# Implementation of UART Transceiver on FPGA

Project Report

Submitted in partial fulfillment of the requirement for the award of degree of
BACHELOR OF TECHNOLOGY

by

**Achyut R. Shegade - 191EE101**
**Banala Rahul Royal - 191EE1208**
**Madhur Sharma - 191EE132**
**Govardhan L.R. - 191EE215**

Under the guidance of
**Dr. K. Panduranga Vittal**
Professor
Department of Electrical and Electronics Engineering


**DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING**
**NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA, SURATHKAL**
**SRINIVASNAGAR, MANGALORE - 575025**
**KARNATAKA, INDIA**

**April 2023**

# Declaration

We hereby declare that the project work entitled " Implementation of UART Transceiver on FPGA" submitted to the National Institute of Technology- Karnataka, is a record of an original work done by us under the guidance of Dr. K. Panduranga Vittal, Professor Dept. of Electrical and Electronics Engineering. This project is submitted in the partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Electrical and Electronics Engineering. The results embodied in this thesis have not been submitted to any other University or Institute for the award of any degree or diploma.

# Certificate

This is to certify that the Major Project Report entitled " Implementation of UART Transceiver on FPGA" is submitted by

**Achyut R. Shegade (191EE101)**

**Banala Rahul Royal (191EE208)**

**Madhur Sharma (191EE132)**

**Govardhan L.R. (191EE215)**

as a record of the work done by them is accepted as the Major Project submission in the partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Electrical and Electronics Engineering, National Institute of Technology Karnataka, Surathkal.

**Project Guide**

# Abstract

Presently FPGAs are coming very strongly in the digital hardware systems as it provides the opportunity for reconfiguration powered with good clock speed and design resources. The usage of FPGA systems in real time domain is also a very fruitful proposition as the FPGA devices are coming with processing cores for Real Time data processing. In a complex system scenario involving a large amount of processing tasks, there is a requirement of building the system using multiple FPGA devices. To make this possible we have to establish a real time data communication between the FPGA devices or between the user and the FPGA.

Universal Asynchronous Receiver and Transmitter (UART) protocol is a kind of serial communication protocol mostly used for short-distance, low speed, low-cost data exchange between computer and its peripherals. This report explores the communication between FPGA and PC, both wired and wirelessly and also between two or more FPGAs using the UART protocol.

# Contents

# List of Figures

# Chapter 1

# Introduction

## 1.1 Background

Digital communication protocols, in general, are standardized methods for exchanging data between devices. They are essential in modern electronics as they enable communication between devices that would otherwise be incompatible. There are many types of digital communication protocols, including UART, SPI, I2C, CAN, Ethernet, and USB, among others. Each protocol has its unique characteristics, advantages, and limitations, and it is essential to select the appropriate protocol for a specific application.

One of the most widely used serial communication protocols is the Universal Asynchronous Receiver-Transmitter (UART), which allows data to be transferred between devices in a serial fashion. The UART protocol is simple and versatile, making it a popular choice for a wide range of applications. But the processing power of conventional microcontrollers is constrained, which can lead to slow data transfer rates.

On the other hand, FPGAs are hardware components that can be configured and run in high parallel to carry out specific tasks. We can take advantage of the parallel processing abilities of the FPGA by implementing a UART controller on it to achieve high-speed data transfer between the microcontroller and other peripheral devices.

In this project, we will use Verilog HDL to design and implement a UART controller on an FPGA board. We will use the Xilinx Vivado development environment to simulate the UART design and generate the corresponding configuration file for programming the FPGA. We will also interface the FPGA with an external device such as a computer or a microcontroller to test the functionality of the UART controller. We will also implement a controller to communicate between two or more FPGAs.

## 1.2 Research Objectives

By completing this project, we aim to gain a deeper understanding of digital communication protocols and FPGA design. We will also develop practical skills in Verilog HDL programming and FPGA implementation, which are highly relevant in the field of embedded systems design.

## 1.3 Outline of the Report

This report is divided into five chapters, each focusing on a different aspect of the project implementation.

Chapter 1 introduces the project and outlines its objectives and scope.

Chapter 2 presents a literature survey on the UART protocol and its implementation on FPGAs. It also reviews related works in the field and discusses the design considerations and challenges of implementing a UART controller on an FPGA.

Chapter 3 describes the project implementation in detail, including the design and implementation of the UART controller on an FPGA using Verilog HDL. It also covers the simulation and verification process, the generation of the configuration file for programming the FPGA, and the hardware implementation of the design on the FPGA board.

Chapter 4 presents the evaluation and testing of the UART controller, including the testing and verification of its functionality. Chapter 5 concludes the report, summarizing the project objectives and achievements, discussing the implications and contributions of the project, outlining future work and potential applications of the design, and presenting final remarks. Finally, appendices are included at the end of the report, which contain additional details on the Verilog HDL code for the UART controller design.

# Chapter 2

# Literature Survey

## 2.1 Introduction

The Universal Asynchronous Receiver-Transmitter (UART) is a widely used serial communication protocol in embedded systems. Its simplicity and versatility have made it a popular choice for many applications. With the increasing demand for high-speed data transfer and low power consumption in embedded systems, the implementation of UART controllers on Field Programmable Gate Arrays (FPGAs) has become an active research area. In this literature survey, we will review the relevant works on the implementation of UART controllers on FPGAs.

## 2.2 Related Work

One of the early works on this topic was by Huang and Lin, who proposed a UART controller design using Verilog HDL and implemented it on an FPGA. Their design achieved a maximum baud rate of 115200 bps and consumed less power than traditional UART controllers implemented on microcontrollers.

In another study, Hasan and Islam presented a low-power UART design that utilized a novel data encoding technique to reduce the number of transitions between high and low states, thus minimizing power consumption. Their implementation on an FPGA achieved a maximum baud rate of 38400 bps with a power consumption of only 11.5 mW.

Furthermore, several studies have explored the use of advanced techniques such as parallel processing, pipelining, and optimization algorithms to enhance the performance of UART controllers on FPGAs. For instance, Wu and Liu developed a pipelined architecture that allowed for parallel processing of data, resulting in improved throughput and reduced latency.

In summary, the reviewed works have demonstrated the feasibility and benefits of implementing UART controllers on FPGAs. The proposed designs have achieved high data transfer rates, low power consumption, and improved system performance. These works can serve as a valuable reference for the design and implementation of UART controllers on FPGAs and have demonstrated the potential of FPGAs in enhancing the performance and efficiency of UART communication in embedded systems.

## 2.3 Background Work

Communication between electronic devices are facilitated by communication protocols. These protocols have a set of rules and format that allows two or more entities to transmit or receive information from one another.
The three most common protocols used for communications in electronic systems are:
1. Serial Peripheral Interface (SPI)
2. Universal Asynchronous Receiver Transmitter (UART)
3. Inter-Integrated Circuit (I2C)

### Serial Peripheral Interface (SPI) protocol

As the name suggests, SPI protocol is a serial communication protocol and one unique benefit of SPI is the fact that data can be transferred without interruption. With I2C and UART, data is sent in packets, limited to a specific number of bits. Start and stop conditions define the beginning and end of each packet, so the data is interrupted during transmission.
Devices communicating via SPI are in a master-slave relationship. The simplest configuration of SPI is a single master, single slave system, but one master can control more than one slave as well.

