VHDL UART Implemetation Assignment 4

Reconfigurable Systems

Instructor: Sergio Brito

sdbrito@ualg.pt

UALg ISE DEE

March 28, 2025

Contents

1 Introduction

This document serves as a companion guide for Assignment 4 of the Reconfigurable Systems course. The objective of the assignment is to design and implement a Universal Asynchronous Receiver-Transmitter (UART) using VHDL, with separate processes for the receiver and the transmitter. In addition, students are required to simulate the functionality of each module using ModelSim.

The companion documentation is organized into the following sections:

- **Theoretical Background:** An overview of UART principles and key concepts relevant to the assignment.
- **Design and Implementation Details:** A detailed explanation of the UART architecture and the VHDL implementation, highlighting the distinct processes for the receiver and transmitter.
- **Simulation Setup:** Instructions for setting up and executing simulations to verify the functionality of the implemented UART.
- **Conclusion and Appendices:** A summary of the document along with additional resources and code listings.

In this document, we will discuss the fundamental concepts of UART communication, the design choices made during the implementation, and the approach taken to validate the system through simulation. For the assignment submission, students are required to include all VHDL source files and a comprehensive report that addresses the simulation of the system. The report must incorporate figures captured from ModelSim to illustrate the behavior of both the receiver and transmitter modules.

2 Theoretical Background

Universal Asynchronous Receiver-Transmitter (UART) is a widely used serial communication protocol that enables data exchange between devices without a shared clock signal. Instead, both the transmitter and receiver rely on pre-defined timing settings, commonly referred to as the baud rate, to accurately interpret the data.

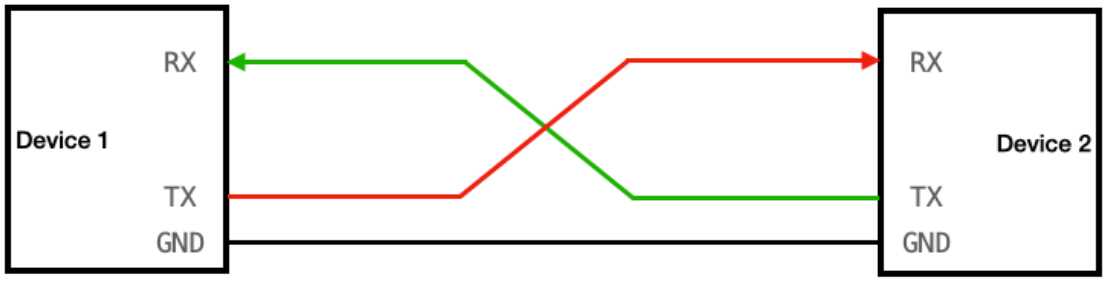

Figure 1: UART diagram.

2.1 UART Communication Protocol

In asynchronous communication, data is transmitted in frames with a specific structure. A typical UART frame consists of:

- **Start Bit:** A single bit (logic 0) that signals the beginning of a data frame.
- **Data Bits:** The main payload of the frame, usually configured for 7 or 8 bits.
- **Parity Bit (optional):** A bit used for basic error checking. In this assignment, only even parity is employed.
- **Stop Bit(s):** One or more bits (logic 1) that indicate the end of the data frame.

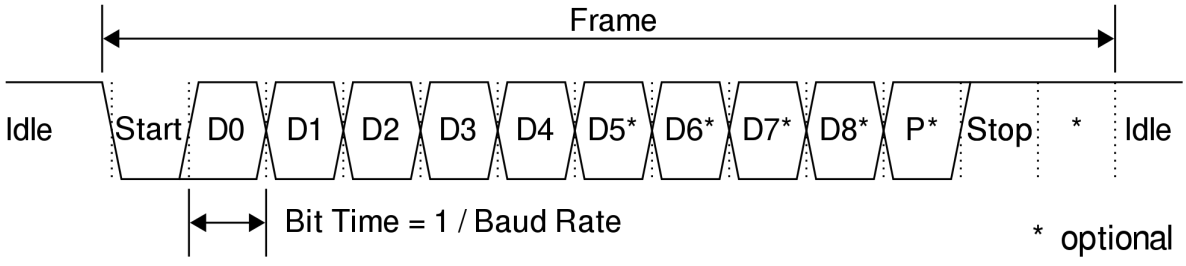

Figure 2: Example UART frame diagram.

