Abdelrahman Mohamed Ali

Senior 1, Faculty of Engineering Ain Shams University

Electronics and Communication Department

UART RTL Implementation using SystemVerilog

Project url: LINK

LinkedIn	**Abdelrahman Mohamed**
GitHub	**Abdelrahman1810**
codeforces	**Salby1810**

UART Project Report

1. Overview

Introduction

This report outlines the design, implementation, and testing of a Universal Asynchronous Receiver-Transmitter (UART) project using SystemVerilog.

Objectives

The primary objectives of this project are:

1. Design and implement a UART module that can transmit and receive data serially.

2. Support multiple baud rates (4800, 9600, 57600, 115200) configurable through preprocessor macros.

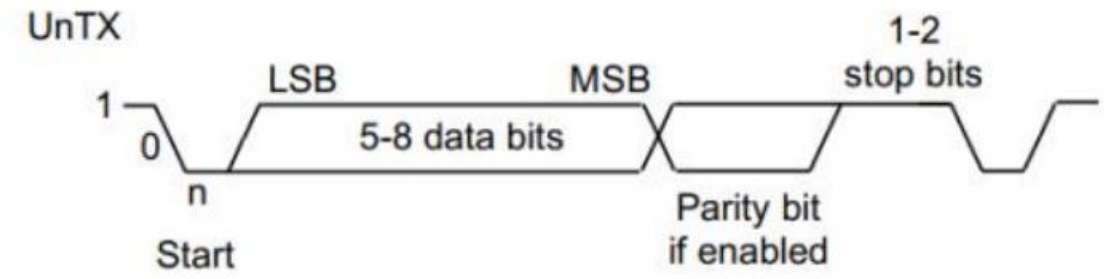
3. Provide a testbench to verify the functionality of the UART module.

Baud Rate Calculation

• BRD = IBRD + FBRD

• BRD = UARTSysClk / (ClkDiv * Baud Rate)

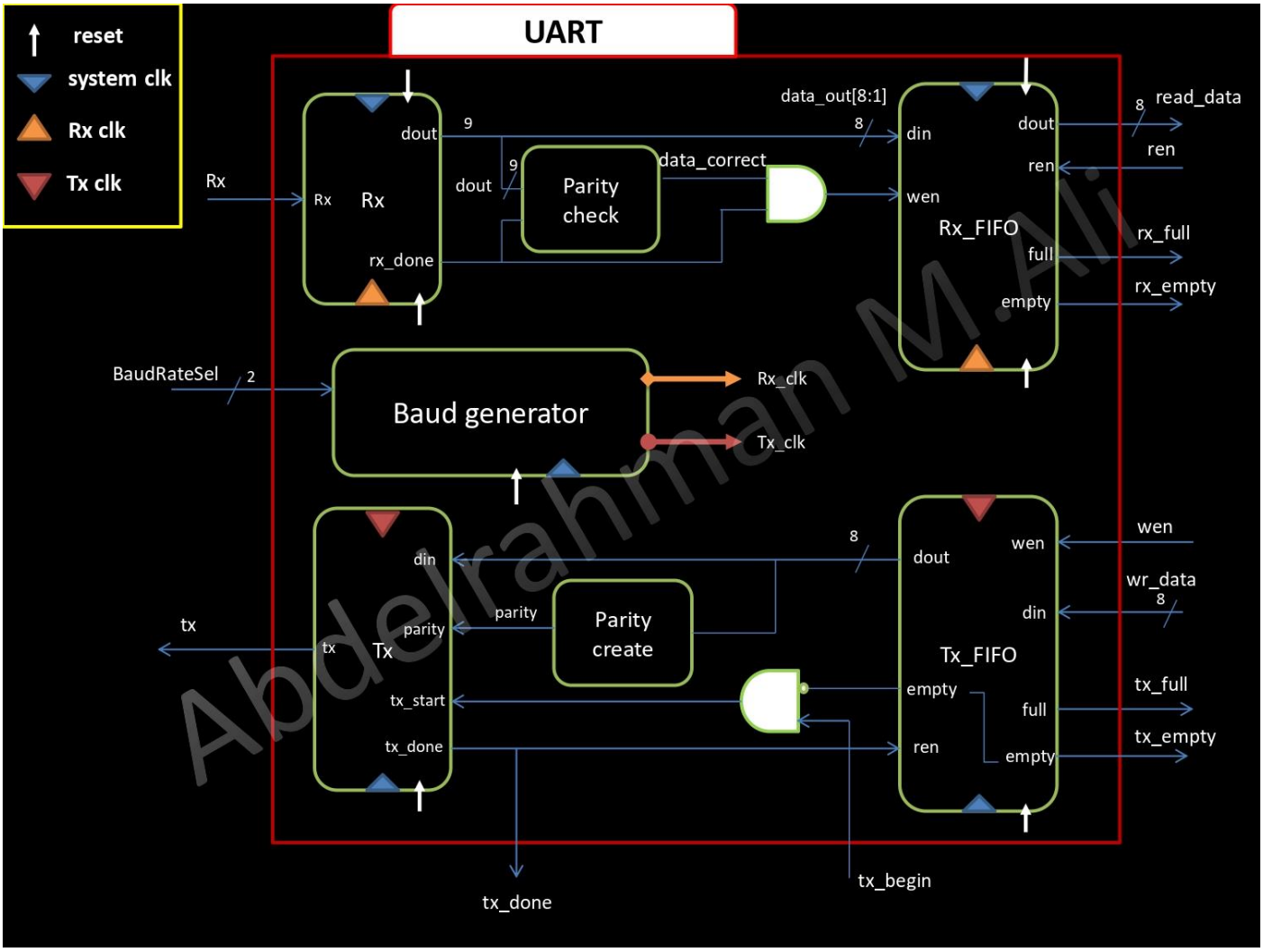
UART Protocol (*Tx*)



