FIFO lab #1 – introduction to a simple FIFO

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

- Make sure we understand how a basic FIFO works.
- Get our first experience in coding with SystemVerilog.

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

- Check the FIFO automatically (you will still be manually checking waveforms!)
- Test the FIFO thoroughly, or even measure how thoroughly we've tested it.

Files

There are two files for this example: *fifo.sv* and *tb_fifo_1.sv*. Download them from the course web site and then add your own code to it as needed.

Fifo.sv contains the "skeleton" of our basic FIFO; *tb_fifo_1.sv* contains the skeleton of a testbench for the FIFO. Your job will be to flesh out the details of both.

Big-picture instructions

- Download the two files. Flesh out the code in those files as described below.
- Run your code, either using *edaplayground.com* or your favorite SystemVerilog simulator. Note that we will check your code using Mentor QuestaSim, so please be sure that you check your code runs on that platform (*edaplayground.com* does support it).
- Check your results by eye, using a waveform viewer
- Turn in your final *fifo.sv* and *tb_fifo_1.sv* files, as well as snapshot of the waveforms that you used to check your results.

What code to write for fifo.sv

The file *fifo.sv* defines a SystemVerilog module called *fifo*. It has the usual inputs and outputs that we discussed in class. Furthermore, it takes two parameters:

- DATA_TYPE describes what type of data the FIFO holds. By default it will hold two-bit signals, but any instance of the FIFO can change that. When we use the FIFO to build our mesh router, we will store much more involved data.
- N_ADDR_BITS says how deep the FIFO is. If, e.g., N_ADDR_BITS=3, then the FIFO will be able to hold up to 2^3=8 pieces of data. In other words, this tells how many address bits the FIFO will need to address its internal memory (not including the extra address bit that distinguishes empty from full).

The file also defines an **always_ff** block and an **always_comb** block, to hold any flops and combinational logic respectively.

Your job is to flesh out the code in this file:

- Add the FIFO internal memory, read pointer and write pointer.
- Drive the FIFO empty and full signals.
- Remember to hook up reset! When *reset* is high, both *rd_ptr* and *wr_ptr* should be driven synchronously to 0 (i.e., they should go to zero at the next clock edge).

What code to write for tb_fifo.sv

The file *tb_fifo.sv* defines a SystemVerilog module called *tb_fifo*. That module has the following pieces:

- It starts by declaring various top-level signals that will connect to the FIFO instance.
- A placeholder for you to instantiate the FIFO (using the signals just defined)
- A placeholder for you to drive the main clock in an **initial** block. The clock should start out low and then toggle every 10 time units forever (or until it is stopped; see below).
- A function to generate random input data.
- The top-level tester routine: an **initial** block that resets the system, and then uses a **repeat** loop to drive new random data into the FIFO *wr_data* input every cycle for 10 cycles. You should write new data at the falling clock edge. You should both write and read data every single cycle.
- Call **$stop** to end the simulation. Without this, the clocking block would run forever.

What to turn in:

- Turn in your final *fifo.sv* and *tb_fifo_1.sv* files, as well as snapshot of the waveforms that you used to check your results.
- Just have one person per team turn in work; no need for every individual to turn in something separate.