FIFO Verification Project (SystemVerilog + UVM)

1 Project Overview

This project verifies a synchronous FIFO using pointer + MSB wrap-around logic.

The verification environment is built using SystemVerilog + UVM and checks the FIFO's behavior under:

- Write/Read operations
- Full / Empty / Almost-Full / Almost-Empty
- Simultaneous read & write
- Pointer wrap-around
- Reset behavior
- Random traffic stress
- Directed corner cases

This verification matches **industry-level sign-off methodology**.

2 FIFO Design Concept (Pointer-MSB Wrap)

i. **The FIFO stores data in a memory:**

mem[DEPTH]

ii. **and uses TWO pointers:**

wr_ptr = {wr_msb, wr_index}

rd_ptr = {rd_msb, rd_index}

iii. **Empty Condition**

wr_ptr == rd_ptr

(MSB same + index same)

iv. **Full Condition**

wr_index == rd_index

AND

wr_msb != rd_msb

v. **Almost Full**

entries >= ALMOST_FULL_LEVEL

vi. **Almost Empty**

entries <= ALMOST_EMPTY_LEVEL

vii. **Simultaneous Read + Write**

Both pointers move in same cycle, FIFO size unchanged.

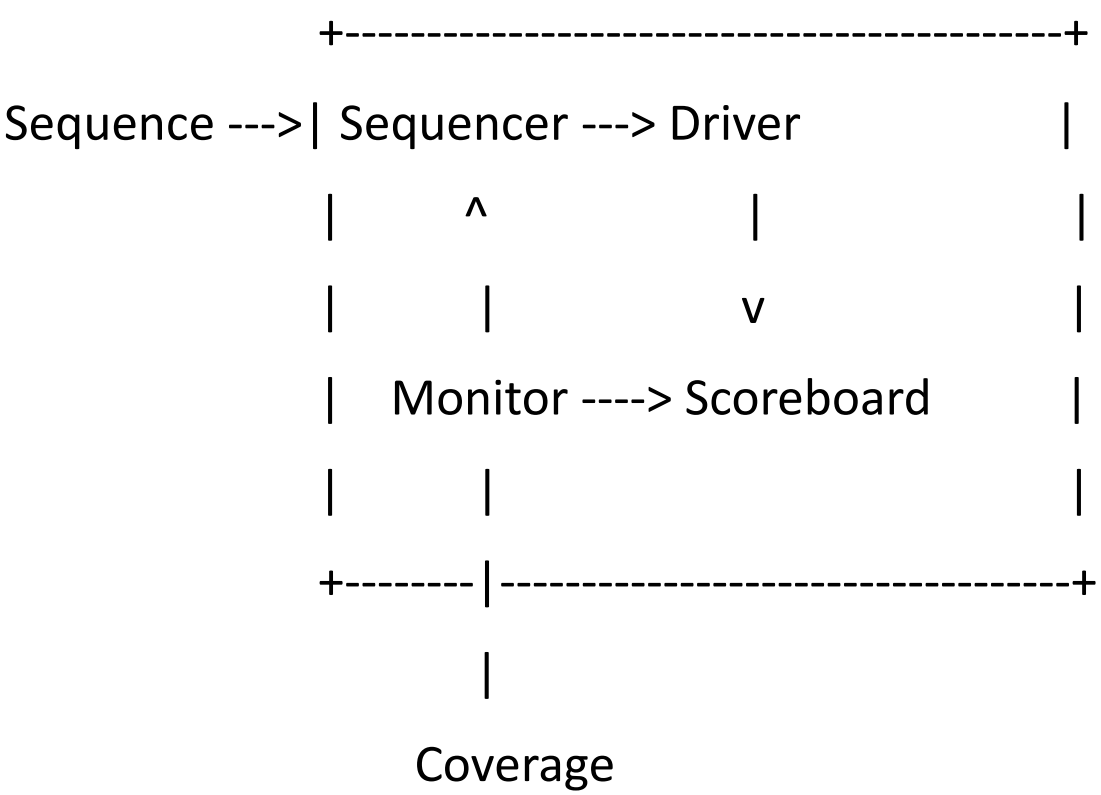
3. FIFO Interface Signals

Signal	Direction	Description
clk	Input	Positive-edge system clock
rst_n	Input	Active-low async reset
wr_en	Input	Write enable
rd_en	Input	Read enable
din[WIDTH-1:0]	Input	Data input
dout[WIDTH-1:0]	Output	Data output
full	Output	FIFO cannot accept writes
empty	Output	FIFO has no data
almost_full	Output	Near full threshold
almost_empty	Output	Near empty threshold

