Mesh lab #3 – building a tracker

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

At this point, after two mesh labs, we now hopefully have a reasonably good RCG that has tested your mesh enough to make the code fairly robust. This lab will:

- add a tracker, at which point you'll be all ready for the debug derby.
- Give you further experience with SystemVerilog classes, including the use of static class instance variables.

Once the tracker is done, we'll be ready for our debug derby rounds!

Big-picture instructions

- Reuse all of your code from the previous lab. Add tracker code to *tb_mesh_2.sv* as described below to make *tb_mesh_3.sv*.
- Run your code and verify that your tracker works.
- Turn in your final *tb_mesh_3.sv* and *mesh_stop.sv* (which may have new bug fixes in it).

The Tracker class

Here is a skeleton for the Tracker class:

```
class Tracker;
        static string signames[$];
        static Ring_slot vals[$];

        static function void add_signal (string signame, Ring_slot RS);
                ...
        endfunction : add_signal

        static function void find_and_print (Ring_slot RS);
                ...
        endfunction : find_and_print

endclass : Tracker
```

The class has two **static** instance variables. Both are *queues*. A SystemVerilog **queue** is similar to a C++ **vector**; an array whose size can grow at runtime (see Spear chapter 2.4 or the LRM 7.10). Because we declare the queues **static**, there is only one copy of each, shared by all *Tracker* instances. The idea is that for every signal in the design where a packet might be hiding, we will put its signal name in *signames*[*i*] (for some *i*) and its current value in the corresponding *vals*[*i*].

The *Tracker* class has two user-visible methods:

- *add_signal* (**string** *signame*, *Ring_slot RS*). This method gives *Tracker* one signal's name and value. The method should update *signames*[] and *vals*[] accordingly.

- *find_and_print* (*Ring_slot RS*). This function should hunt through all the signal name/value pairs. When it finds a value that matches *RS*, it should print the associated signal name. I.e., it hunts through the design to find where a given packet is at the moment.

You may decide to write one or more small "helper" methods in the *Tracker* class if it makes your job easier, but the only mandatory methods are the two above.

Integrating the Tracker class into mesh_tb

You can choose any of three tiers of how difficult to make this lab:

1. The simplest is to track only the rings: *vert_ring*[] and *hori_ring*[]. In this case, you might simply have a loop that iterates through all 2·*MESH_SIZE*^{2} ring locations and calls *add_signal*() appropriately. This is the same idea as in *tracker.pptx* slide #10, but using our *Tracker* class.
2. The next tier is to also track the outputs of the mesh-stop-internal FIFOs. In this case, you would most likely use a trick similar to *tracker.pptx* slide #19, creating a module that you then **bind** to the mesh stops. This tier is worth 5 points; if you choose not to do it then your highest score is a 95.
3. The hardest tier is to also track the internal state of the mesh-stop-internal FIFOs. This is more complex; you might consider creating a specialized version of **module** *tracker_module* that still (similar to the original *tracker_module*) instantiates a *Tracker* instance, but is specialized to FIFOs – e.g., it knows where they store their internal state, and how that state becomes valid or invalid as the read and write pointers advance. This tier is extra credit, worth an extra 5 points.

You may want to directly instantiate one *Tracker* object from within *tb_mesh_3.sv* with code such as

```
Tracker tracker = new();
```

No matter which tier you choose, you still have to call *find_and_print*() from somewhere in *tb_mesh_3.sv*. The best time to do this is right after the falling edge of *clk*; since the mesh state always changes on the rising edge of *clk*, the falling edge will have all signals stable and easily examined without races.

Interestingly, when you call *find_and_print*(), you don't have to call it as *tracker.find_and_print*(). Since the *Tracker* methods are static, you can instead call

```
Tracker::find_and_print (hunt);
```

Questions

1. How would your code change if there were multiple packet types, and each occupied a different number of ring slots?
2. The tracker that we've built is essentially a monitor. Once it has abstracted the bits into high-level packets, we might want to use them to write checkers. What things might you check using the output from this monitor?

What to turn in:

- Turn in your final *tb_mesh_3*.sv, as well as your latest *mesh_stop.sv* and 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.