KLE Technological University

Creating Value
Leveraging Knowledge

School of
Electronics and Communication Engineering

Minor-2 Project Report

on

DESIGN AND IMPLEMENTATION OF UART

By:

1. **Chetan Paranatti** USN: 01FE21BEC163
2. **Manjunath Inamati** USN: 01FE21BEC356
3. **Goutami Naragund** USN: 01FE21BEC177

Semester: VI, 2023-2024

Under the Guidance of

Saroja V Siddamal

K.L.E SOCIETY'S
KLE Technological University,
HUBBALLI-580031
2023-2024

SCHOOL OF ELECTRONICS AND COMMUNICATION ENGINEERING

CERTIFICATE

This is to certify that project entitled " **DESIGN AND IMPLEMENTATION OF UART** " is a bonafide work carried out by the student team of **"Chetan Paranatti (01FE21BEC163), Manjunath Inamati (01FE21BEC356), Goutami Naragund (01FE21BEC177)"**. The project report has been approved as it satisfies the requirements with respect to the minor-2 project work prescribed by the university curriculum for BE (VI Semester) in School of Electronics and Communication Engineering of KLE Technological University for the academic year 2023-2024.

Saroja V Siddamal	**Suneeta V Budihal**	**B. S. Anami**
Guide	**Head of School**	**Registrar**

External Viva:

Name of Examiners	**Signature with date**
1. Saroja V. S	20/6/24
2. Gireesha H.M.	20/6/24

ACKNOWLEDGMENT

We express our gratitude to the faculty and management for their professional guidance through out the project's completion. We would like to extend our thanks to Dr. Ashok Shettar, Vice-Chancellor at KLE Technological University, Hubballi, for their visionary leadership and unwavering support. Special thanks to Dr. Suneeta V Budihal, Professor and Head of SoECE, for providing us with direction and fostering our skills and academic growth. We thank our guide, Dr. Saroja V Siddamal, for their constant guidance during interactions and reviews. We acknowledge the reviewers for their valuable suggestions and inputs. Addition ally, we express our thanks to the Project Coordinator Prof. Rajeshwari M. and other faculty in-charges for their support throughout the completion of the project.

Chetan Paranatti
Manjunath Inamati
Goutami Naragund

ABSTRACT

This project focuses on the design and implementation of a Universal Asynchronous Receiver-Transmitter (UART) system. In response to industry specifications, the primary goal was to ensure reliable serial communication by meeting all the specified baud rates, data bit configurations, parity, and stop bit settings. Extensive testing and verification were conducted to confirm that the UART performs accurately under various conditions. The successful integration and validation on the ARTY-7 demonstrate that our design meets industry standards and specifications.The implementation process involved a systematic approach, starting from initial design and coding to simulation and hardware testing. The results indicate a high degree of reliability and performance, validating our design choices and methodologies. This project showcases our capability to develop industry-standard communication protocols on modern FPGA platforms, providing a valuable contribution to the field of digital communication systems.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Universal Asynchronous Receiver-Transmitter (UART) is a critical component in serial communication systems, facilitating data transmission between devices. UART's widespread use in various applications, including embedded systems, computer peripherals, and communication modules, underscores its significance in modern technology. Given the increasing complexity and demand for reliable data communication, the design and implementation of a high-performance UART that meets industry specifications is essential.

1.1 Motivation

Designing and implementing a UART system offers a comprehensive learning experience in digital design and FPGA programming. This project allows us to apply theoretical knowledge in a practical context, enhancing our understanding of hardware description languages (HDLs) and digital communication protocols.This project hones various technical skills, including Verilog coding, FPGA implementation, and the use of debugging and validation tools. These skills are crucial for careers in electronics, computer engineering, and related fields, making this project an invaluable part of our professional development.

1.2 Objectives

The primary objectives of this project are:

1. To design a UART system in Verilog that adheres to industry-specified parameters.
2. To implement the designed UART on the ARTY-7 FPGA platform.
3. To validate the functionality and performance of the UART using VIO Chipscope.
4. To ensure that UART meets all specified baud rates, data bit configurations, parity, and stop bit settings.

1.3 Literature survey

In designing and implementing the Universal Asynchronous Receiver/Transmitter (UART) on the ARTY-7 platform using VIO Chipscope, we referred to the following key resources:

1. **Texas Instruments KeyStone Architecture Universal Asynchronous Receiver/Transmitter (UART) User Guide (SPRUGP1, November 2010)[2]**

   This guide provides a comprehensive overview of the UART module as part of Texas Instruments' KeyStone Architecture. It details the UART's functional specifications, register descriptions, and operational modes. It was crucial in understanding the foundational aspects of UART design and implementation. It Offers an overview of UART and its significance in serial communication. Describes various registers involved in UART operations, such as Line Status Register (LSR) and Modem Status Register (MSR). Provide additional technical details and guidelines for UART configuration and usage. This guide helped ensure that our implementation adhered to industry standards and best practices for UART design.

2. **"UART16550 Core Technical Manual" by Jacob Gorban[1]**

   This technical manual was indispensable for comprehending the intricacies of the UART16550 core. It detailed the architecture, interface signals, clock requirements, registers, and operational guidelines. The thorough explanations of each register and control mechanism were particularly beneficial for ensuring accurate implementation and functionality of the UART module in our project.

1.4 Problem statement

The objective of this project is to design and implement a Universal Asynchronous Receiver-Transmitter (UART) system that meets industry specifications for serial communication. The UART must support various baud rates, data bit configurations, parity, and stop bit settings, ensuring reliable and efficient data transmission. The implementation will be carried out on the ARTY-7 FPGA platform, utilizing VIO Chipscope for testing and validation.

1.5 Application in Societal Context

The Universal Asynchronous Receiver-Transmitter (UART) system designed in this project has significant applications in various societal contexts, enhancing communication and technology in numerous fields.

- UART is widely used in medical devices such as patient monitors, diagnostic equipment, and wearable health trackers. Reliable data communication between these devices and central systems is crucial for accurate monitoring and timely interventions.
- UART enables communication between smart home devices such as thermostats, security systems, and home automation hubs, creating integrated and efficient home environments.
- UART is used in routers, modems, and other networking devices for configuration and debugging, ensuring robust and efficient network infrastructure.

By ensuring reliable and efficient data communication, the UART system designed in this project has the potential to significantly impact these areas, improving technology and services that are integral to modern society.

1.6 Project Planning

The project was divided into six phases, In the first phase, the focus was on research and requirements gathering. This includes conducting a literature review to understand UART fundamentals and defining project requirements. Phase two, from weeks 5-8, involves the design phase where the overall architecture of the UART module will be created, including detailed schematics for the transmitter and receiver sections, and configuration specifications. The third phase, implementation, involves coding the UART module in Verilog HDL, creating testbenches for initial functionality testing, and debugging any issues. Comprehensive testing is conducted using extensive simulations to validate the design. Hardware testing is also carried out to validate real-world performance.

