Simon Fraser University
Engineering Science Department

Final Project Report: UART

ENSC 350: Digital System Design
Spring 2024

Date: April 1st, 2024

Instructor: Andres Erazo-Sosa

Prepared By:
[ENSC] Ghatarora, Gurnek (301394646)
[ENSC] Malhi, Akashroop (301393341)

1. Illustrator the design of all UART subsystems
    a. Baud Rate Generation
    b. Data Framing
    c. Error Detection and Correction
    d. Handshaking

Baud Rate Generation: The baud rate generator determines the number of clock cycles necessary to transmit one bit of data. It determines the rate at which bits are transmitted per second. This UART subsystem creates synchronization between devices by ensuring they both operate at an identical baud rate for effective communication. A mismatch in baud rates between the transmitting and receiving devices can lead to data corruption and loss of synchronization, leading to communication error. A higher baud rate allows for faster data transmission but will require more precise timing due to errors in the environment such as noise. However, a lower baud rate provides slower data transmission speed but decreases any chances of error.

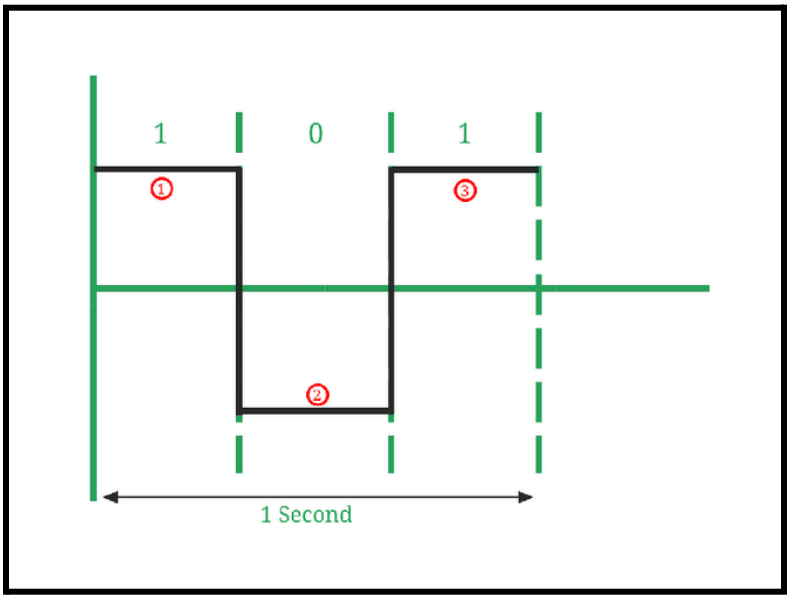

Figure 1: Baud Rate Generation

Data Framing: Data framing is how the data to be sent is structured in order to be transmitted across a communication channel. In the case of a UART, data is transmitted serially, one bit at a time, where character is represented in 7 or 8 bits. Our implementation will have the data line initialized to high. It will consist of 4 key components being the start bit, the stop bit, the data bits, and the parity bit. The start bit sets the data frame to low at the beginning of the data frame to indicate we are about to start transmission. When there is no transmission of any data, we will remain in an idle state, and the start bit will set the data frame to high. After we check the start bit, we can begin sending the data words one by one. The parity bit is used for error checking to ensure the integrity of the data being sent (more on that in Error Detection and Correction).

Finally, once all the data has been sent, the stop bit is used to assert the data frame back to high and return it to an idle state, indicating that all the data has finished being sent.

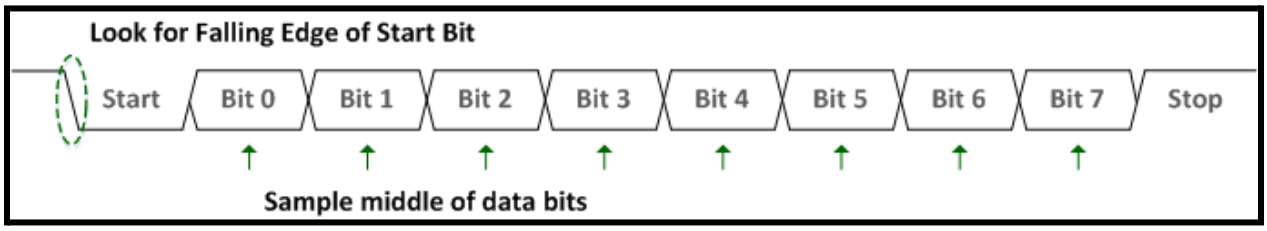

Figure 2: Data Framing

Error Detection and Correction: In UART communication, to ensure the accuracy of transmitted data, the parity checking is a crucial error detection mechanism. We will implement this into our design by appending a parity bit to each data byte and the value of this parity bit is determined based on the number of logic ones (1) in the data byte. When the data byte is received with its party bit the UART checks if the received parity bit matches the expected parity (even or odd). Any difference between the received parity bit and the expected parity indicates a transmission error.

