DESIGN with PIC MICROCONTROLLERS

John B. Peatman

Professor of Electrical and Computer Engineering
Georgia Institute of Technology

PRENTICE HALL, Upper Saddle River, New Jersey 07458

Chapter

11

UART

11.1 OVERVIEW

A UART, *universal asynchronous receiver transmitter*, is a module included in the following parts: PIC16C63, PIC16C65A, PIC16C73A, and PIC16C74A. It is omitted from the following:

PIC16C62A (which is a reduced-feature version of PIC16C63)
PIC16C64A (which is a reduced-feature version of PIC16C65A)
PIC16C72 (which is a reduced-feature version of PIC16C73A).

How this unit works, how it can be used to create a serial interface to a personal computer, and how it can be used to interconnect two PICs will be discussed in this chapter.

11.2 WAVEFORMS AND BAUD-RATE ACCURACY

When serial data is transmitted *asynchronously*, the data stream is generated with the transmitter's clock. The receiver must synchronize the incoming data stream to the receiver's clock.

An example of the transmission of 4 bytes is shown in Figure 11-1. Each 8-bit byte is *framed* by a START bit and a STOP bit. For transmission at 9,600 Bd, each of these bits lasts for a *bit time* (BT) of 1/9,600 second. Before the first frame is transmitted, the line from the transmitter's TX output to the receiver's RX input idles high. The receiver monitors its RX input, waiting for the line to drop low because of the transmission of the (low) START bit. The receiver synchronizes on this high-to-low transition. Then the receiver reads the 8 bits of serial data by sampling the RX input at

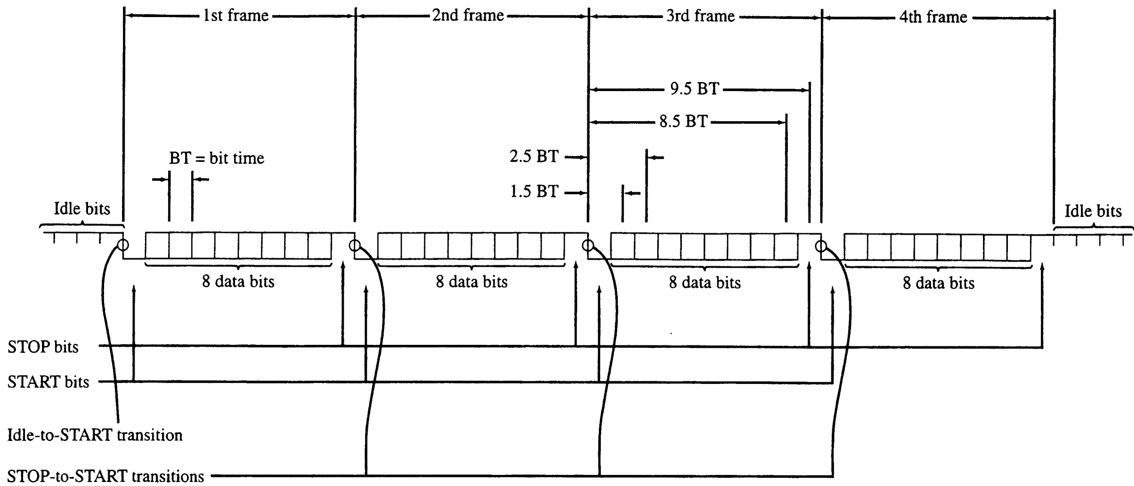


Figure 11-1 Four data frames having a serial protocol of one START bit, eight data bits, and one STOP bit.

1.5 BT, 2.5 BT, 3.5 BT, 4.5 BT, 5.5 BT, 6.5 BT, 7.5 BT, and 8.5 BT

as shown in Figure 11-1. It checks that the framing of the byte has been interpreted correctly by reading what should be a high STOP bit at 9.5 BT. If the RX line is actually low at this time, for whatever reason, the receiver sets a flag to indicate a *framing error*. Regardless of whether or not a framing error occurs, the receiver then begins again, *resynchronizing* upon the next high-to-low transition of the RX line. Because of this resynchronization, the receiver can generate its own baud-rate clock that only approximates the transmitter's baud-rate clock and yet the receiver can recover the serial data perfectly.

