

Memory Verification using UVM (SystemVerilog)

Introduction

This document describes a SystemVerilog and UVM based memory verification project. The project is developed incrementally to demonstrate core UVM concepts, best practices, and an industry-standard verification flow for validating a memory design (DUT). It is intended for learning, practice, and portfolio demonstration in the semiconductor and VLSI verification domain.

Project Objectives

- Verify a memory design using SystemVerilog and UVM
- Understand UVM architecture and phase-based execution
- Implement functional read and write verification
- Build a scoreboard for data checking
- Use TLM analysis ports, uvm_config_db, and uvm_resource_db
- Practice assertions, objections, and sequences
- Develop a reusable and scalable UVM environment

UVM Architecture Overview

The verification environment follows a layered UVM architecture. Stimulus is generated using sequences and sequencers, driven to the DUT via the driver, and observed using a monitor. The monitor sends transactions to the scoreboard using TLM analysis ports for functional checking.

- Transaction (mem_tx)
- Sequence and Sequencer
- Driver
- Monitor
- Coverage
- Agent
- Environment
- Scoreboard
- Test

Data Flow

Sequence → Sequencer → Driver → DUT → Monitor → Scoreboard

Key Features Implemented

- Memory read and write verification
- UVM sequences and sequence items

- TLM analysis port connection for scoreboard
- Scoreboard with data comparison logic
- Configuration using uvm_config_db
- SystemVerilog assertions for protocol checking
- Objection-based run phase control
- Modular and reusable UVM components

Scoreboard Functionality

The scoreboard stores write transactions in an associative array and compares read data against expected values. It tracks matching and mismatching transactions. All checks are performed passively inside the write() method using the analysis port.

Objection Handling

Tests raise objections during the run_phase to prevent premature simulation termination. Objections are dropped only after sequences complete execution, ensuring proper test completion.

Simulation Requirements

A UVM-capable simulator such as Questa, ModelSim, VCS, or Xcelium is required. The project supports UVM 1.2 or simulator built-in UVM libraries.

Learning Outcomes

- Understanding UVM phases (build, connect, run)
- Applying TLM-based communication
- Designing a functional scoreboard
- Managing driver–sequencer handshakes
- Handling objections correctly
- Debugging common UVM issues
- Following verification best practices

Author

Ganesh H R
 Design Verification (UVM/SystemVerilog)
 Fresher | Semiconductor Verification Enthusiast