UART TRANSMITTER Protocol Verification

Introduction

Serial communication protocols play a crucial role in data transfer between digital systems, especially in embedded systems and FPGA-based designs. Among these protocols, **Universal Asynchronous Receiver Transmitter (UART)** is one of the most widely used due to its simplicity, low hardware overhead, and asynchronous nature.

UART is commonly used for:

- Debug communication
- Microcontroller-to-peripheral interfaces
- FPGA-to-host communication
- Bootloaders and configuration interfaces

Although UART is conceptually simple, **correct timing, framing, and synchronization are critical** for reliable data transmission. Any deviation in baud rate timing or bit alignment can result in corrupted data.
Therefore, **functional verification** of UART designs is extremely important before silicon fabrication or FPGA deployment.
This project focuses on the **complete functional verification of a UART Transmitter (TX)** using **SystemVerilog**, employing a **layered, self-checking verification architecture** with randomized stimulus and functional coverage.

Objectives of the Project:

The primary objectives of this verification project are:

1. **To verify correct UART transmitter functionality**, ensuring proper frame generation with one start bit, eight data bits transmitted LSB first, and one stop bit.

2. **To validate baud-rate–controlled serial transmission**, confirming that UART bit timing strictly follows the configured clock frequency and baud rate using the calculated clocks-per-bit mechanism.’

3. **To ensure end-to-end data correctness using a self-checking verification environment**, by comparing randomized transmitted data against monitored serial output and measuring functional coverage across the valid 8-bit data range.

Overview of UART TRANSMITTER Design

UART communication consists of:

- Idle Line: Logic ‘1’
- Start Bit: Logic ‘0’
- Data Bits: 8 bits (LSB first)
- Stop Bit: Logic ‘1’

Each bit is transmitted for a duration of 1 / Baud Rate seconds.

For this design:

- **Clock Frequency:** 100 MHz
- **Baud Rate:** 9600
- **Clocks per Bit:** 100,000,000 / 9600 ≈ 10416 cycles

UART Transmitter implemented using a Finite State Machine (FSM) with the following states:

State	Description
IDLE	Line idle, waits for tx_start
START	Transmits start bit
DATA	Transmits 8 data bits (LSB first)
STOP	Transmits stop bit

Key features:

- Parameterized clock frequency and baud rate
- Busy indication is done
- Accurate baud timing using clock counter
- Clean FSM-based implementation

Verification Architecture:

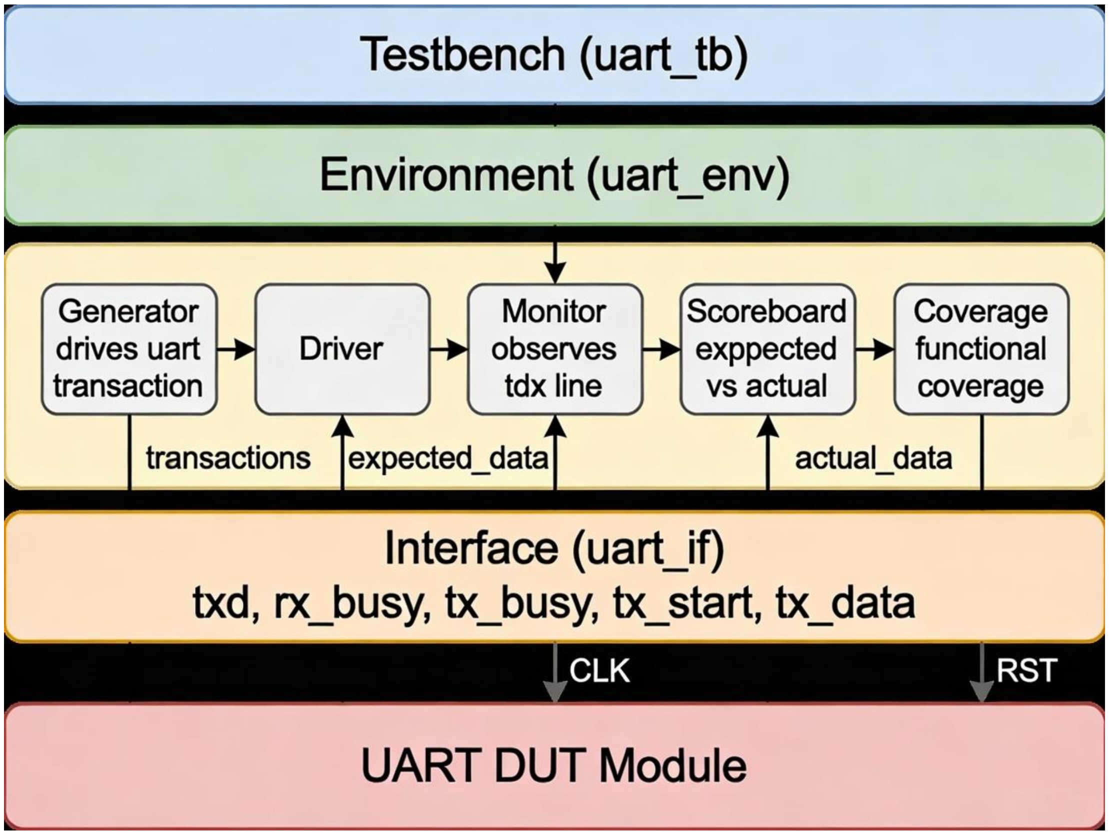

Testbench (uart_tb)
Environment (uart_env)
Generator drives uart transaction
Driver
Monitor observes tdx line
Scoreboard exppected vs actual
Coverage functional coverage
transactions
expected_data
actual_data
Interface (uart_if)
txd, rx_busy, tx_busy, tx_start, tx_data
CLK
RST
UART DUT Module

Fig. 1 Layered Test Bench for UART TX protocol verification

The verification environment follows a **layered architecture**, separating stimulus generation, driving, monitoring, checking, and coverage.

Interface (uart_if)

- Encapsulates DUT signals
- Provides separate **driver** and **monitor** modports
- Ensures clean connectivity between DUT and testbench

Verification Components

1. Transaction (uart_transaction)

Represents a UART packet.

Fields:

- tx_data – Transmitted byte (randomized)
- rx_data – Received byte (monitored)
- pass – Result of comparison
- txn_id – Transaction ID

2. Generator (uart_generator)

- Randomizes UART transactions
- Controls number of packets (num_pkts)
- Sends transactions to the driver via mailbox
- Ensures realistic spacing between packets based on baud rate

3. Driver (uart_driver)

- Drives transactions onto the DUT interface
- Waits for tx_busy deassertion
- Generates a single-cycle tx_start pulse
- Sends expected data to scoreboard and coverage collector

4. **Monitor (uart_monitor)**

- Passively observes UART TX line (txd)
- Detects start bit using falling edge
- Samples data bits at the **center of each bit**
- Reconstructs received byte
- Sends actual data to scoreboard

5. **Scoreboard (uart_scoreboard)**

- Compares expected transmitted data with monitored received data
- Declares PASS or FAIL per transaction
- Maintains pass/fail statistics
- Prints detailed transaction-level results

6. **Functional Coverage (uart_coverage)**

Coverage model ensures:

- Transmission of 0x00
- Transmission of 0xFF
- Coverage of all intermediate values (1–254)

Coverage bins:

- Zero value
- Maximum value
- Mid-range values

7. **Environment (uart_env)**

The environment:

- Instantiates all verification components
- Connects components via mailboxes
- Calculates bit-time based on baud rate
- Controls test execution and synchronization
- Collects final scoreboard and coverage reports

