1.1 Uniqueness in threads -- the FIFO

in_data is pushe into a FIFO upon a *push* control signal, data is popped out as *out_data* upon a *pop* signal.

On second thoughts, the problem with this assertion for the fifo is uniqueness.
Below is a way to fix this using tags to force uniqueness. What I mean here is that yoou don't want a pop to complete 2 separate threads, as shown in the simulation results for ap_data_checker_bad where one pop terminates both threads.

A solution that appears plausible, but has severe issues, is the following:

```
module fifo_aa;
   bit clk, push, pop;
   int ticket, now_serving;
   bit [7:0] in_data, out_data;
   initial forever #5 clk=!clk;

   property p_data_chk_bad; //
     bit [7:0] push_data;
     @(posedge clk) (push, push_data=in_data[7:0])
               |-> ##[1:10] pop ##0 (out_data == push_data);
   endproperty
   ap_data_checker_bad: assert property(p_data_chk_bad);
```

```
Problem is uniqueness,
one pop can terminate
all threads
```

Figure XXX demonstrates the simulation result for this assertion. Note that after 2 push controls with the same value of data, both assertion threads terminate with a single pop; this is obvioulsy not desired.

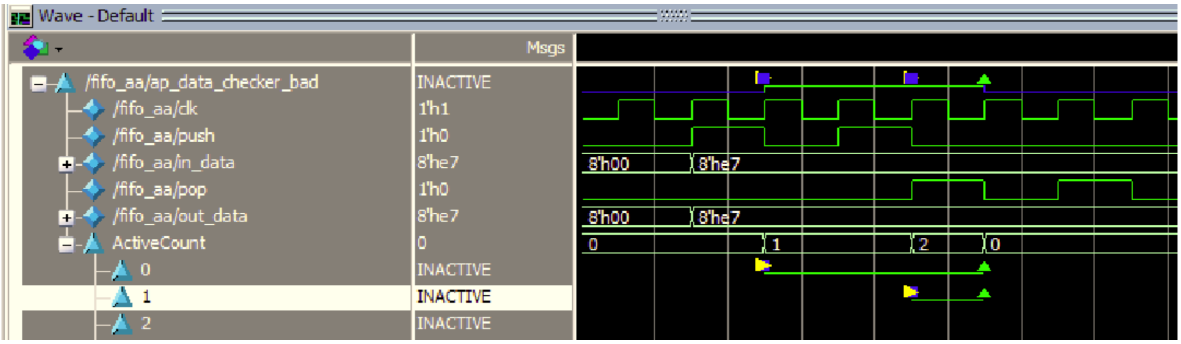


Figure XXX Simulation of a plausible, but incorrect assertion

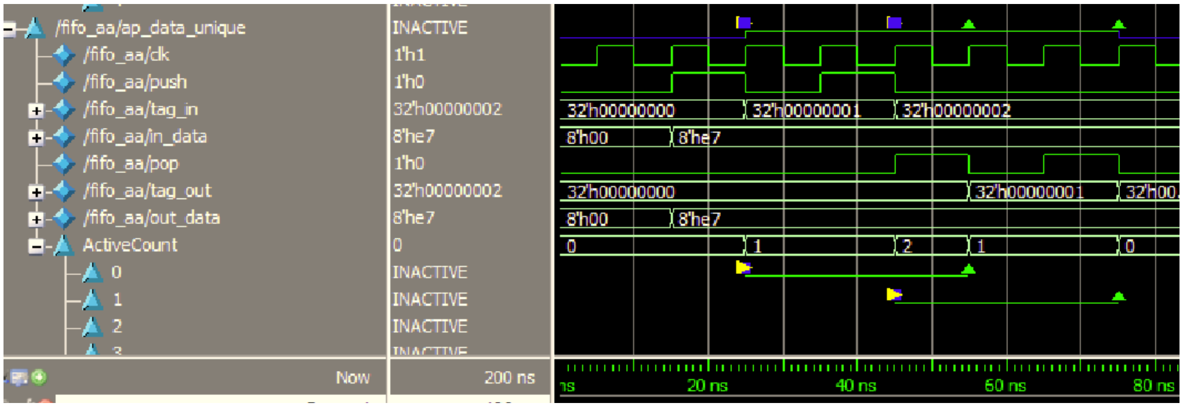
The issue with the above assertion is that there is a lack of uniqueness, or identity for each thread. What is desired for this FIFO assertion is the independence of each attempted thread sequences. To accomplish this, one could use concepts of a model seen in hardware stores in the paint department. There, the store provides a spool of tickets, each with a number. As a customer comes in, he takes a **ticket**. The clerk serving the customers has a sign that reads "**NOW SERVING TICKET #X**". The customer that has the ticket gets served. When done, the number X in incremented, and the next in-line customer gets served. The assertion code could then be written as follows:

```
module fifo_aa;
       bit clk, push, pop;
       int ticket, now_serving;
       bit [7:0] in_data, out_data;
       initial forever #5 clk=!clk;

     function void inc_ticket();
       ticket = ticket + 1'b1;
     endfunction

     property p_data_unique;
       bit [7:0] push_data;
        int v_serving_ticket;
      @(posedge clk) (push, push_data=in_data[7:0],
                              v_serving_ticket=ticket, inc_ticket())
               |-> ##[1:10] pop && now_serving==v_serving_ticket
                   ##0 (out_data == push_data);
     endproperty
     ap_data_unique: assert property(p_data_unique)
                  now_serving =now_serving+1;
             else now_serving =now_serving+1;
```

support variable to achieve attempted thread uniquness

Function needed to increment the ticket spool in the sequence_match_item

now_serving tag incremented at conclusion of assertion for pass or fail case.



/fifo_aa/ap_data_unique INACTIVE
/fifo_aa/clk 1'h1
/fifo_aa/push 1'h0
/fifo_aa/tag_in 32'h00000002
/fifo_aa/in_data 8'he7
/fifo_aa/pop 1'h0
/fifo_aa/tag_out 32'h00000002
/fifo_aa/out_data 8'he7
ActiveCount 0
0 INACTIVE
1 INACTIVE
2 INACTIVE
Now 200 ns

Figure XXXX simulation results with code uniqueness