UART TRANSMITTER AND RECEIVER DESIGN USING SYSTEMVERILOG

INTRODUCTION

- UART allows communication without a shared clock between devices, enabling flexible timing between sender and receiver.
- It transmits data serially bit-by-bit over a single line, making it widely used in microcontrollers, GPS modules, and Bluetooth due to its simplicity and low resource requirements.
- UART data frame is used to structure serial data.
- Standard frame includes:
- 1 Start Bit – signals beginning of transmission (always 0).
- 8 Data Bits – actual information being transmitted.
- Parity Bit – for simple error checking.
- 1 Stop Bit – indicates end of transmission (always 1).
- Data is sent LSB (Least Significant Bit) first.
- Baud rate determines bit transmission speed (e.g., 9600 bps = 9600 bits/sec).
- Timing is crucial due to the lack of a shared clock.

SYSTEM DESIGN OVERVIEW

Main components of the design:

- UART Transmitter (uart_tx)
- UART Receiver (uart_rx)
- Baud Rate Generator – derives the desired baud rate from 100 MHz system clock

Baud Rate Generation:

- Baud Rate Generator is used to convert the 100 MHz system clock into a lower rate suitable for UART transmission.
- For example, to achieve 9600 baud, we divide the clock:

$$\text{Clock Cycles per Bit} = \frac{100,000,000}{9600} \approx 10416 \text{ cycles}$$

Transmitter Block:

- Sends serial data by adding start, data, and stop bits
- Waits for tx_start signal to begin transmission

Receiver Block:

- Monitors the RX line for a start bit
- Samples incoming bits at the center of each bit period
- Assembles received bits into a byte and asserts data_valid

SYSTEMVERILOG IMPLEMENTATION

- The UART system was implemented using System Verilog HDL, following modular and FSM-based design principles.

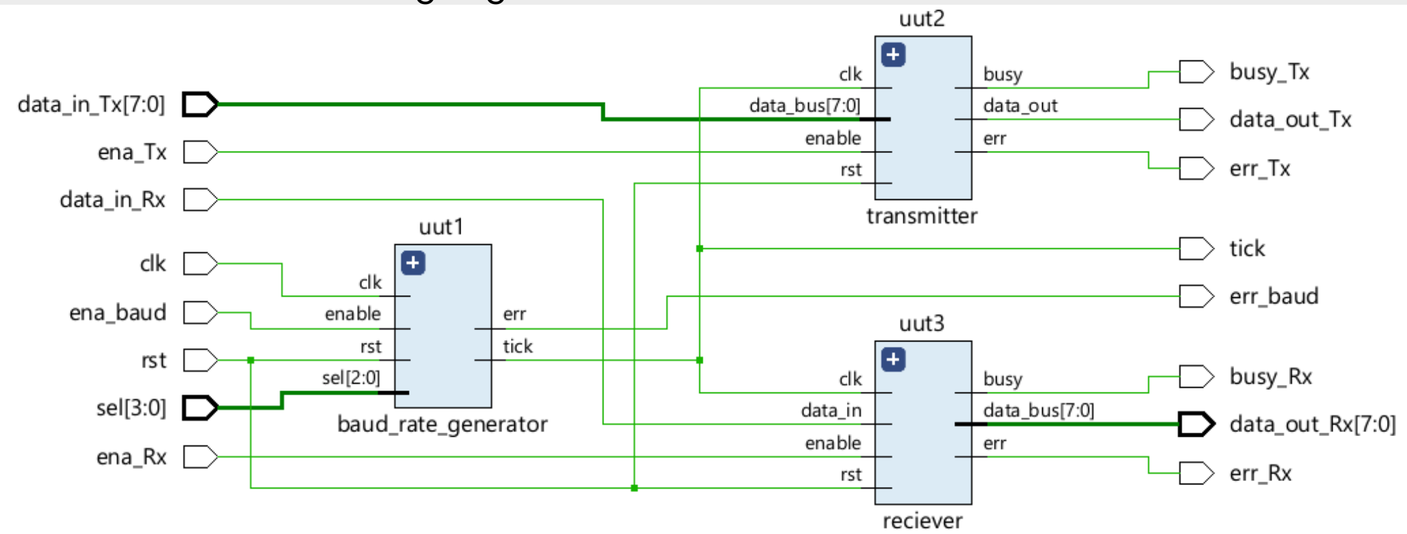
Modules Implemented:

- **transmitter.sv (Transmitter):**
  1. Takes 8-bit parallel data input and converts it to serial.
  2. Adds start and stop bits around the data.
  3. Controlled by a finite state machine (FSM).
  4. Operates based on baud tick from Baud Rate Generator.

- **reciever.sv (Receiver):**
  1. Listens for start bit on the RX line.
  2. Samples bits at mid-bit intervals for accuracy.
  3. Assembles 8-bit data and asserts data valid when complete.
  4. Includes FSM for state control and timing alignment.

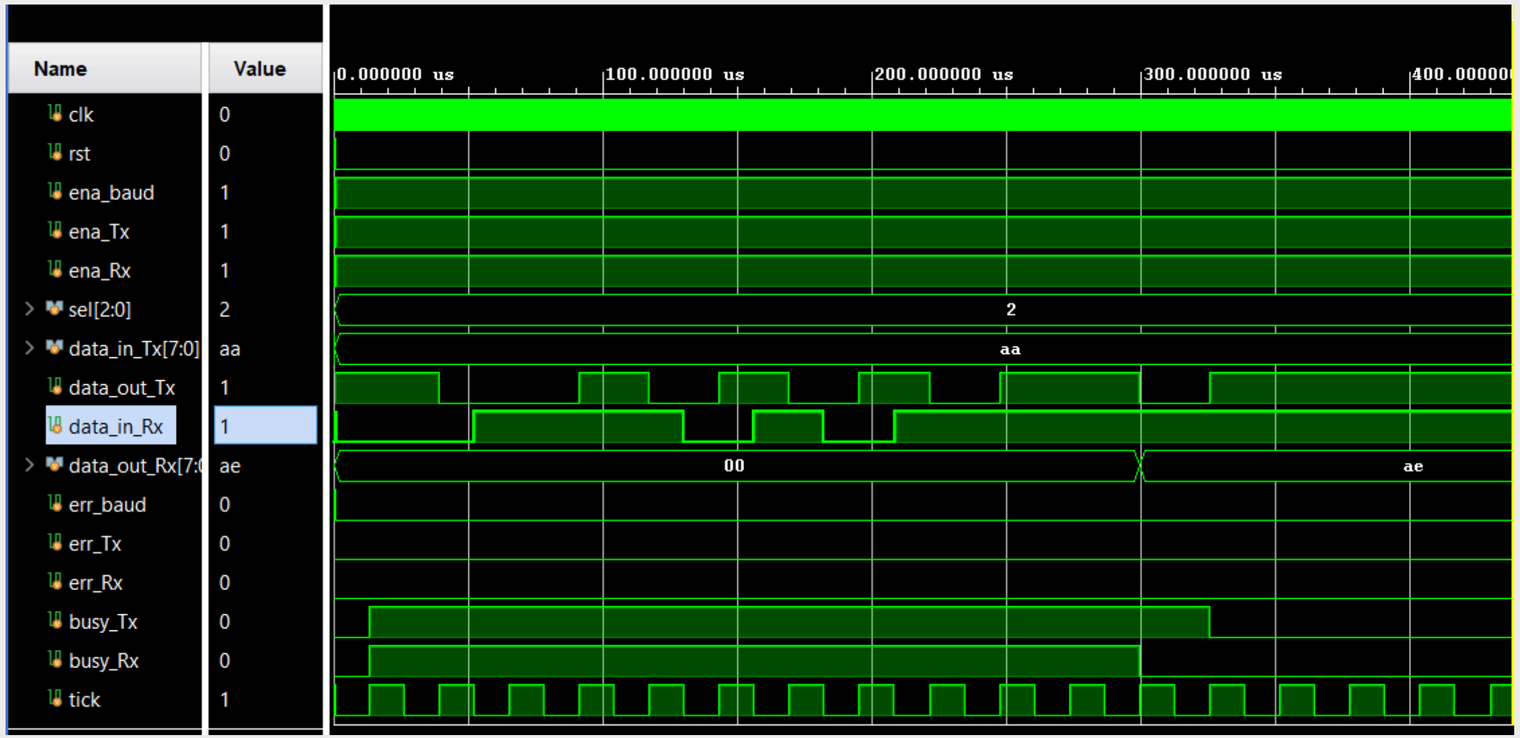
- **baud_rate_generator.sv (Baud Rate Generator):**
  1. Divides 100 MHz clock to generate baud tick (e.g., for 9600 bps).
  2. Provides precise timing pulses for TX and RX modules.

