Data	Even Parity	Odd Parity
0xA0 = 10100000	0	1

FIGURE 5.2 Odd and even parity.

shaking involves a receiver driving a ready signal to the transmitter. The transmitter sends data only when the receiver signals that it is ready. UARTs may support hardware handshaking. Any software handshaking is the responsibility of the UART control program.

Software handshaking works by transmitting special binary codes that either pause or resume the opposite end as it sends data. *XON*/*XOFF* handshaking is a common means of implementing software flow control. When one end of the link is ready to accept data, it transmits a standard character called XON (0x11) to the opposite device. When the receiver has filled a buffer and is unable to accept more data, an XOFF character (0x13) is transmitted. It is by good behavior that most flow control schemes work: the device that receives an XOFF must respect the signal and pause its transmission until an XON is received. It is not uncommon to see an XON/XOFF setting in certain serial terminal configurations.

A generic UART is shown in Fig. 5.3. The UART is divided into three basic sections: CPU interface, transmitter, and receiver. The CPU interface contains various registers to configure parity, bit rate, handshaking, and interrupts. UARTs usually provide three parity options: none, even, and odd. Bit rate is selectable well by programming an internal counter to arbitrarily divide an external reference clock. The range of usable bit clocks may be from several hundred bits per second to over 100 kbps.

Interrupts are used to inform the CPU when a new byte has been received and when a new byte is ready to be transmitted. This saves the CPU from having to constantly poll the UART's status registers for this information. However, UARTs provide status bits to aid in interrupt status reporting, so a simple serial driver program could operate by polling rather than implementing an interrupt service routine. Aside from general control and status registers, the CPU interface provides access to transmit and receive buffers so that data can be queued for transmission and retrieved upon arrival. Depending on the UART, these buffers may be only one byte each, or they may be several bytes

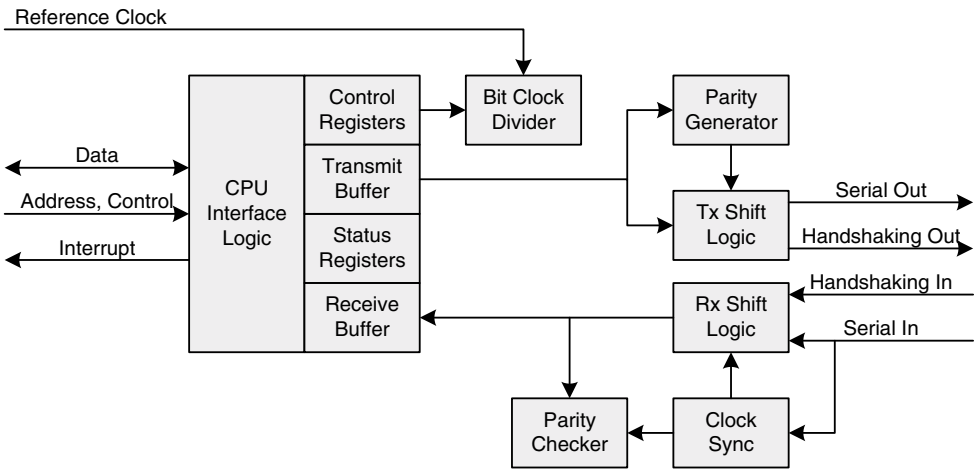

FIGURE 5.3 Generic UART block diagram.

implemented as a small FIFO. Typically, these serial ports run slow enough to not require deep buffers, because even a slow CPU can easily respond to a transmit/receive event before the data link underruns the transmit buffer or overruns the receive buffer.

The transmit section implements a parallel-to-serial shift register, parity generator, and framing logic. UARTs support framing with a start bit and one or two stop bits where the start bit is a logic 0 and stop bits are logic 1s. It is also common to transmit data LSB first. With various permutations of framing options, parity protection, and seven or eight data bits, standard configuration notation is of the form <parity:N/E/O>-<width:8/7>-<stop-bits:1/2>. For example, N-8-1 represents no parity, 8 data bits, and 1 stop bit. E-8-2 represents even parity, 8 data bits, and 2 stop bits. To help understand the format of bytes transmitted by a UART, consider Fig. 5.4. Here, two data bytes are transmitted: 0xA0 and 0x67. Keep in mind that the LSB is transmitted first.

Receiving the serial data is a bit trickier than transmitting it, because there is no clock accompanying the data with which the data can be sampled. This is where the asynchronous terminology in the UART acronym comes from. The receiver contains a clock synchronization circuit that detects the start-bit and establishes a timing reference point from which all subsequent bits in the byte will be sampled. This reference point is created using a higher-frequency receive clock. Rather than running the receiver at 1x the bit rate, it may be run at 16x the bit rate. Now the receive logic can decompose a bit into 16 time units and slide a 16-clock window according to where the start bit is observed. It is advantageous to sample each subsequent bit halfway through its validity window for maximum timing margin on either side of the sampling event. This allows maximum flexibility for settling time around the edges of the electrical pulse that defines each bit.

Consider the waveform in Fig. 5.5. When the start bit is detected, the sampling window is reset, and a sampling point halfway through is established. Subsequent bits can have degraded rising and falling edges without causing the receiver to sample an incorrect logic level.

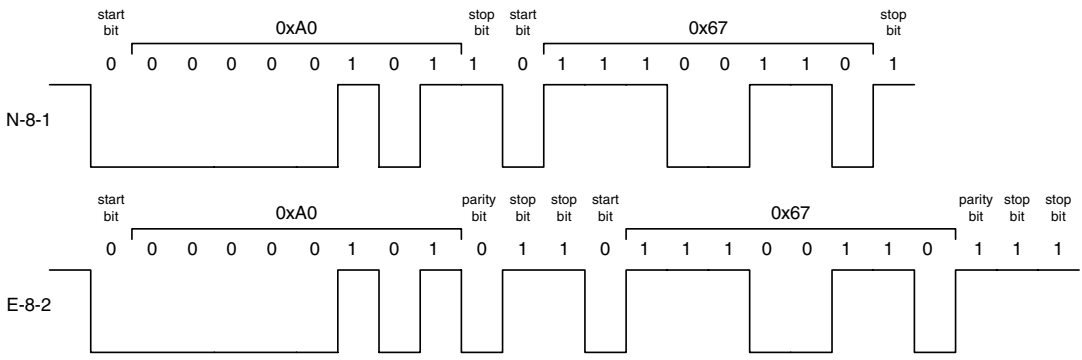

FIGURE 5.4 Common byte framing formats.

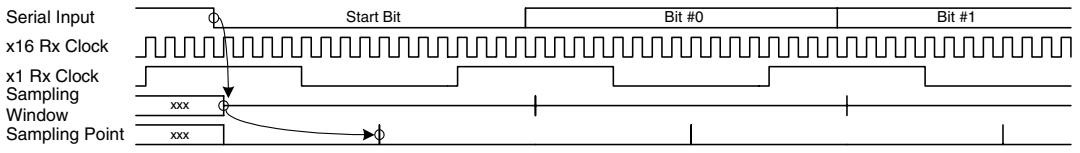

FIGURE 5.5 UART receive clock synchronization.