
Verification Test Plan

for

Asynchronous FIFO

Version 1.0

ECE-593: Fundamentals of Pre-Silicon Validation – Venkatesh Patil

03 February 2024

Prepared by:

Snigdha Reddy Baddam

Venkata Ramana Molabanti

Sri Hari Anne

Acknowledgement

The block diagram and the implementation of gray code counters was taken from the paper “Clifford E. Cummings and Peter Alfke, “Simulation and Synthesis Techniques for Asynchronous FIFO Design with Asynchronous Pointer Comparisons,” *SNUG 2002 (Synopsys Users Group Conference, San Jose, CA, 2002) User Papers*, March 2002, Section TB2, 3rd paper.” and modified for this specifications of the project. Also available at www.sunburst-design.com/papers

Table of Contents

1	Introduction.....	5
1.1	Objective of the verification plan.....	5
1.2	Top Level block diagram	6
1.3	Specifications for the design	6
2	Verification Requirements	7
2.1	Verification Levels.....	7
2.1.1	What hierarchy level are you verifying and why?	7
2.1.2	How is the controllability and observability at the level you are verifying?	7
2.1.3	Are the interfaces and specifications clearly defined at the level you are verifying? List them. 8	
3	Required Tools	8
3.1	List of required software and hardware tool sets needed.	8
3.2	Directory structure of your runs, what computer resources you will be using.	8
4	Risks and Dependencies.....	9
4.1	List all the critical threats or any known risks. List contingency and mitigation plans.	9
5	Functions to be Verified.	9
5.1	Functions from specification and implementation.....	9
5.1.1	List of functions that will be verified. Description of each function.	9
5.1.2	List of functions that will not be verified. Description of each function and why it will not be verified.	10
5.1.3	List of critical functions and non-critical functions for tapeout.....	10
6	Tests and Methods	10
6.1.1	Testing methods to be used: Black/White/Gray Box.	10
6.1.2	Testbench Architecture; Component used (list and describe Drivers, Monitors, scoreboards, checkers etc.)`	11
6.1.3	Verification Strategy: (Dynamic Simulation, Formal Simulation, Emulation etc.)	11
6.1.4	Driving methodology	11
6.1.5	Checking methodology?	11
6.1.6	Testcase Scenarios (Matrix).....	12
7	Results:.....	13
8	Coverage Requirements	16
8.1.2	Assertions.....	16
9	Resources requirements	16

10	Schedule	17
11	References.....	18

1 Introduction

1.1 Objective of the verification plan

The objective of the verification plan for the asynchronous FIFO design is to systematically validate the functionality, performance, and robustness of the FIFO implementation. The primary goals are as follows:

Functional Verification –

- Verifying FIFO works properly for the movement of data based on first in first out.
- Verifying fifo-full, fifo-empty conditions work properly.
- Verifying read pointer, write pointer is satisfying the conditions.

Asynchronous Operation –

- Validating the data is the same between write and read domains.
- Verify that the design handles data transfer asynchronously, allowing for variations in clock frequencies between the read and write interfaces.

