Universal Asynchronous Receiver Transmitter UART

Konstantinos Seraskeris

November 2024

1 Summary

This project involves the implementation of a **serial communication system** using the **UART (Universal Asynchronous Receiver-Transmitter) protocol**. The system includes a **Transmitter**, which sends bits one by one, and a **Receiver**, which receives them. To ensure smooth communication, both must agree on a common sampling rate, defined in **Baud (bits/sec)**.

2 Introduction

The goal was to develop a UART communication system which includes the Baud Controller (for transmission speed control), the Transmitter and the Receiver.

Specifically, the objective of **Part A** is the implementation of a **signal generator for transmission speed**. The module was implemented using a **counter** and a **multiplexer**. The different baud rate values were calculated with precision, as shown later in the report, ensuring the **minimum possible error** in selecting clock cycles for sampling.

Then, in **Part B**, the **Transmitter** was developed, which transmits the data serially according to the **UART protocol (Start, Parity, Stop)**. The transmitter was designed as a **Moore-type Finite State Machine (FSM)**, which includes states for channel idling, data transmission, and bit verification. By synchronizing the active cycles of the baud controller from Part A, **data transmission is successfully carried out** through the communication channel.

After that, in **Part C**, the **Receiver** is implemented. It **receives the bits one by one**, stores them, and raises the appropriate signals indicating the result of the communication. The receiver was also designed as an FSM to **detect the start of communication** with the transition of the signal to 0, **sample at the center of each bit**, and inform the system about the **success or failure of the transmission**.

Finally, in **Part D**, the **Transmitter** and **Receiver** are connected, forming the **complete UART circuit**. The system is tested using a testbench to ensure accuracy and consistency in sampling, and by extension, the reliability and correctness of the communication.

3 Part A - Baud Rate Controller

3.1 Implementation:

The Baud Rate Controller is used internally in the Transmitter and Receiver circuits. Its purpose is to provide the appropriate sampling signal according to the selected Baud Rate. The sampling signal remains positively active for one cycle.

Below are the FPGA clock cycles (100MHz/10ns) for each Baud Rate, the closest approximation of the result, as well as the corresponding relative error:

RECEIVER

$$\textbf{-300}: \frac{100,000,000}{300 * 16} = 20833.3333333 \rightarrow \textbf{20833} \| \| \textbf{Err}: \frac{20833.3333333 - 20833}{20833} = \textbf{0.00001600025}$$

$$\textbf{-1200}: \frac{100,000,000}{1200 * 16} = 5208.33333333 \rightarrow \textbf{5208} \| \| \textbf{Err}: \frac{5208.33333333 - 5208}{5208} = \textbf{0.00006400409}$$

$$\textbf{-4800}: \frac{100,000,000}{4800 * 16} = 1302.08333333 \rightarrow \textbf{1302} \| \| \textbf{Err}: \frac{1302.08333333 - 1302}{1302} = \textbf{0.00006400409}$$

$$\textbf{-9600}: \frac{100,000,000}{9600 * 16} = 651.041666667 \rightarrow \textbf{651} \| \| \textbf{Err}: \frac{651.041666667 - 651}{651} = \textbf{0.00006400409}$$

$$\textbf{-19200}: \frac{100,000,000}{19200 * 16} = 325.520833333 \rightarrow \textbf{326} \| \| \textbf{Err}: \frac{325.520833333 - 326}{326} = \textbf{0.0014698364}$$

$$\textbf{-38400}: \frac{100,000,000}{38400 * 16} = 162.760416667 \rightarrow \textbf{163} \| \| \textbf{Err}: \frac{162.760416667 - 163}{163} = \textbf{0.00146983639}$$

$$\textbf{-57600}: \frac{100,000,000}{57600 * 16} = 108.506944444 \rightarrow \textbf{109} \| \| \textbf{Err}: \frac{108.506944444 - 109}{109} = \textbf{0.00452344546}$$

$$\textbf{-115200}: \frac{100,000,000}{115200 * 16} = 54.2534722222 \rightarrow \textbf{54} \| \| \textbf{Err}: \frac{54.2534722222 - 54}{54} = \textbf{0.00469393004}$$

TRANSMITTER

$$\textbf{-300}: \frac{100,000,000}{300} = 333333.333333 \rightarrow \textbf{333333} \| \| \textbf{Err}: \frac{333333.333333 - 333333}{333333} = \textbf{0.000001}$$

$$\textbf{-1200}: \frac{100,000,000}{1200} = 83333.3333333 \rightarrow \textbf{83333} \| \| \textbf{Err}: \frac{83333.3333333 - 83333}{83333} = \textbf{0.00000400001}$$

$$\textbf{-4800}: \frac{100,000,000}{4800} = 20833.3333333 \rightarrow \textbf{20833} \| \| \textbf{Err}: \frac{20833.3333333 - 20833}{20833} = \textbf{0.00001600025}$$

$$\textbf{-9600}: \frac{100,000,000}{9600} = 10416.6666667 \rightarrow \textbf{10417} \| \| \textbf{Err}: \frac{10416.6666667 - 10417}{10417} = \textbf{0.00003199897}$$

$$\textbf{-19200}: \frac{100,000,000}{19200} = 5208.33333333 \rightarrow \textbf{5208} \| \| \textbf{Err}: \frac{5208.33333333 - 5208}{5208} = \textbf{0.00006400409}$$

$$\textbf{-38400}: \frac{100,000,000}{38400} = 2604.16666667 \rightarrow \textbf{2604} \| \| \textbf{Err}: \frac{2604.16666667 - 2604}{2604} = \textbf{0.00006400409}$$

$$\textbf{-57600}: \frac{100,000,000}{57600} = 1736.11111111 \rightarrow \textbf{1736} \| \| \textbf{Err}: \frac{1736.11111111 - 1736}{1736} = \textbf{0.00006400409}$$

$$\textbf{-115200}: \frac{100,000,000}{115200} = 868.055555556 \rightarrow \textbf{868} \| \| \textbf{Err}: \frac{868.055555556 - 868}{868} = \textbf{0.00006400409}$$

The key components and their functions:

- A circuit with two Flip Flops in series. It prevents metastability in the RESET input signal.
- MUX: A combinational multiplexer that controls:
  - The type of Baud Controller (Transmitter or Receiver), and
  - The baud select.

