Objective

Implement a component for transmitting and receiving data through a standard RS-232 serial port in Verilog. Verify the HDL design with SystemVerilog and advanced testbenches.

Introduction

Serial communication requires only two signals, ***TxD*** and ***RxD***. The former refers to transmitted data and the latter to receiver data. A standard 9-pin serial port found on a personal computer is usually a male connector with pin 2 for RxD and pin 3 for TxD. On the other hand, a 9-pin serial port found on some FPGA's platforms are a female connector that use pin 2 as TxD and pin 3 as RxD. When connected, the TxD of one is connected to the RxD of the other and vice versa so that data transmitted by one is received by the other.

Data is transferred at a specific rate known as a ***baud rate*** given in bits per second. Common baud rates include 4800, 9600, 56000, and 115200. In addition to the baud rate, the ***start bit***, the number of ***data bits***, ***parity***, and number of ***stop bits*** must be selected and match in both the transmitting and receiver devices. A parity bit is added to a group of bits to make the total number of 1's always even or odd. A system operates with a parity or another, but never both. Most implementations transfer from five to eight data bits with even, odd, or no parity including one or two stop bits.

Serial transmission is synchronized using a start bit preceding the transmission and a stop bit indicating that the last bit has been transmitted completing the byte transfer. ASCII codes are often used for communicating between a PC and a deice or between two devices. The standard ASCII code are shown in Table1. Figure 1 shows the bit sequence for transmitting the ASCII code for a capital letter "T" (0x54) with no parity. Figures 2 and 3 shows the same transmission with even and odd parity respectively.

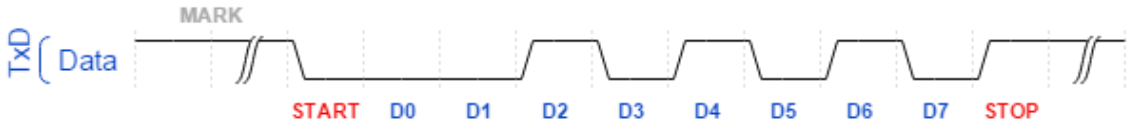

Figure 1. ASCII code 0x54 = 01010100 "T" sent with no parity.

Figure 2. ASCII code 0x54 = 01010100 "T" sent with even parity.

Figure 3. ASCII code 0x54 = 01010100 "T" sent with odd parity.

Functional Description

Transmit Module

Design a component ***uart_tx*** for transferring data using serial communication. Figure 4 shows the block diagram for the module uart_tx, which has ***clr*** and ***clk*** inputs used to reset and synchronize the communication respectively. A byte of data is input using ***tx_data[7:0]***. When ***ready*** is asserted high, the byte of data is transmitted on the TxD output starting with the ***LSB*** (*least significant bit*) first. After the transmission has completed, the ***tdre*** (*transmit data ready*) pin goes high.

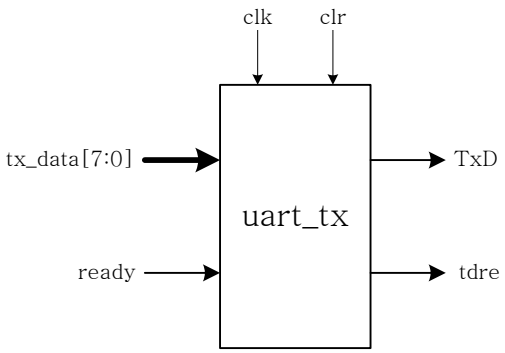

Figure 4. uart_tx.

Transmission begin with the TxD line transitioning from high to low for one ***bit time***. This leading bit is called the ***start bit***. The bit time depends on the ***baud rate***. Immediately following the start bit, the first data bit, the least significant bit, is transferred followed by the next, more significant bit until all eight bits of data have been transferred. Each bit remains on the TxD line for one bit time. After the

MSB (*most significant bit*) has been transferred , TxD goes high for one bit time. This trailing bit is called ***stop bit***.

Receive Module

Receiving data is similar to transmitting data. Figure 5 shows the block diagram for the receiver. eight bits of asynchronous data are input into RxD. Bits are shifted into the shift register, ***rx_data[7:0]***, least significant bit first. When rx_data[7:0] is full, the ***rdrf*** (*received data ready flag*) is set to 1 to signify that rx_data[7:0] contains a complete byte. The output rdrf is set to 0 by setting ***rdrf_clr*** to high. The ***FE*** (*framing error flag*) is set to 1 if the stop bit is not 1. This is one indication that the data on rx_data[7:0] may not be accurate.

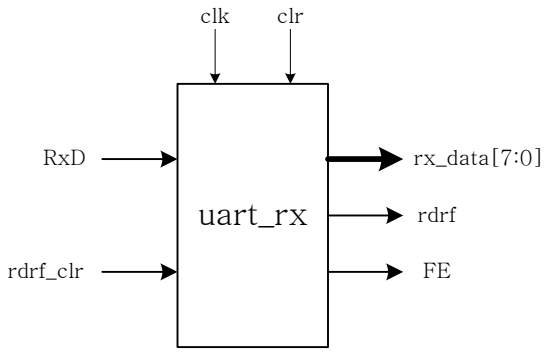

Figure 5. uart_rx.

