TDD-FDP-CCE

Design Project (Technical Design Document)

Project Name	UART		
Student	[1] Phan Minh Nhật [2] Trần Tuấn Kiệt [3] Nguyễn Ngọc Huy [4] Mai Hồng Phong	**ID**	20119147 20119009 20119014 18119192
Major	Computer Engineer Technology	**Supervisor**	Prof.Do Duy Tan

Table of Contents

▪ ABSTRACT

UART, or universal asynchronous receiver-transmitter, is one of the most used device-to-device communication protocols. This article shows how to use UART as a hardware communication protocol by following the standard procedure.

When properly configured, UART can work with many different types of serial protocols that involve transmitting and receiving serial data. In serial communication, data is transferred bit by bit using a single line or wire. In two-way communication, we use two wires for successful serial data transfer. Depending on the application and system requirements, serial communications needs less circuitry and wires, which reduces the cost of implementation.

In this article, we will discuss the fundamental principles when using UART, how to code a FPGA UART, with a focus on packet transmission, standard frame protocol, and customized frame protocols that are value added features for security compliance when implemented, especially during code development. During product development, this document also aims to share some basic steps when checking on a data sheet for actual usage.

At the end of the article, the goal is for better understanding and compliance of UART standards, use Xilinx ISE to code and testbench the FPGA UART to understand the UART protocols and how coding FPGA.

Terms and abbreviations

[UART]	Universal asynchronous receiver-transmitter
[FPGA]	Field Programmable Gate Array.
[ISE]	Integrated software environment.
[ATM]	Asynchronous Transfer Mode.
[ISE]	Integrated software environment.

▪ List of Tables & Figures

1 Introduction

1.1 What is UART?

A universal asynchronous receiver-transmitter is a computer hardware device for asynchronous serial communication in which the data format and transmission speeds are configurable. It sends data bits one by one, from the least significant to the most significant, framed by start and stop bits so that precise timing is handled by the communication channel. The electric signaling levels are handled by a driver circuit external to the UART. Two common signal levels are RS-232, a 12-volt system, and RS-485, a 5-volt system. Early teletypewriters used current loops.

A UART is usually an individual (or part of an) integrated circuit (IC) used for serial communications over a computer or peripheral device serial port. One or more UART peripherals are commonly integrated in microcontroller chips. Specialised UARTs are used for automobiles, smart cards and SIMs.

1.2 Interface

By definition, UART is a hardware communication protocol that uses asynchronous serial communication with configurable speed. Asynchronous means there is no clock signal to synchronize the output bits from the transmitting device going to the receiving end. The Figure.1 describes how two UARTs can communicate with each other.

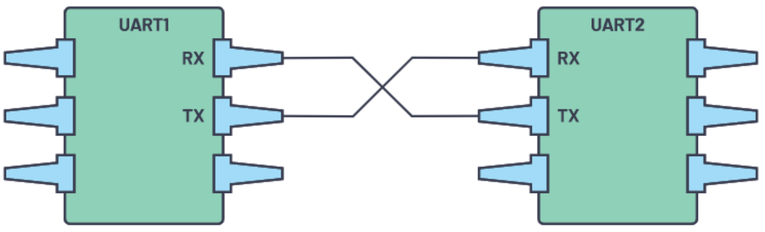


Figure 1: Simple UART interface [1]

The main purpose of a transmitter and receiver line for each device is to transmit and receive serial data intended for serial communication.

1 Image from I2C Communication Protocol: Understanding I2C Primer, PMBus, and SMBus. Access link: https://www.analog.com/ru/analog-dialogue/articles/i2c-communication-protocol-understanding-i2c-primer-pmbus-and-smbus.html

The transmitting UART is connected to a controlling data bus that sends data in a parallel form. From this, the data will now be transmitted on the transmission line (wire) serially, bit by bit, to the receiving UART. This, in turn, will convert the serial data into parallel for the receiving device. This is described in Figure 2.

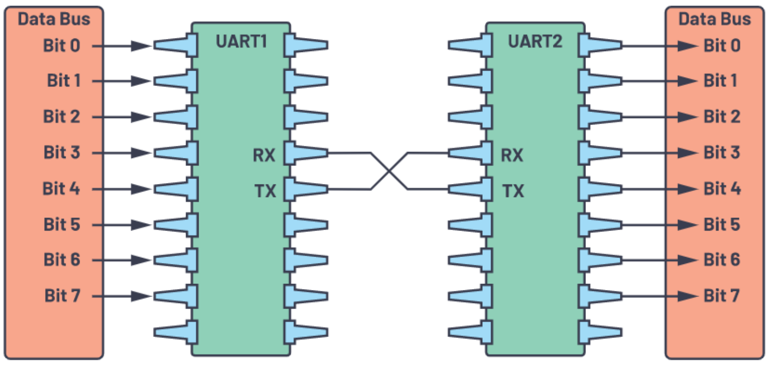


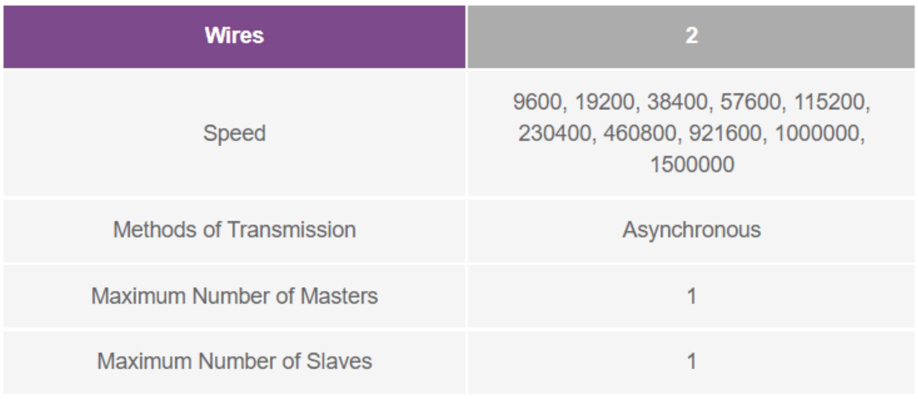
Figure 2. UART interface with data bus [2]