Example 11-1 Assume the transmitter transmits data at exactly 9,600 Bd and assume the receiver measures its sampling times from the exact moment when the STOP-TO-START transition occurs. How far off from 9,600 Bd can the receiver's baud-rate clock be and still recover the data and the STOP bit correctly?

Solution As illustrated in Figure 11-1, the STOP bit is read after 9.5 bit times. Consider the consequence if the receiver's baud rate clock is off sufficiently to cause the sampling to be off by ± 0.5 bit time after 9.5 bit times. The sampling of the first data bit at 1.5 bit times of the receiver's baud-rate clock will occur slightly off center of the bit time generated by the transmitter. This off-centeredness progresses with successive bits to the point where the STOP bit will be read unreliably and where the next STOP-TO-START transition may be missed because the receiver is not yet looking for it. This error in the receiver's baud-rate clock amounts to

$$(\pm 0.5/9.5) \times 100 = \pm 5.3\ \%$$

Example 11-2 The PIC's baud-rate clock operates at either of two ranges, called *high-speed*

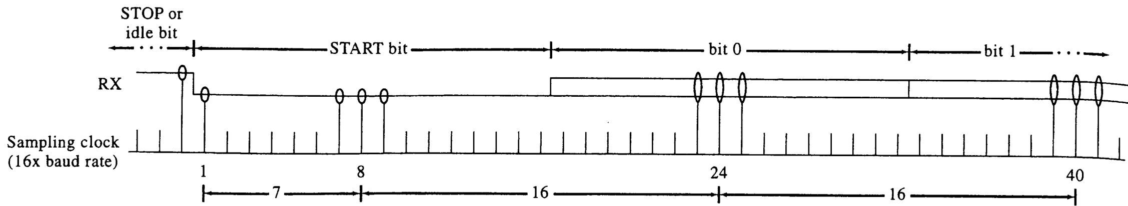


Figure 11-2 Receiver's sampling of RX using its low-speed baud rate circuitry.

baud rate and *low-speed baud rate*. Using the *low-speed* baud rate, the receiver looks for the STOP-TO-START transition by sampling its RX input every 1/16th of one of its bit times, as shown in Figure 11-2. Then it counts six more of these sample times to a point where it reads a cluster of three closely spaced samples of RX and votes among them to ensure that it is seeing the low START bit. Thereafter, it reads successive clusters of three samples spaced 16 sample times apart. In effect, the receiver is reading its input every 16 periods of its sample clock. How far off from 9,600 Bd can the receiver's baud-rate clock be and still recover the data and the STOP bit correctly?

Solution The mechanism for detecting the STOP-TO-START transition can throw the samples off from the center of each bit time by as much as 1/16th of a bit time, even if the receiver's baud rate *exactly* matches the transmitter's baud rate. If the receiver's baud-rate clock is off sufficiently to cause the sampling to be off by

$$\pm(0.5 \text{ bit time} - 1/16 \text{ bit time}) = \pm 0.4375 \text{ bit time}$$

after 9.5 bit times, then an error can occur. This places a baud rate error limit of

$$(\pm 0.4375/9.5) \times 100 = \pm 4.6\%$$

on the receiver's baud-rate clock (assuming the transmitter's baud-rate clock matches its nominal rate exactly).

Example 11-3 Examine the baud rate accuracy requirement for the *high-speed* baud rate. The receiver's sampling scheme is shown in Figure 11-3.

Solution In this case, the sampling rate is eight times higher than the baud rate. Consequently, the pinpointing of when the STOP-TO-START transition occurs may be off by one-eighth of a bit time. When RX is sampled to read the START bit, the data bits, and the STOP bit, again three samples are collected and a vote taken among the three. The samples are collected using the three rising and falling edges of the crystal clock (OSC), as shown in Figure 11-3. For any baud rate much less than the crystal clock rate (e.g., 9,600 Bd << 4 MHz) the RX line is sampled almost exactly at the times of the sampling clock of Figure 11-3. Consequently, this places a baud-rate error limit of

$$(\pm(0.5 \text{ bit time} - 1/8 \text{ bit time})/9.5) \times 100 = \pm 3.9\%$$

on the receiver's baud-rate clock.

