Asynchronous FIFO - Verification Plan

The goal of this verification plan is to ensure the Asynchronous FIFO transfers data between blocks with asynchronous clocks properly without metastability propagation. Key features to verify will include independent read/write clocks as well as FIFO full/empty detection. Binary to gray pointer conversion and synchronization across clock domains will also be important to verify. Directed tests will be used for corner cases along with constrained-random stimulus for stress testing. Assertions will be used to test clock domain crossing (CDC) safety.

Directed Tests

Test	**Description**
Reset	Reset read/write domains independently
	Expect FIFO starts empty and no false pops
Single write -> single read	Read one word, write one word
	Expect exact match
Fill to full FIFO	Fill until fu
	Continuously write until `w_ready` deasserts
	Confirm no overflow: writes stop when full
Drain to empty FIFO	Drain until empty
	Read until `r_valid` deasserts
	Expect no new data stored; `w_fire` must not occur
Full condition	Write when full
Read when empty	Force `r_ready=1` while empty
	Expect no pop; `r_fire` must not occur
Wrap-around	Write/read enough to wrap pointers multiple times

Randomized Tests

- Random read/write block ready signals

- Random read/write clock frequencies
- Long random sequences with random reset assertions
- Random resets

Functional Coverage Plan

Covergroups	
Coverage Type	Behavior
Write-Side Coverage	Read block ready
	Write when FIFO almost full
	Write blocked when FIFO full
Read-Side Coverage	Write block ready
	Ready when FIFO almost empty
	Read blocked when FIFO empty
FIFO State Coverage	Empty <-> non-empty
	Full <-> non-full
	Simultaneous read/write
Pointer and CDC Coverage	Gray pointer transitions (single-bit change)
	Pointer synchronization latency (1 vs 2 cycles)
	Full/empty detection correctness across CDC

Code Coverage Goals

Metric	**Target**
Statement	100%
Branch	100%
Expression	>95%
Toggle	>90%

FSM	If any

Scoreboard

- Ordering preserved (FIFO property)
- No missing/duplicate items (queue size tracking + match comparisons)

Assertions

- No read/write when FIFO empty/full
- No pop when empty
- No push when full
- Gray pointers only increment one bit, single-bit transition (if accessible)
- FIFO ordering preserved
- Pointer synchronization stability (no metastability glitches)
- full/empty stable behavior around reset

Team Member Responsibilities	
Carson	TB development, directed tests
David	Read/write driver, assertions, ordering checks, and explaining class-based TB implementation
Liukee	Code coverage, Functional coverage, coverage analysis, and Writing Verification Plan

Completion Schedule	
Milestone and Due Date	**Task**
M2-M3 (Feb 18th)	Complete RTL (the DUT), compile clean
	Completed TB that proves the DUT works, all class-based TB components + coverage

	Updated verification plan + `run.do` + coverage reports
	Report explaining your class-based TB implementation
	Transcript showing ~15–20 randomized transactions
M4 (Feb 27th)	Define UVM architecture and tests
	Develop UVM driver/monitor, interfaces, and assertions
	Plan UVM coverage strategy and covergroups
M5 (Mar 14th)	Finalize UVM testcases and assertion validations
	Complete code/functional coverage and generate coverage reports
	Submission