Metastability Condition –

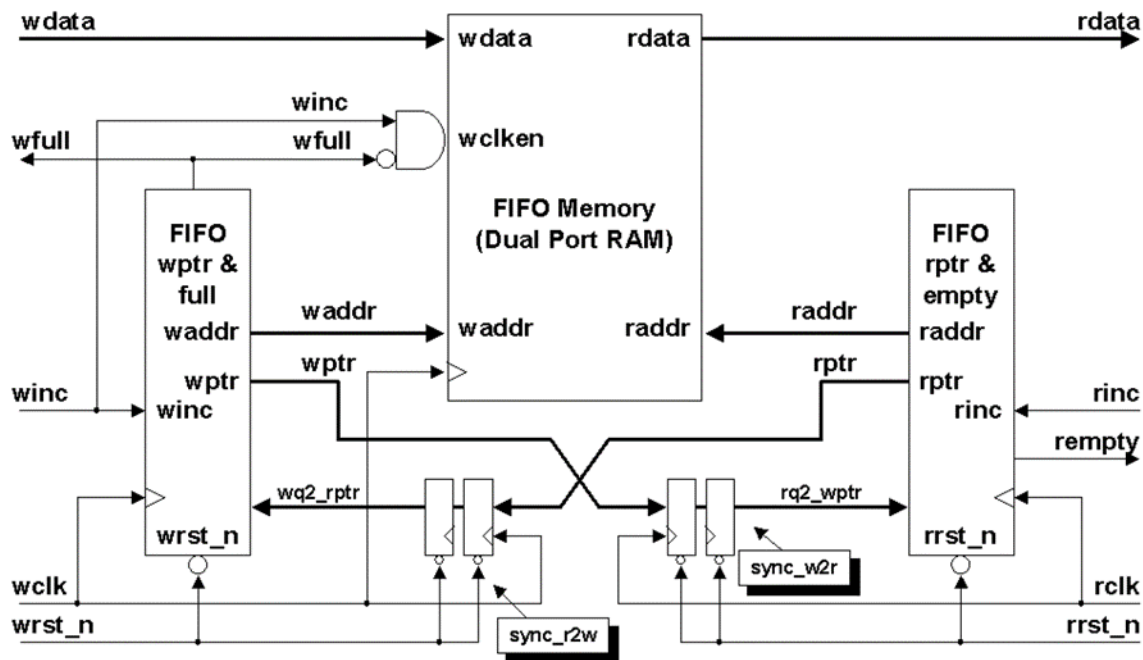
- Evaluate the design's capability to manage metastability challenges arising from asynchronous inputs and confirm the integration of suitable synchronization methods.

Cross-Clock Domain Signals –

- Identify and validate signals that cross clock domains, such as handshaking signals or flags indicating FIFO status.

The verification plan aims to provide comprehensive coverage of the asynchronous FIFO's functional and ensure a high level of confidence in its correct operation and performance across diverse usage scenarios.

1.2 Top Level block diagram



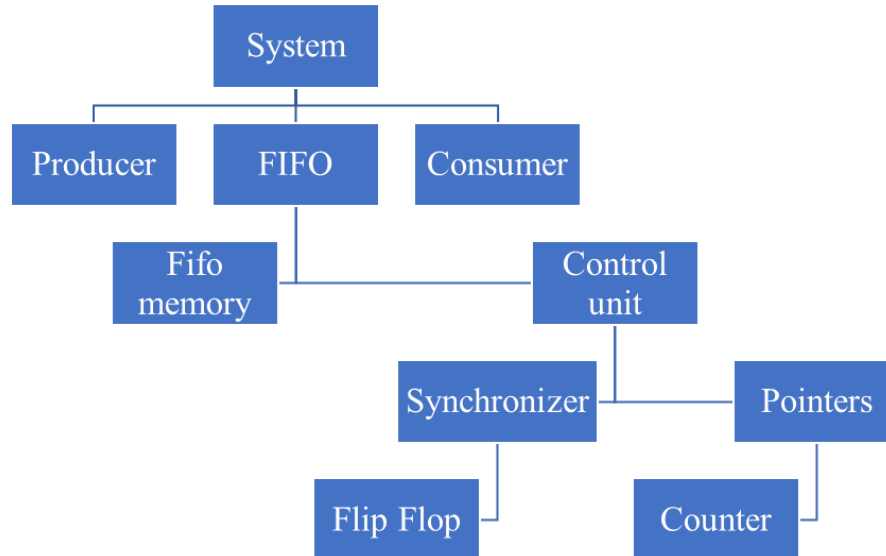
1.3 Specifications for the design

- Producer Frequency: 500MHz
- Consumer Frequency: 500MHz
- Burst Length: 450
- Idle Cycles for Write: 0
- Idle Cycles for Read: 4
- FIFO Depth: 360
- Duty Cycle (Read/Write): 50%
- The data typically needs a positive edge for both read and write.
- The interaction between the two clock cycles happened by using handshaking signals such as FIFO_EMPTY and FIFO_FULL etc.
- To avoid the metastability conditions synchronizers are used. Based on the design conditions two synchronizers are used.
- RESET condition is used to reset the pointers to the initial conditions.
- Counters like write pointer and read pointer are used for managing the data in the FIFO memory.