The start and stop bits provide the necessary framing that allows the receiver to synchronize with the transmitter's data stream and accurately sample the incoming bits.

2.2 Baud Rate and Timing

The baud rate is defined as the number of signal changes (symbols) transmitted per second, and in the context of UART, it typically corresponds to the number of bits transmitted per second. Both the transmitter and receiver must operate at the same baud rate to ensure proper data interpretation. Any discrepancy in baud rate settings can lead to sampling errors, where the receiver may misinterpret the data. Therefore, accurate clock configuration and precise timing are critical to maintaining communication integrity.

2.3 Parity Bit (Even Parity)

The parity bit in this assignment is used for error detection and is generated based on even parity. This means that the parity bit is set such that the total number of 1s in the data bits, including the parity bit, is even. On the receiver side, the parity check is straightforward: the received data bits are XORed together with the parity bit. The result should be 0 if no error has occurred. A result of 1 indicates a discrepancy, signaling that an error has been detected during transmission. Depending on the design, upon detection of a parity error, the system can discard the erroneous frame, request a retransmission, or log an error for further handling.

3 Design and Implementation Details

This section describes the architecture and VHDL implementation of the UART. The design is modular, with separate processes for the transmitter and receiver to simplify simulation and debugging.

3.1 Architecture Overview

The overall architecture is divided into two main modules:

- **Transmitter (Tx):** Converts parallel data into a serial stream. A parallel-to-serial conversion is performed using a shift register and a counter that iterates through the bits. The transmitter appends a start bit, calculates and adds an even parity bit, and appends a stop bit. Registers are used to hold the data during the serialization process.
- **Receiver (Rx):** Detects the start bit and then uses a serial-to-parallel conversion to reconstruct the data. A shift register captures the serial data bits, and a counter ensures proper timing. The receiver performs an even parity check by XORing the received data bits with the transmitted parity bit. Upon detecting an error, appropriate measures such as flagging an error condition may be taken.

