# ECE-593: Fundamentals of Pre-Silicon Validation

Maseeh College of Engineering and Computer Science

Winter, 2025


Project Name: Multiprocessor System

Members:Janvier Mpfizi Rutihunza, Frezewd Debebe,Sal Esmaeil

Date: 02/18/2026

Our testbench uses a class based structure. Each class handles one task. The design separates stimulus, driving, monitoring, checking, and coverage. This improves control and readability.

The transaction class defines one operation. It stores core ID, opcode, address, operands, control signals, output data, and expected value. Constraints keep opcode in a valid range and limit address below 2048. This prevents invalid stimuli. The copy method avoids data overwriting between components. The display method prints transaction details for debug.

The generator creates random transactions. The total number depends on tx_count, for example 100 transactions. The generator randomizes one transaction at a time. It sends the transaction to the driver through a mailbox. It waits for a signal from the driver before sending the next one. This keeps traffic controlled and predictable.

The driver connects to the DUT through a virtual interface. It performs a reset first. It drives reset low for several clock cycles, then releases it. After reset, it reads one transaction from the mailbox. It drives core ID, opcode, and address. It asserts requests and waits for grants. If the operation is stored, it drives operands and asserts write enable. After handshake completion, it deasserts signals. It then signals the generator to continue.

The input monitor watches signals entering the DUT. When request or write enable asserts, it captures the operation into a transaction object. It sends this object to the scoreboard. It also sends it to the coverage collector. This ensures coverage reflects real DUT activity.

The output monitor watches response signals. When rvalid asserts, it captures opcode, address, output core ID, and output data. It builds a transaction and sends it to the scoreboard for checking.

The scoreboard performs prediction and comparison. It uses a reference memory array of size 2048. It keeps four queues, one for each core. When it receives an input transaction, it calculates the expected result. For reading, it fetches data from reference memory. For stores, it updates reference memory. It stores the expected transaction in the correct core queue. When it receives an output transaction, it removes the oldest expected transaction from the same core queue. It compares expected data with actual data. It prints pass or fail.

The coverage collector tracks functional behavior. It measures core ID usage, opcode distribution, and address ranges. It also tracks cross coverage between core ID and opcode. With four cores and 14 opcodes, up to 56 combinations exist. The input monitor triggers coverage sampling.

The environment connects all components. It creates a generator, driver, monitors, scoreboard, and coverage collector. It connects mailboxes. It assigns the virtual interface. It runs reset, then starts all components in parallel. It waits for generation to finish before ending the simulation.

The interface defines driver and monitor clocking blocks. The driver writes through the driver clocking block. The monitors sample through the monitor clocking block. This prevents race conditions.

The top module instantiates clock, interface, DUT, and environment. It sets the transaction count and starts the environment.

The flow is direct. The generator creates operations. The driver applies them. The monitors capture activity. The scoreboard checks for correctness. The coverage collector measures behavior.