In my design I use even parity and one stop bit

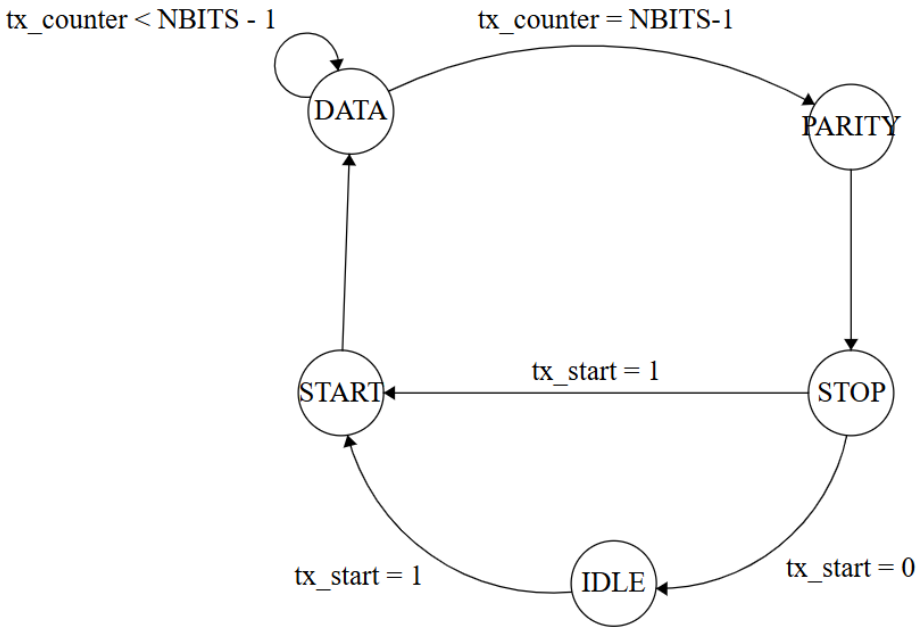
2. Implementation

Design Architecture

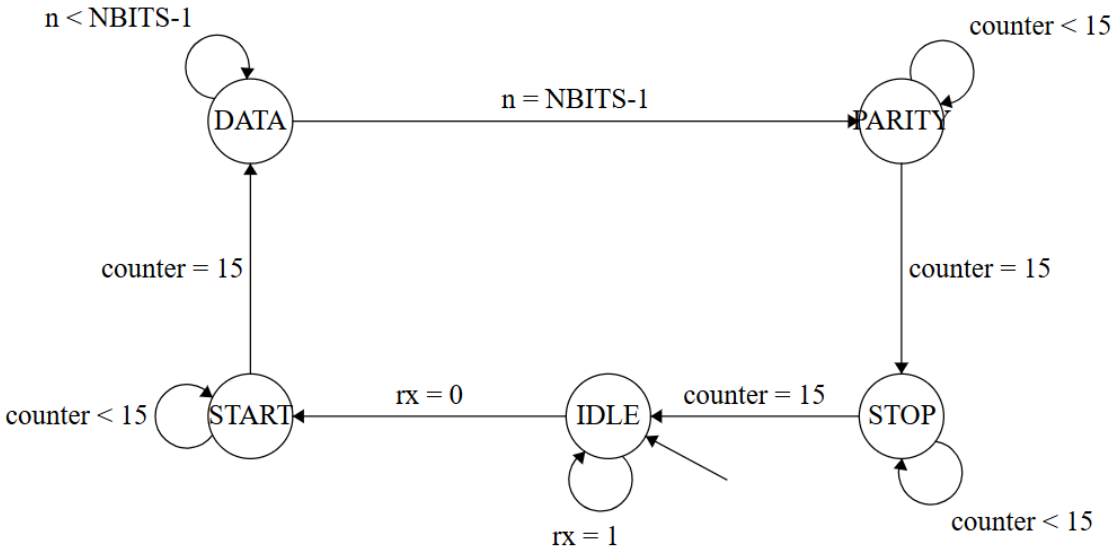


FSM

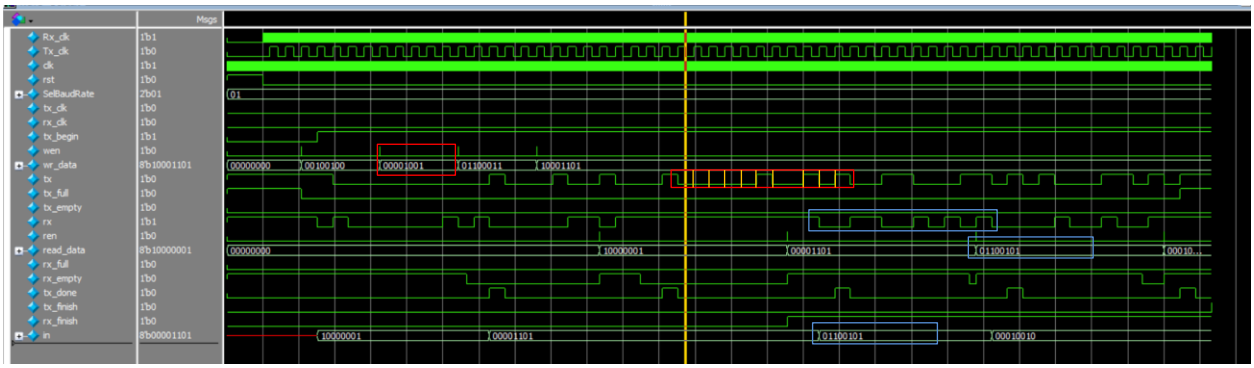
a. Tx


tx_counter < NBITS - 1
tx_counter = NBITS-1
DATA
PARITY
START
tx_start = 1
STOP
tx_start = 1
IDLE
tx_start = 0

b. Rx


n < NBITS-1
counter < 15
DATA
n = NBITS-1
PARITY
counter = 15
counter = 15
counter < 15
START
rx = 0
IDLE
counter = 15
STOP
rx = 1
counter < 15

Snippets



Baud Rate

Baud 4800

```
# ** Note: $finish    : uart_tb.sv(62)
#    Time: 26000100 ns  Iteration: 4  Instance: /uart_tb
# 1
# Break in Module uart_tb at uart_tb.sv line 62
```

Baud 9600

```
# ** Note: $finish    : uart_tb.sv(62)
#    Time: 13657900 ns  Iteration: 4  Instance: /uart_tb
# 1
# Break in Module uart_tb at uart_tb.sv line 62
```

