# **Introduction**

UART, or universal asynchronous receiver-transmitter, is one of the most used device-to-device communication protocols. This article shows how to use UART as a hardware communication protocol by following the standard procedure.

When properly configured, UART can work with many different types of serial protocols that involve transmitting and receiving serial data. In serial communication, data is transferred bit by bit using a single line or wire. In two-way communication, we use two wires for successful serial data transfer. Depending on the application and system requirements, serial communications need less circuitry and wires, which reduces the cost of implementation.

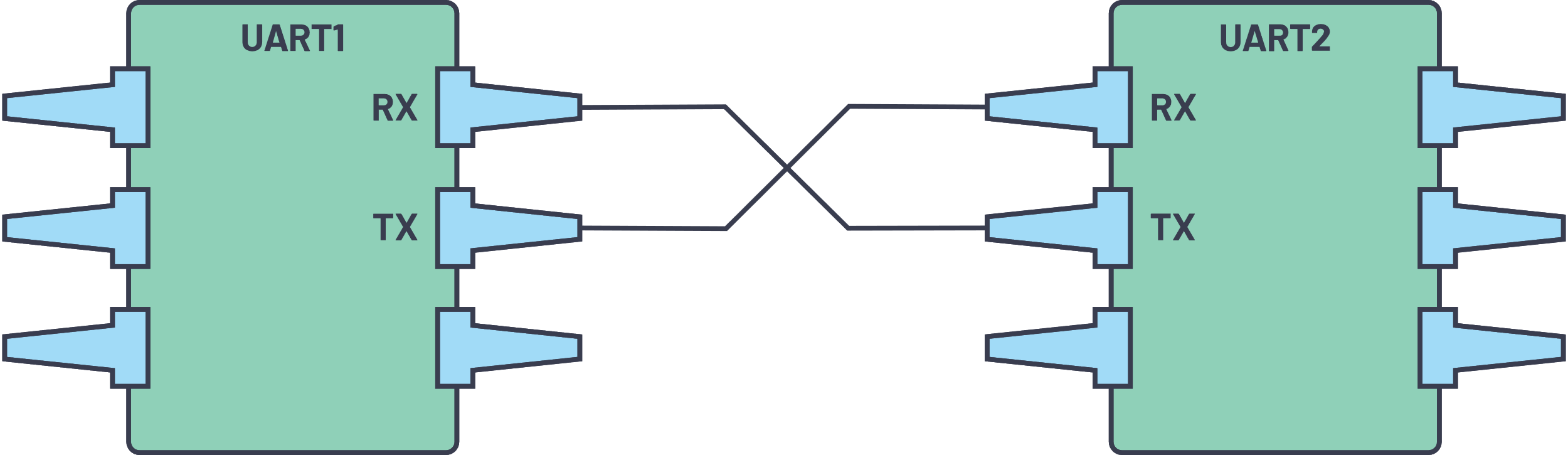
Communication protocol plays a big role in organizing communication between devices. It is designed in different ways based on system requirements, and these protocols have a specific rule agreed upon between devices to achieve successful communication.

Embedded systems, microcontrollers, and computers mostly use UART as a form of device-to-device hardware communication protocol. Among the available communication protocols, UART uses only two wires for its transmitting and receiving ends.

Despite being a widely used method of hardware communication protocol, it is not fully optimized all the time. Proper implementation of frame protocol is commonly disregarded when using the UART module inside the microcontroller.

By definition, UART is a hardware communication protocol that uses asynchronous serial communication with configurable speed. Asynchronous means there is no clock signal to synchronize the output bits from the transmitting device going to the receiving end.