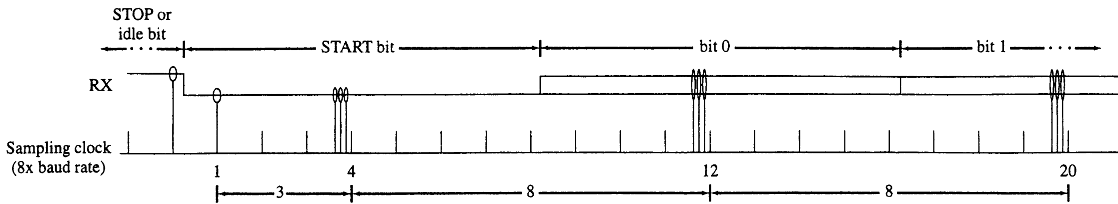


Figure 11-3 Receiver's sampling of RX using its high-speed baud rate circuitry.

11.3 BAUD-RATE SELECTION

Given the considerations of the preceding section, a desired baud rate can now be approximated by the UART's baud-rate generator. If the crystal clock rate were selected to be a carefully chosen multiple of the desired baud rate, then the baud-rate generator would produce the desired baud rate exactly. The clock rates used by Microchip to characterize the three speed grades of their parts

4 MHz 10 MHz 20 MHz

do not provide exact multiples of the popular 9,600 Bd and 19,200 Bd rates commonly used by personal computer serial ports. However, the flexibility of the baud-rate generator circuitry permits close approximations to both 9,600 Bd and 19,200 Bd with any of the standard crystal clock rates. The baud rate is derived from the crystal rate using an 8-bit presettable divider and a fixed divider of either 16 or 64, as shown in Figure 11-4b. The results are tabulated in Figure 11-4a. Even in the worst case, the percent error of the approximate baud rate is only one-third of the percent error that cannot be tolerated by the UART.

Nominal baud rate	OSC = 4 MHz			OSC = 10 MHz			OSC = 20 MHz		
	BRGH	SPBRG	% error	BRGH	SPBRG	% error	BRGH	SPBRG	% error
9,600 baud	1(high)	25	+0.16%	1(high)	64	+0.16%	1(high)	129	+0.16%
19,200 baud	1(high)	12	+0.16%	1(high)	32	−1.4%	1(high)	64	+0.16%

(a) Register contents and accuracy of approximated baud rate

For BRGH = 1 (high-speed baud rate) **For BRGH = 0** (low-speed baud rate)

$$\text{Baud rate} = \frac{\text{OSC}}{16(\text{SPBRG} + 1)} \qquad \text{Baud rate} = \frac{\text{OSC}}{64(\text{SPBRG} + 1)}$$

(b) Relationship between OSC, BRGH, SPBRG, and baud rate

Figure 11-4 Setup for 9,600 baud and 19,200 baud.

11.4 UART DATA HANDLING CIRCUITRY

The transmit data circuit is shown in Figure 11-5a. To transmit a byte of data serially from the TX pin, the byte is written to the **TXREG** register. Assuming there is not already data in the TSR (transmit shift register), the content of **TXREG** will be automatically transferred to the TSR, making **TXREG** available for a second byte even as the first byte is being shifted out of the TX pin, framed by START and STOP bits.

The receive data circuit is similar, with received data shifted into the RSR (receive shift register). When it is in place, the STOP bit is checked and an error flag is set if the STOP bit does not equal one. In any case, the received byte is automatically transferred into a 2-byte FIFO (first-in, first-out memory). If the FIFO was initially empty, the received byte will fall through to the **RCREG** (receive register) virtually immediately, where it is ready to be read by the CPU. If the CPU is slow in reading the **RCREG**, a second byte can be received at the RX pin. When it is in place in the RSR, it will follow the first byte into the 2-byte FIFO. At that point, the FIFO is full. If a third byte enters the RX pin and is shifted all the way across the RSR before at least one of the two bytes in the FIFO has been read, then the new byte will be lost. An *overflow* error flag will be set, alerting the receiver software of the loss of a byte of data.