[2] *Image from I2C Communication Protocol: Understanding I2C Primer, PMBus, and SMBus. Access link: https://www.analog.com/ru/analog-dialogue/articles/i2c-communication-protocol-understanding-i2c-primer-pmbus-and-smbus.html*

2 Background

2.1 The baud rate

The baud rate is the rate at which information is transferred to a communication channel. In UART the baud rate needs to be set the same on both the transmitting and receiving device. Some of the Baudrate is used in the table.



Wires	2
Speed	9600, 19200, 38400, 57600, 115200, 230400, 460800, 921600, 1000000, 1500000
Methods of Transmission	Asynchronous
Maximum Number of Masters	1
Maximum Number of Slaves	1

Table 1: Baud rates are used commonly

Because the UART interface does not use a clock signal to synchronize the transmitter and receiver devices; it transmits data asynchronously. Instead of a clock signal, the transmitter generates a bitstream based on its clock signal while the receiver is using its internal clock signal to sample the incoming data. The point of synchronization is managed by having the same baud rate on both devices. Failure to do so may affect the timing of sending and receiving data that can cause discrepancies during data handling. The allowable difference of baud rate is up to 10% before the timing of bits gets too far off.

2.2 Data Transmission

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.

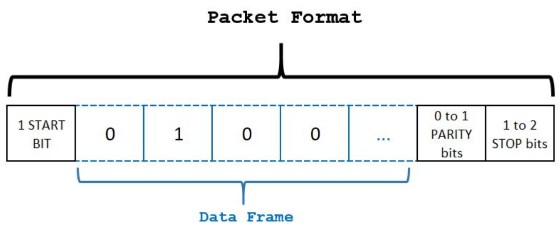


Figure 3: Packet format of transmitting/receiving UART [3]

3 Image from UART Serial Communication. Access link: https://www.romn.io/2020/12/uart-serial-communication.html

In 1 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. The Figure 3 shows how the Start bit work accurately.

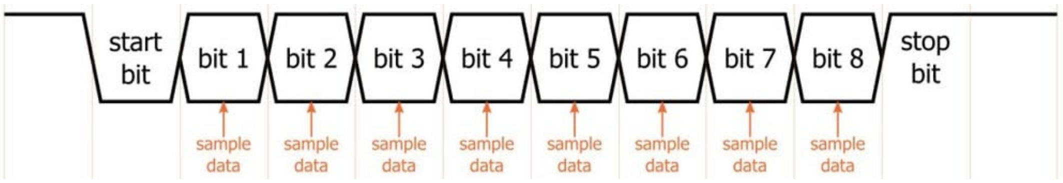


Figure 4. The Start bit is low when the signal start [4]

The data frame contains the actual data being transferred. It can be 5 bits up to 8 bits long if a parity bit is used. If no parity bit is used, the data frame can be 9 bits long. In most cases, the data is sent with the least significant bit first.

Parity describes the evenness or oddness of a number. The parity bit is a way for the receiving UART to tell if any data has changed. 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. If the parity bit is a 0 (even parity), the 1 or logic-high bit in the data frame should total to an even number. If the parity bit is a 1 (odd parity), the 1 bit or logic highs in the data frame should total to 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, or the parity bit is a 1, and the total is even, the UART knows that bits in the data frame have changed.

Stop Bits 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 1 to 2 bit(s) duration.

2.3 Asynchronous Transmission

Asynchronous transmission is a type of data transmission which works on start and stop bits. In Asynchronous transmission, each character contains its start and stop bit and irregular interval of time between them. So we don’t need a same clock pulse for both receiver and transmitter.

In Asynchronous Transmission, data is sent in form of byte or character. This transmission is the half-duplex type transmission. In this transmission start bits and stop bits are added with data. It does not require synchronization.

4 Image from: Back to Basics: The Universal Asynchronous Receiver/Transmitter (UART), access link: https://www.allaboutcircuits.com/technical-articles/back-to-basics-the-universal-asynchronous-receiver-transmitter-uart/

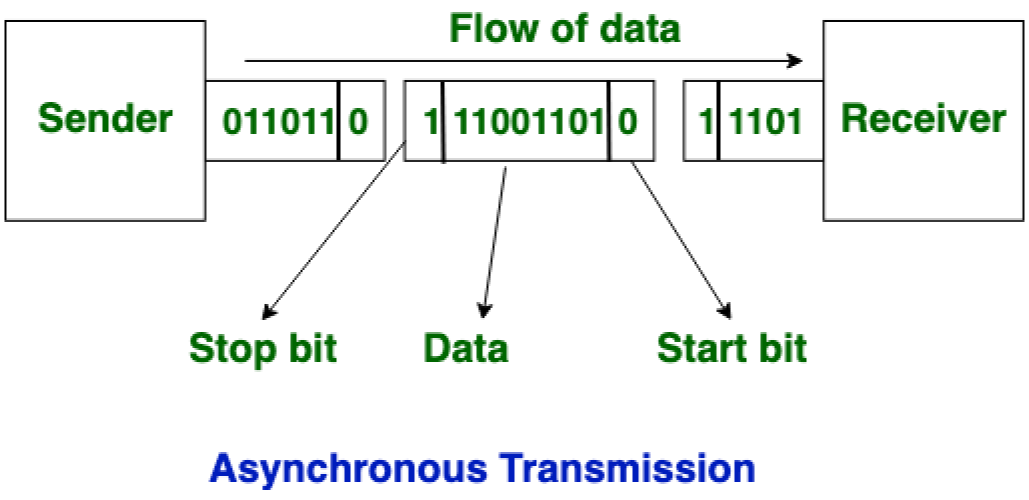


Figure 5. Asynchronous Transmission [5]