# **Interface**

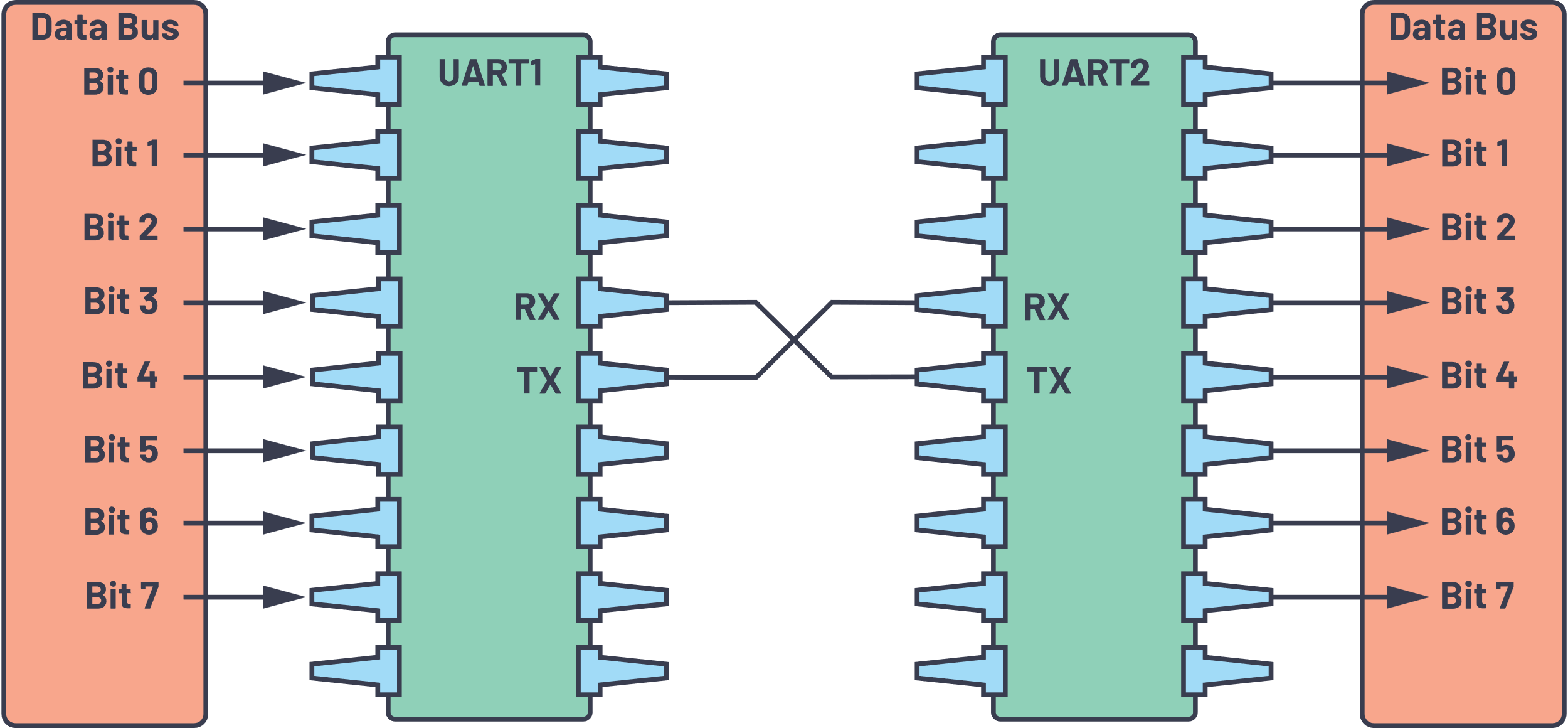


***Figure 1. Two UARTs directly communicate with each other.***

The two signals of each UART device are named:

* Transmitter (Tx)
* Receiver (Rx)

The main purpose of a transmitter and receiver line for each device is to transmit and receive serial data intended for serial communication.



***Figure 2. UART with data bus.***

The transmitting UART is connected to a controlling data bus that sends data in a parallel form. From this, the data will now be transmitted on the transmission line (wire) serially, bit by bit, to the receiving UART. This, in turn, will convert the serial data into parallel for the receiving device.

The UART lines serve as the communication medium to transmit and receive one data to another. Take note that a UART device has a transmit and receive pin dedicated for either transmitting or receiving.

For UART and most serial communications, the baud rate needs to be set the same on both the transmitting and receiving device. The baud rate is the rate at which information is transferred to a communication channel. In the serial port context, the set baud rate will serve as the maximum number of bits per second to be transferred.

