VERIFICATION TEST PLAN

Fundamentals of Pre-Silicon Validation Spring - 2024

Design and Verification of Asynchronous FIFO

Achyuth Krishna

Amrutha Regalla

Sathwik Reddy

Sai Sri Harsha Atmakuri

1 Table of Contents:

2 Introduction

2.1 Objective of the Verification plan

The objective of the verification plan for the asynchronous FIFO is to ensure that the design meets its functional requirements and performs reliably under all expected operating conditions. The plan aims to validate the FIFO's functional correctness, ensuring proper first-in, first-out data management and correct functioning of write and read operations, including the correct identification and handling of empty, almost empty full, almost full states. It also focuses on verifying robustness across different clock domains, addressing metastability issues, and ensuring that synchronization logic effectively manages signal transitions between the write and read clock domains. Additionally, the plan seeks to validate the correct implementation and usage of gray code to minimize metastability during address pointer updates.

2.2 Top Level Block Diagram

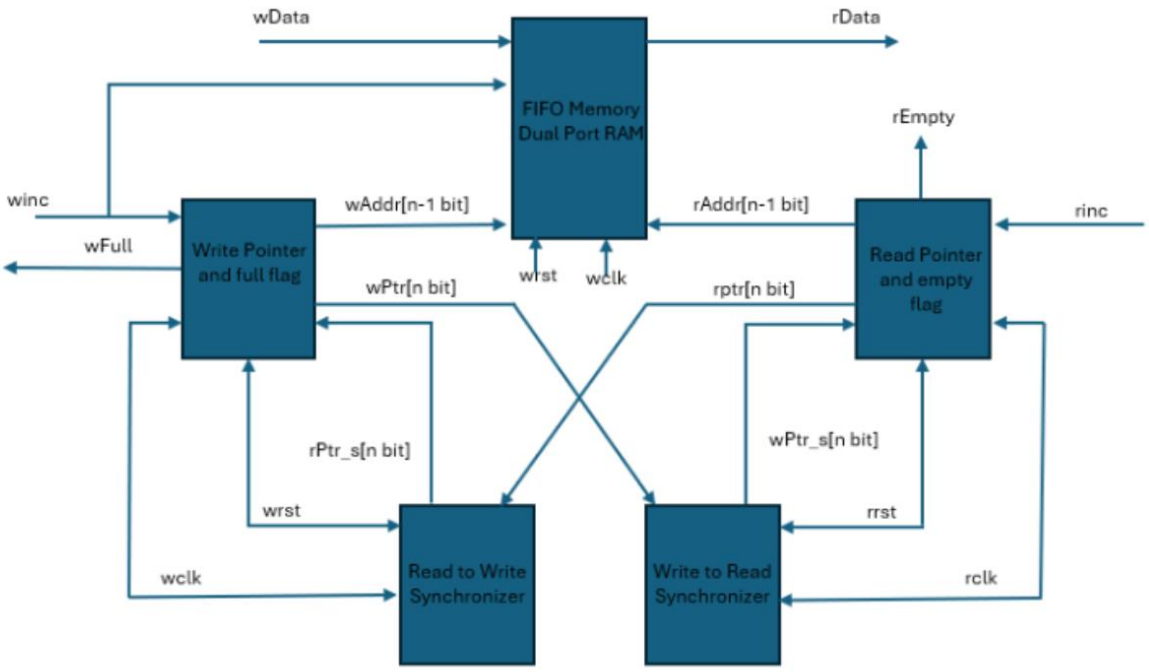

Fig 1: Top Level Block Diagram

2.3 Specifications for the Design

Double the frequency of write and half of the read
Sender Clock Frequency = 120MHz
Number of idle cycles between two successive writes = 3
Receiver Clock Frequency = 50MHz
Number of idle cycles between two successive reads = 2
Write Burst = 1024

Given, Sender Clock Frequency & Receiver Clock Frequency
The number of idle cycles between two successive writes is 3 clock cycles, which means after writing one data, write module is waiting for 3 clock cycles to initiate the next write, meaning every 4 clock cycles one data is written.
The number of idle cycles between two successive reads is 2 clock cycles, which means after reading one data, read module is waiting for 3 clock cycles to initiate next read, meaning every 3 clock cycles on data is read.
Time required to write one data item = 4 * (1/120MHz) = 33.33 ns.
Time required to write the data in the burst = 34,129.92 = 34,129ns.
Time required to read one data item = 3 * (1/50MHz) = 60ns
So, for every 60ns read module is going to read one data item in the burst.
No of data items can be read in a period of 34,129ns = (34,129/60) = 568.81 = 568 items.
Remaining no of bytes to be stored in the FIFO = 1024 – 568 = 456, 600 – 568 = 32
The minimum depth of the FIFO should be 456, rounding to nearest powers of 2 = 512.
Write clock timing = 33.33ns, Read clock timing = 60ns.