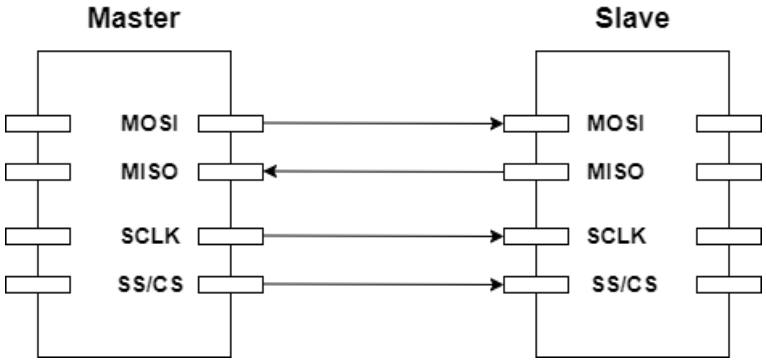


Figure 2.1: SPI master-slave configuration

| | |
|---|---|
| Wires used | 4 |
| Maximum speed | Upto 10 Mbps |
| Synchronous or Asynchronous? | Synchronous |
| Serial or Parallel? | Serial |
| Max number of Masters | 1 |
| Max number of Slaves | Theoretically unlimited |

Table 2.1: SPI features

Advantages of using SPI protocol are that there are no start and stop bits, so the data can be streamed continuously without interruption and it has a separate MISO and MOSI lines, so data can be sent and received at the same time.
Some disadvantages of SPI can be that it uses four wires (I2C and UARTs use two), there is no acknowledgement that the data has been successfully received (as in I2C), no form of error checking (like the parity bit in UART) and it only allows for a single master.

### Universal Asynchronous Receiver Transmitter (UART) protocol

In UART communication, two UARTs communicate directly with each other. The transmitting UART converts parallel data from a controlling device like a CPU into serial form, transmits it in serial to the receiving UART, which then converts the serial data back into parallel data for the receiving device. Only two wires are needed to transmit data between two UARTs. Data flows from the Tx pin of the transmitting UART to the Rx pin of the receiving UART.

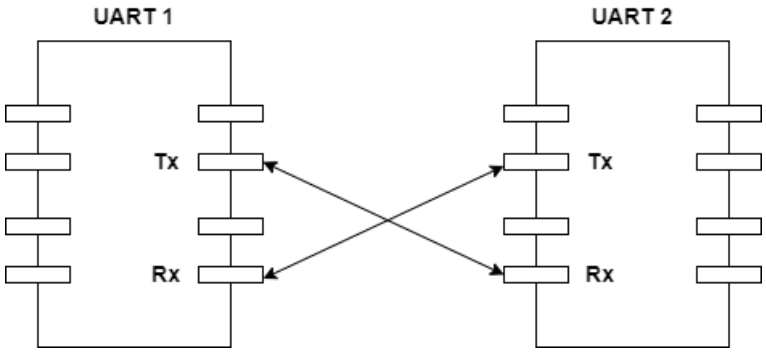


Figure 2.2: UARTconfiguration

| Wires used | 2 |
|---|---|
| Maximum speed | Any speed upto 115200 baud |
| Synchronous or Asynchronous? | Asynchronous |
| Serial or Parallel? | Serial |
| Max number of Masters | 1 |
| Max number of Slaves | 1 |

Table 2.2: UART features

Advantages of using UART protocol are that it uses just 2 wires for communication, no clock signal is necessary, it has a parity bit to allow for error checking and he structure of the data packet can be changed as long as both sides are set up for it.
Some disadvantages of UART can be that the size of the data frame is limited to a maximum of 9 bits, it doesn't support multiple slave or multiple master systems and the baud rates of each UART must be within 10% of each other.

### Inter-Integrated Circuit (I2C) protocol

I2C combines the best features of SPI and UARTs. With I2C, you can connect multiple slaves to a single master (like SPI) and you can have multiple masters controlling single, or multiple slaves.

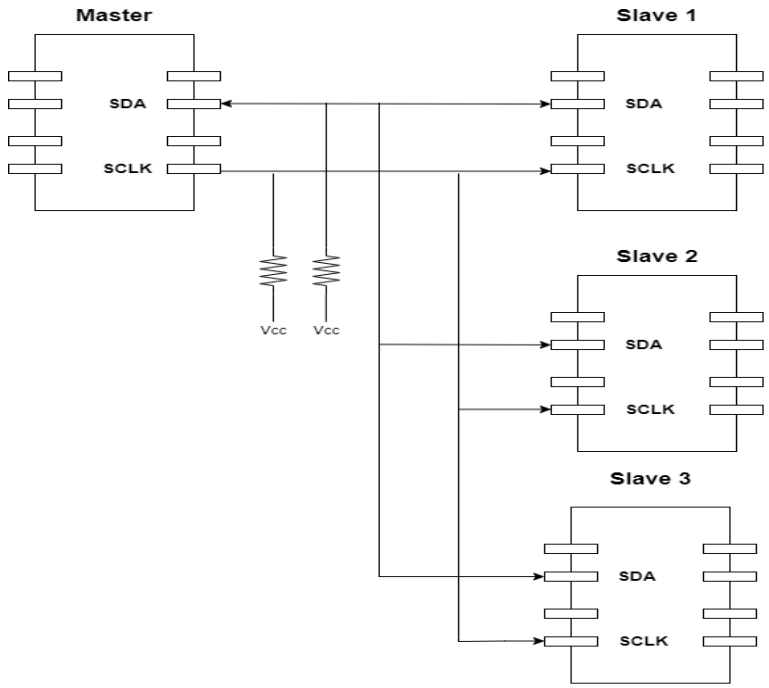


Figure 2.3: I2C configuration

| Wires used | 2 |
|---|---|
| Maximum speed | Standard Mode = 100 kbps<br>Fast Mode = 400 kbps<br>High Speed Mode = 3.4 Mbps<br>Ultra Fast Mode = 5 Mbps |
| Synchronous or Asynchronous? | Synchronous |
| Serial or Parallel? | Serial |
| Max number of Masters | Unlimited |
| Max number of Slaves | Unlimited |

Table 2.3: I2C features

Advantages of using I2C protocol are that it uses only uses two wires, supports multiple masters and multiple slaves, ACK/NACK bit gives confirmation that each frame is transferred successfully and hardware used is less complicated than UARTs.
Some disadvantages of I2C are that it has a slower data transfer rate than SPI, size of the data frame is limited to 8 bits and hardware needed is more complicated to implement than SPI

## 2.4 Data transmission in UART

In UART, the mode of transmission is in the form of a packet. The piece that connects the transmitter and receiver includes the creation of serial packets and controls those physical hardware lines. A packet consists of a start bit, data frame, a parity bit, and stop bits.

| Start bit<br>(1bit) | Data frame<br>(5-9 data bits) | Parity bits<br>(0-1 bit) | Stop bits<br>(1-2 bits) |
|---|---|---|---|

Table 2.4: UART data packet format

**Start bit**

The UART data transmission line is normally held at a high voltage level when it's not transmitting data. To start the transfer of data, the transmitting UART pulls the transmission line from high to low for one clock cycle. When the receiving UART detects the high to low voltage transition, it begins reading the bits in the data frame at the frequency of the baud rate.

**Data frame**

The data frame contains the actual data being transferred. It can be five bits up to eight bits long if a parity bit is used. If no parity bit is used, the data frame can be nine bits

long. In most cases, the data is sent with the least significant bit first.

**Parity**

Parity bit is used for error checking in the data that is being transferred. After the receiving UART reads the data frame, it counts the number of bits with a value of 1 and checks if the total is an even or odd number.
Parity bit of 0 suggests that the number of 1's in the data frame must be an even number while the parity bit of 1 suggests that the number f 1's in the data frame must be an odd number.
When the parity bit matches the data, the UART knows that the transmission was free of errors. But if the parity bit is a 0 and the total is odd (or0 the parity bit is a 1 and the total is even, then the UART knows that bits in the data frame have changed.

**Stop bit**

It is used to signal the end of the data packet, the sending UART drives the data transmission line from a low voltage to a high voltage for one to two bit duration.

## 2.5 Steps of UART transmission

1. The transmitting UART receives data in parallel from its data bus.
2. The transmitting UART adds the start bit, parity bit, and the stop bits to the data frame.
3. The entire packet is sent serially starting from start bit to stop bit from the transmitting UART to the receiving UART. The receiving UART samples the data line at the preconfigured baud rate.
4. The receiving UART discards the start bit, parity bit, and stop bit from the data packet received.
5. The receiving UART converts the serial data back into parallel and transfers it to its receiving data bus.