*Table 1. UART Summary*

|  |  |
| --- | --- |
| Wires | 2 |
| Speed | 9600, 19200, 38400, 57600, 115200, 230400, 460800, 921600, 1000000, 1500000 |
| Methods of Transmission | Asynchronous |
| Maximum Number of Masters | 1 |
| Maximum Number of Slaves | 1 |

The UART interface does not use a clock signal to synchronize the transmitter and receiver devices; it transmits data asynchronously. Instead of a clock signal, the transmitter generates a bitstream based on its clock signal while the receiver is using its internal clock signal to sample the incoming data. The point of synchronization is managed by having the same baud rate on both devices. Failure to do so may affect the timing of sending and receiving data that can cause discrepancies during data handling. The allowable difference of baud rate is up to 10% before the timing of bits gets too far off.

# **Data Transmission**

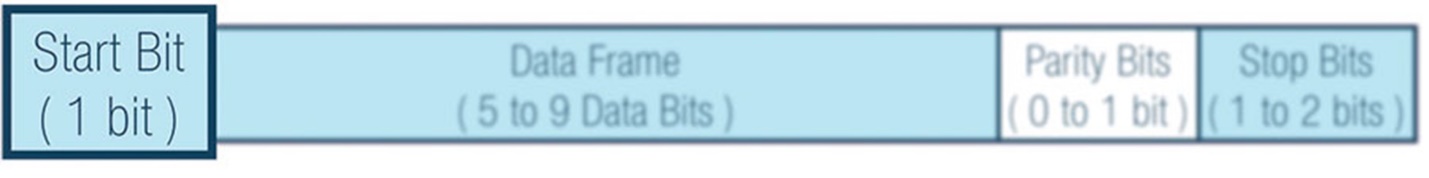
In UART, the mode of transmission is in the form of a packet. The piece that connects the transmitter and receiver includes the creation of serial packets and controls those physical hardware lines. A packet consists of a start bit, data frame, a parity bit, and stop bits.



***Figure 3. UART packet.***

## **Start Bit**

The UART data transmission line is normally held at a high voltage level when it’s not transmitting data. To start the transfer of data, the transmitting UART pulls the transmission line from high to low for one clock cycle. When the receiving UART detects the high to low voltage transition, it begins reading the bits in the data frame at the frequency of the baud rate.

[](https://www.analog.com/-/media/images/analog-dialogue/en/volume-54/number-4/articles/uart-a-hardware-communication-protocol/335962-fig-04.jpg?w=900&imgver=1)

***Figure 4. Start bit***.

## **Data Frame**

The data frame contains the actual data being transferred. It can be five bits up to eight bits long if a parity bit is used. If no parity bit is used, the data frame can be nine bits long. In most cases, the data is sent with the least significant bit first.

[](https://www.analog.com/-/media/images/analog-dialogue/en/volume-54/number-4/articles/uart-a-hardware-communication-protocol/335962-fig-05.jpg?w=900&imgver=1)

***Figure 5. Data frame.***

#### **Parity**

Parity describes the evenness or oddness of a number. The parity bit is a way for the receiving UART to tell if any data has changed during transmission. Bits can be changed by electromagnetic radiation, mismatched baud rates, or long-distance data transfers.

After the receiving UART reads the data frame, it counts the number of bits with a value of 1 and checks if the total is an even or odd number. If the parity bit is a 0 (even parity), the 1 or logic-high bit in the data frame should total to an even number. If the parity bit is a 1 (odd parity), the 1 bit or logic highs in the data frame should total to an odd number.

When the parity bit matches the data, the UART knows that the transmission was free of errors. But if the parity bit is a 0, and the total is odd, or the parity bit is a 1, and the total is even, the UART knows that bits in the data frame have changed.