3 Verification Requirements

3.1 Verification Levels

3.1.1 Module Hierarchy

We are verifying the FIFO design at the block level as it encapsulates the complete functionality of the FIFO.

3.1.2 Controllability and Observability

Controllability and observability are achieved through dedicated input and output ports of the FIFO module, allowing effective stimulus application and result monitoring.

3.1.3 Interface signals

3.1.3.1 FIFO Module Interface:

Definition: The primary interface of the FIFO module, including input and output ports.
Specifications:
winc: Write enable signal.
wclk: Write clock signal.
wrst: Active-low write reset signal.
rinc: Read enable signal.
rclk: Read clock signal.
rrst: Active-low read reset signal.
wdata: Input data bus for write operations.
rdata: Output data bus for read operations.
full: Output signal indicating FIFO full condition.
empty: Output signal indicating FIFO empty condition

3.1.3.2 Memory Interface (FIFO mem):

Definition: The interface between the FIFO module and the memory subsystem (FIFO mem).

Specifications:

winc: Write enable signal.
full: Input signal indicating FIFO full condition.
wclk: Write clock signal.
waddr: Write address bus.
raddr: Read address bus.
wdata: Input data bus for write operations.
rdata: Output data bus for read operations.

3.1.3.3 Read Pointer Empty Module Interface

Definition: The interface of the module handling read pointer and empty condition.

Specifications:

rinc: Read enable signal.
rclk: Read clock signal.
rrst: Active-low read reset signal.
wptr: Input read pointer with write side.
rempty : Output signal indicating FIFO empty condition.
raddr: Output read address.

3.1.3.4 Write Pointer Full Module Interface (wptr_full):

Definition: The interface of the module handling write pointer and full condition.

Specifications:

winc: Write enable signal.
full: Output signal indicating FIFO full condition.
wclk: Write clock signal.
rptr: Input read pointer with write side.
waddr: Output write address.
wptr: Output write pointer.

3.1.3.5 Synchronization Modules (sync_r2w, sync_w2r):

Definition: Modules ensuring synchronization between read and write pointers.

Specifications:

wclk, wrst_n: Write clock and reset signals.
rptr: Read pointer.
rptr_s, wptr_s: Synchronized read and write pointers.

4 Required Tools

4.1 Software and Hardware Tools

Questasim

4.2 Directory structure of your runs, what computer resources you will be using

The directory structure for the verification runs will be organized for clarity and ease of access. Below is a proposed directory structure:

5 Functions to be Verified.

5.1 Functions from specification and implementation

5.1.1 List of Functions that to be Verified

Write Operation:
Function: Verify that data can be successfully written into the FIFO.
Description: Ensure that the FIFO accepts data when the write enable signal is asserted. Check if the data
is correctly stored in the memory.
Read Operation:
Function: Verify that data can be successfully read from the FIFO.
Description: Ensure that the FIFO provides valid data when the read enable signal is asserted. Check if
the data read matches the expected values.
FIFO Full Condition:
Function: Verify the FIFO full condition.
Description: Write data into the FIFO until it reaches its maximum depth. Verify that the FIFO signals a
full condition correctly.
FIFO Empty Condition:
Function: Verify the FIFO empty condition.
Description: Read data from the FIFO until it becomes empty. Verify that the FIFO signals an empty
condition correctly.
Write Error Handling:
Function: Verify error handling during write operations.
Description: Test scenarios where the FIFO receives write requests beyond its capacity. Ensure that
appropriate error handling mechanisms are in place.
Read Error Handling:
Function: Verify error handling during read operations.
Description: Test scenarios where read requests are made when the FIFO is empty. Ensure that appropriate error handling mechanisms are in place.

Reset Operation:
Function: Verify the reset functionality.
Description: Assert the reset signal and verify that the FIFO resets to its initial state, clearing all stored

data.
Sequential Write and Read:
Function: Verify sequential write and read operations.
Description: Perform a series of sequential write and read operations to ensure proper data flow through the FIFO.
Asynchronous Write and Read:
Function: Verify asynchronous write and read operations.
Description: Introduce variations in write and read timing to ensure that the FIFO can handle asynchronous operations.