Difference Between Synchronous and Asynchronous Transmission

S.No.	Synchronous	Asynchronous
1.	In Synchronous transmission a common clock is shared by the transmitter and receiver to achieve synchronisation while data transmission.	In Asynchronous transmission each character contains its own start and stop bits.
2.	In Synchronous transmission data is sent in frames or blocks.	In Asynchronous transmission data is sent in the form of bytes or characters.
3.	Synchronous transmission is faster, as a common clock is shared by the sender and receiver.	Asynchronous transmission is slower as each character has its own start and stop bit.
4.	Synchronous transmission is costlier.	Asynchronous transmission is cheaper.
5.	It is easy to design.	It is complex.
6.	In synchronous transmission there is no gap between the data as they share a common clock.	In asynchronous transmission there is a gap between the data due to the start and stop bit feature.

Table 2: Compare Asynchronous and Synchronous

5 Image from Difference between Synchronous and Asynchronous Transmission, access link: https://www.geeksforgeeks.org/difference-between-synchronous-and-asynchronous-transmission/

❖ Transmitting – receiving step/algorithm

1. UART transmits and receives data in parallel from the data bus, a flags can be used to start the process.

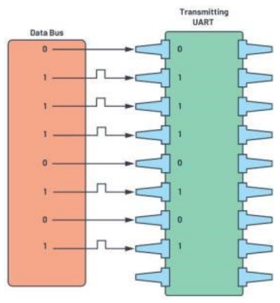


Figure 6. Load data from data bus to UART transmitter [6]

2. UART transmits more start bit, parity bit (may be) and stop bit to data frame.

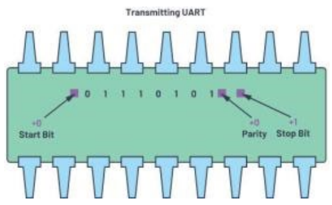


Figure 7: Add more start bit, stop bit (parity bit may be) [7]

3. The entire packet is sent serially from the transmitting UART to the receiving UART. UART receives data line sampling at preconfigured baud rate.

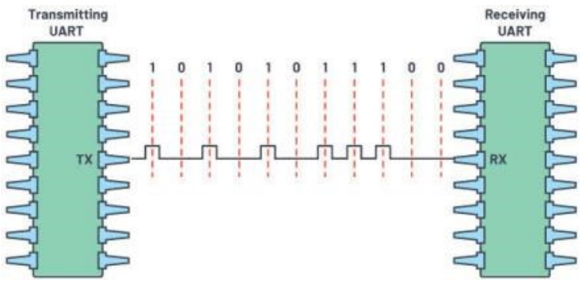


Figure 8: Transmit to receiver UART [8]

[6, 7, 8] *Image from I2C Communication Protocol: Understanding I2C Primer, PMBus, and SMBus. Access link: https://www.analog.com/ru/analog-dialogue/articles/i2c-communication-protocol-understanding-i2c-primer-pmbus-and-smbus.html*

4. The receiving UART removes the start bit, parity bit and stop bit from the data frame.

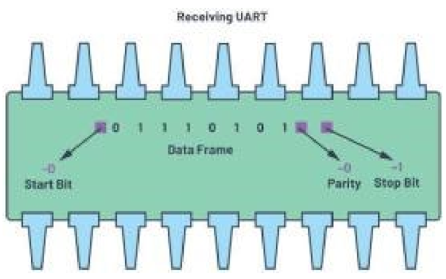


Figure 9. Remove redundant start bit, stop bit (parity bit may be) [9]

5. The receiving UART converts serial data back to parallel and transfers it to the data bus at the receiving end.

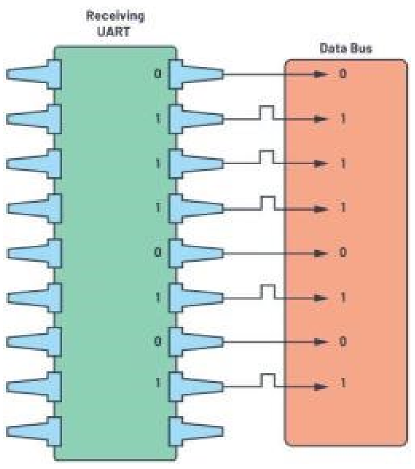


Figure 10. Load the data received to data bus [10]

[9, 10] *Image from I2C Communication Protocol: Understanding I2C Primer, PMBus, and SMBus. Access link: https://www.analog.com/ru/analog-dialogue/articles/i2c-communication-protocol-understanding-i2c-primer-pmbus-and-smbus.html*

3 System Architecture

3.1 I/O connection of UART

Input/Output Connections describes the various input and output connections for the UART. Some I/Os may be hidden on the symbol under the conditions listed in the description of that I/O. The Schematic of UART is described in Figure 6.