[](https://www.analog.com/-/media/images/analog-dialogue/en/volume-54/number-4/articles/uart-a-hardware-communication-protocol/335962-fig-06.jpg?w=900&imgver=1)

***Figure 6. Parity bits.***

#### **Stop Bits**

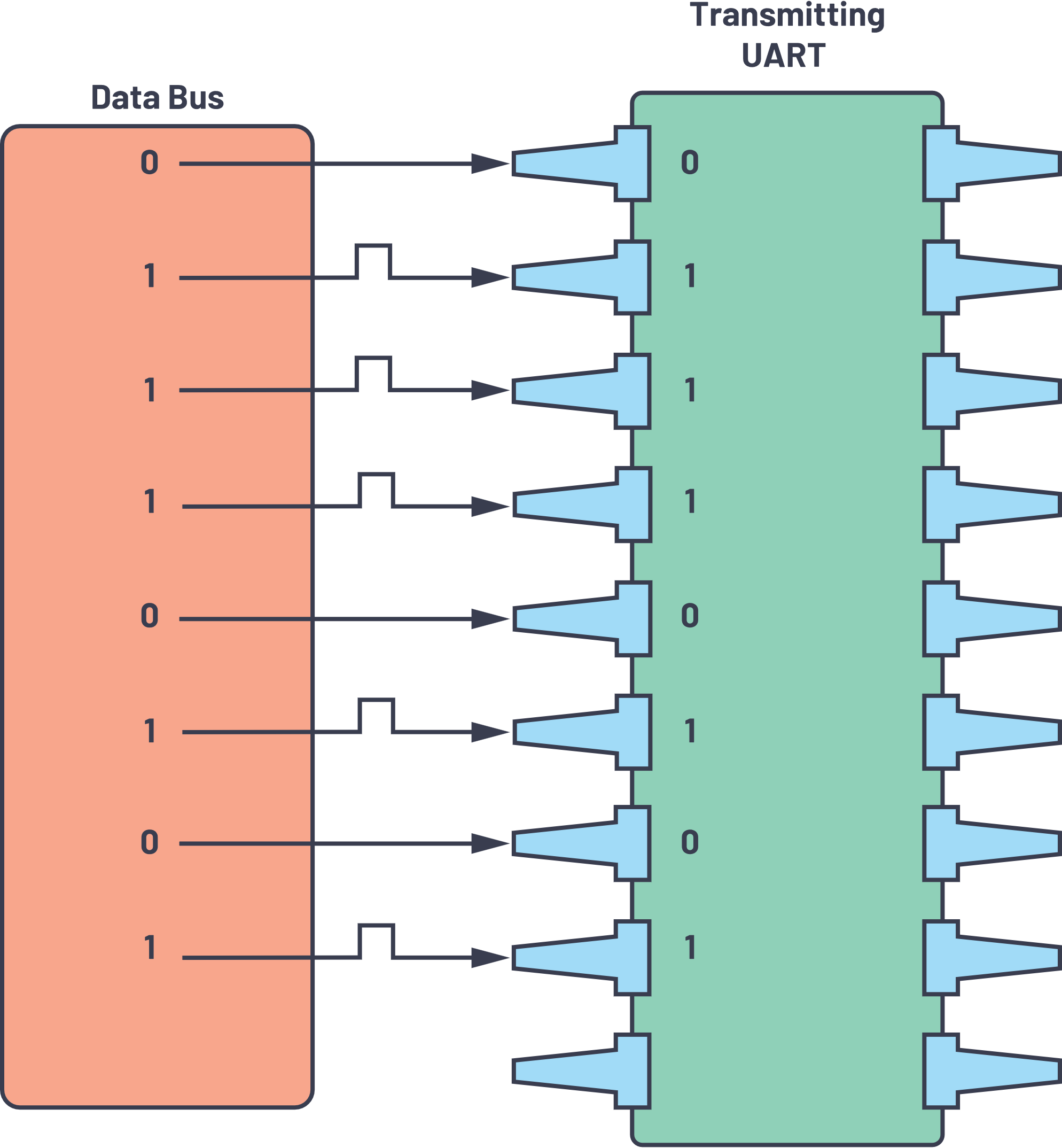
To signal the end of the data packet, the sending UART drives the data transmission line from a low voltage to a high voltage for one to two bit(s) duration.

