Asynchronous FIFO Verification Document

CHAPTER 1

INTRODUCTION AND PROJECT OVERVIEW:

1.1 Introduction

FIFO (First-In, First-Out) is a data structure or buffer in which the first element inserted is the first one to be retrieved. This ensures that data is processed in the exact order it was received. It is widely used in applications where maintaining the sequence of operations is critical.

An Asynchronous FIFO (Async FIFO) is a specialized type of FIFO in which the write and read operations are governed by separate, unsynchronized clock domains. This allows reliable data transfer between different clock domains, ensuring data integrity and synchronization across asynchronous systems.

1.2 Specification

Read and Write Operations

In an asynchronous FIFO, the read and write operations are controlled by independent clock domains.

- The write pointer indicates where the next data will be written. After each write operation, it increments to the next location.
- The read pointer indicates the current data to be read. After each read operation, it increments similarly.
  On reset, both pointers are initialized to zero. When data is first written, the write pointer increments, the empty flag is cleared, and valid data becomes available at the output.

Full, Empty, and Wrapping Conditions

- Empty Condition: Occurs when the read and write pointers are equal, meaning there is no data to read.

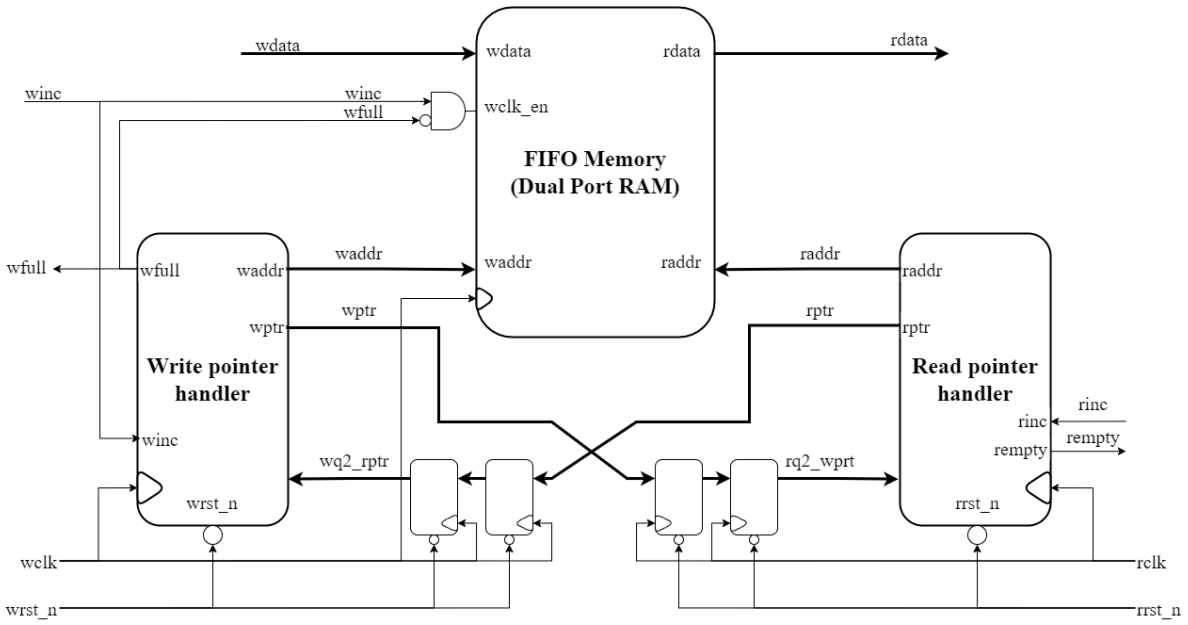
- Full Condition: Occurs when the write pointer wraps around and catches up to the read pointer, indicating no more space to write.
- Wrapping Condition: To distinguish full from empty when pointers are equal, an extra Most Significant Bit (MSB) is added to each pointer. If the MSBs differ, it indicates that the write pointer has wrapped around more times than the read pointer.

This technique ensures accurate detection of **full** and **empty** states.

Gray Code Counter

Gray code counters are used to synchronize pointers between clock domains because only one bit changes per transition. This minimizes synchronization errors and ensures reliable operation across asynchronous domains.

