Verilog Based UART System

USER MANUAL

SYED SHIFAT

Table of Contents

Introduction

This project implements a fully synthesizable UART (Universal Asynchronous Receiver/Transmitter) in Verilog RTL. The system is designed to support 8-bit asynchronous communication with configurable baud rates and uses 16x oversampling for accurate reception.

System Overview

The RTL UART system includes the following core modules:

- **Baud Rate Generator:** Produces 1x and 16x baud ticks.
- **Counter & Comparator:** Drives the timing logic for baud control.
- **UART Transmitter (TX):** Sends 8-bit data serially.
- **UART Receiver (RX):** Receives serial data and reconstructs 8-bit parallel output.
- **Top Module:** Integrates all components.

Features

- 8-bit UART Transmitter and Receiver
- Configurable baud rate via 16-bit divisor input
- 16x oversampling for reliable UART reception
- Modular and testbench-covered design
- Easy simulation with provided Makefile targets

File Structure and Descriptions

```
RTL_UART/
├── baud_gen/
│   ├── baud_gen.v          # Baud rate generator module
│   └── tb_baud.v           # Testbench for baud generator
├── count_comp/
│   ├── comparator.v        # Comparator module for baud timing pulses
│   └── counter.v           # 16-bit counter module
├── Rx_/
│   ├── uart_rx.v           # UART Receiver module
│   └── tb_rx.v             # Testbench for UART Receiver
├── Tx_/
│   ├── uart_tx.v           # UART Transmitter module
│   └── tb_tx.v             # Testbench for UART Transmitter
├── Top_/
│   ├── top.v               # Top-level UART integration module
│   └── tb_top.v            # Full UART system testbench
├── Makefile                # Simulation and waveform helper commands
└── README.md               # Project documentation
```

Input output of top module

Signal	Direction	Description
clk	Input	System clock
rst_n	Input	Active-low reset
data_in	Input	8-bit input data for TX
start_tx	Input	Trigger signal to start TX
tx_line	Output	UART transmit output
rx_line	Input	UART receive input
data_received	Output	Received 8-bit parallel data
tx_busy	Output	TX active indicator
rx_done	Output	RX complete flag
baud_div	Input	16-bit divisor for baud rate

Block Diagram

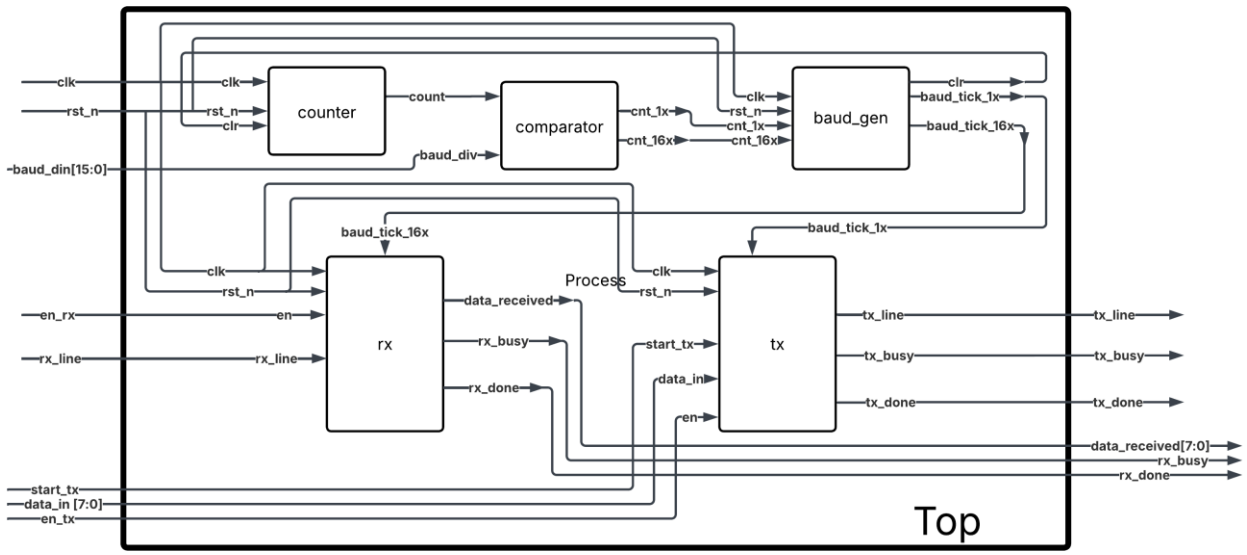


Fig 1. Block Diagram

Working Procedure

1. Clock and Reset Initialization: The system is driven by a clock input (clk) and an active-low reset (rst_n). Reset initializes all modules to their default states.