[](https://www.analog.com/-/media/images/analog-dialogue/en/volume-54/number-4/articles/uart-a-hardware-communication-protocol/335962-fig-07.jpg?w=900&imgver=1)

***Figure 7. Stop bits.***

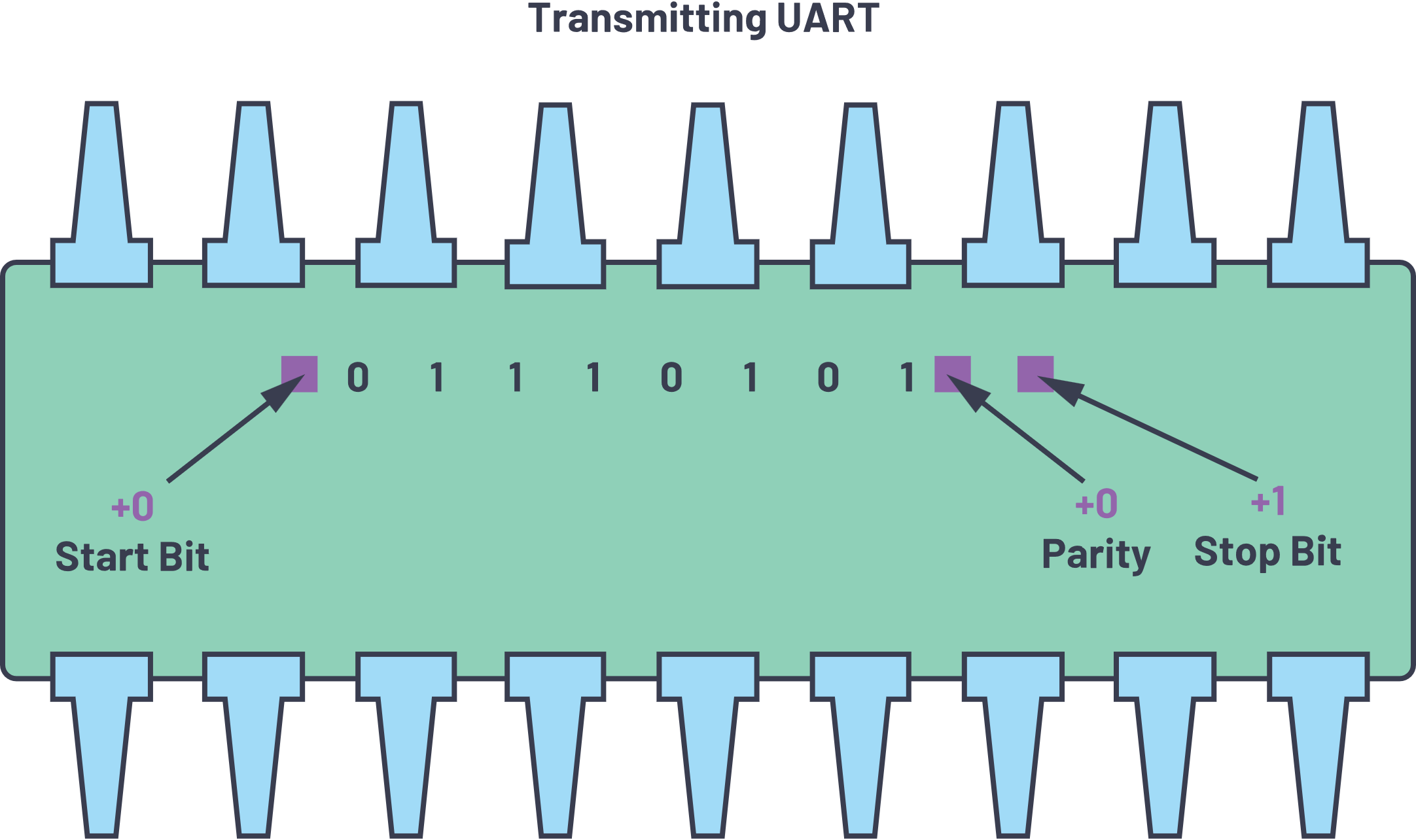
### **UART Transmission**

First: The transmitting UART receives data in parallel from the data bus.