2 Verification Requirements

2.1 Verification Levels

Verification is done at top level.



2.1.1 Controllability and observability at the top level

Controllability

- The data should write whenever **wren** is high
- Whenever **fifo_full** is high **wren** should become low.
- Read and write operations should be performed concurrently.
- If **wrst_n** is asserted the **wptr** and **winc** should come to the initial level of FIFO.
- There are no idle cycles. So the FIFO should get the data without any idle cycles in between.

Observability

- The data whatever is written in FIFO should be equal to the data which is read.
- Checking the FIFO empty condition, whenever **rrst_n** is asserted **rempty** should be high.

2.1.2 Interfaces and specifications at top level

Interfaces :

The mentioned interfaces are Producer and Consumer. Producer is the interface which is used to write the data to FIFO based on the control unit and flags. Consumer is also the other interface which reads the data from the FIFO based on the control unit and flags.

Specifications :

- The important specifications defined for the FIFO design are the clock frequencies and idle cycles. Based on the frequencies and idle clock cycles FIFO depth is calculated.
- Margin is considered as 5-10% on the fifo depth.
- The basic block diagram is provided and gives the basic idea about the design.

Overall, the interfaces and specifications are good but according to the working of the FIFO there is a bit of ambiguity. The control unit and the logics regarding the fifo_full and fifo_empty, utilization of burst length, idle cycles.

3 Required Tools

3.1 List of required software and hardware tool sets needed.

Hardware tool sets :

None

Software Tool sets:

QuestaSim

Windows OS

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

Source Code :

top.sv

fifo_mem.sv

full.sv

empty.sv

sync_r2w.sv

sync_w2r.sv

Test bench :

tb.sv
fifo_UVM_tb.sv (WIP)

Scripts :

run.do

4 Risks and Dependencies

4.1 Critical threats or any known risks and contingency and mitigation plans.

Clock Domain Crossing Issues:

Risk: Metastability issues due to asynchronous clock domains may lead to data corruption or loss.

FIFO Depth Mismatch:

Risk: Incorrectly configured FIFO depth may lead to overflow or underflow issues.

Insufficient Controllability and Observability:

Risk: Lack of effective controllability and observability may result in incomplete verification coverage.

Incorrect Handshaking and Protocol Violations:

Risk: Issues in handshaking signals or protocol violations may disrupt communication between the producer and consumer.

5 Functions to be Verified.

5.1 Functions from specification and implementation

5.1.1 List of functions that will be verified and description of each function.

- Data addition to the FIFO involves consecutive writes until the **winc** (write increment) reaches the tail position.
- The initial position of **winc** is at the head of the FIFO, and upon writing data, it increments and points to the next location.
- When **winc** reaches the tail, **fifo_full** is asserted, indicating that the FIFO cannot accept additional data.
- If **wrst_n** is asserted, **winc** resets to the head position, allowing whatever data present will be cleared and data to be written again from the initial position.
- The FIFO operates asynchronously, ensuring proper synchronization and seamless write and read operations without discrepancies.
- Concurrent read and write operations are supported, maintaining data integrity during simultaneous access.

- Read operations are performed on previously written data, with **rinc** (read increment) pointing to the next data item to be read.
 - If **rinc** is at the head position, **fifo_empty** is asserted, indicating that the FIFO is empty.
 - If **rrst_n** is asserted, **rinc** resets to the initial position by clearing the data in the fifo, allowing new read operations.
 - **almost_full** and **almost_empty** signals are asserted based on the number of data items present in the FIFO, providing warnings for data is almost full and almost empty conditions.
- 5.1.2 List of functions that will not be verified. Description of each function and why it will not be verified.
- This function checks if the FIFO queue is empty. Not verified because it only provides a snapshot of the current state and may become outdated immediately.
- 5.1.3 List of critical functions and non-critical functions for tapeout
- Critical functions:
- **Write clock domain crossing:** Ensuring data integrity when transferring data between clock domains.
 - **Data synchronization:** Proper synchronization of data between the write and read sides.
 - **Empty and full detection:** Accurate detection of FIFO empty and full states.
 - **Almost-Full/Almost-Empty Indicators:** Accurate signaling for almost-full and almost-empty conditions.

