**VERIFICATION TEST PLAN**

ECE-593: Fundamentals of Pre-Silicon Validation  
Maseeh College of Engineering and Computer Science  
Spring, 2025

A close up of a logo

AI-generated content may be incorrect.

**Project Name:** Design and verification of Asysnchrnous FIFO  
**Team\_8 Members:** Vaishnavi Pandhare

Rohith Sri Krishna Inti

Mrudula Chekuri

Venkat Sahith Reddy Cheedu

**Date:** April 20, 2025

Table of Contents

[1](#_heading=h.gjdgxs) Introduction: 3

[1.1](#_heading=h.30j0zll) Objective of the Verification plan 3

[1.2](#_heading=h.1fob9te) Top Level block diagram 3

[1.3](#_heading=h.3znysh7) Design Specifications 4

[2](#_heading=h.2et92p0) Verification Requirements 5

[2.1](#_heading=h.tyjcwt) Verification Levels 5

[2.1.1](#_heading=h.3dy6vkm) What hierarchy level are you verifying and why? 5

[2.1.2](#_heading=h.1t3h5sf) How is the controllability and observability at the level you are verifying? 5

[2.1.3](#_heading=h.4d34og8) Are the interfaces and specifications clearly defined at the level you are verifying. List them. 5

[3](#_heading=h.2s8eyo1) Required Tools 6

[3.1](#_heading=h.17dp8vu) List of required software and hardware toolsets needed. 6

[3.2](#_heading=h.3rdcrjn) Directory structure 6

[4](#_heading=h.26in1rg) Risks and Dependencies 6

[5](#_heading=h.lnxbz9) Functions to be Verified 7

[5.1](#_heading=h.35nkun2) Functions Included 7

[5.2](#_heading=h.1ksv4uv) Critical vs Non-Critical functions 7

[6](#_heading=h.z337ya) Tests and Methods 8

[6.1](#_heading=h.3j2qqm3) Testing Methodologies 8

[6.1.1](#_heading=h.1y810tw) Black box approach 8

[6.1.2](#_heading=h.4i7ojhp) Pros and Cons 8

[6.1.3](#_heading=h.2xcytpi) Verification Strategy 8

[6.2](#_heading=h.3o7alnk) Testcase Scenarios (Matrix) 8

[7](#_heading=h.1v1yuxt) Resources requirements 9

[7.1](#_heading=h.1mrcu09) Team members and who is doing what and expertise. 9

[8](#_heading=h.4f1mdlm) Schedule 9

[8.1](#_heading=h.2u6wntf) Milestone 1 9

[8.2](#_heading=h.19c6y18) Milestone 2 9

[8.3](#_heading=h.3tbugp1) Milestone 3 9

[8.4](#_heading=h.28h4qwu) Milestone 4 9

[8.5](#_heading=h.nmf14n) Milestone 5 9

9References Uses / Citations/Acknowledgements 10

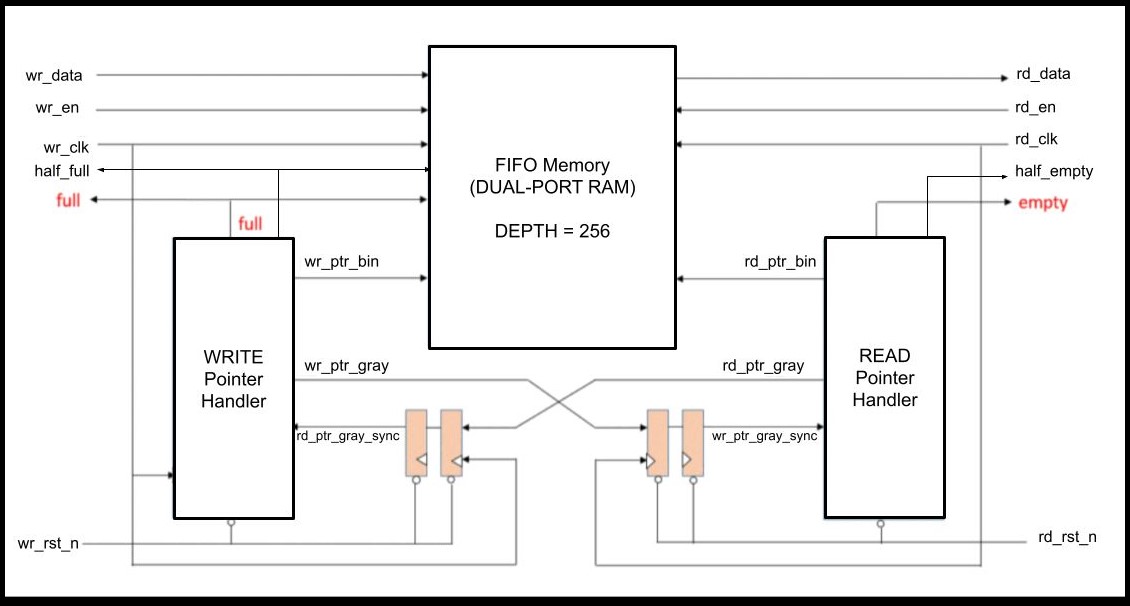
# Introduction:

This document outlines the verification plan for the Asynchronous FIFO designed by Team 8.