  It determines the number of cycles the counter will wait before activating the sampling signal, based on the calculations above. It consists of a combinational always block with a sensitivity list for the type and baud select.
- Counter: Counts the cycles indicated by the multiplexer.
- sample_ENABLER: A sequential circuit that appropriately drives the sampling signal.
  Inside an always block with the sensitivity list consisting of the positive edge of both the clock and the reset signal:
  - If reset is 1, the sampling signal is reset to zero.
  - Otherwise:
    - If the sampling signal is active, it is deactivated in the next clock cycle.
    - Else, if the counter completes its count, the sampling signal is activated.

The dataflow of the circuit components is shown in Figure 1.

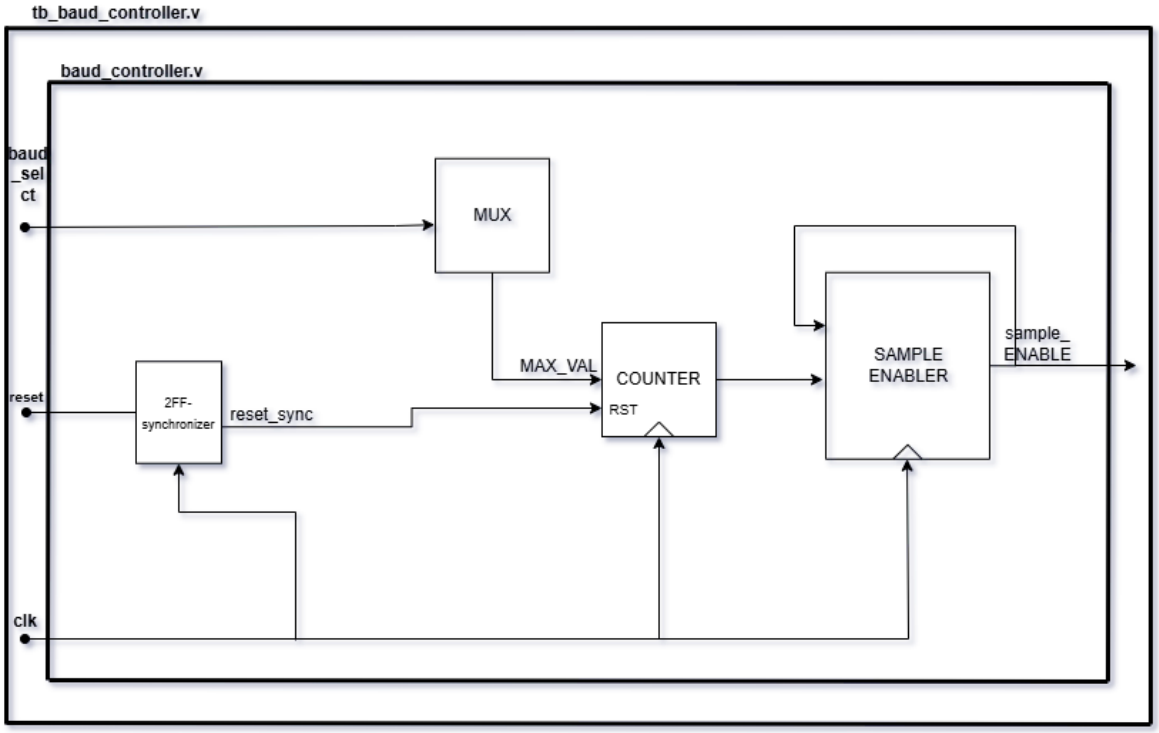

Figure 1: Circuit Dataflow in Part A

3.2 Verification:

For the verification of the circuit's functionality, a testbench was developed that:

- Selects a baud select value from 0 to 7,
- Activates the reset signal for at least one cycle to avoid using the state from previous measurements, and
- Runs the circuit for several activations of the sampling signal.

This process is repeated for all possible values of baud select and circuit type. Then, we examine the waveforms, measure the distances between consecutive activations, and verify them against the calculated values. Verification for some values is shown in Figure 2 and Figure 3.

4 Part B - Implementation of UART Transmitter

4.1 Implementation:

The **UART Transmitter** receives the symbol to be transmitted from the system, performs the transmission over the channel, and simultaneously informs the system of its availability. Specifically, it sends the symbol's bits serially, from **LSB** → **MSB**, enclosed between a **Start Bit (0)** and a **Stop Bit (1)**, along with the **Parity Bit**, which it calculates and transmits. The **Parity Bit** is calculated using the **XOR** expression of all the bits constituting the symbol. When the transmitter is not in transmitting state,

