RS232 Reference Component

Revision: July 25, 2008

This document was produced by Digilent Romania. For questions, contact support@digilent.ro.

Overview

This document describes the Universal Asynchronous Receiver Transmitter (UART) VHDL component which can be used either with the PmodRS232 or with an on-board RS232 port. A UART component is used to convert serial data to parallel data, and parallel data to serial data. The serial data transferred into the UART is placed on an output bus after the UART converts it into parallel information. This bus can then be used as input to other logic in the gate array. The resulting data can then be sent back out serially by using the UART component again.

Features include:

- parallel to serial and serial to parallel data conversion
- user-defined odd or even parity bit (default is odd parity)
- easily changeable baud rate (default at 9600).

Functional Description

The receiving portion of the UART handles the serial to parallel conversion, while the transmitting portion of the UART handles the parallel to serial conversion.

The UART component requires a 50MHz clock input.

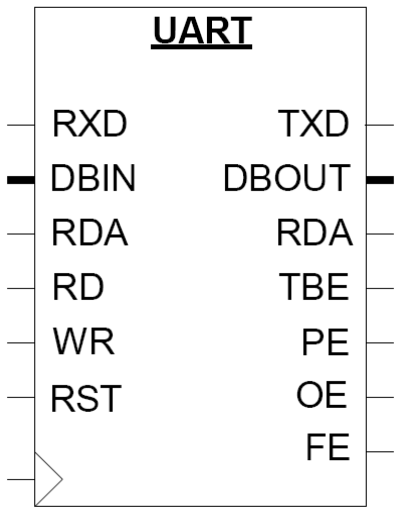

Figure 1 ***The UART Component***

UART Components

The UART has two main functional blocks that can be thought of as two separate circuits packaged in one component. The UART includes a circuit for receiving serial information, and a circuit for transmitting serial information. The receiver takes in a byte of serial data transmitted to the RXD port, and converts it into one byte of parallel information. This byte is then placed on the DBOUT port. The transmitter takes a byte of parallel information found on the DBIN port, and converts it into a byte of serially transmitted data. This data is transmitted on the TXD port. Figure 2 shows the UART separated into its two circuits.

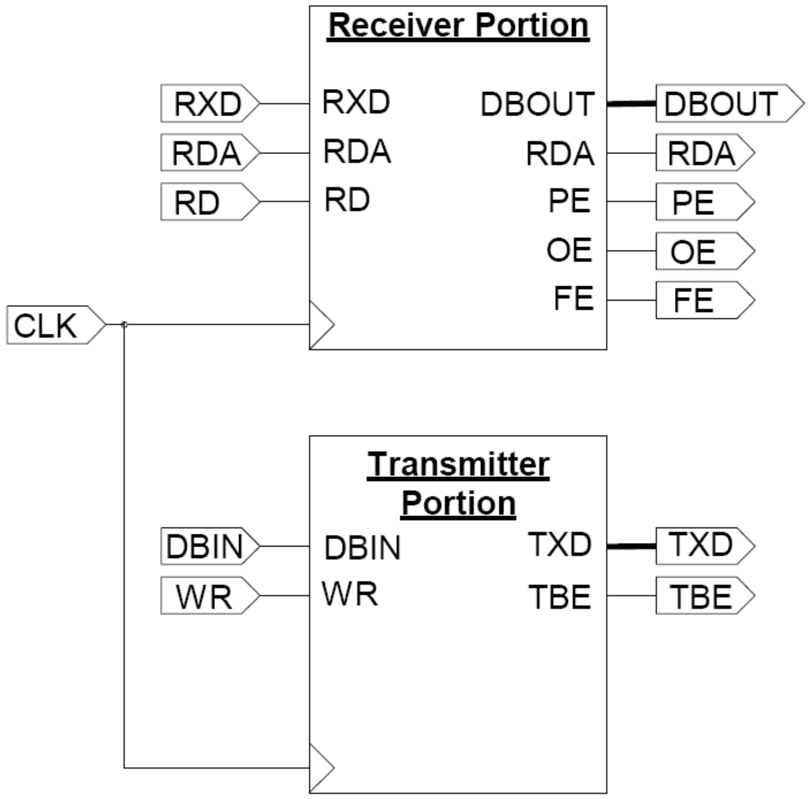

Figure 2 ***Receiver and Transmitter Circuits***

Receiver Circuit Operation

The receiver portion of the UART accepts serial data input, and converts it into parallel data. This portion includes a serial data controller, two counters used for synchronization, a shift register, and an error bit controller. The shift register is used to store the incoming data from the RXD port. When a read cycle is over, the shift register is ready to acquire the received data byte. Because the data being transferred to the RXD port is coming in at a specific baud rate, a controller is needed to synchronize the data acquisition phase. The serial data controller, which uses a state machine and the two synchronization counters, is used for this synchronization. The state machine is calibrated to read the RXD port in the middle of each bit transmission. A diagram of the state machine is shown in Figure 3.