# Chapter 3

# Project Implementation

## 3.1 Implementation of UART

The UART consists of three important modules inside it:
1. Transmission module
2. Receiver module
3. Button Debounce module

These modules are controlled by a top level UART controller module.

### 3.1.1 UART transmission module

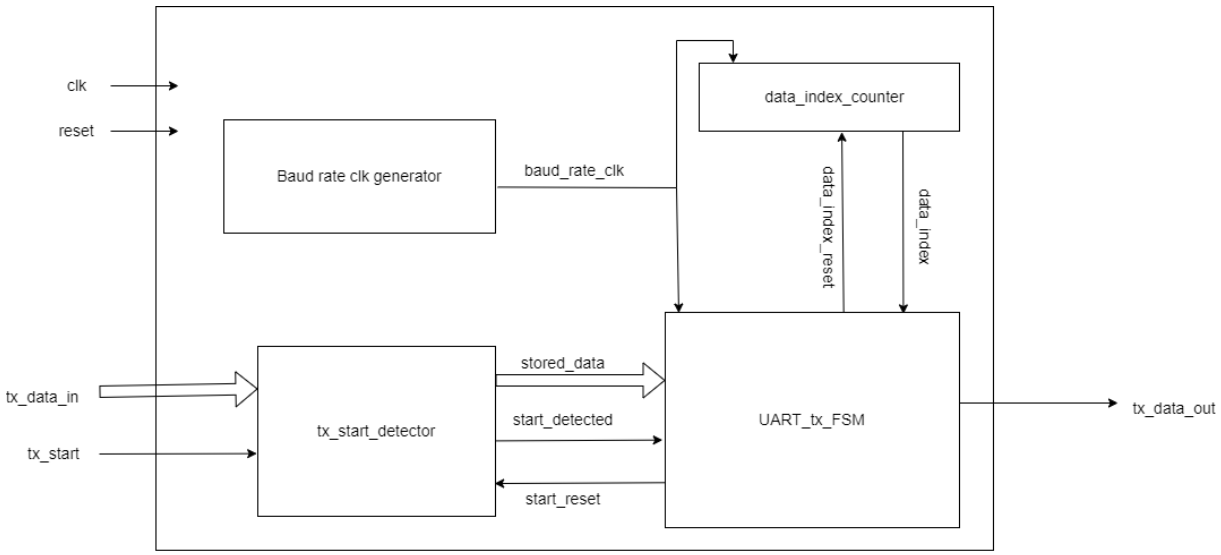


Figure 3.1: UART transmission module

The transmission module of UART contains four basic blocks for baud clock generation, detecting the start of transmission, counter to track the number of bits transmitted and a FSM to control the entire transmission process.

### Baud rate clock generator

The baud rate is the rate at which information is transferred in a communication channel. This module generates that baud rate by dividing the existing clock input.
There are several standard baud rates that are most commonly used for the UART transmission. Example: 9600, 14400, 19200, 38400, 57600, 115200, 128000 and 256000 bits per second.

### Transmission start detector

The UART data transmission line is normally held at a high voltage level when it's not transmitting data and it must be pulled low for atleast one clock cycle to start the data transmission.
The data packet in UART sets the first bit(start bit) to zero. This module detects this start bit and sends the signal to the FSM module to start its operation of transmitting data.

### Data index counter

This module keeps track of the number of bits that have been transmitted. The number of bits in the data packet is fixed depending on the chosen format.

### UART transmission FSM

The FSM module controls the overall data transmission based on the baud rate and start detected signals received from the other modules.
The transmission FSM typically has 4 states: IDLE, START, DATA and STOP.
The IDLE state is the default starting state where the module waits for the transmission request. In this state, the output data is set high and the data index counter is ket on hold.
FSM moves to the START state when it receives the transmission request. This states enables the data index counter and resets the data output line.
Next is the DATA state in which the transmission module continues the transmission of data. Here, the data index counters keep incrementing and data appears on the output data. This happens till the 8 data bits have been transmitted.
The final state in the FSM is the STOP state where the present data packet transmission is completed, the output data is set high again and the module moves back to IDLE state awaiting the next transmission request.

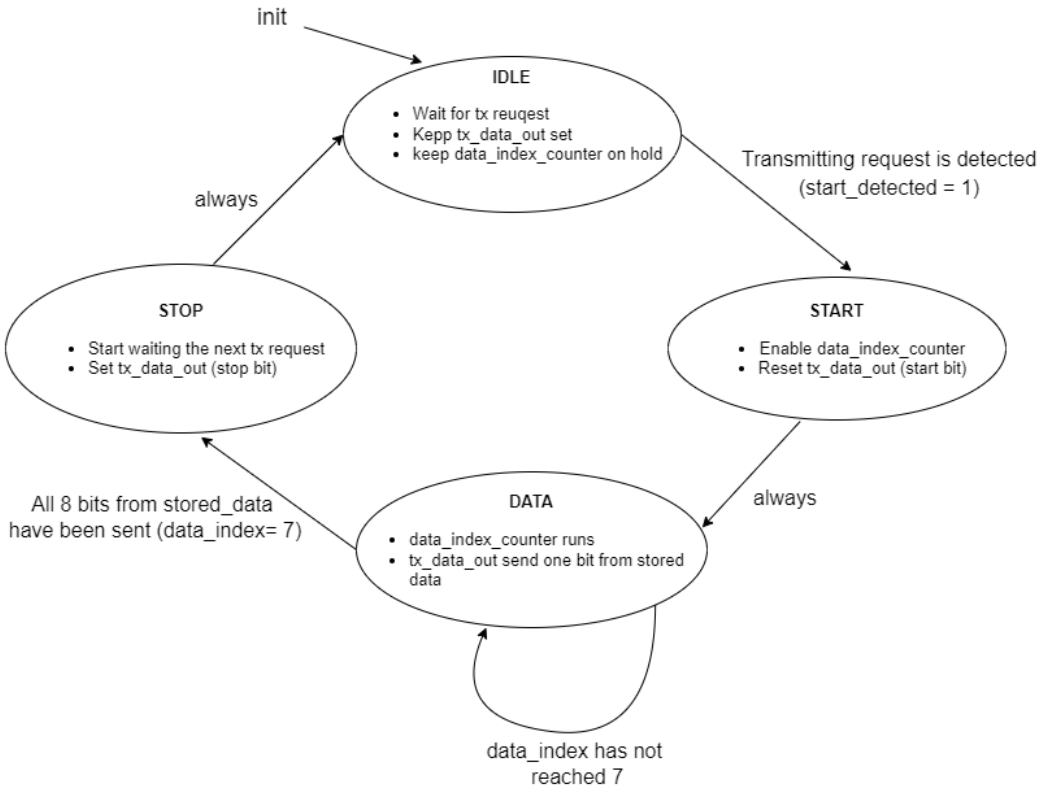


Figure 3.2: State Diagram of FSM of UART transmission module

### 3.1.2 UART receiver module

The receiver module of UART contains the basic blocks for baud clock(x16) generation, counter to track the number of bits received and a FSM to control the entire receiving process.