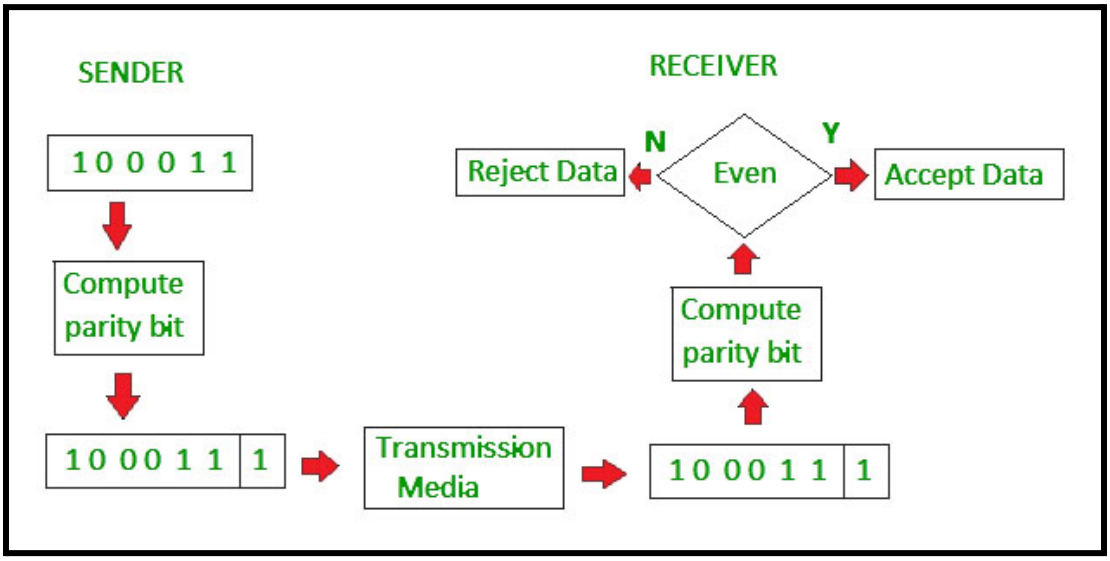

Figure 3: Error Detection and Correction

Handshaking: Handshake manages the data flowing between the transmitting and receiving devices. With UART, there are two forms of handshaking: hardware and software. In our implementation, the transmitter state begins to calculate for the parity bit after a low logic level indicates that the component is ready to send data.

The UART device, when transmitting data, will do the following: It will first check the other device's CTS signal. If the other device's CTS signal is asserted and our device's tx enable is signaled, our device will deassert its own RTS signal, and assert the transmitter's busy signal indicating to the other device that we do not want to receive while transmitting, and begin transmission. In our implementation, this was our logic when creating the transmitter component.

When receiving data the UART will do the following: It will first check if the other device's CTS is asserted, seeing that they are ready to send. They will wait for transmission to finish and then return back to being idle.

2. **Specify your Design (DataPath and Control Unit):**

For our design, we have implemented a half-duplex design, with only transmission supported. Our implementation consists of two main components. The baud generator will use a 50MHz clock and a fixed baud rate of 19200 bps to generate periodic baud pulses. This generated pulse will be input into our transmitter. The transmitter will take Switches 7 down to 0 as data input (tx_data) from our FPGA board, and use KEY(0) as its transmission enabler(tx_enable) button. It will also work on the same clock at 50MHz. The transmitter will consistently check for cts signal from PuTTY as well as tx_en from key(0) press. Once both are detected, we begin transmission. Parity checking is also done within our transmitter, where we XOR the parity bit with our transmission data.