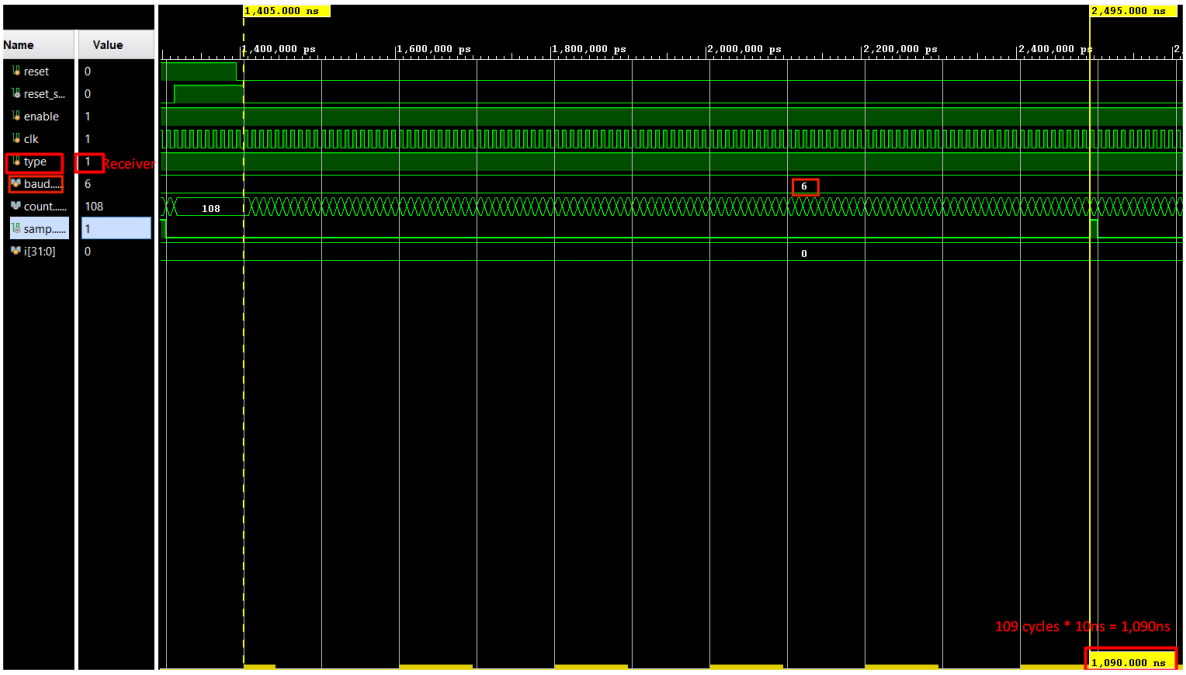

Figure 2: Delay between activations for Receiver and baud_select = 6

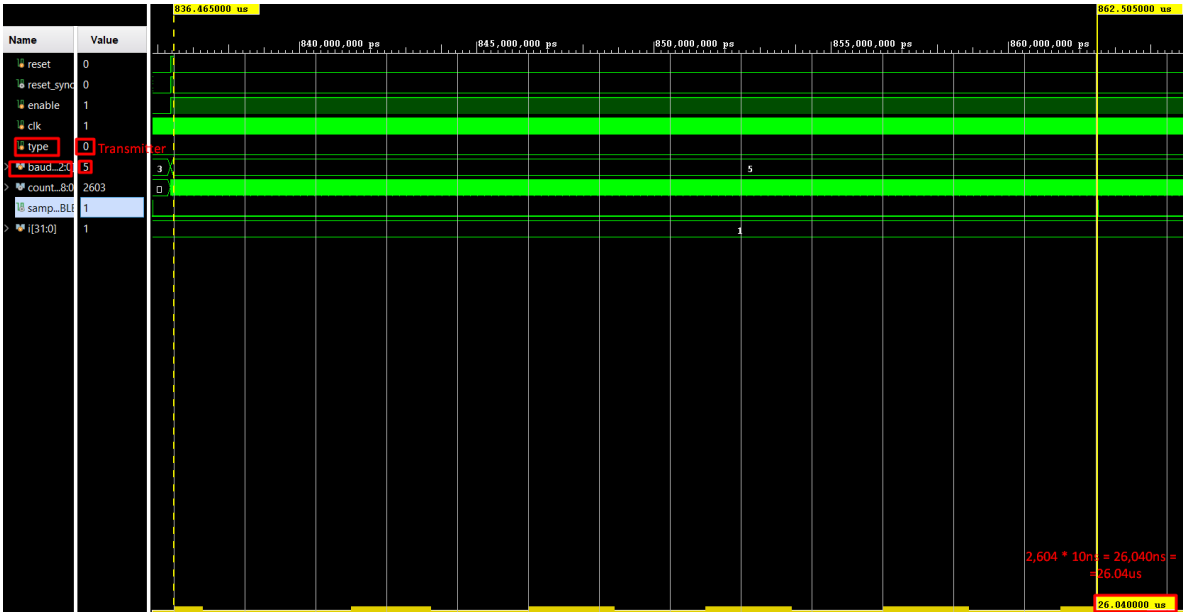

Figure 3: Delay between activations for Receiver and baud_select = 5

it is **available** and awaits the next symbol, while the communication channel remains **idle** at an active value of 1.

The key components and their functions:

- baud controller: The circuit from Part A.
- FSM_transmitter: A Moore-type Finite State Machine consisting of states and outputs that describe the transmitter's functions: The outputs are made up of **a)** the availability of the transmitter with Tx_BUSY, **b)** signal GotData that activates/deactivates the baud controller, and **c)** the output signal on the communication channel, RxD. The FSM with the states and their outputs is displayed in Figure 5 followed by their explanation below:
  - OFF: Transmitter is deactivated.
  - Communication channel is idle.
  - Bit_i: Transmits the i-th bit, where $i = 0, 1, \ldots 7$.
  - Start/Stop Bit: Signals the start and end of communication.
  - Verifies communication integrity with the receiver.

STATE	Outputs		
	Tx_BUSY	GotData	RxD
OFF	0	0	0
IDLE	0	0	1
StartBit	1	1	0
Bit_i	1	1	Data[i]
ParityBit	1	1	ˆ Data
StopBit	1	1	1

The following state transitions occur on positive Tx_sample_ENABLE cycles:

$$Start \rightarrow Bit_i \rightarrow Parity \rightarrow Stop \rightarrow IDLE$$

Yet, transitions between OFF and IDLE are controlled by the Tx_EN signal.

A **key design choice** is the immediate state change from IDLE → StartBit upon receiving new data with the activation of Tx_WR, which also activates the baud controller and starts the cycle count.

The dataflow of the circuit components is shown in Figure 4.