2. Baud Rate Generation: The 16-bit counter increments each clock cycle until cleared. The comparator compares this count to baud_div to generate timing pulses cnt_1x and cnt_16x. These pulses feed into baud_gen, which creates precise baud ticks:

- baud_tick_1x for transmitter timing
- baud_tick_16x for receiver oversampling timing

3. Transmission Flow: When enabled (en_tx) and triggered (start_tx), the UART transmitter (uart_tx) takes 8-bit parallel data (data_in), adds start and stop bits, and serializes the bits onto tx_line. The process is paced by baud_tick_1x. Transmitter status flags tx_busy and tx_done indicate transmission progress.

4. Reception Flow: When enabled (en_rx), the UART receiver (uart_rx) monitors the rx_line for start bits. Using the baud_tick_16x oversampled clock, it samples incoming bits, reconstructs the 8-bit data, and sets rx_done upon completion. The received byte is available on data_received. Receiver status is shown by rx_busy.

5. Top-Level Integration: The top module ties all these pieces together, allowing simultaneous transmission and reception with configurable baud rate.

FSM State Diagram

RX_FSM

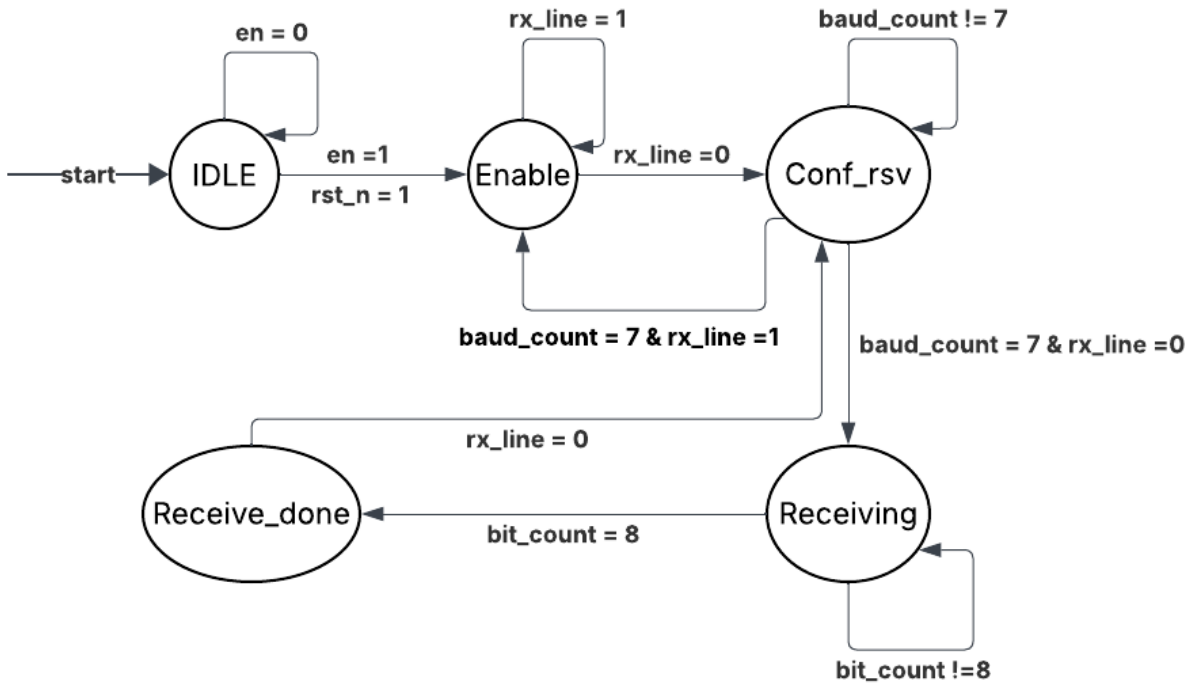


Fig 2. RX_FSM

Tx_FSM

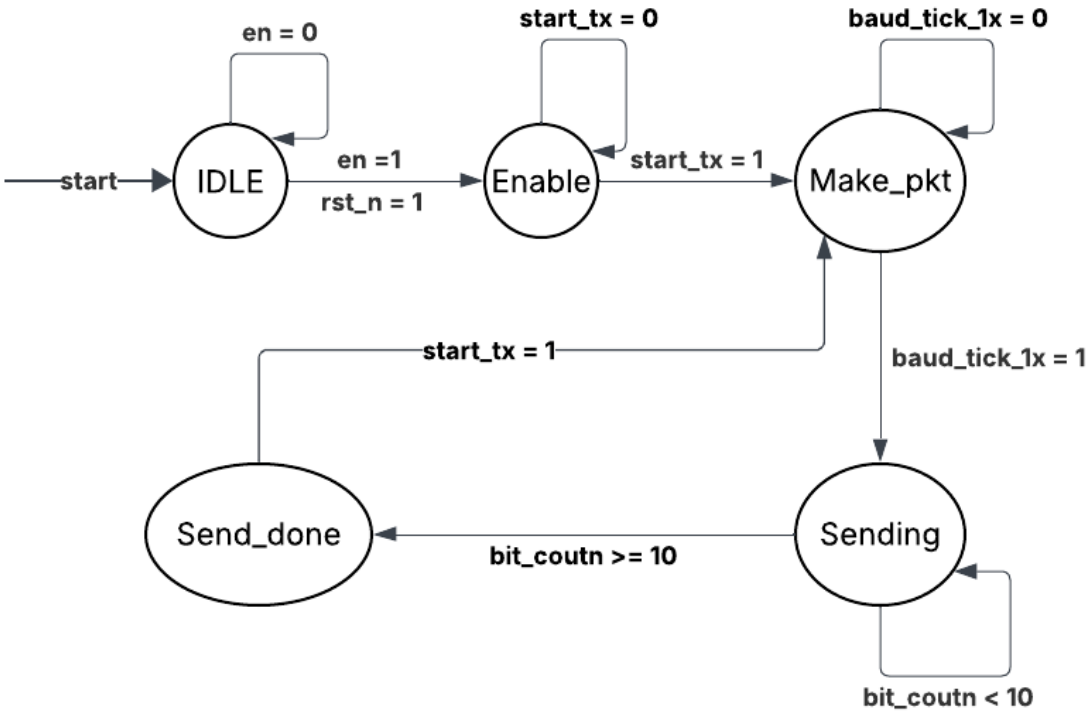


Fig 3. TX_FSM

How to Use

Setting Up the Simulation

Requirments:

- Icarus Verilog (iverilog, vvp)
- GTKWave