6 Tests and Methods

6.1 Testing methods

Black Box Testing: Functional verification based on specifications.
White Box Testing: Code coverage analysis and assertion-based verification.
Gray Box Testing: Scenario-based testing for corner cases.

6.2 PROs and CONs

Black Box Testing: PRO - High-level coverage. CON - Limited visibility into internals.
White Box Testing: PRO - In-depth analysis. CON - May miss system-level issues.
Gray Box Testing: PRO - Comprehensive testing. CON - Increased simulation time.

6.3 Testbench Architecture; Component used (list and describe Drivers, Monitors, scoreboards, checkers etc.)

```
# ----------------------------------------------------------------------------
# Name                          Type                                  Size  Value
# ----------------------------------------------------------------------------
# uvm_test_top                  fifo_test                             -     @472
#   env                         fifo_env                              -     @479
#     cvg_rd                    fifo_read_coverage                    -     @526
#       analysis_imp            uvm_analysis_imp                      -     @533
#     cvg_wr                    wr_fifo_coverage                      -     @507
#       analysis_imp            uvm_analysis_imp                      -     @514
#     rd_agnt                   read_agent                            -     @493
#       rd_drv                  read_driver                           -     @546
#         rsp_port              uvm_analysis_port                     -     @561
#         seq_item_port         uvm_seq_item_pull_port                -     @553
#       rd_mon                  read_monitor                          -     @678
#         rd_monitor_port       uvm_analysis_port                     -     @686
#       rd_seqr                 read_sequencer                        -     @569
#         rsp_export            uvm_analysis_export                   -     @576
#         seq_item_export       uvm_seq_item_pull_imp                 -     @670
#         arbitration_queue     array                                 0     -
#         lock_queue            array                                 0     -
#         num_last_reqs         integral                              32    'd1
#         num_last_rsps         integral                              32    'd1
#     scb                       fifo_scoreboard                       -     @500
#       scoreboard_read_port    uvm_analysis_imp_rd_monitor_port      -     @708
#       scoreboard_write_port   uvm_analysis_imp_monitor_port         -     @700
#     wr_agnt                   write_agent                           -     @486
#       wr_drv                  write_driver                          -     @717
#         rsp_port              uvm_analysis_port                     -     @732
#         seq_item_port         uvm_seq_item_pull_port                -     @724
#       wr_mon                  write_monitor                         -     @849
#         monitor_port          uvm_analysis_port                     -     @857
#       wr_seqr                 write_sequencer                       -     @740
#         rsp_export            uvm_analysis_export                   -     @747
#         seq_item_export       uvm_seq_item_pull_imp                 -     @841
#         arbitration_queue     array                                 0     -
#         lock_queue            array                                 0     -
#         num_last_reqs         integral                              32    'd1
#         num_last_rsps         integral                              32    'd1
# ----------------------------------------------------------------------------
```

6.4 Verification Strategy: (Dynamic Simulation, Formal Simulation, Emulation etc.) Describe why you chose the strategy

Verification Strategy: Dynamic Simulation.
Reasoning: Dynamic simulation balances accuracy and efficiency for FIFO verification.

6.5 Driving Methodology

6.5.1 List the test generation methods (Directed test, constrained random)

Test Generation Methods: Directed Testing and Constraint random testing.
Direct Testing involves creating targeted test cases based on specific design requirements, while Constraint Random Testing (CRT) uses randomized stimulus generation guided by constraints to explore the design space more thoroughly. Both techniques play vital roles in verifying digital designs, with Direct Testing focusing on known requirements and CRT efficiently exploring diverse test scenarios to uncover corner cases and unexpected behaviours.

6.5.2 Checking Methodology

1. **Self-Checking Testbench**: Automates the process of stimulus generation and response verification without manual intervention, using a testbench that compares the DUT's outputs against expected results.
2. **Score boarding**: Utilizes a reference model to predict outputs, facilitating comparison against actual DUT responses for validation of correct behaviour.
3. **Coverage Analysis**: Measures the extent to which the DUT's state space is exercised by the testbench, aiming to identify untested parts of the design.
4. **Randomized Testing**: Generates random inputs to stress-test the DUT across a broad range of scenarios, often uncovering edge cases not considered in directed testing.
5. **Directed Testing**: Focuses on specific and critical DUT functionalities by crafting targeted test cases, ensuring that particular behaviours are correctly implemented.
6. **Logging and Error Reporting**: Captures and outputs simulation data and error messages to track test progress and facilitate debugging of failed checks.