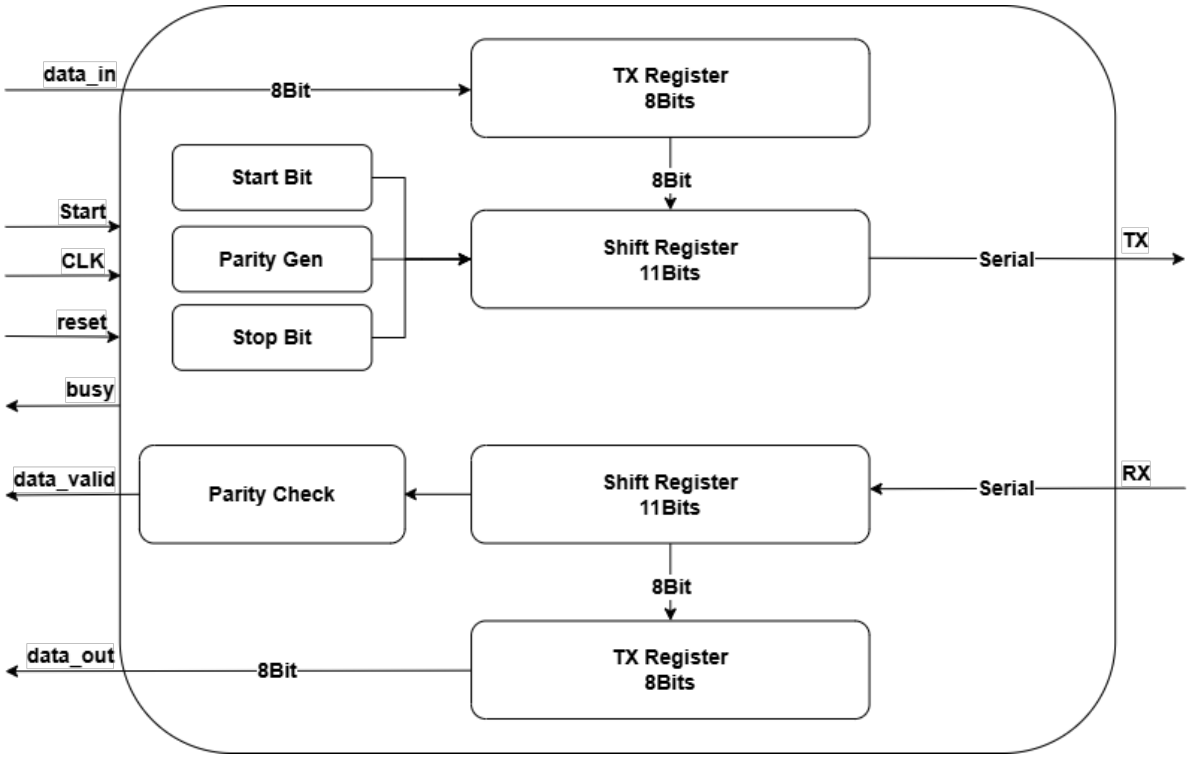

Figure 3: UART diagram.

Figure 3 illustrates the high-level architecture of the UART, including the data registers, shift registers, and control logic for both the transmitter and receiver.

3.2 VHDL Implementation

The VHDL code is structured with clear entity declarations and process-based architectures. The primary components include:

- **Entity Declaration:** Defines the input/output ports.
  - `clk`: System clock.
  - `reset`: Active-high reset signal.
  - `data_in`: 8-bit parallel input for transmission.
  - `tx_start`: Signal to initiate transmission.
  - `tx_out`: Serial output for the transmitter.
  - `busy`: Indicates when the transmitter is active.
  - `rx_in`: Serial input for the receiver.
  - `data_out`: 8-bit parallel output from the receiver.
  - `data_valid`: Indicates when valid data is available at the receiver.
- **Architecture:** Contains separate processes for the transmitter and receiver.
  - **Transmitter Process:** Implements a state machine with states such as IDLE, START, DATA, PARITY, and STOP. It utilizes a shift register for serialization and a counter for bit timing.
  - **Receiver Process:** Implements a state machine that detects the start bit, captures the incoming bits into a shift register, and performs the even parity check. It then reassembles the parallel data and asserts a `data_valid` signal.

3.3 Example Code Snippet

Below is an excerpt of the VHDL code with process skeletons for both the transmitter and receiver.

```
entity UART is
    Port (
        clk        : in  std_logic;
        reset      : in  std_logic;
        data_in    : in  std_logic_vector(7 downto 0);
        tx_start   : in  std_logic;
        tx_out     : out std_logic;
        busy       : out std_logic;
        rx_in      : in  std_logic;
        data_out   : out std_logic_vector(7 downto 0);
        data_valid : out std_logic
    );
end UART;

architecture Behavioral of UART is
    -- Common state definition for both transmitter and receiver
```

```
    type state_type is (IDLE, START, DATA, PARITY, STOP);

    -- Transmitter signals
    signal current_state_tx, next_state_tx : state_type;
    signal shift_reg_tx    : std_logic_vector(7 downto 0);
    signal bit_counter_tx  : integer range 0 to 7;
    signal parity_bit_tx   : std_logic;

    -- Receiver signals
    signal current_state_rx, next_state_rx : state_type;
    signal shift_reg_rx    : std_logic_vector(7 downto 0);
    signal bit_counter_rx  : integer range 0 to 7;
    signal parity_bit_rx   : std_logic;

begin

    -- Transmitter Process
    Tx_Process: process(clk, reset)
    begin
        if reset = '1' then
            current_state_tx <= IDLE;
            busy <= '0';
            tx_out <= '1'; -- Idle state is high
        elsif rising_edge(clk) then
            current_state_tx <= next_state_tx;
            -- Tx Logic
    end process;

    -- Receiver Process
    Rx_Process: process(clk, reset)
    begin
        if reset = '1' then
            current_state_rx <= IDLE;
            data_valid <= '0';
        elsif rising_edge(clk) then
            current_state_rx <= next_state_rx;
            -- Rx Logic
    end process;

end Behavioral;
```