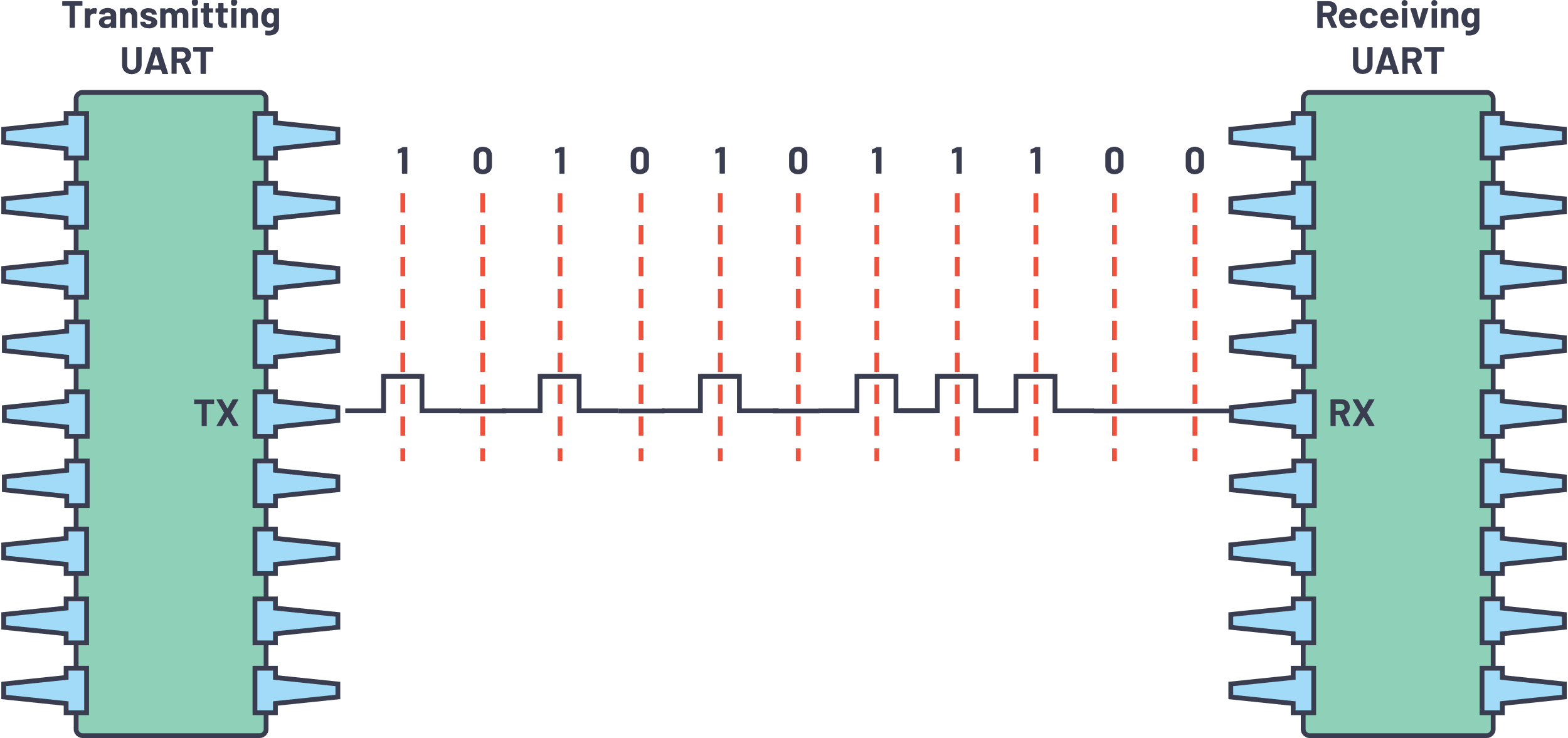
***Figure 8. Data bus to the transmitting UART.***

Second: The transmitting UART adds the start bit, parity bit, and the stop bit(s) to the data frame.