Design

During transmission, the start bit is transmitted for an entire bit time just like the data and the stop bit. When receiving, by waiting only a half bit time after the start bit has been initiated, the data shifted into the shift register is farthest away from the time the signal changes. That is, half way between the beginning and ending time during which the bit is valid. Figure 6 shows the manner the data has to be transmitted and received depending of the bit time (baud rate). Table 1 show the common asynchronous serial baud rates.

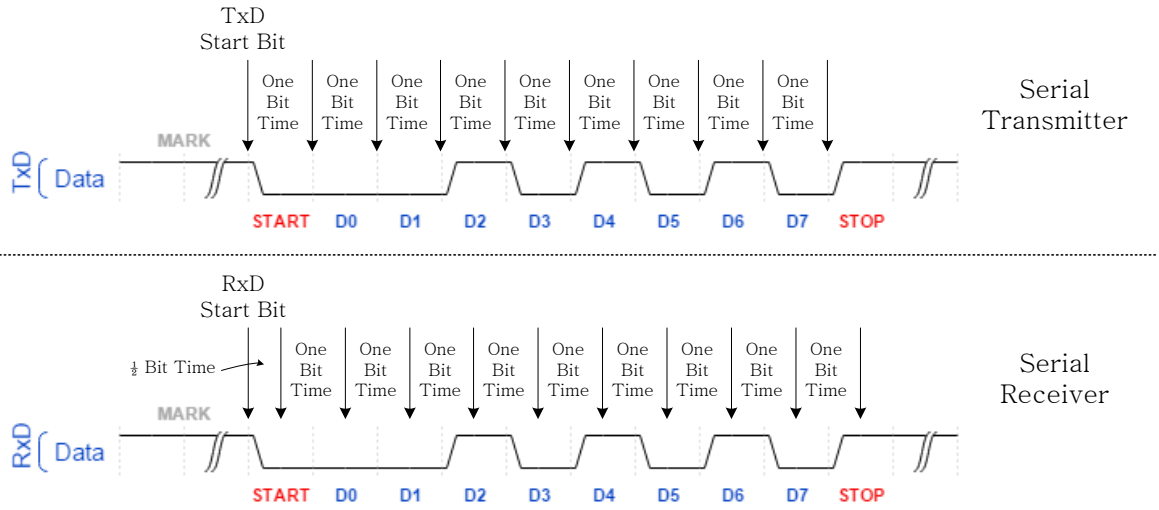

Figure 6. Timing differences between transmitting and receiving.

Table 1. Common asynchronous serial baud rates.

Baud rate	Bit time (msec)	Number of STOP bits
110	9.09	2
300	3.33	1
600	1.67	1
1200	0.833	1
2400	0.417	1
4800	0.208	1
9600	0.104	1
14400	0.069	1
19200	0.052	1
28800	0.035	1
38400	0.026	1

Verification Project

Implement a uart (transmitter - receiver) with 8 bits of data and a parity bit. The kind of parity must be selectable by a signal called *parity[1:0]*. When parity[1:0] is "00 and 11" the transmission has no parity, with "01" has even parity, and with "10" has odd parity. The parity bit has to be calculated by the hardware of the transmitter and selfchecked by the receiver indicating an error with the signal *parity_error*. It means that the tx_data and the rx_data signals have 8 bits of data, and the TxD and RxD signals transmit or receive 10 or 11 bits serially, taking into consideration the start and stop bits.

Figure 7 show the block diagram of the uart for the verification project.

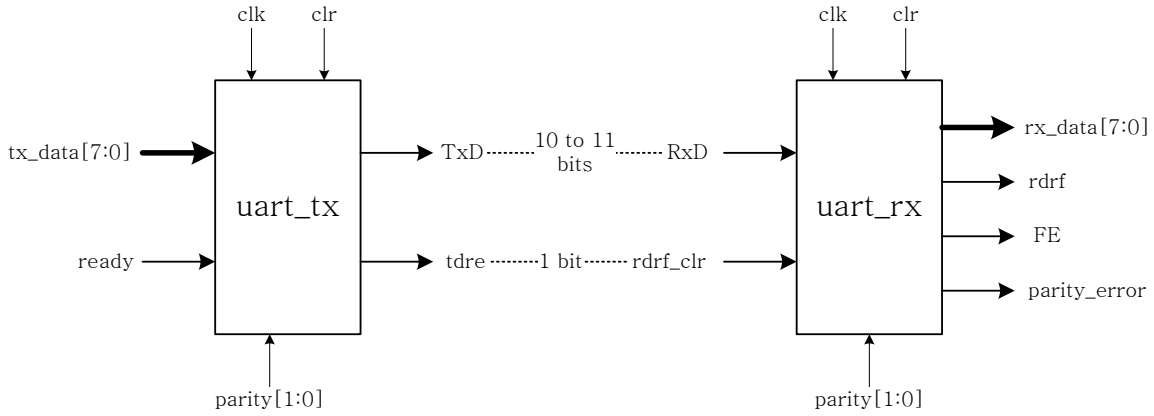

Figure 7. Uart for the verification project.