UART ARCHITECTURE

By Anakh M Nair

Trainer :- Chalam Tirunagari

Mentor:- Gangadhar G

INTRODUCTION

Introduction

In this project, the Universal Verification Methodology (UVM) was used to develop a testbench for the verification of the **Universal Asynchronous Receiver-Transmitter (UART)** protocol. UART is a widely used serial communication protocol that allows for asynchronous data transfer between two devices, commonly used in microcontrollers, sensors, and various communication systems.

The primary goal of this project was to verify the correct functionality of the UART protocol, ensuring that data is transmitted and received accurately between the sender (master) and receiver (slave). The verification process is critical in ensuring the integrity and reliability of the communication, especially under different operating conditions such as varying baud rates, error conditions, and timing constraints.

UVM was chosen for this project due to its ability to create modular, reusable, and scalable verification environments. UVM's object-oriented approach and the automation of testbenches allow for efficient and comprehensive testing of the UART protocol in a structured manner. This project aimed to design a UVM-based testbench, including a master and slave communication model, and implement a scoreboard to validate the correctness of data transfers.

In this report, we will discuss the objectives of the project, the role of the master and slave components, the design and use of the scoreboard, as well as the waveforms and simulations generated during testing. Additionally, we will highlight the challenges faced during the project and the key takeaways from the verification process.

Key Components of UART Architecture

The architecture of UART typically consists of several key components that manage data transmission and reception:

- **Transmitter (TX)**
  - The **transmitter** (TX) is responsible for sending data to the receiver. The data is typically sent in a **serial format**, where bits are transmitted one after the other, rather than in parallel. The transmitter is controlled by the **baud rate**, which determines the speed of data transmission. For example, at a baud rate of 9600, the transmitter sends 9600 bits per second.

- **Receiver (RX)**
  - The **receiver** (RX) accepts the incoming data stream from the transmitter. The receiver synchronizes with the incoming bits based on the predefined baud rate and reconstructs the original data word by aligning start, data, parity, and stop bits.
  - The receiver also monitors for any **framing errors** or **parity errors** in the received data, ensuring data integrity and correctness.
- **Start Bit**
  - Each data frame transmitted via UART begins with a **start bit**. The start bit signals the beginning of a data frame and is typically represented as a low voltage signal (0). It ensures the receiver is ready to receive the data and synchronizes the start of transmission.
- **Data Bits**
  - After the start bit, the **data bits** are transmitted. Typically, UART data frames contain 5 to 9 data bits. The data bits represent the actual information being transmitted.
- **Parity Bit**
  - A **parity bit** is used for error checking and can be configured as **even**, **odd**, or **none**. The parity bit is calculated based on the number of 1's in the data bits to help detect errors in transmission. If the parity configuration is even, the number of 1's should be even; if odd, the number of 1's should be odd. Any mismatch in parity indicates a transmission error.
- **Stop Bit**
  - The data frame ends with one or more **stop bits**, which are used to signal the end of the data transmission. Stop bits are typically represented as a high voltage signal (1), and their number (one or two stop bits) is determined by the UART configuration. The stop bit ensures that the receiver knows when the transmission has finished.
- **FIFO Buffer**
  - Many UART implementations include a **FIFO (First-In-First-Out) buffer** for both transmission and reception of data. The FIFO buffer temporarily stores the data while it's being transmitted or received, allowing the UART to handle multiple bytes of data more efficiently.
  - For instance, the **Transmitter Holding Register (THR)** stores data that will be sent next, while the **Receiver Buffer Register (RBR)** stores incoming data to be read by the receiver.
- **Control Signals**
  - UART communication may also include a few control signals such as **Request to Send (RTS)**, **Clear to Send (CTS)**, and **Data Terminal Ready (DTR)**, which are used for flow control, especially in hardware-assisted flow control configurations. These signals help coordinate data transmission between devices.

Communication Modes

- **Full-Duplex**: In full-duplex mode, data can be transmitted and received simultaneously. Both the transmitter and receiver are active at the same time, allowing for continuous bidirectional communication.
- **Half-Duplex**: In half-duplex mode, data transmission can only occur in one direction at a time. The system alternates between transmitting and receiving data, preventing simultaneous data flow in both directions.

- **Loopback Mode**: Some UART implementations support **loopback mode**, where the transmitted data is fed back to the receiver, allowing the device to test its transmission and reception capabilities.

