**Class-Based Testbench for Asynchronous FIFO**

1. **Introduction:**

This testbench is developed using a class-based verification approach in SystemVerilog to evaluate the behavior and correctness of an asynchronous FIFO design. The setup closely mirrors a Universal Verification Methodology (UVM) structure but is written manually without relying on the UVM library. The verification is modular and leverages object-oriented concepts for flexibility and reusability.

1. **Breakdown of the Testbench Modules:**

**Interface (interface.sv)**This module serves as a communication bridge between the testbench components and the DUT (Device Under Test). It encapsulates all necessary signals such as data lines, read/write enables, and control flags. It also defines clocking blocks to synchronize signal transactions across the verification environment.

**Driver (driver.sv)**The driver is responsible for converting abstract transactions into real signal-level interactions with the DUT. It reads inputs from the generator and sends them through the interface at appropriate times, simulating a realistic FIFO usage pattern.

**Generator (generator.sv)**This block creates randomized stimulus — essentially generating a sequence of input data and control signals that simulate write and read operations. It mimics various operational scenarios by varying timing and data values.

**Input and Output Monitors (monitor\_in.sv, monitor\_out.sv)**Monitors observe signal activity on the interface without influencing the system. The input monitor watches what is written into the FIFO, and the output monitor captures what is read. These monitors store observations for later comparison.

**Scoreboard (scoreboard.sv)**This is the core verification checker. It holds a model of expected behavior and compares it to the actual outputs from the DUT. Any mismatches are flagged, helping to pinpoint functional bugs or violations in FIFO behavior.

**Environment (environment.sv)**The environment is a container module that brings together the generator, driver, monitors, and scoreboard. It handles component instantiation, connections, and coordination of their activities.

**Test File (test.sv)**This file defines the specific test cases, including how many transactions to generate and what corner cases to cover. It also initializes the environment and triggers the start of simulation**.**

**Package (tb\_pkg.sv)**This file packages shared items like class definitions, data structures, and parameters. It ensures consistent data types and interfaces across all verification modules.

**Top-Level Module (top.sv)**The top module connects everything together — the DUT, interface, and testbench. It includes clock and reset generation and is used as the entry point for simulation tools.

**3. DUT Description (async\_fifo.sv)**

The DUT is an asynchronous FIFO that uses separate clocks for reading and writing. It features signals to indicate whether the buffer is full or empty. This design makes it important to verify correct handling of clock domain crossing, synchronization, and data integrity.

**4. Execution Flow**

* The simulation starts from top.sv, which instantiates the DUT and interface.
* The test case initializes the verification environment and begins stimulus generation.
* The driver pushes inputs into the FIFO, while monitors track activity.
* The scoreboard continuously checks output correctness against expectations.
* At the end of simulation, results are analyzed to determine pass/fail status.

**5. Benefits of the Approach**

* Each block is isolated, making debugging easier.
* New test scenarios can be added with minimal changes.
* This architecture allows for future extension to other designs or protocols with similar verification needs.