Application Note
Simple UART Receiver for Quartus II Implementation

Introduction

A Universal Asynchronous Receiver/Transmitter (UART) is a device that can send and receive serial data (usually an ASCII character) without a common clock between the transmitter of one UART and the receiver of another. Both transmitter and receiver clocks must run at the same rate, but they are not required to be synchronous. This device is most commonly used in modems, in the serial ports (COM ports) of PCs, and in embedded systems requiring RS-232 serial communication.

Figure 1 shows the format of an asynchronously transmitted ASCII character at TTL levels. The transmission line remains HIGH when no characters are transmitted. To indicate the start of a character, the line goes LOW for one bit time. This is called the start bit. This is followed by 8 data bits, LSB first. The character ends when the line goes HIGH for one bit time (stop bit).

Other formats are possible, such as 7 data bits, or two stop bits. Some formats also include a parity bit (an error-checking bit) after the data bits. Since the format is Figure 1 has 8 data bits, no parity bit, and 1 stop bit, it is sometimes described as 8N1.

Figure 1 ASCII composite character

UART Receiver Block Diagram and Functional Description

The block diagram in Figure 2 represents a UART receiver that can be implemented in an Altera FPGA. The entire receiver design is shown in the Quartus Block Diagram File **UART_rx.bdf**.

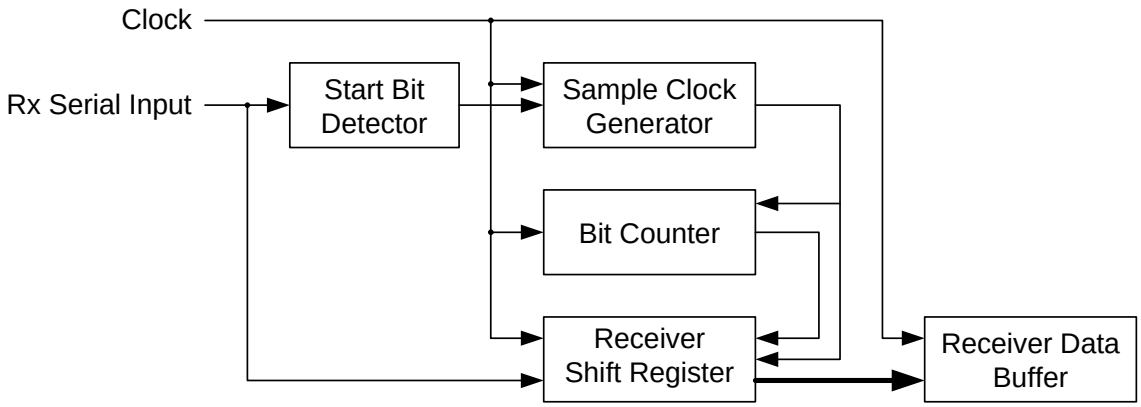

Figure 2 Block Diagram of a UART Receiver

The UART receiver works as follows:

- When the Rx Serial Input is in the Idle state (HIGH), the receiver is quiescent, waiting for a new character.
- The Start Bit Detector detects a negative edge on the Rx Serial Input and enables the Sample Clock Generator.
- The Sample Clock Generator produces a pulse to enable the Receiver Shift Register at or near the centre of each incoming bit, allowing the bit to be serially shifted into the receiver. Figure 3 shows the sample clock relative to the incoming bits. The pulse is placed in the centre of each bit to minimize distortions, which usually occur at or near the bit edges. The sample clock is idle before the start bit is detected and produces exactly 10 pulses after that.
- The bit counter counts 10 bits (1 start bit, 8 data bits, and 1 stop bit).
- When all the bits have been received, the 8 data bits are transferred in parallel to the Receiver Data Buffer and the receiver is automatically reset, making it ready to receive a new character.

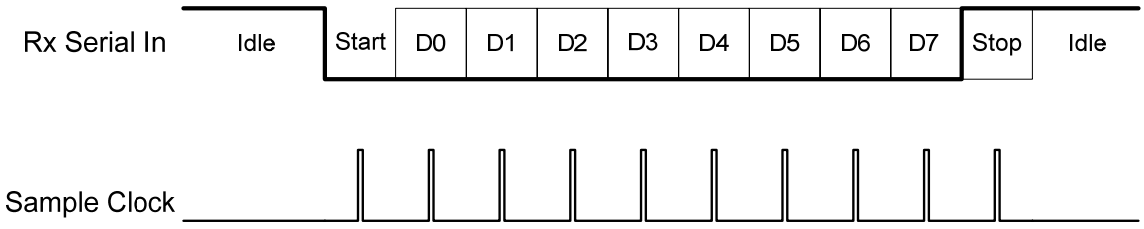

Figure 3 Sample Clock Applied to Incoming Rx Data

UART Receiver Block Symbol

The following symbol represents the UART receiver developed for an Altera FPGA.