Error Handling

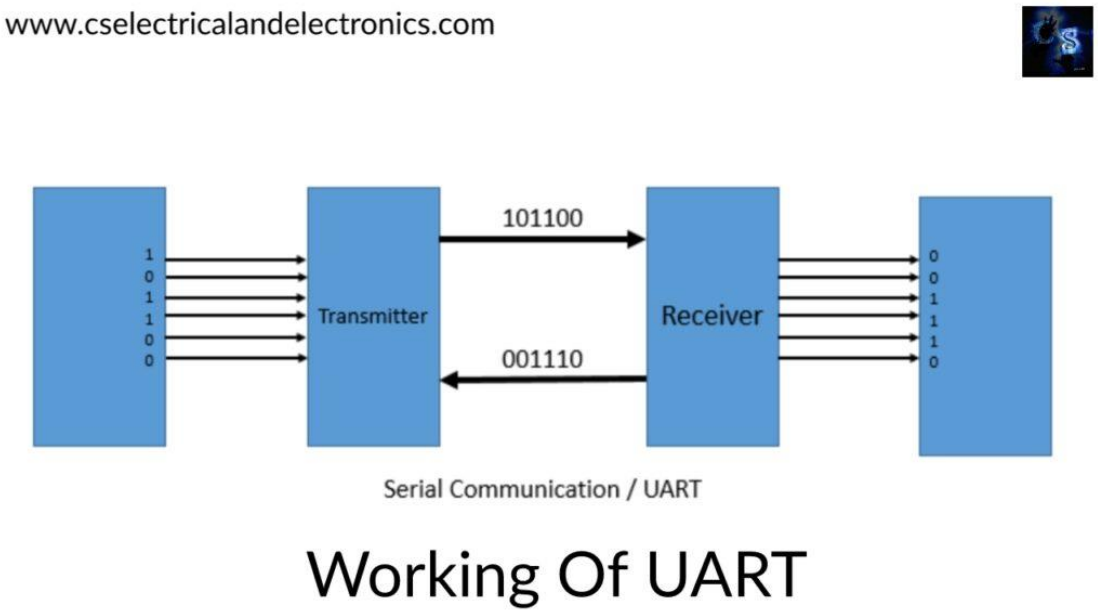
UART communication can also encounter several types of errors during data transmission:

- **Framing Error**: Occurs when the receiver does not detect the correct stop bit, causing misalignment in the data frame.
- **Parity Error**: Happens when the parity bit does not match the expected parity, indicating a corruption in the data.
- **Overrun Error**: Occurs when the receiver's buffer is full and cannot handle additional incoming data, causing data loss.

Key Features of the UART Protocol

- **Asynchronous Communication**: UART does not use a clock signal for synchronization. Instead, the sender and receiver must agree on a **baud rate**, which determines the number of bits transmitted per second. Both devices must operate at the same baud rate for successful communication.
- **Data Transmission**: UART transmits data **serially**, meaning bits are sent one after another on a single data line, rather than in parallel. This serial transmission reduces the number of required I/O pins, making UART a cost-effective solution for many embedded systems.
- **Start and Stop Bits**: Each data frame transmitted in the UART protocol begins with a **start bit** and ends with one or more **stop bits**:
    - **Start Bit**: The start bit is a single low signal (0) that signals the beginning of a data frame. It ensures the receiver is synchronized to the incoming data stream.
    - **Stop Bit(s)**: After the data bits, one or two stop bits are transmitted, represented as high signals (1). Stop bits signify the end of the data frame, allowing the receiver to recognize that the transmission is complete.
- **Data Bits**: The actual data being transmitted is broken down into a set of **data bits**, typically ranging from 5 to 9 bits. The data bits represent the information being sent, such as numeric or ASCII values.
- **Parity Bit**: The **parity bit** is an optional error-detection feature used to check if the data has been transmitted correctly. It helps detect errors in data transmission:
    - **Even Parity**: The number of 1's in the data bits, including the parity bit, must be even.
    - **Odd Parity**: The number of 1's in the data bits, including the parity bit, must be odd.
    - **No Parity**: No parity bit is included, and error detection is not performed for this field.
- **Baud Rate**: The **baud rate** is the rate at which data is transmitted, typically measured in bits per second (bps). Common baud rates include 9600, 19200, 115200, etc. Both the transmitter and receiver must operate at the same baud rate to ensure successful communication.

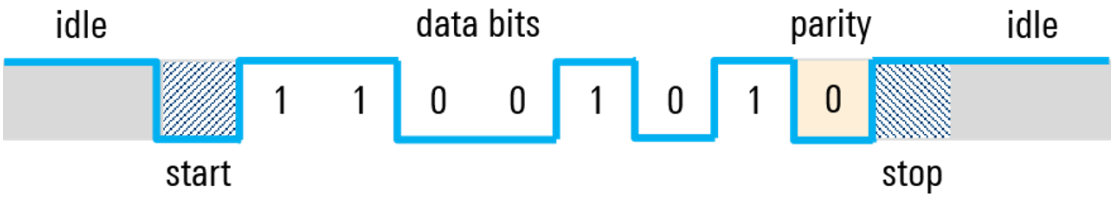
- **Flow Control**: UART communication may include **flow control** to manage the rate of data transmission between devices, ensuring that one device doesn't overwhelm the other. Two common methods of flow control are:
    - **Hardware Flow Control**: Uses control lines like **RTS (Request to Send)** and **CTS (Clear to Send)** to manage data flow.
    - **Software Flow Control**: Uses special characters like **XON/XOFF** to start and stop transmission.