6.6 Testcase Scenarios (Matrix)

6.6.1 Basic Tests

Test Name / Number	Test Description/ Features
1.1.1 Top Module: Write and Read	Base test
1.1.2 FIFO Memory Array check	Check Read and Write into Array at module level
1.1.3 FIFO Memory Full Flag Detection	Checks to see if an asserted full flag
1.1.4 Write Pointer increment	Write Pointer must increment on winc
1.1.5 Read Pointer increment	Read Pointer must increment on rinc
1.1.6 Reset Operation	Reset operation from Top module must propagate through the entire design

6.6.2 Complex Tests

Test Name / Number	Test Description/ Features
1.2.1 Read to Write Synchronizer pipeline	Ensures a Pointer passed is synchronized during clock domain crossings
1.2.2 Write to Read Synchronizer pipeline	Ensures a Pointer passed is synchronized during clock domain crossings
1.2.3 Full Flag generation	Ensures Full flag generation logic functions properly on write pointer wrap around
1.2.4 Empty Flag generation	Ensures Empty flag generation logic functions properly on read pointer equal to write pointer

7 Coverage Requirements

7.1 Describe Code and Functional Coverage goals for the DUV.

1. **Bit Coverage of wdata**:
   - Verify that each bit of **bus_tb.wdata** toggles between 0 and 1.
   - Ensure that each bit is exercised individually to catch any potential stuck-at faults or other issues.
2. **Control Signals**:
   - **Read and Write Increment Signals (rinc, winc):**
     - Verify that these signals applied correctly during read and write operations.
   - **Reset Signals (wrst, rrst):**
     - Ensure that the reset signals transition properly from active to inactive states and vice versa.
     - Confirm that the DUT resets and initializes as expected.
3. **FIFO Full and Empty Signals**:
   - **full Signal:**
     - Validate that the FIFO full signal toggles correctly when the FIFO reaches its maximum capacity.
   - **empty Signal:**
     - Ensure that the FIFO empty signal toggles correctly when the FIFO is empty and ready to accept new data.
     - Test scenarios where the FIFO transitions between empty, partially full, and full states.
4. **Functional Coverage**:
   - **Read and Write Operations**:
     - Ensure that all read and write operations are exercised, including edge cases and corner scenarios.
   - **Data Integrity and Error Handling**:
     - Validate that data integrity is maintained throughout read and write operations.
     - Test error handling mechanisms, such as overflow and underflow conditions.

Additional Considerations:

- **Randomization**: Ensure that the test sequences incorporate randomization to cover a wide range of scenarios.
- **Functional Coverage**: Include bins that cover functional aspects of the DUT's behavior, such as read and write operations, error handling, and boundary conditions.
- **Corner Cases**: Define bins that cover corner cases and extreme values to validate the DUT's behavior under adverse conditions.

8 References Uses / Citations/Acknowledgements

1. S. Cummings, "FIFOs: Fast, predictable, and deep," in Proceedings of SNUG, 2002. [Online]. Available: http://www.sunburst-design.com/papers/CummingsSNUG2002SJ_FIFO1.pdf.

2. S. Cummings, "FIFOs: Fast, predictable, and deep (Part II)," in Proceedings of SNUG, 2002. [Online]. Available: http://www.sunburst-design.com/papers/CummingsSNUG2002SJ_FIFO2.pdf.

3. Putta Satish, "FIFO Depth Calculation Made Easy," [Online]. Available: https://hardwaregeeksblog.files.wordpress.com/2016/12/fifodepthcalculationmadeeasy2.pdf.

4. A. Author et al., "Title of the Paper," in Proceedings of the Conference, 2015, pp. 123-456. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7237325.

5. M. Last Name et al., "Designing Asynchronous FIFO," [Online]. Available: https://d1wqtxts1xzle7.cloudfront.net/56108360/EC109-libre.pdf.

6. A. Author et al., "Title of the Paper," in Proceedings of the Conference, 2011, pp. 789-012. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6041338

7. Author, "Title of the Video," [Online]. Available: https://www.youtube.com/watch?v=UNoCDY3pFh0

8. Open AI, "Chat GPT," [Online]
9. Professor Slides