Mesh lab #1 – introduction to the mesh

How to work as a group

This lab involves coding and simulating. You can work as a group and only turn in one set of files.

The goal of working as a team is to mirror the way you will most likely work in a real job. However, my expectation is that everyone in the group learns the code and runs the simulations. The goal is *not* to have only one person learn the material 😊.

Goals of the lab

In this lab, we'll

- Put together our mesh. Other than bug fixes, it will be the full, complete mesh.
- Use a simple testbench that does not thoroughly test the mesh.
- Thus, our mesh is likely to still have bugs in it.

What we won't do for now (but will do soon enough):

- Test the mesh much more thoroughly by writing a *random-code generator* (lab #2)
- Create a *tracker* to help us debug the mesh (lab #3).

Big-picture instructions

- Download the files *mesh_defs.sv*, *interface.sv*, *mesh_stop.sv*, *mesh_NxN.sv* and *tb_mesh_1.sv*. Flesh out the code in *mesh_stop.sv* as described below.
- Reuse your existing *fifo.sv* (but not your *tb_fifo_3.sv*).
- Run your code and be sure that it passes the rudimentary tests in *tb_mesh_1.sv* (which is self-checking)
- Turn in your final *mesh_stop.sv*.

What is in which file

The code is divided up as follows:

- *mesh_defs.sv*: a set of common definitions needed by all of the other files.
- *mesh_stop.sv*: the basic code for a single mesh stop.
- *mesh_NxN.sv*: instantiates an *N* by *N* grid of individual mesh stops to create the full mesh.
- *tb_mesh_1.sv*: the testbench.

The code for you to write

The file *mesh_stop.sv*, as noted above, is the basic code for a single mesh stop. However, it is not finished. While the datapath logic is in place, the control logic is not. Your job is to write that control code based on what we've discussed in class (*mesh.pptx*).

How the testbench works

You do not need to make any changes in the testbench for this assignment. However, we will be using essentially the same testbench for the next several labs and you may find it useful to understand how it works. Also, the questions below concern the testbench, so you will have to at least slightly understand it.

Tb_mesh_1.sv defines a simple testbench for our mesh. It starts with some basic code:

- Declares various constants that will control how big of a mesh we build (in this case, only 2x2) and how long the test runs (it will stop after sending 20 packets).
- Declares a class called *Sim_control*. For now, *Sim_control* supplies the input vectors for a very simple directed test. However, in the next lab you will rewrite it to make a more powerful RCG.
- Declares the top-level signals that will connect to our DUT (i.e., to the mesh). It also declares the array *packet_history*, used only by verification, which stores the packets we've sent into the mesh so that we can check they exit at their correct destination.
- Instantiates the mesh, as well as a *Sim_control* object to supply test vectors.
- Drives the main clock as usual.

Next comes the top-level *driver* routine: an **initial** block that

- resets the system
- uses a **repeat** loop to drive *N_PACKETS_TO_SEND* packets into the mesh, one at a time.
- Inside that loop, we first create a packet and save it in *packet_history*.
- We then drive it into the mesh for one cycle and stop driving it the next cycle.
- We then wait 10 cycles for it to (hopefully) propagate through the mesh before sending the next packet
- When all packets are done, call **$stop** to end the simulation.

Finally, the top-level checker routine: an **initial** block that mostly runs a big loop, checking every mesh stop to see if a packet has arrived at the mesh-stop output. When one does, we:

- Print the packet (to help debugging).
- Look through *packet_history* to check if this packet was ever sent; if not, complain and end the simulation.

Note that if the simulation does not forcibly end because of the packet-never-sent check, then it is deemed to succeed – no end-of-sim success message gets printed.

Questions

1. The testing that we're doing in this lab is quite basic and inadequate. For example, when we receive a packet we merely check that the packet was previously sent and not yet received. What other checks would be useful to assure that the mesh is faithfully transmitting the packets we have given to it? Note that this question is *not* asking how to give the mesh a more comprehensive set of packets to send, but rather how to ensure that the packets we do send are faithfully received.
2. Now for the other half – how might you better choose which packets to send, when to send them and when to take them once they've arrived at their destination?

What to turn in:

- Turn in your final *mesh_stop.sv* and *fifo_1.sv*, as well as a .pdf with answers to the questions.
- Just have one person per team turn in work; no need for every individual to turn in something separate.