Character Framing in UART

Character framing refers to the structure of the data that is transmitted or received over UART communication. It defines how individual pieces of data (characters) are encoded into a serial bit stream. This process is essential to the operation of UART, as it ensures that both the transmitting and receiving devices interpret the data correctly.

A typical UART frame consists of a series of bits that represent a **single character** (such as an ASCII character), including **start bits**, **data bits**, **parity bits** (optional), and **stop bits**. The configuration of these elements depends on the settings established by both the transmitter and receiver (e.g., baud rate, parity, number of data bits, and stop bits).



Features

- Wishbone interface in 32 bit and 8 bit data bus mode, and its selectable.
- FIFO only operation
- Debug interface in 32 bit data bus mode (debug 1 and debug2)
- Can be used to share data of 5, 6, 7, and 8 bits.
- Supports loopback mode through modem functionality.
- Interupt enable and Interupt identification registers for identifying interupts.
- Can produce parity for error checking.
- Line status register can be used for checking error / working status.
- Break interrupt generation.

Wishbone Protocol Signals

The **Wishbone** protocol is an open-source, highly flexible, and efficient bus protocol used in embedded systems for communication between different components, such as processors, peripherals, and memory. It is typically used in **system-on-chip (SoC)** designs. The protocol defines a set of signals that allow devices to communicate with each other via a shared bus.

Port	Width (bits)	Direction	Description
clk	1	input	Clock signal for UART
wb_rst_i	1	input	Asynchronous reset
wb_adr_i	3	input	Used for register selection
wb_sel_i	4	input	Used to select signal
wb_dat_i	8	input	Data input
wb_dat_o	8	input	Data output
wb_we_i	1	input	Write enable
wb_stb_i	1	input	Strobe signal for UART
wb_cyc_i	1	input	Cycle signal for UART
wb_ack_o	1	output	Acknowledge of transfer
wb_int_o	1	output	Interrupt output

External Signals

Port	Width (bits)	Direction	Description
Int_o	1	output	Interupt output
baud_o	1	output	Optional baudrate output signal
stx_pad_o	1	output	Serial transmission signal
srx_pad_i	1	input	Serial receiver signal
rts_pad_o	1	output	Request To Send
dtr_pad_o	1	Output	Data Terminal Ready
cts_pad_i	1	input	Clear To Send
dsr_pad_i	1	input	Data Set Ready
ri_pad_i	1	input	Ring Indicator
dcd_pad_i	1	input	Data Career Detect

Initialization

Upon reset the core performs the following tasks:

• The receiver and transmitter FIFOs are cleared.

• The receiver and transmitter shift registers are cleared

• The Divisor Latch register is set to 0.

• The Line Control Register is set to communication of 8 bits of data, no parity, 1 stop bit.

• All interrupts are disabled in the Interrupt Enable Register.

For proper operation, perform the following:

• Set the Line Control Register to the desired line control parameters. Set bit 7 to '1' to allow access to the Divisor Latches.

• Set the Divisor Latches, MSB first, LSB next.

• Set bit 7 of LCR to '0' to disable access to Divisor Latches. At this time the transmission engine starts working and data can be sent and received.

• Set the FIFO trigger level. Generally, higher trigger level values produce less interrupt to the system, so setting it to 14 bytes is recommended if the system responds fast enough.

• Enable desired interrupts by setting appropriate bits in the Interrupt Enable register.

Remember that (Input Clock Speed)/(Divisor Latch value) = 16 x the communication baud rate.

Since the protocol is asynchronous and the sampling of the bits is performed in the perceived middle of the bit time, it is highly immune to small differences in the clocks of the sending and receiving sides, yet no such assumption should be made when calculating the Divisor Latch values.

UART Internal Registers

The internal registers present in UART are Receiver Buffer, Transmitter Holding Register, Interrupt Enable Register, Interrupt Identification Register, FIFO Control Register, Line Control Register, Line Status Register, Modem Control Register, Modem Status Register, Divisor Latch LSB and Divisor Latch MSB Register.

Register Name	Address	Width	Access	Description
Receiver Buffer	0	8	R	Receiver FIFO output. It is used to collect the received data.
Transmitter Holding Register	0	8	W	Transmitter FIFO input. It is used to hold the data which is for data transmission.
Interrupt Enable Register	1	8	RW	Enable/ Mask interrupts generated by UART.
Interrupt Identification Register	2	8	R	Used to get the interrupt information.
FIFO Control Register	2	8	W	Used to control FIFO operations.
Line Control Register	3	8	RW	Used to control connections and to configure the character frame and to access the DLR or normal registers.
Modem Control Register	4	8	W	Used to control modem.
Line Status Register	5	8	R	Used to hold the information about the line status interrupts & it is updated by the IP core.
Modem Status Register	6	8	R	Used for modem status.
Divisor Latch Register (LSB)	0	8	RW	It is used to hold the divisor LSB value.
Divisor Latch Register (MSB)	1	8	RW	It is used to hold the divisor MSB value.

Interrupt Enable Register (IER)

This register allows enabling and disabling interrupt generation by the UART.