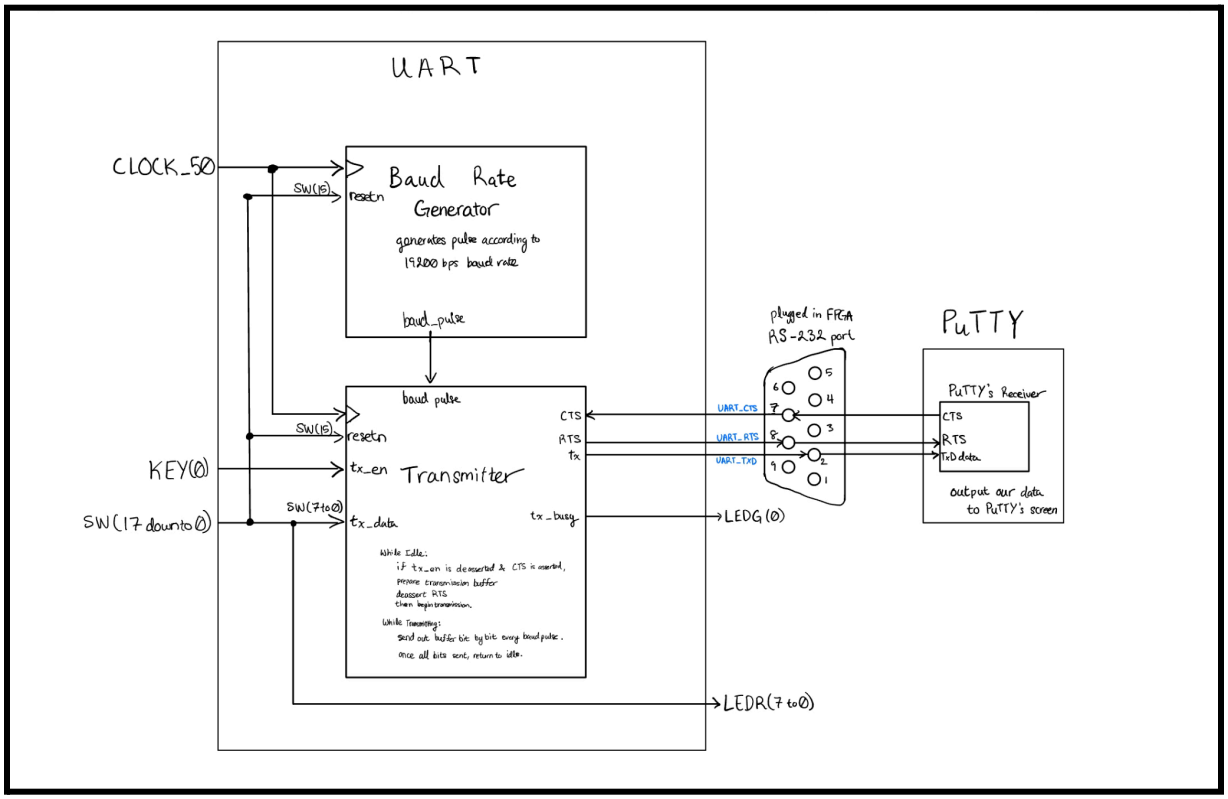

Figure 4: UART Datapath and Control Unit

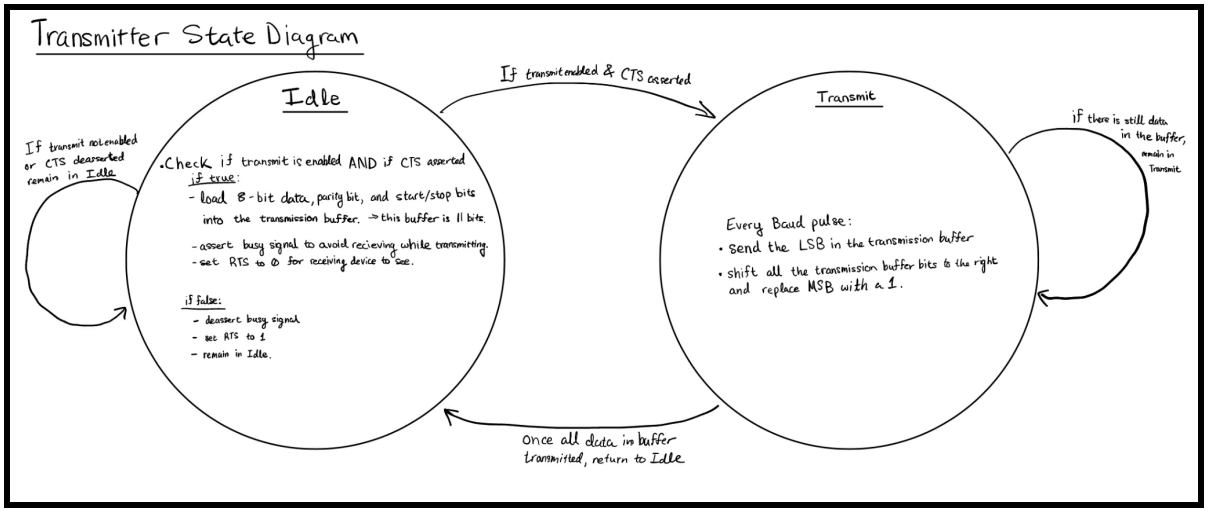

Figure 5: UART State Diagram

Testbench Results

Baud Rate Generator Testbench

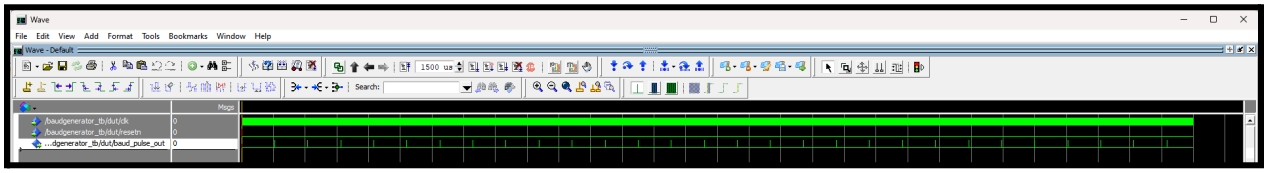

Figure 6: Baud Rate Generator Testbench Output

Here we see our Baud rate generator. We simply use reset to stabilize our signals, and see that we have baud pulse being generated periodically based on the clock. A baud period is 50MHz/19199bps.

Transmitter Testbench

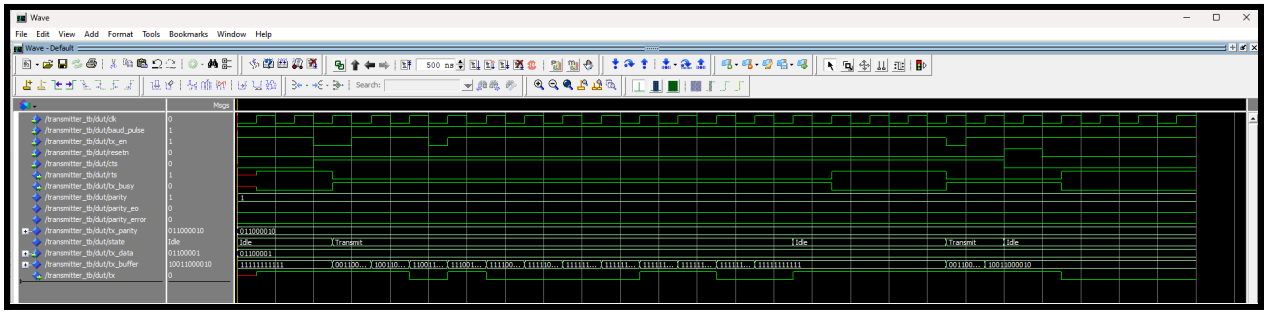

Figure 7: Transmitter Testbench Output

The transmitter testbench shows that all our signals occur on the rising clock edge. We set baud pulse to 1 to simulate a pulse occurring. When tx_en is set to 0 (button pressed) on a rising edge and while CTS is set to 1, we see the state to change from Idle to Transmit. At this point the busy signal is asserted and RTS is deasserted. We can see our checked parity is shown correctly in the tx_parity signal. Finally we see that the buffer shifts right every baud pulse and the LSB is sent out on the tx signal. The tx signal simply contains the bit that is being sent.