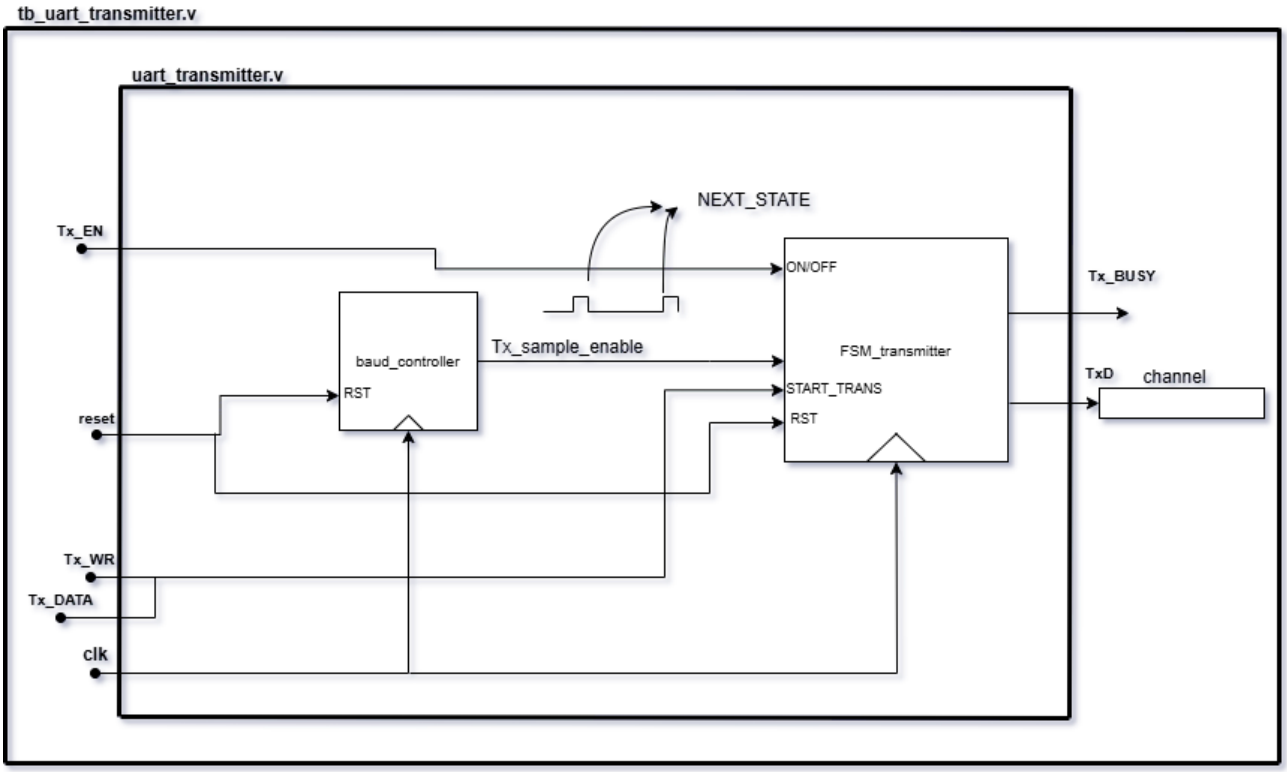

Figure 4: Circuit Dataflow in Part B

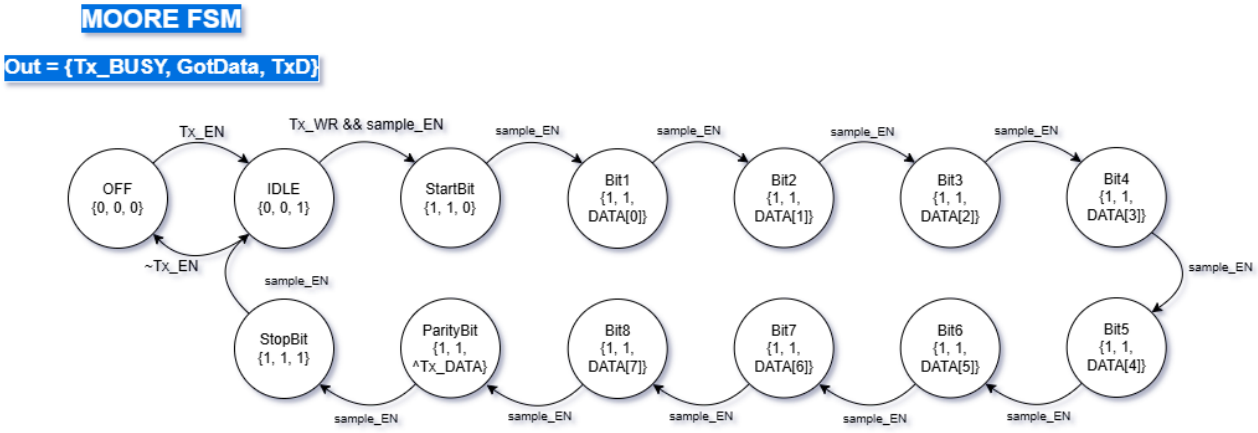

Figure 5: FSM diagram in Part B

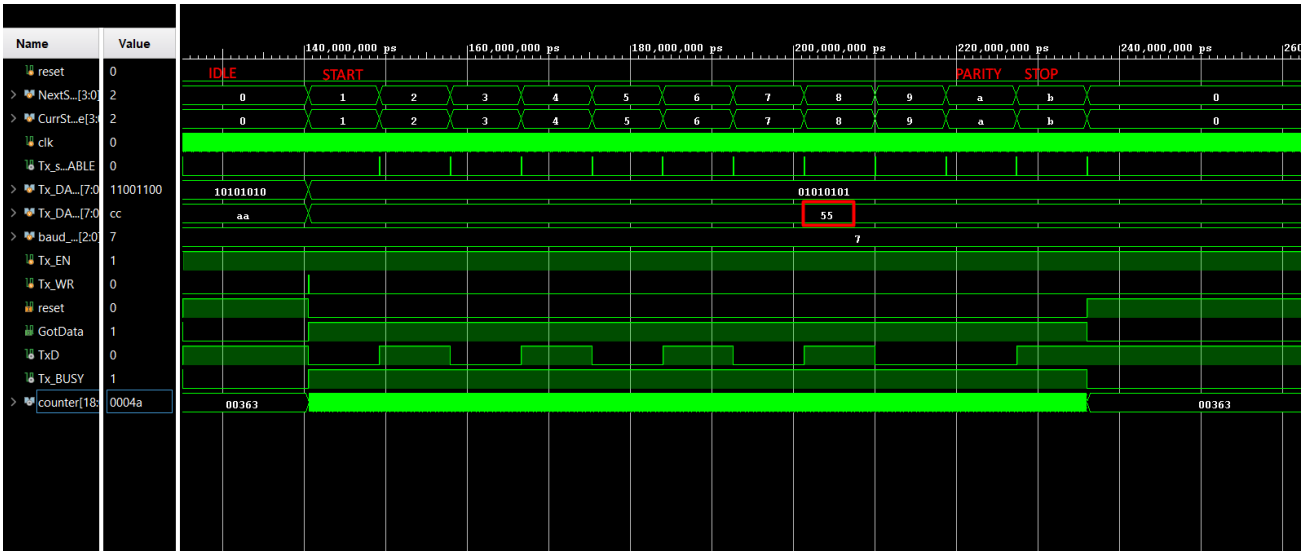

Figure 6: Transmission of symbol '0x55'