Baud 57600

```
#    Time: 3612100 ns  Iteration: 4  Instance: /uart_tb
# 1
# Break in Module uart_tb at uart_tb.sv line 62
```

Baud 115200

```
# ** Note: $finish    : uart_tb.sv(62)
#    Time: 2508700 ns  Iteration: 4  Instance: /uart_tb
# 1
# Break in Module uart_tb at uart_tb.sv line 62
```

Design Overview

The UART project consists of the following top-level modules:

1. `tx_unit`: Responsible for transmitting data serially.

2. `rx_unit`: Responsible for receiving data serially.

3. `uart `: The top-level module that integrates the transmitter and receiver modules.

`tx_unit` Module

The `tx_unit` module is responsible for transmitting data serially. It takes in a parallel data input, a start signal, and a baud rate selection, and converts the data into a serial bit stream with start and stop bits.

The `tx_unit` module uses a finite state machine (FSM) to control the transmission process. The FSM has the following states:

- `IDLE`: Waiting for the start signal to begin transmission.

- `START`: Transmitting the start bit.

- `DATA`: Transmitting the data bits.

- `PARITY`: Transmitting parity bit

- `STOP`: Transmitting the stop bit.

`rx_unit` Module

The `rx_unit` module is responsible for receiving data serially. It takes in the serial data input, the baud rate selection, and outputs the received parallel data.

The `rx_unit` module also uses a finite state machine (FSM) to control the reception process. The FSM has the following states:

- `IDLE`: Waiting for the start bit.

- `START`: Detecting the start bit.

- `DATA`: Receiving the data bits.

- `STOP`: Receiving the stop bit.

3. Testbench

Introduction

This report outlines the testbench for the UART (Universal Asynchronous Receiver-Transmitter) module. The testbench is designed to verify the functionality of the UART module by generating various input scenarios and monitoring the corresponding outputs.

Testbench Structure

The testbench, named `uart_tb()`, instantiates the UART module and defines the necessary parameters and signals. The key components of the testbench are:

1. Parameters:

- `OVERSAMBLE`: The number of samples per bit during the reception process.
- `CLK_FREQ`: The frequency of the system clock.
- `FIFO_WIDTH`: The width of the FIFO (First-In-First-Out) buffer.
- `FIFO_DEPTH`: The depth of the FIFO buffer.
- `NBITS`: The total number of bits, including the data and parity bits.

2. Signals:

- `clk`, `rst`: The system clock and reset signals.
- `SelBaudRate`: The selection signal for the baud rate.
- `tx_begin`, `wen`, `wr_data`: The transmit control signals and write data.
- `tx`, `tx_full`, `tx_empty`: The transmit output signals.
- `rx`, `ren`: The receive control signals.
- `read_data`, `rx_full`, `rx_empty`: The receive output signals.

Testbench Functionality

The testbench includes the following test scenarios:

1. Reset Sequence:
   - The reset signal (`rst`) is asserted for a certain duration to initialize the UART module.

2. Transmit Testbench:
   - Random data is written to the transmit FIFO buffer.
   - The transmit process is initiated, and the transmitted data is monitored.

3. Receive Testbench:
   - Random data is injected into the receive input (`rx`) to simulate the reception of data.
   - The received data is read from the receive FIFO buffer and verified.

The testbench also includes tasks and initial blocks to handle the various test scenarios.

Simulation and Verification

The testbench can be simulated using different baud rate configurations, which are defined using preprocessor macros (`BRD48`, `BRD96`, `BRD57`, `BRD11`). The simulation results can be used to verify the functionality of the UART module and ensure that it operates correctly under various input conditions.

The UART project was implemented using SystemVerilog, and the functionality was verified using a comprehensive testbench.

Conclusion

The UART project was successfully designed, implemented, and tested using SystemVerilog. The modular approach and the use of finite state machines allowed for a clean and efficient implementation. The testbench ensured the overall functionality and robustness of the UART design.

This UART project can be used as a building block for larger systems that require serial communication capabilities.