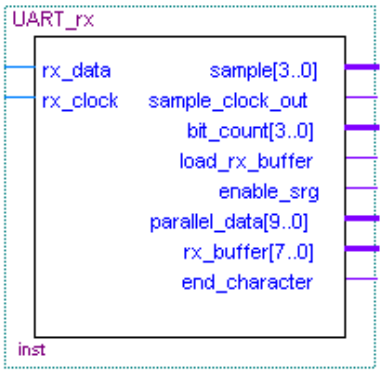

Figure 4 UART Rx Block

Table 1 summarizes the functions of the UART receiver.

Table 1 UART Receiver Functions

Port Name	Function
rx_data	Serial data stream in asynchronous composite character format (Assumes 8N1 format)
rx_clock	Receiver clock (runs at 16× bit rate of **rx_data**)
sample[3..0]	(For monitoring only.) Value of the counter that generates the sample clock.
sample_clock_out	(For monitoring only.) Receiver sample clock, consisting of a HIGH pulse at the centre of each incoming serial bit. This bit enables the bit counter and receiver shift register.
bit_count[3..0]	(For monitoring only.) Counts the number of incoming data bits.
load_rx_buffer	(For monitoring only.) This line generates a HIGH pulse at the end of a received character (when the bit counter is at 1010). The pulse transfers the data bits of **parallel_data[8..1]** to **rx_buffer[7..0]**
enable_srg	(For monitoring only.) This line is HIGH to enable the receiver shift register, from bit counter values of 0000 to 1001. When the entire character is received (at bit 1010), **enable_srg** goes LOW to stop the shift register.
parallel_data[9..0]	(For monitoring only.) Serial shift register that receives serial data, including start and stop bits. When full, 8 data bits are transferred to **rx_buffer**.
rx_buffer[7..0]	Contains last received data byte, excluding start and stop bits
end_character	(For monitoring only.) Goes LOW when a character has been transferred to the **rx_buffer**, thus asynchronously resetting the start bit detector, bit counter, and sample clock counter.

Start Bit Detector

Figure 5 shows a flip-flop that detects the start bit of a serial ASCII character. The flip-flop sets on the first negative edge of the data and is used to enable the sample clock generator. The flip-flop is automatically reset when all bits of an incoming character have been received, thus making it ready to receive a new character.

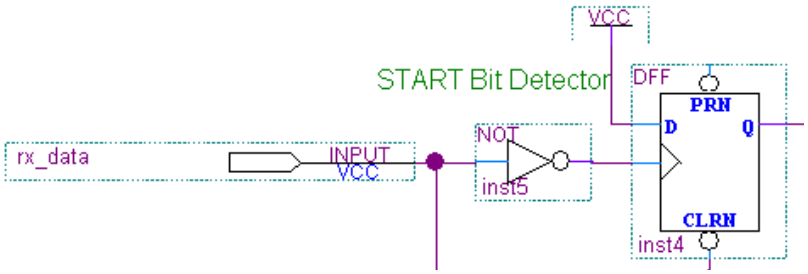

Figure 5 Start bit detector

Figure 6 shows how the start bit detector enables the sample clock generator when a character is being received.

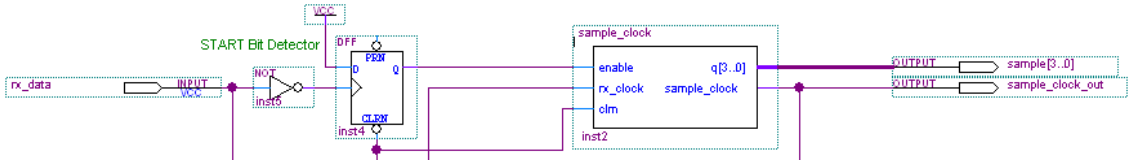

Figure 6 Sample clock generator

Sample clock generator

Everything in the receiver except the start bit detector is synchronously clocked by the receiver clock, **rx_clock**, which runs at a rate of 16 × the incoming bit time. For example, an incoming character transferring at a rate of 9600 baud requires a receiver clock frequency of 16 × 9600 Hz = 153,600 Hz.