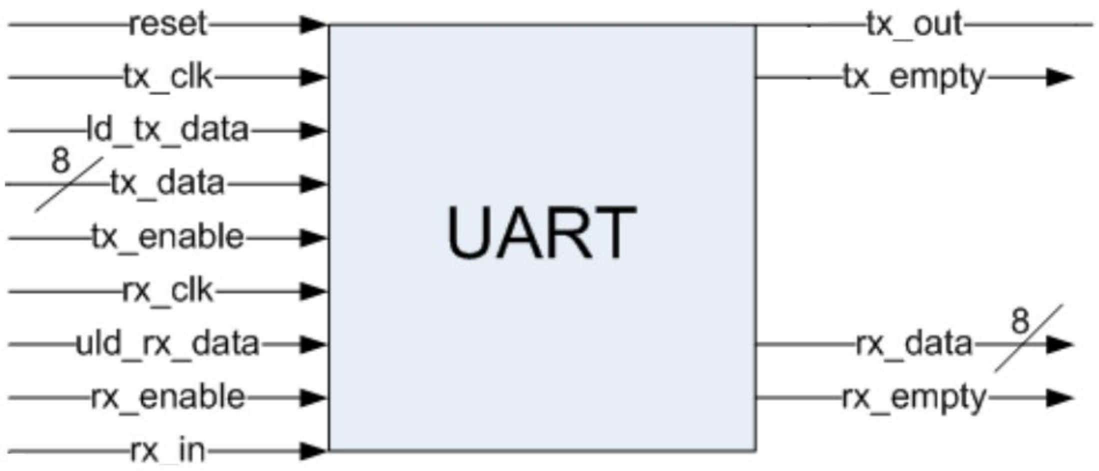


Figure 13: UART module pinout [11]

Input port: reset, tx_clk, ld_tx_data, [7:0] tx_data, tx_enable, rx_clk, uld_rx_data, rx_enable, rx_in

Output port: [7:0] rx_data, rx_empty

3.2 Functional description of I/O pin

Reset: reset all port, reg, memory to appropriate start-up value (0/1), reset will be impacted by user or when our module start-up.

Tx_clk, rx_clk: clock baud rate of Tx and Rx, if Tx and Rx is the same baud rate, frequency of Rx will be 16 times faster than Tx

Ld_tx_data: load data from [7:0] tx_data to memory reg in module (address lock)

Tx_enable, rx_enable: using Tx, Rx when the signal is 1. If it's 0: stop all function.

Rx_in: receiving the signal from UART transmitter.

Tx_out: Transmitting the signal out to UART receiver.

Uld_rx_data: load data received from Rx UART to [7:0] rx_data.

11 Image from: UART Universal asynchronous receiver and transmitter, access link: https://redirect.cs.umbc.edu/~tinoosh/cmpe650/slides/UART.pdf

Tx_empty: transmitting in progress if Tx_empty is 1
Rx_empty: receiving in progress if Rx_empty is 1

3.3 Process of Transmitting and Reciving data in Block Diagram

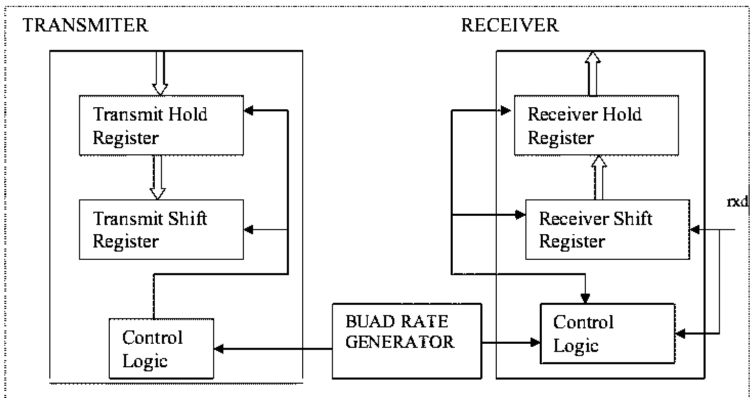


Figure 14. Block Diagram describes the process of transmit and receive data [12]

The parts of the transmitter are transmitted hold registers. shift register and control logic. On the other sides receiver has a hold register, shift register, and control logic. These two has a common configuration of baud rate generator Speed produced by the baud rate generator defines the data sending and receiving speed of the transmitter and receiver. There are data byte that exist in the transmit hold register for sending.

In the Figure 8, the Baud rate Generator play an very important role, it will determine the period of clock pulse in Tx and Rx. Two of them use different clock rate. The receive UART uses a clock that is 16 times the data rate. A new frame is recognized by the falling edge at the beginning of the active-low START bit. This occurs when the signal changes from the active high STOP bit or bus idle condition. The receive UART resets its counters on this falling edge, expects the mid-START bit to occur after 8 clock cycles, and anticipates the midpoint of each subsequent bit to appear every 16 clock cycles thereafter. The START bit is typically sampled at the middle of bit time This will be obvious in chapter 3.