## Objective of the Verification Plan

The purpose of this plan is to comprehensively verify the functionality, performance, and synchronization robustness of the asynchronous FIFO design. Specifically, we will confirm correct operation under a dual-clock domain: 500 MHz write clock and 250 MHz read clock, validate status flags (full, half\_full, empty, half\_empty) across burst transactions (burst length = 1024), and ensure that data integrity is maintained with depth = 256.

## Top Level block diagram

Below is the custom block diagram for our async‑FIFO design. It highlights the dual‑port memory, write and read pointer handlers, and two‑stage CDC synchronizers for pointer crossing between the 500 MHz write and 250 MHz read domains.  
  


**Figure 1.1:** Async FIFO Top‑Level Architecture

Key elements shown in the diagram:

* **FIFO Memory (Dual‑Port RAM):** 256‑deep, 32‑bit width, supports simultaneous write and read.
* **Write Pointer Handler:** Drives wr\_ptr\_bin and wr\_ptr\_gray; synchronizes incoming rd\_ptr\_gray via two flip‑flop CDC (rd\_ptr\_gray\_sync1\_w, rd\_ptr\_gray\_sync2\_w). Generates full and half\_full flags.
* **Read Pointer Handler:** Drives rd\_ptr\_bin and rd\_ptr\_gray; synchronizes incoming wr\_ptr\_gray via two flip‑flop CDC (wr\_ptr\_gray\_sync1\_r, wr\_ptr\_gray\_sync2\_r). Generates empty and half\_empty flags.
* **Status Flags:** full asserted when the write pointer in Gray domain is about to overlap the synchronized read pointer. half\_full when occupancy ≥128. empty when synchronized write pointer equals read pointer Gray. half\_empty when occupancy ≤128.

## Design Specifications

* **Data Width:** 32 bits
* **Depth:** 256 entries (ADDR\_WIDTH = 8 + 1 bit extra MSB)
* **Write Clock (wr\_clk):** 500 MHz (2 ns period)
* **Read Clock (rd\_clk):** 250 MHz (4 ns period)
* **Idle Cycle Assumptions:** write idle = 2 cycles, read idle = 1 cycle;
  + Effective write idle period = 3×2 ns = 6 ns
  + Effective read idle period = 2×4 ns = 8 ns
* **Burst Length:** 1024 words
* **FIFO Depth Calculation:**
  + Total write time = 1024 × 6 ns = 6144 ns
  + Words read in that time = 6144 ns ÷ 8 ns = 768
  + Depth required = 1024 – 768 = 256 entries
* **Key Functions:** Gray↔Binary pointer conversion, CDC synchronization, status flag generation.

# Verification Requirements

## Verification Levels

### What hierarchy level are you verifying and why?

We will verify the async\_fifo module at the RTL block level, as it encapsulates all logic for memory, pointer manipulation, CDC, and flag generation. Verifying at this level allows full control over stimuli and direct observation of internal state.

### How is controllability and observability at the level you are verifying?

**Controllability:** All primary inputs (wr\_clk, wr\_rst\_n, wr\_en, wr\_data, rd\_clk, rd\_rst\_n, rd\_en) are directly driven by the testbench.

**Observability:** Primary outputs (rd\_data, full, half\_full, empty, half\_empty) and internal pointers (wr\_ptr\_bin, rd\_ptr\_bin) are monitored.

### Are the interfaces and specifications clearly defined at the level you are verifying. List them.

**Interface & Specification Listing**

|  |  |  |  |
| --- | --- | --- | --- |
| **Signal** | **Direction** | **Width** | **Description** |
| wr\_clk | input | 1 | Write-side clock (500 MHz) |
| wr\_rst\_n | input | 1 | Active-low reset for write domain |
| wr\_en | input | 1 | Write enable |
| wr\_data | input | 32 | Data bus for write |
| rd\_clk | input | 1 | Read-side clock (250 MHz) |
| rd\_rst\_n | input | 1 | Active-low reset for read domain |
| rd\_en | input | 1 | Read enable |
| rd\_data | output | 32 | Data bus for read |
| full | output | 1 | FIFO full status |
| half\_full | output | 1 | FIFO half-full status |
| empty | output | 1 | FIFO empty status |

# Required Tools

## List of required software and hardware tool sets needed.

* **Simulation:** Mentor Graphics QuestaSim or Synopsys VCS
* **Verification Language:** SystemVerilog
* Source code: <https://github.com/RohithInti/Team_8_Async_FIFO>

## Directory structure

Design, object oriented, and basic testbenches will be in separate source directories.