4.2 Verification:

To verify the transmitter's functionality, a testbench was developed that:

- Selects a baud select value,
- Activates the reset signal for a few cycles,
- Enables the transmitter and provides the symbol for transmission.

This process is repeated for various baud select values and symbols.

It is crucial to wait for the transmitter's availability before sending the next symbol, achieved as follows:

```
1 repeat(1) @(Tx_BUSY == 0);
2
3 #30000;
4
5 Tx_DATA = 8'h...;
6 Tx_WR = 1;
7 #10 Tx_WR = 0;
```

Waveforms are then examined, the distances between consecutive bits are measured, and the Parity calculation is verified to ensure the correct implementation of the transmitter. Verification results for some values are shown Figure 6.

5 Part C - Implementation of UART Receiver

5.1 Implementation:

The UART Receiver receives the symbol from the communication channel, checks its validity, and notifies the system accordingly. Specifically, the Receiver detects the start of transmission from the 1→0 transition of the channel signal and synchronizes with it. Then, it samples once at the center of each bit, aligning with the center of the StartBit and subsequently every 16 counts of the sampling signal Rx sample ENABLE. It notifies the system with success or failure signals about the communication result.

The key components and their functions:

- baud controller: The circuit from Part A, operating at 16 times the frequency of the Transmitter.
- FSM_receiver: A Moore-type Finite State Machine, consisting of states and outputs that describe the Receiver and its functions. The outputs include: **a)** the Receiver's availability indicated by signal Ready, **b)** the Rx_FERROR which signals a FRAME ERROR, **c)** the signal Rx_PERROR which signals a PARITY ERROR, **d)** the signal Rx_VALID which signals a successful transmission, and **e)** the received data. The FSM with the states and their outputs is displayed in Figure 8 followed by their explanation below:
  - OFF: The Receiver is turned off.
  - IDLE: The communication channel is idle.
  - IDLE_F_ERR1: Occurs when the Receiver fails to align with the center of the StartBit. It stays in this state for 176 cycles, for the rest of the communication, signaling Rx_FERROR during this period.
  - IDLE_F_ERR2: Occurs when the Receiver fails to sample the Stop bit at the expected time. It stays in this state for 16 cycles, for the rest of the communication, signaling Rx_FERROR during this period.
  - IDLE_P_ERR: Occurs when the received ParityBit doesn't match the calculated one. It stays in this state for 32 cycles, for the rest of the communication, signaling Rx_PERROR during this period.
  - IDLE_VALID: Occurs when communication is successful. It stays in this state for 16 cycles, for the rest of the communication, signaling Rx_VALID. While Rx_VALID is active, the system (higher level) can retrieve the data from Rx_DATA.
  - Bit_i: Receives the i-th bit, where $i = 0, 1, \ldots 7$.
  - Start/Stop: Samples the start and end of communication.
  - Parity: Samples the parity bit to verify communication integrity.

State Transitions:

- The transition from IDLE_START → StartBit occurs when the channel changes from 1→0, indicating the start of communication. A counter then counts 8 active cycles of the sampling signal Rx sample ENABLE to synchronize with the center of the StartBit.
- Transitions from StartBit → Bit_i → Parity → Stop → IDLE, occur every 16 active cycles of the sampling signal. During the last cycle, the channel is sampled, and the data is stored in memory.
- Transitions OFF⟷IDLE_START, are triggered by the Receiver's enable signal, Rx_EN.

STATE	Outputs		
	Rx_FERROR	Rx_PERROR	Rx_VALID
OFF	0	0	0
IDLE_START	0	0	0
StartBit	0	0	0
IDLE_F_ERR1	1	0	0
Bit_i	0	0	0
ParityBit	0	0	0
IDLE_P_ERR	0	1	0
StopBit	0	0	0
IDLE_F_ERR2	1	0	0
IDLE_VALID	0	0	1

STATE	Outputs			
	ready	start_counter	data_ready	maximum_cycles
OFF	0	0	0	16
IDLE_START	1	0	0	16
StartBit	0	1	0	8
IDLE_F_ERR1	0	1	0	176
Bit_i	0	1	1	16
ParityBit	0	1	0	16
IDLE_P_ERR	0	1	0	32
StopBit	0	1	0	16
IDLE_F_ERR2	0	1	0	16
IDLE_VALID	0	1	0	16

Important Design Choices:

- The state transition from {IDLE_VALID, IDLE_F_ERR1,IDLE_F_ERR2, IDLE_P_ERR} → IDLE_START occurs after the assigned cycle count for each state. Before a new communication begins, the FSM must be in the IDLE START state.
- A sequential always block is used to store bits in the Rx Data register. Using a counter for bit order, the data is stored as follows:

```
1  // handles Rx_DATA based on the counter_Data
2  always@(posedge clk or posedge reset) begin
3      if(reset) begin
4          Rx_DATA = 8'b0;
5      end
6      else if(data_ready) begin
7          case (counter_data)
8              3'b000: Rx_DATA[0] = RxD;
9              3'b001: Rx_DATA[1] = RxD;
10             3'b010: Rx_DATA[2] = RxD;
11             3'b011: Rx_DATA[3] = RxD;
12             3'b100: Rx_DATA[4] = RxD;
13             3'b101: Rx_DATA[5] = RxD;
14             3'b110: Rx_DATA[6] = RxD;
15             3'b111: Rx_DATA[7] = RxD;
16         endcase
17     end
18 end
```

The dataflow of the circuit components is shown in Figure 7.