4 Simulation Setup

This simulation is designed to test the complete UART system by internally connecting the transmitter and receiver via a loopback connection. The testbench instantiates the integrated UART module and uses a direct connection from the transmitter output (`tx_out`) to the receiver input (`rx_in`) to facilitate the complete data flow. As shown in Figure 4, the loopback setup allows the observation of the data flow from `data_in`, through the transmission channel, to `data_out`, while monitoring status signals such as `busy` and `data_valid`.

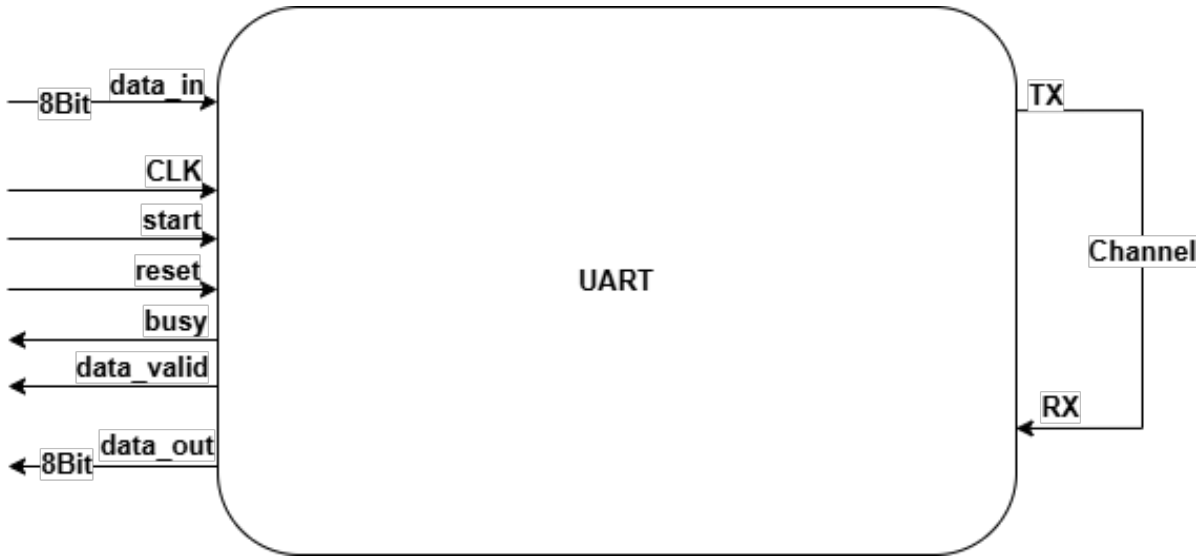

Figure 4: Testbench diagram showing the UART module instantiation and loopback connection.

Additionally, Figure 5 shows the simulation results from ModelSim under normal operation. In these results, students can observe the assertion of the start flag and the busy signal during transmission. The data channel waveform clearly depicts the start bit, the subsequent data bits, the parity bit, and the stop bit. Since no parity errors are detected during normal operation, the parity error indicator remains low; it would only go high in the event of a fault in the transmission.