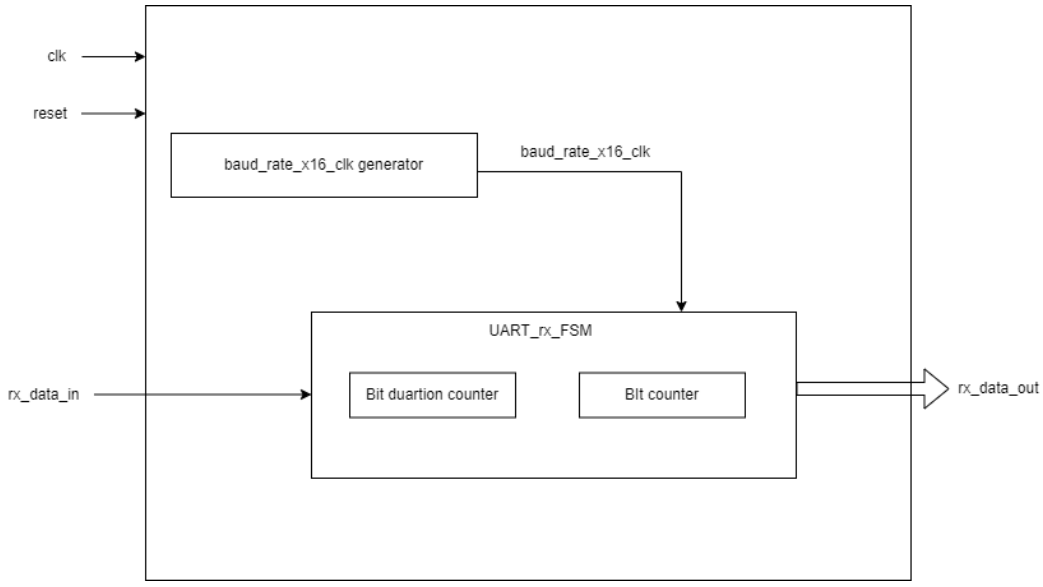


Figure 3.3: UART receiver module

### Baud clock(x16) generator

This module generates the baud clock of frequency 16 times greater than of the baud rate present in the transmitter module. This is done to sample each bit of the receiving data 16 times and the centre value i.e. 8th sample is taken as the correct reading. This is done to avoid reading any wrong values caused because of circuit glitches or any other external disturbances. Reading the exact 8th sample is facilitated by the bit duration counter which keeps track of the number of samples read of an individual bit and the value corresponding to 8th sample is stored as the received data.

### Bit counter

This counter keeps count of the total number of bits received. Once the format of the data packet is set, the counter increments itself to that value and sends a signal to the FSM module to acknowledge that the entire data packet has been received.

### FSM of UART transmission module

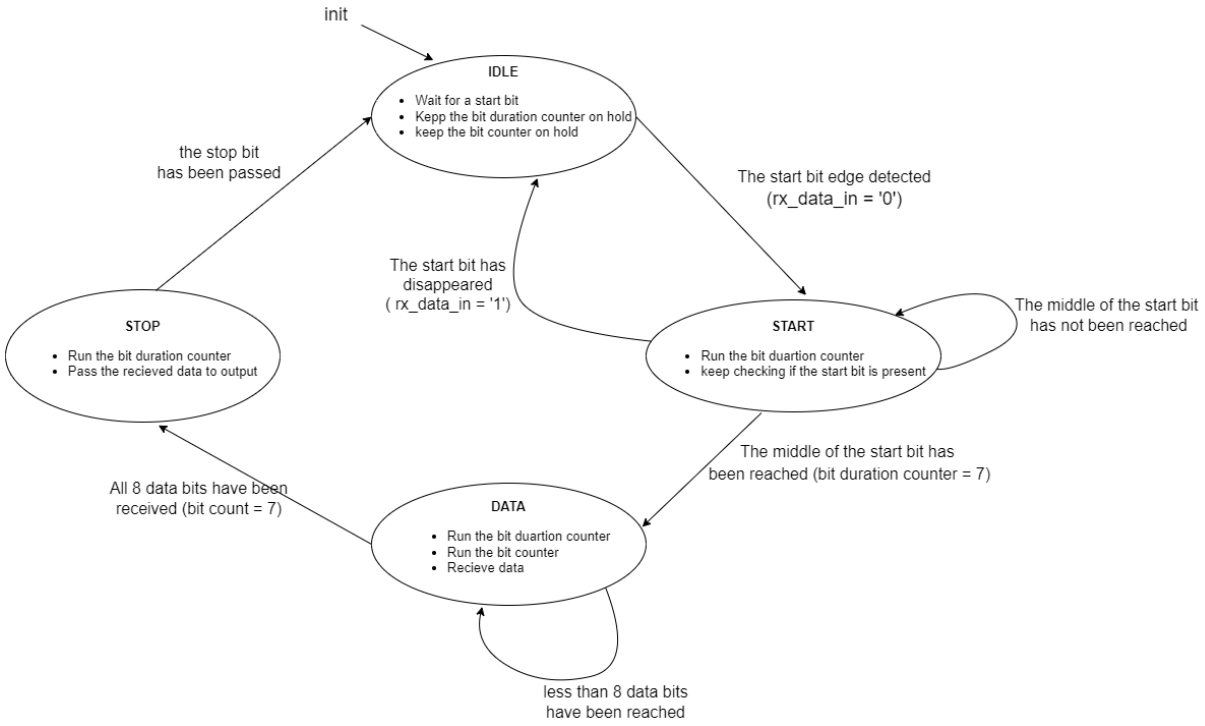


Figure 3.4: FSM of UART receiver module

The FSM module of the transmission typically contains 4 states: IDLE, START, DATA and STOP.

The IDLE start is the default start state where the receiver module waits for the start bit to start receiving the data. Its starts receiving data after the input data which is usually held high is made low for atleast one clock cycle. During this state, bit duration counter

and bit counter are on hold.
FSM moves to START state after detecting the start bit. Here the bit duration counter is run and the middle bit (bit duration counter=7) is read. If the start bit disappears within this time period i.e. if it becomes high again in these 16 samples then the FSM moves back to IDLE state.
Next in DATA state, all the data bits are received in the similar fashion of sampling them 16 times and reading the middle bit. Once all the data packet bits are captured, the FSM moves to the STOP state.
In the STOP state, the bit duartion counter runs to capture the last stop bit. Once the stop bit is received, the FSM moves back to the initial IDLE state where it waits for any other data being sent to be received.

### 3.1.3 UART controller module

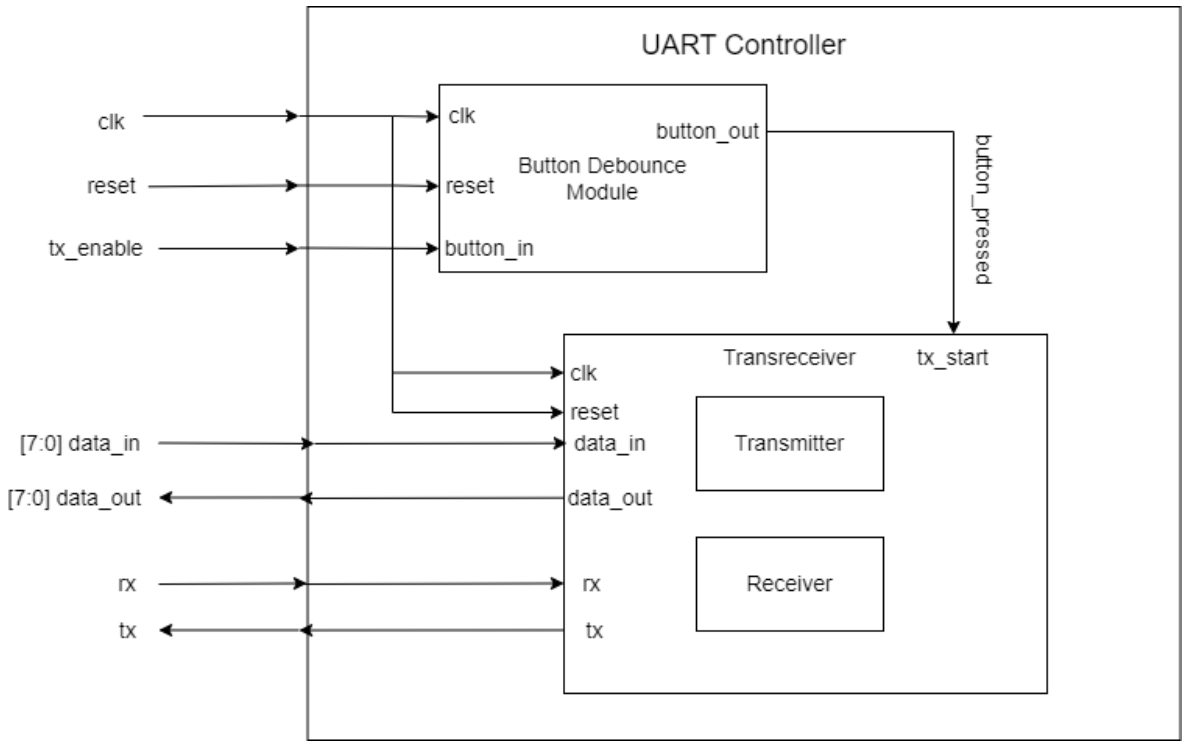


Figure 3.5: UART controller module

UART controller module is the top-level module that controls the entire transmission and receiving operation of UART.
**data_in** is 8-bit wide data that has to be transmitted, the controller sends this to a register present in the transmitter module, which transmits it to Tx port in a serial manner following the operation specified earlier.