12 Image from Research Gate, access link: https://www.researchgate.net/figure/82a-Block-Diagram-of-UART_fig10_294886121

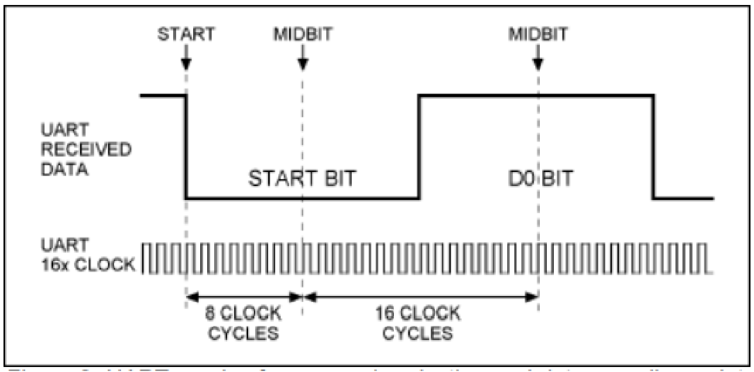


Figure 15. Detect midbit of every bitframe [13]

For example, at 9600 bps a bit time is 104 µs, then it would sample the start bit at T0 + 52 µs, the first data bit at T0 + 52 µs + 104 µs, the second data bit at T0 + 52 µs + 2 × 104 µs, and so on. T0 is the falling edge of the start bit. While sampling the start bit isn't really necessary (you know it's low) it's useful to ascertain that the start edge wasn't a error.

13 Image from UART Communication on Basys 3, FPGA Dev Board Powered by Xilinx Artix 7 Part II, Access link: https://www.instructables.com/UART-Communication-on-Basys-3-FPGA-Dev-Board-Power-1/

Detailing Transmitter block-working (Tx):

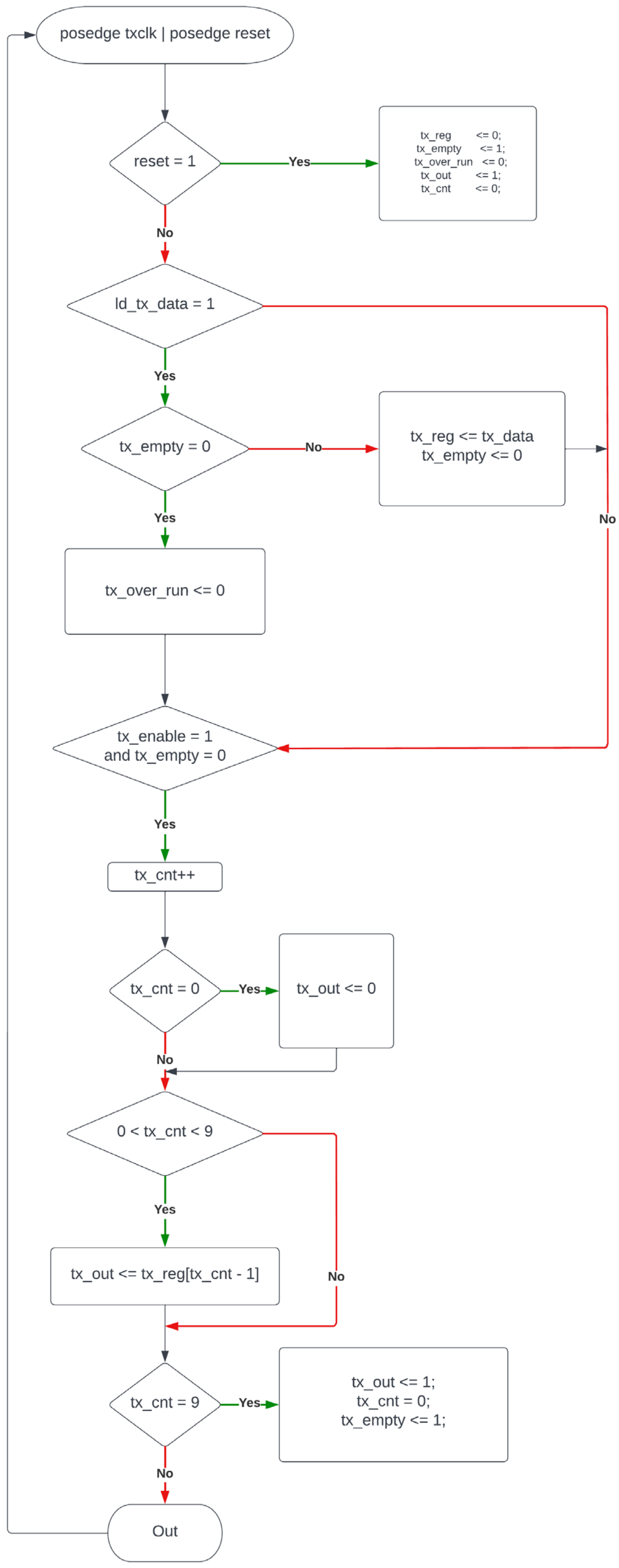


Figure 16: Detailing Transmitter block-working

Detailing Receier block-working (Rx):

