Asynchronous FIFO Verification

CHAPTER 1: PROJECT OVERVIEW AND SPECIFICATIONS

1.1 PROJECT OVERVIEW

An Asynchronous FIFO (First-In-First-Out) buffer is a specialized memory structure designed to handle data transfer between two independent clock domains. In such systems, the write and read operations are governed by separate clocks that are not synchronized with each other. This allows data to be written at one clock rate and read at another, enabling safe and reliable communication between different parts of a digital system operating at varying frequencies.

Asynchronous FIFOs are essential in scenarios where data must cross clock domain boundaries without corruption or loss. They are commonly used in system-on-chip designs, FPGA architectures, and digital communication systems. One major application is interfacing between high-speed processing units and slower peripherals, ensuring smooth data flow despite timing mismatches. They are also employed to bridge modules running on unrelated clocks, such as processors, communication blocks, or external interfaces.

1.2 VERIFICATION OBJECTIVES

- Validate reset functionality to confirm that both read and write pointers are correctly initialized and the FIFO commences operation in an empty state.
- Confirm the accurate assertion and de-assertion of the full and empty flags, particularly during pointer wrap-around scenarios.
- Verify reliable operation during concurrent read and write events across independent clock domains, mitigating data loss or corruption.
- Attain comprehensive functional coverage to ensure the exhaustive testing of the FIFO design.

1.3 DUT INTERFACE

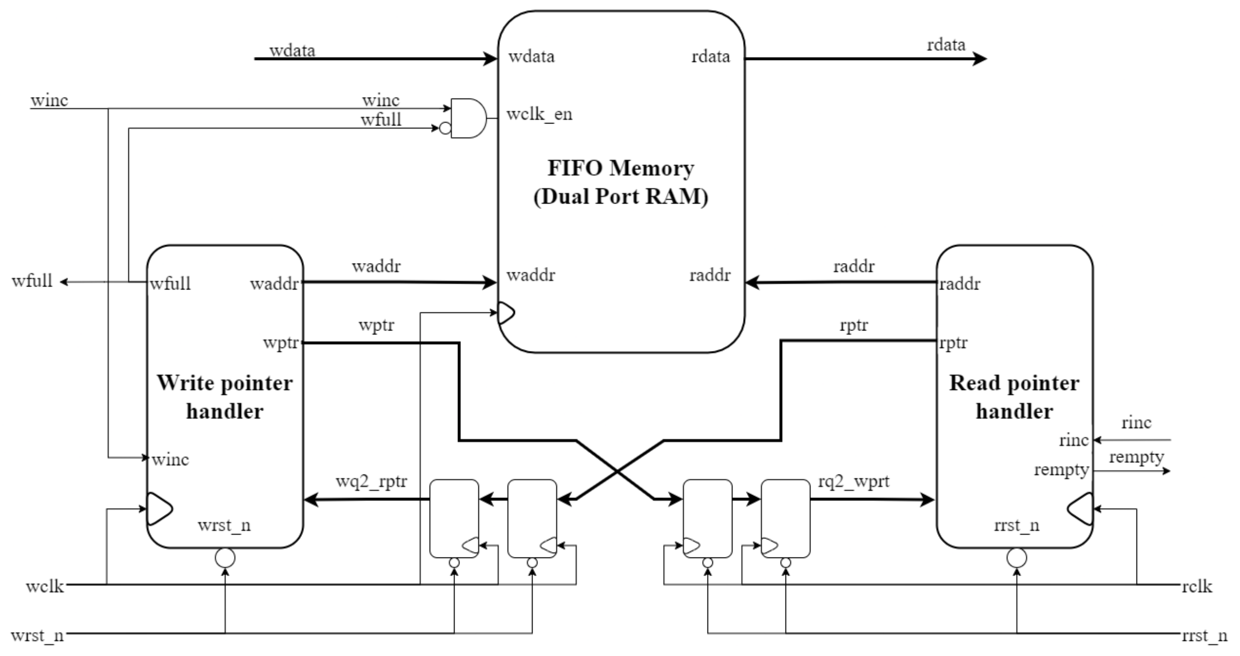


Figure 1: Block diagram of Asynchronous FIFO interface

Inputs:

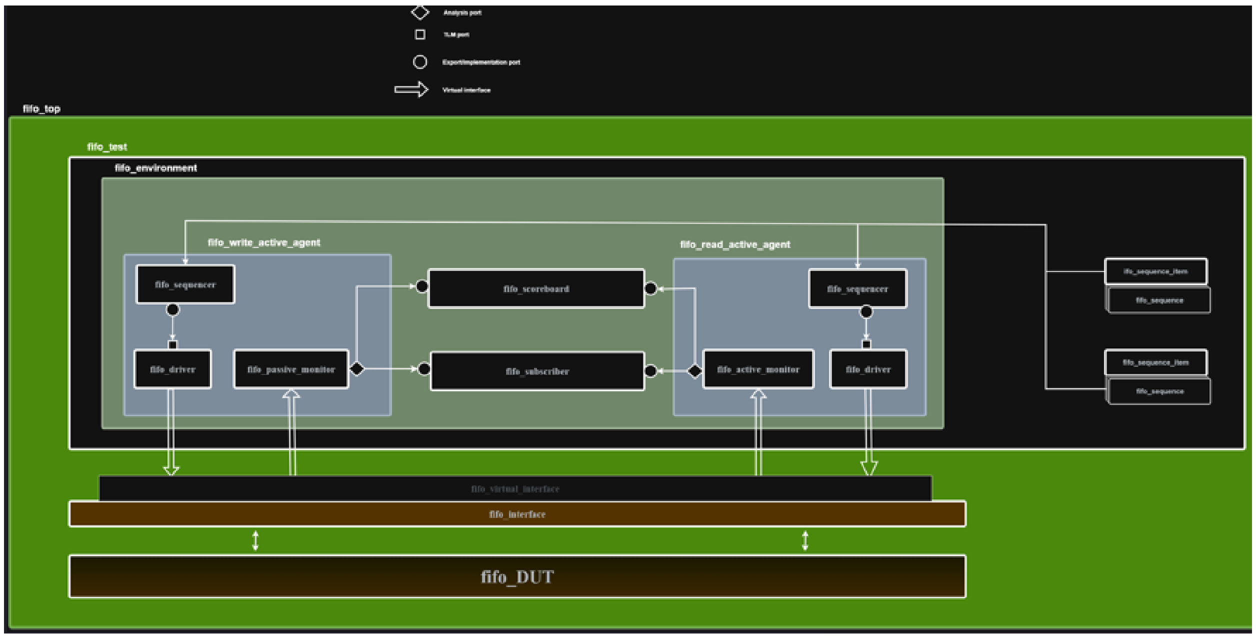
Signal	Width(bits)	Description
wclk	1	Clock signal for the write operation. All the write operations occur on the rising edge of wclk.
wrst_n	1	Resets the write pointer and the writing block logic
winc	1	When the write enable is high, data present on wdata is written into the FIFO on the rising edge of wclk, provided the FIFO is not full.
wdata	DATA_WIDTH	The data input bus, whose width is determined by the DATA_WIDTH design parameter, is used for writing data into the FIFO.
rclk	1	Clock signal for the read operation. All the read operations occur on the rising edge of rclk.
rrst_n	1	The read pointer, read logic, and synchronizers within the read domain are reset.
rinc	1	When the output enable is high, data from the FIFO is driven onto rdata at the rising edge of rclk, provided the FIFO is not empty.

Outputs:

wfull	1	Flag indicates when the FIFO is full by turning HIGH; no further writes are allowed once full.
rdata	DATA_WIDTH	The FIFO's data output bus, used for reading, has a width determined by the DATA_WIDTH design parameter.
rempty	1	When the FIFO is empty, this signal is high, preventing any further read operations.