The sample clock generator is a 4-bit decoded counter that is used to enable three modules within the UART receiver:

- a mod-11 counter that counts the number of incoming bits,
- a 10-bit shift register that receives incoming data, and
- an 8-bit flip-flop that transfers and stores data from the receiver shift register after a character has been received.

This output, called **sample_clock_out**, goes HIGH whenever the counter value equals 8_{10} or 1000_2. This has the effect of producing one pulse, with a width of 1/16 the received data bit time, after 8 receiver clock cycles (T/2 for a bit time T), and then once every 16 cycles (T) thereafter. This signal generates a pulse in the centre of every incoming data bit, as shown in Figure 7. The sampling is done in the centre of the bit to minimize distortion caused by transitions between logic levels on the bit boundaries.

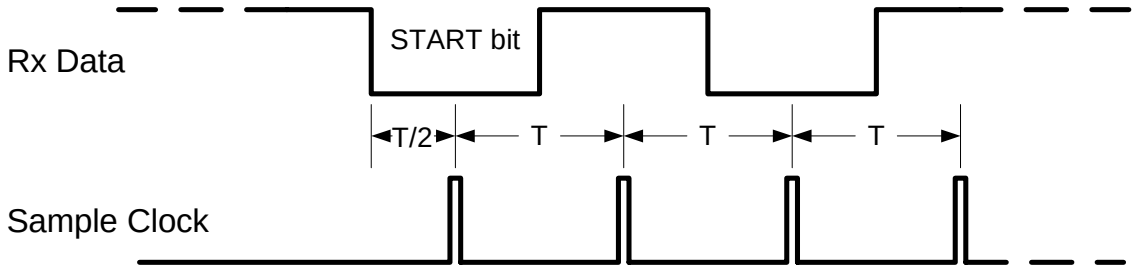

Figure 7 Sample clock timing

The sample clock is used to shift the serial bits into the receiver shift register and also to count the number of bits that have been received by clocking the mod-11 bit counter.

Bit counter

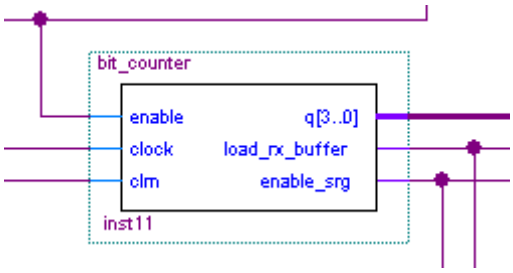

Figure 8 Bit counter

The bit counter, shown in Figure 8, is clocked by the receiver clock. It is enabled by the sample clock generator for 1 of every 16 cycles of the receiver clock. It therefore increments once each bit time, counting the incoming bits. The counter's internal decoder outputs (labeled as **load_rx_buffer** and **enable_srg**) are both activated after the 10^{th} sample clock, when the bit counter holds a value of 10_{10} or 1010_2. (Since the counter holds a maximum count of 10, it has a modulus of 11.) **Enable_srg** goes LOW to disable the receiver shift register and thus stop it from shifting any further once all bits have been received. At the same time **load_rx_buffer** goes HIGH to transfer the received data from the receiver shift register to the receiver data buffer.

Serial Shift Register

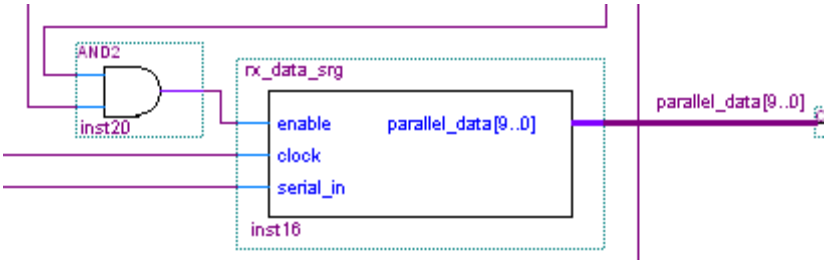

Figure 9 Receiver shift register

The 10-bit serial receiver shift register, illustrated in Figure 9, is enabled by the sample clock generator and the **enable_srg** output of the bit counter. It is synchronously clocked by the receiver clock. Data bits are applied sequentially to the **serial_in** input. The circuit will stop shifting after the **enable_srg** line goes LOW at the end of the received data. At this point, the shift register will contain a start bit, 8-bit data, and a stop bit.

