SANTA CLARA UNIVERSITY	ELEN 127 Fall 2018	Jim Lewis

Laboratory #5: Arbitration

For lab section on October 26, 2018

I. OBJECTIVES

Implement an arbitration scheme between two clients and one server.

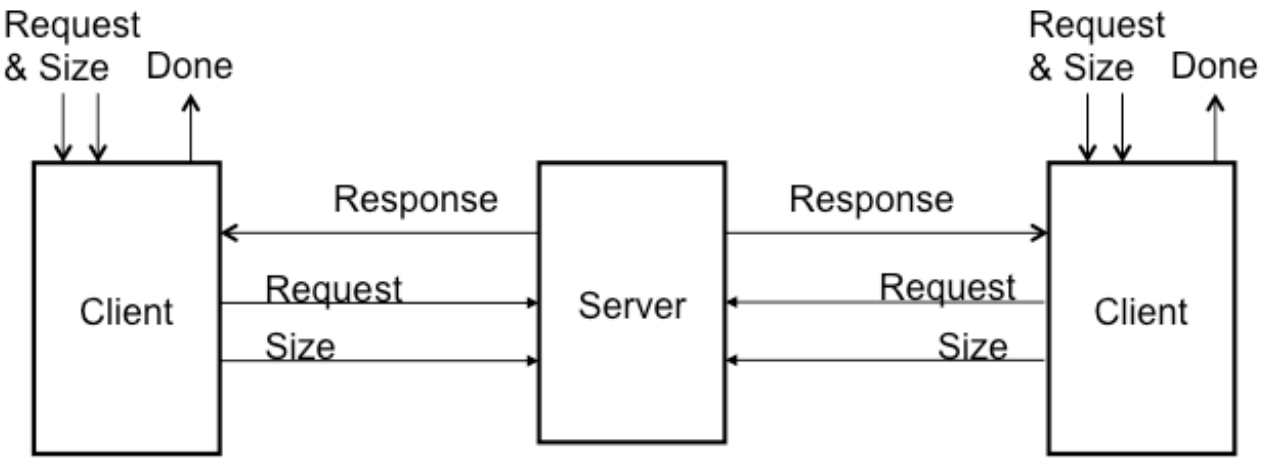
First use of our synthesis tool, Design Compiler from Synopsys.

PROBLEM STATEMENT

We will have three modules interacting with each other: two clients and one server. The clients will be two instances of the same design. Each client will have the ability to request a data transfer with the server of either one, two, or four "beats" (or transfers). The server will have a response interface to each of the two clients. The response will be an acknowledgment signal, and it will need to assert for one, two, or four cycles, depending on the size of the request from the client. Multi-cycle responses need not be in consecutive cycles. If the two clients happen to request at the same time, the server should respond to the client that it did not serve the previous time, so that there is fairness. (We call this round robin arbitration, although with just two clients it's a bit of a degenerate case.)

The requests made by the client will be on behalf of requests coming from another, dedicated interface into the client. But on this interface the client will just return a "done" signal, once the transaction with the server has been completed. See the diagram below.

Note that, while there are two clients shown, the design of the client is just one module, which will be instantiated twice. So the client should only be designed once.



PRE-LAB

(i) Define the module interface of the client. Note that it needs to use different names for the request and size signals on its two interfaces.

(ii) Define the module interface of the server. Note that it also needs to use different names for the requests coming in from the two clients and the responses that it will send back.

(iii) Define a sequence of five requests for each of the two clients. For example, a sequence might look something like

1 beat
2 beats
4 beats
4 beats
1 beat

This would mean the first request would be of size 1, the second of size 2, the third of size 4, etc.

Make the sequences different for the two clients. We will use these in the testbench to generate requests to the two clients.

(iv) Define a state diagram (or SM chart) for a client state machine. The high level behavior we're looking for is to have it receive a request on its higher-level interface, reflect that request to the server, wait for the appropriate number of response cycles, and then assert the done signal back to the higher-level interface and be ready to accept another request.

Note that the SM should be defined to work with any arbitrary sequence. In other words, it should work for both your clients, even though you have them process two different sequences when you test them.

(v) Create the state diagram (or SM chart) for the server. At a high level, the server waits for a request; if there are two requests it needs to decide which it will serve. (Again, a very simple round robin arbitration scheme.) Once it receives a request and notes how many beats the response should be, it should then assert its response signal for the number of cycles corresponding to the size of the request. Once it's done with the request it can accept another, from either the same or a different client.

(vi) Have a look at the Design Compiler tutorial sometime before lab, so that you at least know how to start the tool. Maybe use a Verilog file you already have from a previous lab and see if you can at least get that synthesized. Don't worry about timing constraints.

LABORATORY PROCEDURE

Since you will have defined your own protocol for the client/server interface, the first thing you and your partner will need to do is reconcile your designs, i.e., come to an agreement on how the signals will be defined on the client and server side. Then you can decide how you want to split the work, with one of you working on the client and the other on the server. Whoever finishes first can start working on the testbench.

1. Implement the two clients

   (i) The interface and sequencing state machine should be the same in both cases. The only difference between the two should be the array used to control the request sequence.

2. Implement the server

   (i) Implement the response state machine and the array for the response sequence, which the state machine will use to determine how it responds to each request.

3. Create a testbench and simulate

   (i) Instantiate your two clients and your server, connect the appropriate interface signals, define and connect a clock, and get it running.
   (ii) Once can show that it's working properly, demonstrate to the TA.

4. Synthesize your designs

   (i) Using the guidance of the tutorial, generate gate-level netlists of both your client and server modules.

REPORT

For your lab report, include the source code for your client and server state machines and for the testbench you used to verify the behavior. Also include the synthesized gate-level netlists for your two modules. 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.