Bit	Access	Description
0	RW	Received data available interrupt 0–disabled, 1-enabled
1	RW	Transmitter holding register empty interrupt

		0–disabled, 1-enabled
2	RW	Received line status interrupt 0–disabled, 1-enabled
3	RW	Modem status interrupt 0–disabled, 1-enabled
7 - 4	RW	Reserved, should be logic 0

Interrupt Identification Register (IIR)

The IIR enables the programmer to retrieve what is the current highest priority pending interrupt. Bit 0 indicates that an interrupt is pending when it's logic '0'. When it's '1' – no interrupt is pending. The following table displays the list of possible interrupts along with the bits they enable, priority, and their source and reset control.

Bit 3	Bit 2	Bit1	Priority	Interrupt type	Interrupt source	Interrupt reset control
0	1	1	1st	Receiver Line Status	Parity, Overrun or Framing errors or Break Interrupt	Reading the Line Status Register
0	1	0	2nd	Receiver Data available	FIFO trigger level reached	FIFO drops below trigger level
1	1	0	2nd	Timeout Indication	There's at least 1 character in the FIFO but no character has been input to the FIFO or read from it for the last 4 Char times.	Reading from the FIFO (Receiver Buffer Register)
0	0	1	3rd	Transmitter Holding Register empty	Transmitter Holding Register Empty	Writing to the Transmitter Holding Register or reading IIR.
0	0	0	4th	Modem Status	CTS, DSR, RI or DCD.	Reading the Modem status register.

FIFO Control Register (FCR)

The FCR allows selection of the FIFO trigger level (the number of bytes in FIFO required to enable the Received Data Available interrupt). In addition, the FIFOs can be cleared using this register.

Bit	Access	Description
0	W	Ignored (Used to enable FIFOs in NS16550D). Since this UART only supports FIFO mode, this bit is ignored.
1	W	Writing a '1' to bit 1 clears the Receiver FIFO and resets its logic. But it doesn't clear the shift register, i.e. receiving of the current character continues.
2	W	Writing a '1' to bit 1 clears the Receiver FIFO and resets its logic. But it doesn't clear the shift register, i.e. receiving of the current character continues.
5-3	W	Ignored
7-6	W	Define the Receiver FIFO Interrupt trigger level '00' – 1 byte, '01' – 4 bytes, '10' – 8 bytes, '11' – 14 bytes

Line Control Register (LCR)

The line control register allows the specification of the format of the asynchronous data communication used. A bit in the register also allows access to the Divisor Latches, which define the baud rate. Reading from the register is allowed to check the current settings of the communication.

Bit	Access	Description
1-0	RW	Select number of bits in each character '00' – 5 bits, '01' – 6 bits, '10' – 7 bits, '11' – 8 bits
2	RW	Specify the number of generated stop bits '0' – 1 stop bit '1' – 1.5 stop bits when 5-bit character length selected and 2 bits otherwise
3	RW	Parity Enable '0' – No parity '1' – Parity bit is generated on each outgoing character and is checked on each incoming one.
4	RW	Even Parity selects '0' – Odd number of '1' is transmitted and checked in each word (data and parity combined). In other words, if the data has an even number of '1' in it, then the parity bit is '1'. '1' – Even number of '1' is transmitted in each word.
5	RW	Stick Parity bit. '0' – Stick Parity disabled '1' - If bits 3 and 4 are logic '1', the parity bit is transmitted and checked as logic '0'. If bit 3 is '1' and bit 4 is '0' then the parity bit is transmitted and checked as '1'.

6	RW	Break Control bit ‘1’ – the serial out is forced into logic ‘0’ (break state). ‘0’ – break is disabled
7	RW	Divisor Latch Access bit. ‘1’ – The divisor latches can be accessed ‘0’ – The normal registers are accessed

Modem Control Register (MCR)

The modem control register allows transferring control signals to a modem connected to the UART.

Bit	Access	Description
0	W	Data Terminal Ready (DTR) signal control ‘0’ – DTR is ‘1’ ‘1’ – DTR is ‘0’
1	W	Request To Send (RTS) signal control ‘0’ – RTS is ‘1’ ‘1’ – RTS is ‘0’
2	W	Out1. In loopback mode, connected Ring Indicator (RI) signal input
3	W	Out2. In loopback mode, connected to Data Carrier Detect (DCD) input.
4	W	Loopback mode ‘0’ – normal operation ‘1’ – loopback mode. When in loopback mode, the Serial Output Signal (STX_PAD_O) is set to logic ‘1’. The signal of the transmitter shift register is internally connected to the input of the receiver shift register. The following connections are made: DTR → DSR RTS → CTS Out1 → RI Out2 → DCD
7-5	W	Ignored

Line Status Register (LSR)

Bit	Access	Description
0	R	Data Ready (DR) indicator. ‘ 0’ – No characters in the FIFO ‘1’ – At least one character has been received and is in the FIFO.
1	R	Overrun Error (OE) indicator ‘1’ – If the FIFO is full and another character has been received in the receiver shift register. If another character is starting to arrive, it will overwrite the data in the shift register but the FIFO will remain intact.