Non-Critical Functions

Will be delivered in future milestones.

6 Tests and Methods

6.1.1 Testing methods to be used: Black/White/Gray Box.

Black Box Tesitng:

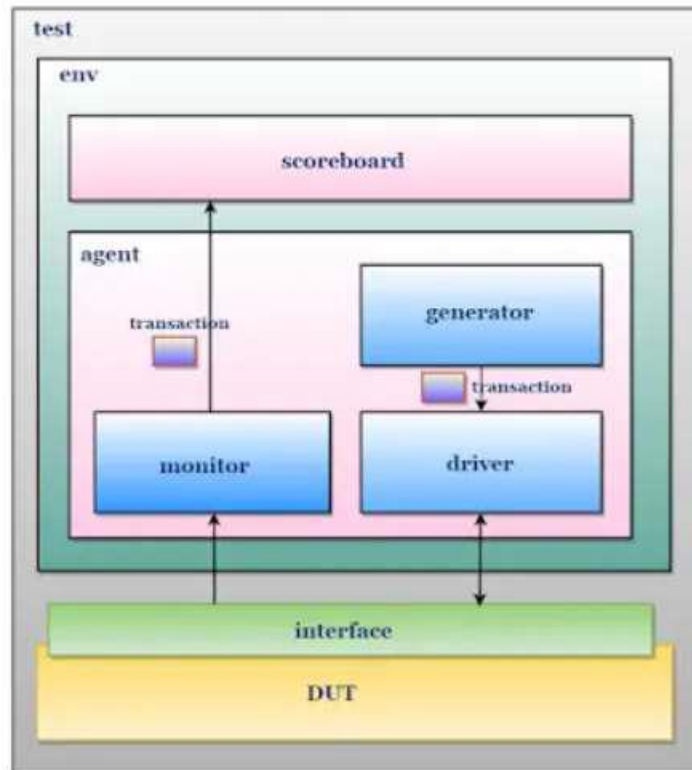
- Testing the asynchronous operation with concurrent read and write operations
- Testing the flags is asserting properly in the specified scenarios.
- Testing the read and write operations for data integrity.
- Testing the burst length is going into fifo within the given specifications.
- Testing the design is working in the specified frequency and able to generate the exact waveforms related to the design

White Box Testing:

- For Assertions in future testbenches, this testing will be used to have access to internal variable values.

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

For Milestone 2, we plan to use the testbench architecture shown below. More details will be provided with Milestone 2 submission.



6.1.3 Verification Strategy: (Dynamic Simulation, Formal Simulation, Emulation etc.)

We chose simulation for testing the functionality considering the nature of the design and the ease of upgrading the testbenches for future milestones.

6.1.4 Driving methodology

6.1.4.1 Some of the testcases were direct. Some functions were verified using random stimulus.

6.1.5 Checking methodology?

6.1.5.1 Assertions were used to verify the tests. Some were verified using waveforms (screenshots provided below)

6.1.6 Testcase Scenarios (Matrix)

Basic Tests

Test Name / Number	Test Description/ Features
1.1.1 Read	Check basic read operation
1.1.2 Read and write	Verifying data write and read in the expected order with the same data

Complex Tests

Test Name / Number	Test Description/ Features
1.2.1	Concurrent events (R+W) Conditions: fifo_full/fifo_empty/always_full/always empty etc.
1.2.2 reset	Verifying reset conditions
1.2.3 extreme levels	Verifying fifo overflow and underflow levels
12.4 concurrent reset	Verifying concurrent reset conditions

Regression Tests (Must pass every time)

Test Name / Number	Test Description/Features
1.3.1	Verifying read and write operations without errors
1.3.2	Verifying fifo full and fifo empty conditions after read and write operations
1.3.3	Verifying that same data should be read whatever the data write is happened