When idle, the serial data pin (RXD in this case) is held high, so the idle state remains unchanged until the RXD port goes low. Once the RXD is asserted low, the EightDelay state is entered. This state is used to assure that the RXD port is read in the middle of each bit transmission. The counter, ctr, increments at a rate that is 16 times faster than the baud rate. When the EightDelay state is entered, the ctr counter is allowed to count up to eight.

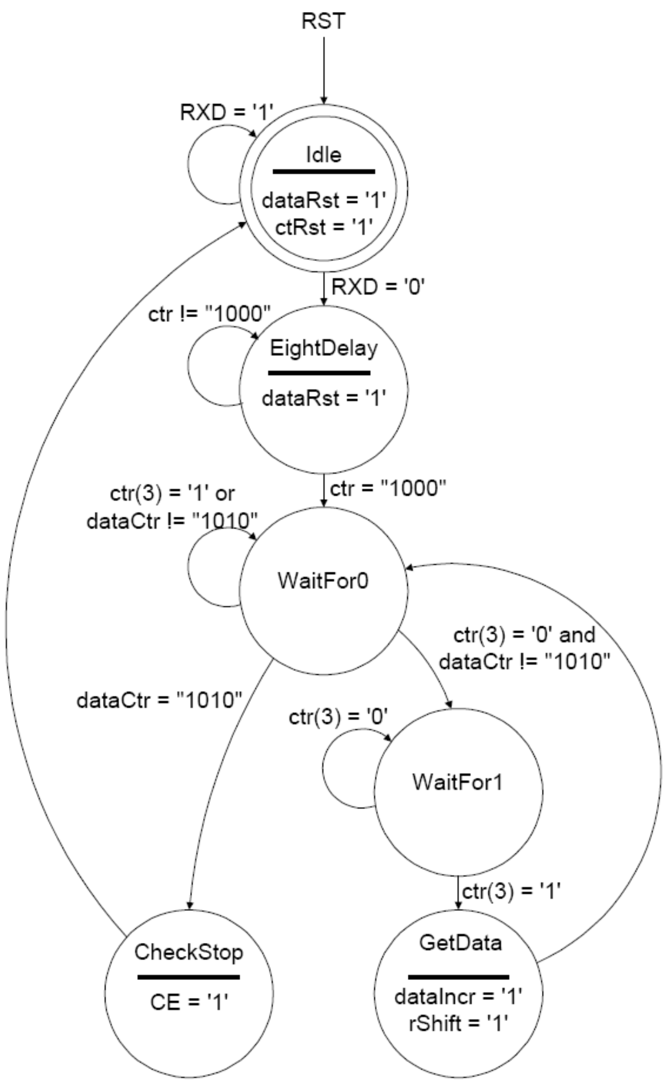

Figure 3 ***Receiving State Machine***

When this occurs, the WaitFor0 state is entered, followed by the WaitFor1 state. These two states rely on the most significant bit of the ctr counter for their next state logic. These states precede the GetData state, which shifts in the value of RXD at the time that the state is entered. The two wait states ensure that the state machine is delayed exactly long enough to read the RXD signal in the middle of its next transmission. The GetData state also increments the dataCtr counter, which is used to keep track of how many data bits have been shifted into the shift register. Once this counter is equal to 10 (8 data bits, 1 parity bit, and 1 stop bit), the CheckStop state is entered. This state enables the error bit controller. The idle state is loaded immediately after the CheckStop state.

A timing diagram of the shifting procedure can be seen in Figure 4. The rShift signal enables the receiving shift register to shift in the value of RXD at the moment that rShift goes high. Notice how the rShift signal is asserted in the middle of each data bit on RXD.