UART Testbench

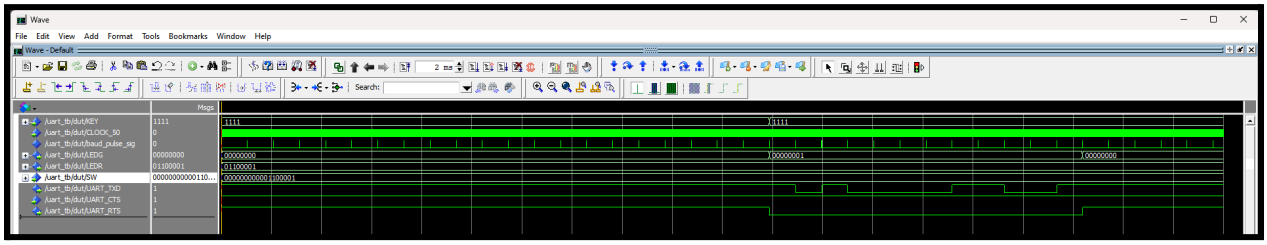

Figure 8: UART Testbench Output

In our full implementation, we can see that the baud pulse is successfully being generated from our baud rate generator. When KEY(0) is pressed and UART_CTS is detected as 1, we see LEDG(0) go to 1 indicating our busy signal is asserted and we have entered transmission state, and UART_RTS go to 0. We can see LEDR matches our SW bits 7 to 0 to show the data being sent. Finally we can

see that the UART_TXD (which is the bit being sent out) changes every baud pulse. In order it matches our SW inputs. This indicates we are successfully transmitting based on the generated baud pulse.

Example of PuTTY Output

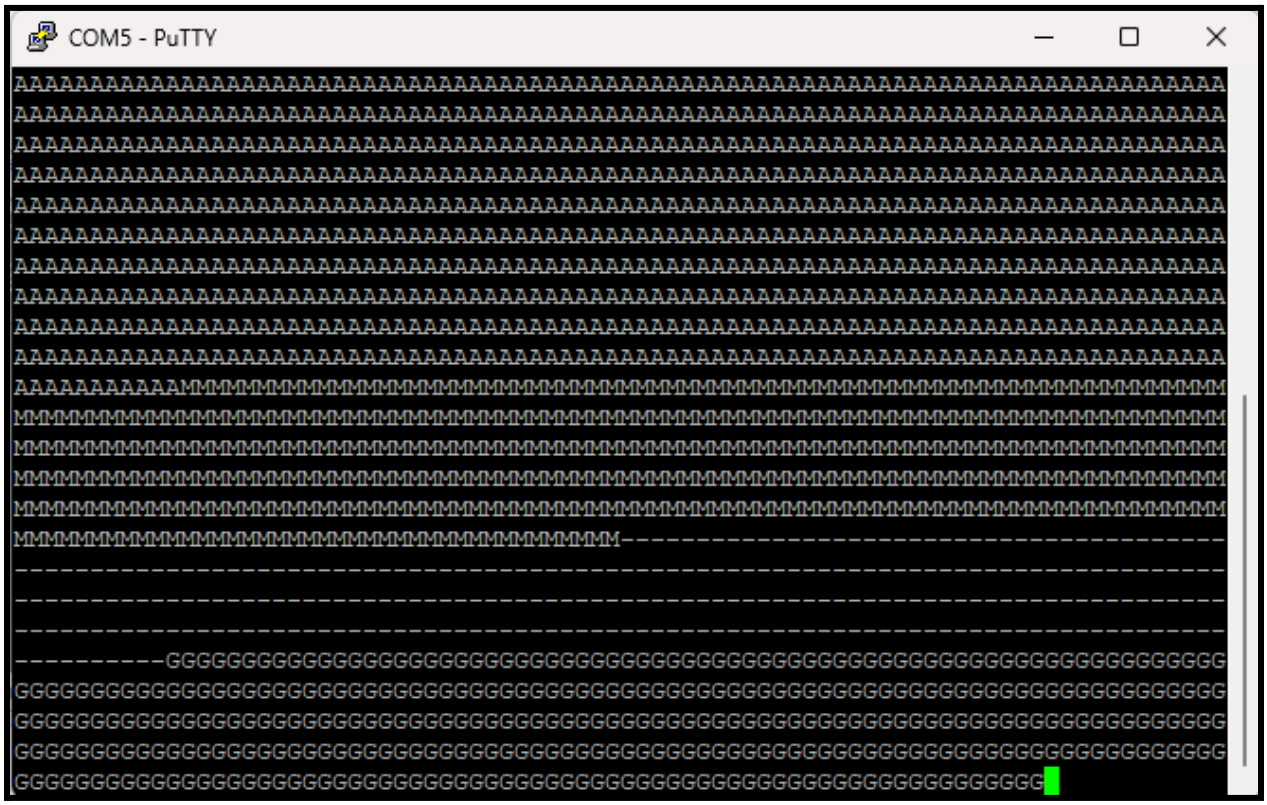

Figure 9: PuTTY Output