes from the server, yet only providing one of those bytes to the processor, you need to decide what to do with the other byte(s). So you'll need to have at least one byte of local storage/buffering in your client, along with a means of determining whether the next request from the processor is for the data that you have now stored. You can be more elaborate than just storing a single byte. If you have exposure to caches, that might inform how you decide to approach this.

(iv) Define a write strategy for your clients. Some of the requests from the processor will be writes rather than reads. You can decide what you do with them, but you have to ensure that data stays consistent. If you choose to push all writes thru to the memory controller, you need to ensure that if there is a write to an address for which you have data stored in your local buffer, you need to either update it or mark the data invalid. Or you can choose to get more sophisticated and define something along the lines of a writeback cache.

LABORATORY PROCEDURE

After reconciling the client/server interface with your partner, each of you will implement one of the clients, specifically implementing the read buffering and write policies that you defined in your pre-lab. Whoever finishes first can then start working on the server. As a starting point you should use a client (or server) module that was demonstrated to work in the previous lab, whether it be one that you worked on or one that your partner worked on.

1. Reconcile your client/server interface

   (i) Since you may come into lab with different thoughts on the client/server interface, come to an agreement with your partner on a common approach.

2. Implement the two clients

   (i) Each of you will implement one of the clients, specifically implementing the read buffering and write policies that you defined in your pre-lab. As a starting point you should use a client module that was demonstrated to work in the previous lab, whether it be one that

you worked on or one that your partner worked on. And you can use the 8-bit load enabled register that you defined for the register file lab as your building block for your buffers.

3. Implement the server

   (i) Whoever finishes first can then start working on the server, again using a working server module from the previous lab as a starting point.

   (ii) You will also need to implement a memory module, which can just be leveraged from the lecture/book.

4. Create a testbench and simulate

   (i) Once the second client is implemented (or the server), you can start working on the testbench, which will need to instantiate the two clients, the server, the memory module, the appropriate interconnections, and then a means for driving the processor interfaces for the two clients. The TA will provide a template for the processor interfaces.
   (ii) Your testcases should cover at least the following scenarios, for each client:
      a. A read request that is satisfied from your local buffering and therefore does not cause an interaction with the server.
      b. A write request to an address that matches the address of some locally buffered data.
      c. Demonstration of data being written out to memory and then, at some later point, read back in from memory.
   (iii) Once you can show that it's working properly, demonstrate to the TA.

REPORT

For your lab report, include the source code for the two separate instances of your clients, the one you implemented and the one your partner implemented. Also include a description of the test stimulus you provided on the processor interfaces of both clients. In addition, include answers to the following questions.

- What problems did you encounter while testing your steps yourself?
- Did any problems arise when demonstrating for the TA? What were they? Explain your thoughts on how/why these testcases escaped your own testing.