1.7 Organization of the report

Name of chapter 2 and brief description about it
Name of Chapter 3 and brief description about it and so on

Chapter 2

System design

A UART data frame starts with a start bit (logic low), followed by the data bits (5 to 9 bits), an optional parity bit for error checking, and ends with one or more stop bits (logic high).

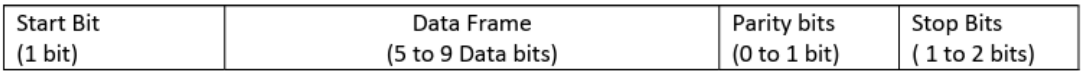

Figure 2.1: Dataframe

Components of the UART Data Frame:

1. **Start Bit (1 bit):**
   - The start bit signals the beginning of a data frame.
   - It is always a logic low (0) signal.
   - It tells the receiver that data transmission is starting and synchronizes the receiver with the transmitter.
2. **Data Frame (5 to 9 Data bits):**
   - This section contains the actual data being transmitted.
   - The number of data bits can vary from 5 to 9, depending on the configuration.
   - The data bits are transmitted from the least significant bit (LSB) to the most significant bit (MSB).
3. **Parity Bit (0 to 1 bit):**
   - Parity bits are optional and can be used for error checking.
   - If enabled, it can be either even parity or odd parity:
   - **Even parity:** The parity bit is set so that the total number of 1s in the data frame (including the parity bit) is even.
   - **Odd parity:** The parity bit is set so that the total number of 1s in the data frame (including the parity bit) is odd.
   - If parity checking is not used, this bit is omitted.

4. **Stop Bits (1 to 2 bits):**
    - Stop bits indicate the end of the data frame.
    - It is used to signal that the data transmission for the current frame is complete.
    - It can be 1, 1.5, or 2 bits long (commonly 1 or 2 bits).
    - The stop bit is always a logic high (1) signal.
    - It provides a pause so the receiver can process the received bits and prepare for the next data frame.

UART Registers

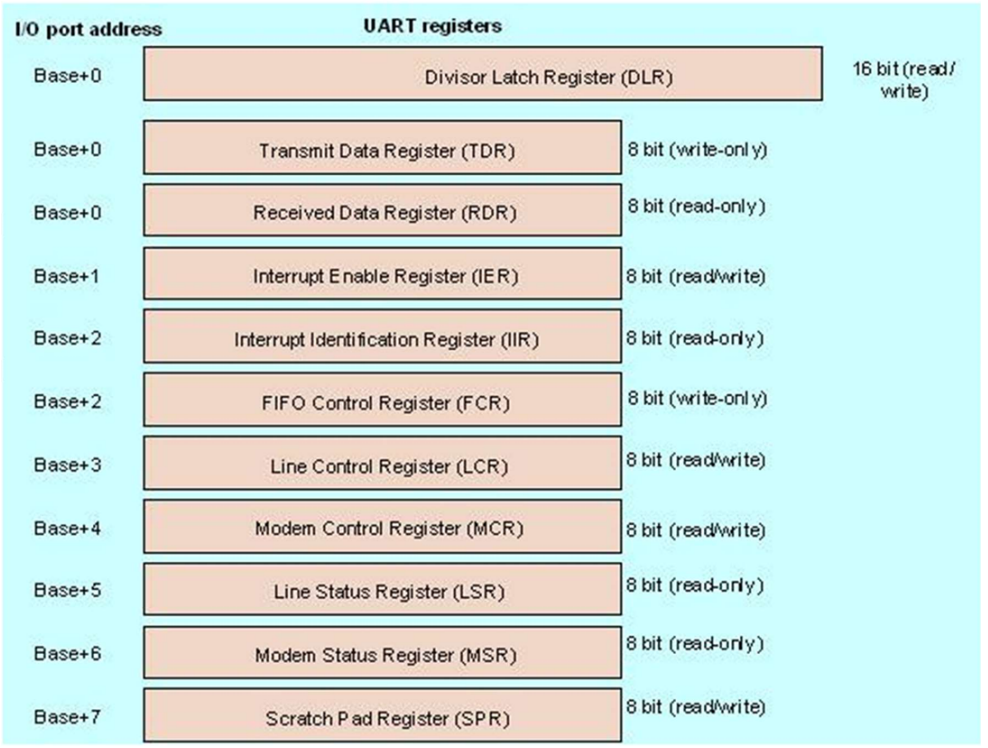

Figure 2.2: Control Registers

1. **Divisor Latch Register (DLR):** The DLR is used to set the baud rate for serial communication by specifying a divisor value. It is 16 bits wide, meaning it uses two 8-bit registers.
2. **Transmit Data Register (TDR) and Received Data Register (RDR):** These registers are used for transmitting and receiving data. The TDR is write-only, meaning data can be written to it for transmission. The RDR is read-only, meaning data can be read from it when received.
3. **Interrupt Enable Register (IER):** This register enables or disables different UART interrupts, such as those for received data, transmit buffer empty, line status, and modem status.

4. **Interrupt Identification Register (IIR):** This read-only register identifies the source of an interrupt, helping the processor to handle the interrupt appropriately.
5. **FIFO Control Register (FCR):** This write-only register controls the FIFO (First-In-First-Out) buffers for the transmitter and receiver, including enabling/disabling FIFOs and clearing them.
6. **Line Control Register (LCR):** This register configures the data format, including the number of data bits, stop bits, and parity settings.
7. **Modem Control Register (MCR):** This register is used to control the modem interface, including signaling like Request to Send (RTS) and Data Terminal Ready (DTR).
8. **Line Status Register (LSR):** This read-only register provides status information about the data transmission and reception, such as data ready, overrun errors, and transmitter empty status.
9. **Modem Status Register (MSR):** This read-only register provides the status of modem control lines like Clear to Send (CTS) and Data Set Ready (DSR).
10. **Scratch Pad Register (SPR):** This register is used for temporary storage of data. It is a general-purpose register with no specific function defined by the UART.

A comprehensive understanding of the bit-wise specifications of each UART register is mentioned in the final design section.