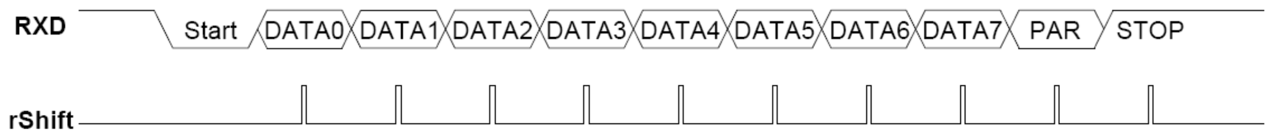

Figure 4 ***Receiver Timing Diagram***

The error bit controller analyzes the received data for three types of errors: parity error, frame error, and over-write error. A parity error occurs if the parity bit does not agree with the number of 1's in the data portion of the shift register. If set for odd parity, the parity bit should be a 1 if there are an odd number of 1's in the data portion of the shift register. If there is an even parity, the parity bit should be a 1 if there are an even number of 1's in the data portion of the shift register. This UART component is set to an odd parity by default, but can easily be changed by deleting the "not" in line 114 and 117 of the RS232RefComp.vhd file. If a parity error is detected, the PE port is held high.

A frame error indicates that the UART is not reading the incoming data at the right time. This occurs if the stop bit is not a 1. In this case the FE port is held high.

The over-write error indicates that the data byte just received wrote over a previously received, yet never read, data byte. Once parallel data is available, after a byte of serial data has been read, the RDA port is held high. When data is read from the DBOUT port, which is set equal to the data portion of the shift register (once all data has been shifted in), the RDA port is brought low. If data has been received and the RDA port is still high, the shifted-in data will replace the data held in the DBOUT port, and the OE port will be held high to indicate an over-write error.

To change the baud rate at which the UART receives data, the baudDivide constant must be changed (line 76). The baudRate must be set to the binary equivalent of [Clock Frequency]/[Baud Rate]. The baudDivide constant is just baudRate divided by 16, so it will be equal to baudRate shifted right four places. A block diagram of the receiving portion of the state machine can be seen in Figure 5.