UART controller receives data from outside UART controller using Rx port at the set baud rate. It receives it and stores in an 8-bit register. This 8-bit register drives 8-bit wide bus **data_out**.
The Button debounce module in the controller helps to ensure that the transmission data is sampled at the correct time i.e. the user sets the data using the switches and then presses the button (usually BTNC or BTNU). This button press is detected by the button debounce module which then sends appropriate signals to start the transmission.
Other signals to this UART controller module include the clock and reset singal. Clock signal is used to generate the baud rates in the transceiver module and reset is used to reset the received data (usually shown by LEDs).

So, this controller design when implemented on FPGA can receive and transmit data with any other device that have UART transceiver(Whatever it may be).

The data being received by the FPGA is indicated by the LEDs present on the Xilinx Basys3 board in the ASCII format while the data that needs to be transmitted is set using the switches on the board.

## 3.2 Communication between FPGA boards

Two or more FPGA boards can send and receive data using the UART communication protocol. The UART controller is implemented on two FPGA boards and the transmission line of one FPGA is fed to the receiver pin of the other FPGA and vice-versa.
These transmission and receiver ports are the PMOD ports present on the FPGA. Note

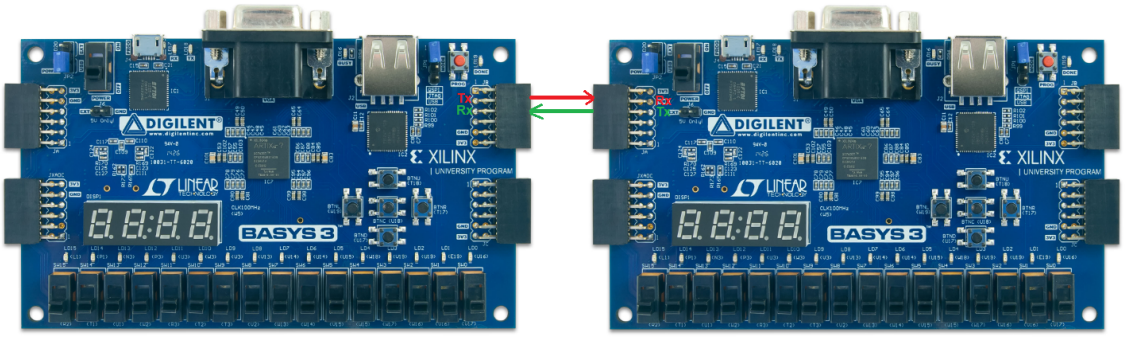


Figure 3.6: Communication between two FPGAs using UART protocol

that the baud rate of communication on both the FPGAs should be the same. To verify whether the data is transmitted and received correctly the following method is used: When Switches from 0-7 of board 1 are set in a particular order, after btnC is pressed LED from 0-7 should light in the corresponding order. Similarly, data can be transmitted from board 2 to 1.

## 3.3 Wireless communication using Bluetooth

One of the applications of the UART transceiver implemented on Artix-7 chip is that we can use it to receive and transmit data wirelessly to any Bluetooth device just by adding a simple Bluetooth module HC05.

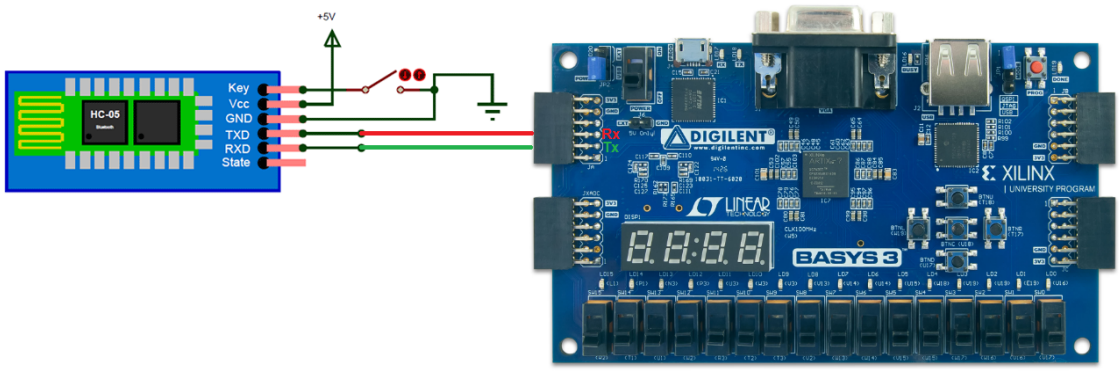


Figure 3.7: Wireless Communication between FPGA and PC using HC05

The HC05 module uses Bluetooth version 2.0 + EDR (Enhanced Data Rate) and supports the Serial Port Profile (SPP) for serial communication. It operates on a frequency range of 2.4 GHz to 2.48 GHz and has a maximum communication range of up to 10 meters (33 feet) in an open space. The data that has to be sent wirelessly is given as input to RXD port using UART transceiver at a default baud rate of 9600bits/s. Similarly, the data received wirelessly is sent to TXD port.

Note that the required power supply voltage for HC05 is 5V, and PMOD port power supply ports in Basys 3 can provide a maximum voltage of only 3.3V so for that purpose external 5V power supply is used.

In PC we have to use Tera-Term COM terminal to establish a connection between HC05 and PC. HC05 uses the default password 1234 for Bluetooth connection.

## 3.4 FPGA I/O ports used for UART transceiver

The designed UART transceiver is implemented on the Xilinx Basys3 board. So here is some information on the related parts of the board.

### PMOD Ports

All the I/O ports that are circled red are PMOD ports. The Pmod ports are arranged in a 2×6 right angle and are 100-mil female connectors that mate with standard 2×6 pin headers.