Double buffering

UART receivers often contain a parallel-loading buffer to transfer received data from the serial receiver shift register. This allows data to be protected from being overwritten by an incoming new character. This feature is called double buffering.

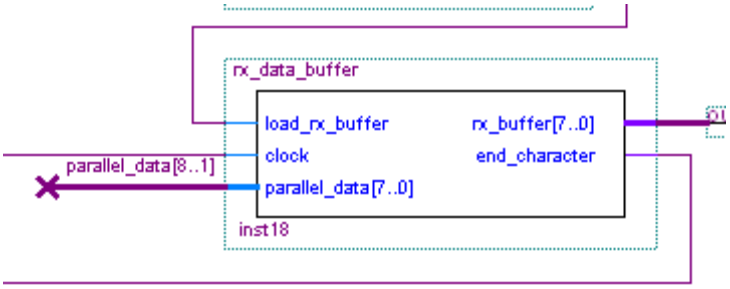

Figure 10 Rx buffer

Figure 10 shows the module that implements double buffering in this receiver. The received data are transferred from the middle 8 bits of the receiver shift register to an 8-bit flip-flop, called the receiver buffer. In other words, **parallel_data[9]** (the stop bit) and **parallel_data[0]** (the start bit) are discarded and the remaining data are transferred from **parallel_data[8..1]** to **rx_buffer[7..0]**.

The transfer is made by making the **load_rx_buffer** input go HIGH and clocking the register with the **rx_clock** input. When the transfer is made, an output called **end_character** goes LOW.

End_character is an active-LOW signal that asynchronously resets the start bit detector, the sample clock generator, and the bit counter, thus preparing the receiver to receive a

new character. **End_character** must return to a HIGH level when **load_rx_buffer** is no longer HIGH, so that the start bit detector, sample clock generator, and bit counter are able to function properly for the next character. Figure 11 shows the reset connections to these three modules.

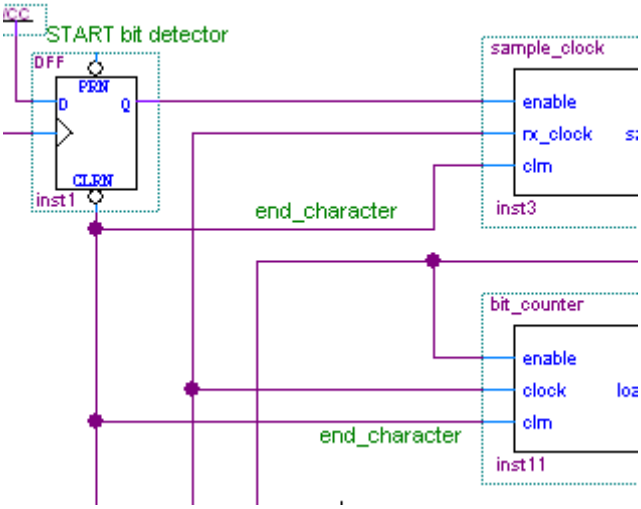

Figure 11 End Character Reset Connections

After 10 sample clocks, the serial shift register now contains the correct configuration of data (1 start bit, 8 data bits, 1 stop bit), the data has been transferred to the receiver buffer, and the circuit has been reset to be ready for a new character.

Quartus II Simulation

Figure 12 shows a Quartus II simulation of the UART receiver, with ASCII characters 55 and 7F shifted in on the serial **rx_data** line. Note that the sample clock begins halfway into the start bit of each character, then generates a pulse at the centre of each bit. On the 10th sample clock pulse of each character, the **load_rx_buffer** line pulses once and enables the transfer of the received data bits to the **rx_buffer**. At this point the bit counter (shown as **bit[3..0]**) and the sample clock generator (shown as **sample[3..0]**) are asynchronously reset in preparation for a new character.