CHAPTER 2: TESTBENCH ARCHITECTURE

2.1 PROPOSED TESTBENCH ARCHITECTURE



2.2 Component Details and Flowchart

asyn_fifo_interfs: The interface serves as a link between the testbench top module and the DUT, consolidating all associated signals. It defines all input and output signals essential for both read and write operations. Clocking blocks are incorporated to ensure synchronized signal sampling and driving by various testbench components, such as drivers and monitors. Modports are utilized to manage the direction and access of signals for each specific testbench component.

asyn_fifo_write_sequence_item: This class defines the write transaction data, enabling information flow between sequences, drivers, monitors, and the scoreboard.

asyn_fifo_read_sequence_item: It defines read transaction data for testbench components like sequences, drivers, monitors, and scoreboards.

asyn_fifo_write_sequence: The UVM write sequence class is responsible for generating write sequence items and transmitting them to the write driver through the write sequencer. This process dictates the method of write stimulus creation.

asyn_fifo_read_sequence: The UVM read sequence class is responsible for generating read sequence items and transmitting them to the read driver through the read sequencer. This process dictates the method of read stimulus creation.

asyn_fifo_write_sequencer: A write sequencer controls write transactions between the write sequence and write driver, forwarding requests from sequences to the driver.

asyn_fifo_read_sequencer: The read sequencer manages read transactions by serving as a link between the read sequence and the read driver. It receives requests from read sequences and then relays them to the read driver.

asyn_fifo_write_driver: The write driver is responsible for converting write transaction-level data, which originates as sequence items from the write sequencer, into pin-level activity on the DUT. It processes various write transactions, including signals such as write_rst, write_en, and write_data.

asyn_fifo_read_driver: The read driver converts read transaction-level data sequence items from the read sequencer into pin-level activity on the DUT interface. It specifically handles incoming read transactions, such as read_rst and read_en.

asyn_fifo_write_monitor: The write monitor observes the DUT's write signals via a virtual interface, converting them into sequence items. These objects are then broadcast to components such as the scoreboard and coverage. Both write monitors continuously sample the DUT interface signals to ensure accurate write transaction data.

asyn_fifo_read_monitor: The read monitor utilizes a virtual interface to observe the DUT's input/output signals. It translates these signals into read transaction-level objects, also known as sequence items. These objects are then disseminated to various verification components, such as the scoreboard and coverage. This ongoing sampling process guarantees the accuracy of transaction-level data, which is crucial for effective verification.

asyn_fifo_write_agent: A write agent comprises a write sequencer, driver, and monitor, connecting the sequencer and driver.

asyn_fifo_read_agent: A read agent contains a read sequencer, driver, and monitor, connecting the sequencer and driver.

asyn_fifo_scoreboard: The scoreboard monitors the DUT's output and compares it to the data written that emulates async FIFO behavior using a TLM Analysis FIFO. This process verifies the DUT's functionality, with any discrepancies being flagged.

asyn_fifo_subscriber: The subscriber class assesses design functionality during simulation. It utilizes write and read cover groups to sample DUT signals and transactions through monitors, identifying both exercised and unexercised scenarios.

asyn_fifo_virtual_sequencer: The async FIFO virtual sequencer coordinates write and read sequencers, synchronizing transactions across clock domains for proper operation sequencing.

asyn_fifo_virtual_sequence: The virtual sequence, executed on the virtual sequencer, orchestrates advanced test scenarios. It manages both write and read sequences simultaneously, facilitating intricate operations such as concurrent processes, resets during transfer, and thorough corner-case testing across asynchronous domains.

asyn_fifo_env: The environment is responsible for consolidating and managing all key verification elements, such as the write/read agents, scoreboard, and coverage, to ensure the DUT's proper verification.

asyn_fifo_test: The test class serves as the principal, user-defined class responsible for overseeing the verification environment. This is achieved by creating, configuring, and initiating stimulus sequences.

top: The System Verilog entry point is the top module, which connects the DUT to UVM components and integrates it with the UVM testbench.