		The bit is cleared upon reading from the register. Generates Receiver Line Status interrupt. ‘0’ – No overrun state
2	R	Parity Error (PE) indicator ‘1’ – The character that is currently at the top of the FIFO has been received with parity error. The bit is cleared upon reading from the register. Generates Receiver Line Status interrupt. ‘0’ – No parity error in the current character
3	R	Framing Error (FE) indicator ‘1’ – The received character at the top of the FIFO did not have a valid stop bit. Of course, generally, it might be that all the following data is corrupt. The bit is cleared upon reading from the register. Generates Receiver Line Status interrupt. ‘0’ – No framing error in the current character
4	R	Break Interrupt (BI) indicator ‘1’ –A break condition has been reached in the current character. The break occurs when the line is held in logic 0 for a time of one character (start bit + data + parity + stop bit). In that case, one zero character enters the FIFO and the UART waits for a valid start bit to receive next character. The bit is cleared upon reading from the register. Generates Receiver Line Status interrupt. ‘0’ – No break condition in the current character
5	R	Transmit FIFO is empty. ‘1’ – The transmitter FIFO is empty. Generates Transmitter Holding Register Empty interrupt. The bit is cleared when data is being been written to the transmitter FIFO. ‘0’ – Otherwise
6	R	Transmitter Empty indicator. ‘1’ – Both the transmitter FIFO and transmitter shift register are empty. The bit is cleared when data is being been written to the transmitter FIFO. ‘0’ – Otherwise
7	R	‘1’ – At least one parity error, framing error or break indications have been received and are inside the FIFO. The bit is cleared upon reading from the register. ‘0’ – Otherwise.

OBJECTIVES

The primary objectives of this project were to verify the functionality of the UART protocol using UVM, covering multiple communication modes and error conditions. Below are the key objectives, along with a description of how each one was achieved during the project:

Objective 1: Verify Full-Duplex Communication

- **Goal**: To ensure that data can be transmitted and received simultaneously in both directions (transmitter and receiver) without interference, in a full-duplex UART setup.

- **How it was met**: A test was created to simulate data transmission from the master to the slave and from the slave to the master simultaneously. The UVM testbench ensured that both sides handled the communication independently while maintaining proper synchronization. The scoreboard verified the accuracy of the transmitted data, ensuring no corruption occurred due to simultaneous transfers.

Objective 2: Verify Half-Duplex Communication

- **Goal**: To test the UART's functionality in half-duplex mode, where data can only flow in one direction at a time.

- **How it was met**: A half-duplex communication test was implemented in the testbench. The master and slave alternated between transmission and reception, ensuring the protocol handled the switching of directions without data loss or timing issues. The scoreboard was used to check if data was properly received after the direction was switched.

Objective 3: Implement and Verify Loopback Mode

- **Goal**: To verify the UART's loopback functionality, where the transmitted data from the transmitter is received by the same device (i.e., the master sends data,

and it is received back by the master or the slave transmits and receives the same data).

- **How it was met**: A loopback mode was simulated where data sent from the master was expected to return from the slave (or vice versa), ensuring that the transmitter and receiver circuits functioned correctly in a self-contained loop. The scoreboard tracked the data flow to confirm that the received data matched the transmitted data, thus verifying correct operation of loopback mode.

Objective 4: Test Various Error Conditions

- **Goal**: To ensure that the UART protocol correctly handles errors, such as framing errors, parity errors, and data corruption.

- **How it was met**: Multiple error scenarios were implemented in the UVM testbench, including:

    - **Framing Errors**: Simulated errors where the no. of bit in transmitted and received character is not the same.
    - **Parity Errors**: Introduced parity mismatches between the transmitted and received data.
    - **Overrun Errors**: simulated overrunning of data from one side of the UART
    - **Break Interrupt Errors**: introduced forced break interupt (logic 0).
    - **THR Empty Errors:** simulated error of transmitted data not being present.
    - **Timeout Errors:** simulated timeout of data.

- The testbench and scoreboard were designed to detect these errors, with the scoreboard verifying that the receiver correctly identifies and handles these

error conditions. Additionally, assertions were added to flag errors during the simulation, helping identify the root causes of failures.

SCOREBOARD

The scoreboard is a critical component in the verification environment, used to monitor the communication between the master and slave sides of the UART protocol. Its primary role is to compare the transmitted and received data, ensuring that they match correctly and that all test scenarios—ranging from standard data transfers to error conditions—are appropriately handled.

Key Features of the Scoreboard:

1. **Data Comparison**: The scoreboard compares the transmitted data from the master with the data received by the slave (or vice versa, depending on the communication direction). It checks that the bits, including start, data, parity, and stop bits, align correctly to validate the integrity of the transmission. Any mismatch or corruption is flagged for further analysis.

