

# ECE 593 WINTER 2025

## TEAM 13

Git link: <https://github.com/Defuse-cfg/UART-PROTOCOL>

### Verification Plan :UART

#### 1. Introduction

This document presents a **Universal Verification Methodology (UVM)-based** approach to verifying the **UART design**, ensuring that its core components—including the **transmitter (uart\_tx)**, **receiver (uart\_rx)**, **FIFO buffer**, and **clock generator**—function as expected.

---

#### 2. Verification Goals

The primary aim is to validate the **accuracy, reliability, and performance** of the UART system. Specific objectives include:

Confirming **correct data transmission and reception**.

Verifying **FIFO buffer operations**, including handling full and empty states.

Ensuring **proper baud rate generation and clock synchronization**.

Checking **error handling mechanisms** (such as parity and framing errors).

Achieving **comprehensive functional coverage** and design compliance.

---

#### 3. Verification Approach

The verification strategy will leverage **Class based &UVM-based verification**, incorporating:

- **Constrained-random stimulus generation** to cover various scenarios.
  - **Coverage-driven verification** to ensure all design aspects are exercised.
  - **Self-checking mechanisms with scoreboards** to validate correctness.
  - **Assertions to enforce protocol compliance and detect violations**.
- 

#### 4. Testbench Structure

The UVM testbench will be composed of the following elements:

##### 4.1. Interface

- Defines **connections between the DUT and testbench**.
- Facilitates **data transactions and signal control**.

#### 4.2. UVM Agent

- **Driver**: Converts transaction-level operations into DUT stimulus.
- **Sequencer**: Generates test sequences for UART communication.
- **Monitor**: Observes DUT activity and forwards transaction details for verification.

#### 4.3. UVM Environment

- Manages multiple **UART agents (Tx and Rx)**.
- Oversees **scoreboarding, functional coverage collection, and assertions**.

#### 4.4. Scoreboard & Coverage Collection

- **Scoreboard**: Compares expected vs. actual UART output for validation.
- **Functional Coverage**: Tracks exercised test scenarios (baud rates, error cases, FIFO status).

#### 4.5. Testcases

- Defines **directed and random tests** for thorough validation.
  - Ensures all UART functionalities are **systematically tested**.
- 

### 5. Key Verification Aspects

#### 5.1. Stimulus Generation

- **Randomized data inputs** with varying parity and baud rate configurations.
- **Constrained-random sequences** to explore a wide range of scenarios.

#### 5.2. Assertions & Checkers

- **Protocol assertions** to verify UART behavior (start/stop bit detection, baud rate accuracy).
- **FIFO assertions** to check buffer operations (full/empty conditions, data retention).

#### 5.3. Coverage Metrics

- **Functional Coverage**: Ensuring all UART functions are tested.

- **Code Coverage:** Measuring toggle, branch, FSM, and condition coverage.
- 

## 6. Test Scenarios

### 6.1. Basic Functionality Checks

Standard UART communication test (verify byte transmission and reception).

FIFO buffer validation (test full and empty states).

Baud rate accuracy (ensure timing correctness).

### 6.2. Edge Cases

Handling FIFO overflows and underflows.

Introducing and detecting parity errors.

Testing extreme baud rate limits.

### 6.3. Stress & Randomized Testing

Executing randomized transaction sequences with varied data patterns.

Continuous high-speed transmission testing.

---

## 7. Verification Metrics

To consider verification complete, the following criteria must be met:

- **100% code coverage.**
  - **At least 90% Functional coverage**
  - **Zero critical bugs**, with minimal non-critical issues.
  - **All assertions must pass** without violations.
- 

## 8. Conclusion

This UVM-based verification plan ensures the UART system meets its functional and performance requirements. By employing randomized and directed testing, self-checking mechanisms, and robust coverage metrics, we guarantee a high-quality, error-free UART implementation.