2.1 Functional block diagram

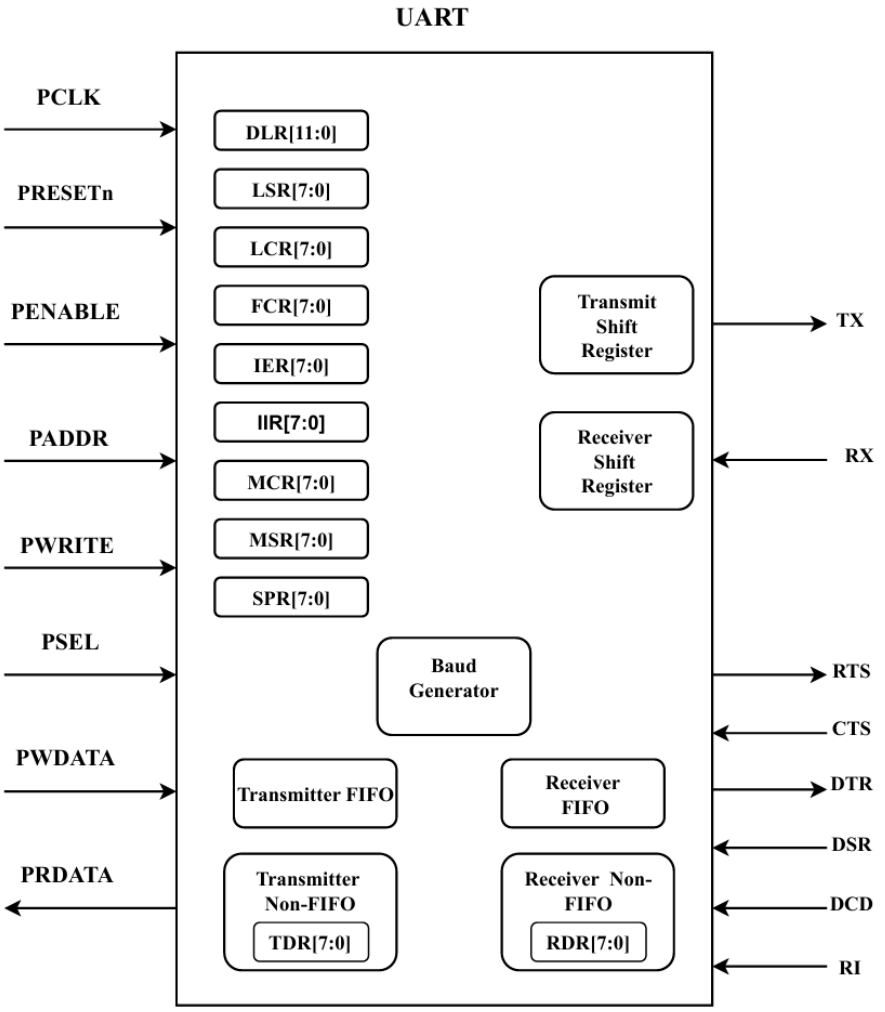

Figure 2.3: Functional Block Diagram

This diagram encapsulates a typical UART setup used for serial communication in embedded systems, providing configurable and reliable data transmission and reception capabilities.

- The various **registers (DLR, LSR, LCR, FCR, IER, IIR, MCR, MSR, SPR)** provide configuration settings, status information, and control for the UART operations. The Baud Generator sets the data transmission rate by dividing the input clock.
- The **FIFO buffers (Transmit FIFO and Receiver FIFO)** temporarily store data to be transmitted and received, respectively. This helps in managing data flow and reducing the chance of data loss, especially at higher data rates.
- The **Transmit Shift Register and Receiver Shift Register** handle the actual serial transmission and reception of data bits through the TX and RX pins.

- **Modem control signals (RTS, CTS, DTR, DSR, DCD, RI)** are used for hardware handshaking, ensuring proper data flow control between the UART and external devices.
- **PCLK** provides the clock signal necessary for synchronous operations within the UART.
- **PRESETn** initializes the UART, setting all registers to their default states when asserted.
- **PENABLE, PSEL, and PADDR** work together to select and access specific peripheral.
- **PWRITE** indicates whether the operation is a read or write.
- **PWDATA** is used to send data to be written to the peripheral, while **PRDATA** is used to read data from the peripheral.

2.2 Final design

2.2.1 Bit-wise specifications of each UART register.

1. **Divisor Latch Register (DLR):**

.

Table 2.1: Divisor Latch Register (DLR)

Name	Access	Description
Divisor latch byte 1 (LSB)	RW	The LSB of the divisor latch
Divisor latch byte 2 (MSB)	RW	The MSB of the divisor latch

The divisor latches can be accessed by setting the 7th bit of LCR to '1'. You should restore this bit to '0' after setting the divisor latches in order to restore access to the other registers that occupy the same addresses. The 2 bytes form one 16-bit register, which is internally accessed as a single number. You should therefore set all 2 bytes of the register to ensure normal operation. The register is set to the default value of 0 on reset, which disables all serial I/O operations in order to ensure explicit setup of the register in the software. The value set should be equal to (system clock speed) / (16 x desired baud rate). The internal counter starts to work when the LSB of DL is written, so when setting the divisor, write the MSB first and the LSB last.

$$\text{Divisor} = \frac{\text{UART Input Clock Frequency}}{\text{Desired Baud Rate} \times 16}$$

Two 8-bit register fields (DLH and DLL), called divisor latches, hold this 16-bit divisor. DLH holds the most significant bits of the divisor, and DLL holds the least significant bits of the divisor. These divisor latches must be loaded during the initialization of the UART to ensure the desired operation of the baud generator. Writing to the divisor latches results in two wait states being inserted during the write access while the baud generator is loaded with the new value.
Note: To access DLR, the 7th bit of LCR should be '1' for some amount of time, after which you should restore the bit to '0'.

2. **Transmit Data Register (TDR):**
The UART transmitter section consists of a transmit data register (TDR) and a transmit shift register (TSR). Transmitter control is a function of the line control register (LCR). The TDR receives data from the internal data bus. When the TSR is idle the UART then moves the data from the TDR to the TSR. The UART serializes the data in the TSR and transmits the data on the TX pin.

Table 2.2: Received Data Register (RDR)

Bit	Description
0-7	Data to transmit

3. **Received Data Register (RDR):**
The UART receiver section consists of a receiver shift register (RSR) and a received data register (RDR). The RSR receives serial data from the RX pin. Then the RSR concatenates the data and moves it into the RDR.

Table 2.3: Received Data Register (RDR)

Bit	Description
0-7	Received data

4. **Interrupt Enable Register (IER):**
This register allows enabling and disabling interrupt generation by the UART.

Table 2.4: Interrupt Enable Register (IER)

Bit	Access	Description
0	RW	**Received Data available interrupt.** '0' - disabled '1' - enabled
1	RW	**Transmit Data Register empty interrupt.** '0' - disabled '1' - enabled
2	RW	**Receiver Line Status Interrupt.** '0' - disabled '1' - enabled
3	RW	**Modem Status Interrupt.** '0' - disabled '1' - enabled
7-4	RW	**Reserved. It should be logic '0'.**

5. **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.

Table 2.5: Interrupt Identification Register (IIR)

Bit	Access	Description
0	R	**Interrupt Pending.** '0' - Interrupt is pending. '1' - Interrupt is not pending.
1	R	**Receiver Line Status.** Overrun error, parity error, and framing error. '0' - No errors. '1' - Either overrun or parity or framing error has occurred.
2	R	**Receiver Data Available.** **In non-FIFO mode:** '0' = Receiver data register (RDR) is empty. '1' = Receiver data register (RDR) is not empty. **In FIFO mode:** '0' = Receiver FIFO is empty. '1' = Receiver FIFO is not empty.
3	R	**Transmit Data Register (TDR) Empty.** **In non-FIFO mode:** '0' = Transmit data register (TDR) is not empty. '1' = Transmit data register (TDR) is empty. **In FIFO mode:** '0' = Transmit FIFO is not empty. '1' = Transmit FIFO is empty.
4	R	**Modem Status.** '0' - Disabled. '1' - Enabled.
7-5	R	**Reserved.**

Bits 5: Logic '0'.
Bits 6 and 7: Logic '1' for compatibility reasons.