- **testbench.sv:**
  1. Simulates full UART communication.
  2. Provides stimulus to TX and observes RX output.
  3. Uses $display/$monitor and waveforms to validate correctness.



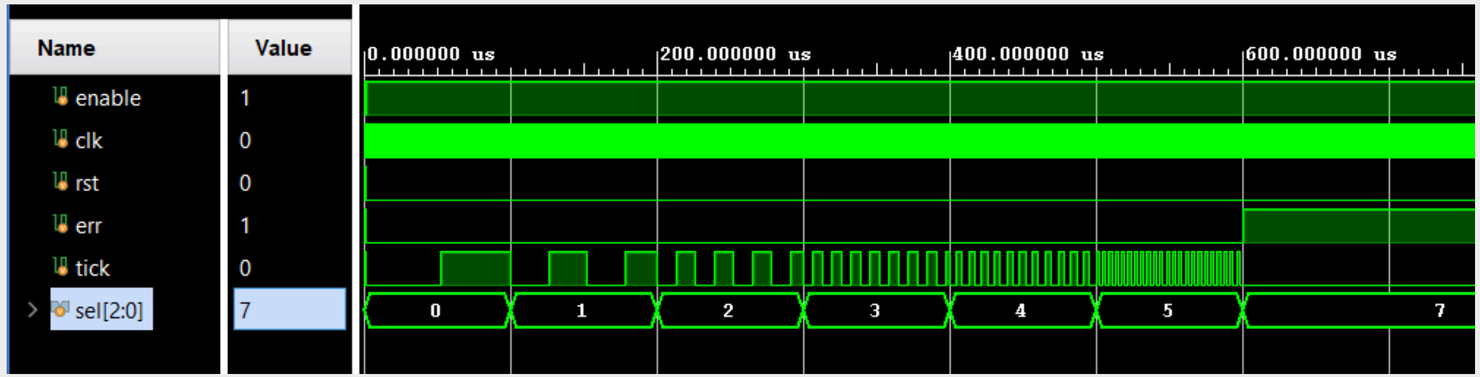
SIMULATION RESULT

UART Transmitter–Receiver Parallel Operation & Busy Status Waveform



- TX and RX work in parallel
- Each module operates independently
- Shows busy when transmitting or receiving
- No data loss during concurrent operation

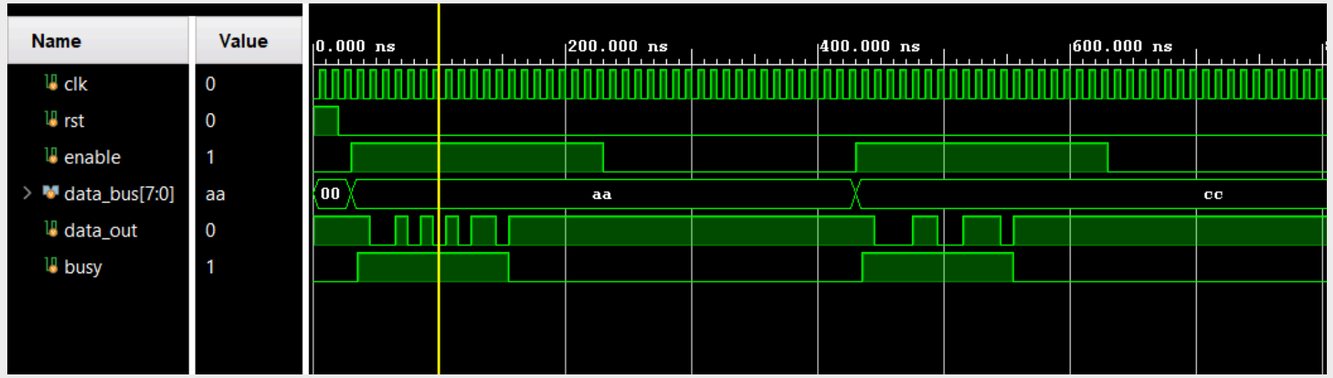
Baud Rate Generator Operation & Error Handling