At 9,600 Bd, it takes 10/9,600 second, or just a little longer than a millisecond, to receive each byte. If the received bytes are handled under interrupt control, each byte should be easily handled in a timely fashion, well before an overrun error can ever occur. No other interrupt handler should be permitted to lock out this or any other interrupt source for anywhere near a millisecond.

11.5 UART INITIALIZATION

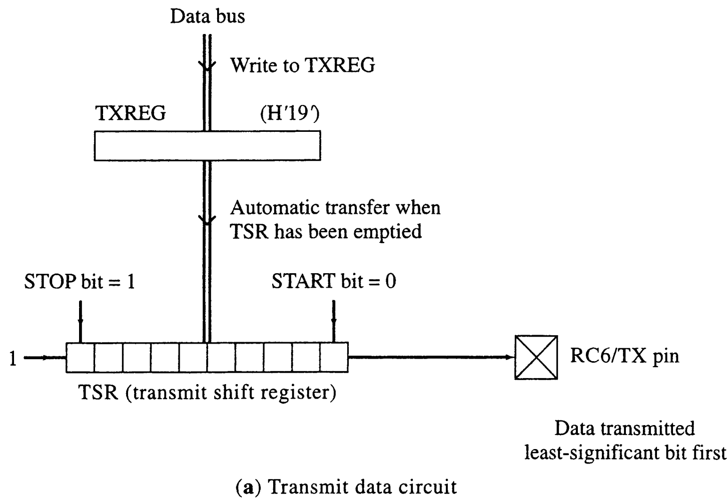
The registers involved with UART use are shown in Figure 11-6. The data direction bits associated with the RC6/TX pin and the RC7/RX pin must both be set up as *inputs*, with ones in bits 6 and 7 of the **TRISC** register. The setting of these two bits disables the general I/O port output circuitry associated with these two pins. (The handling of these bits of **TRISC** stands in contrast to the clearing of bits 3 and 5 of **TRISC** in support of the Serial Peripheral Interface output pins, as shown in Figure 7-2.)

The UART's baud rate and its transmit and receive functions are initialized by writes to **SPBRG**, **TXSTA**, and **RCSTA**, as shown in Figures 11-4 and 11-6. At 9,600 Bd, each transfer takes about a millisecond, so sending or receiving a string of characters is best carried out under interrupt control. The flag and interrupt enable bits of the **PIR1**, **PIE1**, and **INTCON** registers control the timing of the CPU's interactions with the UART.

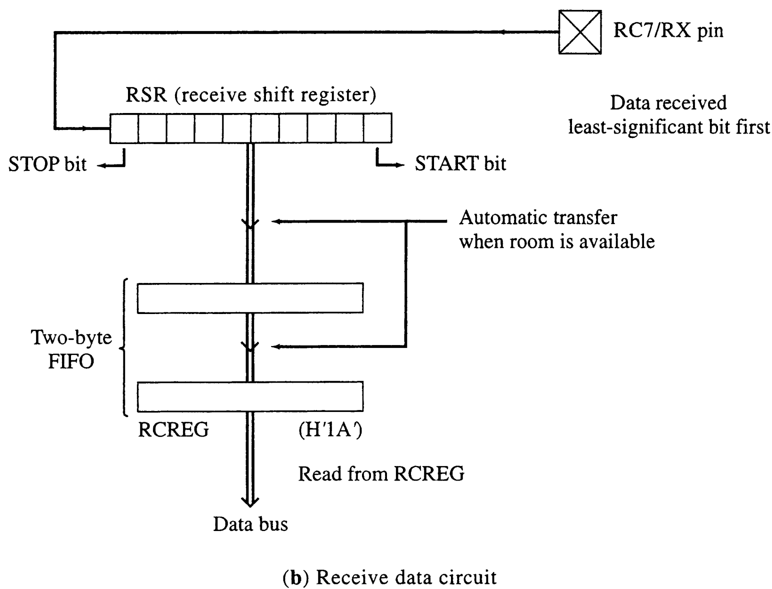
11.6 UART USE

A major application for the PIC's UART is to provide a two-wire (plus ground) serial interface to a personal computer. The circuit of Figure 11-7 uses a Motorola chip to translate between the 0 V and +5 V logic level signal swings on the PIC's RX and TX pins and ± 10 V signal swings that support the RS-232 interface requirements. Both the PIC and the PC should be set up for the same baud rate (e.g., 9,600 Bd) and for one start bit, eight data bits, one stop bit, and no parity.