6. **FIFO Control Register (FCR):**
   The FCR allows the selection of UART operation mode (FIFO / Non FIFO). In addition, the FIFOs can be cleared using this register.

Table 2.6: FIFO Control Register (FCR)

Bit	**Access**	**Description**
0	W	**Transmitter and receiver FIFOs mode enable.** '0' - Non-FIFO mode. The transmitter and receiver FIFOs are disabled, and the FIFO pointers are cleared. '1' - FIFO mode. The transmitter and receiver FIFOs are enabled.
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 2 clears the Transmitter FIFO and resets its logic. The shift register is not cleared, i.e., transmitting of the current character continues.
5-3	W	**Ignored.**
7-6	W	**Reserved.**

7. **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.

Table 2.7: Line Control Register (LCR)

Bit	Access	Description
1-0	RW	**Select number of bits in each character.** '00'- 5bits '01'- 6bits '10'- 7bits '11'- 8bits
2	RW	**Specify the number of generated stop bits.** '0'- 1 stop bit '1'- 1.5 stop bits when 5-bit character length is selected and 2 bits otherwise Note that the receiver always checks the first stop bit only.
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 select.** '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. (Ignored)**
6	RW	**Ignored.**
7	RW	**Divisor Latch Access bit**. '1'- The divisor latches can be accessed '0'- The normal registers are accessed

8. **Modem Control Register (MCR):**
   The modem control register allows the transferring of control signals to a modem connected to the UART.

Table 2.8: Modem Control Register (MCR)

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, the 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. 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	RW	**Ignored.**

9. **Line Status Register (LSR):**
   LSR register provides the status of data transmission and reception. It contains various flags that indicates the status of communication.

Table 2.9: Line Status Register $(LSR)_1$

Bit	Access	Description
0	R	**Data-ready (DR) indicator for the receiver.** **In non-FIFO mode:** '0' = Data is not ready, or the DR bit was cleared because the character was read from the received data register (RDR). '1' = Data is ready. A complete incoming character has been received and transferred into the received data register (RBR). **In FIFO mode:** '0' = Data is not ready, or the DR bit was cleared because all of the characters in the receiver FIFO have been read. '1' = Data is ready. There is at least one unread character in the receiver FIFO. If the FIFO is empty, the DR bit is set as soon as a complete incoming character has been received and transferred into the FIFO. The DR bit remains set until the FIFO is empty again.
1	R	**Overrun Error (OE) indicator.** **In non-FIFO mode:** '0' = No overrun error has been detected, or the OE bit was cleared because the CPU read the content of the line status register (LSR). '1' = Overrun error has been detected. Before the character in the received data register (RDR) could be read, it was overwritten by the next character arriving in the RDR. **In FIFO mode:** '0' = No overrun error has been detected, or the OE bit was cleared because the CPU read the content of the line status register (LSR). '1' = Overrun error has been detected. If data continues to fill the FIFO beyond the trigger level, an overrun error occurs only after the FIFO is full and the next character has been completely received in the shift register. An overrun error is indicated to the CPU as soon as it happens. The new character overwrites the character in the shift register, but it is not transferred to the FIFO.
2	R	**Parity Error (PE) indicator.** **In non-FIFO mode:** '0' = No parity error has been detected, or the PE bit was cleared because the CPU read the erroneous data from the received data register (RDR). '1' = A parity error has been detected with the character in the received data register (RDR). **In FIFO mode:** '0' = No parity error has been detected, or the PE bit was cleared because the CPU read the erroneous data from the receiver FIFO and the next character to be read from the FIFO has no parity error. '1' = A parity error has been detected with the character at the top of the receiver FIFO.

Table 2.10: Line Status Register $(LSR)_2$

Bit	Access	Description
3	R	**Framing Error (FE) indicator.** **In non-FIFO mode:** '0' = No framing error has been detected, or the FE bit was cleared because the CPU read the erroneous data from the received data register (RDR). '1' = A framing error has been detected with the character in the received data register (RDR). **In FIFO mode:** '0' = No framing error has been detected, or the FE bit was cleared because the CPU read the erroneous data from the receiver FIFO and the next character to be read from the FIFO has no framing error. '1' = A framing error has been detected with the character at the top of the receiver FIFO.
4	R	**Ignored.**
5	R	**Transmit Data Register (TDR) Empty.** **In non-FIFO mode:** '0' = Transmit data register (TDR) is not empty. TDR has been loaded by the CPU. '1' = Transmit data register (TDR) is empty (ready to accept a new character). The content of TDR has been transferred to the transmitter shift register (TSR). **In FIFO mode:** '0' = Transmitter FIFO is not empty. At least one character has been written to the transmitter FIFO. The transmitter FIFO may be written to if it is not full. '1' = Transmitter FIFO is empty. The last character in the FIFO has been transferred to the transmitter shift register (TSR).
6	R	**Transmitter Shift Register Empty indicator.** '0' = Transmitter Shift Register (TSR) contains a data character. '1' = Transmitter Shift register (TSR) is empty.
7	R	**Receiver FIFO error.** **In non-FIFO mode:** '0' = There has been no error, or RXFIFOE was cleared because the CPU read the erroneous character from the received data register (RDR). '1' = There is a parity error, framing error, or break indicator in the received data register (RDR). **In FIFO mode:** '0' = There has been no error, or RXFIFOE was cleared because the CPU read the erroneous character from the receiver FIFO and there are no more errors in the receiver FIFO. '1' = At least one parity error, framing error, or break indicator in the receiver FIFO.

10. **Modem Status Register (MSR):**
The register displays the current state of the modem control lines. Also, four bits also provide an indication in the state of one of the modem status lines. These bits are set to '1' when a change in the corresponding line has been detected and they are reset when the register is being read.

Table 2.11: Modem Status Register (MSR)

Bit	Access	Description
0	R	**Delta Clear To Send (DCTS) indicator.** '1'- The CTS line has changed its state.
1	R	**Delta Data Set Ready (DDSR) indicator.** '1'- The DSR line has changed its state.
2	R	Trailing Edge of Ring Indicator (TERI) detector. The RI line has changed its state from low to high state.
3	R	**Delta Data Carrier Detect (DDCD) indicator.** '1'- The DCD line has changed its state.
4	R	Complement of the CTS input or equals to RTS in loopback mode.
5	R	Complement of the DSR input or equals to DTR in loopback mode.
6	R	Complement of the RI input or equals to Out1 in loopback mode.
7	R	Complement of the DCD input or equals to Out2 in loopback mode.

Where CTS – Clear to send.
DSR – Data set ready.
RI – Ring Indicator.
DCD – Data carrier detect.

11. **Scratch Pad Register (SPR):**
The scratch pad register (SCR) is intended for the programmer's use as a scratch pad. It temporarily holds the programmer's data without affecting UART operation.