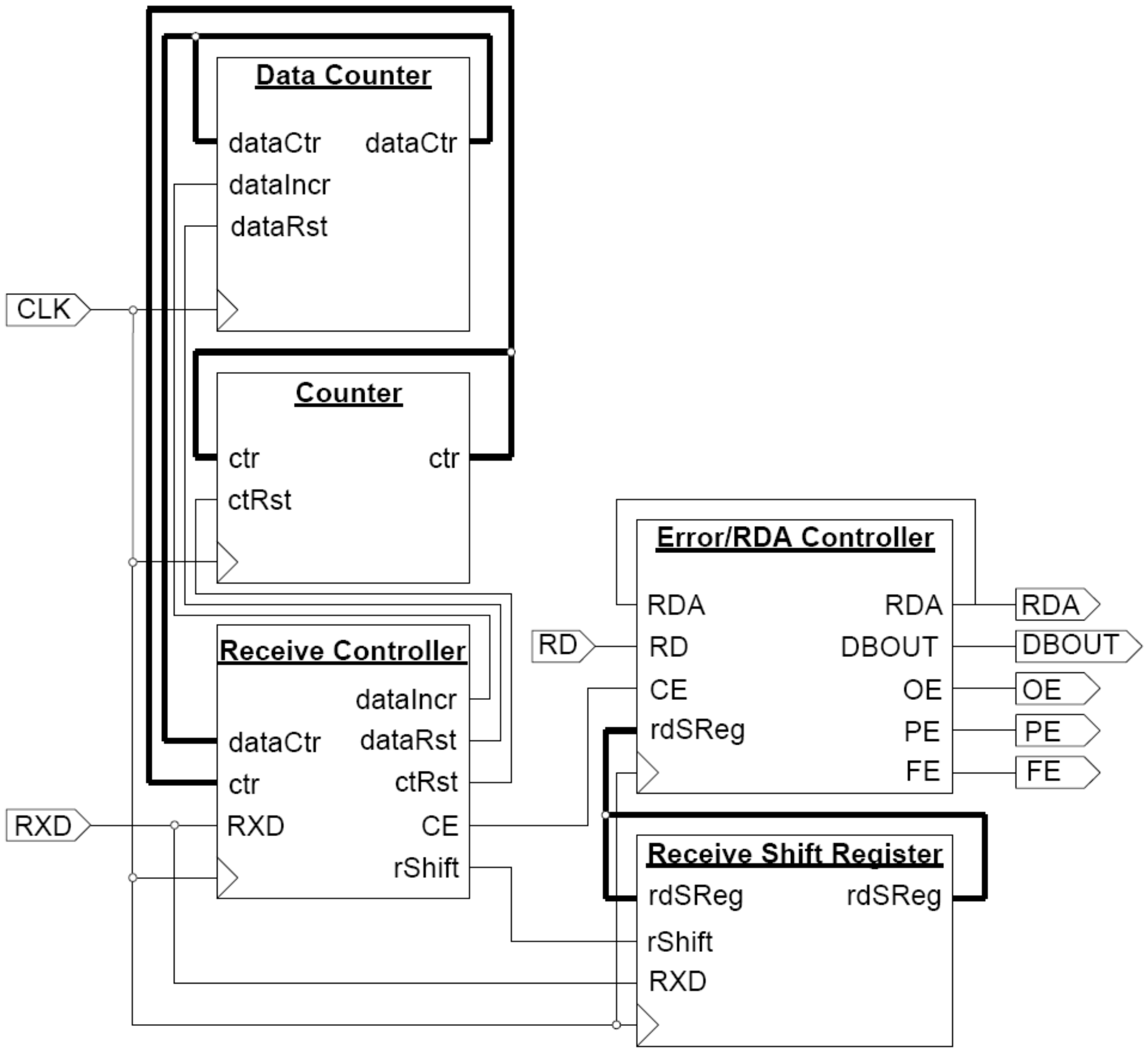

Figure 5 ***The Receiver***

Transmitter Circuit Operation

The transmitter portion of the UART accepts a byte of data from the DBIN port and transmits it as serial data on the TXD port. The baud rate for the transmission is the same as the baud rate for the receiving portion of the UART, so it can be changed in the same way as described in the “Receiving Protocol” section.

To transmit the byte stored in the DBIN port, the transfer portion of the UART must contain a transfer controller, two counters for synchronization, and a transfer shift register. The transfer controller uses the two synchronization counters to control the rate at which the data byte must be transmitted across the TXD port, and the number of bits transmitted. One counter is used to delay the transfer controller between transmissions, while the other counter is used to keep track of how many transmissions have been sent out. The TXD port is set equal to the least significant bit of the transfer shift register, allowing data to be transmitted by simply shifting the register to the right once at the time of each transmission. A state diagram of the transfer controller can be seen in Figure 6.

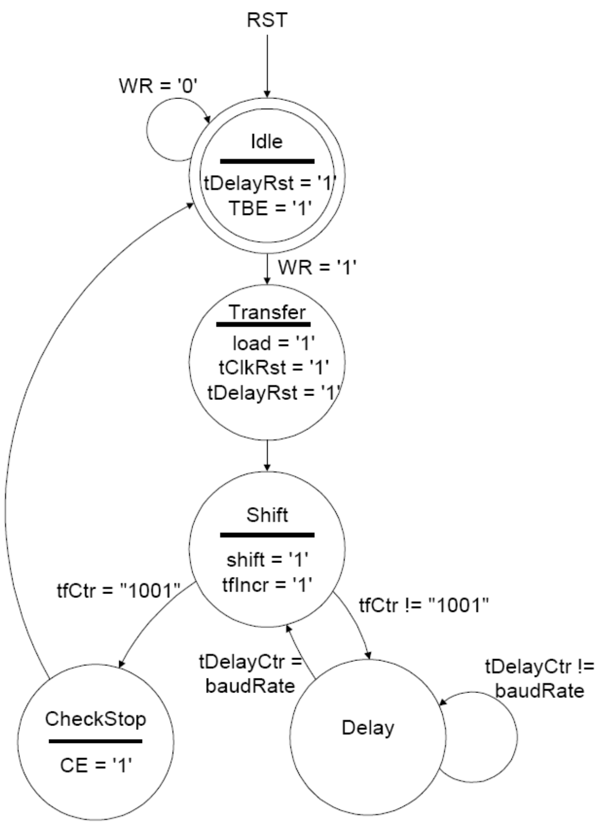

Figure 6 ***The Transferring State Machine***

The transfer portion of the UART stays idle until the WR port goes high. When this happens, the UART is told to write whatever data is in the DBIN port, and the next state, Transfer, is loaded in the transfer state machine.

The Transfer state prepares the transfer shift register to transmit data. By setting load = ‘1’, the shift register is loaded with a start bit, the data byte in DBIN, a proper parity bit, and a stop bit. The two reset signals, tClkRst and tDelayRst, are also set to ‘1’ to reset the two synchronization counters. The next state loaded is the Shift state.

The Shift state sets the shift signal to ‘1’ in order to shift the shift register right one. The tfIncr signal is also set to ‘1’ to increment the data counter. If the data counter is not equal to nine, all of the transfer shift register has not been shifted yet, and the Delay state is gone too. If all of the transfer shift register has been shifted, the WaitWrite state is loaded.

