**Chapter 7, Threads and Interprocess Communication homework 1**

The purpose of this homework is to practice use threads, classes, and a golden model to test a 3-port arbiter. The arbiter design to test will be provided to you in an encrypted format. The arbiter’s ports are in Table 1.

|  |  |  |
| --- | --- | --- |
| **Signal Name** | **Direction** | **Description** |
| clk | input | clock |
| reset | input | active high reset |
| req0 | input | Request for port 0 |
| req1 | input | Request for port 1 |
| req2 | input | Request for port 2 |
| en0 | input | Enable for port 0 |
| en1 | input | Enable for port 1 |
| en2 | input | Enable for port 2 |
| grant0 | output | Grant for port 0 |
| grant1 | output | Grant for port 1 |
| grant2 | output | Grant for port 2 |

Table 1: Arbiter ports

The characteristics of the arbiter are:

1. Priority is maintained in a round robin format, i.e. port 0->port 1->port 2->port 0…
2. At reset the priority is port 0.
3. Priority will be incremented whenever any grant is asserted on the positive edge of the clock.
4. Grant is combinatorial
5. If a port with priority is not enabled or not currently requesting the next port will be given the grant given that port is enabled and requesting the bus, and so on.
6. A sample waveform of the arbiter response is in Figure 1.

The minimum requirements are:

1. Create a class to encapsulate each port transaction. A transaction is made up of at least **en** and **req**.
2. Use an interface for each arbiter port. The interface will be instantiated once for each port.
3. Create a golden model of the arbiter given the requirements above.
4. Your testbench shall:
   1. Run a directed test replicating Figure 1.
   2. Run a random test that issues 3 independent port transactions, i.e. threads, to each port every clock cycle for 100 clock cycles.
   3. Compare at every negative clock edge the value of the 3 grant outputs of the golden model with the 3 grand outputs of the DUT.
   4. Report an error and increment an error counter if the comparison of the golden model vs the DUT does not match.
5. Use a program for the stimulus, a package for the class, and a top module to instantiate the interfaces, DUT, golden model, and program and tie them together.



Figure 1: Arbiter Response

Deliverables:

1. Code for the package, program, top level module, the golden model, and the interface.
2. Waveform of the directed test and the next 10 random cycles (this will take multiple pages) of the following signals in the golden model you created.
   1. error count
   2. req, grant, and enable for each port
   3. priority