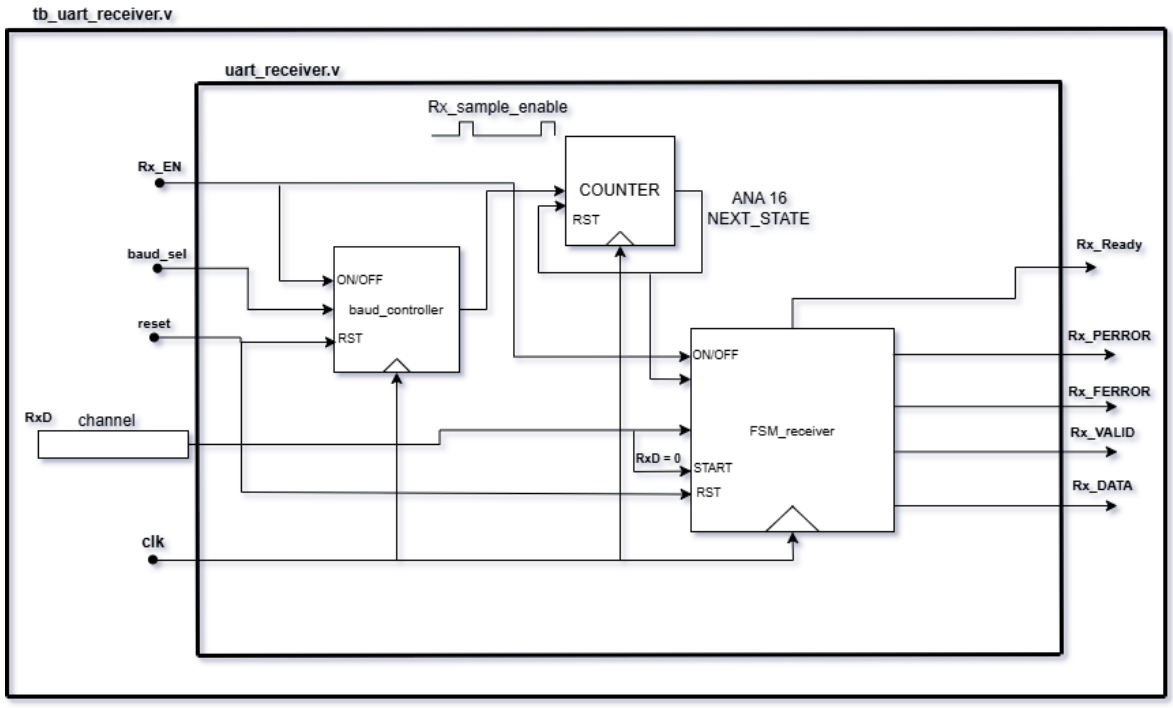

Figure 7: Circuit Dataflow in Part C

5.2 Verification

To verify the functionality of the circuit, the Receiver was directly connected with the Transmitter to obtain an authentic channel signal, avoiding incorrect simulation. Verification with waveforms and further explanations are presented in section 6.2.

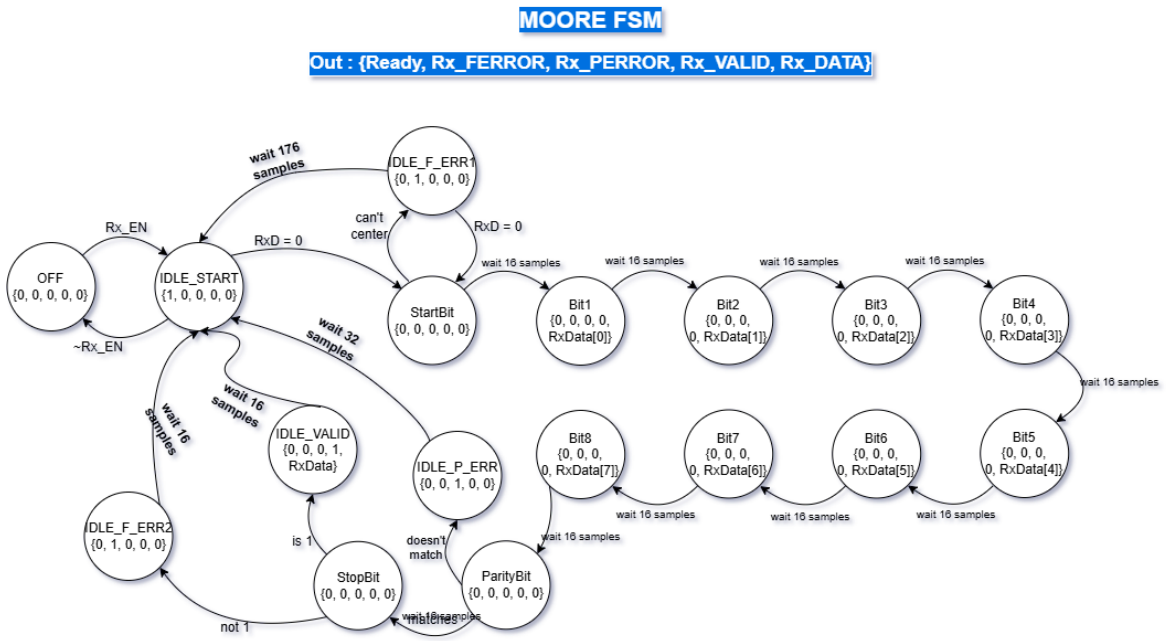

Figure 8: FSM diagram in Part C

6 Part D - Implementation of UART Transmitter-Receiver for Serial Data Transfer

6.1 Implementation:

Part D consists of the integration of the Transmitter and Receiver into a complete UART channel. The Dataflow diagram of this connection is shown in Figure 9.