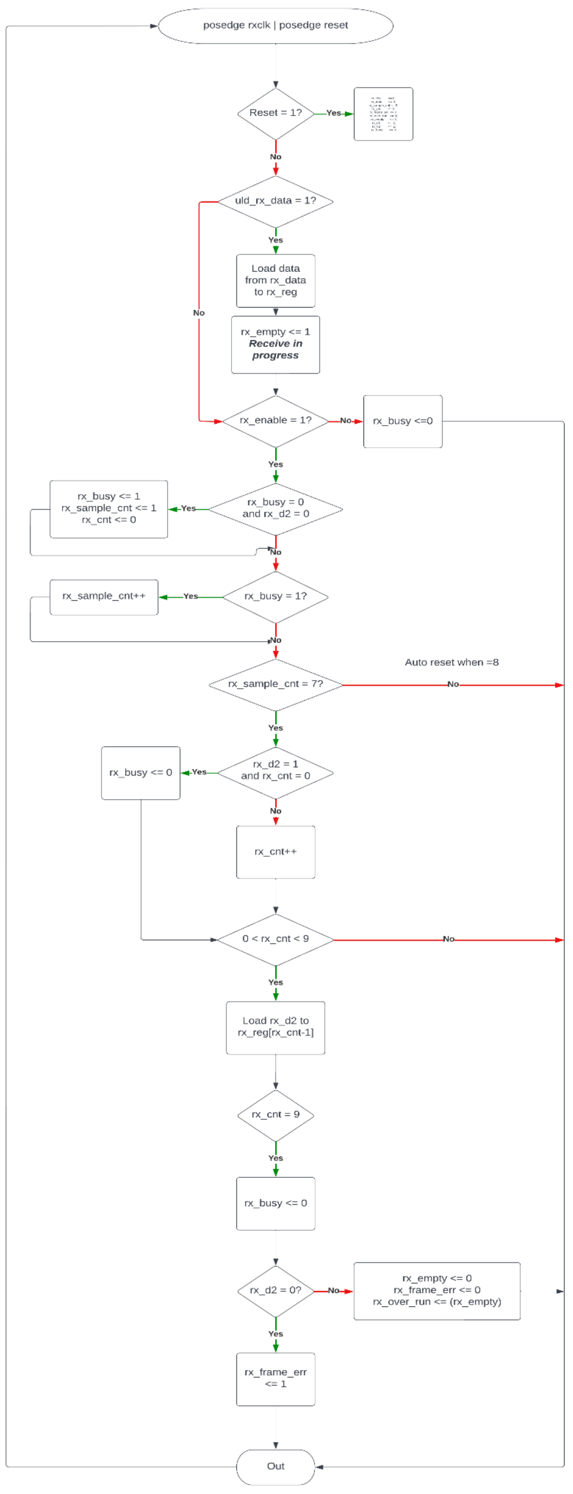


Figure 17: Detailing Receiver block-working

4 Coding of final product

Coding of simple FPGA UART

```
// Design Name : Simple UART with RX & TX
//-----------------------------------------------------
module uart (
                                 reset          ,
                                 txclk          ,
                                 ld_tx_data     ,
                                 tx_data        ,
                                 tx_enable      ,
                                 tx_out         ,
                                 tx_empty       ,
                                 rxclk          ,
                                 uld_rx_data    ,
                                 rx_data        ,
                                 rx_enable      ,
                                 rx_in          ,
                                 rx_empty
);
// Port declarations
input        reset          ;
input        txclk          ;
input        ld_tx_data     ;
input  [7:0] tx_data        ;
input        tx_enable      ;
output       tx_out         ;
output       tx_empty       ;
input        rxclk          ;
input        uld_rx_data    ;
output [7:0] rx_data        ;
input        rx_enable      ;
input        rx_in          ;
output       rx_empty       ;

// Internal Variables
reg [7:0]    tx_reg         ;
reg          tx_empty       ;
reg          tx_over_run    ;
reg [3:0]    tx_cnt         ;
reg          tx_out         ;
reg [7:0]    rx_reg         ;
reg [7:0]    rx_data        ;
reg [3:0]    rx_sample_cnt  ;
reg [3:0]    rx_cnt         ;
reg          rx_frame_err   ;
reg          rx_over_run    ;
reg          rx_empty       ;
reg          rx_d1          ;
reg          rx_d2          ;
reg          rx_busy        ;

// UART RX Logic
always @ (posedge rxclk or posedge reset)
       if (reset)
       begin
         rx_reg        <= 0;
         rx_data       <= 0;
       rx_sample_cnt <= 0;
       rx_cnt        <= 0;
       rx_frame_err  <= 0;
       rx_over_run   <= 0;
       rx_empty      <= 1;
       rx_d1         <= 1;
       rx_d2         <= 1;
       rx_busy       <= 0;
     end
     else
     begin
// Synchronize the asynch signal
       rx_d1 <= rx_in;
       rx_d2 <= rx_d1;
       // Uload the rx data
       if (uld_rx_data)
       begin
            rx_data  <= rx_reg;
            rx_empty <= 1;
       end
// Receive data only when rx is enabled
if (rx_enable) begin
  // Check if just received start of frame
  if (!rx_busy && !rx_d2) begin
    rx_busy       <= 1;
    rx_sample_cnt <= 1;
    rx_cnt        <= 0;
  end
  // Start of frame detected, Proceed with rest of data
  if (rx_busy) begin
     rx_sample_cnt <= rx_sample_cnt + 1;
     // Logic to sample at middle of data
     if (rx_sample_cnt == 7) begin
        if ((rx_d2 == 1) && (rx_cnt == 0)) begin
          rx_busy <= 0;
        end else begin
          rx_cnt <= rx_cnt + 1;
          // Start storing the rx data
          if (rx_cnt > 0 && rx_cnt < 9) begin
            rx_reg[rx_cnt - 1] <= rx_d2;
          end
          if (rx_cnt == 9) begin
             rx_busy <= 0;
             // Check if End of frame received correctly
             if (rx_d2 == 0) begin
               rx_frame_err <= 1;
             end else begin
               rx_empty     <= 0;
               rx_frame_err <= 0;
               // Check if last rx data was not unloaded,
               rx_over_run  <= (rx_empty) ? 0 : 1;
             end
          end
        end
     end
  end
end
if (!rx_enable) begin
  rx_busy <= 0;
end
```

```
end

// UART TX Logic
always @ (posedge txclk or posedge reset)
if (reset) begin
  tx_reg        <= 0;
  tx_empty      <= 1;
  tx_over_run   <= 0;
  tx_out        <= 1;
  tx_cnt        <= 0;
end else begin
   if (ld_tx_data) begin
      if (!tx_empty) begin
        tx_over_run <= 0;
      end else begin
        tx_reg   <= tx_data;
        tx_empty <= 0;
      end
   end
   if (tx_enable && !tx_empty) begin
     tx_cnt <= tx_cnt + 1;
     if (tx_cnt == 0) begin
       tx_out <= 0;
     end
     if (tx_cnt > 0 && tx_cnt < 9) begin
        tx_out <= tx_reg[tx_cnt -1];
     end
     if (tx_cnt == 9) begin
       tx_out <= 1;
       tx_cnt <= 0;
       tx_empty <= 1;
     end
   end
   if (!tx_enable) begin
     tx_cnt <= 0;
   end
end

endmodule
```

