FIFO lab #2 – adding a scoreboard

Races when driving n_items

The variable *n_items* can be a bit tricky. While we won't tell how 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 scoreboard?
   Because checking anything against itself would always call itself correct!
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?
   In the first example, the use of "++" and "--" has the issue that, according to the LRM, both of them are *blocking* rather than non-blocking. Blocking assignments are typically discouraged inside of an **always_ff** block. The main issue is that different **always** blocks can execute in arbitrary order; and and it is hard to tell if a different block reading *n_items* will get the new value or the old one. Thus, the race. Note that the order of statement execution *within* a block is not arbitrary – statements with a block execute in order.
   In the second example, if *rd_en* and *wr_en* are both asserted in the same cycle, then we will schedule *n_items* for update twice in the same cycle. It is indeterminate which assignment happens first.