6.2 Verification:

To verify the functionality of the circuit, a testbench was developed that selects a baud select value, activates the reset signal for a few cycles, and then activates both the Transmitter and Receiver before providing the symbol for transmission. This process is repeated for different baud select values for the Transmitter-Receiver (both identical and different) and for various symbols to be sent.

The purpose of the simulation is to ensure smooth operation for both correct and incorrect transmissions, evaluating the active signals in each case and the data received by the Receiver. **It is important** to wait for the availability of both the Transmitter and Receiver before sending a new symbol. This is achieved as follows:

```
1 repeat(1) @((Tx_BUSY == 0) && (Rx_ready == 1));
2
3 #30000;
4
5 Tx_DATA = 8'h...;
6 Tx_WR = 1;
7 #10 Tx_WR = 0;
```

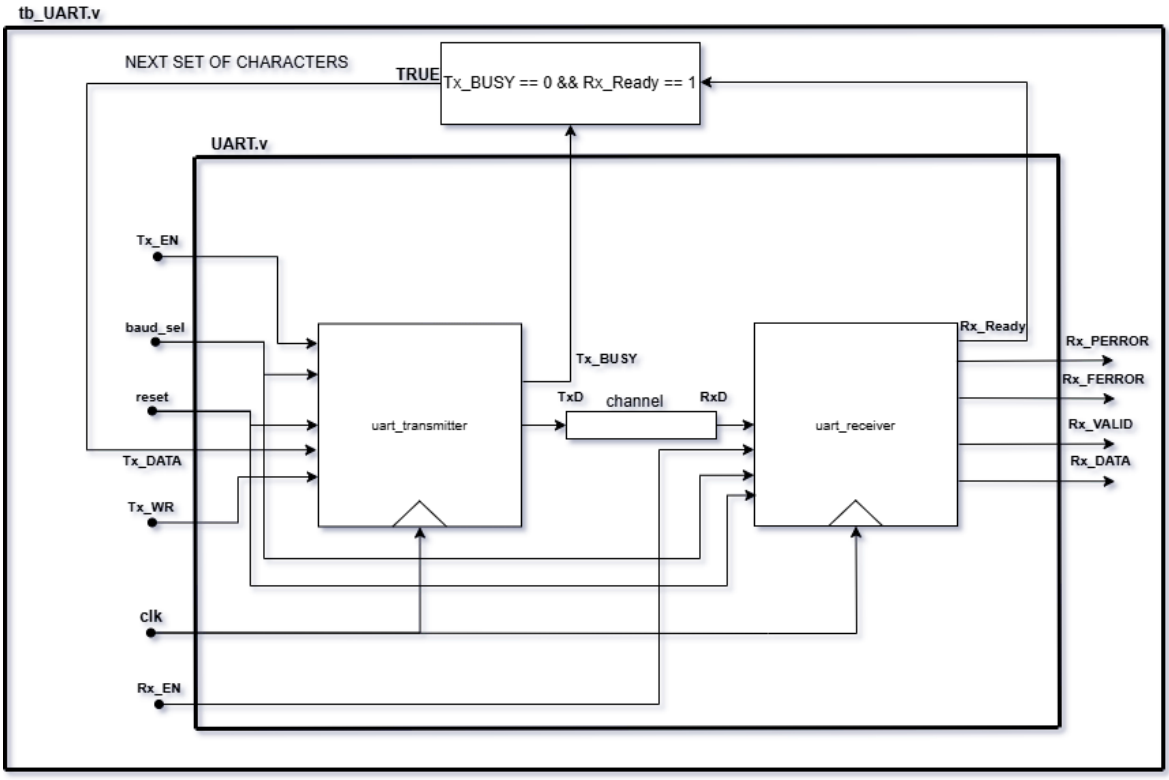

Figure 9: Circuit Dataflow in Part D

Afterwards, the waveforms are checked, the distances between consecutive bits are measured, the Parity calculation is verified, and the correct implementation of the Receiver is confirmed. Verification for some values is shown in Figure 10, Figure 11 and Figure 12.

7 Conclusions

Working with FSMs, understanding their operation, and troubleshooting issues (e.g., latch creation) was the most challenging part of the project, particularly with the FSM of the Receiver. The FSM diagram helped resolve the issues, and the project was completed smoothly.