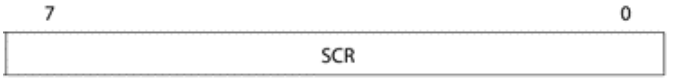

Table 2.12: Scratch Pad Register (SPR)

Bit	Access	Description
0-7	RW	These bits are intended for the programmer's use as a scratch pad in the sense that it temporarily holds the programmer's data without affecting any other UART operation.

2.2.2 Detailed Explanation of UART Components

1. **Transmitter Shift Register:** The Transmitter Shift Register is an essential component of the UART that handles the serial transmission of data. When data is written to the

Transmit Data Register (TDR) or Transmitter buffer, it is moved to the Transmitter Shift Register. This register then serializes the data, shifting it out bit by bit to the TX (transmit) pin. The process is synchronized with the clock signal generated by the Baud Rate Generator, ensuring that the data bits are transmitted at the correct speed. The Transmitter Shift Register's efficient serialization is crucial for maintaining the integrity and timing of the data being sent out.

2. **Receiver Shift Register:** The Receiver Shift Register works in the opposite manner to the Transmitter Shift Register, handling the reception of serial data. Incoming data on the RX (receive) pin is shifted bit by bit into the Receiver Shift Register. Once a complete data frame is received, the data is moved to the Received Data Register (RDR) or Receiver buffer, where it can be accessed by the system. This ensures that the incoming serial data is correctly synchronized and captured according to the timing dictated by the Baud Rate Generator, enabling accurate data reception.

3. **Transmitter Buffer:** The Transmitter Buffer, often implemented as a FIFO (First-In-First-Out) buffer, serves as a temporary storage for data awaiting transmission. This buffer allows the UART to handle data at different rates, storing multiple bytes of data before they are transferred to the Transmitter Shift Register for serial output. The presence of a buffer ensures smooth and continuous data transmission, preventing data loss and handling bursts of data effectively, even if the system's data generation rate is higher than the transmission rate.

4. **Receiver Buffer:** The Receiver Buffer, typically also a FIFO buffer, temporarily stores incoming data bytes received from the RX pin. This buffering allows the UART to manage incoming data efficiently, ensuring that no data is lost even if the system cannot immediately process the received data. By holding multiple bytes of incoming data, the Receiver Buffer helps to prevent overflow and data loss, providing a steady flow of data to the system for processing.

5. **Baud Rate Generator:** The Baud Rate Generator is a critical component that determines the timing of data transmission and reception by generating the appropriate clock signals. It divides the input clock (PCLK) by a divisor value set in the Divisor Latch Register (DLR) to produce the desired baud rate—the speed at which data bits are transmitted and received. Properly setting the baud rate ensures synchronization between the transmitter and receiver, allowing for reliable and accurate serial communication. The Baud Rate Generator's role in defining the communication speed is fundamental to maintaining data integrity and ensuring effective communication between devices.

6. **TX Line (Transmit Line):** The TX line is a critical part of the UART responsible for sending data from the UART to another device. The TX line carries the serialized data from the UART to the receiving device. Data to be transmitted is first loaded into the Transmit Data Register (TDR) or Transmitter buffer parallely, then shifted out bit by bit by the Transmitter Shift Register serially.

7. **RX Line (Receive Line):** The RX line is the counterpart to the TX line, responsible for receiving data sent from another device to the UART. The RX line receives serialized data from the transmitting device. The data is shifted in bit by bit by the Receiver Shift Register serially then moved to Receiver Data Register (RDR) or Receiver buffer parallely.

Chapter 3

Implementation details

3.1 Final system architecture

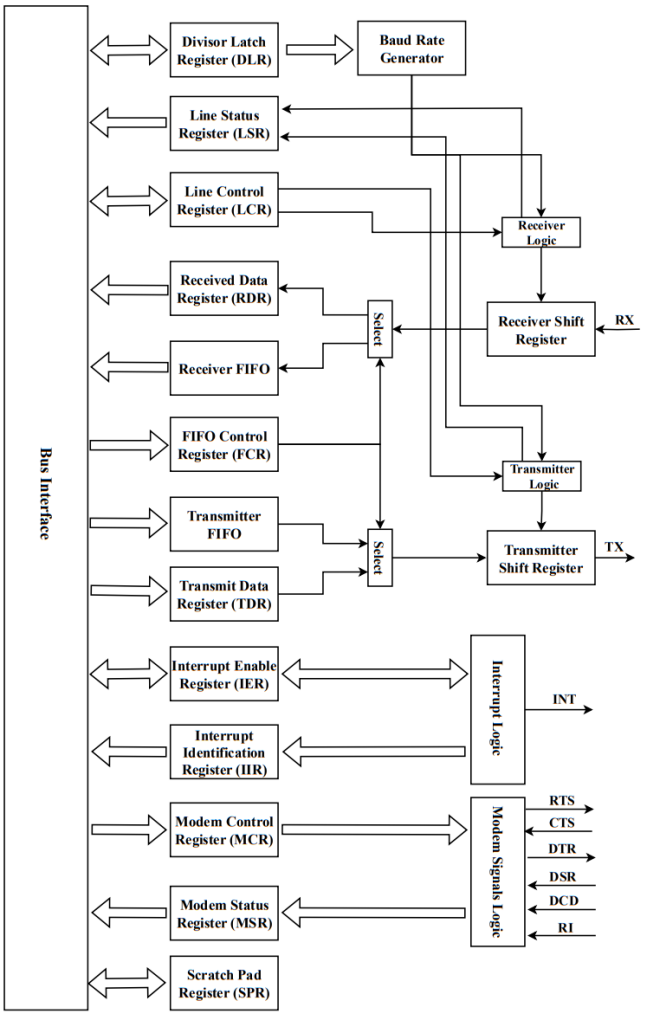

Figure 3.1: Architecture of UART

The diagram illustrates the architecture of a UART module, detailing various registers and their interconnections with the transmit and receive logic.The architecture allows the UART to efficiently handle serial communication by managing data flow, status reporting, error detection, and control signaling through a structured set of registers and logic units.

3.2 Algorithm

1. **Set the Baud rate:** This can be done by configuring the DLR register with appropriate DLR value.The baud rate generator uses this divisor to create the clock rate for UART communication.
2. **Configure Line Control Register (LCR):** Set the desired frame format by configuring word length, stop bits, and parity bits.
3. **Enable FIFOs (Optional):** If using FIFOs, configure the FIFO Control Register (FCR) to enable.
4. **Enable Interrupts (Optional):** Configure the Interrupt Enable Register (IER) to enable desired UART interrupts for efficient handling of data transfer events.
5. **Data Transmission:**
   - **Check Line Status:** Before transmitting data, check the Line Status Register (LSR) to ensure that the Transmitter Holding Register (THR) or Transmitter FIFO is empty and ready to accept new data.
   - **Load Data:** Write the data to be transmitted into the Transmit Data Register (TDR) or Transmitter FIFO.
   - **Shift Data Out:** The data from the TDR or Transmitter FIFO is moved to the Transmitter Shift Register, which converts parallel data into serial form and shifts it out through the TX pin.
   - **Monitor Status:** Continuously monitor the LSR to detect any transmission errors or to know when the transmitter is ready for more data.
6. **Data Reception.**
   - **Check Line Status:** Monitor the LSR to check if there is new data available in the Receive Data Register (RDR) or Receiver FIFO.
   - **Read Data:** When new data is available, write it into the RDR or Receiver FIFO.
   - **Store Data:** Store the received data into the desired buffer or process it as needed.
   - **Handle Errors:** Check the LSR for any reception errors (e.g., framing error, parity error) and handle them accordingly.