Corner cases testcases

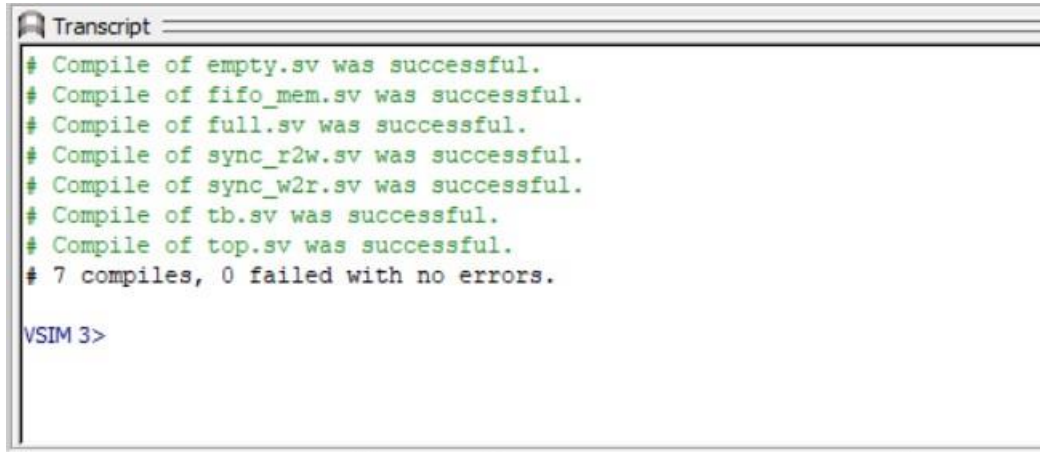
Test Name / Number	Test Description
1.4.1	Special Case testing tests and conditions
1.4.2	Bug injection and testing scenario
1.4.3	To detect stuck at 0 or stuck at 1 faults, write all 0's and 1's to all locations
1.4.4	setting almost empty and almost full at extreme levels

NOTE: Some test cases were not implemented for this milestone but will be part of class based and UVM testbenches.

7 Results:

Transcript output Screenshot:

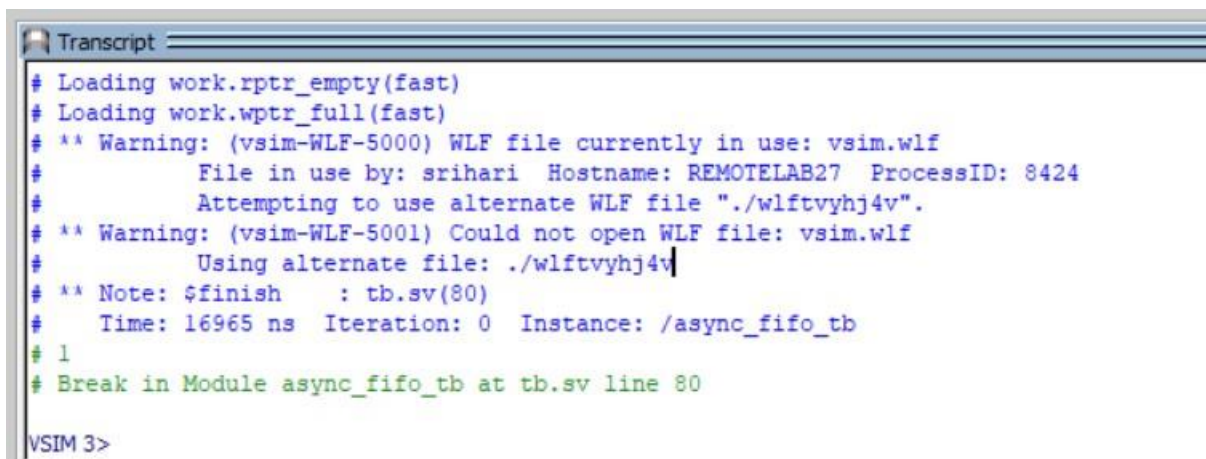
Compile:



```
Transcript
# Compile of empty.sv was successful.
# Compile of fifo_mem.sv was successful.
# Compile of full.sv was successful.
# Compile of sync_r2w.sv was successful.
# Compile of sync_w2r.sv was successful.
# Compile of tb.sv was successful.
# Compile of top.sv was successful.
# 7 compiles, 0 failed with no errors.

VSIM 3>
```

Simulation output (No assertion failures)