Given this setup, the PIC will respond to **RCIF** interrupts by reading each byte from the **RCREG** register sent by the PC. The **RCIF** flag will clear itself when the byte read from **RCREG** leaves the receive circuit's FIFO empty.



(**a**) Transmit data circuit



(**b**) Receive data circuit

Figure 11-5 UART's data-handling circuitry.

The PIC sends out a string of bytes by writing them, one by one under interrupt control, to **TXREG**. The **TXIF** flag takes care of itself, clearing automatically when **TXREG** is written to, and setting again as the data written to **TXREG** are automatically transferred to the transmit shift register. At the completion of sending the string of bytes to the PC, the **TXIE** bit in the **PIE1** register is cleared to disable further "transmit" interrupts until another string needs to be sent to the PC.

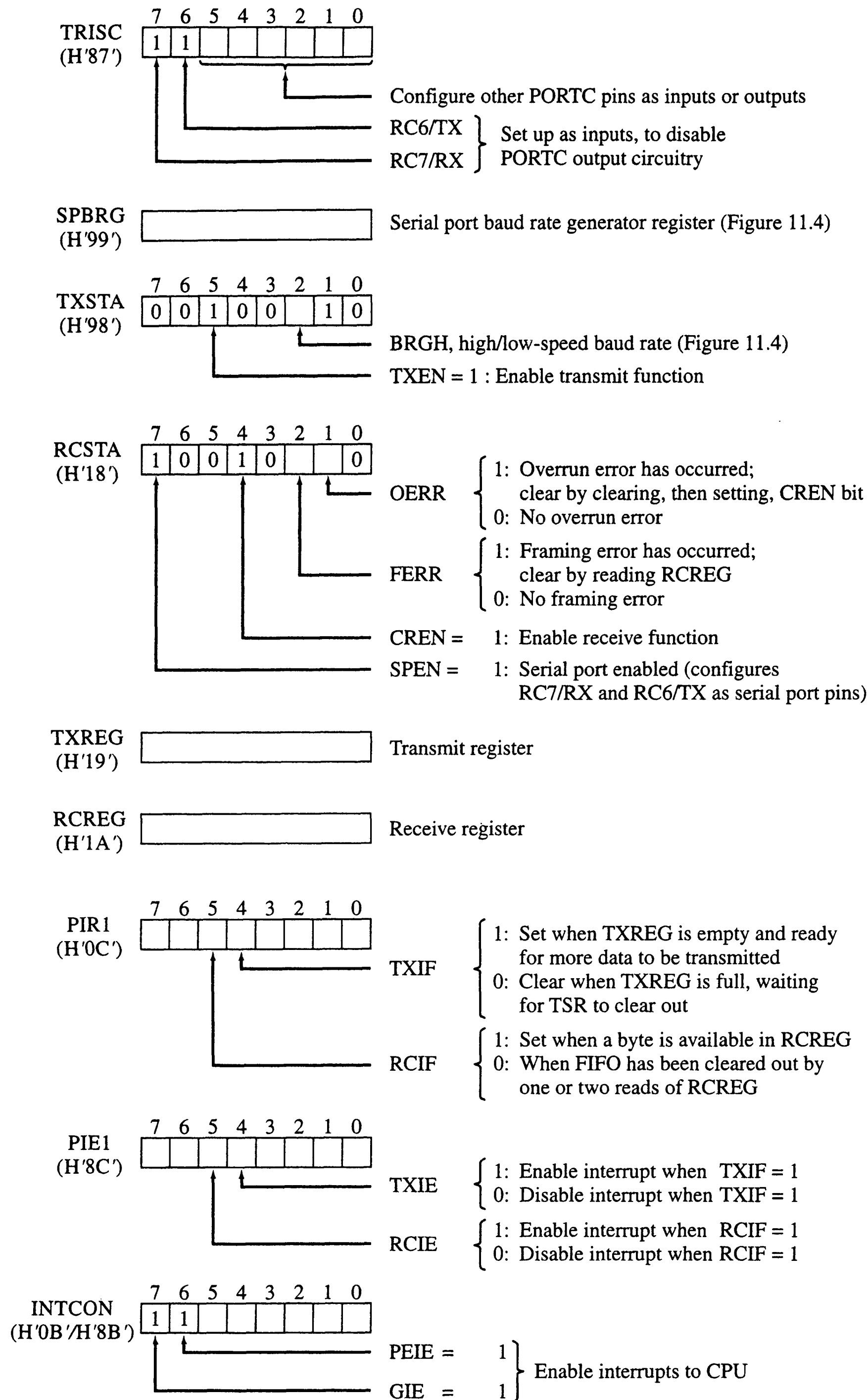


Figure 11-6 UART registers.

Another application of the PIC UART is to couple two PIC's together. In this way some of the work that would be done by one PIC (if only it could do all it needs to do by itself) is off-loaded to a second PIC. Figure 11-8 shows this connection of two PICs, using the maximum possible baud rate to obtain fast coupling between the two PICs. Within 40 internal clock cycles, what is written into one PIC's **TXREG** register appears in the other PIC's **RCREG** register.