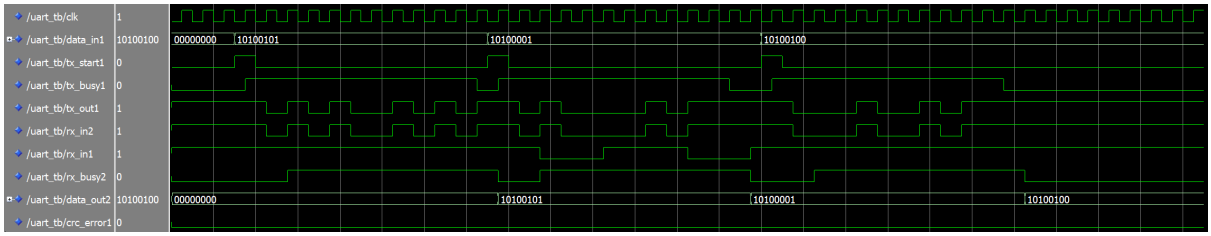

Figure 5: ModelSim simulation results of the UART module in normal operation.

Additionally, the testbench includes an optional section (commented out) that allows the simulation of a faulty transmission by directly overriding the transmission channel and injecting an incorrect parity bit. This can be used to verify that the UART module correctly detects errors. Note that the implementation of parity error handling is optional.

4.1 Testbench Example

Below is a complete testbench example that instantiates the integrated UART module, declares all necessary signals, and connects the transmitter and receiver using a loopback configuration:

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity UART_tb is
end UART_tb;

architecture test of UART_tb is
    -- Signal declarations
    signal clk         : std_logic := '0';
    signal reset       : std_logic := '0';
    signal data_in     : std_logic_vector(7 downto 0) := (others => '0');
    signal tx_start    : std_logic := '0';
    signal tx_out      : std_logic;
    signal busy        : std_logic;
    signal rx_in       : std_logic;
    signal data_out    : std_logic_vector(7 downto 0);
    signal data_valid  : std_logic;
begin

    -- Loopback connection: Connect TX output directly to RX input
    rx_in <= tx_out;

    -- Instantiate the integrated UART module
    UUT: entity work.UART
        port map (
            clk        => clk,
            reset      => reset,
            data_in    => data_in,
            tx_start   => tx_start,
            tx_out     => tx_out,
            busy       => busy,
            rx_in      => rx_in,
            data_out   => data_out,
            data_valid => data_valid
        );

    -- Clock generation process
    clk_gen : process
    begin
        clk <= '0';
        wait for 10 ns;
        clk <= '1';
```

```
        wait for 10 ns;
    end process;

    -- Stimulus process
    stim_proc : process
    begin
        -- Apply reset
        reset <= '1';
        wait for 20 ns;
        reset <= '0';
        wait for 20 ns;

        -- Normal transmission: drive data_in and trigger transmission
        data_in <= "10101010";
        tx_start <= '1';
        wait for 20 ns;
        tx_start <= '0';

        -- Wait for transmission and reception to complete
        wait for 200 ns;

        -- Optional: simulate a faulty transmission by overriding
        -- the loopback channel and sending a full transmission frame
        -- with a wrong parity check, for example:
        -- 0 (start bit)
        -- 11111111 (data bits)
        -- 1 (parity bit) -> should be 0 for even parity if no error
        -- 1 (stop bit)
        % rx_in <= '0';
        % wait for 20 ns;
        % rx_in <= '1';
        % etc...

        wait;
    end process;

end test;
```

5 Conclusion

This assignment required the implementation of a UART module in VHDL, featuring separate transmitter and receiver processes. The design must correctly handle asynchronous communication by framing data with a start bit, 8 data bits, an even parity bit, and a stop bit, while ensuring proper timing through a specified baud rate and accurate parity error detection.

The simulation setup, which uses a loopback connection, demonstrates the complete data flow from `data_in` to `data_out`, with status signals such as `busy` and `data_valid` indicating system operation. The ModelSim simulation results should clearly display the start bit, data bits, parity bit, and stop bit, with the parity error flag remaining low during normal operation (it would only be high if a fault is detected).

Assignment submissions must be delivered online through the designated platform. The submission should include all VHDL source files, the testbench script, and a report that contains a figure of the simulation results and a detailed explanation of the results obtained.