2. **Coverage Model**: To ensure comprehensive verification, a **coverage model** was implemented within the scoreboard. The coverage model tracks and covers the various bits in the registers that are being tested during the UART communication. This includes:

    - **Data Bits**: Ensuring that various data bit sizes are being used.
    - **Parity Bits**: Verifying that various parity methods are being used.
    - **Start and Stop Bits**: tracking if stop bits are being used.
    - **Register bits:** Ensuring various register bits are being triggered according to the sequences being checked.

3. The coverage model helps ensure that all important aspects of the UART protocol, including edge cases and different configurations, are exercised during the simulation. It provides a comprehensive overview of which parts of the communication have been covered, and it ensures that the testbench verifies all possible scenarios defined in the test plan.

4. **Error Detection**: The scoreboard is designed to detect and flag various error scenarios that were implemented in the testbench. This includes:

- **Framing Errors**: The scoreboard checks if the expected bit size is correctly received, and flags framing errors when mismatches occur.
- **Parity Errors**: The scoreboard verifies whether the received data's parity matches the expected parity. Any mismatches in parity bits are flagged as errors.
- **Overrun Errors**: The scoreboard verifies whether the received data overruns. Any mismatches in overrun bits are flagged as errors
- **Break Interrupt Errors**: The scoreboard checks if the UART detects break interupt, and flags when break interupt occur.
- **THR Empty Errors:** The scoreboard verifies whether any data is present in Transmitter Holding Register. If it is found empty, it is flagged as an error.
- **Timeout Errors:** The scoreboard verifies whether any data is pending to be shifted to FIFO. If it is timeout, it is flagged as an error.

5. These error detection mechanisms help identify issues early and ensure that the UART protocol can handle various communication failures effectively.

6. **Scenario Handling**: The scoreboard was specifically written to handle the different communication scenarios tested in this project, such as:

    - **Full-Duplex and Half-Duplex Operations**: It handles the validation of bidirectional data transmission and reception in both full-duplex and half-duplex communication modes.
    - **Loopback Mode**: In loopback mode, where data transmitted by the master is expected to be received back, the scoreboard ensures that the data sent matches the data received.
    - **Error Handling**: It checks whether the system properly handles framing errors, parity mismatches, and any other injected faults.
7. The scoreboard not only checks for expected data correctness but also tracks coverage of each scenario, ensuring that no condition is left untested.

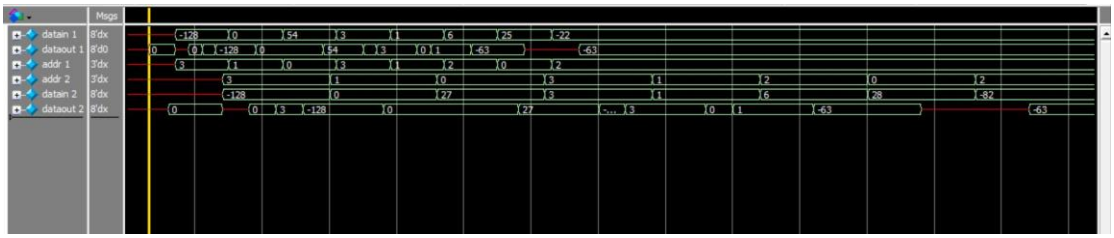
8. **Reporting**: The scoreboard generates detailed reports on the results of the data comparison and error detection. It highlights any discrepancies between transmitted and received data and provides a summary of test coverage. This report is essential for debugging and understanding the behavior of the UART protocol under different conditions.

WAVEFORMS

In this section, we will discuss the waveforms generated during the simulation for various UART test scenarios, including **Full-Duplex**, **Half-Duplex**, **Loopback**, and several **Error Conditions** such as **Parity Error**, **Framing Error**, **Overrun Error**, **Break Interrupt**, and **THR Empty Error**. These waveforms illustrate the communication between the master and slave, showing the transmission of data, timing signals, and error conditions.

1. Full-Duplex Communication

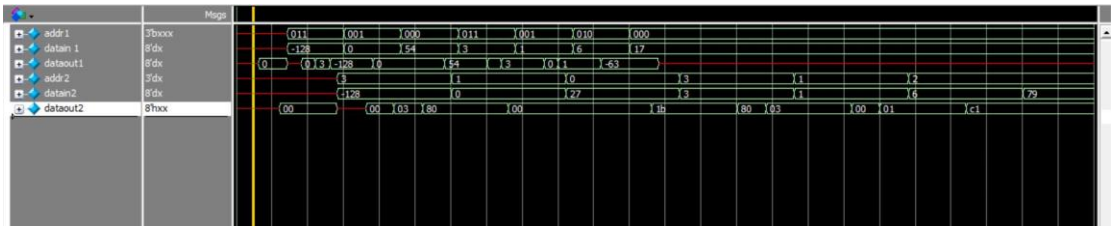
In full-duplex mode, data can be transmitted and received simultaneously in both directions. The waveform should show two bidirectional lines where the master transmits data to the slave while the slave sends data back to the master.