1.3 Design:

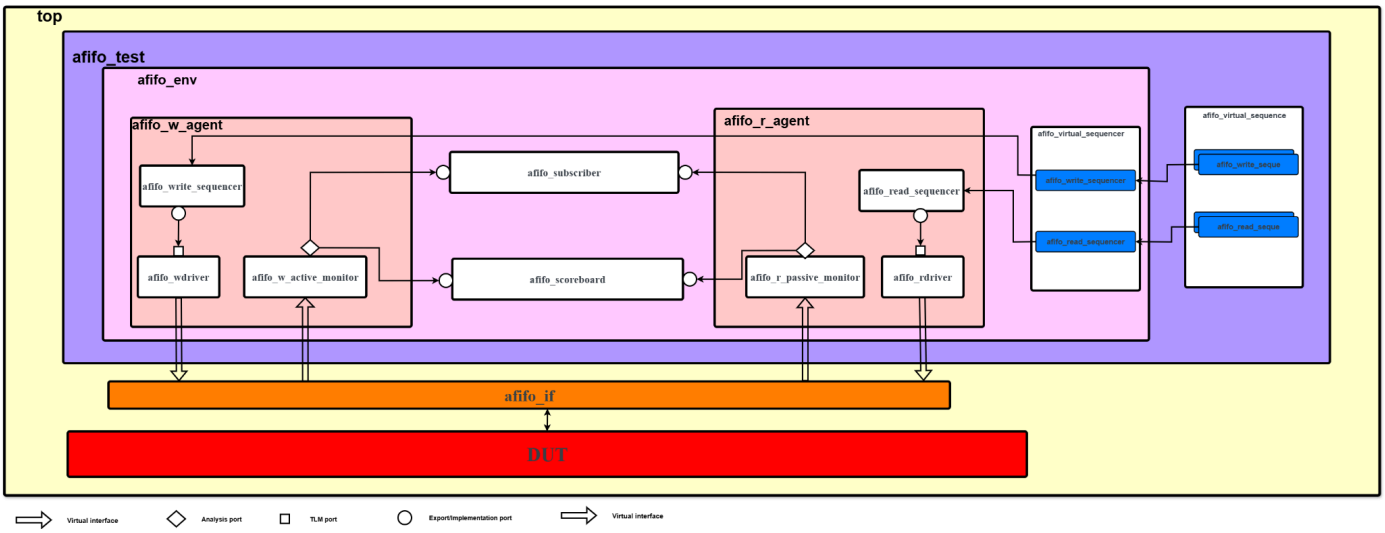


1.4 Interface signals:

Signal	Type	Size (bits)	Description
wclk	INPUT	1	Write Clock Signal
rclk	INPUT	1	Read Clock Signal
wrst_n	INPUT	1	Active-low Asynchronous Write Reset
rrst_n	INPUT	1	Active-low Asynchronous Read Reset
winc	INPUT	1	Write Increment
rinc	INPUT	1	Read Increment
wdata	INPUT	8	Write Data
rempty	OUTPUT	1	Read Empty
wfull	OUTPUT	1	Write Full
rdata	OUTPUT	8	Read Data

CHAPTER 2

TESTBENCH ARCHITECTURE:



Key Components:

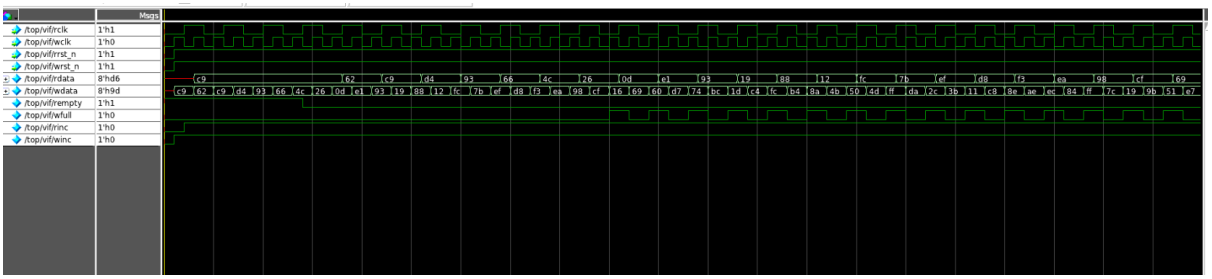
- **Sequence Item** – A user-defined transaction object containing the fields for a single operation, such as input data and control signals.
- **Sequence** – Generates a series of sequence items to stimulate the DUT. It is launched from the **test** to initiate stimulus generation.
- **Sequencer** – Acts as a link between the sequence and the driver, transferring sequence items via a TLM export. It ensures that generated transactions are delivered to the driver.
- **Driver** – Converts the transactions from the sequencer (via a TLM port) into pin-level signals for the DUT, interacting through the **interface**.
- **Monitor** – Passively observes DUT outputs through the virtual interface, converts them into transaction format, and sends them to the scoreboard via a TLM port. Active Monitor is used to capture the DUT inputs. Passive Monitor is used to capture the outputs.
- **Agent -** A reusable block that encapsulates the driver, sequencer, and monitor. Active agent consists of driver, sequencer and a monitor which captures the input.Passive agent consists of only a monitor which captures the DUT response.
- **Scoreboard** - Compares the DUT responses with the expected result.