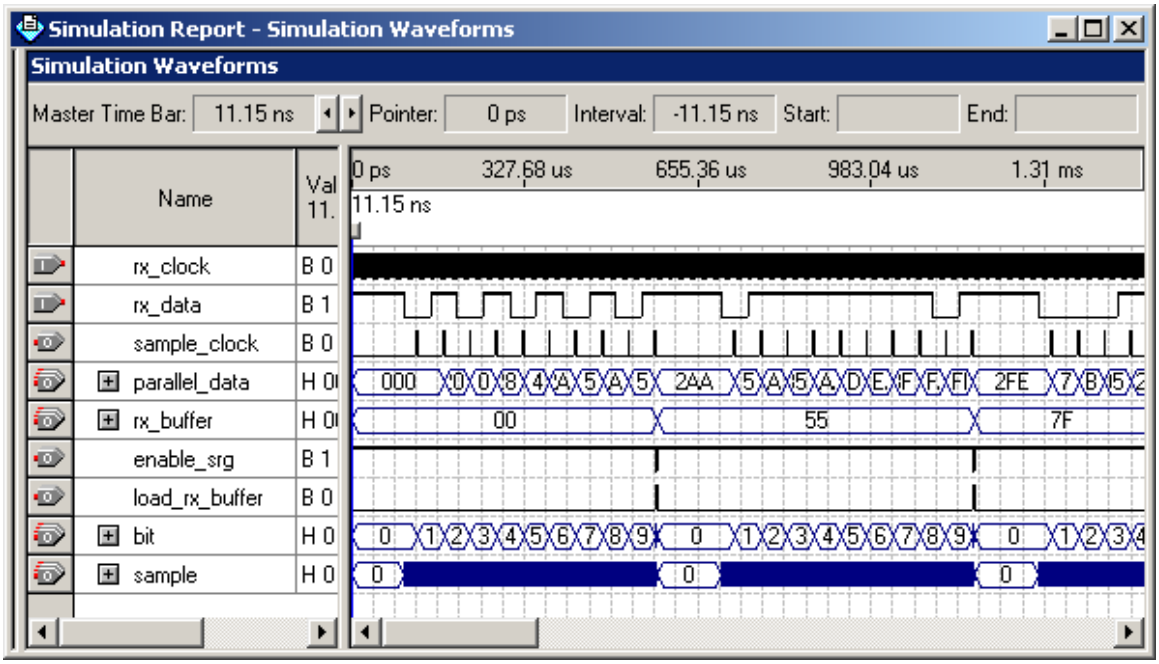

Figure 12 UART_rx Simulation Waveforms

Figure 13 shows a simulation detail that more clearly shows the **rx_clock** waveform and how the sample clock relates to it.

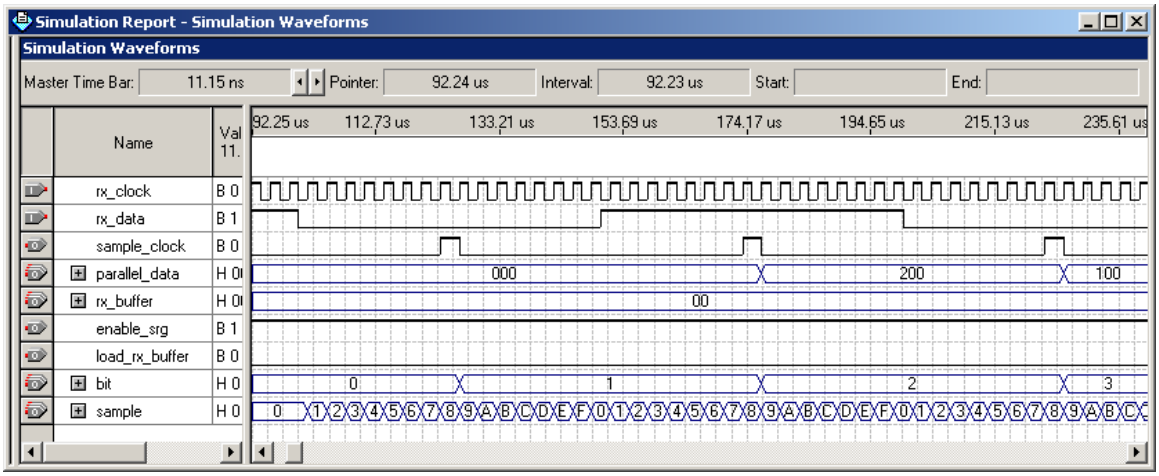

Figure 13 Simulation Detail for UART_rx