ECE593\_Team8

+ rtl\_source

+ testbench

+ work

+ results

# Risks and Dependencies

|  |  |  |
| --- | --- | --- |
| **Risk** | **Impact** | **Mitigation** |
| CDC metastability on pointer sync | Occasional synchronization failures | Two-stage synchronizer; voeg assertion for max latency |
| Pointer arithmetic overflow | Incorrect full/empty signaling | Thorough corner-case tests; formal check of bin2gray/gray2bin |
| Reset-domain crossing | Inconsistent pointer initialization | Reset synchronization; stress tests at reset boundaries |

# Functions to be Verified

## Functions Included

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| |  | | --- | | **Function** | | |  | | --- | | **Description** | | |  | | --- | | **Notes** | |
| |  | | --- | | Binary-to-Gray conversion | | |  | | --- | | bin2gray() function | | |  |  | | --- | --- | |  | Exhaustive input tests | |
| |  | | --- | | Gray-to-Binary conversion | | |  | | --- | | gray2bin() function | | |  | | --- | | Back-to-back conversion checks | |
| |  | | --- | | Write pointer increment & memory write | | |  |  |  | | --- | --- | --- | | Pointer increment when wr\_en && !full |  |  | | |  | | --- | | Verify data integrity and pointer wrap-around | |
| |  | | --- | | Read pointer increment & memory read | | |  | | --- | | Pointer increment when rd\_en && !empty | | |  | | --- | | Verify data matches write history | |
| |  | | --- | | Status flag generation | | |  | | --- | | full, half\_full, empty, half\_empty logic | | |  | | --- | | Boundary and mid-point checks | |
| |  | | --- | | CDC synchronizer | | |  | | --- | | Two-stage Gray-pointer sync across domains | | |  | | --- | | Monitor sync latency | |

## Critical vs. Non‑Critical Functions

|  |  |  |
| --- | --- | --- |
| **Function** | **Criticality** | **Description** |
| Full/Empty flag logic | Critical | Prevents overrun/underrun |
| Memory read/write operations | Critical | Data corruption risk |
| Pointer conversions | Critical | Ensures correct addressing |
| Half-full/half-empty flags | Non‑Critical | Used for flow-control hints, not for fundamental safety |

# Tests and Methods

## Testing Methodology

### Box-Level Approach

* **Black-Box:** Stimuli only on interfaces; ideal for performance metrics.
* **White-Box:** Leverage internal signals for corner-case and internal path validation.
* **Gray-Box:** Combines both; chosen for this DUT to verify both interface behavior and internal pointer state.

### Pros and Cons

|  |  |  |
| --- | --- | --- |
| **Method** | **Pros** | **Cons** |
| Black-Box | Simple stimulus-driven tests; focus on spec. | Cannot target internal corner cases |
| White-Box | Full visibility; targeted tests for internal logic | Requires access to internals; more effort |
| Gray-Box | Balanced; can drive and monitor internal state | Moderate complexity |

### Verification Strategy

**Dynamic Simulation:** Primary strategy for functional and performance validation.  
**Directed Tests:** Cover basic read, write, and reset scenarios explicitly.

## Testcase Scenarios

|  |  |
| --- | --- |
| **Category** | **Test Description** |
| |  | | --- | | Basic Tests | | |  | | --- | | * Single-word write then read, check rd\_data equals wr\_data. |  |  | | --- | | * Multiple writes less than depth, verify half\_full when depth >=128. | |
| |  | | --- | | Corner Cases | | |  |  | | --- | --- | |  | * Assert rd\_en when empty, ensure empty stays asserted, no underflow. | | * Assert wr\_en when full, ensure full stays asserted, no overwrite. | | | |

# Resources requirements

## Venkat Sahith

* Verification test plan

## Mrudula

* DUT Specification document

## Vaishnavi

* RTL code

## Rohith

* Self-Checking testbench

# Schedule

## Milestone 1

* Initial design and verification specifications written
* Block diagram is finalized and integrated into the document.
* RTL implementation of async\_fifo.sv is committed and compiling.
* Standalone tests for bin2gray and gray2bin functions are running.

## Milestone 2

## ● Basic validation of full and empty flags under simple operations. ● half\_full and half\_empty flag behavior verified at mid-depth thresholds. ● Scoreboard and monitor infrastructure set up in testbench.

## Milestone 3

● Constrained-random burst tests (burst length = 1024) will be developed and executed.  
● Wrap-around scenarios and underflow/overflow corner-case tests will be added.  
● Functional coverage models and coverpoints implemented for depth and flag transitions.

## Milestone 4

* Initial UVM testbench
* UVM verification plan

## Milestone 5

* UVM testbench will be completed

● Final verification report will be drafted, including coverage and bug-injection results.  
● Documentation will be updated with test results, coverage metrics,

# References

[1]<https://vlsiverify.com/verilog/verilog-codes/asynchronous-fifo/>

[2]OpenCores, "Asynchronous FIFO IP Core," [https://opencores.org/projects/async\_fifo](https://opencores.org/projects/async_fifo%20)

[3]Xilinx, "FIFO Generator v13.2 User Guide," Xilinx Application Note XAPP 501, 2017.

[4]<http://www.sunburst-design.com/papers/CummingsSNUG2002SJ_FIFO1.pdf>