The Delay state is entered so that the transmission process sends out the data at the appropriate baud rate. Once the tDelayCtr is equal to the baudRate constant, the delay state is left, and the shift state is reloaded.

Once the WaitWrite state has been entered, the transmission is complete. This state is needed to make sure that the WR port that was held high to start the transfer process has been unasserted. Without this state, if the WR port is held high for too long, a garbage transmission will occur.

A block diagram of the transmit portion of the UART can be seen in Figure 7.

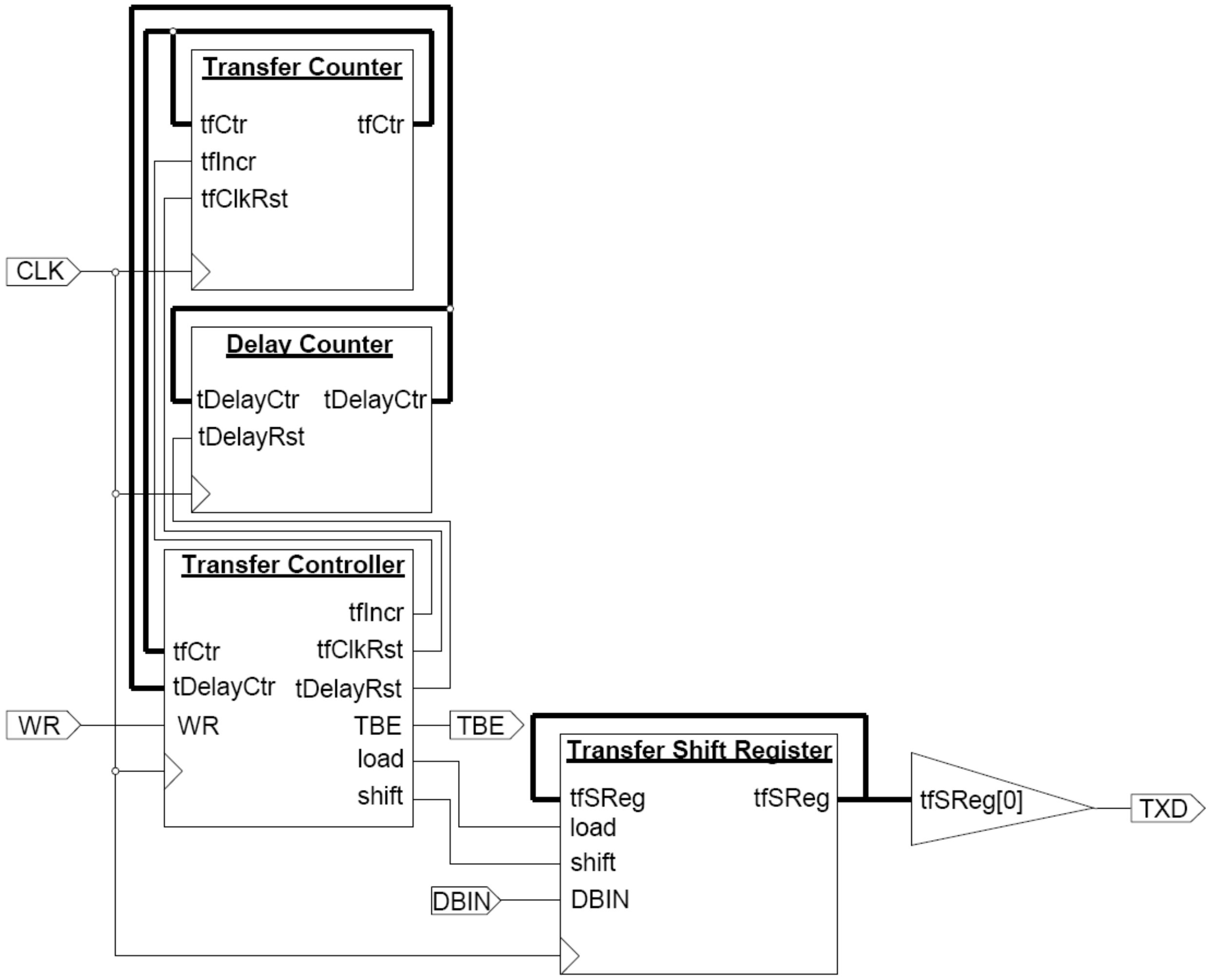

Figure 7 ***The Transmitter***