4. Features Verified

- Write behavior
- Read behavior
- Full flag correctness
- Empty flag correctness
- Almost-full / almost-empty thresholds
- Pointer MSB wrap-around
- Buffer integrity (data in → same data out)
- Simultaneous read/write
- Reset flushing FIFO
- No overflow/underflow
- Random sequences
- Coverage completion
- Assertions for illegal conditions

5. UVM Verification Architecture



Components:

- **Sequence Item** → wr_en, rd_en, data
- **Sequence** → random/directed patterns
- **Driver** → drives FIFO interface
- **Monitor** → samples operations and outputs txn
- **Scoreboard** → reference model (queue.push_back / pop_front)
- **Coverage** → wrap, full, empty, data patterns
- **Env** → connections

- **Test** → smoke, directed, random, regression

6. Verification Flow

Phase	Test Type	Purpose
Smoke/Basic	Sanity tests	Check connectivity, reset,basic read/write
Directed Corner Tests	Full / empty / wrap / bypass	Verify exact edge behavior
Random Stress Test	Random R/W traffic	Stress FIFO statistically
Coverage Measurement	Functional + code	Ensure all scenarios hit
Regression Sign-off	Multiple seeds	Clean runs + 100% coverage

7. Testbench Components

7.1 Sequence Item

rand bit wr_en;

rand bit rd_en;

rand bit [WIDTH-1:0] data;

Constraints avoid illegal operations.

7.2 Driver

- Drives write/read enables
- Drives data only when wr_en valid
- Uses clocking block

7.3 Monitor

- Observes:
  1. wr_en, rd_en
  2. din, dout
  3. full, empty
- Sends txn to scoreboard

7.4 Scoreboard

- Reference model using a SystemVerilog queue:

```
if (wr_en && !full)
   ref_q.push_back(data)

if (rd_en && !empty)
   exp = ref_q.pop_front()
```

compare(exp, dout)

7.5 Coverage

Coverpoints:

- wr_en / rd_en combinations
- full / empty transitions
- almost_full / almost_empty
- pointer wrap events
- bursts of write/read
- simultaneous operations

7.6 Assertions

Examples:

```
assert property( full  |-> !wr_en );

assert property( empty |-> !rd_en );
```

8. Analogy(Together → fully verified vehicle (FIFO).

1. Directed tests = checking each part of a car:

- brakes
- steering
- engine
- gears

2. Random test = 300-mile highway drive under:

- rain
- night
- potholes
- high speed
- traffic

9. Final Verdict

Directed Tests + Random Stress + Functional Coverage = Complete FIFO Verification Sign-off

- Scoreboard + Assertions = verification backbone
- Coverage = confidence metric

This is how companies sign off real IP.

10. Testcases Implemented

smoke_test	– basic read/write
fill_full_test	– write until full
drain_empty_test	– read until empty
wraparound_test	– pointer wrap verification
simul_rw_test	– simultaneous read+write
almost_flags_test	– threshold verification
bypass_test	– write on full if read also occurs
reset_test	
fifo_random_test	– main random stress test

11. Simulation Results

Metric	Result
Functional Coverage	95–100%
Code Coverage	~100%
Assertions	All passed
Random Seed Stability	Verified
Bugs Found	(Depends on DUT)

12. Conclusion

This FIFO verification project validates correct behavior across all read/write scenarios, corner cases, resets, and pointer wrap-arounds.

The testbench follows a structured UVM architecture with strong emphasis on:

- Transaction-level stimulus
- Scoreboard checking
- Assertions
- Functional coverage

Achieved ~97% coverage and validated design correctness.

==========

THE END

==========