3.3 Flowchart

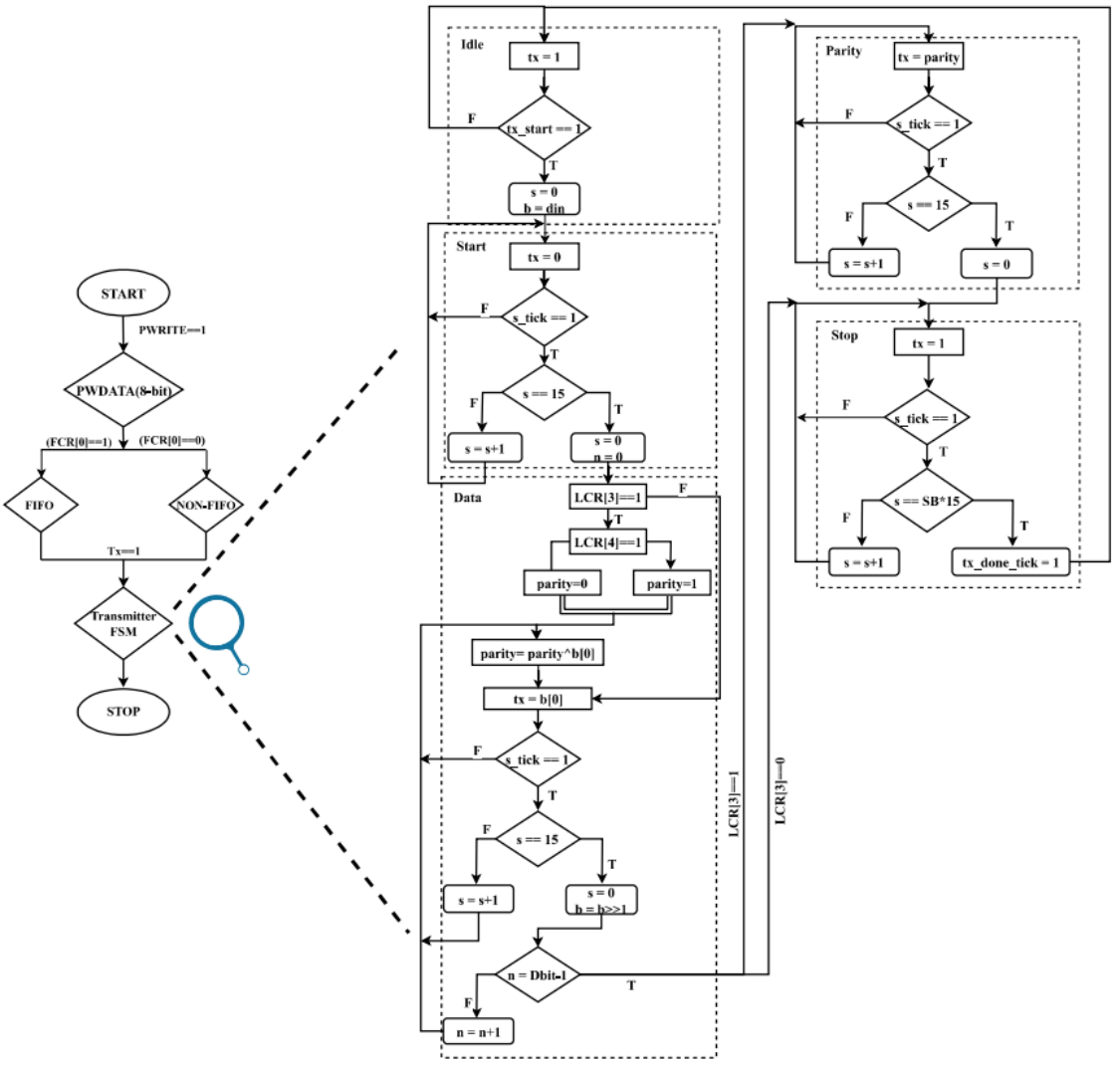

Figure 3.2: Flowchart of Transmitter

The flow chart in Figure 3.2 represents the state machine for a UART transmitter. Initially, it checks the PWRITE control signal. When enabled, the processor writes the PWDATA. Next, it determines whether the UART is operating in FIFO or NON-FIFO mode. It then checks the tx line. If it is enabled, indicating the idle state, the transmitter FSM is executed.