Carrying out transfers at this fast rate calls for some precautions if overrun errors are to be avoided, given PICs that are trying to carry out tasks in addition to monitoring the UART's **RCREG** register. A PIC can only receive 2 bytes into its FIFO without reading them immediately. Any further bytes received will be discarded until the earlier bytes are read out of the FIFO, making room for new bytes.

One application that can easily bypass this limitation is illustrated in Figure 11-9. The slave PIC is used with a phoneme generator chip and a speaker to speak any word that is included in its dictionary of N words. The code representing each word is used to access a string of phoneme codes in the slave PIC's program area and sent to the phoneme generator chip, one by one, producing a vocalization of the desired dictionary word.

Given this scenario, the master PIC can send a 1-byte code to the slave PIC to initiate the vocalization of any word in a dictionary of up to 256 words. When the slave PIC has the time to read the received word, it can respond by sending a byte of acknowledgment back to the master PIC. This handshake procedure ensures that no byte will ever be lost because of an overrun error. With room for 2 bytes in the FIFO, the dictionary can easily be extended to more than 256 words by using a 2-byte code to identify each of the words.

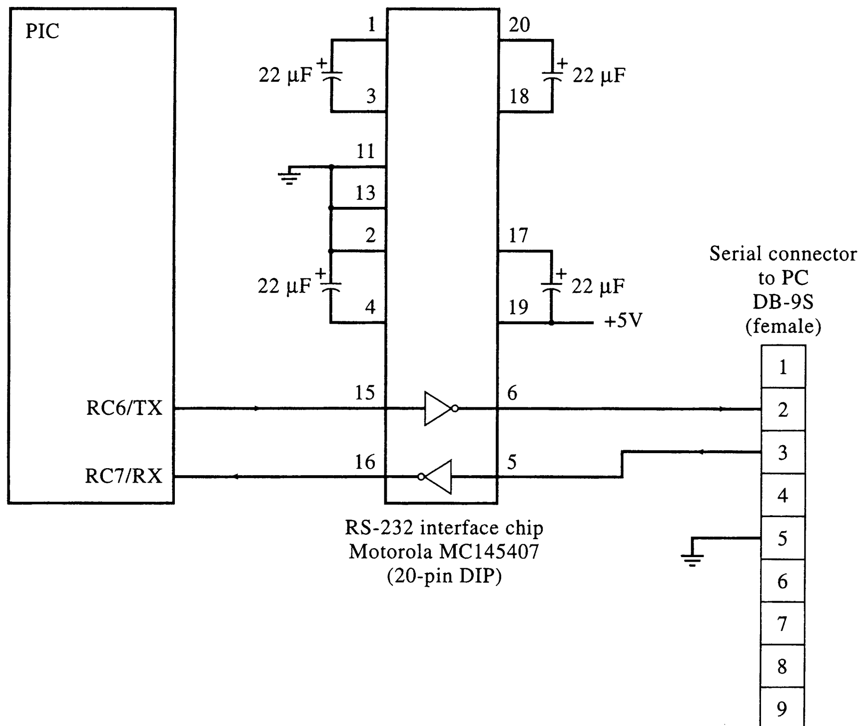
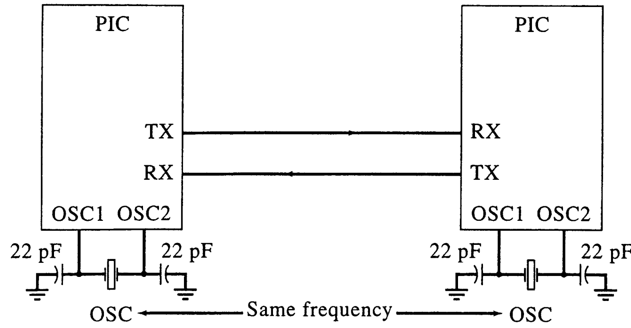