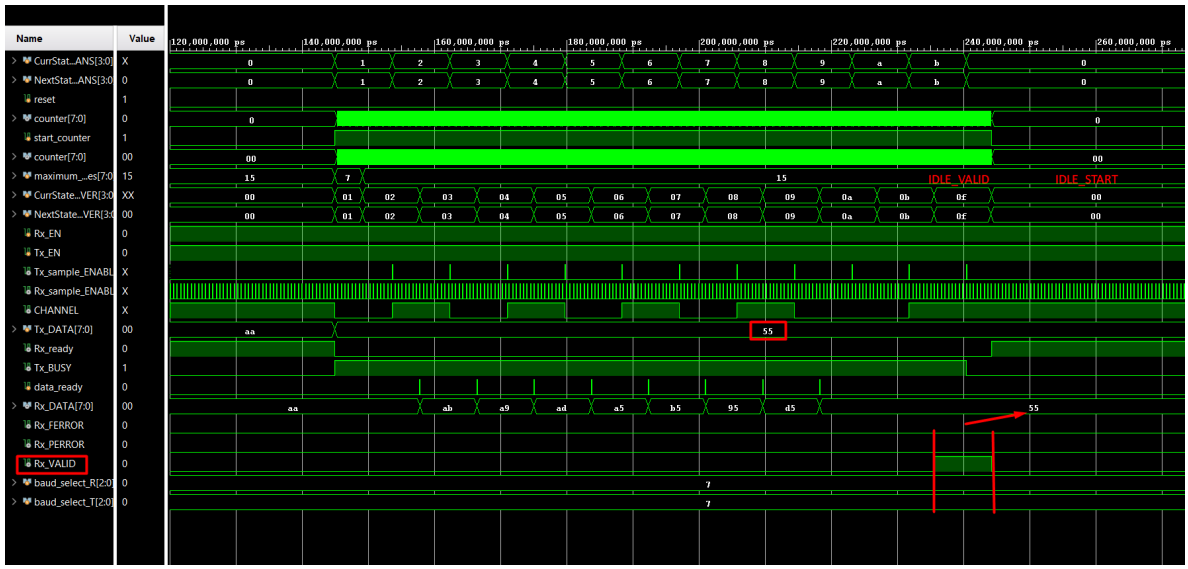

Figure 10: Successful Transmitter-Receiver communication

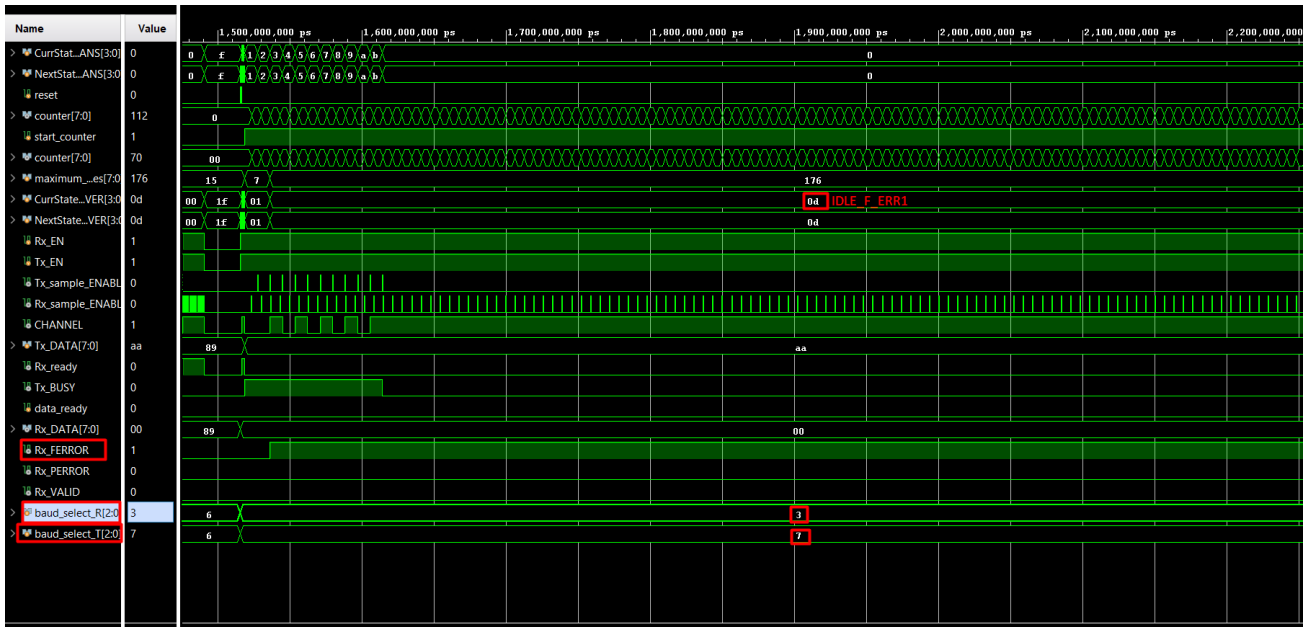

Figure 11: Unsuccessful Transmitter-Receiver communication (FRAME ERROR)

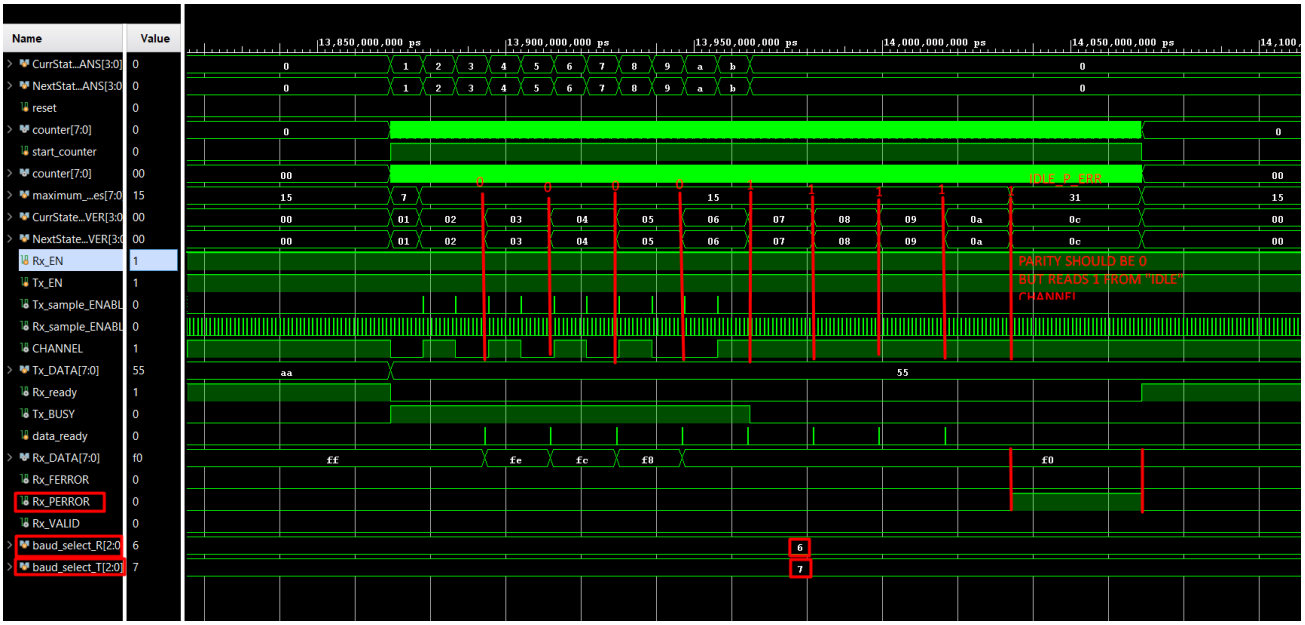

Figure 12: Unsuccessful Transmitter-Receiver Communication (PARITY ERROR)