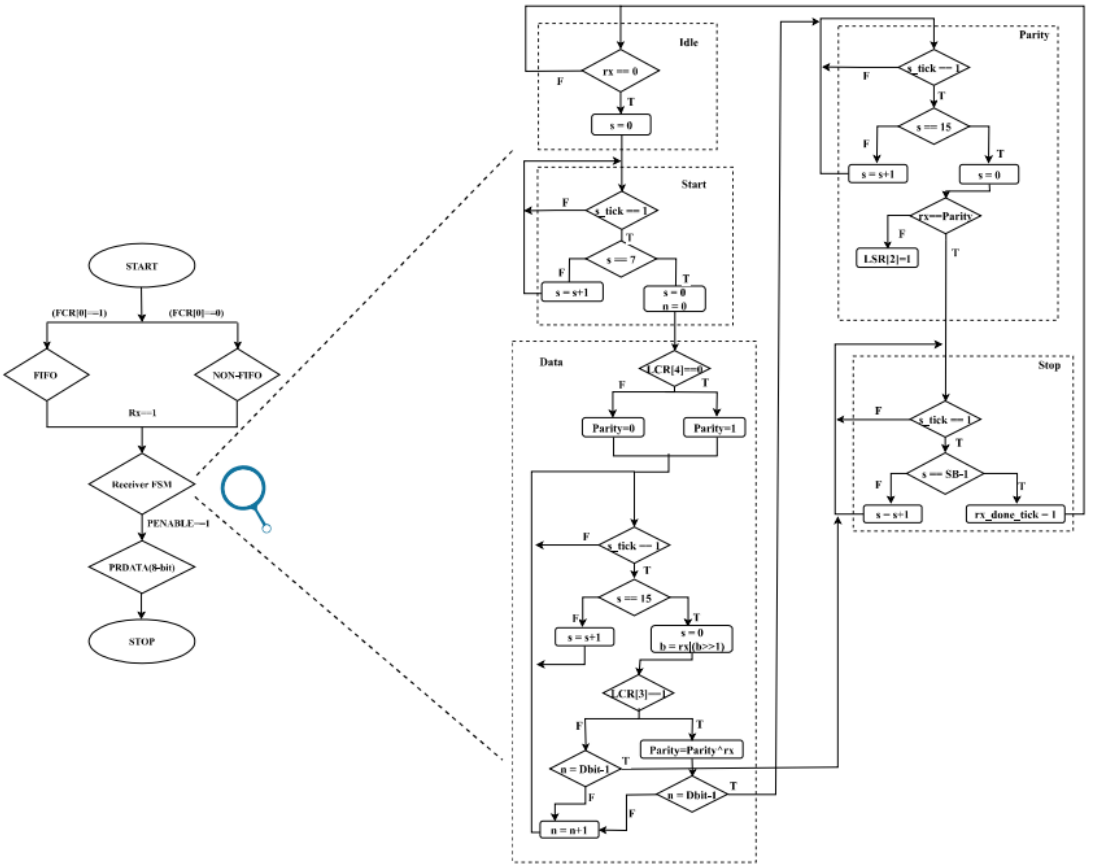

Figure 3.3: Flowchart of Receiver

In the receiver flowchart, we first determine whether the UART is operating in FIFO or NON-FIFO mode. Next, we check the RX line, and if it is enabled, the receiver FSM is activated. When the PENABLE control signal is enabled, PRDATA is read by the processor.

Chapter 4

Results and discussions

4.1 Result Analysis

4.1.1 Simulation Results

Transmitter

Figure 4.1: Transmitter simulation result

The simulation in Figure 4.1 demonstrates the transmission of data over the TX line and also status registers were reflected accordingly.

Receiver

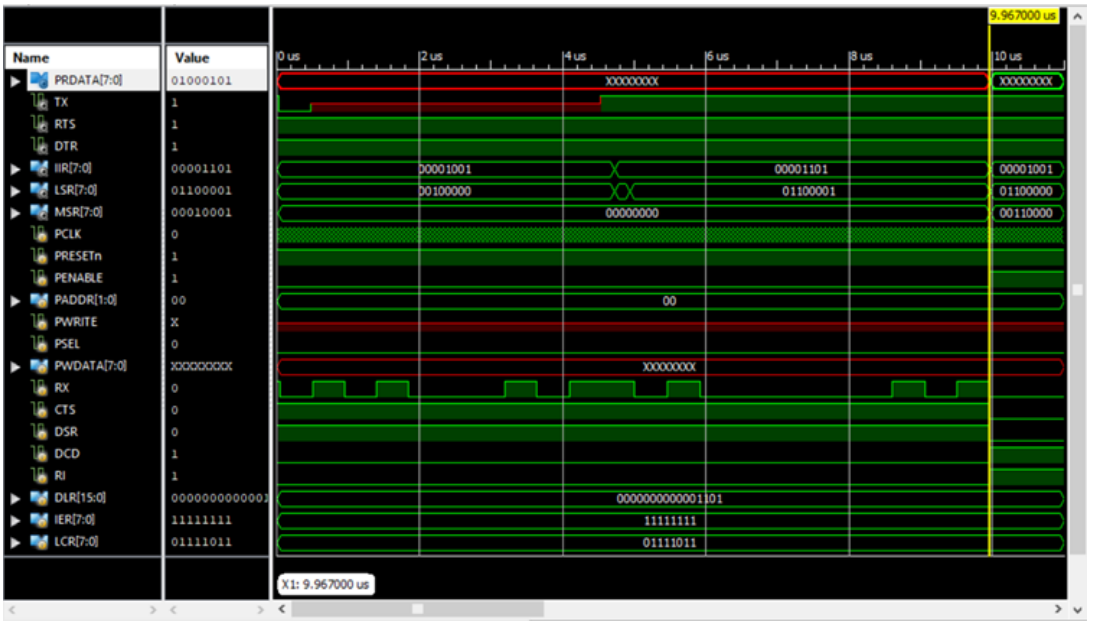

Figure 4.2: Receiver simulation result

Figure 4.2 illustrates the reception of data over the RX line, which is subsequently transferred to the PRDATA when PENABLE is high. Additionally, the status registers are updated to reflect the current state of the system.

Loopback mode

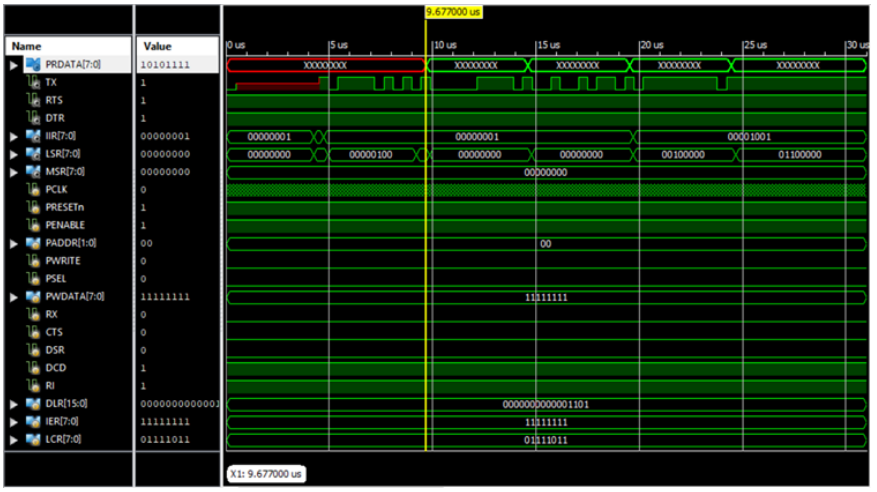

Figure 4.3: Loopback mode simulation result 1

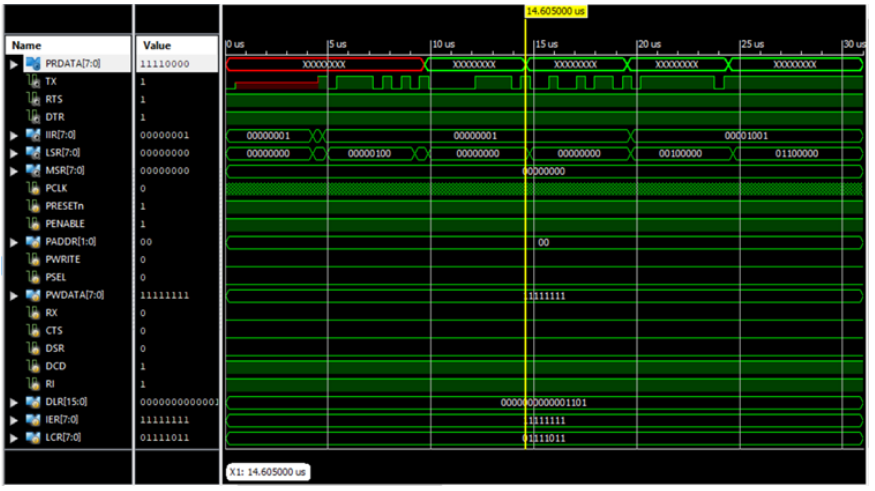

Figure 4.4: Loopback mode simulation result 2

When the 4th bit of MCR is '1', Uart works in loopback mode as shown in Figure 4.3 and 4.4. The RX pin receives the serial data from TX pin directly and then gets stored in RSR, which is then further read from PRDATA.