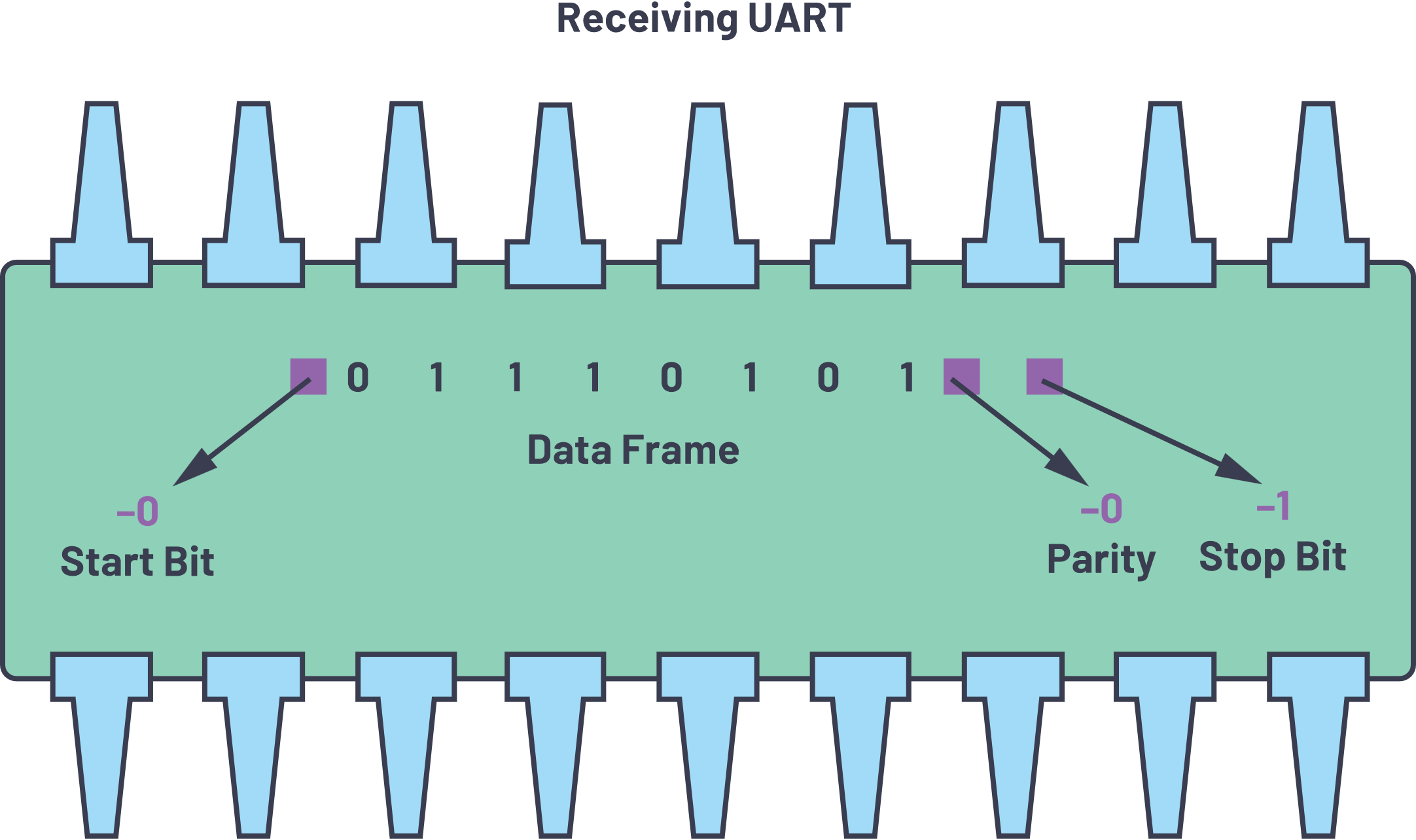
***Figure 11. The UART data frame at the Rx side.***

Third: The entire packet is sent serially starting from start bit to stop bit from the transmitting UART to the receiving UART. The receiving UART samples the data line at the preconfigured baud rate.