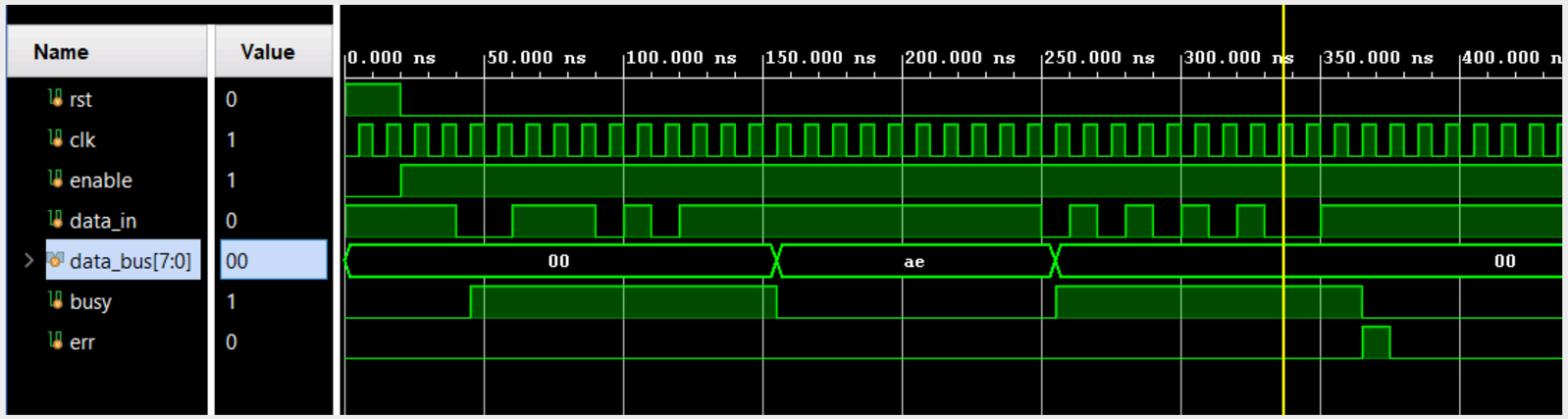
- Supports selection of 6 baud rates
- Generates clock for UART timing
- Error indicated on incorrect baud rate selection
- Ensures synchronization with TX/RX modules

UART Transmitter Waveform: Operation & Control Signals

- Shows busy status during transmission
- Prevents repeated data sending when enable is held high
- Transmits data serially with start, data, and stop bits



UART Receiver Waveform: Data Capture & Error Detection



- Shows busy during data reception
- Detects and flags parity error on mismatch
- Captures serial data with proper framing (start, data, parity, stop)

CONCLUSION

- Successfully implemented a UART communication system using System Verilog.
- Designed modular blocks for Transmitter, Receiver, and Baud Rate Generator.
- Achieved correct serial data transmission and reception at 9600 baud using a 100 MHz system clock.
- Gained practical experience in:

1. Finite State Machine (FSM) design
2. Timing control and synchronization
3. Bit-level serial communication

- Validated functionality through testbenches and waveform simulations.
- Developed better understanding of UART protocol and hardware modeling in HDL.

THANK YOU