- **Waveform Explanation**:
    - The **Tx (Transmit)** line from the master sends out a data frame, which includes start bits, data bits, parity, and stop bits.
    - Simultaneously, the **Rx (Receive)** line from the slave sends data back to the master in the reverse direction.
    - Both transmissions occur concurrently without any interference.
    - The scoreboard verifies that the data sent by the master matches what the slave receives and vice versa.

2. Half-Duplex Communication

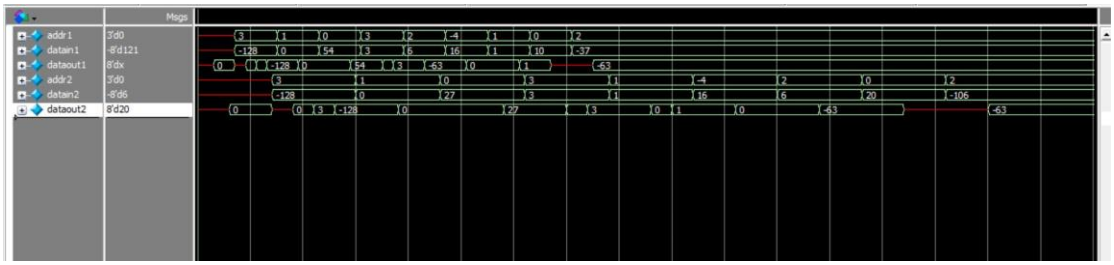
In half-duplex mode, only one device can transmit at any given time—either the master or the slave. The waveform should show alternating transmission between the master and the slave.



- **Waveform Explanation**:
    - Initially, the master sends a data frame. After completion, the master stops transmitting, and the slave starts sending a data frame back.
    - This alternating transmission occurs in one direction at a time.
    - The scoreboard confirms that the data is correctly transmitted and received by the slave after each transmission phase, ensuring synchronization.

3. Loopback Mode

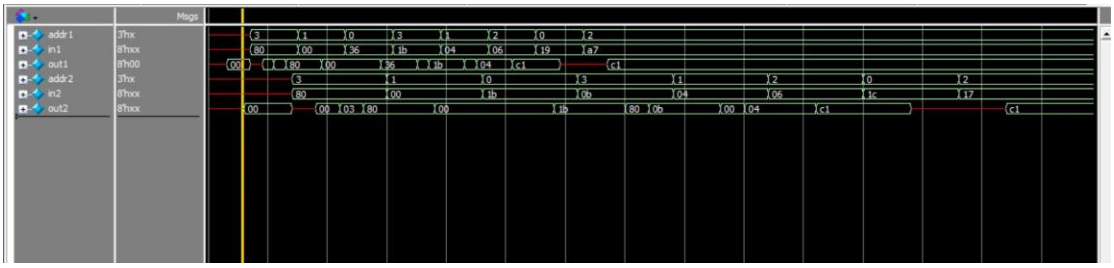
In loopback mode, the transmitted data from the master (or slave) is immediately received back by the same device. This test ensures that the transmit and receive paths within the same device are functioning correctly.



- **Waveform Explanation**:
    - The master sends out data on the **Tx** line, and this data is immediately looped back and received by the **Rx** line, which should match the transmitted data.
    - The waveform shows the master transmitting data followed by the immediate reception of the same data from the loopback.
    - The scoreboard compares the transmitted and received data, ensuring there is no corruption or mismatch.

4. Parity Error

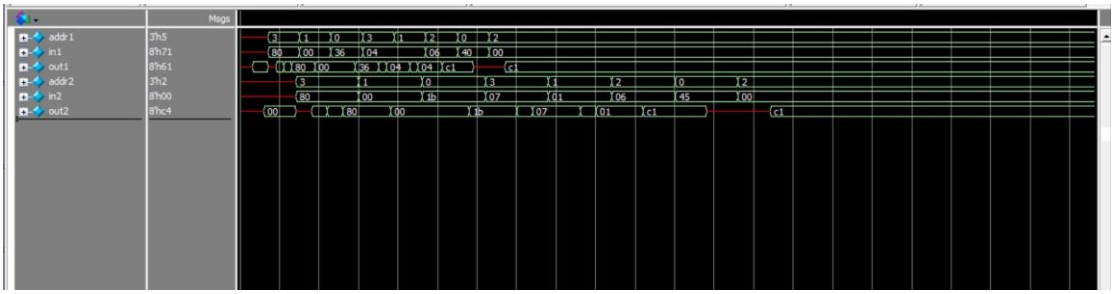
A **Parity Error** occurs when the calculated parity (even or odd) of the transmitted data does not match the expected parity at the receiver.



- **Waveform Explanation**:
    - The waveform shows a transmission from the master with data bits, including a **parity bit** (even or odd).
    - On the receive side, the slave checks the received data's parity. If there's a mismatch between the expected and received parity, a parity error is flagged.
    - The scoreboard detects the error and flags a parity mismatch, ensuring that the UART handles this condition properly.

5. Framing Error

A **Framing Error** occurs when theno. of bit is not set correctly, or the expected framing of data (start, data, parity, and stop bits) is not followed.