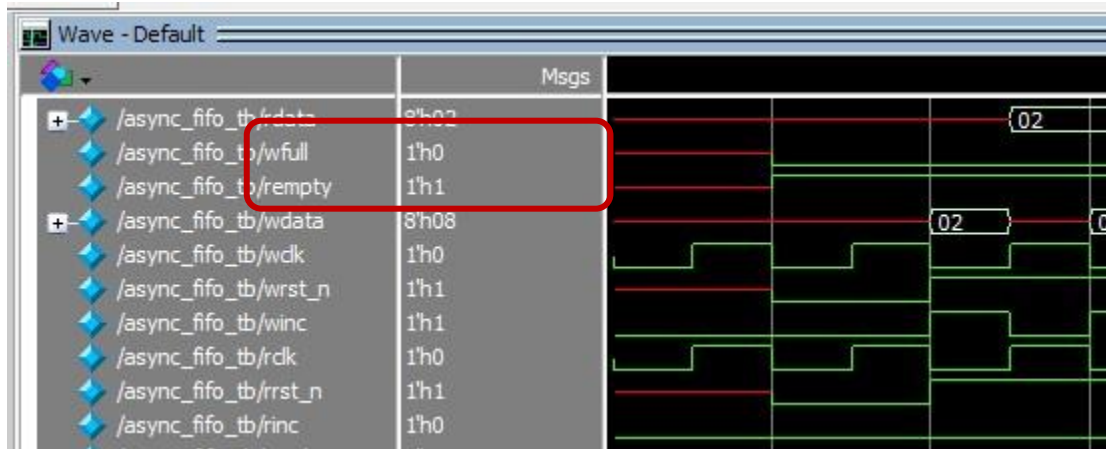


```
Transcript
# Loading work.rptr_empty(fast)
# Loading work.wptr_full(fast)
# ** Warning: (vsim-WLF-5000) WLF file currently in use: vsim.wlf
#       File in use by: srihari  Hostname: REMOTELAB27  ProcessID: 8424
#       Attempting to use alternate WLF file "./wlftvyhj4v".
# ** Warning: (vsim-WLF-5001) Could not open WLF file: vsim.wlf
#       Using alternate file: ./wlftvyhj4v
# ** Note: $finish      : tb.sv(80)
#       Time: 16965 ns  Iteration: 0  Instance: /async_fifo_tb
# 1
# Break in Module async_fifo_tb at tb.sv line 80

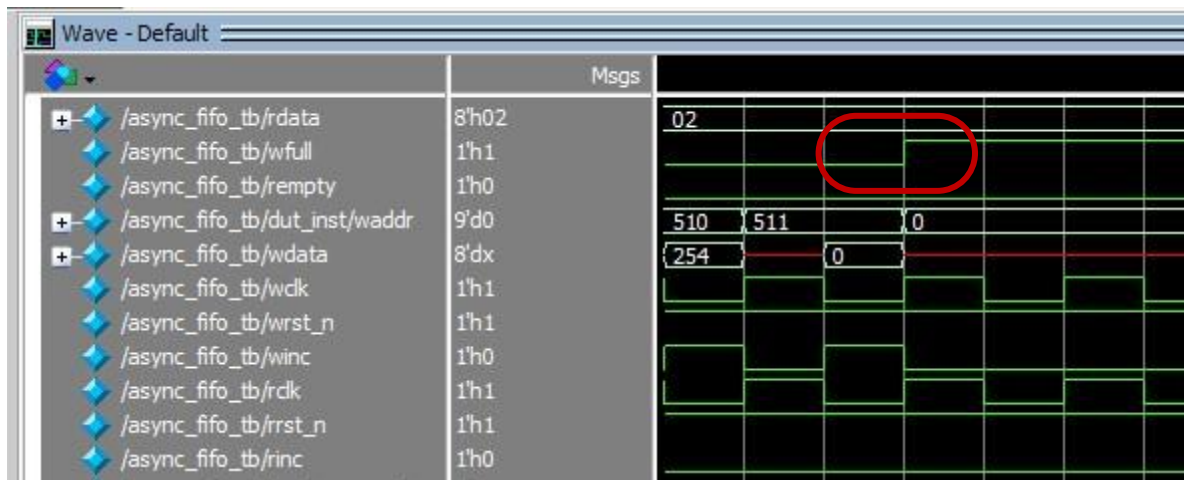
VSIM 3>
```