Testbench file

```
module UART_testbench();
	reg txclk, reset,tx_enable,ld_tx_data,rxclk,rx_enable;
	wire tx_out,tx_empty,rx_in,rx_empty;
	reg[7:0] tx_data;
	wire[7:0] rx_data;
	wire uld_rx_data;
	uart U1 (
		.reset(reset),
		.txclk(txclk),
		.ld_tx_data(ld_tx_data),
		.tx_data(tx_data),
		.tx_enable(tx_enable),
		.tx_out(tx_out),
		.tx_empty(tx_empty),
		.rxclk(rxclk),
		.uld_rx_data(uld_rx_data),
                .rx_data(rx_data),
                .rx_enable(rx_enable),
                .rx_in(rx_in),
                .rx_empty(rx_empty)
                );

        assign rx_in=tx_out;
        assign uld_rx_data=~rx_empty;

        always@(reset,tx_empty)
                begin
                        if(reset)
                                tx_data=170;
                else if (tx_empty)
                        tx_data=tx_data+10;
                end

        initial
        begin
        txclk=0;
        forever #32 txclk=~txclk;
        end

        initial begin
        rxclk=0;
        forever #2 rxclk=~rxclk;
        end

        initial begin
        reset=0;
        ld_tx_data=0;
        tx_enable=0;
        rx_enable=0;
        tx_data=170;
        #10;
        reset=1;
        #20;
        reset=0;

        #50;
        tx_enable=1;
        rx_enable=1;

        #50;
        ld_tx_data=1;
        #400;
        end
endmodule
```

5 UART Simulation and Test

To test the UART module, we will connect the Rx_in and Tx_out to test.

In order to observe the result of the trasmitter and receiver of one UART module , The Pin Tx_out need to be connected to Rx_in and Tx_data connected to Rx_data, which is described in figure 10.

