Asynchronous FIFO test-plan

CHECKS	STATUS	NO. BUGS FOUND	Dependencies
a) **Asserting reset** and checking the output using **reset_test**	DONE	PASSED	N/A
b) **Constraining data_in to reach MAX and MIN values if randomized** data_in_c	DONE	PASSED	N/A
c) **Create a queue *data_to_write_queue* to mirror the FIFO's pointers movements** which is used to compare the read values later on	DONE	N/A	N/A
d) **The mirrored pointers are used to determine the state of the FIFO's flags**	DONE	N/A	N/A
e) **Testing out the limit of the FIFO by writing until its full** while monitoring The flags using **write_all_test/reset_write_read_all_test**	DONE	FAILED (1)	✓
f) **Testing out the limit of the FIFO by reading until its empty** while monitoring the flags **reset_write_read_all_test**	DONE	FAILED (1)	✓
g) **Randomizing write and read operations/tests while monitoring the flags** **write_read_rand_test**	DONE	ALL OF THE ABOVE	✓
h) **Concurrently doing write and read operations/tests while monitoring the flags** **concurrent_write_read_rand_test**	DONE	PASSED	N/A

Coverage Groups

- OPERATION_covgrp: **Covering the write and read and reset operations.**
- INPUTS_covgrp: **Covering the stimulus' data_frames and data_transitions that is applied to the DUT**
- FLAGS_covgrp: **Covering all the flags' data frames and data transitions. *(A habit of mine to preform coverage on outputs too).***

BUG REPORT

EXPECTED	DETECTED
It is expected that the design will not somehow read an older (garbage) value that has been read before: The DUT is set to drive **data_out** instantaneously each time the **reset_pointer** meets the **write_pointer** (when the FIFO becomes empty), since the receiver (the read domain master) will not read the **data_out** unless **r_en** is asserted. This is understandable and is desirable to make the design generally more speed oriented.	**The design reads older and garbage values that has in fact been read before:** After reset is asserted and de-asserted/empty condition is satisfied, the DUT reads the address it stands on instantaneously, without regard to the location of the write pointer. **<u>Sequence to detect:</u>** 1. Reset the DUT 2. Write until its FULL 3. Read until its empty (here, **data_out** will be assigned with a garbage value that has not been re-written and will be read as soon as r_en is asserted). **<u>Dependencies:</u>** - Yes. Quite alot.

Snippet of the BUG:

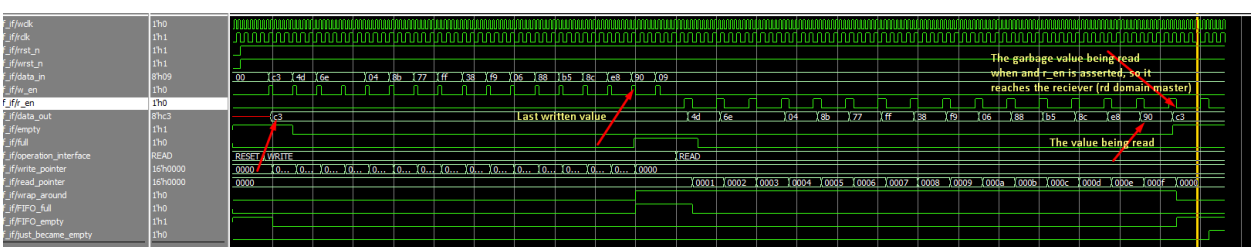

Figure 1: ***The bug appears in the reset_write_read_all_test. This also appears in write_read_rand_test/concurrent_write_read_rand_test.***

SV-Assertions used for debugging & DUT monitoring REPORT

Feature	Assertion
Whenever **write_pointer** increments till it reaches **read_pointer** (wr_p => rd_p), full must be asserted, and when any value is read, **full** must be de-asserted in [1:3] wr_clk cycles	property full_p; @(negedge wclk) disable iff(!rrst_n && !wrst_n) $rose(FIFO_full) \|-> ##[0:$] (!FIFO_full ##[1:3] $fell(full)); endproperty
Whenever **read_pointer** increments till it reaches **write_pointer** (wr_p => rd_p), **empty** must be asserted, and when any value is read, empty must be de-asserted in [1:3] rd_clk cycles	property empty_p; @(negedge rclk) disable iff(!rrst_n && !wrst_n) $rose(FIFO_empty) \|-> ##[0:$] (!FIFO_empty ##[2:3] $fell(empty));

P.S. Check the coverage reports.