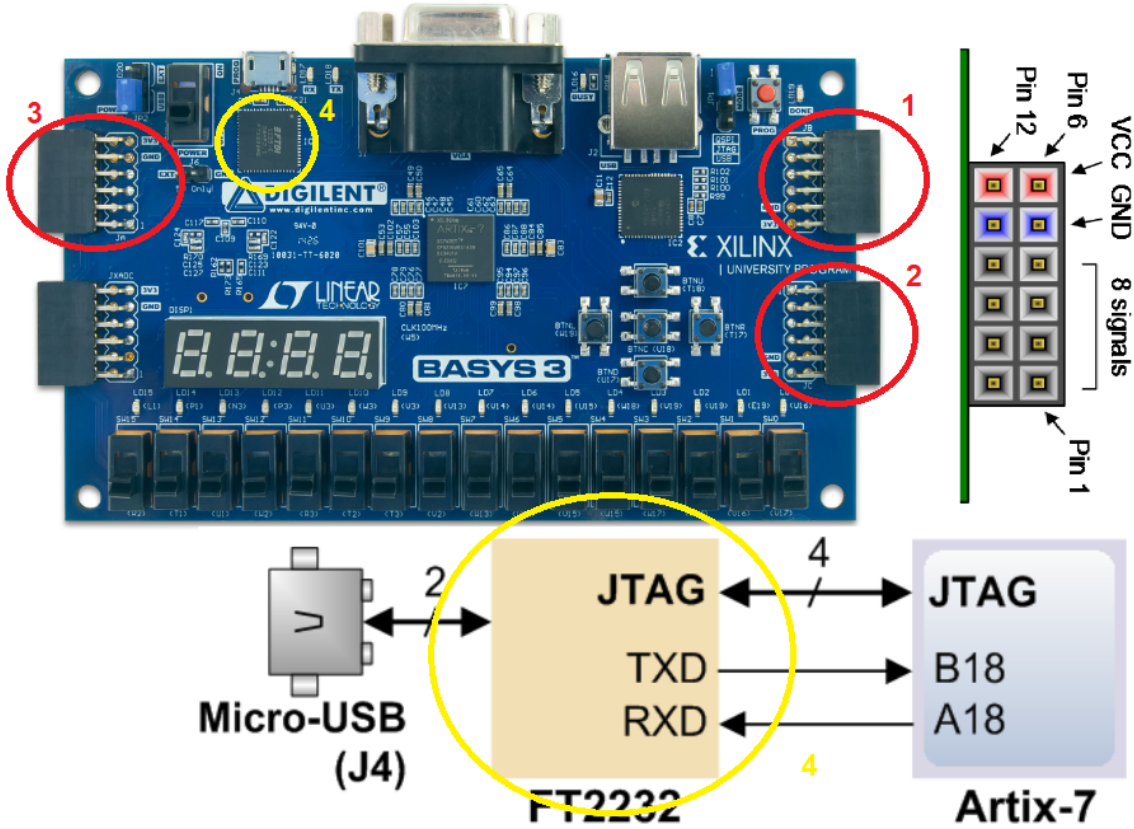


Figure 3.8: Basys 3 FPGA I/O

Each 12-pin Pmod ports provides two 3.3V VCC signals (pins 6 and 12), two Ground signals (pins 5 and 11), and eight logic signals. The VCC and Ground pins can deliver up to 1A of current. Pmod data signals are not matched pairs, and they are routed using best-available tracks without impedance control or delay matching.

### USB-UART Bridge (Serial Port)

The Basys 3 includes an FTDI FT2232HQ USB-UART bridge (attached to connector J4) that allows you to use PC applications to communicate with the board using standard Windows COM port commands. FT2232HQ USB-UART bridge(circled yellow) converts USB packets to UART/serial port data. Serial port data is exchanged with the FPGA using a two-wire serial port (TXD/RXD).
When we need to communicate the data between FPGA and PC, USB-UART bridge has to be used, and in the other type of data communication between any other device and FPGA we will be using PMOD ports.

# Chapter 4

# Evaluation and testing

## 4.1 Synthesis and Implementation Results

The hardware resource utilization for the UART transceiver after implementing on the Artix-7 FPGA can be found below:

| Resource | Utilization | Available | Utilization % |
|---|---|---|---|
| LUT | 122 | 20800 | 0.59 |
| FF | 149 | 41600 | 0.36 |
| IO | 33 | 106 | 31.13 |
| BUFG | 1 | 32 | 3.13 |

| Resource | Estimation | Available | Utilization % |
|---|---|---|---|
| LUT | 123 | 20800 | 0.59 |
| FF | 141 | 41600 | 0.34 |
| IO | 33 | 106 | 31.13 |
| BUFG | 1 | 32 | 3.13 |

Figure 4.1: Post-Implementation and Post-synthesis resource utilization

The power utlizitation report for UART transceiver after mplementation on Atrix-7 can be found below:

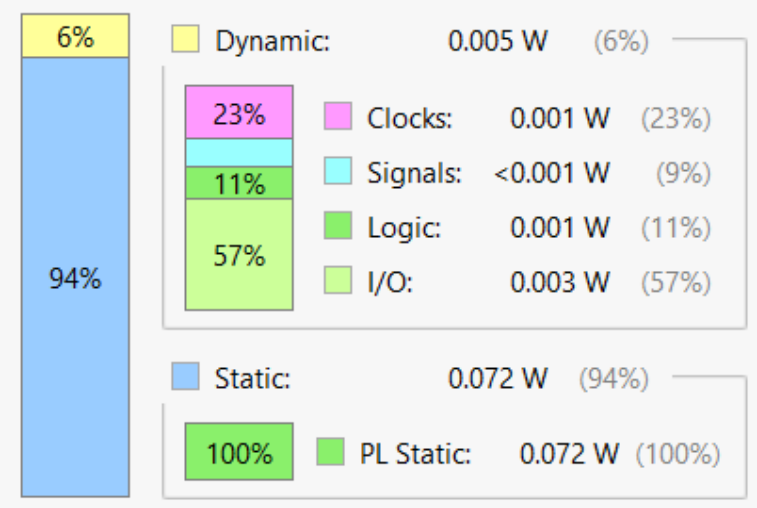


Figure 4.2: Power utilization report

The schematic and mapped hardware of UART transceiver after implementation on Atrix-7 can be found below:

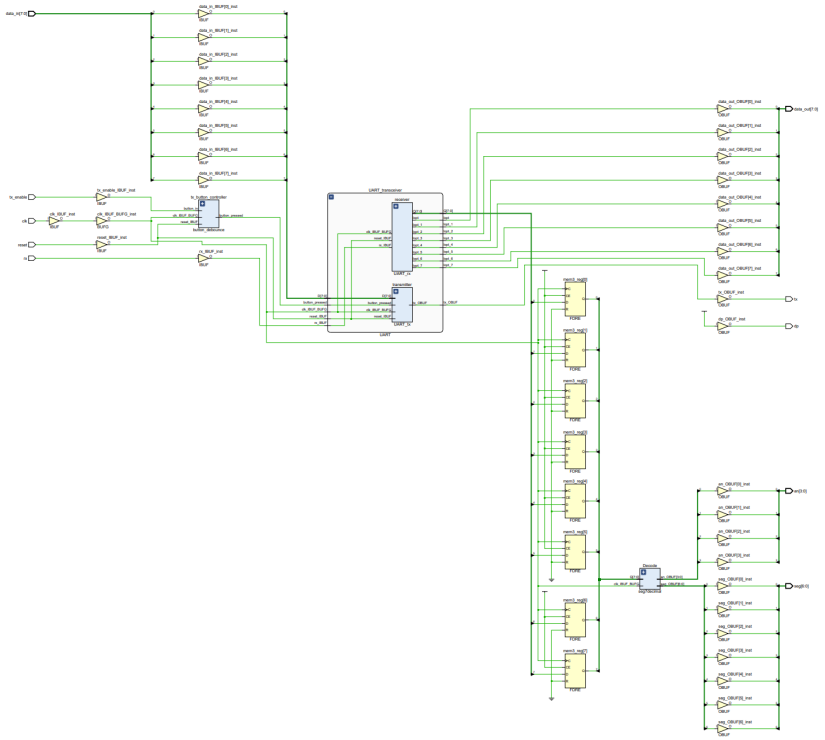

Figure 4.3: Schematic of UART transceiver

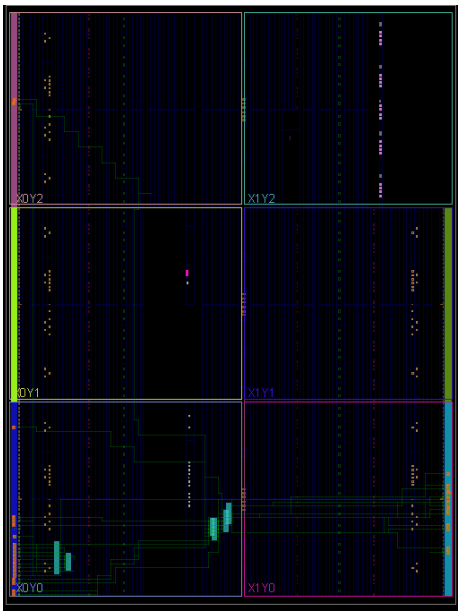


Figure 4.4: Mapped Hardware

Simulation results of UART can be found below:

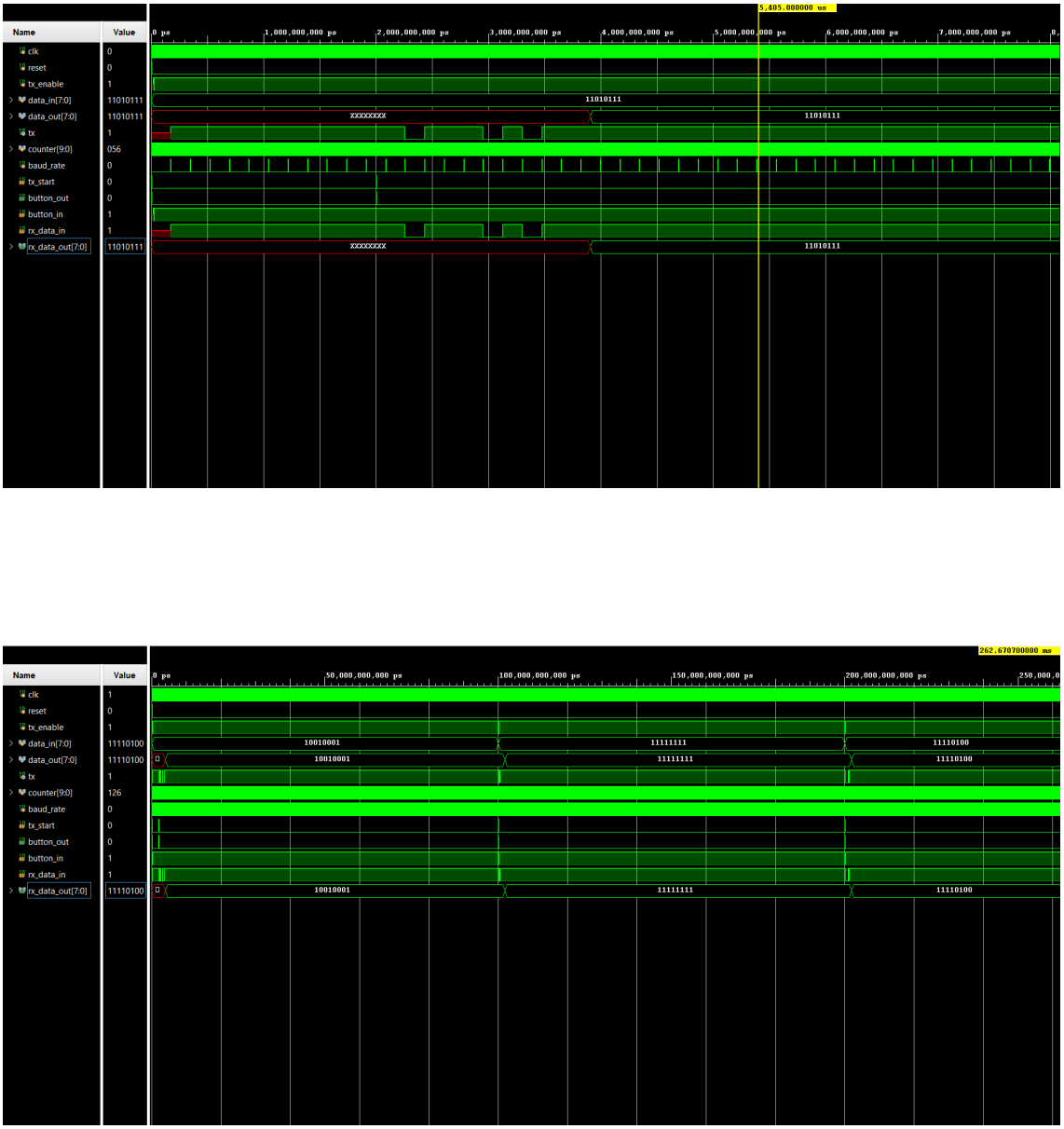


Figure 4.5: Simulation of UART transceiver on vivado

The hardware implementation to represent UART based communication between a FPGA board and a bluetooth device can be found below:

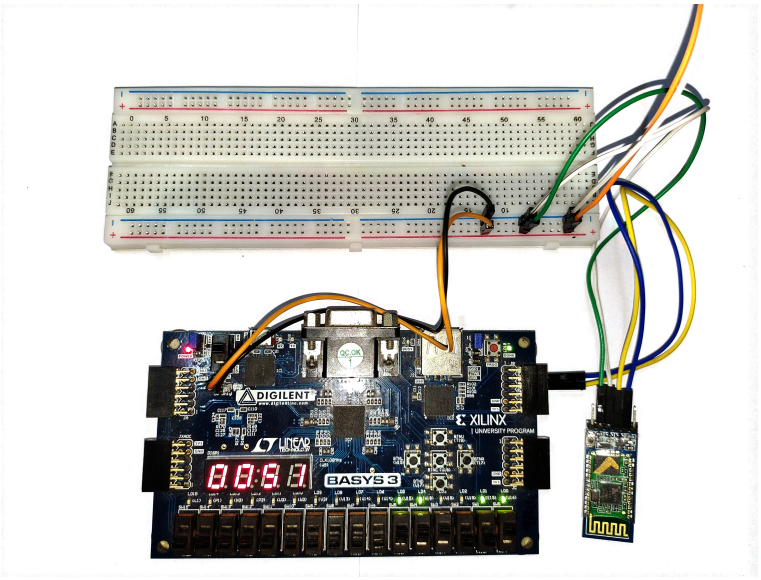


Figure 4.6: UART based communication between FPGA and bluetooth device

The hardware implementation to represent UART based communication between two FPGA boards can be found below:

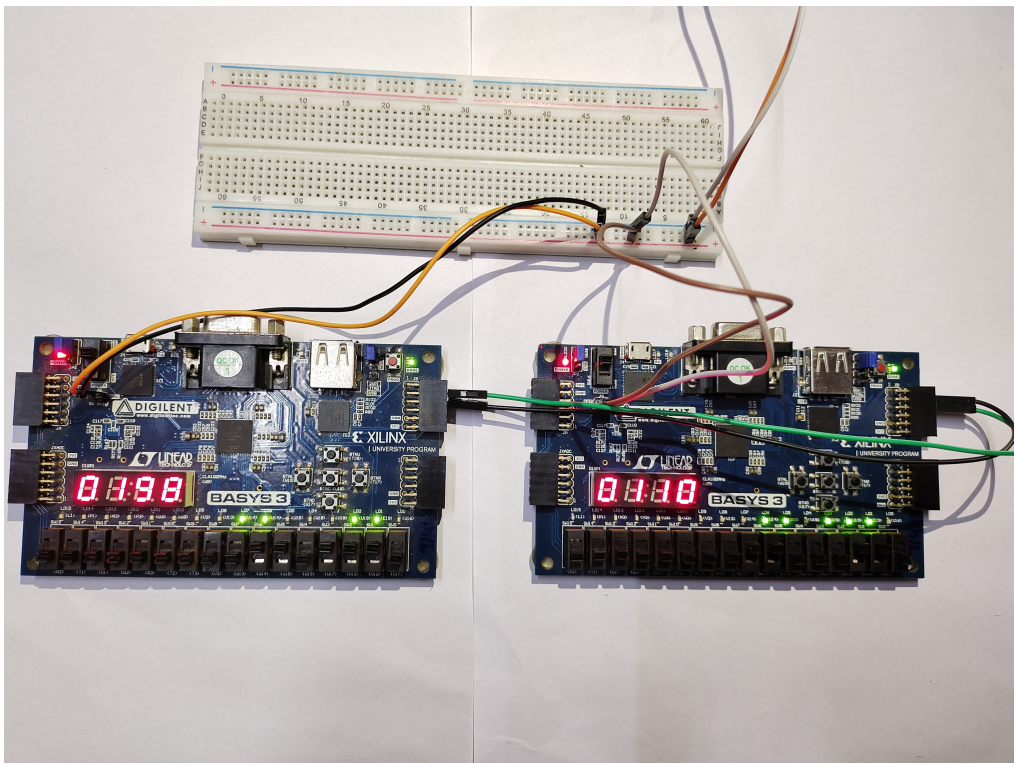


Figure 4.7: UART based communication between two FPGAs

The hardware implementation to represent UART based communication between multiple FPGA boards can be found below:

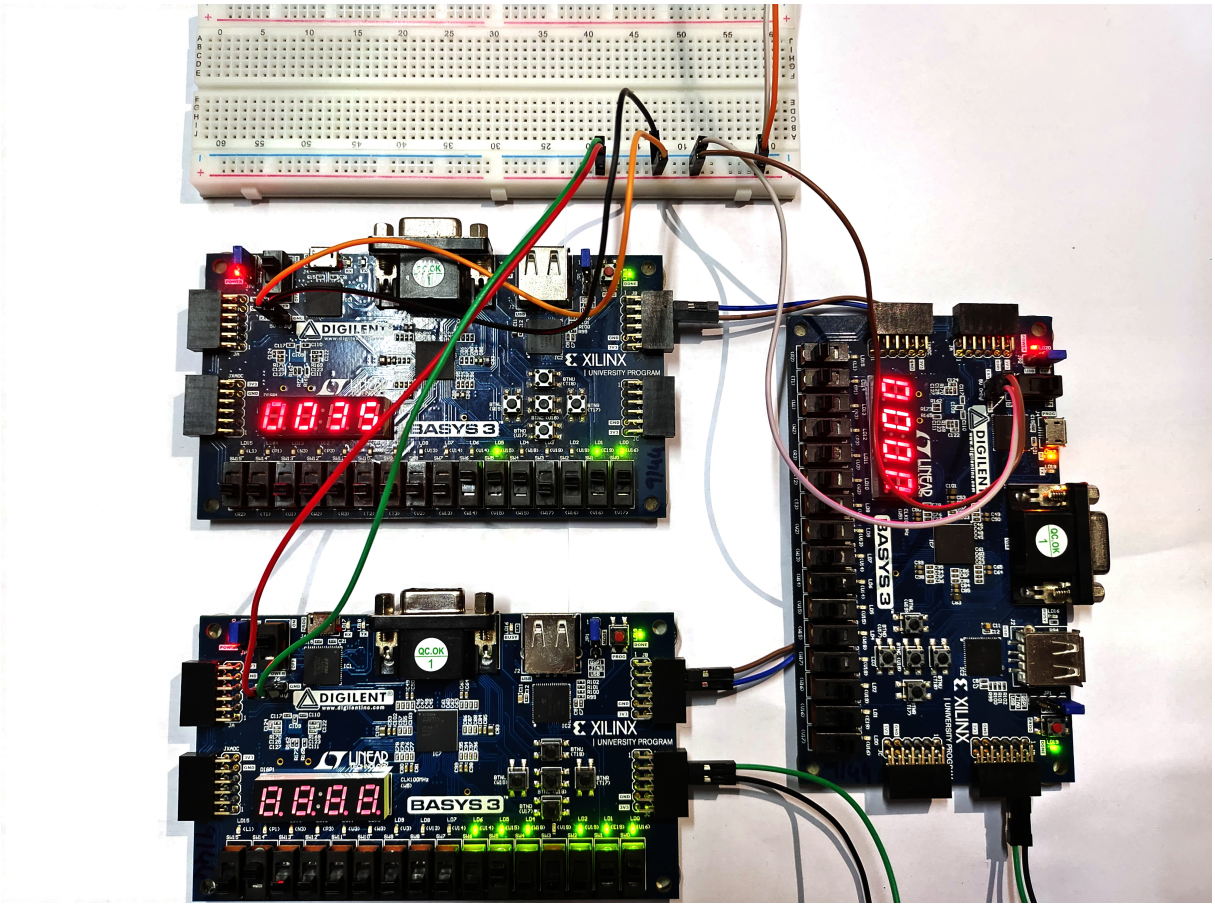


Figure 4.8: UART based communication between multiple FPGAs

# Chapter 5
# Conclusion

In conclusion, the implementation of UART on an FPGA provides a versatile and efficient solution for serial communication. Through the use of hardware design and programming techniques, we were able to successfully integrate UART into the FPGA and develop a working system. This project demonstrated the importance of understanding the principles and specifications of UART, as well as the intricacies of FPGA design and programming.

Furthermore, the implementation of UART on an FPGA has significant practical applications, particularly in the fields of telecommunications and embedded systems. The ability to transmit data in a serial format over a physical connection is a fundamental requirement in many industries, and the implementation of UART on an FPGA provides an optimized and cost-effective solution.

Overall, this project was a valuable learning experience that allowed us to apply theoretical knowledge to practical implementation. The successful integration of UART on an FPGA showcases the potential of hardware design and programming to solve real-world problems, and highlights the importance of continuous research and development in the field of digital systems.

# Bibliography

[Harutyunyan et al., 2020] Harutyunyan, S., Kaplanyan, T., Kirakosyan, A., and Momjyan, A. (2020). Design and verification of autoconfigurable uart controller. In *2020 IEEE 40th International Conference on Electronics and Nanotechnology (ELNANO)*, pages 347–350.

[HU et al., 2007] HU, Z., ZHANG, J., and ling LUO, X. (2007). A novel design of efficient multi-channel uart controller based on fpga. *Chinese Journal of Aeronautics*, 20(1):66–74.

[Kralev, 2014] Kralev, J. (2014). Design of uart controller for fpga with simulink®. In *2014 3rd Mediterranean Conference on Embedded Computing (MECO)*, pages 44–47.

[More et al., 2015] More, J., Suryavanshi, R., Dasarwar, G., Sivanantham, S., and Sivasankaran, K. (2015). Fpga implementation of universal asynchronous transmitter and receiver. In *2015 Online International Conference on Green Engineering and Technologies (IC-GET)*, pages 1–3.

[Poorani and Kurunjimalar, 2016] Poorani, M. and Kurunjimalar, R. (2016). Design implementation of uart and spi in single fgpa. In *2016 10th International Conference on Intelligent Systems and Control (ISCO)*, pages 1–5.

[Wakhle et al., 2012] Wakhle, G. B., Aggarwal, I., and Gaba, S. (2012). Synthesis and implementation of uart using vhdl codes. In *2012 International Symposium on Computer, Consumer and Control*, pages 1–3.

[Wang and Song, 2011] Wang, Y. and Song, K. (2011). A new approach to realize uart. In *Proceedings of 2011 International Conference on Electronic Mechanical Engineering and Information Technology*, volume 5, pages 2749–2752.

[Yu et al., 2007] Yu, S., Yi, L., Chen, W., and Wen, Z. (2007). Implementation of a multi-channel uart controller based on fifo technique and fpga. In *2007 2nd IEEE Conference on Industrial Electronics and Applications*, pages 2633–2638.

# Appendix

## .1 Abbreviations Used

| | | |
|---|---|---|
| FPGA | : | Field Programmable Gate Array |
| UART | : | Universal Asynchronous Receiver Transmitter |
| SPI | : | Serial Peripheral Interface |
| I2C | : | Inter-Integrated Circuit |
| SDA | : | Serial Data |
| SCLK | : | Serial Clock |

## .2 Code Repositories

The codes of all the modules can be found in the following github repository `https://github.com/achyutshegade/UART-Transceiver-on-FPGA.git`