FIFO lab #2 – adding a reference model and scoreboard

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

- Not touch our FIFO itself (other than to fix any issues from lab #1)
- Improve our testbench by adding a *reference model* and a *scoreboard* to make it self-checking. No more having to look at waveforms to determine if it works!

What we won't do for now:

- Test the FIFO thoroughly, and measure how thoroughly we've tested it.
- Those will happen when the FIFO becomes part of the mesh.

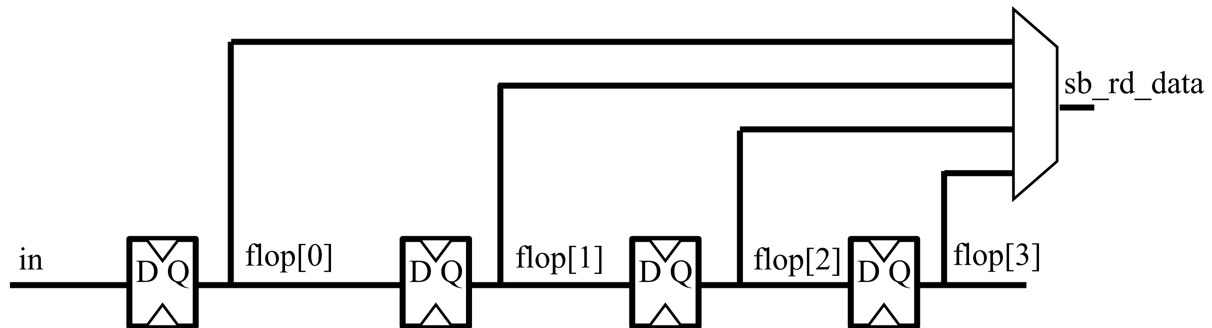
Big-picture instructions

- Reuse your two files from the previous lab. If your FIFO does not meet our standards that we discussed in class, then change it. Otherwise, you can leave it as is.
- Add the new testbench code to *tb_fifo_1.sv*, creating a new file *tb_fifo_2.sv*.
- Run your code, either using *edaplayground.com* or your favorite SystemVerilog simulator.
- In principle, there's no need to check your results by eye, using a waveform viewer. But you'll probably wind up doing some waveform viewing just to debug your work.
- Turn in your final *fifo.sv* and *tb_fifo_2.sv*, as well as a .pdf with answers to the questions below.

What is a reference model and a scoreboard?

A *reference model* is a verification model of what the Device Under Test (DUT) *should* be doing, cycle by cycle, if the DUT is working properly. You drive the DUT and the reference model with the same inputs. The *scoreboard* then compares their outputs – if the DUT and the reference model ever have different outputs then you've found a bug.

The reference model should match what we called "FIFO take 3" in the FIFO Powerpoint slides (see below).



The slides don't give any detail on how to drive the select line(s) of the mux that drives the FIFO output data. The idea is to keep a count *n_items* of how many items are in the FIFO at any time. *N_items* would increment whenever you write the reference-model FIFO and decrement on reads. Since items are always written into flop[0], and move to the right as more items are added, the oldest item will always be in *flop*[*n_items*-1].

This is not only relatively easy to implement in code, it also gives us an easy way for the reference model to compute its version of *empty* and *full*.

Finally, remember to clear *n_items* during reset.

What new code to write for tb_fifo.sv

You should add the reference model and scoreboard to the *tb_fifo* module as follows, piece by piece:

- Declare your new signals. E.g.,
```
// Declare the flops and the ref-model output data.
parameter n_flops = 1 << N_ADDR_BITS;
logic [FIFO_WIDTH-1:0] flops [n_flops-1:0], rm_rd_data;
int n_items;
```
- Add an **always_ff** block to create the flops, and also to drive *n_items*.
- Add an **always_comb** block to drive *rm_read_data*.
- Add a scoreboard, which is simply assertions to complain if there is a mismatch between the reference model and the DUT. The assertions should look something like
```
scoreboard_checker: assert property (@(posedge clk) …
      else $strobe ("T=%0t: SB=%0x, DUT=%0x", $time,
                    sb_rd_data,fifo_rd_data);
```
  It's up to you to decide exactly what to code in place of the "…", but it should compare sb_rd_data with the DUT's *fifo_rd_data*. Then add similar assertions to check *empty* and *full*.

Races when driving n_items

The variable *n_items* can be a bit tricky. While we won't tell how *to* drive it, here are a few ways that will *not* work reliably because of races.

-
```
always_ff @(posedge clk) begin
   if(wr_en) ++n_items
   if (rd_en) --n_items
```
  Hint: check out exactly how "++" and "--" work in Sections 11.4.1 and 11.4.2 of the 2017 SystemVerilog LRM.

-
```
always_ff @(posedge clk) begin
```

```
if(wr_en) n_items <= n_items+1;
if (rd_en) n_items <= n_items-1;
```

Hint: what will this code do if both wr_en and rd_en are active in the same cycle?

Questions to answer:

1. You already wrote a (hopefully!) perfectly fine FIFO in the previous lab. We just wrote a completely different FIFO, which was kind of a lot of work. Why bother? Why should we not have just reused your FIFO code from last week as the reference model?
2. For each of the don't-do-it-this-way scenarios in driving *n_items*, can you explain why that way wouldn't work? What is the nature of the race involved?

What to turn in:

- Turn in your final *fifo.sv* (which is probably the same as in the first lab) and *tb_fifo_2.sv*, as well as a .pdf with your answers to the two questions.
- Just have one person per team turn in work; no need for every individual to turn in something separate.