Figure 11-7 PIC's UART interface to a PC.



(**a**) Circuit

BRGH = 1 SPBRG = H'00' Baud rate = OSC/16

	OSC = 4 MHz	OSC = 10 MHz	OSC = 20 MHz
Baud rate	250 kbaud	625 kbaud	1.25 Mbaud
Time to transfer one byte	40 μs	16 μs	8 μs

(**b**) Setup for maximum transfer rate

Figure 11-8 UART interconnection of two PICs.

The slave PIC in this application can go one step further by letting the master PIC quickly download complete sentences of words. In this way, the master PIC can avoid getting tied up with a slow sequence of transfers dictated by the rate at which the phoneme chip can generate the vocalization of each word. The slave PIC need only handshake for each received byte and then put it in a *queue* (i.e., a FIFO implemented in software). As the vocalization of each word is completed, the slave PIC goes to this queue for the next word to be spoken.

PROBLEMS

11-1 Framing error A UART receiver is triggered by a high-to-low transition on its input. If this is a false trigger caused by an isolated noise spike, then hopefully the UART will automatically detect this and begin again to look for a high-to-low transition to trigger upon.

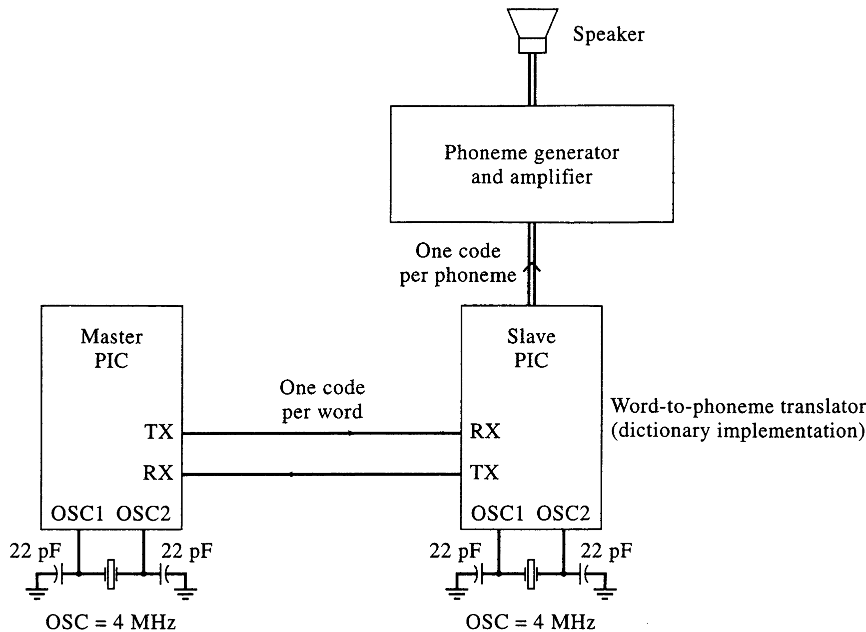


Figure 11-9 Use of UART interface to expand master PIC's resources.

Assuming that the UART input again idles high after the noise spike, when will the UART detect the error? When it reads the START bit as a one? When it reads the data as H'FF'? When it reads the STOP bit as a one?

11-2 Baud-rate selection The entry in Figure 11-4 for OSC = 4 MHz and a nominal baud rate of 9,600 Bd uses the high-speed baud rate choice (**BRGH** = 1) and produces an error from the nominal rate of +0.16%.