8. **Testbench (uart_tb)**

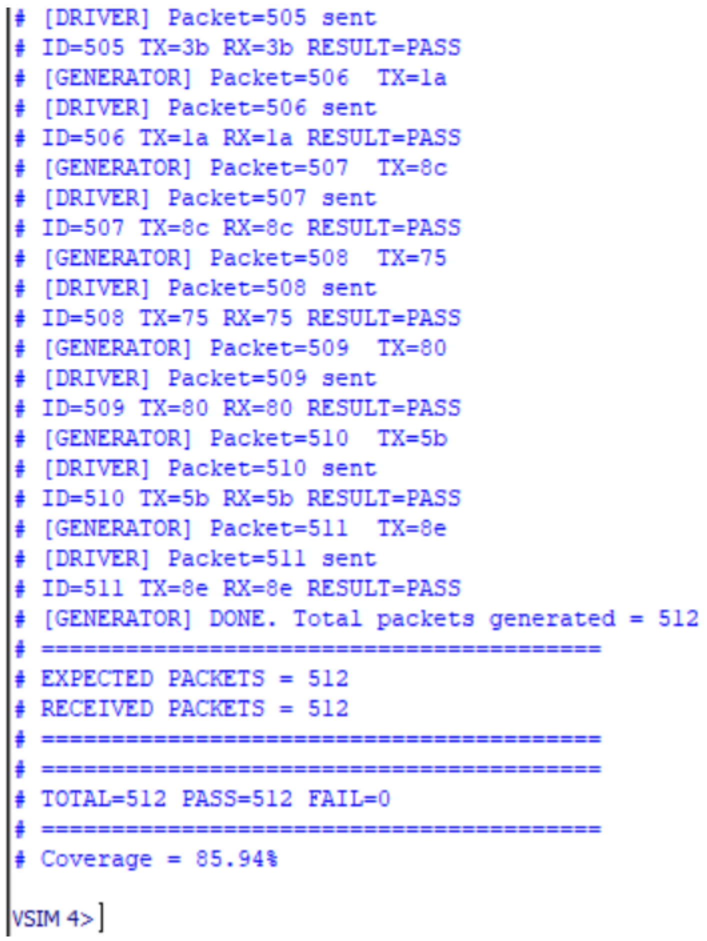
Test Configuration:

- Clock: 100 MHz
- Baud Rate: 9600
- Number of Packets: 512

Test Flow:

1. Reset applied
2. Environment created
3. Generator produces randomized transactions
4. Driver sends data to DUT
5. Monitor captures transmitted bits
6. Scoreboard validates correctness
7. Coverage is collected and reported

Results:



```
# [DRIVER] Packet=505 sent
# ID=505 TX=3b RX=3b RESULT=PASS
# [GENERATOR] Packet=506  TX=1a
# [DRIVER] Packet=506 sent
# ID=506 TX=1a RX=1a RESULT=PASS
# [GENERATOR] Packet=507  TX=8c
# [DRIVER] Packet=507 sent
# ID=507 TX=8c RX=8c RESULT=PASS
# [GENERATOR] Packet=508  TX=75
# [DRIVER] Packet=508 sent
# ID=508 TX=75 RX=75 RESULT=PASS
# [GENERATOR] Packet=509  TX=80
# [DRIVER] Packet=509 sent
# ID=509 TX=80 RX=80 RESULT=PASS
# [GENERATOR] Packet=510  TX=5b
# [DRIVER] Packet=510 sent
# ID=510 TX=5b RX=5b RESULT=PASS
# [GENERATOR] Packet=511  TX=8e
# [DRIVER] Packet=511 sent
# ID=511 TX=8e RX=8e RESULT=PASS
# [GENERATOR] DONE. Total packets generated = 512
# ========================================
# EXPECTED PACKETS = 512
# RECEIVED PACKETS = 512
# ========================================
# ========================================
# TOTAL=512 PASS=512 FAIL=0
# ========================================
# Coverage = 85.94%

VSIM 4>
```

Fig. 2 Result of UART TX protocol verification

The verification was carried out using the **QuestaSim simulator**, which supports **SystemVerilog class-based constrained randomization** through the randomize() method. Random stimulus was applied to the UART transmitter for **512 transactions**, and functional coverage was collected using a SystemVerilog covergroup.

With 512 randomized input transactions, an overall functional coverage of **approximately 86%** was achieved, indicating effective exercise of corner cases and a large portion of the valid 8-bit data space.