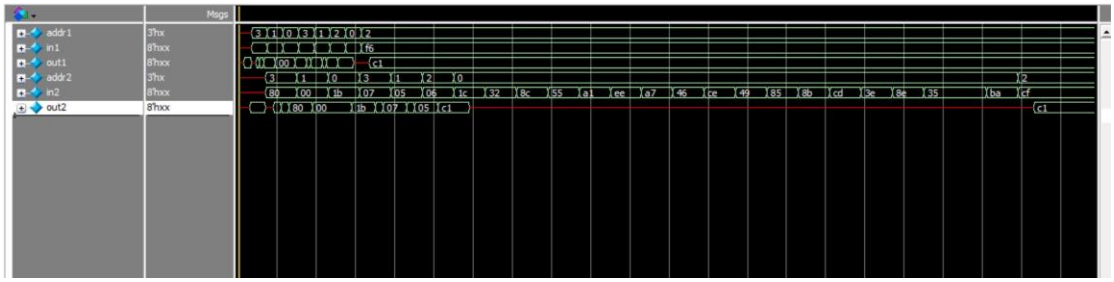


- **Waveform Explanation**:
    - The waveform shows a normal data transmission, but at some point, the stop bit might not be received correctly or is misaligned, causing a framing error.

- The receiver fails to detect the correct stop bit, which leads to a framing error flag being set.
- The scoreboard detects this mismatch and reports the error, ensuring the UART protocol's ability to handle framing errors.

6. Overrun Error

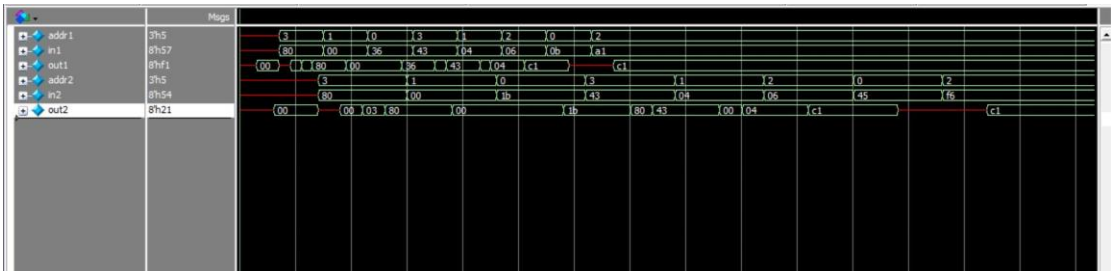
An **Overrun Error** happens when data is being received faster than the receiver can process, causing data to be lost.



- **Waveform Explanation**:
  - The waveform shows the master continuously transmitting data. The slave is unable to process the data in time, and the receiver buffer overflows.
  - The **Rx** line continues to receive data, but due to the overflow, some bits are lost, causing an overrun error.
  - The scoreboard detects this overrun condition by comparing the expected and actual data, flagging any missing or corrupted data.

7. Break Interrupt

A **Break Interrupt** occurs when there's a continuous low signal on the transmission line, indicating a break condition.

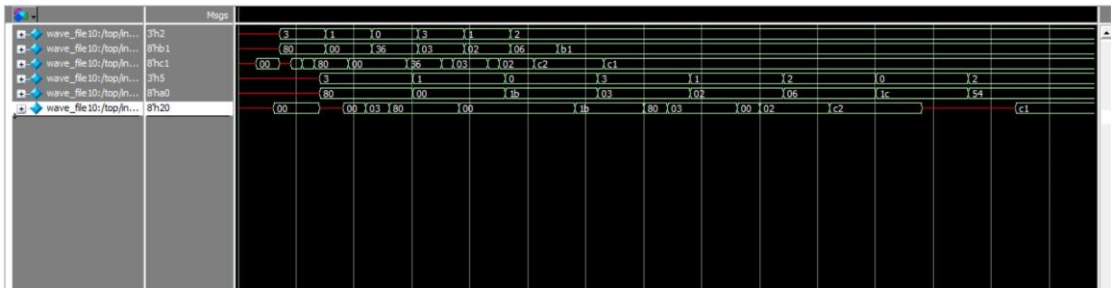


- **Waveform Explanation**:

- The waveform shows a break in transmission, where the data line stays low for a longer period than expected, signaling a break interrupt.
- This break is interpreted as a control signal, interrupting normal communication.
- The scoreboard detects this condition and ensures that the UART system correctly identifies and handles the break interrupt.

8. THR Empty Error

The **THR (Transmitter Holding Register) Empty Error** occurs when the transmitter runs out of data to send (i.e., the THR register becomes empty) before the transmission is complete.



- **Waveform Explanation**:
  - The waveform shows the master starting a data transmission, but it runs out of data before completing the frame. This causes the THR to be empty, resulting in a THR empty error.
  - The UART should handle this by either waiting for new data or generating an error condition.
  - The scoreboard tracks this condition and ensures the UART system properly handles this error without causing undefined behavior.