Waveforms:

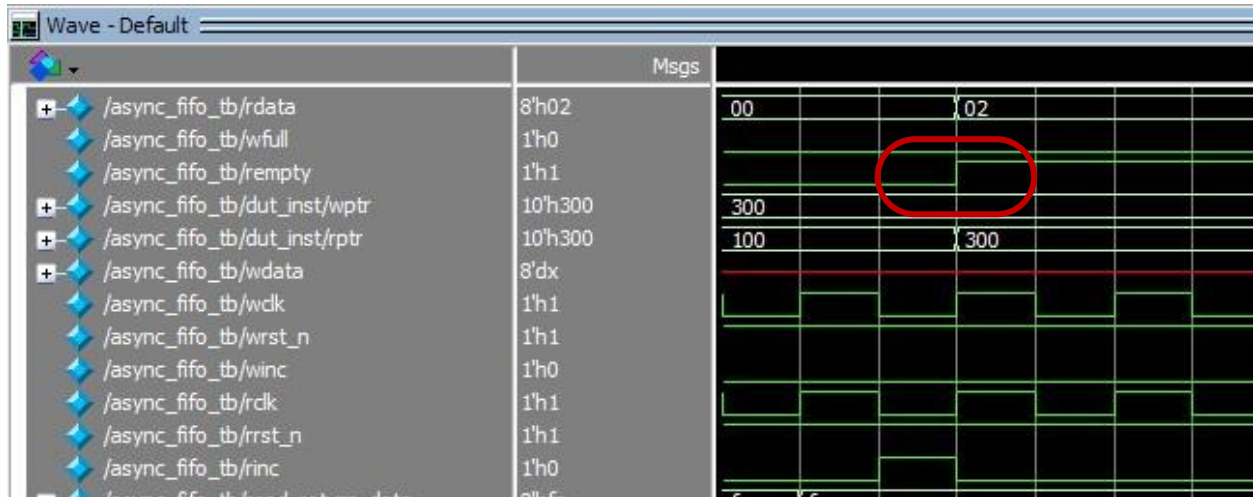
Full and empty on reset:



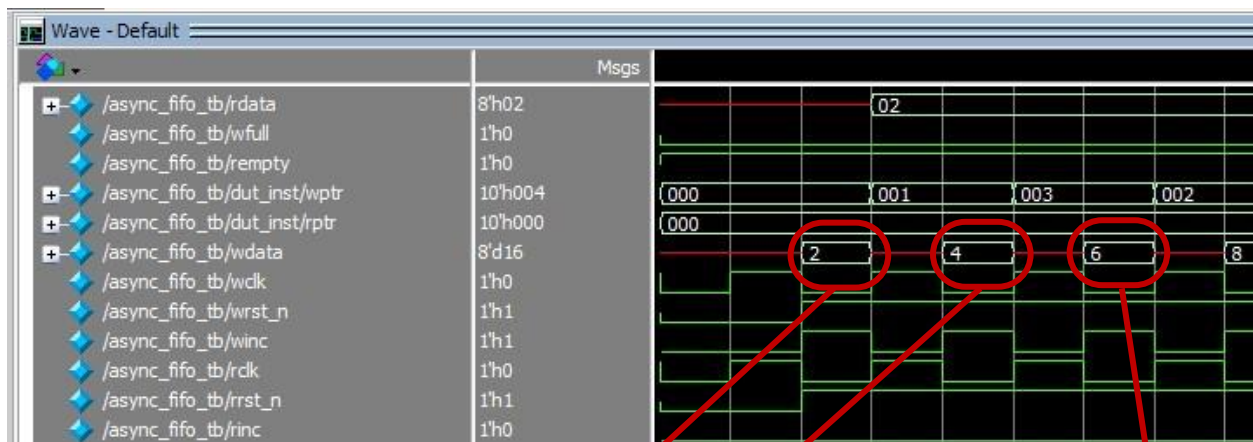
Full asserted when FIFO is full:



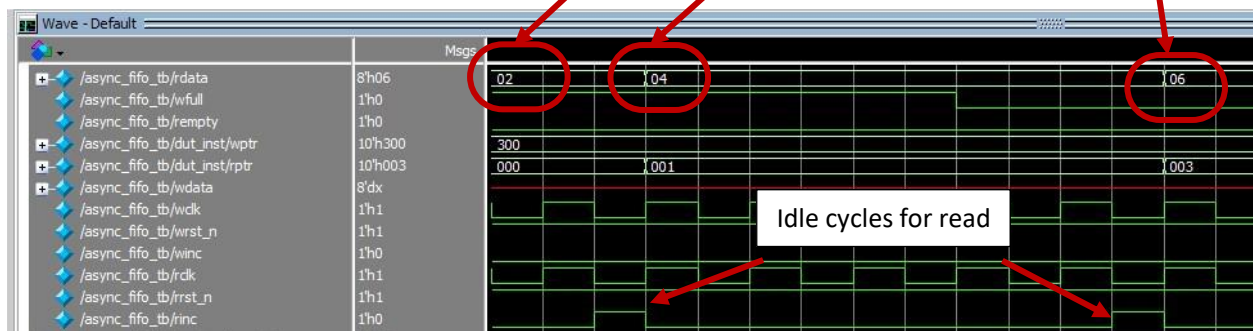
Empty asserted after all the items are read from FIFO:



Write data:



Read data:



8 Coverage Requirements

8.1.1.1 Code and Functional Coverage goals for the DUV

Will be added for Milestone 2

8.1.2 Assertions

Will be added for Milestone 2

9 Resources requirements

For Milestone 1:

Venkata Ramana : Verification Plan and first draft of HLS

Snigdha : Sanity Test bench and also discussing with Ramana on bringing up the Val plan.

Sri Hari : Taking inputs from other Team mates and building up the RTL.
Improving the sanity testbench.

General expertise:

Venkata Ramana : Verification Plan and ideas

Snigdha : Testbench coding

Sri Hari : Assertions and bug fixes for RTL

10Schedule

Task Description	Completion Status
Milestone 1: Submission - 2/3/2024	
High-Level Design (HLD)	✓
Verification Plan	✓
Basic RTL Program (Working)	✓
Milestone 2: Submission - 2/10/2024	
Detailed RTL Design	WIP
Initial Testbench Setup	WIP
Milestone 3: Submission - 2/20/2024	
Implement Class-based Testbench	
Add Synchronization Mechanisms	
Milestone 4: Submission - 3/1/2024	
Develop UVM-Based Testbench	
Perform Testing	
Milestone 5: Final Submission - 3/7/2024	
Finalize Documentation	
Conduct Comprehensive Testing	
Address Bug Fixes and Optimizations	

11 References

- [1] Clifford E. Cummings and Peter Alfke, “Simulation and Synthesis Techniques for Asynchronous FIFO Design with Asynchronous Pointer Comparisons,” *SNUG 2002 (Synopsys Users Group Conference, San Jose, CA, 2002) User Papers*, March 2002, Section TB2, 3rd paper. Also available at www.sunburst-design.com/papers
- [2] Clifford E. Cummings, “Simulation and Synthesis Techniques for Asynchronous FIFO Design,” *SNUG 2002 (Synopsys Users Group Conference, San Jose, CA, 2002) User Papers*, March 2002, Section TB2, 2nd paper. Also available at www.sunburst-design.com/papers
- [3] Putta Satish, “CALCULATION OF FIFO DEPTH MADE EASY” Available at <https://hardwaregeeksblog.files.wordpress.com/2016/12/fifodepthcalculationmadeeasy2.pdf>
- [4] FIFO Architecture, Functions, and Applications by Texas Instruments. Available at <https://www.ti.com/lit/an/scaa042a/scaa042a.pdf>
- [5] W. J. Dally EE 108a Lecture 13: Metastability And Synchronization Failure (Or When Good Flip-Flops Go Bad), *Stanford University Lectures*. Also available at <http://cva.stanford.edu/people/davidbbs/classes/ee108a/winter0607%20labs/lect.9.Metastability-blackschaffer.ppt>