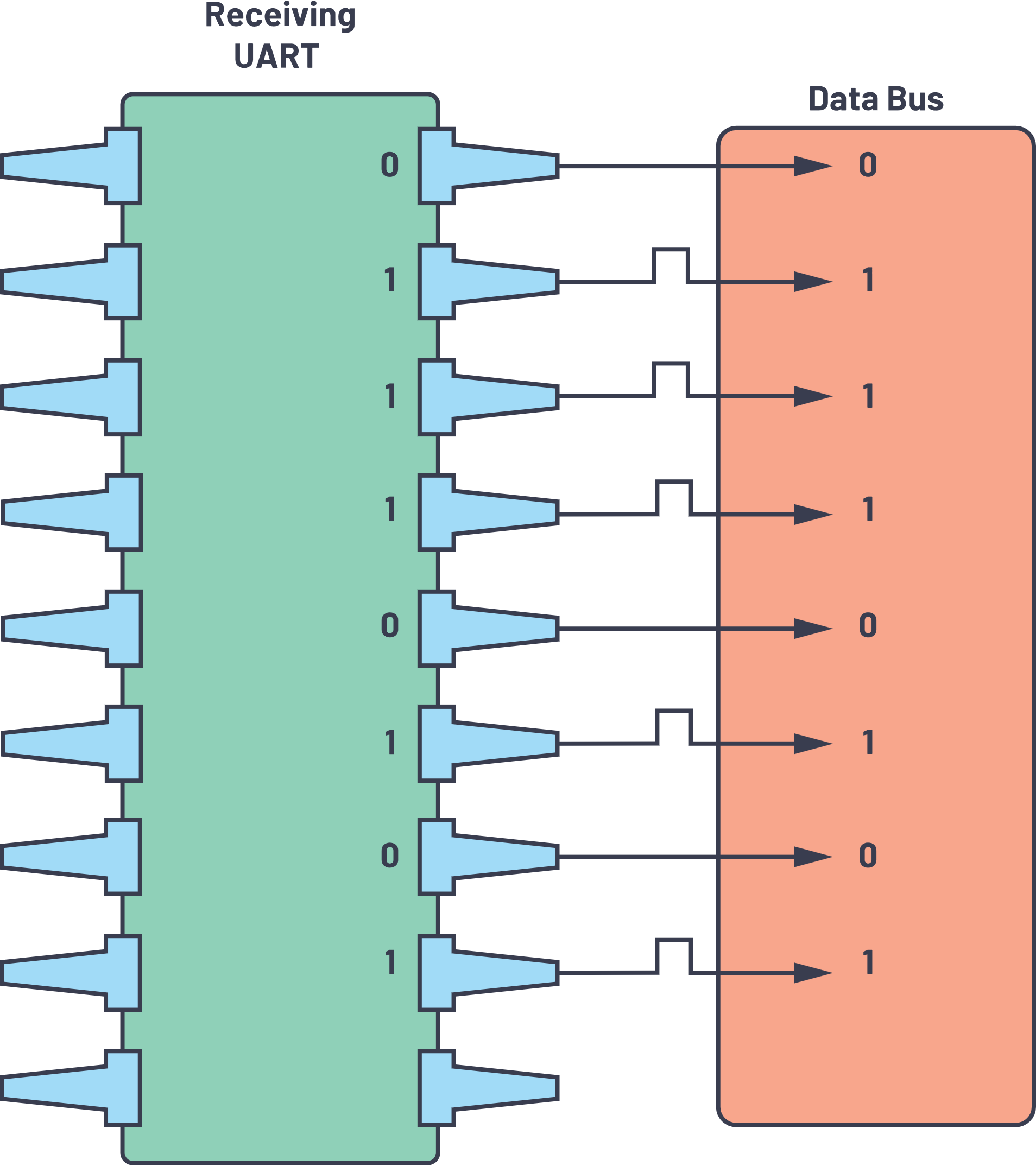
***Figure 10. UART transmission.***

Fourth: The receiving UART discards the start bit, parity bit, and stop bit from the data frame.



***Figure 11. The UART data frame at the Rx side.***

Fifth: The receiving UART converts the serial data back into parallel and transfers it to the data bus on the receiving end.



***Figure 12. Receiving UART to data bus.***