- **Subscriber** - Records functional coverage to ensure test scenarios are well exercised. It connects to both the active and passive monitor via TLM analysis ports.
- **Environment-** A container that organizes agents, scoreboards, and subscriber.
- **Test –** The top UVM testbench component that builds the environment, sets configurations, and initiates stimulus generation.
- **Top –** Instantiates the interface and the DUT, and triggers the UVM phasing mechanism.
- **Interface –** Connects testbench components to the DUT's signals

CHAPTER 3

RESULT:

3.1 Waveforms:



3.2 Coverage :

1. Code coverage :

Coverage Summary By Instance:

Scope	TOTAL	Statement	Branch	FEC Condition	Toggle	Assertion
TOTAL	**95.82**	**100.00**	**100.00**	**100.00**	**99.13**	**80.00**
DUT	**98.07**	--	--	--	**98.07**	--
sync_r2w	**99.01**	**100.00**	**100.00**	--	**97.05**	--
sync_w2r	**99.01**	**100.00**	**100.00**	--	**97.05**	--
fifomem	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	--
rptr_empty	**99.50**	**100.00**	**100.00**	--	**98.52**	--
wptr_full	**99.50**	**100.00**	**100.00**	--	**98.52**	--
afifo_assert_inst	**87.91**	--	--	--	**95.83**	**80.00**

Local Instance Coverage Details:

Total Coverage: 98.07% **98.07%**

Coverage Type	Bins	Hits	Misses	Weight	% Hit	Coverage
Toggles	104	102	2	1	98.07%	**98.07%**

Recursive Hierarchical Coverage Details:

Total Coverage: 98.93% **95.82%**

Coverage Type	Bins	Hits	Misses	Weight	% Hit	Coverage
Statements	29	29	0	1	100.00%	**100.00%**
Branches	14	14	0	1	100.00%	**100.00%**
FEC Conditions	2	2	0	1	100.00%	**100.00%**
Toggles	232	230	2	1	99.13%	**99.13%**
Assertions	5	4	1	1	80.00%	**80.00%**

2. Functional coverage:

write covergroup:

Covergroup type:

cov_write_in

Summary	Total Bins	Hits	Hit %
Coverpoints	7	7	**100.00%**
Crosses	0	0	**0.00%**

Search:

CoverPoints	Total Bins	Hits	Misses	Hit %	Goal %	Coverage %
full_cp	2	2	0	100.00%	**100.00%**	**100.00%**
wdata_cp	3	3	0	100.00%	**100.00%**	**100.00%**
winc_cp	2	2	0	100.00%	**100.00%**	**100.00%**

read covergroup

Covergroup type:

cov_read_out

Summary	Total Bins	Hits	Hit %
Coverpoints	7	7	100.00%
Crosses	0	0	0.00%

Search:

CoverPoints	Total Bins	Hits	Misses	Hit %	Goal %	Coverage %
empty_cp	2	2	0	100.00%	100.00%	100.00
rdata_cp	3	3	0	100.00%	100.00%	100.00
rinc_cp	2	2	0	100.00%	100.00%	100.00

3.3 Assertion Report:

Assertions Coverage Summary:

Search:

Assertions	Failure Count	Pass Count	Attempt Count	Vacuous Count	Disable Count	Active Count	Peak Active Count	Status
/top/DUT/afifo_assert_inst/assert__full_and_empty	0	399	399	0	0	0	1	Covered
/top/DUT/afifo_assert_inst/assert__read_unknown	0	61	200	139	0	0	1	Covered
/top/DUT/afifo_assert_inst/assert__rrst_check	0	0	200	200	0	0	0	ZERO
/top/DUT/afifo_assert_inst/assert__write_unknown	0	200	399	199	0	0	1	Covered
/top/DUT/afifo_assert_inst/assert__wrst_check	0	1	399	398	0	0	1	Covered
/work.afifo_assertions/assert__full_and_empty	0	399	399	0	0	0	1	Covered
/work.afifo_assertions/assert__read_unknown	0	61	200	139	0	0	1	Covered
/work.afifo_assertions/assert__rrst_check	0	0	200	200	0	0	0	ZERO
/work.afifo_assertions/assert__write_unknown	0	200	399	199	0	0	1	Covered
/work.afifo_assertions/assert__wrst_check	0	1	399	398	0	0	1	Covered