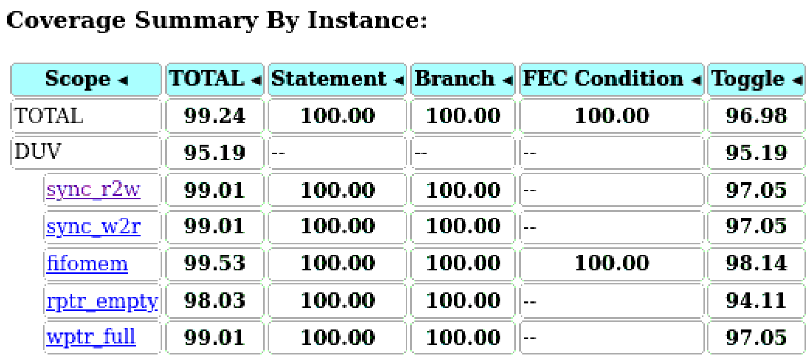
CHAPTER 3: VERIFICATION RESULTS AND ANALYSIS

3.1 ERRORS IN DUT

There is a glitch in the beginning of the Test i.e the rempty is not asserted until the synchronisation of rinc and winc is done for 2 clock cycles it is not an error as its a fifo characteristic in design. The write_full signal does not produce the expected result when the write and read are simultaneously incrementing and decrementing this again is due to the difference in the clocks or the write and read. Thus, leads to incorrect FIFO full detection.

3.2 COVERAGE REPORT

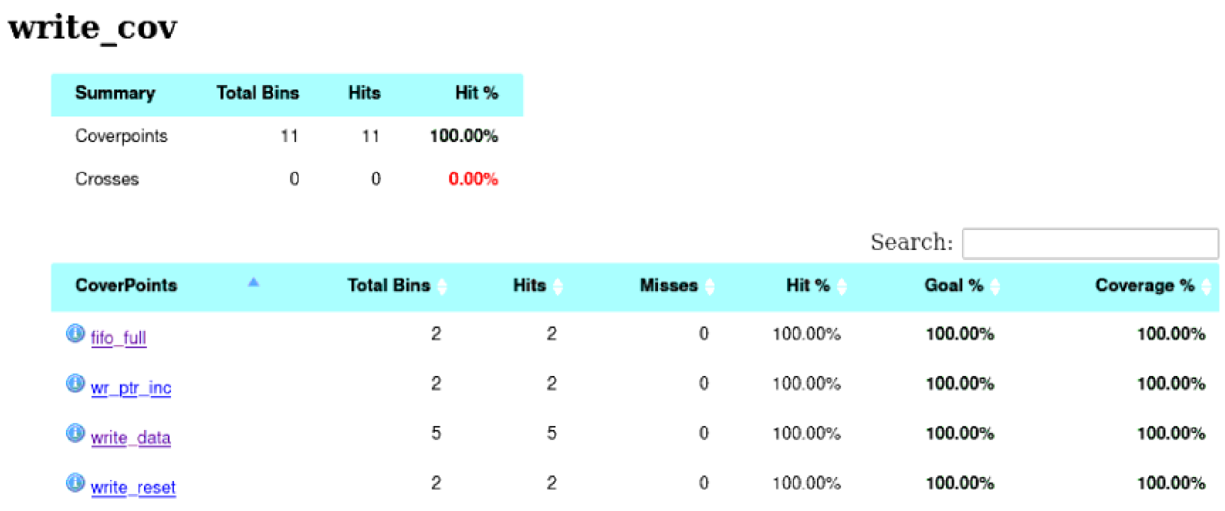
3.2.1 CODE COVERAGE



Coverage Summary By Instance:

Scope	TOTAL	Statement	Branch	FEC Condition	Toggle
TOTAL	99.24	100.00	100.00	100.00	96.98
DUV	95.19	--	--	--	95.19
sync_r2w	99.01	100.00	100.00	--	97.05
sync_w2r	99.01	100.00	100.00	--	97.05
fifomem	99.53	100.00	100.00	100.00	98.14
rptr_empty	98.03	100.00	100.00	--	94.11
wptr_full	99.01	100.00	100.00	--	97.05

3.2.2 WRITE COVERAGE



write_cov

Summary	Total Bins	Hits	Hit %
Coverpoints	11	11	100.00%
Crosses	0	0	0.00%

Search:

CoverPoints	Total Bins	Hits	Misses	Hit %	Goal %	Coverage %
fifo_full	2	2	0	100.00%	100.00%	100.00%
wr_ptr_inc	2	2	0	100.00%	100.00%	100.00%
write_data	5	5	0	100.00%	100.00%	100.00%
write_reset	2	2	0	100.00%	100.00%	100.00%

3.2.3 READ COVERAGE

read_cov

Summary	Total Bins	Hits	Hit %
Coverpoints	11	11	100.00%
Crosses	0	0	0.00%

Search:

CoverPoints	Total Bins	Hits	Misses	Hit %	Goal %	Coverage %
fifo_empty	2	2	0	100.00%	100.00%	100.00%
rd_ptr_inc	2	2	0	100.00%	100.00%	100.00%
read_data	5	5	0	100.00%	100.00%	100.00%
read_reset	2	2	0	100.00%	100.00%	100.00%

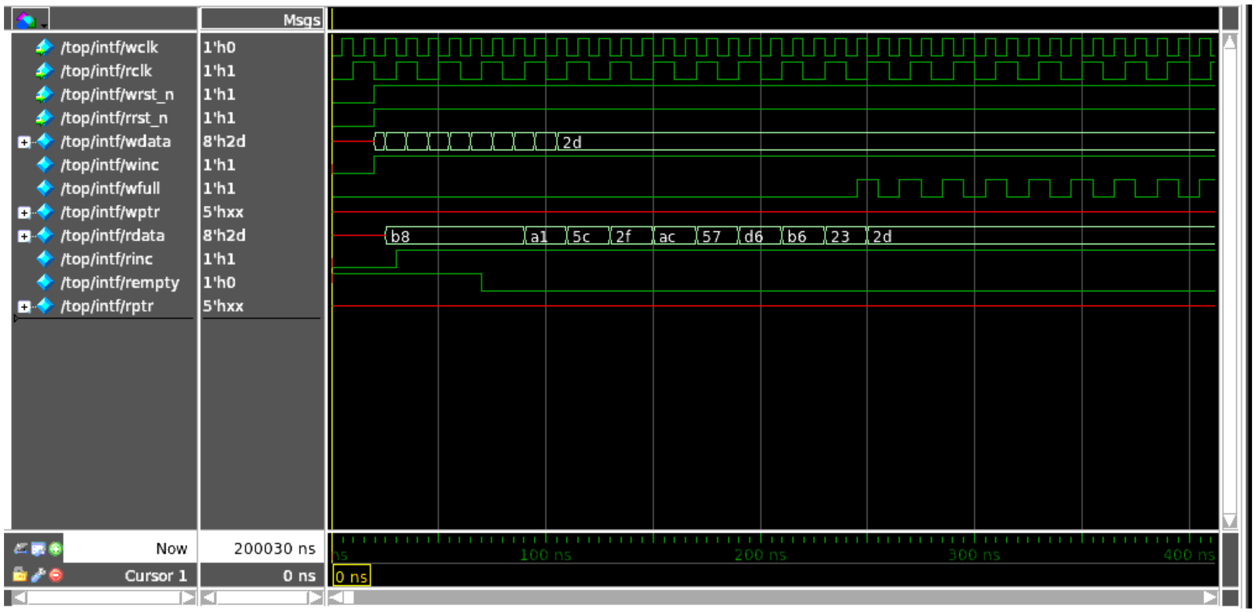
3.2.4 ASSERTION COVERAGE

Assertions Coverage Summary:

Search:

Assertions	Failure Count	Pass Count	Attempt Count	Vacuous Count	Disable Count	Active Count	Peak Active Count	Status
/top/intf/rdata_check	0	9998	10002	3	1	0	1	Covered
/top/intf/wdata_check	0	20001	20003	0	2	0	1	Covered
/top/intf/wdata_stability_check	0	9989	20003	10012	2	0	1	Covered
/work.fifo_if/rdata_check	0	9998	10002	3	1	0	1	Covered
/work.fifo_if/wdata_check	0	20001	20003	0	2	0	1	Covered
/work.fifo_if/wdata_stability_check	0	9989	20003	10012	2	0	1	Covered

3.2.5 WAVEFORM



	Msgs
/top/intf/wclk	1'h0
/top/intf/rclk	1'h1
/top/intf/wrst_n	1'h1
/top/intf/rrst_n	1'h1
/top/intf/wdata	8'h2d
/top/intf/winc	1'h1
/top/intf/wfull	1'h1
/top/intf/wptr	5'hxx
/top/intf/rdata	8'h2d
/top/intf/rinc	1'h1
/top/intf/rempty	1'h0
/top/intf/rptr	5'hxx

Now	200030 ns
Cursor 1	0 ns