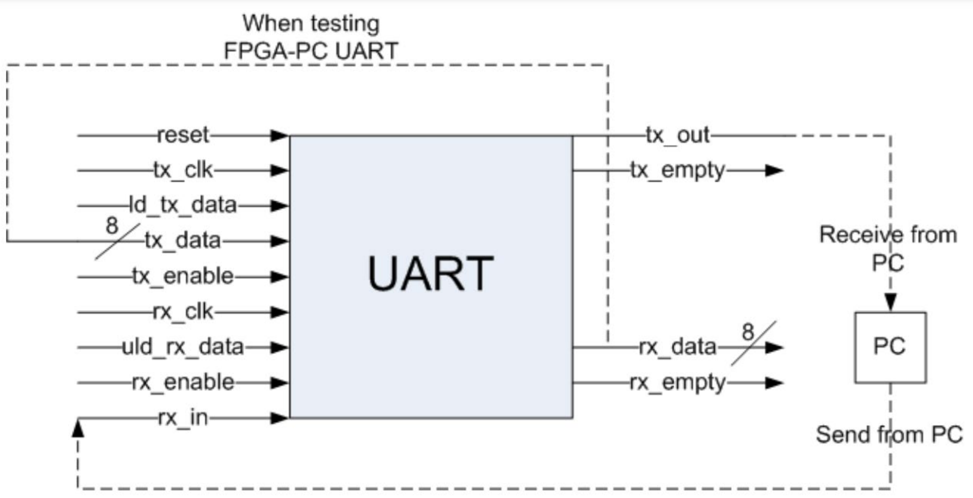
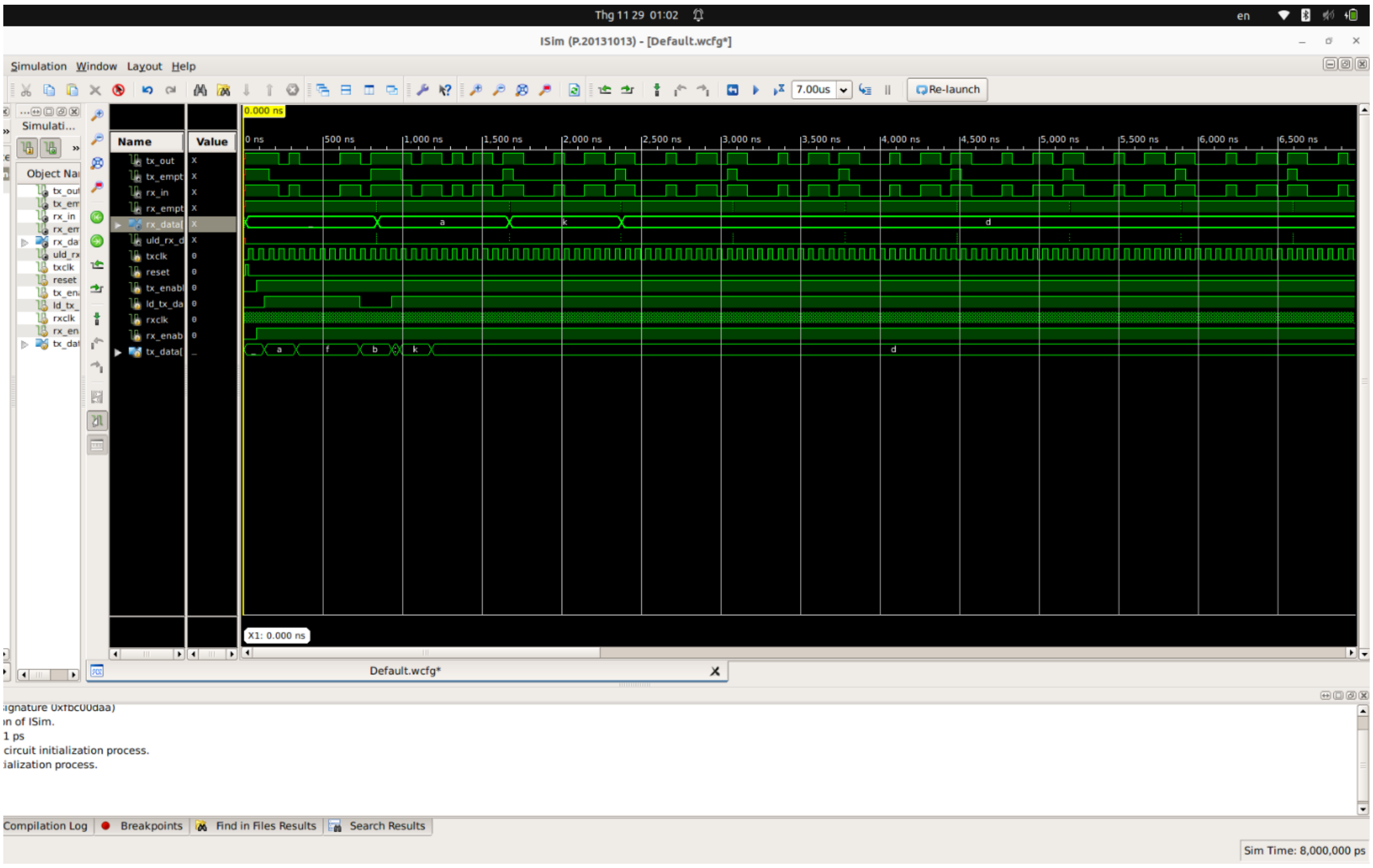


Figure 11: Wiring when testbench [14]

After having all the connection in figure 12, the operating process happens inside the UART is showed in Testbech (Figure 11), which tells the data will be transfer from tx_out line to rx_in. Furthermore, The Tx_data pin stores the data before the progress of transmitting happens, while Rx_data will store the data after the transmitting.

[14] *Image from: UART Universal asynchronous receiver and transmitter, access link: https://redirect.cs.umbc.edu/~tinoosh/cmpe650/slides/UART.pdf*



```
ignature 0xfbc00daa)
on of ISim.
1 ps
circuit initialization process.
ialization process.
```

Figure 12: Testbench UART module

We sent the character, sequently, is “a”, “f”, “b”, “c”, “k” and “d” and watch the Rx_Data to make sure UART module is work correctly.

Since Rx_in and Tx_out is connect together, so the waveform of these two is the same.

The character “f” can’t be sent because of Tx is in transmitting (busy)

The character “b” can’t be sent bacause ld_tx_data is low, so the data from Tx_data can’t be load to memory

The character “c” can’t be sent the wavelenght is small and not in a posedge of Tx_clk, this situation is not able to do in real life, just to testbench the module.

6 Division of Labor

Name	ID	Division of work	Percentage complete
Phan Minh Nhật	20119147	Section 4 and detailing flowchart of Rx, Tx	100%
Trần Tuấn Kiệt	20119009	Coding and Testbench – Section 3 and 5	100%
Nguyễn Ngọc Huy	20119014	Section 4 and graph of UART	100%
Mai Hồng Phong	20119192	Section 1, 2 and slideshow	100%

7 References

[1] PSoC® Creator™ Component Datasheet, Universal Asynchronous Receiver Transmitter (UART), https://www.infineon.com/dgdl/Infineon-Component_UART_V2.20-Software%20Module%20Datasheets-v02_05-EN.pdf?fileId=8ac78c8c7d0d8da4017d0e7f95451159CRC Encoding: https://www.codeproject.com/Articles/5597/CRC-Encoding.
[2] Electrical Engineering, UART receiver clock speed, https://electronics.stackexchange.com/questions/42236/uart-receiver-clock-speed .
[3] Hello03end, Verilog Uart, https://github.com/hell03end/verilog-uart
[4] Maxim Intergrated, Determining Clock Accuracy Requirements for UART Communications, https://pdfserv.maximintegrated.com/en/an/AN2141.pdf
[5] Vinay Sharma, Exp-3- UART X & TX using Xilinx FPGA, https://youtube.com/watch?v=5TVf1hdL4WU&t=56s
[6] WIZnet ,Universal Asynchronous Receive Transmit (UART), https://docs.wiznet.io/Product/iMCU/W7500/Peripherals-internal/uart