**UART in HDL**

User Manual

Syed Shifat

**Table of Contents**

[Introduction 2](#_Toc203675748)

* [System Overview 2](#_Toc203675749)
* [Features 2](#_Toc203675750)
* [File Structure and Descriptions 2](#_Toc203675751)
* [Input output of top module 3](#_Toc203675752)
* [Block Diagram 3](#_Toc203675753)
* [Working Procedure 3](#_Toc203675754)
* [FSM State Diagram 4](#_Toc203675755)
  + [RX\_FSM 4](#_Toc203675756)
  + [Tx\_FSM 5](#_Toc203675757)
* [How to Use 5](#_Toc203675758)
  + [Setting Up the Simulation 5](#_Toc203675759)
  + [Running the UART System 5](#_Toc203675760)
  + [Interpreting GTKWave 6](#_Toc203675761)
* [Waveform Examples 6](#_Toc203675762)
* [Limitations 6](#_Toc203675763)
* [Troubleshooting 7](#_Toc203675764)

# 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

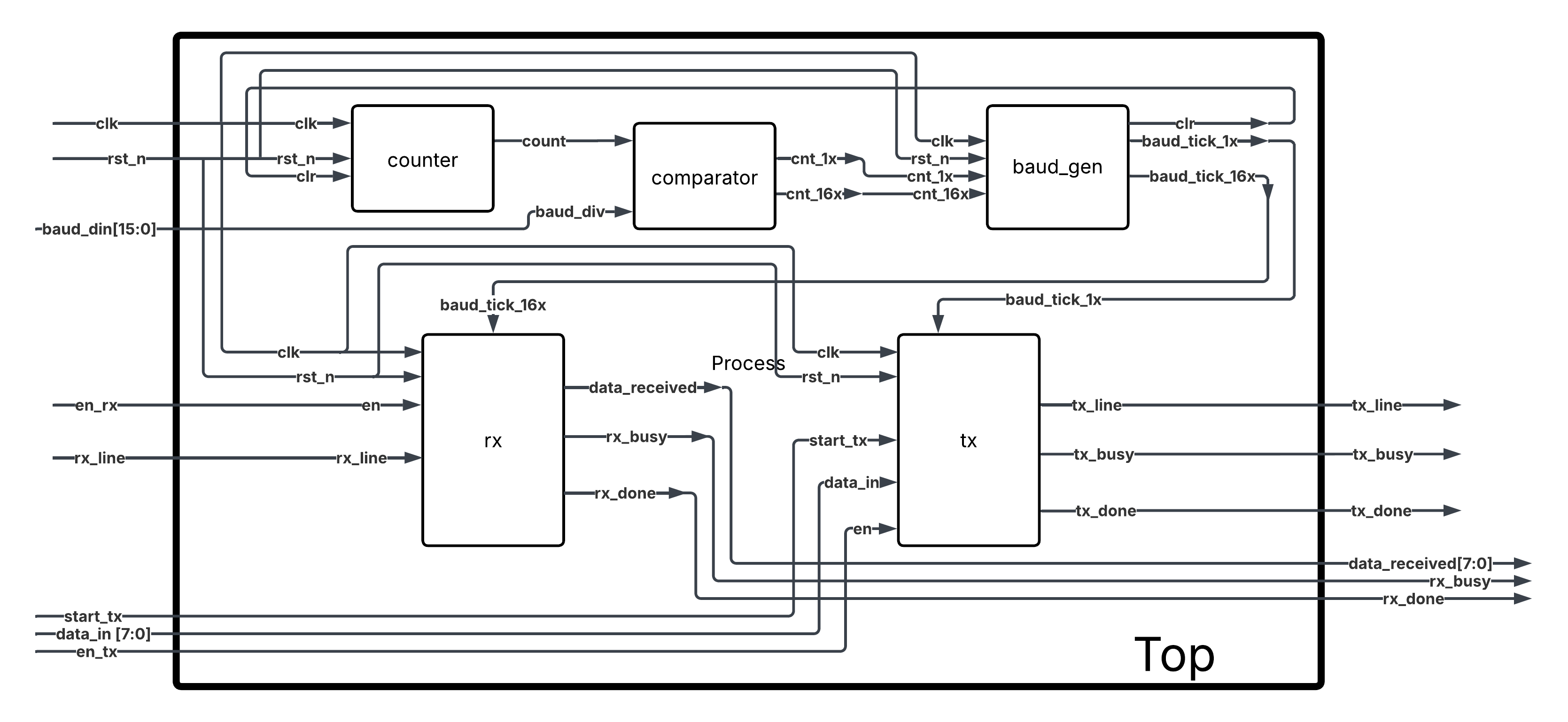


Fig 1. Block Diagram

# 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

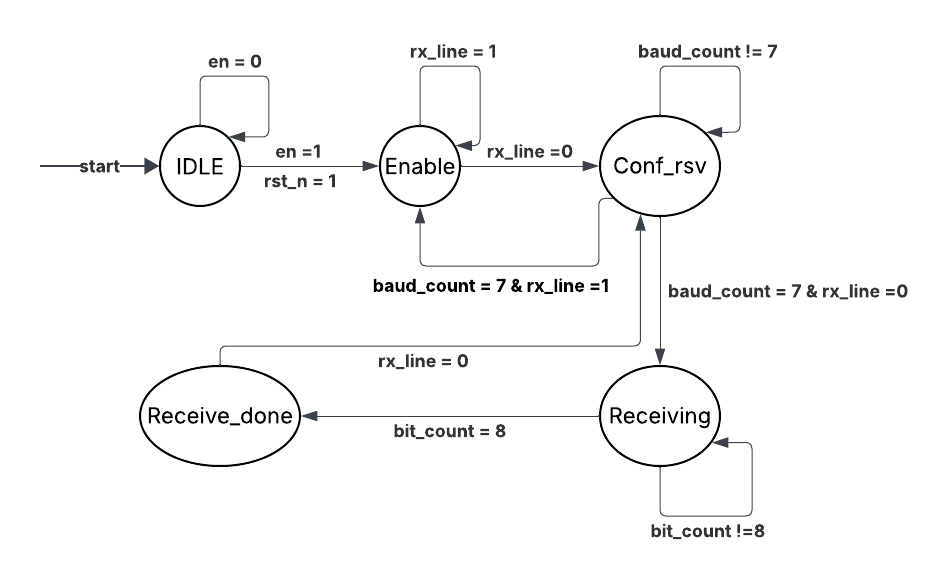


Fig 2. RX\_FSM

## Tx\_FSM

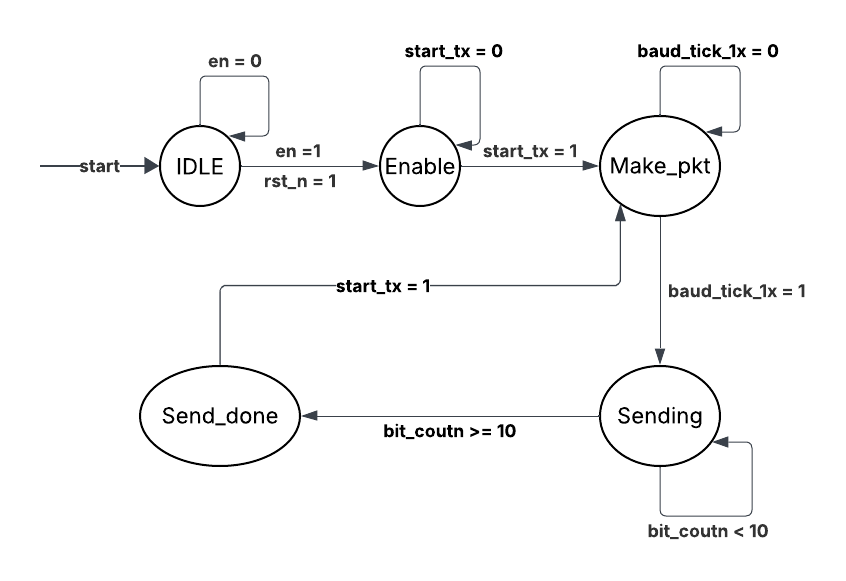


Fig 3. TX\_FSM

# 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

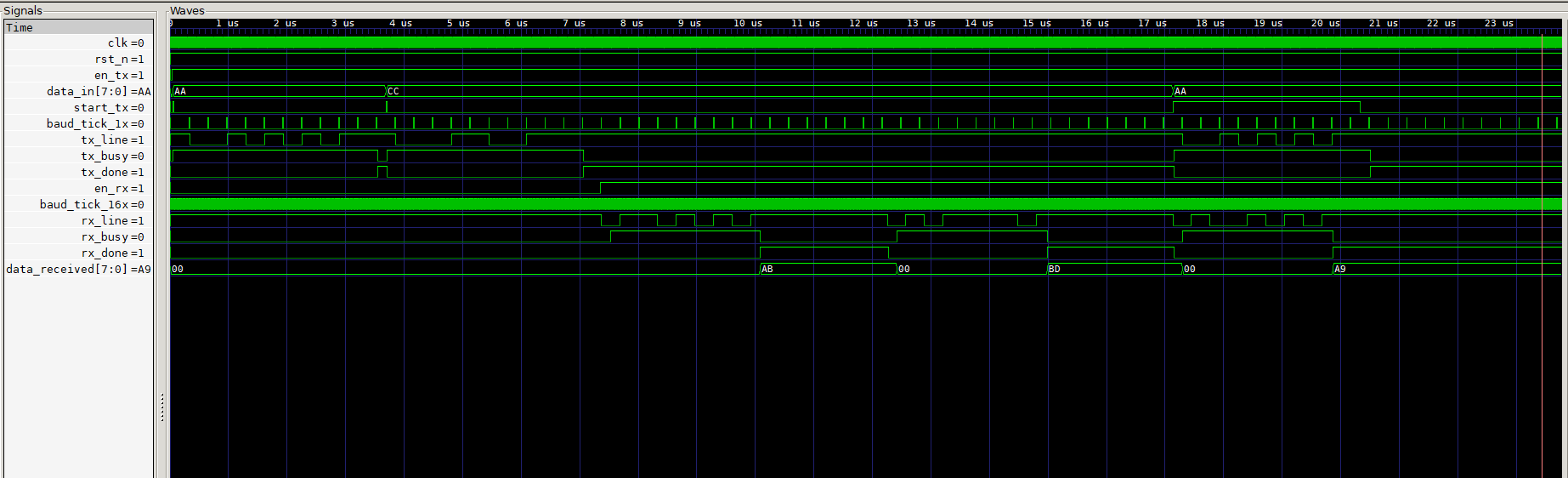
Include screenshots from GTKWave showing:

* 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 |