Clone and navigate to the project directory:

```
>git clone https://github.com/yourusername/RTL_UART.git
>cd RTL_UART
```

Running the UART System

Use the Makefile to run simulations:

```
>make tx      # Simulate transmitter
>make g_tx    # View transmitter waveform

>make rx      # Simulate receiver
>make g_rx    # View receiver waveform
```

```
>make top     # Simulate full UART system
>make g_top   # View full UART waveform
```

Interpreting GTKWave

Observe the following signals:

- **tx_line:** Serialized data bits (start, data[7:0], stop)
- **tx_busy, tx_done:** Transmission state
- **rx_line:** Serial input
- **rx_done, data_received:** Reception result

Waveform Examples

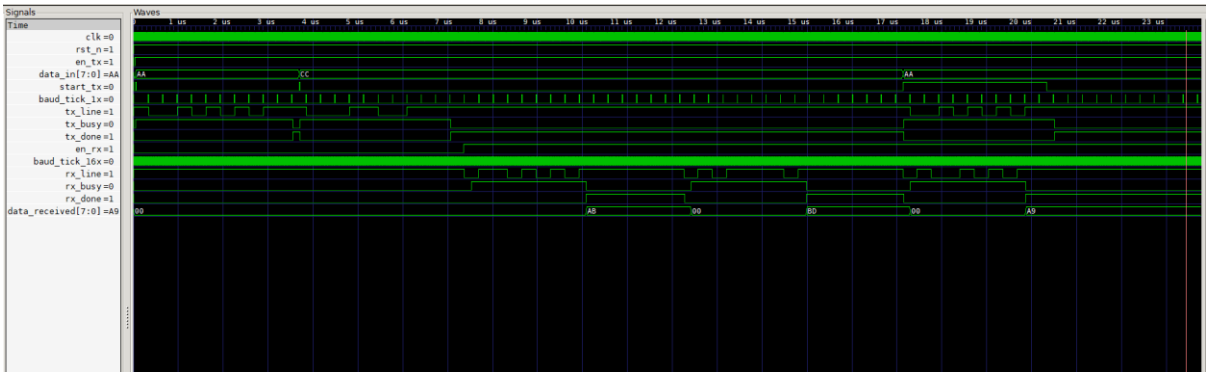
Include screenshots from GTKWave showing:

- A full TX transmission (with start bit, 8 data bits, stop bit)
- RX reconstruction with sampled ticks
- Data match confirmation

Label:

- Start bit, data bits, stop bit
- rx_done, data_received

Sample output:



Limitations

- Only transmits and receives one byte at a time.
- No FIFO buffering.
- No parity or framing error detection.
- Receiver assumes clean and noise-free line.

Troubleshooting

Issue	Solution
rx_done never asserted	Check baud_tick_16x timing and RX line logic
data_received incorrect	Confirm start bit detection is stable
GTKWave not updating	Ensure vcd dump is triggered in testbench