4.1.2 Implementation Results

The design has been successfully implemented on the Arty 7 FPGA, utilizing the Virtual Input/Output (VIO) core.

4.1.3 Using single FPGA

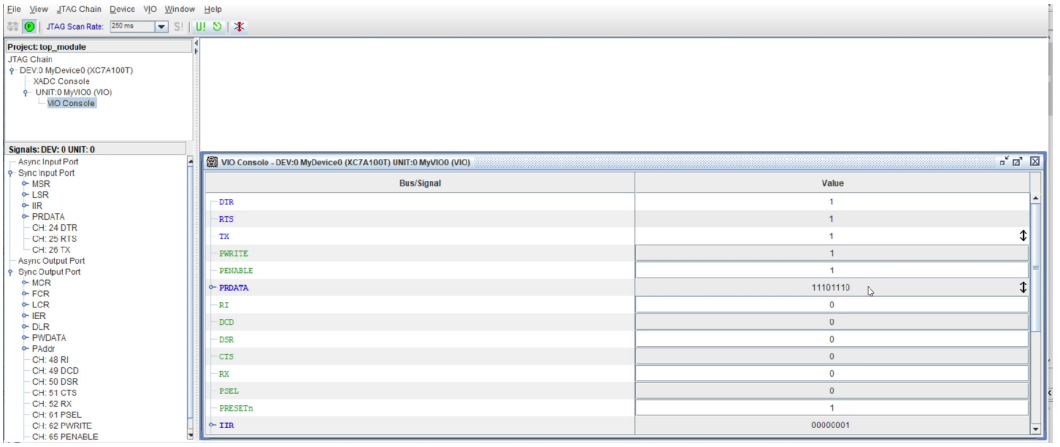

Figure 4.5: Implementation result of single FPGA

We have successfully tested the design in loopback mode on the same FPGA, as illustrated in the provided Figure 4.5.

4.1.4 Using two FPGA

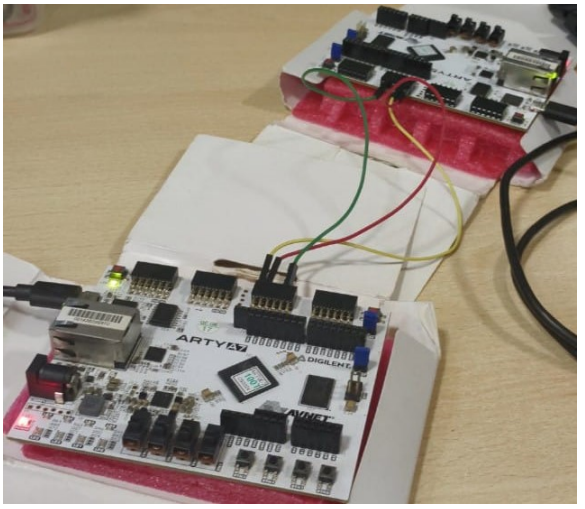

Figure 4.6: View of 2 Arty boards connected

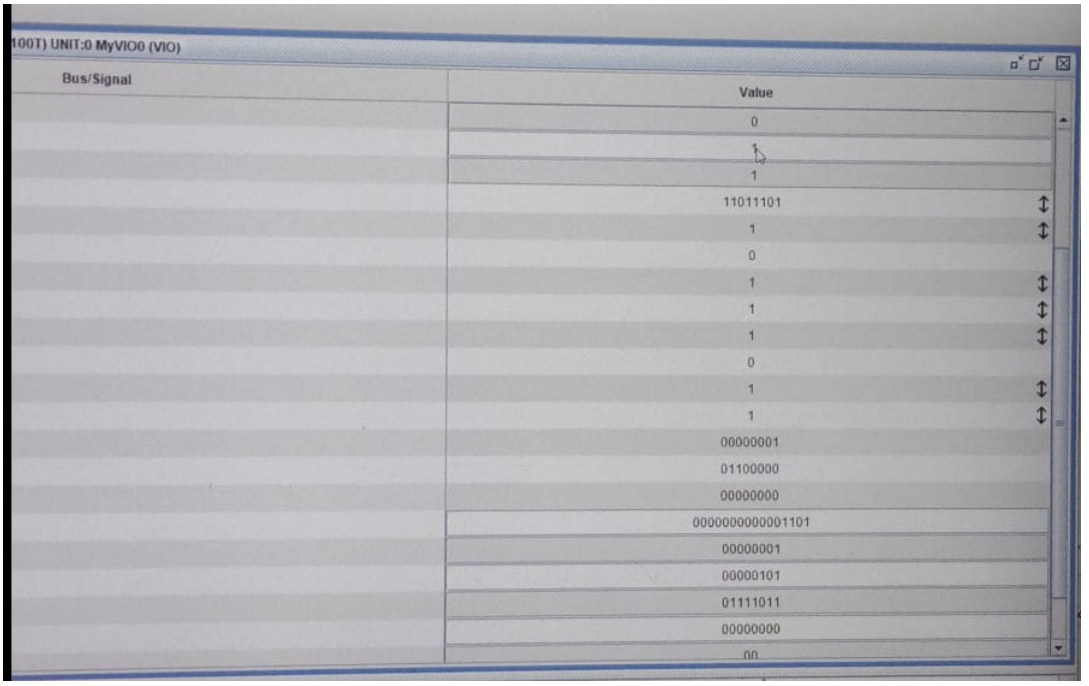

Figure 4.7: Implementation result of two FPGA

To test the functionality of the design, two Arty boards were connected to two different PCs. We successfully captured the **PWDATA** sent from one PC and observed the same data on the other PC in the **VIO** console via **PRDATA** when the respective control signal was enabled.

Chapter 5

Conclusions and future scope

5.1 Conclusion

The implementation of a Universal Asynchronous Receiver/Transmitter (UART) in this project successfully showcases the capability for serial communication between devices. By designing both the transmitter and receiver state machines, the project ensures reliable data transmission and reception, with appropriate handling of the respective control signals. The inclusion of both FIFO and NON-FIFO modes adds flexibility in data handling, accommodating various application requirements. Overall, this project demonstrates the effectiveness of UART in enabling efficient and robust serial communication.

5.2 Future scope

By incorporating various below future enhancements, the UART project can evolve to meet the increasing demands of modern communication systems, delivering robust and efficient solutions for a wide range of applications.

- **Enhanced Error Detection and Correction:** Implement advanced error detection mechanisms like CRC (Cyclic Redundancy Check) and error correction algorithms to further improve data integrity.
- **Higher Baud Rates:** Explore the possibility of increasing the baud rate to support faster data transmission, optimizing the UART for high-speed applications.
- **DMA Integration:** Integrate Direct Memory Access (DMA) support to offload data transfer tasks from the CPU, improving overall system performance.

Bibliography

[1] Lattice Semiconductor Corporation. *UART 16550 IP Core Documentation*, 2024. Accessed: 2024-06-19.

[2] Texas Instruments. *TMS320C6000 Chip Support Library API Reference Guide*, 2009. Accessed: 2024-06-19.