(a) Show the calculations to verify this.

(b) What value of **SPBRG** will couple with **BRGH** = 0 to approximate 9,600 Bd as closely as possible? What is the percent error in this case?

(c) Why does the **BRGH** = 1 choice give a lower percent error than the **BRGH** = 0 choice?

11-3 Baud-rate selection The entry in Figure 11-4 for OSC = 20 MHz and a nominal baud rate of 19,200 Bd chooses the high-speed baud rate circuitry with **BRGH** = 1. What would the content of the 8-bit register, **SPBRG**, have to be with **BRGH** = 0? What would be the resulting percent error in the baud rate? Why is this not listed in the table instead of

```
BRGH = 1  and  SPBRG = 64
```

11-4 Baud-rate selection Add another row to the table of Figure 11-4 to handle a nominal baud rate of 38,400 Bd. Between the two possibilities of **BRGH** = 1 and **BRGH** = 0, can you obtain a percent error for each value of OSC that is less than the worst-case tolerances found in Examples 11-2 and 11-3?

11-5 Transmit data circuitry Consider the circuitry shown in Figure 11-5a and the operation of the **TXIF** bit in the **PIR1** register, described in Figure 11-6. For a variable string of bytes to be transmitted plus a pointer to that string in a 1-byte RAM variable called **TXPTR**, an interrupt handler called **TX** can be written to transmit this string. Assume that each interrupt transmits a single byte.

(a) Assuming the UART's transmit circuit is through transmitting any earlier string, how does the mainline code initiate the transmission of this string pointed to by **TXPTR**?

(b) How long after what is done in part (a) will the interrupt handler write to **TXREG**, the output register? Answer this assuming a baud rate of 9,600 and phrase your answer as $<< 1$ ms or ≈ 1 ms.

(c) How long after part (b) will it be before the second write to **TXREG** occurs?

(d) How long after part (c) will it be before the third write to **TXREG** occurs?

(e) Assuming that H'00' serves as an end-of-string designator, what does the interrupt handler do when it fetches H'00' from the string?

11-6 UART use

(a) Using Figures 11-6 and 11-4 for guidance, modify the code of P4.ASM (Figure 5-7) to initialize the UART for use under interrupt control at 9,600 Bd. Assume OSC = 4 MHz.

(b) Modify **IntService** to poll for both TX and RX interrupts, going to **TX** in the one case and **RX** in the other.

(c) Write the **TX** handler. Describe any design decisions you make. Assume that the characters being sent reside in a variable string located in RAM and pointed to by **TXPTR** and transmit just 1 byte per interrupt.

(d) Write the **RX** handler. Assume that the characters being received will be stored in sequential RAM locations pointed to by **RXPTR**. Increment the pointer after each store. You may assume that the number of bytes received will never overrun the amount of RAM available for it. When an end-of-string designator of H'00' is received, signal the mainline code by setting bit 7 of a 1-byte **FLAGS** RAM variable.

11-7 UART use Consider the interconnection of two PICs, as in Figure 11-8 and operating at 250 kBd (with OSC = 4 MHz). The "master" PIC on the left has strings of up to 10 bytes to send to the "slave" PIC on the right. The slave PIC will devote itself entirely to receiving these characters, but only after it terminates what it is presently doing.

Describe a possible protocol for doing this in which the master PIC sends out a single byte requesting the full attention of the slave, and the slave responds with a single byte (perhaps as long as a tenth of a second later) saying it has stopped what it is doing and is now devoting 100% of its CPU time to the monitoring of the UART. In your description, discuss the role of interrupts and mainline code in both master and slave.

11-8 UART use Consider the word-to-phoneme translator application described in conjunction with Figure 11-9. Describe a coding scheme that can be used to code up to 500 words with two bytes. The slave PIC needs to be able to look at each byte, independent of what it has previously received, and tell which byte is which. That is, it must be able to look at one byte and recognize (by looking at it alone) that it needs the *next* byte to combine with this byte into a 2-byte code. Likewise, it must be able to look at the other byte and recognize (by looking at it alone) that it needs to be combined with the previously received byte into a 2-byte code. The binary value of the resulting 2-byte number should not range beyond 0 to 511.